A FAST LAS VEGAS ALGORITHM FOR TRIANGULATING A SIMPLE POLYGON

Kenneth L. Clarkson
Robert E. Tarjan
Christopher J. Van Wyk

CS-TR-132-88

January 1988

A Fast Las Vegas Algorithm for Triangulating a Simple Polygon

Kenneth L. Clarkson

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

*Robert E. Tarjan**

Department of Computer Science
Princeton University
Princeton, New Jersey 08544
and
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

Christopher J. Van Wyk

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

We show how to use random sampling to triangulate a simple polygon in nearly linear expected time.

1. Introduction

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

Previous work on the triangulation problem has concentrated on finding fast deterministic algorithms to solve it. Garey, Johnson, Preparata, and Tarjan 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 paper we present an algorithm that triangulates a simple polygon in $O(n \log^* n)$ expected time. Our algorithm uses several key ideas:

- divide and conquer;

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

- the "random sampling" paradigm [C87], [C88], [ES], [HW];
- the decomposition of the plane by a set of line segments into trapezoids determined by what is vertically visible from each endpoint of each line segment [CI], [FM];
- Jordan sorting [HMRT]: given the intersections of two simple curves A and B in the order in which they occur along A, find the order in which they occur along B.

2. Outline of the Algorithm

The following is the general recursive step of our algorithm:

The input is a sequence S of line segments that compose a simple polygon P_S.

1. Let S' be the subset of non-vertical line segments in S; let s' be the size of S'. Choose a random sample $R \subset S'$ of size r (r is a function of s', to be determined below).
2. Compute the vertical visibility decomposition $\mathbf{V}(R)$ of the plane according to R. If any region in $\mathbf{V}(R)$ shares a vertical edge with more than two other regions, apply a random rotation to the original set of vertices and restart the algorithm.
3. Trace the boundary of P_S, recording each intersection of P_S with an edge in $\mathbf{V}(R)$. Let I be the number of intersections of P_S with an edge in $\mathbf{V}(R)$. If, during this traversal, I is about to exceed c_{total}, or some region in $\mathbf{V}(R)$ is found 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 below.)
4. For each region in $\mathbf{V}(R)$, Jordan sort the intersection points found in Step 3 around its boundary.
5. Decompose each region in $\mathbf{V}(R)$ into a set of simple polygons (We discuss in Section 4 how to do this using the family trees produced by Jordan sorting in Step 4.); the algorithm recurs on all polygons that contain at least one vertex of P_S that does not lie on a vertical visibility edge.

3. Vertical Visibility Decomposition

To construct the vertical visibility decomposition $\mathbf{V}(S)$ of the plane given a set S of line segments, we extend a ray vertically from each endpoint of each line segment until we hit another line segment in S or we reach infinity. Thus, each vertical segment in $\mathbf{V}(S)$ contains an endpoint of a segment in S. This process produces the following kinds of regions:

(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 by two vertical lines;

(d) those bounded 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 $\mathbf{V}(S)$ is often called a *trapezoidal decomposition* of the plane.

Our interest in the vertical visibility decomposition stems from the observation that computing the vertical visibility of the edges of a simple polygon is linear-time equivalent to triangulating the polygon [CI], [FM]. When all recursive steps of our algorithm have finished, we actually have a vertical visibility decomposition of the edges in hand, so by this observation we have solved the original triangulation problem.

In Step 2 of the algorithm we require that no region share either of its vertical edges with more than two other regions. This certainly happens if no two vertices have the same x-coordinate; when necessary, the random rotation in Step 2 achieves this with probability 1. When this condition holds, it is possible to move from region to neighboring region in the decomposition in $O(1)$ time.

4. Jordan Sorting and Polygon Reconstruction

In Step 4 we need to sort the points at which P_S intersects each region $Q \in \mathbf{V}(R)$ according to their ordering around the boundary of Q. (Figure 2 shows how P_S might meander around R.) Step 3 produces those points in order along P_S. The original Jordan sorting algorithm [HMRT] can be used to sort those points along the boundary of Q in time proportional to the number of points.

During its operation, the Jordan sorting algorithm produces two family trees for each region. The outer tree depicts the way pieces of P_S nest outside the boundary of Q, and the inner tree depicts the way they nest inside the boundary of Q. (Figure 3 shows two family trees from regions in Figure 2.) Thus, each node in the inner tree, together with any children it may have, defines a subpolygon of Q. To perform Step 5, we merely traverse the inner tree of each region $Q \in \mathbf{V}(R)$, constructing the subpolygons of Q and passing appropriate ones to recursive instances of the algorithm.

5. Expected Running Time

In this section we derive a bound on the expected running time of the algorithm on a polygon P with n sides. The proof relies on probabilistic bounds provided by the following theorem:

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

(1) The sum, over all regions $Q \in \mathbf{V}(R)$, of the number of line segments of S that intersect Q is bounded by $k_{\text{total}} s$.

(2) For all $Q \in \mathbf{V}(R)$, the number of line segments of S that intersect Q is bounded by $k_{\max}(s/r)\log r$.

Proof. Both conditions are consequences of general theorems about random sampling in computational geometry. To prove condition (1), we appeal to Theorem 3.2 of [C88]:

> Let S be a set of s regions in E^d, and let $\mathbf{F}$ be another set of regions. Let a, b, and c be fixed nonnegative integers, and let m be a fixed positive real value. Let ν_k, for $1 \le k \le a$, be a collection of mappings from S^b to $\mathbf{F}$. For R a random subset of S of size r, let
>
> $$\mathbf{F}_R = \{\nu_k(\hat{R}) \mid 1 \le k \le a,\ \hat{R} \in R^b\},$$
>
> and define a random variable
>
> $$T_m(R) = \sum_{\substack{A \in \mathbf{F}_R \\ A \cap R = \varnothing}} |A \cap S|^m,$$
>
> where $|A \cap S|$ is understood to mean the number of elements of S that intersect the interior of region A. If the expected value of $T_0(R)$ is $O(r^c)$, then the expected value of $T_m(R)$ is $O((s/r)^m r^c)$.

To apply this theorem to prove condition (1), we take $d = 2$, S to be a set of line segments, $\mathbf{F}$ to be the set of all regions of the plane formed by the intersection of at most four half-planes, $b = 4$, $c = 1$, and $m = 1$. Next we define a collection of a functions ν_k, for $1 \le k \le a$; each of these functions maps a four-tuple (s_1,s_2,s_3,s_4) of members of S to a region in $\mathbf{F}$. We define ν_1 to be the function that maps (s_1,s_2,s_3,s_4) to the half-plane bounded by a vertical line through the left endpoint of s_1, ν_2 to be the function that maps (s_1,s_2,s_3,s_4) to the half-plane bounded by a vertical line through the right endpoint of s_1, ν_3 to be the function that maps (s_1,s_2,s_3,s_4) to the vertical swath defined by the left endpoints of s_1 and s_2, and so on; it is clear that with a finite number (a) of such ν_k we can insure that $\mathbf{F}_R$ contains all regions that could belong to the vertical visibility decomposition induced by any subset of S.

The expected value of $T_0(R)$ is the expected number of regions in $\mathbf{F}_R$ whose interiors do not meet R; this is surely $O(r)$, the number of regions in $\mathbf{V}(R)$. The theorem guarantees that the expected value of $T_1(R)$ is $O(s)$; since $T_1(R)$ is the sum over all regions $Q \in \mathbf{V}(R)$ of the number of line segments of S that intersect Q, this proves that the asserted value of k_{total} exists.

Condition (2) follows similarly from Corollary 4.2 in [C87]. ■

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 a step 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. First, note that 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 chosen in Step 1; this is the only time that a vertex can cause the boundary of P to be cut into pieces. Condition (1) implies that the total number of pieces into which the boundary of P can be cut in this way is $k_{\text{total}} n$. Moreover, since the boundary can contain at most one vertical segment for each non-vertical segment, the number of different vertical segments considered in the algorithm is at most $k_{\text{total}} n$. The subpolygons processed at the ith level of recursion are area-disjoint, so they contain at most $4k_{\text{total}} n$ pieces.

At each level of the recursion, Step 2 can be performed in $O(r \log r) = O(n)$ time [PS], and Steps 3, 4, and 5 can be performed in $O(I)$ time, which is $O(n)$ by the observations in the preceding paragraph. Thus a single level of recursion takes $O(n)$ time. Since there are $O(\log^* n)$ levels of recursion, the total running time is $O(n \log^* n)$.

References

[C87] K. L. Clarkson, "New applications of random sampling in computational geometry," *Discrete and Computational Geometry*, 2 (1987), 195-222.

[C88] K. L. Clarkson, "Applications of random sampling in computational geometry, II," submitted to *Fourth Annual ACM Symposium on Computational Geometry.*

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

[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.

[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*, to appear.

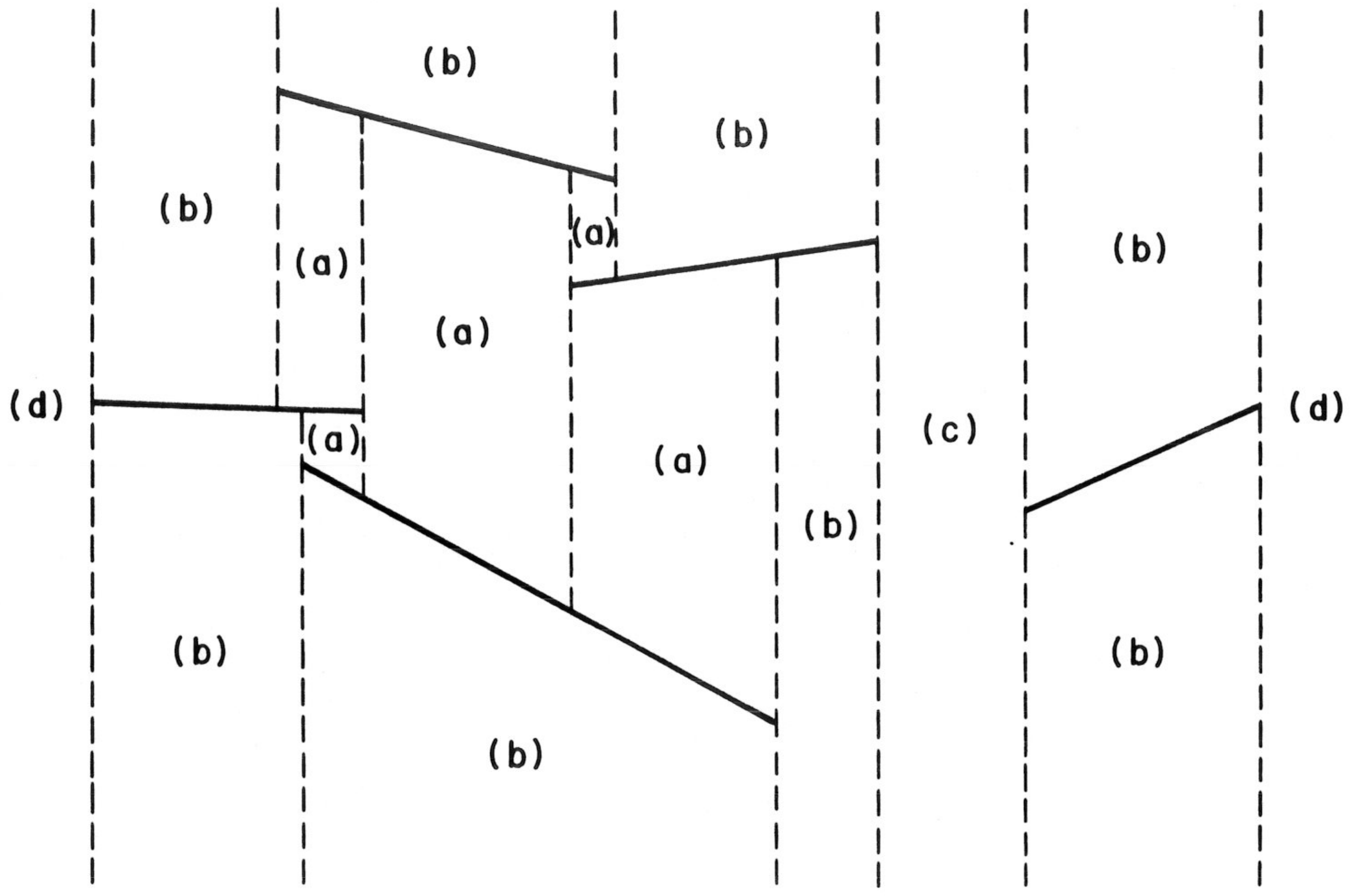

Figure 1. Five line segments and their vertical visibility decomposition.

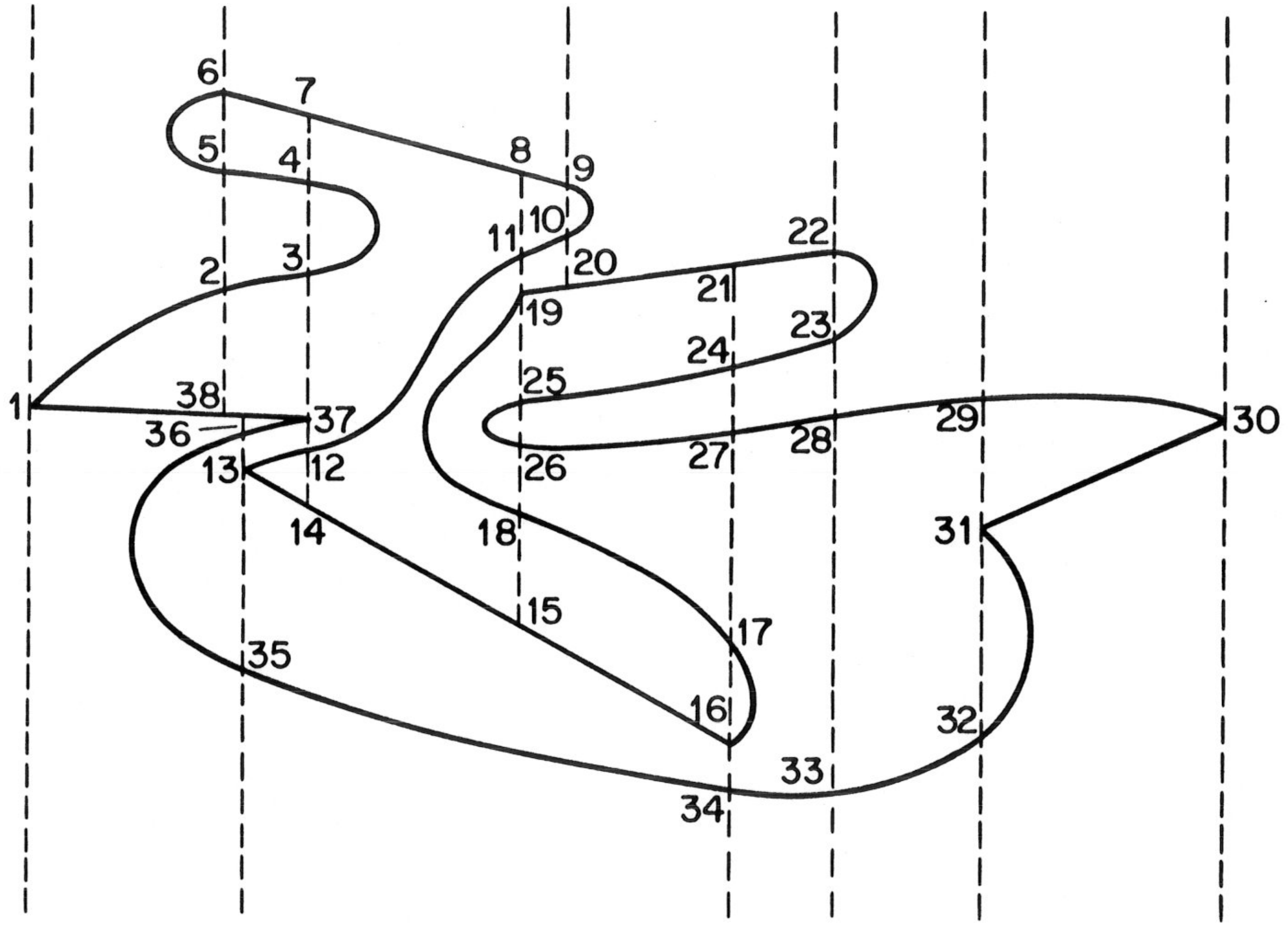

Figure 2. A simple curve through the line segments of Figure 1. The endpoints of the original segments are (1,37),(6,9),(13,16),(19,22), and (30,31). In a polygon the curves would be polygonal chains.

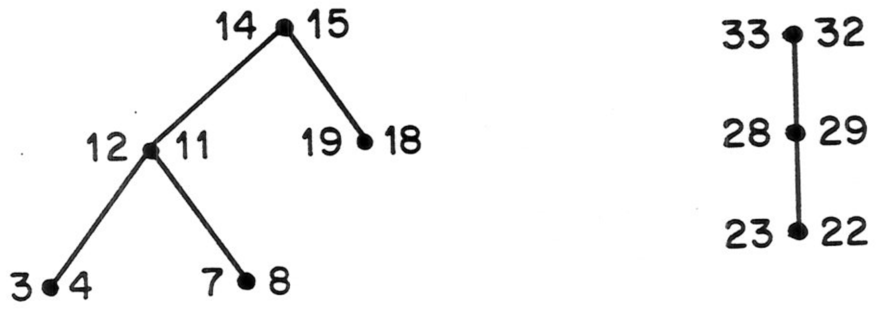

Figure 3. Representative family trees for the curve shown in Figure 2. The tree on the left is the inner family tree for the region whose corners are 7, 8, 15, and 14. The tree on the right is the inner family tree for the region of type (c) in Figure 1.