

# Optimal Mesh Algorithms for Proximity and Visibility Problems in Simple Polygons \*

Sumanta Guha †

Department of Electrical Engineering & Computer Science  
University of Wisconsin-Milwaukee  
3200 N. Cramer Street  
Milwaukee, WI 53211  
USA

Email: guha@cs.uwm.edu

Fax: 414-229-6958

## Abstract

We present optimal parallel algorithms that run in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh-connected computer for a number of fundamental problems concerning proximity and visibility in a simple polygon. These include computing shortest paths, shortest path trees, shortest path partitions, all-farthest neighbors, the visibility polygon of a point, the weak visibility polygon of an edge, and the ray shooting problem.

*Keywords:* Computational geometry, mesh-connected computer, parallel algorithm, simple polygon, shortest path, visibility, farthest neighbor, ray shooting.

## 1 Introduction

Simple polygons form a fundamental class of objects in computational geometry. Proximity and visibility problems in simple polygons, in particular, arise in diverse applications and have been extensively studied.

The distance between two points  $p$  and  $q$  inside a simple polygon  $P$  is, of course, their *geodesic distance*, which is the length of the shortest internal path joining  $p$  and  $q$ . This path is known [18] to be a polygonal line whose corners are vertices of  $P$ . And, if this path is a straight line, then  $q$  is said to be *visible* from  $p$ , and vice versa.

Optimal serial algorithms are now known for almost all proximity and visibility problems in simple polygons. See, e.g., [7, 15] for some recent advances. Recently, interest has also grown in the parallel complexity of various problems concerning simple polygons. Efficient parallel algorithms have been reported for many, mostly on the PRAM (Parallel Random Access Machine) model of computation, e.g., [4, 11, 12, 23] are a few of those relevant to problems considered here. In this paper we consider another platform for parallel processing, the mesh-connected computer (or, simply, mesh), which is not only theoretically attractive but has, in fact, been implemented by various commercial manufacturers. A mesh is an SIMD computer

---

\*A preliminary version of this paper appears in *Proceedings of the 7th IEEE International Parallel Processing Symposium.*, Newport Beach, 1993.

†Supported in part by a UW-Milwaukee Graduate School Research Committee Award and NSF Grant CCR-9710711.

consisting of  $n$  identical PEs arranged in a  $\sqrt{n} \times \sqrt{n}$  array. Each PE is assumed to have  $O(1)$  local memory, and able to perform standard arithmetic operations and communicate with each of its (at most) four neighbors in  $O(1)$  time.

We show the following fundamental proximity and visibility problems in a simple polygon  $P$  can be solved in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh, which is optimal as non-trivial communication in a  $\sqrt{n} \times \sqrt{n}$  mesh requires  $\Omega(\sqrt{n})$  time:

1. Given two points  $p$  and  $q$  in  $P$ , find the *shortest internal path* joining them.
2. Given a point  $p$  in  $P$ , find the *shortest path tree* from  $p$ , and the *shortest path partition* w.r.t.  $p$ .
3. For each vertex of  $P$ , find a *farthest neighbor*, where distances are measured by shortest paths lying inside  $P$ .
4. Given a point  $p$  in  $P$ , compute the *visibility polygon*  $Vis(P, p)$  consisting of all points in  $P$  visible from  $p$ .
5. Given an edge  $e$  of  $P$ , compute the *weak visibility polygon*  $Vis(P, e)$  consisting of all points in  $P$  visible from some point of  $e$ .
6. Given a point  $p$  in  $P$  and a direction  $u$ , compute the first intersection with the boundary of  $P$  of a ray emanating from  $p$  in the direction  $u$  (the *ray shooting problem*).

The main challenge in designing algorithms for the mesh or, in fact, any systolic network of processors is that there is no shared memory in which to store data but, rather, data is distributed over all processors and, additionally, inter-processor communication cost is non-constant. Indeed, a few recent advances in techniques for mesh algorithms form the take-off point for our work: namely, the *multisearch* technique of Atallah, Dehne, Miller, Rau-Chaplin, and Tsay [5], the *multipoint location* technique of Jeong and Lee [17], and the (so-called) *parallel binary search* technique of Miller and Stout [19]. We use these techniques to optimally adapt some existing serial and PRAM algorithms to the mesh. In particular, we claim originality not for any of the underlying algorithms described here, but rather in showing how to implement these optimally on a mesh to solve the problems listed above. We refer to Atallah [3] for a survey and an extensive bibliography of prior work on computational geometry in various parallel models including the mesh.

Throughout this paper we shall assume that an  $n$ -vertex simple polygon  $P$  is represented on a  $\sqrt{n} \times \sqrt{n}$  mesh by assigning to each processor, in an arbitrary manner, one edge of  $P$ , together with the position number and orientation of that edge in some cyclic order of the boundary of  $P$ . Polygonal lines, trees, and other planar structures will also be assumed to be represented similarly in a manner evident from the context.

All our algorithms use some form of *mesh divide-and-conquer*, which, for a  $\sqrt{n} \times \sqrt{n}$  mesh, consists (ideally) of splitting, in  $O(\sqrt{n})$  time, a problem of size  $n$  into four equal subproblems of size  $n/4$ , recursively solving each subproblem in a  $\sqrt{n}/2 \times \sqrt{n}/2$  quadrant of the mesh, and then combining, in  $O(\sqrt{n})$  time, the solutions returned by these recursive calls for a complete solution. In this case, a recurrence of the form

$$T(n) = T(n/4) + O(\sqrt{n}),$$

bounds the running time  $T(n)$  to  $O(\sqrt{n})$ . See, e.g., [17, 19] for applications of mesh divide-and-conquer.

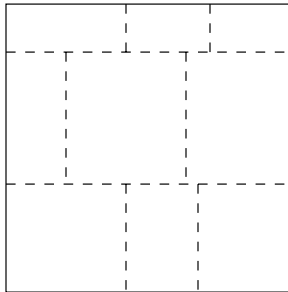


Figure 1: A mesh problem split into 9 subproblems.

For some of our results, we invoke a more generalized form of mesh divide-and-conquer, where we show how to split, in  $O(\sqrt{n})$  time, a problem of size  $n$  into some constant number of subproblems each of size at most  $\alpha n$ , where  $\alpha$  is a constant s.t.  $0 < \alpha < 1$ , and then combine the recursively obtained subsolutions, again in  $O(\sqrt{n})$  time. A little care is needed in bounding the running time here: the claim is that the problem can be solved in  $O(\sqrt{n})$  time on a *rectangular* mesh with *maximum side length*  $\sqrt{n}$  (clearly implying the result for square meshes). To justify the claim, again split each subproblem to get a total of some constant number of subproblems each of which fits on a rectangular submesh with maximum side length  $\leq \alpha\sqrt{n}$  (see Fig. 1). The recurrence bounding the running time then is

$$T(n) = T(\alpha n) + O(\sqrt{n}),$$

which again implies  $T(n) = O(\sqrt{n})$ .

It is very important to note, when using such a generalized mesh divide-and-conquer, that any algorithm that runs in  $O(\sqrt{n})$  time on a square mesh of size  $n$  (e.g., basic mesh operations such as sort, broadcast, etc.) can be run in  $O(\sqrt{n})$  time on a rectangular mesh with maximum side length  $\sqrt{n}$ . This follows from Atallah’s [2, Th. 1] results for simulating a square mesh with a rectangular one. Therefore, when invoking generalized mesh divide-and-conquer, we shall simply show how to split the given problem and combine the subsolutions in  $O(\sqrt{n})$  time, often implicitly performing optimal basic operations on a rectangular mesh.

The rest of the paper is organized as follows. The three fundamental mesh techniques that we make essential use of are briefly described in Sec. 2. Triangulation of simple polygons is reviewed in Sec. 3. In Sec. 4 we show how to compute shortest paths in simple polygons, as also two geometric structures based on shortest paths, the shortest path tree and the shortest path partition. Sec. 5 presents an application of these structures to solve the internal all-farthest neighbors problems for simple polygons. In Sec. 6 we discuss three fundamental visibility problems for simple polygons, including computing the visibility polygon of a point, the weak visibility polygon of an edge, and the ray shooting problem. We conclude in Sec. 7.

To be succinct, we shall throughout omit details concerning well-known basic operations on the mesh such as sort, broadcast, prefix computation, etc. (see [20, 22]), and tree computations (see [6]), that can be performed optimally, as well as avoid amplification when referring to algorithms that have already appeared in accessible literature (especially as these algorithms are, often, contributions of entire separate papers).

## 2 Fundamental Mesh Techniques

In the following three sections we briefly present fundamental mesh techniques that subsequently enable us to optimally adapt serial geometric algorithms to the mesh. In each case, we simply present the main result in a manner suitable for our applications and refer the reader to the original paper for further details.

### 2.1 Multisearch

A *hierarchical DAG* is a directed acyclic graph  $G = (V, E)$ , where each vertex has outdegree  $O(1)$ , and whose vertex set  $V$  can be partitioned into  $h$  levels  $L_0, \dots, L_h$ , where  $h = O(\log n)$ , such that every edge is from some  $L_i$  to  $L_{i+1}$ , and there are positive constants  $c_1, c_2$  and  $\mu$  such that  $|L_0| = 1$  and, for  $1 \leq i \leq h$ ,  $c_1\mu^i \leq |L_i| \leq c_2\mu^i$ . Clearly, a *balanced search tree* directed away from the root is a hierarchical DAG.

Let  $U$  be a universe of possible *search queries* on  $G$ , and  $f : (V \cup \text{start}) \times U \rightarrow V$  be a successor function such that, for every  $v \in V$  and  $q \in U$ ,  $(v, f(v, q)) \in E$ , and such that  $f(v, q)$  can be computed in  $O(1)$  time by a single processor that is given a description of both  $q$  and  $v$  (which must include a list of the outgoing edges from  $v$ ). Define the *search path* of a query  $q$ , denoted  $\text{path}(q)$ , to be the sequence of  $h$  vertices  $(v_1, \dots, v_h)$  of  $G$  given by  $f(\text{start}, q) = v_1$  and  $v_{i+1} = f(v_i, q)$ , for  $1 \leq i \leq h - 1$ . We say that a query  $q$  *visits a node  $v$  at time  $t$*  if, at time  $t$ , the mesh is in a state where there exists a processor which contains a description of both  $q$  and  $v$ . A *search process* for a query  $q$  with search path  $\text{path}(q) = (v_1, \dots, v_h)$  is a sequence of states  $s_1, s_2, \dots$  for the mesh such that, at the conclusion of that sequence, the query  $q$  has visited each node  $v_i$ ,  $1 \leq i \leq h$ .

The *multisearch problem* for a set  $Q = \{q_1, \dots, q_m\}$  of  $m$  queries consists of executing (in parallel)  $m$  search processes, one for each  $q_i$ .

The main result that we need from Atallah et al. [5] is:

**Proposition 1** *Let  $G$  be a hierarchical DAG of size  $n$  and let  $Q$  be a set of  $m = O(n)$  search queries, then the multisearch problem for  $Q$  can be solved in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

### 2.2 Multipoint Location

Let  $S$  be a set of nonintersecting segments (except possibly at endpoints) and let  $P$  be a set of points, all on the plane. Given such an  $S$  and  $P$ , the *multipoint location problem* for  $S$  and  $P$  consists of finding, for each  $p \in P$ , the segments  $u_p, l_p \in S$  that are directly above and below  $p$  (if any), respectively.

The main result that we need from Jeong and Lee [17] is:

**Proposition 2** *Let  $S$  be a set of nonintersecting segments (except possibly at endpoints) and let  $P$  be a set of points, all on the plane, where  $|S| + |P| = O(n)$ , then the multipoint location problem for  $S$  and  $P$  can be solved in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

A useful corollary concerning *trapezoidal decomposition* of simple polygons (which consists of subdividing a polygon into trapezoids by means of vertical rays emanating in both directions from each vertex) is:

**Corollary 1** *Let  $SP$  be a simple polygon with the set  $S$  of edges, where  $|S| = O(n)$ , then the trapezoidal decomposition of  $SP$  can be constructed in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

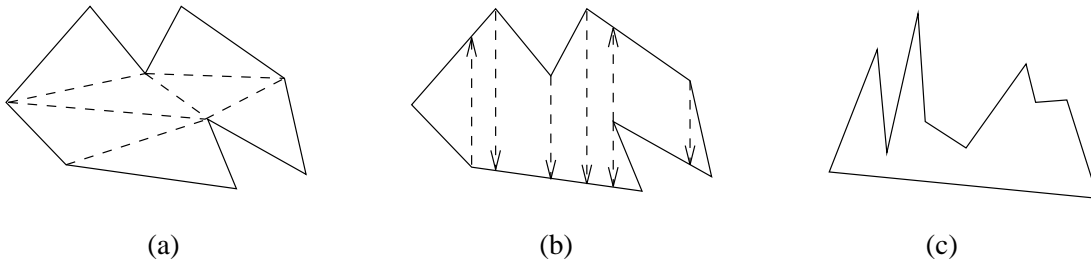


Figure 2: (a) A triangulated polygon, (b) A trapezoidal decomposition, (c) A one-sided monotone polygon.

Another useful corollary concerning *planar point location* in a planar subdivision (which consists of determining, for some given point, which region of the subdivision it is contained in) is:

**Corollary 2** *Let  $PS$  be a planar subdivision with the set  $S$  of edges and let  $P$  be a set of query points on the plane, then the planar point location problem for  $P$  in  $PS$  (i.e., determining, for each  $p \in P$ , which region of  $PS$  it belongs to) can be solved in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

### 2.3 Parallel Binary Search

The *parallel binary search* technique of Miller and Stout [19] is an iterative “search-and-compress” procedure that can be applied to prove, amongst other results, the following:

**Proposition 3** *Let  $C_1$  and  $C_2$  be two convex chains of points on the plane, where  $|C_1| + |C_2| = O(n)$ , then the common tangents between  $C_1$  and  $C_2$  can be found in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

## 3 Triangulation

Jeong and Lee [17, Cor. 2] indicate without elaboration that, using a method of Yap [23], an  $n$ -vertex simple polygon can be *triangulated* optimally on the mesh. Since triangulation is the first phase in all our subsequent algorithms, we record this fact as our first theorem and very briefly discuss the proof.

**Theorem 1 ([17])** *An  $n$ -vertex simple polygon can be triangulated in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

*Proof.* Given an  $n$ -vertex simple polygon  $P$ , we can compute, using a result of Jeong and Lee [17] (see Cor. 1 of Prop. 2), a trapezoidal decomposition of  $P$ , and use this decomposition to split  $P$  into a collection  $\mathcal{Q}$  of *one-sided monotone polygons*, all in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh. See [11, 17] for definitions and details, as also Fig. 2. Next, we intend to triangulate each  $Q \in \mathcal{Q}$  by Yap’s method (see [23, Sec. 3, Step 2’]), after computing its trapezoidal decomposition along the direction parallel to its distinguished edge. However, a technical problem needs to be addressed at this point: if we try to assign each  $Q \in \mathcal{Q}$  a square submesh large enough to hold  $Q$  and solve its individual decomposition problem, we may run out of room on the original mesh.

One way around this problem is to assign each subproblem a rectangular submesh and invoke the simulation results of Atallah [2], which imply that, if problem  $A$  at size  $n$  can be solved in

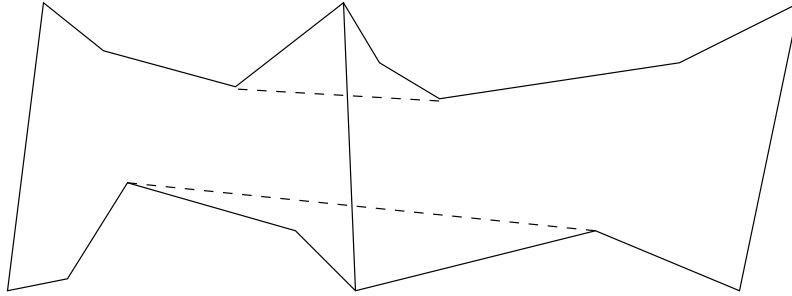


Figure 3: Merging hourglasses.

$O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh  $M$ , then problem  $A$  at size  $m \leq n$  can also be solved in  $O(\sqrt{n})$  time on any rectangular submesh of  $M$  of size  $m$ .

However, an alternate geometric solution that should lead to lower constants is as follows. Rotate each  $Q \in \mathcal{Q}$  so that its distinguished edge is horizontal, and then shift each one horizontally so that no two distinct  $Q$  intersect (this requires a prefix sum computation on the widths of the  $Q$ ). Then, a horizontal trapezoidal decomposition computation over the whole mesh will solve all the individual decomposition problems, provided each edge of a  $Q \in \mathcal{Q}$  checks if its trapezoidal neighbors are, indeed, edges of  $Q$ .  $\square$

## 4 Shortest Paths and Related Structures

### 4.1 Shortest Path

It is well-known [18] that the shortest internal path between two points inside a simple polygon  $P$  is a polygonal line with turns at some vertices of  $P$ . We first prove the following theorem.

**Theorem 2** *Given an  $n$ -vertex simple polygon  $P$  and two point  $p$  and  $q$  in  $P$ , the shortest internal path between them can be computed in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

*Proof.* Our divide-and-conquer proof straightforwardly adapts the ideas of ElGindy and Goodrich [10, Th. 1], who give a PRAM algorithm. We give a quick outline of our adaptation for a  $\sqrt{n} \times \sqrt{n}$  mesh.

Triangulate  $P$  (Th. 1) and construct the dual tree  $D$  of the triangulation  $G$ . Locate  $p$  and  $q$ , in  $O(\sqrt{n})$  time, by the planar point location method of Jeong and Lee [17] (see Cor. 2 of Prop. 2), in triangles of  $G$  corresponding to the nodes, say,  $p'$  and  $q'$  of  $D$ . Determine the path  $S'$  in  $D$  from  $p'$  to  $q'$  (a simple tree computation [6]), and from  $S'$  determine the sequence of *separating diagonals*  $e_1, e_2, \dots, e_m$  that the shortest path from  $p$  to  $q$  must cross in that order. We shall next compute the *hourglass*  $H(e_1, e_m)$  joining  $e_1$  and  $e_m$  (see [15] for a definition and discussion of hourglasses, also Fig. 3).  $H(e_1, e_m)$  can be found recursively in  $O(\sqrt{n})$  time as follows.

If  $m = 2$ , the hourglass is trivially a triangle. If  $m > 2$ , let  $e_k$  be the median separating diagonal between  $e_1$  and  $e_m$ . Recursively compute  $H(e_1, e_k)$  and  $H(e_k, e_m)$ . Merging  $H(e_1, e_k)$  and  $H(e_k, e_m)$  to obtain  $H(e_1, e_m)$  consists essentially of finding the common tangents of two pairs of convex chains (see Fig. 3). This can be done in  $O(\sqrt{n})$  time using the parallel binary search method of Miller and Stout [19], described in Prop. 3. Thus, mesh divide-and-conquer finds  $H(e_1, e_m)$  in  $O(\sqrt{n})$  time.

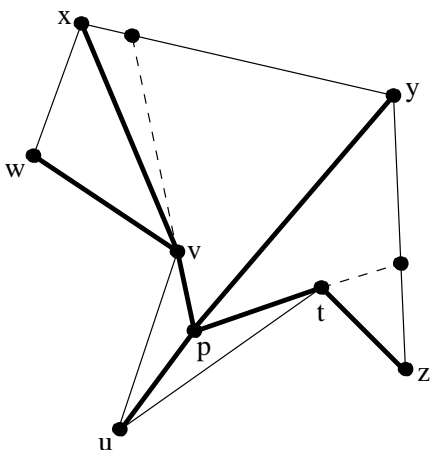


Figure 4: Shortest path tree and partition w.r.t.  $p$ .

Once  $H(e_1, e_m)$  has been found it is not hard to see that the shortest internal path from  $p$  to  $q$  can be determined by computing the tangents from  $p$  and  $q$  to the convex chains of  $H(e_1, e_m)$ , and, possibly, the common tangents between these two chains, each again using parallel binary search.  $\square$

## 4.2 Shortest Path Tree

The union of all shortest paths from a point  $p$  in a simple polygon  $P$  to all vertices of  $P$  is a plane tree, assumed rooted at  $p$ , called the *shortest path tree* from  $p$  (or, simply, the tree from  $p$ ; see Fig. 4, and [15] for a discussion of its properties). However, before discussing the computation of this tree we need some preliminaries. Note that the tree from  $p$  can be represented by  $first(p_i, p)$ , for each vertex  $p_i (\neq p)$  of  $P$ , where  $first(p_i, p)$  is the next vertex after  $p_i$  on the shortest path from  $p_i$  to  $p$ , i.e., the parent of  $p_i$  in the tree from  $p$ . Indeed, we shall compute  $first(p_i, p)$ , for each vertex  $p_i (\neq p)$  of  $P$ .

As another preliminary, we need a method to compute a *centroid* of some given tree  $T$  of bounded node degree. The centroid (see [10]) is defined as a vertex  $v$  which minimizes the maximum cardinality of the subtrees formed by the deletion of  $v$  from  $T$  (each of which will, in fact, be some fraction  $\alpha$  of the size of  $T$ , where  $\alpha$  is a constant,  $0 < \alpha < 1$ ). It suffices to observe that the Euler tour technique of [10] to find a centroid of a tree (on a PRAM) can be implemented on the mesh. In fact, we refer to Atallah and Hambrusch [6, Sec. 3] for details of implementing the Euler tour technique on the mesh, after which we have the following:

**Lemma 1** *We can find a centroid vertex  $v$  of an  $n$ -vertex tree  $T$  of bounded node degree, as well as compute the subtrees resulting from the deletion of  $v$ , in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.  $\square$*

Our theorem is:

**Theorem 3** *Given an  $n$ -vertex simple polygon  $P$  and a point  $p$  in  $P$ , the shortest path tree from  $p$  can be computed in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

*Proof.* Again our divide-and-conquer proof follows ideas for a PRAM algorithm of ElGindy and Goodrich [10, Th. 2]. Triangulate  $P$  and construct the dual tree  $D$  of the triangulation

$G$ . Say  $p$  belongs to some triangle  $s$  of  $G$  corresponding to the node  $s'$  of  $D$ . Delete  $s'$  from  $D$  to leave subtrees  $D_1, \dots, D_c$  ( $c \leq 3$ ). Let  $e_j = (p_{j_1}, p_{j_2})$  be the edge shared by  $s$  and the subpolygon  $P_j$  corresponding to  $D_j$ ,  $1 \leq j \leq c$ . For each  $P_j$  and each vertex  $p_i \in P_j$ , we shall compute  $first(p_i, p_{j_1})$  and  $first(p_i, p_{j_2})$  recursively as follows.

Step 1: If  $D_j$  is singleton return with trivial answers.

Step 2: Use Lemma 1 to find a centroid  $v'_j$  of  $D_j$ . Say  $v'_j$  corresponds to the triangle  $v_j$  with edges  $e_{j,k}$  ( $1 \leq k \leq 3$ ). Split  $D_j$  by deleting  $v'_j$  to leave subtrees  $D_{j,k}$ ,  $1 \leq k \leq 3$ , so that the subpolygon  $P_{j,k}$  corresponding to  $D_{j,k}$  shares edge  $e_{j,k} = (p_{(j,k)_1}, p_{(j,k)_2})$  with  $v_j$ . W.l.o.g. assume that  $P_{j,1}$  shares edge  $e_j$  with  $s$ .

Step 3: Recursively compute  $first(p_i, p_{j_1})$  and  $first(p_i, p_{j_2})$ , where  $p_i \in P_{j,1}$ , and then construct the hourglass  $H(e_{j,1}, e_j)$  following the method of Th. 2.

Step 4: Recursively compute  $first(p_i, p_{(j,k)_1})$  and  $first(p_i, p_{(j,k)_2})$ , where  $p_i \in P_{j,k}$  and  $k = 2, 3$ .

Step 5: For each  $p_i \in P_{j,k}$ ,  $k = 2, 3$ , we can compute  $first(p_i, p_{j_1})$  and  $first(p_i, p_{j_2})$  by determining the intersections, if any, of the rays  $p_i first(p_i, p_{(j,k)_1})$  and  $p_i first(p_i, p_{(j,k)_2})$ , emanating from  $p_i$  and forming the *visibility wedge* of  $p_i$  through  $e_{j,k}$ , with the convex chains of  $H(e_{j,1}, e_j)$ , and then, possibly, computing the tangents to these convex chains from  $p_i$ . This requires a bounded number of binary searches for each  $p_i \in P_{j,k}$ ,  $k = 2, 3$ , provided we represent the convex chains of  $H(e_{j,1}, e_j)$  as balanced search trees.

How do we perform these binary searches for all the  $p_i$  simultaneously, given that there is no concurrently readable global storage in a mesh? One answer to this critical problem is to observe that a balanced search tree is a hierarchical DAG, and invoke the multisearch technique of Atallah et al. [5], described in Prop. 1, to perform these  $O(n)$  binary searches in parallel within  $O(\sqrt{n})$  time.

Thus, we see that that the time to split into subproblems and merge subsolutions in this recursive procedure is  $O(\sqrt{n})$ , so that generalized mesh divide-and-conquer gives  $first(p_i, p_{j_1})$  and  $first(p_i, p_{j_2})$ , for each  $p_i \in P_j$ , in total  $O(\sqrt{n})$  time.

Finally, one can determine  $first(p_i, p)$ , for each  $p_i \in P_j$ , by deciding if  $p$  lies in the visibility wedge formed by the rays  $p_i first(p_i, p_{j_1})$  and  $p_i first(p_i, p_{j_2})$ . Repeating this procedure for each  $P_j$ ,  $1 \leq j \leq c$ , gives  $first(p_i, p)$ , for each vertex  $p_i (\neq p)$  of  $P$ .  $\square$

### 4.3 Shortest Path Partition

A structure related to the shortest path tree that, too, has useful applications is the *shortest path partition*. To describe this structure we need a preliminary definition: given a point  $p$  in a simple polygon  $P$ , the *anchor* w.r.t.  $p$  of a point  $q$  on the boundary of  $P$  is the last vertex on the geodesic path from  $p$  to  $q$ , if there is at least one vertex on that path; or  $p$  itself, otherwise. The shortest path partition of  $P$  w.r.t.  $p$  is then defined to be the partition of its boundary into maximal segments such that all points in a segment share the same anchor w.r.t.  $p$  (see Guha [14], also Fig. 4). The end-points of the segments of the partition (or, simply called, points of the partition w.r.t.  $p$ ) may be obtained as follows:

Let  $T$  be the shortest path tree from  $p$ . For every non-leaf non-root vertex  $v$  of  $T$ , extend the ray  $\overline{par(v)v}$ , where  $par(v)$  is the parent of  $v$  in  $T$ , till it first intersects the boundary of  $P$ . All such intersection points together with the vertices of  $P$  form exactly the points of the partition w.r.t.  $p$ .

To compute the shortest path partition it therefore suffices to decide, for each ray  $\overline{par(v)v}$ , which edge of  $P$  it strikes first. This may be accomplished by an Euler tour method based on a geometric insight that we next describe.

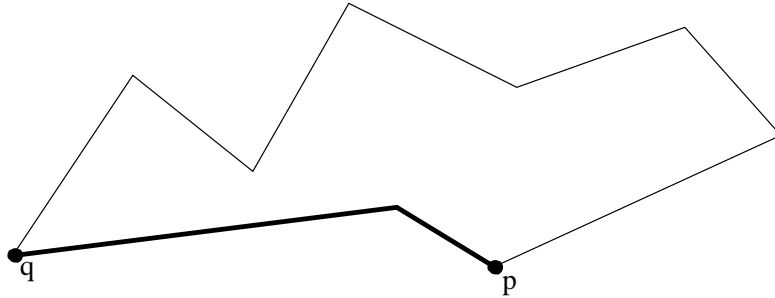


Figure 5: Internal farthest neighbor  $q$  of  $p$  and the shortest path between them.

First, for each vertex  $v$  of  $T$ , order its list of adjacent vertices in a clockwise manner such that, if  $v$  is not the root, then  $par(v)$  is the first in the ordering (the ordering at the root may be arbitrary). For example, in Fig. 4, the ordering of the vertices adjacent to  $v$  would be  $p, w, x$ . Now replace each edge of  $T$  by two oppositely directed edges, and determine an Euler tour  $L$  of the resulting graph, w.r.t. the ordering of adjacency lists just described, that starts and ends at the root  $p$ . Represent  $L$  as the list of vertices in the order in which they are visited (for details about constructing the Euler tour optimally on a mesh and deriving  $L$ , see [6, Sec. 3]). For example, in Fig. 4, an Euler tour is  $pupvwwxvpyptztp$ .

Note that the number of times a vertex  $v$  occurs in  $L$  is one more than the number of children of  $v$ . Denote the first occurrence of  $v$  by  $v_f$  and the last one by  $v_l$  (these may be found, for all  $v$ , with a prefix computation). Consider any non-leaf non-root vertex  $v$  of  $T$ . Then, amongst the edges emerging from  $v$ , edge  $v par(v)$  neighbors both the edges to  $v$ 's first and last child (which may be the same), and makes a reflex angle, say  $A$ , with *exactly one* of them. Two cases arise according as  $A$  is described by rotating  $v par(v)$  in the counter-clockwise or clockwise directions around  $v$ , respectively. In the first (second) case, the ray  $par(v)v$  extends to first intersect the edge of  $P$  that joins the last leaf vertex to occur before  $v_l$  ( $v_f$ ) in  $L$  and the first leaf vertex to occur after  $v_l$  ( $v_f$ ) in  $L$ . These leaf vertices can again be determined with a prefix computation. For example, in Fig. 4, the first case holds for vertex  $v$ , and the leaf nodes occurring just before and after the last occurrence of  $v$  in the Euler tour  $pupvwwxvpyptztp$  are  $x$  and  $y$ , respectively. We have, therefore:

**Theorem 4** *Given an  $n$ -vertex simple polygon  $P$  and a point  $p$  in  $P$ , the shortest path partition of  $P$  w.r.t.  $p$  can be computed in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.  $\square$*

## 5 Farthest Neighbors

In this section we show an application of the geometric structures discussed in the previous section to a proximity problem for a simple polygon  $P$ . The *internal all-farthest neighbors problem* is that of computing, for each vertex  $p$  of  $P$ , a point  $\psi(p)$  of  $P$  at maximum geodesic distance from  $p$ . It can be shown [21] that  $\psi(p)$  is, in fact, another vertex of  $P$ . See Fig. 5 for an illustration.

A serial algorithm for this problem is given, for example, by Suri [21]. Guha [14, Ths. 2] gives a near-optimal PRAM algorithm, and our results use his approach (though Hershberger [16] describes a more efficient PRAM algorithm).

**Theorem 5** *Given an  $n$ -vertex simple polygon  $P$ , the internal all-farthest neighbors problem for  $P$  can be solved in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

*Proof.* The method, as in [14], is to parallelize Suri’s [21] divide-and-conquer algorithm. The fundamental insight is basically the following: given a vertex  $p$  of  $P$  with internal farthest neighbor  $\psi(p)$ , there is an internal farthest neighbor of each vertex of  $P[p, \psi(p)]$  (the piece of the boundary of  $P$  going clockwise from  $p$  to  $\psi(p)$ ) that lies in  $P[\psi(p), p]$ , and, similarly, an internal farthest neighbor of each vertex of  $P[\psi(p), p]$  that lies in  $P[p, \psi(p)]$ .

This insight is the basis for divide-and-conquer. Without repeating details from [21] (in particular, see Algorithm *RFN* in Sec. 3), it is enough to note that a generic “divide” step consists of finding an internal farthest neighbor  $q$  of a vertex  $p$  in an  $m$ -sided subpolygon  $Q$  of  $P$ , and then using  $q$  to split  $Q$  into two subpolygons each of size  $\leq \alpha m$ , where  $\alpha$  is a constant s.t.  $0 < \alpha < 1$ . To find  $q$ , first construct the shortest path tree  $T$  from  $p$  in  $Q$  using Th. 3, in  $O(\sqrt{m})$  time on a submesh of size  $m$ . Then, an  $O(\sqrt{m})$  time tree computation will give a longest root-to-leaf path in  $T$  (see [6], noting that the required computation reduces to evaluating an arithmetic expression tree with edge labels equal to their length, and with node operators *max* and *+*). This determines  $q$ , and the subsequent split of  $Q$  is trivial. The “merge” step is also trivial, as the solution of the problem in  $Q$  is simply the union of the solutions of the problem in either of the two subpolygons that it was split into. Generalized mesh divide-and-conquer, therefore, solves the all-farthest neighbors problem within the stated resource bounds.  $\square$

## 6 Visibility

In this section we consider three fundamental visibility problems for a simple polygon  $P$ , for each of which serial algorithms are well-known (see, e.g., [8, 15]).

### 6.1 Visibility Polygon

First, we deal with the problem of computing the visibility polygon  $Vis(P, p)$ , consisting of all points in  $P$  visible from a given point  $p$  in  $P$ .

**Theorem 6** *Given an  $n$ -vertex simple polygon  $P$  and a point  $p$  in  $P$ , the visibility polygon  $Vis(P, p)$  can be computed in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

*Proof.*  $Vis(P, p)$  is a simple subpolygon of  $P$ , each vertex of which is either a vertex of  $P$  that is visible from  $p$ , or the “shadow” of such a visible vertex cast on the boundary of  $P$  by a “light” at  $p$ . See Fig. 6, as also [15, Sec. 3]. Now, the vertices of  $P$  visible from  $p$  are precisely the children of  $p$  in the shortest path tree from  $p$ , and the shadow cast by such a child  $q$  of  $p$  is the point where the ray  $\overline{pq}$  extends to first hit the boundary of  $P$ , i.e., the point of the shortest path partition of  $P$  w.r.t.  $p$  that corresponds to the edge  $pq$ . Therefore, using Th. 3 to compute the shortest path tree from  $p$ , and Th. 4 to compute the shortest path partition w.r.t.  $p$ , and then performing a straightforward prefix computation to collect the points of  $Vis(P, p)$ , the result follows.  $\square$

We mention that Dehne in [9] shows an optimal mesh algorithm to compute the visibility polygon of a point inside a polygon *with* holes, which generalizes the algorithm of the preceding theorem, but is proved by altogether different methods.

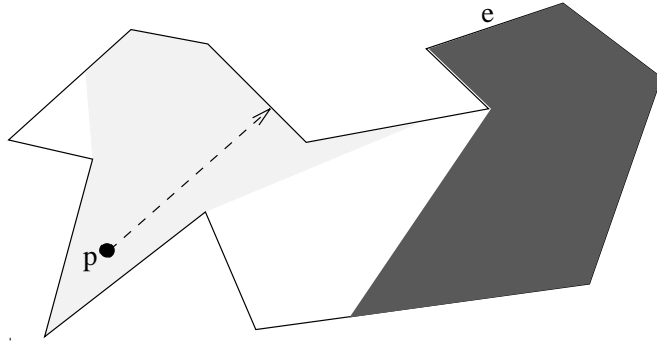


Figure 6: The (lightly shaded) visibility polygon of  $p$ , the (heavily shaded) weak visibility polygon of  $e$ , and a ray from  $p$ .

## 6.2 Geometric Duality

For the next two visibility algorithms we use geometric duality on the two-sided plane. Essentially, we follow the methods of Chazelle and Guibas [8], parallelizing their (serial) algorithms to run on the mesh. Referring the reader to [8] for details, we briefly sketch the application of duality to visibility problems.

For a point  $p$  and ray  $r$ , let  $T_p$  and  $T_r$  denote the dual ray and dual point, respectively. The set of rays intersecting the edge  $e$  and entering into the  $n$ -vertex simple polygon  $P$  dualize to a double wedge  $I_{P,e}$  on the dual plane. Subdivide  $I_{P,e}$  into some  $m(\leq n)$  regions such that two points belong to the same region if and only if the first edges of  $P$  struck by their duals (after, of course, emerging from  $e$ ) are the same. Denote this subdivision by  $S(I_{P,e})$ . Then  $S(I_{P,e})$  has  $O(n)$  edges, and every region of  $S(I_{P,e})$  is convex. We shall show that  $S(I_{P,e})$  can be optimally computed on a mesh.

If  $P$  is a triangle,  $S(I_{P,e})$  can be trivially computed. If  $P$  has more than three edges, then triangulate  $P$  and determine a diagonal  $f$  which separates  $P$  into subpolygons  $P_1$  and  $P_2$ , each of at most some constant fraction of the size of  $P$  (such a separating diagonal can be found using Lemma 1 to compute a centroid of the dual tree of the triangulation). Assume w.l.o.g. that  $e$  is an edge of  $P_1$ . Recursively compute  $S(I_{P_1,e})$  and  $S(I_{P_2,f})$ . Let  $R$  be the region of  $S(I_{P_1,e})$  that is the locus of the duals of rays through  $e$  that first strike  $f$ . Then,

$$S(I_{P,e}) = S(I_{P_1,e}) \cup (R \cap S(I_{P_2,f})).$$

Thus, we have a generalized mesh divide-and-conquer method for computing  $S(I_{P,e})$ , provided we can “clip”  $S(I_{P_2,f})$  to within  $R$  and add this refinement of  $R$  to  $S(I_{P_1,e})$ . To clip  $S(I_{P_2,f})$  within  $R$ , first represent the edges of  $R$  in a balanced search tree (possible as  $R$  is convex). Then, for each edge of  $S(I_{P_2,f})$ , perform a binary search in this tree to determine its intersections, if any, with  $R$ , and clip accordingly. All the searches for each edge of  $S(I_{P_2,f})$  can be done in parallel using the multisearch method of Atallah et al. [5] described in Prop. 1, in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh. We, therefore, have:

**Lemma 2** *Given an  $n$ -vertex simple polygon  $P$  and an edge  $e$  of  $P$ ,  $S(I_{P,e})$  can be computed in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.  $\square$*

The geometric structure  $S(I_{P,e})$  is used in all our subsequent results, following the underlying ideas of Chazelle and Guibas [8].

### 6.3 Weak Visibility Polygon

The next algorithm is to compute the weak visibility polygon  $Vis(P, e)$ , consisting of all points in  $P$  visible from some point of a given edge  $e$  of  $P$  (see Fig. 6).

**Theorem 7** *Given an  $n$ -vertex simple polygon  $P$  and an edge  $e$  of  $P$ , the weak visibility polygon  $Vis(P, e)$  can be computed in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

*Proof.* An edge  $f$  of  $P$  is, possibly partially, visible from  $e$  if the region  $R$  of  $S(I_{P,e})$  associated with  $f$  (i.e.,  $R$  is the locus of the duals of rays emanating from  $e$  that first strike  $f$ ) is non-empty. If  $R$  is, in fact, non-empty we can determine the part of  $f$  visible from  $e$  as follows:

Let  $T_f$  be the dual point of the supporting line of  $f$  (with some orientation). Then the two lines passing through  $T_f$  and tangent to  $R$  are exactly the duals of two points  $a$  and  $b$  on  $f$  which bound the subsegment of  $f$  visible from  $e$ . These two tangents can be computed using binary search, provided the edges bounding the convex polygon  $R$  are represented in a balanced search tree.

Accordingly, on a  $\sqrt{n} \times \sqrt{n}$  mesh, use Lemma 2 to optimally compute  $S(I_{P,e})$ . Then, represent the edges bounding each region of  $S(I_{P,e})$  in a balanced search tree, a distinct search tree for each region. The total size of this data structure is still  $O(n)$ , as each edge of  $S(I_{P,e})$  is represented in exactly two search trees. Now, for each edge  $f$  of  $P$ , the subsegment of  $f$  visible from  $e$  can be determined by a binary search of the tree representing the boundary of the region associated with  $f$ . Invoking the multisearch technique of Atallah et al. [5] for hierarchical DAGs described in Prop. 1, all these searches can be completed in  $O(\sqrt{n})$  time. Finally,  $Vis(P, e)$  can be derived with a prefix computation to collect the end-points of all visible subsegments and join them into a polygon.  $\square$

### 6.4 Ray Shooting

The following result concerns a restricted form of ray shooting.

**Lemma 3** *Given an  $n$ -vertex simple polygon  $P$ , an edge  $e$  of  $P$ , and a ray  $r$  emanating into  $P$  from some point of  $e$ , the first intersection of  $r$  with the boundary of  $P$  can be computed in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

*Proof.* This simply requires locating the region  $R$  of  $S(I_{P,e})$  in which the dual point  $T_r$  lies, as the edge of  $P$  associated with  $R$  is precisely the first one struck by  $r$ .  $S(I_{P,e})$  can be optimally computed on a  $\sqrt{n} \times \sqrt{n}$  mesh by Lemma 2, and the region  $R$  in which  $T_r$  lies can also be optimally determined by the planar point location method of Jeong and Lee [17] (see Cor. 2 of Prop. 2).  $\square$

We consider general ray shooting next (see Fig. 6).

**Theorem 8** *Given a point  $p$  inside an  $n$ -vertex simple polygon  $P$  and a direction  $u$ , the first intersection with the boundary of  $P$  of a ray emanating from  $p$  in the direction  $u$  can be computed in  $O(\sqrt{n})$  time on a  $\sqrt{n} \times \sqrt{n}$  mesh.*

*Proof.* Denote by  $\sigma^P(p, u)$  the first intersection with the boundary of  $P$  of the ray  $r$  emanating from  $p$  in the direction  $u$ . If  $P$  is a triangle, it is trivial to compute  $\sigma^P(p, u)$ . If  $P$  has more than three edges, then triangulate  $P$  and determine (using Lemma 1 to compute a centroid of the dual tree of the triangulation) a diagonal  $f$  that separates  $P$  into two subpolygons  $P_1$  and  $P_2$ , each of at most some constant fraction of the size of  $P$ . Assume w.l.o.g. that  $p$  lies in  $P_1$ . Now, it is not hard to verify the following geometric property [8]:

The point  $\sigma^P(p, u)$  lies in  $P_2$  if and only if the ray from  $p$  in the direction  $u$  intersects (*disregarding* edges of  $P$ )  $f$  at the point  $q$ , and  $p$  lies on the segment  $q\sigma^{P_1}(q, -u)$ . If this holds, then  $\sigma^P(p, u) = \sigma^{P_2}(q, u)$ ; otherwise,  $\sigma^P(p, u) = \sigma^{P_1}(p, u)$ .

Observing that both  $\sigma^{P_1}(q, -u)$  and  $\sigma^{P_2}(q, u)$  can be optimally computed by applying the preceding lemma to the edge  $f$ , it follows that a generalized mesh divide-and-conquer gives the required algorithm.  $\square$

## 7 Conclusions

We have shown many of the fundamental proximity and visibility problems in simple polygons can be solved optimally on a mesh-connected computer. All our algorithms were based on divide-and-conquer recursion. It seems that, with a mesh, we are trading concurrently accessible global storage (as, e.g., in a PRAM) for the more benign recursive recurrence  $T(n) = T(\alpha n) + O(\sqrt{n})$ , which solves to an optimal  $T(n) = O(\sqrt{n})$ . This compares with the recurrence  $T(n) = T(\alpha n) + O(\log n)$ , which is typical for PRAM divide-and-conquer, but solves to  $T(n) = O(\log^2 n)$ , which is usually suboptimal for the PRAM. This makes divide-and-conquer a very attractive design method in mesh algorithms, and especially effective for geometric problems when applied with tools such as the multisearch, multipoint location, and parallel binary search techniques.

There are, of course, several other problems for simple polygons that one can examine and, hopefully, solve optimally on the mesh using methods similar to those here. In particular, a challenging problem that we would like to mention is that of computing the geodesic Voronoi diagram of a set of points inside a simple polygon. Aronov [1] gives a serial algorithm and Guha [13] shows the problem to be in NC, but an efficient algorithm for either the PRAM or mesh seems difficult.

Another direction arises from the fact that, in the absence of a global storage in the mesh, one cannot attempt to build in parallel data structures that would allow a *single* processor to efficiently solve some problem (an approach taken, for example, on the PRAM by Goodrich et al. [12]). Such an approach may be considered, however, on a hybrid model such as the RAM/ARRAY [3].

## References

- [1] B. Aronov, On the geodesic Voronoi diagram of point sites in a simple polygon, *Algorithmica* **4** (1989), 109-140.
- [2] M. J. Atallah, Simulations between mesh-connected processor arrays, *Proc. Annual Allerton Conf. on Communication, Control and Computing*, 1985, 268-269.
- [3] M. J. Atallah, Parallel techniques for computational geometry, *Proc. IEEE* **80** (1992), 1435-1448.
- [4] M. J. Atallah, D. Z. Chen, An optimal parallel algorithm for the visibility of a simple polygon from a point, *Proc. ACM Symp. on Computational Geometry*, 1989, 114-123; *J. ACM* **38** (1991), 516-533.
- [5] M. J. Atallah, F. Dehne, R. Miller, A. Rau-Chaplin, J-J. Tsay, Multisearch techniques for implementing data structures on a mesh-connected computer, *Proc. ACM Symp. on Parallel Algorithms and Architectures*, 1991, 204-214; *J. Parallel and Distributed Computing*, to appear.
- [6] M. J. Atallah, S. E. Hambrusch, Solving tree problems on a mesh-connected processor array, *Information and Control* **69** (1986), 168-187.

- [7] B. Chazelle, Triangulating a simple polygon in linear time, *Proc. IEEE Symp. on Foundations of Computer Science*, 1990, 220-230; *Discrete & Computational Geometry* **6** (1991), 485-524.
- [8] B. Chazelle, L. J. Guibas, Visibility and intersection problems in plane geometry, *Proc. ACM Symp. on Computational Geometry*, 1985, 135-146; *Discrete & Computational Geometry* **4** (1989), 551-581.
- [9] F. Dehne, Solving visibility and separability problems on a mesh-of-processors, *The Visual Computer* **3** (1988), 356-370.
- [10] H. ElGindy, M. T. Goodrich, Parallel algorithms for shortest path problems in polygons, *The Visual Computer* **3** (1988), 371-388.
- [11] M. T. Goodrich, Triangulating a polygon in parallel, *J. Algorithms* **10** (1989), 327-351.
- [12] M. T. Goodrich, S. B. Shauck, S. Guha, Parallel methods for visibility and shortest path problems in simple polygons, *Proc. ACM Symp. on Computational Geometry*, 1990, 73-82; *Algorithmica* **8** (1992), 461-486.
- [13] S. Guha, *Parallel Algorithms for Polygonal and Rectilinear Geometry*, Ph.D. Thesis, Computer Science and Engineering, University of Michigan, Ann Arbor, 1991.
- [14] S. Guha, Parallel computation of internal and external farthest neighbors in simple polygons, *International J. of Computational Geometry and Applications* **2** (1992), 175-190.
- [15] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, R. J. Tarjan, Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons, *Algorithmica* **2** (1987), 209-233.
- [16] J. Hershberger, Optimal parallel algorithms for triangulated simple polygons, *Proc. ACM Symp. on Computational Geometry*, 1992, 33-42.
- [17] C. S. Jeong, D. T. Lee, Parallel geometric algorithms for a mesh-connected computer, *Algorithmica* **5** (1990), 155-177.
- [18] D. T. Lee, F. P. Preparata, Euclidean shortest paths in the presence of rectilinear barriers, *Networks* **14** (1984), 393-410.
- [19] R. Miller, Q. F. Stout, Mesh computer algorithms for computational geometry, *IEEE Trans. Computers* **C-38** (1989), 321-340.
- [20] D. Nassimi, S. Sahni, Data broadcasting in SIMD computers, *IEEE Trans. Computers* **C-30** (1981), 101-106.
- [21] S. Suri, Computing geodesic furthest neighbors in simple polygons, *J. Computer and System Sciences* **39** (1989), 220-235.
- [22] C. D. Thompson, H. T. Kung, Sorting on a mesh-connected parallel computer, *Comm. ACM* **20** (1977), 263-271.
- [23] C. K. Yap, Parallel triangulation of a polygon in two calls to the trapezoidal map, *Algorithmica* **3** (1988), 279-288.