COMPUTER GRAPHICS AND IMAGE PROCESSING 18, 37–57 (1982)

Neighbor Finding Techniques for Images Represented by Quadtrees*

HANAN SAMET

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

Received June 23, 1980; revised June 15, 1981.

Image representation plays an important role in image processing applications. Recently there has been a considerable interest in the use of quadtrees. This has led to the development of algorithms for performing image processing tasks as well as for performing converting between the quadtree and other representations. Common to these algorithms is a traversal of the tree and the performance of a given computation at each node. These computations typically require the ability to examine adjacencies between neighboring nodes. Algorithms are given for determining such adjacencies in the horizontal, vertical, and diagonal directions. The execution times of the algorithms are analyzed using a suitably defined model.

1. INTRODUCTION

Region representation is an important aspect of image processing with numerous representations finding use. Recently, there has emerged a considerable amount of interest in the quadtree [3–8, 11]. This stems primarily from its hierarchical nature, which lends itself to a compact representation. It is also quite efficient for a number of traditional image processing operations such as computing perimeters [14], labeling connected components [13], finding the genus of an image [1], and computing centroids and set properties [18]. Development of algorithms to convert between the quadtree representation and other representations such as chain codes [2, 10], rasters [12, 17], binary arrays [11], and medial axis transforms [15, 16, 19] lend further support to this importance.

In this paper we discuss methods for moving between adjacent blocks in the quadtree. We first show how transitions are made between blocks of equal size and then generalize our result to blocks of different size, where the destination block is either of larger or smaller size than the source block. Such blocks are termed neighbors. Note that the transitions that we discuss also include those along diagonal, as well as horizontal and vertical, directions. The importance of these methods lies in their being the cornerstone of many of the quadtree algorithms (e.g., [1, 2, 10, 12–19]), since they are basically tree traversals with a "visit" at each node. More often than not these visits involve probing a node's neighbors. The significance of our methods lies in the fact that they do not use coordinate information, knowledge of the size of the image, or storage in excess of that imposed by the nature of the quadtree data structure.

*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 and Pat Young in preparing the figures. I have benefitted greatly from discussions with Azriel Rosenfeld.

0146-664X/82/010037-21$02.00/0

2. DEFINITIONS AND NOTATION

The quadtree is an approach to image representation based on the successive subdivision of the image into quadrants. It is represented by a tree of outdegree 4 in which the root represents a block and the four sons represent in order the NW, NE, SW, and SE quadrants. We assume that each node is stored as a record containing six fields. The first five fields contain pointers to the node's father and its four sons, which correspond to the four quadrants. If P is a node and I is a quadrant, then these fields are referenced as FATHER(P) and SON(P, I), respectively. We can determine the specific quadrant in which a node, say P, lies relative to its father by use of the function SONTYPE(P), which has a value of I if SON(FATHER(P), I) = P. The sixth field, NODETYPE, describes the contents of the block of the image which

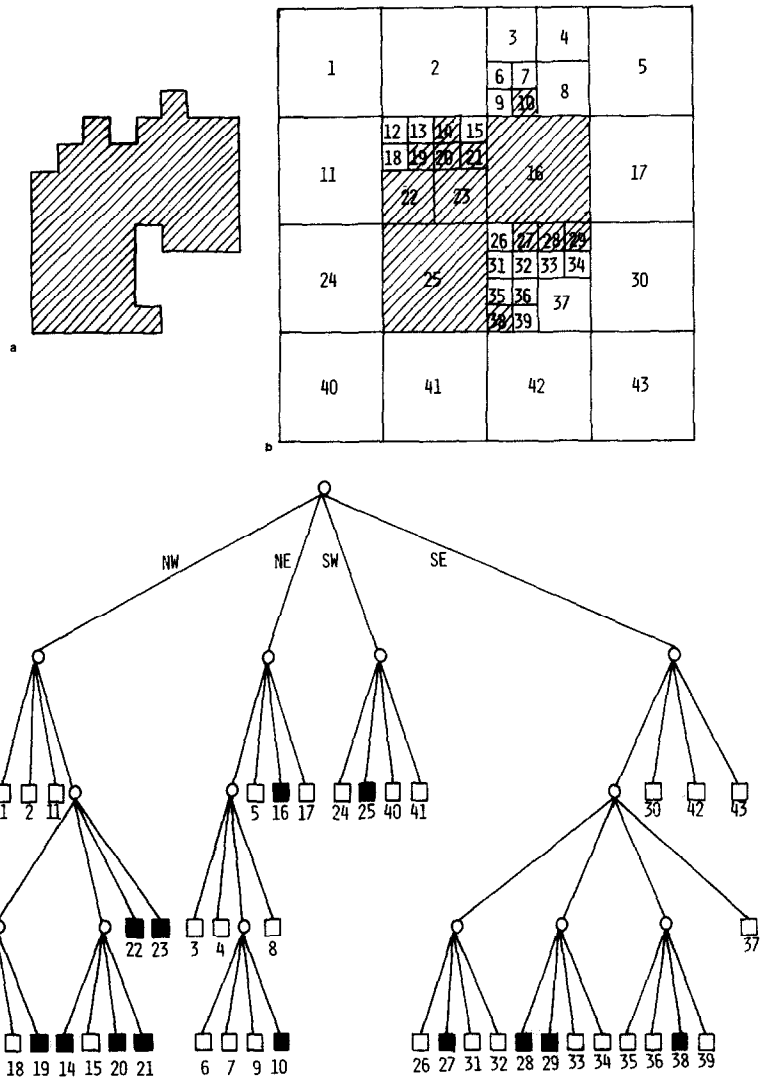

FIG. 1. A region, its maximal blocks, and the corresponding quadtree. Blocks in the region are shaded, background blocks are blank. (a) Region. (b) Block decomposition of the region in (a). (c) Quadtree representation of the blocks in (b).

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 nodes are terminal nodes, while GRAY nodes are nonterminal nodes. For example, Fig. 1b is a block decomposition of the region in Fig. 1a while Fig. 1c is the corresponding quadtree.

Let the four sides of a node's block be called its N, E, S, and W sides. They are also termed its boundaries and at times we speak of them as if they are directions. We define the following predicates and functions to aid in the expression of operations involving a block's quadrants and its 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, SW) is true. REFLECT(B, I) yields the SONTYPE value of the block of equal size that is adjacent to side B of a block having SONTYPE value I, e.g., REFLECT(N, SW) = NW. COMMONSIDE(Q1, Q2) indicates the boundary of the block containing quadrants Q1 and Q2 that is common to them; e.g., COMMONSIDE(SW, NW) = W. If Q1 and Q2 are not adjacent brother quadrants (e.g., NE and SW) or if Q1 and Q2 are the same, then the value of COMMONSIDE is undefined. OPQUAD(Q) is the quadrant which does not share a block boundary with quadrant Q; e.g., OPQUAD(SW) = NE. Figure 2 shows the relationship between the quadrants of a node and its boundaries while Tables 1–4 contain the definitions of the ADJ, REFLECT, OPQUAD, and COMMONSIDE relationships respectively. Ω corresponds to an undefined value.

For a quadtree corresponding to a $2^n \times 2^n$ 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 a level ≥ 0. A

ADJ S \ Q	NW	NE	SW	SE
N	T	T	F	F
E	F	T	F	T
S	F	F	T	T
W	T	F	T	F

Table 1. ADJ(S,Q)

REFLECT S \ Q	NW	NE	SW	SE
N	SW	SE	NW	NE
E	NE	NW	SE	SW
S	SW	SE	NW	NE
W	NE	NW	SE	SW

Table 2. REFLECT(S,Q)

Q	OPQUAD(Q)
NW	SE
NE	SW
SW	NE
SE	NW

Table 3. OPQUAD(Q)

COMMONSIDE

Q1 \ Q2	NW	NE	SW	SE
NW	Ω	N	W	Ω
NE	N	Ω	Ω	E
SW	W	Ω	Ω	S
SE	Ω	E	S	Ω

Table 4. COMMONSIDE(Q1,Q2)

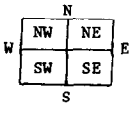

FIG. 2. Relationship between a block's four quadrants and its boundaries.

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. NEIGHBOR FINDING ALGORITHMS

Given a node corresponding to a specific block in the image, its neighbor of equal size in the horizontal or vertical direction is determined by locating a common ancestor. Next, we retrace the path while making mirror image moves about an axis formed by the common boundary between the blocks associated with the two nodes. The common ancestor is simple to determine—e.g., to find an eastern neighbor, the common ancestor is the first ancestor node which is reached via its NW or SW son. For example, the eastern neighbor of node A in Fig. 3a is G. It is located by ascending the tree until the common ancestor, D, is found. This requires going through a NE link to reach B, a NE link to reach C, and a NW link to reach D. Node G is now reached by backtracking along the previous path with the appropriate mirror image moves. This requires descending a NE link to reach E, a NW link to reach F, and a NW link to reach G. Figures 3a and b show how the eastern neighbor of node A is located. The algorithm for locating an equal sized neighbor in a given horizontal or vertical direction is given below using a variant of ALGOL 60 [9]. Note that we assume that the neighbor in the specified direction does indeed exist (i.e., we are not on the border of the image).

```
node procedure EQUAL_ADJ_NEIGHBOR(P, D);
/* Locate an equal-sized neighbor of node P in horizontal or vertical direction D */
begin
  value node P;
  value direction D;
  return (SON(if ADJ(D, SONTYPE(P)) then
                EQUAL_ADJ_NEIGHBOR(FATHER(P), D)
              else FATHER(P),
              REFLECT(D, SONTYPE(P))));
end;
```

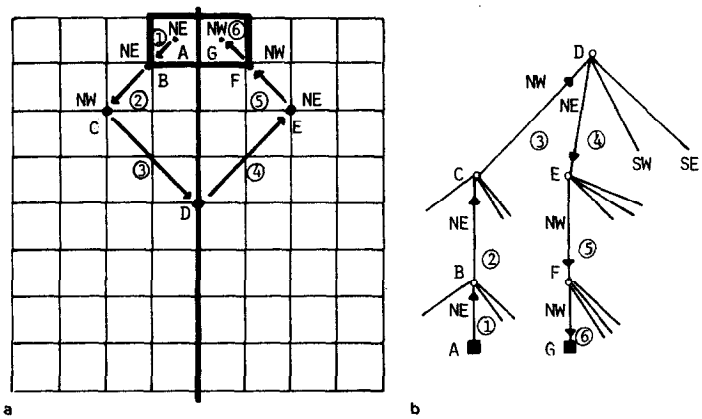

FIG. 3. Process of locating the eastern neighbor of node A (i.e., G). (a) Block decomposition. (b) Tree representation.

Finding a node's neighbor in the diagonal direction (i.e., its corresponding block touches the given node's block at a corner) is more complex. Given a node corresponding to a specific block in the image, its neighbor of equal size in a diagonal direction is determined by a three-step process. First, we locate the given node's nearest ancestor who is also adjacent (horizontally or vertically) to an ancestor of the sought neighbor. Next, we make use of EQUAL_ADJ_NEIGHBOR to access the ancestor of the sought neighbor in the direction of the adjacency. Finally, we retrace the remainder of the path while making directly opposite moves (i.e., 180° opposite so that a NW move becomes a SE move). The nearest ancestor of the first step is the first ancestor which is not reached by a link equal to the direction of the desired neighbor—e.g., to find a SE neighbor, the nearest such ancestor is the first ancestor node that is not reached via its SE son. For example, the SE neighbor of node A in Fig. 4a is G. It is located by ascending the tree until the nearest ancestor, B, which is also adjacent horizontally (in this case) to an ancestor of G, i.e., F, is found. This requires going through a NE link to reach B. Node F is now reached by applying EQUAL_ADJ_NEIGHBOR in the direction of the adjacency (i.e., east). This forces us to go through a NE link to reach C and a NW link to reach D. Backtracking results in descending a NW link to reach E and a NW link to reach F. Finally, we backtrack along the remainder of the path making 180° moves—i.e., we descend a SW link to reach G. Figures 4a and b show how the SE neighbor of node A is located. Note that, at times, EQUAL_ADJ_NEIGHBOR may not need to be applied. This is the case when the nearest ancestor of the first step is reached by a link equal to a direction opposite that of the desired neighbor (e.g., in Fig. 1, the SW neighbor of node 16 is 25 with the nearest ancestor of step 1 being node A). The algorithm for locating an equal size neighbor in a given diagonal direction is given below. Once again, we assume that the neighbor in the specified direction does indeed exist (i.e., we are not on the border of the image).

```
node procedure EQUAL_CORNER_NEIGHBOR(P, C);
/ *Locate an equal-sized neighbor of node P in the direction of quadrant C */
begin
  value node P;
  value quadrant C;
  return(SON(if SONTYPE(P) = OPQUAD(C) then FATHER(P)
             else if SONTYPE(P) = C then
               EQUAL_CORNER_NEIGHBOR(FATHER(P), C)
             else EQUAL_ADJ_NEIGHBOR(
               FATHER(P),
               COMMONSIDE(SONTYPE(P), C)),
             OPQUAD(SONTYPE(P))));
end;
```

It is often the case that neighbors are of different sizes. In such a case, we say that we want the neighboring terminal nodes having equal or greater size (e.g., the eastern neighbor of node 23 in Fig. 1 is 16). If such a node does not exist, then we return a GRAY node of equal size if possible (e.g., the northern neighbor of node 23 in Fig. 1

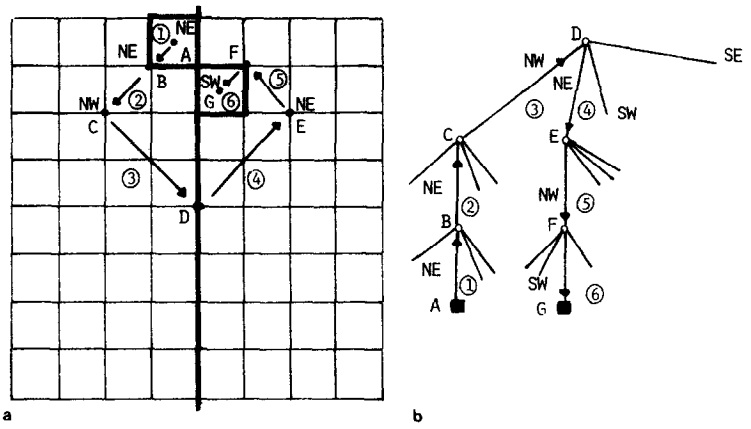

FIG. 4. Process of locating the SE neighbor of node A (i.e., G). (a) Block decomposition. (b) Tree representation.

is J). Otherwise the node is adjacent to the border of the image (not the region) and NULL is returned since there is no neighbor in the specified direction (e.g., the northern neighbor of node 2 in Fig. 1 is NULL). When a node does not have a neighboring terminal node of equal or greater size, returning a GRAY node of equal size is reasonable because the given node whose neighbor is being sought has more than one neighboring terminal node in the given direction. The algorithms for locating neighbors of equal or greater size in horizontal and vertical directions as well as diagonal directions are given below using procedures GTEQUAL_ADJ_NEIGHBOR and GTEQUAL_CORNER_NEIGHBOR respectively. Note that a neighbor in a diagonal direction, say C, will not always abut against corner C of the node whose neighbor is sought (e.g., node 16 is a nonabutting NE neighbor of node 23 in Fig. 1).

```
node procedure GTEQUAL_ADJ_NEIGHBOR(P, D);
/* Locate a neighbor of node P in horizontal or vertical direction D. If such a node
   does not exist, then return NULL */
begin
   value node P;
   value direction D;
   node Q;
   if not NULL(FATHER(P)) and ADJ(D, SONTYPE(P)) then
      /* Find a common ancestor */
      Q ← GTEQUAL_ADJ_NEIGHBOR(FATHER(P), D)
   else Q ← FATHER(P);
   /* Follow the reflected path to locate the neighbor */
   return (if not NULL(Q) and GRAY(Q) then
            SON(Q, REFLECT(D, SONTYPE(P)))
            else Q);
end;
node procedure GTEQUAL_CORNER_NEIGHBOR(P, C);
/* Locate a neighbor of node P in the direction of quadrant C. If such a node does
   not exist, then return NULL */
```

```
begin
  value node P;
  value quadrant C;
  node Q;
  if not NULL(FATHER(P)) and SONTYPE(P) ≠ OPQUAD(C) then
    /* Find a common ancestor */
    if SONTYPE(P) = C then
      Q ← GTEQUAL_CORNER_NEIGHBOR(FATHER(P), C)
    else Q ← GTEQUAL_ADJ_NEIGHBOR(
              FATHER(P),
              COMMONSIDE(SONTYPE(P), C))
  else Q ← FATHER(P);
  /* Follow opposite path to locate the neighbor */
  return (if not NULL(Q) and GRAY(Q) then SON(Q, OPQUAD(SONTYPE(P)))
          else Q);
end;
```

If neighbors are of different sizes, we may wish to know the size of the adjacent or abutting neighbor. In such a case, we want our neighbor finding algorithms to return both a pointer to the neighboring node and a value from which the node's size can be easily computed. This is relatively straightforward when we know the level in the tree at which is found the node whose neighbor is being sought. In fact, such an algorithm need only increment the level counter by 1 for each link that is ascended while locating the common ancestor, and then decrement the level counter by 1 for each link that is descended while locating the appropriate neighbor. The algorithms for locating neighbors of equal or greater size, with their corresponding level positions, in horizontal and vertical directions as well as diagonal directions, are given below using procedures GTEQUAL_ADJ_NEIGHBOR2 and GTEQUAL_CORNER_NEIGHBOR2, respectively. Note the use of reference parameters to transmit and return results. An alternative is to define a record of type **block** having two fields of type **node** and **integer**, whose values are a pointer to the neighboring node and its level, respectively.

```
procedure GTEQUAL_ADJ_NEIGHBOR2(P, D, Q, L);
/* Return in Q the neighbor of node P in horizontal or vertical direction D. 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. If such a node does not exist, then
  return NULL */
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_NEIGHBOR2(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;

procedure GTEQUAL_CORNER_NEIGHBOR2(P, C, Q, L);
/* Return in Q the neighbor of node P in the direction of quadrant C. 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. If such a node does not exist, then return
    NULL */
begin
  value node P;
  value quadrant C;
  reference node Q;
  reference integer L;
  L ← L + 1;
  if not NULL(FATHER(P)) and SONTYPE(P) ≠ OPQUAD(C) then
    /* Find a common ancestor */
    if SONTYPE(P) = C then
      GTEQUAL_CORNER_NEIGHBOR2(FATHER(P), C, Q, L)
    else GTEQUAL_ADJ_NEIGHBOR2(
        FATHER(P),
        COMMONSIDE(SONTYPE(P),C), Q, L)
  else Q ← FATHER(P);
  /* Follow the opposite path to locate the neighbor */
  if not NULL(Q) and GRAY(Q) then
    begin
      Q ← SON(Q, OPQUAD(SONTYPE(P)));
      L ← L − 1;
    end;
end;
```

At times we may wish to locate an adjacent horizontal or vertical neighbor regardless of its size. In such a case, we also specify a corner of the block corresponding to the node whose neighbor is being sought. The neighbor node must be adjacent to this corner (e.g., node 21 is the northern neighbor of node 23 which is adjacent to the NE corner of node 23). The algorithm for computing such a neighbor is given below by procedure CORNER_ADJ_NEIGHBOR, which makes use of GTEQUAL_ADJ_NEIGHBOR.

```
node procedure CORNER_ADJ_NEIGHBOR(P, D, C);
/* Locate a neighbor of node P in horizontal or vertical direction D which
    is adjacent to corner C of node P. If such a node does not exist, then return
    NULL */
begin
  value node P;
```

```
  value direction D;
  value quadrant C;
  P ← GTEQUAL_ADJ_NEIGHBOR(P, D);
  while GRAY(P) do P ← SON(P, REFLECT(D, C)); /* Descend to the desired
    corner */
  return (P);
end;
```

Similarly, in the case of a diagonal neighbor, we may also wish to locate the neighbor in the given direction regardless of its size (e.g., node 20 is a NE neighbor of node 22 in Fig. 1 that is smaller in size). The algorithm for locating an arbitrary-sized diagonal neighbor is given below by procedure CORNER_CORNER _NEIGHBOR which makes use of GTEQUAL_CORNER_NEIGHBOR.

```
node procedure CORNER_CORNER_NEIGHBOR(P, C);
/* Locate a neighbor of node P in the direction of quadrant C that abuts against
  corner C of node P. If such a node does not exist, then return NULL */
begin
  value node P;
  value quadrant C;
  node Q;
  Q ← GTEQUAL_CORNER_NEIGHBOR(P, C);
  while GRAY(Q) do Q ← SON(Q, OPQUAD(C)); /* Descend to the desired
    corner */
  return (Q);
end;
```

It should be clear that procedures similar to CORNER_ADJ_NEIGHBOR and CORNER_CORNER_NEIGHBOR can be constructed that also return the level at which the desired neighboring node is found. This will not be done here.

The procedures outlined above always return NULL when a neighbor in a specified direction does not exist. This situation arises whenever the node whose neighbor is sought is adjacent to the border of the image along the specified direction. At times the NULL pointer is not convenient. Instead, we could assume that the image is surrounded by WHITE blocks as in Fig. 5a or by BLACK blocks as in Fig. 5b. The choice of WHITE or BLACK for the surrounding blocks depends on the particular application. For example, we use WHITE in the case of the quadtree to boundary code conversion algorithm [2] while BLACK is more useful in the case of the computation of distance [15] and the construction of a Quadtree Medial Axis Transform [16].

At times it is useful to determine if certain edges of the blocks corresponding to two neighboring nodes extend past each other or are aligned. For example, in Fig. 1, node 16 extends past node 10 with respect to their western boundaries, while the western boundaries of nodes 9 and 16 are aligned. We assume that the level of the tree at which each of the two nodes, say P and Q at levels LP and LQ, respectively, reside is known. It should be clear that at most | LP-LQ | nodes must be visited. This can be seen by observing that the smaller of the two nodes cannot extend farther

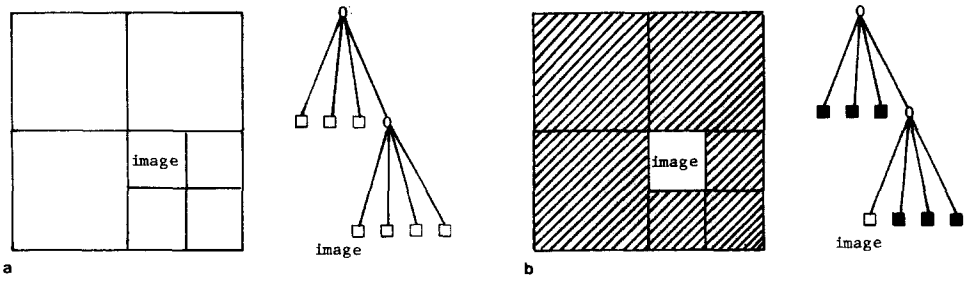

FIG. 5. Technique to avoid lacking a neighbor in a given direction. (a) Image surrounded by WHITE blocks. (b) Image surrounded by BLACK blocks.

than the other because this would imply that the two nodes 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 the designated direction of the nearest common ancestor of the two nodes. The algorithm for computing the aligned relationship is given below by procedure ALIGNED.

```
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;
```

The above techniques should be contrasted with other methods of locating neighbors [3–5, 8]. In [8], a method is described for moving between adjacent blocks of equal size that are brothers (i.e., have the same father node). This method does

not make use of the tree structure; instead, coordinate information and knowledge of the size of the image are used to locate a neighboring brother in a given horizontal or vertical direction. This is accomplished by a number of primitives termed MOVE UP, MOVE DOWN, MOVE RIGHT, and MOVE LEFT. Transitions to nonbrother neighboring blocks require the use of approximations through the use of primitives named MORE, LESS, and GAMMA. The disadvantages of these methods is that they require computation (rather than chasing links) and are clumsy when adjacent blocks are not brothers as well as when they are of different sizes than the block whose neighbor is sought.

In [3–5] a number of algorithms are described for operating on images using quadtrees. Transitions between neighboring blocks are made by use of explicit links from a node to its adjacent neighbors in the horizontal and vertical directions. This is achieved through the use of adjacency trees, "ropes," and "nets." An adjacency tree exists whenever a leaf node, say X, has a GRAY neighbor, say Y, of equal size. In such a case, the adjacency tree of X is a binary tree rooted at Y whose nodes consist of all sons of Y (BLACK, WHITE, and GRAY) that are adjacent to X. For example, for node 16 in Fig. 1, the western neighbor is GRAY node F with an adjacency tree as shown in Fig. 6. A rope is a link between adjacent nodes of equal size at least one of which is a leaf node. For example, in Fig. 1, there exists a rope between node 16 and nodes G, 17, H, and F. Similarly, there exists a rope between node 37 and nodes M and N; however, there does not exist a rope between node L and nodes M and N.

The algorithm for finding a neighbor using a roped quadtree is quite simple. We want a neighbor, say Y, on a given side, say D, of a block, say X. If there is a rope from X on side D, then it leads to the desired neighbor. If no such rope exists, then the desired neighbor must be larger. In such a case, we ascend the tree until encountering a node having a rope on side D, that leads to the desired neighbor. In effect, we have ascended the adjacency tree of Y. For example, to find the eastern neighbor of node 21 in Fig. 1, we ascend through node J to node F, which has a rope along its eastern side leading to node 16.

At times it is not convenient to ascend nodes searching for ropes. A data structure named a net is used in [3–5] to obviate this step by linking all leaf nodes to their neighbors regardless of their relative size. Thus in the previous example there would be a direct link between nodes 21 and 16 along the eastern side of node 21. The advantage of ropes and nets is that the number of links that must be traversed is reduced. However, the disadvantage is that the storage requirements are considerably increased since many additional links are necessary. In contrast, our methods are implemented by algorithms that make use of the existing structure of the tree—i.e., four links from a nonleaf node to its sons, and a link from a nonroot node to its father.

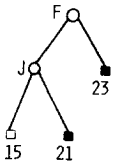

FIG. 6. Adjacency tree for the western neighbor of node 16 in Fig. 1.

4. ANALYSIS

The execution time of the neighbor finding algorithms presented in Section 3 depends on the relative positions of the nodes in question. Clearly, the execution time depends on the number of nodes that must be traversed in locating the desired neighbor. In the following we analyze the average execution time of EQUAL_ADJ_NEIGHBOR, GTEQUAL_ADJ_NEIGHBOR, CORNER_ADJ_NEIGHBOR, EQUAL_CORNER_NEIGHBOR, GTEQUAL_CORNER_NEIGHBOR, CORNER_CORNER_NEIGHBOR, and ALIGNED. GTEQUAL_ADJ_NEIGHBOR2 and GTEQUAL_CORNER_NEIGHBOR2 have the same execution time as GTEQUAL_ADJ_NEIGHBOR and GTEQUAL_CORNER_NEIGHBOR, respectively, since they visit the same number of nodes.

At this point it is appropriate to elaborate 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 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 of the number of nodes visited by* EQUAL_ADJ_NEIGHBOR *is bounded by* 4.

Proof. Given a node P at level i and a direction D, there are $2^{n-i} \cdot (2^{n-i} - 1)$ neighbor pairs of equal sized nodes. $2^{n-1} \cdot 2^0$ have their nearest common ancestor at level n, $2^{n-i} \cdot 2^1$ at level $n - 1, \ldots,$ and $2^{n-i} \cdot 2^{n-i-1}$ at level $i + 1$. For each node at level i having a nearest common ancestor at level j, the number of nodes that will be visited in the process of locating an equal sized neighbor at level i is $2 \cdot (j - i)$. 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 EQUAL_ADJ_NEIGHBOR is

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

The numerator of (1) can be simplified to yield

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

Making use of the identities (3)–(5) in (2) leads to (6)

$$\sum_{j=1}^{n} \frac{j}{2^j} = 2 - \frac{n+2}{2^n}, \tag{3}$$

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

$$\sum_{j=0}^{n} \frac{1}{2^{2j}} = \frac{4}{3} \cdot \left(1 - \frac{1}{2^{2n+2}}\right), \tag{5}$$

$$\frac{4}{3} \cdot 2^{2n+2} - 4 \cdot (n+1) \cdot 2^n - \frac{4}{3}. \tag{6}$$

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

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

Substituting (6) and (7) into (1) yields

$$4 - \frac{3 \cdot (n-1) \cdot 2^{n+2} + 12}{2^{2n+2} - 3 \cdot 2^{n+1} + 2}$$

$\simeq 4$ as n gets large
≤ 4 Q.E.D.

THEOREM 2. *The average number of nodes visited by* GTEQUAL_ADJ_NEIGHBOR *is bounded by* 5.

Proof. Given a node P at level i and a direction D, and recalling Theorem 1, there are $2^{n-i} \cdot \sum_{j=i+1}^{n} 2^{n-j} \cdot (j-i)$ pairs such that the neighboring node is of size greater than or equal to that of P. The index j in the summation corresponds to the level at which the nearest common ancestor is located. For a node at level i, a direction D, and a nearest common ancestor at level j, we have possible neighbor pairs having the initial node at level i and the neighboring nodes at levels $j-1$, $j-2, \ldots, i+1, i$—i.e., $j-i$ possible neighbor pairs. Thus for a node at level i, the number of nodes that will be visited in the process of locating a greater than or equal sized 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 GTEQUAL_ADJ_NEIGHBOR is

$$\frac{\displaystyle\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-i} \cdot 2^{n-j} \cdot \sum_{k=i}^{j-1} (j-i+j-k)}{\displaystyle\sum_{i=0}^{n-1} 2^{n-i} \cdot \sum_{j=1}^{n-i} j \cdot 2^{n-i-j}}. \tag{8}$$

The numerator of (8) can be simplified to yield

$$\sum_{i=1}^{n-1}\sum_{j=1}^{n-i} 2^{2n-2i-j-1}\cdot(3j^2+j). \tag{9}$$

Making use of identities (3)–(5) and (10) in (9) leads to (11)

$$\sum_{j=1}^{n}\frac{j^2}{2^j}=6-\frac{n^2+4n+6}{2^n}, \tag{10}$$

$$\frac{10}{3}\cdot 2^{2n+2}-(3n^2+7n+18)\cdot 2^n+2n+\frac{14}{3}. \tag{11}$$

The denominator of (8) can be simplified in a similar manner to yield

$$\tfrac{2}{3}\cdot 2^{2n+2}-(n+1)\cdot 2^{n+1}-\tfrac{2}{3}. \tag{12}$$

Substituting (11) and (12) into (8) yields

$$5-\frac{(9n^2-9n+24)\cdot 2^n-6n-24}{2^{2n+3}-3\cdot(n+1)\cdot 2^{n+1}-2}$$

$\simeq 5$ as n gets large

≤ 5. Q.E.D.

THEOREM 3. *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 towards corner C of P, using similar reasoning as in Theorem 1 and 2, there are $2^{n-i}\cdot\Sigma_{j=i+1}^{n}2^{n-j}\cdot j$ neighbor pairs where we make no restriction on the relative size of the neighboring node. j corresponds to the level at which the nearest common ancestor is located. For a node at level i, a direction D towards 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 nodes 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

$$\frac{\displaystyle\sum_{i=0}^{n-1}\sum_{j=i+1}^{n} 2^{n-i}\cdot 2^{n-j}\cdot\sum_{k=0}^{j-1}(j-i+j-k)}{\displaystyle\sum_{i=0}^{n-1}\sum_{j=i+1}^{n} 2^{n-i}\cdot 2^{n-j}\cdot j}. \tag{13}$$

The numerator of (13) can be simplified to yield

$$\sum_{i=0}^{n-1} \sum_{j=1}^{n-i} 2^{2n-2i-j-1} \cdot \left((3j^2 + j) + 4ji + i^2 + i\right) \tag{14}$$

Making use of the identities (3), (4), (10), (15), and (16) in (14) leads to (17)

$$\sum_{j=1}^{n} \frac{j}{2^{2j}} = \frac{1}{9}\left(4 - \frac{3n+4}{2^{2n}}\right), \tag{15}$$

$$\sum_{j=1}^{n} \frac{j^2}{2^{2j}} = \frac{1}{27}\left(20 - \frac{9n^2 + 24n + 20}{2^{2n}}\right), \tag{16}$$

$$\frac{98}{27} \cdot 2^{2n+2} - (3n^2 + 11n + 10) \cdot 2^n - \frac{1}{3}n^2 - \frac{11}{9}n - \frac{122}{27}. \tag{17}$$

The denominator of (13) can be simplified in a similar manner to yield

$$\frac{7}{9} \cdot 2^{2n+2} - (n+2) \cdot 2^{n+1} + \frac{2}{3}n + \frac{8}{9}. \tag{18}$$

Substituting (17) and (18) into (13) yields

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

$\approx 14/3$ as n gets large
$\leq 14/3$. Q.E.D.

THEOREM 4. *The average number of nodes visited by* EQUAL_CORNER_NEIGHBOR *is bounded by* 16/3.

Proof. Given a node P at level i and a direction towards quadrant C, there are $(2^{n-i} - 1)^2$ neighbor pairs of equal sized nodes. $4^0 \cdot (2 \cdot (2^{n-i} - 1) - 1)$ have their nearest common ancestor at level n, $4^1 \cdot (2 \cdot (2^{n-i-1} - 1) - 1)$ at level $n - 1, \ldots$ and $4^{n-i-1} \cdot (2 \cdot (2^{n-i-(n-i-1)} - 1) - 1)$ at level $i + 1$. In order to see this, consider Fig. 7, where a grid is shown for $n = 3$. If all BLACK and WHITE nodes are at level

	17		8		22		
14	15	16	9	19	20	21	
	18		10		23		
1	2	3	4	5	6	7	
	27		11		32		
24	25	26	12	29	30	31	
	28		13		33		

FIG. 7. Sample grid illustrating blocks whose nodes are at level 0 and whose nearest common ancestor is at level ≥ 2 when attempting to locate a NE neighbor.

0 (i.e., a complete quadtree), then for a neighbor pair in the NE direction we see that nodes along the fifth row and fourth column have their nearest common ancestor at level 3 (i.e., 13 nodes labeled 1–13). Continuing the process for the NW, NE, SW, and SE quadrants of Fig. 7 we find that all neighbor pairs contained exclusively within these quadrants have their nearest common ancestor at level ≤ 2. In particular, for the NW quadrant, nodes along the third row and second column have their nearest common ancestor at level 2 (i.e., 5 nodes labeled 14–18). The NE, SW, and SE quadrants are analyzed in a similar manner. This process is next applied to the four subquadrants of each quadrant to obtain the neighbor pairs whose nearest common ancestor is at level 1. Note that we had to consider every row in the image when analyzing neighbor pairs in the diagonal directions whereas we only needed to consider one row or one column when analyzing neighbor pairs in the N, E, S, and W directions. This is necessary because for neighbors in the diagonal directions, each row and column in the image has a different number of neighbor pairs with a common ancestor at a given level while this number is constant for each row or column when considering neighbor pairs in the horizontal and vertical directions.

For each node at level i having a nearest common ancestor at level j, the number of nodes that will be visited in the process of locating an equal sized neighbor at level i is $2 \cdot (j - i)$. 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 EQUAL_CORNER_NEIGHBOR is

$$\frac{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 4^{n-j} \cdot \left(2 \cdot (2^{n-i-(n-j)} - 1) - 1\right) \cdot 2 \cdot (j - i)}{\sum_{i=0}^{n-1} (2^{n-i} - 1)^2}. \tag{19}$$

The numerator of (19) can be simplified to yield

$$\sum_{i=0}^{n-1} \sum_{j=1}^{n-i} (2^{2n-2i-j+2} - 3 \cdot 2^{2n-2i-2j+1}) \cdot j. \tag{20}$$

Making use of identities (3)–(5) and (15) in (20) leads to (21)

$$\frac{16}{9} \cdot 2^{2n+2} - (n + 1) \cdot 2^{n+3} + n^2 + \frac{11}{3}n + \frac{8}{9}. \tag{21}$$

The denominator of (19) can be simplified in a similar manner to yield

$$\frac{1}{3} \cdot (2^{2n+2} - 3 \cdot 2^{n+2} + 3n + 8). \tag{22}$$

Substituting (21) and (22) into (19) yields

$$\frac{16}{3} - \frac{(6n - 10) \cdot 2^{n+2} - 3n^2 + 5n + 40}{2^{2n+2} - 3 \cdot 2^{n+2} + 3n + 8}$$

$\simeq 16/3$ as n gets large
$\leq 16/3$. Q.E.D.

THEOREM 5. *The average number of nodes visited by* (GTEQUAL_CORNER_NEIGHBOR *is bounded by* 6.

Proof. Given a node P at level i and a direction towards quadrant C, and recalling Theorem 4, there are $\sum_{j=i+1}^{n} 4^{n-j} \cdot (2 \cdot (2^{n-i-(n-j)} - 1) - 1) \cdot (j - i)$ neighbor pairs such that the neighboring node is of size greater than or equal to that of P. The index j in the summation corresponds to the level at which the nearest common ancestor is located. Recall that a neighbor in the direction of quadrant C will not always abut against corner C of the node whose neighbor is sought (e.g., node 16 is a nonabutting NE neighbor of node 23 in Fig. 1). For a node at level i, a direction towards quadrant C, and a nearest common ancestor at level j, we have possible neighbor pairs having the initial node at level i and the neighbor node at levels $j - 1, j - 2, \ldots, i + 1, i$—i.e., $(j - i)$ possible neighbor pairs. Thus for a node at level i, the number of nodes that will be visited in the process of locating a greater than or equal sized 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 GTEQUAL_CORNER_NEIGHBOR is

$$\frac{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 4^{n-j} \cdot \left(2 \cdot (2^{n-i-(n-j)} - 1) - 1\right) \cdot \sum_{k=i}^{j-1} (j - i + j - k)}{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 4^{n-j} \cdot \left(2 \cdot (2^{n-i-(n-j)} - 1) - 1\right) \cdot (j - i)}. \tag{23}$$

The numerator of (23) can be simplified to yield

$$\sum_{i=0}^{n-1} \sum_{j=1}^{n-i} (2^{2n-2i-j} - 3 \cdot 2^{2n-2i-2j-1}) \cdot (3j^2 + j). \tag{24}$$

Making use of the identities (3)–(5), (10), (15), and (16) in (24) leads to (25)

$$\frac{16}{3} \cdot 2^{2n+2} - (6n^2 + 14n + 32) \cdot 2^n + \frac{1}{2}n^3 + 3n^2 + \frac{13}{2}n + \frac{32}{3}. \tag{25}$$

The denominator of (23) is equal to $\frac{1}{2}$ of the numerator of (19)—i.e., (21):

$$\frac{8}{9} \cdot 2^{2n+2} - (n + 1) \cdot 2^{n+2} + \frac{n^2}{2} + \frac{11}{6}n + \frac{4}{9}. \tag{26}$$

Substituting (25) and (26) into (23) yields

$$6 - \frac{(27n^2 - 45n + 36) \cdot 2^{n+2} - 9n^3 + 81n - 144}{16 \cdot 2^{2n+2} - 18 \cdot (n + 1) \cdot 2^{n+2} + 9n^2 + 33n + 8}$$

≈ 6 as n gets large

≤ 6. Q.E.D.

THEOREM 6. *The average number of nodes visited by* CORNER_CORNER_NEIGHBOR *is bounded by* $6\frac{2}{27}$.

Proof. Given a node P at level i and a direction towards quadrant C, using similar reasoning as in Theorems 3 and 4, there are $\sum_{j=i+1}^{n} 4^{n-j} \cdot (2 \cdot (2^{n-i-(n-j)} - 1) - 1) \cdot j$ neighbor pairs where we make no restriction on the relative size of the abutting neighboring node. j corresponds to the level at which the nearest common ancestor is located. For a node at level i, a direction towards quadrant 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 nodes 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_CORNER_NEIGHBOR is

$$\frac{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 4^{n-j} \cdot \left(2 \cdot (2^{n-i-(n-j)} - 1) - 1\right) \cdot \sum_{k=0}^{j-1} (j-1+j-k)}{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 4^{n-j} \cdot \left(2 \cdot (2^{n-i-(n-j)} - 1) - 1\right) \cdot j}. \quad (27)$$

The numerator of (27) can be simplified to yield

$$= \sum_{i=0}^{n-1} \sum_{j=1}^{n-i} (2^{2n-2i-j} - 3 \cdot 2^{2n-2i-2j-1}) \cdot (3j^2 + 4ji + j + i^2 + i). \quad (28)$$

Making use of identities (3)–(5), (10), (15), and (16) in (28) leads to (29)

$$\frac{164}{27} \cdot 2^{2n+2} - (6n^2 + 22n + 32) \cdot 2^n + n^3 + \frac{17}{3}n^2 + \frac{94}{9}n + \frac{208}{27}. \quad (29)$$

The denominator of (27) can be simplified in a similar manner to yield

$$2^{2n+2} - (n+2) \cdot 2^{n+2} + n^2 + 4n + 4. \quad (30)$$

Substituting (29) and (30) into (27) yields

$$\frac{1}{27}\left(164 - \frac{(162n^2 - 62n - 448) \cdot 2^n - 27n^3 + 11n^2 + 374n + 448}{2^{2n+2} - (n+2) \cdot 2^{n+2} + n^2 + 4n + 4}\right)$$

$\approx 164/27$ as n gets large
$\leq 164/27 = 6\frac{2}{27}$. Q.E.D.

THEOREM 7. *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 Theorems 2 and 3, there are $2^{n-i} \cdot \sum_{j=i+1}^{n} 2^{n-j} \cdot j$ neighbor pairs such that there is no restriction on the size of the neighboring node. j corresponds to the level of the tree at which the nearest common ancestor 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

$$\frac{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-i} \cdot 2^{n-j} \cdot \sum_{k=0}^{j-1} |i-k|}{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-i} \cdot 2^{n-j} \cdot j}. \tag{31}$$

The numerator of (31) can be simplified to yield

$$\sum_{i=0}^{n-1} \sum_{j=1}^{n-i} 2^{2n-2i-j-1} \cdot (j^2 - j - i + i^2). \tag{32}$$

Making use of identities (3)–(5), (10), (15), and (16) in (32) leads to (33)

$$\frac{19}{27} \cdot 2^{2n+2} - (n^2 + n + 6) \cdot 2^n + \frac{1}{3}n^2 + \frac{11}{9}n + \frac{86}{27}. \tag{33}$$

The denominator of (31) is equal to (18) and substituting (33) and (18) in (31) yields

$$\frac{19}{27} - \frac{3}{7} \cdot \frac{(21n^2 - 17n + 50) \cdot 2^n - 7n^2 - 13n - 50}{7 \cdot 2^{2n+2} - 9 \cdot (n+2) \cdot 2^{n+1} + 6n + 8}$$

$\approx 19/21$ as n gets large

$\leq 19/21$. Q.E.D.

The analysis of the ALIGNED relationship performed in Theorem 7 can also be used to yield an estimate of the cost of finding neighbors when using the roping methods of Hunter [3–5]. Recall that roping implies that equal sized neighbors are linked directly regardless of whether or not they are brothers. In the case of a larger sized neighbor, the time required to access it in a roped quadtree is equal to the number of FATHER links that must be ascended to reach a GRAY ancestor node of size equal to that of the desired neighbor. Therefore, the following is the analog of Theorem 2 when roping is used.

THEOREM 8. *The average number of nodes visited when seeking larger sized neighbors in the horizontal and vertical directions in a roped quadtree is bounded by* 1.

Proof. Given a node P at level i and a direction D we have from Theorem 2 that there are $2^{n-i} \cdot \sum_{j=i+1}^{n} 2^{n-j} \cdot (j-i)$ pairs such that the neighboring node is of size greater than or equal to that of P. For a node at level i, the number of nodes that will be visited in locating a greater than or equal sized neighbor at level k ($k \geq i$) where the nearest common ancestor is at level j is $k-i$. The average is obtained as follows:

$$\frac{\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-i} \cdot 2^{n-j} \cdot \sum_{k=i}^{j-1} (k-i)}{\sum_{i=0}^{n-1} 2^{n-i} \cdot \sum_{j=1}^{n-i} j \cdot 2^{n-i-j}}. \tag{34}$$

The numerator of (34) can be simplified to yield

$$\sum_{i=0}^{n-1} \sum_{j=1}^{n-i} 2^{2n-2i-j-1} \cdot (j^2 - j). \tag{35}$$

Making use of identities (3)–(5) and (10) in (35) leads to

$$\frac{2}{3} \cdot 2^{2n+2} - (n^2 + n + 4) \cdot 2^n + \frac{4}{3}. \tag{36}$$

The denominator of (34) is equal to (12) and substituting (36) and (12) in (34) yields

$$1 - \frac{3 \cdot (n^2 - n + 2) \cdot 2^n - 6}{2^{2n+3} - 3 \cdot (n+1) \cdot 2^{n+1} - 2}$$

≈ 1 as n gets large

≤ 1. Q.E.D.

If one is interested in finding a smaller or larger neighbor in a roped quadtree, then we must add to the analysis of Theorem 8 a factor for finding a smaller sized neighbor. In the case of a roped quadtree we merely need to follow the rope and then descend to find the smaller neighbor. However, the cost of finding the smaller and larger sized neighbors is precisely the cost of the ALIGNED procedure. Thus for a roped quadtree, the analog of Theorem 3 is given below.

THEOREM 9. *The average number of nodes visited when seeking neighbors of arbitrary size in the horizontal and vertical directions in a roped quadtree is bounded by* 19/21.

Proof. Given a node P at level i and a direction D we have from Theorem 3 that there are $2^{n-i} \cdot \sum_{j=i+1}^{n} 2^{n-j} \cdot j$ neighbor pairs such that there is no restriction on the size of the neighboring node. For a node at level i, the number of nodes that will be visited in locating a greater than or equal sized neighbor at level k ($k \geq i$) where the nearest common ancestor is at level j is $k - i$. Similarly, in locating a smaller sized neighbor at level k ($k < i$) $i - k$ nodes will be visited. The average is obtained as follows:

$$\frac{\displaystyle\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-i} \cdot 2^{n-j} \cdot \left(\sum_{k=0}^{i-1} (i-k) + \sum_{k=i}^{j-1} (k-i) \right)}{\displaystyle\sum_{i=0}^{n-1} \sum_{j=i+1}^{n} 2^{n-i} \cdot 2^{n-j} \cdot j}. \tag{37}$$

However, (37) is identical to (31) and our result follows. Q.E.D.

Note that the bounds of Theorems 8 and 9 do not reflect the fact that one must also visit one additional link due to the presence of the rope (i.e., the link which the rope represents). It should also be clear that if the quadtree is netted, then no links need ever be traversed except for the link which the net represents.

5. CONCLUDING REMARKS

We have described the neighbor finding techniques for quadtrees in detail. The analyses of the various algorithms demonstrate that the operation is quite efficient. We have used an unusual model of a random image in our analysis. However, as stated in Section 4, had we used a model which attributes a given probability (e.g., 1/2) for a pixel being BLACK or WHITE, the quadtree for a $2^n \times 2^n$ image would most likely have n levels and have neighboring nodes of equal size. In such a case, Theorems 1 and 4 are applicable and show a lower bound on the execution time of adjacent and corner neighbor locating algorithms. Thus our model attempts to present a more realistic view of the time complexity of these algorithms.

We have also analyzed an alternative neighbor finding technique which makes use of a construct termed a rope. In such a case, we saw that neighbors can be located more quickly (at about $\frac{1}{2}$ to $\frac{1}{3}$ of the cost when our technique is used). Nevertheless, if space is at a premium, roping should probably not be used without some careful thought. An upper bound on the number of links necessary to achieve roping is four times the number of leaf nodes since each leaf node may participate in a maximum of four ropes (one for each side).

REFERENCES

1. C. R. Dyer, Computing the Euler number of an image from its quadtree, *Computer Graphics and Image Processing* **13**, 1980, 270–276.
2. C. R. Dyer, A. Rosenfeld, and H. Samet, Region representation: Boundary codes from quadtrees, *Comm. ACM* **23**, 1980, 171–179.
3. G. M. Hunter, *Efficient Computation and Data Structures for Graphics*, Ph.D. dissertation, Department of Electrical Engineering and Computer Science, Princeton University, Princeton, N.J., 1978.
4. G. M. Hunter and K. Steiglitz, Operations on images using quadtrees, *IEEE Trans. Pattern Anal. Mach. Intell.* **1**, 1979, 145–153.
5. G. M. Hunter and K. Steiglitz, Linear transformation of pictures represented by quadtrees, *Computer Graphics and Image Processing* **10**, 1979, 289–296.
6. A. Klinger, Patterns and search statistics, in *Optimizing Methods in Statistics* (J. S. Rustagi, Ed.), Academic Press, New York, 1971.
7. A. Klinger and C. R. Dyer, Experiments in picture representation using regular decomposition, *Computer Graphics and Image Processing* **5**, 1976, 68–105.
8. A. Klinger and M. L. Rhodes, Organization and access of image data by areas, *IEEE Trans. Pattern Anal. Mach. Intell.* **1**, 1979, 50–60.
9. P. Naur (Ed.), Revised report on the algorithmic language ALGOL 60, *Comm. ACM* **3**, 1960, 299–314.
10. H. Samet, Region representation: quadtrees from boundary codes, *Comm. ACM* **23**, 1980, 163–170.
11. H. Samet, Region representation: Quadtrees from binary arrays, *Computer Graphics and Image Processing* **13**, 1980, 88–93.
12. H. Samet, An algorithm for converting from rasters to quadtrees, *IEEE Trans. Pattern Anal. Mach. Intell.* **3**, 1981, 93–95.
13. H. Samet, Connected component labeling using quadtrees, *J. ACM* **28**, July 1981, 487–501.
14. H. Samet, Computing perimeters of images represented by quadtrees, *IEEE Trans. Pattern Anal. Mach. Intell.*, in press.
15. H. Samet, A distance transform for images represented by quadtrees, *IEEE Trans. Pattern Anal. Mach. Intell.*, in press.
16. H. Samet, *A Quadtree Medial Axis Transform*, Computer Science TR-803, University of Maryland, College Park, Md, August 1979.
17. H. Samet, *Algorithms for the Conversion of Quadtrees to Rasters*, Computer Science TR-979, University of Maryland, College Park, Maryland, November 1980.
18. M. Shneier, Linear-time calculations of geometric properties using quadtrees, *Computer Graphics and Image Processing* **16**, 1981, 296–302.
19. M. Shneier, Path-length distances for quadtrees, *Inform. Sci.* **23**, 1981, 45–67.