Discrete Comput Geom 4:423–432 (1989)

Discrete & Computational
Geometry

A Fast Las Vegas Algorithm for Triangulating a Simple Polygon

Kenneth L. Clarkson,[1] Robert E. Tarjan,[1,2,*] and Christopher J. Van Wyk[1]

[1] AT&T Bell Laboratories, Murray Hill, NJ 07974, USA

[2] Department of Computer Science, Princeton University, Princeton, NJ 08544, USA

Abstract. We present a randomized algorithm that triangulates a simple polygon on n vertices in $O(n \log^* n)$ expected time. The averaging in the analysis of running time is over the possible choices made by the algorithm; the bound holds for any input polygon.

1. Introduction

To *triangulate* a simple polygon on n vertices, we add to it $n-3$ line segments between vertices (diagonals) that partition its interior into triangles. Determining the complexity of triangulating a simple polygon is an outstanding open problem in computational geometry.

Previous work on the triangulation problem has concentrated on finding fast deterministic algorithms to solve it. Garey *et al.* gave an algorithm to triangulate an n-gon in $O(n \log n)$ time [GJPT]. Tarjan and Van Wyk devised a much more complicated algorithm that runs in $O(n \log \log n)$ time [TV].

In this revised and expanded version of our conference paper [CTV], we present a randomized algorithm that triangulates a simple polygon on n vertices in $O(n \log^* n)$ expected time. Our algorithm uses the following key ideas:

- divide and conquer;
- the "random sampling" paradigm [C1], [CS], [ES], [HW];
- the vertical visibility decomposition determined by a set of noncrossing line segments in the plane: each endpoint of a line segment defines the vertical boundaries of two generalized trapezoids, generated by vertical rays that

* Research partially supported by the National Science Foundation under Grant No. DCR-8605962.

are extended up and down from the endpoint until they encounter other line segments [CI], [FM];

- Jordan sorting [FNTV], [HMRT]: given the intersections of two simple curves A and B in the order in which they occur along curve A, find the order in which they occur along curve B.

Except for random sampling, these are the same ideas used in the algorithm of Tarjan and Van Wyk [TV]. The addition of random sampling both simplifies the algorithm and improves the time bound.

The rest of this paper is organized as follows. Section 2 describes the vertical visibility decomposition in more detail, and explains its role in the solution of the triangulation problem. Section 3 outlines our algorithm; Sections 4 and 5 describe at greater length how to perform some of its steps. Section 6 contains an analysis of the running time of the algorithm. Throughout Sections 2–6 we assume that no two vertices of the polygon being triangulated share the same x-coordinate; Section 7 describes two ways to cope with input that does not have this nice property. Section 8 explains how to use our algorithm to test whether the input polygon is simple. Section 9 concludes with open problems.

2. Vertical Visibility Decomposition

Given a set S of noncrossing line segments, a line segment e and an endpoint v of another line segment are mutually *vertically visible* if the vertical line segment from v to e does not intersect any other element of S. Each endpoint of a line segment in S can see at most one line segment in S above it and another line segment in S below it, so the number of different vertically visible pairs of edges and vertices is at most $4|S|$.

The boundary of a simple polygon is a set of noncrossing line segments. In this case, the set of vertically visible pairs of vertices and edges composes the *total vertical visibility* information of the polygon. If the open vertical line segment between vertex v and edge e lies inside the polygon, the pair is *internally vertically visible*; if it lies outside the polygon, the pair is *externally vertically visible*; if it lies on the boundary of the polygon, the pair is *vertically visible along the boundary*. To simplify the presentation of the algorithm, we assume that no two vertices of the polygon have the same x-coordinate; thus, there will be no vertex-edge pairs that are vertically visible along the boundary. Section 7 describes two ways to remove this restriction while preserving the bound on expected running time.

The algorithm in this paper computes the total vertical visibility information of the polygon. This contrasts with earlier algorithms [CI], [FM], [TV], which compute only the internal vertical visibility information. Given the internal vertical visibility information for a polygon, we can triangulate it in linear time [CI], [FM]. Since the internal vertical visibility information can be deduced from the total vertical visibility information, we rely on this linear-time reduction to triangulate the polygon given the information computed by our algorithm.

Given a set S of noncrossing line segments in the plane, its vertical visibility decomposition $\boldsymbol{T}(S)$ is a set of open regions. To construct $\boldsymbol{T}(S)$, extend a ray

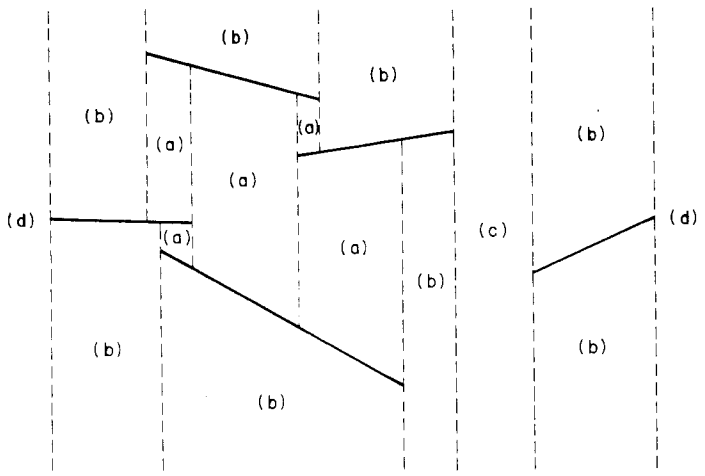

Fig. 1. Five line segments and their vertical visibility decomposition.

vertically from each endpoint of each line segment in S until it hits another line segment in S or it sees infinity. Thus, each vertical segment bounding a region in $T(S)$ contains an endpoint of a segment in S. There are four kinds of regions in $T(S)$:

(a) those bounded by two portions of line segments in S and two vertical line segments;
(b) those bounded by one portion of a line segment in S and two vertical rays;
(c) those bounded only by two vertical lines;
(d) those bounded only by one vertical line.

Figure 1 shows a vertical visibility decomposition whose regions have been labeled with their types according to the above list. Regions of type (a) are trapezoids (or triangles), so $T(S)$ is often called a *trapezoidal decomposition* of the plane.

If S contains s noncrossing line segments, we can compute $T(S)$ in $\Theta(s \log s)$ time. The lower bound follows because we can use the vertical visibility decomposition to sort. The upper bound can be achieved either in the worst case by a plane-sweep algorithm [PS], or on average by a randomized algorithm [CS].

3. Outline of the Algorithm

The algorithm computes the total vertical visibility information of polygon P by computing the total vertical visibility information about a random subset of the edges of P, then using that random subset to partition P into pieces on which the algorithm is applied recursively.

The general step of the algorithm accepts as input a sequence S of line segments that compose a simple polygon P_S. At the top level, the edges of the input polygon P are processed by the algorithm as sequence S. Each recursive call of the

algorithm is on a sequence that defines a polygon bounded by pieces of edges of P together with vertical line segments that correspond to mutually vertically visible points on two edges of P.

We note that the input sequence S never contains two consecutive vertical segments. The assumption that no two vertices of P have the same x-coordinate guarantees that there are no vertical edges in P, so this is certainly true at the top level. At each recursive call, each vertical line segment in S is preceded and followed by nonvertical line segments that are pieces of edges of P.

The following is the general recursive step of the algorithm:

1. [Select random sample] Let S' be the subset of nonvertical line segments in S; let $s' = |S'|$. Choose a random sample $R \subset S'$ of size r (where r is a function of s', to be determined in Section 6).
2. [Compute vertical visibility decomposition] Compute $\boldsymbol{T}(R)$. Only vertices in R that are original vertices of P need participate in this computation: any other endpoints of line segments lie at either the extreme left or right side of P_S, and whatever piece of P they see is already known.
3. [Break polygon edges at vertical visibility segments] Chop each line segment in S each time it crosses the boundary of a region in $\boldsymbol{T}(R)$. Let I be the number of such intersections. If ever I is about to exceed c_{total}, or some region in $\boldsymbol{T}(R)$ is about to intersect more than $c_{\max}$ segments, immediately restart the recursive step at step 1. (Both c_{total} and $c_{\max}$ depend on r and s'; we determine their exact values in Section 6. The value of I also appears in the running-time analyses in Sections 4 and 6.)
4. [Jordan sort] For each region $F \in \boldsymbol{T}(R)$, Jordan sort the intersection points found in step 3 around the boundary of F. Compute the "family trees" associated with the Jordan sorting.
5. [Reconstruct subpolygons and recur] Decompose each region in $\boldsymbol{T}(R)$ into a set of simple polygons, using the family trees computed in step 4. Apply the algorithm recursively to each polygon that contains at least one vertex of P_S that does not lie on a vertical visibility edge.

When all recursive calls are completed, the algorithm has computed a finer partition of the plane than $\boldsymbol{T}(P)$. The internal vertical visibility information for P, and hence a triangulation of P, can be computed from this partition.

4. Breaking the Polygon Boundary at Vertical Visibility Segments

We define the *neighbors* of a region F in a vertical visibility decomposition $\boldsymbol{T}(R)$ to be the regions that we can reach by crossing a vertical edge of F. For our application, the nonvertical edges of F will always be edges of the polygon, which we would never want to cross; thus we do not consider the regions above and below F with which it shares a nonvertical boundary to be neighbors of F.

If the input polygon contains two or more vertices with the same x-coordinate, then a region in the vertical visibility decomposition could have more than two neighbors on each side. We describe in Section 7 how to deal with this anomaly,

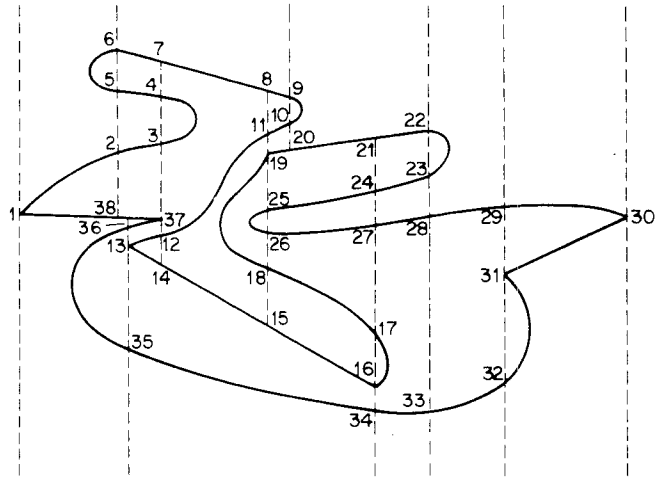

Fig. 2. A simple curve through the line segments of Fig. 1. The endpoints of the original segments are (1, 37), (6, 9), (13, 16), (19, 22), and (30, 31). In a polygon the curves would be polygonal chains.

but for now we assume that the polygon is in general position so this does not happen. That is, each region has at most four neighbors, and we can move from a region to one of its neighbors in $O(1)$ time.

To perform step 3, traverse the boundary of P_S as it moves from region to neighboring region in $\boldsymbol{T}(R)$. (Figure 2 shows how P_S might meander through $\boldsymbol{T}(R)$.) Since each region has at most four neighbors, we can perform step 3 in $O(I + s')$ time.

5. Jordan Sorting and Polygon Reconstruction

In step 4 we need to sort the points at which P_S intersects each region $F \in \boldsymbol{T}(R)$ according to their ordering around the boundary of F, given their order along P_S. We use the simplified Jordan-sorting algorithm of Fung *et al.* [FNTV] to sort the sequence of intersection points found in step 3 into the order in which they lie along the boundary of F in time linear in the number of points.

To apply the Jordan-sorting algorithm as stated [FNTV], we must transform the boundary of each region into a straight line, while preserving the connections defined by the points of intersection between P_S and F. The appropriate transformation depends on the type of the region:

(a) split the trapezoid at the midpoint of its lower nonvertical edge and unfold it into a straight line;
(b) unfold the semi-infinite trapezoid to a straight line;
(c) connect the two bounding lines by a line segment that lies entirely above all pieces of P_S, then unfold the boundary to a straight line;
(d) the boundary is a straight line already.

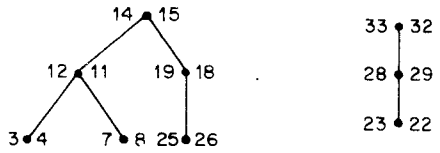

Fig. 3. Representative family trees for the curve shown in Fig. 2. The tree on the left is the inner family tree for the region whose corners are 7, 8, 15, and 14. The tree on the right is the inner family tree for the region of type (c) in Fig. 1.

The *inner family tree* of P_S with respect to F shows how polygon pieces nest with respect to the transformed region boundary. Figure 3 shows the inner family trees for two regions in Fig. 2. The Jordan-sorting algorithm [FNTV] produces family trees as part of its operation.

Each node in the inner family tree, together with any children it may have, defines a subpolygon of F. To perform step 5, we traverse the inner tree of each region $F \in \boldsymbol{T}(R)$, constructing the subpolygons of F and passing nontrivial ones to recursive instances of the algorithm. All of this can be done in time linear in the size of the tree.

6. Expected Running Time

In this section we derive a bound on the expected running time of the algorithm if it is applied to a polygon P with n sides. The analysis relies on general theorems about random sampling in computational geometry. We define enough notation here to state the theorems we need; readers who wish to see the definitions and theorems stated in full generality should consult Clarkson and Shor's paper [CS].

As in Sections 2-5, S is a set of s line segments in the plane. The set $S^{(b)}$ contains all subsets of S of at most b elements. The binary operator δ relates each member of $S^{(b)}$ to one or more regions that belong to a set $\boldsymbol{F}$ of regions in the plane. In our application, the set $\boldsymbol{F}$ is the set of all planar regions that can be represented as the intersection of at most four open halfplanes, and relation δ is defined as follows: if X is a set of line segments, then for any $F \in \boldsymbol{T}(X)$, $F\delta X$. The set $\boldsymbol{F}_S$ is the set of all generalized trapezoidal regions defined by any collection of at most four line segments in S:

$$\boldsymbol{F}_S = \{F \mid F \in \boldsymbol{F},\ F\delta X,\ X \in S^{(4)}\}.$$

For any $F \in \boldsymbol{F}_S$, let X_F be a set in $S^{(4)}$ of minimum cardinality such that $F\delta X_F$. If no two endpoints of line segments in S share the same x-coordinate, then X_F is unique for each $F \in \boldsymbol{F}_S$, and we say that relation δ is *functional.*

Let R be any subset of S. For any $F \in \boldsymbol{F}_R$, $|F|$ denotes the number of elements

of S that have nonempty intersection with F. We define

$$T_c(R) = \sum_{F \in \boldsymbol{T}(R)} |F|^c.$$

Thus, $T_0(R)$ is the number of trapezoids in $\boldsymbol{T}(R)$, and $T_1(R)$ is the number of segments into which $S \backslash R$ would be divided were we to chop segments in $S \backslash R$ along the boundaries of regions in $\boldsymbol{T}(R)$.

Let $T_c(r)$ be the expected value of $T_c(R)$ where $R \subset S, |R| = r$, and each R is equally likely to be chosen. Finally, let

$$\tau_0(r) = \max_{1 \le z \le r} T_0(r).$$

The definition of $\boldsymbol{T}(R)$ implies that $T_0(r)$, and hence $\tau_0(r)$, are both at most $4r$.

We have the following theorem:

Theorem. *Assume S is a set of noncrossing line segments of size s, and R is a random subset of S of size r; let $\boldsymbol{T}(R)$ be the vertical visibility decomposition that R defines on the plane. There exist constants k_{total} and $k_{\max}$ such that with probability at least $1/2$, the following two conditions hold simultaneously:*

(1) $T_1(R) \le k_{\text{total}} s$.
(2) *For each* $F \in \boldsymbol{T}(R), |F| \le k_{\max}(s/r) \log r$.

Proof. We appeal to Corollary 3.8 of [CS], which says that if δ is functional and if K exists such that $\boldsymbol{F}_S$ contains at most $K\binom{n}{b}$ elements, then with probability $3/4 - 1/e^2$, both of the following conditions hold:

$$T_1(R) \le O(n/r)\tau_0(r); \tag{A}$$

for some constant $z = O(\log r) + 2 + \log_e K$,

$$\max_{F \in \boldsymbol{T}(R)} |F| \le zs/r. \tag{B}$$

We can satisfy the hypotheses of the corollary by taking $K = 4$. Since $\tau_0(r) \le 4r$, equation (A) implies that there exists a constant k_{total} such that $T_1(R) \le k_{\text{total}} s$, which is condition (1). Equation (B) implies that there exists a constant $k_{\max}$ such that for each $F \in \boldsymbol{T}(R), |F| \le k_{\max}(s/r) \log r$, which is condition (2).

Since $3/4 - 1/e^2 > 1/2$, the probability that conditions (1) and (2) hold simultaneously is at least $1/2$, as required. □

In the algorithm we take $c_{\text{total}} = k_{\text{total}} s'$ and $c_{\max} = k_{\max}(s'/r) \log r$. The theorem implies that with probability at least 1/2, step 3 will "succeed," and not restart the recursive step with another random sample; in other words, the expected number of times we need to restart step 3 is $O(1)$. If we take $r = s'/\log s'$, then condition (2) further implies that the maximum depth of recursion is $O(\log^* n)$.

Next we compute the work done during the recursive steps of the algorithm. A vertex sends out visibility segments above and below during exactly one recursive step of the algorithm, when it is an endpoint of an edge that is chosen for the random sample at that step; this is the only time that a vertex can cause the boundary of P to be cut into pieces.

Condition (1) implies that over the course of the entire algorithm, the total number of pieces into which the boundary of P can be cut is $k_{\text{total}} n$. Since the boundary can contain at most one vertical segment for each nonvertical segment, the number of different vertical segments considered during the algorithm is also at most $k_{\text{total}} n$. Therefore, at a single level of recursion, the algorithm considers at most $2k_{\text{total}} n$ pieces of the boundary. Since each piece can serve as the boundary of at most two regions, and the subpolygons processed at a single level of recursion have disjoint interiors, they contain at most $4k_{\text{total}} n$ pieces.

At each level of the recursion, the vertical visibility decomposition can be computed in $O(r \log r) = O(n)$ time, and the boundary can be chopped, the crossing points Jordan sorted, and the pieces reconstructed into subpolygons in $O(I + n)$ time, which is $O(n)$ by the preceding observations. Thus a single level of recursion can be performed in $O(n)$ time, and all $O(\log^* n)$ levels of recursion can be completed in $O(n \log^* n)$ time.

7. Dealing with Singularities

Figure 4(a) shows how vertically aligned vertices could cause a visibility region to have more than four neighbors. This could cause a problem in the analysis of the running time, since it could take longer than $O(1)$ time to move from region to neighboring region. A conceptually simple way to deal with the singular case is to apply a random rotation to the original input polygon whenever we detect

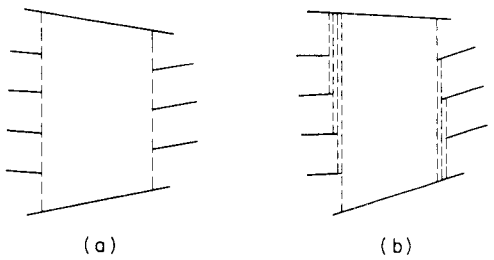

Fig. 4. The region in (a) has more than two neighbors on each side. By rotating it slightly, as in (b), we can construct a vertical visibility decomposition in which no region has more than two neighbors.

a trapezoid that has more than four neighbors. With probability 1, such a rotation avoids any vertical alignment of vertices.

By careful consideration of Fig. 4, however, we can avoid performing any random rotation at all. Let S be the sequence of segments that gave rise to the region in Fig. 4, and let S' be S rotated slightly so that no vertices are vertically aligned. Figure 4(b) depicts the effect of this rotation on the region in Fig. 4(a): a layer of thin regions appears on either side of the original region. When we trace the boundary of S' through an edge of the region in Fig. 4(b), the sequence of steps is equivalent to performing a linear search in clockwise order through the multiple vertices on an edge of the unrotated region in Fig. 4(a). Since the same time bound holds whether the algorithm runs on S or S', the algorithm can simply use clockwise linear search to perform step 3 even when the input polygon is not in general position; in order for the running-time analysis to apply to this version of the algorithm, the count of intersections I must be incremented at each step in searches along region edges, as well as at each intersection of P_S and $T(R)$.

8. Simplicity Testing

The algorithm described in this paper decomposes the plane into regions and considers all parts of the input polygon that lie in each region. If the polygon is not simple, then the algorithm will detect a self-crossing of the boundary during one of the following operations:

- the random sample may contain crossing edges, which will be detected during the computation of the vertical visibility decomposition of the sample;
- the boundary may cross one edge of the random sample, which will be detected during step 3;
- the Jordan sorting in step 4 may fail because the pieces of the polygon do not nest properly.

Thus, this algorithm can test whether an input polygon is simple in $O(n \log^* n)$ expected time.

9. Remarks

A nonrecursive version of this algorithm, which uses a single level of random sampling to partition the polygon and then the straightforward algorithm on each piece, would run in $O(n \log \log n)$ time [C2].

The foremost remaining open problem is to produce a triangulation algorithm that runs in $o(n \log \log n)$ time or in $o(n \log^* n)$ expected time. A related problem is to devise a parallel algorithm whose time-processor product is $o(n \log n)$ [ACG]; some progress has been reported on this problem [CCT].

References

[ACG] M. J. Atallah, R. Cole, and M. T. Goodrich, Cascading divide-and-conquer: a technique for designing parallel algorithms, *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, 1987, pp. 151–160.

[CI] B. Chazelle and J. Incerpi, Triangulation and shape complexity, *ACM Transactions on Graphics*, **3** (1984), 135–152.

[C1] K. L. Clarkson, New applications of random sampling in computational geometry, *Discrete and Computational Geometry*, **2** (1987), 195–222.

[C2] K. L. Clarkson, Applications of random sampling in computational geometry, II, *Proceedings of the Fourth Annual Symposium on Computational Geometry*, 1988, pp. 1–11.

[CCT] K. L. Clarkson, R. Cole, and R. E. Tarjan, private communication.

[CS] K. L. Clarkson and P. W. Shor, Applications of random sampling in computational geometry, II, *Discrete and Computational Geometry*, this issue, 387–421.

[CTV] K. L. Clarkson, R. E. Tarjan, and C. J. Van Wyk, A fast Las Vegas algorithm for triangulating a simple polygon, *Proceedings of the Fourth Annual Symposium on Computational Geometry*, 1988, pp. 18–22.

[ES] P. Erdos and J. Spencer, *Probabilistic Methods in Combinatorics*, Academic Press, New York, 1974.

[FM] A. Fournier and D. Y. Montuno, Triangulating simple polygons and equivalent problems, *ACM Transactions on Graphics*, **3** (1984), 153–174.

[FNTV] K. Y. Fung, T. M. Nicholl, R. E. Tarjan, and C. J. Van Wyk, Simplified linear-time Jordan sorting and polygon clipping, *ACM Transactions on Graphics*, submitted.

[GJPT] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan, Triangulating a simple polygon, *Information Processing Letters*, **7** (1978), 175–180.

[HW] D. Haussler and E. Welzl, ε-nets and simplex range queries, *Discrete and Computational Geometry*, **2** (1987), 127–151.

[HMRT] K. Hoffman, K. Mehlhorn, P. Rosenstiehl, and R. Tarjan, Sorting Jordan sequences in linear time using level-linked search trees, *Information and Control*, **68** (1986), 170–184.

[PS] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

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

Received July 25, 1988, and in revised form March 25, 1989.