Hierarchical Coding of Binary Images

YOAV COHEN, MICHAEL S. LANDY, AND M. (MISHA) PAVEL, MEMBER, IEEE

Abstract—Quadtrees are a compact hierarchical method of representation of images. In this paper, we explore a number of hierarchical image representations as applied to binary images, of which quadtrees are a single exemplar. We discuss quadtrees, binary trees, and an adaptive hierarchical method. Extending these methods into the third dimension of time results in several other methods. All of these methods are discussed in terms of time complexity, worst case and average compression of random images, and compression results on binary images derived from natural scenes. The results indicate that quadtrees are the most effective for two-dimensional images, but the adaptive algorithms are more effective for dynamic image sequences.

Index Terms—Binary images, hierarchical coding, image coding, image compression, image processing, quadtrees.

I. Introduction

THE basic data idea behind hierarchical coding is to segment a picture into the largest possible uniform areas and to transmit a hierarchical representation of these areas. The representation takes the form of a rooted tree where the root represents the entire image, and nodes on lower levels represent increasingly smaller subregions of the picture. Leaves of the tree represent uniform regions of the picture, which therefore require no further decomposition. If the particular decomposition of the image used splits an initially square image by halving the vertical and horizontal dimensions, yielding four squares within a square, we arrive at the quadtree image representation. In this particular scheme, each node in the tree is either a leaf or has four children, each of which in turn describes the four quadrants created by splitting the square described by the parent node. The quadtree representation has received a lot of attention in recent years, both for its usefulness in pattern recognition and computer vision schemes, and as a method for image compression [1].

In this paper we discuss the special case of binary images, wherein only one bit per pixel is contained in the original image (black or white). With the goal of maximal errorless compression of binary images, we show that quadtree representation is not necessarily the best method among methods which hierarchically decompose the image. We analyze several hierarchical coding methods both theoretically and by applying each method to several types of binary images. We describe adaptive hierarchical coding methods in which the design of the hierarchy is coupled more closely to the local statistics of the picture. We also include analyses of dynamic images, which are encoded by hierarchical methods that include the third dimension of time in the hierarchical decomposition. We consider six hierarchical coding schemes. The efficiency of these schemes will be compared in terms of their worst case and average performance on random and natural images.

Manuscript received April 9, 1984; revised September 20, 1984. This work was supported in part by the National Science Foundation, Science and Technology to Aid the Handicapped, under Grant PFR-80171189.

Y. Cohen is with the National Institute for Testing and Evaluation, Jerusalem, Israel.

M. S. Landy is with the Human Information Processing Laboratory, Department of Psychology, New York University, New York, NY 10003.

M. Pavel is with the Department of Psychology, Stanford University, Stanford, CA 94305.

II. Overview

The idea of representing images as hierarchies of features or objects is a natural one, and arose independently in several contexts of picture processing. In the area of formal picture languages, Rosenfeld [2] (following Hunter and Steiglitz [3]) suggested quadtree grammers. These grammars generate families of square pictures by application of production rules that replace a node in a tree by a subtree. In the resulting trees, all the nonterminal nodes are of degree four, and stand for square subblocks of the picture.

In recent years the use of quadtrees for picture processing has become quite popular. Quadtrees have been used for hidden line elimination [4], picture segmentation [5], and a host of other picture processing needs [3], [6]-[16]. The field is reviewed in Samet and Rosenfeld [17] and a number of algorithms are given in Pavlidis [1]. Hierarchical schemes are also prevalent in image-understanding systems [18]; in these schemes however, the nodes of the tree represent features of the scenes rather than areas of uniform gray level. For representations of three dimensional objects, oct-tree representations like those described here have been used [19]-[21].

In the area of image compression and transmission, a representation of pictures by binary trees was explored by Knowlton [22]. In his scheme a binary tree which describes the image is transmitted top-down (using a prefix notation). If the image is uniformly black or white, only one symbol is transmitted—the gray level of the image. If not, the transmitter sends a metasymbol which indicates that the picture is not uniform, and that the code that follows represents two subareas of the picture. The two subareas are created by a single line (a "*cut*") that divides the whole picture into two equal parts. The coding proceeds in this fashion recursively until the transmitted code exhausts all the uniform sub- and sub-subareas of the picture. In the worst case, the coding descends down to the level of single pixels which are, of course, uniform.

This recursive method has two advantages.

1) Compression Efficiency: Pictures which are composed of large uniform areas can be highly compressed.

2) Progressive Transmission: For applications in which the needed resolution is variable, the transmission can be termin-

0162-8828/85/0500-0284$01.00 © 1985 IEEE

ated by the receiver as soon as sufficient information about the transmitted picture has been gathered. This idea, which was first suggested by Sloan and Tanimoto [23], is implemented as a scheme in which the tree is sent in top-down order, so that large uniform areas can appear to the receiver first, with finer details progressively filled in as the lower levels of the tree are reached.

The use of quadtrees for image compression has been studied recently by a number of researchers. Most of these studies have concentrated on theoretical results on compression ratios for simple synthetic images. Examples include the work of Dyer [24] and Grosky and Jain [25]. Kawaguchi and Endo [26] were the first to describe the use of quadtrees for compression as described here, and an alternative method of quadtree compression (for image representation rather than coding efficiency) was developed by Gargantini [27]. Types of hierarchical methods used for compression in image databasing include quadtrees [28] and a more general version of binary trees than that presented here [29], [30].

In this report, we examine the effectiveness of hierarchical methods for *compression* of binary pictures obtained from natural scenes. Any hierarchical method can be used for *progressive transmission* if the coded tree is pre-ordered and sent in top-down order. The motivation for this research is for efficient compression schemes, but the new schemes we introduce, like previous hierarchical representations, also have relevance for the representation of dynamic images and as possible data structures for image segmentation and recognition schemes.

III. Hierarchical Representation Schemes

In this section we introduce six hierarchical encoding schemes including quadtree representation, binary trees, and a new scheme: adaptive hierarchical coding. Three-dimensional counterparts to these methods will be explored, where the time dimension is included. These methods include the octtree, three-dimensional adaptive hierarchical coding, and a hybrid scheme called binquad coding.

A. Example

Quadtrees are trees in which every nonterminal node has exactly four children. In addition, the trees have labeled nodes, and the children of a given node are ordered. Any given node in the quadtree represents a square subimage with sides of length 2^n. If this subimage is uniform in gray level, then the node level indicates the gray level, and this node is a terminal leaf of the tree. Otherwise, the label indicates that the subimage is nonuniform, and four subtrees are created describing the four subblocks obtained by splitting the square into quadrants. Fig. 1 illustrates this process.

In quadtree encoding, the tree is transmitted in a specified order (in our case, depth-first). For each node a particular binary code for the node's label is transmitted. For the tree in Fig. 1, the encoding would be, in symbolic form:

G
G *G* *W W*
G *W B W* *W G* *B W*
W W B B *W W W B*

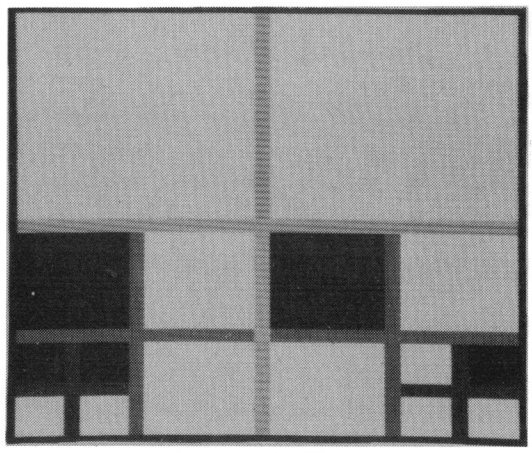


Fig. 1. (a) An image. (b) A quadrant ordering. (c) A quadtree representation of the image. Each node stands for a particular subimage. The top node represents the entire image. If a node has children, then the children represent the four quadrants of the subimage represented by the parent. The node label describes the subimage associated with that node as either black (B), white (W), or nonuniform (G, for gray). Each G node has four children representing the four quadrants of its subimage, where the ordering of the quadrants is as given in (b).

The vertical spacing here is for clarity only (i.e., the tree structure is still discernible); the symbols are transmitted as they appear left-to-right on the page.

B. Quadtree Compression

Quadtree encoding can be implemented by recursive descent in the following manner:

1) Start from the top level (i.e., the area to be considered is the whole $2^n \times 2^n$ picture).

2) Divide the area, if possible, into four subareas.

3) If the subareas are all uniform (of the same gray-level), or if we are dealing with a single pixel, output *B* or *W* and exit.

4) If not, output the symbol for a nonterminal node, here symbolized by "*G*," and then repeat steps 2)-4) for each of the subareas in the prescribed order.

Applying this procedure to the picture of Fig. 1(a) results in the code given in the previous section, which is 21 symbols long over the alphabet $\{G, W, B\}$.

Next, a bit assignment must be selected for the symbols. In this paper we will not discuss the conditions for optimality, and assume the following mapping from symbol to bit string:

$$\{G, W, B\} \rightarrow \{1, 01, 00\}.$$

This decision will be discussed shortly. Applied to the previous example this results in a compression ratio of 37/64 = 0.578 bits per pixel.

Further Improvement in Efficiency: It is assumed that the receiver knows the size of the picture. Hence, when it encounters a "G" that cuts a 2×2 subblock into four single pixels, it knows that the next four symbols describe single pixels, and that no further cuts are possible. Thus, a block of four pixels can be represented using four bits instead of eight.

Another source of redundancy derives from the fact that the receiver knows that if a subblock was cut, then the subblock must be nonuniform. If the first three quadrants are uniform and of the same gray level, then the fourth must not be uniform and of the same gray level. If the following partial code

$$G\ W\ W\ W$$

for a block of size $2^n \times 2^n$ is received, then if $n = 1$, the receiver concludes that the next symbol is B, and therefore the code for B can be omitted. If $n > 1$, the receiver concludes that the next symbol is either G or B and therefore a single bit suffices for distinguishing between the two symbols.

Our example results in the following code:

$$\begin{array}{l} G1 \\ \quad G1 \qquad\qquad\qquad\qquad G1 \qquad\qquad\qquad\qquad W2\ W2 \\ \quad\quad G1 \qquad\qquad W2\ B2\ W2 \quad W2\ G1 \qquad\qquad B2\ W2 \\ \quad\quad\quad W1\ W1\ B1\ B1 \qquad\qquad W1\ W1\ W1\ \mathbf{B0} \end{array}$$

The number of bits used to represent each symbol is given. The savings on uniform codes at the single pixel level can be seen at the bottom level. The code printed in boldface is the one case where the second savings is applicable. The three W codes *must* be followed by a B code, and therefore no bits are used to represent that B. The code length is 28 bits, yielding a rate of 28/64 = 0.436 bits per pixel.

The choice of bit mapping above is partially justified by the redundancy savings we employed. The only case in which the full bit mapping is used is for nodes where the full alphabet is possible, which is for nodes which describe regions larger than a single pixel and where the second savings does not apply. Our choice of bit mapping is justified if the number of G nodes is greater than either the number of B or W nodes among this set, which is pretty much guaranteed for the methods (described below) which result in binary trees. For quadtrees and other nonbinary tree methods, the optimal choice depends in a complicated way upon the image statistics. For images where the number of black and white pixels are relatively comparable, our choice is probably correct. On the other hand, for the images described in Section IV-D, a choice favoring the more frequent pixel may have given slightly better results.

Time Complexity of the Algorithm: Consider the size of a quadtree. Clearly, in a full quadtree which represents every single pixel with $2^n \times 2^n$ terminal nodes there are

$$N_{QT} = \sum_{i=0}^{n} 2^i \times 2^i = \frac{(4^{n+1} - 1)}{3} \tag{1}$$

terminal and nonterminal nodes altogether. An efficient algorithm visits each node once, including single pixels, and executes a constant number of operations for each, regardless of its level. Such an algorithm uses a bottom-up traversal of the decision tree. At each level, uniformity is determined based on the previously computed colors of the next lower level's nodes. By (1), the time complexity is linear in the number of pixels (which is $2^n \times 2^n$ or 4^n).

C. Binary Tree Coding

The binary tree scheme is similar to the quadtree algorithm, but utilizes a binary tree structure to represent the picture. The first *cut* of the square picture results in two rectangular subblocks of equal size. In order to keep the aspect ratio of the subblocks close to one, the next cut is in a direction perpendicular to that of the preceding cut. The transmitter and the receiver of the code must agree as to which cut direction *dominates,* which defines the order of the cuts. Given a block size of $2^n \times 2^m$, if n is not equal to m the cut is across the longer dimension, but when $n = m$, the cut is taken along the dominant direction. The binary encoding of a binary tree representation is quite similar to that used in quadtree coding, including the mapping from symbols to bits, the elimination of redundancy, and the linear time complexity.

D. Adaptive Hierarchical Coding

In quadtree representation, any uniform region in the picture which is not square is decomposed into smaller square areas. Thus, the compression efficiency of this method is relatively poor for pictures which are dominated by elongated regions. A new method, *adaptive hierarchical coding* (AHC), overcomes this disadvantage. In AHC, given a nonuniform region of the picture, the decision whether to cut this region horizontally or vertically is made dynamically by choosing the direction which minimizes the code length.

In terms of number of symbols, the code that results from AHC cannot be longer than that which is generated by BT coding. But for AHC, four different symbols are needed instead of three; the direction of the cuts must be specified, since it is no longer predetermined. The code utilizes two symbols to represent nonuniform areas (H for a horizontal cut and V for vertical cut), and two symbols to represent the colors of uniform areas (B and W). The method of mapping symbols to bit strings, and the elimination of redundancy in the code is somewhat more complicated in AHC than in QT or BT, but the general idea is the same. The alphabet $\{B, W, V, H\}$ is

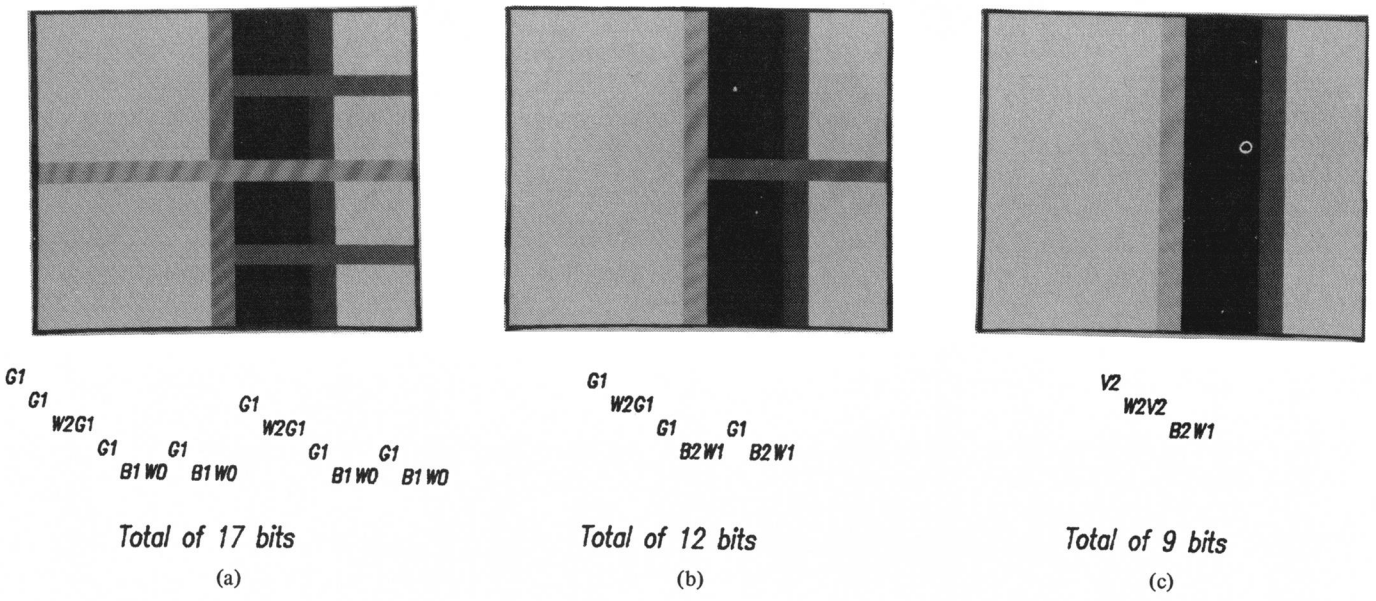


Fig. 2. A simple image processed by three hierarchical coding schemes. (a) Binary tree coding with horizontal dominance. (b) Binary tree coding with vertical dominance. (c) Adaptive hierarchical coding. The cuts made by each method are shown in gray. Below each figure the symbolic representation of the tree is given along with the number of bits needed for each symbol.

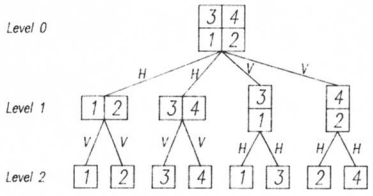


Fig. 3. The complete decision tree for a 2 × 2 image used by the adaptive hierarchical coding method. This method requires that all possible combinations of vertical and horizontal cuts be examined.

initially represented with two bits each. Savings arise when the potential alphabet is smaller, either because fewer cuts are possible (for $m \times 1$, $1 \times m$, and 1×1 subimages), or fewer colors are possible (for the last child when the first is uniform).

The differences between binary tree coding and AHC are illustrated in Fig. 2. The example emphasizes the fact that AHC can result in a code which is shorter than the best BT code, in this example both in the number of symbols and the number of bits which are required to encode the symbols.

Complexity of AHC: AHC can be implemented by a recursive descent algorithm. This implementation, however, leads to unacceptable time complexity, because many images are examined more than once in this procedure. Consider Fig. 3, where an AHC decision tree for a 2×2 image is drawn in detail. Note that on level 2 each pixel of the picture is examined twice. To avoid the repetitious computation, the algorithm is modified so that the code length for each possible subarea is computed only once. The question of complexity then becomes equivalent to determining the number of subareas to be examined: i.e., how many rectangles with sides which are powers of 2, and which are offset from the picture boundary by an integer multiple of their size, are contained in a $2^n \times 2^n$ picture?

Clearly, there are $(n+1)(n+1)$ different sizes of such rectangles, ranging from the size of a single pixel to the size of the entire picture. For a given size, $2^i \times 2^j$, there are $(2^n/2^i)(2^n/2^j)$ different rectangles. The total number of rectangles is therefore

$$R = \sum_{i=0}^{n} \sum_{j=0}^{n} \frac{2^n}{2^i} \times \frac{2^n}{2^j} = 2^{2n} \sum_{i=0}^{n} 2^{-i} \sum_{j=0}^{n} 2^{-j} \tag{2}$$

and since

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

(2) can be rewritten as

$$R = 2^{2n}(2 - 2^{-n})^2. \tag{3}$$

R is bounded by 4×2^{2n}, and the computational complexity of AHC is thus *linear in the number of pixels.* By starting the computation from the smallest areas and proceeding in the correct order, one can assure that the computation for a given block will not require more than four look-up operations and a constant number of comparisons and additions.

E. Hierarchical Coding in Three Dimensions

The hierarchical coding methods that were described thus far are easily extended from two into three dimensions, from the coding of single images to that of dynamic images sequences in which two dimensions are spatial and the third is temporal. The analog to quadtrees in two dimensions is an oct-tree in three dimensions (denoted OT). (Note that the use of the term oct-tree here differs from the octrees of Meagher [20], where the third dimension is spatial depth.) On each level of

an oct-tree, if the three dimensional subblock is not uniform it is subdivided further into eight sub-subblocks. Otherwise, oct-tree coding is similar to QT in all respects. AHC trees can also be extended to three dimensions by adding a symbol, T, for the case in which a subdivision is made in the temporal dimension.

We also consider a compromise between fully adaptive coding in three dimensions and oct-trees. This method, called *binquad encoding* (denoted BQ), is a hybrid between quadtrees in the spatial dimension and adaptive coding otherwise. At each point a decision is made whether to cut the image spatially or temporally. A temporal cut is binary, whereas a spatial cut is quaternary, and the order of cuts is chosen so as to minimize the code length. This tree labeling involves an alphabet of four symbols: B, W, S (spatial cut), and T (temporal cut).

There are similar issues with the three-dimensional methods as with the previous methods with respect to actual binary coding. Again, redundancy is eliminated by considering the potential alphabet at a given point given the subimage size (which indicates the possible cuts) and the color of the previous children of the same parent (which indicates the potential uniform colors). The only question is the priority for making various letters have shorter codes. We always use the shorter codes for symbols indicating cuts for the reasons indicated previously. All of the three-dimensional methods have the same computational complexity—they are all linear in the number of pixels. However, they differ in the number of operations; in our implementation, QT, BT, and OT are roughly comparable, BQ takes three times as long, AHC takes five times as long, and three-dimensional AHC takes nearly 14 times as long.

Under what conditions can we expect superior compression from three-dimensional methods as compared with two-dimensional methods? Hierarchical coding methods are particularly suitable for pictures in which boundaries are parallel to the borders of the picture. A picture of an upright rectangle is coded more efficiently than a picture in which the same rectangle is tilted. In practice, however, pictures with this attribute are not easy to come by (except, perhaps, in document transmission applications, where print lines and graphs are parallel to the boundaries of the document). But in three-dimensional sequences, the boundaries of objects which are stationary relative to the camera are always parallel to the *temporal* axis. Thus, three-dimensional hierarchical coding of sequences might benefit from this naturally available parallelism. This observation was the impetus for the development of the BQ and three-dimensional AHC methods.

F. Generalizations of Hierarchical Coding Schemes

Up until now, we have discussed the encoding of an image as a single rooted tree, representing the entire image or image sequence. This can easily be extended to a representation in terms of a forest of rooted trees which are sent in a known order. This is equivalent to having the receiver and transmitter agree in advance upon a known number of cuts, and therefore not sending the top nodes of the tree representing those cuts. For large images, the size of the decision tree can get to be quite large, as are the space requirements for the method, and so one might pragmatically decide to limit the regions upon which the method is used. This will result in slightly longer codes if huge uniform regions exist, but otherwise results in shorter codes, as the top of the (virtual) tree is not transmitted.

Thus, we have implemented the various methods with a limitation on the spatial extent to which the method is applied. It is assumed that the receiver and transmitter both know this coding parameter. For oct-trees, the limited spatial extent is a cube whose sides evenly divide the total image dimensions. For three-dimensional AHC, the limitation for the various dimensions are effectively independent, and need not be equal. For binquad, the spatial extents must be equal. With this generalization, two-dimensional AHC is merely a subcase of three-dimensional AHC (with temporal extent of one), and quadtrees are a subcase of binquad (similarly).

Lastly, the restriction we have been observing to powers of two and square images (for quadtrees and binquad) is not necessary. As long as the receiver and transmitter agree on how a single cut works (especially in the situation of cutting an odd number of pixels), then any shape will do, and the procedure still recurses until single pixels are reached. Powers of 2 are just convenient, and probably yield better compression ratios for that reason. Also, note that single cuts in a given dimension need not be the rule. For example, one might envision a "ternary-tree" method, which at any given stage makes a double cut, splitting that dimension into three pieces, and yielding three sons.

As a means of summarizing the coding methods, Fig. 4 gives the hierarchy output by the various techniques as applied to a single edge-detected image.

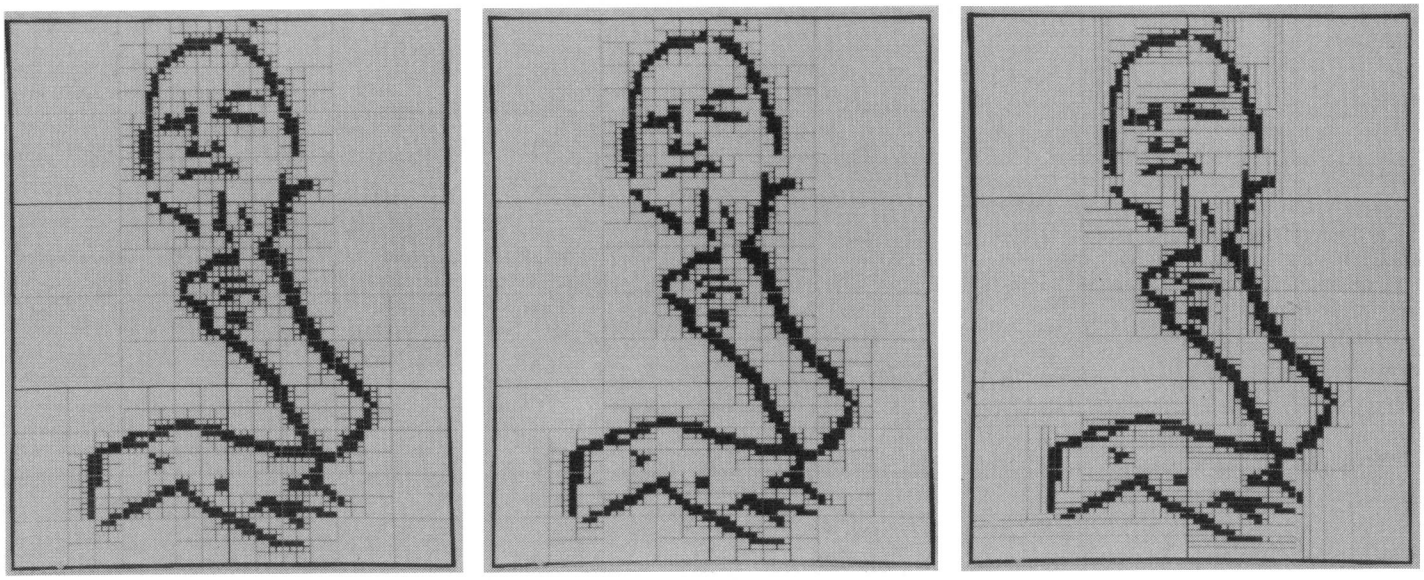
IV. Evaluation

To evaluate the various coding methods, the algorithms were applied to a number of images. Their efficiencies are compared in terms of the number of bits per pixel which can then be compared with the information content (e.g., standard entropy measures) of the original picture(s).

A. Worst Case Analysis

We will now analyze the worst case for each of the methods. That is, we will determine the longest code length which might be generated for a given image size. For all six methods (QT, BT, AHC, OT, three-dimensional AHC, and BQ), the worst case analysis follows the same outline. A full encoding tree, with total branching until every single pixel has a corresponding node in the lowest level of the tree, always gives the longest code. For all methods, a checkerboard with black and white alternating in all directions (including the temporal, for three-dimensional methods) is an example of a worst case image.

The reason that a full tree yields the worst case code is easy to prove. Given a full encoding tree, the only way to create a tree with less nodes is via a merge operation, where the sons of a gray node are eliminated and the parent node becomes uniform. In all of our methods, a merge can never result in a longer code than the unmerged version. In some cases, how-



(a) (b) (c)

Fig. 4. Summary of the two-dimensional methods. This figure depicts the cuts made by three hierarchical coding schemes. (a) Quadtree coding. (b) Binary tree coding with horizontal dominance. (c) Adaptive hierarchical coding. The cuts appear as gray lines in the otherwise binary image.

ever, it can result in a code with the same length. We now examine each method in turn.

Quadtrees: The number of nodes in a full quadtree for an image of size $2^n \times 2^n$ was derived above in order to prove that the time complexity of the algorithm was linear. We repeat that result here

$$N_{QT} = \sum_{i=0}^{n} 2^i \times 2^i = \frac{(4^{n+1} - 1)}{3}. \tag{4}$$

In the worst case, every node in the tree requires 1 bit. Higher level nodes are all gray, and therefore require 1 bit each. The lowest level nodes also require 1 bit each. They require 1 bit instead of 2 because they are lowest level nodes (single pixels), but we restrict ourselves to images in which no other savings are possible in order to reach the worst case. Thus, (4) also gives the number of bits in the worst case image encoding

$$B_{QT} = \sum_{i=0}^{n} 2^i \times 2^i = \frac{(4^{n+1} - 1)}{3}. \tag{5}$$

Dividing by the number of pixels in the image yields a compression ratio slightly less than $\frac{4}{3}$ bits per pixel.

Binary Trees: The full binary encoding tree is precisely a full binary tree of $2n + 1$ levels, and thus the number of nodes is given by

$$N_{BT} = \sum_{i=0}^{2n} 2^i = 2^{2n+1} - 1. \tag{6}$$

In such a tree, every node is represented by 1 bit, except that half of the bottom nodes are not represented at all. Thus, the number of bits in the encoding is

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

The bit rate is slightly less than $\frac{3}{2}$.

Adaptive Hierarchical Coding: For the adaptive method, the same logic applies as in the binary coding method. The full tree is a full binary tree just as in BT, and the number of nodes is as given in (6). For this tree, the adaptive scheme chooses the order of cuts which yields the minimum code. Because of the saving of bits when all cuts in one direction are exhausted, the order chosen will be to do all cuts in one direction first, then all cuts in the other direction. Thus, the top n levels will require 2 bits per node, the next n levels will require 1 bit per node, and the bottom level will require 1 bit per two nodes. The number of bits will be

$$\begin{aligned} B_{\mathrm{AHC}} &= 2 \times (2^n - 1) + 1 \times 2^n \times (2^n - 1) + \tfrac{1}{2} \times (2^{2n}) \\ &= \tfrac{3}{2} \times 2^{2n} + 2^n - 2. \end{aligned} \tag{8}$$

For large n this will yield a bit rate slightly over $\frac{3}{2}$.

Oct-Trees: The analysis of the full tree for the oct-tree method is analogous to that for quadtrees, and so we merely state the answer here. The number of nodes in the worst case tree is

$$N_{\mathrm{OT}} = \sum_{i=0}^{n} 8^i = \frac{8^{n+1} - 1}{7}. \tag{9}$$

There is 1 bit per node, so the same expression yields the number of bits. The worst case bit rate is $\frac{8}{7}$ bits per pixel.

Three-Dimensional AHC: The analysis of three-dimensional AHC is analogous to two-dimensional AHC. The number of nodes in a full tree is given by

$$N_{\text{3D-AHC}} = \sum_{i=0}^{3n} 2^i = 2^{3n+1} - 1. \tag{10}$$

The algorithm will choose the cuts so as to minimize the code length. Thus, it will perform the cuts in dominance order (using the minimum code length cut at each point, and reducing the number of possible cuts as quickly as possible). There will be n T cuts, followed by n V cuts, followed by n H cuts. The T and V nodes will then require 2 bits each, and the H cuts will require a single bit. Only half of the single pixels need be coded. This results in the following bit count

$$B_{\text{3D-AHC}} = 2 \times (2^{2n} - 1) + 1 \times 2^{2n} \times (2^n - 1) + \tfrac{1}{2} 2^{3n}$$
$$= \tfrac{3}{2} 2^{3n} + 2^{2n} - 2. \tag{11}$$

For large n this yields a bit rate slightly higher than $\frac{3}{2}$.

Binquad: The binquad method is a little more difficult to analyze because of the possibility of both binary and quaternary cuts. Even the number of nodes in the tree depends on the order in which the cuts are made. Since the method chooses the tree which minimizes code length, we must determine the form of that tree. This problem is not necessarily solved, in general, by a single approach. Therefore we have examined three cases.

1) All binary cuts are made first, then all quad cuts.

2) All quad cuts are made first, then all binary (temporal) cuts.

3) All but one binary cuts are made first, then all quad cuts, and lastly the final binary cut is made.

The rationale for examining these cases is as follows. In the first case, we make the quaternary cuts lower in the tree, which keeps the highest branching part of the tree lower, and minimizes the number of nodes in the tree (i.e., the number of symbols to be coded). In the second case, the number of symbols will be higher, but on the bottom level of the tree (single pixels), we will save half the bits by the saving of redundancy rule, which does not apply for the final quaternary cut of case 1). In the third case we combine the good points of the first two cases, but we do not save on bits on any cuts before the last, because we never exhaust either type of cut until then.

Working out the three cases in detail, it turns out that the third possibility is the most efficient scheme, and is the scheme chosen by our algorithm for any size image. In this scheme, all but one of the binary cuts are made first, and require 2 bits each. Then the quaternary cuts are made, and each requires 2 bits. Then the last binary cut is made, requiring a single bit. Finally, the single pixels are coded, requiring 1 bit for each pair. This yields

$$N_{\text{BQ}} = (2^{n-1} - 1) + 2^{n-1} \times \frac{4^n - 1}{3} + 2^{n-1} \times 4^n + 2^n \times 4^n$$
$$= \frac{5}{3} 2^{3n} + \frac{1}{3} 2^n - 1$$
$$B_{\text{BQ}} = 2 \times (2^{n-1} - 1) + 2 \times 2^{n-1} \times \frac{4^n - 1}{3} \tag{12}$$
$$+ 1 \times 2^{n-1} \times 4^n + \frac{1}{2} 2^n \times 4^n$$
$$= \frac{4}{3} 2^{3n} + \frac{2}{3} 2^n - 2.$$

TABLE I
WORST CASE COMPRESSION RATIOS

N	2	3	4	5	6	7	8	theoretical bound as $N \to \infty$
QT	1.313	1.328	1.332	1.333	1.333	1.333	1.333	1.333
BT	1.438	1.484	1.496	1.499	1.500	1.500	1.500	1.500
AHC	1.625	1.594	1.555	1.529	1.515	1.507	1.504	1.500
OT	1.141	1.143	1.143	1.143	1.143	1.143	1.143	1.143
3D-AHC	1.719	1.621	1.562	1.531	?	?	?	1.500
BQ	1.344	1.340	1.335	1.334	?	?	?	1.333

The question marks represent compression ratios which our programs were unable to compute for lack of storage.

Thus, for binquad, the worst case compression ratio is slightly higher than $\frac{4}{3}$.

Table I gives the actual compression ratios for checkerboards as computed by our programs, and the theoretical limits we have just derived. In examining the code output by binquad, one can see that case 3) is always used, as predicted.

The reader may have noted something curious in these analyses. To wit, what is the point of using a hierarchical scheme in the worst case if that worst case is more than 1 bit per pixel, since the original image only required 1 bit per pixel to start with? There are two answers to this point. First, the algorithms can easily be modified so as to lower the worst case to 1 bit per pixel. Second, the relative worst case of the unmodified algorithms is of interest in its predictions of the operation of the various methods on nonworst case images. We discuss these two points in turn.

One way to modify these algorithms so as to lower their worst case values without damaging their efficiency involves sending the image as a forest rather than a single tree by agreeing upon an initial decomposition of the image. Thus, the transmitter precedes the coded forest with an indication of the *level* of the forest that follows, or the shape of the blocks which are to be coded. If this shape is only single pixels, then the image is simply coded as an array of pixels. But this scheme allows one to adapt the level of hierarchical description to the image characteristics. In the QT scheme, the initial indicator need only give the level in the full tree at which the forest is, which for a $2^n \times 2^n$ image is a number from 0 to n. In the adaptive schemes, the most general scheme would allow the transmitter to choose the most efficient block size and include a description of that shape in the transmission.

The worst case analyses are also useful indications of the efficiency of the various methods despite the possibility of their being circumvented by modifying the algorithms. In the next section we will be comparing these methods as applied to random images where the probability of one of the two colors is varied. As the two colors become equiprobable, the random images approach the worst case image, and the relative efficiency of the methods approaches their relative worst case efficiency. The worst case as we have derived it becomes a predictor for the effectiveness of the methods on some nonworst case images.

B. Expected Code Length for Random Images

Quadtrees: Consider a square picture which is generated by a random independent pixel process. In particular, let p be the probability of a black pixel. For quadtrees, the expected code-

TABLE II
EXPECTED COMPRESSION RATIO FOR QT

	p	.50	.75	.90	.95	.99	.995	.999	.9995
	H(p)	1.000	.811	.469	.286	.081	.045	.011	.006
n	dimensions								
1	2	1.125	.980	.739	.628	.527	.514	.503	.501
2	4	1.187	1.039	.721	.498	.217	.172	.135	.130
3	8	1.203	1.054	.736	.510	.175	.109	.048	.040
4	16	1.207	1.058	.740	.514	.177	.105	.030	.019
5	32	1.208	1.059	.741	.515	.178	.106	.029	.016
6	64	1.208	1.059	.741	.515	.178	.106	.029	.016

TABLE III
EXPECTED COMPRESSION RATIO FOR BT

	p	.50	.75	.90	.95	.99	.995	.999	.9995
	H(p)	1.000	.811	.469	.286	.081	.045	.011	.006
n	dimensions								
1	2	1.125	.992	.754	.638	.530	.515	.503	.501
2	4	1.309	1.140	.745	.494	.211	.169	.134	.129
3	8	1.355	1.187	.789	.521	.166	.102	.046	.039
4	16	1.367	1.199	.800	.533	.170	.098	.028	.018
5	32	1.370	1.201	.803	.536	.173	.101	.026	.015
6	64	1.371	1.202	.804	.536	.174	.101	.027	.015
7	128	1.371	1.202	.804	.537	.174	.102	.027	.015

length in bits of a $2^n \times 2^n$ image, $\bar{L}(n)$, is given by the following recursion:

$$\bar{L}(n) = 4\bar{L}(n-1) + 1 - 7(p^{4^n} + q^{4^n}) - p^{3\times 4^{n-1}} q^{4^{n-1}} - p^{4^{n-1}} q^{3\times 4^{n-1}} \qquad (13)$$
$$\bar{L}(1) = 5 - (p^3 q + pq^3) - 3(p^4 + q^4)$$

where $q = 1 - p$, the probability of a white pixel (see Appendix A.1 for derivation). Table II shows the compression ratio for pictures of several sizes parameterized by the probability of black (white) pixels. For comparison, the entropies per pixel, denoted by $H(p)$, which are associated with different values of p, are listed in the first row of the table.

As indicated by the table, QT coding is inefficient for compression of independent pixel-process pictures, which is to be expected given that it is a method which takes advantage of correlation between neighboring pixels. However, the method is well behaved in the sense that its efficiency increases (the compression ratio decreases) as the entropy decrease. It is also evident from Table II that for each value of p there is an optimal picture size. For a particular p, large pictures are more efficiently transmitted as a forest instead of as a single tree. The exact level at which the picture should be divided into subpictures depends on p.

Binary Trees: By a derivation similar to that for quadtrees (see Appendix A.2), the expected code length for the binary tree representing a random picture of 2^n pixels is

$$\bar{L}(X_n) = 2\bar{L}(X_{n-1}) + 1 - 2(p^{2^n} + q^{2^n}) - (p^{2^{n-1}} + q^{2^{n-1}})^2$$
$$\bar{L}(X_1) = 2. \qquad (14)$$

Table III lists the expected code length in bits per pixel for pictures of different sizes, for several values of p. BT compression is more efficient than QT for pictures with p's greater than 0.99.

AHC: Because of its adaptive nature, an analysis of the expected code length produced by AHC on random pictures is difficult. Therefore, we investigated this method using a Monte Carlo procedure. Three types of images were used in order to compare the efficiency of AHC with that of BT and QT coding.

TABLE IV
COMPRESSION EFFICIENCY ON 256 × 256 PICTURES

p	.501	.749	.900	.950	.990	.995	.999	.9995
method								
BT	1.370	1.203	0.805	0.539	0.175	0.101	0.028	0.015
AHC	1.374	1.202	0.798	0.533	0.175	0.102	0.028	0.016
QT	1.208	1.060	0.742	0.517	0.180	0.106	0.030	0.016

(a)

p	.501	.749	.900	.950	.990	.995	.999	.9995
method								
BT	.0020	.0025	.0049	.0082	.0052	.0047	.0034	.0025
AHC	.0017	.0024	.0046	.0072	.0052	.0051	.0035	.0024
QT	.0026	.0020	.0030	.0074	.0053	.0048	.0037	.0027

(b)

(a) Mean compression-efficiency. (b) Standard deviations.

TABLE V
COMPRESSION EFFICIENCY ON UNIFORM RECTANGLES

size	mean efficiency BT	AHC	QT	number of such rectangles
4X4	.823	.782	.848	100
8X8	.480	.394	.496	1296
16X16	.259	.181	.269	18496
	Proportion of cases in which the method is superior to others.			
4X4	.478	.384	.138	100
8X8	.276	.658	.066	1296
16X16	.132	.849	.019	18496

First, the three algorithms were tested on randomly generated pictures of size 256 × 256. The pictures were generated by an independent pixel process with different p's (where p is the probability of the more probable color). The results are listed in Table IV. Note that the p values reported in the table are the *observed* proportions which were obtained in the generated pictures and not the parameters that were used to generate the pictures. For each p, ten pictures were generated which were then compressed by the three algorithms. As can be seen in Table IV, in no case is there an advantage to the adaptive method over the other methods. This is understandable because in the absence of any dependency between neighboring pixels, AHC comes out the worse because of the extra symbol in its alphabet.

More promising results were obtained with highly structured pictures. In our second test of AHC coding we followed the method of Dyer [24]. For a given picture size (4 × 4, 8 × 8, 16 × 16), we generated all of the possible upright rectangles that would fit anywhere in a picture of that size. Thus, for a given size of $n \times n$, $(n^2 + n)^2/4$ pictures were generated and then compressed by the three algorithms (the expression for the number of different rectangles is computed by summing over all vertical and horizontal dimensions, and all displacements for a given rectangle). As shown in Table V, the results indicate a marked superiority of AHC coding over BT and QT coding, an advantage which increases with picture size.

The superiority of AHC and BT over the QT method in the last set of data is probably due to the fact that the majority of the synthetic pictures contain elongated rectangles. This observation leads to the expectation that for random pictures in which elongated rectangles dominate, the AHC method will be superior. In our third experiment, we tested this theory by

TABLE VI
COMPRESSION EFFICIENCY ON PICTURES OF RANDOM ELONGATED RECTANGLES

p	.5	.75	.9	.95	.99	.995	.999	.9995
QT	1.211	1.155	.987	.777	.311	.182	.052	.030
AHC	1.376	1.286	.961	.697	.257	.155	.045	.026
Standard Deviations (n=10 for each entry)								
QT	.003	.006	.010	.012	.018	.011	.005	.006
AHC	.004	.005	.009	.011	.015	.009	.004	.005

TABLE VII
COMPRESSION EFFICIENCY ON THREE-DIMENSIONAL SEQUENCES OF RANDOM PICTURES

p	.5	.75	.9	.95	.99	.995	.999	.9995
QT	1.208	1.056	.739	.515	.178	.107	.029	.015
OCT	1.136	1.053	.764	.553	.212	.132	.037	.019
AHC	1.387	1.206	.797	.533	.178	.107	.030	.015
3D-AHC	1.389	1.205	.790	.526	.174	.104	.028	.015
Standard Deviations (n=10 for each entry)								
QT	.002	.004	.005	.011	.011	.005	.005	.003
OCT	.002	.005	.005	.009	.010	.007	.006	.004
AHC	.002	.005	.006	.009	.010	.006	.006	.003
3D-AHC	.002	.004	.005	.010	.010	.006	.005	.003

applying the methods to a new set of random images. The pictures were of size 128 × 128, and were generated by exclusive bitwise ORing of six pictures. In one picture the rectangles were of size 32 × 4, in the second the rectangles were of size 16 × 2, and so on with sizes: 8 × 1, 4 × 1, 2 × 1, and 1 × 1. In each picture, the probability of a pixel to be covered by a rectangle was fixed at p (the same for all the six pictures). Table VI contains the coding efficiency for such random pictures.

As can be seen from Table VI, the AHC method is superior to the QT method by up to 17 percent, for p's of 0.9995 and down to 0.9. The only cases in which QT is superior to AHC are those for which the compression ratio is greater than 1, in which case the use of these compression methods is not justified to begin with.

Three-Dimensional Methods: A comparison was made between the compression efficiency of the two-dimensional and three-dimensional methods by applying the methods to sequences of random pictures (32 frames, each of 32 × 32 pixels). The results are presented in Table VII. As can be seen from the table, on this type of material the three-dimensional coding methods are not always more efficient. In fact, the QT method is more efficient than other methods for p values of 0.90 and 0.95.

C. "Natural" Images

Our first experimental results which are reported below are based on two sets of pictorial material. The first set comprises 32 frames which are part of a sequence of binary pictures which depict a signer of American Sign Language (ASL), signing the word *tomato*. This image sequence comes from our research on extremely low bandwidth transmission of American Sign Language [31]-[33]. The pictures were transformed using an edge-detection method based on the techniques developed by Marr and Hildreth [34], yielding a line-drawing-like image (see also [33] for a fuller description). The size of the picture is 128 × 128 binary pixels, and resembles the image shown in Fig. 5(e). The second set is a single 512 × 512 binary picture which is a digitized and thresholded image of a portion of a printed page (about 10 percent of the area of a full page). The results are given in Table VIII.

TABLE VIII
COMPRESSION RATIOS FOR THE TWO DATA SETS

set	A	B
QT	.078	.205
H(1 X 1)	.108	.596
H(2 X 2)	.083	.303
H(4 X 4)	.061	.178
QT + Huffman code	.076	.201
Huffman code for 2 X 2	.281	.379
Huffman code for 1 X 8	.165	.285
BT	.081	.217
AHC	.080	.198

The quadtree compression algorithm was applied to the two sets of data just described. The compression ratios of the two data sets and the entropies associated with these sets are listed in Table VIII. For both data sets, quadtree encoding is superior to the lower-bound set by the entropies of 2 × 2 blocks (where the picture is divided into nonoverlapping 2 × 2 blocks and entropy is calculated over the alphabet of 16 possible 2 × 2 blocks). In fact, it is not too far from the entropies of 4 × 4 blocks. The results fare even better when compared with the efficiency of Huffman coding [35] on various block sizes.

Table VIII also gives the results for a modified form of quadtree compression. The modification consists of using Huffman coding for the code of the lowest level in the quadtrees (the 2 × 2 blocks of pixels). This modification should improve the efficiency to the extent that some of the overall redundancy of the picture is preserved in the 2 × 2 nonuniform blocks. In practice, this modification increases the efficiency by a further 0.2-0.4 percent, as indicated in Table VIII. The BT-compression method was applied to our data sets, and as can be seen in Table VIII, resulted in slightly poorer compression compared with QT compression. Our expectation was that AHC would be most useful for more structured pictures. The results in Table VIII suggest that for highly structured pictures (e.g., documents), there is a slight advantage to the AHC coding over QT compression.

D. Cartoon Images–Statistical Analysis

In this section we apply the hierarchical coding methods to a wider body of images. All too often image processing methods are discussed in the context of one or two sample images. In our opinion, this is poor scientific practice, allowing the researcher to present only the "best" result, and insufficiently predicting the behavior of an algorithm in a real situation. Therefore, we have applied the various hierarchical methods to a large number of images in order to gain some knowledge of the statistical behavior of the methods' compression characteristics.

As we have already mentioned, this laboratory is involved in research concerning low bandwidth transmission of visual information, notably images of people speaking American Sign Language, which is a manual form of communication used with and among the deaf and hard of hearing [31]-[33]. In carrying out this research, we have created a library of image sequences of various isolated signs (the *words* of ASL), and have transformed them in a number of ways. A number of these transformations yield binary images which resemble car-

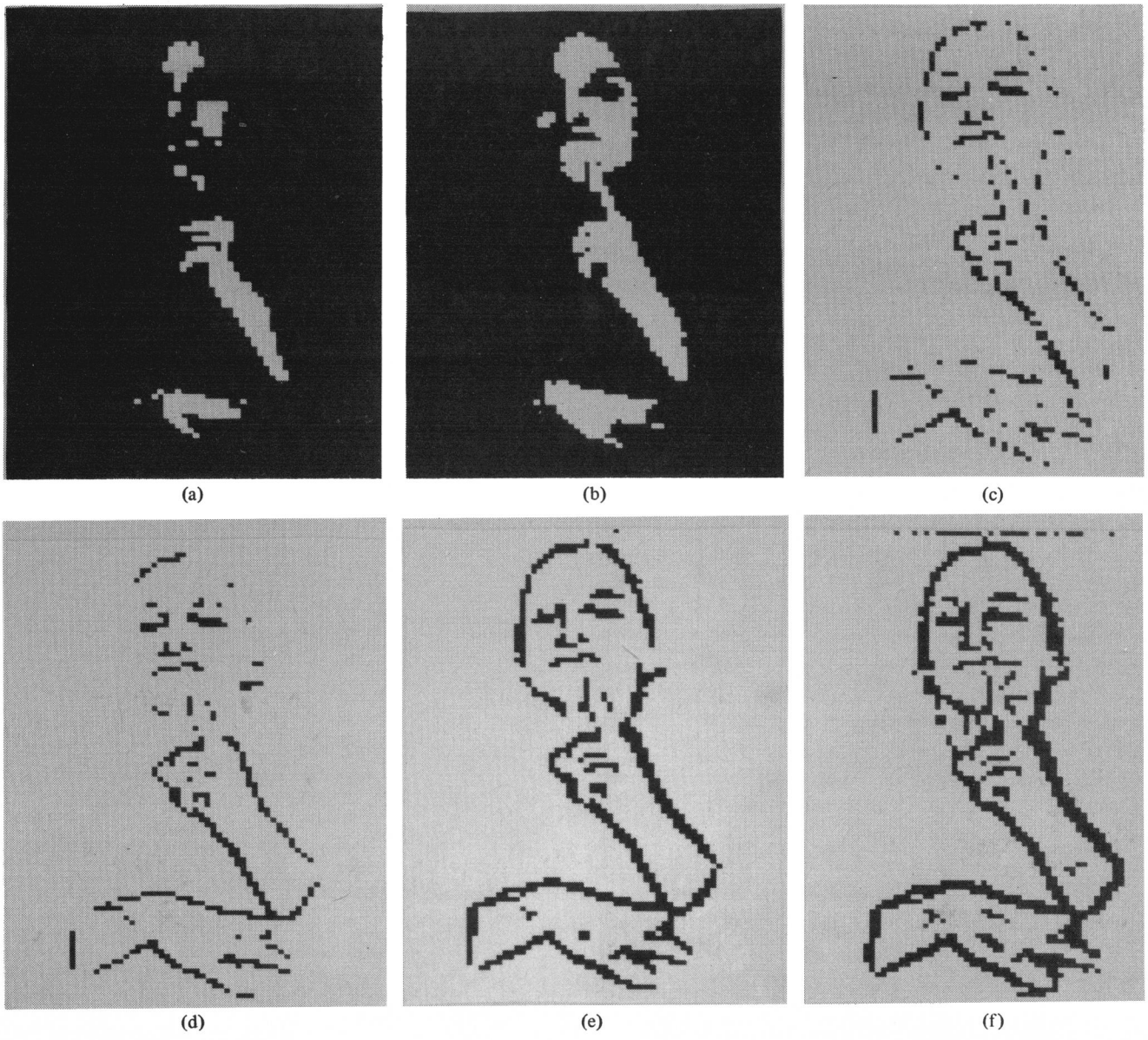


Fig. 5. An example of an image processed by the six transformations we have used. These image transformations include the following. (a) Thresholding the gray-scale image so that 5 percent of the pixels remain above threshold ($t5$). (b) The same procedure with a 10 percent threshold ($t10$). The other four images start from an edge-enhanced image resulting from a convolution of the gray-scale image with a difference-of-Gaussian filter (or *dog* function). (c) The zero crossings in the edge-enhanced image are computed and thresholded based on their slope so that 5 percent of the pixels remain above threshold ($z5$). The last three images involve thresholding the negative-going peaks in the edge-enhanced image, where the thresholding is set so as to control the number of pixels below the threshold. The thresholds are: (d) 5 percent ($d5$), (e) 10 percent ($d10$), and (f) 15 percent ($d15$).

toons or line drawings. It was this body of images which led to our research in compression of binary pictures.

For this study, we have used a set of 84 single signs. These were digitized and all subsequent processing of the images utilized the image processing system HIPS [36], [37]. The images are cropped and subsampled in time and space resulting in images of 96 X 64 8-bit pixels, and sequence lengths of 15 to 45 pictures (1 to 3 s at 15 frames/s). Six transformations will be examined in this section. Examples of the end result of these transformations for the same initial picture are given in Fig. 5. Two of the transformations involve a simple absolute threshold applied to the image. These thresholds were set so as to have 5 percent ($t5$) or 10 percent ($t10$) of the pixels remain above the threshold.

The remaining transformations involve edge detection. All four of these schemes begin by applying a difference-of-Gaus-

TABLE IX
COMPRESSION EFFICIENCY ON THREE-DIMENSIONAL SEQUENCES OF CARTOON IMAGERY

	Mean Compression Ratios (n=84 for each entry)									
	Method									
Transf.	RLE	BT	AHC	QT	BQ2	OT	BQ4	BQ8	3D-AHC	BQ16
d15	.553	.421	.389	.388	.365	.354	.298	.273	.272	.266
d10	.460	.333	.317	.312	.29	.282	.236	.216	.214	.211
z5	.400	.288	.279	.273	.257	.249	.22	.205	.206	.201
d5	.338	.25	.237	.237	.215	.206	.174	.159	.155	.155
t10	.335	.166	.164	.162	.142	.145	.114	.104	.102	.102
t5	.303	.124	.123	.123	.108	.11	.089	.082	.081	.081
	Standard Deviations									
d15	.038	.02	.026	.018	.017	.025	.019	.023	.03	.027
d10	.030	.014	.019	.012	.011	.019	.014	.018	.026	.022
z5	.017	.008	.009	.007	.006	.012	.009	.013	.016	.015
d5	.022	.012	.015	.011	.01	.016	.012	.015	.02	.018
thr10	.016	.015	.016	.013	.012	.019	.012	.014	.018	.016
thr5	.010	.01	.01	.008	.008	.013	.008	.009	.012	.011

sians filter (as in [34]). This convolved image is then further transformed. In the zero-crossing method ($z5$), the zero crossings are found in the convolved image (as in Marr and Hildreth [34]), and these are further thresholded based on their slope, so that 5 percent of the pixels remain. The other methods apply an absolute threshold to the convolved image which picks out the most negative-going portions of the convolved image (the negative peaks). We apply this method with absolute thresholds that leave 5, 10, and 15 percent of the pixels above threshold ($d5$, $d10$, and $d15$, respectively). The "d" here stands for "dog" or ***difference-of-Gaussians.***

The six transformations ($d5$, $d10$, $d15$, $z5$, $t5$, and $t10$) were applied to all 84 original image sequences, yielding 504 binary sequences. These sequences were then encoded with QT, BT, AHC, OT, three-dimensional AHC, and BQ, along with run-length encoding (RLE), for comparison's sake. The limited extents used were as follows.

QT, BT, AHC	32 X 32
OT	16 X 16 X 16
3D-AHC	32 X 32 X 16
BQ	32 X 32 X (2, 4, 8, and 16).

The last dimension given in the three-dimensional methods is time or number of frames. We applied BQ with various temporal extents in order to gauge the benefit of varying the depth of encoding. These applications of the method will be referred to as BQ2, BQ4, BQ8, and BQ16. Note that BQ with temporal depth of 1 is identical to QT, as previously discussed.

The results are given in Table IX. Each cell in the table represents a statistic based on a sample of 84 images. We have ordered the transformations and the methods so that the means generally decrease to the right and down. Thus, BQ16 yields the best compression ratio no matter which transformation is involved, relative to the other algorithms, and $t5$ yields images which are easier to encode for any hierarchical method compared to any other transformation. This latter fact is understandable since the thresholding methods yield large uniform regions in the image, for which hierarchical methods are best suited. The edge-detection methods give images with thin lines which require the methods to descend further in the encoding tree, using more symbols in the encoding.

A comparison of the methods is made clearer in Fig. 6,

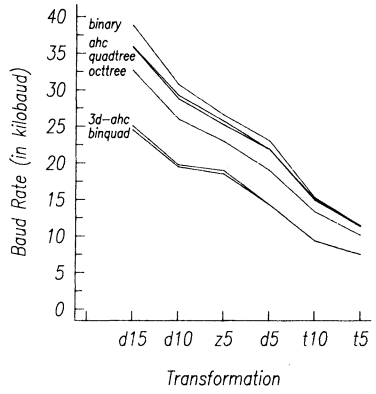


Fig. 6. Average baud rate for each transformation. The parameter is the encoding method used.

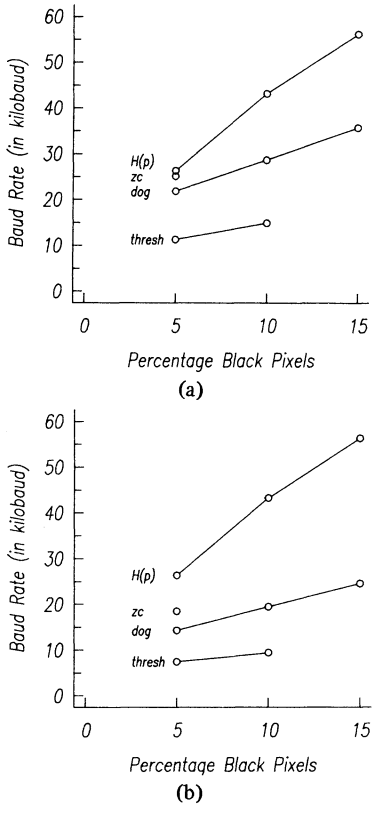


Fig. 7. Average baud rate as a function of the percentage p for quadtrees (a) and binquad (b).

where the method's action is plotted as a function of the transformation. Clearly, with these images the methods act with a certain amount of consistency. Thus, the ordering from best to worst method is preserved across transformations. Second, although AHC is slightly worse than QT in the two-dimensional case, in three dimensions the adaptive methods are clearly better than OT, and the hybrid method BQ has a slight edge over three-dimensional AHC.

As a comparison with our previous results on random pictures for various proportions p, Fig. 7 gives the results in terms of the percentage of black pixels in the images for QT and BQ16. For comparison's sake, these results are plotted with the corresponding information rate for that probability. These results clearly show the advantage of taking regional dependen-

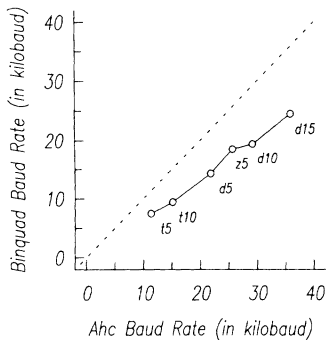


Fig. 8. A comparison of binquad and adaptive hierarchical coding. The dashed line represents the 45 degree line of equality between the two methods, for comparison's sake.

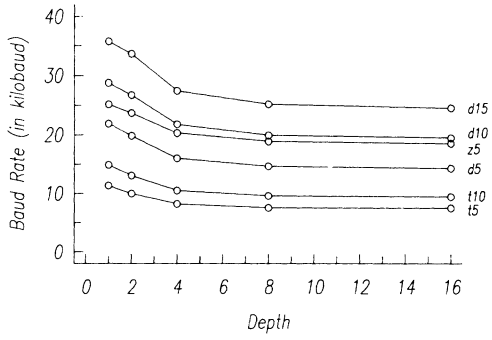


Fig. 9. Average baud rate as a function of encoding depth for the binquad coding method.

cies into account. They also show further how the method is dependent upon the type of images, with large uniform regions yielding the best results (the threshold methods), and small noisy spotty images being the most difficult to encode (as in *z*5).

Fig. 8 compares two methods, BQ16 and AHC, across the six transformations. Here we see a further demonstration of the consistency of the comparison between the methods on "natural" images (i.e., the graph is monotonic).

One obvious problem with the BQ method concerns its space requirements. In the case of real-time usage of the method, the encoder in BQ16 would need to buffer 16 frames at a time. First of all, this is potentially a large buffer space problem. More importantly, in the case of information transmission, this means that transmission will involve at least a 15-frame delay (in our case a delay of 1 s), which may cause some difficulties in an interactive communication system. In Fig. 9 we replot the BQ data in terms of encoding depth, or number of frames stored. Thus, we have points for each transformation corresponding to BQ16, BQ8, BQ4, BQ2, and QT (which is the same as BQ1). As can be seen in the figure, the major gain from the depth is made in the first four frames, and so a practical application might conceivably limit depth of encoding in images such as ours to between 0.25 and 0.5 s, without undue decrement in compression performance.

V. Summary and Conclusions

In this paper we have examined a class of binary image coding methods based upon a recursive hierarchical decomposition of the image into uniform areas. These methods provide very low compression ratios (as low as 0.08 bits per pixel) while there is no destruction of information; every pixel can be regained from the encoded form.

Hierarchical methods share a number of features. Given the commonality of how the algorithms operate, all of these algorithms have linear time complexity with the number of pixels. Also, since they all globally combine locally gathered information, they all lend themselves to parallel hardware implementations. Since buffer space and decision tree size is at a premium in any such implementation, our results concerning temporal depth are salient.

There is one important consideration which has not been discussed in this paper. We have been considering hierarchical methods for transmission of binary images which allow for the possibility of progressive transmission discussed earlier. Unfortunately, the transmission is dependent upon an errorless communication channel. By eliminating redundancy entirely in the bit stream, any single bit error can totally confuse the receiver. Thus, any real implementation of such a method in a noisy environment (such as modems and phone lines) will need some form of noise protection, and the consequent increase in baud rate. The most appropriate method for solving the transmission problem depends on the nature of the possible degradations within the communication system. A discussion of the communication systems is beyond the scope of this paper.

We have introduced a number of hierarchical methods, and have proposed a number of new extensions incuding adaptivity, and the incorporation of the time dimension into the encoding. The performance of these methods has been analyzed theoretically with respect to time complexity and worst case coding, and statistically for random images and for a body of images derived from natural scenes. From this analysis we can see that adaptivity is of little use with two-dimensional images unless they are highly structured, but in three-dimensions a substantial savings can be derived thereby. With our images, the hybrid method we call binquad coding appears to be the most effective.

Appendix

A.1. Expected Code Length for the Quadtree Representation of Random Pictures

Notation:

$B(n) = \{x_i\}$ is the set of all pictures with 4^n spatially adjacent pixels (e.g., all possible squares of sides 2^n).

$B_b(n) \subset B(n)$ such that $\forall x \in B_b(n), \quad x = 0$

$B_w(n) \subset B(n)$ such that $\forall x \in B_w(n), \quad x = 1$

$B_g(n) = B(n) - (B_b(n) \cup B_w(n))$, the set of all nonuniform images.

$u(n) = \text{Prob}\ \{B_u(n) \cup B_b(n)\}$, the probability that $B(n)$ is uniform.

$g(n) = \text{Prob}\ \{B_g(n)\}$, the probability that $B(n)$ is nonuniform.

$\bar{L}(n)$ = the expected code length for $B(n)$.

$\bar{L}_u(n)$ = the code length for $B_b(n)$ and $B_w(n)$, the uniform blocks.

$\bar{L}_g(n)$ = the expected code length for a nonuniform block $B_g(n)$.

p = the probability that a single pixel is black.

$q = 1 - p$ = the probability that a single pixel is white.

S_g is the length in bits of the "meta-symbol" or gray node.

The following simple relation holds

$$u(n) = p^{4^n} + q^{4^n}. \tag{A.1}$$

We begin with a derivation which utilizes none of the tricks used to eliminate redundancy. We will then incorporate those savings one by one. The basic relation from which we start is that the expected code length of a given block is a linear combination of the expected code length in the case that the block is uniform, and of the expected code length in the case that the block is nonuniform; each of these cases is weighted by the appropriate probability

$$\bar{L}(n) = u(n)\,\bar{L}_u(n) + g(n)\,\bar{L}_g(n). \tag{A.2}$$

In the case that the block is nonuniform, we can break it up into different combinations of uniform and gray subblocks. The nonuniform block can be divided either into four nonuniform subblocks, into any combination of uniform and nonuniform subblocks, or into four uniform subblocks which are not of the same color. Each of the five last lines in (A.3) below represents one of these combinations weighted by the appropriate probability.

$$\begin{aligned}\bar{L}(n) &= u(n)\,\bar{L}_u(n)\\ &\quad + g^4(n-1)\,(S_g + 4\bar{L}_g(n-1))\\ &\quad + 4g^3(n-1)\,u(n-1)\,(S_g + 3\bar{L}_g(n-1) + \bar{L}_u(n-1))\\ &\quad + 6g^2(n-1)\,u^2(n-1)\,(S_g + 2\bar{L}_g(n-1) + 2\bar{L}_u(n-1))\\ &\quad + 4g(n-1)\,u^3(n-1)\,(S_g + \bar{L}_g(n-1) + 3\bar{L}_u(n-1))\\ &\quad + (u^4(n-1) - u(n))\,(S_g + 4\bar{L}_u(n-1)).\end{aligned} \tag{A.3}$$

(A.3a)

The last term involves a factor of $(u^4(n-1) - u(n))$ rather than simply $u^4(n-1)$ in order to account for a block whose four quadrants are all uniform, *but not of the same color.*

Rearranging terms

$$\begin{aligned}\bar{L}(n) &= u(n)\,\bar{L}_u(n)\\ &\quad + S_g((g(n-1) + u(n-1))^4 - u(n))\\ &\quad + 4g(n-1)\,\bar{L}_g(n-1)\,(g(n-1) - u(n-1))^3\\ &\quad + 4\bar{L}_u(n-1)\,u(n-1)\,((g(n-1)\\ &\qquad + u(n-1))^3 - u(n)/u(n-1)).\end{aligned} \tag{A.4}$$

We can now substitute the following: $\bar{L}_u(n) = 2$ since a uniform block is coded as 2 bits, $S_g = 1$ since a gray node is coded as a single bit, $g(n-1) + u(n-1) = 1$ by definition, and lastly

$$\bar{L}_g(n-1) = (\bar{L}(n-1) - u(n-1)\,\bar{L}_u(n-1))/g(n-1)$$

which follows directly from (A.2); (A.4) can be now written as

$$\begin{aligned}\bar{L}(n) &= 2u(n)\\ &\quad + 1 - u(n)\\ &\quad + 4(\bar{L}(n-1) - 2u(n-1))\\ &\quad + 8u(n-1) - 8u(n)\\ &= 4\bar{L}(n-1) + 1 - 7u(n).\end{aligned} \tag{A.5}$$

Finally, substituting the value of $u(n)$ according to (A.1) we get

$$\bar{L}(n) = 4\bar{L}(n-1) + 1 - 7(p^{4^n} + q^{4^n}). \tag{A.6}$$

It remains to specify the starting equations for the recursion. If the simplest code is selected, namely, that uniform blocks are represented by 2 bits and gray nodes are represented by a single bit, the recursion is started with

$$\bar{L}(0) = 2. \tag{A.7}$$

If the lowest level nonuniform subblocks are represented by a gray node followed by a 4 bit nibble, then, since 5 bits are needed to represent a nonuniform block with a probability of $1 - u(1)$, and 2 bits are needed to represent a uniform subblock that occurs with a probability of $u(1)$, the starting equation is

$$\begin{aligned}\bar{L}(1) &= 2u(1) + 5(1 - u(1))\\ &= 5 - 3u(1)\\ &= 5 - 3(p^4 + q^4).\end{aligned} \tag{A.8}$$

If the more compact scheme is used in which only 4 bits, instead of 5, are used to code a nonuniform 2×2 subblock when the first three pixels are of the same color, then

$$\begin{aligned}\bar{L}(1) &= 2u(1) + 4(p^3q + qp^3) + 5(1 - p^3q - pq^3 - u(1))\\ &= 5 - (p^3q + pq^3) - 3(p^4 + q^4).\end{aligned} \tag{A.9}$$

Next, we derive $\bar{L}(n)$ under the coding scheme in which on every level of the tree, if the first three subblocks are uniform and of the same color, then the fourth can be coded in one bit if it is uniform (and of the opposite color), since it has only to be distinguished from the case in which the fourth subblock if nonuniform. Clearly, the only line in (A.3) that has to be modified is the last line (A.3a) which takes care of the case in which 4 uniform subblocks are coded. The term (A.3a) now takes the form

$$\begin{aligned}&(u^4(n-1) - u(n))\,(S_g + 4\bar{L}_u(n-1))\\ &\quad - (w^3(n-1)\,b(n-1) + w(n-1)\,b^3(n-1)).\end{aligned} \tag{A.10}$$

Replacing line (A.3a) by line (A.10) gives the expected code length under the modified coding scheme

$$\begin{aligned}\bar{L}(n) &= 4\bar{L}(n-1) + 1 - 7(p^{4^n} + q^{4^n})\\ &\quad - (w^3(n-1)\,b(n-1) + w(n-1)\,b^3(n-1)).\end{aligned} \tag{A.11}$$

Expressing $b(n-1)$ and $w(n-1)$ in terms of p and q we get the final form

$$\bar{L}(n) = 4\bar{L}(n-1) + 1 - 7(p^{4^n} + q^{4^n}) - p^{3\times 4^{n-1}} q^{4^{n-1}} - p^{4^{n-1}} q^{3\times 4^{n-1}}. \quad \text{(A.12)}$$

A.2. Expected Code Length for Binary-Tree Representation of Random Pictures

The same notation will be employed as previously, with one minor change. The subscript n now denotes a block-size of 2^n instead of 4^n pixels. The quadtree equation (A.2) applies to binary trees as well. (A.1), however, has to be modified to reflect the fact that blocks in binary trees are divided into two subblocks instead of four

$$u(n) = p^{2^n} + q^{2^n}. \quad \text{(A.13)}$$

The derivation of an expression for $\bar{L}(n)$ for binary trees is similar to that of quadtrees. The coding assumptions are as follows. A nonuniform block is represented by 1 bit followed by the codes for the two subblocks. A uniform subblock is, in most cases, represented by 2 bits. There are two exceptions to this: first, if the subblock is on the lowest level (i.e., it is a single pixel) it is represented by a single bit. Second, when the subblock is the second of two uniform subblocks that constitute a nonuniform block, it can be represented by 1 bit less. Thus, if a single pixel is the second in a pair of pixels, then it need not be coded at all.

We start the derivation from (A.2) which is reproduced here

$$\bar{L}(n) = u(n)\,\bar{L}_u(n) + g(n)\,\bar{L}_g(n). \quad \text{(A.2)}$$

Expanding the second term

$$\begin{aligned}\bar{L}(n) = {} & u(n)\,\bar{L}_u(n) \\ & + g^2(n-1)\,(S_g + 2\bar{L}_g(n-1)) \\ & + 2g(n-1)\,u(n-1)\,(S_g + \bar{L}_g(n-1) + \bar{L}_u(n-1)) \\ & + (u^2(n-1) - u(n))\,(S_g + 2\bar{L}_u(n-1) - \tfrac{1}{2}\,\bar{L}_u(n-1)).\end{aligned} \quad \text{(A.14)}$$

Rearranging the terms

$$\begin{aligned}\bar{L}(n) = {} & u(n)\,\bar{L}_u(n) \\ & + S_g(g(n-1) + u(n-1))^2 - u(n) \\ & + 2g(n-1)\,\bar{L}_g(n-1)\,(g(n-1) + u(n-1)) \\ & + 2u(n-1)\,\bar{L}_u(n-1)\,(g(n-1) + u(n-1)) \\ & - \bar{L}_u(n-1)\,(\tfrac{1}{2}\,u^2(n-1) - \tfrac{1}{2}\,u(n) + 2u(n)).\end{aligned} \quad \text{(A.15)}$$

Substituting in (A.15):

$$\bar{L}_u(n) = 2;$$

$$S_g = 1;$$

$$g(n-1) + u(n-1) = 1,$$

and

$$\bar{L}_g(n-1) = (\bar{L}(n-1) - u(n-1)\,\bar{L}_u(n-1))/g(n-1)$$

[which follows directly from (A.2)], we get

$$\begin{aligned}\bar{L}(n) = {} & 2u(n) \\ & + 1 - u(n) \\ & + 2\bar{L}(n-1) - 4u(n-1) \\ & + 4u(n-1) \\ & - 2(\tfrac{1}{2}\,u^2(n-1) - \tfrac{1}{2}\,u(n) + 2u(n)) \\ = {} & 2\bar{L}(n-1) + 1 - 2u(n) - u^2(n-1).\end{aligned} \quad \text{(A.16)}$$

Finally, substituting the value of $u(n)$ according to (A.13)

$$\bar{L}(n) = 2\bar{L}(n-1) + 1 - 2(p^{2^n} + q^{2^n}) - (p^{2^{n-1}} + q^{2^{n-1}})^2. \quad \text{(A.17)}$$

The recursion is started with

$$\bar{L}(1) = 2 \quad \text{(A.18)}$$

ACKNOWLEDGMENT

Special thanks to G. Sperling, who conceived of the American Sign Language project and provided the support and encouragement that made this image processing system a reality. We also wish to acknowledge the assistance of T. Riedl, A. van der Beek, and R. Picardi.

REFERENCES

[1] T. Pavlidis, *Algorithms for Graphics and Image Processing.* Rockville, MD: Computer Science, 1982.

[2] A. Rosenfeld, "Quadtree grammars for picture languages," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-12, pp. 401–405, 1982.

[3] G. M. Hunter and K. Stieglitz, "Operations on images using quadtrees," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-1, pp. 145–153, 1979.

[4] J. E. Warnock, "A hidden-surface algorithm for computer generated half-tone pictures," Dep. Comput. Sci., Univ. Utah, Salt Lake City, Tech. Rep. 4-15, 1969.

[5] S. L. Horowitz and T. Pavlidis, "Picture segmentation by a tree traversal algorithm," *J. ACM*, vol. 23, pp. 368–388, 1976.

[6] A. Klinger and C. R. Dyer, "Experiments on picture representation using regular decomposition," *CGIP*, vol. 5, pp. 68–105, 1976.

[7] S. L. Tanimoto, "Hierarchical approaches to picture processing," Ph.D. dissertation, Dep. Elec. Eng., Princeton Univ., Princeton, NJ, 1975.

[8] S. L. Tanimoto and T. Pavlidis, "A hierarchical data structure for picture processing," *CGIP*, vol. 2, pp. 104–119, 1975.

[9] H. Samet, "Region representation: Quadtrees from boundary codes," *Comm. ACM*, vol. 23, pp. 163–170, 1980.

[10] —, "An algorithm for converting rasters to quadtrees," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-3, pp. 93–95, 1981.

[11] —, "Connected component labeling using quadtrees," *J. ACM*, vol. 28, pp. 487–501, 1981.

[12] C. R. Dyer, A. Rosenfeld, and H. Samet, "Region representation: Boundary codes from quadtrees," *Comm. ACM*, vol. 23, pp. 171–179, 1980.

[13] G. M. Hunter, "Efficient computation and data structures for graphics," Ph.D. dissertation, Dep. Elec. Eng. Comp. Sci., Princeton Univ., Princeton, NJ, 1978.

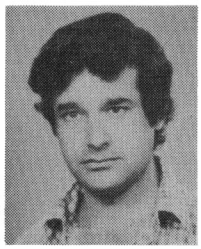
[14] G. M. Hunter and K. Stieglitz, "Linear transformations of pictures represented by quadtrees," *CGIP*, vol. 10, pp. 289–296, 1979.

[15] I. Gargantini and H. H. Atkinson, "Linear quadtrees: A blocking technique for contour filling," *Pattern Recognition*, vol. 17, pp. 285–293, 1984.

[16] L. Jones and S. S. Iyengar, "Representation of a region as a forest of quadtrees," in *Proc. IEEE PRIP-81 Conf.*, Dallas, TX, pp. 57–59, 1981.

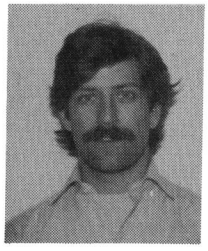
[17] H. Samet and A. Rosenfeld, "Quadtree representations of binary images," in *Proc. 5th Intern. Conf. Pattern Recognition*, Miami Beach, FL, pp. 815–818, Dec. 1980.

[18] L. G. Shapiro, "Data structures for picture processing: A survey," *CGIP*, vol. 11, pp. 162–184, 1979.
[19] C. L. Jackins and S. L. Tanimoto, "Oct-trees and their use in representing three-dimensional objects," *CGIP*, vol. 14, pp. 249–270, 1980.
[20] D. Meagher, "Geometric modeling using octree encoding," *CGIP*, vol. 19, pp. 129–147, 1982.
[21] I. Gargantini, "Linear oct-trees for fast processing of three-dimensional objects," *CGIP*, vol. 20, pp. 365–374, 1982.
[22] K. Knowlton, "Progressive transmission of grey scale and B/W pictures by simple, efficient and lossless encoding schemes," *Proc. IEEE*, vol. 68, pp. 885–896, 1980.
[23] K. R. Sloan and S. L. Tanimoto, "Progressive refinement of raster images," *IEEE Trans. Comput.*, vol. C-28, pp. 871–874, 1979.
[24] C. R. Dyer, "The space efficiency of quadtrees," *CGIP*, vol. 19, pp. 335–348, 1982.
[25] W. I. Grosky and R. Jain, "Optimal quadtrees for image segments," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-5, pp. 77–83, 1983.
[26] E. Kawaguchi and T. Endo, "On a method of binary-picture representation and its application to data compression," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-2, pp. 27–35, 1981.
[27] I. Gargantini, "An effective way to represent quadtrees," *Comm. ACM*, vol. 25, pp. 905–910, 1982.
[28] A. Klinger and M. L. Rhodes, "Organization and access of image data by areas," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-1, pp. 50–60, 1979.
[29] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Comm. ACM*, vol. 18, pp. 509–517, 1975.
[30] M. Tamminen, "Performance analysis of cell based geometric file organizations," *CVGIP*, vol. 21, pp. 160–181, 1983.
[31] G. Sperling, "Bandwidth requirements for video transmission of American Sign Language and finger spelling," *Science*, vol. 210, pp. 797–799, 1980.
[32] —, "Video transmission of American Sign Language and finger spelling: Present and projected bandwidth requirements," *IEEE Trans. Commun.*, vol. COM-29, pp. 1993–2002, 1981.
[33] G. Sperling, M. Pavel, Y. Cohen, M. Landy, and B. Schwartz, "Image processing in perception and cognition," in *Physical and Biological Processing of Images. Rank Prize Funds International Symposium at the Royal Society of London*, O. J. Braddick and A. C. Sleigh, Eds. Berlin: Springer-Verlag, 1982.
[34] D. Marr and E. Hildreth, "Theory of edge detection," *Proc. Royal Soc. London*, vol. B207, pp. 187–217, 1980.
[35] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, pp. 1098–1101, 1952.
[36] M. S. Landy, Y. Cohen, and G. Sperling, "HIPS: A UNIX-based image processing system," *CGIP*, vol. 25, pp. 331–347, 1984.
[37] —, "HIPS: Image processing under UNIX. Software and applications," *Behav. Res. Methods, Instr., Comput.*, vol. 16, pp. 199–216, 1984.



Yoav Cohen received the M.A. degree in psychology from the Hebrew University of Jerusalem, Israel, in 1977 and the M.Sc. degree in computer science and the Ph.D. degree in cognitive psychology, both from the University of Oregon, Eugene, in 1981.

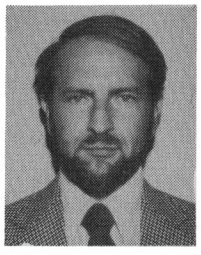
From 1981 to 1983 he was a Research Associate at the Human Information Processing Laboratory in the Psychology Department of New York University, New York. He is currently the Assistant Director of the National Institute for Testing and Evaluation in Jerusalem, Israel.



Michael S. Landy received the B.S. degree in electrical engineering and computer science from Columbia University, New York, in 1974, and the M.S. and Ph.D. degrees in computer and communication sciences from the University of Michigan, Ann Arbor, in 1977 and 1981, respectively.

From 1981 to 1984 he was a Research Scientist at the Human Information Processing Laboratory in the Department of Psychology at New York University, New York. He is currently an Assistant Professor in the Department of Psychology at NYU. His research interests include models of visual perception, applications of image processing to psychology, and computer assisted instruction.

Dr. Landy is a member of the Association for Research on Vision and Ophthalmology and the Pattern Recognition Society.



M. (Misha) Pavel (S'67–M'75) received the B.S. degree in electrical engineering from the Polytechnic Institute of New York, New York, in 1970, the M.S.E.E. degree from Stanford University, Stanford, CA, in 1971, and the Ph.D. degree in experimental psychology from New York University, New York, in 1980.

From 1970 to 1976, he was a member of the Technical Staff at Bell Laboratories, Holmdel, NJ, working on the problems of modeling human and network behavior. From 1976 to 1980 he was the Director of Technical Services at the Department of Psychology at New York University, and until 1983 a Research Assistant Professor in the same department. Currently he is Assistant Professor in the Department of Psychology at Stanford University where he is organizing the Applied Psychology Laboratory. In addition, he is consulting with a number of industrial organizations including Xerox Palo Alto Research Center. His research activities include acoustic and image processing, auditory and visual perception, and human–computer interaction.