Discrete Comput Geom 7:329–346 (1992)

Discrete & Computational
Geometry

Polygon Triangulation in $O(n \log \log n)$ Time with Simple Data Structures*

David G. Kirkpatrick,[1] Maria M. Klawe,[1] and Robert E. Tarjan[2]

[1] Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada V6T 1Z2

[2] Department of Computer Science, Princeton University, Princeton, NJ 08540, USA, and NEC Research Institute

Abstract. We give a new $O(n \log \log n)$-time deterministic algorithm for triangulating simple n-vertex polygons, which avoids the use of complicated data structures. In addition, for polygons whose vertices have integer coordinates of polynomially bounded size, the algorithm can be modified to run in $O(n \log^* n)$ time. The major new techniques employed are the efficient location of horizontal visibility edges that partition the interior of the polygon into regions of approximately equal size, and a linear-time algorithm for obtaining the horizontal visibility partition of a subchain of a polygonal chain, from the horizontal visibility partition of the entire chain. The latter technique has other interesting applications, including a linear-time algorithm to convert a Steiner triangulation of a polygon into a true triangulation.

1. Introduction

Let P be a simple polygon with n vertices. The *diagonals* of P are the open line segments whose endpoints are vertices of the polygon and that lie entirely in the interior of P. A *triangulation* of P is a partition of its interior into $n - 2$ triangles by adding $n - 3$ nonintersecting diagonals. The problem of triangulating a simple polygon, that is, determining the set of nonintersecting diagonals, has attracted considerable attention in computational geometry literature and elsewhere.

* This research was partially supported by the following grants: NSERC 583584, NSERC 580485, ONR-N00014-87-0467, and by DIMACS, an NSF Science and Technology Center (NSF-STC88-09648).

The practical motivation for developing efficient triangulation algorithms is based on the widespread applications of this problem in related disciplines such as computer graphics [FM], [FFR], [L], pattern recognition [T1], and computational morphology [T2], [T4] as well as its role as a fundamental building block in the solution of a variety of (superficially) more complex problems in computational geometry such as geometric decomposition [FM], [LTL], visibility [C1], [GHL$^+$], shortest path [GHL$^+$], separability [BT], and subdivision preconditioning [EGS], [K] problems.

An $O(n \log n)$ upper bound on the complexity of triangulating arbitrary simple polygons was first established by Garey *et al.* [GJPT]. Their algorithm is based on a linear-time method for triangulating monotone polygons. Like the subsequent triangulation algorithm of Chazelle [C1], it involves an explicit sorting step. The first improvements on the $O(n \log n)$ bound came by expressing the complexity in terms of structural characteristics of the input polygon (such as the number of reflex angles [HM], [FM] or its sinuosity [CI]) or the output [T3]. A separation between the worst-case complexities of polygon triangulation and sorting was not established, however, until Tarjan and Van Wyk [TV] presented an $O(n \log \log n)$ algorithm for the former. Subsequently, Clarkson *et al.* [CTV] described a randomized algorithm that gives an $O(n \log^* n)$ upper bound on the expected time complexity of the problem.

Other notable developments expanded the class of families of simple polygons for which linear-time algorithms were known [TA], established a family of problems which are linear-time equivalent to polygon triangulation [FM], and specified some fundamental primitives out of which efficient triangulation algorithms could be constructed [C1], [CI], [HMRT], [FNTV].

Very recently [C2], Chazelle has given an $O(n)$ deterministic algorithm, thus finally settling the question of the asymptotic complexity of polygon triangulation. Chazelle's result is very beautiful and unquestionably will stand as one of the landmarks of computational geometry for many years to come. Although the underlying ideas in Chazelle's algorithm are both simple and elegant, at least in its current state, the algorithm uses some fairly substantial machinery, making an intuitive understanding of its correctness a fair challenge for the nonexpert.

Our contributions in this paper are best appreciated in conjunction with these related results. We present a new $O(n \log \log n)$-time algorithm for triangulating simple polygons that uses only elementary data structures and is formulated in terms of three basic geometric subproblems (two of which, including planar subdivision search, have been identified and studied previously), each of which is of interest in its own right.

Like all of the recent approaches to triangulation [CI], [FM], [TV], [CTV], [C2] ours focuses on the problem of constructing the horizontal visibility partition (HVP) of the edges of a given polygonal chain, i.e., the partition of the plane obtained by adding horizontal edges connecting each vertex to the closest point on the chain on each side if such a point exists, and to a point at infinity otherwise. This problem is well known to be linear-time equivalent to the problem of triangulating a simple polygon [FM], [CI]. Two natural operations on HVPs form the building blocks of our algorithm. First we need to be able to construct

the HVP of a polygonal chain P given the HVP of some prefix subchain P_1 of P together with the HVP of the remainder $P \backslash P_1$. We refer to this as a *merging* of *HVPs*. Chazelle and Incerpi [CI] show that merging of HVPs can be done in time linear in the size of the input partitions. They use this as the merge step in a natural divide-and-conquer algorithm for building the HVP of an arbitrary chain. We use merging in a fundamentally different way. Our algorithm uses a balanced divide-and-conquer approach somewhat like that of [TV], but our splitting is achieved without Jordan sorting and without the use of finger search trees, which are (both conceptually and practically) the most demanding components of the earlier algorithm.

In addition to the merge operation of HVPs we also need to perform its inverse, which we call a *splitting* operation. Specifically, given an HVP for P we need to construct the HVP for P_1, a prefix of P. We show that this can also be done in linear time.

The splitting operation can be viewed as the problem of updating an HVP upon the removal of certain of its nonhorizontal edges. Exploiting the linear-time equivalence between polygon triangulation and HVP construction, our splitting algorithm has the following interesting implication for the dynamic maintenance of arbitrary point-set triangulations: given a triangulation T of a point set S and a subset $D \subseteq S$ such that the connected components of the set formed by taking the union of the triangles (including their interiors) adjacent to the vertices in D are simply connected, it is possible to update T to a triangulation of $S \backslash D$ in time $O(\deg_T(D))$ where $\deg_T(D)$ denotes the sum, over all vertices v in D, of the degree of v in T. This immediately implies that the problem of triangulating a simple polygon *allowing Steiner points* is linear-time equivalent to the (Steiner-point-free) polygon triangulation problem.

The remainder of the paper is organized as follows. Section 2 gives the $O(n \log \log n)$-time algorithm for computing the HVP of a simple polygon with n vertices. Section 3 describes the linear-time HVP splitting algorithm. In Section 4 we show how the algorithm can be modified to run in $O(n \log^* n)$ time for polygons whose vertices have integer coordinates of polynomially bounded size.

2. The HVP Algorithm

In this section we give our $O(n \log \log n)$ algorithm for computing the HVP of a simple polygon with n vertices. We work with a variant of HVP called the wrap-around partition, and we compute the wrap-around partitions of directed polygonal chains as well as polygons. Moreover, we assume that the chains and polygons have no self-crossings, and are nondegenerate in the sense that two distinct vertices cannot have the same vertical coordinate. None of these modifications alters the asymptotic complexity of the problem.

If e is a directed edge, the *positive* side of e is its right-hand side with respect to its direction. The positive side of a directed chain is defined analogously. The horizontal line through a vertex on a directed chain C may intersect C on its positive sides at that vertex from zero, one, or two directions (see Fig. 1).

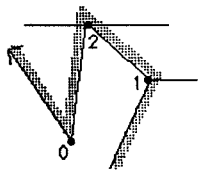

Fig. 1

Wrap-around partitions regard chains and polygons as being embedded on an infinite vertical cylinder, in which horizontal lines going off to infinity wrap-around and reappear on the other side. The *one-sided wrap-around partition* of a directed polygonal chain C is the partition of the cylinder obtained by adding, for each vertex v of C, each horizontal line (if any) which starts at v and continues on a positive side of C at v, until it intersects C again. Note that horizontal lines will extend out on both sides of v if both sides of v are on the positive side of C as in the case of the vertex labeled 2 in Fig. 1, and we also adopt the convention that lines go out from both sides when v is an endpoint of the chain C. If C is a polygon directed so its positive side is the polygon's interior side, then the one-sided wrap-around partition is the usual HVP of the polygon. The *doubling* of a directed chain is the simple polygon obtained by taking the concatenation of C with a slight perturbation of its reversal, with the perturbation chosen so that the doubling satisfies the nondegeneracy condition and so that the perturbed reversal lies on the negative side of C. Note that with the exception of the arbitrarily narrow region trapped between C and its perturbed reversal, the interior of the doubling of C is everything outside C. The two-sided wrap-around partition adds both sets of horizontal edges from the one-sided partitions of C and its (unperturbed) reversal. Although our construction could be modified to deal only with one-sided wrap-around partitions, a number of the steps seem both simpler and more intuitive in the context of two-sided partitions. For this reason we use the "doubling" trick to move between one-sided and two-sided partitions. For any point x on a chain C, its *horizontal neighbors* are the points on the intersection of C with the horizontal line through x which are closest to x on each side. The interior horizontal neighbors of a point on a polygon are defined analogously. These definitions are illustrated in Fig. 2. For convenience, we abbreviate wrap-around partition by WP.

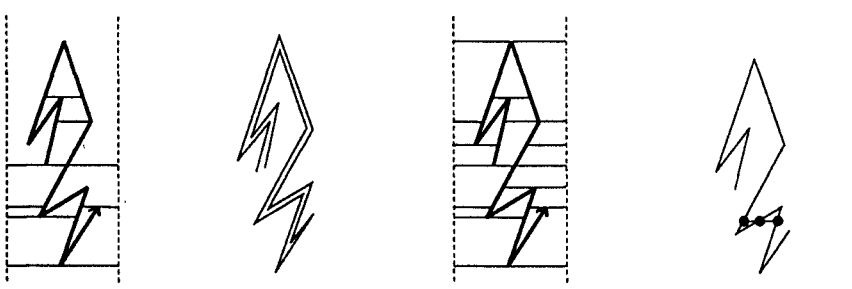

Fig. 2

We use the term *length* of a chain or polygon to mean the number of its edges. A *k-uniform partition* of a chain or polygon is a partition (with all divisions occurring at vertices) into segments of length between $k/2$ and k inclusive, such that we know the two-sided WP of each segment in the partition. Note that we use the term segment to mean a subchain of a chain rather than, as is often the case, as an abbreviation for line segment. The number of segments in a k-uniform partition of an r-vertex chain or polygon is between r/k and $2r/k$. The main component of our algorithm is the following theorem.

Theorem 2.1. *Suppose $k \geq (r/2)^{2/3}$. Given a k-uniform partition of an r-vertex nondegenerate polygon P, we can compute the WP of the interior of P in $O(r)$ time.*

Given Theorem 2.1, it is easy to give an $O(n \log \log n)$-time algorithm for computing the two-sided WP of an n-vertex nondegenerate chain. We partition C into segments of length between $n^{2/3}/2$ and $n^{2/3}$, and use our algorithm recursively to compute the two-sided WP of each of the segments. Next, by Theorem 2.1, regarding the doubling of C as a $2n$-vertex polygon, P, in $O(n)$ time we compute the WP of the interior of P, from which we can easily obtain the two-sided WP of C in $O(n)$ time by correcting the perturbations made in forming the doubling of C. (Recall that the interior of the doubling of C is essentially everything outside C and hence the WP of P is, modulo the perturbations, the two-sided WP of C.) The time, $T(n)$, required by this algorithm satisfies the recurrence $T(n) = O(n) + \sum_i T(n_i)$ where $\sum_i n_i = n$ and $n_i \leq n^{2/3}$, which implies $T(n) = O(n \log \log n)$.

In the assumption $k \geq (r/2)^{2/3}$ in the statement of Theorem 2.1, the use of $r/2$ instead of r is a technicality caused by the fact that the theorem is applied to the doubling which has twice as many vertices as the original chain. The choice of two-thirds as the exponent is arbitrary, and could have been any constant strictly between one-half and one. The need to use an exponent less than one in order to achieve the bound $T(n) = O(n \log \log n)$ is obvious. The requirement that the exponent be greater than one-half will become clear later, namely in the proof of Lemma 2.2.

Proof of Theorem 2.1. The overall idea is as follows. The theorem is proved by induction on r. We fix a constant r_0, specified in Lemma 2.6, and for $r < r_0$ compute the WP of P by merging together all the WPs of the segments in the uniform partition. Thus we assume $r \geq r_0$. We wish to partition the interior of P into regions by adding horizontal edges joining some of the vertices on P to their interior horizontal neighbors. We call these regions *chunks*, and compute the WP of P by computing the WP of each chunk separately. To achieve the $O(r)$ running time, we want the partition into chunks to have the property that at least a fixed fraction of the vertices of P will lie on regions whose WP we can compute easily from the WP information about the segments. To do this we introduce the notion of special point. A point x is a *special point* of a segment in a k-uniform partition of P if it has the following two properties:

(i) x has an interior horizontal neighbor $h(x)$ in P such that if P is cut at x and $h(x)$, both the resulting subpaths will contain at least $k/36$ of the vertices

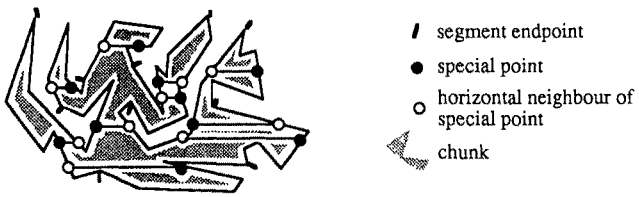

Fig. 3

of P. We call $h(x)$ the special horizontal neighbor of x (note that x might have more than one horizontal neighbor).

(ii) If the segment is cut at x both the resulting subsegments will contain (counting x if x is a vertex) at least $k/36$ of the vertices of the segment.

The complete proof of the theorem depends on several straightforward facts about special points, k-uniform partitions, and chunks, which we prove later in this section. In particular, by proving that each segment contains a special point (Lemma 2.5), that we can identify a special point in a segment in $O(k)$ time (Lemma 2.5), and that we can find the special horizontal neighbors in P of all the special points in $O(r)$ total time (Lemma 2.2), we see that in $O(r)$ time we can partition the interior of P into chunks by adding the horizontal edges joining each special point to its special horizontal neighbor. Figure 3 illustrates the segments, special points, and chunks as they are used in the algorithm.

The boundary of a chunk alternates between horizontal edges and pieces of P, where each such piece is the concatenation of at most two subsegments of the segments in the uniform partition. We call the number of horizontal edges on the boundary the *degree* of a chunk. We call a chunk *good* if it has degree at most 2, and *bad* otherwise. Using the linear-time splitting and merging algorithms, we can compute the WP of a good chunk in $O(k)$ time, and hence we compute the WPs of all the good chunks in $O(r)$ time. Using the linear-time splitting and merging algorithms, we show that in $O(r)$ time we can obtain a k-uniform partition of the boundary of each bad chunk. The bad chunks are degenerate since any pair of vertices forming the endpoints of an added horizontal edge obviously have the same vertical coordinate, but the bad chunks can easily be made nondegenerate by slightly perturbing one endpoint of each horizontal edge. After doing this, we then complete the computation of the WP of P by applying Theorem 2.1 inductively to each bad chunk.

Finally, in order to prove that the total running time used in the computation of WP of P is $O(r)$, it suffices to show that the total number of vertices lying on the boundaries of bad chunks is less than cr for some fixed constant $c < 1$, which we do in Lemma 2.6. □

We now prove the lemmas needed for Theorem 2.1.

Lemma 2.2. *Given any set of $O(r/k)$ points $x_1, \ldots, x_s$ on P we can find their interior horizontal neighbors in $O(r)$ time.*

Proof. Let S_i be the segments in the uniform partition, and let $WP(S_i)$ denote the WP of S_i. By using planar point location [EGS], in $O(r)$ time we create a data structure for each $WP(S_i)$, so that, for any point x and each i, in $O(\log|S_i|)$ time we can locate the region of $WP(S_i)$ containing x. We next find, in

$$O((r/k)^2 \log k) = O(r)$$

time (since $k \geq (r/2)^{2/3}$), the interior horizontal neighbors in P of each x_i as follows. For each x_i and each S_j we find the region in $WP(S_j)$ containing x_i and the two edges, $L_j(x_i)$ and $R_j(x_i)$, which x_i sees to its left and right, respectively, in the region. Using the planar point location data structures this can be done in $O((r/k)^2 \log k)$ time in total. Now for each x_i, depending on where the interior of P is with respect to x_i, we examine one or both of the two sets of edges $\{L_j(x_i)\}$ and $\{R_j(x_i)\}$ to find the edge(s) which x_i sees in the interior of P. This requires at most $O((r/k)^2) = O(r)$ time. □

Lemma 2.3. *If a polygon C is partitioned into t segments of length at most k whose two-sided WPs we know, then in $O(tk)$ time we can produce a k-uniform partition of C.*

Proof. Let $x_1, \ldots, x_s$ be vertices in clockwise order on C that partition C so that all segments have length between $k/2$ and k. We call these the uniform segments. Note that $s/2 \leq t$. We show that we can find the two-sided WPs of the uniform segments in $O(tk)$ time. We first split into subsegments the original segments whose two-sided WPs we know, by splitting at each x_i and computing the WP of each of these subsegments using the linear-time splitting algorithm. Since each segment that gets split has length at most k and s splits are performed, the time used in splitting is $O(sk) = O(tk)$. We compute the WP of each uniform segment by merging the WPs of its subsegments. Since each merge involves chains of length at most k and at most t merges are performed, the total time used is $O(tk)$. □

Lemma 2.4. *The total time needed to produce a k-uniform partition of the boundary of every chunk is $O(r)$.*

Proof. Since the chunks are formed by adding at most $2r/k$ horizontal edges, in $O(r)$ time we can split each segment in the k-uniform partition of P at the endpoints of the added horizontal edges and compute the WPs of these subsegments. Note that the total number of subsegments is still $O(r/k)$ and that every subsegment is of length at most k. Now, using Lemma 2.3, the total time to produce a k-uniform partition of the boundary of every chunk is $O(r)$. □

Before proving the remaining facts about special points and chunks we need to discuss the dual graphs associated with visibility partitions. Given a WP of a chain (either one-sided or two-sided) we define the *dual graph of the partition* to be the graph whose vertices are the connected components of the partition of the cylinder induced by the WP, such that two vertices are adjacent if their components share a horizontal edge of the partition. Figure 4 shows examples. The main reason we use WPs is the easily proved fact that the dual of an (either one-sided or

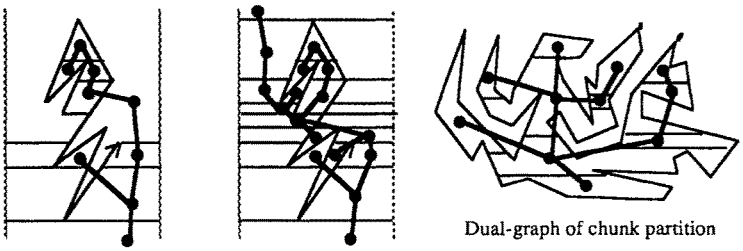

Fig. 4

two-sided) WP of a chain is always a tree. In addition, the nondegeneracy condition implies that the tree has maximum degree at most 4. The dual graph of a one-sided WP of a polygon is defined similarly except that the vertices are restricted to the connected components lying in the interior of polygon, and again the dual graph is always a tree of maximum degree at most 4.

More generally, the dual graphs of partitions obtained by adding any set of edges joining points to their (wrap-around) horizontal neighbors are trees, though not necessarily of maximum degree 4. Note that the degree of a chunk is simply the degree of the corresponding vertex in the dual graph of the chunk partition.

Lemma 2.5. *If N is a segment in a k-uniform partition of a polygon P, then N has a special point, and we can identify it in $O(k)$ time.*

Proof. Let Q be the middle subsegment of N obtained by chopping off $\lfloor k/36 \rfloor$ edges at each end of N. The length of Q is at least $\lceil 4k/9 \rceil$ since the length of N is at least $k/2$. It is easy to see that every point of Q satisfies special point property (ii), so it suffices to show that some point of Q satisfies special point property (i) and that given the WP of N we can identify such a point in $O(k)$ time.

Let T be the set of edges in the dual graph of the WP of the interior of P whose corresponding horizontal edges have at least one endpoint lying on Q. Since Q is a connected portion of P, T is connected and hence is a tree of degree at most 4. It is not hard to prove that every pair of consecutive edges of Q must contain an endpoint of a horizontal edge of the WP of P and hence there are at least $\lceil 2k/9 \rceil$ edges in T. Now, since the maximum degree of T is 4, it is easy to prove that there is some edge e in T such that both connected components in $T - e$ have at least $k/18 - 1$ edges. Moreover, given T we can find e in time linear in the size of T. Let T_1, T_2 be the two connected components of $T - e$, let x be an endpoint of the horizontal edge corresponding to e which lies on Q, and let $h(x)$ be the other endpoint. If we cut P at x and $h(x)$, then the resulting subpaths of P correspond in an obvious way to T_1 and T_2. The number of vertices on a subpath is at least the number of vertices which are an endpoint of a horizontal edge corresponding to an edge in $\{e\} \cup T_i$, where T_i is the appropriate connected component of $T - e$. Since each vertex is the endpoint of at most two horizontal edges corresponding

to edges in T, we see that each subpath has at least $k/36$ vertices, and hence x satisfies property (i).

Finally, we must show that we can identify x in $O(k)$ time. Let P' be the polygon obtained by doubling N. Applying exactly the same argument regarding Q as a subsegment of P' shows that in $O(k)$ time we can find a point x on Q with an interior horizontal neighbor $h'(x)$ on P' such that cutting P' at x and $h'(x)$ yields two subpaths with at least $k/36$ vertices on each subpath. We claim that x is a special point of N. As before, since x is on Q it suffices to show x satisfies property (i). Let $h(x)$ be the interior horizontal neighbor of x on P in the same direction as $h'(x)$. If $h(x) = h'(x)$ we are done. If not, then $h(x)$ is not on N. Now since x is at least $\lfloor k/36 \rfloor$ edges away from both ends of N, it is clear that cutting P at both x and $h(x)$ results in subpaths of P with at least $k/36$ vertices. □

Lemma 2.6. *There are constants $c < 1$ and r_0 such that if $r \geq r_0$, then at most cr vertices lie on the boundaries of bad chunks.*

Proof. The total number of vertices lying on all chunks is at most $r + 8r/k$ since at most $2r/k$ edges are added and each horizontal edge adds at most four vertices to the total. Since $r \geq r_0$ and $k \geq (r/2)^{2/3}$ we have $k \geq (r_0/2)^{2/3}$ and hence $r/k \leq r/(r_0/2)^{2/3}$. Given this bound on r/k, it suffices to show that there is a positive constant α such that at least αr vertices lie on the boundaries of good chunks. As noted before, the dual graph of the chunks is a tree with at least r/k vertices. By the definition of special point, the chunks of degree 1 have at least $k/36$ vertices on their boundary. In addition, it is straightforward to check that, for any $m \geq 2$, the total number of vertices on the boundaries of a set of m chunks of degree 2 that form a path is at least $mk/36$. Putting these facts together with the easily proved lemma that, for any p-vertex tree T, the number of leaves in T plus the number of degree 2 vertices in T that are adjacent to a vertex of degree 2 is at least $p/4 + 1$, we see that at least $(r/4k)(k/36) = r/144$ vertices lie on the boundaries of good chunks. □

3. The Splitting Algorithm

In this section we use the standard definition of HVP, which is trivially translatable to and from the two-sided wrap-around version in linear time. If E is a collection of edges that are disjoint except possibly at their endpoints, the HVP is the partition obtained by adding horizontal edges as follows. For each endpoint z of an edge e in E, we add a horizontal line segment through z that extends in both directions until it intersects another edge of E on each side. We also add two dummy vertical edges at $x = -\infty$ and $x = +\infty$, so that each region (except for the top and bottom) in the HVP is a trapezoid. An example is shown in Fig. 5. We refer to the two nonhorizontal edges of a trapezoid as the *sides* of the trapezoid, and use the term *slice* an edge to mean dividing an edge into two edges by inserting a new vertex on the edge.

For the HVP algorithm in the preceding section, we need an algorithm that,

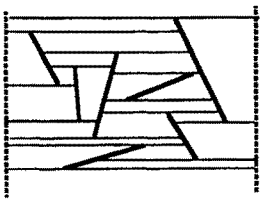

Fig. 5. A horizontal visibility partition.

given the HVP of a polygonal chain C, and a subchain C_1, computes the HVP of C_1 in time linear in the length of C. Let us call this the *chain-splitting problem.* Our splitting algorithm actually works, and seems much easier to prove correct, in a more general context. To give this context, we must introduce the concept of hole-free. We say that a set of edges A *trap* an edge e in the HVP of E if there is a simple polygon P containing e in its interior, where each edge in P is either horizontal and lies in the interior of a trapezoid of the HVP of E with both sides in A or is a subsegment of an edge in A. An example is shown in Fig. 6. We say A is *hole-free with respect to* E if no edge of $E \backslash A$ is trapped by A. It is straightforward to check that modifying the set E (and possibly A) by either removing an edge of A or slicing an edge in two does not affect the condition that A is hole-free with respect to E. An edge e of E and a trapezoid are said to be *adjacent* if part of e forms one of the sides of the trapezoid, and two edges e and e' *share a trapezoid* if it is adjacent to both of them. We define the degree of the edge e, denoted by $d(e)$, to be the number of trapezoids adjacent to it, and set $D(A) = \sum_{e \in A} d(e)$.

Suppose $E = E_1 \cup E_2$ is a collection of edges that are disjoint except possibly at their endpoints, such that E_2 is hole-free with respect to E. Our splitting algorithm solves the following problem which we call *hole-free edge removal.* Given the HVP of E, compute the HVP of E_1 in $O(D(E_2))$ time.

It is worth observing that some restriction such as the hole-free condition must be made on the set of edges E_2 to allow their removal in linear time, since it is easy to give an $\Omega(n \log n)$ lower bound for the general problem of removing $n/2$ edges from an HVP of n edges by a reduction from sorting. Figure 7 illustrates the idea underlying the reduction, i.e., the HVP of the vertical edges is trivial to compute, but once the long vertical edges are removed, the sorted order of the vertical coordinates of the short vertical edges can be computed from their HVP.

We begin by showing that chain splitting is reducible to hole-free edge removal.

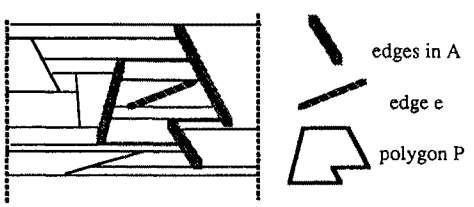

Fig. 6. The edge e is trapped by A.

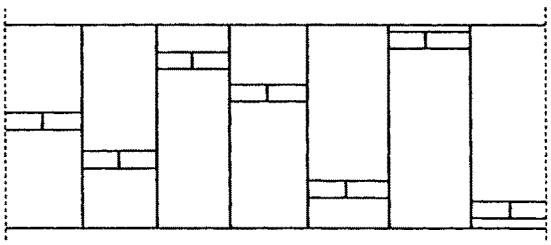

Fig. 7. The $\Omega(n \log n)$ lower bound for the general splitting problem.

Let C be a polygonal chain such that we know its HVP, and let C_1 be a subchain of C. Let E_1 be the set of edges in C_1 together with an infinite vertical edge L through the leftmost vertex on C_1, let E_2 be the set of edges of C which are not on C_1, with each edge split into two edges if it intersects L, and let $E = E_1 \cup E_2$. It is easy to see that we can obtain the HVP of E from the HVP of C, and the HVP of C_1 from the HVP of E_1, in linear time. Moreover, because of the addition of L and the fact that $C_1 \cup L$ is connected, it is easy to see that no edge in E_1 is trapped by E_2.

We now describe the hole-free edge removal algorithm. We use a simple data-structure to represent the HVP of E with the property that any edge can be removed in time proportional to its degree. Specifically, we represent each trapezoid, T, in the HVP by the list of its corner points in clockwise order, plus pointers to the edges to which it is adjacent. In addition, we add four pointers, left-up, left-down, right-up, and right-down. If e is the left side edge of T, the pointer left-up points to the trapezoid, if any, which also has e as its left side edge and is immediately above T in the HVP. The other pointers are defined analogously. Finally, for each edge e we provide pointers to its top and bottom trapezoids on both sides. Thus, in effect we have, for each edge e in E, two doubly linked lists of the trapezoids on its left and right sides, ordered according to their order along e. To remove an edge, e, we simultaneously walk along its two lists of adjacent trapezoids, extending each horizontal edge which ended at the interior of e through the trapezoid on the other side, splitting and merging the trapezoids from the two lists as necessary (see Fig. 8). Since a bounded amount of work is needed for each horizontal edge with an endpoint on e, the total time to remove e is clearly $O(d(e))$. As a consequence of this data structure, we may assume that all the edges in E_2 have degree at least 25, by pulling out small degree edges one at a time until no more remain.

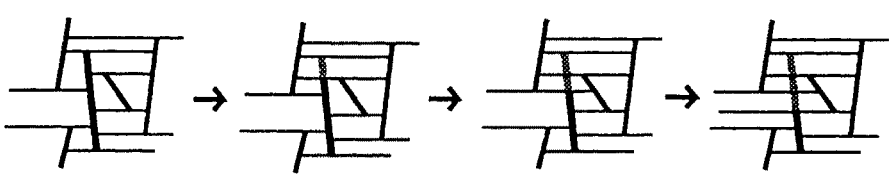

Fig. 8. The process of removing an edge.

The strategy underlying the algorithm is to reduce, in $O(D(E_2))$ time, the problem of computing E_1 to a problem which is smaller by a constant fraction, namely to computing the HVP of E_1' from the HVP of $E_1' \cup E_2'$ where $D(E_2') \leq 24D(E_2)/25$. Moreover, we must ensure that E_2' is hole-free with respect to $E_1' \cup E_2'$. By applying our algorithm recursively to the smaller problem, we thus obtain an algorithm whose running time is bounded by $O(D(E_2))$. To perform the reduction, we identify the subset E_2' of edges (or actually pieces of edges) in E_2 such that $D(E_2')$ is small and such that after removing E_2' the set, F, of remaining (pieces of) edges in E_2 will be easy to remove. Before defining E_2' and F, we prove a lemma giving the basic method we use to remove the edges in F. This lemma is based on the observation that although removing an edge e from an HVP may increase the degree of remaining edges, it can only increase the degrees of edges e' such that e and e' shared a trapezoid in the HVP, and the sum of the increases is at most $d(e)$.

Lemma 3.1. *Suppose that E_1 and F are disjoint sets of edges such that in the HVP of $E_1 \cup F$ each edge of F has at most one side on which it shares trapezoids with other edges in F. Then the HVP of E_1 can be computed from the HVP of $E_1 \cup F$ in $O(D(F))$ time.*

Proof. Let $F_L = \{e \in F: e$ has a trapezoid on its left side that it shares with another edge of $F\}$ and let $F_R = F \backslash F_L$. Note that no pair of edges in F_L (F_R) share a trapezoid. Let f_L, f_R be the number of trapezoids adjacent to edges in F_L, F_R, respectively in the HVP of $E_1 \cup F$. We first pull out the edges in F_L one by one. Removing an edge in F_L can only increase degrees of edges in $E_1 \cup F_R$, so the total time for removing the edges in F_L is $O(f_L)$, and the total increase of the degrees of edges in F_R is also $O(f_L)$. Now removing any edge in F_R only increases the degrees of edges in E_1, so the total time for removing the edges in F_L is $O(f_L + f_R) = O(D(F))$. □

Given this lemma, our goal is to slice up the edges in E_2 to obtain a set $E_3 = F \cup E_2'$ such that E_2' is hole-free with respect to $E_1 \cup E_3$ and $D(E_2') \leq 24D(E_2)/25$. In addition, we want to choose the slices so that in $O(D(E_2))$ time we can both compute the HVP of $E_1 \cup E_3$ from the HVP of E and, via Lemma 3.1, obtain the HVP of E_1 from the HVP of $E_1 \cup F$. In order to describe how the slicing of edges in E_2 is done, we first color the trapezoids in the HVP of E. A trapezoid is white if it is adjacent to an edge in E_1 and gray otherwise (i.e., if both its side edges are subsegments of edges in E_2). See Fig. 9(a) for an illustration. For each edge in E_2, we slice it at each of its adjacent horizontal edges which border both gray and white trapezoids, to obtain the set of edges E_3, and compute the new HVP of $E' = E_1 \cup E_3$ (see Fig. 9(b)). Our data-structure supports doing this in time proportional to the degree of the edge in the current HVP, and it is not hard to check that the total amount of time used is proportional to $D(E_2)$. Let F be the set of edges in E_3 which share a trapezoid of the HVP of E' with an edge in E_1, and let $E_2' = E_3 \backslash F$. By our previous observations on the preservation of the hole-free property under edge removal and slicing, and because every edge in F shares a trapezoid with an edge in E_1, it is not hard to prove that E_2' is hole-free

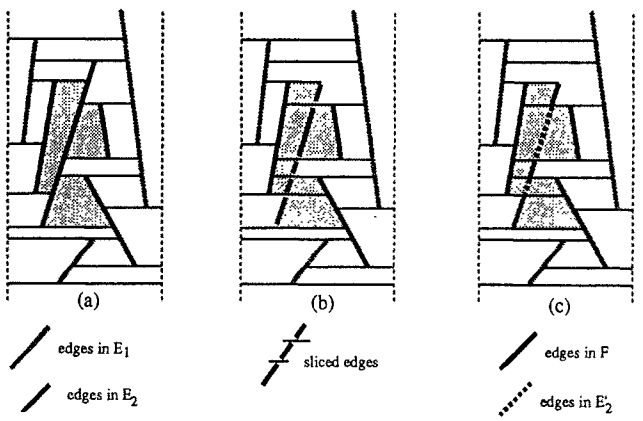

Fig. 9. Slicing the edges.

with respect to E'. The remainder of this section consists of a series of lemmas which establish that F and E'_2 have the other properties we need in order to justify our claim that the algorithm runs in $O(D(E_2))$ time.

Lemma 3.2. *If A is hole-free with respect to E, and if $d(e) \geq 9$ for each e in A, then every pair of edges in A form the sides of at most one trapezoid in the HVP of E.*

Proof. Suppose e, e' are edges in A and h, h' are horizontal edges in the HVP of E such that both h and h' join e to e' with h the highest such horizontal edge and h' the lowest such horizontal edge. Let T be the trapezoid formed by h, h' and the appropriate subsegments of e and e'. It suffices to show that no edges of the HVP of E lie inside the interior of T. Since A is hole-free with respect to E, no edges of $E \backslash A$ lie inside T. Let m be the number of edges of A lying inside T. We will show that $m = 0$. It is easy to see that the number of trapezoids of the HVP of E which lie inside T is at most $4m + 1$ because the m edges of E lying inside T generate at most $4m$ horizontal edges. This implies that if $m > 0$, then some edge inside T has degree at most 8, but this contradicts the assumption that every edge in A has degree at least 9. □

Lemma 3.3. *Given the HVP of $E_1 \cup F$ we can obtain the HVP of E_1 in $O(D(E_2))$ time.*

Proof. It follows immediately from the construction of E_3 that $|F| \leq |E_3| = O(D(E_2))$. Thus we may assume that $d(e) \geq 9$ for each edge e in F, since otherwise we can remove edges of F from the HVP one at a time until this is satisfied. Let Z be the set of trapezoids in the HVP of $E_1 \cup F$ which are adjacent to edges in F, and let Z_1 be the trapezoids adjacent to an edge of F on one side and an edge of E_1 on the other. We now show that $|Z| = O(D(E_2))$. We have $|Z_1| = O(D(E_2))$ since each trapezoid in Z_1 is a trapezoid in the HVP of $E_1 \cup E_3$ and $D(E_3) = O(D(E_2))$. Thus it suffices to show that $|Z \backslash Z_1| = O(D(E_2))$. From the observations

on the preservation of the hole-free property under edge removal and slicing, we see that F is hole-free with respect to $E_1 \cup F$. Thus Lemma 3.2 implies that $|Z \backslash Z_1|$ is bounded by the number of trapezoids in the HVP of F, i.e., by $O(|F|) = O(D(E_2))$. Finally we observe that in the HVP of $E_1 \cup F$, each edge in F has at least one side on which all the trapezoids are shared with edges in E_1. This is because the slicing used to form E_3 guarantees that the edges in F have this property in the HVP of $E_1 \cup E_3$, and since E_2' and E_1 are disjoint, removing the edges in E_2' cannot affect this property. Thus by Lemma 3.1 we can obtain the HVP of E_1 in $O(|Z|) = O(D(E_2))$ time. □

The final lemma proves the desired bound on $D(E_2')$, thus completing the proof that the algorithm runs in $O(D(E_2))$ time.

Lemma 3.4. *We have* $D(E_2') \leq 24D(E_2)/25$.

Proof. We color a trapezoid in the HVP of E' black if it is adjacent to an edge in E_2', and let r (r') be the number of gray (black) trapezoids in the HVP of E (E'). Since $D(E_2') \leq 2r'$, it will suffice to show that $r' \leq 3r$ and $r < 4D(E_2)/25$.

We first show that $r' \leq 3r$. We define the function q from the black trapezoids in the HVP of E' to the gray trapezoids in the HVP of E as follows. Let U be a black trapezoid. If U is also a gray trapezoid, then we set $q(U) = U$. If not, then at least one of the horizontal edges of U must be the extension of a horizontal edge h of the HVP of E that bordered both gray and white trapezoids, and we set $q(U)$ to be the gray trapezoid bordered by h. For any gray trapezoid T, there are at most three black trapezoids that q maps to T. Specifically, for each horizontal edge h of T there is at most one black trapezoid with a horizontal edge formed by extending h, and adding the possibility that q also maps T to itself yields a maximum of three.

We now show $r < 4D(E_2)/25$. By Lemma 3.2, the number of gray trapezoids is at most the number of trapezoids in the HVP of E_2, which is at most $4|E_2|$. Finally, since each edge in E_2 had degree at least 25 we have $D(E_2) \geq 25|E_2|$. □

4. An $O(n \log^* n)$-Time Algorithm for Rasterized Polygons

We call an n-vertex chain or polygon *rasterized* if the coordinates of its vertices are integers whose size is bounded by a fixed polynomial $p(n)$. In this section we show how our algorithm can be modified to compute the HVP of an n-vertex rasterized polygonal chain in $O(n \log^* n)$ time, where the constant depends on the polynomial $p(n)$.

The bottleneck of the algorithm given in Section 2 is the finding of the horizontal neighbors of the special points (Lemma 2.2). More precisely, this is the only place where we are forced to have the length of segments, to which we apply the algorithm recursively, be large, in order to have sufficiently few special points so that we can find their horizontal neighbors in linear time. The key to obtaining

an $O(n \log^* n)$-time algorithm for rasterized polygons is that given any n-vertex rasterized chain we can construct a data-structure in linear time such that the horizontal neighbors of any point can be found in $O(\log^2 n)$ time. This will allow us to make the length of the segments $\log^2 n$ instead of $n^{2/3}$, and still find the horizontal neighbors of the $2n/\log^2 n$ special points in $O(n)$ time.

We say that a rational number is $p(n)$-*bounded* if the absolute values of its numerator and denominator are both bounded by $p(n)$. The key property of a rasterized chain or polygon is that the x-coordinate of the intersection of any edge with the horizontal line $y = y_0$ where y_0 is a $p(n)$-bounded integer, will be a $(p(n))^2$-bounded rational. Constructing our data-structure for finding horizontal neighbors in rasterized chains relies on the fact that we can sort n $(p(n))^2$-bounded rationals in $O(n)$ time (see Lemma 3.2 in [KK] for example). One slight complication that arises is that as we recursively apply our algorithm, after the first recursion we are no longer working with rasterized chains since the vertex coordinates are not polynomially bounded by the length of the chains. We sidestep this difficulty by batching together the creation of the data-structures at the beginning of the algorithm (Lemma 4.3).

We say that a set of edges is *inner-disjoint* if they are disjoint except possibly at their endpoints. We use the term *fence* for a set of inner-disjoint edges $\{e_1, \ldots, e_k\}$ and a horizontal line L that intersects all the e_i. We refer to the line L as the *wire* of the fence, and the x-coordinate of the intersection of an edge of the fence with the wire as its *wire-coordinate*. We say the fence is *ordered* if we know both the sorted order of the vertical coordinates of the endpoints of the e_i, and the left to right order of the wire coordinates of the e_i.

Lemma 4.1. *If $F = (\{e_1, \ldots, e_k\}, L)$ is an ordered fence, then in $O(k)$ time we can construct a data structure such that for any point z we can find its horizontal neighbors in F in $O(\log k)$ time.*

Proof. It suffices to show how to obtain the HVP of F in $O(k)$ time since given the HVP we can use planar point location [EGS] to construct a data-structure to answer queries in the HVP of F in $O(\log k)$ time. By symmetry it suffices to find the horizontal neighbors of the top endpoints of the edges in F. Since we know both orders, in $O(k)$ time we can create a data-structure with a doubly linked list joining the edges in left to right order according to their wire-coordinates, and a linked list connecting the edges according to increasing vertical coordinate of their top endpoint. We now obtain the left and right neighbors of each top endpoint in increasing order of vertical coordinate, by assigning the current lowest top endpoint its left and right neighbors in the doubly linked list as its horizontal neighbors, and then deleting that endpoint's edge from the data-structure. Since this can be done in constant time for each endpoint the total time used is $O(k)$. □

It is possible to prove the preceding lemma, without the assumption that we know the sorted order of the vertical endpoints, by reducing finding the HVP of F to finding the HVP of a monotone polygon. However, since we can easily obtain

the sorted order of the vertical coordinates we use this version since the proof is so easy and direct.

Lemma 4.2. *Suppose E is a set of inner-disjoint edges such that we know the sorted order of the vertical coordinates of the endpoints of the edges. Then in $O(|E|)$ time we can partition E into a family of fences, $F^{i,j}$, such that for any point z there are $O(\log|E|)$ fences in the family which can contain the horizontal neighbors of z, and those fences can be determined in $O(\log|E|)$ time.*

Proof. Let $m = |E|$ and let $y_1, \ldots, y_{2m}$ be the vertical coordinates of the endpoints of the edges of E in (increasing) sorted order. We can assume we know the rank of each vertical coordinate, since otherwise we can compute this is $O(|E|)$ time from the sorted order. For each edge e let $l(e) = e_2 - e_1$, where y_{e_1} and y_{e_2} are the vertical coordinates of the two endpoints, with $y_{e_1} \leq y_{e_2}$. We define

$$F^{i,j} = \{e: 2^i \leq l(e) < 2^{i+1}, j2^i \leq e_2 < (j+1)2^i\}.$$

It is easy to see that each nonempty $F^{i,j}$ is a fence since every edge intersects the line $y = y_{j2^i}$. For each edge in E we can clearly determine to which $F^{i,j}$ it belongs in constant time, and hence we can obtain the partition into fences in $O(|E|)$ time. Finally, for any point z let z_y be the vertical coordinate of z. By binary search we can determine the index p such that $y_p \leq z_y < y_{p+1}$ in $O(\log|E|)$ time. Now for each i there are at most two values of j such that $F^{i,j}$ can contain a horizontal neighbor of z, and these can be clearly determined in constant time once we know p. □

The construction used to separate the edges into fences in the preceding proof is essentially equivalent to (static) interval trees, which were introduced by Edelsbrunner [E].

Lemma 4.3. *Suppose E is a set of inner-disjoint rasterized edges, and that we are given a partition of E into disjoint sets, $E = E_1 \cup \cdots \cup E_t$. Specifically, for each edge e in E we are given the index i of the set E_i containing e. Then in $O(|E|)$ time we can construct data structures $D_1, \ldots, D_t$ such that for any point z we can find its horizontal neighbors in E_i in $O((\log|E_i|)^2)$ time.*

Proof. First note that in $O(|E|)$ total time we can obtain, for each E_i, the order of the vertical coordinates of its edges, by performing a double radix sort. Next, using the technique described in Lemma 4.2, we partition the edges in each E_i into the appropriate fences. Now for each edge in E we compute its wire-coordinate with respect to its fence. Again by performing a double radix sort, in $O(|E|)$ time we compute the sorted order of wire-coordinates and the sorted order of the vertical coordinates of the endpoints of the edges in each fence. Next, by Lemma 4.1, in $O(|E|)$ total time, we construct a data structure for each fence such that for any point we can find its horizontal neighbors in the fence in time logarithmic in the number of edges in the fence. Now for any point z and any i,

we can find the horizontal neighbors of z in E_i as follows. In $O(\log|E_i|)$ time we identify the $O(\log|E_i|)$ fences of edges in E_i which could possibly contain the horizontal neighbors of z. Now, for each of these fences, in $O(\log|E_i|)$ time we locate the horizontal neighbors of z in the fence. This takes a total of $O((\log|E_i|)^2)$ time. Finally we compare the horizontal neighbors found in each of these fences to find the true horizontal neighbors of z in E_i. □

Given Lemma 4.3, it is fairly straightforward to modify the algorithm in Section 2 to run in $O(n \log^* n)$ time on rasterized chains. The overall structure of the algorithm is that we partition the chain into subchains of length $\log^2 n$, apply the algorithm recursively to compute the WP of each subchain, and then compute the WP of the entire chain using a modification of Theorem 2.1, which we state below.

Theorem 2.1 (Rasterized Version). *Suppose $k \geq \log^2(r/2)$. Given a k-uniform partition of an r-vertex nondegenerate polygon P and a data structure so that for any point z we can find the interior horizontal neighbors of z in P in $O(k)$ time, we can compute the WP of the interior of P in $O(r)$ time.*

The proof of this version of Theorem 2.1 is identical to the proof of the previous one except for two points. First, the assumption about the data structure eliminates the need for Lemma 2.2, and, second, we need to have data structures for the bad chunks in order to apply the theorem recursively. However, this is trivial since the data structure for P works for all the chunks as well. Thus, the only remaining issue is now to ensure that the necessary data structure is available every time Theorem 2.1 is applied at a "top" level. We do this in $O(n \log^* n)$ time by $\log^* n$ applications of Lemma 4.3 right at the beginning of the overall algorithm, so that all the necessary data structures are computed before doing anything else. More precisely, we first let E be the set of edges in the doubling of the original chain and apply Lemma 4.3 to the trivial partition $E = E_1$. This provides the data structure that will be needed when Theorem 2.1 is applied to the doubling of the original chain.

We next partition the chain into subchains of length $\log^2 n$ and let $E = E_1 \cup \cdots \cup E_t$ where E_i is the set of edges in the doubling of the ith subchain. Applying Lemma 4.3 provides the necessary data structures for the applications of Theorem 2.1 where the polygon is the doubling of one of these subchains. We repeat this process with partitions of the original chain into subchains of length $\log^2(\log^2 n)$, then $\log^2(\log^2(\log^2 n))$, etc., corresponding to the recursive structure of the overall algorithm. Since each application of Lemma 4.3 requires $O(n)$ time, the total time needed is $O(n \log^* n)$.

References

[BT] Bhattacharya, B. K., and Toussaint, G. T., A linear algorithm for determining the translation separability of two simple polygons, Report SOCS-86.1, School Comput. Sci., McGill University, Montreal, 1986.

[C1] Chazelle, B., A theorem on polygon cutting with applications, *Proc. 23rd Annual Symp. on Foundations of Computer Science*, 1982, pp. 339–349.

[C2] Chazelle, B., Triangulating a simple polygon in linear time, Princeton Tech. Report CS-TR-264-90, 1990.

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

[CTV] Clarkson, K., Tarjan, R., and Van Wyk, C., A fast Las Vegas algorithm for triangulating a simple polygon, *Proc. 4th ACM Symp. on Computational Geometry*, 1988, pp. 18–22 (Princeton Tech. Report CS-TR-157-88, 1988).

[E] H. Edelsbrunner, Dynamic data structures for orthogonal intersection queries, Report f59, Techn. Univ. Graz, Instit. Informationsverarb., Graz, 1980.

[EGS] Edelsbrunner, H., Guibas, L. J., and Stolfi, J., Optimal point location in a monotone subdivision, *SIAM J. Comput.* **15** (1986), 317–340.

[FFR] Fiume, E., Fournier, A., and Rudolph, L., A parallel scan conversion algorithm with anti-aliasing for a general-purpose ultra-computer, *ACM Trans. Comput. Graphics* **17**(3) (1983), 141–150.

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

[FNTV] Fung, K. Y., Nicholl, T. M., Tarjan, R. E., and Van Wyk, C. J., Simplified linear-time Jordan sorting and polygon clipping, Princeton Tech. Report CS-TR-189-88, 1988.

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

[GHL$^+$] Guibas, L., Hershberger, J., Leven, D., Sharir, M., and Tarjan, R. E., Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons, *Algorithmica* **2** (1987), 209–233.

[HM] Hertel, S., and Mehlhorn, K., Fast triangulation of a simple polygon, *Proc. Conf. on Foundations of Computer Theory*, Lecture Notes on Computer Science, Vol. 158, Springer-Verlag, Berlin, 1983, pp. 207–218.

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

[KK] Keil, J. M., and Kirkpatrick, D. G., Computational geometry on integer grids *Proc. 19th Annual Allerton Conference on Communication Control and Computing*, 1981, pp. 41–50.

[K] Kirkpatrick, D. G., Optimal search in planar subdivisions, *SIAM J. Comput.* **12**(1) (1983), 28–35.

[L] Lee, D. T., Shading of regions on vector display devices, *ACM Trans. Comput. Graphics* **15**(3) (1981), 34–44.

[LTL] Liou, W. T., Tan, J. J. M., and Lee, R. C. T., Minimum partitioning simple rectilinear polygons in $O(n \log \log n)$ time, *Proc. 5th ACM Symp. on Computational Geometry*, 1989, pp. 344–353.

[TV] Tarjan, R. E., and Van Wyk, C. J., An $O(n \log \log n)$-time algorithm for triangulating a simple polygon, *SIAM J. Comput.* **17**(1) (1988), 143–178.

[T1] Toussaint, G., Pattern recognition and geometrical complexity, *Proc. 5th Intern. Conf. on Pattern Recognition*, 1980, pp. 1324–1347.

[T2] Toussaint, G., Computational geometry and morphology, Tech. Report SOCS-86.3, McGill University, Montreal, 1986.

[T3] Toussaint, G., An output-complexity-sensitive polygon triangulation algorithm, Tech. Report SOCS-88.11, McGill University, Montreal, 1988.

[T4] Toussaint, G. (ed.), *Computational Morphology*, North-Holland, Amsterdam, 1988.

[TA] Toussaint, G., and Avis, D., On a convex hull algorithm for polygons and its application to triangulation problems, *Pattern Recognition* **15**(1) (1982), 23–29.

Received May 29, 1990, and in revised form January 25, 1991.