A FAST LAS VEGAS ALGORITHM FOR TRIANGULATING A SIMPLE POLYGON

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

CS-TR-157-88

May 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 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 (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, Johnson, Preparata, and Tarjan gave an algorithm to triangulate an n-gon in $O(n \log n)$ time [GJPT]. Tarjan and Van Wyk

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

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 [C87], [C88], [ES], [HW];
- the vertical visibility decomposition determined by a set of non-crossing line segments in the plane: each endpoint of a line segment defines the vertical boundaries of two generalized trapezoids, generated by vertical rays that are extended up and down from the endpoint until they encounter other line segments [CI], [FM];
- Jordan sorting [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.

2. Vertical Visibility Decomposition

Given a set S of non-crossing 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 $O(|S|)$.

The boundary of a simple polygon is a set of non-crossing 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 vertical line segment between vertex v and edge e lies inside the polygon, the pair is *internally vertically visible*, otherwise it is *externally vertically visible*.

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.

To construct the vertical visibility decomposition $\mathbf{V}(S)$ of the plane given a set S of non-crossing line segments, extend a ray vertically from each endpoint of each line segment until it hits another line segment in S or it sees infinity. Thus, each vertical segment in $\mathbf{V}(S)$ contains an endpoint of a segment in S. This process produces four 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.

Given a set S of s non-crossing line segments, we can compute $\mathbf{V}(S)$ in $\Theta(s \log s)$ time. The lower bound follows because one 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 [C88].

3. Outline of the Algorithm

The general step of the algorithm accepts as input a sequence S of line segments that compose a simple polygon P_S. No two consecutive segments in S are vertical.

At the top level, the edges of the 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 vertically visible points on two edges of P.

The following is the general recursive step of the algorithm:

1. Let S' be the subset of non-vertical 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 below).
2. Compute the vertical visibility decomposition $\mathbf{V}(R)$ of the plane defined by R.
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 edges in $\mathbf{V}(R)$. If, during this traversal, I is about to exceed c_{total}, or some region in $\mathbf{V}(R)$ 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 below.)
4. For each region in $\mathbf{V}(R)$, Jordan sort the intersection points found in Step 3 around the boundary of $\mathbf{V}(R)$. Compute the ''family trees'' associated with the Jordan sorting.
5. Decompose each region in $\mathbf{V}(R)$ 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.

The value of I in Step 3 appears below in the analysis of the running time of the algorithm.

Some of the vertices of P_S are original vertices of P, while others may be corners introduced by chopping the boundary of P_S in Steps 3 and 4. The latter vertices can only lie on vertical visibility edges, however, so they do not cause the boundary of P_S to be chopped during subsequent recursive calls.

4. Tracing the Boundary through a Vertical Visibility Decomposition

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

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 below how to deal with this anomaly, but for now we shall 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, we follow the boundary of P_S as it moves from region to neighboring region in $\mathbf{V}(R)$. (Figure 2 shows how P_S might meander through $\mathbf{V}(R)$.) Since each region has at most four neighbors, we can perform Step 3 in $O(I)$ time.

5. 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, given their order along P_S. We use the Jordan sorting algorithm of Hoffman, Mehlhorn, Rosenstiehl, and Tarjan [HMRT] to sort the sequence of intersection points found in Step 3 into the order in which they lie along the boundary of Q in time linear in the number of points.

To apply the Jordan sorting algorithm as stated [HMRT], 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 Q. The appropriate transformation depends on the type of the region:

(a) split the trapezoid at the midpoint of its lower non-vertical 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.

The *inner family tree* of P_S with respect to Q shows how polygon pieces nest with respect to the transformed region boundary. Figure 3 shows the inner family trees for two regions in Figure 2. The Jordan sorting algorithm [HMRT] 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 Q. To perform Step 5, we traverse the inner tree of each region $Q \in \mathbf{V}(R)$, constructing the subpolygons of Q and passing non-trivial ones to recursive instances of the algorithm. All of this can be done in linear time.

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 proof relies on probabilistic bounds. To state them, we first define some notation: For a region Q and a set of line segments S, let $\#(Q,S)$ be the number of segments in S that intersect the interior of Q. Then we have the following theorem:

Theorem. Suppose S is a set of non-crossing 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 on the plane that R defines. There exist constants k_{total} and $k_{\max}$ such that with probability at least $1/2$, the following two conditions hold simultaneously:

(1) $\sum_{Q \in \mathbf{V}(R)} \#(Q,S) \leq k_{\text{total}} s$.

(2) For each $Q \in \mathbf{V}(R)$, $\#(Q,S) \leq k_{\max}(s/r)\log r$.

Proof. This theorem is a consequence of general theorems about random sampling in computational geometry. First, we appeal to Theorem 3.2 of [C88] to conclude that the expected value of $\sum_{Q \in \mathbf{V}(R)} \#(Q,S)$ is at most $(k_{\text{total}}/4)s$. Since the random variable $\sum_{Q \in \mathbf{V}(R)} \#(Q,S)$ takes on non-negative values, we can use Markov's inequality [R] to conclude that condition (1) holds with probability at least $3/4$.

To prove that condition (2) holds with probability at least $3/4$, we begin with the following restatement of Corollary 4.2 of [C87]:

Let S and $\mathbf{F}$ be sets of regions in the plane, with $|S| = s$. For fixed integers i and n, let ν_k, $1 \leq k \leq n$, be a mapping from S^i to $\mathbf{F}$. Let R be a random sample of S, of size r, and let $\mathbf{F}_R$ denote

$$\{\nu_k(\hat{R}) \mid 1 \leq k \leq n,\ \hat{R} \in R^i\},$$

the images of R^i under the ν_k's. Then for fixed i and n,

$$\text{Prob}\{\exists A \in \mathbf{F}_R \text{ with } \#(A,R) = 0 \text{ and } \#(A,S) > \alpha s\} \leq O(r^i)(1-\alpha)^{r-i}.$$

For suitable $\alpha = O(\log r / r)$, this probability is no more than $1/4$.

When we apply this result, S will be the set S of non-crossing line segments, and $\mathbf{F}$ will be the set of generalized trapezoidal regions in the plane. In the definitions of functions ν_k, we refer to regions that lie above, below, and to the left or right of segments; the words "above" and "below" are used in a stronger sense than the words "left" and "right," as follows. Let $z = (x,y)$ be a point and let s be a segment from (x_1,y_1) to (x_2,y_2) with $x_1 < x_2$. Then z lies to the *right* (*left*) of s if $x \geq x_2$ ($x \leq x_1$); the points that lie to the right of a segment form a halfplane. On the other hand, z lies *above* (*below*) s if z lies in the upper (lower) halfplane defined by the line through s, and z lies neither to the left nor the right of s; the points that lie above a segment form a semi-infinite trapezoid. We take $i = 4$ and $k = 6$, defining the following functions to construct regions of various types:

(a) let $\nu_1(s_1,s_2,s_3,s_4)$ be the maximal trapezoid that lies below s_1, to the right of s_2, to the left of s_3, and above s_4;

(b) let $\nu_2(s_1,s_2,s_3,s_4)$ be the maximal semi-infinite trapezoid that lies below s_1, to the right of s_2, and to the left of s_3, and let $\nu_3(s_1,s_2,s_3,s_4)$ be the semi-infinite trapezoid whose interior lies to the right of s_2, to the left of s_3, and above s_4;

(c) let $\nu_4(s_1,s_2,s_3,s_4)$ be the maximal bi-infinite swath that lies to the right of s_2 and to the left of s_3;

(d) let $\nu_5(s_1,s_2,s_3,s_4)$ be the maximal half-plane that lies to the right of s_2, and $\nu_6(s_1,s_2,s_3,s_4)$ be the maximal half-plane that lies to the left of s_3.

A region $Q \in \mathbf{F}_R$ with $\#(Q,R) = 0$ is a generalized trapezoid in $\mathbf{V}(R)$. The corollary implies that with probability at least 3/4, $\#(Q,S) = O((s/r)\log r)$ for each $Q \in \mathbf{F}_R$.

Since each condition holds with probability at least 3/4, the probability that both conditions hold 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 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 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 non-vertical 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, Step 2 can be performed in $O(r \log r) = O(n)$ time, and Steps 3, 4, and 5 can be performed 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 vertices that are vertically aligned. With probability one, such a rotation avoids any vertical alignment of vertices.

By careful consideration of Figure 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 Figure 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 Figure 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 Figure 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 Figure 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 $\mathbf{V}(R)$.

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 $O(n \log^* n)$ time.

9. Open Problems

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].

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, 151-160.

[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,'' *Discrete and Computational Geometry*, submitted.

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

[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, to appear.

[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.

[R] M. M. Rao, *Probability Theory with Applications*, Academic Press, Orlando, Florida, 1984.

[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.

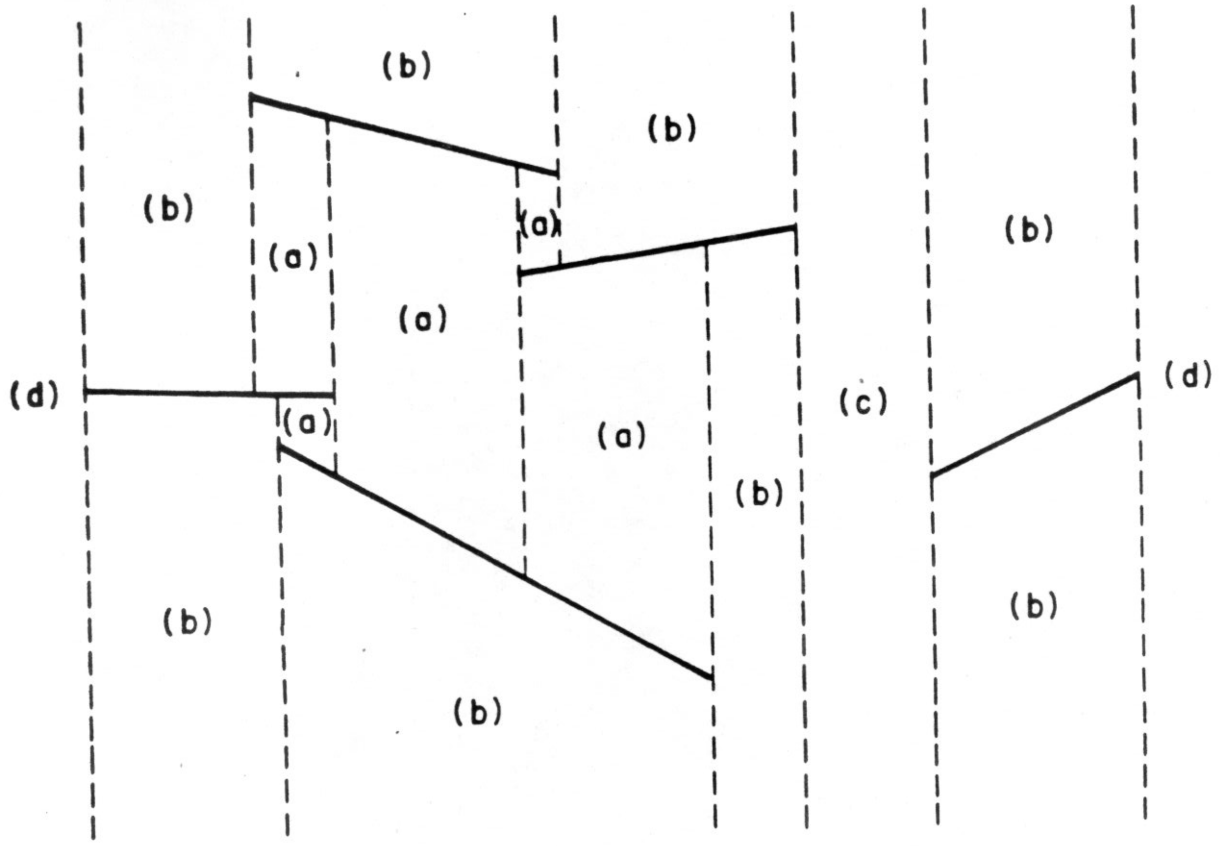

Figure 1. Five line segments and their vertical visibility decomposition.

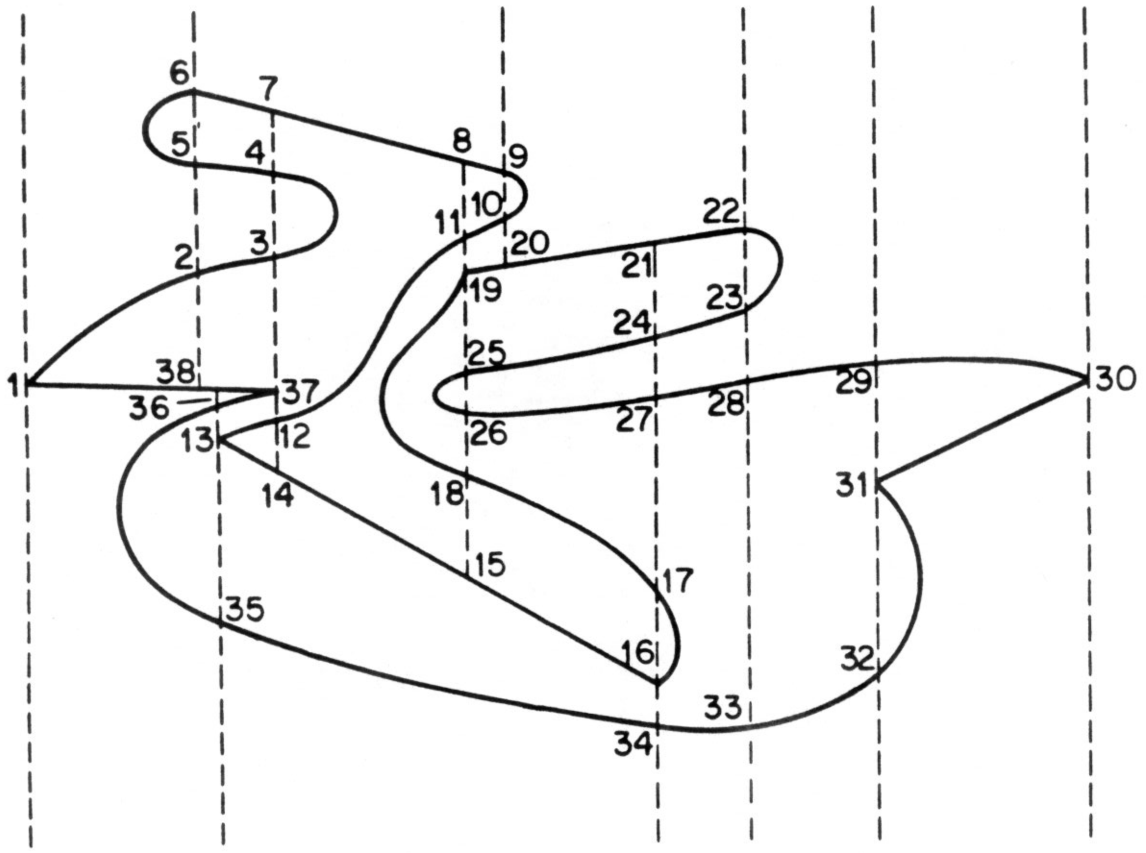

Figure 2. A simple curve through the line segments of Figure 1. The endpoints of the original segments are (1,37), (6,9), (13,16), (19,22), and (30,31). In a polygon the curves would be polygonal chains.

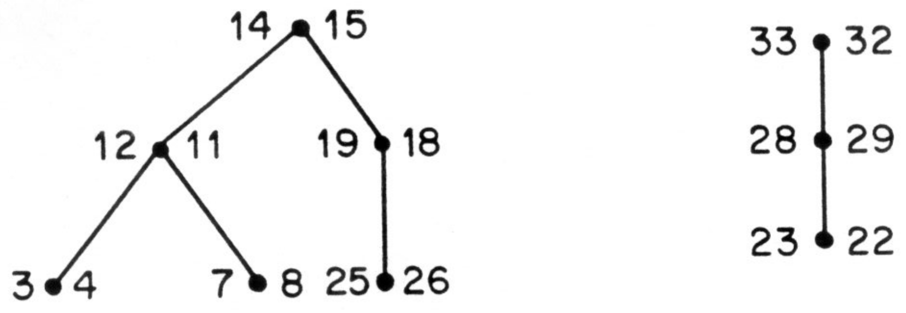

Figure 3. Representative family trees for the curve shown in Figure 2. The tree on the left is the inner family tree for the region whose corners are 7, 8, 15, and 14. The tree on the right is the inner family tree for the region of type (c) in Figure 1.

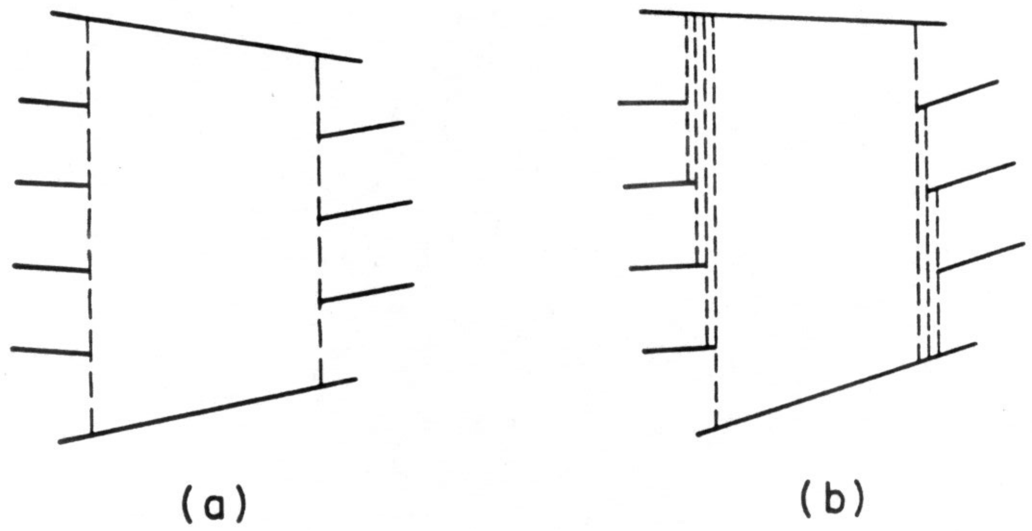

Figure 4. The region in (a) has more than two neighbors on each side. By rotating it slightly, as in (b), we can construct a vertical visibility decomposition in which no region has more than two neighbors.