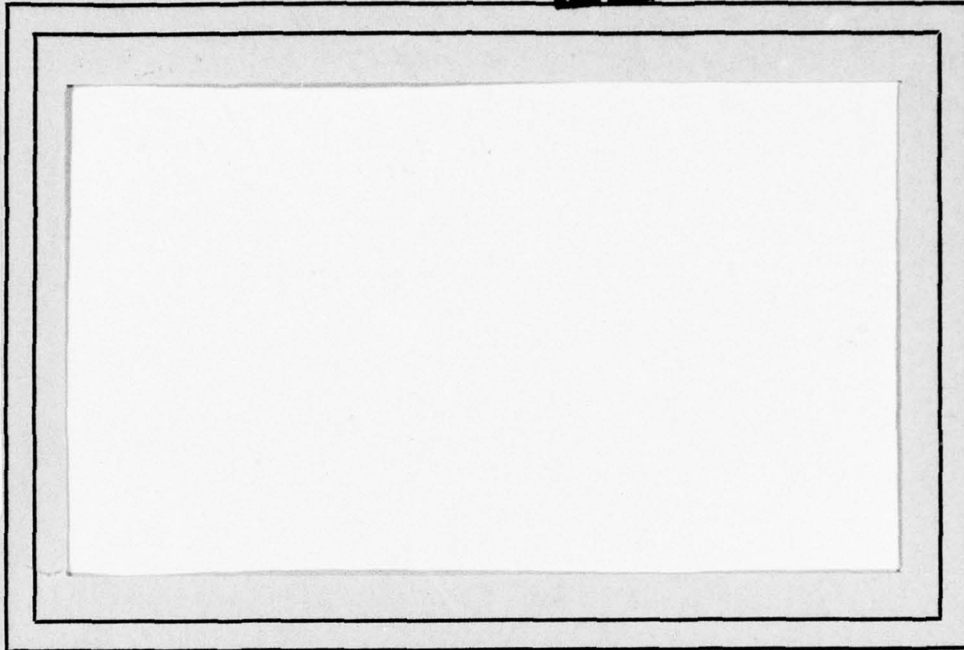
LEVEL II

AD A078086



DDC
DEC 3 1979
E

DDC FILE COPY

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

This document has been approved for public release and sale; its distribution is unlimited.

79 11 30 008

(15) DAAG53-76-C-Ø138, DARPA Order-32Ø6

(1)

(9) Technical rept.,

(14) TR-766
DAAG-53-76C-0138

(11) May 1979

(6) REGION REPRESENTATION: RASTER-TO-QUADTREE CONVERSION.

(12) 3Ø.

(10) Hanan Samet
Computer Science Department
University of Maryland
College Park, MD 20742

DDC
DEC 3 1979
RECEIVED
E

ABSTRACT

An algorithm is presented for constructing a quadtree for a binary image given its row-by-row description. The algorithm processes the image one row at a time and merges identically colored sons as soon as possible so that a minimal size quadtree exists after processing each pixel. This method is spacewise superior to one which reads in an entire array and then attempts to build the quadtree. Analysis of the algorithm reveals that its execution time is proportional to the number of pixels comprising the image.

This document has been approved for public release and sale; its distribution is unlimited.

The support of the Defense Advanced Research Projects Agency and the U.S. Army Night Vision Laboratory under Contract DAAG-53-76C-0138 (DARPA Order 3206) is gratefully acknowledged. as is the help of Kathryn Riley in preparing this paper. The author has also benefited greatly from discussions with Charles R. Dyer and Azriel Rosenfeld. He also thanks Pat Young for her help with the figures.

409 022

JOB

1. Introduction

Region representation is an important issue in image processing, cartography, and computer graphics. There are numerous representations currently in use (see [DRS] for a brief review). In this paper we present an algorithm for obtaining the quadtree representation [Klinger] given the row-by-row description of a binary image. Such an algorithm is useful because each representation is well-suited for a set of operations or may be desirable for its compactness. For example, the row-by-row representation is especially useful for interaction with raster-like display devices since input and output requires very little additional computation. In addition, it is also a useful technique when memory size limitations preclude storing in core an array corresponding to the image (e.g., [Samet4]). On the other hand, the quadtree is a compact hierarchical representation, thereby facilitating search.

In the remainder of this section we briefly define the representations used. Sections 2-5 present and analyze our algorithm. Included is a formal description of the algorithm in addition to a rationale for its various steps. The formal presentation of the algorithm is made using a variant of ALGOL 60 [Naur].

Assume that the image is a 2^n by 2^n array. Each row of the image is thus a bit string of length 2^n. The quadtree is an approach to image representation based on successive subdivision of the image into quadrants. In essence, we repeatedly subdivide the

array into quadrants, subquadrants,... until we obtain blocks (possibly single pixels) which consist entirely of either 1's or Ø'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, Figure 1b is a block decomposition of the region in Figure 1a while Figure 1c is the corresponding quadtree. In general, BLACK and WHITE square nodes represent blocks consisting entirely of 1's and Ø's respectively. Circular nodes, also termed GRAY nodes, denote non-terminal nodes.

Accession For
NTIS GRA&I
DDC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or special
A

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, 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, named NODETYPE, describes the contents of the block of the image which the node represents--i.e., WHITE if the block contains no 1's, BLACK if the block contains only 1's, and GRAY if it contains 0's and 1's. Alternatively, BLACK and WHITE are terminal nodes while GRAY nodes are non-terminal nodes.

Let the four sides of a node's block be called its N, E, S, and W sides. They are also termed its boundaries. The spatial relationships between the various sides are specified by use of the functions OPSIDE and CCSIDE. OPSIDE(B) is a side facing side B; e.g., OPSIDE(E)=W. CCSIDE(B) corresponds to the side adjacent to side B in the clockwise direction; e.g., CCSIDE(E)=N. We also define the following predicates and functions to facilitate 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(N,NW) is true. REFLECT(B,I) yields the

quadrant which is adjacent to quadrant I along boundary B of the block represented by I; e.g., REFLECT(N,NE)=SE, REFLECT(E,NE)=NW, REFLECT(S,NE)=SE, and REFLECT(W,NE)=NW. QUAD(B,C) is the quadrant which is bounded by boundaries B and C (if B and C are non-adjacent boundaries, then the value of QUAD(B,C) is undefined); e.g., QUAD(N,E)=NE. OPQUAD(Q) is the quadrant which is non-adjacent to quadrant Q; e.g., OPQUAD(NE)=SW, OPQUAD(SE)=NW, OPQUAD(SW)=NE, and OPQUAD(NW)=SE. Figure 2 shows the relationship between the quadrants of a node and its boundaries.

Given 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.

3. Algorithm

The key to the raster-to-quadtree algorithm is that at any instant of time (i.e., after each pixel in a row has been processed), a valid quadtree exists with all unprocessed pixels presumed to be WHITE. Thus as the quadtree is built, nodes are merged to yield maximal blocks. This is in contrast to an algorithm which first builds a complete quadtree with one node per pixel and then attempts to merge--i.e., replace all GRAY nodes with four sons of the same color by a node of the same color. The disadvantage of the complete quadtree method is that it requires more space. In particular, for a 2^n by 2^n image, 2^{2n} BLACK and WHITE nodes may be required in addition to $\frac{1}{3} 2^{2n}$ non-terminal GRAY nodes. This is clearly undesirable when compared with a maximum of 2^{2n} bits required by the binary array representation. Note that a hybrid method was used in [Samet1] to construct a quadtree from the boundary code of the image--i.e., the first pass left a number of links unspecified while the second pass filled in the links and attempted to merge the resulting nodes.

The given binary image is assumed to be partitioned into rows. Clearly, no odd-numbered row can lead to a merge of nodes unless it is the last row. Thus odd-numbered rows don't require as much processing as even-numbered rows. Assume that the image contains an even number of rows. If the image

contains an odd number of rows, then it is presumed that one extra row of WHITE has been added.

For an odd-numbered row, the tree is constructed by processing the row from left to right adding a node to the tree for each pixel. For example, Fig. 3a through 3i shows the construction of a quadtree corresponding to the first four pixels of the binary image of Fig. 1a (i.e., pixels 1, 2, 3, and 4). This is done by invoking a procedure called FIND_NEIGHBOR. As the quadtree is constructed, non-terminal nodes must also be added. Since we wish to have a valid quadtree after processing each pixel, whenever we add a non-terminal node we also add, as is appropriate, three or four WHITE nodes as its remaining sons.

More formally, finding a neighbor of a node in a specified direction consists of traversing FATHER links until a common ancestor is found. Once the ancestor is found, we descend along a path that is reflected about the axis formed by the common boundary between the two nodes. If a common ancestor does not exist, then a non-terminal node is added with its three remaining sons being WHITE (e.g., Fig. 3c and 3f). Once the common ancestor and its three sons have been added, we once again descend along a path reflected about the axis formed by the boundary of the node whose neighbor we seek. During this descent, a WHITE node is converted to a GRAY node and four

WHITE sons are added (e.g., Fig. 3g). As a final step, the terminal node is colored appropriately (e.g., Fig. 3d and 3h). In the example, Fig. 3a, 3b-3d, 3e-3h, and 3i are snapshots of the quadtree construction process for the nodes corresponding to pixels 1, 2, 3, and 4 respectively of Fig. 1a.

Even-numbered rows require more work since merging may also take place. In particular, a check for a possible merge must be performed at every even numbered vertical position (i.e., every even-numbered pixel in a row). Once a merge occurs, we may have to check if another merge is possible. In particular, for pixel position $(a2^i, b2^j)$ where a mod 2 = b mod 2 = 1, a maximum of k=min(i,j) merges is possible. For example, at pixel 60 of Fig. 1a, i.e., position (8,4), a maximum of 2 merges is possible and indeed this is how block E of Fig. 1b has been obtained. The fact that merging does take place causes an additional amount of bookkeeping. In particular, we wish to maintain the position in the tree where the next pixel is to be added as well as the next row. Prior to attempting a merge, a node corresponding to the next pixel in the image is added to the quadtree (e.g., node 11 is added to the quadtree in Fig. 4 prior to attempting to merge nodes 1, 2, 9, and 10 of Fig. 1a). Similarly, we precede the processing of each even-numbered row by adding to the quadtree a node corresponding to the first pixel in the next row

(e.g., the addition of node 33 to the tree of Fig. 5 prior to processing row 4 of Fig. 1a). This type of lookahead was also employed for different reasons in the perimeter computation and connected component labeling algorithms [Samet2, Samet3].

As an example of the application of the algorithm, consider the image given in Figure 1a. Figure 1b is the corresponding block decomposition and Figure 1c is its quadtree representation. All of the nodes that are a result of merging have been labeled with letters (i.e., A through I) and the alphabetical order corresponds to the order in which the merged nodes were created. Figures 3a through 3i show the steps in the construction of the quadtree corresponding to the first part of the first row. Figures 6 and 7 show the resulting tree after the first and second rows have been processed.

The following ALGOL-like procedures specify the algorithm. The main procedure is called QUADTREE and is invoked with a pointer to the first row. QUADTREE controls the construction of the quadtree by invoking procedures ODDROW and EVENROW to add odd-numbered and even-numbered rows respectively to the tree. This is facilitated by the use of procedures FIND_NEIGHBOR and CREATENODE to locate neighboring nodes and creation of nodes for pixels which have no corresponding node in the tree. Procedure MERGE is responsible for replacing any GRAY node having four sons of the same color by a node of the same color.

COLOR is a function that converts the pixel's boolean value to the appropriate color (e.g., BLACK and WHITE for 1 and Ø respectively).

```
node procedure QUADTREE(P,WIDTH);
/* build a quadtree corresponding to the image whose binary
   representation is contained in a list of rows, WIDTH pixels
   wide, pointed at by P */
begin
    list P;
    Boolean array Q [1:WIDTH];
    node FIRST;
    integer I;
    ODDROW(Q←GETROW(P), FIRST←CREATENODE(NULL,NULL,Q[1]),WIDTH);
    P←NEXT(P);
    FIRST←EVENROW(NULL(NEXT(P)),GETROW(P),FIND_NEIGHBOR(FIRST,'S'),
                  I←2,WIDTH);
    while not NULL(P←NEXT(P)) do  /* assume an even number of rows */
        begin
            ODDROW(GETROW(P),FIRST,WIDTH);
            P←NEXT(P);
            FIRST←EVENROW(NULL(NEXT(P)),GETROW(P),
                          FIND_NEIGHBOR(FIRST,'S'),I+I←2,WIDTH);
        end;
    while not NULL(FATHER(FIRST)) do FIRST←FATHER(FIRST);
    return(FIRST);  /*return the root of the quadtree */
end;
```

```
procedure ODDROW(Q,R,W)
/* add the odd-numbered row of width W represented by Q to a
   quadtree whose node R corresponds to the first pixel in
   the row */
begin
    integer W,I;
    Boolean array Q[1:W];
    node R;
    NODETYPE(R)←COLOR(Q[1]);
    for I←2 step 1 until W do
        begin
            R←FIND_NEIGHBOR(R,'E');
            NODETYPE(R)←COLOR(Q[I]);
        end;
end;
```

```
node procedure EVENROW(LASTROW,Q,FIRST,I,W);
/* add even numbered row I of width W represented by Q to a
   quadtree whose node FIRST corresponds to the first pixel in
   the row.  During this process, merges of nodes having four
   sons of the same color are performed.  LASTROW indicates if
   row I is the last row in the image */
begin
    integer I,J,W;
    Boolean array Q[1:W];
    node FIRST,P,R,T;
    Boolean LASTROW;
    P←FIRST;
    if not LASTROW then /*remember the first node of the next row */
        FIRST←FIND_NEIGHBOR(P,'S');
    for J←1 step 1 until W-1 do
        begin
            R←FIND_NEIGHBOR(P,'E');
            NODETYPE(P)←COLOR(Q[J]);
            if J mod 2=Ø then MERGE(I,J,FATHER(P));
            P←R;
        end;
    NODETYPE(P)←COLOR(Q[W]); /* don't invoke FIND_NEIGHBOR for
    last pixel in a row */
    if W mod 2≠Ø then T←MERGE(I,W,FATHER(P));
    return (if LASTROW then T
            else FIRST);
end;
```

```
node procedure FIND_NEIGHBOR(Q,S);
/* return a node P which is adjacent to side S of node Q.
   This is done by finding a common ancestor of the two nodes
   and creating one if it does not exist.  Whenever a common
   ancestor or other nodes are created, all created sons are
   set to WHITE.  They are later reset to GRAY or BLACK as
   appropriate */
begin
    node P,Q;
    side S;
    quadrant I;
    if NULL(SONTYPE(Q)) then  /* common ancestor does not exist */
        begin
            P←CREATENODE(NULL,NULL,GRAY); /*create a common ancestor */
            SONTYPE(Q)←QUAD(CCSIDE(S),OPSIDE(S));
            SON(P,SONTYPE(Q))←Q;
            FATHER(Q)←P;
            /* create three sons */
            CREATENODE(P,OPQUAD(SONTYPE(Q)),WHITE);
            CREATENODE(P,OPQUAD(REFLECT(S,SONTYPE(Q)),WHITE);
            return (CREATENODE(P,REFLECT(S,SONTYPE(Q)),WHITE));
        end
    else if ADJ(S,SONTYPE(Q)) then P←FIND_NEIGHBOR(FATHER(Q),S)
    else P←FATHER(Q);
    /* trace a path from the common ancestor to the adjacent node
       creating WHITE sons and relabeling non-terminal nodes
       to GRAY as necessary */
```

```
    if NULL(SON(P,REFLECT(S,SONTYPE(Q)))) then
        begin
            NODETYPE(P)←GRAY;
            for I in {NW,NE,SW,SE} do CREATENODE(P,I,WHITE);
        end;
    return(SON(P,REFLECT(S,SONTYPE(Q))));
end;
```

```
node procedure CREATENODE(ROOT,S,T);
/* create a node P with color T which corresponds to son S
   of node ROOT and return P */
begin
    node P,ROOT;
    quadrant I,S;
    Boolean T;
    P←GETNODE();
    if ROOT then SON(ROOT,S)←P; /* created node has a father */
    SONTYPE(P)←S;
    FATHER(P)←ROOT;
    NODETYPE(P)←COLOR(T);
    for I in {NW,NE,SW,SE} do SON(P,I)←NULL;
    return(P);
end;
```

4. Analysis

The running time of the quadtree construction process is measured by the number of nodes that are visited. Thus we only need to analyze the amount of time used by procedures FIND_NEIGHBOR and MERGE. In our analysis we assume that the image is a 2^n by 2^n array of pixels. We first prove the following lemma:

Lemma 1: The number of nodes visited by FIND_NEIGHBOR is bounded by 4 times the number of pixels.

Proof: For each row in the image, FIND_NEIGHBOR is invoked 2^n-1 times to find a neighbor in the eastern direction. 2^0 of the nodes corresponding to the pixels in each row have their nearest common ancestor at level n, 2^1 at level n-1,...,2^i at level n-i, and 2^{n-1} at level 1. There are 2^n rows which invoke FIND_NEIGHBOR in the easterly direction. In addition, each row invokes FIND_NEIGHBOR once, for the first column, in the southerly direction. Once again, 2^0 of the nodes corresponding to the pixels in the first column have their nearest common ancestor at level n, 2^1 at level n-1,..., 2^i at level n-i, and 2^{n-1} at level 1. Therefore, the total number of nodes visited by FIND_NEIGHBOR in locating a common ancestor is obtained as follows:

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

$$\sum_{i=1}^{n} \frac{i}{2^i} = \frac{1}{2}\sum_{i=0}^{n-1} \frac{i+1}{2^i}$$

$$= \frac{1}{2}\sum_{i=0}^{n-1} \frac{i}{2^i} + \frac{1}{2}\sum_{i=0}^{n-1} \frac{1}{2^i}$$

$$= \frac{1}{2}\sum_{i=1}^{n-1} \frac{i}{2^i} + \frac{1}{2}\,\frac{1-\frac{1}{2^n}}{1-\frac{1}{2}}$$

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

$$\therefore \quad \sum_{i=1}^{n} \frac{i}{2^i} = 2 - \frac{n+2}{2^n}$$

$$(2^n+1)2^n \sum_{i=1}^{n} \frac{i}{2^i} = (2^{2n}+2^n)(2-\frac{n+2}{2^n})$$

$$= 2^{2n+1} + 2^{n+1} - 2^n(n+2) - (n+2)$$

$$= 2^{2n+1} - n2^n - (n+2)$$

$$< 2^{2n+1}$$

Once a common ancestor has been found, FIND_NEIGHBOR must descend an equal number of links in order to locate the neighbor. Thus FIND_NEIGHBOR will visit a maximum of 2^{2n+2} nodes. However, the image, being a 2^n by 2^n array, contains 2^{2n} pixels and thus FIND_NEIGHBOR will visit a number of nodes bounded by 4 times the number of pixels.

Q.E.D.

As noted earlier, merging is only attempted at specified grid positions. An upper bound on the number of nodes checked for merging is attained when all of the pixels are of the same color. This leads to the following lemma:

Lemma 2: The number of nodes examined for merging is bounded by the number of pixels.

Proof: Each time a merge is attempted, we are at a node corresponding to a pixel position $(a2^i, b2^j)$ where a mod 2 = b mod 2 = 1. Letting $k = \min(i,j)$, we see that $k=1$ implies that three additional nodes must be visited and checked if they are of the same color as the node at pixel position $(a2^i, b2^j)$. If $k>1$, and if the node corresponding to pixel position $(a2^i, b2^j)$ has the same color as the nodes in pixel positions $(a2^i, b2^j-1)$, $(a2^i-1, b2^j)$, and $(a2^i-1, b2^j-1)$, then we reapply the test to the three nodes which are neighbors of the merged node, etc. For a 2^n by 2^n image there are 2^{2n-2} pixels such that $k=1$, 2^{2n-4} pixels such that $k=2, \ldots, 2^{2n-2\ell}$ pixels such that $k=\ell$, and 2^0 pixels such that $k=n$. Therefore the maximum number of nodes that can be visited by MERGE is obtained as follows:

$$\sum_{k=1}^{n} 3 \cdot 2^{2n-2k} = 3 \cdot 2^{2n} \sum_{k=1}^{n} \frac{1}{4^k}$$

$$\leq 3 \cdot 2^{2n} \left(\frac{1}{3}\right)$$

$$= 2^{2n}$$

Q.E.D.

We now come to the main result:

Theorem: The quadtree can be constructed from a sequence of rows in time proportional to the number of pixels.

Proof: From Lemma 1, the maximum number of nodes visited by procedure FIND_NEIGHBOR is bounded by four times the number of pixels. From Lemma 2 the maximum number of nodes visited by the merging process is bounded by the number of pixels. Thus the maximum number of nodes that are visited is bounded by five times the number of pixels.

Q.E.D.

5. Concluding Remarks

An algorithm has been presented for converting a row-by-row representation of a binary image into a quadtree representation of the image. The algorithm's execution time has been shown to have a time complexity proportional to the number of pixels in the image. The algorithm is also spacewise efficient in that merging is attempted whenever possible. Thus after processing each pixel in a row, the resulting quadtree contains a minimal number of nodes.

The algorithm is one-dimensional in the sense that it processes the image a row at a time. Thus it can be used in conjunction with the run length representation [RK] which views each row as a sequence of maximal runs of pixels having the same value. A row is thus completely determined by specifying the lengths of these runs and the value of the first run. When only a few runs are present, this representation is very economical. For example, consider the image in Fig. 1a. Its run length coding is B4121, W2141, W53, W53, W422, W422, W8, W8 where commas serve as separators between rows.

The algorithm can be contrasted with two other approaches. If sufficient memory is available to store the array in core, then the technique of [Samet4] which only creates nodes corresponding to maximal blocks, and hence requires no merging, should be used. Alternatively, we could attempt to find maximal blocks by processing several rows at once (e.g., 2^m

consecutive rows with runs of length $\geq 2^m$ at the NW-most corner of the image yield a block of size 2^m). The disadvantage of such an approach is that it requires searching and a substantial amoung of bookkeeping as the rows are being processed. Instead, we have found maximal blocks by merging at appropriate pixel positions.

6. References

[DRS] C. R. Dyer, A. Rosenfeld, and H. Samet. Region representation: boundary codes from quadtrees, Computer Science TR-732, University of Maryland, College Park, Maryland, February 1979.

[Klinger] A. Klinger and C. R. Dyer, Experiments in picture representation using regular decomposition, Computer Graphics and Image Processing 5, 1976, 68-105.

[Naur] P. Naur (Ed.), Revised report on the algorithmic language ALGOL 60, Communications of the ACM 3, 1960, 299-314.

[RK] A. Rosenfeld and A. C. Kak, Digital Picture Processing, Academic Press, New York, 1976.

[Samet1] H. Samet, Region representation: quadtrees from boundary codes, Computer Science TR-741, University of Maryland, College Park, Maryland, March 1979.

[Samet2] H. Samet, Computing perimeters of images represented by quadtrees, Computer Science TR-755, University of Maryland, College Park, Maryland, April 1979.

[Samet3] H. Samet, Connected component labeling using quadtrees, Computer Science TR-756, University of Maryland, College Park, Maryland, April 1979.

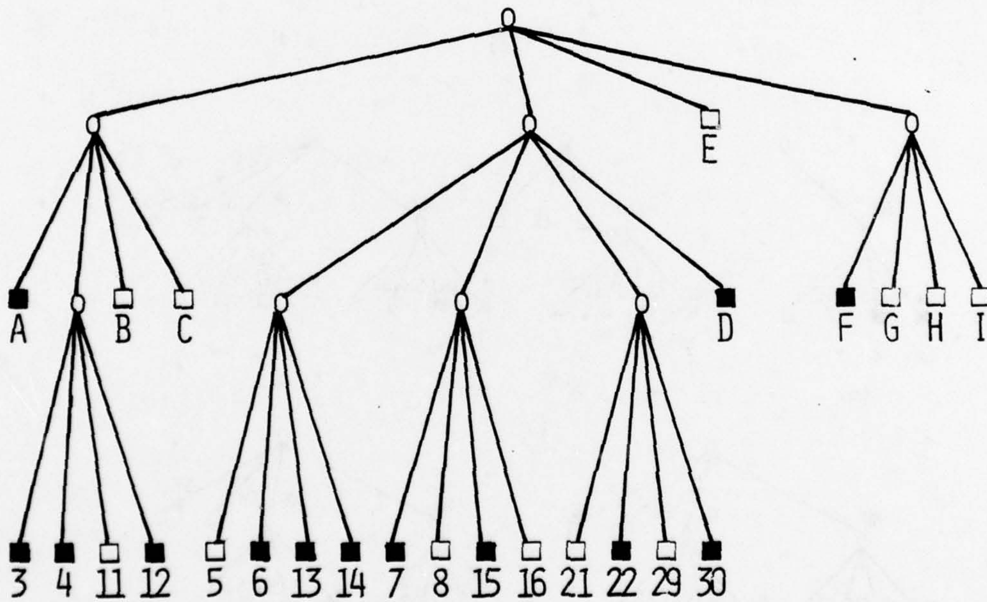
[Samet4] H. Samet, Region representation: quadtrees from binary arrays, Computer Science TR-767, University of Maryland, College Park, Maryland, May 1979.

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

a. Sample image.

A 3 4 5 6 7 8
11 12 13 14 15 16
B C 21 22 D
29 30
F G
E
H I

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



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

Figure 1. An image, its maximal blocks, and the corresponding quadtree. Blocks in the image are shaded.

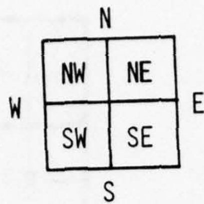


Figure 2. Relationship between a block's four quadrants and its boundaries.

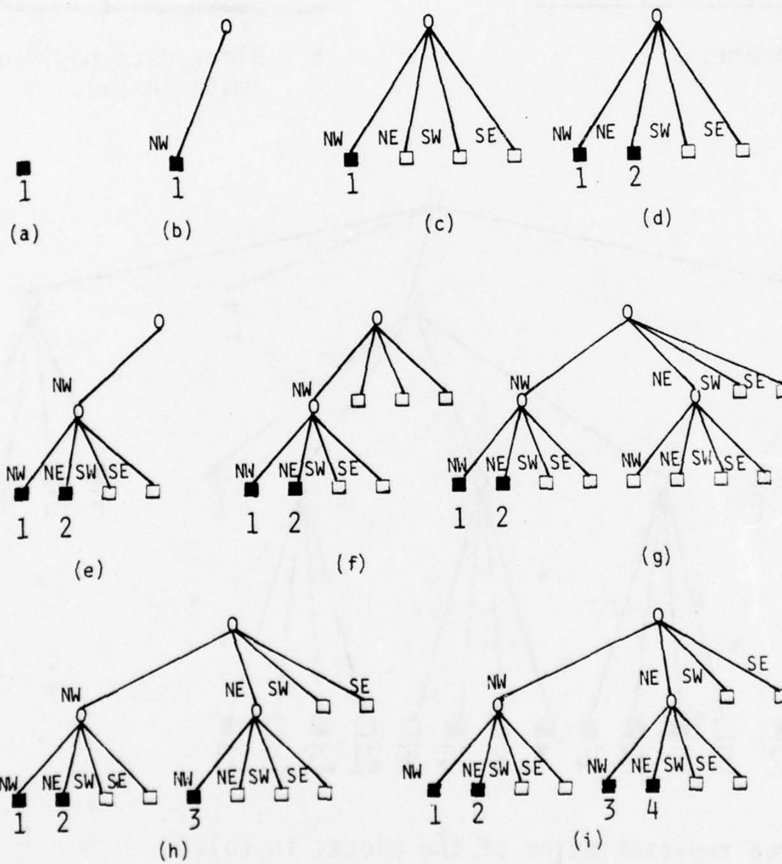


Figure 3. Intermediate trees in the process of obtaining a quadtree for the first part of the first row in Figure 1a.

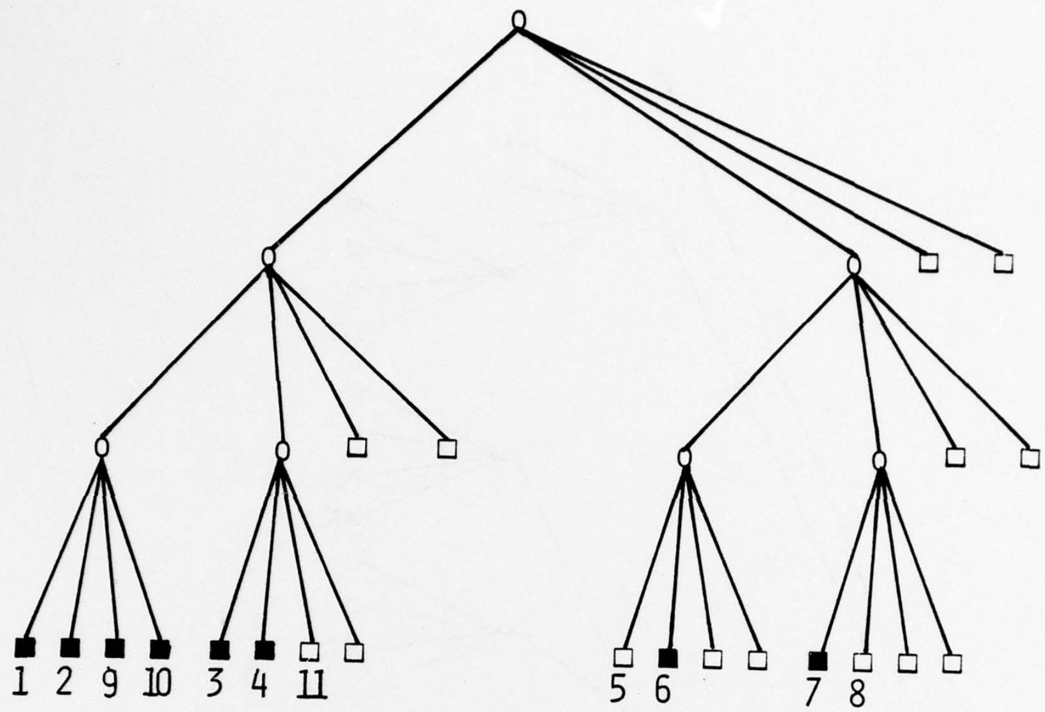


Figure 4. Quadtree prior to merging nodes 1, 2, 9, and 10.

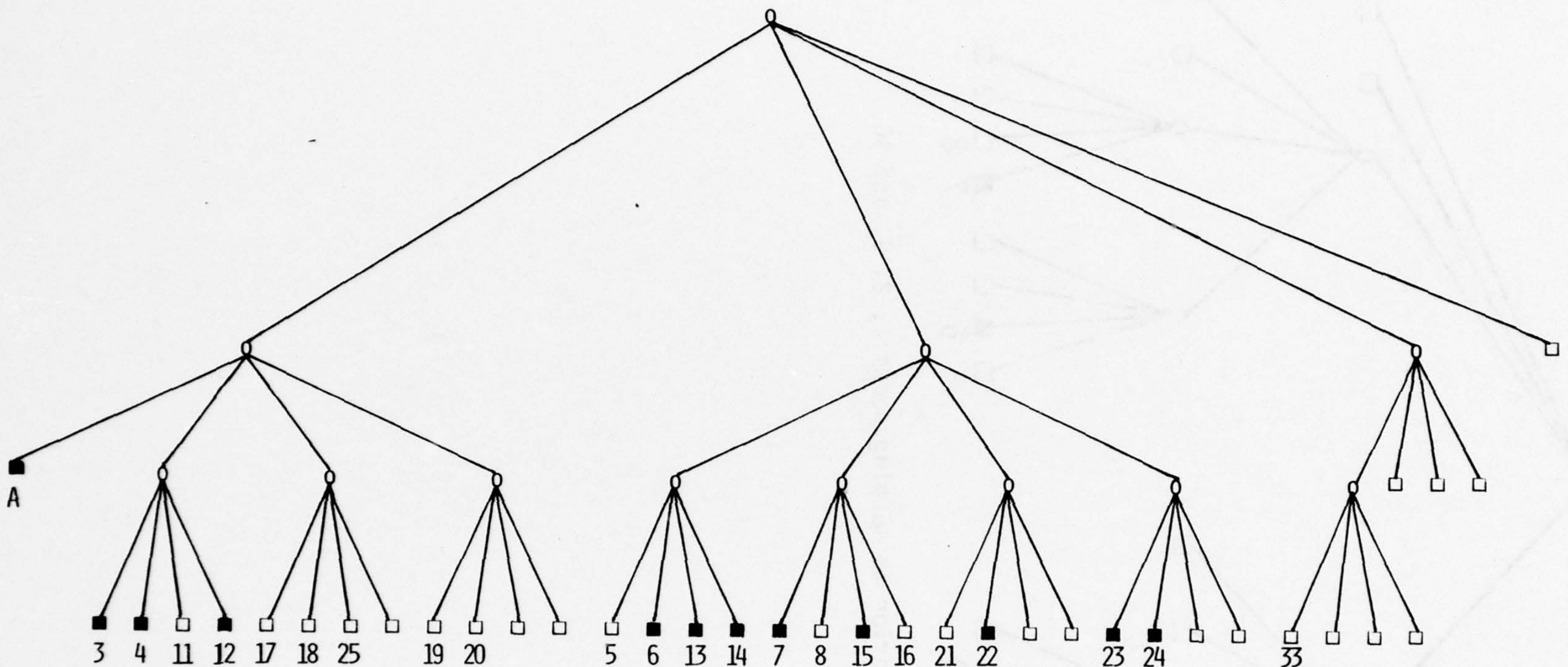


Figure 5. Quadtree at the start of processing the fourth row in Figure 1a.

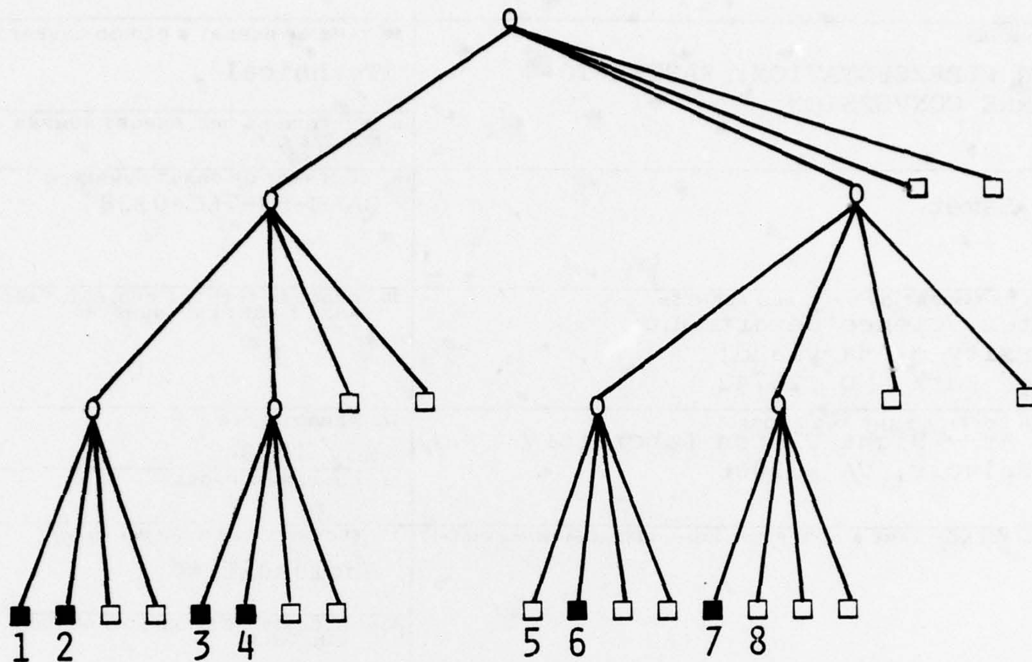


Figure 6. Quadtree after processing the first row in Figure 1a.

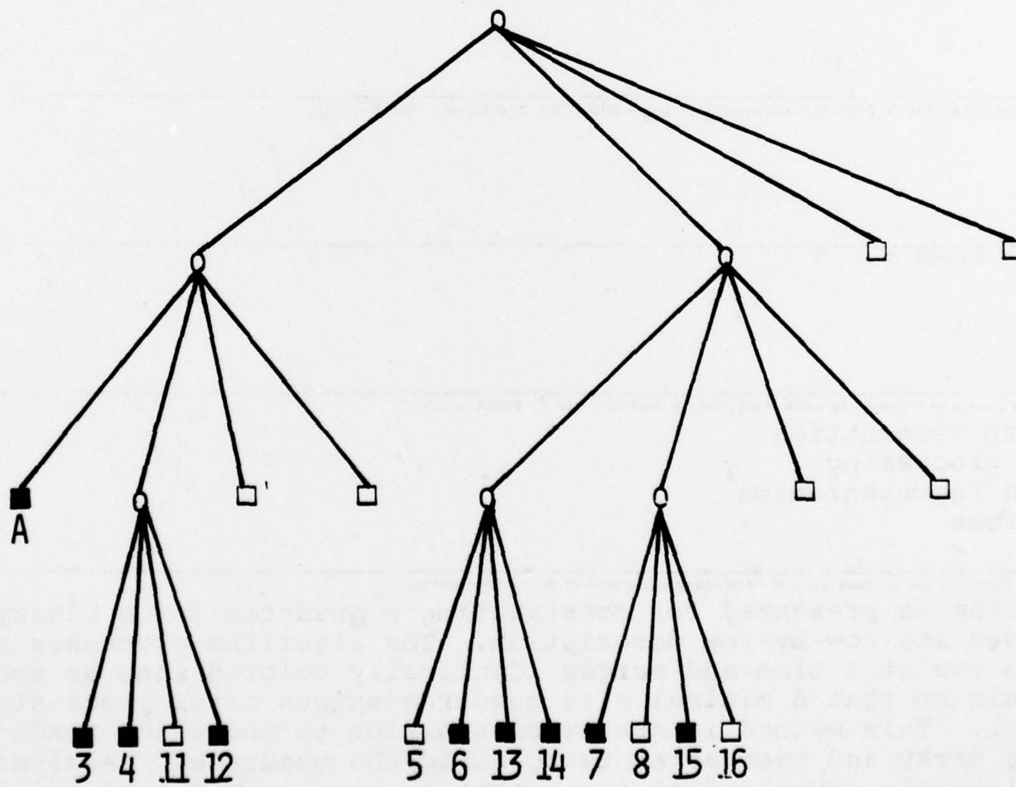


Figure 7. Quadtree after processing the second row in Figure 1a.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) REGION REPRESENTATION: RASTER-TO-QUADTREE CONVERSION		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-766
7. AUTHOR(s) Hanan Samet		8. CONTRACT OR GRANT NUMBER(s) DAAG-53-76C-0138
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department University of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Night Vision Laboratory Fort Belvoir, VA 22060		12. REPORT DATE May 1979
		13. NUMBER OF PAGES 30
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Pattern recognition
Image processing
Region representation
Quadtrees

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An algorithm is presented for constructing a quadtree for a binary image given its row-by-row description. The algorithm processes the image one row at a time and merges identically colored sons as soon as possible so that a minimal size quadtree exists after processing each pixel. This method is spacewise superior to one which reads in an entire array and then attempts to build the quadtree. Analysis of the algorithm reveals that its execution time is proportional to the number of pixels comprising the image.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)