Graphics and Image Processing

J.D. Foley Editor

Region Representation: Boundary Codes from Quadtrees

Charles R. Dyer, Azriel Rosenfeld, and Hanan Samet
University of Maryland

There has been recent interest in the use of quadtrees to represent regions in an image. It thus becomes desirable to develop efficient methods of conversion between quadtrees and other types of region representations. This paper presents an algorithm for converting from quadtrees to a simple class of boundary codes. The algorithm is shown to have an execution time proportional to the perimeter of the region.

Key Words and Phrases: quadtrees, chain codes, regions, borders, data structures

CR Categories: 3.63, 8.2

This work was sponsored by the Defense Advanced Research Projects Agency and the U.S. Army Night Vision Laboratory under contract no. DAAG-53-76C-0138 (DARPA Order 3206).

Authors' present address. C.R. Dyer, Department of Information Engineering, University of Illinois at Chicago Circle, Chicago, IL 60680; A. Rosenfeld, Computer Science Center, and H. Samet, Computer Science Department, University of Maryland, College Park, MD 20742.

1. Introduction

Region representation plays a key role in image and scene analysis, computer cartography, and computer graphics. There are a variety of approaches to representing regions, based on their boundaries or their "skeletons"; some of these are reviewed in the following paragraphs. Recently, a tree representation has been proposed which offers a number of advantages; it is also described here. Since each type of representation has its own advantages, it becomes desirable to develop efficient methods of converting from one representation to another. Sections 2–6 present and analyze an algorithm for converting the tree representation into a simple type of boundary representation. Section 7 briefly considers several related problems.

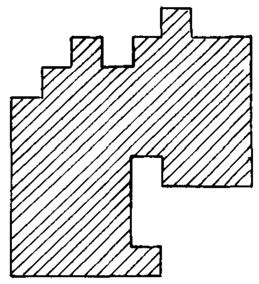
We assume in what follows that a region is a simply connected subset of a 2^n-by-2^n array, which we regard as being made up of unit-square "pixels." (The treatment of regions that have holes will be discussed in Section 7.) The boundary of such a region can thus be specified, relative to a given starting point, as a sequence of unit vectors in the principal directions. We can represent the direction by numbers, e.g. let i, an integer quantity ranging from 0 to 3, represent $90° * i$. For example, the direction sequence for the boundary of the region in Figure 1(a), moving clockwise starting from the left of the uppermost border points, is

$$0\ 3\ 0^2\ 3^5\ 2^3\ 1\ 2\ 3^3\ 0\ 3\ 2^5\ 1^6\ 0\ 1\ 0\ 1\ 0\ 3\ 0\ 1\ 0\ 1.$$

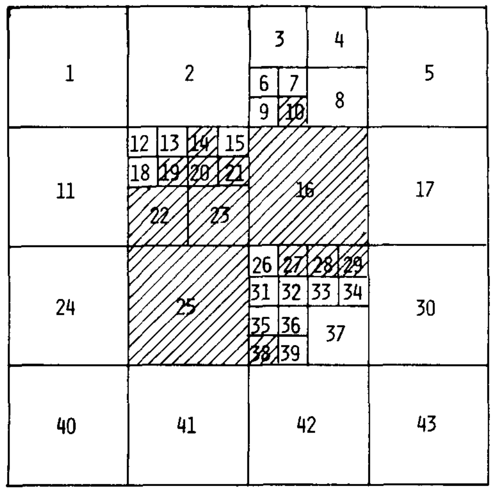
This type of boundary representation is called a *chain code*. Generalized chain codes, involving more than four directions, can also be used. Chain codes provide a very compact region representation, and make it easy to detect features of the region boundary, such as sharp turns ("corners") or concavities. On the other hand, it is harder to determine properties such as elongatedness from a chain code, and it is also difficult to perform operations such as union and intersection on regions represented by chain codes. A general introduction to chain codes and their uses can be found in [4].

Another class of region representations involves various types of maximal "blocks" that are contained in a given region. For example, we can represent a region R as a linked list of the runs (of pixels) in which R meets the successive rows of the array [15]. Here each "block" is a 1-by-m rectangle, where m is the run length; the runs are the largest such blocks that R contains, and R is determined by specifying the initial points (or centers) and lengths of the runs. Alternatively, we can represent R by the set of maximal square blocks (or blocks of any other desired shape) that it contains; here R is determined by specifying the centers and radii of these blocks. This representation is called the *medial axis transformation*, or MAT [2, 14]. It is somewhat less compact than the chain code [12], but it has advantages with respect to performing union and intersection operations or detecting prop-

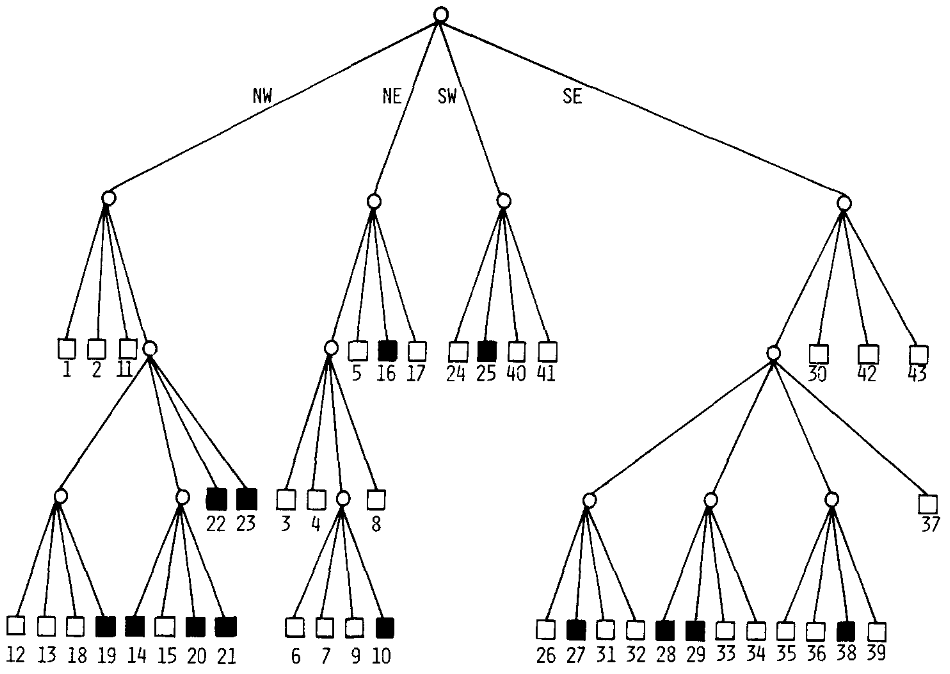
Fig. 1. A Region, Its Maximal Blocks, and the Corresponding Quadtree. Blocks in the region are shaded, background blocks are blank.



(a) Region.



(b) Block decomposition of the region in (a).



(c) Quadtree representation of the blocks in (b).

erties such as elongatedness (in terms of the smallness of the radii relative to the number of centers).

There has been recent interest in an approach to region representation based on successive subdivision of the array into quadrants. If the region does not cover the entire array, we subdivide the array, and repeat this process for each quadrant, each subquadrant, . . . , as long as necessary, until we obtain blocks (possibly single pixels) that are entirely contained in the region or entirely disjoint from it. The resulting blocks for the region of Figure 1(a) are shown in Figure 1(b). This process can be represented by a tree of degree 4 (for brevity: a *quadtree*) in which the entire array is the root node, the four sons of a node are its quadrants, and the leaf nodes correspond to those blocks for which no further subdivision is necessary.[1] The quadtree representation for Figure 1(b) is shown in Figure 1(c). Note that here again we are representing the region as a union of maximal blocks, but this time the blocks must have standard sizes and positions (powers of 2). Since the array was assumed to be 2^n-by-2^n, the tree height is at most n. This method of region representation was proposed by Klinger [1, 9]; it has also been used for image representation (e.g., [5, 10, 13, 19, 20]). It is relatively compact [9], and is also well-suited to operations such as union and intersection [6, 7], and to detecting various region properties [6–9]. A

[1] The quadtree region representation described here should not be confused with the quadtree representation of two-dimensional point data introduced by Finkel and Bentley [3].

Fig. 2(a). Relationship between a block's four quadrants and its chain code representation.

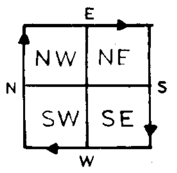
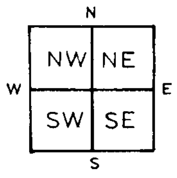


Fig. 2(b). Relationship between a block's four quadrants and its boundary.



recent Ph.D. thesis by Hunter [6–8] in the domain of computer graphics develops a variety of algorithms for the manipulation of quadtree region representation.

Since the quadtree and border representations have different computational advantages, it is of interest to develop methods of converting from one representation to the other. We shall now present an algorithm for deriving a clockwise boundary code from the quadtree representation of a given region.

2. Definitions and Notation

Let each node in a quadtree be stored as a record containing six fields. The first five fields contain pointers to the node's father and its four sons, labeled NW, NE, SW, and SE. Given a node P and a son I, these fields are referenced as FATHER(P) and SON(P, I) respectively. At times it is preferable to use the function SONTYPE(P) where SONTYPE(P) = Q iff SON(FATHER(P), Q) = P. The sixth field, named NODETYPE, describes the contents of the block of the region which the node represents, i.e., WHITE if the block contains no pixels in the region, BLACK if the block contains only pixels in the region, and GRAY if it contains pixels of both types. We often examine the information in this field by use of the predicates WHITE, BLACK, and GRAY respectively.

Let the four sides of a node's block be called its N, E, S, and W sides. Two nodes are said to be adjacent along the northern side of the first, for example, if the pair of blocks represented by these nodes touch along that side (not just at a corner). The sides of a node's block are also termed its boundaries, and at times we speak of them as if they are directions (e.g., in Figure 1, node 23 is adjacent to the eastern side of node 22; alternatively, we say that node 23 is node 22's neighbor in the eastern direction). The four directions that can be represented by a four-direction chain code are also labeled N, E, S, W. We use a clockwise chain code, which means that the region is always to the right of the boundary represented by the code. The function LINK(T) yields the chain code direction associated with side T. Using the coding described in Section 1, we have that LINK('N') = 0, LINK('E') = 3, LINK('S') = 2, and LINK('W') = 1. Figure 2(a) shows the relationship between quadrants of a node and directions of a chain code, and Figure 2(b) shows the relationship between quadrants of a node and its boundaries.

There are a number of functions and predicates which are useful in describing the relationship between nodes, quadrants, and boundaries. CSIDE(T) is the node which is adjacent to side T in the clockwise direction, e.g., CSIDE('N') = 'E'. Similarly, define CCSIDE(T) to be the side which is adjacent in the counterclockwise direction to T, and OPSIDE(T) as the side opposite to T. For example, CCSIDE('N') = 'W' and OPSIDE('N') = 'S'. The predicate ADJ(T, K) is true if and only if quadrant K is adjacent to side T of the node's block, e.g., ADJ('N', 'NW') = TRUE. REFLECT (T, K) yields the SONTYPE value of the block of equal size that is adjacent to side T of a block having SONTYPE value K; e.g., REFLECT('W', 'NE') = 'NW', and REFLECT('N', 'NW') = 'SW'. QUAD(T, U) is the quadrant that touches the corner formed by sides T and U (if T and U are opposite sides, then QUAD(T, U) is undefined), e.g., QUAD('N', 'E') = 'NE'.

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.

3. The Boundary Following Algorithm

Before tracing the boundary we must determine an initial (BLACK, WHITE) node pair. This is achieved by starting at the root and descending the quadtree in the priority order NW, NE, SW, SE avoiding WHITE nodes until a BLACK node P is reached. This node cannot have a BLACK node adjacent to its northern side because we always descend to northern sons unless both northern sons are WHITE. Assuming that the region does not reach the border of the corresponding image, we see that the node Q that is adjacent to P's northern side and touching P's northwestern corner must be WHITE. The previous assumption can be made a reality by surrounding the image with WHITE blocks as in Figure 3. At this point nodes P and Q form an initial (BLACK, WHITE) node pair.

Given an arbitrary pair of adjacent BLACK and WHITE nodes, P and Q respectively, we first output the chain link associated with that part of P's border which is adjacent to Q. Next, we must determine the (BLACK, WHITE) node pair that defines the subsequent link in the chain as we traverse the boundary in the clockwise direction. To find the new pair, we must consider the three possible relative positions of *P* and *Q* with respect to the direction of traversal—either *P* extends past *Q* (Figure 4(a)), or *Q* extends past *P* (Figure 4(b)), or *P* and *Q* end at the same point (Figure 4(c)). Note that in Figure 4, *P* is always to the north of *Q*; however, it can also be to the east, south, or west.

In order to determine the next pair, we first find the adjacent nodes *X* and *Y* as shown in Figure 4. Given the NODETYPE information for these nodes, we obtain the next pair, for the three cases illustrated in Figure 4, in the following manner. Note that the relationships involving Figure 4(c) are the same whether or not *P* is smaller than *Q*. Also whenever we speak of a side of a node, we mean the entire width of the side.

(1) *Figure 4(a)*: In this case *X* cannot extend beyond *P* although it may be larger than *Q*.

(a) If *X* is WHITE, then the new pair is (*P*, *X*), and the boundary does not turn. The new link is the north side of *X* (e.g., Figure 5(a)).

(b) If *X* is BLACK, then the new pair is (*X*, *Q*) and the boundary turns left. If *X* is larger than *Q*, then the new link is the west side of *Q* (e.g., Figure 5(b)); otherwise, it is the east side of *X*.

(2) *Figure 4(b)*: In this case *X* cannot extend beyond *Q*, although it may be larger than *P*.

(a) If *X* is WHITE, then the new pair is (*P*, *X*), and the boundary turns right. If *P* is no larger than *X*, then the new link is the west side of *P*; otherwise, it is the east side of *X* (e.g., Figure 5(c)).

(b) If *X* is BLACK, then the new pair is (*X*, *Q*) and the boundary does not turn. The new link is the south side of *X* (e.g., Figure 5(d)).

(3) *Figure 4(c)*: Assume that the region is 4-connected so that blocks touching only at a corner are not regarded as adjacent.

(a) If both *X* and *Y* are BLACK (they need not be distinct nodes), then the new pair is (*Y*, *Q*), and the boundary turns left. If *X* and *Y* are distinct and *Y* is no bigger than *Q*, then the new link is the east side of *Y* (e.g., Figure 5(i)); otherwise (i.e., *Y* is bigger than *Q*), the new link is the west side of *Q*. Note that if *X* and *Y* are the same node (e.g., Figure 5(f)), then *Y* must extend at least to the end of *Q* because neighboring nodes cannot properly overlap.

(b) If *X* is BLACK and *Y* is WHITE, then the new pair is (*X*, *Y*) and the boundary does not turn. If *X* is not bigger than *Y*, then the new link is the south side of *X*; otherwise, it is the north side of *Y* (e.g., Figure 5(h)).

(c) If *X* is WHITE, then the new pair is (*P*, *X*), and the boundary turns right regardless of the type of node *Y* by virtue of the 4-connected property. If *P* is no bigger than *X*, then the new link is the west side of *P* (e.g., Figure 5(e)); otherwise, it is the east side of *X* (e.g., Figure 5(g)).

Fig. 3. An image totally surrounded by background.

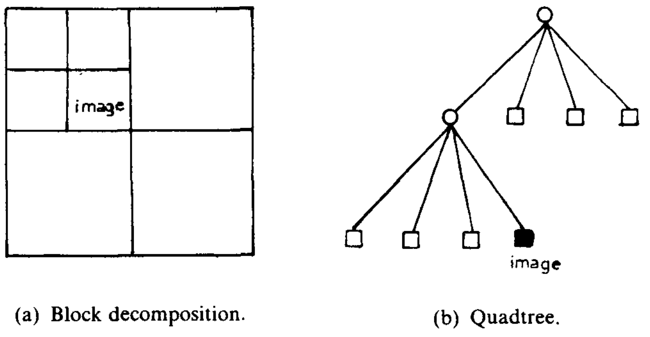


Fig. 4. Possible overlap relationships between the (BLACK, WHITE) adjacent node pair (*P*, *Q*). The heavy arrow indicates the boundary segment just output.

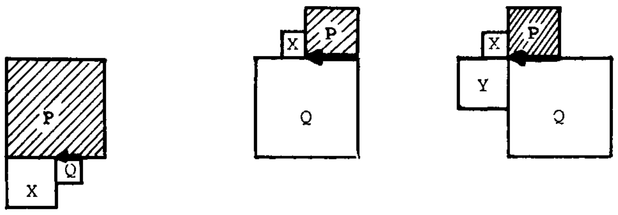
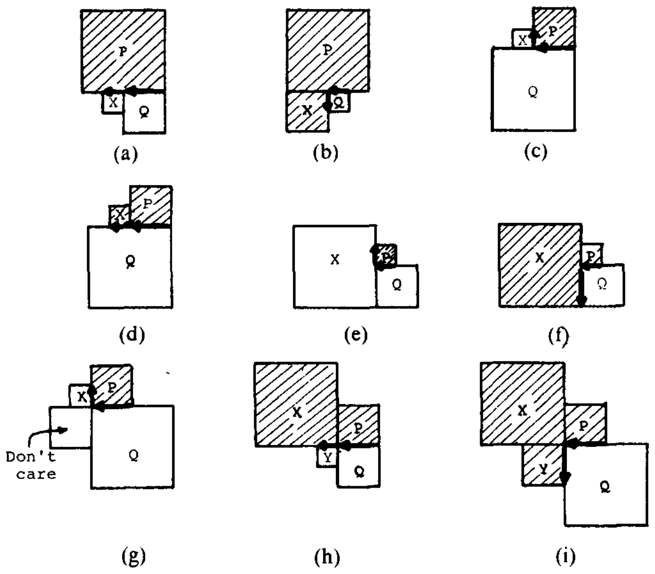


Fig. 5. Possible configurations of *P*, *Q* and their neighbor blocks in determining the next (BLACK, WHITE) pair. Arrows indicate the boundary segments associated with the old and new pairs.



It should be clear that we could specify a similar set of rules if we considered the region to be 8-connected.

4. Formal Statement of the Algorithm

The following ALGOL-like procedures [11] specify the complete boundary following algorithm. The main procedure is termed CHAINCODE and is invoked with a pointer to the root of the quadtree representing the image and an integer corresponding to the log of the diameter of the image (e.g., n for a 2^n by 2^n image array). It finds the initial (BLACK, WHITE) pair of nodes and then invokes procedure NEXT_LINK which controls the process of tracing the boundary and outputting the appropriate links of the chain code. Procedure CORNER_ADJ_NEIGHBOR is used in conjunction with procedure GTEQUAL_ADJ_NEIGHBOR to find the adjacent nodes (e.g., X and Y in Figures 3 and 4). Boolean procedure ALIGNED aids in determining which of the three cases illustrated in Figure 3 is applicable to the current (BLACK, WHITE) pair of nodes. MARK and MARKED aid in the detection of the end of the tracing process.

```
procedure CHAINCODE(ROOT, LEVEL);
/* Find the chaincode corresponding to the quadtree rooted at ROOT
   representing a 2 ↑ LEVEL by 2 ↑ LEVEL image */
begin
    value node ROOT;
    value integer LEVEL;
    node P, Q;
    integer LP, LQ;
    P ← ROOT;
    LP ← LEVEL; /*LEVEL and LP are used to determine the
       length of sides of blocks */
    while GRAY(P) do
        /* Find a BLACK node P which is on the region boundary
           and has no BLACK nodes adjacent to its north side. Q, a
           WHITE node, is the western-most of P's northern neigh-
           bors. The pair (P, Q) defines the initial chain segment */
        begin
            LP ← LP − 1;
            P ← SON(P, if not WHITE(SON(P, 'NW')) then 'NW'
                       else if not WHITE(SON(P, 'NE')) then
                         'NE'
                       else if not WHITE(SON(P, 'SW')) then
                         'SW'
                       else 'SE');
        end;
    LQ ← LP;
    CORNER_ADJ_NEIGHBOR(P, 'N', 'NW', Q, LQ);
    NEXT_LINK(P, LP, Q, LQ, 'N'); /*Trace the boundary*/
end;

procedure NEXT_LINK(P, LP, Q, LQ, D);
/* Given BLACK node P at level LP and WHITE node Q at level Q
   that are adjacent along side D of P, output the corresponding chain
   code description, and determine the next pair of adjacent BLACK
   and WHITE nodes */
begin
    value node P, Q;
    value integer LP, LQ;
    value side D;
    node X, Y;
    integer I, LX, LY;
    if MARKED(P, Q) then HALT /* Done */
    else MARK(P, Q);
    for I ← step 1 until 2 ↑ MIN(LP, LQ) do PRINT(LINK(D));
    /* Determine the next pair of adjacent BLACK and WHITE
       nodes */
    if ALIGNED(P, LP, Q, LQ, CSIDE(D)) then
        /* P and Q are aligned along direction CSIDE(D) */
        begin
            LX ← LP;
            CORNER_ADJ_NEIGHBOR(P, CSIDE(D),
                                QUAD(CSIDE(D), D), X, LX);
            if WHITE(X) then /* Figures 5(e) and 5(g) */
                NEXT_LINK(P, LP, X, LX, CSIDE(D))
            else
                begin
                    LY ← LQ;
                    CORNER_ADJ_NEIGHBOR(
                      Q, CSIDE(D),
                      QUAD(CSIDE(D), OPSIDE(D)),
                      Y, LY);
                    if BLACK(Y) then /* Figures 5(f) and 5(i) */
                        NEXT_LINK(Y, LY, Q, LQ,
                                  CCSIDE(D))
                    else NEXT_LINK(X, LX, Y, LY, D);
                        /* Figure 5(h) */
                end;
        end
    else if LP > LQ then
      /* BLACK extends farther than WHITE */
        begin
            LX ← LQ;
            CORNER_ADJ_NEIGHBOR(
              Q, CSIDE(D),
              QUAD(CSIDE(D), OPSIDE(D)),
              X, LX);
            if WHITE(X) then NEXT_LINK(P, LP, X, LX, D)
                /* Figure 5(a) */
            else NEXT_LINK(X, LX, Q, LQ, CCSIDE(D));
                /* Figure 5(b) */
        end
    else /* WHITE extends farther than BLACK */
        begin
            LX ← LP;
            CORNER_ADJ_NEIGHBOR(
              P, CSIDE(D),
              QUAD(CCSIDE(D), D), X, LX);
            if WHITE(X) then /* Figure 5(c) */
                NEXT_LINK(P, LP, X, LX, CSIDE(D))
            else NEXT_LINK(X, LX, Q, LQ, D);
                /* Figure 5(d) */
        end;
end;

Boolean procedure ALIGNED(P, LP, Q, LQ, D);
/* Given two nodes P and Q, at levels LP and LQ respectively, which
   are adjacent along side CCSIDE(D) of node P, determine whether
   either of P or Q extends farther in direction D than the other (return
   FALSE), or their two sides in direction D are aligned (return TRUE)
   */
begin
    value node P, Q;
    value integer LP, LQ;
    value direction D;
    node R;
    integer I;
    if LP = LQ then return (TRUE)
    else if LP > LQ then R ← Q
    else R ← P;
    /* The smaller of the two nodes cannot extend farther than the
       other because this would imply that P and Q properly overlap,
       which is impossible. At best, the smaller node can be aligned
       with the other node, and this occurs if and only if the smaller
       node is adjacent to the extreme side in direction D of the nearest
       common ancestor of nodes P and Q */
    for I ← 1 step 1 until ABS(LP − LQ) do
        begin
            if not ADJ(D, SONTYPE(R)) then return(FALSE)
            else R ← FATHER(R);
        end;
    return(TRUE);
end;
```

```
procedure CORNER_ADJ_NEIGHBOR(P, D, C, Q, L);
/* Return in Q the neighbor of node P in horizontal or vertical
   direction D which is adjacent to corner C of node P. L denotes the
   level of the tree at which node P is initially found and the level of
   the tree at which node Q is ultimately found */
begin
    value node P;
    value direction D;
    value quadrant C;
    reference node Q;
    reference integer L;
    GTEQUAL_ADJ_NEIGHBOR(P, D, Q, L);
    while GRAY(Q) do
        begin
            Q ← SON(Q, REFLECT(D, C));
            L ← L − 1;
        end;
end;

procedure GTEQUAL_ADJ_NEIGHBOR(P, D, Q, L);
/* Return in Q the neighbor of node P in horizontal or vertical
   direction D which is greater than or equal in size to P. If such a node
   does not exist, then a GRAY node of equal size is returned. If this
   is also impossible, then the node is adjacent to the border, and
   NULL is returned. L denotes the level of the tree at which node P
   is initially found and the level of the tree at which node Q is
   ultimately found */
begin
    value node P;
    value direction D;
    reference node Q;
    reference integer L;
    L ← L + 1;
    if (not NULL (FATHER(P))) and ADJ(D, SONTYPE(P)) then
        /* Find a common ancestor */
        GTEQUAL_ADJ_NEIGHBOR(FATHER(P), D, Q, L)
    else Q ← FATHER(P);
    /* Follow the reflected path to locate the neighbor */
    if not NULL (Q) and GRAY (Q) then
        begin
            Q ← SON(Q, REFLECT(D, SONTYPE(P)));
            L ← L − 1;
        end;
end;
```

5. Example

Let us consider the quadtree shown in Figure 1. Notice that this quadtree has 57 nodes whereas an array representation would have required a 16 by 16 (=256-cell) logical array. Procedure CHAINCODE finds the initial pair (19, 13) and then invokes procedure NEXT_LINK to follow the boundary starting at this point. Table I specifies the arguments of NEXT_LINK at each recursive call along with the link that is output as a result of this pairing.

6. Analysis

The speed of our boundary coding algorithm is determined by procedure NEXT_LINK which is called for each boundary segment associated with a (BLACK, WHITE) adjacent node pair. Recall that NEXT_LINK has two parts. First, it must output the chain code description for each boundary segment. The time required to output the individual links is proportional to the region's perimeter, where the perimeter is defined to be the number of unit-square pixels on the region's boundary (not the number of BLACK nodes which are adjacent to WHITE nodes). Second, it must determine the next (BLACK, WHITE) adjacent node pair. The time required to do this is dependent on the execution time of procedures ALIGNED and CORNER_ADJ_NEIGHBOR. Given a pair (*P*, *Q*) of adjacent (BLACK, WHITE) nodes respectively, procedure ALIGNED is called once and procedure CORNER_ADJ_NEIGHBOR is called at most two times (once for Figures 4(a) and 4(b) and twice for Figure 4(c)). The amount of time required by these procedures is proportional to the number of nodes that must be visited and depends on their relative positions in the quadtree. Note that the number of times that the determination of the next (BLACK, WHITE) adjacency must be done is also bounded by the region's perimeter.

The average execution time of procedures CORNER_ADJ_NEIGHBOR and ALIGNED is analyzed below. However, it is appropriate to comment on our notion of average. We assume a random image in the sense that a node is equally likely to appear in any position and level in the quadtree. This means that we

Table I. Sequence of calls of NEXT_LINK for the quadtree in Figure 1. The exponent on the output indicates the number of unit length links associated with the boundary specified by the given (*P*, *Q*) pair.

BLACK node *P*	WHITE node *Q*	Side *T*	Link output
19	13	N	0^1
14	13	W	1^1
14	2	N	0^1
14	15	E	3^1
21	15	N	0^1
16	15	W	1^1
16	9	N	0^1
10	9	W	1^1
10	7	N	0^1
10	8	E	3^1
16	8	N	0^2
16	17	E	3^4
29	30	E	3^1
29	34	S	2^1
28	33	S	2^1
27	32	S	2^1
27	26	W	1^1
16	26	S	2^1
25	26	E	3^1
25	31	E	3^1
25	35	E	3^1
38	35	N	0^1
38	39	E	3^1
38	42	S	2^1
25	41	S	2^4
25	24	W	1^4
22	11	W	1^2
22	18	N	0^1
19	18	W	1^1
19	13	N	—

Fig. 6. Illustration of nearest neighbor computation.

1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63
8	16	24	32	40	48	56	64

assume that all neighbor pairs (i.e., 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.

THEOREM 1. *The average number of nodes visited by CORNER_ADJ_NEIGHBOR is bounded by 14/3.*

PROOF. Given a node P at level i, and a direction D toward corner C of P, there are

$$2^{n-i} \sum_{j=i+1}^{n} 2^{n-j} \cdot j$$

possible neighbor pairs where j corresponds to the level at which is located the nearest common ancestor of the members of the neighbor pair. The 2^{n-i} factor is a direct result of there being 2^{n-i} possible rows (or columns) containing nodes at level i. For each such row, 2^0 neighbor pairs have their nearest common ancestor at level n, 2^1 at level $n-1, \ldots,$ and 2^{n-i-1} at level $i+1$. For example, in the 2^3 by 2^3 grid of Figure 6, nodes corresponding to blocks 25 to 32 have eastern neighbors 33 to 40 respectively and nearest common ancestors at level 3; nodes corresponding to blocks 9 to 16 and 41 to 48 have eastern neighbors 17 to 24 and 49 to 55 respectively and nearest common ancestors at level 2; nodes corresponding to blocks 1 to 8, 17 to 24, 33 to 40, and 49 to 55 have eastern neighbors 9 to 16, 25 to 32, 41 to 48, and 57 to 64 respectively and nearest common ancestors at level 1. For a node at level i, a direction D toward corner C, and a nearest common ancestor at level j, we have possible neighbor pairs having the initial node at level i and the neighboring node at levels $j-1, j-2, \ldots, i+1, i, i-1, \ldots, 2, 1, 0$—i.e., j possible neighbor pairs. Thus for a node at level i, the number of node pairs that will be visited in the process of locating a neighbor at level k with a nearest common ancestor at level j is $(j-i+j-k)$. This is obtained by observing that the nearest common ancestor is at a distance of $j-i$. Therefore, the average number of nodes visited by CORNER_ADJ_NEIGHBOR is

$$\frac{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-i} \cdot 2^{n-j} \cdot \sum_{k=0}^{j-1} (j-i+j-k)}{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-i} \cdot 2^{n-j} \cdot j} \tag{1}$$

The numerator of (1) can be simplified to yield

$$\sum_{i=0}^{n-1} \sum_{j=1}^{n-i} 2^{2n-2i-j-1} \cdot (3j^2 + j + 4ji + i^2 + i) \tag{2}$$

Making use of the following identities in (2) leads to (9)

$$\sum_{j=0}^{n} \frac{1}{2^j} = 2 \cdot \left(1 - \frac{1}{2^{n+1}}\right) \tag{3}$$

$$\sum_{j=0}^{n} \frac{j}{2^j} = 2 - \frac{n+2}{2^n} \tag{4}$$

$$\sum_{j=0}^{n} \frac{j^2}{2^j} = 6 - \frac{n^2+4n+6}{2^n} \tag{5}$$

$$\sum_{j=0}^{n} \frac{1}{2^{2j}} = \frac{1}{3}\left(4 - \frac{1}{2^{2n}}\right) \tag{6}$$

$$\sum_{j=0}^{n} \frac{j}{2^{2j}} = \frac{1}{9}\left(4 - \frac{3n+4}{2^{2n}}\right) \tag{7}$$

$$\sum_{j=0}^{n} \frac{j^2}{2^{2j}} = \frac{1}{27}\left(20 - \frac{9n^2+24n+20}{2^{2n}}\right) \tag{8}$$

$$\tfrac{98}{27} \cdot 2^{2n+2} - (3n^2 + 11n + 10) \cdot 2^n - \tfrac{1}{3}n^2 - \tfrac{11}{9}n - \tfrac{122}{27} \tag{9}$$

The denominator of (1) can be manipulated in a similar manner to yield

$$\tfrac{7}{9} \cdot 2^{2n+2} - (n+2) \cdot 2^{n+1} + \tfrac{2}{3}n + \tfrac{8}{9} \tag{10}$$

Substituting (9) and (10) into (1) results in

$$\frac{14}{3} - \frac{(27n^2+15n-78) \cdot 2^n + 3n^2 + 39n + 78}{7 \cdot 2^{2n+2} - 9 \cdot (n+2) \cdot 2^{n+1} + 6n + 8}$$

$$\sim \frac{14}{3} \text{ as } n \text{ gets large}$$

$$\leq \frac{14}{3} \qquad \square$$

Procedure ALIGNED takes time proportional to the difference in the levels of the two neighboring nodes for whom this relationship is to be verified. In other words, given nodes P and Q at levels LP and LQ respectively, we must visit at most $|LP - LQ|$ nodes. For a rationale for this bound see the comment in the formal statement of procedure ALIGNED in Section 4. In fact, at times we can use information about the spatial relationship between P and Q so that ALIGNED need not be invoked for each call to NEXT_LINK. For example, having determined that P extends beyond Q and that X is smaller in size than Q (this can be detected while searching for X in CORNER_ADJ_NEIGHBOR), then the new direction for the chain code is as illustrated in Figures 5(a) and 5(b) for X being WHITE or BLACK respectively (disregard the fact that X is larger than Q in Figure 5(b)). A similar result is obtained when Q extends beyond P. It should be clear that the ALIGNED relationship is false in these cases. This can be seen by observing that whenever two adjacent nodes (in the vertical, horizontal, or diagonal directions), say P and Q, correspond to blocks of size SP and SQ respectively, such that $SP > SQ$, then there exist additional nodes of size less than or equal to SQ adjacent to the opposite side of Q. This is a direct consequence of the property of the quadtree that a node, say P, corresponding to a block of size SP cannot be adjacent to node Q and R corresponding to blocks of size SQ and SR respectively on opposite sides of P where both SQ and SR are greater than SP. Using our model of a random image, we have:

THEOREM 2. *The average number of nodes visited by ALIGNED is bounded by 19/21.*

PROOF. Given a node P at level i, and a direction D, using similar reasoning as in Theorem 1, there are

$$2^{n-i} \sum_{j=i+1}^{n} 2^{n-j} \cdot j$$

possible neighbor pairs where j corresponds to the level at which the nearest common ancestor of the members of the neighbor pair is located. Given i and j as defined above, we have possible neighbor pairs having an initial node at level i and a neighboring node at levels $j-1$, $j-2, \ldots, i+1, i, i-1, \ldots, 2, 1, 0$—i.e., j possible neighbor pairs. For a node at level i and a neighbor at level k, at most $|i-k|$ nodes must be visited in determining the aligned relationship. Therefore, the average number of nodes visited by ALIGNED is

$$\frac{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-i} \cdot 2^{n-j} \cdot \sum_{k=0}^{j-1} |i-k|}{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-i} \cdot 2^{n-j} \cdot j} \tag{11}$$

The numerator of (11) can be simplified to yield

$$\sum_{i=0}^{n-1} \sum_{j=1}^{n-i} 2^{2n-2i-j-1} \cdot (j^2 - j - i + i^2) \tag{12}$$

Making use of the identities (3) − (8) in (12) leads to (13)

$$\tfrac{19}{27} \cdot 2^{2n+2} - (n^2 + n + 6) \cdot 2^n + \tfrac{1}{3}n^2 + \tfrac{11}{9}n + \tfrac{86}{27} \tag{13}$$

The denominator of (11) is equal to (10) and substituting (13) and (10) into (11) yields:

$$\frac{19}{21} - \frac{3}{7}\,\frac{(21n^2 - 17n + 50) \cdot 2^n - 7n^2 - 13n - 50}{7 \cdot 2^{2n+2} - 9 \cdot (n+2) \cdot 2^{n+1} + 6n + 8}$$

$$\sim \frac{19}{21} \text{ as } n \text{ gets large}$$

$$\leq \frac{19}{21}$$ □

At this point we can prove the following theorem.

THEOREM 3. *The chain code description of a region represented by a quadtree can be obtained in time proportional to the region's perimeter.*

PROOF. Procedure NEXT_LINK requires time proportional to the region's perimeter to output the individual links. Determining the next (BLACK, WHITE) adjacency takes time which on the average is proportional to $2 \cdot 14/3 + 19/21$ for each invocation of procedures CORNER_ADJ_NEIGHBOR and ALIGNED respectively. However, the number of times these procedures are involved is bounded by the region's perimeter. □

7. Concluding Remarks

An algorithm has been presented for converting the quadtree representation of a simply connected region into the chain code description of the region's boundary. The algorithm is different from [17] where a method is reported for computing the perimeter of an image represented by a quadtree, in that we must visit the boundary nodes in a specified sequence. We have shown that our algorithm's execution time is bounded by the perimeter of the region. It is interesting to note that our concept of a random image yields higher averages for the number of nodes that are visited than when a random image is defined as stipulating that every node at level 0 has an equal probability of being BLACK or WHITE. Recall that in such a case there is a very low likelihood of nodes corresponding to blocks of size larger than 1. In fact, for such an image, it is shown in [16] that the average number of nodes visited in determining neighbors is bounded by 4 whereas our model of the image yields an average bounded by 14/3.

In the case where a region may have holes, we may extend our algorithm by simply adding a quadtree traversal procedure which systematically visits all BLACK nodes upon completion of the first boundary following sequence. If such a scan results in the discovery of a BLACK boundary node with a boundary edge that was not marked by the boundary follower, then the scan is temporarily interrupted so that the boundary of its corresponding region can be followed.

Some related work concerned with quadtrees and the establishment of their applicability for efficiently operating on areal data includes the following. In [16] an algorithm is reported for the converse operation of constructing the quadtree from the chain code of a region. Also, in [18] an efficient algorithm is given for determining the connectivity of BLACK nodes. This is interesting because the algorithm presented herein assumes a single connected region.

Acknowledgments. We thank K. Riley for typing this paper, P. Young for drawing the figures, and J. Antonisse and P. McMullin for their comments.

Received February 1979; revised November 1979; accepted December 1979

References

1. Alexandridis, N., and Klinger, A. Picture decomposition, tree data-structures, and identifying directional symmetries as node combinations. *Comptr. Graphics and Image Processing 8* (1978), 43–77.
2. Blum, H. A transformation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*, W. Wathen-Dunn, Ed., M.I.T. Press, Cambridge, Mass., 1967, pp. 362–380.
3. Finkel, R.A., and Bentley, J.L. Quadtrees: A data structure for retrieval on composite keys. *Acta Informatica 4* (1974), 1–9.
4. Freeman, H. Computer processing of line-drawing images. *Comptg. Surv. 6* (1974), 57–97.
5. Horowitz, S.L., and Pavlidis, T. Picture segmentation by a tree-traversal algorithm. *J. ACM 23* (1976), 368–388.
6. Hunter, G.M. Efficient computation and data structures for graphics. Ph.D. dissertation, Dept. of Electrical Engineering and Comptr. Sci., Princeton Univ., Princeton, N.J., 1978.
7. Hunter, G.M., and Steiglitz, K. Operations on images using quadtrees. *IEEE Trans. on Pattern Analysis and Machine Intell. 1* (1979), 145–153.
8. Hunter, G.M., and Steiglitz, K. Linear transformation of pictures represented by quadtrees. *Comptr. Graphics and Image Processing 10* (1979), 289–296.
9. Klinger, A., and Dyer, C.R. Experiments in picture representation using regular decomposition. *Comptr. Graphics and Image Processing 5* (1976), 68–105.
10. Klinger, A., and Rhodes, M.L. Organization and access of image data by areas. *IEEE Trans. on Pattern Analysis and Machine Intell. 1* (1979), 50–60.
11. Naur, P., Ed. Revised report on the algorithmic language ALGOL 60. *Comm. ACM 3*, 5(May 1960), 299–314.
12. Pfaltz, J.L., and Rosenfeld, A. Computer representation of planar regions by their skeletons. *Comm. ACM 10*, 2 (Feb. 1967), 119–122, 125.
13. Riseman, E.M., and Arbib, M.A. Computational techniques in the visual segmentation of static scenes. *Comptr. Graphics and Image Processing 6* (1977), 221–276.
14. Rosenfeld, A., and Pfaltz, J.L. Sequential operations in digital picture processing. *J. ACM 13* (1966), 471–494.
15. Rutovitz, D. Data structures for operations on digital images. In *Pictorial Pattern Recognition*, G.C. Cheng et al., Eds., Thompson Book Co., Washington, D.C., 1968, pp. 105–133.
16. Samet, H. Region representation: Quadtrees from boundary codes. Comptr. Sci. TR-741, Univ. of Maryland, College Park, Md., March 1979; *Comm. ACM 23* 3(March 1980), 163–170.
17. Samet, H. Computing perimeters of images represented by quadtrees. Comptr. Sci. TR-755, Univ. of Maryland, College Park, Md., April 1979.
18. Samet, H. Connected component labeling using quadtrees. Comptr. Sci. TR-756, Univ. of Maryland, College Park, Md., April 1979.
19. Tanimoto, S.L. Pictorial feature distortion in a pyramid. *Comptr. Graphics and Image Processing 5* (1976), 333–352.
20. Tanimoto, S.L., and Pavlidis, T. A hierarchical data structure for picture processing. *Comptr. Graphics and Image Processing 4* (1975), 104–119.

Professional Activities Calendar of Events

ACM's calendar policy is to list open computer science meetings that are held on a not-for-profit basis. Not included in the calendar are educational seminars, institutes, and courses. Submittals should be substantiated with name of the sponsoring organization, fee schedule, and chairman's name and full address.

One telephone number contact for those interested in attending a meeting will be given when a number is specified for this purpose.

All requests for ACM sponsorship or cooperation should be addressed to Chairman, Conferences and Symposia Committee, Seymour J. Wolfson, 643 MacKenzie Hall, Wayne State University, Detroit, MI 48202, with a copy to Louis Fiora, Conference Coordinator, ACM Headquarters, 1133 Avenue of the Americas, New York, NY 10036; 212 265-6300. For European events, a copy of the request should also be sent to the European Representative. Technical Meeting Request Forms for this purpose can be obtained from ACM Headquarters or from the European Regional Representative. Lead time should include 2 months (3 months if for Europe) for processing of the request, plus the necessary months (minimum 3) for any publicity to appear in *Communications*.

■ This symbol indicates that the Conferences and Symposia Committee has given its approval for ACM sponsorship or cooperation.

In this issue the calendar is given to November 1981. New Listings are shown first; they will appear next month as Previous Listings.

NEW LISTINGS

20–21 March 1980
Computers in Elementary Education, Washington, D.C. Sponsor: AEDS. Contact: Shirley Easterwood, AEDS Workshops, 1201 16th St., N.W., Washington, D.C. 20036; 202 833-4100.

13–15 April 1980
IVth International Conference on Collective Phenomena, Moscow, U.S.S.R. Sponsors: New York Academy of Sciences, Committee of Concerned Scientists with support of SIAM, AMS, APS. Contact: Dorothy Hirsch, Committee of Concerned Scientists, Inc., 9 East 40th St., New York, NY 10016; 212 686-8862.

13–16 April 1980
Spring 80 Conference, Atlanta, Ga. Sponsor: COMMON. Contact: David G. Lister, Administrative Director, COMON-Sz, 435 N. Michigan Ave., Suite 1717, Chicago, IL 60611; 312 644-0828.

25–26 April 1980
Symposium in Cognitive Science, Poughkeepsie, N.Y. Sponsor: Vassar College Cognitive Science Group. Contact: Cognitive Science Symposium, Vassar College, Box 525, Poughkeepsie, NY 12601.

28–30 April 1980
Workshop on Array Architecture for Computing in the 80s and 90s, Hampton, Va. Sponsor: ICASE. Contact: Robert G. Voigt, ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665; 804 827-2513.

1 May 1980
Social Science Data Workshop (preceding IASSIST Annual Conference 1980), Washington, D.C. Sponsor: IASSIST. Workshop chm: Barbara Aldrich, 2715A S. Walter Reed Drive, Arlington, VA 22206.

15–17 May 1980
9th ASIS Mid-Year Meeting, Pittsburgh, Pa. Sponsor: American Society for Information Science. Conf. chm: K. Leon Montgomery, Information Science, 721 LIS Building, University of Pittsburgh, Pittsburgh, PA 15260.

19–23 May 1980
Conference on Mathematical Models for Energy and Public Policy, Charleston, S.C. Host: The Citadel. Contact: John I. Moore Jr., Dept. of Mathematics, The Citadel, Charleston, SC 29409.

24–31 August 1980
Data Processing in Chemistry 80, Rzeszów, Poland. Organizers: Committee of Chemical Sciences, Polish Academy of Sciences, I. Lukasiewicz Technical University. Contact: Institute of Chemical Technology, I. Lukasiewicz Technical University, 35-959 Rzeszów, P.O. Box 85, Poland.

22–25 September 1980
12th Annual Conference of the Society for Management Information Systems, Philadelphia, Pa. Sponsor: SMIS. Contact: SMIS Headquarters, 111 East Wacker Drive, Chicago, IL 60601.

24–27 September 1980
Tenth Annual Conference of the Society for Computer Medicine, San Diego, Calif. Sponsor: SCM. Contact: Society for Computer Medicine, 1901 N. Ft. Myer Drive, Suite 602, Arlington, VA 22209.

30 September–2 October 1980
Annual Conference 80, Universität des Saarlandes, Saarbrücken, West Germany. Sponsor: Gesellschaft für Informatik. Prog. chm: R. Wilhelm, FB 10—Informatik, Universität des Saarlandes, D-6600 Saarbrücken 11, West Germany.

1–3 October 1980
Twenty-First Annual Symposium on Foundations of Computer Science, Lake Placid, N.Y. Sponsor: IEEE-CS Tech. Comm. on Mathematical Foundations of Computing. Prog. chm: Andrew C. Yao, Computer Science Dept., Stanford University, Stanford, CA 94305.

8–10 October 1980
Eighteenth Annual Allerton Conference on Communication, Control, and Computing, Allerton House, near Monticello, Ill. Sponsor: University of Illinois at Urbana-Champaign. Conf. co-chm: D.V. Sarwate and W.R. Perkins, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

20 October 1980
National Information Systems: Getting Ready for 1984, Washington, D.C. Sponsor: Computer Law Association. Contact: Michael Yourshaw, 1776 K St., N.W., Washington, DC 20006; 202 857-5029.

27–31 October 1980
Compsac 80, Fourth International Computer Software and Applications Conference, Chicago, Ill. Sponsor: IEEE-CS. Gen. chm: Richard Merwin, Dept. of EE and Computer Science, The George Washington University, Washington, DC 20052; 202 676-4951.

8–11 December 1980
■ **ACM SIGPLAN Symposium on the Ada Programming Language,** Boston, Mass. Sponsor: ACM SIGPLAN. Conf. chm: Robert M. Graham, COINS Graduate Research Center, University of Massachusetts, Amherst, MA 01003; 413 545-2742.

8–9 January 1981
■ **14th Annual Hawaii International Conference on Systems Sciences,** Honolulu. Sponsor: University of Hawaii in cooperation with ACM SIGBDP. Conf. chm: Ralph H. Sprague Jr., University of Hawaii at Manoa, College of Business Administration, 2404 Maile Way, Honolulu, HI 96822; 808 948-7430.

22–26 June 1981
XXV International Meeting of the Institute of Management Sciences, Cairo, Egypt. Sponsor: TIMS. Gen. chm: Mostafa El Agizy, Exxon Corporation, Box 153, Florham Park, NJ 07932.

PREVIOUS LISTINGS

17–19 March 1980
IECI 80, Sixth Annual Conference and Exhibit on Industrial and Control Applications of Microprocessors, Philadelphia, Pa. Sponsors: Industrial Electronic and Control Instrumentation Society, IEEE. Gen. chm: Paul M. Russo, RCA Laboratories, Princeton, NJ 08540; 609 452-2700.

18–20 March 1980
Electric Power Problems: The Mathematical Challenge, Seattle, Wash. Sponsor: SIAM. Contact: Albert M. Erisman, Boeing Computer Services Co., 565 Andover Park West, M/S 9C-01, Tukwila, WA 98188.

19–21 March 1980
■ **13th Annual Simulation Symposium,** Tampa, Fla. Sponsors: ACM SIGSIM, IEEE-CS, SCS. Symp. chm: Harvey Fisher, Alcan Products, Box 511, Warren, OH 44482; 216 841-3416.

(Calendar continued on p. 184)