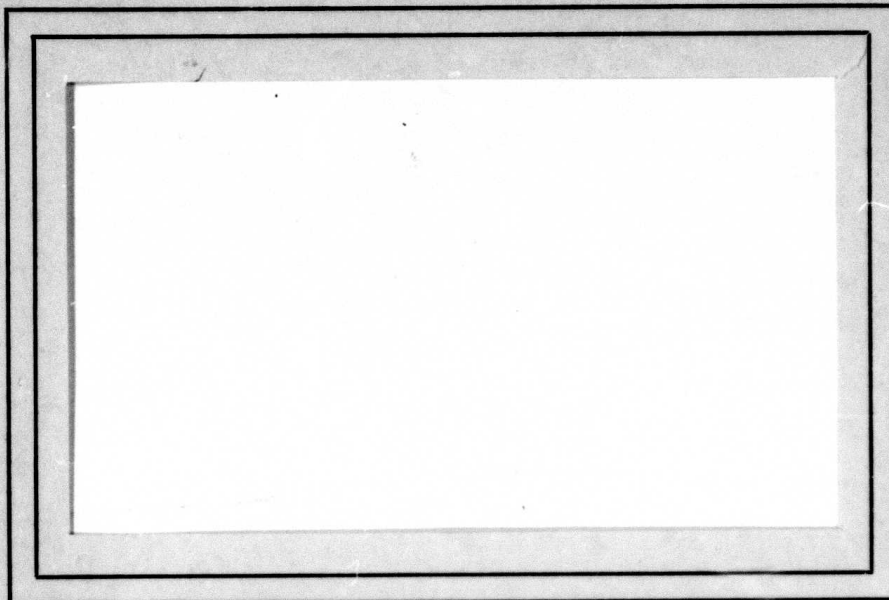


ADA 084288

①

LEVEL

II



UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

DTIC
ELECTE
S MAY 19 1980 D
E

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

80 5 19 120

FILE COPY

15 DAAG-53-76-C-0138,
DARPA Order-3206

12/28

14 TR-76B
DAAG-53-76C-0138

11 June 1979

6 REGION REPRESENTATION:
QUADTREE-TO-RASTER CONVERSION.

70 Hanan Samet
Computer Science Department
University of Maryland
College Park, MD 20742

9 Technical rept.

Accession For	
NTIS Grant	☒
DOC TAB	☐
Unannounced	☐
Justification	☐
By	
Distribution/	
Classification Codes	
Dist	Avail and/or special
A	

ABSTRACT

An algorithm is presented for obtaining a raster representation for an image given its quadtree. For each raster row the algorithm visits the appropriate nodes in the quadtree and, for each such node, outputs a run of length equal to the width of the corresponding block. Each block's node is visited as many times as it is high. Analysis of the algorithm reveals that its execution time is proportional to the sum of the heights of the blocks comprising the image. The total number of terminal and non-terminal nodes visited by the algorithm is also computed and shown to be a function of the number of maximal black and white blocks in the image. This means that the algorithm's execution time is directly proportional to the complexity of the image.

DISTRIBUTION STATEMENT A
Approved for public release
Distribution 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.

1409022

JCR

1. Introduction

In this paper we continue our investigation [DRS, Samet1, Samet2, Samet3, Samet4, Samet5] of region representation by use of quadtrees [Klinger]. This is an important question in applications such as image processing, computer graphics, and cartography (see [DRS] for a brief review of representations currently in use). In this paper we present an algorithm for obtaining a raster (i.e., row-by-row) representation given the quadtree representation of a binary image. Such conversion algorithms (e.g., [DRS, Samet1, Samet4, Samet5]) are useful because each representation is well suited for particular operations on the image. The quadtree is a compact hierarchical representation which, depending on the nature of the image, saves space as well as facilitates operations such as search. The algorithm described here is important because it enables a quadtree to be output by raster-like display devices without requiring space in excess of one row of pixels.

In the remainder of this section we briefly define the representations used. Sections 2-5 contain the algorithm and an analysis of its execution time. We also include a formal presentation of the algorithm using a variant of ALGOL 60 [Naur] as well as motivations for its various steps.

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 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, 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 0's respectively. Circular nodes, also termed GRAY nodes, denote non-terminal nodes.

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 $\text{SONTYPE}(P)=Q$ iff $\text{SON}(\text{FATHER}(P),Q)=P$. The sixth field, named NODETYPE, describes the contents of the block of the image which the node represents--i.e., BLACK, WHITE, or GRAY.

The four sides of a node's block are called its N, E, S, and W sides. They are also termed its boundaries. The predicate ADJ and the function REFLECT 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(W,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., $\text{REFLECT}(N,SW)=NW$, $\text{REFLECT}(E,SW)=SE$, $\text{REFLECT}(S,SW)=NW$, and $\text{REFLECT}(W,SW)=SE$. 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. Also we say that a block is of size 2^S if it is found at level S in the tree.

3. Algorithm

The quadtree-to-raster algorithm traverses the quadtree in a row by row order. For each row in the image, every BLACK or WHITE node corresponding to a block which intersects it is visited from left to right (e.g., for pixels 1-8 in the first row of Figure 1a, node A of Figure 1c is visited first followed by nodes B, C, D, and E). Each BLACK and WHITE node at level L in the tree is visited 2^L times (i.e., its height in pixels). The result of each such visit is that a run of length 2^L is output (e.g., a run of length 2 for node A of Figure 1b).

The main procedure is called RASTEROUTPUT and it 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). RASTEROUTPUT must first find the NW-most BLACK or WHITE node in the image, say P at level L (e.g., node A corresponding to block A in Figure 1b). Once this is done, it controls the exploration of all of the adjacent BLACK and WHITE nodes in the eastern direction along each of the 2^L rows corresponding to P . When all of the rows of P have been output, the process is repeated for P 's westernmost southern neighbor (e.g., the rows corresponding to node H are processed after being done with the rows associated with node A in Figure 1c). Procedure OUTROW is responsible for the exploration of the adjacencies along

each row. For each BLACK or WHITE node at level L that participates in a row, OUTPUTRUN outputs a run of length 2^L of the appropriate color (e.g., a BLACK run of length 2 for node A and row 1 of Figure 1b). OUTPUTENDOFROW outputs a separator symbol to mark the end of the row.

FIND_NEIGHBOR locates the smallest neighboring node of greater or equal size along a specified side. If the node is on the edge of the image, and no neighbor exists in the direction searched, then NULL is returned. This signals that either processing of a row is finished (e.g., the eastern neighbor of node E in Figure 1b) or that the raster output procedure is finished (e.g., the southern neighbor of node y in Figure 1b). If the neighboring node does exist, then a pointer to that node is returned. If it is a GRAY node (e.g., the eastern neighbor of node A in Figure 1b), then procedure FIND_ADJACENT is used to find the adjacent BLACK or WHITE node which intersects the row currently being processed (e.g., the eastern neighbor of node A in Figure 1b is B for the first row and F for the second row). Note that FIND_ADJACENT can also be invoked in the vertical direction. In such a case, we always seek the NW-most node of the adjacent nodes in the southern direction (e.g., node H is the NW-most node of the southern neighbors of node A in Figure 1b).

As an example of the application of the algorithm, consider the image given in Figure 1a. Figure 1b is the cor-

responding block decomposition and Figure 1c is the quadtree representation. Nodes R_i correspond to non-terminal nodes. The terminal nodes in Figure 1b have been labeled in the order in which they were visited for the first time. The result of the algorithm is the string W332, W242, W242, W17, W17, W17, W314, W8 where the comma serves as a separator symbol denoting the end of a row. The term W332 indicates that the first run is of length 3 and corresponds to WHITE, the second run is of length 3 and corresponds to BLACK, and the third run is of length 2 and corresponds to WHITE. When outputting the first row we start at node $R_\emptyset$ and then successively visit nodes $R_1, A, R_1, R_5, B, R_5, C, R_5, R_1, R_\emptyset, R_2, D, R_2, E, R_2, R_\emptyset$. For the second row we start at node A and successively visit nodes $R_1, R_5, F, R_5, G, R_5, R_1, R_\emptyset, R_2, D, R_2, E, R_2, R_\emptyset$. For the third row we must first get to node H--i.e., visit nodes R_1 and R_6 . The third row is output by successively visiting nodes $R_6, I, R_6, R_1, J, R_1, R_\emptyset, R_2, K, R_2, R_7, L, R_7, M, R_7, R_2, R_\emptyset$. For the fourth row we get to node N by visiting node R_6 and we output the row by visiting nodes $R_6, O, R_6, R_1, J, R_1, R_\emptyset, R_2, K, R_2, R_7, P, R_7, Q, R_7, R_2, R_\emptyset$. For the remainder of the image we successively visit nodes $R_6, R_1, R_\emptyset, R_3, R_8, R, R_8, S, R_8, R_3, T, R_3, R_\emptyset, R_4, U, R_4, V, R_4, R_\emptyset, R_8, W, R_8, X, R_8, R_3, T, R_3, R_\emptyset, R_4, U, R_4, V, R_4, R_\emptyset, R_8, R_3, Y, R_3, R_9, Z, R_9, AA, R_9, R_3, R_\emptyset, R_4, BB, R_4, CC, R_4, R_\emptyset, R_3, R_9, DD, R_9, EE, R_9, R_3, R_\emptyset, R_4, BB, R_4, CC, R_4, R_\emptyset, R_3, R_\emptyset$. In total 136 nodes have been visited. Note that in our discussion we assume no difference in the amount of work performed when visiting a node and when outputting a run.

```

procedure RASTEROUTPUT(P,LEVEL);
/* output a raster representation of the  $2^{\uparrow}LEVEL$  by  $2^{\uparrow}LEVEL$ 
   image corresponding to the quadtree rooted at node P */
begin
   node P,Q;
   integer LASTROW,LEVEL,ROW;
   while not NULL(SON(P,'NW')) do
      begin /* find NW-most block */
         LEVEL $\leftarrow$ LEVEL-1;
         P $\leftarrow$ SON(P,'NW');
      end;
   LASTROW $\leftarrow$ 0;
   do
      begin
         for ROW $\leftarrow$ LASTROW+1 step 1 until LASTROW+ $2^{\uparrow}LEVEL$ 
            do OUTROW(P,ROW,LEVEL);
         LASTROW $\leftarrow$ LASTROW+ $2^{\uparrow}LEVEL$ ;
         FIND_NEIGHBOR(P,'S',Q,LEVEL);
         if GRAY(Q) then FIND_ADJACENT(Q,0,LEVEL);
         P $\leftarrow$ Q;
      end
   until NULL(P);
end;

```

```

procedure OUTROW(P,ROW,L);
/* output a raster corresponding to row RUN starting with node
   P at level L */
begin
   node P,Q;
   integer L,ROW;
   do
     begin
       Q←P;
       OUTPUTRUN (NODETYPE(Q),L);
       FIND_NEIGHBOR(Q,'E',P,L);
       if GRAY(P) then FIND_ADJACENT(P,ROW,L);
     end
     until NULL(P);
   OUTPUTENDOFROW();
end;

```

procedure FIND_NEIGHBOR(P,S,Q,L);

/* given node P, return in Q the node which is adjacent to
side S of node P. Q is at level L in the tree */

begin

node P;

reference node Q;

side S;

reference integer L;

L←L+1;

if not NULL(FATHER(P)) and ADJ(S,SONTYPE(P)) then

/* find a common ancestor */

FIND_NEIGHBOR(FATHER(P),S,Q,L)

else Q←FATHER(P);

/* follow reflected path to locate the neighbor */

if not NULL(Q) and GRAY(Q) then

begin

Q←SON(Q,REFLECT(S,SONTYPE(P)));

L←L-1;

end;

end;

```
procedure FIND_ADJACENT(P,POS,L);
```

```
/* given GRAY node P, find a terminal node that contains row
   (column) POS and is an extreme western (northern) son of P.
   P and L will contain the extreme node and its level respectively.
   FIND_ADJACENT can be invoked in the horizontal direction for any row value;
   however, in the vertical direction it is only invoked for column 0.
   In the latter case, the extreme NW son is sought */
```

```
begin
```

```
  reference node P;
```

```
  integer POS,BASE;
```

```
  reference integer L;
```

```
  BASE←POS-POS mod 2↑L;
```

```
  do
```

```
    begin
```

```
      L←L-1;
```

```
      if BASE+2↑L POS then P←SON(P,'NW')
```

```
        /* horizontal or vertical extreme */
```

```
      else
```

```
        begin
```

```
          BASE←BASE+2↑L;
```

```
          P←SON(P,'SW') /* horizontal extreme */
```

```
        end;
```

```
    end
```

```
    until not GRAY(P);
```

```
end;
```

4. Analysis

The running time of the quadtree-to-raster conversion algorithm is measured by the number of nodes that are visited. Thus we only need to analyze the amount of time spent by procedures FIND_NEIGHBOR and FIND_ADJACENT. It is clear that the number of horizontal adjacencies that are explored is equal to the sum of the heights of the blocks comprising the image. This is true because each block will be visited once for each row in which it is a member.

Our analysis first assumes a simpler algorithm than the one given in Section 3. The difference is that we do not use FIND_NEIGHBOR in the vertical direction. Instead, we traverse each row by starting at the root of the tree (e.g., in Figure 1, once the first row has been processed, we return to the root of the tree, R_0 , relocate node A and process the second row (i.e., revisit nodes R1 and A), then return to R_0 and locate node H,...). This will simplify the analysis although we will also show that our original algorithm at times will yield a superior result. Assume that the image is a 2^n by 2^n array of pixels. If the image is a complete quadtree, i.e., all blocks in the image are at level 0, then we have the following lemma:

Lemma 1: In a complete quadtree, the number of nodes visited by the modified quadtree-to-raster algorithm is bounded by

4 times the number of blocks in the image (more precisely, it is equal to four times the difference between the area and the diameter of the image).

Proof: Starting at the root node of the quadtree, for each row in a 2^n by 2^n image, n nodes are visited when locating the node corresponding to the leftmost block. Once this is done, FIND_NEIGHBOR is invoked 2^n times to find neighbors in the eastern direction. 2^0 of the nodes corresponding to blocks in the row have their nearest common ancestor at level n , 2^1 at level $n-1, \dots, 2^i$ at level $n-i$, and 2^{n-1} at level 1. Once the nearest common ancestor has been found, a path of equal length must be traversed to locate the adjacent neighbors. In addition, the node corresponding to the rightmost block in each row has no eastern neighbor. This is detected by attempting to locate a non-existent common ancestor--a process which traverses a path of length n (i.e., to the root of the quadtree and including it). Since there are 2^n rows, the number of rows visited is:

$$\begin{aligned}
 2^n \left(n+2 \sum_{i=1}^n i 2^{n-i} + n \right) &= 2^n \left(2n+2^{n+1} \sum_{i=1}^n \frac{i}{2^i} \right) \\
 &= 2^n \left(2n+2^{n+1} \left(2 - \frac{n+2}{2^n} \right) \right) \\
 &= 2^n \left(2n+2 \left(2^{n+1} - (n+2) \right) \right) \\
 &= 2^{n+1} \cdot n + 2^{2n+2} - 2^{n+1} \cdot n - 2^{n+2} \\
 &= 2^{2n+2} - 2^{n+2}
 \end{aligned}$$

However, there are 2^{2n} blocks in the image. Thus the number of nodes that have been visited is bounded by 4 times the number of blocks in the image.

Q.E.D.

We are now ready to prove the main result.

Theorem 1: For any image, the number of nodes visited by the modified quadtree-to-raster algorithm is bounded by 4 times the sum of the heights of the blocks in the image.

Proof: By Lemma 1 the theorem is true for a complete quadtree. We shall use induction on the size of the blocks to show the result for any quadtree.

Consider a 2 by 2 pixel block in the complete quadtree and assume that the four blocks corresponding to the pixels have been merged to yield one block. Since we are processing the image in a row by row manner, the only adjacencies that are eliminated by the merge are the horizontal ones between the blocks being merged (e.g., between 19 and 20 and 27 and 28 in Figure 1a). This means that four less nodes will be visited by our algorithm. In addition, the node corresponding to the merged block (e.g., node J for the blocks corresponding to 19, 20, 27, and 28 in Figure 1a and 1b) is 1 node closer to its horizontal neighbors to the left and right (e.g., blocks 18, 26, 21, and 29 in Figure 1a). Recall that in general, the left and right edges of the image are also neighbors.

Thus we find that $4+1+1+1+1=8$ nodes less will be visited. However, the size of the blocks in the image has decreased by 2 (initially there were 4 blocks of size 1 and now there is one block of size 2) and our theorem holds.

More generally, consider a 2^{S+1} by 2^{S+1} block--i.e., we are merging four 2^S by 2^S blocks. Once again, since we are processing the image in a row by row manner, the only adjacencies that are eliminated by the merge are the horizontal ones between the blocks being merged. Since the blocks are of size 2^S , 2^{S+1} adjacencies are eliminated. Moreover, each block of size 2^S is at a distance of 2 from its horizontal neighbor with whom it is being merged. Thus the elimination of 2^{S+1} adjacencies results in $2 \cdot 2^{S+1}$ less nodes being visited by the algorithm. In addition, the node corresponding to the merged block is 1 node closer to its horizontal neighbors to the left and right. However, there are 2^{S+1} neighbors in each direction. Thus the total number of nodes that will be visited has decreased by $4 \cdot 2^{S+1}$. However, the size of the blocks in the image has decreased by 2^{S+1} (initially there were 4 blocks of size 2^S and now there is one block of size 2^{S+1}) and our theorem holds.

Q.E.D.

The construction used in the proof of Theorem 1 is worthy of further attention. It can be used in conjunction with the result of Lemma 1 to compute exactly how many nodes will be

visited for any image given the number of blocks comprising it and their respective sizes. Thus different images will require the same number of node visits. For example, the image in Figure 3 requires the same number of node visits as the image in Figure 1 when the modified algorithm is applied. Intuitively, this is not surprising since the modified algorithm does process the rows independently of each other.

We have the following theorem:

Theorem 2: Given a 2^n by 2^n image with b_i blocks of size 2^i , the number of nodes visited by the modified quadtree-to-raster algorithm is:

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

Proof: From the proof of Lemma 1 we have that traversing a complete quadtree of size 2^i requires $2^{2i+2} - 2^{i+2}$ nodes to be visited. This represents a traversal starting and terminating at a common ancestor. Since the 2^i by 2^i array of pixels has been replaced by one block, $2^{2i+2} - 2^{i+2}$ less nodes will be visited for a block of size 2^i . Recall that if the array contains 2^n by 2^n blocks of size 1, then $2^{2n+2} - 2^{n+2}$ nodes will be visited. Subtracting the contribution of b_i blocks of size 2^i yields the desired result.

Q.E.D.

At this point we return to our original algorithm (i.e., the one presented in Section 3) and compare it with the modified algorithm. Recall that the only difference between the two algorithms was in the way they handled the vertical transitions between rows. For a complete quadtree for a 2^n by 2^n image, the modified algorithm required $n \cdot 2^n$ node visits to locate the first block in each row whereas the original algorithm required a number of nodes to be visited equal to that obtained by Lemma 1 for a single row--i.e., $2^{n+2}-4$. Thus the original algorithm requires $2^{2n+2}-2^{n+2}-n \cdot 2^{n+2}+2^{n+2}-4=2^{2n+2}-n \cdot 2^{n+2}-4$ node visits. We have the following results:

Theorem 3: Given a 2^n by 2^n image with v_i blocks of size 2^i in the first column, the number of nodes visited by the original quadtree-to-raster algorithm in locating the nodes corresponding to the first blocks in each column is

$$2^{n+2}-4 - \sum_{i=1}^n v_i (2^{i+2}-4)$$

Proof: From Lemma 1 we have that traversing a row of size 2^i pixels in a 2^i by 2^i image requires $2^{i+2}-4$ nodes to be visited. This represents a traversal starting and terminating at a common ancestor. Since the 2^i pixels have been replaced by one block, $2^{i+2}-4$ fewer nodes will be visited for a block of size 2^i . Recall that if the row contains 2^n blocks of size 1, then $2^{n+2}-4$ nodes will be visited. Subtracting the contribution of v_i blocks of size 2^i yields the desired result.

Q.E.D.

Theorem 4: Given a 2^n by 2^n image with v_i blocks of size 2^i in the first column, the number of nodes visited by the modified quadtree-to-raster algorithm in locating the nodes corresponding to the first blocks in each column is

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

Proof: For each row in a block of size 2^i , the leftmost node is i nodes closer to the root of the quadtree than it would be were it in a block of size 1. Since there are v_i such blocks, the result follows.

Q.E.D.

Theorems 3 and 4 enable us to compare the original and modified algorithms. We find that for blocks of size 2^i where $i \geq 4$, the ratio $\frac{i2^i}{2^{i+2}-4}$ is greater than 1. Thus neither algorithm is always superior to the other one. However, as n gets large and a majority of the nodes appear at low levels in the tree, it would appear that the original algorithm is superior. Note that for the example in Figure 1 both the original and modified algorithms will visit the same number of nodes (i.e., 136).

We may also obtain a result analogous to Theorem 2 for the original algorithm.

Theorem 5: Given a 2^n by 2^n image with b_i blocks of size 2^i , and v_i blocks of size 2^i in the first column, the number of nodes

visited by the original quadtree-to-raster algorithm is:

$$2^{2n+2} - n \cdot 2^{n+4} - \sum_{i=1}^n b_i (2^{2i+2} - 2^{i+2}) + \sum_{i=1}^n v_i ((i-4) 2^{i+4})$$

Proof: Subtract the result of Theorem 4 from the sum of the results of Theorems 2 and 3.

Q.E.D.

Thus we see that the original algorithm also has a degree of configuration independence. The difference between it and the modified algorithm is that in its case the first columns must have the same number of blocks of the different sizes. Thus Figures 1 and 3 do not result in the same number of nodes being visited whereas Figures 1 and 4 do.

5. Concluding Remarks

An algorithm has been presented for converting a quadtree representation of a binary image to a raster representation of the image. The algorithm's running time was shown to be proportional to the sum of the heights of the blocks comprising the image. In other words, the amount of work required is directly proportional to the complexity of the image--i.e., two different images will require the same amount of work if they have the same number of blocks of each size. This is not surprising when we recall the row by row nature of our algorithm. Note that the complexity of the algorithm is directly proportional to the resolution of the image--i.e., as the resolution is increased by a factor of 2, so does the sum of the heights of the blocks comprising the image. Also observe that the order of the algorithm's running time ranges between the diameter of the image (i.e., when the image consists of one node) and the area of the image (i.e., when all of the blocks in the image are of unit pixel size).

The analysis of the algorithm was facilitated by considering a somewhat simplified version which nevertheless was shown to exhibit superior behavior for some images. The simplified version rendered possible the proof of Theorem 1. The original algorithm has the same upper bound of 2^{2n+2} nodes being visited; however, the result of Theorem 1 is not as easily derivable.

The problem with the original algorithm is that the number of nodes visited is sometimes only reduced by 3 rather than by 4 when blocks in the first column are merged.

The algorithm is essentially a bottom-up tree traversal. It can be contrasted with two other approaches. One method traverses the tree in a top down manner looking for extreme nodes (e.g., in the northern direction) and works its way to the bottom of the image. An alternative approach avoids the work required by FIND_NEIGHBOR and FIND_ADJACENT in locating neighboring nodes for each row in the image by linking such nodes. This is similar to the concept of roping [Hunter]. The disadvantage of such a technique is the amount of extra space required to store the links. In addition, as we have shown, the cost of our neighbor finding techniques is not very high (i.e., four nodes must be visited per adjacency rather than 1 as is the case when the nodes are linked).

6. References

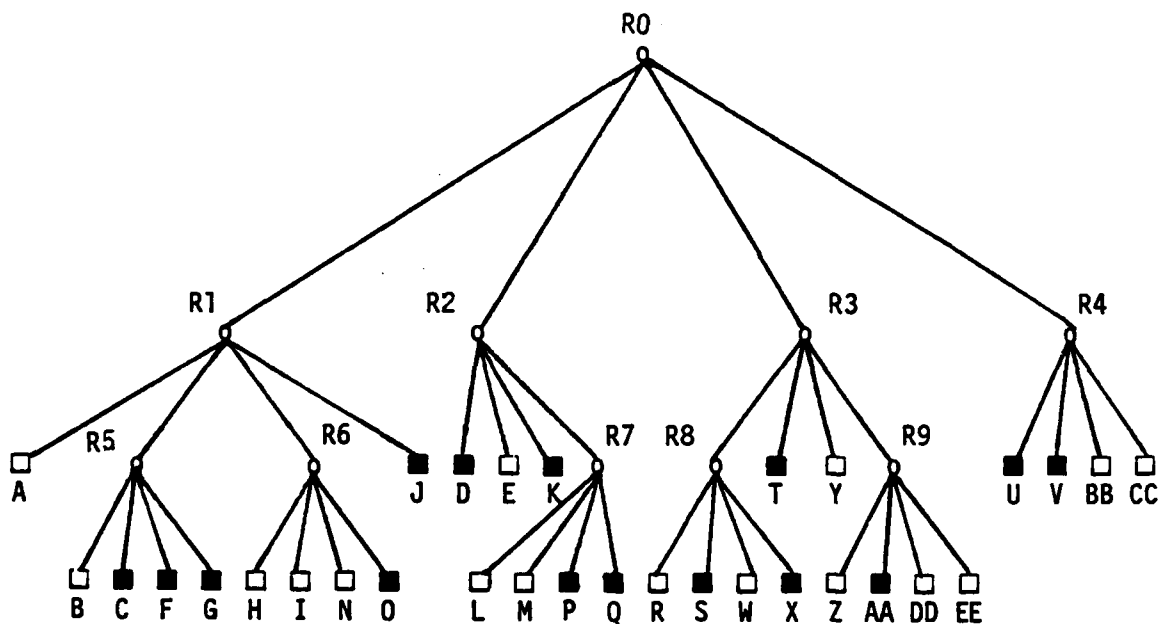
- [DRS] C. R. Dyer, A. Rosenfeld, and H. Samet, Region representation: boundary codes from quadtrees. Computer Science TR-732, University of Maryland, College Park, Maryland, February 1979.
- [Hunter] G. M. Hunter, Efficient computation and data structures for graphics, Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey, 1978.
- [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.
- [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 quad-trees, Computer Science TR-756, University of Maryland, College Park, Maryland, April 1979.
- [Samet4] H. Samet, Region representation: raster-to-quadtree conversion, Computer Science TR-766, University of Maryland, College Park, Maryland, May 1979.
- [Samet5] 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	B	C	D	E
H	I	F	G	L
N	O	J	K	P
R	S	T	U	V
W	X	AA	BB	CC
Y	DD	EE		

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.

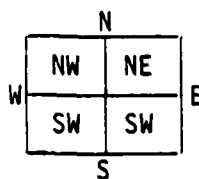
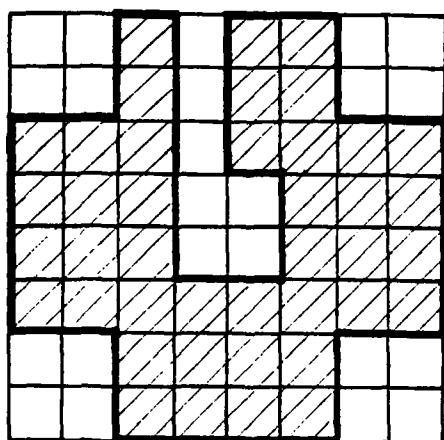
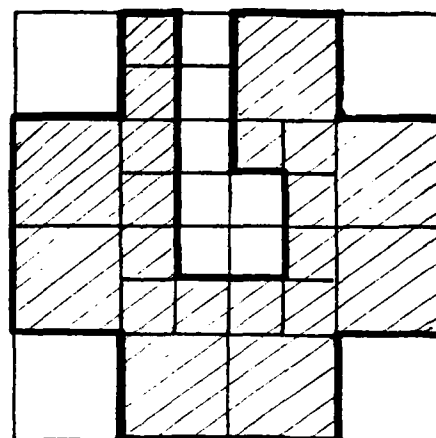


Figure 2. Relationship between a block's four quadrants and its boundaries.

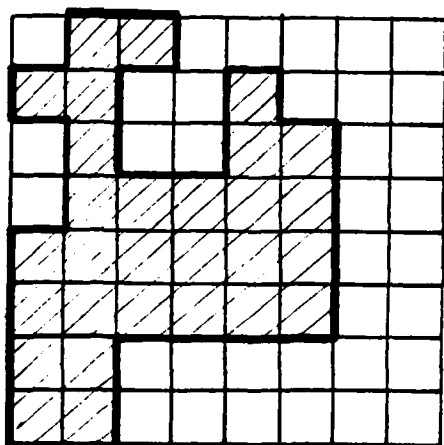


a. Sample image

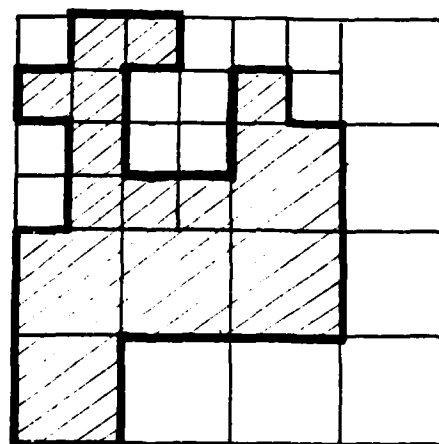


b. Block decomposition of the image in (a).

Figure 3. An image and its maximal blocks which requires the same number of nodes to be visited by the modified quadtree-to-raster algorithm as does Figure 1.



a. Sample image



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

Figure 4. An image and its maximal blocks which requires the same number of nodes to be visited by the original quadtree-to-raster algorithm as does Figure 1.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A084288	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) REGION REPRESENTATION: QUADTREE-TO- RASTER CONVERSION		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-768
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 June 1979
		13. NUMBER OF PAGES 27
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 Computer graphics Quadrees		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) An algorithm is presented for obtaining a raster representation for an image given its quadtree. For each raster row the algorithm visits the appropriate nodes in the quadtree and, for each such node, outputs a run of length equal to the width of the corresponding block. Each block's node is visited as many times as it is high. Analysis of the algorithm reveals that its execution time is proportional to the sum of the heights of the blocks comprising the image. The total number of terminal and non-terminal nodes visited by the		

DD FORM 1473

JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

algorithm is also computed and shown to be a function of the number of maximal black and white blocks in the image. This means that the algorithm's execution time is directly proportional to the complexity of the image.



Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)