The Visual Computer

Solving visibility and separability problems on a Mesh-of-Processors

Frank Dehne*

School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6

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

- An $O(\sqrt{N}$ *time algorithm for computing on a Mesh-of-Processors of size* N the visibility polygon from a point located in an N-vertex polygon, possibly with holes.
- $O(\sqrt{N})$ time algorithms for computing on a Mesh-of-Processors of size N the set of all points on the boundary of an N-vertex polygon $\mathbb{P}$ which are visible in a given direction d as well as the visibility hull of $\mathbb{P}$ for a given direction d.
- An $O(\sqrt{N})$ time algorithm for detecting on a Mesh-of-Processors of size $2N$ whether two N-vertex polygons are separable in a given direction and an $O(\sqrt{MN})$ time algorithm for detecting on a Mesh-of-Processors of size MN whether M N-vertex polygons are sequentially separable in a given direction.

All proposed algorithms are asymptotically optimal (for the Mesh-of-Processors) with respect to time and number of processors.

Key words: Computational geometry – Mesh-of-Processors – Parallel algorithms – Separability-Visibility

* Research supported by NSERC grant No. A9173

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 [11], the shortest path problem for points in a plane with polygonal obstructions [1], and the separability problem for planar polygonal objects [19, 18, 10, 7]. Due to their importance, visibility and related problems have been thoroughly studied in the standard sequential model of computation and several efficient (in some cases asymptotically optimal) algorithms have been presented.
In recent years several authors have started studying geometric problems for parallel architectures; see e.g. [4, 5, 6, 8, 9, 14, 15, 16]. There are mainly two reasons why parallel algorithms for geometric problems have become of special importance:

- A steadily increasing number of parallel machines has become commercially available.
- Geometric algorithms are mainly used for online applications (e.g. CAD workstations) where short response times are a necessity. However, these geometric applications often require large amounts of data to be processed which makes it hard to obtain reasonable response times on standard sequential computers.

In this paper, we study methods for efficiently solving visibility and related separability problems on a Mesh-of-Processors, i.e., a parallel computer which consists of a set of processing elements arranged on a square grid where each *PE* is connected to its direct neighbors only ([UL 84] Chapt. 4.1, 4.2, and 4.4).
In the remainder of this section we will

- introduce the geometric problems studied in this paper,
- describe the Mesh-of-Processors architecture,
- describe how to use divide-and-conquer for solving a problem on a Mesh-of-Processors, and
- give an overview of the results presented in the subsequent sections.

1.1 Geometric problems

Consider an N-vertex *simple polygon* $\mathbb{P}$ in the Euclidean plane; a polygon $\mathbb{P}$ is *simple* if there is no pair of nonconsecutive edges which share a point (see, e.g., [17] p. 18). For the remainder of this paper all polygons considered are simple and planar. A polygon $\mathbb{P}$ partitions the plane into three regions: the interior, the boundary, and the exterior of $\mathbb{P}$. A point p is *contained in* $\mathbb{P}$ if p is contained either in the interior or the boundary of $\mathbb{P}$.

The Visual Computer (1988) 3:356 370

A polygon $\mathbb{P}$ may have *holes*:
It contains other (pairwise disjoint) polygons $\mathbb{P}_1, \ldots, \mathbb{P}_h$ (holes) in its interior; i.e., all points of the interior or boundary of $\mathbb{P}_1, \ldots, \mathbb{P}_h$ are contained in the interior of $\mathbb{P}$.
The interiors and boundaries of the holes are subtracted from the interior of $\mathbb{P}$ and added to the exterior and boundary of $\mathbb{P}$, respectively.

For the remainder, when referring to an N-vertex polygon with holes, N denotes the total number of vertices of $\mathbb{P}, \mathbb{P}_1, \ldots, \mathbb{P}_h$.

Given a polygon $\mathbb{P}$ (with or without holes) and a point p contained in $\mathbb{P}$, then another point q in the Euclidean plane is called *visible* from p if the *open line segment* from p to q, i.e., the open interval (q, q) of the line through p and q, does not share a point with any edge of $\mathbb{P}$.

Obviously, any point q which is not contained in $\mathbb{P}$ is not visible from p.

The *visibility polygon* from a point p contained in a polygon $\mathbb{P}$ (with or without holes) is the polygon containing all those points visible from p.

See Fig. 1 for an illustration of the above definitions.

The problem of computing the visibility polygon from a point has been studied extensively under the standard sequential model of computation. [12] and [13] introduced linear time sequential algorithms for computing the visibility polygon from a point inside an N-vertex polygon without holes; [2] described an $O(N \log h)$ time sequential algorithm for computing the visibility polygon from a point contained in an N-vertex polygon with h holes.

In contrast to the notion of visibility from a point contained in a polygon, one can also consider the model of *parallel visibility* from outside the polygon:

Given a 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 starting at q in direction $-d$, i.e. the direction exactly opposite to direction d, does not share a point, except q, with any edge of $\mathbb{P}$.

The *visibility hull* of $\mathbb{P}$ for a direction d is the polygon $\mathbb{P}'$ defined as follows: the union of boundary and interior of $\mathbb{P}'$ is the set of all points contained either in $\mathbb{P}$ or any line segment $[a, b]$ parallel to d where a and b are contained in $\mathbb{P}$.

See Fig. 2 for an illustration of these definitions.

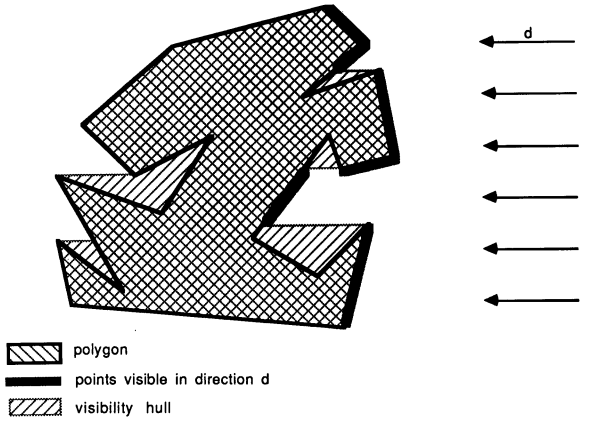

Fig. 2. Parallel visibility, visibility hull

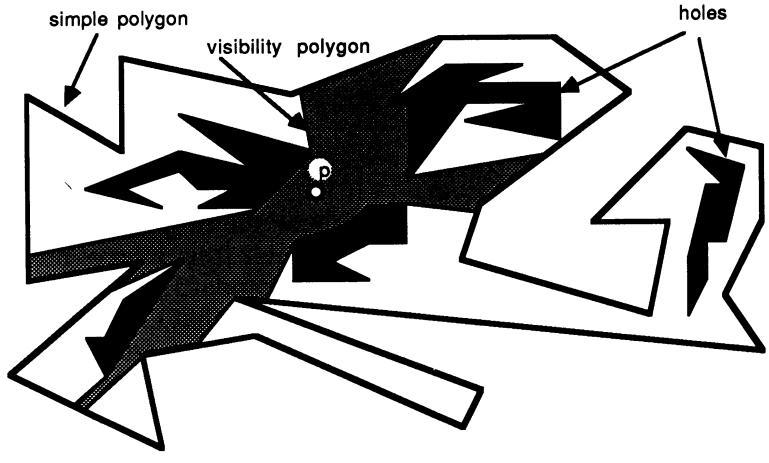

Fig. 1. Visibility polygon from a point

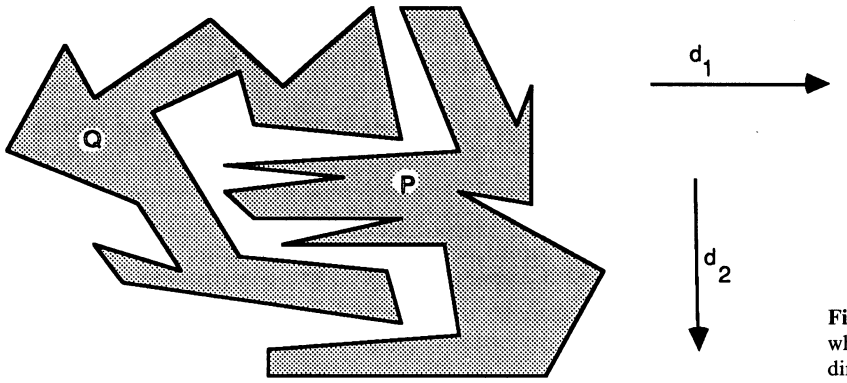

Fig. 3. Two polygons $\mathbb{P}$ and $\mathbb{Q}$ where $\mathbb{P}$ is separable from $\mathbb{Q}$ in direction d_1 but not in direction d_2

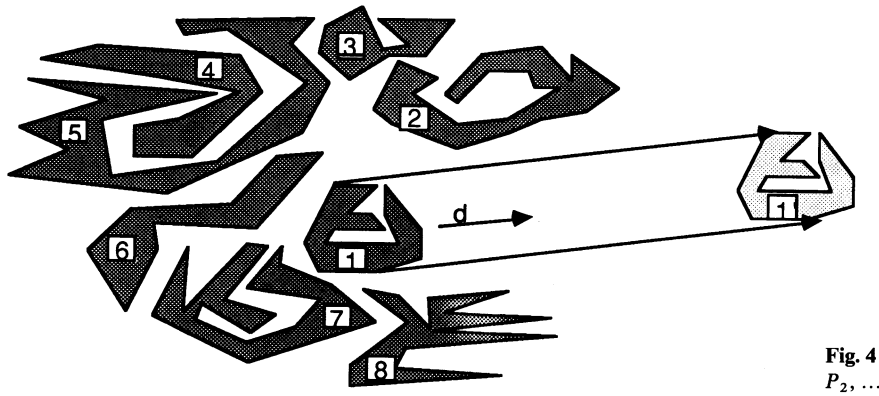

Fig. 4 [10]. P_1 is separable from $P_2, \ldots, P_8$

Note, that in the standard sequential model of computation the points visible in direction d, as well as the visibility hull of an N-vertex polygon can be computed in time $O(N)$; see [12, 13, 19].

An application of visibility algorithms arises when detecting separability of polygons [7, 10, 18, 19]. Consider two nonintersecting N-vertex polygons $\mathbb{P}$ and $\mathbb{Q}$. $\mathbb{P}$ is called (translation) *separable* from $\mathbb{Q}$ in a given direction d if $\mathbb{P}$ can be translated by an arbitrary distance in direction d without colliding with $\mathbb{Q}$; see Fig. 3.

The concept of separability can be further generalized to *sets* of polygons. A set of M non intersecting polygons is called sequentially separable in a given direction d if the set of polygons can be moved to infinity by a sequence of M translations in direction d, one for each polygon, such that no collision occurs.

Formally, this is defined as follows: A set $\{\mathbb{P}_1, \ldots, \mathbb{P}_M\}$ of M nonintersecting N-vertex polygons is *sequentially separable* in a given direction d if there exists an ordering $\mathbb{P}_{i_1}, \ldots, \mathbb{P}_{i_M}$ of these polygons such that every $\mathbb{P}_{i_j}$ is separable (in direction d) from $\mathbb{P}_{i_{j+1}}, \ldots, \mathbb{P}_{i_M}$ (the set of polygons not yet separated). For more details see e.g. [19] and [10].

Figure 4 shows an example of a polygon which is separable from a set of polygons.

For the standard sequential model of computation [19] showed that the problem of detecting whether two N-vertex polygons are separable and whether M N-vertex polygons are sequentially separable in a given direction d can be solved in time $O(N)$ and $O(MN \log(MN))$, respectively.

Summarizing, the geometric problems we will study in this paper are the following:

- computation of the visibility polygon from a point contained in a polygon with holes,
- computation of the points on the boundary of a polygon which are visible in a given direction d,
- computation of the visibility hull of a polygon for a given direction d,
- detection of separability of two polygons in a given direction d, and
- detection of sequential separability of a set of polygons.

1.2 Mesh-of-Processors architecture

The parallel architecture considered in this paper for solving the above problems 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 connected to its direct neighbors by bi-directional communication links (see Fig. 5).

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

A more detailed description of the Mesh-of-Processors can be found, e.g., in [UL 84] (Chapts. 4.1, 4.2, and 4.4).

Several standard techniques which will be used frequently in the remainder of this paper have been developed for designing algorithms on a Mesh-of-Processors (see e.g. [20, 15, 21, 8];

- sending data from one PE to another (non-adjacent) PE,

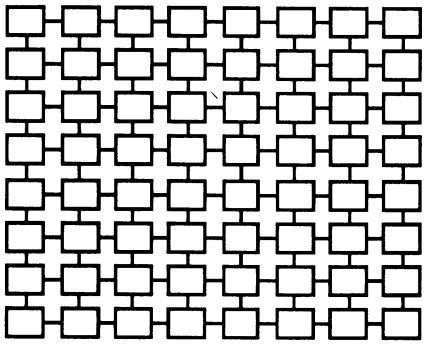

Fig. 5. A Mesh-of-Processors

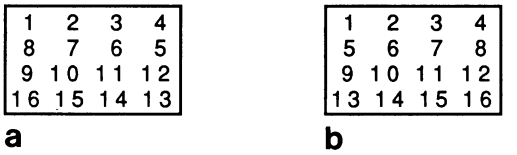

Fig. 6. a Snake-line and **b** row-major ordering of a 4×4 Mesh-of-Processors

- broadcasting information from one PE to all other PEs,
- rotating data within rows or columns of PEs,
- sorting with respect to a register; i.e., permutating the data stored in the PEs such that they are sorted with respect to the given register in e.g. snake-like or row-major ordering (see Fig. 6).

All these operations can be performed in time $O(\sqrt{N})$.

Note, that on a Mesh-of-Processors a worst-case time complexity of $O(\sqrt{N})$ is optimal for solving any nontrivial problem since for comparing the contents of two PEs it is necessary to route the data through the mesh which may take $\Omega(\sqrt{N})$ steps (see [15]).

1.3 Divide-and-conquer on a Mesh-of-Processor

Divide-and-conquer is a very important and frequently used method for the design of efficient mesh algorithms. The basic idea is the same as for the standard sequential machine model: a problem is divided into two subproblems of equal size, each subproblem is solved separately, and finally the solutions of the subproblems are used to compute the solution of the entire problem. For a Mesh-of-Processors it is different in that, when the problem is divided into two subproblems, the mesh is split into two submeshes and the subproblems are solved in parallel, one in each submesh.

(1) *Split*:
   (a) The problem is divided into two subproblems of equal size.
   (b) The mesh is split into two submeshes of equal size (by either a horizontal or vertical split line to minimize the diameter of the submeshes).

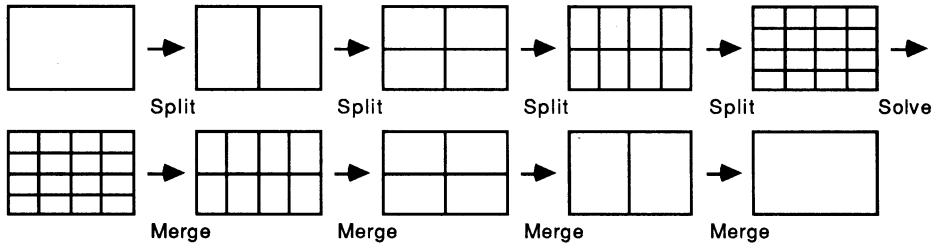

Fig. 7. Trace of the submeshes involved in a divide-and-conquer algorithm

(c) The data for each subproblem is moved into a submesh; i.e., the registers of the PEs of a submesh.

(2) **IF** the size of each submesh is smaller than a given constant C
THEN
Solve:
All submeshes (in parallel) solve their subproblem.
ELSE
Recur:
All submeshes (in parallel) recursively solve the subproblems for the data stored in the submeshes.

(3) *Merge*:
The solutions of the subproblems computed by the submeshes are used to compute the solution of the entire problem.

A description of how to implement a recursive algorithm on a Mesh-of-Processors can be found in [21], Chapt. 4.4.
Note, that the total overhead for implementing a divide and conquer algorithm on a Mesh-of-Processors of size N is $O(\sqrt{N})$ time and linear space (one additional register for each PE).
Thus, given that the split and merge phase can be computed in time $O(\sqrt{N})$, we get the following recurrence for the time $T(N)$ of the entire algorithm:

$$T(C) = O(1)$$

$$T(N) = T\left(\frac{N}{2}\right) + O(\sqrt{N});$$

therefore, $T(N) = O(\sqrt{N})$.
It is easy to observe that the same result follows if, instead of splitting a problem into two subproblems of equal size, a problem of size N is split into two subproblems of size αN and $(1-\alpha)N$, respectively, for any $0 < \alpha < 1$.

1.4 Overview of results

The remainder of this paper is organized as follows:

- In Sect. 2 we will present an $O(\sqrt{N})$ time algorithm for computing, on a Mesh-of-Processors of size N, the visibility polygon from a point contained in an N-vertex polygon with holes.
- In Sect. 3.1 we will describe how the above algorithm can be modified to yield an $O(\sqrt{N})$ time algorithm for computing, on a Mesh-of-Processors of size N, the set of all points on the boundary of an N-vertex polygon which are visible in a given direction d as well as the visibility hull for direction d.
- The above algorithms will be utilized in Sect. 3.2 for detecting separability of polygons in a given direction d. We will obtain an $O(\sqrt{N})$ time algorithm for detecting on a Mesh-of-Processors of size $2N$ whether two N-vertex polygons are separable, and an $O(\sqrt{N})$ time algorithm for detecting on a Mesh-of-Processors of size MN whether M N-vertex polygons are sequentially separable.

All proposed algorithms are asymptotically optimal with respect to time and number of processors. Recently [5] presented an $O(N)$ time parallel algorithms for solving both of the above visibility problems on a linear processor array of size N; i.e., a linear arrangement of N PEs where each PE is connected to its both neighbors (if they exist). Their solutions are clearly asymptotically optimal for linear processor arrays. However, since these problems can be solved on the standard sequential architecture in time $O(N \log h)$ and $O(n)$, respectively, these solutions do not provide a significant speed-

up, especially when taking into account the amount of additional hardware necessary.
Their solutions are based on an exhaustive search strategy which involves $O(N^2)$ comparisons. It is easy to see that $O(N^2)$ comparisons can not be executed on a Mesh-of-Processors of size N using fewer than $O(N)$ steps. Thus, such methods cannot be generalized to yield $O(\sqrt{N})$ time solutions on a Mesh-of-Processors (Umeo, H., private communications).
Compared to the algorithms in [5] the solutions presented in this paper have the following advantages:
- They yield a *significant speedup.*
- From a VLSI point of view, the hardware complexity (area) of a linear array and a mesh differ only in a constant factor. Hence, our $O(\sqrt{N})$ time solutions for the Mesh-of-Processors also yield a significant *increase in efficiency.*
- The solutions presented in this paper are more general and include the solutions in [5]. Atallah [3] has shown that any $O(\sqrt{N})$ time algorithm on a mesh can be simulated on a linear array to run in linear time (but not vice versa).

2 Computing the visibility polygon from a point

Consider a given polygon $\mathbb{P}$ with holes $\mathbb{P}_1, \ldots, \mathbb{P}_h$ and a point p (from which to compute the visibility polygon) contained in $\mathbb{P}$. Let N be the total number of edges.
Assume further the polygon is given by the set $\mathbb{E}$ of edges of $\mathbb{P}$, $\mathbb{P}_1, \ldots, \mathbb{P}_h$ where each edge e is directed such that the interior or $\mathbb{P}$ is to the left of e. Edges of $\mathbb{P}$ are directed in counterclockwise order around the border of $\mathbb{P}$ whereas the edges of the holes are directed in clockwise order (see Fig. 8 for an illustration).
For an edge $e=xy$ we will refer to vertex x as the *start vertex* and to vertex y as the *end vertex* of e.

Description of the required input

The algorithm for computing the visibility polygon from p assumes the following initial configuration:
- Each PE of the Mesh-of-Processors stores the coordinates of the two vertices of one distinct (arbitrary) edge $e \in \mathbb{E}$.
- Each PE also stores the coordinates of the point p.

Description of the output

The visibility polygon will be reported as the set of all visible intervals where a *visible interval* is defined as a maximal interval of an edge $e \in \mathbb{E}$ that is entirely visible from p; see Fig. 9. The same edge may contain several different visible intervals.
In particular, when the algorithm terminates then each of the visible intervals will be stored in one (arbitrary) PE. Note, that the number of visible intervals is at most $O(N)$.
From the set of all visible intervals a list of the edges of the visibility polygon in clockwise ordering (represented, e.g., as snake-like ordering in the mesh) can be easily obtained in time $O(\sqrt{N})$ by sorting the visible intervals by the angle of the polar coordinates (with respect to center p) of their

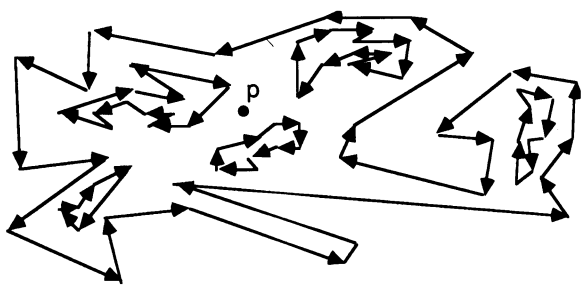

Fig. 8. Representation of a polygon with holes by a Set $\mathbb{E}$ of directed edges

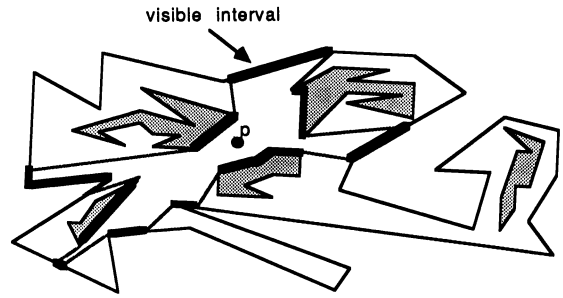

Fig. 9. The set of all visible intervals

start vertices; see [20] for an $O(\sqrt{N})$ time sorting algorithm.

2.1 Global structure of the algorithm

To compute the visible intervals, the algorithm will consider polygon $\mathbb{P}$ together with its holes as an unordered set $\mathbb{E}$ of directed edges and compute for each edge in the visible intervals (if they exist). A point q on an edge $e \in \mathbb{E}$ is visible from p if the open line segment from p to q does not share a point with any other edge $e' \in \mathbb{E}$.

Consider the two horizontal rays R_l and R_r emanating from p to the left and right, respectively (see Fig. 10). In the remainder we will describe how to compute in time $O(\sqrt{N})$ the visible intervals for the set $\mathbb{E}'$ of all edges $e \in \mathbb{E}$ below R_l and R_r; for edges intersecting R_l or R_r only their part below R_l and R_r will be considered (as shown in Fig. 10). Note, that the number of edges $e \in \mathbb{E}'$ may be N in the worst case. Hence, for simplicity, we will also refer to N to denote the number of edges in $\mathbb{E}'$.

The visible intervals for the set of edges above the rays R_l and R_r can be computed in a second analogous step.

Since the complete description of the algorithm is quite involved we will first sketch the algorithm omitting important details to be further explained in the subsequent Sects. 2.2 and 2.3.

The basic idea for computing in parallel the visible intervals for all edges $e \in \mathbb{E}'$ is based on the following observation:

Consider a ray R originating at p which splits $\mathbb{E}'$ into two sets $\mathbb{E}_l$ and $\mathbb{E}_r$ of edges to the left and right of R (see Fig. 10); those edges intersecting R are split into two edges, one for each subset. For any point q on an edge $e \in \mathbb{E}_r$, $q \notin R$, the straight line segment from p to q cannot be intersected by any edge $e' \in \mathbb{E}_l$, and vice versa.

Hence, the visible intervals for $\mathbb{E}_l$ and $\mathbb{E}_r$ can be computed independently in parallel. The union of both sets of visible intervals yields the set of visible intervals of $\mathbb{E}'$ except that adjacent visible intervals, on those edges $e \in \mathbb{E}'$ which intersect R, have to be merged to form one contiguous interval.

The global structure of the proposed algorithm for computing the visible intervals for all edges below R_l and R_r can therefore be expressed using the following divide-and-conquer approach:

(1) *Split*:
The ray R originating at p to the vertex with median polar angle (with respect to center p) of the vertices of all $e \in \mathbb{E}'$ is computed
- R splits the area below R_l and R_r into two sectors Sect_l and Sect_r to the left and right of R, respectively.
- R splits $\mathbb{E}'$ into two subsets $\mathbb{E}'_l$ and $\mathbb{E}'_r$, where $\mathbb{E}'_l$ and $\mathbb{E}'_r$ are the sets of segments $e \in \mathbb{E}'$ contained in Sect_l and Sect_r, respectively; edges (properly) intersecting R are split into two parts, one for each sector.

The mesh is split into two submeshes.
Each subset of edges, $\mathbb{E}'_l$ and $\mathbb{E}'_r$, is moved into one submesh. (Each edge is stored in one arbitrary PE of the submesh)

(2) **IF** the maximum number of edges stored in a submesh is smaller than some constant C
THEN
Solve:
Each submesh (in parallel) computes the visible intervals for the current set of edges.

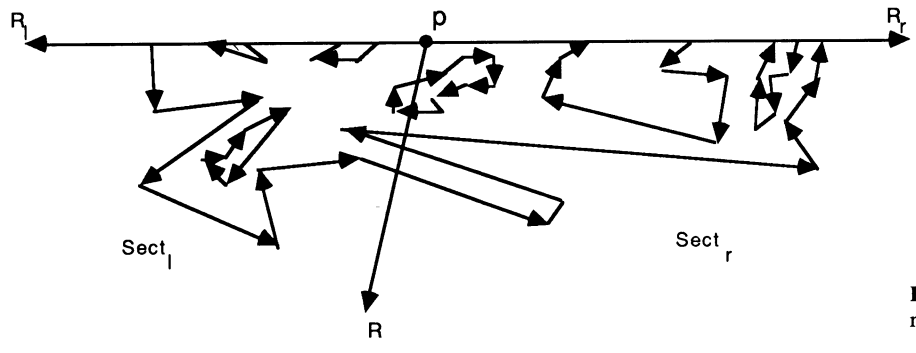

Fig. 10. Basic divide-and-conquer mechanism

ELSE
Recur:
Each submesh which stores more than C edges (in parallel) recursively computes the visible intervals for its current set of edges.

(3) *Merge*:
All pairs of visible intervals which share the same common point with R and are contained in the same edge $e \in \mathbb{E}'$ are determined; each such pair is merged to form one contiguous interval.

2.2 A solution to the problem of edges duplication

The major problem encountered by the above recursive algorithm is that in the split phase all edges intersecting the ray R are split into two edges, one for each sector. If, e.g., all N edges intersect R then the total number of edges doubles since each edge is duplicated and the number of edges for each subproblem is again N.
If this happens repeatedly then the mesh may overflow since it can only store $O(N)$ edges and, furthermore, the recursion may not terminate since the problem size does not decrease.
To overcome this problem we consider the following classification of edges $e \in \mathbb{E}'$:
An edge e is called *luff edge* if p is on the left side of e and *lee edge* if p is on the right side of e; see Fig. 11a and b for an illustration.
With these definitions the following observations are a straightforward consequence of the Jordan Curve Theorem:

Observation 1:
(a) Lee edges cannot contain a visible interval.
(b) The edges in set $\mathbb{E}$ represent a polygon with holes; i.e., every edge is part of a closed curve (see Figs. 9, 10). Thus, every point q contained in $\mathbb{P}$ which is invisible from p because of a lee edge $e \in \mathbb{E}'$ (i.e., the straight line segment between q and p is intersected by e) is also invisible from p because of a luff edge $e' \in \mathbb{E}'$ (i.e., the straight line segment between q and p is intersected e').
(c) From (a) and (b) it follows that the deletion of lee edges does not change the set of visible intervals; i.e., after deletion of a lee edge e every point is visible from p if and only if it is visible from p for the complete set of edges including e.
(d) The edges $e \in \mathbb{E}'$ intersecting the ray R form an alternating sequence of luff and lee edges which is starting and ending with a luff edge; see Fig. 10 and 11c.
Two edges which are of the same type (luff or lee) and are intersected by R at a common point (see Fig. 11d) count as one edge (of type luff or lee, respectively); two edges of different type which are intersected by R at a common point (see Fig. 11e) count as two distinct edges.

Observation 1 leads to the basic idea on how to overcome the duplication problem: After the ray R has been selected to divide the problem into two subproblems of equal size as described in Sect. 2.1 every *luff edge* intersecting R is *split* into two edges, one for each sector, and every *lee edge* intersecting R is deleted.
From observation 1d it follows that the number of luff edges (intersected by R) exceeds the number of lee edges (intersected by R) by one. Every luff edge intersecting R except e_{last}, the luff edge intersecting R furthest from p (see Fig. 11c), has a subsequent lee edge intersecting R. Therefore, after eliminating all lee edges and splitting all luff edges intersected by R, the total total number of edges increases only be one. The edge e_{last} can be eliminated by broadcasting e_{last} to all edges in Sect_l and Sect_r, reducing them to the part which is not invisible from p because of e_{last}, and eliminating e_{last} from further consideration in subsequent split phases. The visible intervals contained in e_{last} can be easily computed at the end of the merge phase.
Furthermore, it is obvious that the numbers of edges added and deleted in Sect_l and Sect_r, respectively, are the same.
Therefore (with the above modifications), after the split phase terminates the total number of edges remains (at most) constant and every sector contains (at most) $\frac{N}{2}$ edges.
However, when the algorithm continues to recursively compute the visible intervals in Sect_l and Sect_r on one half of the mesh, each, Observation 1d may no longer be valid:
Assume that Sect_l is again split by another ray into two subsectors then the edges intersecting this ray may no longer be an alternating sequence of luff and lee edges since some of the lee edges may have already been deleted in a previous split phase.
Consider, in general (for any stage of the algo-

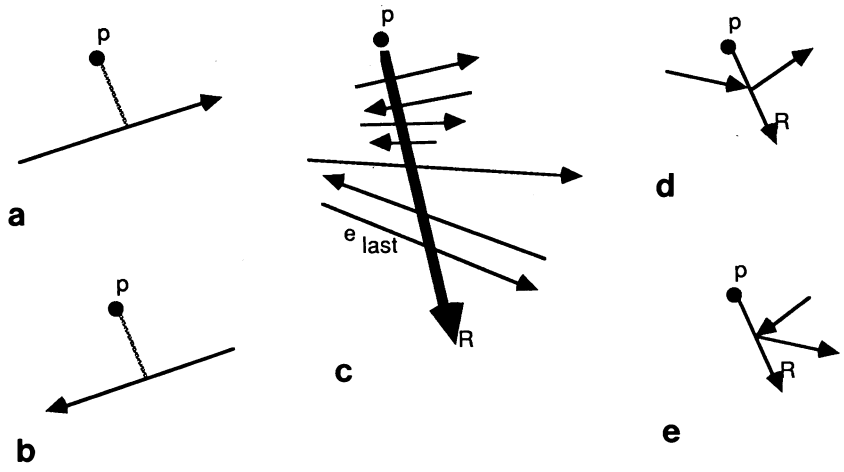

Fig. 11a–e. Luff and lee edges. **a** A luff edge. **b** A lee edge. **c** Luff and lee edges intersecting the ray R. **d** Two luff edges intersected by R at a common point. **e** A lee and luff edge intersected by R at a common point

rithm), a ray R subdividing a sector between two rays R_1 and R_2 into two subsectors Sect_1 and Sect_2, and the sequence of edges intersecting R sorted in increasing order with respect to the distance of the point of intersection with R from p.

Assume that there are $k>1$ luff edges intersecting R which are not succeeded by a lee edge. Hence, when all luff edges intersecting R are split into two luff edges and all lee edges intersecting R are deleted then the total number of edges increases by k. If k is large then this may lead to the same problems as described at the beginning of this section.

Consider the first luff edge e_{first} intersecting R which is not succeeded by a lee edge; see Fig. 12a, (Note, that from all edges intersecting R and succeeding e_{first} only luff edges whose successor is also a luff edge are depicted in Fig. 12.) The lee edge e_{lee} missing between e_{first} and its successor has been deleted in a previous split phase. Thus, e_{lee} must also intersect either R_1 or R_2. Assume, we know that e_{lee} intersects R and R_1 then for all luff edges intersecting R which are successors of e_{first} the part of these edges contained in Sect_1 is not visible from p because of e_{lee}. Hence, these edges do not have to be split into two parts (one for each sector). Instead, they can simply be reduced to the part contained in Sect_2 (see Fig. 12b). In fact, all other active edges in Sect_1 which are not visible from p because of e_{lee} can also be deleted without changing the result.

This yields the result that the total number of edges increases by at most one. This additional edge can again be avoided by eliminating e_{last} as described above. With this, the total number of edges does

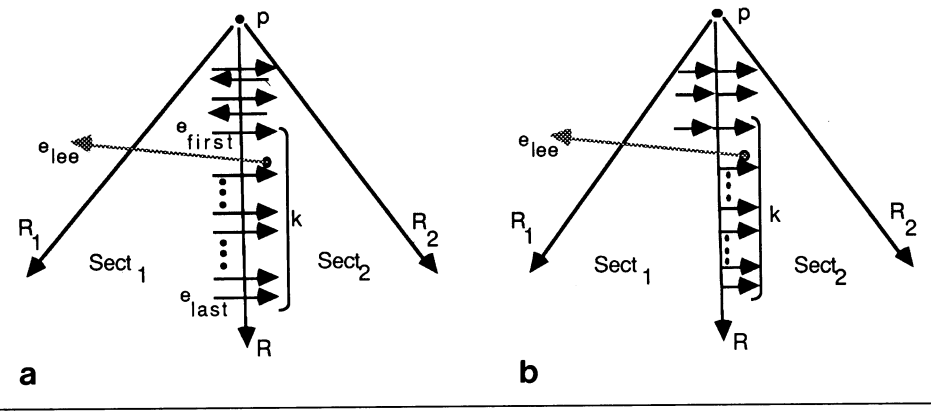

Fig. 12. Active luff edges intersecting R without a subsequent intersecting lee edge

not increase. However, $Sect_2$ may now contain many more edges than $Sect_1$. In the worst case $k=N$ and, hence, $Sect_2$ contains N edges while $Sect_1$ contains one edge only. This difference in the sizes of the subproblems may unbalance the recursion scheme such that the algorithm may take more than $O(\sqrt{N})$ steps.
In the following Sect. 2.3 we will introduce an upper bound k_f for k such that if $k \leq k_f$ for each split phase then the divide-and-conquer mechanism is sufficiently balanced to yield an optimal time algorithm.
In case $k > k_f$ a rebalancing step which selects another "better" ray R' to divide the problem is performed as follows:
Consider the set $\mathbb{E}^*$ of all luff edges in $Sect_2$ which share their start vertex with the ray R; see Fig. 13. Note, that $|\mathbb{E}^*| \geq k$.
For all $e \in \mathbb{E}^*$ the visible intervals for e, pretending that except for the set $\mathbb{E}^*$ there was no other edge contained in $Sect_2$, are computed. Since all $e \in \mathbb{E}^*$ share their start vertex with R this can be performed in time $O(\sqrt{|\mathbb{E}^*|})$ by applying some sorting procedures (the details will be described in Sect. 2.3).
Obviously, each $e \in \mathbb{E}^*$ has at most one such visible interval which also contains the end vertex of e. If we shrink each $e \in \mathbb{E}^*$ to its visible interval, if if exists, and delete it otherwise then it is easy to see that for the entire set of edges in $Sect_2$ the visible intervals remain unchanged.

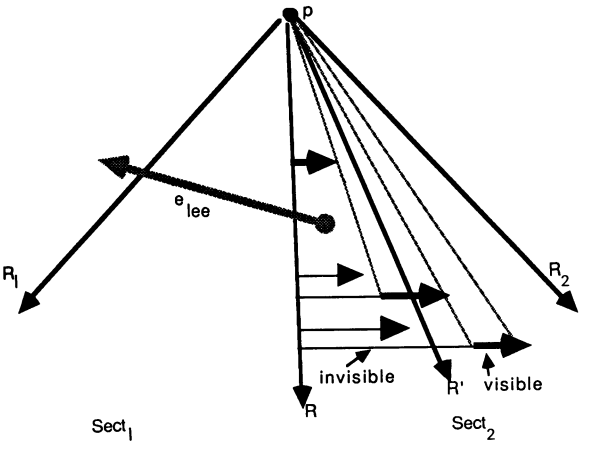

Fig. 13. Computing the Visible Intervals for the Set $\mathbb{E}^*$

The new ray R' is defined by the median of the angles of the polar coordinates (with respect to center p) of all vertices of the final set of edges between R_1 and R_2.
Compared to the initial set of edges in $Sect_1$ and $Sect_2$, all edges in $Sect_1$ which are not visible from p because of e_{lee} have been deleted; in $Sect_2$, edges $e \in \mathbb{E}^*$ have been shrunk such that the new end vertex is now closer to R_2.

Observation 2:
(a) R' lies between R and R_2 and (b) R' is intersected by at most one (modified) edge $e \in \mathbb{E}^*$.

It will be shown in Sect. 2.3 that from this observation it follows that the number of luff edges intersecting R' which are not succeeded by a lee edge is smaller then k_f.
Therefore, R' splits the problem into two sufficiently balanced subproblems.
In the following section, the implementation of the split and merge phase of the algorithm will be explained in detail.
Note, that we assumed e_{lee} to intersect R and R_1. However, it is necessary to determine whether in fact this is the case or whether the symmetric case, i.e. e_{lee} intersects R and R_2, occurs. The problem arising here is that e_{lee} has been deleted in previous steps to limit the number of edges the Mesh-of-Processors has to store. It turns out that we can not simply delete lee edges intersecting a split ray R but have to retain some data to determine in subsequent split phases which of the above cases occurs. On the other hand it must be ensured that these data are not again duplicated during split phases.

2.3 Complete description of the algorithm

In this section we will describe the complete version of the algorithm to compute in time $O(\sqrt{N})$ the visible intervals for the set $\mathbb{E}'$ of all edges $e \in \mathbb{E}$ below the two opposite horizontal rays R_l and R_r starting at p (where edges $e \in \mathbb{E}$ intersecting R_l or R_r are reduced to the part below the intersection point); see Fig. 10. As shown in Sect. 2.1, this yields an optimal $O(\sqrt{N})$ time algorithm for computing the visibility polygon from a point contained in a polygon with holes.

In order to solve the problem of duplicate edges as described in Sect. 2.2 each edge $e \in \mathbb{E}'$ will be initially stored as follows:

- If e is a luff edge then one copy of e, which will be referred to as *active edge*, is stored in an arbitrary PE.
- If e is a lee edge then three copies of e, which will be referred to as *active edge*, *passive entering edge*, and *passive leaving edge*, respectively, are stored in an arbitrary PE.

A lee edge which is either a passive entering edge or a passive leaving edge will be referred to as *passive edge*; the initial set of all active and passive edge as described above will be referred to as Σ_i.
The active edges represent the actual set $\mathbb{E}'$. The purpose of the passive edges is to retain information about lee edges which have been deleted. The major difference between active and passive edges is that active edges are duplicated if they intersect a ray R which is introduced to subdivide the problem (see Fig. 14a) whereas passive edges are not duplicated in such a case. Instead, if a passive edge e and ray R share a point u then e is modified as follows:

- If e is an entering passive edge with end vertex v then e is reduced to the edge with start vertex u and end vertex v (see Fig. 14b).
- If e is a leaving passive edge with start vertex v then e is reduced to the edge with start vertex v and end vertex u (see Fig. 14c).

Entering passive edges will always be stored in the sector which contains their end vertex whereas leaving passive edges will be stored in the sector which contains their start vertex.
In the remainder of this section we will present in detail the algorithm to compute in time $O(\sqrt{N})$ the visible intervals for the edge set $\mathbb{E}'$. In particular, we describe

- the initial configuration of the Mesh-of-Processors, (the output is, as before, the set of visible intervals of $\mathbb{E}'$ where each visible interval is stored in one arbitrary PE)
- the split phase,
- the recur/solve phase, and
- the merge phase.

After all steps of the algorithm have been presented it's correctness and time complexity will be established.

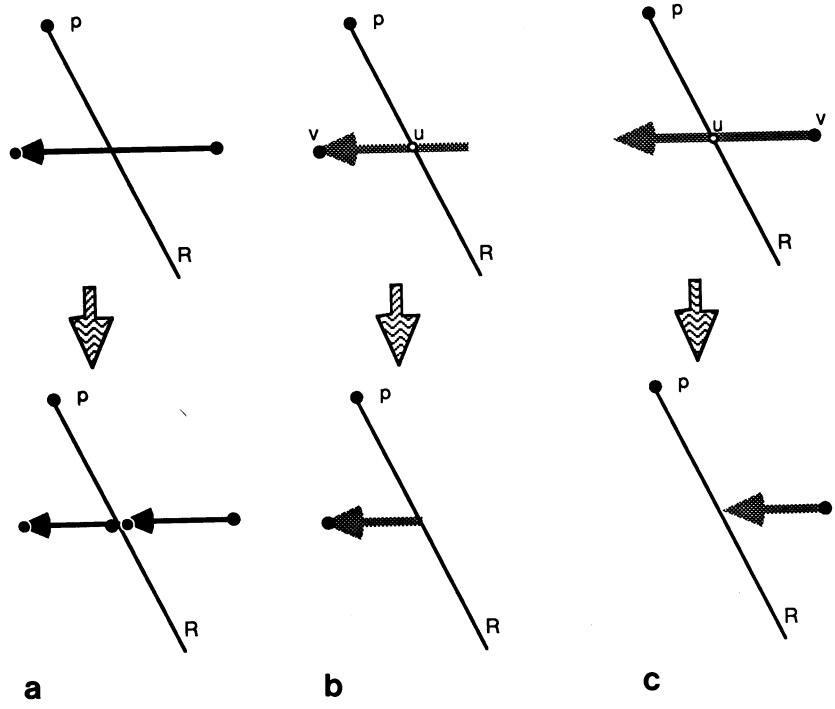

Fig. 14. Active (**a**), entering passive (**b**), and leaving passive (**c**) edges intersecting a split ray R

Initial configuration of the Mesh-of-Processors

The Mesh-of-Processors is initialized as follows:
- Each PE of the Mesh-of-Processors stores the coordinates of the vertices of at most three edges $e \in \mathbf{\Sigma}_i$.
- Each PE stores the coordinates of the point p and the angle of the left and right boundary R_l and R_r of the current sector.

The split phase

The split phase is the most complicated part of the algorithm. The sequence of steps given below describes the split phase for a set $\mathbf{\Sigma}$ of active and passive edges in the sector between two rays R_1 and R_2 (R_1 and R_2 are emanating downwards from p and R_1 is to the left of R_2). Initially, $\mathbf{\Sigma}=\mathbf{\Sigma}_i$, $R_1=R_l$, and $R_2=R_r$.
Let vert($\mathbf{\Sigma}$) denote the set of start and end vertices of all active edges $e \in \mathbf{\Sigma}$, start vertices of all leaving passive edges $e \in \mathbf{\Sigma}$, and end vertices of all entering passive edges $e \in \mathbf{\Sigma}$. Furthermore, let $n := |\text{vert}(\mathbf{\Sigma})|$.

Step S1:
A split ray R is selected such that the direction of R is the median angle of the polar coordinates (with respect to center p) of the vertices $v \in \text{vert}(\mathbf{\Sigma})$. Let Sect_1 and Sect_2 denote the subsector between R_1 and R, and the subsector between R and R_2, respectively.

Step S2:
(a) All active edges $e \in \mathbf{\Sigma}$ intersecting R are sorted by increasing distance of their intersection with R from p.
(b) k, e_{last}, e_{first}, and e_{lee} (if exist) as defined in Sect. 2.2 are determined.

Step S3:
(a) All active luff edges intersecting R (except e_{last}) are split into two active luff edges, one for each subsector Sect_1 and Sect_2, respectively, and active lee edges intersecting R are deleted. All entering and leaving passive edges which intersect R are reduced to the part contained in Sect_1 and Sect_2, respectively, as described at the beginning of this section.
(b) All active edges are reduced to the portion which is not invisible from p because of e_{last}; e_{last} is deleted as an active edge. (However, information about e_{last} will be retained for the merge phase; see Step S6).

Step S4:
If $k>1$ then the following is executed.
If e_{lee} is a leaving passive edge then all active edges in Sect_1 which are not visible from p because of e_{lee} are deleted (see Fig. 12). If e_{lee} is an entering passive edge then all active edges in Sect_2 which are not visible from p because of e_{lee} are deleted.
Note, that every passive edge whose active copy (active lee edge) was deleted in a previous split phase is contained in a sector only as either entering or leaving passive edge (see Step S3a).

Step S5:
If $k>k_f=\frac{2}{5}n$ then the following Steps (a) and (b) are executed.
(a) A new split ray R' is determined as follows:
Assume, e_{lee} is a leaving passive edge; otherwise replace Sect_2 by Sect_1 and 'clockwise' by 'counterclockwise'.
The set $\mathbb{E}^*$ of all active luff edges in Sect_2 which share their start vertex with R (see Fig. 13) is determined. For all $e \in \mathbb{E}^*$ the visible intervals, pretending that except for the set $\mathbb{E}^*$ there was no other edge in Sect_2, are computed as follows:
All $e \in \mathbb{E}^*$ are sorted in clockwise ordering with respect to the angle of the polar coordinates (with respect to center p) of their end points. This ordering will be referred to as *polar ordering*.
All $e \in \mathbb{E}^*$ are sorted in increasing order with respect to the distance of their start points (all start points lie on R) from p. This ordering will be referred to as *R ordering*.
For each $e \in \mathbb{E}^*$, from all those edges in $\mathbb{E}^*$ that have lower rank with respect to R ordering, the edge e' with maximum rank with respect to polar ordering is computed as described in [9] pp. 305–306. If the line through p and the end vertex of e' has a common point u with e then the visible interval of e (with respect to $\mathbb{E}^*$) is the interval from u to the end vertex of e, otherwise it does not exist.
Each $e \in \mathbb{E}^*$ is reduced to the visible interval (with respect to $\mathbb{E}^*$) if exists; otherwise, e is deleted. Let $\mathbf{\Sigma}'$ denote the set of all remaining active and passive edges in Sect_1 and Sect_2.

The new split ray R', whose direction is the median angle of the vertices $v \in \text{vert}(\mathbb{\Sigma}')$, is computed.

(b) The split phase is started again at Step S2 with R' and $\mathbb{\Sigma}'$ instead of R and $\mathbb{\Sigma}$, respectively.
(We will show that for R' and $\mathbb{\Sigma}'$ the above condition for the execution of Steps S5a and S5b does not hold and, therefore, in the second execution of the split phase the algorithm will proceed with Step S6.)

Let $\mathbb{\Sigma}_1$ and $\mathbb{\Sigma}_2$ denote the sets of all remaining active and passive edges in Sect_1 and Sect_2, respectively. Let e_{last} and e'_{last} denote the edge e_{last} of Step S3b with respect to R and R' (if $k > k_f$), respectively.

Step S6:
The Mesh-of-Processor is split into two submeshes M_1 and M_2 (by either a horizontal or vertical split line to minimize the diameter of the submeshes) such that the ratio of number of PEs in M_1 to number of PEs in M_2 is $\frac{|\mathbb{\Sigma}_1|}{|\mathbb{\Sigma}_2|}$.
The edges in $\mathbb{\Sigma}_1$ and $\mathbb{\Sigma}_2$ are moved to the PEs in M_1 and M_2, respectively (at most three edges per PE). Furthermore, each PE updates its current sector (all PEs in M_1 belong to the sector between R_1 and R; all PEs in M_2 belong to the sector between R and R_2). A PE located at the border between M_1 and M_2 stores the edges e_{last} and e'_{last} (if exist).

The recur/solve phase

Let $n_1 := |\text{vert}(\mathbb{\Sigma}_1)|$ and $n_2 := |\text{vert}(\mathbb{\Sigma}_2)|$.
If $\max\{n_1, n_2\} < 7$ then M_1 and M_2 (in parallel) directly compute the visible intervals for $\mathbb{\Sigma}_1$ and $\mathbb{\Sigma}_2$, respectively. Otherwise, each submesh M_i ($i = 1, 2$) with $n_i \geq 7$ recursively computes (in parallel) the visible intervals for its set $\mathbb{\Sigma}_i$ of edges.

The Merge phase

All pairs of visible intervals which share the same common point with R and are contained in the same edge $e \in \mathbb{\Sigma}'$ are determined; each such pair is merged to form one contiguous interval.
Furthermore, e_{last} and e'_{last} (if $k > k_f$) are broadcasted through M_1 and M_2, and the visible intervals contained in e_{last} and e'_{last} determined.

Correctness and time complexity of the algorithm

We will now prove the correctness of the algorithm. In particular, we will prove for the above algorithm that the total problem size does not increase and that the sizes of the subproblems are sufficiently balanced. To simplify exposition we will prove this not in terms of the total number of edges and number of edges of the subproblems, respectively, but in terms of number of vertices of these edges which is equivalent. (For passive entering and leaving edges only their and and start vertices, respectively, are counted.)

Lemma 3. *Let $\mathbb{\Sigma}, \mathbb{\Sigma}_1, \mathbb{\Sigma}_2, n, n_1, n_2, e_{\text{lee}}$ and k be defined as in the algorithm and let k' denote the value of k for the second execution of Steps S2 to S5* (if $k > k_f$).
(a) $n_1 + n_2 \leq n$
(b) If $k \leq k_f = \frac{2}{5}n$ then $n_1 \leq \alpha n$ and $n_2 \leq \alpha n$ for some $0 < \alpha < 1$.
(c) If $k > k_f = \frac{2}{5}n$ then, for the second execution of Steps S2 to S5, $k' \leq k_f = \frac{2}{5}n$.

Proof.
(a) Follows from Sect. 2.2 and the fact that passive edges are not duplicated.
(b) Assume that $k \leq k_f = \frac{2}{5}n$ and that e_{lee} is a leaving passive edge as shown in Fig. 12 (otherwise exchange for the remainder n_1 and Sect_1 with n_2 and Sect_2, respectively). From Steps S3 and S4 of the algorithm, and Sect. 2.2, it follows that the number of vertices in Sect. 1 decreases by at least k whereas the number of vertices in Sect_2 increases by k.
Hence, $n_1 \leq \frac{n}{2} - k \leq \frac{1}{2}n$ and $n_2 \leq \frac{n}{2} + k \leq \frac{9}{10}n$.
Therefore, (b) holds for $\alpha = \frac{9}{10}$.
(c) Let $k > k_f = \frac{2}{n}n$. From Step S5 of the algorithm and Observation 2b it follows that $k' \leq m - |\mathbb{E}^*| + 2$, where $\mathbb{E}^*$ is defined as in the algorithm and m is the number of active edges at the end of Step S5a (when R' is determined).

Since $m \leq \frac{n}{2}$ and $|\mathbb{E}^*| > k > \frac{2}{5}n$ (see Sect. 2.2) it follows that $k' \leq \frac{n}{2} - \frac{2}{5}n + 2 \leq \frac{2}{5}n$, for $n \geq 7$. Since for $n < 7$ the visible intervals are computed directly (see recur/solve phase), part (c) follows. □

Lemma 3a shows that the total number of vertices, and therefore also the total number of edges, does not increase. From Lemma 3b it follows that the selection of the upper bound $k_f = \frac{2}{5}n$ for k provides that the sizes of the subproblems are in the worst case balanced with a ratio of 9 to 1 (with respect to number of vertices). Furthermore, Lemma 3c shows that in case $k > k_f$ the modification of the edge set and selection of the new split ray R' (Step S5) ensures that in the second execution of Step S2 to S5 of the split phase $k' \leq k_f$.

Note that the selection of $k_f = \frac{2}{5}n$ is not the only possible one. In fact, it is easy to see that every $k_f = \beta n$ for $\frac{1}{4} < \beta < \frac{1}{2}$, provided that $n \geq \frac{1}{\beta - \frac{1}{4}}$, is possible.

Theorem 4. *The visibility polygon from a point p contained in an N-vertex polygon (possibly with holes) can be computed on a Mesh-of-Processors of size N in asymptotically optimal time* $O(\sqrt{N})$.

Proof. The correctness of the algorithm follows from Sects. 2.1, 2.2, and Lemma 3. Furthermore, it is easy to see that the split and merge phase can be performed in time $O(\sqrt{n}) = O(\sqrt{N})$ since they can both be implemented by a constant number of the standard $O(\sqrt{N})$ time operations listed in Sect. 2.1 and the technique described in [9] which also has a time complexity of $O(\sqrt{N})$. Therefore, it follows from Sect. 1.3 and Lemma 3b that the time complexity of the algorithm is $O(\sqrt{N})$.
From Lemma 3 it also follows that, in addition to p and its current sector, every PE has to store at most 3 active or passive edges. Furthermore, each PE which is located at the boarder of a submesh may have to store the edges e_{last} and e'_{last} (if exist) created in the respective split phase. Since every PE can be located at no more than 4 borders, it follows that for each PE the total space requirement is a constant number of registers. □

3 Extensions and applications

3.1 Parallel visbility, the visibility hull

The parallel visibility problem for direction d can be seen as a special case of the visibility from a point p located at infinity in direction $-d$.
In fact, the algorithm in Sect. 2 for computing the visible intervals from a point p can be easily modified to compute the visible intervals with respect to parallel visibilty 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 positive x-coordinate and (2) replacing for every vertex with coordinates (x, y), the angle and distance of the polar coordinates (with respect to center p) by y and x, respectively.

Theorem 5. *The visible intervals and the visibility hull of an N-vertex polygon* $\mathbb{P}$ *with respect to parallel visibility in a given direction d can be computed on a Mesh-of-Processors of size N in asymptotically optimal time* $O(\sqrt{N})$.

3.2 Separability of polygons

An application of visibility hulls arises when detecting separability of polygons. Consider two N-vertex polygons $\mathbb{P}$ and $\mathbb{Q}$. Toussaint [19] has shown that $\mathbb{P}$ 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 d 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 polygons can be detected in time $O(\sqrt{N})$ as described in [16], we get the following

Theorem 6. *On a Mesh-of-Processors of size* $2N$ *it can be decided in asymptotically optimal time* $O(\sqrt{N})$ *whether an N-vertex polygon is separable from another N-vertex polygon in a given direction d.*

Toissaint [19] has also proved that a set of 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, Miller and Stout [16] showed that for M N-vertex polygons it can be decided on a Mesh-of-Processors of size M N in time $O(\sqrt{MN})$ whether any of these polygons intersect. This yields

Theorem 8. *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 polygons are sequentially separable in a given direction d.*

References

1. Asano T, Asano T, Guibas L, Hersberger J, Imai H (1985) Visibility polygon search and Euclidean shortest paths. Proc IEEE Symp FOCS, pp 154–164
2. Asano T (1985) An efficient algorithm for finding the visibility polygon for a polygonal region with holes. Trans IECE Jpn, vol E-68, no 9, pp 557–559
3. Atallah MJ (1985) Simulations between mesh-connected processor arrays. Proc 20th Ann Allerton Conf Commun Control Comput, Monticello, III, (October 1985), pp 268–269
4. Aggarval A, Chazelle B, Guibas L, O'Dunlaing C, Yap C (1985) Parallel computational geometry. Proc 26th IEEE Symp FOCS, Portland Oregon (October 1985), pp 468–477
5. Asano T, Umeo H (1987) Systolic algorithms for computing the visibility polygon and triangulation of a polygonal region. Proc Int Workshop Parallel Algorithm Architect, Suhl, GDR (May 1987), pp 77–85
6. Chazelle B (1984) Computational geometry on a systolic chip. IEEE Trans Comput, C-33 (9):774–785
7. Chazelle B, Ottmann T, Soisalon-Soinen E, Wood D (1983) The complexity and decidability of Separation^{TM}. Tech Rep CS-83-34, Data Structuring Group, Univ Waterloo
8. Dehne F (1986) Parallel computational geometry and clustering methods. Tech Rep SCS-TR-104, School Comput Sci, Carleton Univ, Ottawa
9. Dehne F $O(n^{1/2})$ Algorithms for the maximal elements and ECDF searching problem on a mesh-connected parallel computer. Inf Proc Lett 22:303–306
10. Dehne F, Sack J-R (1987) Translation separability of sets of polygons. The Visual Computer 3 (4):225–233
11. Freeman H, Loutrel PP (1967) An algorithm for the two-dimensional "hidden line" problem. IEEE Trans Electron Comput EC-16 (6):784–790
12. El Gindy H, Avis D (1981) A linear algorithm for computing the visibility polygon from a point. J Algorithms 2:186–197
13. Lee DT (1983) Visibility of a simple polygon. Comput Vision Graph Image Proc 22:207–221
14. Lodi E, Pagli L (1986) A VLSI solution to the vertical segment visibility problem. IEEE Trans Comput C-35 (10):923–928
15. Miller R, Stout QF (1984) Computational geometry on a mesh-connected computer. Proc IEEE Int Conf Parallel Proc, pp 66–73
16. Miller R, Stout QF (1987) Mesh computer algorithms for line segments and simple polygons. Proc IEEE Int Conf Parallel Proc, pp 282–285
17. Preparata FP, Shamos MI (1985) Computational geometry. Springer, Tokyo Berlin Heidelberg New York
18. Sack J-R, Toussaint GT (1985) Translating polygons in the plane. Proc STACS '85, Saarbrücken, Federal Republic of Germany, pp 310–321
19. Toussaint GT (1985) Movable separability of sets. In: Toussaint GT (ed) Computational Geometry. North Holland, Amsterdam New York Oxford Tokyo, pp 335–376
20. Thompson CD, Kung HT (1977) Sorting on a mesh-connected parallel computer. Commun ACM 20 (4):263–271
21. Ullman JD (1984) Computational aspects of VLSI. Principles of Computer Science Series, Computer Science Press, Rockville, MD