COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 29, 311–328 (1985)

Reconstruction of Quadtrees from Quadtree Medial Axis Transforms*

HANAN SAMET

Computer Science Department, University of Maryland, College Park, Maryland 20742

Received January 22, 1984; revised May 24, 1984

An algorithm is presented for reconstructing a quadtree from its quadtree medial axis transform (QMAT). It is useful when performing operations for which the QMAT is well suited (e.g., thinning of an image). The algorithm is a postorder tree traversal which propagates the subsumption of each BLACK QMAT node in the eight possible directions. Analysis of the algorithm shows that its average execution time is proportional to the number of leaf nodes in the quadtree. The algorithm also serves to reinforce the appropriateness of the definition of the quadtree skeleton which does not permit a BLACK quadtree node to require more than one element of the quadtree skeleton for its subsumption.

1. INTRODUCTION

The quadtree [9, 26] is a technique of region representation which has been the subject of much research in recent years. This research has focussed on its interchangeability with more common representations such as borders [3, 16] arrays [17], and rasters [18, 27] as well as performing basic image processing operations [4, 19, 20, 29]. It has been used as a basis for a cartographic information system [14], computer graphics [5, 6], and also for modeling 3-dimensional objects [7, 12, 30].

The development of the quadtree was motivated to a large degree by a desire to save storage. It was seen as a means of aggregating blocks having similar properties. Unfortunately, the amount of space that is required depends on the positioning of the regions in the image. Shifting the contents of an image, represented by a quadtree, by one pixel can easily lead to a dramatic increase of the space required. Thus it is desirable to have an image representation that is less sensitive to shift. The quadtree medial axis transform (QMAT) [24] is a data structure that has such a property while at the same time occupying at most the same amount of space as the quadtree. In essence, a quadtree is a partition of a region into a set of disjoint squares having sides of lengths that are powers of two, whereas the QMAT results in a partition of space into a set of possibly nondisjoint squares having sides whose lengths are sums of powers of two. An alternative to the shift sensitivity problem is to try to construct an optimal (space-wise) quadtree by trying a subset of all possible shifts [11] which is nevertheless an expensive process.

It is important to note that the motivation for our study of QMAT is its decreased sensitivity to shift operations and *not the usual purpose of obtaining skeleton-like approximations* of the image. Nevertheless, algorithms such as the one we present here should be useful when the QMAT is used to facilitate operations such as thinning [13] and it is desired to have the quadtree representation of the thinned

* The support of the U.S. Army Engineer Topographic Laboratories under Contract DAAG-70-81-C-0059 is gratefully acknowledged.

0734-189X/85 $3.00

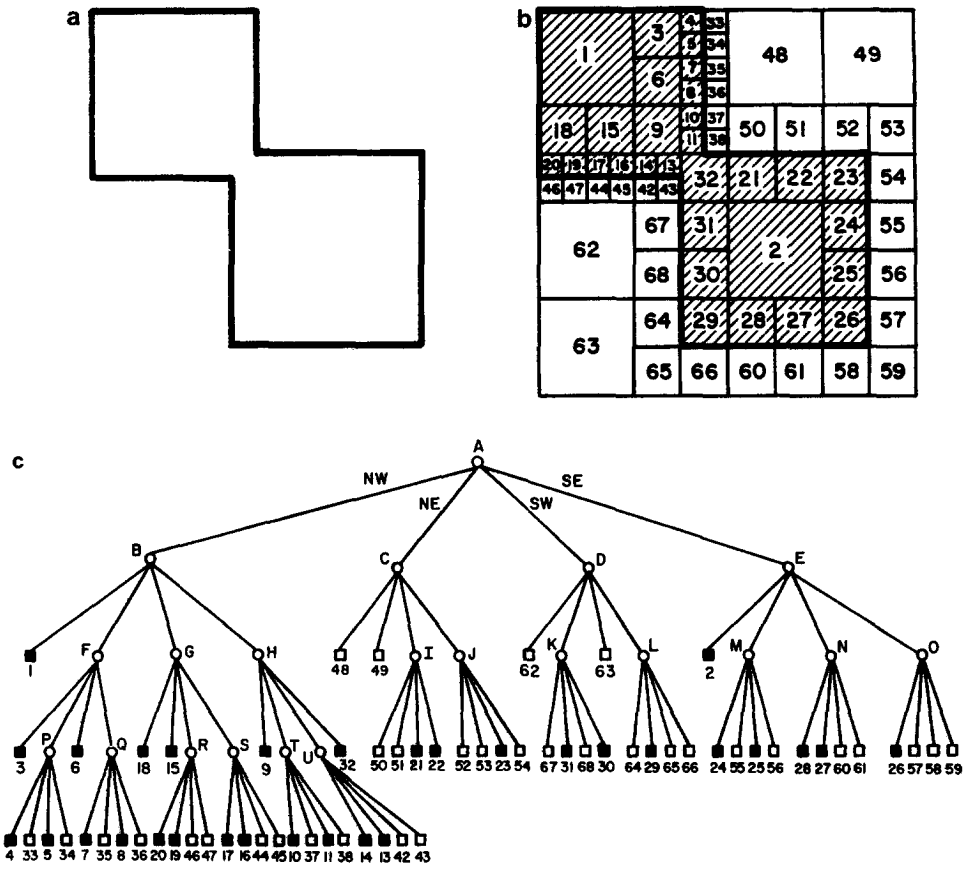

FIG. 1. An image, its maximal blocks, the corresponding quadtree, the chessboard distance transform, the block decomposition of the QMAT, and the QMAT. Blocks in the image and in the QMAT are shaded: (a) Sample image; (b) Block decomposition of the image in (a); (c) Quadtree representation of the blocks in (b); (d) Chessboard distance transform of (b); (e) Block decomposition of the QMAT of (b) (radius values are within parentheses); (f) QMAT representation of the blocks in (b) (radius values are within parentheses).

image. To see this, we note that traditional thinning requires examination of the image pixel by pixel. In contrast, using a quadtree representation, we can take advantage of its hierarchical nature and perform thinning by traversing the image block by block and possibly replacing a block by a smaller block. The QMAT representation stores the radius of the area spanned by the QMAT block thereby facilitating the detection of stopping points for thinning (i.e., when connectivity is lost). Now, thinning may be implemented by traversing the image block by block and at times just changing the value of the radius. Since the QMAT is often even more compact than the quadtree, thinning is speeded up as well, since there are fewer elements (i.e., blocks) to examine. Further discussion of thinning and skeletons is beyond the scope of this paper.

An algorithm for the construction of a QMAT from its quadtree is given in [24]. In this paper we present an algorithm for the reverse process—i.e., the reconstruction of a quadtree from its QMAT. Section 2 contains a discussion of the representa-

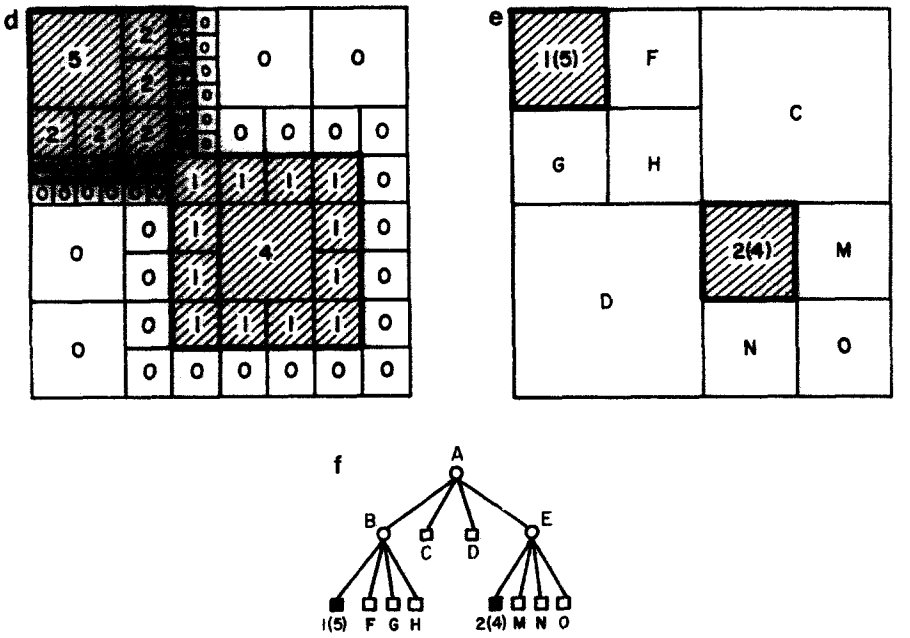

FIG. 1.—*Continued.*

tion that we use. Section 3 reviews the concepts of distance, quadtree skeletons, and QMATs. In Section 4 we present our algorithm both informally and formally using a variant of ALGOL 60, while Section 5 contains an analysis of its execution time.

2. DEFINITIONS AND NOTATION

We assume that the given image is a 2^n by 2^n array of unit square "pixels." The quadtree is an approach to image representation based on a successive subdivision of the array into quadrants. In essence, we repeatedly subdivide the array into quadrants, subquadrants, ..., until we obtain blocks (possibly single pixels) which consist entirely of 1's or 0's. This process is represented by a tree of out-degree 4 in which the root node represents the entire array, the four sons of the root node represent the quadrants, and the terminal nodes correspond to those blocks of the array for which no further subdivision is necessary. For example, Fig. 1b is a block decomposition of the region in Fig. 1a while Fig. 1c is the corresponding quadtree. In general, BLACK and WHITE square nodes represent blocks consisting entirely of 1's and 0's respectively. Circular nodes, also termed GRAY nodes, denote nonterminal nodes.

Each node in a quadtree is stored as a record containing seven fields. The first five fields contain pointers to the node's father and its four sons, labeled NW, NE, SE, and SW. Given a node P and a son I, these fields are referenced as FATHER(P) and SON(P, I), respectively. At times it is useful to use the function (SONTYPE(P), where SONTYPE(P) = Q iff SON(FATHER(P), Q) = P. The sixth field, NODETYPE, describes the contents of the block of the image which the node represents—i.e., BLACK, WHITE, or GRAY. The seventh field, DIST, indicates the

distance [22] to the nearest WHITE node according to the specified distance metric. This field is only meaningful for BLACK nodes. Both WHITE and GRAY nodes are said to have distance zero.

Let the four sides of a node's block be called its N, E, S, and W sides. They are also termed its boundaries and at times we speak of them as if they are directions. Figure 2 shows the relationship between the quadrants of a node's block and its boundaries. The specification of the spatial relationships between the various sides is facilitated by use of the functions OPSIDE, CSIDE, and CCSIDE. Given side B, OPSIDE(B) corresponds to the side facing B; e.g., OPSIDE(N) = S. CSIDE(B) and CCSIDE(B) correspond to the sides adjacent to side B in the clockwise and counterclockwise directions respectively; e.g., CSIDE(N) = E and CCSIDE(N) = W. We also define the following predicates and functions to aid in the expression of operations involving a block's quadrants and boundaries. ADJ(B, I) is true if and only if quadrant I is adjacent to boundary B of the node's block, e.g., ADJ(E, SE) is true. REFLECT(B, I) yields the SONTYPE value of the block of equal size that is adjacent to side B of the block having SONTYPE value I; e.g., REFLECT(N, SE) = NE, REFLECT(E, SE) = SW, REFLECT(S, SE) = NE, and REFLECT(W, SE) = SW. COMMONSIDE($Q1, Q2$) indicates the boundary of the block containing quadrants $Q1$ and $Q2$ that is common to them (if $Q1$ and $Q2$ are not adjacent brother quadrants, then the value of COMMONSIDE($Q1, Q2$) is undefined); e.g., COMMONSIDE(SE, SW) = S while COMMONSIDE(NW, SE) is undefined. QUAD($S1, S2$) is the quadrant bounded by boundaries $S1$ and $S2$ (if $S1$ and $S2$ are not adjacent boundaries, then QUAD($S1, S2$) is undefined); e.g., QUAD(S, E) = SE while QUAD(S, N) is undefined. Similarly, OPQUAD(QUAD($S1, S2$) = QUAD(OPSIDE($S1$), OPSIDE($S2$)).

For a quadtree corresponding to a 2^n by 2^n array we say that the root is at level n, and that a node at level i is at a distance of $n - i$ from the root of the tree. In other words, for a node at level i, we must ascend $n - i$ FATHER links to reach the root of the tree. Note that the farthest node from the root of the tree is at level ≥ 0. A node at level 0 corresponds to a single pixel in the image. Also, we say that a node is of size 2^S if it is found at level S in the tree—i.e., it has a side of length 2^S.

A natural by product of the treelike nature of the quadtree representation is that many basic operations are implemented as tree traversals. The difference between them is in the nature of the computation that is performed at the node. Often, these computations involve the examination of some nodes that are adjacent to the node being processed. We shall speak of these adjacent nodes as neighbors. In order to be more precise, given node P, corresponding to block P and a direction D, we say that node Q, corresponding to block Q, is the *neighbor* of node P in direction D (i.e., *neighbor*(P, D) = Q), when both of the following conditions are satisfied.

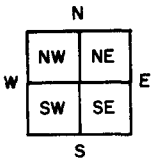

FIG. 2. Relationship between a block's four quadrants and its boundaries.

(1) P and Q share a common border, even if only a corner.

(2) The block corresponding to Q is the smallest block (it may be GRAY) of size greater than or equal to the block corresponding to P.

For example, block 50 in Figs. 1b and c has neighbors 48, 51, 22, 21, 32, T, and Q. Note that the neighboring nodes need not be distinct—i.e., a node may serve as a neighbor in more than one direction. For example, for node 50 in Fig. 1, node 48 overlaps the N and NE neighboring directions.

3. QUADTREE SKELETONS AND QUADTREE MEDIAL AXIS TRANSFORMS

Before presenting the QMAT to the quadtree conversion algorithm, let us briefly review the concepts behind the QMAT. The QMAT of an image is based on the notions of a medial axis. The medial axis of a region [1, 13] is a subset of its points each of which has a distance from the complement of the region (using a suitably defined distance metric) which is a local maximum. The *medial axis transform* (MAT) consists of the set of medial axis or *skeleton* points and their associated distance values.

Distance is measured using the *chessboard* distance metric, given below,

$$d_M(p,q) = \max\{\,|p_x - q_x|, |p_y - q_y|\,\}.$$

In [19] it is shown to be most appropriate for a quadtree since it has the property that for a given point p, the set of points $\{q\}$ such that $d(p,q) \leq t$ is a square. This metric is used to define the chessboard distance transform for a quadtree as a function DIST which yields for each BLACK block in the quadtree the chessboard distance from the center of the block to the nearest point which is on a BLACK-WHITE border. More formally, letting x be the center of a BLACK block b, z be a point on the border of the WHITE block w, say $B(w)$, we have

$$F(b,w) = \min_{z \in B(w)} d(x,z)$$

$$\mathrm{DIST}(b) = \min_{w} F(b,w).$$

In addition, we say that DIST of a WHITE block is zero and that the border is BLACK for the purpose of the computation of F and DIST.

We are now ready to define the concept of a quadtree skeleton. Given a BLACK block, b_i, it is convenient to use $S(b_i)$ to refer to that part of the image spanned by a square with side width $2 \cdot \mathrm{DIST}(b_i)$ centered about b_i. Let the set of BLACK blocks in the image be denoted by B. A *quadtree skeleton* is the set T of BLACK blocks satisfying the following properties:

(1) $area(B) = \cup_i S(t_i)$

(2) for any $t_j \in T$, $\nexists b_k \in B (b_k \neq t_j) \ni S(t_j) \subseteq S(b_k)$

(3) $\forall b_i \in B, \exists t_j \in T \ni S(b_i) \subseteq S(t_j)$.

For example, for the quadtree of Figs. 1b and c, the quadtree skeleton consists of nodes 1 and 2 with chessboard distance transform values of 5 and 4, respectively, assuming a 2^4 by 2^4 image. Property (1) insures that the entire image is spanned by

the quadtree skeleton. Property (2) is termed the *subsumption property* and we say that b_j is *subsumed* by b_k when $S(b_j) \subseteq S(b_k)$. Property (2) means that the elements of the quadtree skeleton are the blocks with the largest distance transform values. Property (3) insures that no block in B and not in T requires more than one element of T for its subsumption—e.g., the case that one half of the block is subsumed by one element of T and the other half is subsumed by another element of T is not permitted. In [24] it is shown that the quadtree skeleton of an image is unique.

The QMAT of an image is the quadtree whose BLACK nodes correspond to the BLACK blocks comprising the quadtree skeleton and their associated chessboard distance transform values. All remaining nodes in the QMAT are WHITE and GRAY with distance value zero. For example, Fig. 1d contains the chessboard distance transform corresponding to the region given in Fig. 1a. Figs. 1e and f contain the block and tree representations, respectively, of the QMAT of Figs. 1b and c. The algorithm for constructing the QMAT from its quadtree is given in [24].

4. ALGORITHM

Construction of a quadtree from its QMAT is relatively straightforward. We perform a postorder traversal of the QMAT, and for each element of the quadtree skeleton (i.e., a BLACK node in the QMAT), say t_i, we add all elements of $S(t_i)$ to the quadtree. This is done by examining the neighbors of t_i. There are at most eight such neighbors as shown in [22]. Once a neighbor, say g, is found to be subsumed by an element of the QMAT, the same process of neighbor examination is performed for the neighbors of g in the specified direction. For example, in Fig. 1, when processing the eastern edge of node 1 of the QMAT, we add nodes 3 and 6 by expanding node F. Next, we add nodes 4 and 5, and nodes 7 and 8 by expanding nodes P and Q, respectively, as they are the eastern neighbors of 3 and 6.

The main procedure is termed QMAT_TO_QUADTREE and is invoked with a pointer to the QMAT representing the image and an integer corresponding to the log of the diameter of the image (e.g., n for a $2^n \times 2^n$ image array). QMAT_TO_QUADTREE traverses the QMAT and controls the examinations of the eight neighbors of each BLACK node which is an element of the QMAT. Note that QMAT_TO_QUADTREE is an "in place" algorithm in the sense that it overwrites the current tree (i.e., the QMAT) in the process of constructing the corresponding quadtree. An alternative algorithm could be devised which would create a copy of the QMAT as it constructs the quadtree thereby processing the two trees in parallel.

The neighbors of each node are obtained by procedures MAKE_EQUAL_ADJ_NEIGHBOR and MAKE_EQUAL_CORNER_NEIGHBOR. In particular, we require a neighbor of equal size along each of the directions. If the neighbor is of larger size, then by use of ADD_FOUR_WHITE_SONS it is decomposed into four WHITE sons as many times as is required to get the appropriate size. For example, the western neighbor of node 2 in Fig. 1e is D and is larger than 2. Therefore, we replace it by a GRAY node of four WHITE sons (i.e., 62, K, 63, and L) and return K as the appropriate neighbor. Of course, if the node is on the edge of the image in the desired direction, then no neighbor exists and NULL is returned (e.g., the northern neighbor of node 1 in Fig. 1e). Note that MAKE_EQUAL_ADJ_NEIGHBOR locates a neighboring node of equal size along directions N, E, S, and W while MAKE_EQUAL_CORNER_NEIGHBOR locates a neighboring node of equal size along directions NE, SE, SW, and NW.

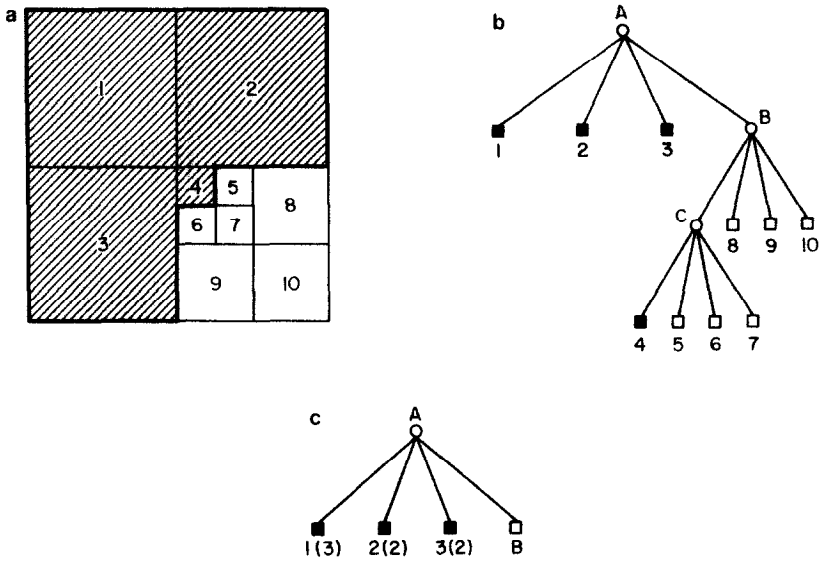

FIG. 3. An image, its quadtree, and its QMAT showing that PROPAGATE_ADJACENT need not be applied when propagating the subsumption in the eastern and southern directions for node 1: (a) Image and its block decomposition; (b) Quadtree representation of the blocks in (a); (c) QMAT of the image in (a) (radius values are within parentheses).

Once a neighboring node of equal size has been located, we must propagate the subsumption and thereby add BLACK nodes as is necessary. This may require changing WHITE nodes to BLACK or decomposing them into smaller nodes. Procedure PROPAGATE_ADJACENT propagates subsumption in the horizontal and vertical directions. It checks if the neighboring node, say Y, is subsumed entirely by the node currently being processed, say X. If Y is not, then there are three cases depending on Y's type. If Y is BLACK, then Y is left alone (e.g., when propagating subsumption in the eastern and southern directions for node 1 in Fig. 3 and encountering BLACK nodes 2 and 3, respectively). If Y is WHITE, then Y is decomposed into four WHITE sons and PROPAGATE_ADJACENT is reapplied recursively to the two sons closest to the node whose subsumption is being propagated, i.e., X. For example, node F of the QMAT in Fig. 1e is an eastern neighbor of node 1 but it is not totally subsumed by node 1 and thus it is decomposed into WHITE sons 3, 6, P, and Q. PROPAGATE_ADJACENT is now reapplied to nodes 3 and 6. The final case is when the node is GRAY which means that we recursively reapply PROPAGATE_ADJACENT to the two adjacent sons as was done for a WHITE node. When the neighboring node Y is subsumed entirely, we perform the following two steps. First, we convert Y to a BLACK node if it is WHITE or GRAY. Second, we determine if there is further subsumption in the direction being processed. For example, while processing node 1 in Fig. 1e, node 3 is subsumed by node 1 but so is part of node P. This forces us to recursively reapply PROPAGATE_ADJACENT to the eastern neighbor of 3—i.e., P.

Procedure PROPAGATE_CORNER is analogous in spirit to PROPAGATE_ADJACENT in the sense that it propagates subsumption in the corner directions (i.e., NE, SE, SW, and NW). It checks if the neighboring node, say Y, is subsumed

entirely by the node currently being processed, say X. If Y is not, then there are three cases depending on Y's type. If Y is BLACK, then Y is left alone. If Y is WHITE, then Y is decomposed into four WHITE sons and PROPAGATE_CORNER is reapplied recursively to the son closest to the node whose subsumption is being propagated, i.e., X. For example, node H of the QMAT in Fig. 1e is a southeastern neighbor of node 1 but it is not totally subsumed by node 1 and thus it is decomposed into WHITE sons 9, T, U, and V. PROPAGATE_CORNER is now reapplied to node 9. The final case is when the node is GRAY which means that we recursively reapply PROPAGATE_CORNER to the nearest son as was done for a WHITE node. When the neighboring node Y is subsumed entirely, then we perform the following two steps. First, we covert Y to a BLACK node if it is WHITE or GRAY. Second, we determine if there is further subsumption in the corner direction being processed as well as its two adjacent directions. For example, while processing node 1 in Fig. 1e, node 9 is subsumed by node 1 but so are parts of nodes T, U, and V. This forces us to recursively reapply PROPAGATE_CORNER to the southeastern neighbor of 9—i.e., V and PROPAGATE_ADJACENT to the eastern and southern neighbors of 9—i.e., T and U, respectively.

Figures 4a and b provide a snapshot of the QMAT to quadtree construction process by showing the intermediate quadtree immediately after finishing processing node 1. Note that node V is a GRAY node having sons 12, 39, 40, and 41 of which 12 is BLACK while 39, 40, and 41 are WHITE. In contrast, in Figures 1b and c node V is BLACK (and is termed 32). This is because when processing node 2 and checking its northeastern neighbor (i.e., H) for subsumption we were compelled to reapply PROPAGATE_CORNER to node V which was a GRAY node. However, as far as node 2 is concerned, the entire space spanned by node V is subsumed by node 2 and hence it was converted to a BLACK node. A similar situation can arise when PROPAGATE_ADJACENT encounters a GRAY node.

The process of recursive application of procedures PROPAGATE_ADJACENT and PROPAGATE_CORNER is guaranteed to terminate by virtue of Theorem 1 of [22] which shows that a QMAT BLACK block of size 2^s by 2^s has an upper bound of $3 \cdot 2^{(s-1)}$ for its distance transform value. Practically speaking, this means that in Fig. 1e, when checking the space subsumed by node 1 we need not examine nodes to the east, south, and southeast of nodes F, G, and H, respectively. Also note that whenever a BLACK node is generated by procedure PROPAGATE_ADJACENT or PROPAGATE_CORNER, its DIST field is set to zero so that procedure QMAT_TO_QUADTREE need not waste its time trying to propagate subsumption for it.

As an example of the application of the algorithm, consider the region given in Fig. 1a. Figure 1b is the corresponding block decomposition while Fig. 1c is its quadtree representation. Figure 1d contains the chessboard distance transform corresponding to Fig. 1b. Figures 1e and f contain the block decomposition of the QMAT and the quadtree representation of QMAT corresponding to Fig. 1b, respectively. Figure 4 shows the quadtree resulting after propagating the subsumption of node 1. All the BLACK nodes have labels ranging from 1 to 32 while the WHITE nodes have labels ranging from 33 to 68. The GRAY nodes have labels ranging between A and V. Nodes 1 and 2 are the elements of the quadtree skeleton (i.e., the BLACK nodes comprising the QMAT). The remainder of the BLACK nodes have been labeled in the order in which they were visited (and thereby created) by procedures PROPAGATE_ADJACENT and PROPAGATE_CORNER.

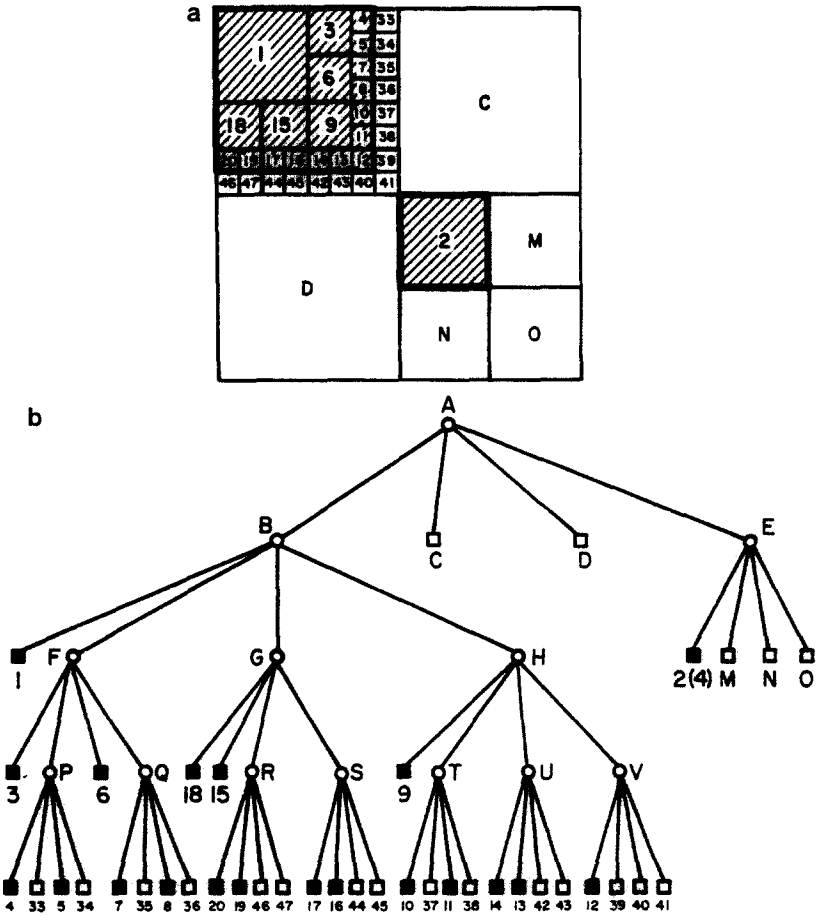

FIG. 4. Intermediate quadtree immediately after propagating the subsumption of node 1: (a) Image and its block decomposition; (b) Quadtree representation of the blocks in (a).

Note the presence of GRAY node V; BLACK node 12; and WHITE nodes 39, 40, and 41 in Fig. 4; and their replacement by BLACK node 32 in Fig. 1. This was explained earlier to be a result of checking the subsumption of node 2.

```
procedure QMAT_TO_QUADTREE(P, LEVEL);
/* Given a QMAT rooted at node P spanning a 2^LEVEL × 2^LEVEL space, find its
   corresponding quadtree. */
begin
    value node P;
    value integer LEVEL;
    node Q;
    direction D;
    quadrant I;
    if BLACK(P) then
        begin
            if DIST(P) > 2↑(LEVEL − 1) then
```

```
                begin/* Node P subsumes some of its neighbors */
                    for D in {'N', 'E', 'S', 'W'} do
                        begin
                            MAKE_EQUAL_ADJ_NEIGHBOR(P, D, Q);
                            if not NULL(Q) then
                                PROPAGATE_ADJACENT(Q, LEVEL,
                                                   DIST(P)-2↑
                                                   (LEVEL - 1), D);
                            MAKE_EQUAL_CORNER_NEIGHBOR(P, QUAD
                                                       (D, CSIDE
                                                       (D)), Q);
                            if not NULL(Q) then
                                PROPAGATE_CORNER(QLEVEL,
                                                 DIST(P)-2↑
                                                 (LEVEL - 1), D;
                        end;
                end;
            DIST(P) ← 0;
        end
    else if GRAY(P) then
        begin
            for I in {'NW', 'NE', 'SE', 'SW'} do
                QMAT_TO_QUADTREE(SON(P, I), LEVEL - 1);
        end;
end; /* WHITE nodes are left alone */


procedure MAKE_EQUAL_ADJ_NEIGHBOR(P, D, Q);
/*  Return in Q the neighbor of node P in horizontal or vertical direction D which
    is equal in size to P. If P is adjacent to the border of the image along the
    specified direction, then NULL is returned. If the neighbor is of size greater
    than P and is WHITE, then it is broken up into four WHITE quadrants in
    order to obtain the desired neighbor. If the neighbor is BLACK, then it is not
    broken up further. */
begin
    value node P;
    reference node Q;
    value direction D;
    if not NULL(FATHER(P)) and ADJ(D, SONTYPE(P)) then
        /* Find a common ancestor */
        MAKE_EQUAL_ADJ_NEIGHBOR(FATHER(P), D, Q)
    else Q ← FATHER(P);
    /* Follow the reflected path to locate the neighbor */
    if not NULL(Q) and not BLACK(Q) then
        Q ← SON(if GRAY(Q) then Q
                else ADD_FOUR_WHITE_SONS(Q),
                REFELCT(D, SONTYPE(P)));
end;
```

```
procedure MAKE_EQUAL_CORNER_NEIGHBOR(P, C, Q);
/* Return in Q the neighbor of node P in the direction of corner C of P which is
   equal in size to P. If P is adjacent to the border of the image along the
   specified direction, then NULL is returned. If the neighbor is of size greater
   than P and is WHITE, then it is broken up into four WHITE quadrants in
   order to obtain the desired neighbor. If the neighbor is BLACK, then it is not
   broken up further. */
begin
    value node P;
    reference node Q;
    value quadrant C;
    if not NULL(FATHER(P)) and SONTYPE(P) ≠ OPQUAD(C) then
        if SONTYPE(P) = C then
            MAKE_EQUAL_CORNER_NEIGHBOR(FATHER(P), C, Q)
        else MAKE_EQUAL_ADJ_NEIGHBOR(FATHER(P),
                                     COMMONSIDE
                                     (SONTYPE(P), C), Q)
    else Q ← FATHER(P);
    /* Follow the opposite path to locate the neighbor */
    if not NULL(Q) and not BLACK(Q) then
        Q ← SON(if GRAY(Q) then Q
                else ADD_FOUR_WHITE_SONS(Q),
                OPQUAD(SONTYPE(P)));
end;

node procedure ADD_FOUR_WHITE_SONS(P);
/* Return node P after converting it to a GRAY node and adding four WHITE
   sons. */
begin
    value node P;
    node Q;
    quadrant I, J;
    NODETYPE(P) ← 'GRAY';
    for I in {'NW', 'NE', 'SE', 'SW'} do
        begin
            Q ← createnode ( );
            SON(P, I) ← Q;
            FATHER(Q) ← P;
            NODETYPE(Q) ← 'WHITE';
            DIST(Q) ← 0;
            for J in {'NW', 'NE', 'SE', 'SW'} do SON(Q, J) ← NULL;
        end;
    return (P);
end;

procedure PROPAGATE_ADJACENT(P, L, T, D);
/* Node P at level L is adjacent to side D of a QMAT node whose span exceeds
   its width by T—i.e., P is subsumed or partially subsumed by that node. P or
```

```
     its appropriate sons are to be converted to BLACK nodes if they are not
     already BLACK. */
begin
     value node P;
     value integer L, T;
     value direction D;
     if 2↑L > T then
        begin /* P is too large to be totally subsumed by its adjacent QMAT
                 node */
           if BLACK(P) then return;
           if WHITE(P) then ADD_FOUR_WHITE_SONS(P);
           PROPAGATE_ADJACENT(SON(P, QUAD(OPSIDE(D),
                                          CCSIDE(D))), L-1, T, D);
           PROPAGATE_ADJACENT(SON(P, QUAD(OPSIDE(D),
                                          CSIDE(D))), L-1 T, D);
        end
     else
        begin /* P is subsumed by its adjacent QMAT node */
           if GRAY(P) then /* Account for nondisjointess of the QMAT—see
                              Fig. 4 */
              return SONS to avail (P);
           NODETYPE(P) ←'BLACK'; /* Change P to BLACK if not already
                                    so */
           if 2↑L < T then /* Propagate subsumption to neighbors of P on side
                              D */
              PROPAGATE_ADJACENT(SON(FATHER(P),
                                     REFLECT(SONTYPE(P), D)), L,
                                     T-2↑L, D);
        end;
end;

procedure PROPAGATE_CORNER(P, L, T, D);
/* Node P at level L is adjacent to corner QUAD(D, CSIDE(D)) of a QMAT
     node whose span exceeds its width by T—i.e., P is subsumed or partially
     subsumed by that node. P or its appropriate sons are to be converted to
     BLACK nodes if they are not already BLACK. */
begin
     value node P;
     value integer L, T;
     value direction D;
     if 2↑L > T then
        begin
           /* P is too large to be totally subsumed by its corner adjacent QMAT
              node */
           if BLACK(P) then return;
           if WHITE(P) then ADD_FOUR_WHITE_SONS(P);
           PROPAGATE _CORNER(SON(P, QUAD(OPSIDE(D),
                                         CCSIDE(D))), L-1, T, D);
```

```
        end
    else
        begin /* P is subsumed by its adjacent QMAT node */
            if GRAY(P) then /* Account for nondisjointness of the QMAT—see
                               Fig. 4 */
                return SONS to avail (P);
            NODETYPE(P) ← 'BLACK'; /* Change P to BLACK if not already
                                      so */
            if 2↑L < T then
                begin
                    /*Propagate subsumption to neighbors of P on its corner and
                      the two sides adjacent to the corner */
                    PROPAGATE_ADJACENT(SON(FATHER(P),
                                           QUAD(D, CCSIDE(D))), L,
                                           T-2↑L, D);
                    PROPAGATE_CORNER(SON(FATHER(P),
                                         QUAD(D, CSIDE(D))), L,
                                         T-2↑L, D);
                    PROPAGATE_ADJACENT(SON(FATHER(P),
                                           QUAD(OPSIDE(D),
                                           CSIDE(D))),
                                           L, T-2↑L, CSIDE(D)));
                end;
        end;
end;
```

5. ANALYSIS

The running time of the QMAT to quadtree algorithm, measured by the number of nodes visited, is dependent on the ultimate size of the quadtree and on the number of elements in the quadtree skeleton (i.e., BLACK nodes in the QMAT). The analysis of the execution time is quite similar to that performed for the chessboard distance computation algorithm in [15, 22]. In essence, we must propagate subsumption for each BLACK element of the QMAT. Actually, we only need to do this work for those elements of size 2^s whose chessboard distance transform value is $> 2^{(s-1)}$. Elements of the QMAT satisfying this criterion are termed *subsuming elements*. Propagating subsumption requires the examination of the eight neighbors of the subsuming element. This is achieved by procedures MAKE_EQUAL_ADJ_NEIGHBOR, PROPAGATE_ADJACENT, MAKE_EQUAL_CORNER_NEIGHBOR, and PROPAGATE_CORNER_NEIGHBOR. Each of these procedures is invoked a maximum of four times for each subsuming element. Note that the neighbor is of equal size here rather than greater than or equal size as is common in most of the other analyses of quadtree algorithms that examine neighbors [21]. The amount of work performed by these procedures depends on the value of the chessboard distance transform of the subsuming element. In the worst case, the subsuming element is found at level $n - 1$ and has a chessboard distance transform value of $2^{(n-1)} + 2^{(n-2)} - 1$. In such a case we will have to visit $2^{(n+1)} - 2$ nodes in the case of a horizontal or vertical neighbor, and $2^{(n+2)} - 2 \cdot (2n + 1)$ nodes in the

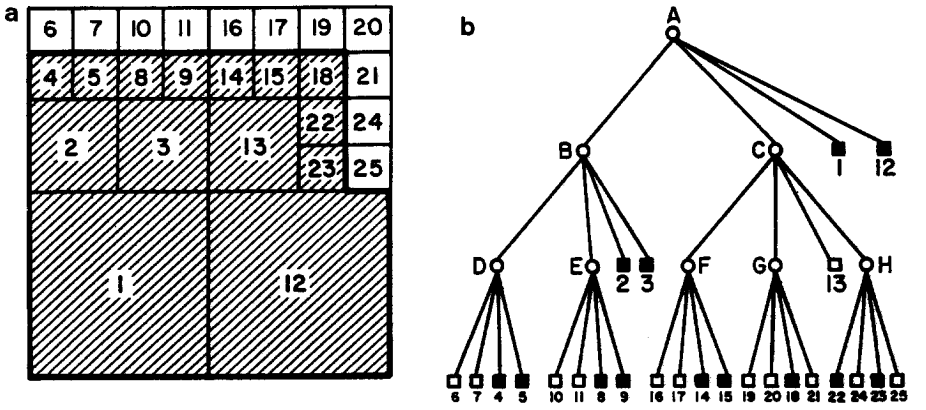

FIG. 5. Image and its quadtree showing the worst case for the N and NE neighbors when $n = 3$: (a) Image and its block decomposition; (b) Quadtree representation of the blocks in (a).

case of a diagonal neighbor. For example, consider Fig. 5, where $n = 3$ and nodes are labeled in the order in which the neighbors of node 1 and their progeny have been visited. Starting with the northern neighbor and proceeding clockwise, we visit the nodes *B*, 2, 3, *D*, 4, 5, 6, 7, *E*, 8, 9, 10, 11. Next, we visit the northeastern neighbor and its progeny in the order *C*, 13, *F*, 14, 15, 16, 17, *G*, 18, 19, 20, 21, *H*, 22, 23, 24, 25.

In the following we analyze the average execution time of the QMAT to quadtree algorithm. Our analysis assumes a $2^n \times 2^n$ image in the sense that a node is equally likely to appear in any position and level in the quadtree. This means that all configurations of adjacent nodes of varying sizes have equal probability. This is different from the more conventional notion of a random image which implies that every block at level 0 (i.e., pixel) has an equal probability of being BLACK or WHITE. Such an assumption would lead to a very low probability of any nodes corresponding to blocks of size larger than 1. Clearly, for such an image the quadtree is the wrong representation. We have used this equal configuration model in our prior work [21] and experimental results in [28] have vindicated this assumption. The amount of work performed by the combination of procedures MAKE_EQUAL_ADJ_NEIGHBOR and PROPAGATE_ADJACENT is the same as that performed by procedures GTEQUAL_ADJ_NEIGHBOR and DIST_ADJACENT in the computation of the chessboard distance transform as reported in [15, 22]. The same equivalence holds for the work performed by the combination of procedures MAKE_EQUAL_CORNER_NEIGHBOR and PROPAGATE_CORNER and procedures GTEQUAL_CORNER_NEIGHBOR and DIST_CORNER in [15, 22]. Both of these tasks have average execution times of $O(1)$. They are invoked a total of eight times for each BLACK node in the tree. We also must visit each node in the quadtree as part of the traversal. Since a quadtree with B and W BLACK and WHITE leaf nodes, respecitively, has $O(B + W)$ nodes we have proven the following theorem:

THEOREM 1. *The average execution time of the* QMAT *to quadtree algorithm is* $O(B + W)$.

6. CONCLUDING REMARKS

An algorithm has been presented for reconstructing a quadtree from its QMAT. It is useful when performing operations for which the QMAT is well suited. For example, thinning an image is quite simple when it is represented by a QMAT. The algorithm's running time was shown to have an average execution time of $O(B + W)$ where B and W correspond to the number of blocks comprising the objects and the background of the image respectively. Actually, the number of BLACK nodes (i.e., the image complexity) dominates the execution time of the algorithm. In fact, it can be said that the algorithm takes time proportional to the number BLACK nodes in the QMAT since subsumption is only propagated for these nodes. Clearly, this is a better bound than the number of BLACK blocks in the image because the size of the quadtree is an upper bound on the size of the QMAT (see [24]). Subsumption is not propagated for the remaining BLACK nodes (i.e., those that are generated during subsumption propagation) because their DIST field is set to zero.

In Section 3 the quadtree skeleton was defined in terms of properties (1), (2), and (3). In [24] an alternative definition is proposed which replaces property (3) by property (3′) below:

$$(3') \qquad \nexists t_j \in T \ni S(t_j) \subseteq \bigcup S(t_k) \qquad t_k \neq t_j.$$

Property (3) does not yield a minimal quadtree skeleton because it does not permit a quadtree block to require more than one block of T for its subsumption—i.e., one half of the block would be subsumed by one element of T and the other half would be subsumed by another element of T. For example, in the image of Fig. 6a, property (3) requires that the quadtree skeleton contains blocks 5, 14, and 15 while property (3′) only requires blocks 5 and 15 since together they subsume block 14. Property (3) was used instead of (3′) because by definition of the QMAT the tree size is unaffected by which property is used since the only difference is that the additional blocks are represented by BLACK nodes instead of WHITE nodes (e.g., node 14 in Figs. 6b and c). Furthermore, property (3) was shown to lead to a simpler QMAT creation algorithm. In [24] it is stated that use of property (3′) should lead to a faster quadtree reconstruction algorithm since the quadtree skeleton is potentially smaller. However, use of such a QMAT definition would require an additional step in procedure QMAT_TO_QUADTREE to check for merging. This would be performed at the conclusion of processing a GRAY node. Since property (3) does not permit a node to require more than one element of the quadtree skeleton for its subsumption, this merger is impossible when using our algorithm and thus need not be checked.

It should be clear that our algorithm for reconstructing the quadtree from its QMAT is not unique. We have adopted an approach similar to that used in the computation of the perimeter of an image represented by a quadtree [20]—i.e., for each BLACK node, its neighbors in the four directions N, E, S, and W were examined to see if they were GRAY or WHITE and if yes, then the appropriate values were added to the perimeter. In the discussion of the perimeter computation algorithm, it was proposed that instead of examining the four neighbors of each BLACK node, the perimeter could also be obtained by traversing the quadtree and only examining the S and E boundaries of BLACK and WHITE nodes. In the case

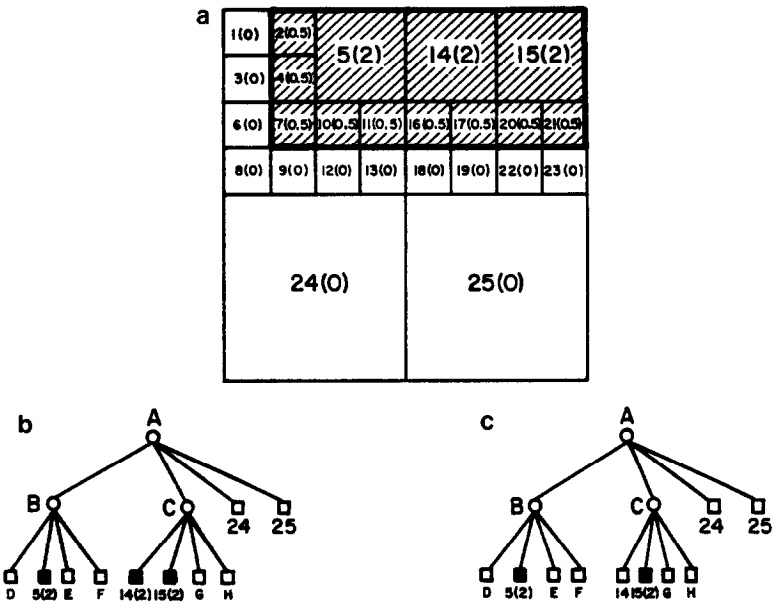

FIG. 6. An image and its corresponding QMATS using properties (3) and (3′) for the quadtree skeleton definitions. Blocks in the QMAT are shaded: (a) Image and its block decomposition (The value of the chessboard distance transform is within parentheses.); (b) QMAT of the image in (a) using property (3) (radius values are within parentheses); (c) QMAT of the image in (a) using property (3′) (radius values are within parentheses).

of the BLACK nodes we examine WHITE adjacent nodes, and in the case of WHITE nodes, we examine BLACK adjacent nodes. This removed redundant checks since each boundary has a node on each side. An analogous algorithm for the reconstruction of the quadtree from its QMAT would always examine E, SE, S, and SW neighbors in the propagation of subsumption. In the case of BLACK elements of the QMAT we would proceed in the same way as procedure QMAT_TO_QUADTREE in Section 4. However, WHITE nodes are considerably more cumbersome to process. For example, when processing the eastern side of WHITE QMAT node *D* of Fig. 1f, we must determine if there are any BLACK nodes (of any size) adjacent to its eastern side and then propagate their subsumption back to node *D*. This requires us to examine all of the adjacent BLACK nodes. The result is a QMAT reconstruction algorithm that is considerably more complex.

Our algorithm makes use of the bottom-up neighbor finding techniques of [21]. In essence, a node's neighbor in a given direction is located by ascending the tree until a nearest common ancestor is found, and then descending down the tree in search of the appropriate node. This technique has an average worst case of four nodes being visited for each neighbor that is sought. The worst case for a 2^n by 2^n image is $2 \cdot n$. An alternative method, termed top-down, makes use of a neighbor vector, containing the eight neighbors of each node in the eight directions, as an actual parameter to the tree traversal [23]. Variants of this technique are discussed in [8, 14]. It could be used in our implementation thereby obviating, in part, the need for procedures MAKE_EQUAL_ADJ_NEIGHBOR and MAKE_EQUAL_CORNER_NEIGHBOR. However, it would not remove the average worst case characterization of the

entire QMAT_TO_QUADTREE algorithm since we still must account for the average worst case of procedures PROPAGATE_ADJACENT and PROPAGATE _CORNER. Nevertheless, the procedure would be faster since the eight neighbors would be immediately accessible.

ACKNOWLEDGMENTS

I have benefitted greatly from discussions with Robert E. Webber. I would like to thank Diana Greenberg and Janet Salzman for their help in the preparation of the paper.

REFERENCES

1. H. Blum, A transformation for extracting new descriptors of shape, in *Models for the Perception of Speech and Visual Form* (W. Wathen-Dunn, Ed.), pp. 362–380, MIT Press, Cambridge, Mass., 1967.
2. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley-Interscience, New York, 1973.
3. C. R. Dyer, A. Rosenfeld, and H. Samet, Region representation: Boundary codes from quadtrees, *Comm. ACM* **23**, No. 3 (1980), 171–179.
4. C. R. Cyer, Computing the Euler number of an image from its quadtree, *Comput. Graphics Image Process.* **13**, No. 3 (1980), 270–276.
5. G. M. Hunter and K. Steiglitz, Operations on images using quad trees, *IEEE Trans. Pattern Anal. Mach. Intell.* **1**, No. 2 (1979), 145–153.
6. G. M. Hunter and K. Steiglitz, Linear transformation of pictures represented by quad trees, *Comput. Graphics Image Process.* **10**, No. 3 (1979), 289–296.
7. C. L. Jackins and S. L. Tanimoto, Oct-trees and their use in representing three-dimensional objects, *Comput. Graphics Image Process.* **14**, No. 3 (1980), 249–270.
8. C. L. Jackins and S. L. Tanimoto, Quad-trees, oct-trees, and k-trees—A generalized approach to recursive decomposition of Euclidean space, *IEEE Trans. Pattern Anal. Mach. Intell.* **5**, No. 5 (1983), 533–539.
9. A. Klinger, Patterns and search statistics, in *Optimizing Methods in Statistics* pp. 303–337, (J. S. Rustagi, Ed.), Academic Press, New York, 1971.
10. A. Klinger and M. L. Rhodes, Organization and access of image data by areas, *IEEE Trans. Pattern Anal. Mach. Intell.* **1**, No. 1 (1979), 50–60.
11. M. Li, W. I. Grosky, and R. Jain, Normalized quadtrees with respect to translations, *Comput. Graphics Image Process.* **20**, No. 1 (1982), 72–81.
12. D. Meagher, Geometric modeling using octree encoding, *Comput. Graphics Image Process.* **19**, No. 2 (1982), 129–147.
13. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, New York, 1976.
14. A. Rosenfeld, H. Samet, C. Shaffer, and R. E. Webber, *Application of Hierarchical Data Structures to Geographical Information Systems*, Computer Science TR-1197, University of Maryland, College Park, Md., June 1982.
15. H. Samet, *A Distance Transform for Images Represented by Quadtrees*, Computer Science TR-780, University of Maryland, College Park, Md., July 1979.
16. H. Samet, Region representation: Quadtrees from boundary codes, *Comm. ACM* **23**, No. 3 (1980), 163–170.
17. H. Samet, Region representation: Quadtrees from binary arrays, *Comput. Graphics Image Process.*, **13**, No. 1 (1980), 88–93.
18. H. Samet, An algorithm for converting rasters to quadtrees, *IEEE Trans. Pattern Anal. Mach. Intell.* **3**, No. 1 (1981), 93–95.
19. H. Samet, Connected component labeling using quadtrees, *J. Assoc. Comput. Mach.* **28**, No. 3 (1981), 487–501.
20. H. Samet, Computing perimeters of images represented by quadtrees, *IEEE Trans. Pattern Anal. Mach. Intell.* **3**, No. 6 (1981), 683–687.
21. H. Samet, Neighbor finding techniques for images represented by quadtrees, *Comput. Graphics Image Process.* **18**, No. 1 (1982), 37–57.

22. H. Samet, Distance transform for images represented by quadtrees, *IEEE Trans. Pattern Anal. Mach. Intell.* **4**, No. 3 (1982), 298–303.
23. H. Samet, A top-down quadtree traversal algorithm, *IEEE Trans. Pattern Anal. Math. Intell.*, in press; Computer Science TR-1237, University of Maryland, College Park, Md., December 1982.
24. H. Samet, A quadtree medial axis transform, *Comm. ACM* **26**, No. 9 (1983), 680–693.
25. H. Samet, A. Rosenfeld, C. Shaffer, and R. E. Webber, Quadtree region representation in cartography: Experimental results, *IEEE Trans. Systems Man Cybernet.* **13**, No. 6 (1983), 1148–1154.
26. H. Samet, The quadtree and related hierarchical data structures, *ACM Comput. Surveys* **16**, No. 2 (1984).
27. H. Samet, Algorithms for the conversion of quadtrees to rasters, *Comput. Vision Graphics Image Process.* **26**, No. 1 (1984), 1–16.
28. H. Samet and C. A. Shaffer, A model for the analysis of neighbor finding in pointer-based quadtrees, Computer Science TR-1432, University of Maryland, College Park, Md., August 1984.
29. M. Shneier, Calculations of geometric properties using quadtrees, *Comput. Graphics Image Process.* **16**, No. 3 (1981), 296–302.
30. M. Yau and S. N. Srihari, A hierarchical data structure for multidimensional digital images, *Comm. ACM* **26**, No. 7 (1983), 504–515.