A GENERALIZED COMPARISON OF QUADTREE AND BINTREE STORAGE REQUIREMENTS

Clifford A. Shaffer
Ramana Juvvadi
Lenwood S. Heath

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

ABSTRACT

The quadtree and the bintree data structures are two variants on the principle of hierarchical regular decomposition applied to image representation. A comparison of the storage requirements for images represented by these two methods is presented. The relative storage efficiency of quad- and bintrees is determined by two factors: the relative node sizes for the two representations as determined by the data structure implementation, and the number of quadtree leaf node pairs that merge to form a single leaf node after conversion to the bintree representation. A probabilistic model for images is developed to analyze the merging probability of quadtree leaf node pairs. The analysis reveals that exactly one half of such pairs are expected to merge. Empirical results closely match these calculations. The resulting storage efficiency for a number of representative implementations is discussed. Each of the data structures has implementations (and associated applications) for which it is more space efficient.

Keywords and phrases: hierarchical data structures, quadtrees, bintrees.

February 8, 2010

1. INTRODUCTION

The widespread use of raster-based image representation methods for computer graphics, image processing, and computer aided cartography has motivated the investigation of more efficient raster-based data structures. The quadtree and bintree data structures [Same89, Same90] are two variants on the principle of hierarchical regular decomposition applied to image representation. The goal of hierarchical representations is to increase the efficiency of spatial data processing by reducing the storage required to represent an image, at the same time reducing the time required to process the image by minimizing the data to be processed.

Given a binary image represented by a $2^n \times 2^n$ array of pixels, both the quadtree and the bintree take advantage of the grouping of like-valued pixels into blocks. The *region* quadtree (henceforth referred to simply as a quadtree*) represents a homogeneous array of pixels as a single block, corresponding to a leaf node in the quadtree. Arrays whose pixel values are not all the same are subdivided into four equal-sized sub-arrays represented by an internal node with four children (labeled NW, NE, SW, and SE as in Figure 1), each representing a corresponding sub-array. These sub-arrays are recursively divided into smaller sub-arrays until each such block is homogeneous. Figure 1 illustrates the quadtree representation for an image.

The region bintree is similar to the region quadtree in that it is a hierarchical representation with each leaf node of the tree representing a homogeneous portion of the array. The bintree subdivides an array into two equal-sized portions by splitting the array parallel to one axis if it is not homogeneous. Non-homogeneous sub-arrays are then split parallel to the second axis. Non-homogeneous sub-sub-arrays would again be split parallel to the first axis. In general, the split axis alternates between the first and second axis. Figure 2 illustrates the bintree decomposition for the image of Figure 1; here the first split is parallel to the x-axis.

* We should point out that the region quadtree is by no means the only form of quadtree, as might appear from our use of the word "quadtree" in this paper. A large number of quadtree variants exist for the purpose of storing other types of data, such as points, lines, rectangles and higher level primitives, in two, three or more dimensions (see [Same89, Same90] for an introduction). While the analysis of leaf merging presented in Section 3 is only relevant to images, the methodology of Sections 2 and 4 can be used to compare the quad- and bintree versions for any of these representations.

The purpose of this paper is to compare the relative space requirements for the quadtree and the bintree. Others [Tamm84b, Cohe85] have previously studied the relative space requirements when the trees are to be encoded for the purpose of transmission or archiving. In particular, the approach of Tamminen [Tamm84b] was to represent the quadtree and bintree as DF expressions [Kawa80] with further compression achieved by using Huffman-like encoding on the values of each node. The space requirements using appropriate variants of this implementation for quadtrees and bintrees were then compared analytically and empirically. Under these conditions, Tamminen concluded that the bintree should occupy about 10% less space than the quadtree based on a random line model for images and an optimal coding scheme. Tamminen's empirical results indicate that the bintree is slightly more space efficient than the quadtree for multicolor images, and slightly less space efficient for binary images using this particular coding scheme.

In this paper we consider the relative storage requirements for the two data structures in more general terms. We first derive equations representing the storage requirements for any implementation by parameterizing the space required for the various components of the tree structure. We then analyze analytically and empirically the expected number of nodes for the bintree relative to the corresponding quadtree. Finally, we consider the expected storage requirements for a number of representative implementations. Again, our aim is not to compare the merits of particular *implementations* for the two data structures, since these merits may be application dependent. Rather, we provide a methodology for comparing space requirements of the *data structures*.

2. AN EQUATION FOR QUADTREE AND BINTREE SPACE REQUIREMENTS

We seek to compare the relative space requirements for the quadtree and the bintree. As such, we are not trying to analyze the absolute size of a quadtree or bintree for any image or class of images. Given the quadtree for an image, we wish to determine the relative size of the bintree for that same image. Thus, we are solely interested in the relationship between the structures of the two trees.

A quadtree leaf at level i represents a square block of size $2^i \times 2^i$ pixels. A bintree leaf can represent either a square (equivalent to a quadtree leaf node) or a rectangle (equivalent to joining two quadtree leaf nodes). Specifically, if we define the bintree decomposition to first divide a block by means of a line parallel to the x-axis, then the equivalent quadtree leaf blocks that may combine to form a single bintree leaf node must, in the quadtree, be leaf siblings which are either NW and NE children of their mutual parent, or SW and SE children of their mutual parent.* We define any two quadtree nodes with this relationship to be *bin siblings*. If two bin siblings in a quadtree are represented by a single leaf in the bintree (i.e., they have the same leaf value in the quadtree), then we say that the two quadtree leaves *merge*.

It must be understood that a bintree leaf node may never represent the merging of two pairs of bin siblings, because the nodes comprising these pairs of bin siblings would correspond to four sibling quadtree leaf nodes with the same value. In such a case, these four quadtree leaf nodes would already be represented by a single quadtree leaf node.

At first it might appear that a bintree always takes less space than a quadtree because it usually has fewer leaf nodes (it cannot have more leaf nodes). In addition, bintree internal nodes have two children as opposed to four for the quadtree, resulting in less space required by internal nodes for any implementation requiring pointers from parent nodes to children. However, each pair of bin siblings that is not merged requires additional internal nodes for the bintree, as illustrated in Figure 2b. Furthermore, any internal node in the quadtree whose children are also internal nodes must be represented by three internal nodes in the bintree. Thus, the relative space requirements for bin- and quadtrees will depend on how often bin siblings merge, and the relative storage requirements for leaf and internal nodes.

Before deriving a general equation for the relative storage costs of the two data structures, we first derive upper and lower bounds on the total number of leaf and internal nodes in the

* Alternatively, if the first split were parallel to the y-axis, the NW children could merge with SW children, and NE children could merge with SE children. We arbitrarily assume the first split axis to be parallel to the x-axis unless specified otherwise.

trees. Let L_q and I_q be the number of leaves and internal nodes, respectively, in the quadtree for a particular image. Let L_b and I_b be the number of leaves and internal nodes, respectively, in the corresponding bintree. If all leaf nodes of the quadtree merge with their bin siblings, then the bintree has half as many leaves as the quadtree, i.e., $L_b = \frac{1}{2}L_q$. Since (by basic theorems for k-ary tree structures) $I_b \approx L_b$ while $I_q \approx \frac{1}{3}L_q$, the total number of nodes in the resulting bintree ($2 \cdot L_b = 2 \cdot \frac{1}{2}L_q = L_q$) will require three quarters as many nodes as the quadtree (which has a total of $\frac{4}{3}L_q$ nodes) when all quadtree leaf nodes merge with their bin siblings. When no bin siblings merge, the bintree has as many leaves as the quadtree, i.e., $L_b = L_q$. In this case, the total number of nodes in the bintree ($2L_q$) will be one and a half times as many nodes as in the quadtree.

Define α to be the factor of merging, i.e., $\alpha \cdot L_q$ leaves of the quadtree merge into $\alpha \cdot \frac{L_q}{2}$ leaves of the bintree, $0 \leq \alpha \leq 1$. Then, $L_b = \frac{\alpha \cdot L_q}{2} + (1-\alpha) \cdot L_q = (1 - \frac{\alpha}{2}) \cdot L_q$ and $I_b \approx (1 - \frac{\alpha}{2}) \cdot L_q$.

The second variable affecting the total amount of space required by either the quadtree or bintree structure is the amount of space required by each type of node in the tree, which is dependent on the implementation used. In particular, we must recognize that a leaf node data value, an internal node data value, and a pointer may each require different amounts of storage. Let i_b, l_b, i_q, and l_q be the space occupied by the internal node of a bintree, a leaf node of a bintree, an internal node of a quadtree, and a leaf node of a quadtree, respectively. Let s_b be the total space occupied by a bintree and s_q be the space occupied by a quadtree. Then

$$\frac{s_b}{s_q} = \frac{I_b i_b + L_b l_b}{I_q i_q + L_q l_q} \approx \frac{(1 - \frac{\alpha}{2}) L_q i_b + (1 - \frac{\alpha}{2}) L_q l_b}{\frac{1}{3} L_q i_q + L_q l_q} = \frac{(1 - \frac{\alpha}{2})(i_b + l_b)}{\frac{1}{3} i_q + l_q} = \beta(1 - \frac{\alpha}{2})$$

where $\beta = \frac{i_b + l_b}{\frac{1}{3} i_q + l_q}$.

The value of α ranges between 0 and 1 (i.e., between no quadtree bin siblings merged in the bintree, and all quadtree bin siblings merged in the bintree), which in turn allows the expression $(1 - \frac{\alpha}{2})$ to range between $\frac{1}{2}$ and 1. β quantifies the relationship between the relative number of internal and leaf nodes in each structure (1:1 in the bintree, 1:3 in the quadtree) and the relative

node sizes for the trees. The value of β ranges between 0 and ∞. When β is less than 1, $\frac{s_b}{s_q}$ is also less than 1, which means that the bintree is always more space efficient than the corresponding quadtree. When β is greater than 2, $\frac{s_b}{s_q}$ is always greater than 1, which means the bintree is always less space efficient than the quadtree. When β lies between 1 and 2, the relative space efficiency of the two data structures depends on the value of α. We shall see in Section 4 that "reasonable" values of β range from slightly less than 1 to slightly greater than 2. For the bintree to occupy less space than the quadtree, α must be greater than $2(1 - \frac{1}{\beta})$. These relationships are illustrated by Figure 3. In Section 3 we analyze the expected value of α. In Section 4 we derive the value of β for several representative implementations.

3. AN ANALYSIS OF α

Recall that α is the fraction of quadtree leaves that merge with their bin siblings to form the bintree representation. Therefore, α is dependent on the image represented. What we actually want is an average value of α over all images to be represented.

The main result of this section are Theorems 1 and 2, which together state that the average value of α is $\frac{1}{2}$. The result is proved through a detailed argument based on geometric probability theory, and is supported by empirical evidence.

As the set of images is neither known nor characterized in a manner suited to analysis, we make three simplifying assumptions and argue that they are reasonable for both a typical image and a typical position in the quadtree. These assumptions may not be valid for very complicated images; however, our empirical evidence indicates that the assumptions provide an adequate model for all of the images that we tested. Using these three assumptions, we are able to calculate an average value of α that is approximately correct in the general case.

Consider an internal node in the quadtree. Our first assumption is that a *single* boundary between two regions of the image passes through that node. This is not unreasonable since we only need to consider internal nodes with at least one leaf child, the only nodes that can affect

α. In particular, this assumption is likely to be true for a node at a lower level of the quadtree since the smaller the node, the less likely it is that it contains more than two polygons. Our second assumption is that the single boundary within an internal node can, at least locally, be approximated by a line. This is, again, an assumption that images are not too complicated locally, and again, more likely for the smaller nodes which dominate the space requirements for a typical quad- or bintree [Same85]. Tamminen [Tamm84b] also makes this assumption in his analysis.

Our third assumption concerns the manner in which images are digitized into a matrix of pixels. For us, an image is a partition of the interior of a $2^n \times 2^n$ square into homogeneous subsets. The area of a homogeneous subset of any reasonable image is at least measurable (in the sense of measure theory [Rudi74]). Given a 1×1 square that is represented by a single pixel, our first assumption requires that it contain at most two regions, which we arbitrarily label red and green since the image as a whole may be multicolored. The measure of the red part of the pixel and the measure of the green part of the pixel sum to 1. If the measure for the red part is $< \frac{1}{2}$, then the pixel is green. If the measure for the red part is $> \frac{1}{2}$, then the pixel is red. (We assume that the probability that the measure is *exactly* $\frac{1}{2}$ is 0.) This reasonable assumption simplifies the calculations of the average value of α.

Henceforth, we make these three assumptions and proceed to the calculation of the average value of α. These assumptions should yield a slightly higher value for α than may actually be the case, since the assumptions limit the complexity of the image, thus allowing greater opportunities for bin siblings to merge. As a result of the second and third assumptions, a pixel has the same color as its center. Also, for those pixels having a boundary line passing through them, half are red and half are green on average.

To make use of the second assumption, we draw on results from geometric probability [Sant76]. To a set S of lines randomly chosen from a uniform distribution, there is an associated measure, denoted $\mu(S)$ (all sets of lines that we consider are measurable in the sense of measure

theory and have finite measure). We may view this measure as being *proportional* to the probability that a line is in S. We are particularly interested in the set of lines that intersect a bounded convex set K (such as a line segment, triangle, convex quadrilateral, or circle). Such a K has a boundary, denoted ∂K, with a well-defined perimeter, $\mathrm{P}(\partial K)$. If p and q are points, then $\mathrm{d}(p, q)$ is the Euclidean distance from p to q. As a special and degenerate case, the perimeter of the line segment (p, q) is taken to be $\mathrm{P}((p, q)) = 2\mathrm{d}(p, q)$. The following result is demonstrated in [Sant76].

Proposition. Let K be a bounded convex set in the plane. Let S be the set of lines that intersect K. Then, the measure of S is $\mu(S) = \mathrm{P}(\partial K)$, the perimeter of K.

In other words, the perimeter of a bounded convex set is the measure of the set of lines that intersect it. To apply the proposition to line segments, let (p, q) be a line segment. Let L be the set of lines that intersect (p, q). Then, the measure of L is $\mu(L) = 2\mathrm{d}(p, q)$, twice the Euclidean distance from p to q.

We now derive two lemmas needed to calculate the average value of α.

Lemma 1. Suppose $\triangle abc$ is a triangle with sides of length $r = d(a, b)$, $s = d(b, c)$, and $t = d(a, c)$. Let L be the set of lines that intersect $\triangle abc$, but avoid the side (a, c). Then,

$$\mu(L) = r + s - t.$$

Proof: Let M be the set of lines that pass through $\triangle abc$, and let N be the set of lines that pass through (a, c). By the Proposition, $\mu(M) = r + s + t$ and $\mu(N) = 2t$. But, $L = M - N$. Therefore,

$$\mu(L) = \mu(M) - \mu(N)$$

$$= r + s - t.$$

□

Lemma 2. Suppose p_1, p_2, p_3, p_4 are the vertices of a convex quadrilateral with sides of length

$$s_1 = d(p_1, p_2)$$

$$s_2 = d(p_2, p_3)$$

$$s_3 = d(p_3, p_4)$$

$$s_4 = d(p_1, p_4).$$

Let the diagonals of the quadrilateral have length

$$d_1 = d(p_1, p_3)$$

$$d_2 = d(p_2, p_4).$$

Let L be the set of lines that pass through the opposite sides (p_1, p_2) and (p_3, p_4). Then,

$$\mu(L) = d_1 + d_2 - s_2 - s_4.$$

Proof: Consider the triangle $\triangle p_1 p_2 p_3$. By Lemma 1, the measure of the set of lines that intersect the sides (p_1, p_2) and (p_2, p_3) (and therefore avoid the diagonal (p_1, p_3)) is $s_1 + s_2 - d_1$. Similarly, the measure of the set of lines that intersect the sides (p_1, p_2) and (p_1, p_4) (and therefore avoid the diagonal (p_2, p_4)) is $s_1 + s_4 - d_2$. The measure of the set of all lines that intersect the side (p_1, p_2) is $2s_1$. Therefore, the measure of L is

$$\begin{aligned}\mu(L) &= 2s_1 - (s_1 + s_2 - d_1) - (s_1 + s_4 - d_2) \\ &= d_1 + d_2 - s_2 - s_4.\end{aligned}$$

□

With these lemmas behind us, we return to the calculation of the average value of α. There are two cases to consider. The first case (Theorem 1) is that of a 2×2 pixel internal node, which always has four leaf children. The second case (Theorem 2) is that of a $2^{k+1} \times 2^{k+1}$ pixel internal node, $k > 0$. Such a node may have one, two, or three leaf children, with the remaining children always being internal nodes.

Theorem 1. Consider 2×2 pixel internal nodes only. Then the average value of α is $\frac{1}{2}$.

Proof: Figure 4 illustrates the situation. For a 2×2 matrix of pixels, there is a *constricted* square (shown dashed) that connects the centers of the four pixels. This square controls whether the node is internal or leaf. If a boundary line intersects the constricted square, then the node is internal; otherwise, the node is leaf. This follows from the earlier observation that the color of a pixel is the color of its center.

We identify two cases in which a line intersects the constricted square. In the first case, the line passes through adjacent sides of the square (Figure 4a). In the second case, the line passes through opposite sides of the square (Figure 4b).

In the first case, there are always three pixels of one color and one pixel of the other color. Two of the three pixels of the same color always merge. Thus, out of the four leaves in the quadtree, two always merge.

In the second case, there is always a pair of pixels of each color. Also, two pixels of the same color are never diagonally opposite each other. Thus, the pairs are either vertically or horizontally aligned. When the pairs are horizontally aligned, the four leaves merge to form two distinct bin siblings. When the pairs are vertically aligned, no leaves merge. Vertical and horizontal alignment occur with equal probability. Therefore, on average, two of the four leaves merge.

In both cases, on average, two of the four leaves merge. Thus, the average value of α for 2×2 pixel internal nodes is $\frac{1}{2}$. □

Theorem 2. Consider $2^{k+1} \times 2^{k+1}$ pixel internal nodes only, $k > 0$. Then, the average value of α is $\frac{1}{2}$.

Proof: Figure 5 illustrates the situation. There is a *constricted* square (shown dashed) with sides of length $2^{k+1} - 1$ that connects the centers of the four corner pixels. For the corresponding node in the quadtree to be internal, a boundary line of the image must intersect the square. Otherwise,

the node is a leaf. Thus, the measure of the set I of lines that force an internal $2^{k+1} \times 2^{k+1}$ node is

$$\mu(I) = 4\Big(2^{k+1} - 1\Big) = 2^{k+3} - 4.$$

Now we wish to know the configuration of red, green, and internal nodes among the four $2^k \times 2^k$ sons of the internal node. The configuration is controlled by the relationship between the boundary line and the constricted squares of the four central squares shown dashed in Figure 6. Each (constricted) central square has sides of length $2^k - 1$ and controls whether its corresponding $2^k \times 2^k$ node is internal or leaf. In particular, a son is internal if and only if the boundary line passes through its central square. We identify six cases with respect to the boundary line (shown in Figure 6).

(a) The line passes through three central squares.

(b) The line passes through two central squares that are adjacent horizontally or vertically, and no others.

(c) The line passes through two central squares that are opposite diagonally, and no others.

(d) The line passes through only one central square and does not pass between two other central squares.

(e) The line passes through only one central square and passes between two other central squares.

(f) The line does not pass through any central square but has two central squares on one side and two on the other.

As no line can pass through all four central squares, these six cases cover all possibilities for a $2^{k+1} \times 2^{k+1}$ internal node.

Our next task is to determine the measure of the set of lines for each case. Each case has either two, four, or eight symmetric subcases. For example, case (d) has four symmetric subcases,

depending on whether the line passes through the NW, SW, NE, or SE central square. Our strategy for calculating the measure of the set of lines corresponding to any case is first to calculate the measure for one subcase and then to multiply by the number of subcases. During the calculations, the following function of an integer j will prove useful

$$f(j) = \sqrt{(2^j - 1)^2 + 1}.$$

$f(j)$ gives the hypotenuse of a right triangle having sides of length $2^j - 1$ and 1. In particular, we will need the values $f(k)$ and $f(k+1)$.

Case (a). See Figure 6a. Consider first the set L_3 of lines that pass through the NW, NE, and SE central squares only. L_3 is characterized by the property that any such line intersects both edges (a, b) and (b, c) of triangle $\triangle abc$. The sides of triangle $\triangle abc$ have lengths $d(a, b) = d(b, c) = f(k)$, and $d(a, c) = 2^k\sqrt{2}$. By Lemma 1, the measure of L_3 is

$$\mu(L_3) = 2f(k) - 2^k\sqrt{2}.$$

There are four symmetric subcases for Case (a). Therefore, the measure for Case (a) is

$$4\Big(2(f(k) - 2^k\sqrt{2}\Big) = 8f(k) - 2^{k+2}\sqrt{2}.$$

Case (b). See Figure 6b. Consider the subcase of lines that pass through the upper two central squares only. Such lines must pass through sides (p_1, p_2) and (p_3, p_4) of rectangle p_1, p_2, p_3, p_4. By Lemma 2, the measure of the set of such lines is $2f(k) - 2$. From this set we must eliminate two sets of lines of type L_3 which also pass through (p_1, p_2) and (p_3, p_4) (those passing through the NW, NE and SE quadrants, and those passing through the SW, NW and NE quadrants). Therefore, the measure of the set of lines that pass through the upper two central squares only is

$$2f(k) - 2 - 2\Big(2f(k) - 2^k\sqrt{2}\Big) = 2^{k+1}\sqrt{2} - 2 - 2f(k).$$

There are four symmetric subcases of Case (b). Therefore, the measure of Case (b) is

$$4\Big(2^{k+1}\sqrt{2} - 2 - 2f(k)\Big) = 2^{k+3}\sqrt{2} - 8 - 8f(k).$$

Case (c). See Figure 6c. Consider the subcase of lines that pass through the NW and SE central squares, and no others. Such lines pass through edges (p_1, p_2) and (p_3, p_4) of quadrilateral p_1, p_2, p_3, p_4. By Lemma 2, the measure of the set of such lines is $2f(k+1) - 2 \cdot 2^k\sqrt{2}$. We must eliminate two sets of lines of type L_3. The remaining measure is therefore

$$2f(k+1) - 2^{k+1}\sqrt{2} - 2\Big(2f(k) - 2^k\sqrt{2}\Big) = 2f(k+1) - 4f(k).$$

There are two symmetric subcases of Case (c). Therefore, the measure of Case (c) is

$$2\Big(2f(k+1) - 4f(k)\Big) = 4f(k+1) - 8f(k).$$

Case (d). See Figure 6d. Consider the set of lines that pass through the NW central square only and do not pass between the other squares. Such lines are characterized by the property of passing through edges (a, b) and (b, c) of triangle $\triangle abc$. By Lemma 1, the measure of the set of such lines is $2^{k+1} - 2^k\sqrt{2}$. There are four symmetric subcases for Case (d). Therefore, the measure of Case (d) is

$$4\left(2^{k+1} - 2^k\sqrt{2}\right) = 2^{k+3} - 2^{k+2}\sqrt{2}.$$

Case (e). See Figure 6e. Consider only lines that pass through the NW central square and between the NE and SE central squares. Such lines are characterized by the property of passing through edges (p_1, p_2) and (p_3, p_4) of quadrilateral p_1, p_2, p_3, p_4. By Lemma 2, the measure of the set of such lines is

$$f(k) + \Big(2^{k+1} - 1\Big) - \Big(2^k - 1\Big) - f(k+1) = 2^k + f(k) - f(k+1).$$

There are eight symmetric subcases of Case (e). Therefore, the measure of Case (e) is

$$8\Big(f(k) - f(k+1) + 2^k\Big) = 2^{k+3} + 8f(k) - 8f(k+1).$$

Case (f). See Figure 6f. Consider lines with the NW and SW central squares to the left and the NE and SE central squares to the right. Such lines are characterized by the property of passing through edges (p_1, p_2) and (p_3, p_4) of rectangle p_1, p_2, p_3, p_4. By Lemma 2, the measure of the set of such lines is

$$2f(k+1) - 2\Big(2^{k+1} - 1\Big) = 2f(k+1) + 2 - 2^{k+2}.$$

There are two symmetric subcases of Case (f). Therefore, the measure of Case (f) is

$$2\Big(2f(k+1) + 2 - 2^{k+2}\Big) = 4f(k+1) + 4 - 2^{k+3}.$$

The results of these calculations are summarized in Table 1. The second column gives the calculated measures. As a check, the sum of the measures for the six cases is $2^{k+3} - 4$ which is the perimeter of the constricted square and the measure of I. The third column gives the number of leaves for each case. The number of leaves always equals the number of central squares that are not intersected by the line. The fourth column gives the expected number of leaves that merge. Note that this is not always the number of leaves that merge. In Case (b) for example, there are always two internal children and two leaf children. If the bintree axis splits the internal from the leaf children, then the two leaves merge; otherwise, no leaves merge. These two possibilities occur with equal probability. Therefore, the expected number of leaves that merge is one, as shown in Table 1.

The number of leaves weighted by the measures is

$$\begin{aligned} A = {} & 1\Big(8f(k) - 2^{k+2}\sqrt{2}\Big) + 2\Big(2^{k+3}\sqrt{2} - 8 - 8f(k)\Big) + 2\Big(4f(k+1) - 8f(k)\Big) + \\ & 3\Big(2^{k+3} - 2^{k+2}\sqrt{2}\Big) + 3\Big(2^{k+3} + 8f(k) - 8f(k+1)\Big) + 4\Big(4f(k+1) + 4 - 2^{k+3}\Big) \\ = {} & 2^{k+4} \end{aligned}$$

Table 1. Results of Theorem 2.			
Case	Measure	Leaves	Merged
(a)	$8f(k) - 2^{k+2}\sqrt{2}$	1	0
(b)	$2^{k+3}\sqrt{2} - 8 - 8f(k)$	2	1
(c)	$4f(k+1) - 8f(k)$	2	0
(d)	$2^{k+3} - 2^{k+2}\sqrt{2}$	3	2
(e)	$2^{k+3} + 8f(k) - 8f(k+1)$	3	1
(f)	$4f(k+1) + 4 - 2^{k+3}$	4	2

The number of leaves that merge weighted by the measures is

.

$$B = 0\Big(8f(k) - 2^{k+2}\sqrt{2}\Big) + 1\Big(2^{k+3}\sqrt{2} - 8 - 8f(k)\Big) + 0\Big(4f(k+1) - 8f(k)\Big) +$$
$$2\Big(2^{k+3} - 2^{k+2}\sqrt{2}\Big) + 1\Big(2^{k+3} + 8f(k) - 8f(k+1)\Big) + 2\Big(4f(k+1) + 4 - 2^{k+3}\Big)$$
$$= 2^{k+3}$$

The average value of α is the ratio of these two, i.e.,

$$\alpha = \frac{B}{A} = \frac{1}{2}.$$

□

Another point to consider is that a high value for α implies a high bias of horizontal borders when the first split is parallel to the x-axis. If the split priority is reversed, then this same image would have a low α. There is a limit on how high α may become without requiring some directional bias in the image. For groups of four leaf siblings, the greatest value for α with no directional bias is $\frac{1}{2}$. Internal nodes may have three leaf children of the same color, and a fourth internal child. Two of the leaf children may merge regardless of orientation, yielding the maximum unbiased value of α approaching $\frac{2}{3}$ (this is never actually realized as there must be some nodes with four leaf children). Thus, only pathological images may have α considerably higher than $\frac{1}{2}$. Images with α significantly lower than $\frac{1}{2}$ may occur for a number of reasons aside from a strong bias in the direction perpendicular to the first split axis. One possibility is that the typical region size is very

small in comparison to the resolution, yielding many cases where more than two regions fall within the four pixels making up the siblings in the quadtree. This is equivalent to more than a single border falling within the 2×2 block (i.e., an image violating the assumptions of our analysis). Images in which this happens for many 2×2 blocks are not well suited to representation by either quad- or bintrees, as this property implies that the quad- or bintree would be very large.

In order to test our hypothesis that the expected value of α is close to $\frac{1}{2}$, we compared the number of nodes required for the quadtree and bintree representations of several images. Three of the images were maps taken from a Geographic Information System testbed reported in [Shaf87a]. These maps include a landuse map, a floodplain map, and the regions resulting from the 100 foot contour lines of a topography map. We also tested two texture images (identified as "stone" and "pebble"), hand thresholded so as to minimize noise and generate homogeneous regions for quadtree encoding. Finally, we also calculated the space requirements for the largest circle that would fit within the region represented by the quad- or bintree, as reported in [Tamm84a].

Table 2 shows both the total number of nodes and the number of internal nodes in the quadtree and bintree for the six test cases. For each image, we calculated the number of merges resulting from choosing both the x-axis and the y-axis as the primary split axis. From the table, we see that the values for α ranged from .423 to .569. The image with the lowest value for α (the stone texture with first split parallel to the x-axis) also had the highest value for α when split parallel to the y-axis. This is consistent with the prediction that even slight deviations from the expected value of $\frac{1}{2}$ are due to directional bias. When we take the average of the x-axis split and y-axis split values for α, we find that the result ranges between .487 and .502 for the six test images.

4. DATA STRUCTURE IMPLEMENTATIONS

The quad- and bintree implementations to be examined generally fall into one of three categories.* These are standard pointer-based tree structures (i.e., a representation with explicit

* A fourth implementation scheme overlays the actual quad- or bintree onto the space required by a pyramid representation for the image (see [Know80, Shaf87b]). While time efficient for large memory machines (since nodes may be located quickly), it is extremely space inefficient.

Table 2. Empirical results for α on six test images.					
Image	Quadtree Nodes Total (internal)	Bintree		α (x)	α (y)
		x Total (internal)	y Total (internal)		
Floodplain	6997 (1749)	7779 (3889)	7981 (3990)	.518	.479
Topography	33349 (8337)	37227 (18613)	38357 (19178)	.512	.466
Landuse	38065 (9516)	43573 (21786)	42939 (21469)	.474	.500
Stone	35009 (8752)	41417 (20708)	37575 (18787)	.423	.569
Pebble	59933 (14983)	68905 (34452)	66671 (33335)	.467	.517
Circle	5317 (1329)	5975 (2987)	5975 (2987)	.502	.502

pointers between parents and children as illustrated by Figures 1d and 2b), linear quad- or bintrees [Garg82] and DF expressions [Kawa80].

The linear quadtree representation stores a list of records where each record corresponds to a leaf node of the pointer-based structure. These records store the data value for the leaf node (e.g., red or green) and an address value describing the position and size of the corresponding block in the image. Some versions of the linear quadtree further reduce space requirements for special types of images. Gargantini's original formulation [Garg82] does not store white (background) nodes for binary images, while Lauzon, et al [Lauz85] use a form of two dimensional runlength encoding by only storing the first of a sequence of leaf nodes with the same value. Since the linear quadtree and linear bintree store only leaf nodes, clearly the bintree will be more space efficient than the corresponding quadtree for any of these implementations.

DF expressions are extremely space efficient. We describe the representation of a binary quadtree or bintree as a DF expression. The symbol '(' represents an internal node, and the symbols 'B' and 'W' represent black and white leaf nodes, respectively, in a binary image. The DF expression is derived by traversing the quad- or bintree in preorder, enumerating the nodes by this coding scheme as they are visited. The quadtree decomposition for Figure 1c would therefore be represented by the string "(W(WWBB(W(WBBBWB(BB(BBBWW". The bintree decomposition of Figure 2 would be encoded as "((W(WB(((W((WBB(WB(B((B(BWW". Using a naive coding scheme, both leaf and internal nodes require a single character of storage (depending on the number of colors represented). If we use these values in our equation for comparing the space efficiencies

of quadtrees and bintrees, we assign i_b, l_b, i_q, and l_q each to be 1. This yields a β value of 1.5. For this implementation, α must be $\frac{2}{3}$ for the bintree to be as space efficient at the quadtree – higher than the average value of α.

Since the symbol "(" appears quite often, we could use the concept of Huffman encoding to reduce the total number of bits required by encoding internal nodes with bit value '0', and leaf nodes with bit value '1' followed by another '1' for black and '0' for white. Thus, 1 bit is required for every internal node and 2 bits for every leaf node. Our values for computing β are then $i_b = 1$, $l_b = 2$, $i_q = 1$, and $l_q = 2$. This implies $\beta = \frac{9}{7}$. This implementation is the one suggested by Tamminen in his space analysis [Tamm84b]. We see therefore that the value for α need only be $\frac{4}{9}$ (below the average value of α) for the bintree to be more space efficient than the quadtree under this implementation.

For a multicolor image, the preceding analysis can be extended by increasing the size of a leaf node value so as to store additional colors. For example, if we wish to represent 128 colors, we require 7 bits of storage. Thus, internal nodes still require a single bit using a Huffman-like coding scheme, while the leaf nodes require 8 bits (1 bit to determine that it is a leaf node, and 7 bits for the color value). Our values for calculating β are then $i_b = 1$, $l_b = 8$, $i_q = 1$, and $l_q = 8$, yielding $\beta = \frac{27}{25}$. Nearly all images (although not every image) would be more efficiently represented by the bintree with this implementation, since α need be only $\frac{4}{27}$. These calculations roughly agree with Tamminen's empirical results showing that the bintree for binary images requires slightly more space than the quadtree, while the bintree for multicolor images requires slightly less space than the corresponding quadtree.

We see from the above analysis that the linear bintree is always more space efficient than the linear quadtree, and that the bintree DF-expression is often more space efficient than the quadtree DF-expression. However, these representations are not always to be preferred over the pointer-based representation. The DF-expression is particularly useful for archival purposes since

it is more space efficient than the other representations discussed. It is also quite efficient for operations that traverse the entire tree in the order stored by the DF-expression. However, the DF-expression does not permit random access to nodes. For example, the value of the leaf node at a specified position can be determined only by sequentially processing the DF-expression until the desired node is encountered.

The linear quadtree has been used as a representation for images maintained in secondary storage (for example, see [Abel84, Shaf87a]). For this purpose it has been found to be more effective than the pointer-based implementation since it lends itself readily to paging techniques to minimize I/O costs. Discovery of an effective paging technique for the pointer-based quadtree representation may reduce interest in the linear representation [Brow92]. In any case, the linear quadtree does not appear to be superior to the pointer-based quadtree for in-core storage applications (such as any real-time application of quadtree-like data structures, e.g., for robotics [Shaf91]). Since the linear quadtree need not store internal nodes and pointers, it might be expected to be more space efficient than pointer-based representations. Surprisingly, Samet and Webber [Same86] demonstrate that for many images, linear quadtree encoding actually requires more space than an efficient implementation of the pointer-based quadtree. This is due to the fact that the address field for the linear quadtree can require more space for relatively sparse trees than do the pointers and internal nodes. "Relatively sparse" often includes what would be considered typical images. Another way to view this result is that, for a given memory size, the length of a pointer (which is determined by memory size) may be significantly less than the length of the linear quadtree address code (which is determined by image resolution). This space savings is confirmed in [Brow92], while the contrary view is taken by Walsh [Wals85] for linear quadtree representing binary images. The pointer-based implementation is usually simpler to implement, and may be more time efficient (although no empirical comparison has been reported for in-core representations).

We will now analyze the storage requirements for a number of pointer-based implementations of the quad- and bintree. We begin with an extremely inefficient (although easily programmed)

implementation in which we assume that a pointer and a data value occupy the same amount of space, and that both leaf nodes and internal nodes maintain space for pointers. This results from implementing all nodes of the tree with the same record type. All bintree nodes have two pointers and one data item while quadtree nodes have four pointers and one data item. Thus, any bintree node requires 3 units of storage whereas any quadtree node requires 5 units of storage. Our values for calculating β are then $i_b = 3$, $l_b = 3$, $i_q = 5$, and $l_q = 5$ which results in $\beta = \frac{9}{10}$. Since β is less than 1, the bintree is always more efficient than the quadtree for this implementation.

In practice, such an inefficient implementation would not be recommended. We can make the representation more efficient by first removing all null pointers from the leaf nodes. Such an approach is described in [Same86], and an actual implementation is reported in [Oska88]. In addition, the data field for a node is likely to be smaller than the pointer fields stored in the internal nodes. Thus, the minimum possible number of bits should be allocated for data. Assume data takes x units of storage and a pointer takes p units. Our values for computing β are $i_b = x + 2p$, $l_b = x$, $i_q = x + 4p$, and $l_q = x$ which yields $\beta = 1.5$ regardless of the values for x and p. In other words, when leaf nodes do not store pointers, the sizes of the data and pointer fields cancel out of the equation. For a β of 1.5, α is required to be $\frac{2}{3}$ for the bintree to be as space efficient as the quadtree. As this is higher than the average value of α, most images will require more space for the bintree than for the corresponding quadtree. This implementation is easily programmed, and should be considered as the minimum standard in space efficiency for a pointer-based tree representation.

Alternatively, the implementation just described can be viewed in terms of pointers to nodes. Internal node pointers point either to other internal nodes, or to leaf nodes. The "pointers" to leaf nodes are replaced by the values of those leaf nodes (since the leaf node data value typically requires less space than a pointer). Since there are no true leaf nodes, there is no need for a data field in the internal nodes. Thus, the total amount of storage required is one "pointer" value for each node.

With some additional programming effort, we can make this representation even more efficient by recognizing that the storage required by data values can be much smaller than the storage required by a pointer. If an internal node is the parent of a leaf, that "pointer" to the leaf is actually replaced by the data value for that leaf, which is smaller than the size required for a true pointer to another internal node. Such a coding scheme is analogous to the Huffman-like compression method used for DF-expressions. However, it is the leaf nodes that most benefit from this compression since there are typically more nodes than data values. Since every node must be pointed to exactly once, and since the "pointers" to leaf nodes are smaller than the pointers to internal nodes, we get the following equation if we set the size of a leaf "pointer" (i.e., its data value) to 1, and an internal node pointer to p (i.e., a true pointer is p times as large as a data value).

$$\frac{S_b}{S_q} = \frac{I_b p + (I_b + L_b)}{I_q p + (I_q + L_q)} = \frac{(1 - \frac{\alpha}{2})3(p+2)}{p+4}$$

This yields $\beta = \frac{3(p+2)}{p+4}$. For a typical implementation the data value might require 1 byte (i.e., 256 possible data values) while a pointer typically requires 4 bytes ($p = 4$). This yields $\beta = 2.25$. If a pointer takes only twice the amount of storage required for a data value, we get $\beta = 2$. As p varies from 0 to ∞, β varies from 1.5 to 3. So for many implementations of this type (whenever $\beta \geq 2$), the quadtree is always more space efficient than the bintree.

The results of this section are summarized in Table 3. Here we list each implementation with a representative value for β. The "α Threshold" column shows the point at which the two data structures will require the same amount of stpace for that implementation; for lower values of α the quadtree is more space efficient.

5. CONCLUSIONS

In this paper we have provided a general equation for the relative space requirements of the quadtree vs. the bintree. We have seen that total storage requirements depend on two factors:

Table 3. β values for various implementations.		
Implementation	β	α Threshold
Linear	< 1	< 0
DF – binary	1.5	2/3
DF – binary (Huffman)	9/7	4/9
DF – multicolor (Huffman)	27/25	4/27
Inefficient pointers	9/10	< 0
Efficient pointers	1.5	2/3

the merging rate of nodes in the bintree *vis a vis* the quadtree (which we have labeled α), and the relative amount of storage required for nodes in the two structures (which we have labeled β). Analysis, confirmed by empirical observation, indicate that the expected value of α is about 0.5 — in other words, half of the quadtree leaf nodes can be expected to merge with their bintree sibling. This result is particularly interesting as it is independent of the type of image represented. The actual values ranged between 0.423 and 0.569 (with these two values occurring for the same image using different initial split axis, indicating a relatively high directional bias). One way to view this result is that the image model affects the quad- and bintree structures equally, and can be factored out when comparing the two. The single exception to this is the issue of directional bias as discussed in Section 3.

In general, DF-expression implementations minimize the space required for an internal node as compared to a leaf node, while the linear representations carry this to the extreme by eliminating the internal nodes entirely. Thus the bintree, which typically requires more internal nodes than the quadtree while requiring fewer leaf nodes, often is more space efficient for typical images under these representations. Conversely, pointer-based implementations minimize the size of leaf nodes as compared to internal nodes. For these representations, we conclude that the quadtree is more space efficient than the bintree for "typical" images in which approximately half of the quadtree leaf nodes merge with their bin siblings.

When considering higher dimensional applications, in particular 3-d graphics, we note that the storage requirements for internal nodes become a smaller factor for the 3-d version of the

"quadtree" (an octree [Meag82]), while for the 3-d bintree the number of internal nodes is still one less than the number of leaf nodes. Thus we expect for three dimensions that the pointer-based octree to be much more space efficient than the pointer-based 3-d bintree.

While the preceding analysis has considered only space comparisons between the quad- and bintrees, we must also observe that time requirements can also be expected to differ. Many quadtree (and corresponding bintree) algorithms can be performed by a simple traversal of the tree, performing some task at each node. For such algorithms, the time required is proportional to the total number of nodes in the tree. Many other algorithms involve examining a particular location by descending the tree from the root to the corresponding leaf node, or by means of neighbor-finding operations [Same82, Same85] that ascend and descend the tree to locate desired neighboring nodes. Recall that internal nodes of the quadtree are represented in the bintree by three internal nodes arranged in a pyramid as illustrated by Figure 2. Operations that require neighbor-finding and searching operations will be required to visit two of these internal nodes in the bintree for each corresponding visit to an internal node in the quadtree. Thus, the total number of nodes visited by the bintree operation will be greater, resulting in greater time cost.

Our methodology for calculating space requirements can also be applied to time requirements. Many useful algorithms visit each node of the tree structure a fixed number of times. In such a case, given time requirements for L_q, I_q, L_b, and I_b, under a given implementation, the total time expected can be calculated.

6. ACKNOWLEDGEMENTS

We wish to extend our thanks to the reviewers for their comments.

7. REFERENCES

1. [Abel84] D.J. Abel, A B^+-tree structure for large quadtrees, *Computer Vision, Graphics, and Image Processing 27*, 1(July 1984), 19-31.

2. [Brow92] P.R. Brown, *A paging scheme for pointer-based quadtrees*, Masters Thesis, Department of Computer Science, Virginia Tech, May 1992.

3. [Cohe85b] Y. Cohen, M.S. Landy, and M. Pavel, Hierarchical coding of binary images, *IEEE Transactions on Pattern Analysis and Machine Intelligence 7* 3(May 1985), 284-298.

4. [Garg82] I. Gargantini, An effective way to represent quadtrees, *Communications of the ACM 25*, 12(December 1982), 905-910.

5. [Kawa80] E. Kawaguchi and T. Endo, On a method of binary picture representation and its application to data compression, *IEEE Transactions on Pattern Analysis and Machine Intelligence 2*, 1(January 1980), 27-35.

6. [Know80] K. Knowlton, Progressive transmission of grey-scale and binary pictures by simple, efficient, and lossless encoding schemes, *Proceedings of the IEEE 68*, 7(July 1980), 885-896.

7. [Lauz85] J.P. Lauzon, D.M Mark, L. Kikuchi and J.A.Guevara, Two-dimensional run-encoding for quadtree representation, *Computer Vision, Graphics, and Image Processing 30*, 1(April 1985), 56-59.

8. [Meag82] D. Meagher, Geometric modeling using octree encoding, *Computer Graphics and Image Processing 19*, 2(June 1982), 129-147.

9. [Oska88] D.N. Oskard, T.-H. Hong, and C.A. Shaffer, Real-time algorithms and data structures for underwater mapping, *SPIE Symposium on Sensor Fusion: Spatial Reasoning and Scene Interpretation*, Cambridge MA, November 1988.

10. [Rudi74] W. Rudin, *Real and Complex Analysis*, McGraw-Hill Book Co., New York, New York, 1974.

11. [Same82] H. Samet, Neighbor finding techniques for images represented by quadtrees, *Computer Graphics and Image Processing 18*, 1(January 1982), 37-57.

12. [Same85] H. Samet and C.A. Shaffer, A model for the analysis of neighbor finding in pointer-based quadtrees, *IEEE Transactions on Pattern Analysis and Machine Intelligence 7*, 6(November 1985), 717-720.

13. [Same86] H. Samet and R.E. Webber, A comparison of the space requirements of multi-dimensional quadtree-based file structures, Computer Science TR-1711, University of Maryland, College Park, MD, September 1986.

14. [Same89] H. Samet, *Applications of Spatial Data Structures; Computer Graphics, Image Processing, and GIS*, Addison-Wesley, Reading MA, 1989.

15. [Same90] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading MA, 1990.

16. [Sant76] L.A. Santalò, *Integral Geometry and Geometric Probability,* Addison-Wesley Publishing Company, Reading, Massachusetts, 1976.

17. [Shaf87a] C.A. Shaffer, H. Samet, and R.C. Nelson, **QUILT**: a geographic information system based on quadtrees, Computer Science TR-1885, University of Maryland, College Park, MD, July 1987.

18. [Shaf87b] C.A. Shaffer and H. Samet, An in-core hierarchical data structure organization for a geographical database, Computer Science TR-1886, University of Maryland, College Park, MD, July 1987.

19. [Shaf91] C.A. Shaffer, Real time robot arm collision detection for telerobotics, to appear in *Journal of Computer & Electrical Engineering.*

20. [Tamm84a] M. Tamminen, Comment on Quad- and Octrees, *Communications of the ACM 27*, 3(March 1984), 248-249.

21. [Tamm84b] M. Tamminen, Encoding pixel trees, *Computer Vision, Graphics, and Image Processing 28*, 1(October 1984), 44-57.

22. [Wals85] T.R. Walsh, On the size of quadtree generalized to d-dimensional binary pictures, *Computers and Mathematics with Applications 11*, 11(1985), 1089-1097.