International Journal of Parallel Programming, Vol. 19, No. 3, 1990

Optimal Visibility Algorithms for Binary Images on the Hypercube[1]

Frank Dehne,[2] Quoc T. Pham,[3] and Ivan Stojmenović[4]

Received June 1988; Revised October 1990

Consider a $n \times n$ binary image. Given a direction D, the parallel visibility problem consists of determining for each pixel of the image the portion that is visible (i.e., not obstructed by any other black pixel of the image) in direction D from infinity. A related problem, referred to as point visibility, is to compute for each pixel the portion that is visible from a given point p. In this paper, we derive $O(\log n)$ time SIMD algorithms for each of these two problems on the hypercube, where one processor is assigned to every pixel of the image. Since the worst case communication distance of two processors in a n^2-processor hypercube is $2 \log n$, it follows that both of the above algorithms are asymptotically optimal.

KEY WORDS: Hypercube Algorithms; Image Processing; Parallel Algorithms; Visibility.

1. INTRODUCTION

A d-dimensional hypercube is a set of 2^d synchronized processing elements $P(i)$, $0 \leqslant i \leqslant 2^d - 1$, where two processors $P(i)$ and $P(j)$ are connected by a bidirectional communication link if and only if the binary representations

[1] This paper summarizes a preliminary version [Ref. 1] and short note on a possible improvement [Ref. 2] presented at the 1988 *IFIP WG 10.3 Working Conference on Parallel Processing* and 1988 *Allerton Conference on Communication, Control and Computing*, respectively. The first and third authors' research are partially supported by the Natural Sciences and Engineering Research Council of Canada.

[2] Center for Parallel and Distributed Computing, School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6.

[3] Department 7H65, Bell-Northern Research, P.O. Box 3511, Ottawa, Canada K1Y 4H7.

[4] Department of Computer Science, University of Ottawa, Ottawa, Canada K1N 6N5.

0885-7458/90/0600-0213$06.00/0

of i and j differ in exactly one bit. (See Refs. 3 and 4 for more details on hypercubes.)

In Refs. 5–7 several hypercube algorithms are proposed for geometric and topological problems on digitized images: labeling of the extreme points, finding the diameter, finding the smallest enclosing box of a figure, component labeling, etc.

In this paper, we continue this line of research by studying two visibility problems: the parallel visibility problem and the point visibility problem. Visibility problems have been extensively studied for the sequential model of computation.[8,9] The notion of visibility in geometric objects is important for a large number of geometric applications; e.g., the hidden line problem in graphics,[10] the shortest path problem for points in a plane with polygonal obstructions,[11] and the separability problem for planar polygonal objects.[12–14] In practice, studying these problems for digitized images (rather than for sets of polygons) is of particular importance, since the input data consists frequently of a binary image. Consider for example the shortest path problem for a robot in a factory hall, where the description of the polygonal obstructions is obtained via a camera mounted at the ceiling of the hall. In this case, an algorithm which can be applied directly to the image is desirable since it does not need a costly image to polygons transformation. (See also Ref. 15.)

Consider a digitized image π consisting of $n \times n = 2^d$ pixels $\pi(i)$ indexed from 0 to $2^d - 1$ in row-major ordering, and stored in the hypercube such that every pixel $\pi(i)$ is assigned to processor $P(i)$. (If the pixels are numbered in Gary code ordering, a transformation can be performed in time $O(\log n)$.[16,17]

For a given direction D, the parallel *visibility* problem for the image π is to compute for each pixel $\pi(i)$ the portion of $\pi(i)$ that is visible in direction D; i.e., the portion of $\pi(i)$ that is illuminated by light in the direction D from a light source at infinity, assuming that the black pixels of π obstruct light; see Fig. 1a. The problem of computing the *visibility* from a *point* p is to compute for each pixel $\pi(i)$ the portion of $\pi(i)$ that is visible from p (i.e., the portion illuminated by a light source located at p); see Fig. 1b.

In this paper we present, for both of the discussed visibility problems, $O(d) = O(\log n)$ time algorithms for the hypercube architecture. Since the worst case communication distance of two processors in a hypercube of size n^2 is $2 \log n$, it follows that both solutions are asymptotically optimal. The remainder of this paper is organized as follows: Section 2 presents the algorithm for determining parallel visibility and, in Section 3, we solve the point visibility problem.

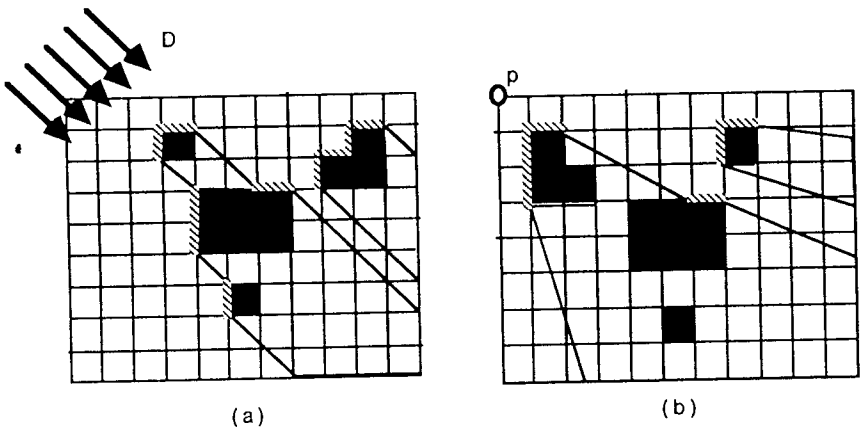

Fig. 1. (a) Parallel visibility in direction D; (b) visibility from point p.

2. PARALLEL VISIBILITY

The basic geometric idea for our algorithm is to divide the image π into strips parallel to direction D (as illustrated in Fig. 2), where each strip is the portion of light intersecting the top edge of one pixel in the top row

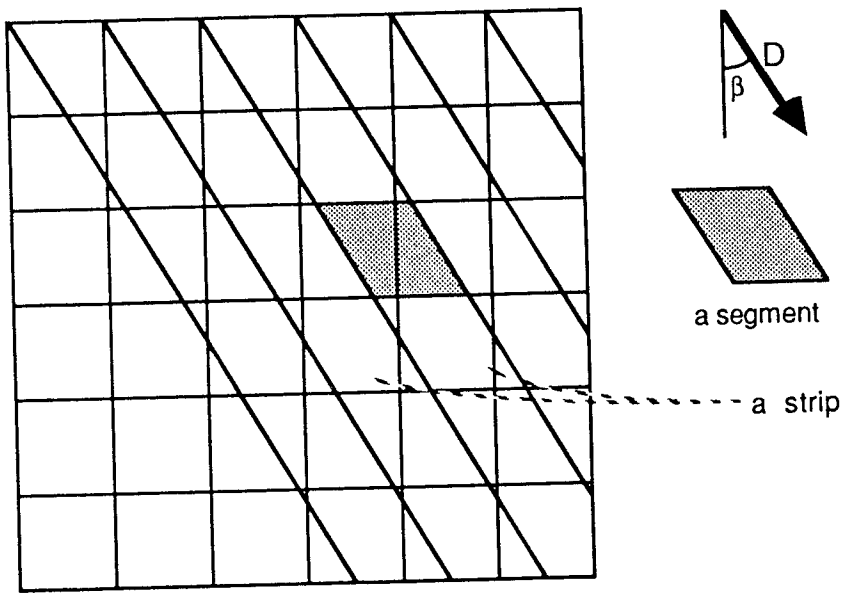

Fig. 2. Divising the image into strips and segments.

of the image (as if it were unobstructed by any black pixel). For the remainder, let w_0 denote the horizontal width of a (square shaped) pixel.

We shall assume without loss of generality that the angle β between the North-South direction and the direction D (in counter-clockwise direction) is between 0 and 45°; otherwise, the algorithm can be obtained similarly by symmetry.

We further divide each strip into *segments*, where a segment is the portion of a strip contained in one row of pixels (see Fig. 2). The leftmost of the pixels intersected by a segment will be referred to as the *representative* pixel of the segment.

For each segment, we define the *black interval* to be the projection of the black portion of the segment onto the cross-section of the strip; the *white interval* is the complement of the black interval (with respect to the cross-section of the strip). The projection of the visible portion of the segment onto the cross-section of the strip will be referred to as *visible interval* of the segment.

Lemma 1.

(a) No pixel $\pi(i)$ is properly contained in a strip (i.e., every $\pi(i)$ intersects either the left or the right border of its strip).

(b) Each segment intersects at most three neighboring pixels.

(c) Every pixel intersects at most three strips.

(d) Each white or visible interval consists of at most two connected components, and both components are adjacent to the boundary of the strip.

Proof. (a) Follows from the fact that the horizontal width of each pixel and strip are equivalent. (b) Follows from (a) and the fact that $\beta \leqslant 45°$. (c) Let w_p and w_s denote the width of a pixel and a strip with respect to the direction perpendicular to D, respectively. From $0 \leqslant \beta \leqslant 45°$ it follows that $w_s \leqslant w_p \leqslant 2w_s$. Hence, a pixel can intersect at most three strips. (d) Follows from $w_s \leqslant w_p$. ∎

Our algorithm solves the parallel visibility problem by computing, for all strips in parallel, the visible intervals for all segments in the strip. From Lemma 1d it follows that each white or visible interval requires only $O(1)$ memory space. For simplicity we describe the algorithm only for the portion of the image covered by strips in Fig. 2; i.e., the region bounded by the top, right, and bottom edges of the image, and the line through the top left corner of the image parallel to direction D. The parallel visibility problem for the remaining pixels can be solved in a second, analogous step.

The general structure of our algorithm is as follows:

(1) All *white intervals* are computed. Every processor determines the segment it represents (if exists) and the corresponding strip number. Then, it examines its local neighborhood and computes the white interval for the segment it represents.

(2) All *visible intervals* are computed by using the following simple geometric idea: Consider, within each strip, the sorted ordering of the segments with respect to direction D such that the topmost segment is the first in this ordering. The visible interval of each segment is the intersection of the white intervals of its predecessors (the visible interval of the topmost segment is the entire cross-section of the strip).

We employ four standard hypercube operations, all of which can be executed in time $O(\log n)$ on a hypercube with n^2 processors (see Refs. 3, 18, and 19):

Distribute: Assume that processors $P(i)$, $0 \leqslant i < j \leqslant n^2$, each store a record $r(i)$ and a processor destination address $\text{dest}(i)$ such that $\text{dest}(i) < \text{dest}(i+1)$ for $0 \leqslant i < j$. The distribute operation consists of routing, for each of these $P(i)$, the record $r(i)$ to processor $P(\text{dest}(i))$.

Concentrate: The concentrate operation is the reverse of the distribute operation.

Shift (special case of the distribute operation): Every processor $P(i)$ representing a pixel $\pi(i)$ sends a record $r(i)$ to the processor $P(j)$ that represents the pixel $\pi(j)$ which is obtained by shifting $\pi(i)$ k pixels to the right (or to the left, upwards, downwards, respectively).

Partial sum: Given N values $a_0,\ldots, a_{n^2}$ (each value a_i is stored in processor $P(i)$) and an associative binary operator *, then the partial sum operation consists of computing the values $a_0, a_0{*}a_1, a_0{*}a_1{*}a_2,\ldots, a_0{*}a_1{*}a_2{*}\ldots{*}a_{n^2}$.

In part 2 of our algorithm, for each strip (in parallel) a partial sum problem has to be solved where the operands for the partial sum operations are the white intervals of the segments and the associative binary operator is set intersection. However, the $O(\log N)$ partial sum algorithm assumes that the operands for each partial sum problem are stored in exactly one-hypercube which, in general, is not the case for the white intervals of a strip. We observe that the processors which store a row or column of pixels form a sub-hypercube. Therefore, our strategy is to move each of the strips into a column-subcube of the hypercube so that the

partial sums, for all strips, can be computed independently in $O(\log n)$ time. Finally, the obtained visibility information is returned to the original segment locations and propagated to the neighboring pixel in the segment.

The following is the final description of our algorithm:

(1) All strips are "rotated" to the left to coincide with the columns as follows: In each row of segments (i.e., the sub-hypercube of processors representing the segments in one row of pixels), all segments are shifted simultaneously to the left (using the strip number to determine how far to shift) such that the leftmost segment is stored in the leftmost processor of the row. For each row-subcube this operation can be performed independently in time $O(\log n)$ using the shift operation described earlier.

(2) In each column-subcube, which now stores the white intervals of the segments of one strip, the partial sum operation can be performed independently in $O(\log n)$ steps.

(3) The invese of Step 1 is performed to return the "rotated" (actually shifted) segments with the values obtained in the previous step back to their original positions.

(4) After Step 3, each processor has received the visible interval for the segment it represents. Finally, for each segment, the visibility information is propagated from the representative pixels to the neighboring pixel in the segment (using a shift operation) and, for each pixel, the visible portion determined.

Since each of these steps can be executed in time $O(\log n)$, we obtain

Theorem 1. The parallel visibility problem for a digitized image of size $n \times n$ can be solved on a d-dimensional hypercube, $2^d = n \times n$, in time $O(d) = O(\log n)$.

3. POINT VISIBILITY

In order to determine the visibility from a point p, we will assume without loss of generality that the point p is located at the upper left corner of the image; otherwise, the digitized image can be split by the horizontal and the vertical lines through p into (at most) four quadrants and the problem can be solved for each quadrant separately.

In the remainder of this section we will show how to compute for all pixels in the area below the 45° ray emanating from p (again, all angles are defined with respect to the north-south axis and in counter-clockwise

direction) the portion that is visible from p. For all pixels above the ray, the visibility problem can be solved in a second analogous step.

Consider the 22.5° ray emanating from p. It splits the image (below the 45° ray) into two strips whose widths (i.e., the horizontal distance between left and right border) increase with the distance from p and will, eventually, exceed width w_0 which ensures that no pixel is properly contained in a strip (cf. Lemma 1a). At the level where the width of the rightmost strip reaches w_0, each strip is split again (by rays emanating from p) into two strips such that the angles between the borders of the four strips are equal. When the width of the rightmost of these four strips reaches w_0, they are bisected again. This process is repeated until the entire image below the 45° ray is covered (see Fig. 3). We define a sector to be the section of a strip between two consecutive splittings.

Lemma 2.

(a) No pixel is properly contained in a sector (i.e., every pixel intersects either the left or right border of its sector).

(b) Every pixel intersects at most eight sectors.

Proof. (a) Follows from the fact that the horizontal width of a pixel is not smaller than the maximum horizontal width of a sector. (b) For

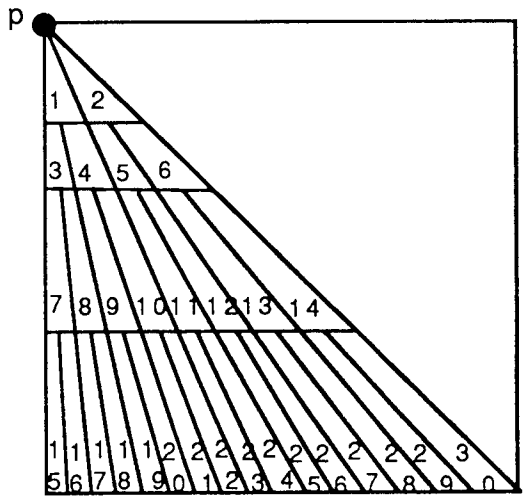

Fig. 3. Subdivising the image into sectors.

every horizontal cut through the image, the width w_l of the leftmost sector intersected by the has the property

$$w_r > w_l = \frac{1+\tan\alpha}{2} w_r \geq \frac{1}{2} w_r$$

where w_r is the width of the rightmost sector (intersected by the cut) and α the angle between the left and right border of the sector ($0° < \alpha \leq 22.5°$). Since $1/2\, w_0 \leq w_r \leq w_0$), and the width of all other sectors on the cut is between w_l and w_r, we obtain for the horizontal width w of any sector (along any horizontal cut): $1/4\, w_0 \leq w \leq w_0$. ∎

Furthermore, we divide each sector into *segments*. A segment of a sector S is the portion of S contained in one row of pixels (see Fig. 4).

For each segment, the *representative pixel*, *white interval*, and *visible interval* are defined in the same way as in Section 2.

The sectors, together with the relation "$<$" defined by

$S_1 < S_2$ if and only if the top horizontal border of S_2 is contained in the bottom horizontal border of S_1,

form a binary tree which will be referred to as sector tree (see Fig. 5). The point visibility problem can be solved by executing for every path from the root to a leaf of the sector tree, for the white initervals of the segments of these sectors, the partial sum operation of Section 2.

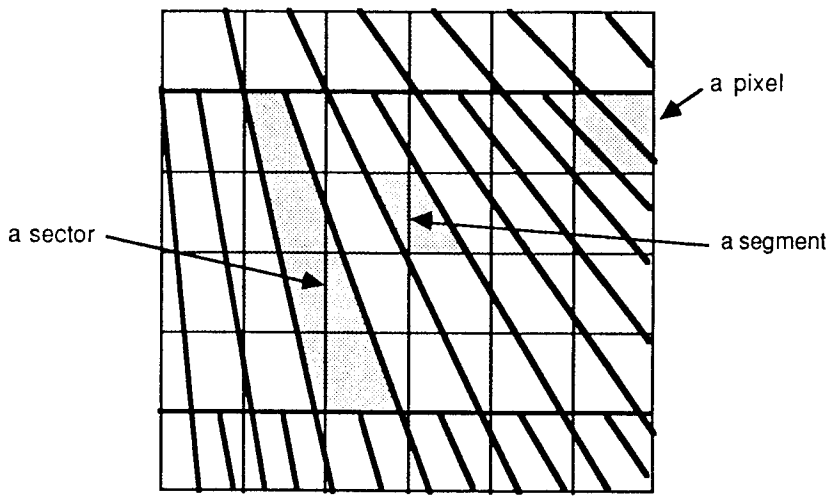

Fig. 4. Sectors and segments.

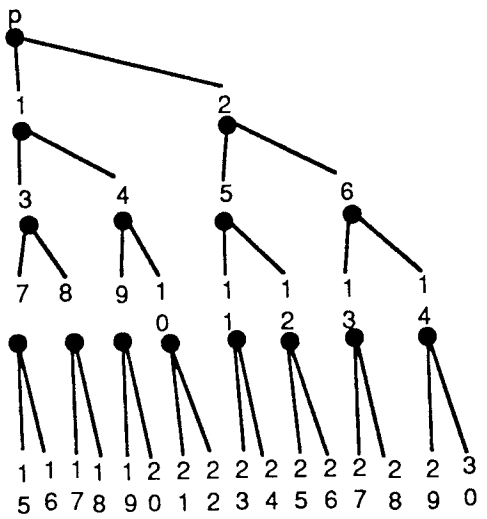

Fig. 5. Sector tree for the subdivision in Fig. 4.

Compared to Section 2, the problem arising here is not only that segments involved in a partial sum operation are (in general) not stored in a sub-hypercube but also that segments in non-leaf sectors are involved in several partial sum operations.

The general idea for solving these problems is to copy, for every path from the root to a leaf of the sector tree, the segments of the involved sectors into a column subhypercube (see Fig. 6).

In order to do this efficiently, we utilize the *generalize* operation[3] of which can be executed in time $O(\log n)$ on a hypercube with n^2 processors:

> Assume that processors $P(i)$, $0 \leqslant i < j \leqslant n^2$, each store a record $r(i)$ and a processor destination address $\text{dest}(i)$ such that $\text{dest}(i) < \text{dest}(i+1)$ for $0 \leqslant i < j$; the result of the generalize operation is that every record $r(i)$ is routed to the processors $P(\text{dest}(i-1)+1),\ldots, P(\text{dest}(i))$.

The following is a detailed description of our hypercube algorithm for solving the point visibility problem (To simplify exposition, we shall assume that every processor representing a pixel simulates four 'virtual' processors, one for each of the at most four respective segments.):

(1) Every processor $P(i)$ determines in constant time which segment, $\text{Seg}(i)$, in which sector, $\text{Sect}(i)$, it represents (if any).

(2) Since the sector tree is a complete binary tree, every $P(i)$ can also determine in constant time the column number, $c(i)$, of the

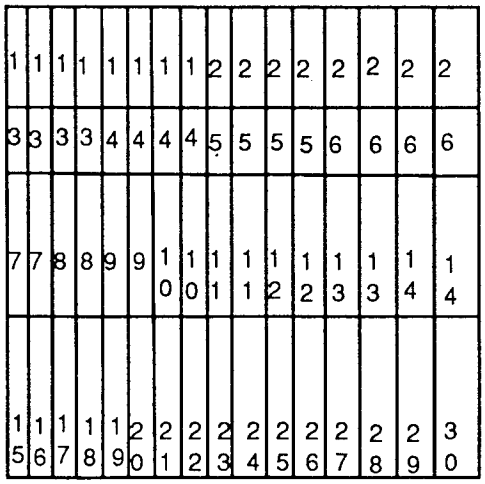

Fig. 6. Copying each path of the sector tree into a column subcube.

representative pixel of the bottommost segment in the rightmost leaf-sector of the sub-sector tree rooted at Sect(i). If Sect(i) is a leaf-sector then $c(i)$ is the column number of the representative pixel of the bottommost segment of Sect(i).

(3) Every $P(i)$ determines in time $O(\log n)$ the white interval, $w(i)$, of Seg(i); see Section 2.

(4) A generalize operation is performed where, for every $P(i)$ representing a segment Seg(i), $r(i) = w(i)$ and dest(i) is the row major index, i^*, of the pixel with the same row number as $\pi(i)$ and column number $c(i)$. (It is easy to see that the requirement $i < j \Rightarrow \text{dest}(i) < \text{dest}(j)$ for generalize operations holds.)

(5) After Step 4, for each path from the root to a leaf of the sector tree, the white intervals of the segments of all sectors on the path are stored in a column subcube in sorter order.
Therefore, for all paths in parallel, the partial sum operation with respect to their white intervals can be computed in time $O(\log n)$.

(6) The visible intervals obtained in Step 5 are returned from each $P(i^*)$ to $P(i)$, the procesor storing the representative pixel of the segment. (This can be implemented with time complexity $O(\log n)$ by using the concentrate operation.) Note that, in Step 5, for all copies of the white interval of a segment the result of the partial sum operation is the same.

Finally (by using the shift operation), for each segment the visible interval is sent in time $O(\log n)$ from the representation pixel to the other pixel in its segment (if exists).

Summarizing, we obtain

Theorem 2. The point visibility problem for a digitized image of size $n \times n$ can be solved on a d-dimensional hypercube, $2^d = n \times n$, in time $O(d) = O(\log n)$.

4. CONCLUSION

In this paper, we presented $O(\log n)$ time hypercube algorithms for solving the parallel visibility problem, and computing the visibility from a point, for a $n \times n$ binary image. Since the worst case communication distance of two processors in a hypercube of size n^2 is $2 \log n$, it follows that both solutions are asymptotically optimal. While the algorithms as described here assume one processor per pixel, they can be immediately generalized, using standard simulation, to $O(n^2/p \log p)$ time algorithms for solving the same problems on an arbitrary size hypercube with $p \leqslant n^2$ processors. Hence, they yield a near optimal speed-up of $O(p/\log p)$ for all $p \leqslant n^2$.

REFERENCES

1. F. Dehne and Q. T. Pham, Visibility algorithms for binary images on the hypercube and the perfect-shuffle computer, *Proc. IFIP WG 10.3 Working Conf. on Parallel Processing*, Pisa (Italy), North-Holland, pp. 117–124 (1968).
2. F. Dehne, Q. T. Pham, and I. Stojmenovic, Optimal visibility algorithms for binary images on the hypercube—preliminary version, *Proc. Allerton Conference on Communication, Control and Computing, Monticello*, Illinois, pp. 1035–1036 (1988).
3. D. Nassimi and S. Sahni, Data broadcasting in SIMD computers, *IEEE Transactions on Computers*, **C-30** (2):101–106 (1981).
4. C. L. Seitz, The cosmic cube, *Comm. of the ACM*, **28**:22–23 (1985).
5. R. Miller and S. E. Miller, Using hypercube multiprocessors to determine geometric properties of digitized pictures, *Proc. IEEE Conf. on Parallel Processing*, pp. 638–640 (1987).
6. R. Miller and Q. F. Stout, Some graph and image processing algorithms on the hypercube, *Proc. Second Conf. on Hypercube Multiprocessors*, pp. 418–425 (1986).
7. Q. T. Pham, Parallel algorithm and architecture for binary image component labeling, Technical Report, Department of Computer Science, University of California, Los Angeles, California (1987).
8. H. ElGindy and D. Avis, A linear algorithm for computing the visibility polygon from a point, *Journal of Algorithms* **2**:186–197 (1981).
9. D. T. Lee, Visibility of a simple polygon, *Computer Vision Graphics and Image Processing* **22**:207–221 (1986).

10. H. Freeman and P. P. Loutrel, An algorithm for the two-dimensional "hidden line" problem, *IEEE Trans. Electron. Comput.*, **EC-16** (6):784–790 (1967).
11. T. Asano, T. Asano, L. Guibas, J. Hersberger, and H. Imai, Visibility polygon search and Euclidean shortest paths, *Proc. of the IEEE Symp. on FOCS*, pp. 154–164 (1985).
12. F. Dehne and J.-R. Sack, Translation separability of sets of polygons, to appear in *The Visual Computer*, Vol. 3, No. 4 (1987).
13. J.-R. Sack and G. T. Toussaint, Translating polygons in the plane, *Proc. STACS '85*, Saarbrücken, Federal Republic of Germany, pp. 310–321 (1985).
14. G. T. Toussaint, Movable separability of sets, *Computational Geometry*, G. T. Toussaint (ed.), North Holland, Amsterdam, New York, Oxford, Tokyo, pp. 335–376 (1985).
15. F. Dehne, A. Hassenklover, J.-R. Sack, and N. Santoro, Computational geometry on a systolic screen, to appear in F. Dehne (ed.), Parallel algorithms for geometric problems on digitized pictures, special issue of *Algorithmica*.
16. S. L. Johnson, Communication efficient basic linear algebra computations on hypercube architectures, *J. on Parallel and Distributed Computing*, **4**:133–172 (1987).
17. I. Stojmenovic, Computational geometry on the hypercube, Technical Report TR-CS-87-100, Computer Science Department, Washington State University, Pullman, Washington (1987).
18. C. P. Kruskal, L. Rudolph, and M. Snir, The power of parallel prefix, *Proc. IEEE Conference on Parallel Processing*, pp. 180–185 (1985).
19. R. E. Tarjan and U. Vishkin, Finding biconnected components and computing tree functions in logarithmic parallel time, *Proc. 25th IEEE Symp. on Foundations of Computer Science*, pp. 12–20 (1984).