1992

Testing a Simple Polygon for Monotonicity Optimally in Parallel

Danny Z. Chen

Sumanta Guha

Report Number:
92-027

TESTING A SIMPLE POLYGON FOR MONOTONICITY OPTIMALLY IN PARALLEL

Danny Z. Chen
Sumanta Guha

CSD-TR-92-027
April 1992
Revised 9/92

Testing a Simple Polygon for Monotonicity Optimally in Parallel

Danny Z. Chen* Sumanta Guha†

Abstract

We show that, in parallel, an n-vertex simple polygon can be tested for monotonicity optimally in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors, and we present two different optimal parallel algorithms for solving this problem. Our result leads to an optimal parallel algorithm for triangulating simple polygons that runs in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors if the polygons are monotone.

Keywords: Parallel algorithms, computational geometry, simple polygons

1 Introduction

In [17], Preparata and Supowit show how to test a simple polygon for monotonicity sequentially in optimal linear time. In this paper, we present two different optimal parallel algorithms for testing the monotonicity of an n-vertex simple polygon, and we discuss a consequence of our result for the parallel triangulation of simple polygons. Our parallel algorithms are based on the geometry of Preparata and Supowit [17]. However, the algorithm in [17] appears to be essentially sequential while our algorithms require considerable machinery from the theory of parallel computation. Both our algorithms run in $O(\log n)$ time using $O(n/\log n)$ processors. The first algorithm is conceptually very simple and consists of a reduction from the monotonicity test problem to the problem of computing in parallel the visibility of nonintersecting line segments from a point [2]. The second algorithm is perhaps more practical and is based on a square-root divide-and-conquer strategy. The parallel computational model we use is the EREW PRAM. For a discussion of the PRAM models and in particular the EREW PRAM, we refer to [13].

We need to introduce some definitions. Let P be a simple polygon with vertices v_0, $v_1, \ldots, v_{n-1}$ in counterclockwise order around its boundary (we denote the boundary of P

*Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556. This research was partially done when this author was with Dept. of Computer Science, Purdue University, West Lafayette, Indiana and was supported in part by the Office of Naval Research under Grants N00014-84-K-0502 and N00014-86-K-0689, the National Science Foundation under Grant DCR-8451393, and the National Library of Medicine under Grant R01-LM05118.

†EE&CS Department, University of Wisconsin-Milwaukee, PO Box 784, Milwaukee, WI 53201. This research was partially done when this author was with the EECS Dept., University of Michigan, Ann Arbor and was supported in part by NSF/DARPA Grant CCR-9004727.

by $bd(P)$). The edge $\overline{v_i v_{i+1}}$ of P is denoted by e_i. (Throughout this paper, all indices are taken modulo of n.) A *chain* $C_{ij} = (e_i, e_{i+1}, \ldots, e_{j-1})$ is a sequence of consecutive edges on $bd(P)$. Chain C_{ij} is said to be *monotone with respect to a straight line* l if for any line l' that is orthogonal to l, $C_{ij} \cap l'$ consists of at most one point. Polygon P is *monotone* if and only if there exists a line l such that $bd(P)$ can be split into two chains C_{ij} and C_{ji} both monotone with respect to l. If this is the case, then P is said to be monotone with respect to l (see Figure 1 for example). Two points p and q in P are *visible* to each other if and only if segment $\overline{pq}$ (whose endpoints are p and q) lies completely within P. P is said to be *star-shaped* if there exists a point $q \in P$ such that q is visible to every other point of P (see Figure 2 for example).

In this paper, we shall prove the following theorem:

Theorem 1 *Whether an n-vertex simple polygon P is monotone can be decided optimally in $O(\log n)$ time using $O(n/\log n)$ processors in the EREW PRAM model, and, if so, a line l with respect to which P is monotone can be obtained within the same complexity bounds.*

Chazelle's recent breakthrough in finding an optimal sequential linear time algorithm for triangulating a simple polygon [4] settled probably one of the most important outstanding questions in computational geometry. However, parallel algorithms for triangulating simple polygons still lag by a factor of $\log n$ (except in the CRCW PRAM model). Currently, the best PRAM algorithms for triangulating an n-vertex simple polygon run in $O(\log n)$ time using either $O(n)$ processors on the CREW PRAM [10, 18] or $O(n/\log n)$ processors on the CRCW PRAM [11]. (The EREW PRAM, of course, is less powerful than the CREW PRAM, which is less powerful than the CRCW PRAM.) Goodrich [10] shows that if it is already given that a polygon is monotone with respect to a particular line, then the polygon can be triangulated optimally in $O(\log n)$ time using $O(n/\log n)$ CREW PRAM processors. Cole and Goodrich [7] show how to test a simple polygon for star-shapedness optimally in $O(\log n)$ time using $O(n/\log n)$ CREW PRAM processors. Chen [5] gives optimal parallel algorithms for testing the star-shapedness of simple polygons and for triangulating monotone and star-shaped polygons, all in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM.

A consequence of Theorem 1 is that simple polygons of certain types, in particular monotone (and star-shaped) polygons, can be detected and triangulated optimally in parallel. Together with Theorem 1, we then have:

Theorem 2 *An n-vertex simple polygon can be triangulated optimally in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors if the polygon is monotone or star-shaped.* □

The rest of the paper is for proving Theorem 1. The next section reviews some definitions and observations from [17]. Sections 3 and 4 present our two algorithms, respectively. Section 5 mentions some open problems.

2 Preliminaries

To avoid undue repetition, we assume familiarity with the paper by Preparata and Supowit [17] and restrict ourselves to quoting a few relevant definitions and results.

The *length* of $C_{ij} = (e_i, e_{i+1}, \ldots, e_{j-1})$, denoted by $|C_{ij}|$, is k if C_{ij} consists of k edges. For example, $|C_{ij}| = j - i$ for $i < j$, and $|C_{ij}| = n - i + j$ for $i > j$. The boundary of P, $bd(P)$, is a *close* chain of length n.

The *polar diagram* of the input polygon P is defined as follows. For each edge e_i of P, draw a semi-infinite ray from the origin O in the direction from v_i to v_{i+1}. Without risk of confusion we denote this ray by e_i too. The polar rays $e_0, e_1, \ldots, e_{n-1}$ together partition the polar range $[0, 2\pi)$ into n consecutive wedges (a *wedge* is a sector in the polar diagram bounded by two polar rays). Note, of course, that e_{i+1} may not be adjacent to e_i in the polar ordering. Suppose these wedges are $\beta_0, \beta_1, \ldots, \beta_{n-1}$ in counterclockwise order starting from β_0, where β_0 is the wedge on the counterclockwise side of e_0. Let α_i, $0 \leq i \leq n-1$, be the wedge from e_i counterclockwise to e_{i+1} if the angle from e_i counterclockwise to e_{i+1} is $\leq 180°$, and the wedge from e_i clockwise to e_{i+1}, otherwise. Given a chain C_{ij}, define $\alpha(C_{ij})$ to be the wedge $\cup_{k=i}^{j-1}\alpha_k$. Obviously, $\alpha(C_{ij})$ is one contiguous wedge because $\alpha_k \cup \alpha_{k+1}$ is a connected component for every $k \in \{i, i+1, \ldots, j-2\}$.

For a wedge $\beta_{k'} \subseteq \alpha(C_{ij})$, we define the *multiplicity* of $\beta_{k'}$ *with respect to* $\alpha(C_{ij})$ to be $|\{\alpha_k \mid \beta_{k'} \subseteq \alpha_k, k \in \{i, i+1, \ldots, j-1\}\}|$, i.e., the number of wedges α_k that contain the given $\beta_{k'}$. It is not difficult to see that for every $0 \leq k' \leq n-1$, the multiplicity of $\beta_{k'}$ with respect to $\alpha(bd(P))$ is no smaller than 1 as the boundary of P is not self-intersecting. If each of a sequence of consecutive wedges $\beta_{k'}$ has multiplicity m with respect to $\alpha(C_{ij})$, then we say that the wedge which is the union of all the $\beta_{k'}$'s in the sequence also has multiplicity m with respect to $\alpha(C_{ij})$. Two wedges are said to be *antipodal* if their union contains a line passing through the origin.

The following lemma characterizes the monotonicity of a simple polygon.

Lemma 1 (Preparata and Supowit [17]) *A simple polygon P is monotone if and only if its polar diagram contains at least one pair of antipodal wedges β_i and β_j both of multiplicity 1 with respect to $\alpha(bd(P))$. If this is the case, then P is monotone with respect to any infinite line contained in the union of two such antipodal wedges.* □

Because of Lemma 1, whenever the multiplicity of a wedge is larger than 1, that multiplicity is simply taken to be 2. The following lemma is also useful in our algorithms.

Lemma 2 (Preparata and Supowit [17]) *Given a chain C_{ij}, one can in time $O(|C_{ij}|)$ compute the wedge $\alpha(C_{ij})$, as well as a partition of $\alpha(C_{ij})$, by some subset of the polar rays from chain C_{ij}, into $O(|C_{ij}|)$ consecutive wedges alternately having labels 1 and 2. Wedges with label 1 all have multiplicity 1 with respect to $\alpha(C_{ij})$, and wedges with label 2 all have multiplicity 2 with respect to $\alpha(C_{ij})$.* □

Suppose that we partition $bd(P)$ into $n/\log n$ chains $C_{0,\log n}, C_{\log n,2\log n}, \ldots, C_{n-\log n,0}$ of length $\log n$ each. We denote each $C_{(i-1)\log n,i\log n}$ by bd_i. Then by Lemma 2, each $\alpha(bd_i)$ (together with its partition as stated in Lemma 2) can be computed in $O(\log n)$ time using one processor. Furthermore, this computation can be done for all the bd_i's in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors (by assigning one processor to each bd_i). Henceforth, we assume that this computation has been done for all the bd_i's, and we store the partition of each $\alpha(bd_i)$ (with its labels) in an array of size $O(\log n)$.

3 The First Algorithm

This section gives a parallel algorithm for testing the monotonicity of P that is conceptually very simple. This algorithm reduces the monotonicity problem to a problem that can be solved by using the cascading divide-and-conquer technique [2, 6].

Suppose that each $\alpha(bd_i)$ is bounded by two polar rays p_i and q_i and is from p_i counterclockwise to q_i. Without loss of generality (WLOG), we assume that no $\alpha(bd_i)$ consists of only a single polar ray (i.e., for every i, $\alpha(bd_i) \neq \{p_i\}$). Let the polar angle of a polar ray r be $\theta(r)$.

We now transform each $\alpha(bd_i)$ to a horizontal line segment s_i in a plane $\mathcal{P}$. We assume that plane $\mathcal{P}$ has the x and y coordinates, and that the points (x, y) and $(x + 2\pi, y)$ are identical in $\mathcal{P}$ for all values x and y. Hence topologically $\mathcal{P}$ is a cylinder. The transformation is as follows. Let point $l_i = (\theta(p_i), i)$ and point $r_i = (\theta(q_i), i)$ in $\mathcal{P}$. If $\theta(p_i) < \theta(q_i)$, then let $s_i = \overline{l_i r_i}$; if $\theta(p_i) > \theta(q_i)$ or $|\alpha(bd_i)| = 2\pi$, then let $s_i = \overline{(0, i)r_i} \cup \overline{l_i(2\pi, i)}$. Note that s_i is one contiguous segment in $\mathcal{P}$. This transformation gives us $n/\log n$ horizontal line segments in $\mathcal{P}$. Later in this section, when we refer to "visibility" in $\mathcal{P}$, we assume that the only "opaque" objects in $\mathcal{P}$ are the $n/\log n$ segments s_i. Note that this transformation can be easily done in $O(1)$ time using $O(n/\log n)$ EREW PRAM processors.

The following lemmas are essential to the algorithm in this section.

Lemma 3 *For any $\alpha(bd_i)$, suppose that a wedge $w \subseteq \alpha(bd_i)$ has label 1 with respect to $\alpha(bd_i)$. Then w has multiplicity 1 with respect to $\alpha(bd(P))$ if and only if the portion of the horizontal segment in $\mathcal{P}$ corresponding to w is completely visible to* both *the points $(0, +\infty)$ and $(0, -\infty)$ in $\mathcal{P}$.*

Proof. Let the portion of the segment in $\mathcal{P}$ corresponding to w be $s(w)$. If $s(w)$ is completely visible to $(0, +\infty)$ (resp., $(0, -\infty)$) in $\mathcal{P}$, then there is no chain bd_j (resp., $bd_{j'}$) such that $j > i$ (resp., $j' < i$) and that $\alpha(bd_j)$ (resp., $\alpha(bd_{j'})$) contains a polar ray in the interior of w. Hence the multiplicity of w with respect to $\alpha(bd(P))$ is the same as the multiplicity of w with respect to $\alpha(bd_i)$. If the multiplicity of w with respect to $\alpha(bd(P))$ is 1, then there is no $e_k \subset bd_j$ such that $j \neq i$ and that a polar ray in the interior of w is contained in α_k. But this implies that $s(w)$ is completely visible to both $(0, +\infty)$ and $(0, -\infty)$ in $\mathcal{P}$. □

Lemma 4 *The portions of every s_i that are visible to $(0, +\infty)$ (resp., $(0, -\infty)$) form at most two connected components.*

Proof. We only prove the case for $(0, +\infty)$ because the case for $(0, -\infty)$ is similar. Among the $n/\log n$ horizontal segments in $\mathcal{P}$, only those s_j's such that $j > i$ may affect the visibility of s_i from $(0, +\infty)$ because they are all above s_i. Let $B_i = \cup_{j>i} bd_j$. Then B_i is a contiguous chain, and hence $\alpha(B_i)$ is a contiguous wedge. Thus the portion of s_i that is hidden by these s_j's from $(0, +\infty)$ in $\mathcal{P}$ forms at most one connected component, and the lemma follows. □

Corollary 1 *The portions of every s_i that are visible to* both *$(0, +\infty)$ and $(0, -\infty)$ form $O(1)$ connected components.*

Proof. An easy consequence of Lemma 4. □

The rest of the algorithm goes as follows. (1) Compute the portions of each s_i that are visible from both $(0, +\infty)$ and $(0, -\infty)$ in $\mathcal{P}$. (2) Sort the portions of all the s_i's that are visible to both $(0, +\infty)$ and $(0, -\infty)$ according to the x-coordinates of their endpoints. (3) For each i, based on the outcome of step (1), obtain from the partition of $\alpha(bd_i)$ the set W_i of wedges with multiplicity 1 with respect to $\alpha(bd(P))$. (4) Find all the antipodal pairs of wedges in the union of all the W_i's.

Step (1) can be done in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors by using the algorithm in [2]. Step (2) takes $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors by using [6] since there are totally $O(n/\log n)$ endpoints. Step (3) is easily done in $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors (by using one processor for each W_i). Step (4) can be done by using an algorithm which is very similar to the one

used by Goodrich for computing in parallel a farthest pair of points in a convex polygon [9]. The algorithm in [9] computes all the antipodal pairs of wedges for n wedges sorted by the cyclic ordering of polar angles, in $O(\log n)$ time using $O(n/\log n)$ CREW PRAM processors. In our situation, the number of wedges with multiplicity 1 is $O(n)$ and those wedges are already sorted (based on step (2)). We basically follow the steps in [9], except that we use an EREW PRAM merging algorithm [3, 12] in Step 4 of [9]. All other steps of [9] can be easily implemented on the EREW PRAM in the same complexity bounds. In total, the algorithm for testing the monotonicity of a simple polygon in this section takes $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors.

4 The Second Algorithm

We now present an algorithm that uses a square-root divide-and-conquer strategy. This algorithm is perhaps much easier to be adapted for other parallel computational models than the algorithm in the previous section. We focus only on the computation of finding all the wedges with multiplicity 1 with respect to $\alpha(bd(P))$ because the rest of the algorithm (i.e., for obtaining all the antipodal pairs of wedges with multiplicity 1) is same as in Section 3.

In this section, instead of transforming each $\alpha(bd_i)$ to a line segment in a 2-dimensional space, we view $\alpha(bd_i)$ as an arc $A_i \subseteq [0, 2\pi)$ on the unit circle C centered at the origin ($A_i = [0, 2\pi)$ if $|A_i| = 2\pi$). In total, there are $n/\log n$ arcs A_i. For a chain C on $bd(P)$, we denote the arc corresponding to $\alpha(C)$ by $A(C)$.

The following fact can be easily observed: For a bd_i, a wedge $w \subseteq \alpha(bd_i)$ has multiplicity 2 if and only if either **(i)** w has multiplicity 2 with respect to $\alpha(bd_i)$ or **(ii)** w is contained in $\alpha(bd(P) - bd_i)$. If the information for both (i) and **(ii)** is available for all i, then the rest of the computation is same as in the previous section (i.e., using Corollary 1 to find in each $\alpha(bd_i)$ the wedges with multiplicity 1 with respect to $\alpha(bd(P))$ and using a modification of the algorithm in [9] to compute all the antipodal pairs of wedges with multiplicity 1). The information on **(i)** is already computed in Section 2. The computation for **(ii)**, that is, finding $\alpha(bd_i) \cap \alpha(bd(P) - bd_i)$ (i.e., $A(bd(P) - bd_i) \cap A_i$) for each bd_i, is the main task of this algorithm, and we only discuss how to compute $A(bd(P) - bd_i) \cap A_i$ for every i, $i = 1, 2, \ldots, n/\log n$.

The algorithm is based on the following lemma.

Lemma 5 *For any two chains C' and C'' on $bd(P)$ that are disjoint except at their endpoints, let w be the wedge corresponding to the intersection $A(C') \cap A(C'')$ (i.e., $w = \alpha(C')$*

$\cap\ \alpha(C''))$. *Then w has multiplicity 2 with respect to $\alpha(bd(P))$.*

Proof. Easy and omitted. □

By Lemma 5, $A(C') \cap A(C'')$ can be represented by a single arc on C (which corresponds to $w = \alpha(C') \cap \alpha(C'')$) with multiplicity 2. Using this representation, if $A(C')$ (resp., $A(C'')$) is partitioned into $k_{C'}$ (resp., $k_{C''}$) subarcs which alternately have multiplicities 1 and 2, then $A(C' \cup C'')$ can be partitioned into at most $k_{C'} + k_{C''} + 1$ subarcs which alternately have multiplicities 1 and 2. We use an array to represent the partition of $A(C)$ (into subarcs which alternately have multiplicities 1 and 2) for each chain C processed in the algorithm. The algorithm recursively computes the partition for a chain C from the partitions for its subchains based on a square-root divide-and-conquer strategy.

Now, because only $A(bd(P) - bd_i) \cap A_i$ is computed for every i, we simply assume that A_i is one single arc with multiplicity 1 with respect to $\alpha(bd_i)$ (i.e., we ignore the partitioning information for (i) in this computation). Hence the input to the procedure below consists of an ordered set $S^* = \{A_1, A_2, \ldots, A_{n/\log n}\}$ of $n/\log n$ arcs, and each arc A_i has multiplicity 1. We call a subset S of S^* consisting of consecutive A_i's a *contiguous subset* of S^*. Note that for each contiguous subset $S = \{A_j, A_{j+1}, \ldots, A_k\}$ of S^*, $\cup_{l=j}^{k} A_l$ is one contiguous arc, and we denote it by $U(S)$.

The procedure for computing $A(bd(P) - bd_i) \cap A_i$ for all i is as follows.

Input. A contiguous subset S of S^* with $|S| = m$.

Procedure P(S, m)

If $m = 1$, then return the only element in S;

otherwise,

(1) partition S into $g = m^{1/2}$ contiguous subsets $S_1, S_2, \ldots, S_g$ of size $m^{1/2}$ each,

(2) recursively solve the g subproblems in parallel, and

(3) compute the partition of $U(S)$ by using the partitions of $U(S_1), U(S_2), \ldots, U(S_g)$ (each stored in an array), with m processors and in $O(\log m)$ time, storing the partition of $U(S)$ in an array.

If we could perform the conquer stage of P$(S, |S|)$ in $O(\log |S|)$ time using $O(|S|)$ EREW PRAM processors, then it is easy to see that P$(S^*, n/\log n)$ would run in totally $O(\log n)$ time using $O(n/\log n)$ EREW PRAM processors. Therefore, we only need to show how to perform the conquer stage computation in the claimed complexity bounds.

In the conquer stage, we have already computed recursively the partitions of $U(S_1)$, $U(S_2)$, $\ldots$, $U(S_g)$, which are stored, say, in arrays $L_1, L_2, \ldots, L_g$, respectively. We want

to compute the partition of $U(S)$ (whose subarcs alternately have multiplicities 1 and 2) and store the partition in array L_S. In order to do that, we compute in parallel, for each S_j, $U(S - S_j) \cap S_j$, and store $U(S - S_j) \cap S_j$ and the partition of $U(S_j) - U(S - S_j)$ in appropriate locations of L_S.

For a subset S_j of S, we let $Pre_j = \{U(S_1), U(S_2), \ldots, U(S_{j-1})\}$ and $Suc_j = \{U(S_{j+1}), U(S_{j+2}), \ldots, U(S_g)\}$. If both $U(Pre_j)$ and $U(Suc_j)$ are available, then the portions of $U(S_j)$ that overlap with $U(Pre_j) \cup U(Suc_j)$ can be easily obtained, because each of $U(S_j)$, $U(Pre_j)$, and $U(Suc_j)$ is a contiguous arc. WLOG, we only show the computation with respect to Pre_j since the computation with respect to Suc_j is similar.

The computation of the conquer stage on Pre_j is as follows. First, for each j, broadcast in parallel the endpoints of each $U(S_j)$ to all Pre_k's, for $k < j$. The two endpoints of each $U(S_j)$ defines an arc on C. Hence each Pre_j consists of $O(m^{1/2})$ arcs. Then for every j in parallel, $U(Pre_j)$ can be computed in $O(\log m)$ time using $O(m^{1/2})$ processors on the EREW PRAM by using the preprocess procedure in the parallel algorithm for computing a minimum circle-cover [1] (given a set of arcs on a circle, the minimum circle-cover problem is to compute a subset of arcs of minimum size such that the union of the arcs in the subset covers the whole circle). Since $U(Pre_j)$ is a contiguous arc, $U(Pre_j) \cap U(S_j)$ is also a contiguous arc (with multiplicity 2) and can be computed in $O(\log m)$ time using one processor.

Given, for every j, $U(S - S_j) \cap U(S_j)$ (which has multiplicity 2) and the partition of $U(S_j) - U(S - S_j)$ (which is available from L_j), we compute L_S that contains the partition of $U(S)$ whose subarcs alternately have multiplicities 1 and 2, one subarc per element in L_S. We claim that $|L_S| = O(m)$. This claim can be easily proved by induction (by assuming $|L_j| = O(m^{1/2})$ and by using the fact that $U(Pre_j) \cap U(S_j)$ is at most one subarc of $U(S)$ with multiplicity 2 for each j). Now, observe that the arcs in $\Omega = \{U(S_j) - U(S - S_j) \mid j = 1, 2, \ldots, m^{1/2}\}$ are pairwise disjoint (except possibly at their endpoints). Remove from Ω all the empty arcs and denote the remaining set still by Ω. Sort the $O(m^{1/2})$ arcs in Ω according to their endpoints (in $O(\log m)$ time using $O(m^{1/2})$ EREW PRAM processors [6]). Then we have a partition of $U(S)$ of the form

$$X_S = U(S'_1) - U(S - S_j),\ I_1,\ U(S'_2) - U(S - S_j),\ I_2,\ \ldots,\ U(S'_{|\Omega|}) - U(S - S_j),\ I_{|\Omega|},$$

where every $U(S'_j) = U(S_k)$ for some k and I_j is the arc between $U(S'_j) - U(S - S_j)$ and $U(S'_{j+1}) - U(S - S_j)$. Note that it is possible that $I_j = \emptyset$ and that for all but at most one j, if $I_j \neq \emptyset$, then I_j has multiplicity 2. By using parallel prefix [14, 15], the partition X'_S of $U(S)$ whose subarcs alternately have multiplicities 1 and 2 can be easily obtained from

X_S in $O(\log m)$ time using $O(m/\log m)$ EREW PRAM processors, because $|X_S| = O(m)$. Then X'_S is stored in array L_S in $O(1)$ time using $O(m)$ EREW PRAM processors (the k-th subarc in X'_S is stored in the k-th element of L_S).

Overall, the conquer stage takes $O(\log m)$ time using $O(m)$ EREW PRAM processors.

5 Conclusion

An important problem is, of course, that of finding an optimal parallel algorithm to triangulate arbitrary simple polygons. Goodrich has solved this problem on the CRCW PRAM [11]. Whether the same bounds as in [11] can be obtained for triangulating simple polygons on the CREW PRAM and EREW PRAM still remains open. Till that is achieved, however, an interesting endeavor would be to prove theorems like Theorem 2 for other classes of simple polygons. This is in the spirit of the papers by ElGindy and Toussaint [8] and Lee and Chwa [16] where the authors, prior to Chazelle's discovery [4], try to identify "triangulation-linear" classes of simple polygons which admit of triangulation in optimal sequential time.

References

[1] M. J. Atallah and D. Z. Chen, "An optimal parallel algorithm for the minimum circle-cover problem," *Information Processing Letters* **32** (1989), pp. 159-165.

[2] M. J. Atallah, R. Cole, and M. T. Goodrich, "Cascading divide-and-conquer: a technique for designing parallel algorithms," *SIAM J. Computing* **18** (1989), pp. 499-532.

[3] G. Bilardi and A. Nicolau, "Adaptive bitonic sorting: An optimal parallel algorithm for shared-memory machines," *SIAM J. Computing*, **18** (2) (1989), pp. 216-228.

[4] B. Chazelle, "Triangulating a simple polygon in linear time," *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990, pp. 220-230.

[5] D. Z. Chen, "Efficient geometric algorithms in the EREW-PRAM," *Proc. 28th Annual Allerton Conf. on Communication, Control, and Computing*, 1990, pp. 818-827.

[6] R. Cole, "Parallel merge sort," *SIAM J. Computing* **17** (1988), pp. 770-785.

[7] R. Cole and M. T. Goodrich, "Optimal parallel algorithms for polygon and point set problems," *Proc. 4th ACM Symp. on Computational Geometry*, 1988, pp. 201-210.

[8] H. ElGindy and G. T. Toussaint, "On geodesic properties of polygons relevant to linear time triangulation," *The Visual Computer* 5 (1989), pp. 68-74.

[9] M. T. Goodrich, "Efficient parallel techniques for computational geometry," Dept. of Comp. Sci., Purdue University, West Lafayette, Indiana, 1987.

[10] M. T. Goodrich, "Triangulating a polygon in parallel," *J. Algorithms* **10** (1989), pp. 327-351.

[11] M. T. Goodrich, "Planar separators and parallel polygon triangulation," *Proc. of the 24th Annual ACM Symposium on Theory of Computing*, 1992.

[12] T. Hagerup and C. Rub, "Optimal merging and sorting on the EREW PRAM," *Information Processing Letters* **33** (4) (1989), pp. 181-185.

[13] R. Karp and V. Ramachandran, "Parallel Algorithms for Shared-Memory Machines," "*Handbook of Theoretical Computer Science*. Edited by J. van Leeuwen, Volume 1, Elsevier Science Publishers, 1990,

[14] C. P. Kruskal, L. Rudolph, and M. Snir, "The power of parallel prefix," *IEEE Trans. Computers* **C-34** (10) (1985), pp. 965-968.

[15] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *J. of the ACM* **27** (4) (1980), pp. 831-838.

[16] S. H. Lee and K. Y. Chwa, "A new triangulation-linear class of simple polygons," *International J. Computer Mathematics* **22** (1987), pp. 135-147.

[17] F. P. Preparata and K. J. Supowit, "Testing a simple polygon for monotonicity," *Information Processing Letters* **12** (1981), pp. 161-164.

[18] C. K. Yap, "Parallel triangulation of a polygon in two calls to the trapezoidal map," *Algorithmica* **3** (1988), pp. 279-288.