

# Computing Convexity Properties of Images on a Pyramid Computer<sup>1</sup>

Russ Miller<sup>2</sup> and Quentin F. Stout<sup>3</sup>

**Abstract.** We present efficient parallel algorithms for using a pyramid computer to determine convexity properties of digitized black/white pictures and labeled figures. Algorithms are presented for deciding convexity, identifying extreme points of convex hulls, and using extreme points in a variety of fashions. For a pyramid computer with a base of  $n$  simple processing elements arranged in an  $n^{1/2} \times n^{1/2}$  square, the running times of the algorithms range from  $\Theta(\log n)$  to find the extreme points of a convex figure in a digitized picture, to  $\Theta(n^{1/6})$  to find the diameter of a labeled figure,  $\Theta(n^{1/4} \log n)$  to find the extreme points of every figure in a digitized picture, to  $\Theta(n^{1/2})$  to find the extreme points of every labeled set of processing elements. Our results show that the pyramid computer can be used to obtain efficient solutions to nontrivial problems in image analysis. We also show the sensitivity of efficient pyramid-computer algorithms to the rate at which essential data can be compressed. Finally, we show that a wide variety of techniques are needed to make full and efficient use of the pyramid architecture.

**Key Words.** Pyramid computer, Convexity, Digitized pictures, Digital geometry, Image processing, Parallel computing, Parallel algorithms.

**1. Introduction.** Pyramid-like parallel computers have long been proposed for performing high-speed low-level image processing [D1]–[D3], [R3], [St1], [St2], [TaK], [Ta1], [Ta2], [Uh1], [Uh2] and several are under construction [BV], [CFLS], [CM], [Fr], [Sc], [SHBV], [Ta2]. The pyramid has a simple geometry that adapts naturally to many types of problems, and which may have ties to human vision processing. Furthermore, the pyramid can be projected onto a regular pattern in the plane, which makes it ideal for VLSI implementation [D2], and provides the possibility of constructing pyramids with thousands or millions of processing elements.

A pyramid computer can be viewed as being constructed from layers of mesh computers. A *mesh computer (mesh)* of size  $n$  is a collection of  $n$  simple *processing elements (PEs)* arranged in an  $n^{1/2} \times n^{1/2}$  grid, where each PE, except those along the border, is connected to its four nearest neighbors. A *pyramid computer (pyramid)* of size  $n$  has a base which is a mesh of size  $n$ . The base is called the

---

<sup>1</sup> This research was partially supported by National Science Foundation Grants MCS-83-01019, DCR-8507851, DCR-8608640, IRI-8800514 and an Incentives for Excellence Award from the Digital Equipment Corporation.

<sup>2</sup> Department of Computer Science, 226 Bell Hall, State University of New York, Buffalo, NY 14260, USA. email: miller@cs.buffalo.edu.

<sup>3</sup> Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122, USA. email: qstout@eecs.umich.edu.

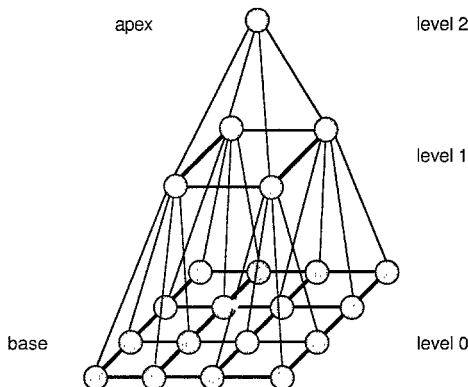


Fig. 1. Pyramid computer of size 16.

zeroth level. The level above the base, level 1, is a mesh of size  $n/4$ , and so on to the apex at level  $\log_4 n$ . Except for PEs along the boundary, each PE at level  $k$  is connected to its four mesh neighbors at level  $k$ , four children at level  $k-1$ , and parent at level  $k+1$ . (See Figure 1.) This model is the same as that used in [D2], [MS4], [St2], and [Ta3]. (A complete description of a pyramid computer is given in Section 2.)

A serial computer operating on  $n$  pieces of data must take  $\Omega(n)$  time to solve the problems considered in this paper, since all the data needs to be examined. For problems involving images, a mesh is a natural data structure, which means that the mesh computer is a logical alternative to a serial machine for solving the problems considered in this paper. Notice that for a mesh computer of size  $n$ , any algorithm that requires global communication of data must take  $\Omega(n^{1/2})$  time. While algorithms that run in  $\Theta(n^{1/2})$  time are a significant improvement over  $\Omega(n)$ -time serial algorithms, the relatively long communication diameter of the mesh, with respect to other parallel architectures, has been a major criticism of that architecture.

The pyramid computer's parent-child links provide the possibility of  $\Theta(\log n)$  algorithms, and some simple problems, such as finding a maximum and computing parity, can be solved in this time. However, many problems cannot be solved in  $O(\log n)$  time on the pyramid. For example, [St2] showed that sorting requires  $\Omega(n^{1/2})$  time on a pyramid of size  $n$ . Since  $\Theta(n^{1/2})$ -time sorting is possible on a mesh [ThK], the pyramid structure does not need to be utilized in order to obtain a  $\Theta(n^{1/2})$ -time pyramid-computer sorting algorithm. Sorting requires  $\Omega(n^{1/2})$  time because of the large amount of data movement required, while finding a maximum value can be completed in  $\Theta(\log n)$  time by rapidly eliminating values from consideration and moving the remaining values up the pyramid. This extreme sensitivity to the amount of data movement makes the pyramid a rather unique parallel architecture.

In this paper we present algorithms to solve a variety of problems, all of which deal with convexity. It will become clear that the differences in the running times of the algorithms are related to the rate at which data can be eliminated from

further consideration during the course of an algorithm. Convexity is a fundamental property of figures, and convexity algorithms for meshes have been widely studied (see [DR], [KR1], [MS3], [Sk]), but the only previous pyramid algorithms we know of are preliminary (conference) announcements of the results presented in this paper [MS1]–[MS3].

For an arbitrary black figure  $F$  on a white background, the extreme points representing the convex hull of  $F$  (i.e., the vertices of the smallest enclosing convex polygon of  $F$ ) often give a satisfactory compressed approximation of the figure. These extreme points can also be used to obtain important geometric classifications of the figure, such as determining a smallest enclosing box, smallest enclosing circle, and the external diameter of the figure. Efficient pyramid-computer algorithms that generate and use the extreme points of a single or multiple figures are given in this paper.

In Section 2 we give formal definitions of the pyramid computer, define the types of input that are considered, define convexity and related properties, and define some fundamental data movement operations that are used in the algorithms. Section 3 contains algorithms for deciding convexity and finding extreme points of single figures or sets of PEs with the same label, and Section 4 contains algorithms for determining convexity properties of multiple figures or sets of labels. Section 5 contains some algorithms that make use of extreme points to determine properties of sets, such as linear separability, diameter, smallest enclosing box, smallest enclosing circle, and so forth. Section 6 considers smaller pyramids for obtaining geometric information about image data, and Section 7 discusses related machine models such as the 4-ary (*quad*) tree, *mesh-of-trees*, and *hypercube*. In Section 8 we conclude with observations about optimality and some situations in which the results can be improved.

**2. Definitions.** In this section we define standard notation, formally define the mesh and pyramid computers, specify the forms of input that our problems use, define convexity and related properties, and discuss some fundamental data movement operations for the pyramid computer.

*2.1. Notation.* We use  $\Theta$  to mean “order exactly,”  $O$  to mean “order at most,” and  $\Omega$  to mean “order at least.” That is, given nonnegative functions  $f$  and  $g$  defined on the positive integers, we write  $f = \Theta(g)$  if and only if there are positive constants  $C_1$ ,  $C_2$ , and a positive integer  $N$  such that  $C_1g(n) \leq f(n) \leq C_2g(n)$ , whenever  $n > N$ . We write  $f = O(g)$  if and only if there is a positive constant  $C$  and an integer  $N$  such that  $f(n) \leq Cg(n)$ , for all  $n > N$ , and we write  $f = \Omega(g)$  if and only if there is a positive constant  $C$  and an integer  $N$  such that  $Cg(n) \leq f(n)$ , for all  $n > N$ .

*2.2. Models of Computation.* The *mesh computer (mesh)* of size  $n$  is a machine with  $n$  processing elements (PEs) arranged in a square lattice. To simplify exposition, we assume  $n = 4^c$  for some integer  $c$ . For all  $i, j \in [0, \dots, n^{1/2} - 1]$ , PE( $i, j$ ), representing the PE in row  $i$  and column  $j$ , is connected by a distinct bidirectional unit-time communication link to each of the following four neighbors that exist:

$PE(i, j + 1)$ ,  $PE(i, j - 1)$ ,  $PE(i + 1, j)$ , and  $PE(i - 1, j)$ . Each PE has a fixed number of registers (words), each of size  $O(\log n)$ . In one unit of time, each PE can perform standard arithmetic and Boolean operations on the contents of its registers and can also send and receive a word of data from each of its neighbors. Each PE contains its row and column indices, as well as a unique identification register, the contents of which is initialized to the concatenation of the PE's row and column indices.

A *pyramid computer (pyramid)* of size  $n$  is a machine that can be viewed as a full, rooted, 4-ary tree of height  $\log_4 n$ , with additional horizontal links so that each horizontal *level* is a mesh. (See Figure 1.) Another convenient view of the pyramid is as a tapering array of mesh computers. A pyramid computer of size  $n$  has at its base a mesh of size  $n$ , and a total of  $\frac{4}{3}n - \frac{1}{3}$  PEs. The levels are numbered so that the base is level 0 and the apex is level  $\log_4 n$ . A PE at level  $i$  is connected by a distinct bidirectional unit-time communication link to each of the following nine neighbors that exist: four adjacent PEs of the mesh at level  $i$ , four children at level  $i - 1$ , and a parent at level  $i + 1$ . Similar to the mesh, it is assumed that each PE has a fixed number of words, each of size  $O(\log n)$ , and that all arithmetic, Boolean, and communication operations take unit time. Further, each PE contains registers with its row, column, and level coordinates, the concatenation of which provides a unique index (label) for the PE. These registers can be initialized in  $\Theta(\log n)$  time if necessary.

**2.3. Input Data.** There are two different forms the data can take. One form of input consists of each base PE containing a label, where there are no assumptions made concerning the origin of the labels. In particular, the set of PEs with a given label may be disconnected. For example, classification labels give rise to such input [To].

The other form of input consists of an  $n^{1/2} \times n^{1/2}$  black/white picture  $M = \{m_{i,j}\}$  stored in the base of the pyramid so that base  $PE(i, j)$  stores pixel  $m_{i,j}$ . Two black pixels are called *4-neighbors (neighbors)* if they are stored in PEs sharing a communication link, and are *connected* if there is a path of neighboring black pixels between them. This divides the black pixels into (maximal) connected components, which we call *figures*. (We could also define figures and connected in terms of 8-neighbors, where a black pixel stored in  $PE(i_1, j_1)$  is an *8-neighbor* of a black pixel stored in  $PE(i_2, j_2)$  if  $\max(|i_1 - i_2|, |j_1 - j_2|) = 1$ . It should be noted that all of the results presented in this paper work with trivial modifications for this definition as well.) For many of the problems considered in this paper we assume that the figures have been *labeled*, which means that each black pixel has been assigned a label, where two black pixels are assigned the same label if and only if they are in the same figure. The following result was presented in [MS4] for using a pyramid computer to label the figures of a digitized black/white picture, and we make frequent use of it throughout the paper.

**LEMMA 1 [MS4].** *Given an arbitrary black/white picture stored one pixel per PE in the base of a pyramid computer of size  $n$ , the figures (connected components) can be labeled in  $\Theta(n^{1/4})$  time.*

*2.4. Lower Bounds.* Notice that the pyramid has a communication diameter of  $\Theta(\log n)$ , meaning that any two PEs can exchange messages in  $O(\log n)$  time, and some pairs require  $\Omega(\log n)$  time. This sets a lower bound on any problem which may require information to be exchanged between arbitrary processors. For very simple problems, such as counting the number of black pixels or locating the northernmost black pixel, it is quite easy to construct standard bottom-up collection, top-down distribution algorithms that finish in  $\Theta(\log n)$  time.

As noted earlier, sorting provides an example of a different class of problems for which the extensive data movement required forces the time to be  $\Omega(n^{1/2})$ . For many of the problems considered in this paper, however, neither of these bounds is appropriate. The logarithmic bound is still true, but overly optimistic, while the  $n^{1/2}$  bound does not apply because not as much data needs to be moved. We show that by exploiting properties of image data, many geometric problems can be solved in about  $n^{1/4}$  time. In some cases we can achieve  $O(n^{1/6})$ -time solutions, and for a few problems we achieve solutions that run in *polylogarithmic time*, i.e.,  $O(\log^k n)$  time, for some integer  $k > 0$ . A lower bound more appropriate to these situations is contained in the following theorem due to [MS4].

LEMMA 2 [MS4]. *In a pyramid computer of size  $n$ , suppose that there are  $B$  bits in the first column of the base,  $1 \leq B \leq Cn^{1/2} \log(n)$ , for some constant  $C > 0$ , and suppose that these are to be moved to the last column of the base. Then  $\Omega(\log(n) + [B/\log(n)]^{1/2})$  time is required.*

An example of where Lemma 2 can be applied follows. In [MS4] it is shown that labeling the figures of an arbitrary black/white picture stored in the base of a pyramid of size  $n$  may require  $O(n^{1/2})$  bits to be sent from the leftmost column in the base to the rightmost column of the base. Therefore, from Lemma 2, it is known that  $\Omega(n^{1/4}/\log^{1/2} n)$  time is required to label the base picture.

*2.5. Convexity.* Throughout this paper we identify the base PE( $i, j$ ) with the integer lattice point ( $i, j$ ). A set of base PEs is said to be *convex* if and only if the corresponding set of integer lattice points is convex, i.e., the smallest convex polygon containing them contains no other integer lattice points. This is the proper notion of convexity for integer lattice points, but it does have the annoying property that some disconnected sets of points, such as  $\{(1, 1), (3, 4)\}$ , are convex. The reader might wish to consult [KR1] and [MS3] for further discussions of convexity and digitized data. Some uses of convexity are illustrated in [MS3] and [To].

Given a set  $S$  of base PEs, the *convex hull* of  $S$ , denoted  $\text{hull}(S)$ , is the smallest convex set of PEs containing  $S$ . As for standard planar convexity, the convex hull of  $S$  is the intersection of all convex sets containing  $S$ . A PE  $P \in S$  is an extreme point of  $S$  if  $P \notin \text{hull}(S - P)$ . The extreme points of  $S$  are the corners of the smallest convex polygon containing  $S$ . We say that we have *marked the extreme points* of  $S$  if every PE  $P$  in the base of the pyramid has a Boolean variable that is true if and only if  $P$  is an extreme point of  $S$ . We say that we have *enumerated the extreme points* of  $S$  if we have marked the extreme points of  $S$ , and each extreme point has

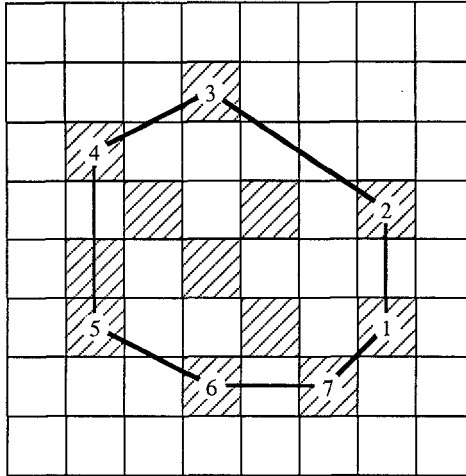


Fig. 2. Enumerated extreme points of a set.

been assigned a number as follows. The rightmost extreme point has the number 1 (if there are two rightmost extreme points then the lower one is number 1), and the numbering continues around the extreme points in counterclockwise order. Each extreme point must know its number, the total number of extreme points, and the locations of the preceding and following extreme points. (See Figure 2).

2.6. *Data-Movement Operations.* Efficient parallel algorithms for regular architectures often rely on a variety of fundamental operations to move data around [M]. Several such operations were introduced for the pyramid computer in [MS4], one of which is the *pyramid write*. The pyramid write is used to move data up the pyramid from a given mesh level to a desired mesh level that contains enough PEs to hold all of the distinct pieces of data being sent. For the algorithms in this paper we need only a restricted version of this operation.

LEMMA 3 [MS4]. *Given a pyramid computer of size  $n$ , fix constants  $p$  and  $C$ , where  $0 < C$  and  $0 < p < 1$ . Suppose there are no more than  $Cn^p$  PEs in the base that have a piece of data to be sent to level  $\lfloor \log_4 n^{1-p}/C \rfloor$ . (This level is the highest one with at least  $Cn^p$  PEs.) The pyramid write will move the data to its proper location in  $\Theta(n^{p/2} \log^{1/2} n)$  time.*

We emphasize that Lemma 3 considers  $p$  and  $C$  fixed, and only considers  $n$  to vary. This is all we need for our results, but in other circumstances we may need to determine completely the dependence on  $p$  and  $C$ , as well as on  $n$ .

We now introduce a closely related data-movement operation, the *sparse pyramid write*.

PROPOSITION 4. *Given a pyramid computer of size  $n$ , fix constants  $p$  and  $C$ , where  $0 < C$  and  $0 < p < 1$ . Suppose there are no more than  $Cn^p$  PEs in the base that have a piece of data to be sent to level  $\lfloor \log_4 n^{1-p}/C \rfloor$ . Further, in each base subsquare of*

size  $k$ ,  $0 \leq k \leq n$ , assume that there are no more than  $Ck^p$  PEs sending data. Then a sparse pyramid write will move the data to level  $\lceil \log_4 n^{1-p}/C \rceil$  in  $\Theta(n^{p/2})$  time.

PROOF. To perform a sparse pyramid write, fix  $p$  and  $C$ , and in parallel perform a sparse pyramid write in each quadrant. The level that the data is written to is either the same as the desired final level, or else it is one level below. Merge the data together using a mesh-computer operation such as random access read [MS3], and move up one level if necessary. For fixed  $p$  and  $C$ , the time obeys an equation of the form  $T(n) = T(n/4) + dn^{p/2}$ , which has a solution of  $\Theta(n^{p/2})$ .  $\square$

Our algorithms use one additional data-movement operation from [MS4], called *reducing a function*. Given sets  $A$  and  $B$  (which in this paper will always be the same), given a function  $f$  mapping  $A \times B$  into some set  $C$ , and given an associative, commutative operation  $*$  on  $C$ , let  $g$  be defined on  $A$  by  $g(a) = f(a, b_1) * f(a, b_2) * \dots * f(a, b_k)$ , where  $B = \{b_1, \dots, b_k\}$ .  $g$  is called *the reduction of  $f$* . For example, if  $A$  and  $B$  are sets of points in the plane, if  $C$  is the set of real numbers, if  $f$  computes the distance between points, and if  $*$  is the maximum operation, then  $g(a)$  is the farthest distance from  $a$  to any point in  $B$ .

LEMMA 5 [MS4]. Suppose that  $f$  and  $*$  can be computed in unit time, and that  $A$  and  $B$  are stored one item per PE at level  $l$  with  $m$  PEs,  $1 \leq m \leq n^{1/2}$ . The operation of reducing a function will compute  $g$  in  $\Theta(m^{1/2})$  time, storing  $g(a)$  in the PE at level  $l$  storing  $a$ .

We also need to use an extended reduction operation for the situation where there are three sets  $A_1, A_2$ , and  $A_3$ , a function  $f$  mapping  $A_1 \times A_2 \times A_3$  into  $C$ , and an associative commutative operation  $*$  on  $C$ . The reduction of  $f$  is the function  $g$  mapping  $A_1$  to  $C$  given by

$$g(a) = \underset{(x, y) \in A_2 \times A_3}{*} f(a, x, y).$$

This operation is presented in [M].

LEMMA 6 [M]. Suppose that  $f$  and  $*$  can be computed in unit time, and that  $A_1, A_2$ , and  $A_3$  are stored one item per PE at a level with  $m$  PEs,  $1 \leq m \leq n^{1/3}$ . Then the reduction of  $f$  can be computed in  $\Theta(m^{1/2})$  time.

**3. Single Figures or Labels.** In this section we consider input at the base of the pyramid that consists of a single digitized black figure on a white background (i.e., there is only one connected black component in the picture) or a figure with a single label (i.e., all base PEs that contain a label, contain the same label). Pyramid solutions to problems involving single figures are important for a variety of reasons, including the recent result presented in [MS6] which shows how to transport a *single figure pyramid algorithm* to a variety of architectures, including

the pyramid, mesh, hypercube, mesh-of-trees, and several bus-augmented mesh machines, and solve the corresponding problem for *multiple* figures. The approach used in [MS6] is to create a *simulated essential pyramid* over every figure and then use these to run the single-figure algorithms simultaneously. For a variety of architectures, it is shown in [MS6] that the multiple-figure version will run nearly as fast as the single-figure pyramid-computer version.

Before solving some general convexity problems for single figures, we first prove a lemma that will be quite useful. The lemma shows that once the extreme points of a figure have been marked, they can be enumerated in  $\Theta(\log n)$  time on a pyramid of size  $n$ .

LEMMA 7. *In a pyramid computer of size  $n$ , suppose the extreme points of a set  $S$  of base PEs have been marked. Then in  $\Theta(\log n)$  time the pyramid can enumerate the extreme points of  $S$ .*

PROOF. The algorithm requires that the processors determine certain basic information, as follows:

1. By using a bottom-up report, followed by a top-down broadcast operation, in  $(2 \log_4 n)$  steps all PEs in the pyramid computer can know the identity of the base processor that contains the extreme point that will be labeled 1. This may be accomplished as follows. Pass data up the pyramid so that at step  $i$ ,  $1 \leq i \leq \log_4 n$ , each PE at level  $i$  will know the identity of the extreme point in the base beneath it that would be labeled 1 if the extreme points of  $S$  were restricted to the subset of  $S$  in the base beneath it. After  $(\log_4 n)$  steps, the apex of the pyramid knows the identity of the extreme point in the subset of  $S$  beneath it (i.e., in the entire base) that is to be labeled 1. This information is broadcast to all PEs in the pyramid in  $(\log_4 n)$  steps by a straightforward top-down broadcast operation that is initiated by the apex.
2. In a similar fashion, in  $(2 \log_4 n)$  steps all PEs can simultaneously determine the total number of extreme points of  $S$ .
3. Using a bottom-up report procedure, in  $(\log_4 n)$  steps the apex of the pyramid can know the locations of the rightmost-bottommost, rightmost-topmost, topmost-rightmost, topmost-leftmost, leftmost-topmost, leftmost-bottommost, bottommost-leftmost, and bottommost-rightmost extreme point of  $S$ . These eight (not necessarily distinct) extreme points partition the extreme points of  $S$  into eight “triangular regions,” as shown in Figure 3.

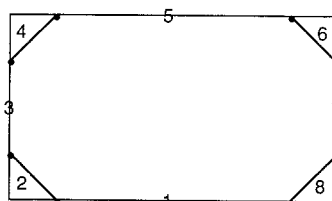


Fig. 3. The eight “triangular regions.”



4. In  $(2 \log_4 n)$  steps, every PE in the pyramid can know the total numbers of extreme points of  $S$  in the base under it that are in each of these eight regions. Notice that the boundaries of these (possibly degenerate) triangular regions may be generated in unit-time by every PE of the pyramid once they are informed as to the locations of these eight points.

Once this information has been determined, the extreme points of  $S$  can be labeled in  $\Theta(\log n)$  steps by having the apex recursively distribute ranges of the numbers to its children for each of the eight regions. Distributing the proper numbers to the children is straightforward since the extreme points represent a convex polygon. Notice that within each region, this is a prefix computation.

It only remains to show that each base PE containing an extreme point of  $S$  can determine the location of the preceding and succeeding extreme points of  $S$  in  $\Theta(\log n)$  steps. During the numbering process, as every PE passes ranges of numbers to its children, it also determines if any of its children are responsible for extreme points that have a preceding or succeeding extreme point in another one of its children. For each such case, the PE creates a *neighbor record*, which consists of the numbers of the extreme points involved, as well as the identity of the PE creating the record. When the numbering phase of the algorithm terminates, these neighbor records are sent in lockstep fashion down to the base. When a base PE receives a neighbor record, it is examined to determine if either of the numbers in the record correspond to its extreme point number. If there is a match, then the location of the extreme point is appended to the record, and the record is sent back up to the PE that generated it, while otherwise the record is discarded. Finally, the neighboring information is sent down to the base in lockstep fashion so every base PE in  $S$  will know not only its number, but the location of its predecessor and successor. Hence, the extreme points of  $S$  have been enumerated.

The algorithm requires a fixed number of  $\Theta(\log n)$ -time top-down and bottom-up tree-like operations. Therefore, the running time of the algorithm is  $\Theta(\log n)$ .  $\square$

We now look at the general problem of enumerating the extreme points of a single convex set of base PEs (i.e., marking and numbering the PEs that contain extreme points of the figure). This is important in many image-processing applications that require a compact description of a single convex figure for storage or transmission purposes. One such description is given by the extreme points of the figure.

Before we give our result for enumerating the extreme points of a single convex figure, we first give a simple technical lemma which we find extremely useful. The lemma is concerned with the fact that it is possible to take a digitized convex figure, divide it into two parts by a straight line parallel to one of the grid axes, and have points which are extreme points of the parts but not of the entire figure. An important consequence of the following lemma is that there are only  $O(\log n)$  such points.

**LEMMA 8.** *Given a convex figure  $F$  on a grid, suppose the grid is divided vertically in half and the extreme points of the restriction of  $F$  to the right half are determined.*

Suppose  $p$  and  $q$  are extreme points along the top of the right-hand portion. Further, suppose  $p$  and  $q$  are not extreme points of  $F$ , and that  $q$  is further from the dividing line than is  $p$ . Then  $q$  is more than twice as far from the dividing line as  $p$  is.

PROOF. Since  $p = (p_x, p_y)$  and  $q = (q_x, q_y)$  are not extreme points of  $F$ , it must be that the line segment  $L$  from  $q$ , passing through  $p$  and continuing on to the dividing line, lies in the convex hull of  $F$  when viewed as a figure in the real plane (rather than just on the grid). If  $q$  were less than twice  $p$ 's distance to the dividing line, then the grid point  $r = (2p_x - q_x, 2p_y - q_y)$  would lie on  $L$  and be on the same side of the dividing line as  $p$  and  $q$ . This means that  $r$  would be in  $F$  and, specifically, in the restriction of  $F$  to the right half. However, since  $p$  is halfway between  $q$  and  $r$ , this contradicts the assumption that  $p$  is an extreme point of the restriction of  $F$  to the right half.  $\square$

The next result is concerned with enumerating the extreme points of a convex set of processors. The algorithm used to solve this problem is based on a bottom-up divide-and-conquer solution strategy.

**THEOREM 9.** *In a pyramid computer of size  $n$ , suppose the PEs with a given label form a convex set  $S$ . Then in  $\Theta(\log n)$  the time the extreme points of  $S$  can be enumerated.*

PROOF. The algorithm proceeds in two phases. The first phase of the algorithm marks the extreme points of  $S$ , and the second phase of the algorithm enumerates them by applying the algorithm of Lemma 7. Therefore, we need only describe the marking phase of our algorithm.

Our algorithm for marking extreme points works in a bottom-up fashion, where at step  $k$ ,  $0 \leq k \leq \log_4 n$ , decisions regarding extreme points are made by PEs at level  $k$ . Consider the PEs at level  $k$  in the pyramid. These are the apices of disjoint subpyramids with bases of size  $2^k \times 2^k$ . Call the base of each of these disjoint subpyramids a *subsquare*. At the end of step  $k$ , suppose that in each  $2^k \times 2^k$  subsquare those points which are not extreme points of the restriction of the figure to their subsquare have been marked as not being extreme, while those that are extreme points in their subsquare remain as candidate extreme points for the entire figure. Suppose further that for each way of forming a square of four subsquares, each point which is not an extreme point in the larger square has also been marked as not being extreme. Notice that these larger squares overlap, and some correspond to bases of subpyramids of height  $k + 1$ , while others do not.

Now consider step  $k + 1$ , where for PEs at level  $k + 1$  we call the base of each of the corresponding subpyramids a *block*. Since each block is a square of four  $2^k \times 2^k$  subsquares, we know that at the beginning of step  $k + 1$  those points which are not extreme points in their block have already been marked as not extreme. The purpose of step  $k + 1$  (i.e., the recursive step) is to identify those points which were candidate extreme points at the end of step  $k$ , but which are not extreme points in some square of four blocks. This must be done for all possible squares

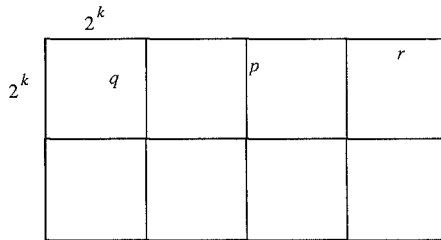


Fig. 4. Eliminating extreme points.

of four blocks, not just those corresponding to the base of a subpyramid of height  $2 + k$ .

Each square of four blocks can be viewed as merging two blocks together (for two sets in parallel), and then merging these rectangles together. Both merge steps are similar, so only the first is described. Since all squares of four  $2^k \times 2^k$  subsquares have been considered during step  $k$ , for a candidate extreme point  $p$  to be marked as not being a candidate during step  $k + 1$ , there must be a triangle containing  $p$  with one of its vertices being an extreme point  $q$  more than  $2^k$  away. An example is given in Figure 4, where  $q$  can be taken to be the rightmost remaining candidate extreme point in its subsquare, and  $p$  the leftmost remaining candidate extreme point in its subsquare. Further, if  $q$  causes two remaining candidate extreme points to be eliminated, then an argument as in Lemma 8 shows that the second, call it  $r$ , must be more than twice as far from  $q$  as  $p$  is. Therefore,  $r$  must be in a different subsquare than  $p$ , and since it must also be the next candidate extreme point after  $p$ , in left–right order,  $r$  must be the leftmost extreme point in its subsquare. This same consideration shows that  $q$  cannot eliminate more than two candidate extreme points within this square of four blocks. Thus, by knowing for each subsquare only the leftmost and rightmost remaining candidate extreme points along the top, the leftmost and rightmost remaining candidate extreme points along the bottom, the topmost and bottommost remaining candidate extreme points along the left, and the topmost and bottommost remaining candidate extreme points along the right, we can determine all false extreme points in the merger of the blocks.

The apex of each block maintains this information about its block. By exchanging information with its neighbors at level  $k + 1$ , in constant time, simultaneously for every apex, an apex can determine for each possible square of four blocks which of its candidate extreme points should be eliminated from further consideration. To finish step  $k + 1$ , every PE at level  $k + 1$  initiates a top-down broadcast message of the information to its block, and supplies its parent with the information necessary to start step  $k + 2$ .

The time between the start of step  $k + 1$  and the start of step  $k + 2$  is  $\Theta(1)$ . The time it takes to finish the final top-down broadcast after the last step is complete is  $\Theta(\log n)$ . Therefore, the total running time is  $\Theta(\log n)$ .  $\square$

The next problem we consider is that of marking the convex hull of a single figure that is described by a set of enumerated extreme points. If the original figure

was not convex, then the figure that we generate will be an approximation of the original figure. However, if the original figure is “blob”-like, then this operation can be viewed approximately as the inverse operation to that of generating the extreme points of the figure.

**THEOREM 10.** *In a pyramid computer of size  $n$ , suppose the extreme points of the PEs with a given label have been enumerated, then in  $\Theta(\log n)$  time the PEs in the convex hull of this set can be marked.*

**PROOF.** Since the extreme points have been enumerated, all base processors containing an extreme point know the location of the extreme points preceding and succeeding it according to the counterclockwise ordering of extreme points. Assume that there are  $p$  extreme points in the figure, where the base PE containing the  $i$ th extreme point is denoted  $P_i$ ,  $1 \leq i \leq p$ . Each base PE  $P_i$  assumes responsibility for the hull edge, call it  $e_i$ , between its (extreme) point and the extreme point that follows it in the counterclockwise ordering. All  $P_i$  can now determine, in  $\Theta(1)$  time, the processor in the pyramid at maximum level (closest to the apex), denoted  $P_{i(m)}$ , that is an ancestor of  $P_i$  such that  $e_i$  crosses the boundary between the subpyramids rooted at the children of  $P_{i(m)}$ . All base processors,  $P_i$  now pass up to their respective  $P_{i(m)}$ , the hull edge  $e_i$  that they are responsible for, as well as a flat indicating which side of  $e_i$  is on the inside of the hull. Notice that no processor in the pyramid will be responsible for more than four such edges. After  $\log_4 n$  units of time, all processors in the pyramid will know the (at most) four edges in the base that cross the boundaries of the subpyramids of its children. This information is then passed down the pyramid in lockstep fashion from all  $P_{i(m)}$  to their descendants. As each base PE receives such information, it decides in  $\Theta(1)$  time whether or not it is in the convex hull.  $\square$

The next result provides an optimal solution to the problem of deciding whether or not a marked set of PEs is convex. The algorithm is straightforward, combining the results just presented in Theorems 9 and 10. First, use the algorithm associated with Theorem 9 to mark the “extreme points” of the set. Using the preceding two “extreme points” and the succeeding two “extreme points”, every PE determines whether or not it can decide that the figure is not convex. Combining these results, it can be decided whether or not the “extreme points” are convex. If the set of “extreme points” are not convex, then the algorithm halts and it is known that the original marked set of PEs is not convex. Otherwise, use the algorithm associated with Theorem 9 to mark the convex hull of the extreme points, and compare those marked PEs with the original marked set of PEs. This gives the following result.

**COROLLARY 11.** *In a pyramid computer of size  $n$ , in  $\Theta(\log n)$  time the set of PEs with a given label can decide whether or not they are convex.*

The next problem considered is that of enumerating the extreme points of an arbitrary set of base PEs. This extreme-point generation algorithm degrades by a

factor of  $\Theta(\log n / \log \log n)$  over the convexity-query algorithm just presented. This is counterintuitive in that the solution to the convexity-query problem can be obtained faster than generating the extreme points of a given set of base PEs. It should be noted that a  $\Theta(\log n)$ -time extreme-point generation algorithm is an open problem.

The extreme-point generation algorithm that is presented in Theorem 12 follows a top-down divide-and-conquer solution that exploits the following fact about extreme points. A point is an extreme point if and only if it is the first point of the figure contacted as some line is moved toward the figure from infinity. By way of an example, suppose that for a given digital figure embedded in an  $n^{1/2} \times n^{1/2}$  grid, there exists a unique topmost, bottommost, leftmost, and rightmost extreme point (which may be detected by finding the first point contacted as lines of slope 0 come from the top and bottom, and lines of slope  $\infty$  come from the left and right, respectively). Then for any extreme point  $p$  of the figure that is between the topmost point and the leftmost point (in the ordering of extreme points), there must be a slope in the range  $(n^{-1/2}, n^{1/2})$  such that  $p$  is the first point of the figure contacted as a line with this slope comes toward the figure from the upper-left direction. If the line with slope  $(n^{-1/2} + n^{1/2})/2$  is used to detect an extreme point between the topmost and leftmost extreme points, then

- (1) if the first point contacted is the topmost extreme point, then there are no extreme points of the figure between these two that will be detected by slopes in the range  $[(n^{-1/2} + n^{1/2})/2, n^{1/2})$ , while
- (2) if the first point contacted is the leftmost extreme point, then there are no extreme points of the figure between these two that will be detected by slopes in the range  $(n^{-1/2}, (n^{-1/2} + n^{1/2})/2]$ , while
- (3) if a first point contacted was not the topmost or leftmost extreme point, then this first point (or, in the case of a multiple detection, the outermost points contacted) is an extreme point.

These situations define a recursive search procedure that is used to detect extreme points. Notice that if a single new extreme point is found in an interval, then this new extreme point is used to create two subintervals, both of which are searched for additional extreme points. These observations form the basis of the algorithm that follows.

**THEOREM 12.** *In a pyramid computer of size  $n$ , the extreme points of the base PEs with a given label can be enumerated in  $\Theta(\log^2 n / \log \log n)$  time.*

**PROOF.** The algorithm uses a top-down divide-and-conquer solution strategy. First an algorithm requiring  $O(\log^2 n)$  time is given, and then we show how to modify it so as to reduce the running time to  $\Theta(\log^2 n / \log \log n)$ . Let  $S$  be the set of base PEs with a given label. Observe that starting with an arbitrary line  $l$  far away from  $S$  and moving it in toward  $S$  (without changing its slope), then the element of  $S$  that  $l$  reaches first must be an extreme point of  $S$ . (If several elements of  $S$  are reached simultaneously, then only the two extreme points of this one-dimensional set of points are extreme points of  $S$ .) Notice that a PE  $P$  that is

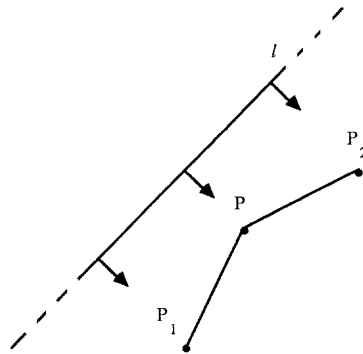


Fig. 5. Detecting  $P$  as an extreme point.

an extreme point of  $S$ , with  $P_1$  and  $P_2$  the preceding and succeeding extreme points, respectively, will be detected as an extreme point of  $S$  by a line  $l$  that has a slope between  $\text{slope}(\overline{PP_1})$  and  $\text{slope}(\overline{PP_2})$  and moves toward  $S$  from the concave side of the angle formed by  $P_1PP_2$ . (See Figure 5.)

The set  $S$  of base PEs is embedded in an  $n^{1/2} \times n^{1/2}$  grid. Therefore, the difference of slopes between any two distinct pairs of these grid points must be greater than  $1/n$ . Except for vertical lines, all lines through two PEs have slopes between  $-n^{1/2}$  and  $n^{1/2}$ , so by checking all multiples of  $1/n$  between these two values, all extreme points of  $S$  will be detected. In fact, while only  $\Theta(n)$  different slopes can actually occur, the algorithm will check  $\Theta(n^{3/2})$  slopes. This will cause no significant time penalty, and it is much simpler to consider just multiples of  $1/n$ .

In  $\Theta(\log n)$  time, the apex determines the (not necessarily distinct) rightmost-bottommost, rightmost-topmost, topmost-rightmost, topmost-leftmost, leftmost-topmost, leftmost-bottommost, bottommost-leftmost, and bottommost-rightmost members of  $S$ . These are all extreme points of  $S$ , and they divide the perimeter of  $S$  into eight (or fewer) intervals. (Refer back to Figure 3.) Four of these intervals, e.g., between the topmost-rightmost and the topmost-leftmost points, contain no more extreme points, while the other four intervals, e.g., between the topmost-rightmost and the rightmost-topmost points, might contain more extreme points. For each of the four intervals that might contain more extreme points there is a corresponding interval of line slopes that may be used to locate the extreme points in the interval. For example, in the interval between the topmost-rightmost and the rightmost-topmost extreme points, the slopes are in the range of  $-n^{-1/2}$  to  $-n^{1/2}$ . From now on an *interval* means a pair of endpoint coordinates, along with the associated interval of slopes. Notice that when a slope  $m$  is being used, if each base PE computes the inner product of its  $(x, y)$  position with  $(1/m, 1)$ , then the base PE with the greatest inner product is the one that would be reached first. (If the line approaches from the opposite side, then the base PE with the least inner product is the one reached first.)

Initially, the apex of the pyramid is responsible for the four intervals that may contain more extreme points. The algorithm proceeds in stages, where a PE is

responsible for at most eight intervals during any stage. At the beginning of each stage, if the endpoints of an interval in a PE lie beneath the same child, then responsibility for that interval is passed on to that child (which may in turn pass it further down). Next, for each interval that a PE is responsible for, the PE creates a record corresponding to the interval's endpoints and the middle slope. In a top-down fashion, starting with the apex, copies of these records are then sent from every PE to each of its four children. Every PE receiving such a record ignores it if none of its descendants could be an extreme point as discovered by that slope, while otherwise it passes the record down to its children, along with any such records it may generate. Notice that no PE passes more than eight such records to any of its children.

When these records reach the base, each element of  $S$  determines its inner product with the indicated slope and appends this to the record, along with the PE's coordinates, and passes this record back to its parent. This information is passed up through the pyramid, where when a parent receives multiple copies of an interval, it passes along only the one with the largest inner product. (If there are ties, then the two outermost extreme points among the ties are passed up.) When this information returns to the PE generating the request, several possibilities can occur. For example, if two new extreme points, say  $N_1$  and  $N_2$ , were discovered between extreme points  $P_1$  and  $P_2$ , as in Figure 6, then the original  $P_1P_2$  interval is divided into three new intervals, namely,  $P_1N_1$  and  $N_2P_2$ , both of which have no more than half as many slopes as the original  $P_1P_2$  interval, and  $N_1N_2$  which requires no further work. Other possibilities are treated similarly. Finally, each time an extreme point is found it is marked.

Each stage of the algorithm takes  $\Theta(\log n)$  time. Since there are at most  $\Theta(n^{3/2})$  slopes, and each stage subdivides an interval's slopes by at least half, then there are at most  $\Theta(\log n^{3/2}) = \Theta(\log n)$  steps. When finished, all extreme points have been marked, and in an additional  $\Theta(\log n)$  time the extreme points can be enumerated by applying the algorithm of Lemma 7.

The algorithm as described requires  $\Theta(\log^2 n)$  time. To reduce the time of the algorithm to  $\Theta(\log^2 n / \log \log n)$ , have each PE that is responsible for an interval divide that interval's slopes into  $\log_2 n$  pieces, instead of two pieces. These records are sent down in serial fashion (i.e., pipelined), where no PE passes more than  $8 \log_2 n$  records to its children. Each stage still takes  $\Theta(\log n)$  time, but because the intervals are being broken up faster only  $\Theta(\log n / \log \log n)$  stages are needed. Therefore, the algorithm finishes in the time indicated.  $\square$

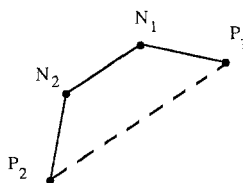


Fig. 6. Discovering two new extreme points in an interval.

**4. Multiple Figures or Labels.** In this section we consider the problems of enumerating extreme points and deciding convexity for multiple sets of base PEs (i.e., multiple figures or labels). Since there may be  $\Theta(n)$  disjoint sets of base PEs, applying the algorithms of Section 3 to one set of PEs at a time would yield substantially suboptimal running times in the worst case. In order to determine convexity properties for multiple sets of base PEs efficiently, it appears that the algorithms must be designed to work on multiple sets simultaneously. Further, since  $\Omega(n^{1/2})$  time is required if only the base mesh of the pyramid is used, faster algorithms must use both the parent-child and mesh links that are available in the pyramid. Finally, the algorithms must avoid having many figures trying to send data through the apex, for then the apex becomes a bottleneck.

The running times of algorithms presented in this section are slower than the running times of algorithms from Section 3 that involved single figures. Nevertheless, the results presented in this section are at most a logarithmic factor from optimal for the pyramid computer. The first result of this section describes an algorithm to mark and enumerate the extreme points for each of an arbitrary number of digitized figures.

**THEOREM 13.** *In a pyramid computer of size  $n$  with a digitized black/white picture in its base, in  $\Theta(n^{1/4} \log n)$  time the extreme points of every figure can be enumerated.*

**PROOF.** The algorithm uses a bottom-up divide-and-conquer approach. For each figure, first enumerate the extreme points of the restriction of the figure to each of the four quadrants of the picture. For a figure in two or more quadrants, as in Figure 7, we need to determine which extreme points in the quadrant are not extreme points in the entire figure. These form an interval, e.g., in Figure 7 they are the ones between the dotted lines. To find these dotted lines, we use a binary search on the hull edges of the (at most four) pieces of the figure. For example, in Figure 7, the topmost dotted line can be found as follows. Find the leftmost and rightmost extreme points of the restriction of the figure to the right subimage. Using this information, find and send the top hull edge which is in the middle of these two extreme points in the enumeration ordering (as restricted to the right subimage) to the left subimage. Next, determine if the line collinear with this edge passes above the restriction of the figure to the left subimage, passes through or

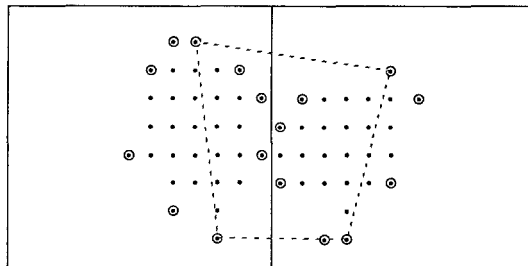


Fig. 7. Not all extreme points of the subregions are extreme points of the region.



below it, or is tangent to it (and hence is the dotted line). In the first case, the edge and all hull edges preceding it in the counterclockwise ordering (with respect to the restriction of the figure to the right subimage) and eliminated from further consideration, while in the second case the edge and all hull edges following it are eliminated.

Next, the left-hand piece sends over its middle edge, and a similar check eliminates half of the hull edges with respect to the restriction of the figure to the left subimage. A binary search for the top dotted line continues in a natural fashion, alternating between the halves. Eventually, either an edge on the dotted line is found, or else both pieces locate an extreme PE representing an extreme point such that the edge on one side is too high, and the edge on the other side is too low. In this case the dotted line passes through the PE. Once the intervals of extreme points between the dotted lines have been determined, it is easy to enumerate the remaining points using their old enumeration information.

There may be  $\Theta(n^{1/2})$  figures merging pieces together, so for each step of the binary search, for all figures, simultaneously, we move an edge up to a level of size  $\Theta(n^{1/2})$ , across the level, and down to the piece on the other side. We use a sparse pyramid write, with  $p = \frac{1}{2}$  to move the data up. This sparse pyramid write can be used since, in any subsquare of size  $k$ , if a piece of data is being moved up, then it is in a figure crossing the border of the subsquare, and there are  $O(k^{1/2})$  such figures. A similar operation moves the data down. The time obeys a recurrence equation of the form  $T(n) = T(n/4) + cn^{1/4} \log n$ ,  $c$  a constant, which has a solution of  $T(n) = \Theta(n^{1/4} \log n)$ .  $\square$

Suppose we know *a priori* that the digitized figures in the base of the pyramid are all convex. Then by incorporating the approach of Theorem 9, the time of the previous theorem for enumerating the extreme points of each figure can be reduced by a factor of  $\Theta(\log n)$ .

**COROLLARY 14.** *In a pyramid computer of size  $n$  with a digitized picture in its base, suppose all the figures are convex. Then the extreme points of each figure can be enumerated in  $\Theta(n^{1/4})$  time.*

In Section 3 the algorithm associated with Corollary 11 can be used to decide whether or not a digitized figure is convex. This algorithm was designed by making a minor modification to the algorithm in Theorem 9 that enumerates the extreme points of a convex digitized figure. A similar modification can be made so that we can detect for each digitized figure whether or not it is convex.

**COROLLARY 15.** *In a pyramid computer of size  $n$  with a digitized picture in its base, in  $\Theta(n^{1/4})$  time every figure can decide whether or not it is convex.*

Suppose that we are given an arbitrary number of (not necessarily connected) labeled sets of PEs in the base of the pyramid. Further, suppose that we are interested in enumerating the extreme points of each labeled set of PEs. By a fairly straightforward wire-counting argument, it is easy to show that in the worst case,

$\Theta(n)$  pieces of data may have to cross from the left half of the pyramid to the right half of the pyramid. Therefore, any pyramid-computer algorithm to solve this problem will require  $\Omega(n^{1/2})$  time. Since a mesh algorithm to solve this problem in  $\Theta(n^{1/2})$  time is given in [MS3], we simply ignore the pyramid structure above the base mesh, and use the mesh-computer algorithm to enumerate the extreme points of each set of base PEs.

**THEOREM 16.** *In a pyramid computer of size  $n$ , in  $\Theta(n^{1/2})$  optimal time we can enumerate the extreme points of the PEs with the same label, for all labels simultaneously.*

**5. Applications of Extreme Points.** In this section we solve problems by giving algorithms that make use of enumerated extreme points. These algorithms solve problems such as deciding if two sets of PEs are linearly separable, determining a smallest enclosing box, determining the smallest enclosing circle, and determining the diameter of a set of multiple sets of PEs. Additional applications of extreme points can be found in [MS3] and [To].

A set  $A$  of base PEs is *linearly separable* from a set  $B$  of base PEs if and only if there is a straight line in the plane such that all elements of  $A$  lie on one side of the line, and all elements of  $B$  lie on the other side. A well-known observation is that two sets are linearly separable if and only if their convex hulls are disjoint. Given the enumerated extreme points of two sets of (not necessarily distinct) base PEs, in  $\Theta(\log n)$  time it can be determined whether or not these two sets are linearly separable as follows. Mark the convex hull of  $A$  and the convex hull  $B$  such that a base PE has the value  $\alpha$  if it is in the convex hull of  $A$ , and the value  $\beta$  if it is in the convex hull of  $B$ . This takes  $\Theta(\log n)$  time by applying the algorithm associated with Theorem 10 once for  $A$  and a second time for  $B$ . All base PEs send to the apex a Boolean flag that is set to “true” if the PE is labeled  $\alpha$  and  $\beta$ , and that is set to “false” otherwise. As each PE in the pyramid receives the four Boolean values from its children, they are logically “or”ed together and passed up. In  $\Theta(\log n)$  the apex knows the answer to the query which it propagates to all PEs in the pyramid in  $\Theta(\log n)$  time. Hence the algorithm is complete in  $\Theta(\log n)$  time.

**COROLLARY 17.** *In a pyramid computer of size  $n$ , suppose the extreme points corresponding to a set  $A$  of base PEs have been enumerated, as have the extreme points of a set  $B$  of base PEs. Then in  $\Theta(\log n)$  time it can be decided whether or not  $A$  is linearly separable from  $B$ .*

Given a metric  $d$  and a set  $S$  of base PEs, the diameter of  $S$  with respect to  $d$  is  $\max\{d(P,Q) | P, Q \in S\}$ . We assume  $d$  is one of the  $l_p$  metrics, such as the  $l_1$  (taxicab) metric,  $l_\infty$  (chessboard) metric, or  $l_2$  (Euclidean) metric. The  $l_p$  distance from  $(a, b)$  to  $(c, d)$  is  $(|a - c|^p + |b - d|^p)^{1/p}$  for  $1 \leq p \leq \infty$ , and the  $l_\infty$  distance from  $(a, b)$  to  $(c, d)$  is  $\max(|a - c|, |b - d|)$ . The metrics can be computed in unit time, and for them the diameter is  $\max\{d(P, Q) | P \text{ and } Q \text{ are extreme points of } S\}$ . Metrics

other than the  $l_p$  metrics could also be used, but a complete discussion of appropriate metrics is outside the focus of this paper.

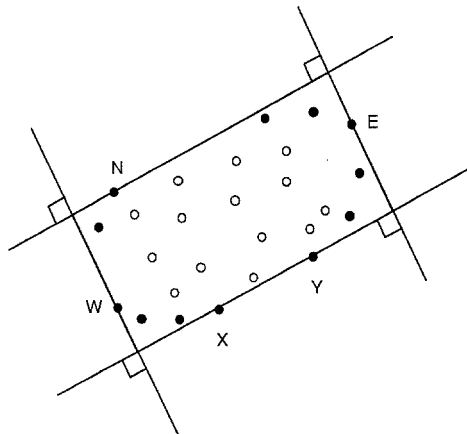
Given a set  $S$  of points in the plane, a *smallest enclosing rectangle* (also known as a *smallest box*) is a rectangle of least area containing  $S$ . (If rectangles of zero area contain  $S$ , then we want the smallest line segment containing  $S$ .) If  $S$  is finite, then it can be shown that a smallest enclosing rectangle must contain an extreme point of  $S$  on each side, and at least one side must contain two consecutive extreme points [FS]. The *smallest enclosing circle* is the circle of least area containing  $S$ . Smallest enclosing rectangles and smallest enclosing circles appear in [FS], [MS3], and [To].

**THEOREM 18.** *In a pyramid computer of size  $n$ , suppose the extreme points of a labeled set of PEs have been marked. Then in  $\Theta(n^{1/6})$  time the diameter (measured with any given  $l_p$  metric), smallest enclosing circle, and a smallest enclosing rectangle of this set of PEs can be determined.*

**PROOF.** We use the number-theoretic fact that for a set of lattice points in a square of size  $k$  there are  $O(k^{1/3})$  extreme points [VK]. Using a sparse pyramid write, in  $\Theta(n^{1/6})$  time move the extreme points to a level in the pyramid that consists of a mesh of size  $\Theta(n^{1/3})$ .

To determine the diameter, let  $E$  be the set of extreme points and let  $d$  compute the given metric. Let  $g$  be defined for  $e \in E$  by  $g(e) = \max\{d(e, x) | x \in E\}$ . So,  $g(e)$  is the maximum distance from  $e$  to any other labeled PE, and  $g$  can be computed in  $\Theta(n^{1/6})$  time by reducing  $d$  with respect to maximum, as mentioned in Section 2.6. The diameter of the set is just  $\max\{g(e) | e \in E\}$ , which can be computed in  $\Theta(\log n)$  time once  $g$  has been computed.

A smallest enclosing rectangle can be found in a similar manner. For each edge, assume an orientation of the points that has this edge as the southernmost edge parallel to the  $x$ -axis, and use reduction to find the northernmost, westernmost, and easternmost points. For each hull edge, these three points determine the minimum-area enclosing rectangle that includes the edge. (See Figure 8.) A smallest



**Fig. 8.** Determining a smallest enclosing box for hull edge XY.

enclosing rectangle of the entire set is found by taking a minimum over these rectangles.

The smallest enclosing circle is the largest circle either passing through three of the extreme points or having two of the extreme points as a diameter [MS3]. Thus the smallest enclosing circle can be found by using a reduction of a function over a triple cross product of the extreme points, which too can be done in  $\Theta(n^{1/6})$  time, as described in Section 2.6.  $\square$

Much work in digital image processing and pattern recognition has been spent on the fundamental problem of deciding whether a digitized figure could have arisen as the digitization of a straight line segment [R1], [R2], [RK], [K2], [K3], [KR1], [G]. Our next theorem proves that a pyramid computer of size  $n$  can determine in  $\Theta(\log n)$  time whether or not a digitized figure could have arisen as the digitization of a straight line segment. The theorem will combine a result about digital arcs [KR1] with Corollary 11 in order to arrive at this asymptotically optimal algorithm.

Digitization can make the detection of even basic properties of a figure nontrivial to determine. The digitization scheme that we use is the standard grid-intersection scheme [R3] for digitizing arcs. (For a further discussion of digitization schemes, see [R2], [RK], [G], [K1]–[K3], [KR1], [KR2], [KS], and [DS]). Given a coordinate grid superimposed on an arc  $A$ , then as we traverse  $A$  we cross a succession of grid lines. Whenever  $A$  crosses a grid line, the PE associated with the integer lattice point nearest to the crossing line becomes a part of  $A$ 's digitization. In the case where  $A$  crosses a grid line halfway between two lattice points, the tie is resolved by choosing the PE associated with the lattice point that lies to the right of  $A$  (in the sense that we are traversing  $A$ ) to be a member of the digitization of  $A$ . Define  $PEs(i \pm 1, j \pm 1)$  to be the 8-neighbors of  $PE(i, j)$ , assuming they exist. Given a set  $S$  of PEs, and two PEs  $P, Q \in S$ ,  $P$  and  $Q$  are defined to be 8-connected if and only if there exists a finite connected path of 8-neighbors in  $S$  from  $P$  to  $Q$ . A set  $S$  of PEs is an 8-connected set if and only if for all PEs  $P, Q \in S$ ,  $P$  and  $Q$  are 8-connected.

An 8-connected set  $D$  of two or more PEs is a *digital arc* if all but two of the PEs in  $D$  have exactly two 8-neighbors in  $D$ , and the exceptional two, called the *endpoints*, each have exactly one 8-neighbor in  $D$  [KR1]. Given two lattice points  $p$  and  $q$ , corresponding to two PEs in  $D$ , the line segment  $\overline{pq}$  is said to *lie near*  $D$  if, for any point  $(x, y)$  of  $\overline{pq}$ ,  $(x, y) \in \mathbb{R}^2$ , there exists a lattice point  $(a, b)$  corresponding to a  $PE(a, b) \in D$  such that  $\max\{|a - x|, |b - y|\} < 1$ .  $D$  is said to have the *chord property* if, for every  $p, q \in D$ , the line segment  $\overline{pq}$  lies near  $D$  [R1].

LEMMA 19 [R1]. *A digitized arc has the chord property if and only if it is the digitization of a straight line segment.*

LEMMA 20 [KR1]. *A set of processors  $S$  has the chord property if and only if  $S$  is convex.*

From Lemmas 19 and 20, we see that  $D$  could have arisen as the digitization of a straight line segment if and only if it is a convex digitized arc. By the result

in [KR1], this implies that a convex set  $D$  of two or more PEs is the digitization of a straight line segment if and only if

- (1) all but two of the PEs of  $D$  have exactly two 8-neighbors in  $D$ , and the exceptional two have exactly one 8-neighbor in  $D$ , and
- (2)  $D$  is 8-connected.

Further, we can show that if  $D$  is convex and satisfies property (1), then it satisfies property (2) as well. (This is false for nonconvex sets, as can be seen by considering a disconnected set consisting of digitizations of a circle and a line). Thus a convex set  $D$  of two or more PEs is the digitization of a straight line if and only if it satisfied property (1).

This characterization yields an efficient algorithm to determine whether or not a set  $D$  of lattice points could have arisen as the digitization of a straight line segment. (It is assumed that  $D$  corresponds to a set of labeled PEs). From Corollary 11, it can be decided in  $\Theta(\log n)$  time whether or not  $D$  is convex. If  $D$  is not convex, then the algorithm halts and it is known that  $D$  could not have arisen as the digitization of a straight line segment, while otherwise the algorithm continues in an effort to determine whether or not  $D$  is a digital arc (property (1)). To determine if property (1) holds, each base PE that is a member of  $D$  determines in  $\Theta(1)$  time the number of its 8-neighbors that are members of  $D$ . By passing these results up to the apex and combining them at each level, after  $\Theta(\log n)$  time the apex will know whether or not property (1) holds, and hence knows whether  $D$  could have arisen as the digitization of a straight line segment. This gives the following.

**THEOREM 21.** *Given a digitized black/white picture stored in the base of a pyramid computer of size  $n$ , in  $\Theta(\log n)$  time it can be decided whether or not the set of black pixels could have arisen as the digitization of a straight line segment.*

Corollary 15 gives an algorithm to decide whether or not each of an arbitrary number of figures in a digitized picture is convex in  $\Theta(n^{1/4})$  time. Combining the algorithms associated with Lemma 1, Corollary 15, and Theorem 21, we obtain the following.

**COROLLARY 22.** *In a pyramid computer of size  $n$ , if there are multiple sets, then the ones that could have arisen as the digitization of straight line segments can be determined in  $\Theta(n^{1/4})$  time.*

**6. Smaller Pyramids.** There are at least three ways we could consider using smaller pyramids for obtaining geometric information about image data. In the first, suppose it is known that no figure has an  $l_1$  diameter greater than  $D$ . Then a pyramid computer of size  $n$  can be conceptually partitioned into subpyramids of size  $\Theta(D^2)$ , where problems such as labeling or marking extreme points are solved in the subpyramids, exchanging data as needed between neighboring subpyramids. This would result in faster algorithms, replace “ $n$ ” with “ $D^2$ ”

in the time bounds. This method has been used in pyramid algorithms appearing in [Ta2] and [St2].

A second way to use small pyramids is to consider a picture of size  $N$  and a pyramid of size  $n$  with  $n \leq N$ . The pyramid could work on subsquares of size  $n$ , gluing results together as necessary. While this is often required in practice, and efficient gluing is not always easy, we do not consider this class of smaller pyramids further.

A third possibility is to consider again a picture of size  $N$  and a pyramid of size  $n$ , with  $n \leq N$ , but with a slight change in the pyramid. (For convenience, assume that  $N = 4^i n$ , for  $i \geq 0$  an integer.) This *modified pyramid computer* (*modified pyramid*) of size  $n$  has PEs with wordlengths of size  $O(\log N)$ . Each of the  $n$  base PEs has  $O(N/n)$  words of memory, while PEs above the base need only  $\Theta(1)$  words of memory. The input to the algorithm consists of a picture of size  $N$  that is partitioned in a natural fashion into subsquares so that each base PE is given a subsquare of size  $N/n$ .

Many of the multiple figure algorithms presented in this paper assume that the figures (connected components) have been labeled. For a picture of size  $n$  stored 1 pixel per base PE on a pyramid of size  $n$ , [MS4] gives a  $\Theta(n^{1/4})$ -time labeling algorithm, as mentioned in Section 2.3. Before discussing convexity algorithms for digitized pictures on a modified pyramid, we first discuss some fundamental results related to labeling connected components of a digitized black/white picture of size  $N$  on a modified pyramid of size  $n$ .

**THEOREM 23.** *Given a digitized black/white picture of size  $N$  stored in the base of a modified pyramid computer of size  $n$ ,  $n \leq N$ , in  $\Theta(N/n + N^{1/4})$  time the connected components can be labeled.*

**PROOF.** The component-labeling algorithm in [MS4] works by moving data up the pyramid, where PEs at level  $i$  need  $\Theta(2^i)$  time to perform their calculations. In a modified pyramid, the base PEs simulate the bottom  $\log_4 N/n$  levels. Each base PE must simulate  $(N/n)/4^i$  PEs at level  $i$ ,  $0 \leq i \leq \log_4 N/n$ , so the time required to perform computations at level  $i$  is  $\Theta((N/n)/2^i)$ . Therefore, the base PEs finish their simulations in  $\Theta(N/n)$  time. The higher PEs in [MS4] need  $\Theta(N^{1/4})$  time, so the total time is as claimed.  $\square$

Given a digitized black/white picture of size  $N$  distributed  $\Theta(N/n)$  pixels per base PE on a modified pyramid of size  $n$ , then in order to minimize the running time of the component-labeling algorithm, the relationship between  $n$  and  $N$  should be  $n = \Omega(N^{3/4})$ . This gives the following.

**COROLLARY 24.** *To within a multiplicative factor, on a picture of size  $N$  a modified pyramid computer of size  $N^{3/4}$  can perform component labeling as fast as a pyramid computer of size  $N$ .*

**COROLLARY 25.** *For  $n \leq N^{3/4}$ , a modified pyramid computer of size  $n$  can label components with linear speed-up on a picture of size  $N$ .*

For algorithms presented in this paper that consist of straightforward top-down and bottom-up divide-and-conquer strategies, modified pyramid algorithms may be patterned after the component-labeling algorithm associated with Theorem 23. The crucial technique is to let the base of the modified pyramid simulate the bottom  $\log_4 N/n$  levels of the (unmodified) pyramid. For example, for the  $\Theta(\log n)$ -time pyramid-computer algorithms that are presented in this paper, the base level simulation technique just described will yield  $\Theta(N/n + \log n)$ -time algorithms on a modified pyramid of size  $n$  for input of size  $N$ . This comes from letting each base processor simulate  $(N/n)/4^i$  PEs at level  $i$ ,  $0 \leq i \leq \log_4 N/n$ , where processors at level  $i$  need  $\Theta(1)$  time to perform their calculations, and higher PEs need  $\Theta(\log n)$  time to complete the operation.

For other algorithms presented in this paper, a combination of serial and parallel (pyramid) algorithms may be more efficient than a base-level simulation. A good example is the algorithm associated with Theorem 12, which enumerates the extreme points of an arbitrary set of lattice points. In order to enumerate the extreme points of a set of lattice points of size  $N$  on a modified pyramid of size  $n$ , we could first let each base processor determine the extreme points of its subset, simultaneously for all base processors, and then apply the rest of the pyramid algorithm in a natural fashion. This would give a running time of  $\Theta(N/n + \log^2 n / \log \log n)$ , since the extreme points of a set of lattice points of size  $k$  can be enumerated in  $\Theta(k)$  time on a serial machine.

Finally, there are algorithms in this paper that rely on data-movement operations such as the (sparse) pyramid write and the (extended) reduction of a function. A discussion of their implementation on a modified pyramid is outside the focus of this paper, but the reader might wish to consult [MS4] for insight into extending these operations, and hence the algorithms that rely on these operations, to a modified pyramid.

**7. Related Machine Models.** There are several machine models that are related to the pyramid computer. One is the 4-ary (*quad*) tree, i.e., a pyramid without the nearest-neighbor links. Like the pyramid, the quadtree has a logarithmic communication diameter, but it is easy to show that the apex of a quadtree often acts as a bottleneck. For example, a simple wire-counting argument shows that a quadtree needs  $\Omega(n)$  time to sort data and  $\Omega(n^{1/2})$  time to label components or determine nearest neighbors, in the worst case. Further, these time bounds remain even if higher PEs have additional memory, as suggested in [AS]. It should be noted that algorithms attaining these lower bounds are quite straightforward.

The mesh computer also has shortcomings due to the fact that its communication diameter is  $\Theta(n^{1/2})$ . This shows that  $\Omega(n^{1/2})$  time is required for all of the problems in this paper. Mesh algorithms solving these problems in this time appear in [NS] and [MS3].

A more interesting model is known as *orthogonal trees or mesh-of-trees* [UI]. This model has a base mesh of size  $n$  augmented with a tree of processors over each row and over each column (for a total of  $3n - 2n^{1/2}$  PEs), with these trees being disjoint except at their leaves. In this model  $\Theta(n^{1/2} \log^2 n)$  bits can be

moved from the leftmost  $\log n$  columns to the rightmost  $\log n$  columns in  $\Theta(\log n)$  time. This is a significant improvement over the pyramid bound in Lemma 2, though not enough to permit polylogarithmic time sorting. The mesh-of-trees has not received much consideration as an image-processing machine, but it can perform all of the problems considered here in polylogarithmic time [MS6].

The mesh-of-trees does, however, have some drawbacks. The pyramid computer can be laid out on a chip using area proportional to that required by the base mesh [D1], but the mesh-of-trees needs a factor of  $\log^2 n$  more area [UI]. Further, the pyramid computer has close ties to other objects of interest to research in image processing, including (region) quadtrees and animal optic systems.

Additional models that solve all of the geometric problems mentioned in this paper in polylogarithmic time are the hypercube [MS5], the reconfigurable mesh [MPRS1], [MPRS2], the PRAM, and some related pyramid models that have been proposed in [St3].

**8. Final Remarks.** Our results indicate a progression of complexity in finding extreme points of convex hulls. Single convex figures are the most constrained, enabling a bottom-up approach to eliminate points from further consideration rapidly. Next come arbitrary single figures and sets of PEs with a given label. Multiple figures require much more data movement than single figures, but far less data movement than PEs with arbitrary labels. It is also interesting to note that, at least for our algorithms, it is sometimes slightly easier to decide convexity than it is to find the extreme points.

It has been shown in [MS6] that a pyramid algorithm designed for a *single* figure can be implemented on a variety of architectures to yield efficient solutions to the same problem for *multiple* figures. Therefore, designing efficient single-figure pyramid algorithms, such as those presented in Section 3, are important for a variety of reasons. Many of the results presented in this paper are optimal, and we believe all are near optimal for the pyramid computer. Since each problem can have inputs that require combining information at opposite edges of the base, all algorithms must have a worst-case time of  $\Omega(\log n)$ . Therefore, Lemma 7, Theorem 9, Corollary 11, Theorem 10, and Corollary 17 are all optimal. The only algorithms for a single object that may be nonoptimal are Theorem 18 and the  $\Theta(\log^2 n / \log \log n)$ -time algorithm to enumerate the extreme points of a single arbitrary set of PEs. In fact, as mentioned previously, a  $\Theta(\log n)$ -time algorithm to solve this problem is an interesting open problem. For multiple figures or sets of processors, all of the algorithms presented in this paper are either optimal or within a polylogarithmic factor of optimal. It appears that for these problems lower bound arguments that are stronger than the bit-counting argument of Lemma 2 are needed for proofs of optimality.

Despite the optimality of our results, there are some situations that arise in practice for which faster algorithms can be developed. One such situation is when there is a bound on the number of different labels or figures. This is



obvious from a comparison of the results of Sections 3 and 4, and there are various ways we can interpolate those results. For example, suppose it is known that there are no more than  $n^c$  labels, where  $0 < c < 1$ . By slightly modifying the algorithm in Theorem 13 in  $\Theta(n^{c/2} \log n)$  time we can enumerate the extreme points for all labels.

## References

- [AS] N. Ahuja and S. Swamy, Multiprocessor pyramid architectures for bottom-up image analysis, *IEEE Trans. Pattern. Anal. Mach. Intell.*, **6** (1984), 463–474.
- [B] P. Burt, The pyramid as a structure for efficient computation, in *Multiresolution Image Processing and Analysis* (A. Rosenfeld, ed), Springer-Verlag, Berlin, 1984, pp. 6–35.
- [BV] P. J. Burt and G. S. van der Wal, Iconic image analysis with the pyramid vision machine (PVM), *Proc. IEEE 1987 Workshop on Pattern Analysis and Machine Intelligence*, pp. 137–144.
- [CFLS] V. Cantoni, M. Ferretti, S. Levialdi, and R. Stefanelli, Papi: pyramidal architecture for parallel image processing, *Proc. Computer Arithmetic Conf.*, Urgana, Ill. (1985).
- [CM] Ph. Clermont and A. Merigot, Real time synchronization in a multi-SIMD massively parallel machine, *Proc. IEEE 1987 Workshop on Pattern Analysis and Machine Intelligence*, pp. 131–136.
- [DS] L. Dorst and W. M. Smeulders, Discrete representation of straight lines, *IEEE Trans. Pattern Anal. Mach. Intell.*, **6** (1984), 450–463.
- [D1] C. R. Dyer, A VLSI pyramid machine for hierarchical parallel image processing, *Proc. IEEE Conf. on Pattern Recognition and Image Processing*, (1981), pp. 381–386.
- [D2] C. R. Dyer, A Quadtree Machine for Parallel Image Processing, Tech. Report KSL 51, University of Illinois at Chicago Circle, 1981.
- [D3] C. R. Dyer, Pyramid algorithms and machines, in *Multicomputers and Image Processing Algorithms and Programs* (K Preston and L. Uhr, eds.), Academic Press, New York, 1982, pp. 409–420.
- [DR] C. R. Dyer and A. Rosenfeld, Parallel image processing by memory augmented cellular automata, *IEEE Trans. Pattern. Anal. Mach. Intell.*, **3** (1981), 29–41.
- [FS] H. Freeman and R. Shapira, Determining the minimal-area enclosing rectangle for an arbitrary closed curve, *Comm. ACM*, **18** (1975), 409–413.
- [Fr] G. Fritsch *et al.*, EMSY85—the Erlangen multi-processor system for a broad spectrum of applications, *Proc. 1983 Int. Conf. on Parallel Processing*, pp. 325–330.
- [G] M. Gaafar, Convexity verification, block-chords, and digital straight lines, *Comput. Graphics Image Process.*, **6** (1977), 361–370.
- [K1] C. E. Kim, On the cellular convexity of complexes, *IEEE Trans. Pattern. Anal. Mach. Intell.*, **3** (1981), 617–625.
- [K2] C. E. Kim, Digital convexity, straightness, and convex polygons, *IEEE Trans. Pattern Anal. Mach. Intell.*, **4** (1982), 618–626.
- [K3] C. E. Kim, Three-dimensional digital line segments, *IEEE Trans. Pattern Anal. Mach. Intell.*, **5** (1983), 231–234.
- [KR1] C. E. Kim and A. Rosenfeld, Digital straight lines and convexity of digital regions, *IEEE Trans. Pattern Anal. Mach. Intell.*, **4** (1982), 149–153.
- [KR2] C. E. Kim and A. Rosenfeld, Convex digital solids, *IEEE Trans. Patterns Anal. Mach. Intell.*, **4** (1982), 612–618.
- [KS] C. E. Kim and J. Sklansky, Digital and cellular convexity, *Pattern Recognition*, **15** (1982), 359–387.
- [KV] J. J. Koenderink and A. J. van Dorn, New type of raster scan preserves the topology of the image, *Proc. IEEE*, **67** (1979), 1465–1466.
- [LZ] A. Lempel and J. Ziv, Compression of two-dimensional data, *IEEE Trans. Inform. Theory*, **32** (1986), 2–8.

- [M] R. Miller, Ph.D. thesis, State University of New York at Binghamton, 1985. *Pyramid Computer Algorithms*.
- [MS1] R. Miller and Q. F. Stout, The pyramid computer for image processing, *Proc. 7th Int. Conf. on Pattern Recognition*, (1984), pp. 240–242.
- [MS2] R. Miller and Q. F. Stout, Convexity algorithms for pyramid computers, *Proc. 1984 Int. Conf. on Parallel Processing*, pp. 177–184.
- [MS3] R. Miller and Q. F. Stout, Geometric algorithms for digitized pictures on a mesh-connected computer, *IEEE Trans. Pattern Anal. Mach. Intell.*, **7** (1985), 216–228.
- [MS4] R. Miller and Q. F. Stout, Data movement techniques for the pyramid computer, *SIAM J. Comput.*, **16** (1987), 38–60.
- [MS5] R. Miller and Q. F. Stout, Some graph and image processing algorithms for the hypercube, *Hypercube Multiprocessors 1987*, SIAM, Philadelphia, Pa., pp. 418–425.
- [MS6] R. Miller and Q. F. Stout, Simulating essential pyramids, *IEEE Trans. Comput.*, **37** (1988), 1642–1648.
- [MS7] R. Miller and Q. F. Stout, *Parallel Algorithms for Regular Architectures*, MIT Press, Cambridge, Mass, 1992.
- [MPRS1] R. Miller, V. K. Prasanna Kumar, D. Reisis, and Q. F. Stout, Image computations on reconfigurable VLSI arrays, *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, (1988), pp. 925–930.
- [MPRS2] R. Miller, V. K. Prasanna Kumar, D. Reisis, and Q. F. Stout, Data movement operations and applications on reconfigurable VLSI arrays, *Proc. 1988 Int. Conf. on Parallel Processing*, Vol. I, pp. 205–208.
- [NS] D. Nassimi and S. Sahni, Finding connected components and connected ones on a mesh-connected parallel computer, *SIAM J. Comput.*, **9** (1980), 744–757.
- [R1] A. Rosenfeld, Digital straight line segments, *IEEE Trans. Comput.*, **23** (1974), 1264–1269.
- [R2] A. Rosenfeld, Digital topology, *Amer. Math. Monthly*, **86** (1979), 621–630.
- [R3] A. Rosenfeld (ed.), *Multiresolution Image Processing and Analysis*, Springer-Verlag, Berlin, 1984.
- [RK] A. Rosenfeld and C. E. Kim, How a digital computer can tell whether a line is straight, *Amer. Math. Monthly*, **89** (1982), 230–235.
- [Sa] B. Sakoda, Parallel Construction of Polygonal Boundaries from Given Vertices on a Raster, Tech. Report CS81 1–21, Department of Computer Science, Pennsylvania State University, 1981.
- [Sc] D. H. Schaefer *et al.*, The PMMP—a pyramid of MPP processing elements, *Proc. 18th Hawaiian Int. Conf. on Systems Science*, Vol. 1 (1985), pp. 178–184.
- [SHBV] D. H. Schaefer, P. Ho, J. Boyd, and C. Vallejos, The GAM pyramid, in *Parallel Computer Vision* (L. Uhr, ed.), Academic Press, New Yorks, 1987, pp. 15–42.
- [Sk] J. Sklansky, Recognition of convex blobs, *Pattern Recognition*, **2** (1970), 3–10.
- [St1] Q. F. Stout, Drawing straight lines with a pyramid cellular automation, *Inform. Process. Lett.*, **15** (1982), 233–237.
- [St2] Q. F. Stout, Sorting, merging, selecting, and filtering on tree and pyramid machines, *Proc. 1983 Int. Conf. on Parallel Processing*, pp. 214–221.
- [St3] Q. F. Stout, Pyramid computer algorithms optimal for the worst-case, in *Parallel Computer Vision* (L. Uhr, Ed.), Academic Press, New York, 1987, pp. 147–168.
- [Ta1] S. L. Tanimoto, Towards Hierarchical Cellular Logic: Design Considerations for Pyramid Machines, Tech. Report 81-02-01, Department of Computer Science, University of Washington, 1981.
- [Ta2] S. L. Tanimoto, Programming techniques for hierarchical parallel image processors, in *Multicomputers and Image Processing Algorithms and Programs* (K. Preston and L. Uhr, eds.), Academic Press, New York, 1982, pp. 421–429.
- [Ta3] S. L. Tanimoto, Sorting, Histogramming, and Other Statistical Operations on a Pyramid Machine, Tech. Report 82-08-02, Department of Computer Science, University of Washington, 1982.
- [TaK] S. L. Tanimoto and A. Klinger, *Structured Computer Vision: Machine Perception Through Hierarchical Computation Structures*, Academic Press, New Yorks, 1980.

- [ThK] C. D. Thompson and H. T. Kung, Sorting on a mesh-connected parallel computer, *Comm. ACM*, **20** (1977), 263–271
- [To] G. T. Toussaint, Pattern recognition and geometrical complexity, *Proc. 5th Int. Conf. on Pattern Recognition*, (1980), pp. 1324–1347.
- [Uh1] L. Uhr, Layered “recognition cone” networks that preprocess, classify, and describe, *IEEE Trans. Comput.*, **21** (1972), 758–768.
- [Uh2] L. Uhr, *Algorithm-Structured Computer Arrays and Networks*, Academic Press, New York, 1984.
- [UI] J. D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Rockville, Md., 1984.
- [VK] K. Voss and R. Klette, On the Maximum Number Edges of Convex Digital Polygons Included into a Square, *Forschungsergebnisse Nr. N/82/6*, Friedrich-Schiller Universitat, Jena, 1982.