TRIANGULATING A SIMPLE POLYGON *

Michael R. GAREY, David S. JOHNSON
Bell Laboratories, Murray Hill, NJ 07974, USA

Franco P. PREPARATA **
University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Robert E. TARJAN ***
Stanford University, Stanford, CA 94305, USA

Received 8 November 1977

Closest-point problems, computational geometry, polygon, triangulation

1. Introduction

Let P be a simple n-sided polygon in the plane, not necessarily convex. A *diagonal* of P is a line segment joining two non-adjacent vertices of P. We consider here the problem of *triangulating* P, that is, of finding $n-3$ diagonals which intersect neither each other nor the boundary of P and which divide the interior of P into $n-2$ triangles

Applications of triangulation arise in closest point problems [1,2] and in evaluating functions by interpolation [3,6]. An elegant algorithm for triangulating a set S of n points in the plane has been given by Shamos [4,5], using the Voronoi diagram of S, and it requires only $O(n \log n)$ steps on a random access machine with real-number arithmetic. The problem of triangulating a simple polygon appears to be more difficult. Indeed, no previously known method improves on the $O(n^2)$ brute force algorithm. (The method of [4,5] is not sufficient, since it cannot ensure that the boundary edges of P will belong to the constructed triangulation.) We shall give an algorithm which triangulates an arbitrary n-vertex simple polygon in time $O(n \log n)$.

Our algorithm depends heavily upon results in [1]. We use the "regularization" procedure of that paper to preprocess the given polygon, subdividing it into polygons having a very simple structure. This special structure permits us to complete the triangulation using a straightforward algorithm for triangulating each of the simpler polygons. Section 2 of this paper presents the algorithm for triangulating such a specially-structured polygon, and Section 3 discusses the property of "regularization" which allows us to use this algorithm for triangulating an arbitrary simple polygon.

* The results in this paper were obtained independently by the third author and by the first, second, and fourth authors.

** Departments of Electrical Engineering and of Computer Science, and the Coordinated Science Laboratory. Research of this author supported in part by National Science Foundation grant MCS76-17321 and the Joint Services Electronics Program Contract DAAB-07-72-C-0259.

*** Computer Science Department. Research of this author supported in part by National Science Foundation grant MCS75-22870, Office of Naval Research Contract N00014-76-C-0330, and Bell Laboratories.

2. Triangulating a monotone polygon

Let P be a simple polygon in the plane having boundary vertices $p_1, p_2, \ldots, p_n$. We choose the y-axis as a preferred direction and assume throughout this paper that no two vertices of P have the same y-coordinate (this assumption is not crucial to the results, but serves

merely to simplify the presentation). Suppose without loss of generality that p_1 is the vertex with the largest y-coordinate. The polygon P is said to be *monotone* if there is a vertex p_k such that the vertices $p_1, p_2, \ldots, p_k$ are in decreasing order by y-coordinate and the vertices $p_k, p_{k+1}, \ldots, p_n, p_1$ are in increasing order by y-coordinate. In this section we describe an algorithm for triangulating a monotone polygon.

To triangulate such a polygon P, we first sort its vertices in order of decreasing y-coordinate. Because P is a monotone, this can be done in time $O(n)$ by merging the two sequences of vertices given by the definition. Suppose $q_1, q_2, \ldots, q_n$ is the resulting sequence, with $q_1 = p_1$. It is easy to prove that the following property (*) is equivalent to the monotone property:

(*) For $2 \leqslant i \leqslant n$, there is some $j < i$ such that q_i and q_j are adjacent on the boundary of P.

We process the boundary vertices in the order $q_1, q_2, \ldots, q_n$. When we process a vertex, we may add certain diagonals to the polygon. Each added diagonal cuts a triangle off of the polygon and leaves a polygon of one less side still to be triangulated. We shall call the part of the original polygon still to be triangulated the "remaining polygon".

During the processing, we maintain a stack that contains all vertices which have been processed so far and which lie on the boundary of the remaining polygon. The contents $x_1, x_2, \ldots, x_i$ of the stack satisfy the following properties during the processing:

(i) $x_1, x_2, \ldots, x_i$ are in decreasing order by y-coordinate;

(ii) $x_1, x_2, \ldots, x_i$ form a chain on the boundary of the remaining polygon;

(iii) the internal angles of the remaining polygon at $x_2, x_3, \ldots, x_{i-1}$ are at least 180°; and

(iv) the next vertex to be processed is adjacent to either x_1 or x_i (or both).

Here is the triangulation algorithm.

Initial step: Place the first two vertices q_1 and q_2 in the stack.

General step: Let $x_1, x_2, \ldots, x_i$ denote the stack contents and x the next vertex to be processed.

(a) If x is adjacent to x_1 but not x_i, add diagonals $(x, x_2), (x, x_3), \ldots, (x, x_i)$. Replace the stack contents by x_i, x.

(b) Otherwise, if x is adjacent to x_i but not x_1, repeat the following until $i = 1$ or the internal angle at x_i is at least 180°: add diagonal (x_{i-1}, x), delete x_i from the stack, and replace i by $i - 1$. When this step is no longer applicable, add x to the top of the stack.

(c) Otherwise, if x is adjacent to both x_1 and x_i, add diagonals $(x, x_2), (x, x_3), \ldots, (x, x_{i-1})$ and stop. Vertex x is the last one to be processed.

Repeat the general step until case (c) applies.

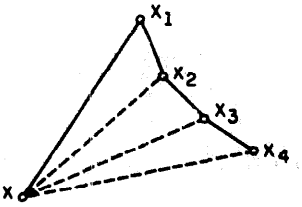

Fig. 1. Case (a) of the monotone polygon triangulation algorithm. Dashed lines are added diagonals. The remaining polygon is formed from the old polygon by replacing the polygonal chain x, x_1, x_2, x_3, x_4 by the line segment x, x_4.

Figs. 1 through 3 illustrate cases (a), (b) and (c). The correctness of the algorithm depends upon the fact that the added diagonals lie completely inside the polygon. Consider for example the diagonal (x, x_2) constructed in case (a). None of the vertices $x_3, x_4, \ldots, x_i$ can lie inside or on the boundary of the triangle formed by x_1, x_2, and x because the internal angles of at least 180° at $x_2, x_3, \ldots, x_{i-1}$ force x and $x_3, x_4, \ldots, x_i$ to lie on opposite sides of the line through x_1, x_2 (Fig. 4). No other vertex of the polygon lies inside or on the boundary of this triangle because all such vertices have smaller y-coordinates than x. No point inside the triangle can be exterior to the polygon, because the polygon would then have to pass through the interior of the triangle, and at least one of its vertices would be

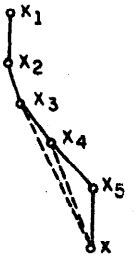

Fig. 2. Case (b) of the monotone polygon triangulation algorithm. The remaining polygon is formed from the old polygon by replacing the polygonal chain x_3, x_4, x_5, x by x_3, x.

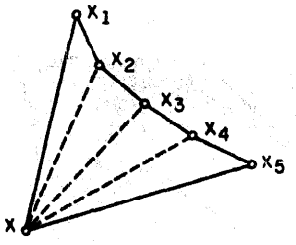

Fig. 3. Case (c) of the monotone polygon triangulation algorithm.

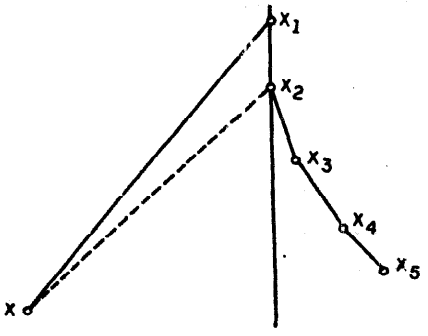

Fig. 4. Vertices x and x_3, x_4, x_5 lie on opposite sides of the line through x_1 and x_2.

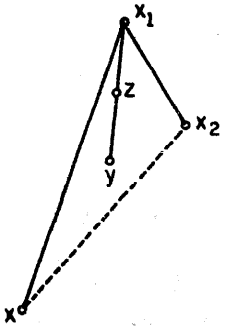

Fig. 5. Point y is on the outside of the polygon. Some point z on line segment (x_1, y) must lie on the polygon boundary. At least one end of the boundary edge containing z must lie inside the triangle x_1, x_2, x.

in the interior of the triangle (Fig. 5). Thus, the diagonal (x, x_2) lies completely within the polygon. A proof by induction verifies that the other diagonals constructed in case (a) lie inside the polygon. The proofs for cases (b) and (c) are similar.

It is routine to verify that properties (i) through (iii) are preserved by the general step. The validity of (iv) follows from (*).

The algorithm requires $O(n)$ time for the initial merge-sort and $O(n)$ time to process all the vertices. The space required is also $O(n)$. Thus, except for constant factors, neither the time complexity nor the space requirements can be improved.

3. Regularization and monotone polygons

To complete the triangulation algorithm we need a method for subdividing an arbitrary simple polygon into one or more monotone polygons. For this purpose we use the regularization algorithm of [1]. This algorithm adds diagonals to the given polygon P, none of which intersects either a boundary edge or another diagonal, so that the following properties hold:

(i) Each vertex (except the one with largest y-coordinate) is joined directly to at least one vertex with a larger y-coordinate; and

(ii) Each vertex (except the one with smallest y-coordinate) is joined directly to at least one vertex with a smaller y-coordinate.

The regularization algorithm may add some diagonals which are exterior to P (see Fig. 6(a)), but for our purposes these can be ignored. As shown in Fig. 6(b), the diagonals added to the interior of P subdivide P into a number of smaller polygons, each of which contains no portion of any other diagonal in its interior. We claim that each of these polygons must be monotone.

Let us say that a vertex q of a simple polygon Q is an *interior cusp* of Q if the internal angle at q exceeds 180° and the two vertices adjacent to q on the boundary of Q either both have larger y-coordinates than q or both have smaller y-coordinates than q. We then observe that, if Q is any one of the polygons obtained as described above from regularizing P, then no vertex of Q can be an interior cusp. This is an immediate consequence of the two regularization properties and the fact that Q was chosen so that none of the diagonals introduced in the regularization of P intersects its interior. The following theorem then implies that each of the polygons obtained by regularizing P is a monotone polygon.

Theorem 1. *If Q is a simple polygon and no vertex of Q is an interior cusp, then Q is monotone.*

Proof. Let Q be a polygon satisfying the hypothesis of the theorem, and let $q_1, q_2, \ldots, q_m$ denote the vertices of Q, listed in clockwise order around the bound-

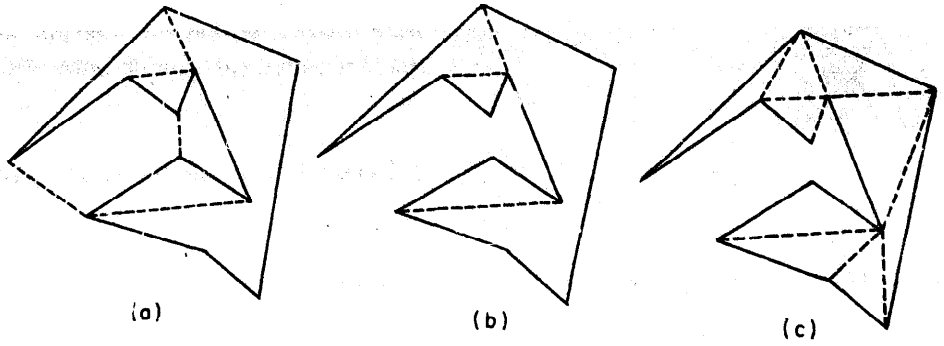

Fig. 6. A simple polygon P: (a) regularized; (b) decomposed into monotone polygons; (c) triangulated.

ary. Suppose without less of generality that q_1 has the largest y-coordinate among these and that q_k, $1 < k \leqslant n$, has the smallest y-coordinate. If Q is not monotone, then one of the two chains from q_1 to q_k formed by the boundary edges of Q is not strictly decreasing by y-coordinate. Consider the case in which the chain passing through q_2 fails to be strictly decreasing (the other case is symmetric). Choose q_i, $1 < i < k$, to be the first vertex on this path such that the y-coordinate of q_{i+1} exceeds that of q_i.

We first observe that the edge from q_i to q_{i+1} must lie to the left of the edge from q_{i-1} to q_i, for otherwise q_i would be an interior cusp of Q (see Fig. 7(a)). Now consider the line through q_i and q_k (see Fig. 7(b)), and let $r \neq q_i$ be the first point on the boundary of Q encountered when traveling from q_i to q_k along this line (r might be q_k). Then the line segment joining q_i to r divides the exterior of Q into two parts, one of which is a finite polygon Q', as shown in Fig. 7(c). Except for r, the vertices of Q' are all vertices of Q. Among all the vertices of Q', let s be the one having largest y-coordinate. Then s is also a vertex of Q, and furthermore s must be an interior cusp; it therefore follows that Q must be monotone.

The following algorithm triangulates an arbitrary simple polygon.

Step 1. Apply the regularization algorithm to the polygon. Delete all added diagonals which are exterior to the polygon (in fact the regularization algorithm can be modified to add only the interior diagonals).

Step 2. Apply the algorithm of Section 2 to triangulate each monotone polygon in the subdivision of the given polygon constructed by Step 1.

Fig. 6(c) shows the results of applying this algorithm to a simple polygon.

The regularization algorithm given in [1] requires $O(n \log n)$ time for an n-vertex polygon, and the algorithm of Section 2 requires a total of $O(n)$ time to triangulate all the polygons in the resulting subdivision. Thus the total time required for triangulating an n-ver-

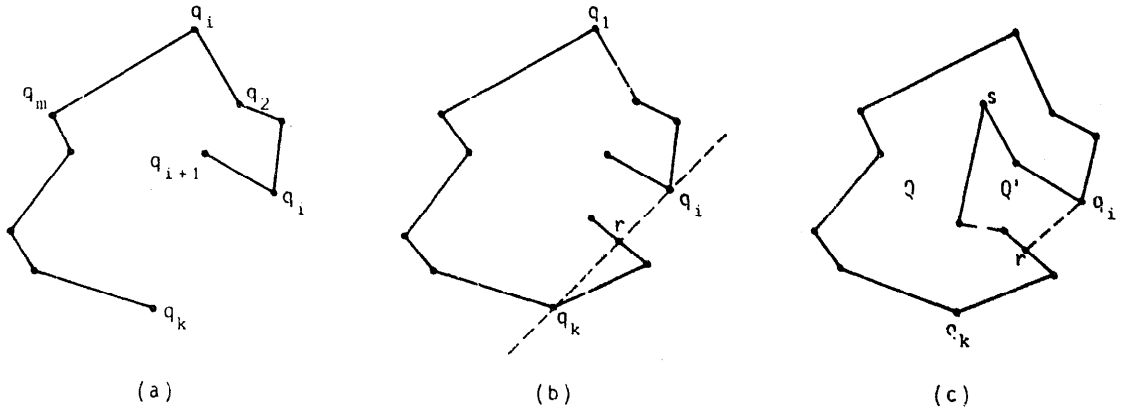

Fig. 7. Steps in the proof of Theorem 1: (a) q_i is the first vertex on the path $q_1, q_2, \ldots, q_k$ such that q_{i+1} has larger y-coordinate than q_i; (b) line from q_i to q_k intersects polygon boundary for the first time at point r; (c) polygon Q' exterior to Q formed by the line segment (q_i, r); s is the vertex of Q' having largest y-coordinate and must be an interior cusp of Q.

tex simple polygon is $O(n \log n)$. The space required is $O(n)$.

References

[1] D.T. Lee and F.P. Preparata, Location of a point in a planar subdivision and its applications, SIAM J. Comput. 6 (1977) 594–606.

[2] R.J. Lipton and R.E. Tarjan, Applications of a planar separator theorem, to appear.

[3] D.H. McLain, Two-dimensional interpolation from random data, Computer Journal 19 (1976) 178–181.

[4] M.I. Shamos, Computational Geometry, Dept. of Computer Sci., Yale University (1977). To be published by Springer–Verlag.

[5] M.I. Shamos and D. Hoey, Closest-point problems, Proc. IEEE Symp. on Foundations of Comput. Sci. 16 (1975) 151–162.

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