AN $O(n^2 \log n)$ TIME ALGORITHM FOR THE MINMAX ANGLE TRIANGULATION *

HERBERT EDELSBRUNNER†, TIOW SENG TAN† AND ROMAN WAUPOTITSCH†

Abstract. We show that a triangulation of a set of n points in the plane that minimizes the maximum angle can be computed in time $O(n^2 \log n)$ and space $O(n)$. The algorithm is fairly easy to implement and is based on the edge-insertion scheme that iteratively improves an arbitrary initial triangulation. It can be extended to the case where edges are prescribed, and, within the same time- and space-bounds, it can lexicographically minimize the sorted angle vector if the point set is in general position. Experimental results on the efficiency of the algorithm and the quality of the triangulations obtained are included.

Key words. Computational geometry, two dimensions, triangulations, minmax angle criterion, iterative improvement, edge-insertion

AMS(MOS) subject classifications. 68C05, 65M50

Appear in: *SIAM J. Scientific & Statistical Computing*, **13 (4)**, 994–1008, (1992)

1. Introduction. Let S be a finite set of points in the Euclidean plane. A *triangulation* of S is a maximally connected straight line plane graph whose vertices are the points of S. By maximality, each face is a triangle except for the exterior face which is the complement of the convex hull of S. Occasionally, we will call a triangulation of a finite point set a *general triangulation* in order to distinguish it from a *constrained triangulation* which is a triangulation of a finite point set where some edges are prescribed. A special case of a constrained triangulation is the so-called *polygon triangulation* where S is the set of vertices of a simple polygon and the edges of the polygon are prescribed. In this paper only the triangles *inside* the polygon will be of interest.

For a given set of n points there are, in general, exponentially many triangulations. Among them one can choose those that satisfy certain requirements or optimize certain objective functions. Different properties are desirable for different applications in areas such as finite element analysis [1, 3, 23], computational geometry [21], and surface approximation [12, 18]. The following are some important types of triangulations that optimize certain objective functions.

(i) The *Delaunay triangulation* has the property that the circumcircle of any triangle does not enclose any vertex [5].

(ii) The *constrained Delaunay triangulation* has the same property except that visibility constraints depending on the enforced edges are introduced [13].

(iii) The *minimum weight triangulation* minimizes the total edge length over all possible triangulations of the same set of points and prescribed edges [10, 17].

It is known that the Delaunay triangulation maximizes the minimum angle over all triangulations of the same point set [22]. This result can be extended to a similar statement about the sorted angle vector of the Delaunay triangulation [6] and to the constrained case [13]. The Delaunay triangulation of n points in the plane can be constructed in time $O(n \log n)$ [6, 19], and even if some edges are prescribed its

* Research of the first author was supported by the National Science Foundation under grants CCR-8714565 and CCR-8921421. The second author is on study leave from the Department of Information Systems and Computer Science, National University of Singapore, Republic of Singapore.

† Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, USA.

constrained version can be constructed in the same amount of time [20]. There is no polynomial time algorithm known for the minimum weight triangulation if the input is a finite point set, but dynamic programming leads to a cubic algorithm [10] if the input is a simple polygon.

In this paper, we study the problem of constructing a triangulation that minimizes the maximum angle, over all triangulations of a finite point set, with or without prescribed edges. We call such a triangulation a *minmax angle triangulation.* Although avoiding small angles is related to avoiding large angles, the Delaunay triangulation does not minimize the maximum angle — four points are sufficient to give an example to this effect. Triangulations that minimize the maximum angle have potential applications in the area of finite element and surface approximation [1, 2, 8]. Our main result is summarized in the following statement.

MAIN THEOREM. *A minmax angle triangulation of a set of n points in the plane, with or without prescribed edges, can be computed in time $O(n^2 \log n)$ and space $O(n)$.*

Curiously, our algorithm has the same complexity for point sets and for simple polygons. Prior to this paper no polynomial time algorithm for constructing a minmax angle triangulation for a finite point set was known. On the other hand, if the input is a simple n-gon then a cubic time and quadratic space solution can be derived simply by substituting the angle criterion for the edge-length criterion in the dynamic programming algorithm of [10]. Thus, it seemed that the problem for simple polygons is much simpler than for point sets. Indeed, our attempts to apply popular techniques such as local edge-flipping [11, 9], divide-and-conquer [21] and plane-sweep [7] to construct a minmax angle triangulation for a point set were not successful; see also [15].

Instead, we solve the problem by an iterative improvement method based on what we call the *edge-insertion scheme*. An *edge-insertion* step adds some new edge qs to the current triangulation, deletes edges that cross qs, and retriangulates the resulting polygonal regions to the left and the right of qs. The difference to the simpler edge-flip operation is that qs can cross up to a fraction of the current edges, whereas an edge added in an edge-flip crosses only one edge. This difference turns out to be crucial in the case of minimizing the maximum angle: the edge-flip scheme can get stuck in a non-global optimum [15] whereas the edge-insertion scheme is powerful enough to always reach the optimum. A proof of the latter property is sufficient to design a polynomial time implementation of the edge-insertion scheme. Clever strategies to find an edge qs that leads to an improvement of the current triangulation and to retriangulate the created polygonal regions are needed to obtain the claimed $O(n^2 \log n)$ time bound.

Section 2 presents the algorithm to construct a minmax angle triangulation, and section 3 proves the crucial piece needed to show that the algorithm is correct. Section 4 gives the algorithmic details that lead to an efficient implementation of the algorithm. Section 5 discusses the extensions to the constrained case and to the problem of lexicographically minimizing the sorted angle vector. Finally, section 6 presents experimental results, and section 7 mentions some related open problems.

2. The Global Algorithm. In general, there is more than one minmax angle triangulation for a given set of points. Below we outline an algorithm that constructs one such triangulation for a set S of n points in the plane. The maximum angle of a given triangulation $\mathcal{A}$ is denoted by $\mu(\mathcal{A})$.

Construct an arbitrary triangulation $\mathcal{A}$ of S.
repeat
(M1) Find a largest angle $\angle pqr$ of $\mathcal{A}$.
(M2) Apply the ear cutting procedure (section 4) to modify $\mathcal{A}$ by
adding a 'suitable' edge qs to $\mathcal{A}$, where $s \in S - \{p, q, r\}$ and $pr \cap qs \neq \emptyset$,
removing edges that intersect qs (this step creates polygons P and R which have qs as a common edge), and
constructing triangulations $\mathcal{P}$ of P and $\mathcal{R}$ of R so that $\mu(\mathcal{P}), \mu(\mathcal{R}) < \angle pqr$.
until the ear cutting procedure fails to find such a qs.

To show that this algorithm is correct, we need the following two lemmas and some forward references to the cake cutting lemma of section 3 and the ear cutting procedure of section 4. We define $\angle xsy = 0$ if any two of the three points are identical.

LEMMA 2.1. *If xy is an edge in a triangulation $\mathcal{A}$ of a point set S then $\mu(\mathcal{A}) \geq \max_{s \in S} \angle xsy$.*

Proof. Let t be a point so that $\angle xty = \max_{s \in S} xsy$. Thus no points of S lies inside the triangle xty. Clearly, if xty is a triangle in $\mathcal{A}$ then there is nothing to be proved. Otherwise, there exists a triangle utv in $\mathcal{A}$ so that either $u = x$, $v \in S - \{y, t\}$, and uv intersects ty or $u, v \in S - \{x, y, t\}$ and uv intersects both xt and ty. In both cases, $\mu(\mathcal{A}) \geq \angle utv > \angle xty$. $\square$

The proof of the next lemma makes use of the cake cutting lemma to be presented in section 3. We suggest that the reader reads the statement of that lemma (Lemma 3.1) and then returns to the current discussion leading to Lemma 2.2. We call a triangulation $\mathcal{B}$ of S an *improvement* of $\mathcal{A}$ if

(i) $\mu(\mathcal{B}) < \mu(\mathcal{A})$, or
(ii) $\mu(\mathcal{B}) = \mu(\mathcal{A})$, every triangle abc in $\mathcal{B}$ with $\angle abc = \mu(\mathcal{B})$ is also a triangle in $\mathcal{A}$, and $\mathcal{B}$ has at least one fewer maximum angle than $\mathcal{A}$.

The next lemma asserts that the algorithm makes progress as long as the current triangulation is not yet a minmax angle triangulation. It does this by proving that there is at least one suitable edge qs. In its current version, the algorithm can be thought of as trying all possible edges going out of q, so if there exist edges qs that lead to an improvement of $\mathcal{A}$, then the algorithm finds one such edge.

LEMMA 2.2. *Assume that $\mathcal{A}$ is not yet a minmax angle triangulation. Then an iteration of the repeat-loop constructs an improvement of $\mathcal{A}$.*

Proof. Step (M1) of the repeat-loop finds a triangle pqr in $\mathcal{A}$ so that $\angle pqr = \mu(\mathcal{A})$. The main observation is that there is some edge qs that intersects pr and belongs to a minmax angle triangulation $\mathcal{T}$ of S. This is because $\mu(\mathcal{T}) < \mu(\mathcal{A})$ implies that $\angle pqr$ cannot exist in $\mathcal{T}$, and consequently, $pr \notin \mathcal{T}$ (by the previous lemma). Therefore, there exists a point $s \in S - \{p, q, r\}$ such that $qs \cap pr \neq \emptyset$ and qs is an edge of $\mathcal{T}$. With this edge qs, the cake cutting lemma (section 3) ensures that there are polygon triangulations of P and R such that the largest angle of any triangle within P and R is still smaller than $\angle pqr$. Section 4 shows that the ear cutting procedure of step (M2) indeed finds such a point s and produces triangulations $\mathcal{P}$ and $\mathcal{R}$ of P and R such that $\mu(\mathcal{P}), \mu(\mathcal{R}) < \angle pqr$. $\square$

The above two lemmas can now be used to analyze the running time of the algorithm. First, we address the number of iterations of the repeat-loop which is 1 plus the number of successful iterations of step (M2).

LEMMA 2.3. *The above algorithm reaches a minmax triangulation after at most $O(n^2)$ iterations of the repeat-loop.*

Proof. Each iteration produces a triangulation with a smaller maximum angle than before, or with fewer maximum angles of the same size. Since the number of different triangulations is finite an optimum must be reached. To get an upper bound on the number of iterations notice that the edge pr removed from $\mathcal{A}$ during some iteration will not reappear in the future. The claim follows because S allows only $\binom{n}{2}$ different edges. □

We are now ready to argue that the above algorithm runs in time $O(n^2 \log n)$ and space $O(n)$. There are two data structures needed for the algorithm. First, the quad-edge structure of Guibas and Stolfi [9] is used to represent $\mathcal{A}$; it permits common operations, such as removing an edge, adding an edge, and walking from one edge to the next, in constant time each. Second, the angles of $\mathcal{A}$ are stored in a priority queue that admits insertions, deletions, and finding the maximum. Standard implementations support each such operation in time $O(\log n)$, see e.g. [4]. The space needed for both data structures is $O(n)$.

With these preliminaries we can give the analysis of the algorithm. By Lemma 2.3, the number of times the priority queue is consulted to get a largest angle is $O(n^2)$, which implies that step (M1) takes total time $O(n^2 \log n)$. Section 4 will show that the ear cutting procedure performs only a total of $O(n^2)$ operations on the quad-edge structure, each in constant time, and only $O(n^2)$ insertions into and deletions from the priority queue, each in time $O(\log n)$. We conclude that the running time of the algorithm is $O(n^2 \log n)$ as claimed.

3. The Cake Cutting Lemma. The result of this section is a technical lemma which is nevertheless the heart of this paper. It assures that for some edge qs the generated regions, P and R, can be triangulated without angles that are too large. We first discuss the shape of these regions and then state and prove the lemma.

The regions P and R are generated in step (M2) of the algorithm by adding an edge qs and removing all edges that intersect qs. It follows that P (and by symmetry R) is very similar to a simple polygon, that is, it is simply connected and bounded by straight line edges. The only difference is that there can be edges surrounded by P on both sides; these are the edges contained in the interior of the closure of P (see Figure 1). To simplify the forthcoming discussion (and also in the implementation of the algorithm) we treat each such edge as if it consisted of two edges, one for each side. Effectively, this means that we can talk about P and R as if they were simple polygons.

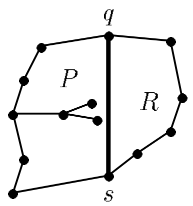

FIG. 1. *Regions P and R.*

With this note we now state and prove the cake cutting lemma. The intuition behind the proof is that we look at a piece of an optimal triangulation $\mathcal{T}$ and argue about its edges. Keep in mind, however, that during the algorithm we have no way of knowing what $\mathcal{T}$ really is; we only know that it exists.

LEMMA 3.1. *Let $\mathcal{T}$ be a minmax angle triangulation of S, $\mathcal{A}$ a triangulation of S with $\mu(\mathcal{A}) > \mu(\mathcal{T})$, pqr a triangle in $\mathcal{A}$ so that $\angle pqr = \mu(\mathcal{A})$, and qs an edge in*

$\mathcal{T}$ that intersects pr. Let P and R be the polygons generated by adding qs to $\mathcal{A}$ and removing all edges that intersect qs. Then there are triangulations $\mathcal{P}$ and $\mathcal{R}$ of P and R so that $\mu(\mathcal{P}), \mu(\mathcal{R}) < \mu(\mathcal{A})$.

Proof. We prove the claim for P; it follows for R by symmetry. Imagine we have $\mathcal{A}$ and $\mathcal{T}$ on separate pieces of transparent paper that we lay on top of each other so that the points match. Following step (M2) of the algorithm we add qs to $\mathcal{A}$ and remove intersecting edges from $\mathcal{A}$, thus creating P and R. Next, we clip everything outside P. In $\mathcal{A}$ only P without intersecting edges is left, and in $\mathcal{T}$ there will in general be edges that cut through P. By assumption, qs is also in $\mathcal{T}$ which implies that none of these edges meets qs. We define a *clipped edge* as a connected component of such an edge of $\mathcal{T}$ intersected with P. Since P is not necessarily convex, some clipped edges can belong to the same edge of $\mathcal{T}$. Given a point x on the boundary of P, let the *path* from x to q (or x to s) be the part of the boundary between x and q (or x and s) that does not contain qs. We have four classes of clipped edges xy, see Figure 2.

I. Both endpoints, x and y, are not vertices of P and thus lie on edges of P.
II. Both endpoints are vertices of P.
III. Endpoint x is a vertex of P, y is not, and y lies on the path from x to s.
IV. The same as class III except that y lies on the path from x to q.

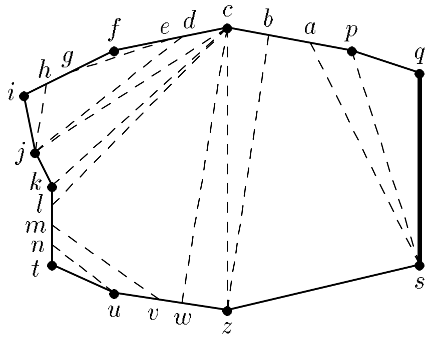

FIG. 2. *The class I edges in this example are eg and mv, the class II edges are cj, ck, cz and sp, the class III edges are cl and cw, and the class IV edges are jh, jd, un, zb and sa.*

At any vertex x of P, the clipped edges with one endpoint at x define angles at x which are all smaller than $\mu(\mathcal{A})$, because the clipped edges come from $\mathcal{T}$ and $\mu(\mathcal{T}) < \mu(\mathcal{A})$ holds by assumption. The only disadvantage of the partition of P defined by the clipped edges is that some of their endpoints lie on edges of P rather than at the vertices. We will now construct a triangulation of P based on the clipped edges. It proceeds step by step where each step either removes or rotates a clipped edge or introduces a new edge.

1. All class I edges are removed. This does not harm any angle.
2. All class II edges remain where they are.
3. Let xy be a class III edge with y on the edge $\alpha\beta$ of P, where α precedes β on the path from x to s. We replace xy by $x\beta$.

Note first that $x\beta$ is indeed a diagonal of P. Otherwise, it intersects the boundary of P, which implies that either x or β is not visible from qs. This is a contradiction to the way P is constructed. Note second that the angle at x that precedes xy in the counterclockwise order increases in step 3. Still, the angle formed by $x\beta$ is strictly contained in an angle at x in $\mathcal{A}$ because all edges of $\mathcal{A}$ that intersect P also intersect qs. It follows that the angle formed by $x\beta$ is smaller than $\mu(\mathcal{A})$. Another issue that comes up is that there can be class IV edges $x'y'$ with y' on the same edge $\alpha\beta$ of P — these edges now intersect $x\beta$. To remedy this situation we replace $x'y'$ by $x'x$. By the same argument as above $x'x$ is a diagonal of P, and the angle at x' that precedes $x'y'$ in the clockwise order and which increases as we replace $x'y'$ by $x'x$ remains smaller than $\mu(\mathcal{A})$.

4. If xy is a class IV edge with y on the edge $\alpha\beta$ of P, where α precedes β on the path from x to q, then we replace xy by $x\beta$.
5. After steps 1 through 4 we have a partial triangulation of P which we complete by adding edges arbitrarily. This finishes the construction of $\mathcal{P}$.

We have $\mu(\mathcal{P}) < \mu(\mathcal{A})$ since we started out with all angles smaller than $\mu(\mathcal{A})$, each time an angle increases it remains smaller than $\mu(\mathcal{A})$ as argued above, and step 5 decomposes angles thus creating only smaller angles. □

Remark. Note that the only property of $\mathcal{T}$ used in the proof of the cake cutting lemma is that $\mu(\mathcal{T}) < \mu(\mathcal{A})$. The lemma thus also holds if we replace $\mathcal{T}$ by an arbitrary triangulation $\mathcal{B}$ of S that satisfies $\mu(\mathcal{B}) < \mu(\mathcal{A})$. In fact, it suffices if $\mathcal{B}$ is an improvement of $\mathcal{A}$ and pqr is not a triangle in $\mathcal{B}$.

4. The Ear Cutting Procedure. The cake cutting lemma in section 3 shows that if $\mathcal{A}$ is not yet a minmax angle triangulation and qs is an edge in $\mathcal{T}$, chosen by the algorithm to improve $\mathcal{A}$, then there are triangulations of the generated polygons P and R with all angles smaller than $\angle pqr$. The two questions that remain are how to find such an edge qs and how to quickly triangulate P and R. One obvious way to find qs (not necessarily in $\mathcal{T}$ but in an improvement of $\mathcal{A}$) is to try all possible points s with $qs \cap pr \neq \emptyset$. For each such s we add qs to $\mathcal{A}$ and remove all edges that intersect qs. The thus created polygons P and R are triangulated with minimum largest new angle using dynamic programming. If the largest new angle is smaller than $\angle pqr$ we have an improvement of $\mathcal{A}$ and thus a desired qs.

Apparently, the implementation of an iterative step sketched in the above paragraph is rather inefficient. We improve the performance by a more clever way to search for an appropriate point s and by a fast procedure for triangulating P and R. The two tasks are woven together to the extent that it is not advisable to look at them as separate steps. For a chosen point s we attempt to triangulate P and R with all angles smaller than $\angle pqr$. If this fails we get some guidance where to look for a better point s. Following this guidance, a next point s is chosen so that we can reuse part of the work done during the unsuccessful triangulation attempt. The fundamental notion in all of this is that of an ear of a polygon triangulation.

4.1. Ears. An *ear* in a polygon triangulation is a triangle bounded by two polygon edges and one diagonal. It is easy to show that any triangulation of a simple polygon with more than three vertices has at least two ears [14].

In order to efficiently triangulate P and R, with all angles smaller than $\mu = \mu(\mathcal{A}) = \angle pqr$, we need two properties. The first guarantees that no expensive testing is necessary to recognize when an edge is a diagonal.

LEMMA 4.1. *Let P' be a polygon obtained from P by repeatedly removing ears not incident to qs. If a, b, c are three consecutive vertices of P' with $\{q, s\} \not\subseteq \{a, b, c\}$ and $\angle abc < \pi$ then ac is a diagonal of P'.*

Proof. By construction of P each of its vertices can be connected by a straight line segment within P to a point on qs. This property is maintained whenever we remove an ear not incident to qs, so it also holds for P'. In particular, it holds for the vertices a, b, and c of P'. The edge ac can avoid being a diagonal only if it intersects the boundary of P' (it cannot lie outside P' because $\angle abc < \pi$). But this contradicts the above property for either a or c or for both. $\square$

By symmetry, Lemma 4.1 also holds for R. It is now easy to identify ears because only one angle has to be checked. This is because the angles at a and c inside abc are always smaller than μ as they are properly contained in angles of $\mathcal{A}$. Thus, all three angles of abc are smaller than μ if and only if $\angle abc < \mu$.

The second property we need is that it does not matter which ears we remove, and in what sequence we remove them, as long as their angles are small enough. This property is implied by the following lemma whose proof is omitted because it is identical to that of the cake cutting lemma.

LEMMA 4.2. *Let P' be a polygon obtained from P by repeatedly removing ears not incident to qs. If qs is an edge of $\mathcal{T}$ then there exists a triangulation of P' without angles larger than or equal to μ.*

The two lemmas suggest that we triangulate P and R simply by repeatedly finding consecutive vertices a, b, c, with $\angle abc < \mu$, and removing the ear abc. We remark that this strategy can also be used to get an inductive proof of the cake cutting lemma. The next two subsections show how ear cutting and the search for an appropriate point s can be combined to yield an efficient implementation of an iterative step.

4.2. How to Cut. The way we search for a point s (section 4.3) guarantees a certain property of the polygons P and R which simplifies their triangulation by ear cutting. To be accurate we should mention that at the time we start the triangulation process for P and R, some ears will already have been removed as a result of earlier attempts to triangulate polygons generated for other points s. Consistently with our earlier notation, we therefore denote the two polygons that we attempt to triangulate by P' and R'. We state the mentioned property as an invariant of the algorithm after introducing some notation.

As justified above we pretend that P' and R' are simple polygons; by construction they share the edge qs. Let $k+2$ be the number of vertices of P' and $m+2$ the number of vertices of R', and label them consecutively as $q = p_0, p_1, \ldots, p_k, p_{k+1} = s$ and $q = r_0, r_1, \ldots, r_m, r_{m+1} = s$ (see Figure 3). Define $\phi_i = \angle p_{i-1}p_ip_{i+1}$ for $1 \leq i \leq k$ and $\rho_j = \angle r_{j-1}r_jr_{j+1}$ for $1 \leq j \leq m$. We can now state the property of P' and R'.

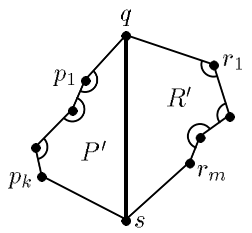

FIG. 3. *The circular arcs indicate angles that are known to be at least as large as μ.*

Invariant. $\phi_i \geq \mu$ for all $1 \leq i < k$ and $\rho_j \geq \mu$ for all $1 \leq j < m$.

This implies that p_{k-1}, p_k, s are the only three vertices that possibly define an ear of P' that is not incident to qs (provided $k > 1$) and has all three angles smaller than μ. Symmetrically, r_{m-1}, r_m, s are the only such three vertices of R'. If $\phi_k < \mu$ then $p_{k-1}p_ks$ is indeed such an ear and we can remove it from P'. This operation decreases ϕ_{k-1}, the angle at p_{k-1}, and leaves all other ϕ_i unchanged. Thus, P' still satisfies the invariant after setting $k := k-1$. Similarly, the invariant is maintained if we remove $r_{m-1}r_ms$ from R' and set $m := m-1$.

We now describe this process more formally as a procedure that alternates between removing an ear from P' and removing an ear from R'. It either completes its task of triangulating P' and R' or it stops because it encounters a situation where $\phi_k \geq \mu$ or $\rho_m \geq \mu$. To avoid repetition we separate out the code that tests an angle and removes an ear if the angle is small enough.

procedure CUTEARP'.
  if $\phi_k < \mu$ **then**
    if $k > 1$ **then** add the edge $p_{k-1}s$ to the triangulation **endif**;
    remove the triangle $p_{k-1}p_k s$ from P' and set $k := k - 1$
  else set *stop* := **true**
endif.

Similarly, we define a procedure CUTEARR' which either removes $r_{m-1}r_m s$ from R' or raises the flag by setting *stop* := **true**. The attempt to triangulate P' and R' first alternates between the two polygons and, if one polygon is successfully triangulated, attempts to complete the polygon that remains.

stop := **false**;
while $k > 0$ **and** $m > 0$ **and not** *stop* **do**
  CUTEARP'; **if not** *stop* **then** CUTEARR' **endif**
endwhile;
while $k > 0$ **and not** *stop* **do** CUTEARP' **endwhile**;
while $m > 0$ **and not** *stop* **do** CUTEARR' **endwhile**.

If the procedure finishes without raising the flag (*stop* = **false**) then we must have $k = m = 0$ and the triangulation is complete. Otherwise, the flag is raised either while testing P' or while testing R' (so we should really have used two flags to be able to distinguish the two cases — and we pretend we did).

Assume the flag was raised because of $\phi_k \geq \mu$. Let $\vec{qs}$ be the half-line that starts at q and goes through s, and let p' be the point among $p_1, \ldots, p_k$ so that $\angle p'qs$ is a minimum. Note that p' is not necessarily equal to p_k, but $p' = p_k$ if P' is convex. We have the following lemma which will be useful in searching for a new point s.

LEMMA 4.3. *Assuming $\phi_k \geq \mu$, there is no point $t \in S$ so that qt is an edge in a minmax angle triangulation $\mathcal{T}$ of S, $qt \cap p_k s \neq \emptyset$, and $qt \cap p's \neq \emptyset$.*

Proof. Suppose there is a point t that contradicts the assertion. Because $qt \cap p_k s \neq \emptyset$, this edge qt generates a polygon P'' so that $q = p_0, p_1, \ldots, p_k$ is a contiguous subsequence of its vertices (after removing appropriate ears). Let $p_{k+1}, \ldots, p_{k''}, p_{k''+1} = t$ be the other vertices of P''. By assumption we have $\angle p_{i-1}p_i p_{i+1} \geq \mu$ for $1 \leq i \leq k-1$. Furthermore, $\angle p_{k-1}p_k p_j \geq \mu$ for all $k+1 \leq j \leq k''+1$ because all these angles are larger than ϕ_k, the angle at p_k in P'. Hence, any attempt to triangulate P'' by removing ears (not incident to qs with angles all smaller than μ) must fail to cut off ears at p_i for all $1 \leq i \leq k$. $\square$

Remark. Similar as in the remark after the cake cutting lemma we can argue that Lemma 4.3 is also true if we replace $\mathcal{T}$ by an arbitrary triangulation that is an improvement of $\mathcal{A}$.

Lemma 4.3 suggests that the search for a new s continue between $\vec{qr'}$ and $\vec{qs}$ if the flag is raised while testing P', where r' is the counterpart of p' in R' and s is the old s. Thus, all ears removed from P' are safe and do not have to be considered again. However, all ears removed from R' have to be added back because they will intersect any future edge qs. Simultaneously, the value of m has to be adjusted. The amount of time needed to add these ears back in is proportional to the number of ears removed from P', because the ear cutting alternates between P' and R'. Symmetric actions are in order when the flag is raised while testing R'.

4.3. How to Search. Let us go back to the triangulation $\mathcal{A}$ of S that is not yet a minmax angle triangulation, and as usual let p, q, r be the points so that pqr is a triangle in $\mathcal{A}$ and $\angle pqr = \mu = \mu(\mathcal{A})$. The first vertex s that we test is the third vertex of the other triangle of pr (if no such triangle exists then pr is an edge of the convex hull of S and no appropriate point s exists). Thus, we add qs and remove pr. If the new angles at p and r are both smaller than μ, then we are done. If $\angle qps < \mu$ and $\angle qrs \geq \mu$ then, by Lemma 4.3, the edges we should test must intersect ps. Symmetrically, if $\angle qps \geq \mu$ and $\angle qrs < \mu$ then we must search for edges that intersect sr. If both angles are at least μ then no appropriate edge exists.

We now generalize and formalize this idea. For given polygons P' and R' we define vertices p' and r' as above, and we denote the open wedge between $\vec{qp'}$ and $\vec{qr'}$ by W. This wedge will get progressively smaller as we proceed with the search, and only points s within the wedge will be considered as endpoints of new edges qs. Initially, $p' = p$ and $r' = r$. We are now ready to describe the algorithm that searches for an appropriate point s.

Input. A triangulation $\mathcal{A}$ of S with maximum angle $\angle pqr = \mu = \mu(\mathcal{A})$.
Output. An improved triangulation or a message that the maximum angle cannot be decreased. In the latter case, the input triangulation is a minmax angle triangulation of S.
Define. $\text{THIRD}(a, b)$ is the vertex c of the triangle abc so that q and c lie on opposite sides of the line through a and b. If such a vertex does not exist, which is the case if ab is an edge of the convex hull of S, then $\text{THIRD}(a, b)$ is undefined. As before, W denotes the open wedge defined by p', q, and r'.

Initialize $k := 1$, $p_1 := p' := p$, $m := 1$, and $r_1 := r' := r$.
loop
 if $\text{THIRD}(p_k, r_m)$ is not defined **then**
  return the message that the maximum angle cannot be decreased and **stop**.
 else
  set $s := \text{THIRD}(p_k, r_m)$ and remove $p_k r_m$ from $\mathcal{A}$.
  if $s \in W$ **then**
   add qs to $\mathcal{A}$ and attempt the triangulation of P' and R' as described in §4.2.
   case 1. The attempt succeeds. Return the new triangulation and **stop**.
   case 2. The flag was raised while testing P'. Set $k := k + 1$ & $p_k := p' := s$.
   case 3. The flag was raised while testing R'. Set $m := m + 1$ & $r_m := r' := s$.
  else (i.e. $s \notin W$)
   if sr_m intersects W **then**
    set *stop* := **false**; **while not** *stop* **do** CUTEARP' **endwhile**;
    set $k := k + 1$ and $p_k := s$.
   else (i.e. sp_k intersects W)
    set *stop* := **false**; **while not** *stop* **do** CUTEARR' **endwhile**;
    set $m := m + 1$ and $r_m := s$.
   endif
  endif
 endif
forever.

We would like to point out a subtlety of the algorithm needed to prove its correctness. That is, the polygons P' and R' defined by any edge qs are obtained from $\mathcal{A}$ by removing *only* edges that intersect qs. Of course, some edges not in $\mathcal{A}$ have been added already to remove some ears. In other words, P' is the polygon P (as defined in section 2) with some ears removed, and the same is true for R' and R.

4.4. The Final Analysis. The running time of an iterative step (the above algorithm) is proportional to the number of removed ears. Because of the alternation between removing an ear from P' and one from R', only at most one more than half of the removed ears are added back to the polygon. This is also true if one polygon is completely triangulated while ears are still removed from the other polygon, because in this case only the ears of the former polygons need to be added back in, and their number is smaller than the number of ears cut off from the other polygon. It follows that the total number of removed ears is $\mathrm{O}(n)$. A single iteration therefore takes only $\mathrm{O}(n)$ time. Together with Lemma 2.3, which states that there are only $\mathrm{O}(n^2)$ iterations, this implies a cubic upper bound on the time-complexity of our algorithm (if implemented without priority queue).

Below we argue that its running time is actually $\mathrm{O}(n^2 \log n)$. To achieve this bound it is necessary to store the angles of the current triangulation in a priority queue, for otherwise finding all maximum angles costs time $\Omega(n^3)$. The crucial observation is that the time spent in an iterative step is proportional to the number of edges in the input triangulation that intersect the new edge qs. Each such edge has been removed and we argue that it will never be added again because every future triangulation will have an edge qt that intersects $p_k r_m$, the last edge before s. First note that every future triangulation is an improvement of $\mathcal{A}$. By Lemma 4.3 and the remark following it, every improvement of $\mathcal{A}$ has an edge qt in the final wedge W as maintained by the algorithm. Both, p_k and r_m, lie outside W (possibly on its boundary) and the edge $p_k r_m$ intersects W. The claim follows because all points of $W \cap S$ lie beyond $p_k r_m$ as seen from q. This implies the $\mathrm{O}(n^2 \log n)$ bound because we have only $\binom{n}{2} = \mathrm{O}(n^2)$ edges to work with. It should be noted that the maintenance of the priority queue storing the angles is the sole reason for the $\log n$ term in the $\mathrm{O}(n^2 \log n)$ bound; all other operations take total time $\mathrm{O}(n^2)$.

5. Extensions. We address two types of extensions of our algorithm for constructing minmax angle triangulations. The first extension is to the constrained case where the input consists of a set of n points plus some pairwise disjoint edges defined by the points that are required to be in the triangulation. The second extension discusses the optimization of the entire angle vector rather than just the maximum angle.

Only minor changes are necessary to adapt the algorithm presented in sections 2 and 4 to the constrained case. The most important change is that no prescribed edge will be removed to give way to searching for a new point s. This modification takes no extra time which implies the part of the main theorem that deals with prescribed edges.

Before we introduce angle vectors notice that for a given point set S all triangulations (whether constrained or not) have the same number of triangles and therefore the same number of angles. By Euler's formula for planar graphs the number of triangles is $t = 2n - h - 2$, where $n = |S|$ and h is the number of points of S that lie on the boundary of its convex hull. For any triangulation $\mathcal{A}$ of S we define its *angle vector* $V_{\mathcal{A}} = (\alpha_1, \alpha_2, \ldots, \alpha_{3t})$, with $\alpha_1 \geq \alpha_2 \geq \ldots \geq \alpha_{3t}$ the $3t$ angles of the t triangles. If $\mathcal{B}$ is another triangulation of S with angle vector $V_{\mathcal{B}} = (\beta_1, \beta_2, \ldots, \beta_{3t})$ we define

$V_{\mathcal{B}} < V_{\mathcal{A}}$ if there is an index $1 \leq j \leq 3t$ so that $\beta_i = \alpha_i$ for $1 \leq i < j$ and $\beta_j < \alpha_j$. For example, $V_{\mathcal{B}} < V_{\mathcal{A}}$ if $\mathcal{B}$ is an improvement of $\mathcal{A}$, but the reverse is not necessarily true.

The problem of finding a triangulation with minimum angle vector is at least as difficult as finding a minmax angle triangulation. If any two angles defined by three points of S each are different we can construct the minimum angle vector triangulation — which is unique in this case — as follows.

> First, construct a minmax angle triangulation, $\mathcal{T}_1$, and declare the three edges of the triangle that contains the maximum angle as prescribed. Second, construct a minmax angle triangulation $\mathcal{T}_2$ for the thus constrained input and introduce new constraints to enforce the second largest angle in future triangulations. Continue this way and construct triangulations $\mathcal{T}_3$, $\mathcal{T}_4$ and so on until the prescribed edges add up to a triangulation themselves. This triangulation minimizes the angle vector.

An $O(n^3 \log n)$ time bound for this algorithm is obvious because it just iterates the minmax angle triangulation algorithm a linear number of times. Even better, we have an $O(n^2 \log n)$ time bound if we use $\mathcal{T}_i$ as the input triangulation for the construction of $\mathcal{T}_{i+1}$. The improved bound follows because an edge once removed cannot appear in any future triangulation. We thus get the following result by the same argument as in section 4.4.

THEOREM 5.1. *Given a set of n points in the plane so that no angles defined by three points each are equally large, the triangulation that lexicographically minimizes the angle vector can be constructed in time $O(n^2 \log n)$ and space $O(n)$.*

Remark. In the presence of multiple angles it is not clear how to adapt the approach of this paper without requiring an exponential amount of time in the worst case. We pose the existence of a polynomial algorithm for minimizing the angle vector in the presence of multiple angles as an open problem. A case where multiple angles can be handled relatively easily is that of a simple polygon. The straightforward cubic time algorithm for minimizing the maximum angle, derived from the dynamic programming algorithm of Klincsek [10], can be extended to an $O(n^4)$ time algorithm for minimizing the angle vector as follows. Instead of characterizing a (partial) triangulation by its maximum angle we store its sorted angle vector. The best triangulation of a sequence of vertices is then selected on the basis of these vectors. The cubic time increases to $O(n^4)$ because comparing two angle vectors takes $O(n)$ time in the worst case, in contrast to constant time for comparing maximum angles.

6. Experimental Results. To demonstrate that the results of the preceding sections, which we believe are of theoretical interest, are significant also from a practical viewpoint, we implemented the algorithm along with a few other triangulation algorithms from the literature. Using these implementations, we perform a small-scale comparative study of the triangulations they produce. A more extensive study and complete description of the findings will be available as the master thesis of the third author. The difference between two triangulations is expressed in terms of their angles and edges (as in [16]).

The experimental study is based on implementations of four different triangulations algorithms. Three work by iterative improvement, and to construct an initial triangulation we use a plane-sweep strategy (see e.g. [6, section 8.3.1]). Triangulations constructed by plane-sweep are denoted by PS. The implementation of the edge-insertion algorithm of this paper minimizes the angle vector as discussed in section 5.

Its triangulations are referred to as MV. To avoid the difficulty that arises when two angles are equally large (see the remark at the end of section 5), we use a heuristic that breaks ties in a consistent manner. Delaunay triangulations, *DEL*, are constructed by flipping the diagonals of convex quadrilaterals as long as the smallest angle involved increases (see e.g. [11]). The third incremental improvement algorithm flips the diagonal of a convex quadrilateral if the largest of the six involved angles decreases. As shown in [15], this heuristic typically gets stuck in a local optimum depending on the initial triangulation as well as on the way the flips are scheduled. We use this algorithm to construct triangulations *FPD, FPN, FDD, FDN* where the middle letter distinguishes between *PS* and *DEL* as the initial triangulation, and the final letter distinguishes between deterministic (largest angle first) and 'non-deterministic' (first in first serve) scheduling.

The point sets chosen for our experimental study are drawn uniformly either inside a square or near a circle (see Figure 4). To allow for exact arithmetic all points are chosen on the integer grid. For each of various point set sizes, 30 experiments are carried out and average statistics is compiled.

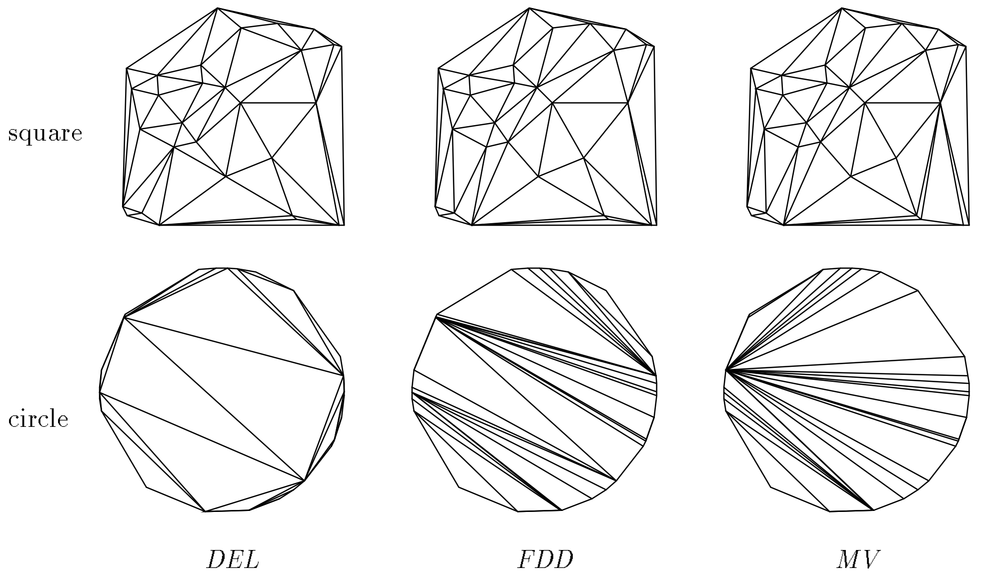

FIG. 4. *The Delaunay triangulation, DEL, a locally optimal triangulation, FDD, and the globally optimal triangulation, MV, for two small point sets.*

Table 1 compares triangulations and their quality. More specifically, it compares each triangulation $X \in \{PL,\ DEL,\ FPD,\ FPN,\ FDD,\ FDN\}$ with MV, the optimum triangulation. The parameter Δe gives the number of edges in X that are not in MV. The angle vectors of X and MV are compared using parameters $\angle_{eq}$, $\angle_{sm}$, and $\frac{\angle}{\angle_{MV}}$. Their meaning is that the $\angle_{eq}$ largest angles of X and MV are the same and that the next $\angle_{sm}$ largest angles are smaller for MV. $\frac{\angle}{\angle_{MV}}$ is the ratio between the $\angle_{eq}+1$ largest angle of each triangulation. The statistics shows that for points uniformly distributed in a square the edge-flip heuristic produces triangulations that come close to the optimum. Consistent with the findings reported in [16], *DEL* differs from MV by slightly less than 10% of its edges. This is in sharp contrast to the relative performance of the algorithms for points chosen on or close to a circle. In

TABLE 1
Comparison of MV with other triangulations.

	50 points (in square)				100 points (in square)				200 points (in square)			
	Δe (%)	$\angle_{eq}$	$\angle_{sm}$	$\frac{\angle}{\angle_{MV}}$	Δe (%)	$\angle_{eq}$	$\angle_{sm}$	$\frac{\angle}{\angle_{MV}}$	Δe (%)	$\angle_{eq}$	$\angle_{sm}$	$\frac{\angle}{\angle_{MV}}$
PS	51.9	0	91	1.020	62.5	0	190	1.012	70.8	0	391	1.006
DEL	7.3	7	29	1.033	8.0	8	53	1.018	8.8	11	66	1.008
FPD	1.5	97	5	1.011	3.1	36	42	1.022	3.3	24	66	1.011
FPN	1.6	89	6	1.012	3.2	35	32	1.022	3.2	26	68	1.012
FDD	2.4	54	7	1.015	2.2	74	13	1.017	2.5	28	30	1.010
FDN	2.6	29	7	1.017	2.5	91	15	1.022	2.9	28	34	1.010
	500 points (in square)				1000 points (in square)				50 points (near circle)			
PS	79.0	0	995	1.002	84.0	0	2001	1.001	45.7	0	33	1.058
DEL	8.8	17	185	1.005	9.1	21	231	1.002	46.5	0	47	1.072
FPD	3.3	40	195	1.005	3.0	55	438	1.003	43.1	1	26	1.046
FPN	3.3	41	151	1.006	3.2	57	454	1.003	43.7	1	26	1.045
FDD	2.7	45	92	1.009	2.7	62	200	1.007	42.0	2	24	1.039
FDN	3.0	43	91	1.008	3.0	60	201	1.006	40.6	2	22	1.035
	100 points (near circle)				200 points (near circle)				500 points (near circle)			
PS	43.5	4	79	1.003	47.1	7	104	1.001	44.5	2	176	1.00006
DEL	38.6	5	28	1.001	46.5	7	209	1.001	43.4	24	358	1.00003
FPD	39.4	16	34	1.005	46.4	15	94	1.004	43.4	77	215	1.00018
FPN	39.3	16	34	1.005	46.6	15	93	1.004	43.4	76	242	1.00018
FDD	38.2	18	28	1.004	46.1	16	88	1.004	43.3	80	335	1.00024
FDN	39.8	18	17	1.005	46.8	16	85	1.006	43.3	80	202	1.00011

this case, *DEL* and *MV* share very few non-convex hull edges. The edge-flip heuristic produces triangulations that are superior in terms of angles to *DEL*, but they hardly share any more edges with *MV*.

It is interesting to note that the amount of work needed to construct *MV* is far less for points in a square than for points near a circle. Table 2 shows the number of edges removed during the construction of *MV*. While the difference between the two point distributions is striking, the choice of the initial triangulation seems to have far less influence on the running time of the edge-insertion algorithm. In general, we observe that the edge-insertion algorithm is much faster on the average than expressed by the worst-case analysis in section 4. We would also like to remark that there are no polynomial time bounds known for the edge-flip heuristic used in our experimental study.

TABLE 2
The number of edges removed by the edge-insertion algorithm when it computes MV from either PS or DEL.

	square					circle			
	50 pts	100 pts	200 pts	500 pts	1000 pts	50 pts	100 pts	200 pts	500 pts
PS	240	647	1607	5067	11847	1301	2340	17136	63003
DEL	153	390	946	2887	6658	1303	2276	16660	58934

7. Conclusions. The main result of this paper is an $O(n^2 \log n)$ time algorithm for constructing a minmax angle triangulation of a set of n points in the plane, with or without prescribed edges. This seems fairly efficient considering that it is the first polynomial time algorithm for the problem and that it somehow avoids to look at all the $\binom{n}{3}$ angles defined by the n points. On the other hand, our algorithm is a factor n slower than the best algorithms for constructing Delaunay triangulations, at least in the worst case. We thus pose the question whether a minmax angle triangulation can be constructed in $o(n^2 \log n)$ time.

In the non-degenerate case where no two angles defined by three points each are equal, the algorithm can be extended to compute the triangulation that lexicographically minimizes the sorted vector of angles. The running time is till $O(n^2 \log n)$ in the worst case, and our experiments indicate that the average run-time is significantly less.

A problem related to minimizing the maximum angle is to construct a triangulation that minimizes the number of obtuse angles. It seems that the edge-insertion scheme does not work for this criterion. The problem thus remains open for point sets, although dynamic programming yields a cubic time algorithm if the input is a simple polygon. Still, the authors of this paper believe that the edge-insertion scheme is more generally applicable and plan to further investigate this paradigm.

Acknowledgement. The second author thanks Professor C. L. Liu for his constant supports and encouragement.

REFERENCES

[1] I. Babuška and A. K. Aziz, *On the angle condition in the finite element method*, SIAM J. Numer. Anal., 13 (1976), pp. 214–226.

[2] R. E. Barnhill and F. F. Little, *Three- and four-dimensional surfaces*, Rocky Mountain J. Math., 14 (1984), pp. 77–102.

[3] J. Cavendish, *Automatic triangulation of arbitrary planar domains for the finite element method*, Internat. J. Numer. Methods Engin., 8 (1974), pp. 679–696.

[4] T. H. Cormen, C. E. Leiserson and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Mass., 1990.

[5] B. Delaunay, *Sur la sphère vide*, Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk, 7 (1934), pp. 793–800.

[6] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, Germany, 1987.

[7] S. J. Fortune, *A sweepline algorithm for Voronoi diagrams*, Algorithmica, 2 (1987), pp. 153–174.

[8] J. A. Gregory, *Error bounds for linear interpolation on triangles*, in The Math. of Finite Element and Applications II, J. R. Whiteman, ed., Academic Press, 1975, pp. 163–170.

[9] L. J. Guibas and J. Stolfi, *Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams*, ACM Trans. Graphics, 4 (1985), pp. 74–123.

[10] G. T. Klincsek, *Minimal triangulations of polygonal domains*, Annals Discrete Math., 9 (1980), pp. 121–123.

[11] C. L. Lawson, *Generation of a triangular grid with applications to contour plotting*, Jet Propul. Lab. Techn. Memo. 299, 1972.

[12] C. L. Lawson, *Software for C^1 surface interpolation*, in Mathematical Software III, J. R. Rice, ed., Academic Press, 1977, pp. 161–194.

[13] D. T. Lee and A. K. Lin, *Generalized Delaunay triangulations for planar graphs*, Discrete Comput. Geom., 1 (1986), pp. 201–217.

[14] G. H. Meisters, *Polygons have ears*, Amer. Math. Monthly, 82 (1975), pp. 648–651.

[15] G. M. Nielson, *An example with a local minimum for the minmax ordering of triangulations*, Manuscript, Lawrence Livermore Nat. Lab., Livermore, California, 1987.

[16] G. M. Nielson and R. Franke, *Surface construction based upon triangulations*, in Surfaces in Comp. Aided Geom. Design, R. E. Barnhill and W. Boehm, eds., North-Holland Publishing, 1983, pp. 163–177.

[17] D. A. Plaisted and J. Hong, *A heuristic triangulation algorithm*, J. Algorithms, 8 (1987), pp. 405–437.

[18] M. J. D. Powell and M. A. Sabin, *Pairwise quadratic approximation on triangles*, ACM Trans. Math. Software, 3 (1977), pp. 316–325.

[19] F. P. Preparata and M. I. Shamos, *Computational Geometry – an Introduction*, Springer-Verlag, New York, 1985.

[20] R. Seidel, *Constrained Delaunay triangulations and Voronoi diagrams with obstacles*, in "1978-1988, 10-Years IIG" a report of the Inst. Informat. Process., Techn. Univ. Graz, Austria, 1988, pp. 178–191.

[21] M. I. Shamos and D. Hoey, *Closest point problems*, Sixteenth Annual IEEE Symposium on the Foundations of Computer Science, 1975, pp. 151–162.

[22] R. Sibson, *Locally equiangular triangulations*, Comput. J., 21 (1978), pp. 243–245.

[23] G. Strang and G. Fix, *An Analysis of the Finite Element Method*, Prentice-Hall, Englewood Cliffs, NJ, 1973.