Asymmetric Binary Tree Coding for Contour Images

Ahsan Shamim
Design Engineer, DSP SW
Nokia Mobile Phones, R&D
Salo 24101, Finland.
Ahsan.Shamim@nokia.com

John. A. Robinson *(corresponding author)*
Department of Electronics,
University of York,
Heslington, York, YO10 5EE, UK.
Jar11@ohm.york.ac.uk
Tel: +44 1904 432353
Tel: +44 1904 432355

(This work was conducted when both authors were at Faculty of Engineering and Applied Science, Memorial University of Newfoundland, St. John's, Newfoundland, Canada A1B3X5)

Abstract **- We consider the coding of featured contours, i.e. texture, object or motion boundaries in images. The MPEG-4 and MPEG-7 video coding standards provide for a content-based representation of video information, so efficient coding of boundaries can play an important role. We propose a new coding scheme that uses an asymmetric binary tree. We show that the new scheme outperforms conventional quadtree, binary tree, contour and READ coding algorithms applied to typical boundary images and also offers competitive performance for general-purpose two-level image coding.**

Keywords: **Hierarchical coding, contour coding.**

I. INTRODUCTION

Asymmetric Binary Tree (ABT) Coding is a hierarchical method for representing and compressing the boundaries of regions, objects or textures in images. Applied to motion-segmented video frames, the scheme efficiently compresses the boundary component in MPEG-4 or MPEG-7 object-based coding [1]. More broadly, it can code all types of segmented image more economically than standard alternatives and a variant has competitive performance for bilevel images in general. We describe the method in detail and report experiments that substantiate these claims.

Segmented images can be represented with two levels, corresponding to region interiors and boundaries. In contrast to other two-level image types, such as scanned documents, segmented images consist of thin, continuous contours. Some of the methods used for coding segmented images exploit this; others do not [2]. We first review scan-based schemes, most often used for document compression, which do not assume thin contours, and then discuss chain coding schemes, which do. Section II reviews tree-based methods in more detail, since the new scheme falls into this class. Section III describes the new scheme, Asymmetric Binary Tree coding, and its variants, then Section IV compares these with the alternatives on a range of images

I.1 Scan Based Schemes

The simplest two-level coding scheme is runlength coding [3]. Here the distances between adjacent transitions in the picture level are transmitted. Runs more than the maximum possible

runlength are coded using a make-up codeword plus a terminator. Variable-word-length coding can be applied to the runlengths.

The principle of relative address coding [4] is that transitions on the current line most often occur close to related transitions on the previous line. By coding the displacements between the current line's transitions and their neighbors on the previous line, this closeness is exploited. Special cases where transitions on one line do not match transitions on the previous are coded with runlengths and "pass codes". READ coding, as used in G4 fax and the TIFF image format is a version of RAC.

I.2 Contour Schemes

Contour, or chain, coding [5-6] is often more suited to representing object or motion boundaries than scan-based coding. The start point of a contour is identified by an absolute address, and then the contour is traced by transmitting the direction of the current pixel relative to the previous pixel. The total number of transmitted bits in a chain-coding scheme depends on the number of active pixels in an image. Differential coding of directions improves the efficiency when the contours are mostly straight or slowly curving.

II. TREE-STRUCTURED SCHEMES

The basic idea behind tree-structured coding schemes is to segment a picture into the largest possible uniform areas and to transmit a hierarchical representation of these areas.

II.1 Quadtree Coding

Quadtree coding [7] works by successive subdivision of rectangular (usually square) blocks. In general, three different codewords are used: **All_White (W)** is transmitted if the block is uniformly white, **All_Black (B)** if it is uniformly black, and **Split (S)** if it contains both black and white pixels. All blocks coded as **S** are quartered and the same coding scheme is applied to the subblocks. The process continues until all subblocks to be analyzed are exhausted. At the 1x1 pixel level, only two codewords are required, to distinguish white from black.

Quadtree coding has numerous variants. For boundary/contour images, a black pel represents *activity*. **All_Black (B)** blocks correspond to blocks where every pixel is active and have low probability of occurrence. The codeword **B** may therefore be omitted (except for the 1x1 level): all active blocks are instead coded by subdivision down to individual pixels. Figure 1(a) illustrates this variant of quadtree coding.

II.2 Binary Tree Coding

A representation of pictures by binary trees was first explored by **Knowlton** [8]. In this coding scheme a block with activity is divided into two sub-blocks. **Cohen et al.** [9] proposed an improved binary coding technique where the total number of cuts is optimized by finding the most effective cut direction (horizontal or vertical) for each block. The idea is to choose cut directions to minimize the number of blocks with activity. This scheme requires direction of cut and the structural information of the block to be passed to represent a block. For an (m×n) block, where both m and n are greater than 1, codeword **Split (S)** is passed as a structural representation, if the block has activity, otherwise **All_White (W)** is passed. (Again, the basic scheme also uses an **All-Black (B)** codeword, but since we are focussing on the coding of contours, we consider a variant that omits this case for m×n blocks.) In order to split a block, the direction information is

passed before the structural codeword of sub-blocks. If the effective cut is in the same direction as the previous cut direction, codeword **Unchanged (U)** is transmitted, otherwise **Changed (C)** is transmitted. For single pixel thick blocks such as (m×1) and (1×m) blocks with m>1, an **All_Black (B)** codeword *is* required (even for contour images). In these cases direction information is not required as there is only one possible cut direction. The coding of a (1×1) block is trivial but the symbols **B**(black) and **W**(white) can be used for consistency.

Codes can be converted to binary via a back-end entropy coder, or in a simple mapping such as the following:

For (m×n) blocks:

US→10, CS→11, W→0.

For (m×1) and (1×m) blocks (m>1):

S→0, W→10, B→11

For (1×1) block:

B→0, W→1.

Fixed simple mappings such as these mean the coding is very fast but slightly suboptimal. We consider this issue for ABT coding later. Figure 1(b) illustrates the binary tree coding scheme.

III. ASYMMETRIC BINARY TREE (ABT) CODING

Asymmetric tree coding divides blocks unequally if to do so will increase coding efficiency. The schemes we describe here are all binary and therefore involve only the decision about where to cut a block into two. Moreover all cuts are horizontal or vertical, so all subblocks are rectangles. In describing the variants of asymmetric binary tree coding we specify how cuts are made and how they are coded. The allocation of particular fixed codewords is described here, but justified in section IV.1, where the statistical properties of the scheme are analyzed experimentally.

III.1 Asymmetric Binary Tree (ABT) Coding with Logarithmic Extension

We consider three types of blocks. First: (m×n) blocks where **m** and **n** are both greater than 1. Second: (m×1) and (1×m) blocks where **m>2** and Third: (1×2) and (2×1) blocks. Initially, all blocks are considered to be in the *Primary Stage* and a specific cut direction is specified. Once the cut direction is fixed, we divide the block in the middle. If there is activity in both blocks, we send a codeword representing a split into two equal blocks. Both blocks will then be analyzed separately in the later stage. If there is activity in one block only, we send a codeword identifying which of the two this is. We then continue to subdivide the block with activity and check whether the block adjacent to that without activity has activity or not. If it has no activity then we send a continuation bit (value 1) and continue the subdivision and checking process until there is activity in the adjacent sub-block. For example, if the initial block had width 32, and the first split yielded a left subblock (width 16) without activity and a right subblock (also width 16) with activity, the left subblock would be extended by 8, 4, 2, then 1 until further extension would include active

pixels. This expansion is called “Logarithmic Extension” because the amount added to the inactive subblock halves at each step. When continuation ends before the maximum possible number of continuation steps, we send a termination bit (value 0). Otherwise, the maximum possible continuation indicates termination of splitting. That is, no cut needs to be encoded because its position is implicit from preceding continuation marks. The block with activity is analyzed further in later stages. If there is a continuation mark with blocks having activity in one sub-block only, or if there is activity in both blocks, the cut direction of the resultant blocks will be opposite to the previous direction. Otherwise, it will be the same as the previous direction. In the latter case, where the block resulted from splitting in the previous stage and activity was in one sub-block only, we know that the location of the activity in the sub-block prevented a further split. Such blocks are termed *Advanced Blocks* and require fewer bits than blocks in the *Primary Stage*. For (1×m) and (m×1) blocks the cut direction is always the same, cutting the longer axis. The coding is the same as that of an (m×n) block, but there is an additional codeword for a block that is completely black. (1×2) and (2×1) blocks are regarded in the same way as (1×m) and (m×1) blocks. Again, blocks at *advanced stages* are more efficiently coded than those at *primary stages*.

Codewords and a mapping to binary for the ABT scheme are shown in Table 1. Here the first column corresponds to *primary stage* blocks and the second column corresponds to *advanced stage* blocks.

For (m×n) blocks, codewords E and F represent activity in one sub-block only. As the location of activity in one sub-block of an *advanced stage* block is known from the previous stage it is not necessary to encode whether the first or the second sub-block is active. The same applies to

codewords L and M in (m×1) and (1×m) blocks. As the location of activity in one pixel of a (2×1) and (1×2) block in the *advanced stage* is known, codeword **T** represents non-activity while **R** represents activity in the remaining pixel.

Figure 1(c) illustrates ABT with logarithmic extension. The block to be coded is a (16×16) pixel area which has activity in four pixels at row 5 and columns 13-16. The proposed algorithm needs the initial cut direction to be transmitted, which is either horizontal (H), or vertical (V) and it can be coded by one bit. With the initial cut direction codeword **V,** the block will be cut in the vertical direction. The block is initially at the Primary stage. With the cut position at the middle of the block it is found that there is activity in the second sub-block. So pseudocode **C** is transmitted. Then the continuation process will start and the sub-block with activity will be halved in the vertical direction. It is found that the first sub-subblock has no activity. So a continuation mark **1** is transmitted. Then the remaining sub-subblock is examined and halved. It is found that now both sub-subblocks have activity. So a termination mark **0** is transmitted. There will be no more cuts in that direction and the previous cut position is the final cut location at this stage. The sub-block with activity will be revisited in the next level of the tree for further coding. As it results from the continuation process and code **1** was transmitted, the cut direction will be changed in the next stage to the horizontal direction. In the next stage **B0** is transmitted, as no continuation is possible. The next cut direction is unchanged and the sub-block with activity is in the advanced stage, because activity in the second sub-block blocked the continuation process in the previous stage. Codeword **F0** is transmitted. The remaining block with activity is still in the advanced stage as the cut direction has not been changed, and it resulted from one non-active sub-block in the previous stage. At this stage **E1** is transmitted. Here no termination mark **0** is required because after transmitting **1** the remaining sub-block is reduced to one pixel wide. So the

continuation process reaches its maximum level which is an implicit termination mark. Then the cut direction will be changed. The remaining block is now a (1×4) block and it has activity in all pixels. So codeword **J** is transmitted.

The decoder will perform the reverse process. At first it receives **V** and it becomes ready to cut in the vertical direction. Then it receives **C**, meaning there is activity in the second sub-block. Then it looks for **1** or **0**, and continues division of the second sub-block until a termination mark is found. The termination mark will locate the cut position and all pixels of the first sub-block are made non-active (i.e. represented by white pixels). As there was continuation, so the cut direction is changed and the process continues until all codewords are decoded.

The number of processing steps required by ABT in this example is five, whereas the number of steps required by the quadtree and the binary tree algorithms are five and seven respectively.

III.2 Asymmetric Binary Tree (ABT) Coding with Linear Extension

The advantage of ABT with logarithmic extension is that it does not include much overhead from failing to pass the continuation-checking test. On the other hand, its drawback is that the scheme does not locate the exact position of the first active pixel in the block that is to be analyzed in the next stage. This means inefficient coding of the block during later stages. This leads to the modification of the ABT coding scheme to allow linear extension of inactive blocks. Though the extra bits to indicate the exact location are costly, later coding may be more efficient.

In the linear extension approach, an inactive subblock is extended to the exact location of activity in its active neighbour. For example, if the initial block had width 32, and the first split yielded a left subblock (width 16) without activity and a right subblock (also width 16) with activity, the left subblock would be extended to the exact location of the activity by transmission of a 4-bit number specifying the size of the additional extension. In general, if the subblock size is d then an additional $\log_2 d$ bits are required to locate the exact activity position. With this approach two different cases are considered to choose the cut direction. In case 1, the cut direction will alternate each time when the block of interest enters a new processing stage. In case 2, the cut direction will not alternate if the block having activity in one sub-block fails at the very first continuation checking stage.

Figure 1(d-e) explains the two cases of the linear extension of ABT coding scheme.

In case1, with the initial vertical cut direction, the code sequence **VC10** is transmitted to code the initial block until the first termination point (as in the case of ABT with logarithmic extension). The *critical distance* that causes the termination is 2. This requires a single bit code **0** to locate the exact position of activity. The next cut direction for the resulting sub-block is horizontal. The code sequence **B0** is transmitted to code the sub-block until the first termination point. The *critical distance* is then 4 and requires a two-bit code **11** to locate the position of the activity. The next cut direction is vertical and the same process goes on until all sub-blocks to be analyzed are exhausted.

In Case 2, the first code sequence is **VC100** as with case 1. Then follows the code sequence **B011**. Here the sub-block fails in the continuation-checking test after the codeword **B** is

transmitted. So the next cut direction remains horizontal and the same process will continue until all sub-blocks are coded.

III.3 ABT with Merging for Thicker Textual Images

ABT as so far described is suitable for thin boundary images. However we seek a scheme in which the cost of misapplication – to an image with thick black lines, for example – is not too high. We now explain a modification that allows ABT to code areas of black more efficiently, and therefore enables it to be used as a general-purpose two-level coder that can be applied to textual images, for example. In this modification an extra codeword, **Y**, is transmitted when an (m×n) block is completely black. Inclusion of this extra code requires a reassignment of bits to codewords. To make the method effective, different coding techniques (with and without code **Y**) are applied on a processing stage basis. A processing_stage_flag to select one out of the two sets of codewords is transmitted at the start of a processing stage. If the processing_stage_flag is **0**, the codewords will be 0(A, D), 10(B, E), 11(C, F) for an (m×n) block. If the processing_stage_flag is **1**, the codewords will be 0(A, D), 100(B, E), 101(U), 11(C, F) or 0(A, D), 10(B, E), 110(C, F), 111(U).

A merging algorithm is then applied to code a group of totally black blocks efficiently. At the start of a processing stage, all blocks are arranged according to their location, scanning from the Top-Left to the Bottom-Right. If the codeword of the block is **Y**, the coder looks for continuation in its longer direction (if *width* > *height* then in the horizontal direction, otherwise in the vertical direction), because there is a higher probability of continuation in the longer direction. The continuation criteria for a totally black block are given below:

- Mergable blocks must have same *height* (horizontal direction) or same *width* (vertical direction).
- The processing stage of the block must be greater than or equal to the processing stage of the previous totally black block.
- The block must be aligned with the previous block having codeword **Y,** where the alignment is done at the time of creation of the block.

For each block which matches these criteria, a continuation codeword **1** is transmited. If any of the above mentioned conditions is violated or the search point (the next possible position of a totally black block) is not yet coded, then there will be a termination mark **0**. This also helps in reducing the number of totally black blocks to be processed at higher stages as these blocks are merged at lower stages.

To select the codeword to overload, the number of blocks having a white first sub_block (First_white) at all processing stages with processing_stage_flag = 1 is determined. Similarly, the number of blocks having a white second sub_block (Second_white) is determined. If the number of First_white is greater than the number of Second_white, then codewords **C** and **F** are overloaded. Otherwise, codewords **B** and **E** are overloaded.

IV. EXPERIMENTS

IV.1 Efficiency of ABT Codeword Allocation

We first report statistical results on the efficiency of the simple codeword mapping scheme described in Table 1 for the following six cases:

Case 1: coding of (m×n) *primary stage* blocks.

Case 2: coding of (m×n) *advanced stage* blocks.

Case 3: coding of (m×1) or (1×m) *primary stage* blocks.

Case 4: coding of (m×1) or (1×m) *advanced stage* blocks.

Case 5: coding of (2×1) or (1×2) *primary stage* blocks.

Case 6: coding of (2×1) or (1×2) *advanced stage* blocks.

The analysis is performed on four two-level images of increasing complexity: the first two are images with thin contours and the last two are thicker textual images. Table 2 summarizes the results, comparing the bit costs of using the code symbols given in Table 1 against the entropy of the symbol set used in that case. The entropy is calculated by $H = -\sum_i p_i \log_2 p_i$, where p_i is the probability of occurrence of the *i*th symbol for that coding case.

From Table 2, it can be observed that the codewords selected in the *primary stages* of the proposed technique are very close to the entropy for all four experiments. There is a scope to improve the codeword selection in the *advanced stages*. But this will not affect the total number

of transmitted bits too much as the number of blocks in *advanced stages* is small compared to that of the number of blocks in the *primary stages*.

As the fixed coding scheme is faster than the use of a back-end entropy coder and the average number of bits in all cases are very close to the entropy, we have used the simple codeword mapping (Table 1) for the performance analysis of ABT coding scheme.

IV.2 Comparison of Tree structures for quadtree, binary tree and ABT

Figure 2(a) shows a motion boundary image with size (240×352), where each white segment corresponds to an independent moving region. The extraction of the motion boundaries is by hierarchical global-to-local motion segmentation as explained in [1]. In this section, the performance of ABT is compared against quadtree and conventional binary tree coding for the sample image Figure 2(a).

IV.2.1 Quadtrees

Figure 2(b) shows a magnified contour image superimposed with grids to illustrate the application of a quadtree algorithm. Here each block requires one of the two possible codewords, **W** indicating a complete white block and **S** otherwise. As the quadtree algorithm quarters each block with activity, the rectangular boundary image is partitioned into a number of 64×64 blocks to obtain an unbiased quadtree. In order to obtain equal sized blocks the contour image is padded with pixels having no activity. With the example shown in Figure 2(b), 5576 bits are required to transmit the exact motion boundary information of the tested boundary image of size 240 rows by 352 columns.

IV.2.2 Binary Trees

Figure 2(c) illustrates the result using the binary tree algorithm suggested by **Cohen et al.** [3] with the same sample image. Here the total number of transmitted bits is 5515, which shows slightly better performance than the quadtree coding. The advantage comes from the minimization of blocks with activity using effective cut direction. But there is no substantial amount of savings in terms of bits with this complex algorithm relative to the simple quadtree coding.

IV.2.3 ABT Coding

Figure 3 shows the application of ABT (logarithmic) coding on motion boundary images. For figure 3(a) (the same image as used in figure 2), the coding scheme requires 4351 bits, which is about 22% improvement over the traditional quadtree and binary algorithms in terms of number of bits. For figure 3(b), the algorithm requires 3393 bits whereas quadtree and conventional binary tree schemes require 4178 and 4566 bits respectively. Figure 4 shows the application of the ABT coding scheme with linear extension for both cases on a motion boundary image. The number of bits is 4182 for case 1 and 4279 for case 2.

Figure 5 shows the application of ABT (linear, Case 1) with the merging technique on a textual image. The merging technique improves the coding of thicker textual images except on some occasions where number of merged totally black blocks is insufficient to overcome the cost of coding the black blocks.

ABT allows progressive transmission of two-level image structure, where the resolution of the transmitted image increases with the processing stage. A processing stage corresponds to every

active block in the picture being split once. The maximum number of processing stages for an image to be coded using ABT is given by,

$$P = \log_2(number \quad of \quad rows) + \log_2(number \quad of \quad columns)$$

because there may be a block to be coded at the final stage that is obtained by splitting equally the corresponding previous stage blocks until the final stage sub-block is reduced to a single active pixel. Figure 6 shows the variation of the resolution of the transmitted image with processing stages. The figure shows that the largest possible uniform regions are coded at the initial stages facilitating image transmission at significantly lower bit rates with perhaps usable quality (for example, in very low data-rate video). Figure 6(d) is the original and final stage image.

The number of bits transmitted for the example shown in Figure 6 at processing stages 7, 12, 14 and 17 are 137, 1478, 2809, and 4351 respectively.

IV.3 Compression Performance of Asymmetric Binary Tree Coding

Finally we compare the compression performance of ABT against READ coding, contour coding, conventional quadtree coding, and conventional binary tree coding. The READ coder used in the experiments here is a full implementation and represents the current general-purpose standard for document image compression as used in CCITT Group 4 facsimile and the Tagged Image File Format (TIFF with Group 4 compression). The contour coder is an adaptive differential contour coder that automatically selects between coding 4-connected and 8-connected contours and between direct representation of black-pixel chains in the image and coding region boundaries for thick black areas. It uses a switched set of variable word length codes for

representing differential directions, allowing for efficient coding of straight line segments without compromising performance on curly contours. It codes the start and branch locations of contours and their lengths efficiently. The contour coder is therefore an effective implementation, which we would expect to perform very well on boundary images. The quadtree and binary tree implementations are conventional and use the codeword allocation described in Section II.

Figure 7 shows the representative set of images used for results here. The class of images with which we are most concerned are typified by Figure 7(a-d) – boundary images, either of motion or spatial segmentation. The first two of these represent the sparse end of the spectrum of possible images. They are derived according to the object-based motion estimation method of [1]. The busier boundary images represent more detailed segmentations and are obtained from a conventional bottom-up segmentation algorithm [10]. Figure 7(e-h) show textual images for which, as explained earlier, we want ABT to perform reasonably well.

Figure 8 and Table 3 give the coding results for the eight test cases in figure 7. ABT with merging is shown only for the thicker images.

Asymmetric Binary Tree coding always outperforms conventional quadtree and binary tree coding. It is always better than READ coding for contour images, and, with merging, only slightly inferior for textual images. ABT is substantially better than contour coding for images with many contours and for textual images with fine detail (e.g. small fonts). However, when there are few contours, it cannot perform as well as contour coding. The worst case found in all our experiments is coding of Figure 7(a) for which ABT requires 3298 bits against contour coding's 2469 bits. Contour coding also performs well for the thick image in Figure 7(g) where it encodes the boundaries of the black areas.

Table 4 shows the proportion of ABT's coded bits required for compression using READ and contour coding. A number below 1 means the alternative outperforms ABT. This table confirms

that ABT is never the worst of the three alternatives and for contour images with more than a few lines it is the best.

V. CONCLUSION

Asymmetric Binary Tree Coding is an extension of binary tree coding for two-level images that divides blocks unequally to increase efficiency. We have described three variants corresponding to different ways of extending the size of coded subblocks, and a final variant which allows for merging of all-black areas. Experiments show that the variation in performance between the three approaches to size extension is small compared to the difference in performance between ABT and alternative schemes. All variants outperform previous versions of quadtree and binary tree coding for all image types. On segmented images, they outperform scan-based schemes, but their performance relative to contour coding depends on the number of segments. With very few segments, contour coding is superior. The fourth variant – the inclusion of merging – has little effect on performance for contour images, but greatly improves ABT when applied to document images. With merging, ABT is competitive with scan-based coding even for documents. Therefore, this final version, ABT with merging, can substitute for READ coding as a general-purpose bilevel compression scheme, while giving particularly good compression of segmented or contour images. The scheme is implemented efficiently using simple codewords.

VI. REFERENCES

[1] **Shamim, A., Robinson, J.A**., "Object-Based Video Coding by Global-to-Local Motion Segmentation", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol 12, No 12, December 2002.

[2] **Robinson, J. A.,** "Low-data-rate Visual Communication using Cartoons: a Comparison of Data Compression Techniques", *IEE Proceeddings*, Vol. 133, No. 3, pp. 236-256, June 1986.

[3] Golomb, S., "Run-length encodings", *IEEE Transactions on Information Theory*, Vol 12, pp 399-401, 1966.

[4] **Bodson, D., Urban, S. J., Deutermann, A. R., Clarke, C. E.,** "Measurement of Data Compression in Advanced Group 4 Facsimile System", *Proceedings of the IEEE*, Vol 73, No. 4, pp 731-739, April 1985.

[5] **Freeman, H.,** "On the Encoding of Arbirtrary Geometric Configurations", *IRE Transactions*, EC-10, pp. 260-268, 1961.

[6] **Freeman, H., Saghri, A.,** "Generalized Chain Coders for Planar Curves", *Proceedings of the fourth International Joint Conference on Pattern Recognition*, pp 701-703, Japan, November 1978.

[7] **Samet, H., Rosenfeld, A.,** "Quadtree Representation of Binary Images", *Proceedings of the fifth International Conference on Pattern Recognition*, Miani, FL, pp 815-818, December 1980.

[8] **Knowlton, K.,** "Progressive Transmission of Grey Scale and B/W Pictures by Simple Efficient and Lossless Encoding Schemes", *Proceedings of IEEE*, Vol. 68, pp. 885-896, 1980.

[9] **Cohen, Y., Landy, M. S.,** and **Pavel, M.,** "Hierarchical Coding of Binary Images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 3, pp 284-298, May 1985.

[10] **Gonzalez, R. C., Woods, R. E.,** "Digital Image Processing", 3rd edition. Addison Wesley 1992.

List of Tables and Figures

For (m×n) blocks,

Activity in both blocks (Split Equally):	A →0	D →0
Activity only in the first block:	B →10	E →1
Activity only in the second block:	C →11	F →1

For (m×1) and (1×m) blocks,

Activity in both blocks (Split Equally):	G →00	K →0
Activity only in the first block:	H →01	L →10
Activity only in the second block:	I →10	M →10
All_black block:	J →11	N →11

For (2×1) and (1×2) blocks,

Activity in both blocks:	O →0	R →0
Activity only in the first block:	P →10	T→1
Activity only in the second block:	Q →11	T→1

Table 1: Codewords used in the ABT coding scheme.

Experiment 1 (Sparse contour image)						
	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case5**	**Case6**
Average number of bits with the ABT coding scheme	1.5353	1	2	1.4348	1.5438	1
Entropy	1.5314	0.5506	1.976	1.1916	1.5211	0.5917
Experiment 2 (Busier contour image)						
	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case5**	**Case6**
Average number of bits with the ABT coding scheme	1.4836	1	2	1.6482	1.5443	1
Entropy	1.4776	0.937	1.987	1.58	1.5131	0.9299
Experiment 3 (Small-font textual image)						
	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case5**	**Case6**
Average number. of bits with the ABT coding scheme	1.4804	1	2	1.7667	1.5115	1
Entropy	1.4789	0.9951	1.8679	1.4984	1.4952	0.9627
Experiment 4 (Textual image with thicker characters)						
	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case5**	**Case6**
Average number. of bits with the ABT coding scheme	1.4123	1	2	1.5	1.5271	1
Entropy	1.3897	0.6415	1.9711	1.4549	1.4317	0.5505

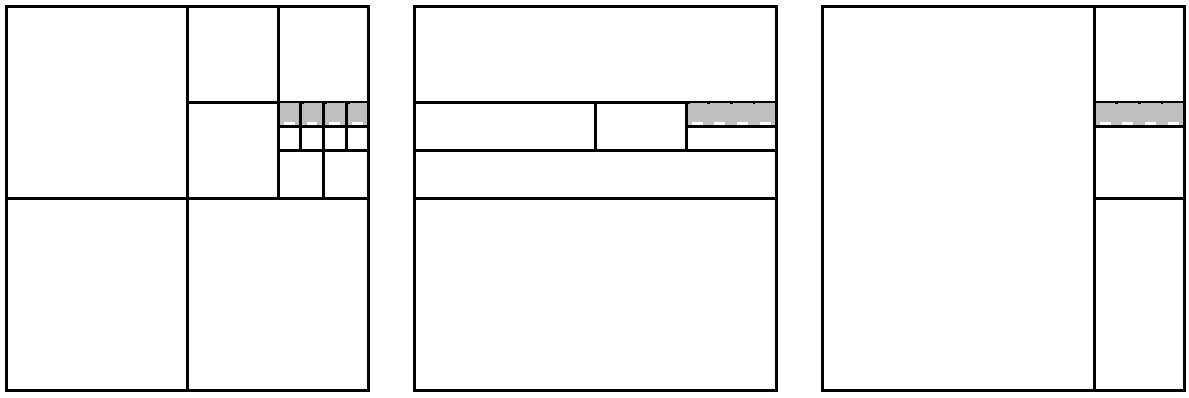
Table 2: The efficiency of ABT codeword allocation.

Input Image	Compressed size in bits							
	READ	Contour	Quadtree	Binary tree	ABT (Log.)	ABT (Lin., Case 1)	ABT (Lin., Case 2)	ABT with merging
Table tennis	5920	2469	3996	4486	3420	3365	3298	
Toy train	6560	3843	5576	5515	4351	4182	4279	
Lenna	34592	29113	31232	33481	25582	25448	25869	
Goldhill	35600	34032	34594	37816	29217	28957	30280	
Caere corp	12416	16544	13578	16476	12985	12177	12688	12309
Photocell	10048	14305	16172	16418	13802	13510	13605	11241
Capacitor	5936	5175	12930	12692	10001	9850	9593	5677
Primaries	8064	10216	18971	20394	15131	15206	14995	9867

Table 3: Performance comparison of coding algorithms

	Table tennis	Toy train	Lenna	Goldhill	Caere corp	Photocell	Capacitor	Primaries
ABT	1	1	1	1	1	1	1	1
READ	1.79	1.53	1.34	1.23	1.01	0.89	1.05	0.82
Contour	0.75	0.92	1.13	1.18	1.34	1.27	0.91	1.04

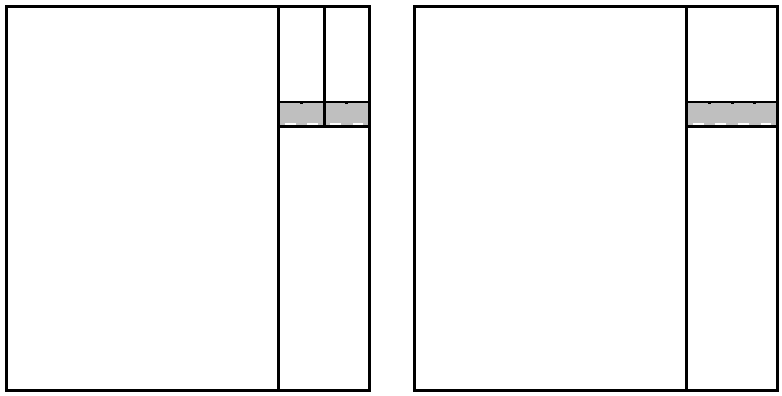
Table 4: Relative compressed sizes of the test images, normalized so ABT=1.



(a) (b) (c)

Code-string:

S	(US)	VC10
W S W W	(US) W	B0
W W W S	W (US)	F0
S S W W	(CS) W	E1
B B W W B B W W	W (US)	J
	W (CS)	
	B W	

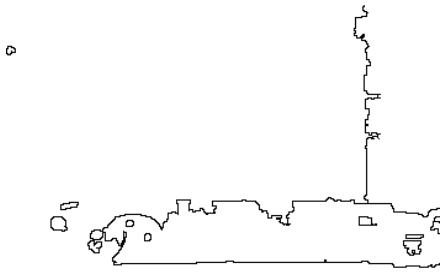


(d) (e)

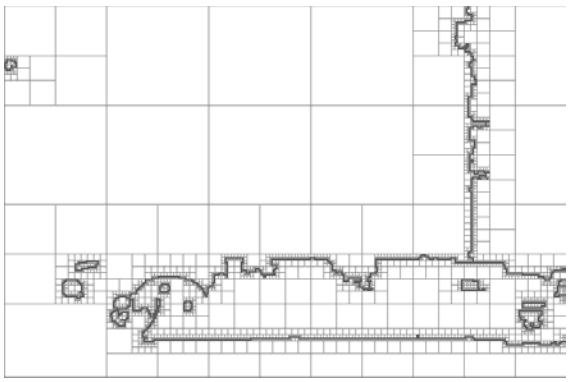
Code-string:

VC100	VC100
B011	B011
A	F11
C11C11	J
OO	

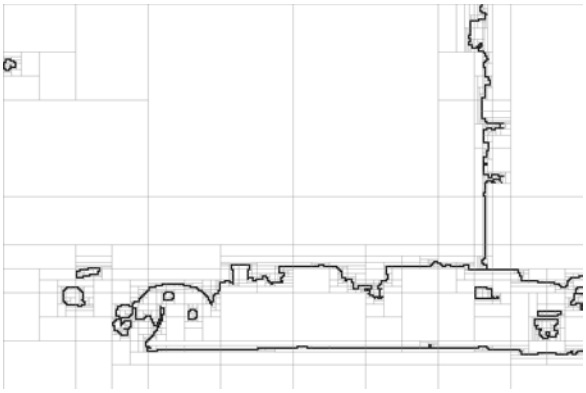
Figure 1. Tree coding examples. (a) Quadtree, (b) Binary Tree, (c) ABT coding with logarithmic extension, (d) ABT with linear extension (Case 1), (e) ABT with linear extension (Case 2).



(a)

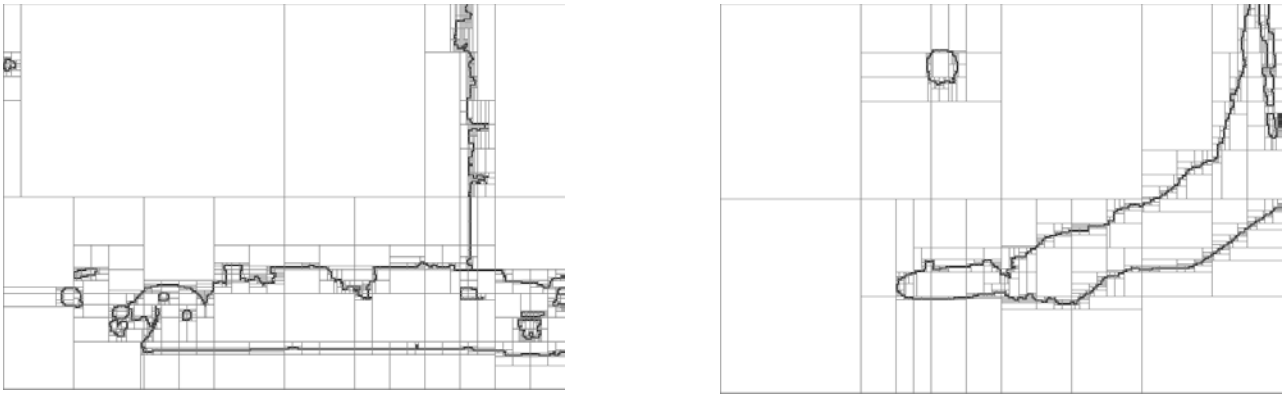


(b)



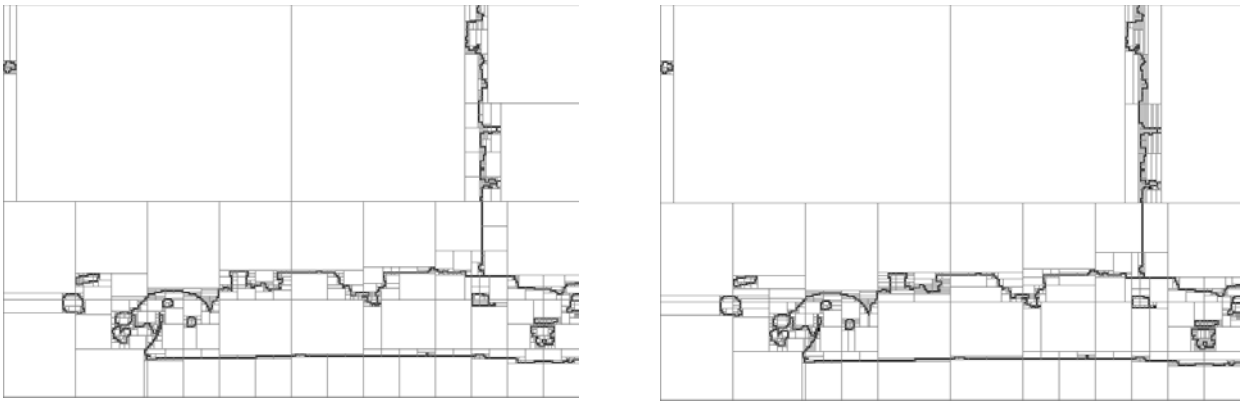
(c)

Figure 2. (a) Sample boundary image, (b) Test image superimposed with grids to illustrate quadtree coding, (c) Application of conventional binary tree coding.



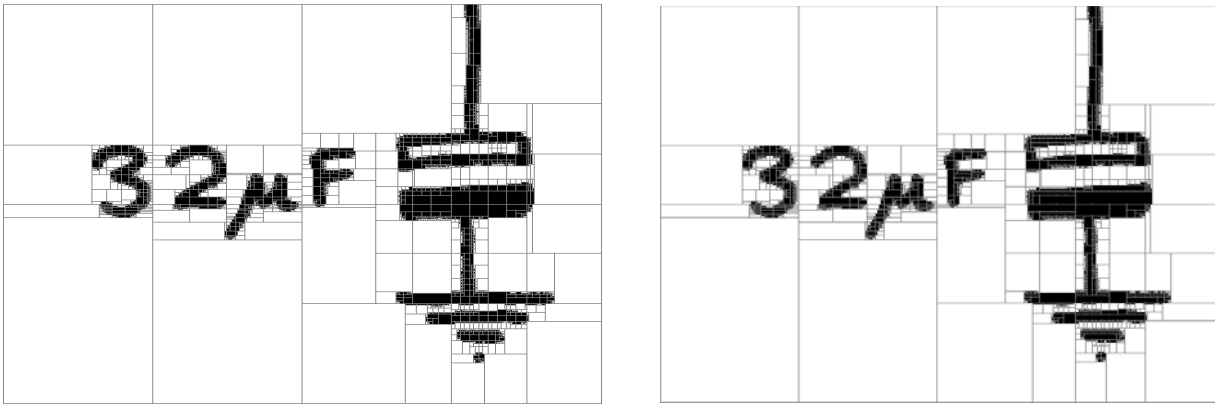
(a) (b)

Figure 3. Application of the ABT (logarithmic) coding scheme on a motion boundary image of (a) the *toy train* sequence, (b) *table tennis* sequence.



(a) (b)

Figure 4. Application of the ABT (linear) coding scheme on a motion boundary image of the *toy train* sequence for (a) Case 1, (b) Case2.



(a) (b)

Figure 5. ABT coding scheme for a thicker textual image (a) before merging, (b) after merging.

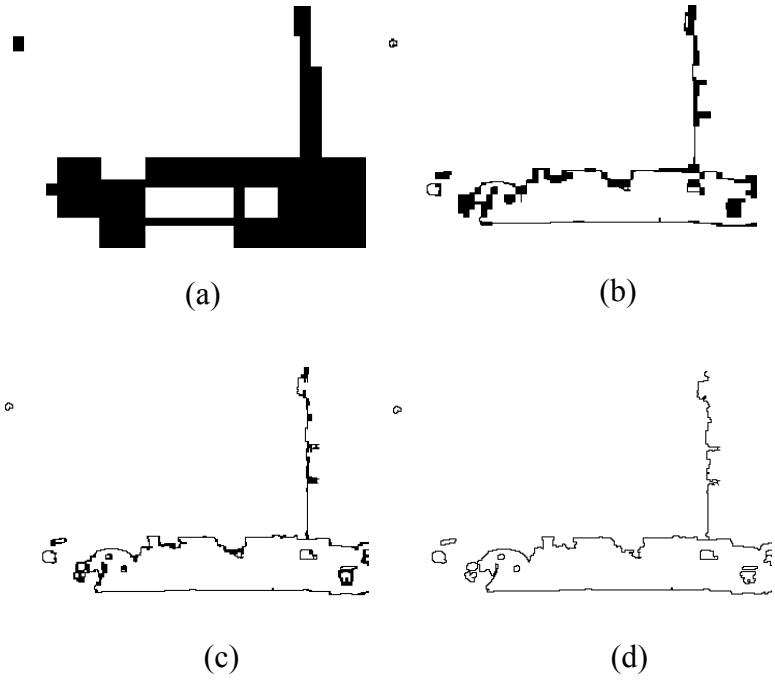
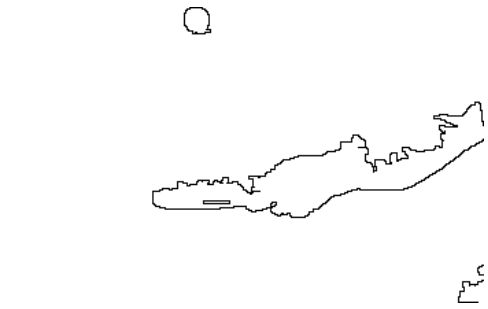
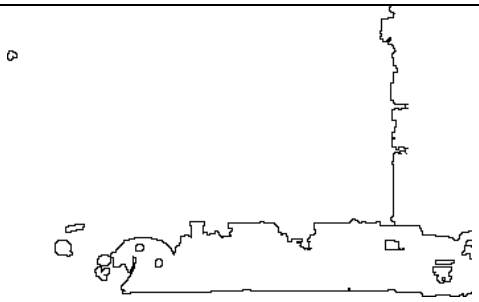


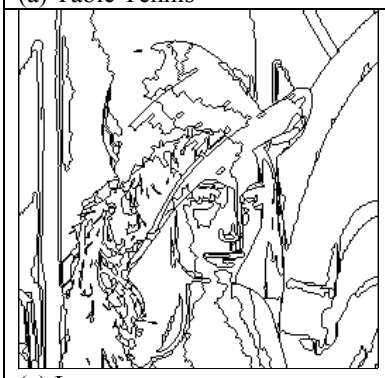
Figure 6. Transmitted and decoded boundary image at processing stage (a) 7, (b) 12, (c) 14, (d) 17.



(a) Table Tennis



(b) Toy Train (Mobile and Calendar)



(c) Lenna

(d) Goldhill

Caere Corporation

475 Potrero Avenue

Sunnyvale, CA 94086

USA

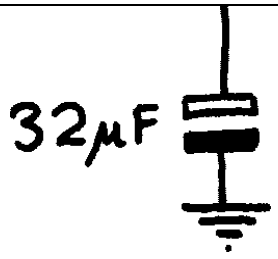
Tel: 408-720-8300

Fax: 408-720-1330

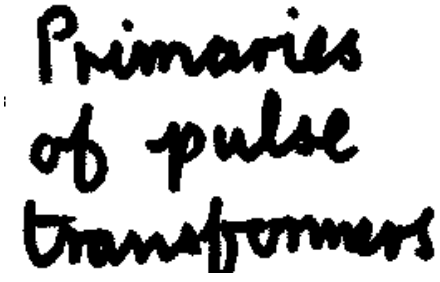
(e) Caera

a photocell is caused
The variations of pri
l to generate an analc

(f) Photocell



(g) Capacitor



(h) Primaries

Figure 7. Test cases representing (a)(b) sparse contour images, (c)(d) busy contour images, (e)(f) fine textual images, (g)(h) thick textual images

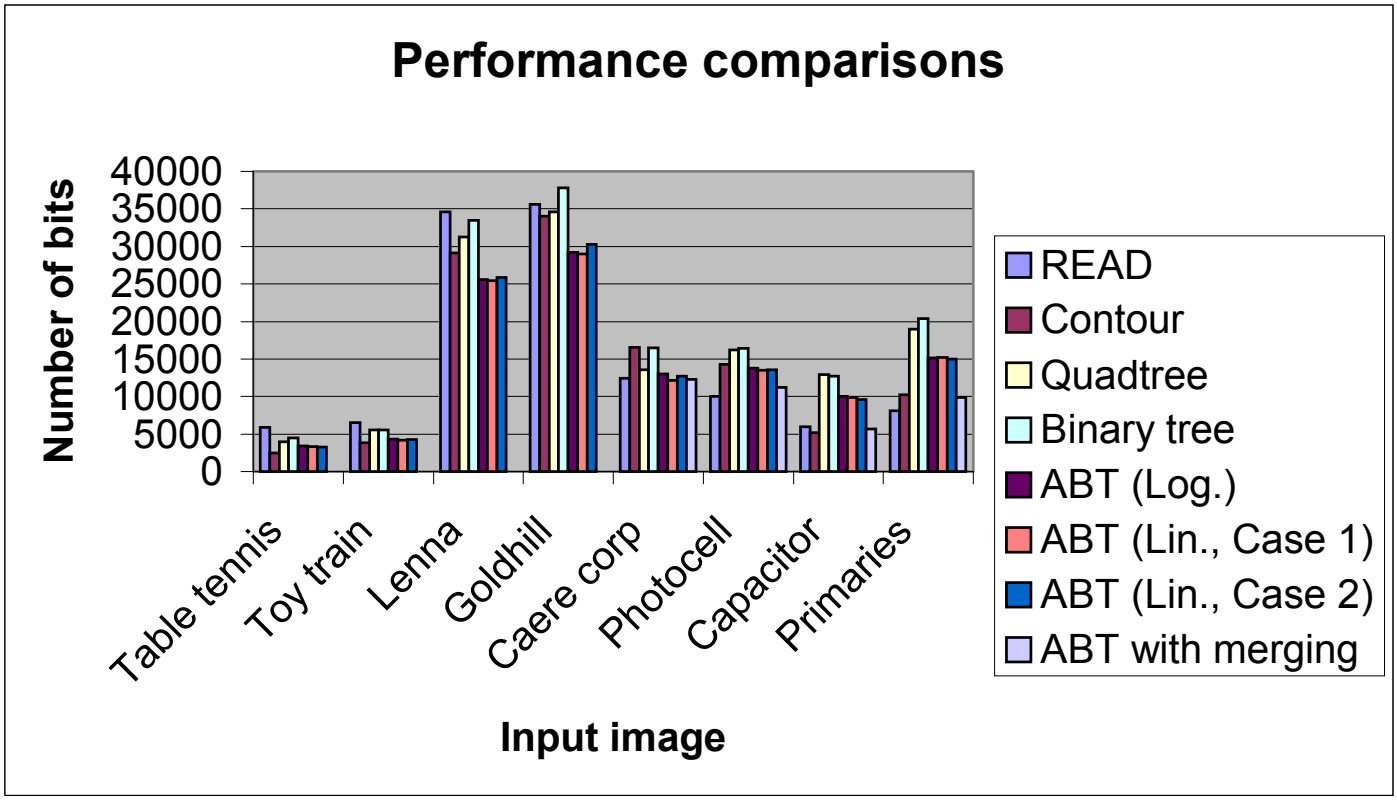


Figure 8. Performance comparison of coding algorithms.