Comp. & Maths. with Appls. Vol. 11, No. 11, pp. 1089–1097, 1985
Printed in Great Britain.

0097-4943/85 $3.00 + .00

ON THE SIZE OF QUADTREES GENERALIZED TO d-DIMENSIONAL BINARY PICTURES

T. R. WALSH
Department of Computer Science, Univ. of Western Ontario

(*Received October* 1984; *in revised form March* 1985)

Communicated by Ervin Y. Rodin

Abstract—Some results about the size of quadtrees and linear quadtrees, used to represent binary $2^n \times 2^n$ digital pictures, are generalized to d-dimensional $2^n \times \ldots \times 2^n$ pictures. Among these results are a comparison of the space-efficiency of linear vs regular trees, in terms of both the number of nodes of the tree and the number of bits needed to store each node, and an upper bound on the number of nodes as a function of n and the perimeter of the picture.

INTRODUCTION

Quadtrees[1, 2] have recently emerged as a convenient means of encoding 2-dimensional black-and-white pictures because they enable large picture fragments of the same color to be represented by a single node while retaining spatial information in a form suitable for processing. Raster-algorithms using quadtrees have been formulated for filling, chain-code determination, labelling of maximally-connected subsets, medial-axis transform, conversion from and to array representation[3] and others[4]. Most recently hardware implementations have been announced for processing[5] and display[6].

Octtrees, similar to quadtrees, were introduced in [7] for representing 3-dimensional pictures, and have been used for a number of raster-algorithms including translation and rotation[7] and serial section image analysis[4]. The generalization of quadtrees and octtrees to d-dimensional pictures was called hyperocttrees or 2^d-trees in [8]; the space-complexity of this data-structure will be analysed in this article.

When a quadtree is stored in a computer, each node must have five pointers, one to its father and one to each of its four sons. A variety of quadtree linearizations[9–15], mainly aimed at eliminating or reducing the number of pointers, has recently appeared: these linearizations represent, totally or partially, compression techniques and are, therefore, quite useful in archiving very large amounts of pictorial data; a few of these[9–11] preserve some of the dynamic aspects of regular quadtrees such as quick insertion and deletion capabilities. In particular, linear quadtrees[9] have proved quite useful in designing several efficient raster algorithms[16–18]. Linear octtrees were introduced in [19] and used for 3-D border-determination in [16] and for 3-D filling in [20]. Here we introduce linear 2^d-trees and compare their space-complexity with that of regular 2^d-trees.

In Sec. 1 we review the definitions of regular and linear quadtrees and regular 2^d-trees, introduce linear 2^d-trees, and compare linear quadtrees with some other picture representations with regard to space complexity. The number of bits per node of a regular quadtree[21] and a linear quadtree[18] are compared, and these results are generalized to 2^d-trees. In particular, it is shown that a regular quadtree takes about 2.5 times as many bits per node as a linear one, and that this ratio increases exponentially with the dimension.

The space saved by using linear instead of regular trees depends not only on the number of bits per node but also on the number of nodes in the respective trees. The ratio of the number of nodes in a regular tree to the number of nodes in the corresponding linear tree is never less than 1: in Sec. 2 we find the maximum and minimum of this ratio. Specializing these results to quadtrees and comparing with the results of Sec. 1 we find that it takes between 3.3 and 26.5 times as many bits to store a regular quadtree as a linear quadtree. At this point the reader may think that the "better" space utilization has been gained at the expense of time efficiency for the frequently-used algorithms, such as filling, border-determination, search and set operations: this is indeed not the case, since the most recently-introduced algorithms[16–18, 20,

22] are simpler, have better worst-case time complexity than those using regular quadtrees, and can be extended to higher dimensions with very minor modifications.

In analysing the complexity of a raster-algorithm one must take into account the complexity of the picture and not just the size of the raster; a one-black-pixel picture on a large raster can be expected to be processed quickly. A reasonable measure of this complexity is its perimeter, which is roughly the size of its chain code. In [1] it was shown that the number of nodes in the quadtree-representation of a 2-dimensional picture with polygonal boundary is $O(n + L)$, where n is the logarithm of the diameter of the raster and L is the perimeter of the picture. For general 2-dimensional pictures this asymptotic bound does not hold; it was shown in [17] that the number of nodes of a linear quadtree can get arbitrarily close to $0.75nL$ but cannot exceed $1.5nL$. In section 3 we generalize the upper bound to d dimensions, showing in particular that a linear 2^d-tree cannot have more than $((2^d - 1)/d)nL$ nodes. The number of nodes of a regular 2^d-tree is also shown to be $O(nL)$, so that the time- and space-complexity of algorithms using either regular or linear trees can now be bounded in terms of quantities which are independent of the data structure.

1. BITS PER NODE: REGULAR VS LINEAR TREES IN 2 AND d DIMENSIONS

The *quadtree* of a $2^n \times 2^n$ binary picture is a rooted directed tree generated by the recursive subdivision of the picture into quadrants. Each node of the quadtree represents a square sub-picture of the picture, with the root representing the entire picture. If the sub-picture represented by a given node is all of one color then the node is a leaf of the same color; otherwise the sub-picture is divided into 4 equal quadrants and the node is colored gray and given 4 sons, one for each quadrant. An example of a black-and-white picture with $n = 3$ is given in Fig. 1a and its corresponding quadtree is shown in Fig. 1b (these figures are taken from [9]).

We denote by N, G and W the number of black, gray and white nodes, respectively.

In a *linear quadtree*[9] only the black nodes are stored; each one contains a quaternary code (defined below) and a pointer to the next node (the nodes are stored as a linked list to enable quick insertion and deletion, and are also sorted according to their codes for conversion to a sorted array should searches become necessary). To construct the code for a node, one follows the path from the root to the node, writing 0, 1, 2 or 3 each time one goes from a node to its NW, NE, SW or SE son, respectively, and then pads the code with 0, if necessary, until it has n digits; a second integer in the code, called the *level* or grouping factor, counts the padding zeros. The code for each black pixel is given in Fig. 1(a), and the linear quadtree for this picture is shown in Fig. 1(c).

The quaternary code for a (black) node of a linear quadtree requires $2n$ bits and the level requires $\log_2 n$ bits. Since N, the number of black nodes, could be as large as $3 \times 4^{n-1}$ (for a picture with 3 black and 1 white pixel in each 2×2 square), the pointer requires $2n$ bits. For border-determination and filling an additional 4 bits are needed, one for each of the four principal directions (E, S, W, N). Details of how these extra bits are used can be found in [16, 18, 20]; suffice it to say here that the eastern bit of a given node is 0 if all the easternmost pixels in the corresponding square sub-picture share their eastern sides with a black pixel. Altogether, then, one node of a linear quadtree requires $4n + \log_2 n + 4$ bits, and the whole quadtree $(4n + \log_2 n + 4)N$ bits.

For a fair comparison between linear and regular quadtrees one must note that for most algorithms on regular quadtrees white nodes can be replaced by nil pointers on their fathers, so that we need to analyse space requirements only in terms of number of gray nodes (G) and black nodes (N). Adapting the analysis recently done in [21], $2n + 1$ bits are required for each of the 5 pointers (father and 4 sons) and 1 bit for the color (gray or black) so that a node requires $10n + 6$ bits and the whole quadtree $(10n + 6)(N + G)$ bits. Thus, for large n one node of a regular quadtree requires about 2.5 times the space of one nodes of a linear quadtree; the ratio $(N + G)/N$ will be examined in Sec. 2.

We note here that there do exist more compact representations for storing pictures. The Srihari representation[4] requires $(N + W + 2G)$ bits and can be further compressed by using the Kawasuchi–Endo technique[23]. And the chain code takes $O(n + L)$ bits to represent an L-pixel border, as opposed to $O(nL)$ for a quadtree. But these data structures are often converted

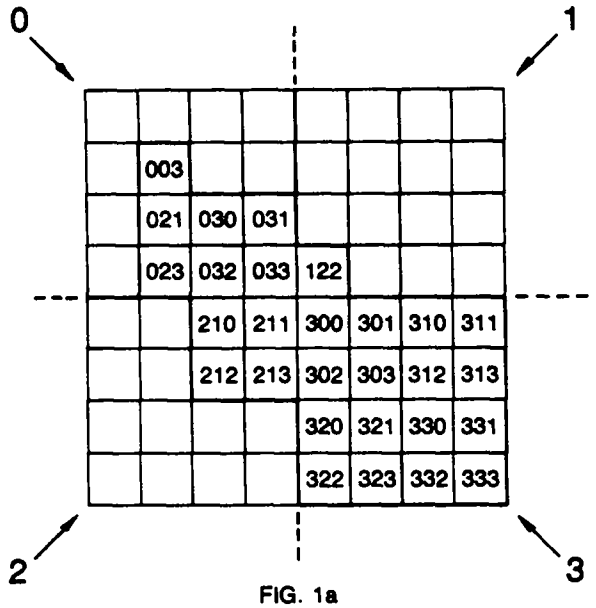

FIG. 1a

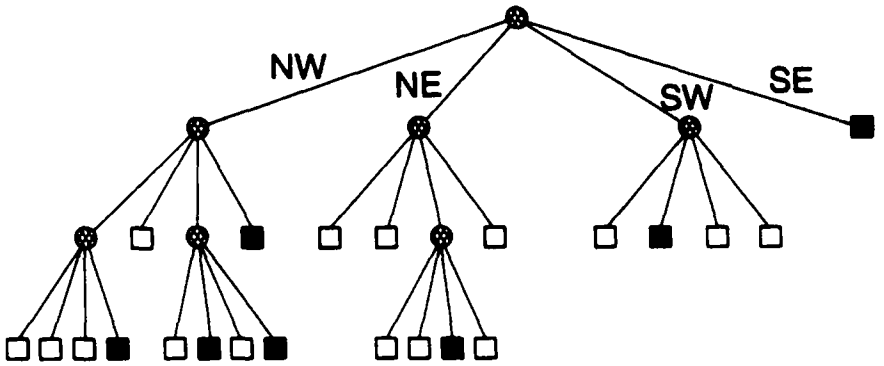

FIG. 1b

(003,0), (021,0), (023,0), (030,1), (122,0), (210,1), (300,2)

FIG. 1c

Fig. 1. A $2^1 \times 2^1$ picture with its black pixels coded [Fig. 1(a)], its regular quadtree [Fig. 1(b)] and its linear quadtree [Fig. 1(c)].

to quadtrees since quadtrees are more convenient for processing pictures–for example, simple operations such as union, intersection, difference and search can be performed directly on a quadtree but not on a chain code, and besides, chain codes cannot be extended to higher dimensions—whence the relevance of a comparison between regular and linear quadtrees.

Quadtrees and octtrees in both complete and condensed form were generalized to arbitrary dimensions in [8]; we summarize the condensed form, introduce a linear version and find the number of bits required by a node of each.

Given a binary d-dimensional $2^n \times \ldots \times 2^n$ picture, the generalization of a quadrant (for $d = 2$) and an octant (for $d = 3$) will be called an *ant* (instead of hyperoctant as in [8]): one of 2^d sub-pictures into which d bisecting hyperplanes divide the picture. The generalization of a quadtree ($d = 2$) and an octree ($d = 3$) will be called a *2^d-tree* (instead of hyperoctree): the root represents the whole picture, and for each node, if the sub-picture it represents is either all black or all white then the node is a leaf of the same color, otherwise the node is colored gray and has 2^d sons, one for each ant of that sub-picture. As with 2-dimensional pictures, we

call n the *resolution*, and we still use the word "pixel" instead of "hypervoxel" for a sub-picture of unitary dimensions.

For a (regular) 2^d-tree we need $nd + 1$ bits to store an address (since there could be as many as $1 + 2^d + 2^{2d} + \ldots + 2^{nd}$ nodes) and each node has $2^d + 1$ pointers: adding 1 bit for the color (black or gray) we now need $(2^d + 1)(nd + 1) + 1$ bits per node.

The generalization of the linearization of a quadtree to d dimensions is quite straightforward. Only the black nodes are stored, so that we do not need a bit for the color. The code for a node consists of n digits, each with d bits instead of 2, and a $\log_2 n$-bit integer for the level of the node in the tree. The pointer to the next node requires nd bits (there can be no more than 2^{nd} black nodes). Since there are now $2d$ principal directions we would need $2d$ bits for border-determination and filling (see [16] and [20] for the case $d = 3$). So all together a node of a linear 2^d-tree requires $2dn + 2d + \log_2 n$ bits. Comparing this number with the $(2^d + 1)(nd + 1) + 1$ bits needed to store a node of a regular 2^d-tree, one can see how rapidly the gain in space-efficiency afforded by using linear instead of regular 2^d-trees increases with d.

2. THE RATIO OF GRAY NODES TO BLACK NODES

We now consider the problem of finding the maximum and minimum values of the ratio G/N over all 2^d-trees representing binary $2^n \times \ldots \times 2^n$ pictures. The importance of this ratio is that if white nodes are replaced by nil pointer in a regular tree then the ratio of the number of nodes in a regular tree to the number of nodes in the corresponding linear tree is $(G + N)/N = (G/N) + 1$. We show, in particular, that this ratio can be as great as $n + 1$. We also find the maximum and minimum values of G/N over all trees whose leaves are all at the same level; such trees are often encountered in the case of border determination and reconstruction.

THEOREM 1

Excluding the trivial cases of all-white and all-black pictures, we have the following bounds on the ratio G/N:

(1) The minimum is $1/(2^d - 1)$, for a picture which is all black except for one white pixel (or, more generally, one white ant at any level).

(2) The maximum is n, attained only for a picture which is all white except for one black pixel.

(3) Among the 2^d-trees all of whose leaves are at the same level, the maximum is $(2^d/(2^d - 1))(1 - 2^{-dn})$, for a picture with one black pixel in each $2 \times 2 \times \ldots \times 2$ hypercube (all leaves are at level 0), and the minimum is $1/(2^d - 1)$, for a picture which is all black except for one white ant at level $n - 1$ (all leaves are at level $n - 1$). The minimum over all 2^d-trees all of whose leaves are at level 0 is $(2^d/(2^d - 1)^2)(1 - 2^{-dn})$, for a picture with one white pixel in each $2 \times \ldots \times 2$ hypercube.

Proof. We first note

Property a. If the 2^d sons of some non-leaf node of a 2^d-tree are all leaves then at least one of them is black and at least one of them is white.

We now seek the minimum value of G/N over all 2^d-trees with resolution n. Since (by Property a) every non-leaf node can have at most $2^d - 1$ black leaves for sons, $G/N \geq 1/(2^d - 1)$. This minimum value is attained for the 2^d-tree of Fig. 2(a) (representing a picture which is all black except for one white pixel), establishing point (1) of the theorem.

We now seek the maximum value of G/N over all 2^d-trees with resolution n. Consider the tree of Fig. 2(b), representing a picture which is all white except for one black pixel. For this tree, $N = 1$ and $G = n$, so that $G/N = n$, and we claim that this is the unique 2^d-tree for which $G/N \geq n$. To this end we recall that the depth of a node in a tree is the difference between the level of the root (n) and the level of the node, and the height of a tree is the greatest depth of any node.

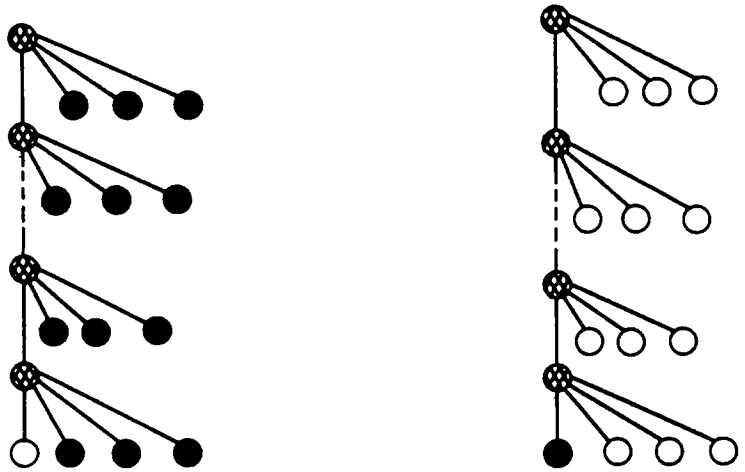

Fig. 2. The quadtrees which minimize [Fig. 2(a)] and maximize [Fig. 2(b)] the ratio G/N.

LEMMA 2

For any 2^d-tree of height $h \leq n$ we have $G/N \leq n$, with equality only in the case of the tree in Fig. 2(b).

Proof. For any 2^d-tree, let S be the set of non-leaf nodes all of whose sons are leaves. By Property a, $N \geq |S|$. For each node x in S, the number of nodes in the path from the root to x is equal to the depth of the sons of x. Hence the sum of the cardinalities of these paths over all the nodes in S cannot exceed $h|S|$, with equality if and only if every node x in S is of depth $h - 1$.

But every non-leaf node v in the tree belongs to at least one path from the root to a node in S, since the subtree at v is of finite length, and its deepest non-leaf node is in S. Thus the union of these paths is just the set of non-leaf nodes, with cardinality G; so the sum of the cardinalities of the paths cannot be less than G, with equality if and only if $|S| = 1$ since two of these paths cannot be disjoint as they have at least the root in common.

Thus $G \leq h|S| \leq hN \leq nN$ (since $h \leq n$), with equality if and only if all of the following hold: $h = n$, only one node v has 2^d leaves for sons, and every leaf is white except for one of these 2^d sons of v. Only the tree of Fig. 2(b) meets all of these conditions; in every other case $G < nN$, so that $G/N < n$.□

This establishes point (2) of the theorem.

To prove point (3) we need another lemma, which extends to higher dimensions a result quoted in the second page of [21].

LEMMA 3

The total number F of leaves is $(2^d - 1)G + 1$.

Proof (by induction on G). If $G = 0$ we have a single leaf, and $1 = (2^d - 1)0 + 1$. Suppose that $G > 0$ and that the lemma is true for any 2^d-tree with $G - 1$ non-leaf nodes. Take any 2^d-tree with G non-leaf nodes. Its height $h > 0$. Any non-leaf node at depth $h - 1$ has 2^d leaves for sons. Choose one such node v and remove all its 2^d sons, changing v into a leaf. This reduces G by 1 (to $G - 1$) and F by $2^d - 1$ (to $(2^d - 1)(G - 1) + 1$ by the induction hypothesis). It follows that

$$F = (2^d - 1)(G - 1) + 1 + (2^d - 1) = (2^d - 1)G + 1,$$

which completes the induction.□

We can now examine the case when all the leaves are at the same depth $h \geq 1$ (we are excluding the tree of height 0 consisting of only the root, since the picture would then have to be all black or all white). In this case the number F of leaves is 2^{dh}. Also, these leaves can be grouped into sets of 2^d, the 2^d sons of the same node, so that in each set the number of black

leaves is in the closed interval $[1, 2^d - 1]$ (by Property a). Thus the minimum number of black leaves is $2^{d(h-1)}$ and the maximum number of black leaves is $2^{d(h-1)}(2^d - 1)$. By Lemma 3 we have $G = (2^{dh} - 1)/(2^d - 1)$. Substituting into G/N, taking the extreme values of h (1 and n) and simplifying we can establish point 3 of the theorem, thus completing the proof.□

Specializing points (1) and (2) to $d = 2$ we find that for practical values of n ($n \leq 10$) a regular quadtree can have between 4/3 and 11 times as many nodes as the corresponding linear quadtree. Multiplying these ratios by the bits-per-node ratio of 2.5 we find that a regular quadtree requires from 3.3 to 26.5 times as many bits as the corresponding linear one. For quadtrees all of whose leaves are at level 0, a regular quadtree can have between $(4/9)(1 - 4^{-n}) + 1$ and $(4/3)(1 - 4^{-n}) + 1$ times as many nodes as the corresponding linear one; taking the extreme values of n (1 and 10) and multiplying by the bits-per-node ratio of 2.5 we find that a regular quadtree of this type requires from 3.3 to 5.8 times as many bits as the corresponding linear one. We note that these gains are under-estimations in the case of region reconstruction, since in that particular algorithm white nodes are indeed necessary.

3. AN UPPER BOUND ON THE NUMBER OF NODES GIVEN THE PERIMETER

The absolute upper bound on the size of a tree representing a binary picture is the size of the raster (divided by $1 - 2^{-d}$ to account for the gray nodes), but for many pictures this bound is unduly pessimistic. A reasonable measure of the complexity of a picture is its perimeter, and we bound N and G as functions of n and the perimeter.

Two pixels are called *neighbors* if they differ in exactly one co-ordinate and by one unit. A black pixel is called *exposed* if it has white neighbor, and a *border-pixel* if it is either exposed or on the *raster-border* (at least one of its co-ordinates is either 0 or $2^n - 1$). The *perimeter* L of a picture is the number of its border-pixels.

In [17] we showed that the number of black nodes of a quadtree representing a $2^n \times 2^n$ picture of perimeter L cannot exceed $1.5nL$ (but may set close to $0.75nL$ for large n). Here we show that the number N of black nodes in a 2^d-tree representing a $2^n \times \ldots \times 2^n$ binary picture of perimeter L cannot exceed $((2^d - 1)/d)nL$, and that the number G of gray nodes cannot exceed nL (independently of d).

LEMMA 4 (Lemma 6 of [17])

If W and B are white and black pixels, respectively, in a binary picture of any dimension and resolution, then there is an exposed pixel.

Proof. Let $B = Q_0, Q_1, \ldots, Q_k = W$ be any path (sequence of pixels such that Q_j and Q_{j-1} are neighbors for all j) and let i be the first index such that Q_i is white. Then $i \geq 1$ and Q_{i-1} is the desired exposed pixel.□

THEOREM 5.

The number G of gray nodes in a 2^d-tree representing a binary $2^n \times \ldots \times 2^n$ picture of perimeter L cannot exceed nL (independently of d).

Proof. If the picture is all white or all black then $G = 0$ and $nL \geq 0$; so we assume that there is at least one white pixel and at least one black pixel.

Let A be the number of exposed pixels. We show by induction on n that $G \leq nA$, which implies that $G \leq nL$ since $A \leq L$ (every exposed pixel is a border-pixel).

For $n = 1$ the inequality holds since $G = 1$ and $nA \geq 1$ by Lemma 4.

Suppose that $G \leq nA$ for a given n, and consider any picture with resolution $n + 1$ and at least one white pixel and at least one black pixel. Define G_i and A_i, $i = 1, 2, \ldots, 2_d$, to be the number of gray nodes and the number of exposed pixels, respectively, in ant $\#i$ of the picture.

By the induction hypothesis we have

$$G_i \leq nA_i \tag{1}$$

if ant $\#i$ is neither all white nor all black, and otherwise (1) holds anyway since in this case $G_i = nA_i = 0$.

Since the picture is neither all white nor all black, the root of its 2^d-tree is a gray node with 2^d sons, and we have

$$G = G_1 + \ldots + G_{2^d} + 1. \tag{2}$$

Also, we have

$$A \geq A_1 + A_2 + \ldots + A_{2^d}, \tag{3}$$

with strict inequality arising if a black pixel on the raster-border of some ant does not have a white neighbor in that ant but does have a white neighbor on the raster-border of a neighboring ant (see Fig. 3, which was taken from [17]).

From these three inequalities we derive

$$G \leq nA + 1. \tag{4}$$

But by Lemma 4 we have $A \geq 1$, whence $G \leq (n + 1)A$, which completes the induction. □

We note that the inequality $G \leq nL$ is sharp: the upper bound is attained by the tree of Fig. 2(b).

To prove a similar result for black nodes we need the following lemma which generalizes to higher dimensions Lemma 7 of [17]:

LEMMA 6

In a d-dimensional $2^n \times \ldots \times 2^n$ binary picture, suppose that there is at least 1 white pixel P and exactly b black ones. Then

(1) if $b \leq d$ then every black pixel is exposed, and

(2) if $b \geq d$ then at least d black pixels are exposed.

Proof. To prove (1), we assume that there exists a black pixel Q with no white neighbor and show that $b \geq d + 1$. In d dimensions, Q must have at least d neighbors, all of which must be black. Together with Q, these neighbors constitute $d + 1$ black pixels, so that $b \geq d + 1$.

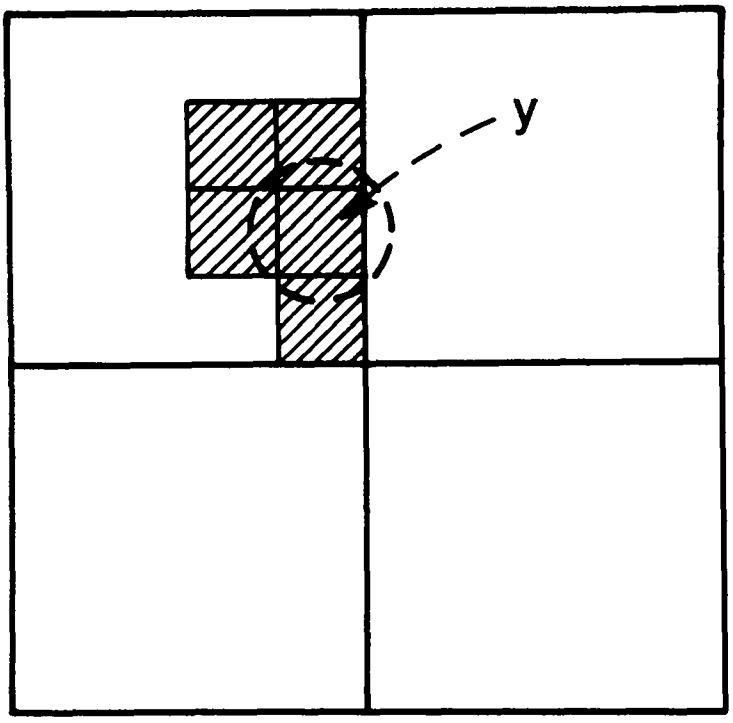

Fig. 3. The pixel y has a white neighbor in the neighboring $2^2 \times 2^2$ quadrant.

To prove (2), we assume that fewer than d black pixels are exposed, so that since there are at least d black pixels there must be some unexposed black pixel Q. Since any path from the white pixel P to the black pixel Q must pass through some exposed pixel (the first black pixel met on the path), it follows that there is a set S of fewer than d pixels which separates P from Q. We show this to be impossible (by proving that the graph whose vertices are the pixels with edges joining neighboring pixels is d-connected in the graph-theoretical sense).

Suppose that P and Q are in the same $(d-1)$-dimensional hyperplane perpendicular to some co-ordinate axis. If not all the pixels in S are in H, then H contains fewer than $d-1$ pixels in S, so that, by the induction hypothesis, there is a path from P to Q in H which avoids S. If all the pixels in S are in H, then we can avoid S via a neighboring hyperplane, since even if H is part of the raster-border it must have one neighboring hyperplane.

Now suppose that P and Q are not in the same $(d-1)$-dimensional hyperplane; we can find a path from P to Q which avoids S in 2 steps, using an intermediate pixel R. Drop perpendiculars from P to each of the d $(d-1)$-dimensional hyperplanes containing Q. The feet of these perpendiculars are d distinct pixels; so at least one of them is not in S, and we call one of these pixels R. Since R and Q are in the same hyperplane, by what has already been shown there is a path from R to Q which avoids S. Since P and R are in the same line there are $d-1 \geq 1$ hyperplanes containing P and R, so that there is also a path from P to R which avoids S. The union of these two paths may not be a path, but it contains a path from P to Q which avoids S. This completes the proof. □

THEOREM 7

The number N of black nodes in a 2^d-tree representing a $2^n \times \ldots \times 2^n$ picture (with $n \geq 1$ and $d \geq 1$) with L border-pixels cannot exceed $((2^d-1)/d)nL$.

Proof. If the picture is all black then $N = 1$ and $((2^d-1)/d)nL \geq 1$. So we assume that there is at least one white pixel. As in Theorem 5 we let A be the number of exposed pixels and we show by induction on n that

$$N \leq ((2^d-1)/d)nA. \tag{5}$$

Suppose $n = 1$. In this case N is the number of black pixels (since a black node at a level >0 would imply an all-black picture). By Lemma 6 we have either $N \leq d$ and $A = N$ (which satisfies (5)) or $N > d$ and $A \geq d$. Assume the latter. The maximum value for N is $2^d - 1$ (one white pixel), but since $A \geq d$, (5) holds in this case too (actually $A = d$ since the one white pixel has d neighbors).

Now suppose that the theorem holds for a given n and consider any picture with resolution $n+1$ and at least one white pixel. We will show that

$$N \leq ((2^d-1)/d)(n+1)A. \tag{6}$$

Define N_i and A_i, $i = 1, 2, \ldots, 2^d$, to be the number of black nodes and the number of exposed pixels, respectively, in ant $\#i$. By the induction hypothesis we have

$$N_i \leq ((2^d-1)/d)nA_i \tag{7}$$

if ant $\#i$ is not all black, while

$$N_i = 1 \text{ and } A_i = 0 \tag{8}$$

if ant $\#i$ is all black.

Since not all the ants are all black we have

$$N = N_1 + \ldots + N_{2^d}. \tag{9}$$

Also, as in the proof of Theorem 5 we have

$$A \geq A_1 + A_2 + \ldots + A_{2^d}. \tag{10}$$

Substituting (7) and (8) into (9) we have

$$N \leq ((2^d - 1)/d)n(A_1 + \ldots + A_{2^d}) + k, \tag{11}$$

where k is the number of all-black ants. Substituting (10) into (11) we have

$$N \leq ((2^d - 1)/d)nA + k. \tag{12}$$

But since there is at least one white pixel, $k \leq 2^d - 1$; so from (12) we have

$$N \leq ((2^d - 1)/d)(nA + d). \tag{13}$$

If at least d pixels are black, then by Lemma 6 we have $A \geq d$ so that, from (13), we have

$$N \leq ((2^d - 1)/d)(nA + A) = ((2^d - 1)/d)(n + 1)A,$$

so that (6) holds.

If fewer than d pixels are black then $N \leq$ number of black pixels $= A$ (by Lemma 6) $\leq ((2^d - 1)/d)(n + 1)A$, so that (6) holds in this case as well.

This completes the induction.□

Thus for any dimension d the number of nodes in both a linear and a regular 2^d-tree is $O(nL)$, with a multiplicative constant depending on d. In [17] it was shown that the constant for linear quadtrees (1.5) could not be improved by more than a factor of 2; we leave the generalization of this result to higher dimensions as an open problem.

Acknowledgement—The author wishes to thank Prof. I. Gargantini for posing these problems and giving valuable assistance in the preparation of the Introduction and Section 1 of this article.

REFERENCES

1. G. M. Hunter and K. Steiglitz, Operations on images using quadtrees. *IEEE PAMI* **1**, 145–153 (1979).
2. A. Klinger and M. L. Rhodes, Organization and access of image data by areas. *IEEE PAMI* **1**, 50–60 (1979).
3. H. Samet, The quadtree and related hierarchical data structures. *ACM Comp. Surveys* **16**, 187–260 (1984).
4. S. N. Srihari, Representations of three-dimensional images. *Comp. Surveys* **13**, 400–424 (1981).
5. C. R. Dyer, *The pyramid machine*, oral communication.
6. D. J. Milford, P. J. Willis and J. R. Woodwark, Quad encoded display. *IEE Proc.* **131**, 70–75 (1984).
7. C. L. Jackins and S. L. Tanimoto, Oct-trees and their use in representing three-dimensional objects. *Comp. Graphics Image Proc.* **14**, 249–270 (1980).
8. M. Yau and S. N. Srihari, A hierarchical data structure for multidimensional digital images. *Comm. ACM* **26**, 504–515 (1983).
9. I. Gargantini, An effective way to store quadtrees. *Comm.* ACM **25**, 905–910 (1982).
10. D. J. Abel and J. L. Smith, A data structure based on a linear key for rectangle retrieval. *Comp. Vision Graph. Image Proc.* **24**, 1–13 (1983).
11. D. J. Abel, A B^+-tree structure for large quadtrees. *Comp. Vision Graph. Image Proc.* **27**, 19–31 (1984).
12. L. Jones and S. S. Iyengar, Representation of a region by a forest of quadtrees. *IEEE PRIP 81 Conference*, Dallas, 57–59 (1981).
13. M. A. Oliver and N. E. Wiseman, Operations on quadtree encoded images. *Comp. J.* **26**, 82–91 (1983).
14. V. Raman and S. S. Iyengar, Properties and applications of forests of quadtrees for pictorial data representation. *BIT* **23**, 472–486 (1983).
15. J. R. Woodwark, The explicit quad tree as a structure for computer graphics. *Comp. J.* **25**, 235–238 (1982).
16. H. H. Atkinson, I. Gargantini and M. V. S. Ramanath, Determination of the 3D border by repeated elimination of internal surfaces. *Comp. J.* **32**, 279–295 (1984).
17. H. H. Atkinson, I. Gargantini and T. R. S. Walsh, Counting regions, holes and their level of nesting in time proportional to the border. *Comp. Vision Graph. Image Proc.* 29 (1985), 196–215.
18. I. Gargantini and H. H. Atkinson, Linear quadtrees: a blocking technique for contour filling. *Pattern Recognition* **17**, 285–293 (1984).
19. I. Gargantini, Linear oct-trees for fast processing of three-dimensional objects. *Comp. Graph. Image Proc.* **20**, 365–374 (1982).
20. H. H. Atkinson, I. Gargantini and T. R. Walsh, Filling by quadrants or octants, to appear.
21. C. R. Dyer, The space efficiency of quadtrees. *Comp., Vision Graph. Image Proc.* **19**, 335–348 (1982).
22. M. A. Bauer, Set operations with linear quadtrees. *Comptr. Vision Graph.* Image Proc. 29 (1985), 248–258.
23. E. Kawaguchi and T. Endo, On a method of binary picture representation and its application to data compression. *IEEE Trans. PAMI* **5**, 27–35 (1980).