# SOLVING VISIBILTIY AND SEPARABILITY PROBLEMS ON A MESH-OF-PROCESSORS

Frank Dehne*

SCS-TR-123

Nov. 1987

*) Research supported by NSERC grant No. A9173

**ABSTRACT**

In this paper we study parallel algorithms for the Mesh-of-Processors architecture to solve visibility as well as related separability problems for sets of simple polygons in the plane; in particular, we solve the following problems:

( 1 ) compute for a point p inside a N-vertex simple polygon with holes the visibility polygon containing all points visible from p

( 2 ) for the parallel visibility model (visibility from a point with infinite distance) compute the portion of a N-vertex simple polygon visible in a given direction d and the visibility hull with respect to direction d

( 3 ) determine whether two N-vertex simple polygons are separable (without collision) by a single translation in a given direction d

( 4 ) determine for a set of M N-vertex simple polygons whether they are sequentially separable in a given direction d, i.e. whether there exists a sequence of translations in direction d, one for each polygon, which allows to separate them without collisions between polygons.

The parallel algorithms presented in this paper have an asymptotically optimal time complexity of $O(\sqrt{N})$ for problems 1-3 and $O(\sqrt{M\,N})$ for problem 4, respectively, and require a linear number of processing elements.

**Keywords:** Computational Geometry, Mesh-of-Processors, Parallel Algorithms, Separability, Visibility

## 1. INTRODUCTION

The notion of visibility in geometric objects is important for a large number of geometric applications; e.g. the hidden line problem in graphics ([FL67]), the shortest path problem in a plane with polygonal obstructions ([AAG85]), and the separability problem for planar polygonal objects ([T85], [ST85], [DS87], [COSW83]).

The visibility problem for planar simple polygons has been studied for several models of visibility: e.g. visibility from a point ([A85], [GA81], [L83], [LP79]), visibility from an edge ([AT81]), and parallel visibility, i.e. visibility from a point with infinite distance ([T85], [GA81], [L83]).

In this paper we study parallel algorithms for solving visibility as well as related separability problems involving sets of simple polygons in the plane; the considered parallel model of computation is the Mesh-of-Processors (see e.g. [MS84]).

The motivation for studying parallel algorithms on the mesh which solve geometric problems is that (1) in the recent past an increasing number of parallel machines of this type has become available and (2) geometric applications are major candidates for parallel implementations since they often require huge amounts of data to be handled on-line (e.g. for CAD workstations).

In the following section 2 we will briefly describe the Mesh-of-Processors model and state a well known lower bound argument for the running time of parallel algorithms executed on a mesh.

In section 3 we will introduce an asymptotically optimal $O(\sqrt{N})$ time parallel algorithm to compute for a point p inside a N-vertex simple polygon with holes the visibility polygon containing all points visible from p.

In section 4 we will consider parallel visibility and describe an asymptotically optimal $O(\sqrt{N})$ time parallel algorithm to determine the portion of a N-vertex simple polygon visible in a given direction d as well as the visibility hull ([T85]) with respect to direction d.

The above parallel solutions for visibility problems imply new asymptotically optimal mesh algorithms to (1) detect for a given direction d whether two simple polygons are separable (without collision) by a single translation (to infinity) in direction d and (2)

detect for a set of polygons whether they are sequentially separable, i.e. whether there exists a sequence of translations, one for each polygon, which allows to separate them while avoiding collisions between polygons. These algorithms will be presented in section 5.

## 2. MODEL OF COMPUTATION

The model of computation considered in this paper is the *Mesh-of-Processors* of size N, i.e. a set of N synchronized processing elements (PEs) arranged on a $\sqrt{N} \times \sqrt{N}$ grid where each PE is connect to its direct neighbors by bidirectional communication links (see Figure 1).

Each processor has a constant number of registers and within one time unit it can simultaneously send an output to and receive an input from each of its communication links.

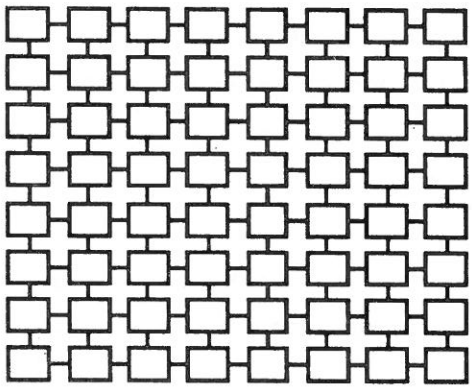


Figure 1: A Mesh-of-Processors

The Mesh-of-Processors is a very realistic model of computation; a steadily increasing number of parallel machines which have an array architecture or can be set up as an array are available; e.g. the Programmable Systolic Chip ([S82]), the Connection Machine (Thinking Machines Inc., [Hi85]), and the Hypercube (Intel).

Note, that on a Mesh-of-Processors a worst-case time complexity of $O(\sqrt{N})$ is optimal for any nontrivial problem, since for comparing the contents of two PEs it is necessary to route the data through the mesh which may take $O(\sqrt{N})$ steps (see e.g. [MS84]).

Several standard techniques (which will be used frequently in the remaining of this paper) have been developed for designing algorithms on a Mesh-of-Processors; e.g. broadcasting information from one PE to all other PEs, rotating data within rows or columns of PEs, sorting, and recursion. Note, that all these operations can be performed in time $O(\sqrt{N})$; see e.g. [TK77], [MS84], [UL84], [D86a].

## 3. VISIBILITY FROM A POINT

Consider a N-vertex simple polygon $\mathbb{P}$ with holes which contains a point p in its interior (a polygon $\mathbb{P}$ is *simple*, if no two of its edges intersect).

A point q is called *visible* from p if the straight line segment from p to q does not intersect any edge of $\mathbb{P}$.

The *visibility polygon* from a point p (with respect to a polygon $\mathbb{P}$) is the polygon containing all those points visible from p (see Figure 2).

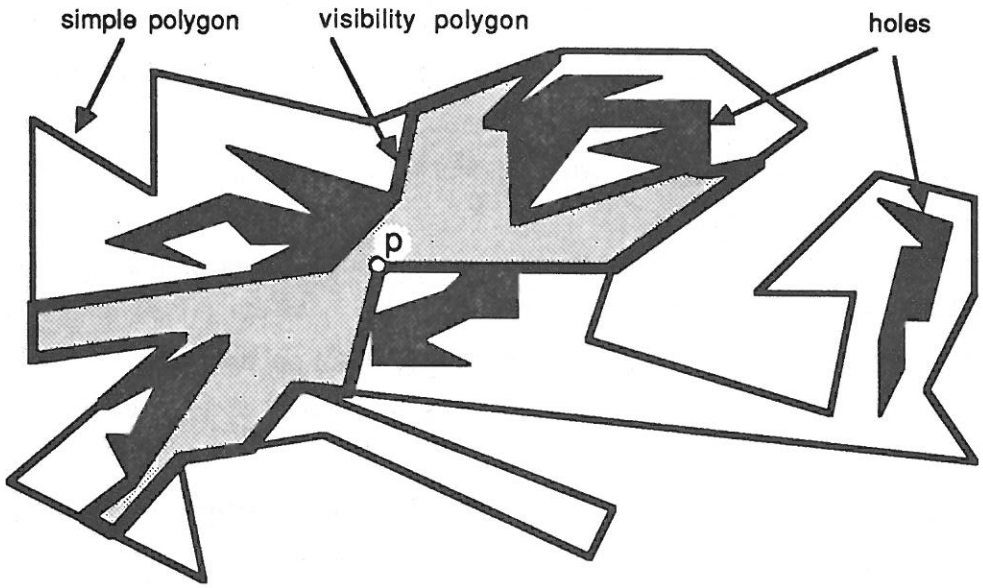



Figure 2: Visibility Polygon from a Point

The problem of computing the visibility polygon from a point has been studied extensively under the standard sequential model of computation. [GA81] and [L83] introduced linear time sequential algorithms for computing the visibility polygon from a point inside a simple polygon without holes. For the general case [A85] described an O(N log h) time sequential algorithm for computing the visibility polygon from a point within a polygon with h holes.

Recently [AU87] presented an O(N) time parallel algorithm for solving the visibility problem for simple polygons with holes on a linear array of N processors, i.e. a linear arrangement of N PEs where each PE is connected to its both neighbors (if exist). Similar to the Mesh-of-Processors model it is easy to observe that their solution is asymptotically optimal for linear processor arrays. Unfortunately, their solution is based on an exhaustive search strategy which apparently can not be generalized to yield an $O(\sqrt{N})$ time solution on a Mesh-of-Processors ([U87]). However, from a VLSI point of view, the hardware complexity (area) of a linear array and a mesh differ only in a constant factor. Hence, finding an optimal $O(\sqrt{N})$ time solution for a Mesh-of-Processors yields a significant increase in efficiency.

In the following we will present an optimal $O(\sqrt{N})$ time solution for computing the visibility polygon from a point in the interior of a N-vertex simple polygon with holes on a Mesh-of-Processors of size N.

The algorithm assumes the following initial configuration:

- All edges of the polygon $\mathbb{P}$ are directed such that the interior of the polygon (exterior of the holes) is to the left of each edge (see Figure 3). For an edge e=(x,y), directed from x to y, we will refer to vertex x as its *start vertex* and vertex y as its *end vertex*.
- Each PE of the Mesh-of-Processors stores the coordinates of the vertices of an (arbitrary) edge of polygon $\mathbb{P}$. (A sorted sequence of the edges of the exterior boundary of polygon and its holes, respectively, is not necessary.)
- Each PE of the Mesh-of-Processors stores the coordinates of the point p.

The result, i.e. the visibility polygon from p, is reported as follows:

- A list of all *visible intervals* of the polygon $\mathbb{P}$ is reported, where a visible interval is defined as a maximal interval of an edge of $\mathbb{P}$ that is entirely visible from the point p.
- Each of these visible intervals is stored in one (arbitrary) PE. Note, that the number of visible intervals is at most O(N).

From this information, a list of the vertices of the visibility polygon in sorted order can be easily obtained in time $O(\sqrt{N})$ by sorting the vertices of the visible intervals by the angle of their polar coordinates with respect to center p (which will be referred to as *polar p-coordinates*).

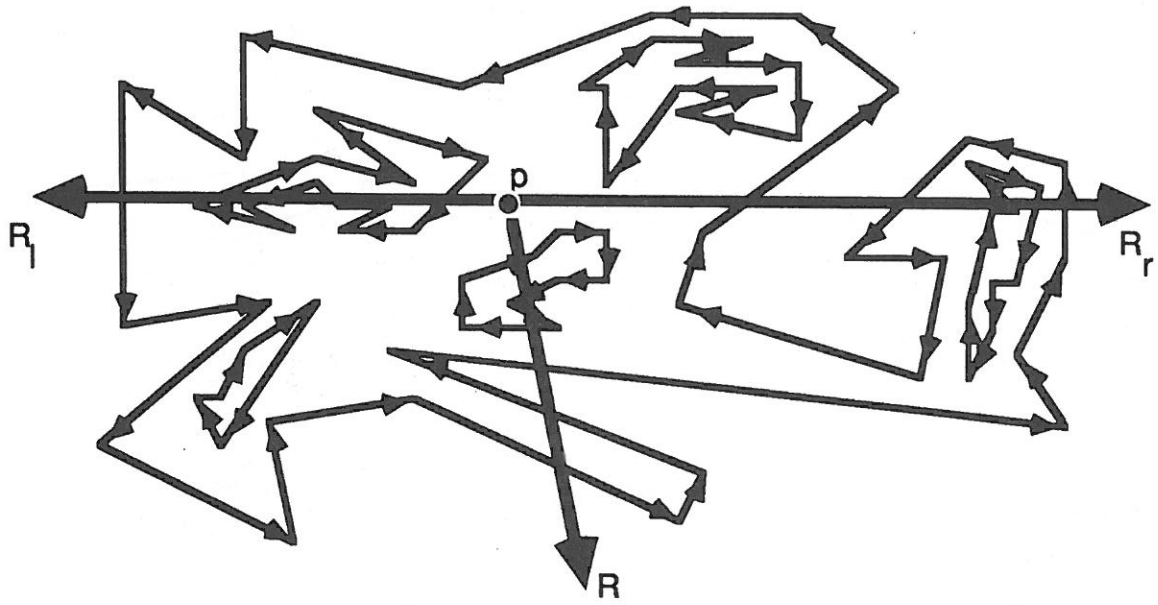
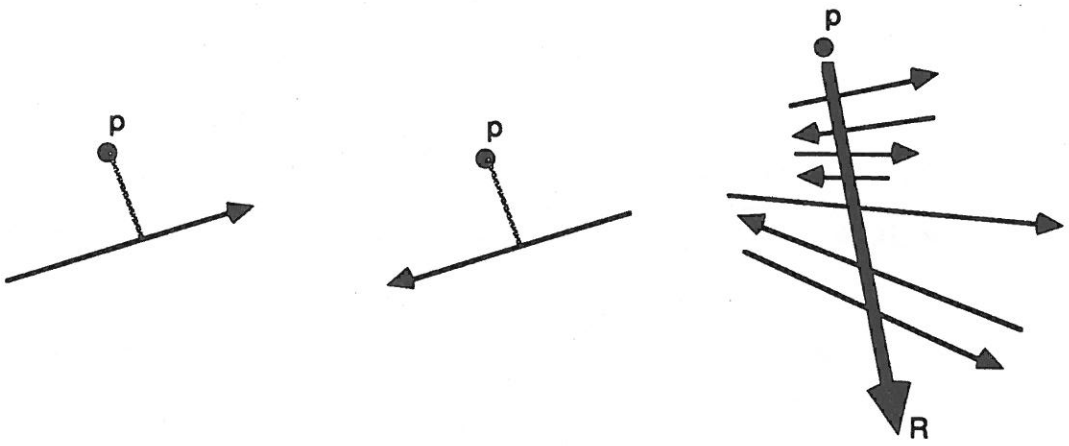



Figure 3: A Polygon with Directed Edges; Basic Divide-and-Conquer Mechanism

Consider the two horizontal rays $R_l$ and $R_r$ emanating from p to the left and right, respectively (see Figure 3). In the following we will show how to compute the visible intervals below these rays. The visible intervals above the rays $R_l$ and $R_r$ can be computed in a second analogous step.

Before we proceed to describing the algorithm we need some basic definitions and observations.

Consider an edge e of the polygon $\mathbb{P}$. If the point p is on the left side of e, then e is called a *luff edge*, otherwise it is called a *lee edge* (see Figure 4a and 4b). Obviously, a lee edge can not contain a visible interval.




(a)
a Luff Edge

(b)
a Lee Edge

(c)
Luff and Lee Edges
Intersected by R

Figure 4: Luff and Lee Edges

Consider the edges intersected by a ray R emanating from p (e.g. the ray shown in Figure 3). It is easy to observe that these edges always form an alternating sequence of luff and lee edges (beginning and ending with a luff edge); see Figure 4c.

During the execution of the algorithm, edges which are still under consideration for containing a visible interval will be referred to as *active edges*.

From those edges which are not under consideration as active edges any more, some (which are lee edges) will be stored as *passive edges*. Although they can not contain a visible interval, they may cause other edges to be invisible from p and, hence, be necessary in subsequent stages of the algorithm. (Passive edges will depicted as shaded edges in the subsequent figures.) Vertices of active and passive edges will be referred to as active and passive vertices, respectively; for the remaining, the joint vertex of two adjacent edges will be considered as two vertices, one for each edge.

To compute all visible intervals below $R_l$ and $R_r$ all edges below these rays which do not intersect $R_l$ or $R_r$ are initially marked as active edges; from those edges e intersecting $R_l$ or $R_r$ the part of e below the rays is stored as an active edge if e is a luff edge and as a passive edge if e is a lee edge, respectively.

Let $n \leq 2N$ be the total number of active vertices stored in the mesh.

The algorithm described below is recursive and will further subdivide the sector between $R_l$ and $R_r$ by rays emanating downwards from p (e.g. the ray R shown in figure 3); the edges of the polygon P will then be stored as follows:

A submesh of the Mesh-of-Processors is assigned to each subsector. For active edges e, the portion of e in each subsector intersected by the edge will be explicitly stored in the respective submesh. Passive edges, however, will only be stored in the submeshes of the (at most) two sectors which contain at least one of their vertices. A passive edge e will be called *entering passive edge* in a sector S if S contains the end vertex of e and *leaving passive edge* in sector S if S contains the start vertex of e.

The algorithm recursively computes all visible intervals in the sector below $R_l$ and $R_r$ as follows:

1) Split

1a) A ray R emanating downwards from p is selected such that the set of all vertices of the polygon $\mathbb{P}$ below $R_l$ and $R_r$ is split into two subsets of equal size. (This can be implemented by sorting all vertices by the angle of their polar p-coordinates)

1b) Let $Sect_l$ and $Sect_r$ be the left and right subsector created by R, respectively (see Figure 3).
The Mesh-of-Processors is split into two submeshes $M_l$ and $M_r$ of equal size. (Either a horizontal or vertical split line is selected to minimize the diameter of the submeshes.) All active edges entirely contained in $Sect_l$ or $Sect_r$ are stored in $M_l$ or $M_r$, respectively.

1c) Consider the sequence of all active edges intersecting R. The number k of all active luff edges intersecting R which are not followed by a subsequent active lee edge ( which intersects R) is computed. (Initially, k =1)

1d) Each active luff edge intersecting R is split into two active luff edges, one for each subsector; each of them is stored in the respective submesh.

1e) Each active lee edge intersecting R is deleted as an active edge and inserted as a passive edge.

1 f) All entering and leaving passive edges with respect to the sector between $R_l$ and $R_r$ which intersect R (not those computed in step 1e) are updated:
For $Sect_r$ [$Sect_l$] all entering passive edges e which intersect R (see Figure 5a) are considered; each e is stored as an entering [leaving] passive edge of $Sect_l$ [$Sect_r$] and the edge $e_r$ [$e_l$] with its intersection point with R closest to p is computed; all active edges in $Sect_r$ [$Sect_l$] on the left side of $e_r$ [$e_l$] are deleted; see figure 5b.
All entering and leaving passive edges with respect to the sector between $R_l$ and $R_r$ which do not intersect R are stored as entering and leaving passive edges of $Sect_r$ and $Sect_l$, respectively.

1 g) IF k>1 THEN
(i) Unnecessary edges are removed
(ii) IF $k > \frac{2}{5}n$ THEN
The mesh is "rebalanced" and the split phase restarted at step 1b
(The details of step 1g will be given below).

2) Recur

For each submesh (in parallel):
Recursively, all visible intervals in the respective subsector are computed.

3) Merge: (The visible intervals in $Sect_l \cup Sect_r$ are computed)

3a) All pairs of visible intervals in $Sect_l$ and $Sect_r$, respectively, that are part of the same edge of the polygon $\mathbb{P}$ and have a common point q are appended to form one interval, each:
For each such pair of intervals, q is the common point of each of these intervals with the ray R. Hence, all intervals that have a common point with R are sorted by the distance of that common point from p; visible intervals that have to be appended are subsequent in this ordering.

3b) All other visible intervals determined in $Sect_l$ and $Sect_r$ are also visible intervals in $Sect_l \cup Sect_r$.

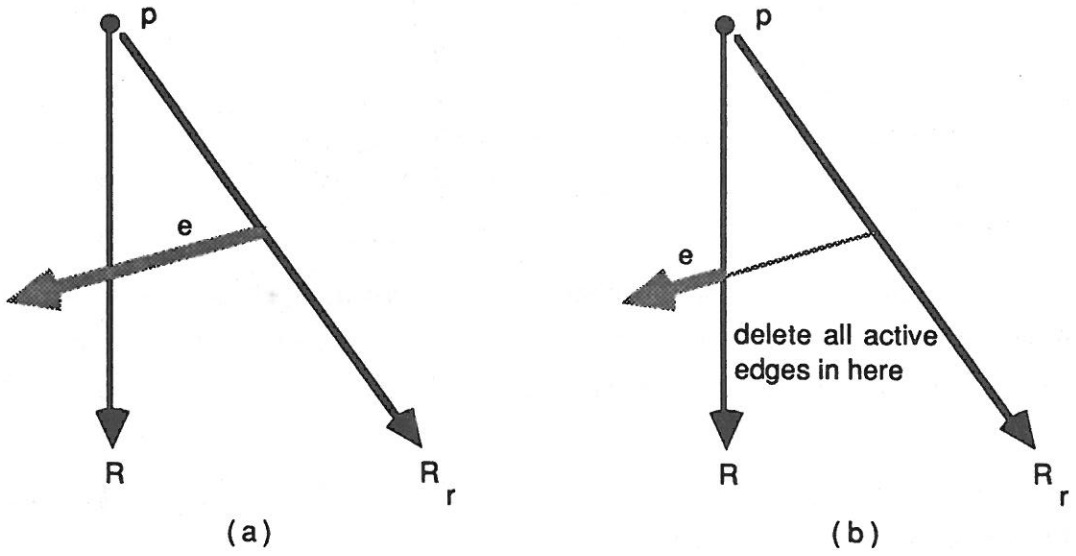



Figure 5: Update of Entering Passive Edges

When the problem is split by a ray R emanating from p into two subproblems of equal size, it is easy to see that both subproblems can be solved independently in parallel since the edges in one subsector have no impact on the computation of the visible intervals in the other subsector. However, before finally reporting the visible intervals, those pairs of intervals computed with respect to a subsector, each, that are part of the same edge of P and have a common point q have to be appended to form one interval (step 3a).

The major problem in such a recursive solution arises from the fact that all active edges intersecting R must be split into two edges, on for each sector. The number of active vertices in the sector between $R_l$ and $R_r$ is n, i.e. the sector contains $\frac{n}{2}$ active edges. If each edge intersects R, then the number of edges in $Sect_l$ and $Sect_r$ may again be $\frac{n}{2}$, each; i.e. the total number of active vertices and edges doubles. If this happens repeatedly then the recursion never terminates since the problem size does not decrease; even if it decreases, then the total number of edges to be stored in the mesh may grow exponentially with respect to the number of recursions.

The split phase of the algorithm described above is mostly concerned with solving this problem. In particular, the algorithm has the following property (*):

Let $n_l$ and $n_r$ be the number of all <u>active</u> vertices stored in $M_l$ and $M_r$ after the problem has been split into two subproblems, then

(i) $n_l + n_r \leq n + C$ , C constant

(ii) $n_l \leq \alpha\, n$ , $n_r \leq \alpha\, n$ , $\alpha < 1$ .

Property (*) ensures that the total number of active vertices increases by at most a constant in each stage of the recursive algorithm and that the sizes of the subproblems (with respect to the number of active vertices) are decreasing.

The same property is obvious for the <u>passive</u> vertices since passive edges are not duplicated (except for the initialization).

In the following, we will specify step 1g of the algorithm and show that, indeed, property (*) holds.

Consider the set of all edges intersection R in the split phase of the algorithm. As we have already observed, these edges form an alternating sequence of luff and lee edges (see figure 6).

<u>Case 1:</u>

Assume that all these edges are active (i.e. k=1 in step 1c); then, for each (except the last) active luff edge which is duplicated in step 1d (adding one active vertex to $Sect_l$ and $Sect_r$, each) there exist a subsequent active lee edge which is deleted in step 1e (subtracting one active vertex from $Sect_l$ and $Sect_r$, each); hence, the total number of active vertices is increased by two and $n_l = n_r$, thus property (*) holds.

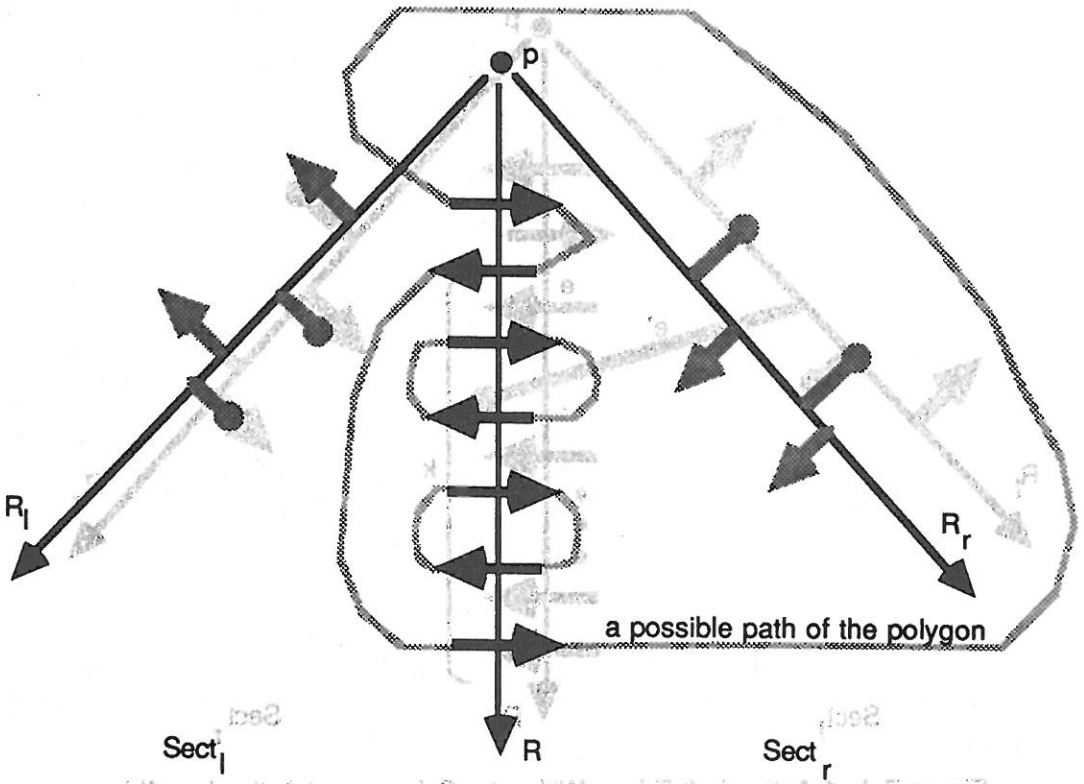



Figure 6: Alternating Active Luff and Lee Edges

Case 2:

Assume that there are k>1 active luff edges that do not have a subsequent active lee edge; these edges will be referred to a *single edges.* Consider the single edge e whose intersection with R is closest to p (see Figure 7).

The subsequent lee edge intersecting R is a passive edge e' which has been deleted as an active edge in a previous step. It is easy to observe that e' must be a passive entering or leaving edge stored in this sector since at least one of its vertices must be located between $R_l$ and $R_r$; otherwise, all edges below e', in particular all k-1 single edges below e, would have been deleted in a previous step (see step 1f).

Hence, e' can be determined by a broadcast to all passive edges in the submesh.

Assume that e' is a passive leaving edge as depicted in Figure 7; if e' is a passive entering edge, exchange "left" and "right" for the following.

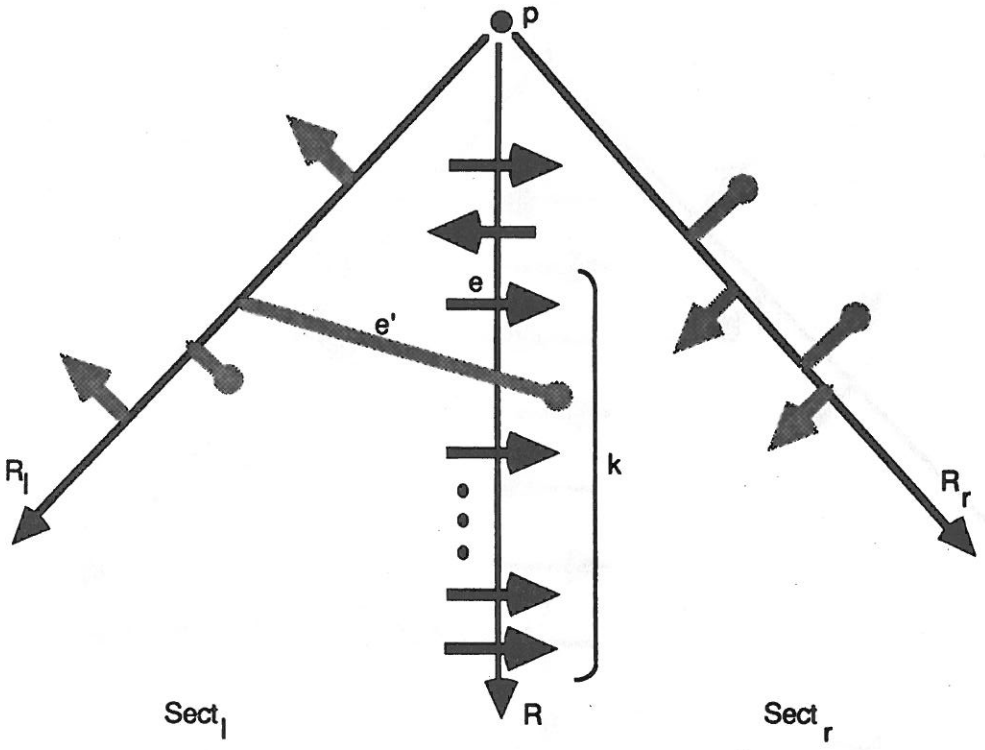



Figure 7: k>1 Active Luff Edges Without a Subsequent Active Lee Edge

For all single edges, except e, it is obvious that after they have been duplicated in step 1d, all those edges to the left of R below e' can be deleted since they are not visible from p. This specifies part (i) of step 1g in the algorithm.

Therefor, the total number of vertices increases by two; thus, part (i) of property (*) holds. To ensure that part (ii) of property (*) holds, too, one has to consider two more cases.

Case 2a: Assume that $k \leq \frac{2}{5} n$ , then

$n_r \leq \frac{1}{2} n + \frac{2}{5} n = \frac{9}{10} n$ and $n_l \leq \frac{1}{2} n - \frac{2}{5} n + 1 \leq \frac{1}{2} n$ (for $n \geq 3$),

hence, part (ii) of property (*) holds, too.

Case 2b: Assume that $k > \frac{2}{5} n$ .

The problem encountered in this case is that, although the total number of active vertices increases by two, only, $Sect_r$ may contain n active vertices.

We will refer to all active luff edges created in step 1d which lie in $Sect_r$ as *entering active edges* (in $Sect_r$). Part (ii) of step 1g in the above algorithm is then specified as follows (see Figure 8):

α) In $Sect_r$, the visibility problem is solved with respect to all entering active edges, only, as follows :

- All entering active edges are sorted in clockwise order with respect to the angle of the polar p-coordinates of their end points (*polar ordering*).
- All entering active edges are sorted in increasing order with respect to the distance of their start points (all start points lie on R) from p (*R ordering*).
- For each entering active edge e, from all those entering edges that have a lower rank in R ordering, the edge e' with maximum rank in polar ordering is computed.
  This can be implemented on a Mesh-of-Processors (for all entering active edges in parallel) in time $O(\sqrt{n})$ by one global row and column rotation; see [D86b] pp. 305-306 .
- Each entering active edge e is replaced by the portion of e which is not obstructed by the respective e'.

β) For the resulting set of (active and passive) vertices and edges in the sector between $R_l$ and $R_r$, the ray R' with median angle with respect to the polar p-coordinates of the vertices is computed and the split phase restarted at step 1b with the new ray R' instead of R.

Note, that in the second pass of the split phase the case 2b, i.e. $k > \frac{2}{5} n$, can not occur again, since after the above processing R' lies between R and $R_r$ and no more than one of the entering active edges processed above can intersect R'; hence,
$k \leq \frac{n}{2} - \frac{2}{5} n = \frac{1}{10} n \leq \frac{2}{5} n$ .

Summarizing, with the above specification of step 1g, property (*) holds in any case after at most two executions of the split phase.

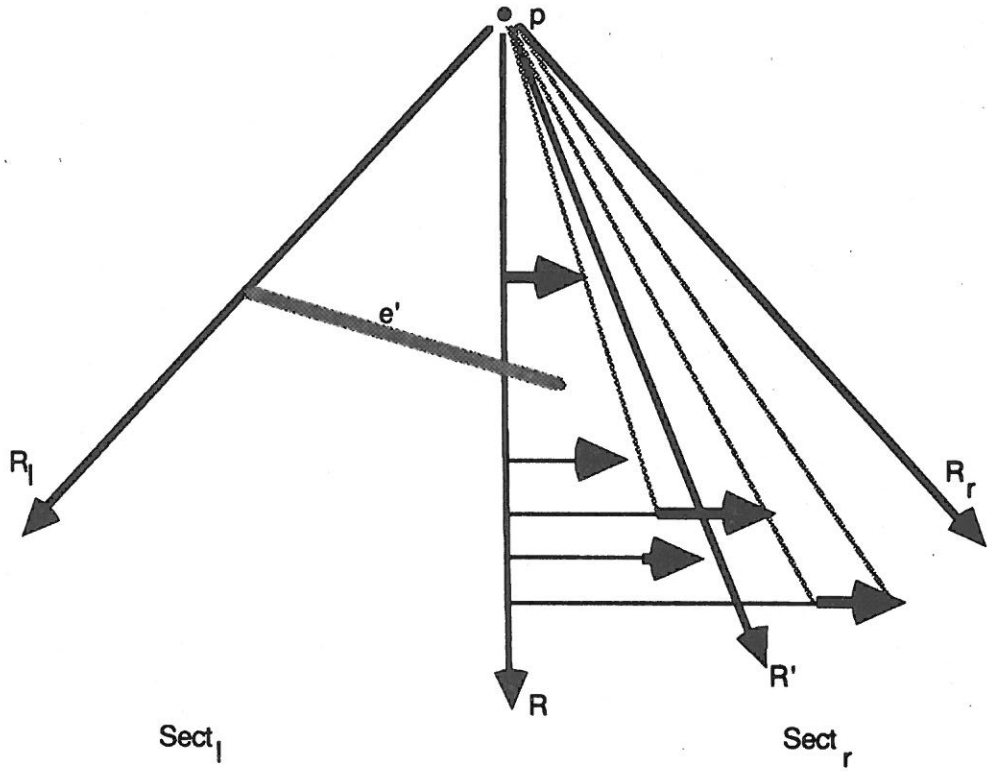



Figure 8: Solving the Visibility Problem for All Entering Active Edges in $Sect_r$

This yields the following

**Theorem 1:**

The visibility polygon from a point inside a simple N-vertex polygon with holes can be computed on a Mesh-of-Processors of size N in time $O(\sqrt{N})$ which is asymptotically optimal.

**Proof:**

It is easy to observe that the split and merge phase of the algorithm can be implemented with a constant number of standard operations (sorting, broadcast, row and column rotations); hence, the time complexity of the split and merge phase is $O(\sqrt{N})$.

Thus, it follows from part (ii) of property (*) that the algorithm has an optimal time complexity $T(N)=O(\sqrt{N})$ since the following recurrence relation holds:

$$T(2) = O(1)$$
$$T(N) \leq T(\alpha N) + O(\sqrt{N}),\ \alpha < 1.$$

Furthermore, it follows from part (i) of property (*) that the total space requirement is O(N); i.e. the total number of vertices grows by at most a constant factor. ♦

## 4. PARALLEL VISIBILITY, THE VISIBILITY HULL

In contrast to the notion of visibility from a point inside a polygon one can also consider the problem of parallel visibility ([T85], [GA81], [L83], [AU87], [LP86]):

Given a simple polygon $\mathbb{P}$ and a direction d then a point q on the boundary of $\mathbb{P}$ is *visible in direction d* if the ray r starting at q in direction -d does not intersect any edge of P (where direction -d is exactly opposite to direction d); see Figure 9.

A *visible interval* is defined as a maximal interval of an edge of $\mathbb{P}$ that is entirely visible in direction d. Obviously, all visible intervals form a monotone sequence with respect to the direction d' perpendicular to d; the *visibility chain* of $\mathbb{P}$ with respect to direction d is the polygonal chain defined by this sequence of all visible intervals.

The *visibility hull* of $\mathbb{P}$ with respect to direction d is the polygon defined by the two visibility chains of $\mathbb{P}$ with respect to d and -d, respectively. (The *visibility hull* of $\mathbb{P}$ can also be defined as the set obtained by taking the union of P with all line segments [a, b] parallel to d where a and b are contained in $\mathbb{P}$.)

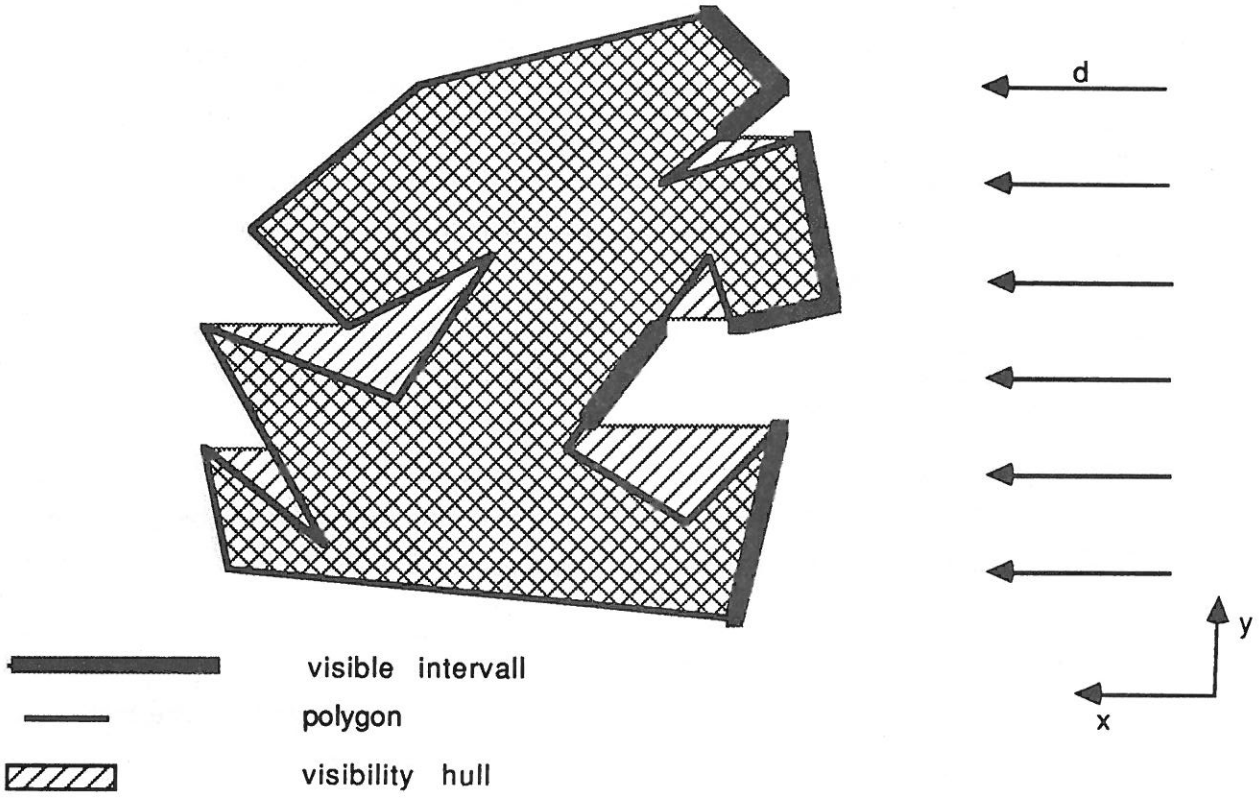



Figure 9: Parallel Visibility, Visibility Hull

Obviously, the parallel visibility problem can be seen as a special case of the visibility problem from a point p located at infinity in direction -d.

In fact, the algorithm in section 3 for computing the visible intervals with respect to visibility from a point p can be easily modified to compute the visible intervals with respect to parallel visibility in direction d by (1) introducing perpendicular x- and y-coordinate axes where the x-axis is parallel to d and all vertices of $\mathbb{P}$ have positiv x-coordinate and (2) replacing for every vertex with coordinates (x,y), the angle in polar p-coordinates and distance from p in the algorithm in section 3 by y and x, respectively.

**Corollary 2:**

The visible intervals, visibility chain, and visibility hull of a N-vertex simple polygon $\mathbb{P}$ with respect to parallel visibility in a given direction d can can be computed on a Mesh-of-Processors of size N in time $O(\sqrt{N})$ which is asymptotically optimal.

## 5. SEPARABILITY OF SIMPLE POLYGONS

A major field of application of visibility hulls is the problem of detecting separability of polygons ([COSW83], [DS87], [ST85], [T85]).

Consider two N-vertex simple polygons $\mathbb{P}$ and $\mathbb{Q}$. $\mathbb{P}$ is called separable from $\mathbb{Q}$ in a given direction d if $\mathbb{P}$ can be translated an arbitrary distance in direction d without colliding with $\mathbb{Q}$; see Figure 10.

The following lemma shows that the algorithm for computing visibility hulls can be utilized to detect whether $\mathbb{P}$ is separable from $\mathbb{Q}$ in a given direction d.

**Lemma 3 [T85]:**

$\mathbb{P}$ is separable from $\mathbb{Q}$ in a given direction d if and only if the visibility hulls of $\mathbb{P}$ and $\mathbb{Q}$ with respect to direction d do not intersect.

Hence, the separability of two N-vertex polygons in a given direction can be detected on a Mesh-of-Processors of size 2N (where each polygon is stored on one half of the mesh) by computing for each polygon its visibility hull and, then, determining whether both visibility hulls intersect. Since intersection of two N-vertex simple polygons can be detected in time $O(\sqrt{N})$ as described in [MS87], we get the following corollary.

**Corollary 4:**

On a Mesh-of-Processors of size 2N it can be decided in asymptotically optimal time $O(\sqrt{N})$ whether a N-vertex simple polygon is separable from another N-vertex simple polygon in a given direction d.

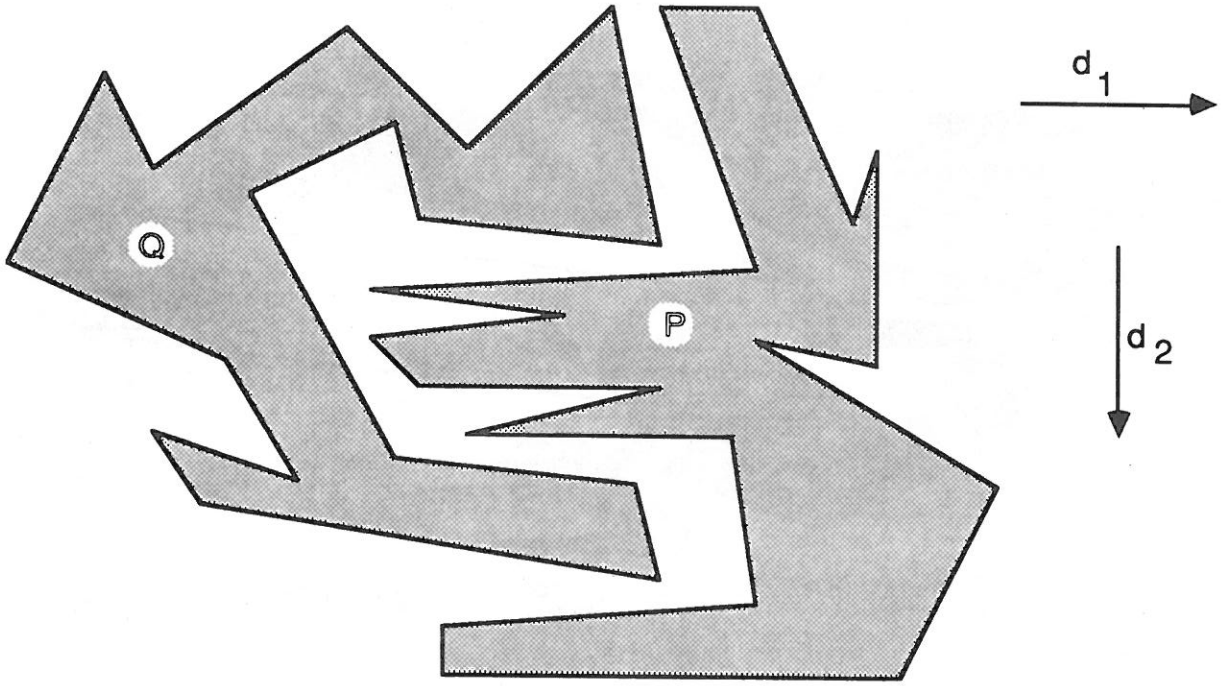



Figure 10: Two Simple Polygons $\mathbb{P}$ and Q Where $\mathbb{P}$ is Separable From Q in Direction $d_1$ But Not in Direction $d_2$

The concept of separability can be further generalized to sets of simple polygons. A set $\{\mathbb{P}_1,...,\mathbb{P}_M\}$ of M N-vertex simple polygons is called sequentially separable in a given direction d if there exists an ordering $\mathbb{P}_{i_1},..., \mathbb{P}_{i_M}$ of these polygons such that every $\mathbb{P}_{i_j}$ is separable (in direction d) from $\mathbb{P}_{i_1},...,\mathbb{P}_{i_{j-1}}$ , each; see figure 11 (cf. e.g. [T85], [DS87]).

[T85] proved that a set of simple polygons is sequentially separable in a direction d if and only if for each pair of polygons their visibility hulls with respect to direction d do not intersect. Furthermore, [MS87] showed that for M N-vertex polygons it can be decided on a Mesh-of-Processors of size M N in time $O(\sqrt{M\,N})$ whether any of these polygons intersect. This yields

**Corollary 5:**

On a Mesh-of-Processors of size M N it can be decided in asymptotically optimal time $O(\sqrt{M N})$ whether M N-vertex simple polygons are sequentially separable in a given direction d.

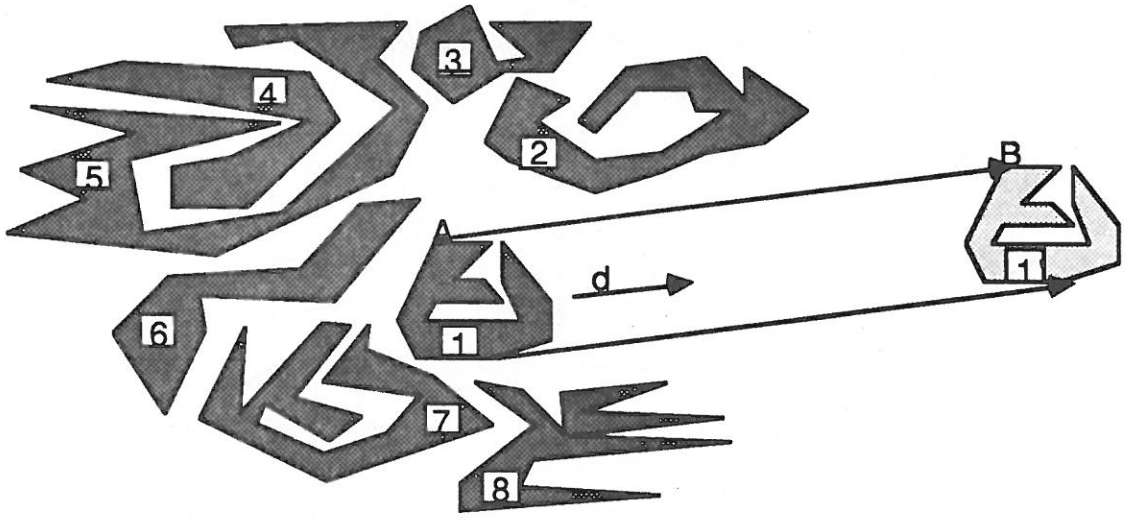



Figure 11 ([DS87]): $P_1$ is Separable from $P_2$ , ... , $P_8$ , Each

## REFERENCES


[A85] T. Asano, "An efficient algorithm for finding the visibility polygon for a polygonal region with holes", Trans. of IECE of Japan, Vol. E-68, No.9, 1985, pp.557-559

[AAG85] T. Asano, T. Asano, L. Guibas, J. Hersberger, H. Imai, "Visibility polygon search and Euclidean shortest paths", Proc. of the IEEE Symp. on FOCS, 1985, pp. 154-164

[AT81] D. Avis, G.T. Toussaint, "An optimal algorithms for detecting the visibility of a polygon from an edge", *IEEE Trans. on Computers*, Vol.. C-30, No.12, Dec. 1981, pp. 910-914

[AU87] T. Asano, H.Umeo, "Systolic algorithms for computing the visibility polygon and triangulation of a polygonal region", Proc. of the Int. Workshop on Parallel Algorithms and Architectures, Suhl, GDR, May 25-30, 1987, pp.77-85

[COSW83] B. Chazelle, T. Ottmann, E. Soisalon-Soinen, and D. Wood, "The complexity and decidability of Separation$^{TM}$ ", Tech. Rept. CS-83-34, Data Structuring Group, University of Waterloo, 1983.

[D86a] F.Dehne, "Parallel Computational Geometry and clustering methods", Tech. Rep. SCS-TR-104, School of Computer Science, Carleton University, Ottawa, Canada K1S5B6,1986

[D86b] F.Dehne, "$O(n^{1/2})$ Algorithms for the maximal elements and ECDF searching problem on a mesh-connected parallel computer", *Information Processing Letters* 22, 1986, pp. 303-306

[DS87] F.Dehne, J.-R. Sack, "Translation separability of sets of polygons", to appear in *The Visual Computer*, Vol.3, No.4, 1987

[FL67] H. Freeman, P.P. Loutrel, "An algorithm for the two-dimensional "hidden line" problem, *IEEE Trans. Electron. Comput.*, Vol. EC-16, No.6, 1967, pp. 784-790

[GA81] H.El Gindy, D.Avis, "A linear algorithm for conputing the visibility polygon from a point", *Journal of Algorithms*, Vol.2, 1981, pp.186-197

[Hi85] W.D.Hillis, "The Connection Machine", The MIT Press, 1985

[L83] D.T.Lee, "Visibility of a simple polygon", *Computer Vision, Graphics, and Image Processing*, Vol.22, 1983, pp.207-221

[LP79] D.T.Lee, F.P. Preparata, "An optimal algorithm for finding the kernel of a polygon, *Journal of the ACM*, Vol. 26, No. 3, 1979, pp. 415-421

[LP86] E.Lodi, L. Pagli, "A VLSI solution to the vertical segment visibility problem", IEEE Trans. on Computers, Vol. C-35, No.10, 1986, pp. 923-928

[MS84] R. Miller, Q.F. Stout, "Computational Geometry on a mesh-connected computer", Proc. IEEE Int. Conf. on Parallel Processing, 1984, pp. 66-73

[MS87] R.Miller, Q.F.Stout, "Mesh computer algorithms for line segments and simple polygons", Proc. IEEE Int. Conf. on Parallel Processing, 1984, pp. 282-285

[S82] L. Snyder, "Introduction to the Configurable Highly Parallel Computer", *Computer* 15(1), 1982, pp. 47-56

[ST85] J.-R. Sack, G.T. Toussaint, "Translating polygons in the plane", Proc. STACS '85, Saarbrücken, Federal Republic of Germany, 1985, pp. 310-321.

[T85] G.T. Toussaint, "Movable separability of sets", in *Computational Geometry*, Ed. G.T. Toussaint, North Holland, 1985, pp.335-376.

[TK77] C.D.Thompson, H.T.Kung, "Sorting on a mesh-connected parallel computer", *Comm. of the ACM*, Vol.20, No.4, April 1977

[U87] H.Umeo, private communications

[UL84] J.D.Ullman, "Computational aspects of VLSI", Principles of Computer Science Series, Computer Science Press, 1984

Carleton University, School of Computer Science
Bibliography of Technical Reports
**Publications List (1985 -->)**

**School of Computer Science**
**Carleton University**
**Ottawa, Ontario, Canada**
**KIS 5B6**

SCS-TR-66 **On the Futility of Arbitrarily Increasing Memory Capabilities of Stochastic Learning Automata**
_____ B.J. Oommen, October 1984. Revised May 1985.

SCS-TR-67 **Heaps in Heaps**
_____ T. Strothotte, J.-R. Sack, November 1984. Revised April 1985.

SCS-TR-68 **Partial Orders and Comparison Problems**
out-of-print M.D. Atkinson, November 1984. See Congressus Numerantium 47 ('86), 77-88

SCS-TR-69 **On the Expected Communication Complexity of Distributed Selection**
_____ N. Santoro, J.B. Sidney, S.J. Sidney, February 1985.

SCS-TR-70 **Features of Fifth Generation Languages: A Panoramic View**
_____ Wilf R. LaLonde, John R. Pugh, March 1985.

SCS-TR-71 **Actra: The Design of an Industrial Fifth Generation Smalltalk System**
_____ David A. Thomas, Wilf R. LaLonde, April 1985.

SCS-TR-72 **Minmaxheaps, Orderstatisticstrees and their Application to the Coursemarks Problem**
_____ M.D. Atkinson, J.-R. Sack, N. Santoro, T. Strothotte, March 1985.

SCS-TR-73 **Designing Communities of Data Types**
Wilf R. LaLonde, May 1985.
Replaced by SCS-TR-108

SCS-TR-74 **Absorbing and Ergodic Discretized Two Action Learning Automata**
out-of-print B. John Oommen, May 1985. See IEEE Trans. on Systems, Man and Cybernetics, March/April 1986, pp. 282-293.

SCS-TR-75 **Optimal Parallel Merging Without Memory Conflicts**
_____ Selim Akl and Nicola Santoro, May 1985

SCS-TR-76 **List Organizing Strategies Using Stochastic Move-to-Front and Stochastic Move-to-Rear Operations**
_____ B. John Oommen, May 1985.

SCS-TR-77 **Linearizing the Directory Growth in Order Preserving Extendible Hashing**
_____ E.J. Otoo, July 1985.

SCS-TR-78 **Improving Semijoin Evaluation in Distributed Query Processing**
_____ E.J. Otoo, N. Santoro, D. Rotem, July 1985.

Carleton University, School of Computer Science
Bibliography of Technical Reports

SCS-TR-79 — **On the Problem of Translating an Elliptic Object Through a Workspace of Elliptic Obstacles**
B.J. Oommen, I. Reichstein, July 1985.

SCS-TR-80 — **Smalltalk - Discovering the System**
W. LaLonde, J. Pugh, D. Thomas, October 1985.

SCS-TR-81 — **A Learning Automation Solution to the Stochastic Minimum Spanning Circle Problem**
B.J. Oommen, October 1985.

SCS-TR-82 — **Separability of Sets of Polygons**
Frank Dehne, Jörg-R. Sack, October 1985.

SCS-TR-83 out-of-print — **Extensions of Partial Orders of Bounded Width**
M.D. Atkinson and H.W. Chang, November 1985. See Congressus Numerantium, Vol. 52 (May 1986), pp. 21-35.

SCS-TR-84 — **Deterministic Learning Automata Solutions to the Object Partitioning Problem**
B. John Oommen, D.C.Y. Ma, November 1985

SCS-TR-85 out-of-print — **Selecting Subsets of the Correct Density**
M.D. Atkinson, December 1985. To appear in Congressus Numerantium, Proceedings of the 1986 South-Eastern conference on Graph theory, combinatorics and Computing.

SCS-TR-86 — **Robot Navigation in Unknown Terrains Using Learned Visiblity Graphs. Part I: The Disjoint Convex Obstacles Case**
B. J. Oommen, S.S. Iyengar, S.V.N. Rao, R.L. Kashyap, February 1986

SCS-TR-87 — **Breaking Symmetry in Synchronous Networks**
Greg N. Frederickson, Nicola Santoro, April 1986

SCS-TR-88 — **Data Structures and Data Types: An Object-Oriented Approach**
John R. Pugh, Wilf R. LaLonde and David A. Thomas, April 1986

SCS-TR-89 — **Ergodic Learning Automata Capable of Incorporating Apriori Information**
B. J. Oommen, May 1986

SCS-TR-90 — **Iterative Decomposition of Digital Systems and its Applications**
Vaclav Dvorak, May 1986.

SCS-TR-91 — **Actors in a Smalltalk Multiprocessor: A Case for Limited Parallelism**
Wilf R. LaLonde, Dave A. Thomas and John R. Pugh, May 1986

SCS-TR-92 — **ACTRA - A Multitasking/Multiprocessing Smalltalk**
David A. Thomas, Wilf R. LaLonde, and John R. Pugh, May 1986

SCS-TR-93 — **Why Exemplars are Better Than Classes**
Wilf R. LaLonde, May 1986

SCS-TR-94 — **An Exemplar Based Smalltalk**
Wilf R. LaLonde, Dave A. Thomas and John R. Pugh, May 1986

SCS-TR-95 — **Recognition of Noisy Subsequences Using Constrained Edit Distances**
B. John Oommen, June 1986

SCS-TR-96 — **Guessing Games and Distributed Computations in Synchronous Networks**

_____ J. van Leeuwen, N. Santoro, J. Urrutia and S. Zaks, June 1986.

SCS-TR-97 **Bit vs. Time Tradeoffs for Distributed Elections in Synchronous Rings**
_____ M. Overmars and N. Santoro, June 1986.

SCS-TR-98 **Reduction Techniques for Distributed Selection**
_____ N. Santoro and E. Suen, June 1986.

SCS-TR-99 **A Note on Lower Bounds for Min-Max Heaps**
_____ A. Hasham and J.-R. Sack, June 1986.

SCS-TR-100 **Sums of Lexicographically Ordered Sets**
_____ M.D. Atkinson, A. Negro, and N. Santoro, May 1987.

SCS-TR-102 **Computing on a Systolic Screen: Hulls, Contours, and Applications**
_____ F. Dehne, J.-R. Sack and N. Santoro, October 1986.

SCS-TR-103 **Stochastic Automata Solutions to the Object Partitioning Problem**
_____ B.J. Oommen and D.C.Y. Ma, November 1986.

SCS-TR-104 **Parallel Computational Geometry and Clustering Methods**
_____ F. Dehne, December 1986.

SCS-TR-105 **On Adding *Constraint Accumulation* to Prolog**
_____ Wilf R. LaLonde, January 1987.

SCS-TR-107 **On the Problem of Multiple Mobile Robots Cluttering a Workspace**
_____ B. J. Oommen and I. Reichstein, January 1987.

SCS-TR-108 **Designing Families of Data Types Using Exemplars**
_____ Wilf R. LaLonde, February 1987.

SCS-TR-109 **From Rings to Complete Graphs - $\Theta$(n logn) to $\Theta$(n) Distributed Leader Election**
_____ Hagit Attiya, Nicola Santoro and Shmuel Zaks, March 1987.

SCS-TR-110 **A Transputer Based Adaptable Pipeline**
_____ Anirban Basu, March 1987.

SCS-TR-111 **Impact of Prediction Accuracy on the Performance of a Pipeline Computer**
_____ Anirban Basu, March 1987.

SCS-TR-112 **$\varepsilon$-Optimal Discretized Linear Reward-Penalty Learning Automata**
_____ B.J. Oommen and J.P.R. Christensen, May 1987.

SCS-TR-113 **Angle Orders, Regular n-gon Orders and the Crossing Number of a Partial Order**
_____ N. Santoro and J. Urrutia, June 1987.

SCS-TR-115 **Time Is Not a Healer: Impossibility of Distributed Agreement in Synchronous Systems with Random Omissions**
_____ N. Santoro, June 1987.

SCS-TR-116 **A Practical Algorithm for Boolean Matrix Multiplication**
_____ M.D. Atkinson and N. Santoro, June 1987.

Carleton University, School of Computer Science
Bibliography of Technical Reports

SCS-TR-117 **Recognizing Polygons, or How to Spy**
James A. Dean, Andrzej Lingas and Jörg-R. Sack, August 1987.

SCS-TR-118 **Stochastic Rendezvous Network Performance - Fast, First-Order Approximations**
J.E. Neilson, C.M. Woodside, J.W. Miernik, D.C. Petriu, August 1987.

SCS-TR-120 **Searching on Alphanumeric Keys Using Local Balanced Trie Hashing**
E.J. Otoo, August 1987.

SCS-TR-121 **An $O(\sqrt{n})$ Algorithm for the ECDF Searching Problem for Arbitrary Dimensions on a Mesh-of-Processors**
Frank Dehne and Ivan Stojmenovic, October 1987.

SCS-TR-122 **An Optimal Algorithm for Computing the Voronoi Diagram on a Cone**
Frank Dehne and Rolf Klein, November 1987.

SCS-TR-123 **Solving Visibility and Separability Problems on a Mesh-of-Processors**
Frank Dehne, November 1987.