Retrospective Theses and Dissertations

1988

Three-level block truncation coding

Deborah Ann Lee
University of Central Florida

Part of the [Systems and Communications Commons](#)
Find similar works at: <https://stars.library.ucf.edu/rtd>
University of Central Florida Libraries <http://library.ucf.edu>

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Lee, Deborah Ann, "Three-level block truncation coding" (1988). *Retrospective Theses and Dissertations*. 4304.
<https://stars.library.ucf.edu/rtd/4304>

THREE-LEVEL
BLOCK TRUNCATION CODING

BY

DEBORAH A. LEE
B.S., University of Florida, 1981

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Graduate Studies Program
of the College of Engineering
University of Central Florida
Orlando, Florida

Fall Term
1988

ABSTRACT

Block Truncation Coding (BTC) techniques, to date, utilize a two-level image block code. This thesis presents and studies a new BTC method employing a three-level image coding technique. This new method is applied to actual image frames and compared with other well-known block truncation coding techniques.

The method separates an image into disjoint, rectangular regions of fixed size and finds the highest and lowest pixel values of each. Using these values, the image block pixel value range is calculated and divided into three equal sections. The individual image block pixels are then quantized according to the region into which their pixel value falls; they are quantized to a 2 if they fall in the upper third, a 1 in the middle third, and a 0 if in the lower third range region. Thus, each pixel now requires only two bits for transmission. This is one bit per pixel more than the other well-known BTC techniques and thus it has a smaller compression ratio.

When the BTC techniques were applied to actual images, the resulting 3LBTC reconstructed images had the smallest mean-squared-error of the techniques applied. It also produced favorable results in terms of the entropy of the reconstructions as compared to the entropy of the original images. The reconstructed images were also very good replicas of the originals and the 3LBTC process had the fastest processing speed.

For applications where coding and reconstruction speed are crucial and bandwidth is not critical, the 3LBTC provides an image coding solution.

ACKNOWLEDGEMENTS

I would like to thank the faculty of the Department of Electrical Engineering, especially my advisor, Dr. Y. A. Alsaka, and Dr. N. S. Tzannes, for guidance and encouragement as I endeavored to complete this thesis and the degree requirements for a Master of Science.

Special thanks are extended to Brian E. Bateman, for his time and effort spent photographing the images included within this thesis.

This thesis is dedicated to my husband, Gary, without whose patience and support this thesis would not have become reality.

TABLE OF CONTENTS

LIST OF TABLES	v
LIST OF FIGURES	vi
LIST OF ILLUSTRATIONS	vii
Chapter	
I. INTRODUCTION	1
II. CURRENT CODING TECHNIQUES	6
Block Truncation Coding	6
Absolute Moment Block Truncation Coding	7
Entropy-Preserving Block Truncation Coding	8
III. THREE-LEVEL BLOCK TRUNCATION CODING	10
The Method	10
Variations of Three-Level Block Truncation Coding	15
IV. CURRENT CODING TECHNIQUE COMPARISONS	35
Compression Ratio	35
Mean-Square-Error	35
Entropy	38
The Effects of Errors in b on a	39
Processing Time	43
V. APPLICATIONS TO ACTUAL IMAGES	44
Compression Ratio	61
Mean-Square-Error	76
Entropy	79
Processing Time	82
VI. CONCLUSIONS	83
APPENDICES	84
A. F77 Listing of Three-Level BTC Variations Comparison	85
B. Results of Three-Level BTC Variations Comparison	96
C. F77 Listing of Current Coding Techniques Comparison	119
D. Results of Current Coding Techniques Comparison	126
E. PASCAL Listings of Color Image Computer Application	138
REFERENCES	169

LIST OF TABLES

1.	The Original 3LBTC Requires Less Bandwidth Than the Variations	30
2.	The Compression Ratio of the Original 3LBTC and Its Technique Variations.	29
3.	The 3LBTC Mathematically Outperforms the Variations in Terms of MSE .	31
4.	Entropy Comparison of Eleven Reconstructed Image Blocks	33
5.	3LBTC Has the Fastest Processing Time of Its Variation Techniques . .	34
6.	BTC Methods Reduce the Overall Bandwidth Required for Image Transmission	36
7.	BTC Techniques Compression Ratio Comparison	37
8.	MSE Results of the Reconstructed Images As Compared to the Original . .	37
9.	The Reconstructed Images Entropy Results As Compared to the Originals .	39
10.	3LBTC has the Fastest Processing Time of the BTC Techniques Compared .	43
11.	The 3LBTC Method Has the Largest Image Transmission Bandwidth	62
12.	BTC Methods Yield Compression Ratios as Great as 3.75:1	63
13.	The Quality of the Reconstructed Images Are Mathematically Compared in Terms of MSE	76
14.	The AMBTC Has the Smallest MSE for the Green RGB Color Component . .	77
15.	The 3LBTC and the AMBTC Have the Smallest MSE for the Blue RGB Color Component	78
16.	The Amount of Information in the Original Images	79
17.	The EBTC Has the Largest Entropy for the Red RGB Color Component . .	79
18.	The 3LBTC Has the Smallest Entropy for the Green RGB Color Component .	80
19.	The EBTC and the 3LBTC Have the Smallest Entropy for the Blue RGB Color Component	81
20.	3LBTC Processes the Images at Least Twice as Fast as the Current Coding Techniques	82

LIST OF FIGURES

1.	The Block Truncation Coding Algorithm	4
2.	Block Truncation Coding Reconstruction Algorithm	5
3.	The Three-Level Block Truncation Coding Process	11
4.	The Three-Level BTC Reconstruction Algorithm	12
5.	A Sample Image Block to Show the Simplicity of the 3LBTC Process . . .	13
6.	All Sample Image Block Pixel Values Fall Into One of Three Sections . . .	13
7.	The Section in Which the Individual Image Block Pixels Lie Determine Their Coded Value	14
8.	The Reconstructed Image Block Is Similar to the Original	14
9.	The High/Low Coding Algorithm Is Similar to the 3LBTC	16
10.	The Mid Section Is Used to Calculate Both Absolute Moments	15
11.	The Mid/Low Coding Algorithm Is a Variation of the 3LBTC	18
12.	The Mid Section Is Used to Calculate the Low Absolute Moment	17
13.	The Mid/High Coding Algorithm Is Similar to the Mid/Low	20
14.	The Mid Section Is Used to Calculate the High Absolute Moment	19
15.	The Mean-Splits-Mid Coding Algorithm Divides the Mid Section	22
16.	The Block Mean Separates the Pixels for Calculating the High and Low Absolute Moments	21
17.	The High/Low Mean Coding Algorithm Uses the Mean Instead of the Range . .	23
18.	The Block Mean Divides the Pixels	24
19.	The 3LBTC Variation Reconstruction Process Is Similar to the AMBTC Process	26
20.	The Pixels Are Reconstructed According to Their Section	25

21.	High/Low Reconstructed Sample Image Block	28
22.	Mid/Low Reconstructed Sample Image Block	28
23.	Mid/High Reconstructed Sample Image Block	28
24.	Mean-Splits-Mid Reconstructed Sample Image Block	28
25.	High/Low Reconstructed Sample Image Block	28
26.	Slight Errors in b Do Not Affect the Reconstruction	40
27.	Large Errors in b Affect the Reconstruction MSE	41
28.	Large Errors in b Affect the Reconstruction Entropy	42
29.	Each RGB Component Has a Maximum Value of 31	44

LIST OF ILLUSTRATIONS

1.	Blocking an Image Frame Is the First Step in the BTC Process	2
2.	Blocking An Image Frame Is Also the First Step in the 3LBTC Process . . .	10
3.	The Original Images As Seen at the Transmission Side	45
4.	The BTC Technique Applied to the CAT Image Produces Favorable Results . .	46
5.	BTC Blocking Effect Is Evident in Minute Details of the CARTOON Image . .	47
6.	The BABY Image Is Recognizable in the BTC Reconstructions	48
7.	The BTC DEER Images Were Subjectively the Best Reconstructions . . .	49
8.	AMBTC Reconstructions Are Similar to Those of the BTC	50
9.	Minute Detail Is Lost in AMBTC Reconstructions	51
10.	AMBTC Reconstructions of the BABY Produce Favorable Results	52
11.	AMBTC Reconstructions Closely Resemble the Originals	53
12.	EBTC Reconstructions Are Comparable to Those of the BTC and AMBTC . .	54
13.	EBTC Reconstructions Lose Minute Details	55
14.	Distortions Occur as the EBTC Blocking Size Is Increased	56
15.	EBTC DEER Reconstructions Produce Favorable Results	57
16.	3LBTC Reconstructions Are Closest to the Originals	58
17.	Minute Details Are Still Distinguishable with a 3LBTC 20 x 32 Blocking Size	59
18.	Distortions Are Not As Evident with the 3LBTC	60
19.	The 3LBTC Produces the Brightest Reconstructed Images	61
20.	Reconstructed CAT Images Using a Blocking Size of 5 x 8 Produce Favorable Results	64
21.	Reconstructed CAT Images Using a 10 x 16 Blocking Size Show Evidence of the Blocking Technique	65

22.	Reconstructed CAT Images Using a Blocking Size of 20 x 32 Are Abstracts of the Original	66
23.	CARTOON Reconstructions Employing a Blocking Size of 5 x 8 Produce Acceptable Images	67
24.	CARTOON Reconstructions Employing a 10 x 16 Blocking Size Show Evidence of the Blocking Technique	68
25.	CARTOON Reconstructions Employing a 20 x 32 Blocking Size Are Abstracts of the Original	69
26.	BABY Reconstructions Using a 5 x 8 Blocking Size Are Good Facsimiles of the Original	70
27.	Details Are Lost in the BABY Reconstructions Utilizing a Blocking Size of 10 x 16	71
28.	Abstract BABY Reconstructions Are Produced with a Blocking Size of 20 x 32	72
29.	DEER Reconstructions, Employing A 5 x 8 Blocking Size, Produce Excellent Replicas of the Original	73
30.	Minute Details of the DEER Reconstructions Are Lost with a Blocking Size of 10 x 16	74
31.	DEER Reconstructions, Employing a 20 x 32 Blocking Size, Still Produce Distinguishable Replicas of the Original	75

I. INTRODUCTION

The demands on telecommunication systems to transmit data, voice and video are rapidly increasing. Simpler, less expensive solutions to satisfy these demands, other than sending additional satellites into orbit, building additional microwave stations, or laying additional fiber optic lines, have emerged in the development of data compression and coding techniques.

Of the numerous digital image coding techniques developed, some of the most widely recognized are the Fast Fourier Transform (FFT), Huffman coding, the Karhunen-Loeve transform and Block Truncation Coding. The FFT requires many complex computations, and both the Huffman coding and the Karhunen-Loeve transform require statistical knowledge of the image [1].

Block Truncation Coding techniques have an advantage of being easily implemented. Many variations of the original Block Truncation Coding (BTC) [2] technique have recently been developed. Among them are Adaptive Block Truncation Coding [3], Absolute Moment Block Truncation Coding (AMBTC) [4], and Entropy-Preserving Block Truncation Coding (EBTC) [5].

The Block Truncation Coding process requires that an image frame be divided into a set of smaller blocks. Illustration 1 is an example of a divided image frame. Statistical calculations, usually the mean and moments, are performed on each of the smaller blocks. One of these statistical values, normally the block mean, is used to reduce the pixel values within its block to a single bit. If a pixel value is greater than or equal to the mean, the pixel value is coded (or set) to a value of one. If the pixel value is less than the mean, it is coded to a zero. This reduces the pixel value to a single bit, a 1 or a 0. The coded block and the statistical moments are then transmitted.

Illustration 1. Blocking an Image Frame Is the First Step in the BTC Process.

Figure 1 shows the flow diagram of the coding process, where N is the total number of blocks the image is divided into, row and $column$ are the respective row and column within the current block, $xsize$ is the total number of rows and $ysize$ is the total number of columns within the block, and $pixel [row,column]$ is the pixel value at location $[row,column]$ within the block.

At the receiver, calculations, involving the transmitted moments and thus preserving the original image block moments, are performed to obtain quantized values of the coded image block. Two values will be calculated, one for the pixels set to a 1, one for the pixels set to a 0. The coded image block pixels are then set to either one of the two calculated values. Figure 2 shows the flow diagram of the reconstruction process, where the variables are identical to those defined for Figure 1.

Block Truncation Coding techniques applied to actual images produce reconstructions that are close facsimiles of the original images as long as the blocking size is kept very small. Thus, the transmission bandwidth is not as small as one might like, or need, even though it is still less than that of transmitting the original image without applying a coding technique. The blocking effect is also visible in the reconstructed Block Truncation Coded images.

The efforts of trying to improve upon the current BTC techniques, by reducing the visual blocking effect in the reconstructed images and increasing the acceptable blocking size used within the method and thus decreasing the transmission bandwidth, are presented in this thesis.

Figure 1. The Block Truncation Coding Algorithm.

Figure 2. Block Truncation Coding Reconstruction Algorithm.

II. CURRENT CODING TECHNIQUES

The underlying design objective of all coding techniques is to represent and transmit images in as little information as possible and still maintain a usable picture. Block Truncation Coding (BTC) is a method of data compression that divides an image into blocks of data. Normally, statistical moments of each block are used as quantizers [2-5]. The pixels within each block are then reduced to a single bit depending upon its comparison with one of the quantizers, namely the block mean. The encoded image blocks and the quantizers for each are then transmitted. At the receiver, the image blocks are decoded using the transmitted quantizers as reconstruction parameters to preserve as much information as possible from the original image.

Block Truncation Coding

The first BTC method, proposed by Delp and Mitchell in 1979 [2], uses the mean and standard deviation of an image block as its quantizers. The mean and standard deviation of each block are calculated as follows:

$$\mu = 1/m \sum_{i=1}^m x_i \quad (1)$$

$$\sigma = \left[1/m \sum_{i=1}^m (x_i - \mu)^2 \right]^{1/2} \quad (2)$$

where m is the total number of pixels in the block and x is the value of each pixel. For color images, equations (1) and (2) must be performed for each RGB (red, green, blue) color components of each pixel value, x . Monochrome images do not have separate color components per pixel. Their pixel values correspond to a gray level shade between black and white; thus equations (1) and (2) are performed once per image

block. For the remainder of Chapters II and III, we will assume that the discussions involve monochrome images. In Chapter IV, these BTC techniques will be applied to color images.

After calculating the mean and standard deviation, each pixel in the block is then coded based upon its value as compared to the block mean value. Every pixel greater than or equal to the mean is set to 1; all others are set to 0, thus reducing the image to a single bit for each pixel. This coded image, along with the mean values and standard deviations, is then transmitted.

The image block(s) is then reconstructed using a two-level quantizer that preserves the block mean and variance, calculated as such:

$$a = \mu - \sigma \left[\frac{q + (m - q)}{2} \right] \quad , x_i < \mu \quad (3)$$

$$b = \mu + \sigma \left[\frac{(m - q) + q}{2} \right] \quad , x_i \geq \mu \quad (4)$$

where q is the number of pixels greater than or equal to the block mean, μ (i.e., the number of pixels set to 1). The pixels whose values equal a zero are assigned to the calculated low quantizer value, a , and the one valued pixels are assigned to the high quantizer value, b .

Absolute Moment Block Truncation Coding

Since the original paper by Delp and Mitchell, several papers have been published implementing the BTC using moment-preserving quantizers [3-5]. Absolute Moment BTC (AMBTC), presented by Lema and Mitchell [4], preserves the absolute moments rather than the standard moment as in the BTC. The image to be transmitted is once again divided into blocks, the block sample mean is calculated as in (1) and the standard deviation is replaced by the absolute moment:

$$c = 1/m \sum_{i=1}^m |x_i - \mu| \quad (5)$$

The block is quantized and transmitted as before. At the receiver the image is reconstructed using the following equations to replace the 0 value pixels and the 1 value pixels, respectively.

$$a = \mu - \left[mc + [2(m-q)] \right] \quad , x_i < \mu \quad (6)$$

$$b = \mu + \left[mc + 2q \right] \quad , x_i \geq \mu \quad (7)$$

Entropy-Preserving Block Truncation Coding

Entropy-Preserving BTC, as presented by Tzannes, Thacker, and Tzannes [5], preserves the mean and the entropy of the image block. The image is treated as a non-normalized probability mass function, where the individual pixels are non-negative. Again the image block mean (1) and the block entropy are calculated. The block entropy is given by

$$H = -1/m \sum_{j=1}^M \sum_{k=1}^N f(j,k) \log_2 f(j,k) \quad (8)$$

where m is the total number of pixels in the block, M is the height of the block, N is the width of the block, and $f(j,k)$ is the value of the pixel in the j th row and k th column of the block. Thus, equation (1), calculation of the mean, may now be written as

$$\mu = 1/m \sum_{j=1}^M \sum_{k=1}^N f(j,k) \quad (9)$$

which represents the normalization factor of the image. The image block is quantized as in the BTC and is transmitted along with the mean and entropy.

At the receiving end, the image will be reconstructed using b to replace the high level pixels and a the low level pixels.

$$a = \frac{m\mu - qb}{(m - q)}, \quad x_i < \mu \quad (10)$$

$$f(b) = (qb - m\mu) \log_2 \left(\frac{m\mu - qb}{(m - q)} \right) - qb \log_2 b - mH, \quad x_i \geq \mu \quad (11)$$

The zero crossings of equation (11) correspond to the solution for b . It can be shown, by taking the derivative of $f(b)$ and setting it equal to zero, that the minimum of $f(b)$ always occurs when $b = \mu$. Thus, the zero crossing that is greater than μ is the needed solution for b . Once b is found, it may be substituted into equation (10) to solve for a .

The method proposed in this paper calculates the range of the pixel values within the image block, divides it by 3, thus separating the block into 3 equal sections. The block is then coded using a three level quantizer: high, mid, and low values. A value is assigned for each of the three sections, thus the name Three-Level BTC.

III. THREE-LEVEL BLOCK TRUNCATION CODING

To reduce the blocking effect produced by the traditional Block Truncation Coding techniques, a new method, Three-Level Block Truncation Coding (3LBTC), employing a three-level quantizer is introduced. The coding and reconstruction process, described step-by-step below, are shown in flow chart format in Figures 3 and 4, respectively.

The Method

An image to be coded is first divided into $n \times m$ pixel blocks, as shown in Illustration 2. Assume that the image is 40 pixels wide and 30 pixels high and is divided into 4×5 pixel blocks. The pixel values range from 0 to 255, where 0 is white and 255 is black.

Illustration 2. Blocking an Image Frame Is Also the First Step in the 3LBTC Process.

For each of the pixel blocks created, the range of the pixel values within the block is calculated and divided by three, subdividing the original image block into thirds. Using the second row, first column pixel block in Illustration 2 as an example, the individual pixel values are shown in Figure 5.

Figure 3. The Three-Level Block Truncation Coding Process.

Figure 4. The Three-Level BTC Reconstruction Process.

per pixel. Figure 7 shows the sample block in its coded format. The coded block and the highest (HI) and lowest (LO) values are transmitted.

1	1	1	0
1	1	1	0
1	1	0	0
1	1	0	0
2	1	2	2

Figure 7. The Section in Which the Individual Image Block Pixels Lie Determine Their Coded Value.

At the receiver, the range and high and low "mean" values are again calculated as in (12), (13), and (14). The image is reconstructed using the following equations:

$$b = (hi_n + HI [pixel[j,k]]) / 2 \quad , \quad pixel[j,k] \geq hi_n \quad (15)$$

$$a = (lo_n + LO [pixel[j,k]]) / 2 \quad , \quad pixel[j,k] \leq lo_n \quad (16)$$

$$mid = (hi_n + lo_n) / 2 \quad , \quad lo_n < pixel[j,k] < hi_n \quad (17)$$

where the highest quantized values are set equal to b, the lowest quantized values set equal to a, and the middle values set equal to mid. For the sample image block, the equations yield 213, 43, and 128 for b, a, and mid respectively. The reconstructed sample image block is shown in Figure 8.

127	127	127	42
127	127	127	42
127	127	42	42
127	127	42	42
212	127	212	212

Figure 8. The Reconstructed Image Block Is Similar to the Original.

Variations of Three-Level Block Truncation Coding

As an extra measure of assurance that the best method was chosen for the 3LBTC, variations of the new method were executed and compared with the original in terms of several mathematical performance areas: compression ratios, mean-squared-error results, entropy results, and the overall processing time required for each technique.

Variations of the 3LBTC utilizing the absolute moment were tested. The range, high mean and low mean are calculated for each of these variations as in the original in equations (12), (13), and (14). Each of the absolute moment variations calculate a high and low absolute moment. They are distinct from each other in the manner in which they calculate the moments. The variation known as *high/low*, whose coding flow chart is shown in Figure 9, uses the high and mid range sections in calculating the high absolute moment, and the mid and low range sections to calculate the low absolute moment. Thus the mid section is used in both calculations, as shown in Figure 10.

Figure 10. The Mid Section Is Used to Calculate Both Absolute Moments.

Equations (18) and (19) show the high and low absolute moment calculations, respectively:

$$\begin{array}{c} \text{ROW COLUMN} \\ \text{hi_c} = 1/n\text{hi} \sum_{j=1} \sum_{k=1} |\text{pixel}[j,k] - \text{hi_n}|, \text{pixel}[j,k] > \text{lo_n} \end{array} \quad (18)$$

$$\begin{array}{c} \text{ROW COLUMN} \\ \text{lo_c} = 1/n\text{lo} \sum_{j=1} \sum_{k=1} |\text{pixel}[j,k] - \text{lo_n}|, \text{pixel}[j,k] < \text{hi_n} \end{array} \quad (19)$$

Figure 9. The High/Low Coding Algorithm Is Similar to the 3LBTC.

where hi_n and lo_n are the range high and low mean values calculated in (13) and (14), and nhi and nlo are the total number of pixels greater than lo_n and less than hi_n , respectively.

The *mid/low* variation uses the mid range section only in conjunction with the low range section in calculation of the low absolute moment. The high range section is used to calculate the high absolute moment. The *mid/low* coding algorithm is shown in flow chart format in Figure 11. Figure 12 pictorially displays this process.

Figure 12. The Mid Section Is Used to Calculate the Low Absolute Moment.

The calculations for the high and low absolute moments are shown in equations (20) and (21), respectively.

$$hi_c = \frac{1}{nhi} \sum_{j=1}^{ROW} \sum_{k=1}^{COLUMN} |pixel[j,k] - hi_n|, \quad pixel[j,k] \geq hi_n \quad (20)$$

$$lo_c = \frac{1}{nlo} \sum_{j=1}^{ROW} \sum_{k=1}^{COLUMN} |pixel[j,k] - lo_n|, \quad pixel[j,k] < hi_n \quad (21)$$

Here, nhi is the total number of pixels whose values are greater than or equal to the high mean, and nlo is the number of pixels less than the high mean, hi_n and lo_n are the range high and low mean values calculated in (13) and (14).

Figure 11. The Mid/Low Coding Algorithm Is a Variation of the 3LBTC.

Figure 13. The Mid/High Coding Algorithm Is Similar to the Mid/Low.

Two other variations of the 3LBTC utilize not only the absolute moment values, but also the mean calculation, shown in equation (1). The first, *mean-splits-mid*, uses the mean of the image block as the dividing point in the mid range section for the calculation of the high and low absolute moment. Its coding algorithm is depicted in Figure 15. If a pixel is greater than or equal to the mean, it is used in the calculation of the high absolute moment. If it is less than the mean, it is used to calculate the low absolute moment, as shown in Figure 16.

Figure 16. The Block Mean Separates the Pixels for Calculating the High and Low Absolute Moments.

Thus, the high and low absolute moment calculations, respectively, are :

$$\text{hi_c} = \frac{1}{n_{hi}} \sum_{j=1}^{\text{ROW}} \sum_{k=1}^{\text{COLUMN}} |\text{pixel}[j,k] - \text{hi_n}| \quad , \quad \text{pixel}[j,k] \geq \mu \quad (24)$$

$$\text{lo_c} = \frac{1}{n_{lo}} \sum_{j=1}^{\text{ROW}} \sum_{k=1}^{\text{COLUMN}} |\text{pixel}[j,k] - \text{lo_n}| \quad , \quad \text{pixel}[j,k] < \mu \quad (25)$$

where n_{hi} is the total number of pixels greater than or equal to the mean, μ , and n_{lo} is the total number of pixels less than μ .

The *high/low mean* variation, whose coding flow chart is shown in Figure 17, does not calculate the range of the pixel block; instead it calculates the mean and uses all pixels greater than or equal to it to calculate a "high mean", $\text{hi_}\mu$, and all pixels less than it to calculate a "low mean", $\text{lo_}\mu$. Thus the high and low means separate the pixel

Figure 15. The Mean-Splits-Mid Coding Algorithm Divides the Mid Section.

Figure 17. The High/Low Mean Coding Algorithm Uses the Mean Instead of the Range.

block into thirds, the upper third being those pixels greater than or equal to $hi_μ$, the lower third being those pixels less than or equal to $lo_μ$, and the mid all pixel values in between, illustrated in Figure 18.

Figure 18. The Block Mean Divides the Pixels.

The calculations for the high and low means are shown in equations (26) and (27), respectively. The calculations for the high and low absolute moments are shown in equations (28) and (29), respectively.

$$hi_μ = \frac{1}{nhi} \sum_{j=1}^{ROW} \sum_{k=1}^{COLUMN} pixel[j,k] \quad , \quad pixel[j,k] \geq \mu \quad (26)$$

$$lo_μ = \frac{1}{nlo} \sum_{j=1}^{ROW} \sum_{k=1}^{COLUMN} pixel[j,k] \quad , \quad pixel[j,k] < \mu \quad (27)$$

$$hi_c = \frac{1}{nhi} \sum_{j=1}^{ROW} \sum_{k=1}^{COLUMN} |pixel[j,k] - hi_μ| \quad , \quad pixel[j,k] \geq \mu \quad (28)$$

$$lo_c = \frac{1}{nlo} \sum_{j=1}^{ROW} \sum_{k=1}^{COLUMN} |pixel[j,k] - lo_μ| \quad , \quad pixel[j,k] < \mu \quad (29)$$

Again, nhi is the total number of pixels whose value is greater than or equal to the mean, $μ$, and nlo is the total number of pixels whose value is less than $μ$.

Figure 19. The 3LBTC Variation Reconstruction Process Is Similar to the AMBTC Process.

Figure 19. Continued.

133	133	133	3
133	133	133	3
133	133	3	3
133	133	3	3
290	133	290	290

Figure 21. *High/Low* Reconstructed Sample Image Block.

144	144	144	3
144	144	144	3
144	144	3	3
144	144	3	3
213	144	213	213

Figure 22. *Mid/Low* Reconstructed Sample Image Block.

122	122	122	3
122	122	122	3
122	122	3	3
122	122	3	3
213	122	213	213

Figure 23. *Mid/High* Reconstructed Sample Image Block.

137	137	137	7
137	137	137	7
137	137	7	7
137	137	7	7
238	137	238	238

Figure 24. *Mean-Splits-Mid* Reconstructed Sample Image Block.

70	70	70	0
70	70	70	0
70	70	0	0
70	70	0	0
212	70	212	212

Figure 25. *High/Low Mean* Reconstructed Sample Image Block.

Compression Ratio

To calculate the compression ratio of each of the BTC methods, we first need to determine the transmission bandwidth, i.e., the number of bits per pixel required to represent the coded image. This constitutes the number of bits required to transmit each pixel in the coded image block and the bits required for the mean and the moments transmitted with each block. The typical transmission bandwidth for each of the 3LBTC variations is shown in Table 1 in terms of bits/pixel, assuming that the image is divided into 4×4 pixel blocks.

The results of Table 1 show that the original 3LBTC has a smaller transmission bandwidth, at 3 bits/pixel, than the variations, all at 4 bits/pixel. Table 2 shows the results of converting the transmission bandwidth values to image compression ratios, assuming that the original image requires 8 bits/pixel for transmission. As was expected, the original 3LBTC performed better than the variations with only a 0.67 lead.

TABLE 2

THE COMPRESSION RATIOS OF THE ORIGINAL 3LBTC AND ITS TECHNIQUE VARIATIONS.

COMPRESSION RATIOS		
<u>METHOD</u>	<u>FORMULA</u>	<u>RATIO</u>
3LBTC	$8/2.69 = 2.98$, $8/2.94 = 2.72$	2.98:1, 2.72:1
high/low	$8/3.06 = 2.61$, $8/3.375=2.37$	2.61:1, 2.37:1
mid/low	$8/3.06 = 2.61$, $8/3.375=2.37$	2.61:1, 2.37:1
mid/high	$8/3.06 = 2.61$, $8/3.375=2.37$	2.61:1, 2.37:1
mean-splits-mid	$8/3.06 = 2.61$, $8/3.375=2.37$	2.61:1, 2.37:1
high/low mean	$8/3.25 = 2.46$, $8/3.875=2.07$	2.46:1, 2.07:1

TABLE 1

THE ORIGINAL 3LBTC REQUIRES LESS BANDWIDTH THAN THE VARIATIONS.

TRANSMISSION RESULTS		
<u>METHOD</u>	<u>FORMULA</u>	<u>BITS/PIXEL</u>
3LBTC	$\frac{N_{hi} + N_{lo} + 2m}{m}$	2.688 - 2.938
high/low	$\frac{N_{hi} + N_{lo} + 2N_c + 2m}{m}$	3.063 - 3.375
mid/low	$\frac{N_{hi} + N_{lo} + 2N_c + 2m}{m}$	3.063 - 3.375
mid/high	$\frac{N_{hi} + N_{lo} + 2N_c + 2m}{m}$	3.063 - 3.375
mean-splits-mid	$\frac{N_{hi} + N_{lo} + 2N_c + 2m}{m}$	3.063 - 3.375
high/low mean	$\frac{N_{hi\mu} + N_{lo\mu} + 2N_c + 2m}{m}$	3.250 - 3.875

where,

N_c	= # of bits required for the absolute moment	(3 - 7) bits
N_{hi}	= # of bits required for the highest pixel value	(7 - 8) bits
N_{lo}	= # of bits required for the lowest pixel value	(4 - 7) bits
$N_{hi\mu}$	= # of bits required for the high mean value	(7 - 8) bits
$N_{lo\mu}$	= # of bits required for the low mean value	(7 - 8) bits
m	= total # of pixels in the image block	(16) bits

Mean-Square-Error

Another measure of coding quality is the mean-square-error (MSE) calculation. The MSE is calculated by summing the difference squared of the reconstructed and the original image pixels and dividing the total sum by the total number of pixels, as shown in equation (34).

$$\text{MSE} = \frac{1}{M} \sum_{j=1}^{\text{ROW}} \sum_{k=1}^{\text{COLUMN}} (\text{pixel}[j,k] - \text{recon}[j,k])^2 \quad (34)$$

where M is the total number of pixels in the image, ROW is the total number of image pixels rows, COLUMN is the total number of image pixel columns, pixel[j,k] is the value of the original image pixel in the jth row kth column, and recon[j,k] is the value of the reconstructed image pixel in the jth row kth column.

To see how well the 3LBTC technique variations compare in terms of mean-square-error (MSE) results, each method was applied to eleven sample 4 by 4 image blocks. The FORTRAN 77 code, in Appendix A, produced the results of the reconstructions found in Appendix B. Table 3 is a summary of the MSE results of the reconstructions.

TABLE 3
THE 3LBTC MATHEMATICALLY OUTPERFORMS
THE VARIATIONS IN TERMS OF MSE.

MEAN-SQUARE-ERROR SUMMARY					
3LBTC	HIGH/LOW	MID/LOW	MID/HIGH	MEAN-SPLITS-MID	HIGH/LOW MEAN
805.6	909.1	1167.	1090.	1243.	1234.
4.000	4.500	5.500	5.500	4.750	6.063
3.500	5.000	5.000	6.438	6.438	2.000
6.500	6.750	6.875	6.750	6.875	7.438
2.750	2.750	2.750	3.500	3.500	2.500
2.500	3.500	3.500	4.813	2.938	6.063
312.3	294.9	404.4	373.6	400.2	240.2
650.0	559.5	809.8	728.7	802.5	750.0
515.4	859.2	889.8	950.9	690.6	635.3
448.5	386.0	594.7	482.2	533.2	567.2
0000	.0000	.0000	.0000	.0000	.0000

The results of Table 3 show that the original 3LBTC outperformed the *high/low* and *high/low mean* variations eight out of eleven times and eleven out of eleven times better than the *mid/low*, *mid/high*, and *mean-splits-mid*. The MSE results are even more striking when you realize that the original 3LBTC has a larger compression ratio than the variations; one would have expected the variations to have outperformed the original 3LBTC.

Entropy

Entropy is defined as the amount of information stored within an image. The equation to calculate the entropy, shown previously in equation (8), is

$$H = -1/m \sum_{j=1}^M \sum_{k=1}^N \text{pixel}[j,k] \log_2 \text{pixel}[j,k] \quad (35)$$

Where m is the total number of pixels, M is the number of rows in the image, N is the number of columns, and $\text{pixel}[j,k]$ is the value of the pixel at location (j,k) .

The entropy of the 3LBTC reconstructions and its technique variation reconstructions is compared to the entropy of the original eleven sample image blocks, found in Appendix B. The variation technique with the reconstruction entropy value closest to the original sample, shown in Table 4, is *high/low mean* four out of the eleven samples, second is *mid/high* and *mean-splits-mid* with three, third is *mid/low* with two, and last is *high/low* with one. The *mid/low* variation had the highest entropy six times out of the eleven samples, the *high/low mean* was second with 3 samples highest, the *high/low* a close third with two, the *mid/high* fourth with one. Recall that the variation techniques have a smaller compression ratio than the original 3LBTC.

TABLE 4

ENTROPY COMPARISON OF ELEVEN RECONSTRUCTED IMAGE BLOCKS.

ENTROPY SUMMARY						
<u>IMAGE</u>	<u>3LBTC</u>	<u>HI/LO</u>	<u>MID/LO</u>	<u>MID/HI</u>	<u>MEAN-SPLITS-MID</u>	<u>HI/LO MEAN</u>
-721.8	-703.4	-776.1	-761.1	-788.7	-769.5	-774.3
-428.5	-425.5	-421.8	-419.8	-425.5	-423.6	-429.8
-430.3	-428.4	-422.7	-422.7	-423.1	-423.1	-431.2
-441.7	-438.8	-442.5	-443.5	-442.5	-443.5	-450.7
-466.4	-465.4	-463.5	-465.4	-465.4	-465.4	-462.6
-451.1	-447.3	-443.6	-444.5	-447.7	-444.9	-439.2
-1220.	-1191.	-1188.	-1159.	-1212.	-1186.	-1224.
-1038.	-1109.	-1049.	-1013.	-1060.	-1034.	-1012.
-660.2	-699.5	-810.7	-764.6	-822.2	-787.0	-644.1
-1094.	-1097.	-1112.	-1063.	-1129.	-1089.	-1147.
-2037.	-2037.	-2037.	-2037.	-2037.	-2037.	-2037.

Processing Time

A Macintosh SE, equipped with Novy Systems 25 MHz Accelerator card, was used to execute the FORTRAN 77 3LBTC Variations Comparison program included as Appendix A. The greatest precision achievable to time each of the procedures was one sixtieth of a second (1/60 sec). Two sets of timing runs were executed. The first calculated the time required to process each sample image block separately. The second calculated the time required to process all eleven sample image blocks as a single file. Input/Output processing to the results file was not included in the processing time. Of the eleven sample image blocks run, the 3LBTC processed them the quickest at less than 1/60 second each; it produced an execution time of zero which meant that the timing clock had not yet reached 1/60 of a second. The 3LBTC technique variations each averaged a processing time of 1/60 second for the individual sample image blocks. The 3LBTC produced the fastest processing time again, with 4/60 of a second, when timing all eleven samples as a single run. The *mid/low*, *mid/high*, and *mean-splits-mid* variations again had the second fastest processing time of 8/60 of a second. The

high/low variation finished third with 9/60 of a second and the *high/low mean* variation finished last with 10/60 of a second. Table 5 summarizes the processing time results.

TABLE 5

3LBTC HAS THE FASTEST PROCESSING TIME
OF ITS TECHNIQUE VARIATIONS.

PROCESSING TIME		
<u>METHOD</u>	<u>EACH SAMPLE</u>	<u>11 SAMPLES COMBINED</u>
3LBTC	<1/60 second	4/60 second
HIGH/LOW	1/60 second	9/60 second
MID/LOW	1/60 second	8/60 second
MID/HIGH	1/60 second	8/60 second
MEAN-SPLITS-MID	1/60 second	8/60 second
HIGH/LOW MEAN	1/60 second	10/60 second

The results of all the mathematical comparisons performed using the 3LBTC and its technique variations, proved the original 3LBTC technique to be the superior method. Not only does it have a greater compression ratio, but it also outperformed the variation techniques in the MSE comparison tests and has a processing speed 1/2 that of the next closest variation technique. Thus, the Three-Level Block Truncation Coding, 3LBTC, method chosen has proven to be the superior of those tested.

IV. CURRENT CODING TECHNIQUE COMPARISONS

Each of the Block Truncation Coding techniques discussed previously is compared in terms of mathematical performance in several areas: compression ratios, mean-squared-error results, entropy results, the effects of errors in the high quantizer value, b , on the low quantizer value, a , and the overall processing time required for each technique.

For each of the areas under comparison, we will assume that all images are 4×4 pixels and that the pixel values range from 0 to 255, thus requiring 8 bits per pixel representation.

Compression Ratio

The typical transmission bandwidth, i.e., the number of bits required to represent the coded image, for each of the BTC techniques is shown in Table 6 in terms of bits/pixel, assuming that the image is a 4×4 pixel block. The results of Table 6 show that the original BTC and the AMBTC will require only 2 bits/pixel for transmission, while the EBTC may sometimes require 3, whereas the 3LBTC will always require 3. Table 7 shows the results of converting these results to image compression ratios, assuming that the original image requires 8 bits/pixel for transmission. Here again, the BTC and the AMBTC yield greater compression ratios than the 3LBTC in all cases for images of 4×4 pixel blocks.

Mean-Square-Error

Each of the BTC techniques were applied to the same eleven sample image blocks, used in the Variations of Three-Level Block Truncation Coding section, and their image reconstructions compared in terms of mean-square-error, as shown previously in equation (34). These sample image blocks, along with each of the BTC methods' reconstructed image blocks, are detailed in Appendix D. The FORTRAN 77 code that produced the results in Appendix D is found in Appendix C. Table 8 is a summary of the MSE results from the FORTRAN 77 program.

TABLE 6

BTC METHODS REDUCE THE OVERALL BANDWIDTH REQUIRED
FOR IMAGE TRANSMISSION.

TRANSMISSION RESULTS		
<u>METHOD</u>	<u>FORMULA</u>	<u>BITS/PIXEL</u>
BTC	$\frac{N_{\mu} + N_o + m}{m}$	1.625 - 1.938
AMBTC	$\frac{N_{\mu} + N_c + m}{m}$	1.625 - 1.938
EBTC	$\frac{N_{\mu} + N_h + m}{m}$	2.000 - 2.188
3LBTC	$\frac{N_{hi} + N_{lo} + 2m}{m}$	2.688 - 2.938

where,

N_{μ}	= # of bits required for the mean	(7 - 8) bits
N_o	= # of bits required for the standard deviation	(3 - 7) bits
N_c	= # of bits required for the absolute moment	(3 - 7) bits
N_h	= # of bits required for the entropy	(9 - 11) bits
N_{hi}	= # of bits required for the highest pixel value	(7 - 8) bits
N_{lo}	= # of bits required for the lowest pixel value	(4 - 7) bits
m	= total # of pixels in the image block	(16) bits

TABLE 7

BTC TECHNIQUES COMPRESSION RATIO COMPARISON.

COMPRESSION RATIOS				
<u>METHOD</u>	<u>FORMULA</u>		<u>RATIO</u>	
BTC	8 / 1.625 = 4.92,	8/1.94 = 4.13	4.92:1,	4.13:1
AMBTC	8 / 1.625 = 4.92,	8/1.94 = 4.13	4.92:1,	4.13:1
EBTC	8 / 2.000 = 4.00,	8/2.19 = 3.66	4.00:1,	3.66:1
3LBTC	8 / 2.688 = 2.98,	8/2.94 = 2.72	2.98:1,	2.72:1

TABLE 8

MSE RESULTS OF THE RECONSTRUCTED IMAGES
AS COMPARED TO THE ORIGINALS.

MEAN-SQUARE-ERROR SUMMARY								
<u>IMAGE SAMPLE</u>	<u>3LBTC (CR)</u>		<u>BTC (CR)</u>		<u>AMBTC (CR)</u>		<u>EBTC (CR)</u>	
1	805.6	(3.13:1)	1684.	(4.28:1)	1605.	(4.28:1)	1660.	(3.88:1)
2	4.000	(2.85:1)	8.000	(4.73:1)	7.250	(4.73:1)	7.500	(4.00:1)
3	3.500	(2.85:1)	9.688	(4.73:1)	9.313	(4.73:1)	9.688	(4.00:1)
4	6.500	(2.85:1)	8.250	(4.73:1)	7.500	(4.73:1)	7.875	(4.00:1)
5	2.750	(2.78:1)	6.000	(4.73:1)	5.438	(4.73:1)	6.000	(4.00:1)
6	2.500	(2.85:1)	7.750	(4.73:1)	7.000	(4.73:1)	7.250	(4.00:1)
7	312.3	(2.72:1)	723.5	(4.27:1)	673.5	(4.27:1)	723.5	(3.66:1)
8	650.0	(2.91:1)	1550.	(4.12:1)	1450.	(4.12:1)	1675.	(3.66:1)
9	515.4	(2.85:1)	1525.	(4.41:1)	1370.	(4.41:1)	1459.	(3.88:1)
10	448.5	(2.78:1)	933.1	(4.27:1)	876.1	(4.27:1)	967.1	(3.66:1)
11	.0000	(2.67:1)	.0000	(5.13:1)	.0000	(5.13:1)	.0000	(3.66:1)

The results in Table 8 show that the 3LBTC has the smallest MSE in all the test cases. This is to be expected since Table 7 shows that it has the smallest compression ratio, in most cases, hence its reconstruction should be closer to the original. The compression ratio of each technique's reconstructed image is included within parenthesis

next to the MSE value. Only those MSEs having like compression ratios (CR) should be compared to each other.

From Table 8, we can see that the AMBTC has the lowest MSE of all the reconstructed sample image blocks with a compression ratio greater than 4:1. The 3LBTC is far superior, by a factor of at least two, than the two level BTC techniques for all the reconstructed sample image blocks. Recall however, that the 3LBTC has the smallest compression ratio and therefore should produce the lowest MSE.

Entropy

Entropy is defined as the amount of information stored within an image. Table 9 compares the entropy of the original eleven sample image blocks, found in Appendix D, with the entropy of the each of the BTC techniques' reconstructed images. Here again, we must compare reconstructed images with like compression ratios.

The results of the reconstructed image blocks with a compression ratio greater than or equal to 4:1 show that in most cases the BTC's entropies are closest to the entropies of the original sample image blocks, whereas the EBTC's reconstructed images have the largest entropy value. The reconstructed EBTC images, with a 3.66:1 compression ratio, have entropies that are much closer to the originals' than the 3LBTC.

Since the EBTC technique is intended to preserve the entropy of the original image, it should produce entropies closer to the originals than the other techniques. For those samples where its compression ratio is comparable to the 3LBTC, its entropy is closer to the original sample. For those samples where its compression ratio is comparable to those of the BTC and AMBTC, its compression ratio was not the closest to the original sample. However, it produced larger entropies than the other methods and the original samples. One possible explanation for the difference in entropies between the EBTC and the original images is the error produced in the floating point calculations, i.e., $\log_2$, b , and H , which are all floating point variables, required for the EBTC technique. The following subsection researches this possibility.

TABLE 9

THE RECONSTRUCTED IMAGES ENTROPY RESULTS AS COMPARE TO THE ORIGINALS.

ENTROPY SUMMARY					
IMAGE ENTROPY	3LBTC (CR)	BTC (CR)	AMBTC (CR)	EBTC (CR)	
1 -721.8	-703.4(3.13:1)	-723.9(4.28:1)	-710.3(4.28:1)	-719.0(3.88:1)	
2 -428.5	-425.5(2.85:1)	-429.4(4.73:1)	-429.3(4.73:1)	-425.6(4.00:1)	
3 -430.3	-428.4(2.85:1)	-430.8(4.73:1)	-429.7(4.73:1)	-430.8(4.00:1)	
4 -441.7	-438.8(2.85:1)	-440.8(4.73:1)	-442.6(4.73:1)	-437.9(4.00:1)	
5 -466.4	-465.4(2.78:1)	-464.5(4.73:1)	-467.8(4.73:1)	-464.5(4.00:1)	
6 -451.1	-447.3(2.85:1)	-454.0(4.73:1)	-450.1(4.73:1)	-448.2(4.00:1)	
7 -1220.	-1191.(2.72:1)	-1219.(4.27:1)	-1217.(4.27:1)	-1219.(3.66:1)	
8 -1038.	-1109.(2.91:1)	-1033.(4.12:1)	-1025.(4.12:1)	-1038.(3.66:1)	
9 -660.2	-699.5(2.85:1)	-661.3(4.41:1)	-653.8(4.41:1)	-658.3(3.88:1)	
10 -1094.	-1097.(2.78:1)	-1092.(4.27:1)	-1088.(4.27:1)	-1091.(3.66:1)	
11 -2037.	-2037.(2.67:1)	-2037.(5.13:1)	-2037.(5.13:1)	-2037.(3.66:1)	

The Effects of Errors in b on a

We have seen in Section II, Current Coding Techniques, that the reconstruction of a transmitted block truncation coded image depends on the calculations of a high and a low quantizer. The only BTC method whose low quantizer, a, depends on the value of its high quantizer, b, is the Entropy-Preserving BTC, as shown below and previously in equation (11).

$$a = \frac{m\mu - qb}{(m - q)}, \quad x_i < \mu \quad (36)$$

$$f(b) = (qb - m\mu) \log_2 \left(\frac{m\mu - qb}{(m - q)} \right) - qb \log_2 b - mH, \quad x_i \geq \mu \quad (37)$$

To see what effect errors in b have on a, I have taken the second and sixth image block samples and selectively introduced errors of 0.01, 0.1, 1.0, 5.0, and 10.0 into the value calculated for b. For example, in the second sample the value of b was calculated to be 75.69 and the corresponding value for a was 64.06. When an error of

0.01 was added to b , hence $b=75.70$, a was calculated to be 64.05 by substituting the new value of b into equation (36). When an error of 5.0 was added to b , $b=80.69$, a was calculated to be 59.06. Originally these samples produced reconstructed images where the $f(b)$ in equation (37) was found to be exactly zero for the calculated value of b . As seen in Figure 26, when the value of b is increased the value of a decreases.

Figure 26. Slight Errors in b Do Not Affect the Reconstruction.

As you can see from the graphs in Figure 26, slight errors in the calculation of b do not affect the calculation of a , and therefore the reconstructed image is not affected. Larger errors will, however, affect the reconstruction as shown in the resulting MSEs in Figure 27 and the entropies shown in Figure 28. The original value of b , where no errors were introduced, is depicted by the 0.0 tick mark on the horizontal axis.

Figure 27. Large Errors in b Affect the Reconstruction MSE.

Figure 28. Large Errors in b Affect the Reconstruction Entropy.

From the graphs in Figures 26 through 28, we can see that errors in the calculation of b greater than 1.0 affect the resulting MSE and entropy calculations of the EBTC reconstructed images. Therefore, it is possible that floating point arithmetic errors in the calculation of b could account for the difference in entropy values of the resulting reconstructed EBTC images and the original images.

Processing Time

A Macintosh SE, equipped with Novy Systems 25 Mhz Accelerator card, was used to execute the FORTRAN 77 Current Coding Techniques Comparison program included as Appendix A. The greatest precision that this author could use to time each of the procedures was one sixtieth of a second (1/60 sec). Two sets of timing runs were executed. The first calculated the time required to process each sample image block separately. The second calculated the time required to process all eleven sample image blocks as a single file. Input/Output processing to the results file was not included in the processing time. Of the eleven sample image blocks run, the 3LBTC processed them the quickest at less than 1/60 second each; it produced an execution time of zero which meant that the timing clock had not yet reached 1/60 of a second. The BTC and the AMBTC each averaged a processing time of 1/60 second for each individual sample; some of their samples had execution times less than 1/60 second, a few others were timed at 2/60 second. The EBTC showed the slowest processing time of an average of 3/60 second for each of the sample image blocks. The 3LBTC produced the fastest processing time again, with 4/60 of a second, when timing all eleven samples in a single run. The AMBTC was a close second, with a total processing time of 5/60 of a second. The BTC finished third with 8/60 of a second. The EBTC, last with 32/60 of a second, was well outperformed by the other three methods. Input/Output processing to the results file was not included in the processing time. Table 10 summarizes the processing time results.

TABLE 10

3LBTC HAS THE FASTEST PROCESSING TIME OF
THE BTC TECHNIQUES COMPARED.

PROCESSING TIME		
<u>METHOD</u>	<u>EACH SAMPLE</u>	<u>11 SAMPLES COMBINED</u>
BTC	1/60 second	8/60 second
AMBTC	1/60 second	5/60 second
EBTC	3/60 second	32/60 second
3LBTC	<1/60 second	4/60 second

V. APPLICATIONS TO ACTUAL IMAGES

The Three-Level BTC, the original BTC, AMBTC, and EBTC were simulated using Turbo PASCAL to code and reconstruct actual color images. An IBM PC, equipped with AT&T's Image Capture Board (ICB), a SONY color RGB monitor, and a SHINKO CHC-355 color thermal printer, was used to display and print the images used. Photographs of the SHINKO printer outputs are included in this section. Had the IBM and its specialized hardware not been in a secure environment, photographs of the RGB monitor would have been taken instead.

The display memory size of AT&T's ICB is 200 rows by 256 columns, thus restricting the images used to 200 x 256 pixels. The AT&T's ICB display memory pixel word specification is shown in Figure 12.

Figure 29. Each RGB Component Has a Maximum Value of 31.

As you can see the red, green, and blue components are each 5 bits in length, limiting each color to a maximum value of 31. An RGB combination of all zeroes is black; all ones is white. Each pixel color component was coded separate from the others. This is not the most efficient technique, in terms of bandwidth, but it will work for both color and monochromatic images. An application using transformations of the RGB color components to NTSC components, YIQ, is presented by M.D. Lema and O.R. Mitchell [3].

Blocking sizes of 4 x 4, 5 x 8, 10 x 16, and 20 x 32 pixels were used in each BTC technique for four color images. Two of the images, CAT and CARTOON, are demos from AT&T. The last two, BABY and DEER, were two monochromatic posters that were digitized using AT&T's ICB into the format shown in Figure 12. These original images are shown in Illustration 3.

Illustration 3. The Original Images as Seen at the Transmission Side.

The original Block Truncation Coding method was first applied to the images in Illustration 3. Illustrations 4 through 7 show the results of this technique using various blocking sizes: 4×4 , 5×8 , 10×16 , and 20×32 . The reconstructed images using the 4×4 pixel blocks resemble the originals very closely. However, the blocking effect is evident along the edges, such as the cat's nose and eyes. Minute details, such as the cartoon girl's eyes, have also lost their sharpness. The blocking effect is a bit more pronounced in the 5×8 pixel block reconstructed images, but the images are still readily distinguishable. The 10×16 pixel block reconstructions begin to look a little distorted as compared to the originals. Finally, the 20×32 pixel block reconstructed images have lost all details; the CARTOON image fairsing the worst; the DEER and CAT subjectively the best. It appears that the more detail within a pixel block the greater the distortion in the reconstructed image. This is expected since the reconstructed block has a limit of two pixel values as opposed to the $n \times m$ values in the original image block.

Illustration 4. The BTC Technique Applied to the CAT Image Produces Favorable Results.

Illustration 5. BTC Blocking Effect Is Evident in Minute Details of the CARTOON Image.

Illustration 6. The BABY Image Is Recognizable in the BTC Reconstructions.

Illustration 7. The BTC DEER Images Were Subjectively the Best Reconstructions.

Next, the Absolute-Moment Block Truncation Coding technique was applied to the original images of Illustration 3. The reconstruction results are shown in Illustrations 8 through 11. The same blocking sizes were used as with the original BTC. The AMBTC reconstructions produce results very similar to that of the BTC method; the 4 x 4 pixel block reconstructions are very close to the originals; the blocking effect is evident along the edges; minute detail is lost with larger blocking sizes; and distortion of the original is noticeably evident with the 20 x 32 blocking size. However, this author believes that the AMBTC reconstructions are a bit brighter and clearer than those of the BTC.

Illustration 8. AMBTC Reconstructions Are Similar to Those of the BTC.

Illustration 9. Minute Detail Is Lost in AMBTC Reconstructions.

Illustration 10. AMBTC Reconstructions of the BABY Produce Favorable Results.

Illustration 11. AMBTC DEER Reconstructions Closely Resemble the Originals.

The reconstruction results of applying the Entropy Preserving Block Truncation Coding technique to the images of Illustration 3 are shown in Illustrations 12 through 15. As with the AMBTC method, the EBTC produces results comparable to those of the BTC; blocking is evident along the edges and details are distorted as the blocking size is increased. Subjectively, this author still believe that the AMBTC reconstructions are the best of the three methods utilized thus far. The EBTC reconstructions appear to be darker than those of the BTC and the AMBTC.

Illustration 12. EBTC Reconstructions Are Comparable to Those of the BTC and AMBTC.

Illustration 13. EBTC Reconstructions Lose Minute Details.

Illustration 14. Distortions Occur as the EBTC Blocking Size Is Increased.

Illustration 15. EBTC DEER Reconstructions Produce Favorable Results.

Finally, the Three-Level Block Truncation Coding technique was applied to the original images of Illustration 3. The reconstruction results are shown in Illustrations 16 through 19. The same blocking sizes were used as with the previous methods. One can clearly see that the 3LBTC results are ALL very similar to the originals. The blocking effect is not noticeable until a blocking size of 20×32 is used. The 3LBTC reconstructed images also appear to be the brightest of the four BTC techniques applied. Recall that the 3LBTC has the smallest compression ratio of the techniques compared, therefore we should see reconstructions better than the previous BTC techniques.

Illustration 16. 3LBTC Reconstructions Are Closest to the Originals.

Finally, the Three-Level Block Truncation Coding technique was applied to the original images of Illustration 3. The reconstruction results are shown in Illustrations 16 through 19. The same blocking sizes were used as with the previous methods. One can clearly see that the 3LBTC results are ALL very similar to the originals. The blocking effect is not noticeable until a blocking size of 20×32 is used. The 3LBTC reconstructed images also appear to be the brightest of the four BTC techniques applied. Recall that the 3LBTC has the smallest compression ratio of the techniques compared, therefore we should see reconstructions better than the previous BTC techniques.

Illustration 16. 3LBTC Reconstructions Are Closest to the Originals.

Illustration 17. Minute Details Are Still Distinguishable
with a 3LBT 20 x 32 Blocking Size.

Illustration 18. Distortions Are Not as Evident with the 3LBTC.

Illustration 19. The 3L BTC Produces the Brightest Reconstructed Images.

Now that we have seen how the reconstructed images compare subjectively, we must compare their mathematical performance in the areas of compression ratios, mean-square-error results, entropy results, and overall processing speeds.

Compression Ratios

As we have seen in the results of the sample image blocks in Chapter IV, to accurately compare the results of the actual image applications we must compare reconstructed images of like compression ratios. The transmission bandwidth of each of the BTC techniques and each of the blocking sizes utilized is tabulated in Table 11. Recall that all the images used are 200 x 256 pixels and each pixel is comprised of 5 bits for the red component, 5 bits for the green component, and 5 bits for the blue

component. Each of the formulas in Table 11 are multiplied by three since the blocks were coded separately for each of the color components. It is interesting to note that the transmission bandwidth for each of the methods stabilized at a blocking size of 5×8 and above. As expected the 3LBTC has the largest transmission bandwidth. The BTC, AMBTC, and EBTC all stabilized at an equal transmission bandwidth, 4, starting with the 5×8

TABLE 11

THE 3LBTC METHOD HAS THE LARGEST
IMAGE TRANSMISSION BANDWIDTH.

TRANSMISSION RESULTS					
METHOD	FORMULA	BITS/PIXEL			
		4x4	5x8	10x16	20x32
BTC	$\frac{3(N_{\mu} + N_{\sigma} + m)}{m}$	3.9-4.7	3.4-3.7	3.1-3.2	3.03
AMBTC	$\frac{3(N_{\mu} + N_{\sigma} + m)}{m}$	3.9-4.7	3.4-3.7	3.1-3.2	3.03
EBTC	$\frac{3(N_{\mu} + N_h + m)}{m}$	4.7-5.4	3.7-3.9	3.2-3.3	3.05
3LBTC	$\frac{3(N_{hi} + N_{lo} + 2m)}{m}$	7.1-7.7	6.5-6.7	6.1-6.2	6.03

where,

N_{μ}	= # of bits required for the mean	(3 - 5)	bits
N_{σ}	= # of bits required for the standard deviation	(2 - 4)	bits
N_C	= # of bits required for the absolute moment	(2 - 4)	bits
N_h	= # of bits required for the entropy	(6 - 8)	bits
N_{hi}	= # of bits required for the highest pixel value	(4 - 5)	bits
N_{lo}	= # of bits required for the lowest pixel value	(2 - 4)	bits
m	= total # of pixels in the image block	(16-640)	bits

blocking size. Table 12 shows the results of converting these results to image compression ratios, assuming that the original image requires 15 bits/pixel for transmission, as shown in Figure 12, ignoring the unused bit.

TABLE 12

BTC METHODS YIELD COMPRESSION RATIOS AS GREAT AS 3.75:1.

COMPRESSION RATIOS				
<u>METHOD</u>	<u>4x4</u>	<u>5x8</u>	<u>10x16</u>	<u>20x32</u>
BTC	3.85:1, 3.19:1	4.41:1, 4.05:1	4.84:1, 4.69:1	4.95:1
AMBTC	3.85:1, 3.19:1	4.41:1, 4.05:1	4.84:1, 4.69:1	4.95:1
EBTC	3.19:1, 2.78:1	4.05:1, 3.85:1	4.69:1, 4.55:1	4.92:1
3LBTC	2.11:1, 1.95:1	2.31:1, 2.24:1	2.46:1, 2.42:1	2.49:1

As with the sample image block compression ratio results in Chapter 3, the 3LBTC has the smallest compression ratios of the BTC techniques compared, as expected. Illustrations 20 through 31 group the reconstructed images by blocking size and compression ratios. The 3LBTC reconstructed images are included by blocking size just for the sake of comparison, even though its compression ratio is not equal to the others.

Illustration 20. Reconstructed CAT Images Using A Blocking Size of 5 x 8 Produce Favorable Results.

Comparison of the reconstructed images in Illustration 20 above, reveals the original BTC, top left, and the AMBTC, top right, to be nearly identical. The AMBTC appears to be slightly lighter in color. The reconstructed EBTC, bottom left, image is not only the darkest, but its blocking effect is the most evident. The 3LBTC, bottom right, seems to be brighter than the others. Its reconstruction also seems to have sharpened the edges within the image; the whiskers and the ear hairs appear more distinct.

Illustration 21. Reconstructed CAT Images Using A 10 x 16 Blocking Size Show Evidence of the Blocking Technique.

Comparisons of the images in Illustration 21 reveal the same results of those using a blocking size of 5 x 8 in Illustration 20. The BTC, top left, and AMBTC, top right, images are nearly identical, the AMBTC being slightly lighter. The blocking effect is more evident in the background of the EBTC, bottom left, reconstruction. The 3LBTC, bottom right, reconstruction sharpens the image's edges. The blocking effect is more evident in all of the reconstructions as compared to those in Illustration 20.

Illustration 22. Reconstructed CAT Image Using A Blocking Size of 20 x 32 Are Abstracts of the Original.

Here again, the results of the reconstructed image comparison of Illustration 22 are the same as those of Illustrations 20 and 22. The AMBTC, top right, and the BTC, top left, are nearly identical. The EBTC, bottom left, reconstruction shows more evidence of background blocking and the 3LBTC, bottom right, sharpens the edges. The blocking is the most evident utilizing a blocking size of 20 x 32 for all the reconstructions. It also produces the most abstract reconstructions of the original, however the cat is still discernible.

Illustration 23. CARTOON Reconstructions Employing a Blocking Size of 5 x 8 Produce Acceptable Images.

For the reconstructions of the CARTOON in Illustration 23, the BTC, top left, appears to be the closest reconstruction to the original (see Illustration 3) of the equal compression ratio reconstructions. The AMBTC, top right, and the EBTC, bottom left, reconstructions seem about the same. The 3LBTC, bottom right, has sharper details than the others, noticeably in the girl's outline.

Illustration 24. CARTOON Reconstructions Employing A 10 x 16 Blocking Size Show Evidence of the Blocking Technique.

The results of Illustration 24 are very much the same as those for Illustration 23; however, the blocking is more evident in all the reconstructions. Subjectively the BTC reconstruction, top left, still appears to be the closest to the original of the three reconstructions with the same compression ratio. The 3LBT reconstruction, bottom right, is much sharper than the others.

Illustration 25. CARTOON Reconstructions Employing A 20 x 32 Blocking Size Are Abstracts of the Original.

Comparisons of the reconstructed images in Illustration 25 produce the same results as those of Illustrations 23 and 24. The BTC reconstruction, top left, still appears to be the best of the three like compression ratio reconstructions shown. All details in the BTC, AMBTC, and the EBTC reconstructions have been lost, the girl's face in particular. The 3LBTC reconstruction, bottom right, still has sharper edges and is brighter than the others.

Illustration 26. BABY Reconstructions Using A 5 x 8 Blocking Size Are Good Facsimile of the Original.

All of the reconstructions in Illustration 26 appear to be about the same. Looking at the edges of the ears and the nostrils, the 3LBTTC has smoother edges than the other three reconstructions. We would expect the 3LBTTC to have a clearer reconstruction since it has the smallest compression ratio.

Illustration 27. Details Are Lost in the BABY Reconstructions Utilizing a Blocking Size of 10 x 16.

Details, such as those in the baby's eyes and nose, are now lost in the BTC, AMBTC, and EBTC reconstructions in Illustration 27. The BABY image, however, is still distinguishable. As expected, the 3LBTC reconstruction is a much clearer and closer replica of the original.

Illustration 28. Abstract BABY Reconstructions Are Produced with a Blocking Size of 20 x 32.

The BTC, AMBTC, and EBTC reconstructions all appear identical, abstracts of the original, yet discernible. The 3LBTC reconstruction, has the smallest compression ratio of the four, has lost some of the details in the baby's eyes and nose, but is still much closer to the original than the other three reconstructions.

Illustration 29. DEER Reconstructions, Employing A 5 x 8 Blocking Size, Produce Excellent Replicas of the Original.

All of the DEER reconstructions, shown in Illustration 29, appear to be very similar. The 3LBTC, however, does not show as much evidence of blocking as the other three reconstructions. They are all excellent replicas of the original.

Illustration 30. Minute Details of the DEER Reconstructions Are Lost with a Blocking Size of 10 x 16.

The BTC, AMBTC, and EBTC DEER reconstructions, appearing to be identical, have lost some of the edge detail around the deer's body, as shown in Illustration 30. They are still discernible replicas of the original. The 3LBTC, due to its smaller compression ratio, has still retained the detail around the body of the deer and is closest in comparison to the original.

Illustration 31. DEER Reconstructions, Employing A 20 x 32 Blocking Size, Still Produce Distinguishable Replicas of the Original.

Even though the blocking technique is the most evident in the reconstructions employing a 20 x 32 blocking size, shown in Illustration 31, the reconstructions are good replicas of the original. All detail has been lost in the BTC, AMBTC, and EBTC reconstructions, but the deer is still very distinguishable. The 3LBTC reconstruction retains some of the details of the original, due to its small compression ratio and its three valued blocking technique.

Mean-Square-Error

The mean-square-error of all the reconstructed images was calculated, for each RGB color component, from the originals. Tables 13 through 15 summarize these results. The AMBTC technique showed the least mean-square-error for the red color component for most all of the images. The 3LBTC and the BTC both had fairly close results to the AMBTC. The 3LBTC, however, did not produce as good results for blocking sizes of 10 x 16 and 20 x 32. For the majority of the results, the EBTC had the highest MSE. These results are somewhat unexpected. Since the 3LBTC has the smallest compression ratio, one would have expected it to have the least MSE.

TABLE 13

THE QUALITY OF THE RECONSTRUCTED IMAGES
ARE MATHEMATICALLY COMPARED IN TERMS OF MSE.

MEAN-SQUARE-ERROR SUMMARY OF RED COMPONENT				
<u>IMAGE</u>	<u>3LBTC</u>	<u>BTC</u>	<u>AMBTC</u>	<u>EBTC</u>
CAT 4x4	37.83	38.62	37.06	56.91
CAT 5x8	43.49	41.98	40.02	61.98
CAT 10x16	54.97	46.66	56.46	61.37
CAT 20x32	63.63	52.02	48.90	69.54
CARTOON 4x4	22.37	23.30	22.21	27.76
CARTOON 5x8	27.31	27.57	26.33	30.22
CARTOON 10x16	39.03	30.93	31.40	33.67
CARTOON 20x32	77.68	34.09	35.36	33.34
BABY 4x4	9.925	10.28	10.28	35.24
BABY 5x8	18.73	16.51	15.34	42.40
BABY 10x16	39.41	26.18	25.27	47.18
BABY 20x32	63.35	37.63	33.26	46.84
DEER 4x4	31.49	36.93	36.19	37.53
DEER 5x8	33.57	30.90	29.56	41.12
DEER 10x16	57.88	45.50	39.44	49.32
DEER 20x32	75.10	54.47	50.01	50.10

TABLE 14

THE AMBTC HAS THE SMALLEST MSE
FOR THE GREEN RGB COLOR COMPONENT.

MEAN-SQUARE-ERROR SUMMARY OF GREEN COMPONENT				
<u>IMAGE</u>	<u>3LBTC</u>	<u>BTC</u>	<u>AMBTC</u>	<u>EBTC</u>
CAT 4x4	52.85	50.87	47.37	64.25
CAT 5x8	64.64	54.44	50.40	67.72
CAT 10x16	69.65	60.12	56.46	71.61
CAT 20x32	63.78	68.60	60.10	75.91
CARTOON 4x4	46.49	46.77	44.10	55.31
CARTOON 5x8	51.84	51.54	47.44	58.60
CARTOON 10x16	55.90	60.35	53.15	67.55
CARTOON 20x32	66.66	61.61	54.29	76.28
BABY 4x4	37.72	34.88	32.36	55.88
BABY 5x8	58.70	43.25	39.00	59.63
BABY 10x16	73.72	62.76	53.41	65.26
BABY 20x32	81.68	74.42	64.23	73.36
DEER 4x4	56.21	56.66	54.58	60.78
DEER 5x8	62.74	57.81	52.66	65.18
DEER 10x16	77.97	69.50	60.45	66.38
DEER 20x32	77.70	71.79	71.90	68.40

For the green color component, the AMBTC again proved to have the least MSE, as shown in Table 14. The 3LBTC and the BTC were a close second and the EBTC the worst. The 3LBTC showed better results, closer to the other methods, for the 10 x 16 and 20 x 32 blocking sizes for the green component than it did for the red.

TABLE 15

THE 3LBTC AND THE AMBTC HAVE THE SMALLEST MSE
FOR THE BLUE RGB COLOR COMPONENT.

MEAN-SQUARE-ERROR SUMMARY OF BLUE COMPONENT				
<u>IMAGE</u>	<u>3LBTC</u>	<u>BTC</u>	<u>AMBTC</u>	<u>EBTC</u>
CAT 4x4	17.56	20.19	19.98	22.18
CAT 5x8	22.22	24.13	23.04	26.16
CAT 10x16	28.42	27.07	26.07	29.03
CAT 20x32	36.75	28.01	27.44	29.45
CARTOON 4x4	22.51	28.29	27.01	29.57
CARTOON 5x8	29.11	36.03	34.35	36.31
CARTOON 10x16	39.83	43.61	40.86	42.89
CARTOON 20x32	64.94	47.96	47.92	47.76
BABY 4x4	17.40	18.48	17.44	32.38
BABY 5x8	30.63	27.35	25.64	37.90
BABY 10x16	44.55	38.52	35.42	42.23
BABY 20x32	38.81	41.20	38.59	42.82
DEER 4x4	23.10	30.92	30.41	29.11
DEER 5x8	26.70	27.17	25.53	33.37
DEER 10x16	43.55	37.63	34.19	42.53
DEER 20x32	37.88	45.46	43.69	48.74

The 3LBTC has the majority of the lowest MSEs for the images in the blue color component, shown in Table 15. The AMBTC was a very close second and the BTC and the EBTC not much worse. All the techniques had results fairly close to one another as compared to the results in Tables 13 and 14. Inspection of the reconstructed image histograms showed that the blue color component had the highest number of pixels in the 24 to 31 color range, the brightest and deepest color range. The red color component was next highest. The green color component did not have any pixels with values above 28.

Entropy

The reconstructed color images were compared next in terms of the amount of information stored within as compared to the entropy of the original color images. The entropies of the original images are shown in Table 16. The entropies for each of the reconstructed images is shown by color component in Tables 17 through 19. Table 17 shows the BTC and the AMBTC as having the most red component entropies closest to the

original images. The 3LBTC also had entropies close to the originals, but not as close as the BTC or AMBTC. The EBTC image entropies had the most information, i.e., they had the highest entropy, but they were not close to the original entropies.

TABLE 16

THE AMOUNT OF INFORMATION IN THE ORIGINAL IMAGES.

ORIGINAL IMAGE ENTROPIES			
IMAGE	RED	GREEN	BLUE
CAT	-80.89	-95.20	-93.47
CARTOON	-12.94	-32.73	-30.24
BABY	-69.47	-42.59	-89.65
DEER	-38.76	-61.51	-93.39

TABLE 17

THE EBTC HAS THE LARGEST ENTROPY
FOR THE RED RGB COLOR COMPONENT.

ENTROPY SUMMARY OF RED COMPONENT				
IMAGE	3LBTC	BTC	AMBTC	EBTC
CAT 4x4	-80.48	-58.66	-58.63	-57.19
CAT 5x8	-80.28	-81.09	-80.67	-79.49
CAT 10x16	-79.63	-81.87	-80.67	-80.26
CAT 20x32	-81.66	-79.51	-81.30	-82.46
CARTOON 4x4	-12.66	-12.72	-12.94	-10.34
CARTOON 5x8	-13.78	-13.06	-13.81	-10.25
CARTOON 10x16	-16.45	-11.24	-13.63	-10.74
CARTOON 20x32	-27.22	-11.99	-13.66	-8.910
BABY 4x4	-66.48	-66.77	-66.77	-63.43
BABY 5x8	-69.07	-69.81	-69.55	-45.02
BABY 10x16	-46.83	-69.67	-68.82	-42.57
BABY 20x32	-41.97	-68.74	-69.00	-40.69
DEER 4x4	-63.95	-49.46	-49.20	-72.78
DEER 5x8	-63.24	-85.59	-86.54	-32.08
DEER 10x16	-49.22	-52.21	-32.37	-33.82
DEER 20x32	-32.75	-45.58	-45.54	-33.77

TABLE 18

THE 3LBTC HAS THE SMALLEST ENTROPY
FOR THE GREEN RGB COLOR COMPONENT.

ENTROPY SUMMARY OF GREEN COMPONENT				
<u>IMAGE</u>	<u>3LBTC</u>	<u>BTC</u>	<u>AMBTC</u>	<u>EBTC</u>
CAT 4x4	-96.16	-71.12	-71.26	-72.37
CAT 5x8	-96.46	-94.76	-95.01	-97.03
CAT 10x16	-96.56	-71.53	-71.74	-73.43
CAT 20x32	-96.17	-72.09	-71.26	-74.47
CARTOON 4x4	-74.37	-19.95	-75.36	-72.18
CARTOON 5x8	-31.56	-32.21	-20.30	-16.98
CARTOON 10x16	-33.71	-30.34	-20.20	-16.06
CARTOON 20x32	-26.14	-30.56	-30.86	-14.42
BABY 4x4	-83.44	-83.99	-84.13	-82.69
BABY 5x8	-39.24	-40.95	-29.40	-26.67
BABY 10x16	-39.63	-27.08	-28.76	-81.51
BABY 20x32	-36.31	-27.35	-27.86	-79.53
DEER 4x4	-96.50	-33.73	-42.60	-27.67
DEER 5x8	-96.02	-53.27	-37.13	-83.01
DEER 10x16	-31.82	-37.69	-38.76	-30.11
DEER 20x32	-50.12	-37.93	-53.51	-32.11

The 3LBTC had the most reconstructed images with entropies closest to the original image entropies for the green color component, as shown in Table 18. The other techniques' reconstructed images had resulting entropies similar to one another but not as close to those of the originals as the 3LBTC.

The 3LBTC again had entropies resulting in being the closest to those of the original images for the blue color component, shown in Table 19. The other techniques had resulting entropies very similar to one another, but again not as close to the originals as the 3LBTC.

TABLE 19

THE EBTC AND THE 3LBTC HAVE THE SMALLEST ENTROPY
FOR THE BLUE RGB COLOR COMPONENT.

ENTROPY SUMMARY OF BLUE COMPONENT				
<u>IMAGE</u>	<u>3LBTC</u>	<u>BTC</u>	<u>AMBTC</u>	<u>EBTC</u>
CAT 4x4	-91.27	-82.75	-82.46	-78.74
CAT 5x8	-79.22	-83.21	-92.88	-78.88
CAT 10x16	-100.9	-93.62	-92.25	-78.44
CAT 20x32	-86.05	-94.20	-92.23	-77.03
CARTOON 4x4	-29.77	-21.87	-21.93	-20.67
CARTOON 5x8	-29.52	-29.45	-21.66	-20.19
CARTOON 10x16	-30.28	-29.13	-22.12	-20.86
CARTOON 20x32	-30.09	-30.20	-32.82	-20.33
BABY 4x4	-89.69	-89.74	-89.82	-90.05
BABY 5x8	-88.88	-89.86	-102.2	-90.10
BABY 10x16	-87.99	-102.2	-102.1	-89.93
BABY 20x32	-75.52	-90.25	-101.3	-87.87
DEER 4x4	-22.81	-91.82	-91.41	-22.05
DEER 5x8	-22.65	-93.19	-93.43	-92.19
DEER 10x16	-93.13	-92.63	-28.95	-27.45
DEER 20x32	-54.29	-92.96	-23.03	-80.48

Processing Time

An IBM PC/AT compatible was used to execute the PASCAL color Image Computer Application program included as Appendix E. The four blocking sizes used, 4x4, 5x8, 10x16, and 20x32, were timed for each of the four Block Truncation Coding Techniques. The coding and reconstruction algorithms and output processing to the resulting image file were included in the processing time. The 3LBTC processed the images the quickest at approximately 2 minutes per image. The AMBTC had the second quickest processing time of about 4 minutes per each picture. The BTC was third fastest with an average of 5.5 minutes. The EBTC showed the slowest processing time of all the techniques. The fastest that it processed an image was approximately 45 minutes. Table 20 summarizes the processing time results.

TABLE 20

3LBTC PROCESSED THE IMAGES AT LEAST TWICE AS FAST
AS THE CURRENT CODING TECHNIQUES.

PROCESSING TIME				
<u>METHOD</u>	<u>4x4</u>	<u>5x8</u>	<u>10x16</u>	<u>20x32</u>
BTC	7 min.	6 min.	5 min.	4 min.
AMBTC	4 min.	4 min.	4 min.	3 min.
EBTC	45 min	1 hr.	1.5 hr.	2.5 hr.
3LBTC	2 min.	2 min.	2 min.	1 min.

VI. CONCLUSIONS

The comparisons of the Three-Level Block Truncation Coding technique to the current coding techniques, the original Block Truncation Coding, the Absolute Moment Block Truncation Coding, and the Entropy-Preserving Block Truncation Coding, in Chapter IV for the sample image blocks showed that the 3LBTC did not have as small of a compression ratio as the current techniques (Table 7). It had the smallest mean-square-error (Table 8), which was expected due to its larger compression ratio. It did not have reconstructed image entropy results (Table 9) as close to the original sample images as the other techniques. Its processing time, shown previously in Table 10, was the quickest with the AMBTC technique one second slower.

When the BTC techniques were applied to actual color images, the 3LBTC again had the largest compression ratio (Table 12), the smallest mean-square-error for the blue color component (Table 15), entropies closest to the original images for both the green and blue color components (Tables 18 and 19), and a processing speed (Table 20) twice as fast as the next fastest technique, AMBTC. Its reconstructed images were subjectively very close to the originals.

If a fast image coding algorithm is required and bandwidth is not a critical factor, the Three-Level Block Truncation Coding technique would be a suitable image coding solution.

APPENDICES

APPENDIX A
F77 LISTING OF THREE-LEVEL BTC
VARIATIONS COMPARISON

```

program three_level_compare

integer    x(16,11), y(16), i, ones, zeroes, lo, hi, num_lo,
&          num_hi,j, mid, hi_ones, lo_zeroes, lo_a, hi_b

real      mean, abs_mnt, entropy, mean_sqrd, mses(6,11),
&          ent(6,11), range, lo_absmnt, hi_absmnt, lo_mean,
&          hi_mean

data      ((x(i,j),i=1,16),j=1,11)
&/121,114, 56, 47, 37,200,247,255, 16,  0, 12,169, 43,  5,  7,251,
& 61, 59, 65, 63, 67, 67, 69, 67, 73, 73, 73, 73, 77, 77, 77, 77,
& 61, 59, 63, 63, 69, 69, 69, 71, 73, 75, 71, 75, 77, 75, 75, 77,
& 69, 63, 63, 69, 71, 71, 65, 69, 69, 69, 75, 79, 81, 77, 75, 81,
& 67, 67, 69, 69, 73, 73, 73, 75, 79, 77, 79, 77, 77, 81, 81, 81,
& 67, 63, 65, 69, 73, 73, 73, 73, 77, 73, 73, 73, 77, 77, 79, 81,
& 240,169,210,100,180,140,130,230,133,  90,  77,180,166,211,222,150,
& 10,  20,  30,  40,250,240,230,220,100,200,110,190,120,180,130,170,
& 100,200,150, 50, 78, 82, 29,240, 40, 30, 20, 60,100,110,120,130,
& 100,124,167,234, 87,136,198,247, 65,120,167,227, 45,115,132,219,
& 255,255,255,255,255,255,254,254,255,255,255,255,255,255,255/

open (20, file='results.dat', status='new')

do (j=1, 11)
  write (20,*) 'sample image block:'

  call write_array (x(1,j))

  entropy = 0.0
  do (i=1, 16)
    if (x(i,j) <> 0)
&      entropy = entropy + (x(i,j) * (log(float(x(i,j))) / log(2.0)))
  repeat
  entropy = -(entropy / 16)

  hi = 0
  lo = 256
  lo_mean = 0.0
  mean = 0.0
  do (i=1, 16)
    if (x(i,j) > hi)    hi = x(i,j)
    if (x(i,j) < lo)    lo = x(i,j)
    mean = mean + x(i,j)
  repeat
  mean = mean / 16.0
  mean_sqrd = mean **2

  write (20,*) 'mean = ',mean,'      entropy = ',entropy
  write (20,*)

```

```

*****      Original Three-Level BTC      *****

range = (1.0 * (hi -lo)) / 3.0
if (range < 1.0) then
    hi_mean = hi
    lo_mean = lo
else
    lo_mean = range + lo
    hi_mean = hi - range
end if

hi_b = (hi + hi_mean) / 2
mid  = (hi_mean + lo_mean) / 2
lo_a = (lo + lo_mean) / 2

write (20,*) '*** Original Three-Level BTC ***'

write (20,*) 'highest=', hi,'   hi_mean=',hi_mean
write (20,*) 'lowest =', lo,'   lo_mean=',lo_mean
write (20,*) 'range=',range,'   b=',hi_b,'   mid=',mid,'   a=',lo_a

do (i=1,16)
    if (x(i,j) >= hi_mean) then
        y(i) = hi_b
    else
        if (x(i,j) <= lo_mean) then
            y(i) = lo_a
        else
            y(i) = mid
        end if
    end if
end do
repeat

call write_array (y)
call calc_errs (x(1,j),y,mSES(1,j),ent(1,j))

```


***** HIGH/LOW VARIATION 3LBTB TECHNIQUE *****

```

range = (1.0 * (hi -lo)) / 3.0
if (range < 1.0) then
    hi_mean = mean
else
    lo_mean = range + lo
    hi_mean = hi - range
end if

hi_ones = 0
mid = 0
lo_zeroes = 0
num_lo = 0
num_hi = 0
lo_absmnt = 0.0
hi_absmnt = 0.0
do (i=1,16)
    if (x(i,j) < hi_mean) then
        num_lo = num_lo + 1
        lo_absmnt = lo_absmnt + abs(x(i,j) - lo_mean)
        if (x(i,j) > lo_mean) then
            mid = mid + 1
        else
            lo_zeroes = lo_zeroes + 1
        end if
    end if
    if (x(i,j) > lo_mean) then
        num_hi = num_hi + 1
        hi_absmnt = hi_absmnt + abs(x(i,j) - hi_mean)
        if (x(i,j) >= hi_mean) hi_ones = hi_ones + 1
    end if
repeat
hi_absmnt = hi_absmnt / num_hi
lo_absmnt = lo_absmnt / num_lo

write (20,*) '*** high/low 3-LEVEL BTC ***'

num_hi = hi_ones + mid
num_lo = lo_zeroes + mid

call reconst (num_hi,num_lo,hi_ones,mid,lo_zeroes,hi_absmnt,
&             lo_absmnt,hi_mean,lo_mean,x(1,j),y)

call write_array (y)
call calc_errs (x(1,j),y,m ses(2,j),ent(2,j))

```

***** MID/LOW 3 LEVEL BLOCK TRUNCATION CODING VARIATION *****

```

hi_ones = 0
mid = 0
lo_zeroes = 0
num_lo = 0
num_hi = 0
lo_absmnt = 0.0
hi_absmnt = 0.0
do (i=1,16)
  if (x(i,j) < hi_mean) then
    num_lo = num_lo + 1
    lo_absmnt = lo_absmnt + abs(x(i,j) - lo_mean)
    if (x(i,j) > lo_mean) then
      mid = mid + 1
    else
      lo_zeroes = lo_zeroes + 1
    end if
  else
    num_hi = num_hi + 1
    hi_absmnt = hi_absmnt + abs(x(i,j) - hi_mean)
    hi_ones = hi_ones + 1
  end if
repeat
  hi_absmnt = hi_absmnt / num_hi
  lo_absmnt = lo_absmnt / num_lo

write (20,*) '**** mid/low 3-LEVEL BTC ****'

num_hi = hi_ones
num_lo = lo_zeroes + mid

call reconst (num_hi,num_lo,hi_ones,mid,lo_zeroes,hi_absmnt,
&      lo_absmnt,hi_mean,lo_mean,x(1,j),y)

call write_array (y)
call calc_errs (x(1,j),y,mse(3,j),ent(3,j))

```

***** MID/HIGH 3 LEVEL BLOCK TRUNCATION CODING VARIATION *****

```

hi_ones = 0
mid = 0
lo_zeroes = 0
num_lo = 0
num_hi = 0
lo_absmnt = 0.0
hi_absmnt = 0.0
do (i=1,16)
  if (x(i,j) > lo_mean) then
    num_hi = num_hi + 1
    hi_absmnt = hi_absmnt + abs(x(i,j) - hi_mean)
    if (x(i,j) < hi_mean) then
      mid = mid + 1
    else
      hi_ones = hi_ones + 1
    end if
  else
    num_lo = num_lo + 1
    lo_absmnt = lo_absmnt + abs(x(i,j) - lo_mean)
    lo_zeroes = lo_zeroes + 1
  end if
repeat
hi_absmnt = hi_absmnt / num_hi
lo_absmnt = lo_absmnt / num_lo

write (20,*) '*** mid/high 3-LEVEL BTC ***'

num_hi = hi_ones + mid
num_lo = lo_zeroes

call reconst (num_hi,num_lo,hi_ones,mid,lo_zeroes,hi_absmnt,
&      lo_absmnt,hi_mean,lo_mean,x(1,j),y)

call write_array (y)
call calc_errs (x(1,j),y,mse(4,j),ent(4,j))

```


***** MEAN-SPLITS-MID 3 LEVEL BLOCK TRUNCATION CODING VARIATION *****

```

hi_ones = 0
mid = 0
lo_zeroes = 0
num_lo = 0
num_hi = 0
lo_absmnt = 0.0
hi_absmnt = 0.0
do (i=1,16)
  if (x(i,j) < mean) then
    num_lo = num_lo + 1
    lo_absmnt = lo_absmnt + abs(x(i,j) - lo_mean)
    if (x(i,j) > lo_mean) then
      mid = mid + 1
    else
      lo_zeroes = lo_zeroes + 1
    end if
  else
    num_hi = num_hi + 1
    hi_absmnt = hi_absmnt + abs(x(i,j) - hi_mean)
    if (x(i,j) < hi_mean) then
      mid = mid + 1
    else
      hi_ones = hi_ones + 1
    end if
  end if
repeat
hi_absmnt = hi_absmnt / num_hi
lo_absmnt = lo_absmnt / num_lo

write (20,*) '*** mean-splits-mid 3-LEVEL BTC ***'

num_hi = hi_ones + mid/2
num_lo = lo_zeroes + mid/2

call reconst (num_hi,num_lo,hi_ones,mid,lo_zeroes,hi_absmnt,
&      lo_absmnt,hi_mean,lo_mean,x(1,j),y)

call write_array (y)
call calc_errs (x(1,j),y,mse(5,j),ent(5,j))

```

***** HIGH/LOW MEAN 3 LEVEL BLOCK TRUNCATION CODING VARIATION *****

```

num_hi = 0
num_lo = 0
hi_mean = 0.0
lo_mean = 0.0
do (i=1,16)
  if (x(i,j) >= mean) then
    hi_mean = hi_mean + x(i,j)
    num_hi = num_hi + 1
  else
    lo_mean = lo_mean + x(i,j)
    num_lo = num_lo + 1
  end if
repeat
hi_mean = hi_mean / num_hi
lo_mean = lo_mean / num_lo

hi_ones = 0
mid = 0
lo_zeroes = 0
num_lo = 0
num_hi = 0
lo_absmnt = 0.0
hi_absmnt = 0.0
do (i=1,16)
  if (x(i,j) >= mean) then
    num_hi = num_hi + 1
    hi_absmnt = hi_absmnt + abs(x(i,j) - hi_mean)
    if (x(i,j) < hi_mean) then
      mid = mid + 1
    else
      hi_ones = hi_ones + 1
    end if
  else
    num_lo = num_lo + 1
    lo_absmnt = lo_absmnt + abs(x(i,j) - lo_mean)
    if (x(i,j) > lo_mean) then
      mid = mid + 1
    else
      lo_zeroes = lo_zeroes + 1
    end if
  end if
repeat

hi_absmnt = hi_absmnt / num_hi
lo_absmnt = lo_absmnt / num_lo

write (20,*) '*** high/low mean3-LEVEL BTC ***'

```

```

num_hi = hi_ones + mid/2
num_lo = lo_zeroes + mid/2

call reconst (num_hi,num_lo,hi_ones,mid,lo_zeroes,hi_absmnt,
&          lo_absmnt,hi_mean,lo_mean,x(1,j),y)

call write_array (y)
call calc_errs (x(1,j),y,mSES(6,j),ent(6,j))

repeat

write      (20,*)      '-----'
write (20,*) '          Mean Square Error Summary'
write (20,*) ' orig    high/low    mid/low    mid/high    mean-splits-mid',
&          'high/low mean'
write (20,*) '-----'
do (j=1,11)
    write (20,*) mSES(1,j),' ',mSES(2,j),' ',mSES(3,j),' ',
&          mSES(4,j),' ',mSES(5,j),' ',mSES(6,j))
    repeat
    write      (20,*)      '-----'

    write (20,*)
    write (20,*) '-----'
    write (20,*) '          Entropy Summary'
    write (20,*) ' orig    high/low    mid/low    mid/high    mean-splits-mid',
&          'high/low mean'
    write (20,*) '-----'
    do (j=1,11)
    write (20,*) ent(1,j),' ',ent(2,j),' ',ent(3,j),' ',
&          ent(4,j),' ',ent(5,j),' ',ent(6,j))
    repeat
    write      (20,*)      '-----'

end

```



```

subroutine reconst (nhi,nlo,ones,mid,zeroes,hi_abs,lo_abs,hmean,lmean,
&                  old,new)

integer nhi,nlo,ones,mid,zeroes,hi_a,hi_b,lo_a,lo_b
integer new(16),old(16)
real hi_abs,lo_abs,hmean,lmean

write (20,*) 'lo_mean=',lmean,' hi_mean=',hmean,
&            ' lo_absmnt=',lo_abs,' hi_absmnt=',hi_abs
write (20,*) 'hi_ones=',ones,' mid=',mid,' lo_zeroes=',zeroes

hi_b = 0
hi_a = 0
lo_a = 0
lo_b = 0

if (ones > 0)
&    hi_b = int(hmean+(((nhi*hi_abs)/2)/ones)+0.5)

if (zeroes > 0)
&    lo_a = int(lmean-(((nlo*lo_abs)/2)/zeroes)+0.5)

if (mid > 0) then
    hi_a = int(hmean-(((nhi*hi_abs)/2)/mid) +0.5)
    lo_b = int(lmean+(((nlo*lo_abs)/2)/mid) +0.5)
end if

do (i=1,16)
    if (old(i) >= hmean) then
        new(i) = hi_b
    else
        if (old(i) > lmean) then
            new(i) = int ((hi_a + lo_b) / 2)
            if (zeroes = 0) new(i) = hi_a
            if (ones = 0) new(i) = lo_b
        else
            new(i) = lo_a
        end if
    end if
end do
repeat

write (20,*) 'hi_b=',hi_b,' hi_a=',hi_a,' lo_b=',lo_b,' lo_a=',lo_a
write (20,*) 'reconstructed image block:'

return
end

```

```

subroutine write_array (array)

integer    array(16)

write (20,10) array(1), array(2), array(3), array(4)
write (20,10) array(5), array(6), array(7), array(8)
write (20,10) array(9), array(10), array(11), array(12)
write (20,10) array(13), array(14), array(15), array(16)
10 format (10x, i3, 2x, i3, 2x, i3, 2x, i3)

return
end

subroutine calc_errs (in, out,mse,hse)

integer in(16), out(16), i, mthd, set
real    mse, hse

mse = 0.0
do (i=1, 16)
    mse = mse + ((out(i) - in(i)) **2)
repeat
mse = mse / 16.0

hse = 0.0
do (i=1, 16)
    if (out(i) > 0)
&        hse = hse + (out(i)*(log(float(out(i)))/log(2.0)))
repeat
hse = -(hse / 16.0)

write (20,*) 'mse= ', mse,'    entropy= ',hse
write (20,*)

return
end

```

APPENDIX B
RESULTS OF THREE-LEVEL BTC
VARIATIONS COMPARISON

sample image block:

121	114	56	47
37	200	247	255
16	0	12	169
43	5	7	251

mean = 98.75 entropy = -721.8

*** Original Three-Level BTC ***

highest= 255 hi_mean= 170.0

lowest = 0 lo_mean= 85.00

range= 85.00 b= 212 mid= 127 a= 42

127	127	42	42
42	212	212	212
42	42	42	127
42	42	42	212

mse= 805.6 entropy= -703.4

*** high/low 3-LEVEL BTC ***

lo_mean= 85.00 hi_mean= 170.0 lo_absmnt= 57.58 hi_absmnt= 54.14

hi_ones= 4 mid= 3 lo_zeroes= 9

hi_b= 217 hi_a= 107 lo_b= 200 lo_a= 47

reconstructed image block:

153	153	47	47
47	217	217	217
47	47	47	153
47	47	47	217

mse= 909.1 entropy= -776.1

*** mid/low 3-LEVEL BTC ***

lo_mean= 85.00 hi_mean= 170.0 lo_absmnt= 57.58 hi_absmnt= 68.25

hi_ones= 4 mid= 3 lo_zeroes= 9

hi_b= 204 hi_a= 125 lo_b= 200 lo_a= 47

reconstructed image block:

162	162	47	47
47	204	204	204
47	47	47	162
47	47	47	204

mse= 1167. entropy= -761.1

*** mid/high 3-LEVEL BTC ***

lo_mean= 85.00 hi_mean= 170.0 lo_absmnt= 60.22 hi_absmnt= 54.14

hi_ones= 4 mid= 3 lo_zeroes= 9

hi_b= 217 hi_a= 107 lo_b= 175 lo_a= 55

reconstructed image block:

141	141	55	55
55	217	217	217
55	55	55	141
55	55	55	217

mse= 1090. entropy= -788.7

*** mean-splits-mid 3-LEVEL BTC ***

lo_mean= 85.00 hi_mean= 170.0 lo_absmnt= 60.22 hi_absmnt= 54.14

hi_ones= 4 mid= 3 lo_zeroes= 9

hi_b= 204 hi_a= 125 lo_b= 185 lo_a= 52

reconstructed image block:

155	155	52	52
52	204	204	204
52	52	52	155
52	52	52	204

mse= 1243. entropy= -769.5

*** high/low mean 3-LEVEL BTC ***

lo_mean= 24.78 hi_mean= 193.9 lo_absmnt= 18.64 hi_absmnt= 50.73

hi_ones= 4 mid= 7 lo_zeroes= 5

hi_b= 238 hi_a= 168 lo_b= 35 lo_a= 10

reconstructed image block:

101	101	101	101
101	238	238	238
10	10	10	101
101	10	10	238

mse= 1234. entropy= -774.3

sample image block:

61	59	65	63
67	67	69	67
73	73	73	73
77	77	77	77

mean = 69.87 entropy = -428.5

*** Original Three-Level BTC ***

highest= 77 hi_mean= 71.00

lowest = 59 lo_mean= 65.00

range= 6.000 b= 74 mid= 68 a= 62

62	62	62	62
68	68	68	68
74	74	74	74
74	74	74	74

mse= 4.000 entropy= -425.5

*** high/low 3-LEVEL BTC ***

lo_mean= 65.00 hi_mean= 71.00 lo_absmnt= 2.750 hi_absmnt= 3.833

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 74 hi_a= 65 lo_b= 68 lo_a= 62

reconstructed image block:

62	62	62	62
66	66	66	66
74	74	74	74
74	74	74	74

mse= 4.500 entropy= -421.8

*** mid/low 3-LEVEL BTC ***

lo_mean= 65.00 hi_mean= 71.00 lo_absmnt= 2.750 hi_absmnt= 4.000

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 73 hi_a= 67 lo_b= 68 lo_a= 62

reconstructed image block:

62	62	62	62
67	67	67	67
73	73	73	73
73	73	73	73

mse= 5.500 entropy= -419.8

*** mid/high 3-LEVEL BTC ***

lo_mean= 65.00 hi_mean= 71.00 lo_absmnt= 3.000 hi_absmnt= 3.833

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 74 hi_a= 65 lo_b= 67 lo_a= 64

reconstructed image block:

64	64	64	64
66	66	66	66
74	74	74	74
74	74	74	74

mse= 5.500 entropy= -425.5

*** mean-splits-mid 3-LEVEL BTC ***

lo_mean= 65.00 hi_mean= 71.00 lo_absmnt= 2.750 hi_absmnt= 4.000

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 74 hi_a= 66 lo_b= 67 lo_a= 63

reconstructed image block:

63	63	63	63
66	66	66	66
74	74	74	74
74	74	74	74

mse= 4.750 entropy= -423.6

*** high/low mean 3-LEVEL BTC ***

lo_mean= 64.75 hi_mean= 75.00 lo_absmnt= 2.813 hi_absmnt= 2.000

hi_ones= 4 mid= 9 lo_zeroes= 3

hi_b= 77 hi_a= 74 lo_b= 66 lo_a= 61

reconstructed image block:

61	61	70	61
70	70	70	70
70	70	70	70
77	77	77	77

mse= 6.063 entropy= -429.8

sample image block:

61	59	63	63
69	69	69	71
73	75	71	75
77	75	75	77

mean = 70.13 entropy = -430.3

*** Original Three-Level BTC ***

highest= 77 hi_mean= 71.00

lowest = 59 lo_mean= 65.00

range= 6.000 b= 74 mid= 68 a= 62

62	62	62	62
68	68	68	74
74	74	74	74
74	74	74	74

mse= 3.500 entropy= -428.4

*** high/low 3-LEVEL BTC ***

lo_mean= 65.00 hi_mean= 71.00 lo_absmnt= 3.714 hi_absmnt= 3.000

hi_ones= 9 mid= 3 lo_zeroes= 4

hi_b= 73 hi_a= 65 lo_b= 69 lo_a= 62

reconstructed image block:

62	62	62	62
67	67	67	73
73	73	73	73
73	73	73	73

mse= 5.000 entropy= -422.7

*** mid/low 3-LEVEL BTC ***

lo_mean= 65.00 hi_mean= 71.00 lo_absmnt= 3.714 hi_absmnt= 3.333

hi_ones= 9 mid= 3 lo_zeroes= 4

hi_b= 73 hi_a= 66 lo_b= 69 lo_a= 62

reconstructed image block:

62	62	62	62
67	67	67	73
73	73	73	73
73	73	73	73

mse= 5.000 entropy= -422.7

*** mid/high 3-LEVEL BTC ***

lo_mean= 65.00 hi_mean= 71.00 lo_absmnt= 3.500 hi_absmnt= 3.000

hi_ones= 9 mid= 3 lo_zeroes= 4

hi_b= 73 hi_a= 65 lo_b= 67 lo_a= 63

reconstructed image block:

63	63	63	63
66	66	66	73
73	73	73	73
73	73	73	73

mse= 6.438 entropy= -423.1

*** mean-splits-mid 3-LEVEL BTC ***

lo_mean= 65.00 hi_mean= 71.00 lo_absmnt= 3.714 hi_absmnt= 3.333

hi_ones= 9 mid= 3 lo_zeroes= 4

hi_b= 73 hi_a= 65 lo_b= 68 lo_a= 63

reconstructed image block:

63	63	63	63
66	66	66	73
73	73	73	73
73	73	73	73

mse= 6.438 entropy= -423.1

*** high/low mean 3-LEVEL BTC ***

lo_mean= 64.71 hi_mean= 74.33 lo_absmnt= 3.673 hi_absmnt= 1.778

hi_ones= 6 mid= 6 lo_zeroes= 4

hi_b= 76 hi_a= 73 lo_b= 67 lo_a= 62

reconstructed image block:

62	62	62	62
70	70	70	70
70	76	70	76
76	76	76	76

mse= 2.000 entropy= -431.2

sample image block:

69	63	63	69
71	71	65	69
69	69	75	79
81	77	75	81

mean = 71.62 entropy = -441.7

*** Original Three-Level BTC ***

highest= 81 hi_mean= 75.00

lowest = 63 lo_mean= 69.00

range= 6.000 b= 78 mid= 72 a= 66

66	66	66	66
72	72	66	66
66	66	78	78
78	78	78	78

mse= 6.500 entropy= -438.8

*** high/low 3-LEVEL BTC ***

lo_mean= 69.00 hi_mean= 75.00 lo_absmnt= 2.000 hi_absmnt= 3.250

hi_ones= 6 mid= 2 lo_zeroes= 8

hi_b= 77 hi_a= 69 lo_b= 74 lo_a= 68

reconstructed image block:

68	68	68	68
71	71	68	68
68	68	77	77
77	77	77	77

mse= 6.750 entropy= -442.5

*** mid/low 3-LEVEL BTC ***

lo_mean= 69.00 hi_mean= 75.00 lo_absmnt= 2.000 hi_absmnt= 3.000

hi_ones= 6 mid= 2 lo_zeroes= 8

hi_b= 77 hi_a= 71 lo_b= 74 lo_a= 68

reconstructed image block:

68	68	68	68
72	72	68	68
68	68	77	77
77	77	77	77

mse= 6.875 entropy= -443.5

*** mid/high 3-LEVEL BTC ***

lo_mean= 69.00 hi_mean= 75.00 lo_absmnt= 2.000 hi_absmnt= 3.250

hi_ones= 6 mid= 2 lo_zeroes= 8

hi_b= 77 hi_a= 69 lo_b= 73 lo_a= 68

reconstructed image block:

68	68	68	68
71	71	68	68
68	68	77	77
77	77	77	77

mse= 6.750 entropy= -442.5

*** mean-splits-mid 3-LEVEL BTC ***

lo_mean= 69.00 hi_mean= 75.00 lo_absmnt= 2.000 hi_absmnt= 3.000

hi_ones= 6 mid= 2 lo_zeroes= 8

hi_b= 77 hi_a= 70 lo_b= 74 lo_a= 68

reconstructed image block:

68	68	68	68
72	72	68	68
68	68	77	77
77	77	77	77

mse= 6.875 entropy= -443.5

*** high/low mean 3-LEVEL BTC ***

lo_mean= 67.80 hi_mean= 78.00 lo_absmnt= 2.480 hi_absmnt= 2.333

hi_ones= 3 mid= 10 lo_zeroes= 3

hi_b= 81 hi_a= 77 lo_b= 69 lo_a= 64

reconstructed image block:

73	64	64	73
73	73	64	73
73	73	73	81
81	73	73	81

mse= 7.438 entropy= -450.7

sample image block:

67	67	69	69
73	73	73	75
79	77	79	77
77	81	81	81

mean = 74.87 entropy = -466.4

*** Original Three-Level BTC ***

highest= 81 hi_mean= 76.33

lowest = 67 lo_mean= 71.67

range= 4.667 b= 78 mid= 74 a= 69

69	69	69	69
74	74	74	74
78	78	78	78
78	78	78	78

mse= 2.750 entropy= -465.4

*** high/low 3-LEVEL BTC ***

lo_mean= 71.67 hi_mean= 76.33 lo_absmnt= 2.750 hi_absmnt= 2.722

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 78 hi_a= 72 lo_b= 74 lo_a= 69

reconstructed image block:

69	69	69	69
73	73	73	73
78	78	78	78
78	78	78	78

mse= 2.750 entropy= -463.5

*** mid/low 3-LEVEL BTC ***

lo_mean= 71.67 hi_mean= 76.33 lo_absmnt= 2.750 hi_absmnt= 2.667

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 78 hi_a= 74 lo_b= 74 lo_a= 69

reconstructed image block:

69	69	69	69
74	74	74	74
78	78	78	78
78	78	78	78

mse= 2.750 entropy= -465.4

*** mid/high 3-LEVEL BTC ***

lo_mean= 71.67 hi_mean= 76.33 lo_absmnt= 3.667 hi_absmnt= 2.722

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 78 hi_a= 72 lo_b= 74 lo_a= 70

reconstructed image block:

70	70	70	70
73	73	73	73
78	78	78	78
78	78	78	78

mse= 3.500 entropy= -465.4

*** mean-splits-mid 3-LEVEL BTC ***

lo_mean= 71.67 hi_mean= 76.33 lo_absmnt= 2.667 hi_absmnt= 2.519

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 78 hi_a= 73 lo_b= 74 lo_a= 70

reconstructed image block:

70	70	70	70
73	73	73	73
78	78	78	78
78	78	78	78

mse= 3.500 entropy= -465.4

*** high/low mean 3-LEVEL BTC ***

lo_mean= 70.14 hi_mean= 78.56 lo_absmnt= 2.449 hi_absmnt= 1.827

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 80 hi_a= 78 lo_b= 71 lo_a= 68

reconstructed image block:

68	68	68	68
74	74	74	74
80	74	80	74
74	80	80	80

mse= 2.500 entropy= -462.6

sample image block:

67	63	65	69
73	73	73	73
77	73	73	73
77	77	79	81

mean = 72.87 entropy = -451.1

*** Original Three-Level BTC ***

highest= 81 hi_mean= 75.00

lowest = 63 lo_mean= 69.00

range= 6.000 b= 78 mid= 72 a= 66

66	66	66	66
72	72	72	72
78	72	72	72
78	78	78	78

mse= 2.500 entropy= -447.3

*** high/low 3-LEVEL BTC ***

lo_mean= 69.00 hi_mean= 75.00 lo_absmnt= 3.636 hi_absmnt= 2.500

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 78 hi_a= 73 lo_b= 72 lo_a= 64

reconstructed image block:

64	64	64	64
72	72	72	72
78	72	72	72
78	78	78	78

mse= 3.500 entropy= -443.6

*** mid/low 3-LEVEL BTC ***

lo_mean= 69.00 hi_mean= 75.00 lo_absmnt= 3.636 hi_absmnt= 3.200

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 77 hi_a= 74 lo_b= 72 lo_a= 64

reconstructed image block:

64	64	64	64
73	73	73	73
77	73	73	73
77	77	77	77

mse= 3.500 entropy= -444.5

*** mid/high 3-LEVEL BTC ***

lo_mean= 69.00 hi_mean= 75.00 lo_absmnt= 3.000 hi_absmnt= 2.500

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 78 hi_a= 73 lo_b= 70 lo_a= 68

reconstructed image block:

68	68	68	68
71	71	71	71
78	71	71	71
78	78	78	78

mse= 4.813 entropy= -447.7

*** mean-splits-mid 3-LEVEL BTC ***

lo_mean= 69.00 hi_mean= 75.00 lo_absmnt= 3.000 hi_absmnt= 2.500

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 77 hi_a= 74 lo_b= 71 lo_a= 66

reconstructed image block:

66	66	66	66
72	72	72	72
77	72	72	72
77	77	77	77

mse= 2.938 entropy= -444.9

*** high/low mean 3-LEVEL BTC ***

lo_mean= 66.00 hi_mean= 75.17 lo_absmnt= 2.000 hi_absmnt= 2.528

hi_ones= 5 mid= 9 lo_zeroes= 2

hi_b= 77 hi_a= 74 lo_b= 67 lo_a= 63

reconstructed image block:

70	63	63	70
70	70	70	70
77	70	70	70
77	77	77	77

mse= 6.063 entropy= -439.2

sample image block:

240	169	210	100
180	140	130	230
133	90	77	180
166	211	222	150

mean = 164.3 entropy = -1220.

*** Original Three-Level BTC ***

highest= 240 hi_mean= 185.7

lowest = 77 lo_mean= 131.3

range= 54.33 b= 212 mid= 158 a= 104

212	158	212	104
158	158	104	212
158	104	104	158
158	212	212	158

mse= 312.3 entropy= -1191.

*** high/low 3-LEVEL BTC ***

lo_mean= 131.3 hi_mean= 185.7 lo_absmnt= 29.73 hi_absmnt= 30.53

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 222 hi_a= 160 lo_b= 155 lo_a= 90

reconstructed image block:

222	157	222	90
157	157	90	222
157	90	90	157
157	222	222	157

mse= 294.9 entropy= -1188.

*** mid/low 3-LEVEL BTC ***

lo_mean= 131.3 hi_mean= 185.7 lo_absmnt= 29.73 hi_absmnt= 36.93

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 204 hi_a= 172 lo_b= 155 lo_a= 90

reconstructed image block:

204	163	204	90
163	163	90	204
163	90	90	163
163	204	204	163

mse= 404.4 entropy= -1159.

*** mid/high 3-LEVEL BTC ***

lo_mean= 131.3 hi_mean= 185.7 lo_absmnt= 32.08 hi_absmnt= 30.53

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 222 hi_a= 160 lo_b= 141 lo_a= 115

reconstructed image block:

222	150	222	115
150	150	115	222
150	115	115	150
150	222	222	150

mse= 373.6 entropy= -1212.

*** mean-splits-mid 3-LEVEL BTC ***

lo_mean= 131.3 hi_mean= 185.7 lo_absmnt= 22.48 hi_absmnt= 25.81

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 206 hi_a= 171 lo_b= 143 lo_a= 112

reconstructed image block:

206	157	206	112
-----	-----	-----	-----

157	157	112	206
-----	-----	-----	-----

157	112	112	157
-----	-----	-----	-----

157	206	206	157
-----	-----	-----	-----

mse= 400.2 entropy= -1186.

*** high/low mean 3-LEVEL BTC ***

lo_mean= 117.1 hi_mean= 200.9 lo_absmnt= 24.12 hi_absmnt= 24.12

hi_ones= 5 mid= 8 lo_zeroes= 3

hi_b= 223 hi_a= 187 lo_b= 128 lo_a= 89

reconstructed image block:

223	157	223	89
-----	-----	-----	----

157	157	157	223
-----	-----	-----	-----

157	89	89	157
-----	----	----	-----

157	223	223	157
-----	-----	-----	-----

mse= 240.2 entropy= -1224.

sample image block:

10	20	30	40
250	240	230	220
100	200	110	190
120	180	130	170

mean = 140.0 entropy = -1038.

*** Original Three-Level BTC ***

highest= 250 hi_mean= 170.0

lowest = 10 lo_mean= 90.00

range= 80.00 b= 210 mid= 130 a= 50

50	50	50	50
210	210	210	210
130	210	130	210
130	210	130	210

mse= 650.0 entropy= -1109.

*** high/low 3-LEVEL BTC ***

lo_mean= 90.00 hi_mean= 170.0 lo_absmnt= 45.00 hi_absmnt= 45.00

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 204 hi_a= 103 lo_b= 135 lo_a= 45

reconstructed image block:

45	45	45	45
204	204	204	204
119	204	119	204
119	204	119	204

mse= 559.5 entropy= -1049.

*** mid/low 3-LEVEL BTC ***

lo_mean= 90.00 hi_mean= 170.0 lo_absmnt= 45.00 hi_absmnt= 40.00

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 190 hi_a= 130 lo_b= 135 lo_a= 45

reconstructed image block:

45	45	45	45
190	190	190	190
132	190	132	190
132	190	132	190

mse= 809.8 entropy= -1013.

*** mid/high 3-LEVEL BTC ***

lo_mean= 90.00 hi_mean= 170.0 lo_absmnt= 65.00 hi_absmnt= 45.00

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 204 hi_a= 103 lo_b= 123 lo_a= 58

reconstructed image block:

58	58	58	58
204	204	204	204
113	204	113	204
113	204	113	204

mse= 728.7 entropy= -1060.

*** mean-splits-mid 3-LEVEL BTC ***

lo_mean= 90.00 hi_mean= 170.0 lo_absmnt= 45.00 hi_absmnt= 40.00

hi_ones= 8 mid= 4 lo_zeroes= 4

hi_b= 195 hi_a= 120 lo_b= 124 lo_a= 56

reconstructed image block:

56	56	56	56
195	195	195	195
122	195	122	195
122	195	122	195

mse= 802.5 entropy= -1034.

*** high/low mean 3-LEVEL BTC ***

lo_mean= 70.00 hi_mean= 210.0 lo_absmnt= 45.00 hi_absmnt= 25.00

hi_ones= 4 mid= 8 lo_zeroes= 4

hi_b= 235 hi_a= 198 lo_b= 93 lo_a= 25

reconstructed image block:

25	25	25	25
235	235	235	235
145	145	145	145
145	145	145	145

mse= 750.0 entropy= -1012.

sample image block:

100	200	150	50
78	82	29	240
40	30	20	60
100	110	120	130

mean = 96.19 entropy = -660.2

*** Original Three-Level BTC ***

highest= 240 hi_mean= 166.7

lowest = 20 lo_mean= 93.33

range= 73.33 b= 203 mid= 130 a= 56

130	203	130	56
56	56	56	203
56	56	56	56
130	130	130	130

mse= 515.4 entropy= -699.5

*** high/low 3-LEVEL BTC ***

lo_mean= 93.33 hi_mean= 166.7 lo_absmnt= 36.26 hi_absmnt= 49.58

hi_ones= 2 mid= 6 lo_zeroes= 8

hi_b= 266 hi_a= 134 lo_b= 136 lo_a= 62

reconstructed image block:

135	266	135	62
62	62	62	266
62	62	62	62
135	135	135	135

mse= 859.2 entropy= -810.7

*** mid/low 3-LEVEL BTC ***

lo_mean= 93.33 hi_mean= 166.7 lo_absmnt= 36.26 hi_absmnt= 53.33

hi_ones= 2 mid= 6 lo_zeroes= 8

hi_b= 193 hi_a= 158 lo_b= 136 lo_a= 62

reconstructed image block:

147	193	147	62
62	62	62	193
62	62	62	62
147	147	147	147

mse= 889.8 entropy= -764.6

*** mid/high 3-LEVEL BTC ***

lo_mean= 93.33 hi_mean= 166.7 lo_absmnt= 44.71 hi_absmnt= 49.58

hi_ones= 2 mid= 6 lo_zeroes= 8

hi_b= 266 hi_a= 134 lo_b= 123 lo_a= 71

reconstructed image block:

128	266	128	71
71	71	71	266
71	71	71	71
128	128	128	128

mse= 950.9 entropy= -822.2

*** mean-splits-mid 3-LEVEL BTC ***

lo_mean= 93.33 hi_mean= 166.7 lo_absmnt= 44.71 hi_absmnt= 49.58

hi_ones= 2 mid= 6 lo_zeroes= 8

hi_b= 229 hi_a= 146 lo_b= 134 lo_a= 63

reconstructed image block:

140	229	140	63
63	63	63	229
63	63	63	63
140	140	140	140

mse= 690.6 entropy= -787.0

*** high/low mean 3-LEVEL BTC ***

lo_mean= 48.62 hi_mean= 143.8 lo_absmnt= 18.87 hi_absmnt= 39.69

hi_ones= 3 mid= 9 lo_zeroes= 4

hi_b= 190 hi_a= 128 lo_b= 57 lo_a= 30

reconstructed image block:

92	190	190	92
92	92	30	190
30	30	30	92
92	92	92	92

mse= 635.3 entropy= -644.1

sample image block:

100	124	167	234
87	136	198	247
65	120	167	227
45	115	132	219

mean = 148.9 entropy = -1094.

*** Original Three-Level BTC ***

highest= 247 hi_mean= 179.7

lowest = 45 lo_mean= 112.3

range= 67.33 b= 213 mid= 146 a= 78

78	146	146	213
78	146	213	213
78	146	146	213
78	146	146	213

mse= 448.5 entropy= -1097.

*** high/low 3-LEVEL BTC ***

lo_mean= 112.3 hi_mean= 179.7 lo_absmnt= 29.73 hi_absmnt= 43.61

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 232 hi_a= 142 lo_b= 136 lo_a= 71

reconstructed image block:

71	139	139	232
71	139	232	232
71	139	139	232
71	139	139	232

mse= 386.0 entropy= -1112.

*** mid/low 3-LEVEL BTC ***

lo_mean= 112.3 hi_mean= 179.7 lo_absmnt= 29.73 hi_absmnt= 45.33

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 202 hi_a= 163 lo_b= 136 lo_a= 71

reconstructed image block:

71	149	149	202
71	149	202	202
71	149	149	202
71	149	149	202

mse= 594.7 entropy= -1063.

*** mid/high 3-LEVEL BTC ***

lo_mean= 112.3 hi_mean= 179.7 lo_absmnt= 38.08 hi_absmnt= 43.61

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 232 hi_a= 142 lo_b= 123 lo_a= 93

reconstructed image block:

93	132	132	232
93	132	232	232
93	132	132	232
93	132	132	232

mse= 482.2 entropy= -1129.

*** mean-splits-mid 3-LEVEL BTC ***

lo_mean= 112.3 hi_mean= 179.7 lo_absmnt= 24.19 hi_absmnt= 36.00

hi_ones= 5 mid= 7 lo_zeroes= 4

hi_b= 208 hi_a= 159 lo_b= 124 lo_a= 91

reconstructed image block:

91 141 141 208

91 141 208 208

91 141 141 208

91 141 141 208

mse= 533.2 entropy= -1089.

*** high/low mean 3-LEVEL BTC ***

lo_mean= 102.7 hi_mean= 208.4 lo_absmnt= 25.26 hi_absmnt= 26.65

hi_ones= 4 mid= 8 lo_zeroes= 4

hi_b= 235 hi_a= 195 lo_b= 115 lo_a= 77

reconstructed image block:

77 155 155 235

77 155 155 235

77 155 155 235

77 155 155 235

mse= 567.2 entropy= -1147.

sample image block:

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mean = 254.9 entropy = -2037.

*** Original Three-Level BTC ***

highest= 255 hi_mean= 255.0

lowest = 254 lo_mean= 254.0

range= .3333 b= 255 mid= 254 a= 254

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mse= .0000 entropy= -2037.

*** high/low 3-LEVEL BTC ***

lo_mean= 254.0 hi_mean= 254.9 lo_absmnt= .0000 hi_absmnt= .1250

hi_ones= 14 mid= 0 lo_zeroes= 2

hi_b= 255 hi_a= 0 lo_b= 0 lo_a= 254

reconstructed image block:

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mse= .0000 entropy= -2037.

*** mid/low 3-LEVEL BTC ***

lo_mean= 254.0 hi_mean= 254.9 lo_absmnt= .0000 hi_absmnt= .1250

hi_ones= 14 mid= 0 lo_zeroes= 2

hi_b= 255 hi_a= 0 lo_b= 0 lo_a= 254

reconstructed image block:

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mse= .0000 entropy= -2037.

*** mid/high 3-LEVEL BTC ***

lo_mean= 254.0 hi_mean= 254.9 lo_absmnt= .0000 hi_absmnt= .1250

hi_ones= 14 mid= 0 lo_zeroes= 2

hi_b= 255 hi_a= 0 lo_b= 0 lo_a= 254

reconstructed image block:

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mse= .0000 entropy= -2037.

*** mean-splits-mid 3-LEVEL BTC ***

lo_mean= 254.0 hi_mean= 254.9 lo_absmnt= .0000 hi_absmnt= .1250

hi_ones= 14 mid= 0 lo_zeroes= 2

hi_b= 255 hi_a= 0 lo_b= 0 lo_a= 254

reconstructed image block:

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mse= .0000 entropy= -2037.

*** high/low mean 3-LEVEL BTC ***

lo_mean= 254.0 hi_mean= 255.0 lo_absmnt= .0000 hi_absmnt= .0000

hi_ones= 14 mid= 0 lo_zeroes= 2

hi_b= 255 hi_a= 0 lo_b= 0 lo_a= 254

reconstructed image block:

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mse= .0000 entropy= -2037.

APPENDIX C

F77 LISTING OF CURRENT CODING TECHNIQUES COMPARISON

```
program coding_compare
```

```

integer    x(16,11), y(16), i, ones, zeroes, lo, hi, num_lo, num_hi,
&          j, btc_a, btc_b, ambtc_a, ambtc_b, ebtc_a, ebtc_b,
&          mid, hi_ones, lo_zeroes

real       mean, std_dev, abs_mnt, entropy, mean_sqrd, a,
&          log2_a, log2_b, qb, funct_b, range, lo_mean, hi_mean,
&          epsilon, b, delta, derv_fb, sav_delta, sav_delta2,
&          mses(4,11), ent(4,11)

      data ((x(i,j),i=1,16),j=1,11)
&/121,114, 56, 47, 37,200,247,255, 16,  0, 12,169, 43,  5,  7,251,
& 61, 59, 65, 63, 67, 67, 69, 67, 73, 73, 73, 73, 77, 77, 77, 77,
& 61, 59, 63, 63, 69, 69, 69, 71, 73, 75, 71, 75, 77, 75, 75, 77,
& 69, 63, 63, 69, 71, 71, 65, 69, 69, 69, 75, 79, 81, 77, 75, 81,
& 67, 67, 69, 69, 73, 73, 73, 75, 79, 77, 79, 77, 77, 81, 81, 81,
& 67, 63, 65, 69, 73, 73, 73, 73, 77, 73, 73, 73, 77, 77, 79, 81,
& 240,169,210,100,180,140,130,230,133,  90,  77,180,166,211,222,150,
& 10,  20,  30,  40,250,240,230,220,100,200,110,190,120,180,130,170,
& 100,200,150, 50, 78, 82, 29,240, 40, 30, 20, 60,100,110,120,130,
& 100,124,167,234, 87,136,198,247, 65,120,167,227, 45,115,132,219,
& 255,255,255,255,255,255,254,254,255,255,255,255,255,255,255,255/

      open (20, file='results.dat', status='new')

      do (j=1, 11)
        write (20,*) 'sample image block:'

        call write_array (x(1,j))

        hi = 0
        lo = 256
        lo_mean = 0.0
        mean = 0.0
        do (i=1, 16)
          if (x(i,j) > hi)    hi = x(i,j)
          if (x(i,j) < lo)    lo = x(i,j)
          mean = mean + x(i,j)
        repeat
        mean = mean / 16.0
        mean_sqrd = mean **2

        std_dev = 0.0
        do (i=1, 16)
          std_dev = std_dev + ((x(i,j) - mean) **2)
        repeat
        std_dev = (std_dev/16) **0.5

```



```

abs_mnt = 0.0
do (i=1, 16)
  abs_mnt = abs_mnt + abs(x(i,j) - mean)
repeat
abs_mnt = abs_mnt / 16

entropy = 0.0
do (i=1, 16)
  if (x(i,j) <> 0)
    entropy = entropy + (x(i,j) * (log(float(x(i,j))) / log(2.0)))
& repeat
entropy = -(entropy / 16)

ones = 0
do (i=1, 16)
  if (x(i,j) >= mean) ones = ones + 1
repeat
zeroes = 16 - ones

write (20,*) 'mean=',mean,' standard deviation=',std_dev
write (20,*) 'absolute moment=',abs_mnt,' entropy=',entropy
write (20,*) 'ones=',ones,' zeroes=',zeroes
write (20,*)

```

***** 3 LEVEL BLOCK TRUNCATION CODING *****

```

range = (1.0 * (hi -lo)) / 3.0
if (range < 1.0) then
  hi_mean = hi
  lo_mean = lo
else
  lo_mean = range + lo
  hi_mean = hi - range
end if

hi_b = (hi + hi_mean) / 2
mid = (hi_mean + lo_mean) / 2
lo_a = (lo + lo_mean) / 2

write (20,*) '*** Three-Level BTC ***'
write (20,*) 'highest=', hi,' hi_mean=',hi_mean
write (20,*) 'lowest =', lo,' lo_mean=',lo_mean
write (20,*) 'range=',range,' b=',hi_b,' mid=',mid,' a=',lo_a

do (i=1,16)
  if (x(i,j) >= hi_mean) then
    y(i) = hi_b
  else
    if (x(i,j) <= lo_mean) then
      y(i) = lo_a
    else

```

```

        y(i) = mid
    end if
end if
repeat

write (20,*) 'reconstructed image block:'
call write_array (y)
call calc_errs (x(1,j),y,mSES(1,j),ent(1,j))

```

***** BLOCK TRUNCATION CODING *****

```

write (20,*) '*** BLOCK TRUNCATION CODING ***'
btc_a = int(mean-(std_dev*((float(ones)/zeroes) **0.5)) +0.5)
btc_b = int(mean+(std_dev*((float(zeroes)/ones) **0.5)) +0.5)

do (i=1, 16)
    if (x(i,j) >= mean) then
        y(i) = btc_b
    else
        y(i) = btc_a
    end if
repeat

write (20,*) 'a=',btc_a,'    b=',btc_b
write (20,*) 'reconstructed image block:'

call write_array (y)
call calc_errs (x(1,j),y,mSES(2,j),ent(2,j))

```

***** ABSOLUTE MOMENT BLOCK TRUNCATION CODING *****

```

write (20,*) '*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***'
ambtc_a = int (mean - (((16 * abs_mnt) / 2) / zeroes) + 0.5)
ambtc_b = int (mean + (((16 * abs_mnt) / 2) / ones) + 0.5)

do (i=1, 16)
    if (x(i,j) >= mean) then
        y(i) = ambtc_b
    else
        y(i) = ambtc_a
    end if
repeat

write (20,*) 'a=',ambtc_a,'    b=',ambtc_b
write (20,*) 'reconstructed image block:'

call write_array (y)
call calc_errs (x(1,j),y,mSES(3,j),ent(3,j))

```

***** ENTROPY-PRESERVING BLOCK TRUNCATION CODING *****

```

write (20,*) '*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***'
epsilon = .0001
delta = 1.0
sav_delta = 0.5
sav_delta2 = 0.5
b = mean + (std_dev / 10.0) + delta
do while ((abs (delta) > epsilon) .and. abs(delta - sav_delta2) > .0000001))
    sav_delta2 = sav_delta
    sav_delta = delta
    b = b - delta
    qb = ones * b
    a = ((16 * mean) - qb) / zeroes
    if (a > 0.0) then
        log2_a = log (a) / log (2.0)
    else
        log2_a = 0.0
    end if
    log2_b = log (b) / log (2.0)
    funct_b = ((qb-(16*mean))*log2_a) - (qb*log2_b) - (16*entropy)
    derv_fb = ones * (log2_a - log2_b)
    delta = funct_b / derv_fb
repeat

ebtc_b = int (b)
ebtc_a = int (a + 0.5)

do(i=1, 16)
    if (x(i,j) >= mean) then
        y(i) = ebtc_b
    else
        y(i) = ebtc_a
    end if
repeat

write (20,*) 'a=',ebtc_a,'    b=',ebtc_b
write (20,*) 'reconstructed image block:'

call write_array (y)
call calc_errs (x(1,j),y,mses(4,j),ent(4,j))

repeat

```



```

write      (20,*)      '-----'
write (20,*) '          Mean Square Error Summary'
write (20,*) ' 3-Level   BTC      AMBTC      EBTC'
write (20,*) '  -----      -----      -----'
do (j=1,11)
    write (20,*) mses(1,j),'      ',mses(2,j),'      ',mses(3,j),'      ', mses(4,j)
repeat
write      (20,*)      '-----'

write (20,*)
write      (20,*)      '-----'
write (20,*) '          Entropy Summary'
write (20,*) ' 3-Level   BTC      AMBTC      EBTC'
write (20,*) '  -----      -----      -----'
do (j=1,11)
    write (20,*) ent(1,j),'      ',ent(2,j),'      ',ent(3,j),'      ',ent(4,j)
repeat
write      (20,*)      '-----'

end

```

```

subroutine write_array (array)

integer    array(16)

write (20,10) array(1), array(2), array(3), array(4)
write (20,10) array(5), array(6), array(7), array(8)
write (20,10) array(9), array(10), array(11), array(12)
write (20,10) array(13), array(14), array(15), array(16)
10 format (10x, i3, 2x, i3, 2x, i3, 2x, i3)

return
end

```

```

subroutine calc_errs (in, out,mse,hse)

integer in(16), out(16), i, mthd, set
real mse, hse

mse = 0.0
do (i=1, 16)
    mse = mse + ((out(i) - in(i)) **2)
repeat
mse = mse / 16.0

hse = 0.0
do (i=1, 16)
    if (out(i) > 0)    hse = hse + (out(i)*(log(float(out(i)))/log(2.0)))
repeat
hse = -(hse / 16.0)

write (20,*) 'mse= ', mse,'    entropy= ',hse
write  (20,*)

return
end

```

APPENDIX D
RESULTS OF CURRENT CODING TECHNIQUES COMPARISON

sample image block:

121	114	56	47
37	200	247	255
16	0	12	169
43	5	7	251

mean= 98.75 standard deviation= 92.95
 absolute moment= 83.22 entropy= -721.8
 ones= 7 zeroes= 9

*** THREE-LEVEL BLOCK TRUNCATION CODING ***

highest= 255 hi_mean= 170.0
 lowest = 0 lo_mean= 85.00
 range= 85.00 b= 212.5 mid= 127 a= 42
 reconstructed image block:

127	127	42	42
42	212	212	212
42	42	42	127
42	42	42	212

mse= 805.6 entropy= -703.4

*** BLOCK TRUNCATION CODING ***

a= 17 b= 204
 reconstructed image block:

204	204	17	17
17	204	204	204
17	17	17	204
17	17	17	204

mse= 1684. entropy= -723.9

*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***

a= 25 b= 194
 reconstructed image block:

194	194	25	25
25	194	194	194
25	25	25	194
25	25	25	194

mse= 1605. entropy= -710.3

*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***

a= 18 b= 202
 reconstructed image block:

202	202	18	18
18	202	202	202
18	18	18	202
18	18	18	202

mse= 1660. entropy= -719.0

sample image block:

61	59	65	63
67	67	69	67
73	73	73	73
77	77	77	77

mean= 69.87 standard deviation= 5.787
 absolute moment= 5.125 entropy= -428.5
 ones= 8 zeroes= 8

*** THREE-LEVEL BLOCK TRUNCATION CODING ***

highest= 77 hi_mean= 71.00
 lowest = 59 lo_mean= 65.00
 range= 6.000 b= 74.00 mid= 68 a= 62
 reconstructed image block:

62	62	62	62
68	68	68	68
74	74	74	74
74	74	74	74

mse= 4.000 entropy= -425.5

*** BLOCK TRUNCATION CODING ***

a= 64 b= 76
 reconstructed image block:

64	64	64	64
64	64	64	64
76	76	76	76
76	76	76	76

mse= 8.000 entropy= -429.4

*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***

a= 65 b= 75
 reconstructed image block:

65	65	65	65
65	65	65	65
75	75	75	75
75	75	75	75

mse= 7.250 entropy= -429.3

*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***

a= 64 b= 75
 reconstructed image block:

64	64	64	64
64	64	64	64
75	75	75	75
75	75	75	75

mse= 7.500 entropy= -425.6

sample image block:

61	59	63	63
69	69	69	71
73	75	71	75
77	75	75	77

mean= 70.13 standard deviation= 5.655
 absolute moment= 4.734 entropy= -430.3
 ones= 9 zeroes= 7

*** THREE-LEVEL BLOCK TRUNCATION CODING ***

highest= 77 hi_mean= 71.00
 lowest = 59 lo_mean= 65.00
 range= 6.000 b= 74.00 mid= 68 a= 62
 reconstructed image block:

62	62	62	62
68	68	68	74
74	74	74	74
74	74	74	74

mse= 3.500 entropy= -428.4

*** BLOCK TRUNCATION CODING ***

a= 64 b= 75
 reconstructed image block:

64	64	64	64
64	64	64	75
75	75	75	75
75	75	75	75

mse= 9.688 entropy= -430.8

*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***

a= 65 b= 74
 reconstructed image block:

65	65	65	65
65	65	65	74
74	74	74	74
74	74	74	74

mse= 9.313 entropy= -429.7

*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***

a= 64 b= 75
 reconstructed image block:

64	64	64	64
64	64	64	75
75	75	75	75
75	75	75	75

mse= 9.688 entropy= -430.8

sample image block:

69	63	63	69
71	71	65	69
69	69	75	79
81	77	75	81

mean= 71.62 standard deviation= 5.644
 absolute moment= 4.781 entropy= -441.7
 ones= 6 zeroes= 10

*** THREE-LEVEL BLOCK TRUNCATION CODING ***

highest= 81 hi_mean= 75.00
 lowest = 63 lo_mean= 69.00
 range= 6.000 b= 78.00 mid= 72 a= 66
 reconstructed image block:

66	66	66	66
72	72	66	66
66	66	78	78
78	78	78	78

mse= 6.500 entropy= -438.8

*** BLOCK TRUNCATION CODING ***

a= 67 b= 79
 reconstructed image block:

67	67	67	67
67	67	67	67
67	67	79	79
79	79	79	79

mse= 8.250 entropy= -440.8

*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***

a= 68 b= 78
 reconstructed image block:

68	68	68	68
68	68	68	68
68	68	78	78
78	78	78	78

mse= 7.500 entropy= -442.6

*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***

a= 67 b= 78
 reconstructed image block:

67	67	67	67
67	67	67	67
67	67	78	78
78	78	78	78

mse= 7.875 entropy= -437.9

sample image block:

67	67	69	69
73	73	73	75
79	77	79	77
77	81	81	81

mean= 74.87 standard deviation= 4.768
 absolute moment= 4.141 entropy= -466.4
 ones= 9 zeroes= 7

*** THREE-LEVEL BLOCK TRUNCATION CODING ***

highest= 81 hi_mean= 76.33
 lowest = 67 lo_mean= 71.67
 range= 4.667 b= 78.67 mid= 74 a= 69
 reconstructed image block:

69	69	69	69
74	74	74	74
78	78	78	78
78	78	78	78

mse= 2.750 entropy= -465.4

*** BLOCK TRUNCATION CODING ***

a= 69 b= 79
 reconstructed image block:

69	69	69	69
69	69	69	79
79	79	79	79
79	79	79	79

mse= 6.000 entropy= -464.5

*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***

a= 70 b= 79
 reconstructed image block:

70	70	70	70
70	70	70	79
79	79	79	79
79	79	79	79

mse= 5.438 entropy= -467.8

*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***

a= 69 b= 79
 reconstructed image block:

69	69	69	69
69	69	69	79
79	79	79	79
79	79	79	79

mse= 6.000 entropy= -464.5

sample image block:

67	63	65	69
73	73	73	73
77	73	73	73
77	77	79	81

mean= 72.87 standard deviation= 4.768
 absolute moment= 3.438 entropy= -451.1
 ones= 12 zeroes= 4

*** THREE-LEVEL BLOCK TRUNCATION CODING ***

highest= 81 hi_mean= 75.00
 lowest = 63 lo_mean= 69.00
 range= 6.000 b= 78.00 mid= 72 a= 66
 reconstructed image block:

66	66	66	66
72	72	72	72
78	72	72	72
78	78	78	78

mse= 2.500 entropy= -447.3

*** BLOCK TRUNCATION CODING ***

a= 65 b= 76
 reconstructed image block:

65	65	65	65
76	76	76	76
76	76	76	76
76	76	76	76

mse= 7.750 entropy= -454.0

*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***

a= 66 b= 75
 reconstructed image block:

66	66	66	66
75	75	75	75
75	75	75	75
75	75	75	75

mse= 7.000 entropy= -450.1

*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***

a= 65 b= 75
 reconstructed image block:

65	65	65	65
75	75	75	75
75	75	75	75
75	75	75	75

mse= 7.250 entropy= -448.2

sample image block:

240	169	210	100
180	140	130	230
133	90	77	180
166	211	222	150

mean= 164.3 standard deviation= 48.98
 absolute moment= 41.22 entropy= -1220.
 ones= 9 zeroes= 7

*** THREE-LEVEL BLOCK TRUNCATION CODING ***

highest= 240 hi_mean= 185.7
 lowest = 77 lo_mean= 131.3
 range= 54.33 b= 212.8 mid= 158 a= 104
 reconstructed image block:

212	158	212	104
158	158	104	212
158	104	104	158
158	212	212	158

mse= 312.3 entropy= -1191.

*** BLOCK TRUNCATION CODING ***

a= 109 b= 207
 reconstructed image block:

207	207	207	109
207	109	109	207
109	109	109	207
207	207	207	109

mse= 723.5 entropy= -1219.

*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***

a= 117 b= 201
 reconstructed image block:

201	201	201	117
201	117	117	201
117	117	117	201
201	201	201	117

mse= 673.5 entropy= -1217.

*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***

a= 109 b= 207
 reconstructed image block:

207	207	207	109
207	109	109	207
109	109	109	207
207	207	207	109

mse= 723.5 entropy= -1219.

sample image block:

10	20	30	40
250	240	230	220
100	200	110	190
120	180	130	170

mean= 140.0 standard deviation= 79.69
 absolute moment= 70.00 entropy= -1038.
 ones= 8 zeroes= 8

*** THREE-LEVEL BLOCK TRUNCATION CODING ***

highest= 250 hi_mean= 170.0
 lowest = 10 lo_mean= 90.00
 range= 80.00 b= 210.0 mid= 130 a= 50
 reconstructed image block:

50	50	50	50
210	210	210	210
130	210	130	210
130	210	130	210

mse= 650.0 entropy= -1109.

*** BLOCK TRUNCATION CODING ***

a= 60 b= 220
 reconstructed image block:

60	60	60	60
220	220	220	220
60	220	60	220
60	220	60	220

mse= 1550. entropy= -1033.

*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***

a= 70 b= 210
 reconstructed image block:

70	70	70	70
210	210	210	210
70	210	70	210
70	210	70	210

mse= 1450. entropy= -1025.

*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***

a= 55 b= 225
 reconstructed image block:

55	55	55	55
225	225	225	225
55	225	55	225
55	225	55	225

mse= 1675. entropy= -1038.

sample image block:

100	200	150	50
78	82	29	240
40	30	20	60
100	110	120	130

mean= 96.19 standard deviation= 60.27
 absolute moment= 47.56 entropy= -660.2
 ones= 8 zeroes= 8

*** THREE-LEVEL BLOCK TRUNCATION CODING ***

highest= 240 hi_mean= 166.7
 lowest = 20 lo_mean= 93.33
 range= 73.33 b= 203.3 mid= 130 a= 56
 reconstructed image block:

130	203	130	56
56	56	56	203
56	56	56	56
130	130	130	130

mse= 515.4 entropy= -699.5

*** BLOCK TRUNCATION CODING ***

a= 36 b= 156
 reconstructed image block:

156	156	156	36
36	36	36	156
36	36	36	36
156	156	156	156

mse= 1525. entropy= -661.3

*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***

a= 49 b= 144
 reconstructed image block:

144	144	144	49
49	49	49	144
49	49	49	49
144	144	144	144

mse= 1370. entropy= -653.8

*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***

a= 39 b= 153
 reconstructed image block:

153	153	153	39
39	39	39	153
39	39	39	39
153	153	153	153

mse= 1459. entropy= -658.3

sample image block:

100	124	167	234
87	136	198	247
65	120	167	227
45	115	132	219

mean= 148.9 standard deviation= 60.24
 absolute moment= 52.05 entropy= -1094.
 ones= 7 zeroes= 9

*** THREE-LEVEL BLOCK TRUNCATION CODING ***

highest= 247 hi_mean= 179.7
 lowest = 45 lo_mean= 112.3
 range= 67.33 b= 213.3 mid= 146 a= 78
 reconstructed image block:

78	146	146	213
78	146	213	213
78	146	146	213
78	146	146	213

mse= 448.5 entropy= -1097.

*** BLOCK TRUNCATION CODING ***

a= 96 b= 217
 reconstructed image block:

96	96	217	217
96	96	217	217
96	96	217	217
96	96	96	217

mse= 933.1 entropy= -1092.

*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***

a= 103 b= 208
 reconstructed image block:

103	103	208	208
103	103	208	208
103	103	208	208
103	103	103	208

mse= 876.1 entropy= -1088.

*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***

a= 94 b= 219
 reconstructed image block:

94	94	219	219
94	94	219	219
94	94	219	219
94	94	94	219

mse= 967.1 entropy= -1091.

sample image block:

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mean= 254.9 standard deviation= .3307
 absolute moment= .2188 entropy= -2037.
 ones= 14 zeroes= 2

*** THREE-LEVEL BLOCK TRUNCATION CODING ***

highest= 255 hi_mean= 255.0
 lowest = 254 lo_mean= 254.0
 range= .3333 b= 255.0 mid= 254 a= 254
 reconstructed image block:

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mse= .0000 entropy= -2037.

*** BLOCK TRUNCATION CODING ***

a= 254 b= 255
 reconstructed image block:

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mse= .0000 entropy= -2037.

*** ABSOLUTE MOMENT BLOCK TRUNCATION CODING ***

a= 254 b= 255
 reconstructed image block:

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mse= .0000 entropy= -2037.

*** ENTROPY-PRESERVING BLOCK TRUNCATION CODING ***

a= 254 b= 255
 reconstructed image block:

255	255	255	255
255	255	254	254
255	255	255	255
255	255	255	255

mse= .0000 entropy= -2037.

APPENDIX E
PASCAL LISTINGS OF COLOR IMAGE COMPUTER APPLICATION


```
( *****
*
*  PROCEDURE NAME: Block Truncation Coding Schemes
*
*  PURPOSE: This is the main procedure that performs the BTC on a pixel
*           image and then reconstructs the image by reversing the coding process.
*
*  PARAMETERS PASSED:
*    none.
*
*  PROGRAMMER: Deborah A. Lee
*
*  DATE WRITEN: 09-01-88
*
***** )
```

```
PROGRAM BTC_main;
```

```
{ $R - }
```

```
( *****
*
*  Define the image size constants. It is assumed that the image is a
*  square matrix of pixels.
*
***** )
```

```
CONST    pixsize   = 256;
          pixsize_1 = 255;
          pixsize_2 = 254;
          fourth    = 64;
          fourth_1  = 63;
```

```
( *****
*
*  Define the image matrix and coded matrix array types.
*
***** )
```

```
TYPE codes   = ARRAY [0..49, 0..fourth_1] OF REAL;
    codesptr = ^codes;
    picture  = ARRAY [0..199, 0..pixsize_1] OF BYTE;
    pixptr   = ^picture;
    hdrrec   = ARRAY [0..8] OF INTEGER;
    answer   = STRING[1];
```

```
(*****
*
* Define the variables used:
*
*****)
```

```
*****
*
*
*
*****)
```

```
VAR   red, green, blue      : pixptr;
      redmean, bluemean    : codesptr;
      greenmean             : codesptr;
      redmoment             : codesptr;
      greenmoment           : codesptr;
      bluement             : codesptr;
      redentropy            : codesptr;
      greenentropy          : codesptr;
      blueentropy           : codesptr;
      redlorn               : codesptr;
      greenlorn             : codesptr;
      bluelforn             : codesptr;
      xsize, xsize_1        : integer;
      ysize, ysize_1        : integer;
      bsize                 : integer;
      blocks_x, blocks_y    : integer;
      hdr                   : hdrrec;
      choice                : answer;
```

```
(*****
*
* Include the BTC subroutines.
*
*****)
```

```
*****
*
*
*
*****)
```

```
{ $I BTC_Menu.pas }
{ $I Get_Blksz.pas }
{ $I loadrgb.pas }
{ $I readrgb.pas }
{ $I writrgb.pas }
{ $I am_code.pas }
{ $I am_unicode.pas }
{ $I btc_code.pas }
{ $I btc_unicode.pas }
{ $I ent_code.pas }
{ $I ent_unicode.pas }
{ $I rng_code.pas }
{ $I rng_unicode.pas }
```

```

( *****
*
* Let's begin processing.....
* Allocate the dynamic heap area for storing the picture image.
*
***** )

BEGIN
    new (red);
    new (green);
    new (blue);

( *****
*
* Let's find out which BTC we are to use....
*
***** )

    REPEAT
        BTC_Menu;

( *****
*
* Let's see what blocking size we are to use.
* It must divide evenly into 256.
*
***** )

        IF (choice < 'E') THEN
            BEGIN
                Get_Blksz;

( *****
*
* Let's find out which file we are supposed to code.....
*
***** )

                ReadRGB;

( *****
*
* Now allocate the heap memory for the coded blocks mean and moment values.
* Then code the image!
*
***** )

                new (redmean);
                new (greenmean);
                new (bluemean);

```



```
Writeln ('Coding the image....');
```

```
IF ((choice = 'A') or (choice = 'B')) THEN
BEGIN
```

```
    new (redmoment);
    new (greenmoment);
    new (bluemoment);
```

```
IF (choice = 'A') THEN
```

```
    BTC_Code
```

```
ELSE
```

```
    AM_Code;
```

```
END;
```

```
IF (choice = 'C') THEN
```

```
BEGIN
```

```
    new (redentropy);
    new (greenentropy);
    new (blueentropy);
    Ent_Code;
```

```
END;
```

```
IF (choice = 'D') THEN
```

```
BEGIN
```

```
    new (redlorng);
    new (greenlorng);
    new (bluelorng);
    Rng_Code;
```

```
END;
```

```
( *****
```

```
*
```

```
* Now, let's see how well we can reconstruct the image.
```

```
*
```

```
* When we're done release the heap memory for the coded variables.
```

```
*
```

```
*****
```

```
*****
```

```
*
```

```
*
```

```
*
```

```
*
```

```
*
```

```
*****)
```

```
Writeln ('Reconstructing the image from the coded image.....');
```

```
IF (choice = 'A') THEN
```

```
    BTC_Uncode;
```

```
IF (choice = 'B') THEN
```

```
    AM_Uncode;
```

```
IF (choice = 'C') THEN
```

```
BEGIN
```

```
    Ent_Uncode;
    dispose (redentropy);
    dispose (greenentropy);
    dispose (blueentropy);
```

```
END;
```

```

IF (choice = 'D') THEN
BEGIN
    Rng_Uncode;
    dispose (redlorng);
    dispose (greenlorng);
    dispose (bluelorng);
END;

IF ((choice = 'A') or (choice = 'B')) THEN
BEGIN
    dispose (redmoment);
    dispose (greenmoment);
    dispose (bluemoment);
END;

dispose (redmean);
dispose (greenmean);
dispose (bluemean);

```

```

( *****
*
* We're done restoring so save the restored image in another file.
* Then release the rest of the heap memory allocated for the image.
*
*****

```

```

*****
*
*
*
*
***** )

```

```

        WritRGB;
    END;
UNTIL (choice = 'E');

dispose (red);
dispose (green);
dispose (blue);

END.

```

```
( *****
*
* PROCEDURE NAME: Block Truncation Coding Main Menu
*
* PURPOSE: This is the routine that displays the main menu for the Block
* Truncation Coding program.
*
* PARAMETERS PASSED:
* none.
*
* PROGRAMMER: Deborah A. Lee
*
* DATE WRITTEN: 08-21-88
*
***** )
```

```
PROCEDURE BTC_Menu;
```

```
BEGIN
```

```
( *****
* Display the main Menu.
***** )
```

```
TextMode (c80);
ClrScr;
TextColor (LightMagenta);
```

```
Writeln ('-----');
Writeln ('          BLOCK TRUNCATION CODING PROGRAM');
Writeln ('-----');
Writeln;
```

```
TextColor (LightGreen);
Writeln ('          Your options are:');
TextColor (Yellow);
Write ('          A) ');
TextColor (LightCyan);
Writeln ('Original BTC');
TextColor (Yellow);
Write ('          B) ');
TextColor (LightCyan);
Writeln ('Absolute-Moment BTC');
TextColor (Yellow);
Write ('          C) ');
TextColor (LightCyan);
Writeln ('Entropy-Preserving BTC');
TextColor (Yellow);
Write ('          D) ');
TextColor (LightCyan);
Writeln ('Three-Level BTC');
```



```

TextColor (Yellow);
Write ('          E) ');
TextColor (LightCyan);
Writeln ('Quit (Return to DOS)');
TextColor (LightGreen);
Write ('          Please enter your choice: ');

```

```

(****
* Find out what processing technique to apply.
*****

```

```

* * * * *
      *
* * * * * )

```

```

Repeat
  TextColor (Yellow);
  Readln (choice);
  choice := UpCase (choice);

  If ((choice < 'A') or (choice > 'E')) Then
    Begin
      Writeln;
      TextColor (LightRed);
      Write ('  You must enter one of the option letters from above. ');
      Delay (1000);
      DelLine;
      GotoXY (37,11);  ClrEOL;
    End;
  Until ((choice >= 'A') and (choice <= 'E'));

  Writeln;
END; { BTC_Menu }

```

```
PROCEDURE Get_Blksz;
```

```
VAR   good_size      : boolean;
      size, rblock    : real;
```

```
BEGIN
```

```
    good_size := FALSE;
```

```
    Write ('    Enter the x size of the block that you wish to code:');
    REPEAT
```

```
        ReadLn (rblock);
```

```
        size := 200;
```

```
        size := size / rblock;
```

```
        IF (Frac (size) = 0) THEN
```

```
            good_size := TRUE
```

```
        ELSE BEGIN
```

```
            Writeln;
```

```
            TextColor (LightRed);
```

```
            Write ('    The block size MUST divide evenly into 200.');
```

```
            Delay (2000);
```

```
            DelLine;
```

```
            GotoXY (58,12);    ClrEOL;
```

```
        END;
```

```
    UNTIL (good_size);
```

```
    xsize      := TRUNC (rblock);
```

```
    xsize_1     := xsize - 1;
```

```
    good_size := FALSE;
```

```
    Write ('    Enter the y size of the block that you wish to code:');
    REPEAT
```

```
        ReadLn (rblock);
```

```
        size := pixsize;
```

```
        size := size / rblock;
```

```
        IF (Frac (size) = 0) THEN
```

```
            good_size := TRUE
```

```
        ELSE BEGIN
```

```
            Writeln;
```

```
            TextColor (LightRed);
```

```
            Write ('    The block size MUST divide evenly into 256.');
```

```
            Delay (2000);
```

```
            DelLine;
```

```
            GotoXY (58,12);    ClrEOL;
```

```
        END;
```

```
    UNTIL (good_size);
```

```
    ysize      := TRUNC (rblock);
```

```
    ysize_1     := ysize - 1;
```

```
    bsize      := xsize * ysize;
```

```
    blocks_x   := (200 DIV xsize) - 1;
```

```
    blocks_y   := (256 DIV ysize) - 1;
```

```
END; {Get_Blksz }
```

```
( *****
*
* PROCEDURE NAME: Read an Image Pixel File
*
* PURPOSE: This is procedure reads in a 200 x 256 2-byte image file and
* places the image pixel data into the heap allocated pixel data arrays.
*
* PARAMETERS PASSED:
* none.
*
* PROGRAMMER: Deborah A. Lee
*
* DATE WRITEN: 06-25-87
*
***** )
```

```
PROCEDURE ReadRGB;
```

```
{ $R - }
```

```
TYPE filename = string [25];
```

```
VAR i, j, p : integer;
    inpix : file of integer;
    pixname : filename;
```

```
BEGIN
```

```
    REPEAT
```

```
        pixname := ' ';
        Write ('Enter the name of the picture you wish to read: ');
        Readln (pixname);
```

```
    IF ((pixname <> ' ') AND (pixname <> '')) THEN
        BEGIN
```

```
            Assign (inpix, pixname);
            Reset (inpix);
```

```
            FOR i := 0 TO 8 DO
                Read (inpix, hdr [i]);
```

```
            FOR i := 0 TO 199 DO
                FOR j := 0 TO pixsize_1 DO
                    BEGIN
                        read (inpix, p);
                        LoadRGB (p, red^[i,j], green^[i,j], blue^[i,j]);
                    END;
```

```
            Close (inpix);
```

```
        END;
```

```
    UNTIL (pixname <> ' ');
```

```
END; { ReadRGB }
```



```
PROCEDURE LoadRGB (p:INTEGER; VAR redval,greenval,blueval:BYTE);
```

```
CONST redmask      = 31744;  
      greenmask    = 992;  
      bluemask     = 31;
```

```
BEGIN  
    redval      := (p AND redmask) DIV 1024;  
    greenval    := (p AND greenmask) DIV 32;  
    blueval     := p AND bluemask;  
END; {LoadRGB}
```

```
(
*****
*
*  PROCEDURE NAME: Write an Image Pixel File
*
*  PURPOSE: This is procedure writes a 200 x 256 2-byte image file.
*
*  PARAMETERS PASSED:
*    none.
*
*  PROGRAMMER: Deborah A. Lee
*
*  DATE WRITEN: 06-25-87
*
*****
)
```

```
procedure WritRGB;
{$R-}
```

```
type  filename = string [25];
```

```
var   i, j, p : integer;
      outpix  : file of integer;
      pixname : filename;
```

```
begin
```

```
  repeat
```

```
    pixname := ' ';
    write ('Enter the name of the picture you wish to create: ');
    readln (pixname);
```

```
    if ((pixname <> ' ') and (pixname <> '')) then
      begin
```

```
        assign (outpix, pixname);
        rewrite (outpix);
```

```
        FOR i := 0 TO 8 DO
          Write (outpix, hdr [i]);
```

```
        FOR i := 0 TO 199 DO
          FOR j := 0 TO pixsize_1 DO
            BEGIN
```

```
              p := (red^[i,j] * 1024) + (green^[i,j] * 32) + blue^[i,j];
              Write (outpix, p);
```

```
            END;
```

```
          close (outpix);
```

```
        end;
```

```
      until (pixname <> ' ');
```

```
end; { WritRGB }
```

```
PROCEDURE BTC_Code;
```

```
VAR   i, j, k ,l : integer;
```

```
BEGIN
```

```
  FOR k := 0 TO blocks_x DO
```

```
    FOR l := 0 TO blocks_y DO
```

```
      BEGIN
```

```
        redmean^ [k,l] := 0.0;
```

```
        greenmean^[k,l] := 0.0;
```

```
        bluemean^ [k,l] := 0.0;
```

```
        redmoment^ [k,l] := 0.0;
```

```
        greenmoment^[k,l] := 0.0;
```

```
        bluement^ [k,l] := 0.0;
```

```
      FOR i := 0 TO xsize_1 DO
```

```
        FOR j := 0 TO ysize_1 DO
```

```
          BEGIN
```

```
            redmean^ [k,l] := redmean^ [k,l] + red^ [i+(k*xsize),  
                                                    j+(l*ysize)];
```

```
            greenmean^[k,l] := greenmean^[k,l] + green^[i+(k*xsize),  
                                                         j+(l*ysize)];
```

```
            bluemean^ [k,l] := bluemean^ [k,l] + blue^ [i+(k*xsize),  
                                                         j+(l*ysize)];
```

```
          END;
```

```
        redmean^ [k,l] := redmean^ [k,l] / bsize;
```

```
        greenmean^[k,l] := greenmean^[k,l] / bsize;
```

```
        bluemean^ [k,l] := bluemean^ [k,l] / bsize;
```

```
      FOR i := 0 TO xsize_1 DO
```

```
        FOR j := 0 TO ysize_1 DO
```

```
          BEGIN
```

```
            redmoment^ [k,l] := redmoment^ [k,l] +  
            SQR(red^[i+(k*xsize),j+(l*ysize)] - redmean^ [k,l]);
```

```
            greenmoment^[k,l] := greenmoment^[k,l] +  
            SQR(green^[i+(k*xsize),j+(l*ysize)] - greenmean^[k,l]);
```

```
            bluement^ [k,l] := bluement^ [k,l] +  
            SQR(blue^ [i+(k*xsize),j+(l*ysize)] - bluement^ [k,l]);
```

```
          END;
```

```
        redmoment^ [k,l] := SQRT(redmoment^ [k,l] / bsize);
```

```
        greenmoment^[k,l] := SQRT(greenmoment^[k,l] / bsize);
```

```
        bluement^ [k,l] := SQRT(bluement^ [k,l] / bsize);
```



```

FOR i := 0 TO xsize_1 DO
  FOR j := 0 TO ysize_1 DO
    BEGIN
      IF (red^[i+(k*xsize),j+(l*ysize)] >= redmean^[k,l]) THEN
        red^[i+(k*xsize),j+(l*ysize)] := 1
      ELSE
        red^[i+(k*xsize),j+(l*ysize)] := 0;

      IF (green^[i+(k*xsize),j+(l*ysize)] >= greenmean^[k,l]) THEN
        green^[i+(k*xsize),j+(l*ysize)] := 1
      ELSE
        green^[i+(k*xsize),j+(l*ysize)] := 0;

      IF (blue^[i+(k*xsize),j+(l*ysize)] >= bluemean^[k,l]) THEN
        blue^[i+(k*xsize),j+(l*ysize)] := 1
      ELSE
        blue^[i+(k*xsize),j+(l*ysize)] := 0;
    END;
  END;
END; { BTC_Code }

```

```
PROCEDURE BTC_Uncode;
```

```
VAR i, j, k ,l      : integer;
    redones, redzeroes : real;
    greenones, greenzeroes : real;
    blueones, bluezeroes : real;
    reda, redb        : integer;
    greena, greenb     : integer;
    bluea, blueb       : integer;
```

BEGIN

```
FOR k := 0 TO blocks_x DO
  FOR l := 0 TO blocks_y DO
```

BEGIN

```
redones    := 0.0;  
greenones  := 0.0;  
blueones   := 0.0;
```

```
FOR i := 0 TO xsize_1 DO
  FOR j := 0 TO ysize_1 DO
```

BEGIN

```

IF (red^[i+(k*xsize),j+(l*ysize)] = 1) THEN
  redones := redones + 1.0;
IF (green^[i+(k*xsize),j+(l*ysize)] = 1) THEN
  greenones := greenones + 1.0;
IF (blue^[i+(k*xsize),j+(l*ysize)] = 1) THEN
  blueones := blueones + 1.0;

```

END;

```
redzeroes := bsize - redones;
greenzeroes := bsize - greenones;
bluezeroes := bsize - blueones;
```

[illegible][illegible]

```
IF (greenones > 0) THEN
    greenb := Round (greenmean^[k,l] + (greenmoment^[k,l] *
        SQRT(greenzeroes/greenones)));
```

```
IF (greenzeroes > 0) THEN
  greena := Round (greenmean^ [k,l] - (greenmoment^[k,l] *
    SQRT(greenones/greenzeroes)));
```

```

IF (blueones > 0) THEN
  blue := Round (bluemean^[k,l] + (bluemoment^[k,l] *
    SQRT(bluezeroes/blueones)));

```

```

IF (bluezeroes > 0) THEN
    bluea := Round (bluemean^ [k,l] - (bluemoment^[k,l] *
                                                Sqrt(blueones/bluezeroes)));
IF (redb > 31) THEN redb := 31;
IF (greenb > 31) THEN greenb := 31;
IF (blueb > 31) THEN blueb := 31;

IF (reda < 0) THEN reda := 0;
IF (greena < 0) THEN greena := 0;
IF (bluea < 0) THEN bluea := 0;

FOR i := 0 TO xsize_1 DO
    FOR j := 0 TO ysize_1 DO
        BEGIN
            IF (red^[i+(k*xsize),j+(l*ysize)] = 1) THEN
                red^[i+(k*xsize),j+(l*ysize)] := redb
            ELSE
                red^[i+(k*xsize),j+(l*ysize)] := reda;

            IF (green^[i+(k*xsize),j+(l*ysize)] = 1) THEN
                green^[i+(k*xsize),j+(l*ysize)] := greenb
            ELSE
                green^[i+(k*xsize),j+(l*ysize)] := greena;

            IF (blue^[i+(k*xsize),j+(l*ysize)] = 1) THEN
                blue^[i+(k*xsize),j+(l*ysize)] := blueb
            ELSE
                blue^[i+(k*xsize),j+(l*ysize)] := bluea;

        END;
    END;
END; { BTC_Uncode }

```



```
PROCEDURE AM_Code;
```

```
VAR i, j, k, l : integer;
```

```
BEGIN
```

```
  FOR k := 0 TO blocks_x DO
```

```
    FOR l := 0 TO blocks_y DO
```

```
      BEGIN
```

```
        redmean^ [k,l] := 0.0;
```

```
        greenmean^[k,l] := 0.0;
```

```
        bluemean^ [k,l] := 0.0;
```

```
        redmoment^ [k,l] := 0.0;
```

```
        greenmoment^[k,l] := 0.0;
```

```
        bluement^ [k,l] := 0.0;
```

```
      FOR i := 0 TO xsize_1 DO
```

```
        FOR j := 0 TO ysize_1 DO
```

```
          BEGIN
```

```
            redmean^ [k,l] := redmean^ [k,l] +  
                                red^ [i+(k*xsize),j+(l*ysize)];
```

```
            greenmean^ [k,l] := greenmean^[k,l] +  
                                green^[i+(k*xsize),j+(l*ysize)];
```

```
            bluemean^ [k,l] := bluemean^ [k,l] +  
                                blue^ [i+(k*xsize),j+(l*ysize)];
```

```
          END;
```

```
        redmean^ [k,l] := redmean^ [k,l] / bsize;
```

```
        greenmean^[k,l] := greenmean^[k,l] / bsize;
```

```
        bluemean^ [k,l] := bluemean^ [k,l] / bsize;
```

```
      FOR i := 0 TO xsize_1 DO
```

```
        FOR j := 0 TO ysize_1 DO
```

```
          BEGIN
```

```
            redmoment^ [k,l] := redmoment^ [k,l] +  
                                ABS(red^ [i+(k*xsize),j+(l*ysize)] - redmean^ [k,l]);
```

```
            greenmoment^[k,l] := greenmoment^[k,l] +  
                                ABS(green^[i+(k*xsize),j+(l*ysize)] - greenmean^[k,l]);
```

```
            bluement^ [k,l] := bluement^ [k,l] +  
                                ABS(blue^ [i+(k*xsize),j+(l*ysize)] - bluemean^ [k,l]);
```

```
          END;
```

```
        redmoment^ [k,l] := redmoment^ [k,l] / 2;
```

```
        greenmoment^[k,l] := greenmoment^[k,l] / 2;
```

```
        bluement^ [k,l] := bluement^ [k,l] / 2;
```

```

FOR i := 0 TO xsize_1 DO
  FOR j := 0 TO ysize_1 DO
    BEGIN
      IF (red^[i+(k*xsize),j+(l*ysize)] >= redmean^[k,l]) THEN
        red^[i+(k*xsize),j+(l*ysize)] := 1
      ELSE
        red^[i+(k*xsize),j+(l*ysize)] := 0;
      IF (green^[i+(k*xsize),j+(l*ysize)] >= greenmean^[k,l]) THEN
        green^[i+(k*xsize),j+(l*ysize)] := 1
      ELSE
        green^[i+(k*xsize),j+(l*ysize)] := 0;

      IF (blue^[i+(k*xsize),j+(l*ysize)] >= bluemean^[k,l]) THEN
        blue^[i+(k*xsize),j+(l*ysize)] := 1
      ELSE
        blue^[i+(k*xsize),j+(l*ysize)] := 0;
    END;
  END;
END; { AM_Code }

```

```
PROCEDURE AM_Uncode;
```

```
VAR    i, j, k ,l      : integer;
        redones, redzeroes : real;
        greenones, greenzeroes : real;
        blueones, bluezeroes : real;
        reda, redb      : integer;
        greena, greenb   : integer;
        bluea, blueb     : integer;
```

BEGIN

```
FOR k := 0 TO blocks_x DO
  FOR l := 0 TO blocks_y DO
```

BEGIN

```
redones    := 0.0;
greenones  := 0.0;
blueones   := 0.0;
```

```
FOR i := 0 TO xsize_1 DO
  FOR j := 0 TO ysize_1 DO
```

BEGIN

```

IF (red^[i+(k*xsize),j+(l*ysize)] = 1) THEN
  redones := redones + 1.0;
IF (green^[i+(k*xsize),j+(l*ysize)] = 1) THEN
  greenones := greenones + 1.0;
IF (blue^[i+(k*xsize),j+(l*ysize)] = 1) THEN
  blueones := blueones + 1.0;

```

END;

```
redzeroes := bsize - redones;
greenzeroes := bsize - greenones;
bluezeroes := bsize - blueones;
```

IF (redones > 0.0) THEN

```
redb := Round (redmean^[k,l] + (redmoment^[k,l] / redones));
```

```
IF (redzeroes > 0.0) THEN
```

```
reda := Round (redmean^[k,l] - (redmoment^[k,l] / redzeroes));
```

IF (greenones > 0.0) THEN

```
greenb := Round (greenmean^ [k,l] + (greenmoment^[k,l] /
```

```
                                greenones));
```

IF (greenzeroes > 0.0) THEN

```
greena := Round (greenmean^ [k,l] - (greenmoment^[k,l] /  
greenzeroes));
```

IF (blueones > 0.0) THEN

```
blueb := Round (bluemean^ [k,l] + (bluemoment^[k,l] / blueones));
```

```
IF (bluezeroes > 0.0) THEN
```

```
bluea := Round (bluemoment^[k,l] - (bluemoment^[k,l] / bluezeroes));
```



```

FOR i := 0 TO xsize_1 DO
  FOR j := 0 TO ysize_1 DO
    BEGIN
      IF (red^[i+(k*xsize),j+(l*ysize)] >= 1) THEN
        red^[i+(k*xsize),j+(l*ysize)] := redb
      ELSE
        red^[i+(k*xsize),j+(l*ysize)] := reda;

      IF (green^[i+(k*xsize),j+(l*ysize)] >= 1) THEN
        green^[i+(k*xsize),j+(l*ysize)] := greenb
      ELSE
        green^[i+(k*xsize),j+(l*ysize)] := greena;

      IF (blue^[i+(k*xsize),j+(l*ysize)] >= 1) THEN
        blue^[i+(k*xsize),j+(l*ysize)] := blueb
      ELSE
        blue^[i+(k*xsize),j+(l*ysize)] := bluea;
    END;
  END;
END; { AM_Uncode }

```

```
PROCEDURE Ent_Code;
```

```
VAR   i, j, k ,l : integer;
      log2_red   : real;
      log2_green : real;
      log2_blue  : real;
```

```
BEGIN
```

```
  FOR k := 0 TO blocks_x DO
    FOR l := 0 TO blocks_y DO
      BEGIN
```

```
        redmean^ [k,l]   := 0.0;
        greenmean^[k,l] := 0.0;
        bluemean^ [k,l]  := 0.0;
```

```
        redentropy^ [k,l]   := 0.0;
        greenentropy^[k,l] := 0.0;
        blueentropy^ [k,l]  := 0.0;
```

```
        FOR i := 0 TO xsize_1 DO
          FOR j := 0 TO ysize_1 DO
            BEGIN
```

```
              redmean^ [k,l] := redmean^ [k,l] + ABS(red^ [i+(k*xsize),
                                                         j+(l*ysize)]);
              greenmean^[k,l] := greenmean^[k,l] + ABS(green^[i+(k*xsize),
                                                         j+(l*ysize)]);
              bluemean^ [k,l] := bluemean^ [k,l] + ABS(blue^ [i+(k*xsize),
                                                         j+(l*ysize)]);
```

```
            END;
```

```
        redmean^ [k,l]   := redmean^ [k,l] / bsize;
        greenmean^[k,l] := greenmean^[k,l] / bsize;
        bluemean^ [k,l]  := bluemean^ [k,l] / bsize;
```

```
        FOR i := 0 TO xsize_1 DO
          FOR j := 0 TO ysize_1 DO
            BEGIN
```

```
              IF (red^[i+(k*xsize),j+(l*ysize)] > 0) THEN
                log2_red := LN (ABS(red^[i+(k*xsize),j+(l*ysize)])) /
                               ln (2.0)
```

```
              ELSE
                log2_red := 0.0;
```

```
              IF (green^[i+(k*xsize),j+(l*ysize)] > 0) THEN
                log2_green := LN (ABS(green^ [i+(k*xsize),j+(l*ysize)]))
                               / ln (2.0)
```

```
              ELSE
                log2_green := 0.0;
```

```

IF (blue^[i+(k*xsize),j+(l*ysize)] > 0) THEN
    log2_blue := LN (ABS(blue^[i+(k*xsize),j+(l*ysize)])) /
                                     ln (2.0)
ELSE
    log2_blue := 0.0;

redentropy^[k,l] := redentropy^[k,l] +
    (ABS(red^[i+(k*xsize),j+(l*ysize)]) * log2_red);
greenentropy^[k,l] := greenentropy^[k,l] +
    (ABS(green^[i+(k*xsize),j+(l*ysize)]) * log2_green);
blueentropy^[k,l] := blueentropy^[k,l] +
    (ABS(blue^[i+(k*xsize),j+(l*ysize)]) * log2_blue);
END;

redentropy^[k,l] := redentropy^[k,l] * (-1.0 / bsize);
greenentropy^[k,l] := greenentropy^[k,l] * (-1.0 / bsize);
blueentropy^[k,l] := blueentropy^[k,l] * (-1.0 / bsize);

FOR i := 0 TO xsize_1 DO
    FOR j := 0 TO ysize_1 DO
        BEGIN
            IF (red^[i+(k*xsize),j+(l*ysize)] >= redmean^[k,l]) THEN
                red^[i+(k*xsize),j+(l*ysize)] := 1
            ELSE
                red^[i+(k*xsize),j+(l*ysize)] := 0;

            IF (green^[i+(k*xsize),j+(l*ysize)] >= greenmean^[k,l])
                THEN
                green^[i+(k*xsize),j+(l*ysize)] := 1
            ELSE
                green^[i+(k*xsize),j+(l*ysize)] := 0;

            IF (blue^[i+(k*xsize),j+(l*ysize)] >= bluemean^[k,l])
                THEN
                blue^[i+(k*xsize),j+(l*ysize)] := 1
            ELSE
                blue^[i+(k*xsize),j+(l*ysize)] := 0;
        END;
    END;
END; { Ent_Code }

```



```
PROCEDURE Ent_Uncode;
```

```
VAR   i, j, k ,l           : integer;
      redones, redzeroes   : real;
      greenones, greenzeroes : real;
      blueones, bluezeroes  : real;
      a, b                 : real;
      reda, redb           : integer;
      greena, greenb        : integer;
      bluea, blueb          : integer;
      qb, mu, functb        : real;
      log2_a, log2_b        : real;
```

```
BEGIN
```

```
  FOR k := 0 TO blocks_x DO
    FOR l := 0 TO blocks_y DO
      BEGIN
```

```
        redones := 0.0;
        greenones := 0.0;
        blueones := 0.0;
```

```
        FOR i := 0 TO xsize_1 DO
          FOR j := 0 TO ysize_1 DO
            BEGIN
```

```
              IF (red^[i+(k*xsize),j+(l*ysize)] = 1) THEN
                redones := redones + 1.0;
              IF (green^[i+(k*xsize),j+(l*ysize)] = 1) THEN
                greenones := greenones + 1.0;
              IF (blue^[i+(k*xsize),j+(l*ysize)] = 1) THEN
                blueones := blueones + 1.0;
```

```
            END;
```

```
        redzeroes := bsize - redones;
        greenzeroes := bsize - greenones;
        bluezeroes := bsize - blueones;
```

```
        b := redmean^ [k,l];
        mu := bsize * redmean^[k,l];
        functb := 1.0;
        WHILE (functb > 0.0) DO
          BEGIN
            b := b + 0.01;
            qb := redones * b;
            IF (b > 0.0) THEN
              log2_b := LN (b) / LN (2.0)
            ELSE BEGIN
              b := 0.0;
              log2_b := 0.0;
            END;
```

```

IF (redzeroes > 0.0) THEN
    a := (mu - qb) / redzeroes
ELSE
    a := 0.0;
IF (a > 0.0) THEN
    log2_a := LN (a) / LN (2.0)
ELSE BEGIN
    log2_a := 0.0;
    a := 0.0;
END;
functb := ((qb-mu)*log2_a)-(qb*log2_b)-
                                                (bsize*redentropy^[k,l]);
END;

IF (functb < 0.0) THEN
BEGIN
    b := b - 0.01;
    a := (mu - (redones * b)) / redzeroes;
END;

redb := TRUNC (b);
reda := TRUNC (a);

b := greenmean^ [k,l];
mu := bsize * greenmean^[k,l];
functb := 1.0;
WHILE (functb > 0.0) DO
BEGIN
    b := b + 0.01;
    qb := redones * b;
    IF (b > 0.0) THEN
        log2_b := LN (b) / LN (2.0)
    ELSE
        log2_b := 0.0;
    qb := greenones * b;
    IF (greenzeroes > 0.0) THEN
        a := (mu - qb) / greenzeroes
    ELSE
        a := 0.0;
    IF (a > 0.0) THEN
        log2_a := LN (a) / LN (2.0)
    ELSE BEGIN
        a := 0.0;
        log2_a := 0.0;
    END;
    functb := ((qb - mu) * log2_a) - (qb * log2_b) -
                                                (bsize*greenentropy^ [k,l]);
END;
END;

```

```

IF (functb < 0.0) THEN
BEGIN
    b := b - 0.01;
    a := (mu - (greenones * b)) / greenzeroes;
END;
greenb := TRUNC (b);
greena := TRUNC (a);

b := bluemean^ [k,l];
mu := bsize * bluemean^[k,l];
functb := 1.0;
WHILE (functb > 0.0) DO
BEGIN
    b := b + 0.01;
    qb := blueones * b;
    IF (b > 0.0) THEN
        log2_b := LN (b) / LN (2.0)
    ELSE
        log2_b := 0.0;
    IF (bluezeroes > 0.0) THEN
        a := (mu - qb) / bluezeroes
    ELSE
        a := 0.0;
    IF (a > 0.0) THEN
        log2_a := LN (a) / LN (2.0)
    ELSE BEGIN
        a := 0.0;
        log2_a := 0.0;
    END;
    functb := ((qb - mu) * log2_a) - (qb * log2_b) -
               (bsize*blueentropy^ [k,l]);
END;

IF (functb < 0.0) THEN
BEGIN
    b := b - 0.01;
    a := (mu - (blueones * b)) / bluezeroes;
END;

blueb := TRUNC (b);
bluea := TRUNC (a);

```



```

FOR i := 0 TO xsize_1 DO
  FOR j := 0 TO ysize_1 DO
    BEGIN
      IF (red^[i+(k*xsize),j+(l*ysize)] >= 1) THEN
        red^[i+(k*xsize),j+(l*ysize)] := redb
      ELSE
        red^[i+(k*xsize),j+(l*ysize)] := reda;
      IF (green^[i+(k*xsize),j+(l*ysize)] >= 1) THEN
        green^[i+(k*xsize),j+(l*ysize)] := greenb
      ELSE
        green^[i+(k*xsize),j+(l*ysize)] := greena;
      IF (blue^[i+(k*xsize),j+(l*ysize)] >= 1) THEN
        blue^[i+(k*xsize),j+(l*ysize)] := blueb
      ELSE
        blue^[i+(k*xsize),j+(l*ysize)] := bluea;
    END;
  END;
END; { Ent_Uncode }

```

```
PROCEDURE Rng_Code;
```

```
VAR   i, j, k ,l       : integer;
      hi, lo           : integer;
      redhi, redlo      : integer;
      greenhi, greenlo  : integer;
      bluehi, bluelo    : integer;
      redrng, bluerng   : real;
      greenrng          : real;
```

```
BEGIN
```

```
  FOR k := 0 TO blocks_x DO
```

```
    FOR l := 0 TO blocks_y DO
```

```
      BEGIN
```

```
        redmean^ [k,l] := 0.0;
```

```
        greenmean^[k,l] := 0.0;
```

```
        bluemean^ [k,l] := 0.0;
```

```
        redlorng^ [k,l] := 255.0;
```

```
        greenlorng^[k,l] := 255.0;
```

```
        bluelorng^ [k,l] := 255.0;
```

```
      FOR i := 0 TO xsize_1 DO
```

```
        FOR j := 0 TO ysize_1 DO
```

```
          BEGIN
```

```
            IF (red^ [i+(k*xsize),j+(l*ysize)] > redmean^ [k,l]) THEN
              redmean^ [k,l] := red^ [i+(k*xsize),j+(l*ysize)];
```

```
            IF (green^ [i+(k*xsize),j+(l*ysize)] > greenmean^ [k,l]) THEN
              greenmean^[k,l] := green^[i+(k*xsize),j+(l*ysize)];
```

```
            IF (blue^ [i+(k*xsize),j+(l*ysize)] > bluemean^ [k,l]) THEN
              bluemean^ [k,l] := blue^ [i+(k*xsize),j+(l*ysize)];
```

```
            IF (red^ [i+(k*xsize),j+(l*ysize)] < redlorng^ [k,l]) THEN
              redlorng^ [k,l] := red^ [i+(k*xsize),j+(l*ysize)];
```

```
            IF (green^ [i+(k*xsize),j+(l*ysize)] < greenlorng^ [k,l]) THEN
              greenlorng^[k,l] := green^[i+(k*xsize),j+(l*ysize)];
```

```
            IF (blue^ [i+(k*xsize),j+(l*ysize)] < bluelorng^ [k,l]) THEN
              bluelorng^ [k,l] := blue^ [i+(k*xsize),j+(l*ysize)];
```

```
          END;
```

```
      redrng := (redmean^[k,l] - redlorng^[k,l]) / 3.0;
```

```

IF (redrng < 1.0) THEN
BEGIN
    redhi := TRUNC(redmean^[k,l]);
    redlo := TRUNC(redlorng^[k,l]);
END
ELSE BEGIN
    redhi := TRUNC(redmean^[k,l] - redrng);
    redlo := TRUNC(redrng + redlorng^[k,l]);
END;

greenrng := (greenmean^[k,l] - greenlorng^[k,l]) / 3.0;

IF (greenrng < 1.0) THEN
BEGIN
    greenhi := TRUNC(greenmean^[k,l]);
    greenlo := TRUNC(greenlorng^[k,l]);
END
ELSE BEGIN
    greenhi := TRUNC(greenmean^[k,l] - greenrng);
    greenlo := TRUNC(greenrng + greenlorng^[k,l]);
END;

bluerng := (bluemean^[k,l] - bluelorng^[k,l]) / 3.0;

IF (bluerng < 1.0) THEN
BEGIN
    bluehi := TRUNC(bluemean^[k,l]);
    blueho := TRUNC(bluelorng^[k,l]);
END
ELSE BEGIN
    bluehi := TRUNC(bluemean^[k,l] - bluerng);
    blueho := TRUNC(bluerng + bluelorng^[k,l]);
END;

FOR i := 0 TO xsize_1 DO
    FOR j := 0 TO ysize_1 DO
        BEGIN
            IF (red^[i+(k*xsize),j+(l*ysize)] >= redhi) THEN
                red^[i+(k*xsize),j+(l*ysize)] := 2
            ELSE
                IF (red^[i+(k*xsize),j+(l*ysize)] <= redlo) THEN
                    red^[i+(k*xsize),j+(l*ysize)] := 0
                ELSE
                    red^[i+(k*xsize),j+(l*ysize)] := 1;
        END
    END
END

```



```

IF (green^[i+(k*xsize),j+(l*ysize)] >= greenhi) THEN
    green^[i+(k*xsize),j+(l*ysize)] := 2
ELSE
    IF (green^[i+(k*xsize),j+(l*ysize)] <= greenlo) THEN
        green^[i+(k*xsize),j+(l*ysize)] := 0
    ELSE
        green^[i+(k*xsize),j+(l*ysize)] := 1;

IF (blue^[i+(k*xsize),j+(l*ysize)] >= bluehi) THEN
    blue^[i+(k*xsize),j+(l*ysize)] := 2
ELSE
    IF (blue^[i+(k*xsize),j+(l*ysize)] <= bluelo) THEN
        blue^[i+(k*xsize),j+(l*ysize)] := 0
    ELSE
        blue^[i+(k*xsize),j+(l*ysize)] := 1;
END;
END;
END; { Rng_Code }

```

```
PROCEDURE Rng_Uncode;
```

```
VAR   i, j, k ,l           : integer;
      redrng, grnrng, bluerng : real;
      redhi, redlo          : integer;
      greenhi, greenlo       : integer;
      bluehi, bluelo         : integer;
      reda, redb, redmid     : integer;
      greena, greenb, greenmid : integer;
      bluea, blueb, bluemid  : integer;
```

```
BEGIN
```

```
  FOR k := 0 TO blocks_x DO
```

```
    FOR l := 0 TO blocks_y DO
```

```
      BEGIN
```

```
        redrng := (redmean^[k,l] - redlorng^[k,l]) / 3.0;
```

```
        IF (redrng < 1.0) THEN
```

```
          BEGIN
```

```
            redb := TRUNC(redmean^[k,l]);
```

```
            reda := TRUNC(redlorng^[k,l]);
```

```
            redmid := (reda + redb) Div 2;
```

```
          END
```

```
        ELSE BEGIN
```

```
          redhi := TRUNC(redmean^[k,l] - redrng);
```

```
          redlo := TRUNC(redrng + redlorng^[k,l]);
```

```
          redmid := (redhi + redlo) Div 2;
```

```
          redb := TRUNC((redmean^[k,l] + redhi) / 2);
```

```
          reda := TRUNC((redlorng^[k,l] + redlo) / 2);
```

```
        END;
```

```
      grnrng := (greenmean^[k,l] - greenlorng^[k,l]) / 3.0;
```

```
      IF (grnrng < 1.0) THEN
```

```
        BEGIN
```

```
          greenb := TRUNC(greenmean^[k,l]);
```

```
          greena := TRUNC(greenlorng^[k,l]);
```

```
          greenmid := (greena + greenb) Div 2;
```

```
        END
```

```
      ELSE BEGIN
```

```
        greenhi := TRUNC(greenmean^[k,l] - grnrng);
```

```
        greenlo := TRUNC(grnrng + greenlorng^[k,l]);
```

```
        greenmid := (greenhi + greenlo) Div 2;
```

```
        greenb := TRUNC((greenmean^[k,l] + greenhi) / 2);
```

```
        greena := TRUNC((greenlorng^[k,l] + greenlo) / 2);
```

```
      END;
```

```
    bluerng := (bluemean^[k,l] - bluelorng^[k,l]) / 3.0;
```

```

IF (bluerng < 1.0) THEN
BEGIN
    blueb := TRUNC(bluemean^[k,l]);
    bluea := TRUNC(bluelorng^[k,l]);
    bluemid := (bluea + blueb) Div 2;
END
ELSE BEGIN
    bluehi := TRUNC(bluemean^[k,l] - bluerng);
    blueo := TRUNC(bluerng + bluelorng^[k,l]);
    bluemid:= (bluehi + blueo) Div 2;
    blueb := TRUNC((bluemean^[k,l] + bluehi) / 2);
    bluea := TRUNC((bluelorng^[k,l] + blueo) / 2);
END;

FOR i := 0 TO xsize_1 DO
    FOR j := 0 TO ysize_1 DO
        BEGIN
            IF (red^[i+(k*xsize),j+(l*ysize)] = 2) THEN
                red^[i+(k*xsize),j+(l*ysize)] := redb
            ELSE
                IF (red^[i+(k*xsize),j+(l*ysize)] = 1) THEN
                    red^[i+(k*xsize),j+(l*ysize)] := redmid
                ELSE
                    red^[i+(k*xsize),j+(l*ysize)] := reda;

            IF (green^[i+(k*xsize),j+(l*ysize)] = 2) THEN
                green^[i+(k*xsize),j+(l*ysize)] := greenb
            ELSE
                IF (green^[i+(k*xsize),j+(l*ysize)] = 1) THEN
                    green^[i+(k*xsize),j+(l*ysize)] := greenmid
                ELSE
                    green^[i+(k*xsize),j+(l*ysize)] := greena;

            IF (blue^[i+(k*xsize),j+(l*ysize)] = 2) THEN
                blue^[i+(k*xsize),j+(l*ysize)] := blueb
            ELSE
                IF (blue^[i+(k*xsize),j+(l*ysize)] = 1) THEN
                    blue^[i+(k*xsize),j+(l*ysize)] := bluemid
                ELSE
                    blue^[i+(k*xsize),j+(l*ysize)] := bluea;

        END;
    END;
END; { Rng_Uncode }

```


REFERENCES

- [1] Pratt, W.K. 1978. Digital Image Processing. New York: Wiley.
- [2] Delp, Edward J., and Robert O. Mitchell. 1978. Image Compression Using Block Truncation Coding. IEEE Transactions on Communications, 27: 329-336.
- [3] Halverson, D.R., N.C. Griswold, and G.L. Wise. 1987. A Note on Adaptive Block Truncation Coding for Image Processing. IEEE Transactions on Acoustics, Speech, and Signal Processing, 35, no. 8: 1201-1203.
- [4] Lema, Maximo D., and Robert O. Mitchell. 1984. Absolute Moment Block Truncation Coding and Applications to Color Images. IEEE Transactions on Communications, 32: 1148-1157.
- [5] Tzannes, N.S., J.A. Thacker, and M.A. Tzannes. Entropy-Preserving Block Truncation Coding. University of Virginia, Charlottesville. Photocopied.
- [6] Tzannes, N.S. and P.A. Jonnard. 1987. Maximum Entropy Reconstruction of Moment-Coded Images. Optical Engineering, (October).
- [7] Mitchell, Robert O. and Edward J. Delp. 1980. Multilevel Graphics Representation Using Block Truncation Coding. Proceedings of the IEEE, 68, no. 7: 868-873.
- [8] Shoizaki, A. 1986. Edge Extraction Using Entropy Operator. Computer Vision, Graphics, and Image Processing, (March): 1-9.
- [9] Halverson, D.R., N.C. Griswold, and G.L. Wise. 1984. A Generalized Block Truncation Coding Algorithm for Image Compression. IEEE Transactions on Acoustics, Speech, and Signal Processing, 32: 664-668.
- [10] Healy, D.J. and Robert O. Mitchell. 1981. Digital Video Bandwidth Compression Using Block Truncation Coding. IEEE Transactions on Communications, 29: 1809-1817.
- [11] Hall, E.L. 1979. Computer Image Processing and Recognition. New York: Academic Press.
- [12] Oppenheim, A.V. and R.W. Schaffer. 1975. Digital Signal Processing. Englewood Cliffs, N.J.: Prentice-Hall Inc.
- [13] Jain, A.K. 1976. A Fast Karhunen-Loeve Transform for a Class of Random Processes. IEEE Transactions on Communications, (September): 1023-1029.

- [14] Delp, E.J., R.L. Kashyap, and O.R. Mitchell. 1979. Image Data Compression Using Autoregressive Time Series Models. Pattern Recognition, 11: 313-323.
- [15] Modestino, J.W. and V. Bhaskaran. 1981. Robust Two-Dimensional Tree Encoding of Images. IEEE Transactions on Communications, 29: 1786-1798.
- [16] Kassam, S.A. 1978. Quantization Based on the Mean-Absolute-Error Criterion. IEEE Transactions on Communications, 26: 267-270.
- [17] Huang, T.S. 1977. Coding of Two-Tone Images. IEEE Transactions on Communications, 25: 1406-1424.
- [18] Peled, A. and B. Liu. 1985. Digital Image Processing. Malabar, Florida: Robert E. Krieger Publishing Company.