Algorithmica (1991) 6: 734–761

Algorithmica

Computational Geometry Algorithms for the Systolic Screen[1]

F. Dehne,[2] A.-L. Hassenklover,[3] J.-R. Sack,[2] and N. Santoro[2]

Abstract. A *digitized plane* Π of size M is a rectangular $\sqrt{M} \times \sqrt{M}$ array of integer lattice points called pixels. A $\sqrt{M} \times \sqrt{M}$ mesh-of-processors in which each processor P_{ij} represents pixel (i, j) is a natural architecture to store and manipulate images in Π; such a parallel architecture is called a *systolic screen.* In this paper we consider a variety of computational-geometry problems on images in a digitized plane, and present optimal algorithms for solving these problems on a systolic screen. In particular, we present $O(\sqrt{M})$-time algorithms for determining all contours of an image; constructing all rectilinear convex hulls of an image (peeling); solving the parallel and perspective visibility problem for n disjoint digitized images; and constructing the Voronoi diagram of n planar objects represented by disjoint images, for a large class of object types (e.g., points, line segments, circles, ellipses, and polygons of constant size) and distance functions (e.g., all L_p metrics). These algorithms imply $O(\sqrt{M})$-time solutions to a number of other geometric problems: e.g., rectangular visibility, separability, detection of pseudo-star-shapedness, and optical clustering. One of the proposed techniques also leads to a new parallel algorithm for determining all longest common subsequences of two words.

Key Words. Computational geometry, Clustering, Convex hull, Digitized pictures, Hulls, Maxima, Mesh-of-processors, Parallel computing, Separability, Systolic array, Visibility, Voronoi diagram.

1. Introduction. A *mesh-of-processors* of size M is a set of M processors P_{ij} $((i, j) \in \{1, \ldots, \sqrt{M}\}^2)$ positioned on a $\sqrt{M} \times \sqrt{M}$ grid where each processor is connected to its four neighbors (if they exist) via communication links. Such an architecture is ideal for representing a *digitized plane* Π of size M, i.e., a rectangular array of M lattice points (or *pixels*) with integer coordinates $(i, j) \in \{1, \ldots, \sqrt{M}\}^2$. On a mesh-of-processors, a set of n disjoint images $I_1, \ldots, I_n$ (where each image I_i is defined as a subset of Π) can naturally be stored as follows (see Figure 1):

Each processor P_{ij} has a *color-register* C-Reg(i, j) with value

$$\text{C-Reg}(i, j) = \begin{cases} k & \text{if } (i, j) \in I_k \quad (1 \le k \le n), \\ 0 & \text{otherwise.} \end{cases}$$

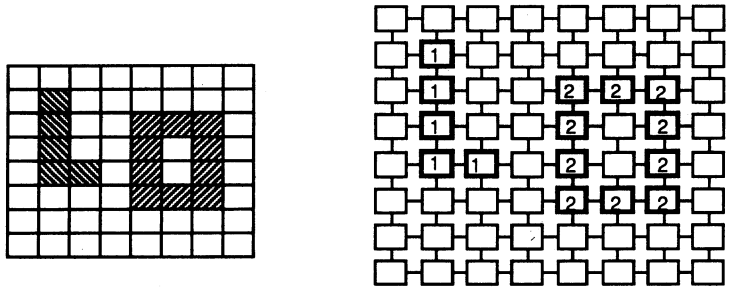
A mesh-of-processors which stores and manipulates images is referred to as a *systolic screen.*

[1] Research supported by the Naural Sciences and Engineering Research Council of Canada. With the Editor-in-Chief's permission, this paper was sent to the referees in a form which kept them unaware of the fact that the Guest Editor is one of the co-authors.

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

[3] Bell-Northern Research, Department 9X41 (Carling), P.O. Box 3511, Station "C," Ottawa, Canada K1Y 4H7.

Received September 20, 1988; revised July 5, 1989. Communicated by Frank Dehne.



(a) (b)

Fig. 1. (a) Two images in Π. (b) Systolic screen representation of these two images.

Over the last three decades, a number of systolic screens have been constructed [R3], [U], e.g., the ILLIAC III [M1], the MPP (designed by NASA for analyzing LANDSAT satellite data) [R1], the CLIP architecture [D5], and more recently the Connection Machine (with its frame buffer) [H1].

While most of the early applications of the systolic screen focused on low-level operations (such as, e.g., edge detection, filtering, or component labeling), recent research has started considering high-level geometric problems as studied in the rapidly growing field of *computational geometry*. This development is similar to that of parallel computational geometry in object space, where geometric objects are represented in analytical form (e.g., via boundary representations) rather than as images. In practice, however, geometric data is often obtained from sensors or photographs; hence, parallel computational-geometry algorithms in image-space can be directly applied to images and do not need costly conversions of images into object-space representation.

Miller and Stout [SM], [MS2] have presented $O(\sqrt{M})$-time algorithms for computing, e.g., the distance between two images as well as the diameter, convex hull, and smallest enclosing circle of an image, and for testing convexity and linear separability of images.

In this paper we consider a variety of other computational-geometry problems on images in the digitized plane of size M, and present $O(\sqrt{M})$-time algorithms for their resolution on a systolic screen; since any nontrivial routing operation on a $\sqrt{M} \times \sqrt{M}$ mesh-of-processors requires $\sqrt{M}$ steps, these algorithms are optimal for the systolic screen. We present $O(\sqrt{M})$-time algorithms for determining all contours and all rectilinear convex hulls (peeling) of an image (Section 2); solving the parallel and perspective visibility problem for N disjoint images, and related separability problems (Section 3); constructing the Voronoi diagram of N planar digitized objects for a large class of object types (e.g., points, line segments, circles, ellipses, and polygons of constant size) and distance functions (e.g., all L_p metrics), and optical clustering (Section 4). Incidentally, one of the proposed techniques also leads to a new parallel algorithm for determining all longest common subsequences (Section 2). These results have been obtained as part of the ongoing investigation by the authors in the field of parallel computational geometry, and preliminary descriptions of some of these results have been

presented at conferences [DHSS], [DSS]; related results can also be found in [D2]–[D4], [DHS], [DP], and [DS2].

Many of the algorithms described in this paper are based on a particular scheme for systematically passing messages to each processor of the systolic screen, referred to as *systolic-screen sweep*. A sweep is specified by a function $f: \{T_0, \ldots, T\} \to \mathbb{P}(\{1, \ldots, \sqrt{M}\}^2)$, where $\mathbb{P}(\{1, \ldots, \sqrt{M}\}^2)$ denotes the powerset of $\{1, \ldots, \sqrt{M}\}^2$, satisfying the condition that for each pair $(i, j) \in \{1, \ldots, \sqrt{M}\}^2$ there exists exactly one $t \in \{T_0, \ldots, T\}$ such that $(i, j) \in f(t)$. The argument t corresponds to a step number, or "time"; a systolic-screen sweep for such a function f is a message-passing mechanism where at time t, every processor P_{ij} with $(i, j) \in f(t)$ receives a message. Unless otherwise stated, we assume that $T_0 = 1$ and $T = \sqrt{M}$.

Assuming that, at time $t = 1$, all P_{ij} with $(i, j) \in f(1)$ contain such a message, a systolic-screen sweep can be efficiently executed in time $O(\sqrt{M})$ (and, hence, $T = O(\sqrt{M})$ if the function f has the property that for every P_{ij} with $(i, j) \in f(t)$ there exists a processor $P_{i'j'}$ with $(i', j') \in f(t)$ such that the processor distance of P_{ij} and $P_{i'j'}$ is $O(1)$; the *processor distance* of two processors P_{ij} and $P_{i'j'}$ is defined as their Manhattan distance $|i - i'| + |j - j'|$.

We utilize three principal systolic-screen sweeps:

- *vertical*, where $f(t) = \{(1, t), (2, t), \ldots, (\sqrt{M}, t)\}$ (or *horizontal*, where $f(t) = \{(t, 1), (t, 2), \ldots, (t, \sqrt{M})\}$),
- *diagonal*, where $T_0 = 2$, $T = 2\sqrt{M}$, and $f(t) = \{(i, j) | i + j = t \text{ and } 1 \le i, j \le \sqrt{M}\}$, and
- *layered* from (i_p, j_p), where

$$f(t) = \{(i, j) | \max\{|i_p - i|, |j_p - j|\} = t - 1 \text{ and } 1 \le i, j \le \sqrt{M}\}.$$

Vertical, horizontal, and diagonal sweeps will also be applied in the direction opposite to that defined above (e.g., for a vertical sweep we can define $f(t) = \{(1, \sqrt{M} - t), (2, \sqrt{M} - t), \ldots, (\sqrt{M}, \sqrt{M} - t)\}$); particular applications also require the sweep direction to be parallel to some direction other than parallel to the screen coordinates axes (see, e.g., Section 3.1 on parallel visibility).

Before we start presenting the algorithms, we introduce some definitions which are used in the remainder of this paper [R2], [K]:

- The *eight neighbors* (for short called *neighbors*) of a pixel $(x, y) \in \Pi$ are the eight pixels $(x \pm 1, y)$, $(x, y \pm 1)$, $(x + 1, y \pm 1)$, and $(x - 1, y \pm 1)$, if they exist. The *four neighbors* of (x, y) are the four pixels $(x \pm 1, y)$ and $(x, y \pm 1)$, if they exist. The *border* Bord(I) of an image $I \subseteq \Pi$ is the set of all pixel of I which have an eight neighbor in $\Pi - I$. The *interior* of I, $I -$ Bord(I), is denoted by Int(I).
- A *path* [4-*path*] from $p \in \Pi$ to $q \in \Pi$ is a sequence of point $p = p_0, \ldots, p_r = q$ such that p_i is a neighbor [4-neighbor] of p_{i-1}, $1 \le i \le r$. An image $I \subseteq \Pi$ is *connected* if for every $p, q \in I$ there exists a path from p to q consisting

entirely of pixels of I. An image I which is connected is referred to as a (*digital*) *object*.

- With each pixel $p = (i, j) \in \Pi$ we associate its *cell*

$$\langle p \rangle := [i - 0.5, i + 0.5] \times [j - 0.5, j + 0.5] \subseteq \mathbb{R}^2$$

and with each image $I \subseteq \Pi$ its *region*

$$\langle I \rangle := \bigcup_{p \in I} \langle p \rangle.$$

- Conversely, we define for a set $R \subseteq \mathbb{R}^2$ of points in the real plane its image

$$\mathrm{Im}(R) := \{p \in \Pi \mid \langle p \rangle \cap R \neq \varnothing\}.$$

- Given a point $s \in \mathbb{R}^2$ and radius $r \in \mathbb{R}$, then $\mathrm{disc}(s, r) := \{x \in \mathbb{R}^2 \mid d(s, x) \leq r\}$ denotes the *disc* with center s and radius r, where d is a distance function to be specified.

2. Contours, Layers, and Applications. We first examine two well-known and extensively studied problems: given an image $S = \{s_1, \ldots, s_n\}$, determine all its contours and construct all its (rectilinear) convex hulls. The former problem and its systolic-screen solution is presented in Section 2.1; the algorithm presented here also leads to a new parallel solution for the (nongeometric) problem of determining all *longest common subsequences* of two words (see Section 2.3). An algorithm for the latter problem, which is often referred to as *peeling*, is discussed in Section 2.2.

2.1. Dominance and Contours. Consider an image $S \subseteq \Pi$ containing two pixels $s = (i, j) \in \Pi$ and $s' = (i', j') \in \Pi$, $s \neq s'$. Then pixel s *dominates* pixel s' (denoted by $s \geq s'$) if $i \geq i'$ and $j \geq j'$; pixel $s \in \Pi$ is called *maximal* in S if no other pixel $s' \in S$ dominates s. The set CONTOUR(S) of all maximal pixels of S, sorted by x-coordinate, is called the *contour* of S. (We define $\mathrm{CONTOUR}(\varnothing) := \varnothing$.) The notion of the contour of S can be generalized to define the *k-contour* of S, denoted by CONTOUR(S, k), $k \in \mathbb{N}$, as follows:

$$\mathrm{CONTOUR}(S, 1) := \mathrm{CONTOUR}(S),$$

$$\mathrm{CONTOUR}(S, k + 1) := \mathrm{CONTOUR}(S - [\mathrm{CONTOUR}(S, 1) \cup \cdots \cup \mathrm{CONTOUR}(S, k)]).$$

Since in a digitized plane different pixels may have the same x- or y-coordinate, the following restricted definition of dominance is also useful: given two pixels $s = (i, j)$ and $s' = (i', j')$, $s \neq s'$, then s *strictly dominates* s' (denoted by $s > s'$) if $i > i'$ and $j > j'$. The k-contour with respect to the strict dominance relation is denoted by CONTOUR*(S, k).

Sequential algorithms for determining the maximal elements as well as all contours of a set of points have been extensively studied in the literature (see, e.g., [PS]); for a set of n points, both problems can be solved in time $O(n \log n)$.

For the mesh-of-processors of size n, where each processor stores the coordinates of one point, $O(\sqrt{n})$-time algorithms for determining the maximal elements (and for solving the related ECDF searching problem) have been presented in [D2]. In the following we present an $O(\sqrt{M})$-time systolic-screen algorithm for computing all nonempty k-contours of an image. For ease of description, we first present (and prove the correctness of) an algorithm for the standard dominance relation; we then extend this algorithm to determine all k-contours for the strict dominance relation.

Assume that an image $S = \{s_1, \ldots, s_n\} \subseteq \Pi$ is stored on a systolic screen of size M. In addition to the register C-Reg(i, j) used to store the image, each P_{ij} contains a second register called K-Reg(i, j). Upon termination of the algorithm, the K-Registers will contain the final result, i.e., all k-contours, as follows:

$$\text{for all } P_{ij} \text{ for which } (i,j) \in S: \quad (\text{K-Reg}(i,j) = k) \Leftrightarrow ((i,j) \in \text{CONTOUR}(S,k)).$$

The following algorithm computes *all* k-contours in one diagonal sweep.

ALGORITHM 1: COMPUTING ALL SETS CONTOUR(S, k).

(1) Every processor P_{ij} initializes its K-Register as follows:

$$\text{K-Reg}(i,j) \leftarrow \text{C-Reg}(i,j).$$

(2) Every processor with K-Reg = 1 sends the content of its K-Register to its lower and left neighbors, if they exist.

(3) Every processor P_{ij}, upon receiving value v_u and/or v_r from its upper and/or right neighbor, respectively, updates its K-Register

$$\text{K-Reg}(i,j) \leftarrow \max\{\text{K-Reg}(i,j), \max\{v_u, v_r\} + \text{C-Reg}(i,j)\}$$

and sends the new content of its K-Register to its lower and left neighbors, if they exist. By definition, v_u and v_r are set to 0 if no value is received.

(4) Step 3 is iterated until there are no more messages transmitted.

THEOREM 1. *For any image* $S \subseteq \Pi$, *Algorithm* 1 *computes all nonempty sets* CONTOUR(S, k) *in time* $O(\sqrt{M})$.

PROOF. Each processor representing a pixel of S originates a message, and every processor forwards a received message (possibly with modified content) to its lower and left neighbors, if they exist. Hence, in the worst case, these messages will proceed from the upper right to the lower left corner of the screen taking time $O(\sqrt{M})$. The correctness of Algorithm 1 is proved by induction on $|S|$: For $|S| = 1$, the algorithm obviously provides the correct result. Assume $|S| > 1$ and let

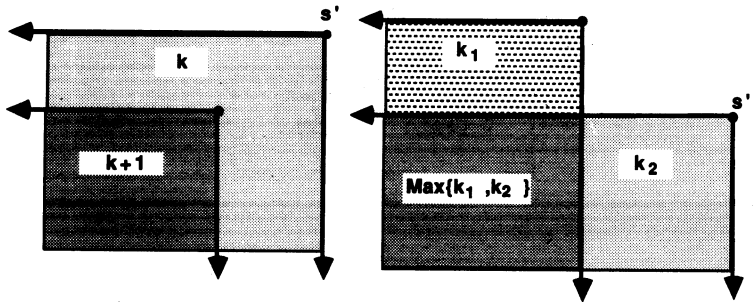


Fig. 2. Update process for nested and overlapping dominance regions.

$s' \in$ CONTOUR(S, 1) be a maximal element of S. Observe that, after execution of the algorithm, the final register contents of a processor is independent of the order of arrival of the messages which originated at other elements. Thus, the execution of the algorithm is equivalent to its execution with respect to $S - \{s'\}$ (which is assumed to be correct), with a subsequent propagation of the message originating at s'. We observe that this final message propagates to all processors which are dominated by s' and correctly updates the solution obtained for $S - \{s'\}$; see Figure 2. □

The algorithm for computing all CONTOUR*(S, k) is essentially identical to Algorithm 1 with the addition of taking into account that a pixel does not dominate pixels with the same x- or y-coordinate. Thus, when a processor receives a message, it needs to determine whether this message has been passed on a horizontal (vertical) path, i.e., a path in which each consecutive pair of pixels are horizontal (vertical) neighbors. To provide the necessary information, an additional bit b is added to each message; by convention, a bit value 0 indicates that this message is traveling on a horizontal or vertical path.

ALGORITHM 2: COMPUTING ALL SETS CONTOUR*(S, k).

(1) Every processor P_{ij} initializes its K-Register as follows:

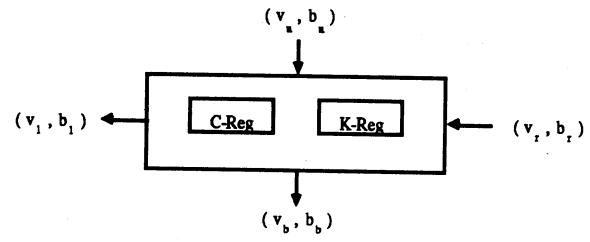
$$\text{K-Reg}(i, j) \leftarrow \text{C-Reg}(i, j).$$

(2) Every processor with K-Reg = 1 sends a message (K-Reg, 0) to its lower and left neighbors, if they exist.

(3) Every processor, upon receiving message (v_u, b_u) and/or (v_r, b_r) from its upper and/or right neighbor, respectively, updates its K-Reg and sends a message (v_b, b_b) and (v_l, b_l) to its lower and left neighbor, respectively; see Figure 3(a). The update of the register K-Reg and determination of the bits b_l and b_b is described in Figure 3(b). The values of v_l and v_b are set to the updated value of register K-Reg.

(4) Step 3 is iterated until there are no more message transmissions.

THEOREM 2. *For any image* $S \subseteq \Pi$, *Algorithm* 2 *computes all nonempty sets* CONTOUR*(S, k) *in time* $O(\sqrt{M})$.



(a)

$b_u b_l$	$v_u > v_r$	$v_u = v_r$	$v_u < v_r$
*	0	[K-Reg]<-max{K-Reg, v_r}; b_l=0; b_b=1-[C-Reg]	
0 *	[K-Reg]<-max{K-Reg, v_u}; b_l=1-[C-Reg]; b_b=0		
*	1	[K-Reg]<-max{K-Reg, v_r+[C-Reg]}; b_l=b_b=1-[C-Reg]	
1 *	[K-Reg]<-max{K-Reg, v_u+[C-Reg]}; b_l=b_b=1-[C-Reg]		
0 0	[K-Reg]<-max{K-Reg, v_r, v_u}		
	b_l=1-[C-Reg]; b_b=0	b_l=b_b=0	b_l=0;b_b=1-[C-Reg]
0 1	[K-Reg]<-max{K-Reg, v_u} b_l=1-[C-Reg]; b_b=0	[K-Reg]<-max{K-Reg, v_r+[C-Reg]} b_l=b_b=1-[C-Reg]	
1 0	[K-Reg]<-max{K-Reg, v_u+[C-Reg]} b_l=b_b=1-[C-Reg]		[K-Reg]<-max{K-Reg, v_r} b_l=0; b_b=1-[C-Reg]
1 1	[K-Reg]<-max{K-Reg,max{v_u,v_r}+[C-Reg]}; b_l=b_b=1-[C-Reg]		

(b)

Fig. 3. (a) Messages sent to/from every processor. (b) Update of register K-Reg and determination of b_l and b_b performed by each processor.

PROOF. The proof follows along the same lines as that of Theorem 1. □

The points of each k-contour define a 4-path of pixels connecting these points, which we refer to as the *k-chain* of S. The set of all pixels which lie on or below the k-chain and above the $(k+1)$-chain are referred to as the *k-strip* of S. We observe that, upon termination of Algorithm 2, the register K-Reg of the processors corresponding to pixels in the k-strip have value k; the processors corresponding to pixels in the k-chain have the additional property that either the upper or the right neighbor has K-Reg $= k - 1$.

COROLLARY 1. *On a systolic streen of size M, all k-chains and k-strips of an image $S \subseteq \Pi$ can be computed in time $O(\sqrt{M})$.*

2.2. Rectilinear Convex Hulls (Peeling). A classical problem in computational geometry, directly related to the maxima determination problem, is the *convex hull construction* problem which has been extensively studied both for sequential (see, e.g., [PS]) and parallel environments (see, e.g., [ACGOY], [MS1], and [MS2]). A useful representation of a set S is the set of its convex layers; this

representation has also been used to obtain an efficient solution for the half-plane range query problem [CGL]. For sets $S \subseteq \mathbb{E}^2$ in the Euclidean plane, a commonly used technique for determining this representation is *peeling*; that is, the iterative process of computing the convex hull of S and removing its vertices from S. Peeling is the two-dimensional analogue to the concept of the alpha-trimmed mean used in robust statistics (see, e.g., pp. 83ff of [S4] and [H3]). Several sequential algorithms for peeling have been proposed by Shamos [S4], Overmars and van Leeuwen [OvL], and Chazelle [C]; an $\Omega(n \log n)$ (sequential) lower bound for this problem has been proved by Shamos [S4].

In a digitized plane, a type of hull of particular interest is the *rectilinear convex hull*, whose sequential determination has been studied, i.e., in [MF], [S1], [S2], and [W]. An image $S = \{s_1, \ldots, s_n\}$ is said to be *rectilinearly convex* if the intersection of its region $\langle S \rangle$ and an arbitrary horizontal or vertical line in $\langle \Pi \rangle$ consists of at most one line segment. The intersection of all rectilinearly convex images $S' \subseteq \Pi$ which contain S is called the *rectilinear convex hull* of S and is denoted by HULL(S). In a digitized plane, peeling an image $S = \{s_1, \ldots, s_n\}$ refers to the following iterative process: compute the *rectilinear* convex hull of S and remove its vertices from S, until S contains no more points.

The kth *rectilinear convex hull* HULL(S, k) of S ($k \in \mathbb{N}$) is therefore defined as follows (see Figure 4):

$$\text{HULL}(S, 0) := \Pi,$$

$$\text{HULL}(S, 1) := \text{HULL}(S),$$

$$\text{HULL}(S, k+1) := \text{HULL}(\text{Int}(\text{HULL}(S, k)) \cap S).$$

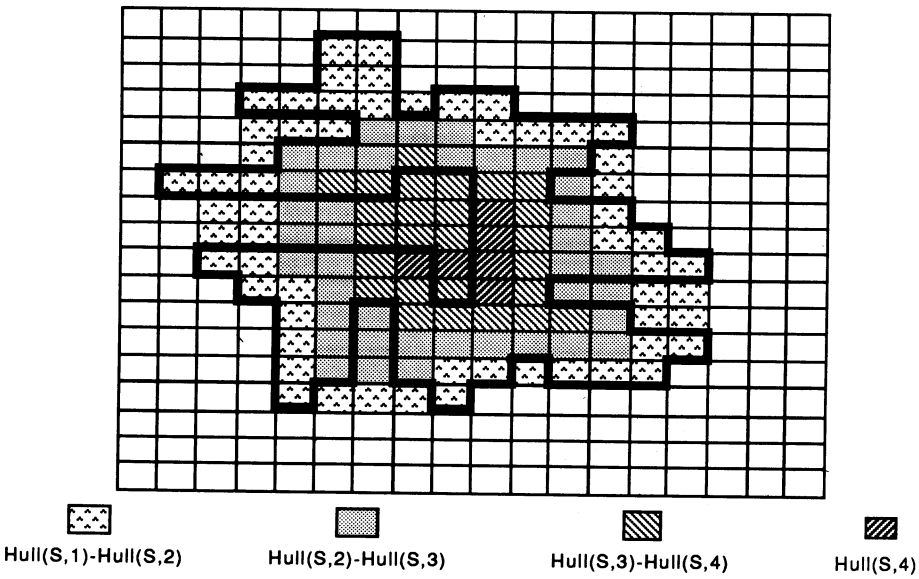


Fig. 4. All hulls Hull(S, k) of an image S (enclosed by the bold line).

Considerable attention has also been given to finding estimators which identify the center of a set S and the depth of points with respect to S (see [S4], [OvL], and [LP]). The *depth* of S, denoted by DEPTH(S), is the largest k such that HULL(S, k) $\neq \emptyset$. For each pixel $s \in \Pi$ we define its depth DEPTH(s, S) in S:

$$\text{DEPTH}(s, S) := k \quad \text{if and only if} \quad s \in \text{Hull}(S, k) - \text{Hull}(S, k+1).$$

Obviously, DEPTH(S) = max{DEPTH(s, S)$|s \in \Pi$}.

Peeling an image and computing the depth of each pixel can be reduced to four executions of the algorithm for computing all k-strips with respect to the strict dominance relation. Given a pixel $s \in \Pi$, we define $k_{NE}(s, S) := k$ ($k_{SE}(s, S) := k$, $k_{SW}(s, S) := k$, $k_{NW}(s, S) := k$) if s is an element of a k-strip of S for the strict dominance relation with respect to the NE direction (SE direction, SW direction, NW direction, respectively); see Figure 5 for an illustration. We observe that, for every pixel $s \in \Pi$, Depth(s, S) = k if and only if $\min\{k_{NW}(s, S), k_{SW}(s, S), k_{NE}(s, S), k_{SE}(s, S)\} = k$. Hence, an image can be peeled by executing Algorithm 2 four times, once for each direction; at each processor the depth of the corresponding pixel is simply the minimum of the four values of its register K-Reg at the end of each iteration.

We obtain

THEOREM 3. *For any image $S \subseteq \Pi$ all* kth *rectilinear convex hulls, the depth of each pixel and the depth of the image can be computed on a systolic screen of size M in time $O(\sqrt{M})$.*

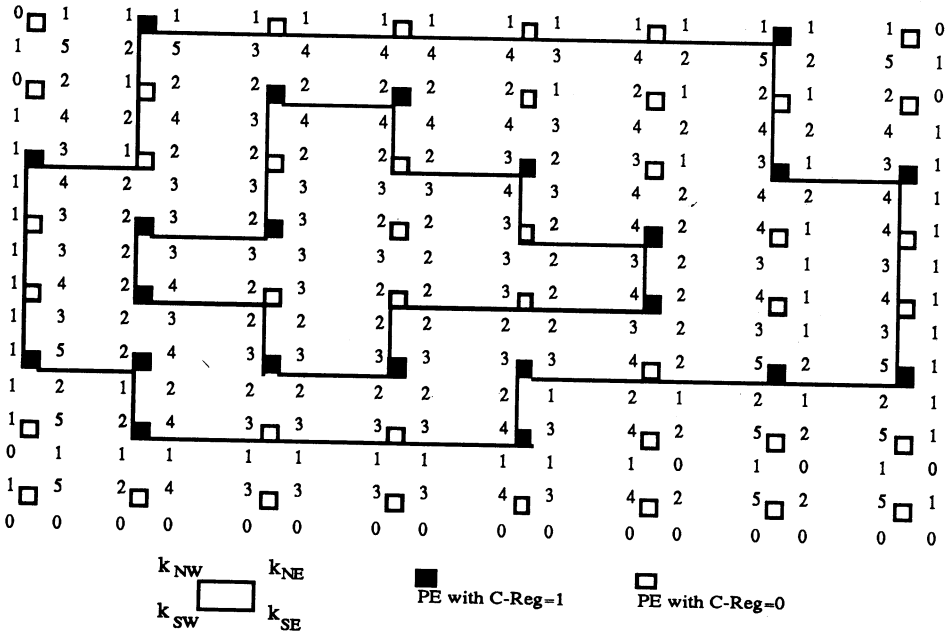


Fig. 5. Peeling an image.

2.3. Longest Common Subsequences. The proposed technique for computing all k-contours (Section 2.1) also yields a new efficient parallel algorithm for finding all longest common subsequences of two given strings.

Given two strings $A = A(1) \cdots A(n)$ and $B = B(1) \cdots B(m)$, $n \geq m$, over some finite alphabet Σ, a *substring* C of A is defined to be any string $C = C(1) \cdots C(r)$ for which there exists a monotone strictly increasing function $f: \{1, \ldots, r\} \to \{1, \ldots, n\}$ with $C(i) = A(f(i))$ for all $1 \leq i \leq r$. The *longest common subsequence problem* is to find a string of maximum length which is á substring of both A and B. This problem has been extensively studied in sequential environments (e.g., see [HS], [H2], and [NKY]). Recently, Robert and Tchuente [RT] introduced a parallel algorithm which computes a longest common subsequence in time $O(n)$ using a one-dimensional systolic array of size m with an additional systolic stack of size n associated with each processor.

We present an algorithm for the more general problem of determining *all* longest common subsequences in time $O(n)$ on a systolic screen of size $n \times m$. All processors are of the same type and thus our solution uses a more homogeneous architecture than that in [RT]; this answers the question posed in [RT]. The central idea which leads to this method is a transformation of the longest common subsequence problem to the k-contour determination; the same reduction was used by Hirschberg in [H2].

LEMMA 1.

(a) *$A(i_1) \cdots A(i_r) = B(j_1) \cdots B(j_r)$ is a common subsequence of A and B if and only if $(i_1, j_1) < (i_2, j_2) < \cdots < (i_r, j_r)$.*

(b) *The length r of a longest common subsequence is*

$$r = \max\{k \in \mathbb{N} \mid \text{CONTOUR}^*(S_{AB}, k) \neq \varnothing\},$$

where $S_{AB} := \{(i, j) \mid A(i) = B(j)\}$.

PROOF. See [H2]. □

As a consequence of Lemma 1, the problem of computing all longest common subsequences can be reduced to the problem of computing a k-contour CONTOUR*(S_{AB}). Every longest common subsequence corresponds to a sequence $s_1, \ldots, s_r$ of points of S_{AB}, where

$$r = \max\{k \in \mathbb{N} \mid \text{CONTOUR}^*(S_{AB}, k) \neq \varnothing\},$$

$s_k \in \text{CONTOUR}^*(S_{AB}, r - k + 1)$, and $s_1 < s_2 < \cdots < s_r$; see Figure 6. Note that there may be an exponential number of longest common subsequences which obviously cannot be reported explicitly in time $O(n)$. Instead, we report for every $(i, j) \in \text{CONTOUR}^*(S_{AB}, k)$ the set

$$\text{Next}(i, j) := \{(i', j') \in \text{CONTOUR}^*(S_{AB}, k - 1) \mid i' > i \text{ and } j' > j\}.$$

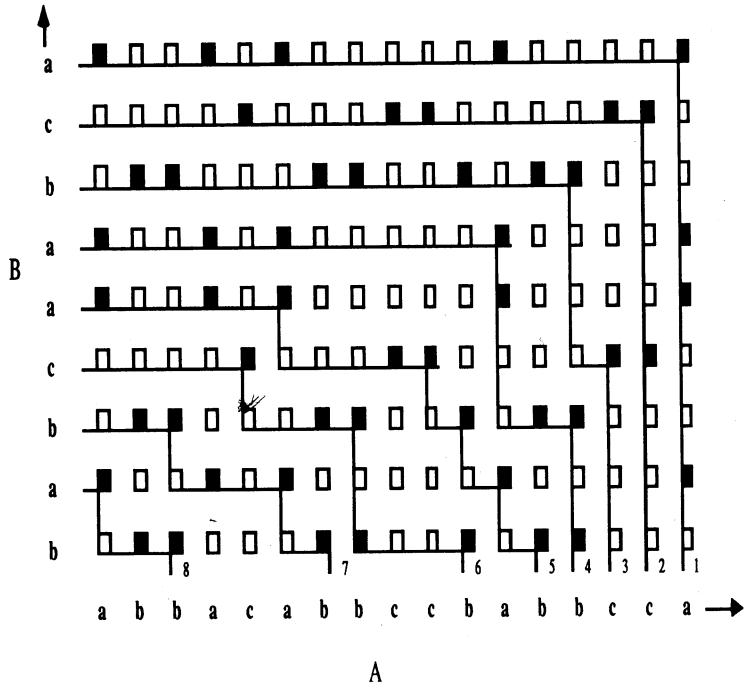


Fig. 6. All k-contours for the set S_{AB}.

With this, the set of all longest common subsequences corresponds to the set of all sequences $s_1, \ldots, s_r$ of points S_{AB} such that $s_1 \in \text{CONTOUR}^*(S_{AB}, r)$ and $s_{k+1} \in \text{Next}(s_k)$, $1 \leq k \leq r-1$. We observe that $\text{Next}(i, j)$ is a sequence of consecutive points (i', j') of $\text{CONTOUR}^*(S_{AB}, k-1)$, sorted by y-coordinate, with $i' \geq \text{Next}_1(i, j)$ and $j' \geq \text{Next}_2(i, j)$ where

$$\text{Next}_1(i, j) := \min\{i'' > i \mid (i'', j'') \in \text{CONTOUR}^*(S_{AB}, k-1)\}$$

and

$$\text{Next}_2(i, j) := \min\{j'' > j \mid (i'', j'') \in \text{CONTOUR}^*(S_{AB}, k-1)\}.$$

Thus, after computing S_{AB} and all its contours with respect to the strict dominance relation, an implicit description of all longest common sub-sequences follows from the two values $\text{Next}_1(i, j)$ and $\text{Next}_2(i, j)$ associated with every point $(i, j) \in S_{AB}$.

All values $\text{Next}_1(i, j)$ can be obtained as follows:

(1) For each $(i, j) \in \text{CONTOUR}^*(S_{AB}, k)$ create two records $(k, i, j, 0)$ and $(k-1, i, j, 1)$.
(2) Sort these records in snake-like ordering [TK] using their first field as the major and their second field as the minor key.

(3) Determine for all records $(k, i, j, 0)$ the next record $(k, i', j', 1)$ with respect to the snake-like ordering (using one row and one column rotation) and send i' back to processor P_{ij} (using a random access write as described in [MS1]).

Since all operations can be performed in $O(n)$ time, we obtain:

THEOREM 4. *All longest common subsequences of two strings $A = A(1), \ldots, A(n)$ and $B = B(1), \ldots, B(m)$, $n \geq m$, can be computed on a mesh-of-processors of size $n \times m$ in time $O(n)$.*

3. Visibility Problems. We now examine problems relating to the visibility of digitzed objects. Visibility problems have been studied extensively in the fields of computer graphics, robotic, and computational geometry. We study these problems in two common models of visibility: the parallel and the perspective model.

Consider a digitized plane Π of size M and a set $I_1, \ldots, I_n$ of n disjoint images in Π stored on a systolic screen of size M. A pixel $(i, j) \in \Pi$ is called *black*, if it is contained in some image I_i $(1 \leq i \leq n)$.

In the *parallel visibility model* it is assumed that a light source is located at infinity, emitting rays parallel to a specified direction d. A point $x \in \langle \Pi \rangle$ is called *visible* in direction d if the ray emanating from x in direction $-d$ (i.e., the direction opposite to d) is not intersected by the region $\langle (i, j) \rangle$ of any black pixel (i, j) (i.e., x is not *obstructed* by any black pixel). In the *perspective visibility model* a light source is located at some point $p \in \Pi$, emitting rays in every direction. A point $x \in \langle \Pi \rangle$ is called *visible* from p if the line segment from p to x is not intersected by the region $\langle (i, j) \rangle$ of any black pixel (i, j).

On a systolic screen, the *visibility problem*, formulated in either the parallel or the perspective model, consists of determining for every pixel $(i, j) \in \Pi$ the set of all visible points on the boundary of $\langle (i, j) \rangle$.

In Sections 3.1 and 3.2, we present $O(\sqrt{M})$-time algorithms to solve the parallel and perspective visibility problem for a set of n disjoint images stored on a systolic screen of size M. These algorithms imply efficient solutions to a variety of other geometric problems on a systolic screen. Some of these applications are presented in Section 3.3. We obtain $O(\sqrt{M})$-time solutions for, e.g., determining the visibility hull of each image and deciding whether a set of images is translation separable (in the sense of [T]), or deciding for each image whether it is pseudo-star-shaped with respect to a point p (in the sense of [DLS]).

3.1. Parallel Visibility. In this section we solve the parallel visibility problem for a set of n disjoint images on a systolic screen of size M and a given direction d. We assume that the angle β between direction d and the horizontal axis is between 0° and 90°; all other cases, are handled symmetrically.

We split $\langle \Pi \rangle$ into m strips $ST_1, \ldots, ST_m$, parallel to direction d; each strip ST_k is bounded by two bounding lines b_k and b_{k+1}, and every strip has the same width w (see Figure 7 for an illustration). A processor P_{ij} belongs to ST_k if $\langle (i, j) \rangle$ intersects ST_k.

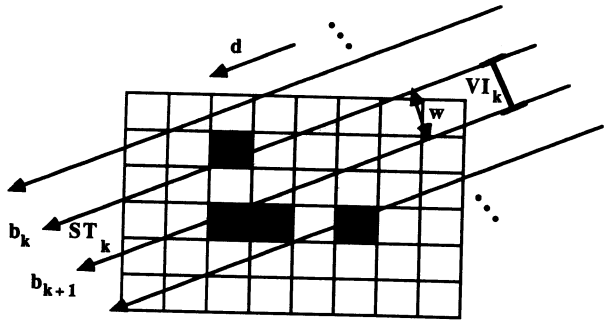


Fig. 7. A strip ST_k of width w.

The visibility in a strip is affected only by the pixels whose region is intersecting that strip. Thus, we can solve the parallel visibility problem independently, and in parallel, for each strip. In geometric terms, for every strip a sweep is performed in direction d which determines for any point of the strip whether it is visible in direction d.

The basic idea for performing such a sweep on a systolic screen is to pass a visibility interval VI_k, which represents the current part of the cross-section of strip ST_k currently visible in direction d, along the processors belonging to ST_k (see Figure 8). Every such processor P_{ij} representing a black pixel (i,j) creates a visibility interval VI_k, equal to the entire cross-section of ST_k minus the part of the cross-section obstructed by $\langle(i,j)\rangle$, and sends VI_k to its successor in ST_k with respect to direction d. Every processor $P_{i'j'}$ belonging to ST_k that receives a visibility interval VI_k updates it (i.e., subtracts the part of the cross-section obstructed by $\langle(i',j')\rangle$), and sends the updated VI_k to its successor in ST_k with respect to direction d (see Figure 8). For every processor P_{ij} belonging to ST_k, the visible part of the cross-section of the strip at $\langle(i,j)\rangle$ is the intersection of all visibility intervals received (or the entire cross-section if no visibility interval was received).

In order to implement the above idea on a systolic screen, a number of issues have to be resolved which are discussed in the following.

One issue is the correct ordering of the processors belonging to ST_k, in which the visibility intervals are passed along the strip. For a correct execution of the algorithm we need a linear ordering of those processors satisfying the following property: if P_{ij} precedes $P_{i'j'}$, then no point of $\langle(i,j)\rangle$ is obstructed by $\langle(i',j')\rangle$.

Such an ordering, which is referred to as O_k, is obtained by projecting the pixel (i,j) represented by every processor P_{ij} belonging to ST_k onto the border s_k of the

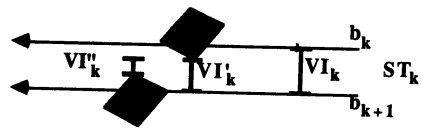


Fig. 8. Update of a visibility interval VI_k.

strip and then sorting these projections in increasing order with respect to direction d.

When a processor P_{ij} receives VI_k from its predecessor in O_k, it has to determine the address of its successor in O_k, and forward the updated visibility interval. In order to obtain the desired time complexity, these two steps must be executed in constant time.

PROPERTY 1. The processor distance between P_{ij} and its successor in O_k is at most three.

From Property 1 it follows that the successor of P_{ij} in O_k can be computed in constant time. Furthermore, VI_k can be forwarded in constant time provided that it can be encoded into a message of constant length.

This requirement leads to another issue, the choice of the width w of every strip. Notice that, if w is chosen to be too large, a visibility interval VI_k being shifted through strip ST_k may consist of several fragments. A large number of fragments will result in too long messages being sent between the processors belonging to a strip.

Notice, on the other hand, that if w is chosen too small, a processor may belong to more than a constant number of strips. Such a situation must not occur since every processor has to execute one process for each strip to which it belongs.

In order to meet the above requirements, we select the following width w:

$$w = \sqrt{2}\cos(45° - \beta).$$

This selection of w has the following properties:

PROPERTY 2. Every visibility interval consists of at most two contiguous parts.

PROPERTY 3. Every processor belongs to at most two strips.

We can now present the algorithm for solving the parallel visibility problem, in the parallel visibility model, for a given direction d on a systolic screen which stores a set of N disjoint images.

We assume that the direction of d of visibility has been entered and broadcast to all processors. Upon termination of the algorithm, every processor P_{ij} will store the set of visible points on the border of $\langle(i,j)\rangle$.

ALGORITHM 3: PARALLEL VISIBILITY

(1) Every processor P_{ij} computes the following initial steps for its local variables w, ST_k, ST_{k+1}, succ_k, succ_{k+1}, VI_k, and VI_{k+1}:
   (a) Calculate the width w.
   (b) Compute the strips ST_k and ST_{k+1} (if it exists) to which P_{ij} belongs.
   (c) Compute the addresses succ_k, succ_{k+1} for the successor of P_{ij} in O_k and O_{k+1}, respectively.

(d) Initialize VI_k and VI_{k+1} to the entire cross-section of ST_k and ST_{k+1}, respectively.

(2) Every processor P_{ij} representing a black pixel (i, j) removes from VI_k and VI_{k+1} the portion obstructed by $\langle (i, j) \rangle$ and sends the updated VI_k and VI_{k+1} to P_{succ_k} and $P_{succ_{k+1}}$, respectively.

(3) Every processor P_{ij} executes the following steps $3\sqrt{M}$ times:
   (a) If a visibility interval VI for strip ST_k is received from the predecessor of P_{ij} in O_h, $h \in \{k, k+1\}$, then VI_h is set to the intersection of VI_h and VI, and the updated VI_h is sent to the successor of P_{ij} in O_h.
   (b) If a received interval VI has a destination other than P_{ij} it is forwarded toward its destination processor.

THEOREM 5. *For a set of digitzed images stored on a systolic screen of size M, Algorithm 3 solves the visibility problem for the parallel visibility model in time* $O(\sqrt{M})$.

PROOF. The correctness of the algorithm follows from the above discussion and the observation that every sweeping process in a strip is completed after at most $3\sqrt{M}$ steps. The time complexity follows from the fact that Steps 1, 2, 3(a) and 3(b) can each be executed in time $O(1)$. □

3.2. Perspective Visibility. We now present a systolic-screen algorithm for solving the visibility problem in the perspective visibility model. Given a point $p = (i_p, j_p) \in \Pi$ emitting rays (radially) in every direction, the task is to determine for every pixel $(i, j) \in \Pi$ the set of all points q on the boundary of $\langle (i, j) \rangle$ that are visible from p (i.e., the line segment from p to q is not intersected by the region $\langle (i', j') \rangle$ of any black pixel $(i', j') \in \Pi$); see Figure 9 for an illustration. Point p is called the *viewpoint*.

For any $q \in \Pi$, let $W(q)$ be the wedge created by the two rays emanating from p and supporting $\langle q \rangle$, i.e., each ray of $W(q)$ is a tangent of $\langle q \rangle$, and $W(q)$ is contained in the closed half-plane defined by each ray (see Figure 10). A *subwedge* of $W(q)$ is a wedge created by two rays emanating from p which is contained in $W(q)$.

The *pixel obstruction wedge* $POW(q)$ is the portion of the plane obstructed by

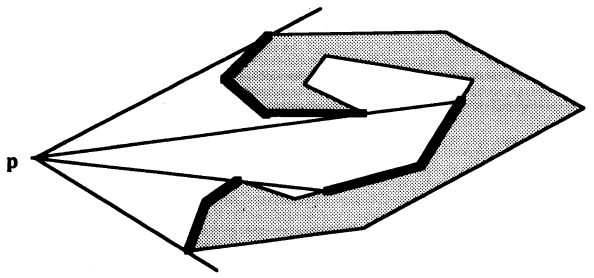


Fig. 9. Perspective visibility from a point p.

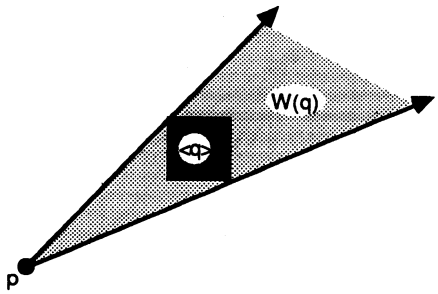


Fig. 10. The wedge W(q) of a pixel q.

a pixel $q \in \Pi$; if q is black, then POW(q) is the portion of W(q) obstructed by $\langle q \rangle$, if q is white, then POW(q) is the empty set.

The algorithm for perspective visibility uses a layered systolic-screen sweep, i.e., a systolic-screen sweep with $f(t) = \{(i,j) | \max\{|i_p - i|, |j_p - j|\} = t - 1\}$, see Section 1. During this sweep, every processor representing a pixel q accumulates the pixel obstruction wedges of all pixels q' for which $\langle q' \rangle$ obstructs at least one point of $\langle q \rangle$; the portion of this set that is contained in W(q) describes exactly the portion of the boundary of $\langle q \rangle$ which is not visible from p and is referred to as its *invisible set* INVIS(q). The *obstruction set* OS(q) of a pixel $q \in \Pi$ is the union of INVIS(q) and POW(q).

We assume that the coordinates of the viewpoint p have been broadcast to all processors. The following is the basic structure of the algorithm for solving the perspective visibility problem. Upon termination of the algorithm, every processor representing a pixel q stores its invisible set INVIS(q).

ALGORITHM 4: PERSPECTIVE VISIBILITY.

(1) Every processor P_{ij} representing a pixel q computes the following initial steps on its local variables INVIS and OS which represent the invisible set INVIS(q) and the obstruction set OS(q), respectively:

$$\text{INVIS} \leftarrow \varnothing,$$
$$\text{OS} \leftarrow \text{POW}(q).$$

(2) The following steps are iterated for $t = 1, \ldots, \sqrt{M} - 1$:
   (a) Every processor representing a pixel in $f(t)$ sends a description of its obstruction set OS to all those processors P_{ij} with $(i,j) \in f(t+1)$ for which $\langle (i,j) \rangle$ intersects OS.
   (b) Every processor representing a pixel $q \in f(t+1)$ sets

$$\text{INVIS} \leftarrow (\text{W}(q) \cap \Sigma),$$
$$\text{OS} \leftarrow \text{INVIS} \cup \text{POW}(q),$$

where Σ is the union of all obstruction sets received.

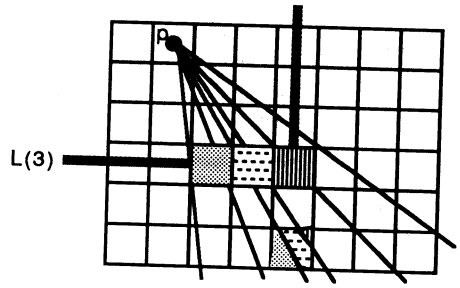


Fig. 11. A maximum number of obstructions sets of a previous layer that intersect a pixel.

In order to implement this algorithm efficiently, it must be ensured that all messages describing obstruction sets are of constant length, and that at each stage of the algorithm every processor representing a pixel $q \in f(t+1)$ receives only a constant number of such messages from the processors representing a pixel $f(t)$.

PROPERTY 4. For every $(i,j) \in f(t)$, there are at most three pixels $q \in f(t-v)$, $1 \leq v < t$, whose obstruction sets intersect $\langle (i,j) \rangle$. (See Figure 11.)

PROPERTY 5. Every invisible set INVIS(q) and obstruction set OS(q) consists of at most two subwedges of W(q).

We also observe that the L_1 (Manhattan) distance between a pixel $q \in f(t)$ and every $p \in f(t+1)$ for which OS(q) intersects $\langle p \rangle$ is at most 2. Therefore, in order to implement Step 2(a) of Algorithm 4, it is sufficient that every processor representing a pixel $q \in f(t)$ first sends its obstruction set only to its direct neighbor(s) representing a pixel $p \in f(t+1)$ (*growth step*), and then all those processors rotate the received obstruction sets by two positions in clockwise and counterclockwise direction (*exchange step*); see Figure 12.

Summarizing, we obtain

THEOREM 6. *For any set of digitized images stored on a systolic screen of size M, Algorithm 4 solves the visibility problem for the perspective model in time $O(\sqrt{M})$.*

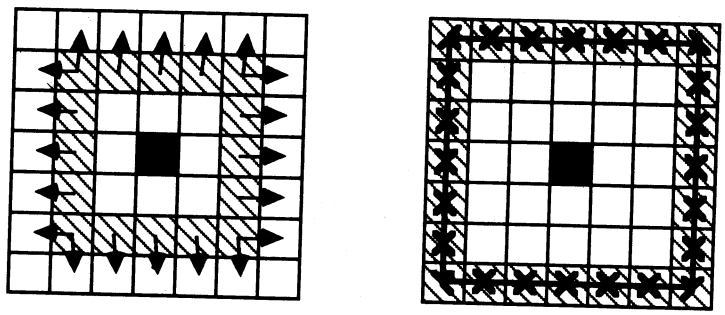


Fig. 12. The (a) growth step and (b) exchange step in the perspective visibility algorithm.

PROOF. The correctness and time complexity of Algorithm 4 follows from the above observations and the fact that the layered systolic-screen sweep guarantees that every processor representing a pixel q accumulates, in its register INVIS, the pixel obstruction wedges of all pixels q' for which $\langle q' \rangle$ obstructs at least one point of $\langle q \rangle$. □

3.3. *Applications of Digitized Visibility*

3.3.1. Rectangular Visibility. Let S be a planar point set of cardinality n. Two points $p, q \in S$ are said to be *rectangularly visible* if the orthogonal rectangle with the two points as diagonally opposite vertices contains no other point of S. $O(n \log n + k)$-time (sequential) algorithms to find all pairs of points that are rectangularly visible (where k is the number of pairs to be reported) have been presented by Güting *et al.* [GNO], as well as Munro *et al.* [MOW]. On a systolic screen, this problem can be easily solved in $O(\sqrt{M})$ time by using a sweep technique similar to that described in Section 3.1 performed in a direction parallel to the screen axes.

3.3.2. Separability. The technique of partitioning the systolic screen into a set of parallel strips developed in Section 3.1 can be employed to solve related problems in motion planning. One motion-planning problem is that of separability of objects (see, e.g., [DS1], [NS2], and [T]). Given a set of objects $\{O_1, \ldots, O_N\}$ and a direction d of translation, does there exist a translation ordering $(O_{\pi(1)}, \ldots, O_{\pi(N)})$ among the objects, where π is a permutation of the index set $\{1, \ldots, N\}$, such that, for all $i = 1, \ldots, N-1$, $O_{\pi(i)}$ can be separated, i.e., translated by an arbitrary amount in d without colliding with any of the objects $O_{\pi(i+1)}, \ldots, O_{\pi(N-1)}, O_{\pi(N)}$ not yet translated (Problem I)? This problem has been studied in robotics and computer graphics. A subproblem arising in this context is to determine for every object O_i whether it can be separated from the object set $\{O_1, \ldots, O_N\} - O_i$ or not (Problem II). We sketch the ideas of the algorithms for solving these problems on the systolic screen.

The idea to solve Problem I is to construct the visibility hull of each object with respect to direction d; the visibility hull of a planar object O_i with respect to direction d is the union of all line segments that are parallel to direction d and whose endpoints are both contained in O_i. It has been shown in [T] that there exists a translation ordering for a set of objects with respect to direction d if and only if the visibility hulls (with respect to direction d) of no two objects intersect. Thus, it remains to be shown how to detect on a systolic screen whether any two visibility hulls of N objects $O_1, \ldots, O_N$ intersect. This can be accomplished by essentially two executions of a sweep similar to the one described in Section 3.1, one in direction d and one in the opposite direction. During these sweeps, every obstruction interval initiated by a black pixel of object O_i considers all pixels except those of O_i as white pixels. In the first sweep, every black pixel of O_i determines whether it is obstructed by another black pixel of O_i. In the second sweep, only the obstructed pixels of O_i initiate an obstruction interval; during this sweep, the cross-section of all encountered black pixels of O_i are subtracted from (rather than added to) these obstruction intervals. All black pixels of O_i and those

"obstructed" by O_i in the second sweep are in the visibility hull of O_i and are colored as such. The objects are separable in direction d, if no two objects attempt to color the same pixel as part of its visibility hull (collision). Note that the visibility hulls may not be computed correctly if collisions occur.

In order to solve Problem II, we observe that an object O_i is separable from the object set $\{O_1, \ldots, O_N\} - O_i$ if and only if none of the obstruction intervals originated by a black pixel of O_i encountered a collision and none of its black pixels was the location of a collision. Obviously, the information about all collisions can also be routed back (by another sweep in direction d) to the black pixels at which the respective obstruction intervals orginated. Then, it remains to be determined for each object O_i, whether any of its pixels found a collision. This can be accomplished in time $O(\sqrt{M})$ by using the connected component algorithm described in [NS1] (where each pixel is labeled with "0" if it found a collision and "1" otherwise) and routing the minimum label of the pixels in each object to all its pixels. Thus, all objects O_i entirely labeled with "1" are separable from $\{O_1, \ldots, O_N\} - O_i$.

3.3.3. Pseudo-Star-Shapedness. The perspective visibility algorithm of Section 3.2 can be used to solve a variant of visibility in which a point p in a polygon P is k-visible from another point q if the line segment pq does not intersect the boundary of P more than k times (for fixed k). A polygon P is called pseudo-star-shaped from p if it is k-visible from p for $k = 1$ (see [DLS]). Pseudo-star-shaped polygons are more general than convex, star-shaped or monotone polygons and exact characterizations of these properties via pseudo-star-shapedness have been given [DLS]. The above perspective visibility algorithm (for a view point p) can be generalized to decide whether a polygon is pseudo-star-shaped from a point p by considering only boundary pixels of a polygon (represented as an image on the systolic screen) as black pixels and introducing for each pixel an "intermediate obstruction set" to store wedges obstructed by only one boundary pixel. Thus, on a systolic screen of size M, pseudo-star-shapedness can be detected, and in general the k-visibility problem (for fixed k) can be solved, in time $O(\sqrt{M})$.

4. Digitized Voronoi Diagrams. In this section we present an $O(\sqrt{M})$-time solution to the problem of computing the (digitized) Voronoi diagram of a set of n disjoint objects taken from a large class of object types which includes points, line segments, circles, ellipses, and polygons of constant size. The algorithm can be used to compute the (digitized) Voronoi diagram for a large class of distance functions which include, e.g., all L_p metrics. The parallel computation of digitized Voronoi diagrams for point sets in the Manhattan (L_1) and Euclidean (L_2) metric has also been studied in [S3].

Since Voronoi diagrams are used in many geometric applications, our result has numerous consequences for the design of efficient image-processing algorithms on a systolic screen.

Consider a set $S = \{s_1, \ldots, s_n\}$ of n geometric objects in $\mathbb{R}^2$ (i.e., objects which are connected point sets), e.g., points, line segments, polygons, circles, ellipses. Let

$d: \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^+$ be a distance function. The well-known Voronoi *diagram* V(*S*) (discussed, e.g., in [SH]) partitions $\mathbb{R}^2$ into *n Voronoi regions*:

$$V(s_i) := \{x \in \mathbb{R}^2 | d(x, s_i) \leq d(x, s_j) \text{ for all } j \neq i\}.$$

See Figure 13 for an illustration. Every Voronoi region $V(s_i)$ consists of two disjoint parts, the *interior*

$$IV(s_i) := \{x \in \mathbb{R}^2 | d(x, s_i) < d(x, s_j) \text{ for all } j \neq i\}$$

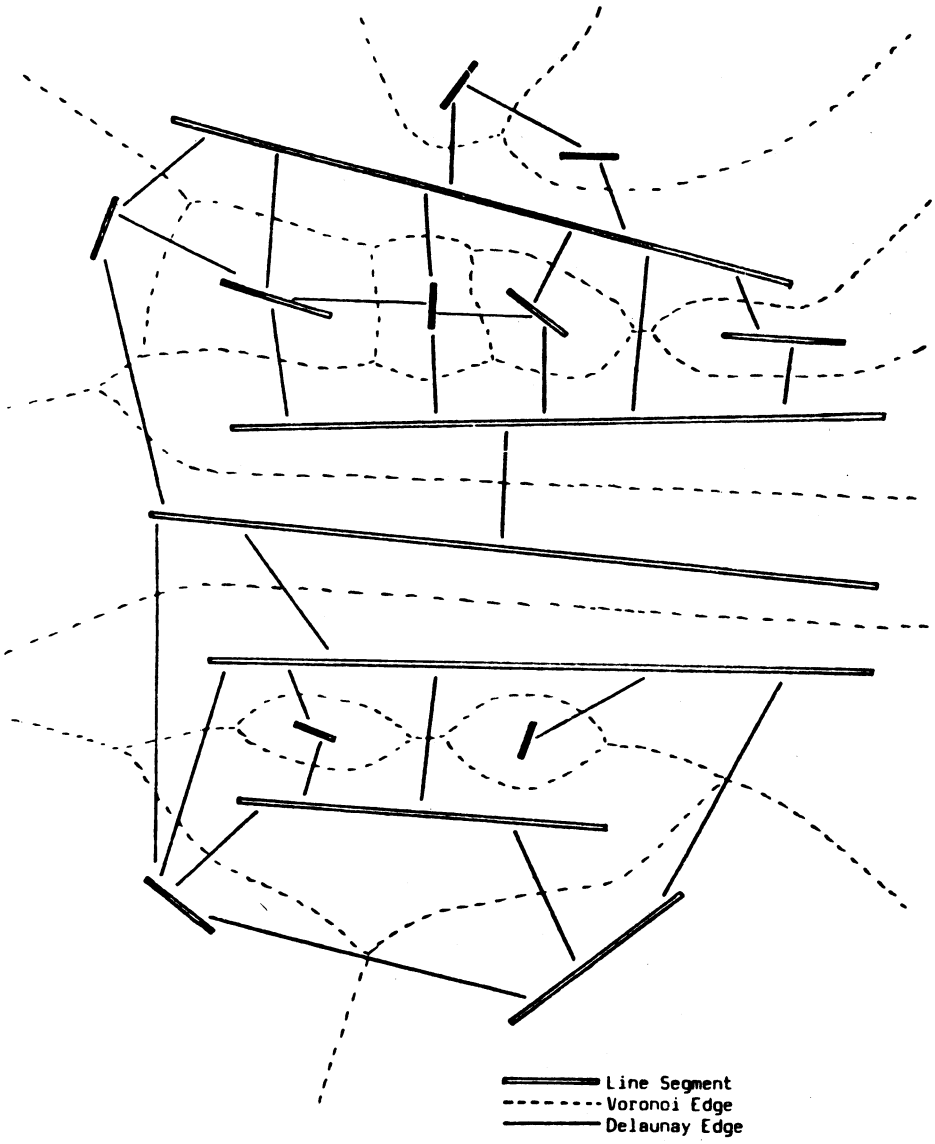


Fig. 13. Voronoi diagram for a set of line segments.

and the *border*

$$\mathrm{BV}(s_i) := \mathrm{V}(s_i) - \mathrm{IV}(s_i).$$

$\mathrm{BV}(S) := \bigcup_{1 \le i \le n} \mathrm{BV}(s_i)$, the union of all borders, is usually referred to as the set of Voronoi *points* of $\mathrm{V}(S)$.

We show how the definition of a Voronoi diagram in $\mathbb{R}^2$ can be translated to the digitized environment (see also [S3]). The *digitized Voronoi diagram* $\mathrm{V_d}(S)$ can be defined as follows:

Consider a set $S = \{s_1, \ldots, s_n\}$ of n geometric objects $s_i \subseteq \langle \Pi \rangle$ such that their image representations in Π do not intersect (i.e., $\langle s_i \rangle \cap \langle s_j \rangle = \varnothing$ for $i \neq j$), and let d: $\mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^+$ be a distance function.

As described above, the standard Voronoi diagram $\mathrm{V}(S)$ induces a Voronoi region $\mathrm{V}(s_i)$ for each object which consists of an interior $\mathrm{IV}(s_i)$ and a border $\mathrm{BV}(s_i)$.

The *digitized Voronoi diagram* $\mathrm{V_d}(S)$ partitions Π into n *digitized Voronoi regions* $\mathrm{V_d}(s_i)$, one for each object s_i. Each digitized Voronoi region consists of an *interior* $\mathrm{IV_d}(s_i)$ and a *border* $\mathrm{BV_d}(s_i)$ defined as follows:

- $\mathrm{BV_d}(s_i) := \mathrm{Im}(\mathrm{BV}(s_i))$.
- $\mathrm{IV_d}(s_i) := \mathrm{Im}(\mathrm{V}(s_i)) - \mathrm{BV_d}(s_i)$.

That is, the border of a digitized Voronoi region is the image of the border of the respective standard Voronoi region; the interior of a digitized Voronoi region consists of the remaining pixels in the image of the respective standard Voronoi region (see Figure 14). Consequently, the set $\mathrm{BV_d}(S)$ of all *Voronoi pixels* of the digitized Voronoi diagram is defined as $\mathrm{BV_d}(S) := \bigcup_{1 \le i \le n} \mathrm{B_d}(s_i)$; i.e., the Voronoi pixels of $\mathrm{V_d}(S)$ are obtained by computing the image of the Voronoi points of $\mathrm{V}(S)$.

Note that Voronoi points which do not intersect $\langle \Pi \rangle$ are not represented in the digitized Voronoi diagram $\mathrm{V}(S)$ and that all Voronoi points which are contained in a cell $\langle p \rangle$, $p \in \Pi$, are represented by one Voronoi pixel only (see also [M2]).

In Sections 4.1 and 4.2 we describe how to compute digitized Voronoi diagrams on a systolic screen. To simplify the exposition we first consider the basic case of a set of points and Euclidean metric, and we then generalize our result to more general sets of objects and distance functions.

4.1. Computing Digitized Voronoi Diagrams for Point Sets and Euclidean Metric. Let $S = \{s_1, \ldots, s_n\}$ be a set of n points in $\langle \Pi \rangle$ ($\mathrm{Im}(s_i) \cap \mathrm{Im}(s_j) = \varnothing$ for $i \neq j$), and let d be the Euclidean metric. We present an $O(\sqrt{M})$-time algorithm for computing the digitized Voronoi diagram $\mathrm{V_d}(S)$ on a systolic screen of size M. (See Figure 15). The algorithm assumes as input that $\mathrm{Im}(s_1), \ldots, \mathrm{Im}(s_n)$ are represented on a systolic screen as described in Section 1. The digitized Voronoi diagram of S is reported by the systolic screen as follows:

Every processor P_{ij} has a *Voronoi register*, V-Reg(i, j), and upon termination of the algorithm their values are

$$\text{V-Reg}(i,j) = \begin{cases} k & \text{if } (i,j) \in \mathrm{IV_d}(s_k) \quad (1 \le k \le n), \\ * & \text{if } (i,j) \in \mathrm{BV_d}(S). \end{cases}$$

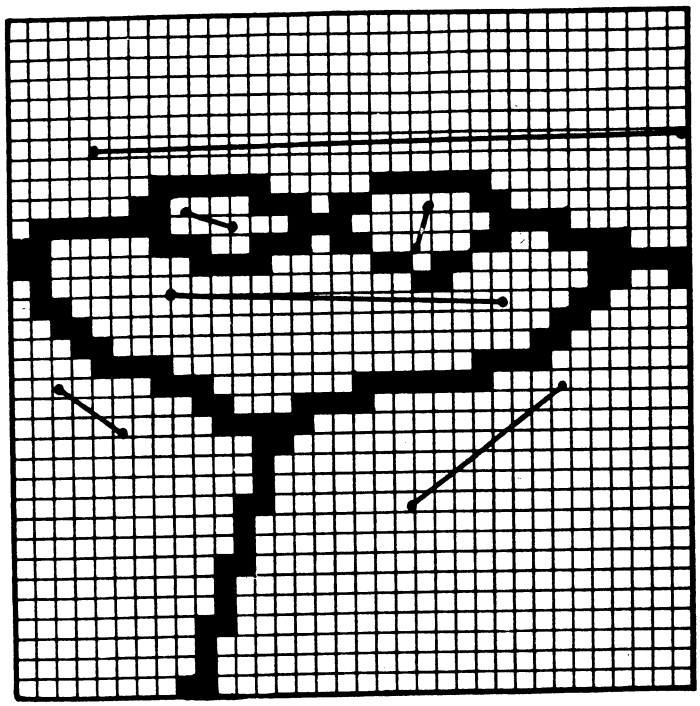


Fig. 14. Digitized Voronoi diagram for the set of line segments of Figure 13. (The black pixels represent the Voronoi pixels).

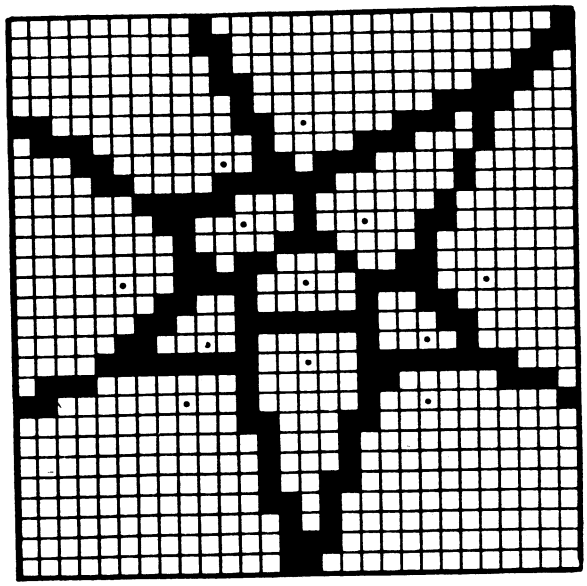


Fig. 15. Digitized Voronoi diagram for a set of points.

The basic idea of the algorithm is to "grow" circles emanating from the objects (recall the layered sweep introduced in Section 1); a similar geometric idea for sequential Voronoi diagram construction can also be found in [CD]. The algorithm has $O(\sqrt{M})$ rounds, in each round the radii of all circles are incremented by a distance μ, where μ is chosen such that, for all $w \in S$, $t \in \mathbb{N}$,

$$\text{Bord}(\text{disc}(w, t\mu)) = \text{disc}(w, t\mu) - \text{disc}(w, (t-1)\mu);$$

for point sets, and Euclidean Metric, we chose $\mu = 1$. The algorithm determines the Voronoi pixels by examining the areas of "colliding" circles.

ALGORITHM 5: COMPUTING THE DIGITIZED VORONOI DIAGRAM.

(1) All P_{ij} initialize their V-Register: V-Reg$(i, j) \leftarrow$ C-Reg(i, j).
(2) For $t := 1$ to $\sqrt{M}/\mu$ do.
   (a) All P_{ij} with V-Reg$(i, j) = k > 0$ send a message "k" to all $P_{i'j'}$ within a constant processor distance λ for which V-Reg$(i', j') = 0$ and $\langle (i', j') \rangle \cap (\text{disc}(s_k, t\mu) - \text{disc}(s_k, (t-1)\mu)) \neq \varnothing$.
   (b) All P_{ij} with V-Reg$(i, j) = 0$ which receive only messages "k" set V-Reg$(i, j) \leftarrow k$.
   (c) All P_{ij} with V-Reg$(i, j) = 0$ which receive at least two different messages "k_1" and "k_2" set V-Reg$(i, j) \leftarrow$ *.
   (d) All P_{ij} with V-Reg$(i, j) = k_1$ which receive a message "k_2" such that

$$(\text{disc}(s_{k_1}, t\mu) - \text{disc}(s_{k_1}, (t-1)\mu)) \cap \langle (i, j) \rangle \neq \varnothing$$

   and

$$(\text{disc}(s_{k_2}, t\mu) - \text{disc}(s_{k_2}, (t-1)\mu)) \cap \langle (i, j) \rangle \neq \varnothing$$

   set V-Reg$(i, j) \leftarrow$ *.

THEOREM 7. *Algorithm 5 computes, on a systolic screen of size M, the digitized Voronoi diagram of a set S of n points in $\langle \Pi \rangle$, for the Euclidean metric, in time $O(\sqrt{M})$.*

PROOF. We refer to the loop index t as time. The minimum distance between a pixel $(i', j') \in \text{disc}(s_k, t+1)$ and some pixel $(i, j) \in \text{disc}(s_k, t)$ is at most a constant, say λ. Thus, to send a message "k" from all pixels in $\text{disc}(s_k, t)$ to the pixels in $\text{disc}(s_k, t+1) - \text{disc}(s_k, t)$, it suffices that each P_{ij} with $\langle (i, j) \rangle \cap \text{disk}(s_k, t) \neq \varnothing$ sends a message "k" to all processors within distance λ. Hence, at time t, a processor P_{ij} with $\langle (i, j) \rangle \cap \text{disc}(s_k, t) \neq \varnothing$ either receives a message "k" or has received some other message earlier.

Consider a processor P_{ij} for which all points in $\langle (i, j) \rangle$ are closer to s_k than to any other $s_{k'}$. Upon termination of the algorithm the value of the processor's V-register must be k. For this processor P_{ij}, there exists some earliest time $t^* \in \{1, \ldots, \sqrt{M}\}$ such that $\langle (i, j) \rangle \subseteq \text{disc}(s_k, t^*)$ but $\langle (i, j) \rangle \cap \text{disc}(s_{k'}, t^*) = \varnothing$ for

all $k' \neq k$. Hence, prior to time t^*, processors P_{ij} did not receive any messages and, at time t^*, P_{ij} receives messages, all of which have value "k." Thus, at time t^*, in Step 2(b), P_{ij} correctly sets its Voronoi register V-Reg(i,j) to k and subsequently this value is not altered.

Consider a processor P_{ij} for which at least one point $x \in \langle(i,j)\rangle$ is equidistant to two objects s_k and $s_{k'}$. Upon termination of the algorithm the value of the processors V-register must be *. For such a P_{ij}, there exists some earliest time $t^* \in \{1, \ldots, \sqrt{M}\}$ at which it receives a message "k_1" and, either at this time or one time step later, it receives a message "k_2." In either case, V-register(i,j) will eventually be set to *. In the former case, this is done at time t^* as per Step 2(b). In the latter case, V-Reg(i,j) is first set to k_1 and subsequently, at time $t^* + 1$, this value changes to * as per Step 2(d). Since a register value * can never be altered, V-Reg(i,j) will, upon termination of the algorithm, correctly contain the value *. Thus, the correctness of Algorithm 5 follows.

Since the execution of Step 1 and parts (a)–(d) of Step 3 each take time $O(1)$, the running time of Algorithm 5 is $O(\sqrt{M})$ □

4.2. Computing Digitized Voronoi Diagrams for Sets of Objects and Convex Distance Functions. After having solved the problem of constructing the digitized Voronoi diagram for point sets using the Euclidean metric, we now generalize our result to other classes of objects and to convex distance functions. Algorithm 5 can be modified to solve these more general problems.

THEOREM 8. *The digitized Voronoi diagram of a set* $S = \{w_1, \ldots, w_n\}$ *of n objects* $w_i \subseteq \langle\Pi\rangle$ *for any convex distance function can be computed on a systolic screen of size M in time* $O(\sqrt{M})$ *provided that the following conditions hold:*

(i) *For any two objects* $w, w' \in S$, $\mathrm{Im}(w) \cap \mathrm{Im}(w') = \varnothing$.

(ii) *There exists a constant* μ *such that, for all* $w \in S, t \in \mathbb{N}$,

$$\mathrm{Bord}(\mathrm{disc}(w, t\mu)) = \mathrm{disc}(w, t\mu) - \mathrm{disc}(w, (t-1)\mu).$$

(iii) *For any object* $w \in S$ *there exists an* $O(1)$ *space description such that from this description it can be decided for every* $p \in \Pi$ *and* $t \in \{1, \ldots, \sqrt{M}\}$ *in* $O(1)$ *time whether* $\langle p\rangle \cap (\mathrm{disc}(w, t\mu) - \mathrm{disc}(w, (t-1)\mu)) = \varnothing$.

(iv) *There exists a constant* λ *such that, for every* $w \in S$, $t \in \{1, \ldots, \sqrt{M}\}$, *and* $p \in \Pi$ *with* $\langle p\rangle \cap \mathrm{disc}(w, t\mu) \neq \varnothing$,

$$\min\{d_1(p, p') \mid p' \in \Pi, \langle p'\rangle \cap \mathrm{disc}(w, (t-1)\mu) \neq \varnothing\} < \lambda,$$

where d_1 *refers to the* L_1 metric (i.e., the processor distance).

PROOF. Algorithm 5 needs only minor modifications to handle the generalized case: disc(s, r) is generalized to the given type of object and the given distance function. Furthermore, the values of λ and μ are adjusted to the particular case.

While, similar to Section 4.1, condition (i) ensures that the images of two objects do not intersect, condition (ii) ensures that a proper constant μ as required by Algorithm 5 exists. Two more conditions are required to show that Algorithm 5 performs correctly and terminates after $O(\sqrt{M})$ steps.

In Steps 2(a) and 2(d) of the algorithm, intersection tests between a disc and a rectangle are performed. While such a test can clearly be executed in $O(1)$ time for point objects using the Euclidean metric, this is no longer the case for arbitrary objects using arbitrary distance functions. In fact, the processor performing this test needs not only the number of objects, but also information about the objects in order to perform the intersection test. Therefore, an $O(1)$ space description of this information must be available, and the test must be executable in $O(1)$ time, i.e., condition (iii) must hold (and this also suffices).

For Step 2(a) of the algorithm, the processor distance λ within which each P_{ij} scans all its neighbors and sends them a message k is modified according to the type of object and the given metric. Condition (iv) ensures that λ is a constant, which may not be the case in general, but is sufficient for the algorithm to work correctly.

Under these conditions, the correctness of Algorithm 5 so generalized follows in the same way as in the proof of Theorem 7, and its asymptotic running time does not change. Thus, the correctness of Theorem 8 is established. □

The class of objects for which the conditions in Theorem 8 hold is fairly general. It contains, e.g., all "simple" geometric objects (i.e., those ones that have an $O(1)$ description) and most of the standard distance function including all L_p metrics.

COROLLARY 2. *On a systolic screen of size M, the digitized Voronoi diagram of a set of points, line segments, circles, ellipses, and polygons of constant size can be computed, for any L_p metric, in time $O(\sqrt{M})$ provided that their images do not intersect.*

4.3. Applications to Optical Clustering. One of the most useful and most thoroughly studied methods in image analysis is clustering, see, e.g., [DJ]. In [D1] a clustering method called "optical clustering" has been presented, which groups objects in a manner similar to human perception. For a given clustering radius r, the clusters are the connected component of the graph connecting two objects if and only if there exists a circle with radius r intersecting both of them.

The sequential method presented in [D1] which solves this problem for n points (and L_p metric) or line segments (and Euclidean Metric) is based on the Voronoi diagram for these objects.

The same type of clustering can be obtained on a systolic screen of size M in time $O(\sqrt{M})$ for any set of digitized objects and distance function for which the conditions of Theorem 8 hold. First, the digitized Voronoi diagram is computed as described in Sections 4.1 and 4.2, but with the following addition:

(i) Each processor P_{ij} has an additional register D-Reg(i, j).

(ii) For each P_{ij}, when V-Reg(i, j) is set to k (or to *, respectively), then D-Reg(i, j) is set to the distance of (i, j) to the corresponding object (the two corresponding objects, respectively).

For any given clustering radius r, the radius is simply broadcast to all processors, and all pixels (i, j) with D-Reg$(i, j) \leq r$ are considered as black pixels of a new image $I_{\text{clus}}(r)$. The clustering is then obtained by applying the connected component algorithm described in [NS1] to $I_{\text{clus}}(r)$.

References

[ACGOY] A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing, C. Yap, Parallel computational geometry, *Algorithmica*, **3**(3), 1988, 293–327.

[AK] M. J. Atallah, S. R. Kosaraju, Graph problems on a mesh-connected processor array, *Journal of the ACM*, **31**(3), 1984, 649–667.

[C] B. M. Chazelle, Optimal algorithms for computing depths and layers, in *Proc. 21st Allerton Conf. on Communication, Control and Computing*, Urbana-Champaign, Ill., 1983, 427–436.

[CGL] B. Chazelle, L. J. Guibas, D. T. Lee, The power of geometric duality, in *Proc 24th IEEE Symp. on Foundations of Computor Science*, Tucson, Ariz., 1983.

[CD] L. P. Chew, R. L. Drysdale, Voronoi diagrams based on convex distance functions, in *Proc. Symp. on Computational Geometry*, Baltimore, 1985, 235–244.

[DLS] J. A. Dean, A. Lingas, J.-R. Sack, Recognizing Polygons: or How To Eavesdrop, *The Visual Computer*, **3**(6), 1988, 344–355.

[D1] F. Dehne, Optical clustering, *The Visual Computer*, **21**(1), 1986, 39–43.

[D2] F. Dehne, $O(n1/2)$ algorithms for the maximal elements and ECDF searching problem on a mesh-conneced parallel computer, *Information Processing Letters*, **22**(6), 1986, 303–306.

[D3] F. Dehne, Solving visibility and separability problems on a mesh-of-processors, *The Visual Computer*, **4**(6), 1988, 356–370.

[D4] F. Dehne, Computing the largest empty rectangle on one- and two-dimensional processor arrays, *Journal of Parallel and Distributed Computing*, **9**(1), 1990, 63–68.

[DHS] F. Dehne, A. Hassenklover, J.-R. Sack, Computing the configuration space for a robot on a mesh-of-processors, *Parallel Computing*, **12**, 1989, 221–231.

[DHSS] F. Dehne, A. Hassenklover, J.-R. Sack, N. Santoro, Parallel visibility on a mesh-connected parallel computer, in *Proc. Int. Conf. on Parallel Processing and Applications*, L'Aquila, Italy, 1987, 173-180.

[DJ] R. Dubes, A. K. Jain, Clustering methodologies in exploratory data analysis, in *Advances in Computers*, Vol. 19 (M. C. Yovits, ed.), 1980, 113–228.

[DP] F. Dehne, Q. T. Pham, Visibility algorithms for binary images on the hypercube and the perfect-shuffle computer, in *Proc. IFIP WG 10.3 Conf. on Parallel processing*, Pisa, 1988, North Holland, Amsterdam, 117–124.

[DS1] F. Dehne, J.-R. Sack, Translation separability of sets of polygons, *The Visual Computer*, **3**(4), 1987, 227–235.

[DS2] F. Dehne, J.-R. Sack, Parallel computational geometry: a survey, invited paper, in *Proc. Parcella '88*, Berlin, Lecture Notes in Computer Science, Vol. 342, Springer-Verlag, Berlin, 1988, 73–89.

[DSS] F. Dehne, J.-R. Sack, N. Santoro, Computing on a systolic screen: hulls, contours and applications, in *Proc. Conf. on Parallel Architectures and Languages*, Eindhoven, The Netherland, 1987, Lecture Notes in Computer Science, Vol. 258, Springer-Verlag, Berlin, 121–133.

[D5] M. J. Duff *et al.*, A cellular logic array for image processing, *Pattern Recognition*, **5**, 1973, 229–247.

[GNO] R. H. Güting, O. Nurmi, T. Ottmann, The direct dominance problem, in *Proc. ACM Symp. on Computational Geometry*, Baltimore, 1985, 81–88.

[H1] D. Hillis, *The Connection Machine*, MIT Press, Cambridge, Mass., 1985.

[H2] D. S. Hirschberg, Algorithms for the longest common subsequence problem, *Journal of the ACM*, **24**(4), 1977, 664–675.

[H3] P. J. Huber, Robust statistics: a review, *Annals of Mathematical Statistics*, **43**(3), 1972, 1041–1067.

[HS] J. W. Hunt, T. G. Szymanski, A fast algorithm for computing longest common subsequences, *Communications of the ACM*, **20**, 1977, 350–353.

[K] C. E. Kim, Digital Disks, Report CS-82-104, Computer Science Department, Washington State University, Dec. 1982.

[LP] D. T. Lee, F. P. Preparata, Computational geometry—a survey, *IEEE Transactions on Computers*, **33**(12), 1984, 1072–1101.

[M1] B. H. McCormick, The Illinois pattern recognition computer—ILLIAC III, *IEEE Transactions on Electronic Computers*, **12**, 1963, 791–813.

[M2] G. U. Montanari, On limit properties of digitzation schemes, *Journal of the ACM*, **17**, 1970, 348–360.

[MF] M. F. Montuno, A. Fournier, Finding the x–y Convex Hull of a Set of x–y Polygons, Technical Report CSRG-148, University of Toronto, Toronto, Nov. 1982.

[MS1] R. Miller, Q. F. Stout, Computational geometry on a mesh-connected computer, in *Proc. Int. Conf. on Parallel Processing*, 1984, 66–73.

[MS2] R. Miller, Q. F. Stout, Geometric algorithms for digitised pictures on a mesh-connected computer, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **7**(2), 1985, 216–228.

[MOW] J. I. Munro, M. H. Overmars, D. Wood, Variations on visibility, in *Proc. ACM Symp. on Computational Geometry*, Waterloo, Canada, 1987, 291–299.

[NKY] N. Nakatsu, Y. Kambayashi, S. Yajima, A longest common subsequence algorithm suitable for similar text strings, *Acta Informatica*, **18**, 1982, 171–179.

[NS1] D. Nassimi, S. Sahni, Finding connected components and connected ones on a mesh-connected parallel computer, *SIAM Journal on Computing*, **9**(4),,1980, 744–757.

[NS2] O. Nurmi, J.-R. Sack, Separating a polyhedron by one translation from a set of obstacles, in *Proc. Workshop on Graph-Theoretic Concepts in Computer Science*, Amsterdam, The Netherlands, June 1988, (J. van Leeuwen, ed.), Lecture Notes in Computer Science, Vol. 344, Springer-Verlag, Berlin, 202–212.

[OvL] M. H. Overmars, J. van Leeuwen, Maintenance of configurations in the plane, *Journal of Computer and System Sciences*, **23**, 1981, 166–204.

[PS] F. P. Preparata, M. I. Shamos, *Computational Geometry, An Introduction*, Springer-Verlag, Berlin, 1985.

[R1] A. P. Reeves, Survey parallel computer architectures for image processing, *Computer Vision, Graphics and Image Processing*, **25**, 1984, 68–88.

[RT] Y. Robert, M. Tchuente, A systolic array for the longest common subsequence problem, *Information Processing Letters*, **21**, 1985, 191–198.

[R2] A. Rosenfeld, Digital topology, *The American Mathematical Monthly*, **86**, 1979, 621–630.

[R3] A. Rosenfeld, Parallel image processing using cellular arrays, *IEEE Transactions on Computers*, **16**(1), 1983, 15–20.

[S1] J.-R. Sack, A simple hidden-line algorithm for rectilinear-polygons, in *Proc. 21st Allerton Conf. on Communication, Control and Computing*, Urbana-Champaign, Ill., Oct. 1983, 437–446.

[S2] J.-R. Sack, Rectilinear Computational Geometry, Ph.D. thesis, McGill University, Montréal, 1984.

[S3] O. Schwarzkopf, Parallel computation of discrete Voronoi diagrams, in *Proc. Symp. on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science, Vol. 349, Springer-Verlag, Berlin, 1989, 193–204.

[S4] M. I. Shamos, Computational Geometry, Ph.D. thesis, Department of Computer Science, Yale University, 1978.

[SH] M. I. Shamos, D. Hoey, Closest point problems, in *Proc. 7th Ann. IEEE Symp. on Foundations of Computer Science*, 1975.

[SM] Q. F. Stout, R. Miller, Mesh-connected computer algorithms for determining geometric properties of figures, in *Proc. 7th Int. Conf. on Pattern Recognition*, Montréal, 1984, 475–477.

[TK] C. D. Thompson, H. T. Kung, Sorting on a mesh-connected parallel computer, *Communications of the ACM*, **20**(4), 1977, 263–271.

[T] G. T. Toussaint, Movable separability of sets, in *Computational Geometry*, (G. T. Toussaint, ed.), North-Holland, Amsterdam, 1985, 335–376.

[U] S. H. Unger, A computer oriented towards spatial problems, *Proceedings of the IRE*, **46**, 1958, 1744–1750.

[W] D. Wood, An isothetic view on computational geometry, in *Computational Geometry*, (G. T. Toussaint, ed.), North-Holland, Amsterdam, 1985, 429–459.