Image Compression Using Block Truncation Coding Techniques

PROJECT REPORT

Submitted By

N. M. R. Krishna
Santosh Antony
G. Vijaykumar

Under the Guidance of

Mr. G. Balasubramanian
B.E., M.S. M.I.S.T.E

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING OF THE BHARATHIAR UNIVERSITY COIMBATORE

1994-95

Department of Computer Science and Engineering

Kumaraguru College of Technology

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

Kumaraguru College of Technology

COIMBATORE - 641 006

CERTIFICATE

This is to certify that the Project Report entitled

IMAGE COMPRESSION USING BLOCK TRUNCATION CODING TECHNIQUES

has been submitted by

Mr. N.M.R. KRISHNA, SANTOSH ANTONY, G.VIJAYAKUMAR.

in partial fulfilment for the award of the degree of

Bachelor of Engineering in Computer Science Engineering

Branch of the Bharathiar University, Coimbatore during the academic year 1994-95

[illegible]
Guide

Head of the Department

Certified that the candidate with University Register No. 9127K0046, 9127K0065, 9127K0064.

was examined in project work Viva-Voce by us on 10/04/'95.

Acknowledgement...

ACKNOWLEDGEMENT

We profoundly thank ***Prof.P.Shanmugam*** *Msc(Engg.),M.S.(Hawaii),SMIEEE,MISTE,* Head of the department of Computer Science and Engineering, for his encouragement and suggestions for the fulfillment of the project.

We express our deep sense of gratitude and heartful thanks to our project guide ***Mr.G.Balasubramanian*** *B.E.,M.S.,MISTE,Senior lecturer,*Department of Computer Science and Engineering, for his valuable guidence in bringing out this project successfully.

We also thank our principal ***Dr. S. Subramanian***, for providing us the facilities, without which we could not have completed our project.

Finally we thank all teaching and non-teaching staff of our department and our friend who rendered immeasurable help at the time of need.

CONTENTS

SYNOPSIS

Image compression has become an important subject of research, since the use of images in several scientific domains has increased. The image compression is a necessary step to reduce bandwidth and storage requirement.

Image compression is the process by means of which an image is represented using as few a number of bits of memory as possible.

In this project work, different algorithms based on Block Truncation Coding (BTC) technique have been developed and implemented. These algorithms give better performance in either the compression or the reconstructed image quality. The algorithms implemented include the Basic Block Truncation Coding (BTC), Modified Block Truncation Coding, Absolute Moment Block Truncation Coding (AMBTC), Absolute Moment Block Truncation Coding using table look up, Adaptive Compression Coding (ACC), Adaptive Compressive Coding using table look up.

Block Truncation Coding involves segmenting the image small blocks and constructing a two-level representation of each block such that the sample mean and variance for each reconstructed block is the same as in the original. BTC uses a two level non parametric quantizer that adapts to the local properties of the image. The levels for each block are chosen such that the first two sample moments are preserved. The technique requires very little computation, no mass storage and can produce compressed images at 1.5 bits /pixel which are comparable to the most advanced complex coding techniques.

such a way that the user can select the

and decompress some sample test images and the performance of the algorithms are evaluated both subjectively and objectively in terms of the compression ratio and compression time as well as the reconstructed image quality.

Turbo C is used to develop the necessary software. It makes the portability of the software to any terminal, easier.

1.0 INTRODUCTION

1.1 Scope of the project

A large amount of memory is needed to represent a typical image. This increases the difficulties in processing the image, storing it in memory or transmission of images to different stations. If image data compression is achieved, the storage and transmission of images would become much more efficient.

1.1.1. Storage

In an image processing computer system, a large amount of memory may be used in just scoring the different image files. Aoting more physical memory is not cost effective if there is need to increase the number of image files in the image data base. So in order to utilize the limited available memory, there is a great need for image data compression algorithms to store images with significantly less storage space.

1.1.2. Transmission

In some image processing applications it would be necessary to send several images over a communication channel. Typical example for this application is the transmission of satellite earth pictures to substations on the earth. The communication channel in such case would be a very long one with applicable delay. In the image representation requires more number of bits, since the channel capacity is limited and increasing it is not cost effective, the number of images that can be transmitted in a given time is limited. If the image data compression [illegible] to limit the image file size, more number of images can be

1.2 What is Image Compression ?

The principle objective of image compression is to represent an image with as few bits as possible, while preserving the level of quality and intelligibility required for the given application.

1.3 Objective of the proposed work

The objective of this work is to develop and implement efficient algorithms for image compression using Block Truncation Coding techniques.

2.0 IMAGE COMPRESSION

2.1 Introduction

There are many algorithms developed for image data compression applications. Each algorithm is based on basic principle of redundancy reduction or decorrelation of the given image. Based on these principles the algorithms can be divided into two basic techniques.

- Spacial Coding technique.
- Transform coding technique.

Each one of these techniques is based on a separate way of reducing the redundancy and correlation in the original image.

2.2 Spatial Coding Technique

In spatial coding techniques, the idea is to generate an array of uncorrelated random variables from the given image using an invertible transformation. These uncorrelated random variables can be represented using a few number of bits. This leads to reduction in memory size for the image. During decompression, the inverse transformation is applied to the random variables to reconstruct the original image.

The structure of transformation is defined by image model which is used to model the image. An image model can be considered as an orderly representation of the represented using fewer bits and this gives rise to image data compression. So, the spatial coding is nothing, but fitting the model to the image, or generating the model of the image.

As an example of describing the spatial coding technique consider a 2 x 2 image model

20	34
26	30

2 x 2

Image block

Fitting the image model to the block, the parameters can be generated.

As m = the average value = 28

The difference values = m (0,0) = -8

m (0,1) = +6

m (1,0) = -2

m (1,1) = +2

If 8 bits are used to represent each of the 4 pixel values of the image block in the model, m needs 8 bits while, (i,j), values needs 4 bits each. The total bits needed for original block = 8 x 4 = 32 bits. Total bits needed for parameters of the image model = 8 + 4 x 4 = 24 bits. Thus image data compression is achieved.

Although in the example a simple model has been taken in actual algorithms different models capable of giving very good compression ratio are used.

2.3 Transform Coding Technique

In the transform coding technique, a linear invertible two dimensional transform is applied to the given image. The transformed coefficients are analyzed and a subset of them are retained, quantized and then stored or transmitted. The coefficients can be represented using less number of bits, since they are decorrelated and also only some of the coefficients are retained. In the decompression process the transform coefficients are reconstructed followed by the application of the inverse transformation. This results in an image which is very close to, it not exactly similar to, the original image.

The best invertible transform is the Karhunen - Loeve transform. This transform produces exactly uncorrelated transform coefficients which leads to a very high compression ratio. But, it is computationally very extensive which makes it almost unusable. Most of the current transform coding algorithms usually use sub optimal transforms such as discrete Cosine transform, Walsh transform, Hadamard transform etc. These transforms are computationally not as intensive as Karhunen - Loeve transform, but their transform coefficients are not exactly uncorrelated which gives rise to lower compression ratio.

2.4 Applications

There are a number of application areas in which image compression is widely used to enable efficient transmission and / or average storage of images. Medical image data compression, video conferencing, remotely piloted vehicles and broadcast television.

♦ ***Medical Image Data Compression***

Despite the high cost of computer tomography systems, they are being used in many hospitals throughout the world. Physicians would like to store the X-rays images and maintain an image database for each patient. Physicians are concerned that data compression might introduce distortion that would remove important information or introduce spurious information, either of which could lead to incorrect diagnosis. Consequently, low error techniques should be considered.

♦ ***Video Conferencing***

Unlike broadcast television, there is rarely a large amount of movement in the pictures used for video conferencing. This allows conditional replenishment coding to be used effectively, which has been demonstrated by hardware realization of complete systems.

♦ ***Remotely Piloted Vehicles***

Remotely piloted vehicles (RPV) are small aircraft, not much larger than model planes, that are used by the military in reconnaissance operations in a military environment, transmission must be protected from jamming by the enemy. This is achieved by the use of spread spectrum techniques in which the information bandwidth is spread and transmitted over a wideband signaling structure, and the signal is retrieved at the receiver by means of correlation techniques. Consequently, the original information bandwidth must be kept as small as possible to allow a number of RPVs to operate simultaneously using the available bandwidth.

♦ ***Broadcast Television***

In broadcast television there is large movement between frames because of scene changes and camera motion, zooming and panning, under such conditions, which are not infrequent, conditional replenishment is rather inefficient. A promising alternative is motion compensation, an adaptive interframe technique that is currently receiving a great deal of attention.

3.0 BLOCK TRUNCATION CODING (BTC)

3.1 Basic BTC Algorithm

This technique uses a one bit non parametric quantizer adaptive over local regions of the image. The quantizer that shows great promise is one that preserves local statistics of the image.

The image is divided into n x n pixel blocks (Generally we take n = 4) and the quantizer will have 2 levels. We can use the classical quantization of Max for quantizer design which minimizes the mean square error. For this we must know, a priori, the probability density function of pixels in each block. Since in general it is not possible to find an adequate density function models for typical imagery we will use non parametric quantizers for this scheme. In this algorithm we use a non parametric quantizer that preserves sample moments denoted as Moment preserving quantizer (MP).

After dividing the picture into n x n blocks, the blocks are coded individually, each into a two level signal. The levels for each block are chosen such that the first two sample moments are preserved.

Let $m = n^2$ and let $X_1, X_2, \ldots X_m$ be the values of the pixels in a block of the original picture.

Let $\overline{X} = (1/m) \sum_{i=1}^{m} X_i$ be the sample mean

$$\overline{\sigma}^2 = (1/m)\sum_{i=1}^{m} (X_i - X)^2 \text{ be the sample variance ------ (2)}$$

As with the design of any one bit quantizer, we find a threshold, Xth and two output levels a and b, such that

if $\overline{X}_i >=$ Xth output = b

if $\overline{X}i <$ Xth output = a for i = 1,2,...m.

For our algorithm Xth will be set to $\overline{X}$, the sample mean of the block. The output levels a and b are found by solving the following equations.

Let q = number of Xi's greater than Xth (=$\overline{X}$)

$$\boxed{\begin{aligned} a &= \overline{X} - \overline{\sigma}\sqrt{q/(m-q)} \\ b &= \overline{X} + \overline{\sigma}\sqrt{(m-q)/q} \end{aligned}}$$

The bit plane is chosen such that all pixel values above $\overline{X}$ are set to 1 and all other values are set to 0. Each block is then described by the values of $\overline{X}$, $\overline{\sigma}$ and an n x n bit consisting of 1's and 0's indicating whether pixels are above or below Xth. Assigning 8 bits each to $\overline{X}$ and $\overline{\sigma}$ and 16 bits for bit plane results in a data rate of 2 bits / pixel. The receiver reconstructs the image block by calculating a and b and placing these values in accordance with the bits in the bit plane. Thus, the sample mean and sample variance in the binary block matches that in the original.

By using this coding technique block boundaries are not visible in the reconstructed picture nor is the quantization noise visible in the regions of the picture where there is little change in luminance. Here the levels a and b are close together. They are widely spaced only in regions where large changes of luminance occur, but large changes of luminance mask noise. Thus BTC encoding makes use of the masking property in human vision. The most noticeable improvement of the reconstructed picture is a little raggedness of sharp edges.

The main advantages of this coding technique are

(1) Calculations involved are relatively simple.

(2) Storage of data is small because only m pixels at a time need be considered, eliminating the need for picture storage and allowing real time coding with a small hardware device.

(3) Suitable to human observer. The largest changes in a block are the ones coded. If no large changes are present, the most significant small variations are coded. The human is also insensitive to small variations in the presence of large variations, so that this technique is neglecting the very thing the human visual system is insensitive to.

(4) The bit plane preserves the original accuracy of an edge or object location with no blurring.

(5) Generates the coded output in one pass through the data.

(6) BTC can be made quite insensitive to channel errors and is easily implemented on a microprocessor.

The two major artifacts occur by using this algorithm are

(1) False contouring due to only two levels in each block.

(2) Misrepresentation of some midrange points due to their assignment to either a high value or low value.

3.2 Modifications To BTC

(1) Mean and variance coding

In many coding schemes it is desired to obtain data rates less than 2 bits/pixel. An with 6 bits and $\bar{\sigma}$ with 4 bits introduces a few perceivable errors. The resulting data rate is 1.63 bits/pixel. Better results could be obtained by quantizing $\bar{X}$ and $\bar{\sigma}$ jointly using 10 bits. $\bar{X}$ is assigned most bits in blocks where $\bar{\sigma}$ is small and fewest bits where $\bar{\sigma}$ is large. If the variance with in block is small, it is possible to omit the bitplane and variance for that particular block, resulting in significant savings for some images.

(2) Bit plane entropy and manipulation

Additional savings could be obtained by efficiently coding the bit plane. Entropy calculations indicate that 0.85 bits/pixel would be sufficient for the bit plane. A modification to reduce the bit plane entropy is to initially assign a "don't' care" value to gray levels close to sample mean and then assign binary values to the "don't cares" such that 1's and 0's tend to cluster together. With this many ragged edges in the reconstructed image will appear somewhat smoother.

(3) Dithering

The receiver can remove some contouring effects due to two level construction by dithering each reconstructed value by a fraction of standard deviation of each block.

(4) Quality improvement through RMS Error control

In a few cases a binary block cannot accurately represent the original. If the transmitter ascertains that a reconstructed will be separated from the original by a mean square error greater than some threshold, additional information can be sent to improve the quality of that block. For example a second variance and bitplane can be coded which essentially creates a four level images block instead of a two level one.

3.3 Other Nonparametric Quantizer Schemes

We can use other nonparametric quantizers instead of moment preserving (MP) quantizer used in the above described algorithm. To use the Mean square error (MSE) fidelity criterion, Xi's). Let Y1, Y2 '... Ym be the sorted X1's. Let q be the number of Xi's greater than Xth.

Then a and b values are

$$a = 1/(m-q) \sum_{i=1}^{m-q-1} Y_i$$

$$b = 1/q \sum_{i=m-q}^{m} Y_i$$

One obvious way to solve this problem is to try every possible threshold (there are m-1 thresholds) and we have to pick the one with smallest JMSE. Assuming a and b have 8 bit resolution, this gives a date rate of 2 bits/pixel.

To use the Mean absolute Error (MAE) fidelity criterion first histogram of Xi's is constructed as in the case of MSE fidelity criterion. the values of a and b are

a = median of $(Y_1, Y_2, \ldots\ldots\ldots\ldots Y_{m-q-1})$

b = median of $(Y_{m-q-1} \ldots\ldots\ldots\ldots Y_m)$

Here also the nonparametric quantizer is obtained by an exhaustive search. The MSE quantizer has the smallest mean square error measure and the MAE quantizer has the smallest mean absolute error measure.

The performance of BTC is quite good when compared to these standard fidelity criteria. The advantage of using a nonparametric moment preserving (MP) quantizer is that the quantizer formulation is available in olosed form, this greatly simplifies the computational load.

By setting the threshold of the nonparametric quantizer at S, removes a possible degree of freedom for optimizing the encoding. Allowing the threshold to be a variable permits the encoding to preserve not only the first sample moments but also the third sample moment.

To analyze this we shall make use of sorted pixel values Yi. The third moment can be expressed as

$$\overline{X^3} = (1/m)\sum_{i=1}^{m} X_i^3$$

$$\overline{X^3} = (1/m)\sum_{i=1}^{m} Y_i^3$$

For finding a, b, and q to preserve X , X^2, and X^3 we have to solve the following equations.

$$m\overline{X} = (m - q)a + qb$$

$$m\overline{X^2} = (m - q)a^2 + qb^2$$

$$m\overline{X^3} = (m - q)a^3 + qb^3$$

The solutions are :

$$\boxed{\begin{aligned} a &= \bar{X} - \bar{\sigma}\sqrt{q/(m-q)} \\ b &= \bar{X} + \bar{\sigma}\sqrt{(m-q)/q} \end{aligned}}$$

In general the use of q obtained will not be an integer, so it must be rounded to nearest integer.

An interesting interpretation is that the threshold is nominally set to the sample median and biased higher or lower depending upon the value of the third sample moment which is a measure of skewness of the Yi's. This threshold selection requires no extra computation at the receiver but the transmitter does have extra processing. This method of threshold selection is far easier to implement than other nonparametric quantizers such as MSE or MAE quantizers because they require an exhaustive search.

3.4 Performance Evaluation Of BTC

Although BTC does not perform as well as transform coding in the photo interpreters evaluation, it proves superior to the other techniques in the presence of many channel errors. BTC requires a significantly smaller computational load and much less memory than transform or hybrid coding. For instance, the chen and smith method requires the two-dimensional cosine transform over every 16x16 image block and also requires multiple passes through the transform data to collect various statistics about the transform coefficients. BTC requires no sophisticated error protection as do other coding methods evaluated. One of the advantages of BTC is that luminance edges are emphasized.

The drawbacks of BTC are it produces artifacts that are very different than the transform coding. They are usually seen in the regions around edges and in low contrast areas containing a sloping gray level. BTC produces sharp edges, these edges have tendency to be ragged. Transform coding usually produces edges that are blurred and smooth. The second problem in low contrast regions is due to inherent quantization noise in the one bit quantizer. Here the sloping gray levels can turn into false contours. By doing pre and post processing of the image the effects of both these artifacts can be reduced while simultaneously reducing the Mean square error and Mean absolute error.

3.5 Hybrid Formulation Of BTC

BTC uses only first order information and does not exhibit the two dimensional structure of the image within each block as do most other forms of image coding. For example in the two dimensional transform coding the transform coefficients contain information about variations in the picture in both directions. Also BTC generally has a poor response near a spatial frequency of 1/2 cycle per block.

One method to overcome this disadvantage is a hybrid formulation. First a highly compressed cosine transform coded image is subtracted from the original image. Then BTC is used on this difference picture and the recombination is performed at receiver. Those method increases the computational load but it show enough significant improvements.

3.6 Absolute Moment Block Truncation Coding (AMBTC)

The principle idea used in Block truncation coding is to achieve compression while preserving standard moments. AMBTC used a nonparametric quantizer which preserves sample absolute moments.

The image is divided into n x n blocks (generally n = 4) and the quantizer will have two levels. Each block is quantized in such a way that each resulting block has the same sample mean and the same sample first absolute central moment of each original block

.

Let $m = n^2$ an let $x_1, X_2 \ldots X_m$ be the values of pixels in a block of the original picture.

Let $\bar{\eta} = (1/m) \sum_{i=1}^{m} X_i$ be the sample mean .... (1)

$\bar{\alpha} = (1/m) \sum_{i=1}^{m} (X_i - \eta)^2$ be the sample first absolute central moment ------ (2)

The mean value contains information about central tendency, this is same as central tendency information used in basic block truncation coding. The sample first absolute central moment contains information about dispersion about the mean. The corresponding value used in BTC is the sample standard deviation. Therefore two of the most important local characteristics of the spatial block gray levels are preserved : central tendency and deviation from the center.

The sample first absolute central moment can be calculated in a simple way as follows.

m

$$\bar{\gamma} = m\,\alpha/2 = \sum_{\text{For } X_i >= \eta} X_i - \eta q$$

$$\boxed{\begin{array}{c} a = \bar{\eta} - \bar{\gamma}/(m-q) \\ b = \bar{\eta} + \bar{\gamma}/q \end{array}}$$

In order to preserve the moments given in eq. (1) & (2) it is necessary to assign two values a and b at the output of the two level quantizer such that
The behavior this quantizer can be analyzed. For the case in which all pixels have equal gray level, the deviation information is equal to zero and both a and b are set to the mean value which is also the correct value of the pixel. On the other hand if the pixels are dispersed about the mean, the value assigned to b is mean plus a bias that depends directly on the dispersion and inversely on the number of pixels above the mean and the value assigned to a is mean minus a bias that depends directly on the dispersion and inversely on the number of pixels below the mean.

The bit plane is chosen such that all pixels above are set to 1 and all other values are set to o. Each block is than described by $\bar{\eta}$, $\bar{\alpha}$, and an n x n bit plane consisting of 1's and 0's indication whether the pixels are above or below Xth. Assigning 8 bits each to $\bar{\eta}$ and $\bar{\alpha}$ and 16 bits for n x n bit plane results in a data rate of 2 bits/pixel. The receiver reconstructs the image block by calculating a and b and placing those values in accordance with the bits in the bit plane. Thus the sample mean and sample first absolute central moment in the reconstructed block matches that in the original block.

Coding $\bar{\eta}$ with 6 bits and $\bar{\alpha}$ with 4 bits introduces only a few perceivable errors and the resulting bit rate is 1.63 bits/pixel. By using this technique the Mean Square error is smaller than the Mean Square error resulting from BTC, the major artifacts occurring by AMBTC are same as BTC which are edge raggedness and misrepresentation of some mid range values due to their assignment to either high or low value.

3.7 Errors In AMBTC

In the order to compare the nonparametric moment preserving quantizer used in AMBTC to the two level MMSE quantizer.

$$a = \bar{\eta} - \bar{\gamma}/(m - q)$$

$$= \bar{\eta} - 1/(m - q) \sum_{\text{For } X_i < \bar{\eta}} X_i - \bar{\eta}q$$

$$b = \bar{\eta} + \bar{\gamma} / q$$

$$= \bar{\eta} + 1 / q \sum_{\text{For } X_i >= \bar{\eta}} X_i - \bar{\eta}q$$

a and b are estimated conditional means given that X_i is less or greater than $\bar{\eta}$, respectively. On the other hand the nonparametric MMSE two levl quantizer would give the same form of equations but with a general threshold t and an iterative procedure is suggested for MMSE quantizer. Therefore the output equations of AMBTC are those of a suboptimum implementation of a nonparametric MMSE quantizer in which the threshold is fixed to the mean. This result could explain the better mean square error performance of AMBTC when compared to BTC.

To analyze the error introduced by this non parametric quantizer let us define a deterministic mean square error as

$$\text{MSE} = (1/m) \sum^{m}$$

If the histogram of the block is symmetric about the sample mean and m is even, we have q = m-q = m/2.

$$g(\bar{\eta}) = (m/2)\,(\bar{\eta} - \bar{\alpha})$$

$$m\bar{\eta} - g(\bar{\eta}) = m/2\,(\bar{\eta} + \bar{\alpha})$$

Replacing these values into expression for mean square error we get

$$\boxed{MSE = \bar{\sigma}^2 - \bar{\alpha}^2}$$

It is important to note here, that in those cases where the standard deviation equals first absolute central moment the error becomes zero. This situation occurs for example when all the pixels have the same gray level or when half of them take a value $\bar{\eta}$ + A and the other half take a value of $\bar{\eta}$ - A for any fixed A.

Other values of threshold could be used. Using t = (Xmin + Xmax)/2 as threshold yields lower mean square error in standard BTC, where xmin are the values of minimum and maximum values of pixels in a block.

3.8 Advantages Of AMBTC

In the case that the quantizer used to transmit an image from aransmitter to a receive, it isnecessary to compute two quantities at the transmitter using either BTC or AMBTC. These two quantities are the sample mean and information about the deviation about the centre, which is the sample standard deviation for BTC and the sample first absolute central moment for AMBTC. Since computation of central information is same for both the methods, let us compare mecessary computations of them squared, while in the case of AMBTC it is only necessary to compute the sum of the values. Since the multiplication time in most digital processore is several times greater then the addition time, by using AMBTC the total computtation time at the transmitter is significantly reduced.

A similar situation occurs at receiver. For standard BTC it is necessary to evaluate two quotients and two square roots. While using AMBTC it is to evaluate only two quotients. Consequently, the processing time at the receiver is also substantially reduced. The savings in time achieved by AMBTC might suggest the name of fast BTC for this method. The rest of the characteristics of AMBTC remain the same as those of BTC since the pholosophy of both methods is substantially the same. An additional advantage of AMBTC is the Minimum Mean Square Error achieved under symmetry conditions.

3.9 Modified Block Truncation Coding

This algorithm is similar to basic BTC algorithm exceptthat this algorithm uses a one bit nonparametric quantizer which preserves the first order statistical information namely the lower mean $\bar{X}_l$ and the higher mean $\bar{X}_h$ of n x n pixel blocks of the input image data. The mean $\bar{X}$ of the pixels in the block is taken as the one bit quantizer threshold.

After dividing the picture into 4 x 4 blocks, the blocks are coded individually each into a two level signal. The levels for each block are chosen such that the first order statistical information $\bar{X}_l$ and $\bar{X}_h$ are preserved.

Let $m = n^2$ and let X_1, X_2 .... X_m be the values of pixels in a block of the original picture.

$$\bar{X} = (1/m) \sum_{i=1}^{m} X_i$$

For our algorithm X is taken as the quanizer threshold Xth and the quantizer output levels are

$$\bar{X}_l = 1/(m - q) \sum_{i=1}^{m} X_i \qquad \text{if and only if } X_i < \bar{X}$$

$$\bar{X}_h = 1/q \sum_{i=1}^{m} X_i \qquad \text{if and only if } X_i >= \bar{X}$$

m is the number of pixels in a block (n x n).

Xi is the value of ith pixels in the block.

q is the number of pixels having value higher then X.

It is also shown that

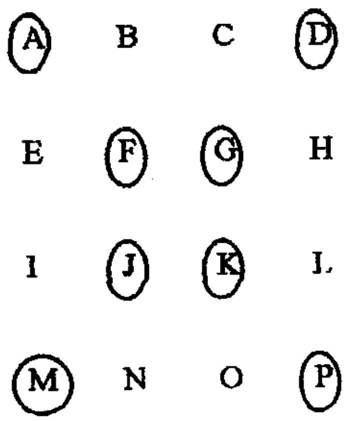
$$\bar{X}_h = \bar{X}_l + (m / q) (\bar{X} - \bar{X}_l)$$

The bit plane is chosen such that all pixels above $\bar{X}$ are set to 1 and all other values are set to 0. Each block is then described by the values of $\bar{X}$, $\bar{X}_l$ and an n x n bit plane consisting of 1's and 0's indicating whether the pixels are above or below the threshold Xth i.e. $\bar{X}$. The receiver reconstructs the image block by calculating $\bar{X}_h$ and placing $\bar{X}_h$ and $\bar{X}_l$ in accordance with the byte in the bit plane. Thus the first order statistical information that is lower mean and higher mean in the binary bolck matches that in the original.

By assigning 8 bits each to $\bar{X}$ and $\bar{X}_l$ and 16 bits for bit plane results in a data rate of 2 bits/pixels. The image reconstructed by this technique given lower MSE then that of resulting from basic BTC algorithm. The rest of the characterstrics of modified BTC remain the same as those of BTC since the philosophy of both these methods is substantially the same.

3.10 AMBTC Using Table Lookup

This method is basically same as AMBTC but only 8 bits are transmitted for the binary bit plane instead of 16 bits as in the case of AMBTC. This method is based on the occurrence probability of certain bit plane parttems. A set of 8 pixels in the 4 x 4 block will specify the values of ther bits in the plane. We define the 8 circled pixels to be independent and the 8 uncircled pixels to be dependent.



The value of the circled pixels are the ones transmitted and the value of the other pixels are deifned accouding to the following logical table.

B = 1 iff [(A and D) = 1] or [(F and J) = 1]

E = 1 iff [(A and M) = 1] or [(F and G) = 1]

H = 1 iff [(D and P) = 1] or [(F and G) = 1]

I = 1 iff [(A and M) = 1] or [(J and K) = 1]

L = 1 iff [(D and P) = 1] or [(J and K) = 1]

N = 1 iff [(M and P) = 1] or [(F and J) = 1]

O = 1 iff [(M and P) = 1] or [(G and K) = 1]

The logic behind the table will prserve the edges of the block. Each block is then described by the values of mean $\overline{X}$, first absolute central moment and 8 bit plane consisting of 1's and o's assigning 8 bits each to x, and bit plane results in a data rate of 1.5 bits/pixel. If we use 6 bits for mean and 4 bits for first absolute central moment we will get a data rate of 1.125 bits/pixel. The receiver constructs the image block by first calculating the values of other 8 bits in accordance with the table lookup and then by calculating a and b and placing those values in accordance with the bit plane. This method preserves all horizontal, vertical and some other diagonal edges. The loss in resolution caused by the dependent bit plane points may be unacceptable.

4.0 ADAPTIVE COMPRESSION CODING

4.1 Introduction

The aim of this algorithm is to obtain high quality compressed images with high compression ratio an with edge preservation characteristics. Edge preservation is more important is many engineering application such as in vision, Robotics and recognition. Example of which rely on edge information include identification of types of specific objects such as apples, oranges, houses, military and civilian aircraft etc. In robotics edge detection is used for the recognition of tools and their positions. In biomedical engineering application such as recognition of the different kinds of blood cells and sizes of tumors also rely on edge information.

Block truncation coding (BTC) is known to give high quality compressed images, however, BTC results in ragged edges and introduce noise at edges. In BTC the correlation remains in the pixels of the binary block generated by the quantizer. The compressed data rate is reduced to 1.375 bits/pixel if the roots of the median filter are used to represent the binary block. A compressed data rate of 1.1875 bits/pixel is achieved if the binary block is coded with trellis coding. These modifications increase the compression ratio but at the expense of increased image degradation and algorithm complexity. These variation of BTC are fundamentally similar relying on a uniform operator which does not adapt to local statistics of the image. A positive aspect of BTC is that it exploits the masking property of the human vision system, where by errors in a textured region of an image are less visible than are errors in a uniform region. As a result it codes heavily detailed or texture regions compactly without introducing obvious artifacts. However BTC does cause a noisy ragged appearance in the regions which contain edges. Also data representation used by BTC is inefficient in regions where the pixel intensity are relatively constant.

An *algorithm which adapts to the local statistics of the image* should be able to preserve edges while coding smooth regions more compactly than BTC. Adaptive compression coding (ACC) is formed by combining BTC with quadtree coding for regions which are smooth and regions which contain an edge. Compression ratio is improved by first representing the regions which contain relatively constant intensity by their average intensity and second by representing the 16 binary bits of BTC by eight bits using table lookup. The later is used only for the BTC blocks where the intensities of the two output levels of BTC differ by greater than 20. Image quality is improved by representing regions which contain an edge by a quadtree containing original pixels. ACC achieves high quality compressed pictures with data rates from 1.1 to 1.2 bits/pixel. Good quality pictures may be obtained at 0.5 - 0.8 bits/pixel ,if a post smoothing filter is obtained. This is to smooth the blocky appearance of the picture at those compression levels. More important is the edge preservation nature of ACC.Edges are preserved at all compression ratios. This makes ACC very useful in applications where edge preservation is important such as robot vision and other recognition applications.The adaptive algorithm is more complex than BTC, resulting in an increased compression time. However the data representation used by ACC result in a decreased decompression time.

4.2 The Adaptive Compression Coding Algorithm

In this algorithm the image is divided in to 4 x 4 blocks. After dividing, each block is categorized into one of the following three categories.

The three categories are

2. Edge block.... containing a definitive edge.
3. Textured block .... having an undefinable picture orientation.

After categorizing each block, if the block is a smooth block, it is represented by its average. If the block is an edge block,it is represented by a quadtree. If the block is a textured block, it is coded using AMBTC. For the textured block, if the two output levels of AMBTC differ by 20 or more, a different version is used. In this only eight bits are used to code the binary bit plane instead of 16 bits used as in the case of AMBTC.

The test to determine whether or not a block is relatively uniform (smooth) and thus could be represented by its average, uses the range which is the difference between the maximum and minimum intensities in the block. This is compared to a user defined preset threshold usually in the range 13 to 50. This threshold shall be called as smoothness threshold. Another choice to detect the smoothness, besides the smoothness range could have been the variance of intensities in the block. However the range is better detecting the small portions of lines or edges which might be present in the block. The range always detects such features while the variance might miss them. A value for smoothness threshold is found to be equal to 18. This value produces best results in terms of image quality, RMSE and image compression. At high values of smoothness threshold more blocks are represented by their averages and better compression ratios are obtained. However a more blockyness appears in the picture and a post smoothing filter is needed. The edge preservation of the filter is not affected by the variations in the smoothness threshold.

If the 4 x 4 block is represented by its average then it is assumed to contain either an edge or to have a texture. A test to determine that uses a different threshold

edge detection threshold. If the range(difference between the maximum and minimum intensities in the block)is less than the threshold, the block is assumed to be of texture nature. In this case AMBTC with table lookup is used for coding Otherwise the 4 x 4 block is assumed to have an edge. If an edge is detected then quadtree encoding is used. If an edge is detected then that block is divvied into four 2 x 2 sub blocks. Afterwards the range for each 2 x 2 sub block is compared to a threshold which is equal to half of the threshold value used for detecting edges in the 4 x 4 original block namely a value of 60 is now used. The reason for using half of the original value of threshold is that a sharp edge is infact represented by a gradual step. This is due to low pass filtering effect of the lens of the camera which results
in slight blurring of the image. The result here is that a sharp edge will look more like a ramp than a step function. In the unlikely event that the lens is used is perfect and the edges are infact perfect step function, then the above procedure does not alter the results.

For the 4 x 4 blocks which have a sharp edge the original intensities of those blocks which contain the edge will be saved, the other 2 x 2 blocks will be represented by their averages.

4.3 AMBTC And Table Lookup

Each textured block is represented by using AMBTC. AMBTC encodes every 4 x 4 block of the image by its average, first absolute central moment and 16 binary bits. Each bit indicates whether or not the original intensity at a pixel is above the average intensity value. After decoding, every pixel takes one of the following two values depending on whether the binary bit is 0 or 1 . The two levels are

$$B = \overline{X} + (8\,\overline{\alpha} / q)$$

where $\overline{X}$ and $\overline{\alpha}$ are the average and the first absolute central moment of the original intensities and Q is the number of pixels with intensities equal or greater than the average intensity value.

AMBTC attains a compression ratio of 8:1.625 if a two dimensional quantizer is used for mean and first absolute central moment. Further compression may be achieved if the binary bits are represented by fewer bits. Methods which make this possible include lookup tables and roots of the median filter. Another simple method is to use lookup tables only for areas with very low contrast. This method uses AMBTC only if the difference between the two output levels is less than or equal to 20, otherwise a lookup table is used. In the table lookup method only 8 are coded instead of 16 bits of the binary bit plane. This is based on occurrence probability of certain bit plane patterns. The 8 bits that are coded will specify the value of other bits in the plane.

4.4 Data Identification

A two bit code word is needed for each 4 x 4 block since for each block four possibilities exist. These are the possibilities representing the block by its average, AMBTC with table lookup and *quadtree*. This code word is followed by the relevant values i.e. the average of intensities, the values for AMBTC with table lookup. If an edge exists within a 4 x 4 quadrant we need to specify which of the 2 x 2 blocks that edge passes through. For such 2 x 2 blocks the most significant bit of the word used to store the information for that block is used to identify whether the block has an edge or the mean of the intensities is stored. If the mean is stored 7 bits are used to represent the mean instead of usual 8 bits

During decompression this value is multiplied by 2. If the 2 x 2 block contains edge then the 4 original intensities have to be stored. In this case one of these intensities is represented by 7 bits.

4.5. Comparison With AMBTC

The adaptive compression coding technique produces images much better than those of AMBTC. The edge of the image from AMBTC are noisy and have ragged appearance. The image from ACC algorithm do not appear as degraded as the RMSE would suggest because most of the edge are perfectly preserved. This indicates that the degree of edge preservation is an important characteristic of any image coding algorithm. At low compression rate degradation becomes significant due to the fact that many textured regions are smoothened as number of blocks represented by their average increases.

Because BTC algorithms are non-adaptive, their compression and decompression times are relatively constant for images. Compression using adaptive algorithm is approximately two Times slower than AMBTC and varies slightly from image to image. Decompression is approximately same as that of AMBTC for high quality images. As the threshold value increases and the compressed data rates becomes lower, the decompression times becomes significantly faster than that of AMBTC.

One of the salient feature in the algorithm is that the smoothness threshold is user defined. The user can determine the threshold he needs and give it as an input to the algorithm. The significance of the threshold lies in the fact, that for each of blocks in the image, the range (difference between the maximum and considered as a smooth block. We have seen that smooth block is encoded only with its mean value. So during the decompression process, the block would contain pixels with single gray level and any small variations, if any would be lost. By giving the threshold the user himself can determine the level of small variations that need not be reproduced in the reconstructed image.

4.6 The Encoding Steps For Adaptive Compression Coding

1. Get the smoothness threshold and image file name as inputs.
2. Divide the image data file into 4 x 4 pixel blocks. This process is accomplished by storing the image data in a temporary memory buffer area and grouping the pixels making up the individual blocks.
3. For each of the block, get the corresponding pixel values from the memory buffer and execute the following steps.
4. Determine the range value of the gray level values of the pixels in the block.
5. Check the range with the given threshold. If the range is less than the threshold to step 6 otherwise go to step 7.
6. The block is considered as a smooth block. Determine the mean value of the gray level value of the pixels in the block. The code for this block is considered to be code for smooth block and the mean value. Go to step 12 .
7. Check for the range of the block with 120. If the range is less than 120 go to step 8 otherwise go to step 9.
8. The block is considered to be a textured block. Determine the mean value and first absolute central moment for the gray level values of the pixels in the block. Compare each pixel gray level value with the mean of the block and is coded either 0 if it is less than mean or 1 if it is greater than or equal to the mean value. The code for this block is considered to be code for textured block, mean,

first absolute central moment and the bit plane consisting of 1's and 0's. Go to step 12.

9. The block is considered to be an edge block. Divide this block into 4 sub blocks and calculate range for each sub block.

10. Compare the range of each sub block with value 60. If it is greater than 60 that sub block is considered to contain edge and store the 4 gray level values in that sub block. Otherwise calculate the average of the gray level values in the sub block. 11. The code for this block is code for the edge block and code for each sub block whether it contain edge or not and the corresponding values.

12. Retain and store the code for the block If all the pixels are encoded. Terminate the algorithm otherwise continue from step 4.

<u>The Decoding Steps</u>

1. Get the compressed image data file.
2. Each code set in the file corresponds to one 4 x 4 block. The first two bits of the first code it each code set is the code for block type.
3. Determine the block type

   If smooth block go to step 4.

   else if textured block go to step 5.

   else go to step 6.
4. For the smooth block extract rightmost 6 bits of the code which would be the mean value of the block. Retain this value as the gray level for all pixels in the block. Go to step 8.
5. For textured block the rightmost 6 bits is the mean value and the next code would be the first absolute central moment. As per the absolute moment block truncation coding determine code value for a and b. From the individual codes for pixels determine their gray levels. If the code is 0 its gray level is 'a' otherwise

6. For the edge block get the next code and extract the leftmost bit and determine the sub block type. If it is sub block containing edge the rightmost seven bits and the next three codes are the gray level values of the pixels in that 2 x 2 sub block. Otherwise the rightmost seven bits is the mean value of the sub block. Retain this value as the gray level value for all the pixels in the sub block.

7. If all the sub block in the block are decompressed go to step 8 otherwise goto step 6.

8. After determining the pixels gray levels for all the pixels in the block, store in the appropriate place in memory buffer. If all blocks are decompressed go to step 9 otherwise go to step 3.

9. Store the image data from the buffer to the image data file.

5.0 RESULTS AND CONCLUSION

Various algorithms for image compression based on block truncation coding techniques were implemented and their performance were evaluated.

5.1. Criteria For Evaluation

The criteria for evaluation should cover both the quality of the reconstructed image and the data compression performance of the algorithms. To evaluate these two features two types of criteria are needed.

5.1.1. Data Compression Performance Criteria

For the purpose of evaluation of the data compression performance two criteria can be selected.

1. The number of bits of storage needed for pixel in the image.

This measure can be easily calculated as the ratio of memory size of the compressed image to the total number of pixels in the original image. An algorithm performs better than another algorithm, if the first algorithm gives less number of bits/pixel the second one, provided all other parameters are same.

2. The compression ratio

The compression ratio is calculated as the ratio of total memory size of the clearly brings out the evaluation of the actual purpose of the algorithm, that is, the reduction in memory size required to store the image. This compression ratio would be the main criteria to evaluate the compression performance of the different the algorithms.

5.1.2 Image Quality Fidelity Criteria

For the evaluation of the quality of the reconstructed image fidelity criteria can be applied.

Fidelity criteria can be divided in to two types.

1. Objective Fidelity Criteria

The objective fidelity criteria would enable us to measure the quality of the reconstructed images exactly as a mathematical quantity. The objective fidelity criteria to be used for the system is called the root mean square error (RMS) between the original and reconstructed image.

Assume that the original image consists of M x N array of pixels.

$f(x, y)$

$x=0,1,2,\ldots\ldots\ldots M-1$

$y=0,1,2,\ldots\ldots\ldots N-1$

Let the reconstructed image, which is also an M x N array of pixels, be represented as ,

$g(x, y)$

$x=0,1,2,\ldots\ldots M-1$

$y=0,1,2,\ldots\ldots N-1.$

For any pixel (x, y) the error between the original image and the reconstructed image is given by, The squared error average over the image area is

$$e^2 = 1/(M \times N) \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} e^2(x,y)$$

The RMS error is defined as

$$e_{rms} = \sqrt{e^2}$$

2. Subjective Fidelity Criteria

Usually the reconstructed images are used for the purpose of viewing by the people. So it is more appropriate to use a subjective fidelite criteria correspoding to how good the images look to human observers. The human system has peculiar characteristics so that two pictures having the same amount of RMS error may appear to have drastically different visual quality.

The human visual system has logarithmic sensitivity to light intensity so that errors in dark areas of an image are much more noticeable tham errors in light areas. Also, human eye is sensitive to abrupt changes in gray level, so that error on or near the edges are more bothersome tham error background texture.

The subjective quality of an image can be evaluated by showing the image to a number of observers and averaging thier evaluations. The absolute scale as used by panel 6 of the television allocations study organisation can be used.

This scale is given below:

1. Excellant: An image of extremely high quality. Not possible to improve further.
2. Fine: An image of high quality. Interference is not objectionable.
3. Passable: An image of acceptable quality.
4. Marginal: An image of poor quality . Interference is some what objectionable.
5. Inferior: A very poor image.Objectionable interference.
6. Unusable: An image so bad that people could not watch it.

5.2 Results

The results of compression and decompressing the test images are analyzed and each of the algorithms is evaluated by their compression performances and aslo the reconstruction image quality.The evaluation is done by means of the criteria as explained in the previous section.

The results of the evaluation criteria can be given in tabular structure. The results are adequate enough to appreciate the performance of each and every algorithm.Compression a rate of the process in the case of application of each algorithm. Clearly the compression performance of the algorithms can be seen.

The generated image quality is also tested by the objectives fidelity criteria using the absolute scale as defined in the previous section.

5.3.Conclusion

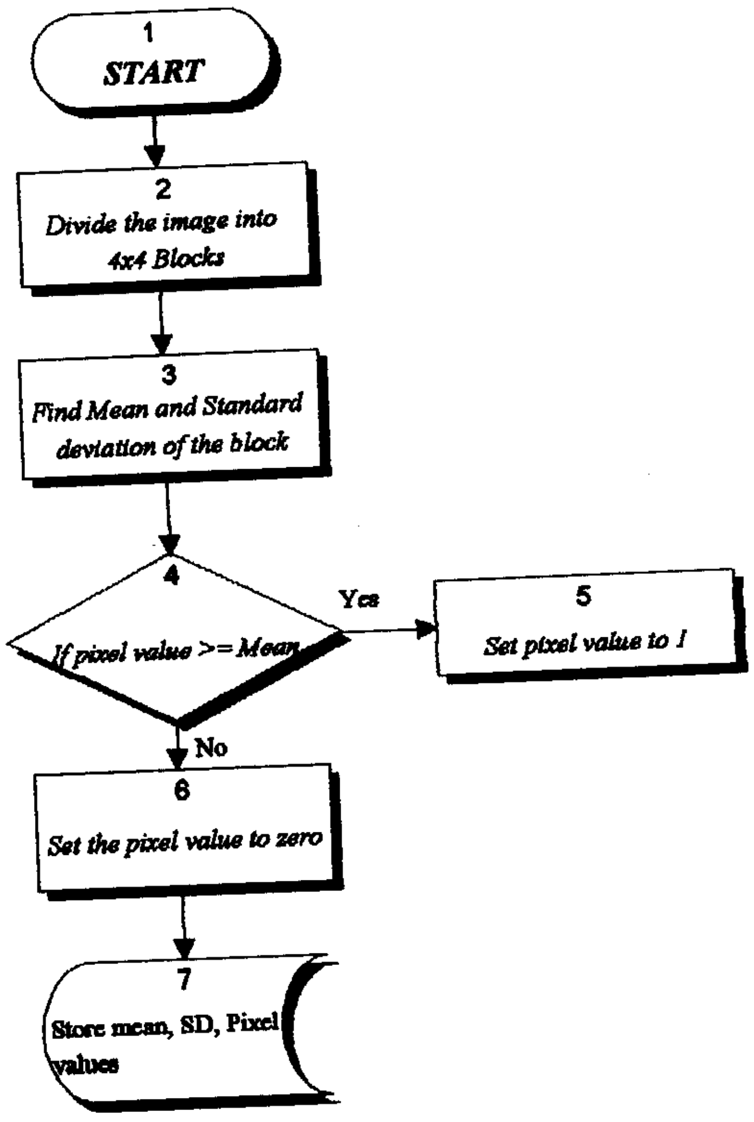
The algorthim developed as part of the project are tested using suitable test images. The results of the tests are computed in terms of compression performance as well as regenerated image quality criteria. The performance of each of the algorithms is shown in the table . The best algorithms are found. From these results, the choise of algorithms for the applications at hand can be easily determined.

5.4.Suggestions For Future Work

- The bit rate can still be reduced by quantizing the mean and variance using two dimensional quantization.
- By using variable threshold, we can also preserve the third moment.

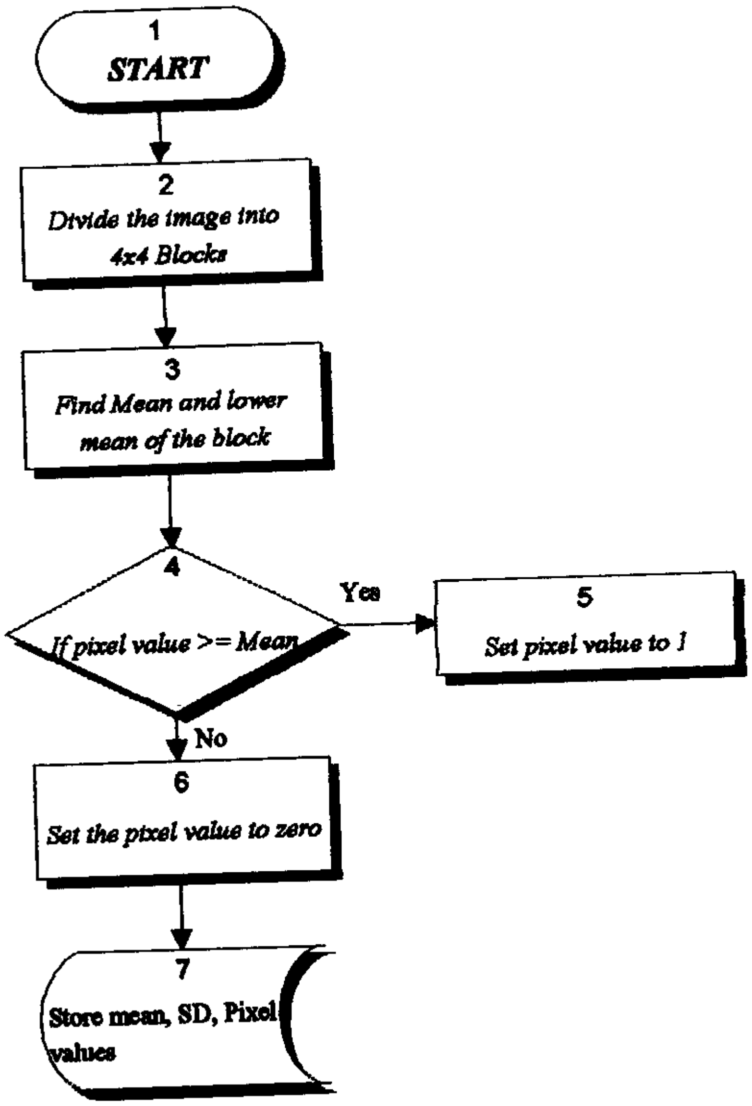
* The overhead statistical information and the truncated block can be compressed using separate vector quantizers(VQ).
* AMBTC can be applied to compress color images.

BASIC BLOCK TRUNCATION CODING

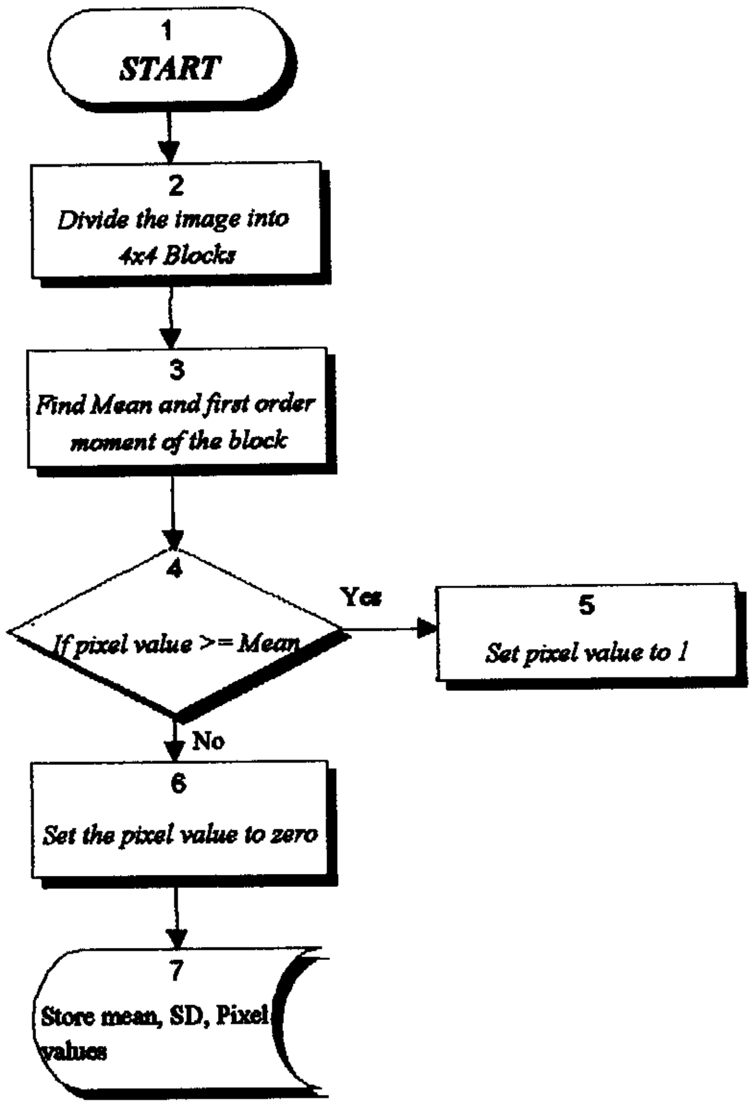


1
START
2
Divide the image into 4x4 Blocks
3
Find Mean and Standard deviation of the block
4
If pixel value >= Mean
Yes
5
Set pixel value to 1
No
6
Set the pixel value to zero
7
Store mean, SD, Pixel values

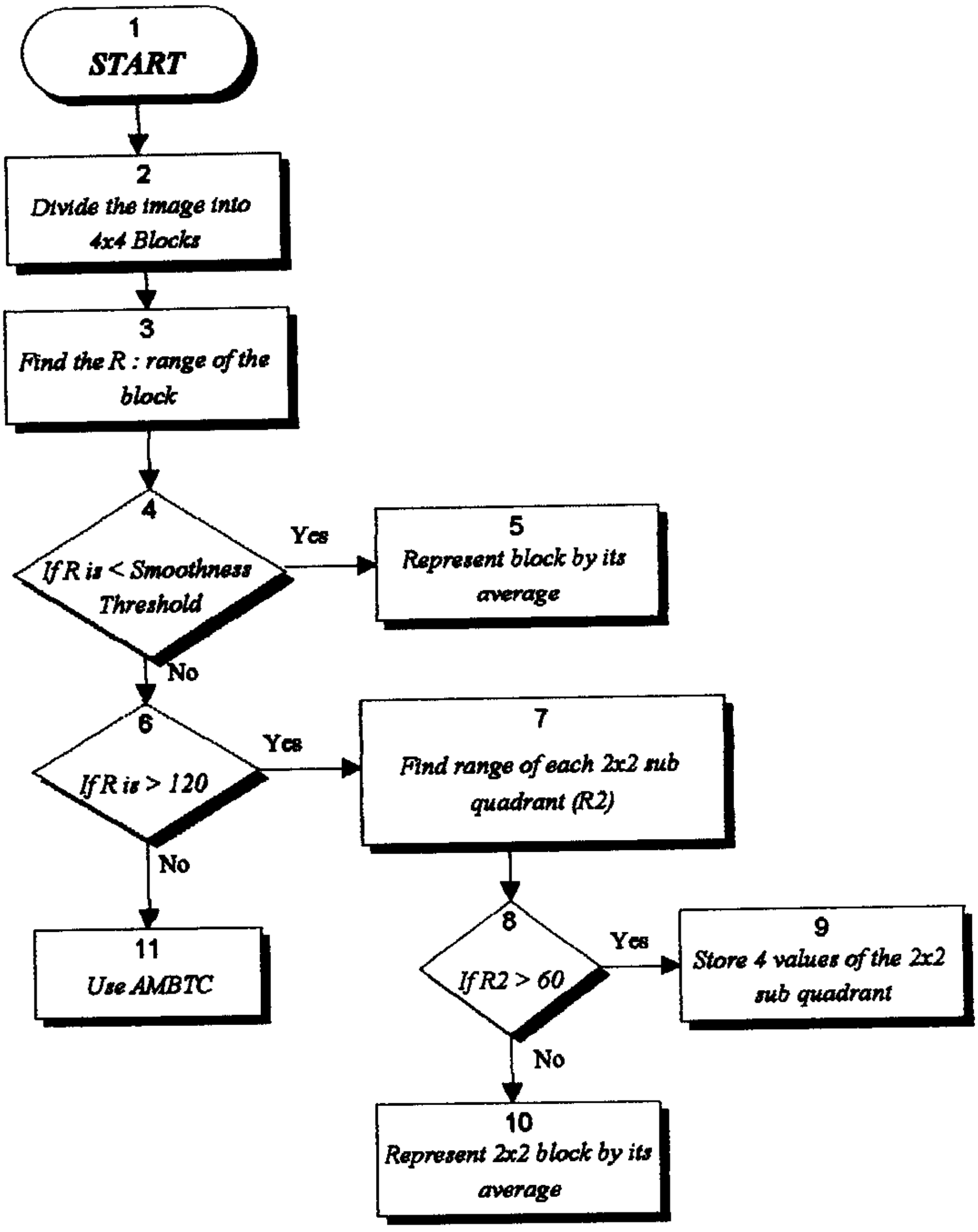
MODIFIED BLOCK TRUNCATION CODING


1
START
2
Divide the image into 4x4 Blocks
3
Find Mean and lower mean of the block
4
If pixel value >= Mean
Yes
5
Set pixel value to 1
No
6
Set the pixel value to zero
7
Store mean, SD, Pixel values

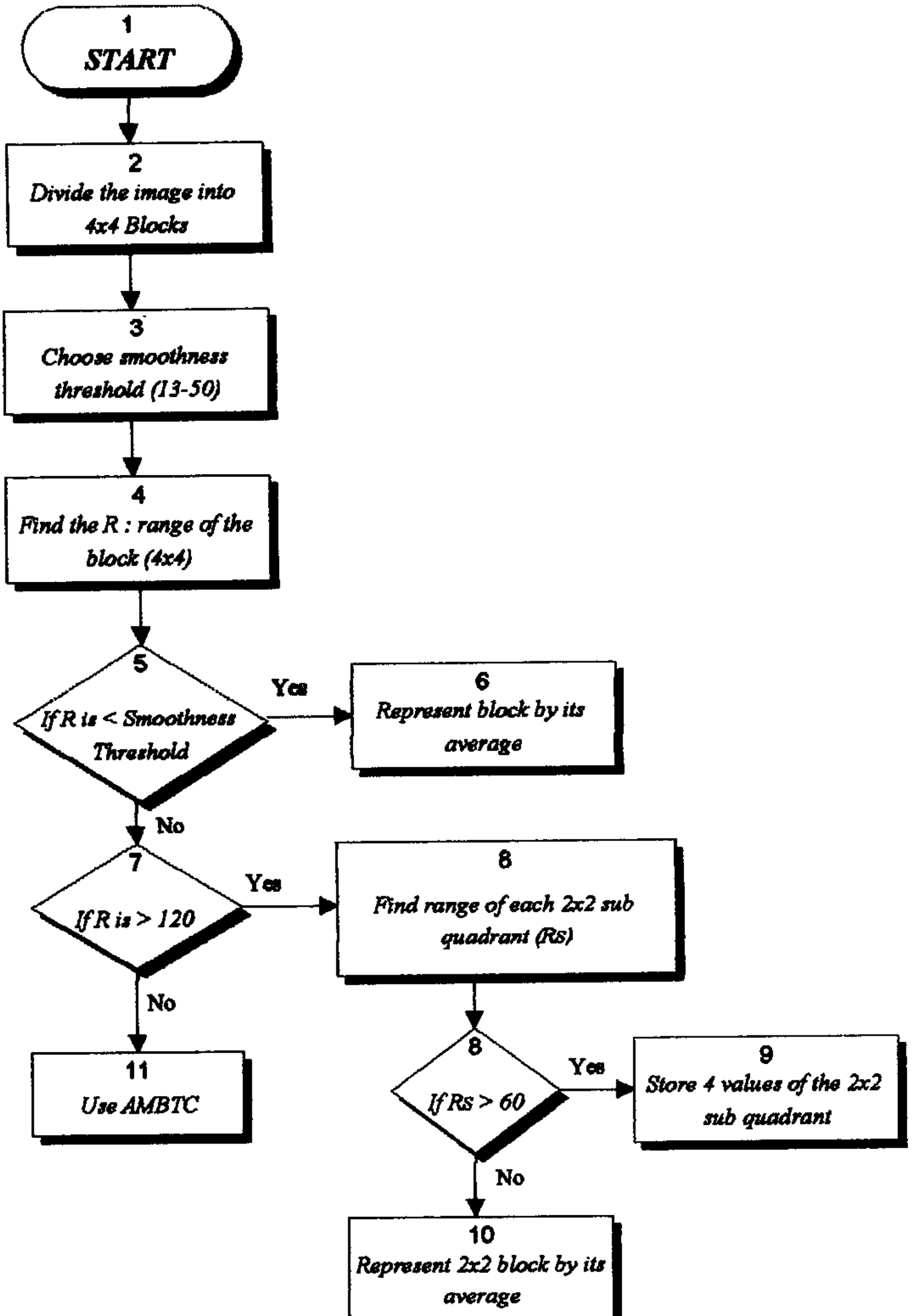
ABSOLUTE MOMENT BLOCK TRUNCATION CODING


1
START
2
Divide the image into 4x4 Blocks
3
Find Mean and first order moment of the block
4
If pixel value >= Mean
Yes
5
Set pixel value to 1
No
6
Set the pixel value to zero
7
Store mean, SD, Pixel values

ADAPTIVE COMPRESSION CODING


1
START
2
Divide the image into 4x4 Blocks
3
Find the R : range of the block
4
If R is < Smoothness Threshold
Yes
5
Represent block by its average
No
6
If R is > 120
Yes
7
Find range of each 2x2 sub quadrant (R2)
No
11
Use AMBTC
8
If R2 > 60
Yes
9
Store 4 values of the 2x2 sub quadrant
No
10
Represent 2x2 block by its average

ADAPTIVE COMPRESSION CODING USING TABLE LOOKUP


1
START
2
Divide the image into 4x4 Blocks
3
Choose smoothness threshold (13-50)
4
Find the R : range of the block (4x4)
5
If R is < Smoothness Threshold
Yes
6
Represent block by its average
No
7
If R is > 120
Yes
8
Find range of each 2x2 sub quadrant (Rs)
No
11
Use AMBTC
8
If Rs > 60
Yes
9
Store 4 values of the 2x2 sub quadrant
No
10
Represent 2x2 block by its average

```
/* ABOUTH.C */

#include<graphics.h>
void abouth1()
{
int gd=DETECT,gm=DETECT;
initgraph(&gd,&gm,"");

setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(160,60,DARKGRAY);

setcolor(WHITE);
rectangle(5,35,635,445);
setfillstyle(SOLID_FILL,BLUE);
floodfill(160,60,WHITE);

setcolor(LIGHTMAGENTA);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
outtextxy(180,57,"ABOUT THE PROJECT");

setcolor(LIGHTGRAY);
settextstyle(SANS_SERIF_FONT,HORIZ_DIR,1);
outtextxy(20,95,"    The main aim of the project is to
compress and decompress the ");
outtextxy(20,120,"test images using the BLOCK TRUNCATION
CODING techniques.This BTC");
outtextxy(20,145,"techniques take less computational time
when compared with other");
outtextxy(20,170,"algorithms.");
outtextxy(20,210,"    The evaluation of these algorithms is
done based on factors ");
outtextxy(20,235,"like computational speed,compression ratio
and the quality of the ");
outtextxy(20,260,"reconstructed images.");
getch();
closegraph();
}


/* ACC.C */

            /* Program for compression using ACC */

#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<dos.h>
#include<alloc.h>
#include<math.h>
#define MAXCOLS 320
void dec()
{
```

```
unsigned int (huge *a)[320];
int temp1,temp2;
time_t time1,time2;
FILE *fp1,*fp2;
char fn1[15],fn2[15];
time(&time1);
clrscr();
printf("enter the compressed file name: ");
scanf("%s",fn1);
printf("enter  the file name in which decompressed image  is
to be stored: ");
scanf("%s",fn2);
fp1=fopen(fn1,"rb");
fp2=fopen(fn2,"wb");
printf("please  note  down  decompressed  file  name  for
display.\n");
printf("press <ENTER> for start of decompression.\n");
getch();
printf("please wait..., decompression is going on.\n");
a=farcalloc(sizeof(*a),200);
i=0;
j=0;
row=0;
col=0;
while(!feof(fp1))
  {
     ch=getc(fp1);
     ch6=ch&0xc0;
     if(ch6==0x40)
       {
         ch1=ch&0x3f;
         i=row;
         j=col;
        for(i=row;i<(row+4);i++)
          for(j=col;j<(col+4);j++)
            *(*(a+i)+j)=ch1*4;
         col=col+4;
        if(col==320)
          {
            col=0;
            row=row+4;
           }
       }
      else  if(ch6==0x80)
        {
         ch1=ch&0x3f;
         ch2=getc(fp1);
         ch4=getc(fp1);
         ch5=getc(fp1);
         inc=0;
         ch3=ch4& 0x080;
         if(ch3!=0) inc++;
         ch3=ch4& 0x040;
         if(ch3!=0) inc++;
  if(ch3!=0) inc++;
  ch3=ch4& 0x010;
  if(ch3!=0) inc++;
  ch3=ch4& 0x08;
  if(ch3!=0) inc++;
  ch3=ch4& 0x04;
  if(ch3!=0) inc++;
  ch3=ch4& 0x02;
 if(ch3!=0) inc++;
 ch3=ch4& 0x01;
 if(ch3!=0) inc++;
 ch3=ch5& 0x080;
 if(ch3!=0) inc++;
 ch3=ch5& 0x040;
 if(ch3!=0) inc++;
 ch3=ch5& 0x020;
if(ch3!=0) inc++;
ch3=ch5& 0x010;
if(ch3!=0) inc++;
ch3=ch5& 0x08;
if(ch3!=0) inc++;
ch3=ch5& 0x04;
if(ch3!=0) inc++;
ch3=ch5& 0x02;
if(ch3!=0) inc++;
ch3=ch5& 0x01;
if(ch3!=0) inc++;
temp=16-inc;
if(temp!=0)
 if(inc!=0)
  {
   temp1=(8*ch2)/temp;
   temp2=(8*ch2)/inc;
  }
else
 {
  temp1=0;
  temp2=0;
  }
c=(ch1*4)-temp2;
b=(ch1*4)+temp1;
i=row;
j=col;
ch3=ch4&0x0001;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4&0x0002;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4&0x0004;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4&0x0008;
```

```
i++;
ch3=ch4&0x0010;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4&0x0020;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4&0x0040;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4&0x0080;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j=col;
i++;
ch3=ch5&0x0001;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch5&0x0002;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch5&0x0004;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch5&0x0008;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j=col;
i++;
ch3=ch5&0x0010;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch5&0x0020;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch5&0x0040;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch5&0x0080;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
col+=4;
if(col==320)
 {
  row+=4;
  col=0;
  }
}
else
 {
      m=0;
      l=0;
      while(m<16)
      {
      m+=4;
      ch1=getc(fp1);
```

```
            {
              ch1=ch1*4;
              for(i=l;i<(l+4);i++)
                d[i]=ch1;
                l=l+4;
            }
             else
             {
               ch2=ch1&0x7f;
               ch2=ch2*4;
               ch3=getc(fp1);
               ch4=getc(fp1);
               ch5=getc(fp1);
               i=l;
               d[i]=ch2;
               d[i+1]=ch3;
               d[i+2]=ch4;
               d[i+3]=ch5;
               l=l+4;
             }
         }
         i=row;
         j=col;
         z=0;
         for(i=row;i<row+4;i++)
          for(j=col;j<col+4;j++)
            *(*(a+i)+j)=d[z++];
         col=col+4;
         if(col==320)
          {
           col=0;
           row=row+4;
          }
    }
  }
     fclose(fp1);
/*     greymode();
     i=0;
     j=0;
     for( f=0;f<200;f++)
      for(g=0;g<320;g++)
      {
        gputpixel(g,f,(*(*(a+i)+j)/4));
        j++;
        if(j==320)
         {
          j=0;
          i+=1;
         }
       } */
       for(i=0;i<200;i++)
        for(j=0;j<320;j++)
         putc(*(*(a+i)+j),fp2);
       time(&time2);
       printf("time    taken   for   decompression   is    %lf
seconds.\n",difftime(time2,time1));
       getch();

}
void acc1()
{
 int check[]={0x01,0x02,0x04,0x08,0x010,0x020,0x040,0x080,
              0x01,0x02,0x04,0x08,0x010,0x020,0x040,0x080};
 unsigned int x,y,i,j,l,t,b[4][4],c[16],p,z,temp,m;
 int r,range,max,min,temp1,u,v,threshold;
 float  sum,ave,med,inc,median;
 unsigned long int ct;
 time_t time1,time2;
 int(huge *a)[MAXCOLS];
 FILE *fp1,*fp2;
 char fn[15],fn1[15];
 a=farcalloc(sizeof(*a),200);
 clrscr();
 gotoxy(5,10);
 printf("ENTER THE IMAGE FILE NAME : ");
 scanf("%s",fn);
 printf("ENTER  THE SMOOTHNESS THRESHOLD(between 13  to  50):
");
 scanf("%d",&threshold);
 printf("ENTER THE FILE NAME IN WHICH COMPRESSED IMAGE IS  TO
BE STORED: ");
 scanf("%s",fn1);
 printf("plese   note   down   compressed   file   name   for
decompression.\n");
 printf("press <ENTER> for start of compression process.\n");
 getch();
 printf("please wait...,compression is going on.\n");
 if((fp1=fopen(fn,"rb"))==NULL)
 {
  printf("file cannot be opened.\n");
  exit(0);
 }
 time(&time1);
 fp2=fopen(fn1,"wb");
 for(y=0;y<200;y++)
  for(x=0;x<320;x++)
   {
    *(*(a+y)+x)=getc(fp1);
    }
  ct=0;
  l=0;
  p=0;
  while(ct<4000)
   {
          z=0;
        for(i=0;i<4;i++)
 i=0;
 j=0;
 for(y=l;y<l+4;++y)
  for(x=p;x<p+4;x++)
  {
    b[i][j]=*(*(a+y)+x);
     j++;
     if(j==4)
       {
          j=0;
          i++;
        }
   }
 max=b[0][0];
 min=b[0][0];
for(i=0;i<4;i++)
 {
  for(j=0;j<4;j++)
  {
   if(max<b[i][j])
    max=b[i][j];
   if(min>b[i][j])
    min=b[i][j];
  }
 }

 range=max-min;
 if(range<threshold)
  {
   sum=0.0;
   ave=0.0;
   for(i=0;i<4;i++)
    for(j=0;j<4;j++)
     sum+=b[i][j];
   ave=sum/16;
   t=0x40;
   t=t|(int)(ave/4);
   putc(t,fp2);
  }
  else if(range<120)
   {
    sum=0.0;
    ave=0.0;
    for(i=0;i<4;i++)
     for(j=0;j<4;j++)
      sum+=b[i][j];
    ave=sum/16;
    t=0x80;
    t=t|(int)(ave/4);
    putc(t,fp2);
    med=0;
    median=0;
    for(i=0;i<4;i++)
  median=med/16;
  putc((int)median,fp2);
  for(i=0;i<4;i++)
   for(j=0;j<4;j++)
    {
     c[z]=b[i][j];
     z++;
    }
  t=0;
   for(z=0;z<8;z++)
   if(c[z]>ave)
    t=t|check[z];
  putc(t,fp2);
  t=0;
  for(z=8;z<=15;z++)
  if(c[z]>ave)
   t=t|check[z];
  putc(t,fp2);
}
else
 {
   t=0xc0;
   u=0;
   v=0;
   putc(t,fp2);
   m=0;
   while(m<16)
    {
       max=b[u][v];
       min=b[u][v];
       sum=0;
       for(i=u;i<(u+2);i++)
        for(j=v;j<(v+2);j++)
         {
          sum+=b[i][j];
          if(max<b[i][j]) max=b[i][j];
          if(min>b[i][j]) min=b[i][j];
         }
       r=max-min;
       if(r<60)
        {
          ave=sum/4;
          t=0x00;
          t=t|(int)(ave)/4;
          putc(t,fp2);
          v=v+2;
          if(v==4)
           {
            v=0;u++;
           }
        }
       else
        {
```

```
                    t=t|(temp1/4);
                    putc(t,fp2);
                    putc(b[u][v+1],fp2);
                    putc(b[u+1][v],fp2);
                    putc(b[u+1][v+1],fp2);
                    v=v+2;
                    if(v==4)
                     {
                       v=0;
                       u++;
                     }
                  }
                 m+=4;
             }
          }
          ct++;
          p=p+4;
          if(p==320)
           {
             l+=4;
             p=0;
           }
     }
    getch();
        fclose(fp2);
        fclose(fp1);
        farfree(a);
        getch();
        time(&time2);
        printf("the image has been compressed.\n");
        printf("\n   time   taken   for   compression   is
%lf\n",difftime(time2,time1));
        dec();
        getch();
}        /* Program for decompression for ACC coding */

/* ACCH.C */

#include<graphics.h>
void acch1()
{
int gd=DETECT,gm=DETECT;
initgraph(&gd,&gm,"");

setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(160,60,DARKGRAY);

setcolor(WHITE);
rectangle(5,35,635,445);
setfillstyle(SOLID_FILL,BLUE);
floodfill(160,60,WHITE);

setcolor(WHITE);
```

```
ALGORITHM");
setcolor(RED);
outtextxy(111,57,"THE    ADAPTIVE    COMPRESSION    CODING
ALGORITHM");
setcolor(LIGHTGRAY);
settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
outtextxy(30,95,"    In  this   algorithm   the   image   is
divided  into  blocks of  size  4x4.");
outtextxy(30,120,"Then each  block is  categorized  into  one
of the  following three types.");
outtextxy(30,145,"1.  smooth  block  ........closely   spaced
pixel values.");
outtextxy(30,170,"2.   edge  block     ........containing   a
definitive edge.");
outtextxy(30,195,"3.   textured   block.......having      an
undefinable picture orientation.");
outtextxy(30,235,"   After  categorizing each block, if  the
block is a smooth  block ,it is");
outtextxy(30,260,"represented by its average.If the block  is
a edge  block,it is represented");
outtextxy(30,285,"by  a quadtree.If the block is a  textured
block,it is  coded  using AMBTC");
outtextxy(30,325,"    In  the decompression process  for  the
textured block,AMBTC decompression");
outtextxy(30,350,"algorithm  is  used.For  the   compression,
the user can define the threshold");
outtextxy(30,375,"value.");
getch();
closegraph();
}

/* ACCLOOKH.C */

#include<graphics.h>
void aclookh1()
{
int gd=DETECT,gm=DETECT;
initgraph(&gd,&gm,"");

setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(160,60,DARKGRAY);

setcolor(WHITE);
rectangle(5,35,635,445);
setfillstyle(SOLID_FILL,BLUE);
floodfill(160,60,WHITE);

setcolor(WHITE);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
outtextxy(110,57,"THE ACC TECHNIQUE USING LOOKUP TABLE");
setcolor(RED);
outtextxy(111,57,"THE ACC TECHNIQUE USING LOOKUP TABLE");
setcolor(LIGHTGRAY);
```

```
are coded instead of 16 bits");
outtextxy(30,120,"of  the binary bit plane.This is  based  on
the occurence  probability of");
outtextxy(30,145,"certain bit plane patterns.The 8 bits  that
are coded will  specify the");
outtextxy(30,170,"value of the other bits in the plane.");
outtextxy(30,205,"        In this  ACC  the  image is  divided
into  blocks of  size  4x4.");
outtextxy(30,230,"Then each  block is  categorized  into  one
of the  following three types.");
outtextxy(30,255,"1.  smooth  block  ........closely   spaced
pixel values.");
outtextxy(30,280,"2.   edge  block     ........containing   a
definitive edge.");
outtextxy(30,305,"3.   textured   block.......having     an
undefinable picture orientation.");
outtextxy(30,340,"    After  categorizing each block, if  the
block is a smooth  block ,it is");
outtextxy(30,365,"represented by its average.If the block  is
a edge  block,it is represented");
outtextxy(30,390,"by  a quadtree.If the block is a  textured
block,it is  coded  using AMBTC.");
getch();
closegraph();
}

/* ACCLOOKUP.C */
      /* Program for compression using ACC with Table lookup
*/

#include<stdio.h>
#include<time.h>
#include<conio.h>
#include<dos.h>
#include<alloc.h>
#include<math.h>
#define MAXCOLS 320
void c();
void dc();

void aclook1()
{
c();
dc();
}
void c()
{
 int check[]={0x01,0x02,0x04,0x08,0x010,0x020,0x040,0x080};
 unsigned int x,y,i,j,l,t,b[4][4],c[16],p,z,temp,m;
 int r,range,max,min,temp1,u,v,threshold;
 float  sum,ave,med,inc,median;
 long int ct;
 time_t time1,time2;
 int(huge *a)[MAXCOLS];
 char fn[15],fn1[15];
 a=farcalloc(sizeof(*a),200);
 clrscr();
 gotoxy(5,10);
 time(&time1);
 printf("enter the image file name : ");
 scanf("%s",fn);
 if((fp1=fopen(fn,"rb"))==NULL)
 {
  printf("file cannot be opened.\n");
  exit(0);
 }
 printf("enter the smoothness threshold : ");
 scanf("%d",&threshold);
 printf("enter the file name in which compressed image is  to
be stored : ");
 scanf("%s",fn1);
 printf("please  note  down  the  compressed  file  name  for
decompression purpose.\n");
 printf("press<ENTER>for start of compression process.\n");
 getch();
 printf("please wait...., compression is going on.\n");
 fp2=fopen(fn1,"wb");
 for(y=0;y<200;y++)
  for(x=0;x<320;x++)
    *(*(a+y)+x)=getc(fp1);

 ct=0;
 l=0;
 p=0;
 while(ct<64000)
  {
         z=0;
       for(i=0;i<4;i++)
        for(j=0;j<4;j++)
         b[i][j]=0;
       i=0;
       j=0;
       for(y=l;y<l+4;++y)
        for(x=p;x<p+4;x++)
        {
           b[i][j]=*(*(a+y)+x);
            j++;
            if(j==4)
              {
                 j=0;
                 i++;
               }
          }
        max=b[0][0];
        min=b[0][0];
       for(i=0;i<4;i++)
        {
```

```
  if(max<b[i][j])
   max=b[i][j];
  if(min>b[i][j])
   min=b[i][j];
 }
}
range=max-min;

if(range<threshold)
 {
  sum=0.0;
  ave=0.0;
  for(i=0;i<4;i++)
   for(j=0;j<4;j++)
    sum+=b[i][j];
  ave=sum/16;
  t=0x40;
  t=t|(int)(ave/4);
  putc(t,fp2);
}
else if(range<120)
 {
  sum=0.0;
  ave=0.0;
  for(i=0;i<4;i++)
   for(j=0;j<4;j++)
    sum+=b[i][j];
  ave=sum/16;
  t=0x80;
  t=t|(int)(ave/4);
  putc(t,fp2);
  med=0;
  median=0;
  for(i=0;i<4;i++)
   for(j=0;j<4;j++)
    med=med+abs(b[i][j]-ave);
  median=med/16;
  putc((int)median,fp2);
  for(i=0;i<4;i++)
   for(j=0;j<4;j++)
    {
     c[z]=b[i][j];
     z++;
    }
  t=0;
  if(c[0]>ave)
    t=t|check[0];
  if(c[3]>ave)
    t=t|check[1];
  if(c[5]>ave)
    t=t|check[2];
  if(c[6]>ave)
    t=t|check[3];
  if(c[10]>ave)
    t=t|check[5];
  if(c[12]>ave)
    t=t|check[6];
  if(c[15]>ave)
    t=t|check[7];
  putc(t,fp2);
}
else
 {
   t=0xc0;
   u=0;
   v=0;
   putc(t,fp2);
   m=0;
   while(m<16)
    {
       max=b[u][v];
       min=b[u][v];
       sum=0;
       for(i=u;i<(u+2);i++)
        for(j=v;j<(v+2);j++)
         {
          sum+=b[i][j];
          if(max<b[i][j]) max=b[i][j];
          if(min>b[i][j]) min=b[i][j];
         }
       r=max-min;
       if(r<60)
        {
          ave=sum/4;
          t=0x00;
          t=t|(int)(ave)/4;
          putc(t,fp2);
          v=v+2;
          if(v==4)
           {
            v=0;u++;
           }
        }
       else
        {
          t=0x80;
          temp1=b[u][v];
          t=t|(temp1/4);
          putc(t,fp2);
          putc(b[u][v+1],fp2);
          putc(b[u+1][v],fp2);
          putc(b[u+1][v+1],fp2);
          v=v+2;
          if(v==4)
           {
             v=0;
```

```
                }
                m+=4;
            }
         }
         ct+=16;
         p=p+4;
         if(p==320)
          {
            l+=4;
            p=0;
          }
   }
        fclose(fp2);
        fclose(fp1);
        farfree(a);
        time(&time2);
        printf("the image has been compressed.\n");
        printf("time    taken    for    compression    is    %lf
seconds.\n",difftime(time2,time1));
}

void dc()
{
 unsigned                                                   int
b,c,ch,ch1,ch2,ch3,ch4,inc,temp,ch5,ch6,ch7,ch8;
 unsigned int row,i,j,u,v,m,l,z,d[16],f,g;
 unsigned long int col;
 unsigned int (huge *a)[320];
 int temp1,temp2;
 time_t time1,time2;
 char fn1[10],fn2[10];
 FILE *fp1,*fp2;
 time(&time1);
 clrscr();
 printf("enter the compressed image file name : ");
 scanf("%s",fn1);
 printf("enter  the file name in which decompressed image  is
to be stored : ");
 scanf("%s",fn2);
 printf("please  note down the decompressed image  file  name
for display purpose.\n");
 printf("press<ENTER>for start of decompression process.\n");
 getch();
 printf("please  wait....,  decompression  process  is  going
on.\n");
 fp1=fopen(fn1,"rb");
 fp2=fopen(fn2,"wb");
 a=farcalloc(sizeof(*a),200);
 i=0;
 j=0;
 row=0;
 col=0;
```

```
ch6=ch&0xc0;
if(ch6==0x40)
 {
   ch1=ch&0x3f;
   i=row;
   j=col;
  for(i=row;i<(row+4);i++)
    for(j=col;j<(col+4);j++)
      *(*(a+i)+j)=ch1*4;
   col=col+4;
  if(col==320)
    {
      col=0;
      row=row+4;
     }
 }
else  if(ch6==0x80)
  {
    ch1=ch&0x3f;
    ch2=getc(fp1);
    ch4=getc(fp1);
    inc=0;
    ch3=ch4& 0x080;
    if(ch3!=0) inc++;
    ch3=ch4& 0x040;
    if(ch3!=0) inc++;
    ch3=ch4& 0x020;
    if(ch3!=0) inc++;
    ch3=ch4& 0x010;
    if(ch3!=0) inc++;
    ch3=ch4& 0x08;
    if(ch3!=0) inc++;
    ch3=ch4& 0x04;
    if(ch3!=0) inc++;
    ch3=ch4& 0x02;
   if(ch3!=0) inc++;
   ch3=ch4& 0x01;
   if(ch3!=0) inc++;
   ch3=ch4& 0x01;
   ch5=ch4&0x02;
   ch7=ch4&0x04;
   ch8=ch4&0x10;
   if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0))  inc++;
   ch3=ch4& 0x01;
   ch5=ch4&0x02;
   ch7=ch4&0x08;
   ch8=ch4&0x20;
   if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0))  inc++;
   ch3=ch4& 0x01;
   ch5=ch4&0x40;
   ch7=ch4&0x04;
   ch8=ch4&0x08;
   if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0))  inc++;
```

```
ch7=ch4&0x04;
ch8=ch4&0x08;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0))   inc++;
ch3=ch4& 0x01;
ch5=ch4&0x40;
ch7=ch4&0x10;
ch8=ch4&0x20;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0))   inc++;
ch3=ch4& 0x02;
ch5=ch4&0x80;
ch7=ch4&0x10;
ch8=ch4&0x20;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0))   inc++;
ch3=ch4& 0x04;
ch5=ch4&0x10;
ch7=ch4&0x40;
ch8=ch4&0x80;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0))   inc++;
ch3=ch4& 0x08;
ch5=ch4&0x20;
ch7=ch4&0x40;
ch8=ch4&0x80;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0))   inc++;
temp=16-inc;
if(temp!=0)
if(inc!=0)
  {
   temp1=(8*ch2)/temp;
   temp2=(8*ch2)/inc;
  }
else
 {
  temp1=0;
  temp2=0;
  }
c=(ch1*4)-temp2;
b=(ch1*4)+temp1;
i=row;
j=col;
ch3=ch4&0x0001;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4& 0x01;
ch5=ch4&0x02;
ch7=ch4&0x04;
ch8=ch4&0x10;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0)) *(*(a+i)+j)=b;
else  *(*(a+i)+j)=c;
j++;
ch3=ch4& 0x01;
ch5=ch4&0x02;
ch7=ch4&0x08;
ch8=ch4&0x20;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0)) *(*(a+i)+j)=b;
```

```
j++;
ch3=ch4&0x0002;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j=col;
i++;
ch3=ch4& 0x01;
ch5=ch4&0x40;
ch7=ch4&0x04;
ch8=ch4&0x08;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0)) *(*(a+i)+j)=b;
else  *(*(a+i)+j)=c;
j++;
ch3=ch4&0x0004;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4&0x0008;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4& 0x02;
ch5=ch4&0x80;
ch7=ch4&0x04;
ch8=ch4&0x08;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0)) *(*(a+i)+j)=b;
else  *(*(a+i)+j)=c;
j=col;
i++;
ch3=ch4& 0x01;
ch5=ch4&0x40;
ch7=ch4&0x10;
ch8=ch4&0x20;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0)) *(*(a+i)+j)=b;
else  *(*(a+i)+j)=c;
j++;
ch3=ch4&0x0010;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4&0x0020;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4& 0x02;
ch5=ch4&0x80;
ch7=ch4&0x10;
ch8=ch4&0x20;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0)) *(*(a+i)+j)=b;
else  *(*(a+i)+j)=c;
j=col;
i++;
ch3=ch4&0x0040;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
j++;
ch3=ch4& 0x40;
ch5=ch4&0x80;
ch7=ch4&0x10;
ch8=ch4&0x04;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0)) *(*(a+i)+j)=b;
```

```
else  *(*(a+i)+j)=c;
j++;
ch3=ch4& 0x40;
ch5=ch4&0x80;
ch7=ch4&0x20;
ch8=ch4&0x08;
if((ch3!=0&&ch5!=0)||(ch7!=0&&ch8!=0)) *(*(a+i)+j)=b;
else  *(*(a+i)+j)=c;
j++;
ch3=ch4&0x0080;
if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
col+=4;
if(col==320)
{
 row+=4;
 col=0;
 }
}
else
 {
      m=0;
      l=0;
      while(m<16)
      {
      m+=4;
      ch1=getc(fp1);
      ch2=ch1&0x00;
      if(ch2==0x00)
        {
          ch1=ch1*4;
          for(i=l;i<(l+4);i++)
            d[i]=ch1;
            l=l+4;
        }
         else
         {
           ch2=ch1&0x7f;
           ch2=ch2*4;
           ch3=getc(fp1);
           ch4=getc(fp1);
           ch5=getc(fp1);
           i=l;
           d[i]=ch2;
           d[i+1]=ch3;
           d[i+2]=ch4;
           d[i+3]=ch5;
           l=l+4;
         }
     }
     i=row;
     j=col;
     z=0;
     for(i=row;i<row+4;i++)
```

```
          col=col+4;
          if(col==320)
           {
            col=0;
            row=row+4;
           }
    }}

        for(i=0;i<200;i++)
         for(j=0;j<320;j++,col++)
          {putc(*(*(a+i)+j),fp2);
          /*printf("\n in progress%ld",col);*/}
        fclose(fp1);
        fclose(fp2);
        farfree(a);
       time(&time2);
       printf("The image has been decompressed.\n");
       printf("time    taken    for    decompression     is     %lf
seconds.\n",difftime(time2,time1));
}

/* AMBTC.C */
                /* program for compression using AMBTC */

#include<stdio.h>
#include<alloc.h>
#include<process.h>
#include<math.h>
#include<dos.h>
#include<time.h>
#include<conio.h>
#define MAXCOLS 320
void deco()
 {
  unsigned int b,c,ch,ch1,ch2,ch3,ch4;
  float inc,temp;
  unsigned int row,col,i,j;
  long int count;
  float  temp1,temp2;
  time_t time1;
  time_t time2;
  unsigned int (huge *a)[320];
  FILE *fp1,*fp2;
  char fn[15],fn1[15];
  time(&time1);
  clrscr();
  printf("ENTER COMPRESSED FILE NAME :");
  scanf("%s",fn);
  printf("\nENTER DECOMPRESSED FILE NAME :");
  scanf("%s",fn1);
  printf("please   note   down   decompressed   file   name   for
 display.\n");
  printf("press<ENTER>    for    start    of    decompression
 process.\n");
```

```
printf("please wait...,decompression is going on.\n");
fp1=fopen(fn,"rb");
fp2=fopen(fn1,"wb");
a=farcalloc(sizeof(*a),200);
i=0;
j=0;
row=0;
col=0;
count=0;
while(count<16000)
{
   ch=getc(fp1);
   ch1=getc(fp1);
   ch2=getc(fp1);
   ch4=getc(fp1);
   count+=4;
   inc=0;
   ch3=ch2& 0x080;
   if(ch3!=0) inc++;
   ch3=ch2& 0x040;
   if(ch3!=0) inc++;
   ch3=ch2& 0x020;

   if(ch3!=0) inc++;
   ch3=ch2& 0x010;
   if(ch3!=0) inc++;
   ch3=ch2& 0x08;
   if(ch3!=0) inc++;
   ch3=ch2& 0x04;
   if(ch3!=0) inc++;
   ch3=ch2& 0x02;
   if(ch3!=0) inc++;
   ch3=ch2& 0x01;
   if(ch3!=0) inc++;
   ch3=ch4& 0x080;
   if(ch3!=0) inc++;
   ch3=ch4& 0x040;
   if(ch3!=0) inc++;
   ch3=ch4& 0x020;
   if(ch3!=0) inc++;
   ch3=ch4& 0x010;
   if(ch3!=0) inc++;
   ch3=ch4& 0x08;
   if(ch3!=0) inc++;
   ch3=ch4& 0x04;
   if(ch3!=0) inc++;
   ch3=ch4& 0x02;
   if(ch3!=0) inc++;
   ch3=ch4& 0x01;
   if(ch3!=0) inc++;
   temp=16-inc;
   if(temp!=0)
    if(inc!=0)
   temp2=(8*ch1)/temp;
   }
else
{
 temp1=0.0;
 temp2=0.0;
 }
c=ch-(int)temp2;
b=ch+(int)temp1;
  i=row;
  j=col;
  ch3=ch2&0x0001;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0002;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0004;

  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0008;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j=col;
  i++;
  ch3=ch2&0x0010;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0020;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0040;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0080;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j=col;
  i++;
  ch3=ch4&0x0001;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch4&0x0002;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch4&0x0004;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;.
  j++;
  ch3=ch4&0x0008;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j=col;
  i++;
  ch3=ch4&0x0010;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
```

```
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=ch4&0x0040;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=ch4&0x0080;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    col+=4;
    if(col==320)
     {
       row+=4;
       col=0;
      }
  }

     fclose(fp1);
     for( i=0;i<200;i++)
      for(j=0;j<320;j++)
       putc(*(*(a+i)+j),fp2);
       fclose(fp2);
       farfree(a);
       time(&time2);
     printf("time    taken    for   decompression    is    %lf
seconds.\n",difftime(time2,time1));
     getch();
 }
void ambtc1()
{
 int
check[]={0x01,0x02,0x04,0x08,0x010,0x020,0x040,0x080,0x01,0x02
 unsigned int x,y,l,t,b[16],p,z;
 unsigned int  sum,ave,median;
 long int ct,med;
 double diff;
 time_t time1;
 time_t time2;
  int(huge *a)[MAXCOLS];
 FILE *fp1,*fp2;
 char fn[15],fn1[15];
 a=farcalloc(sizeof(*a),200);
 time(&time1);
 clrscr();
 gotoxy(5,10);
 printf("ENTER THE IMAGE FILE NAME : ");
 scanf("%s",fn);
 printf("\nENTER THE IMAGE FILE NAME OF COMPRESSED IMAGE: ");
 scanf("%s",fn1);
 printf("Please  note  down  the  compressed  file  name  for
decompression process.\n");
 printf("press <ENTER> for start of compression process.\n");
 getch();
 printf("please wait...,compression is going on.\n");
 if((fp1=fopen(fn,"rb"))==NULL)
 exit(0);
}
fp2=fopen(fn1,"wb");
for(y=0;y<200;y++)
 for(x=0;x<320;x++)
   *(*(a+y)+x)=getc(fp1);
ct=0;
l=0;
p=0;
while(ct<64000)
 {
          z=0;
        for(y=l;y<l+4;++y)
        for(x=p;x<p+4;x++)
         {
           b[z]=*(*(a+y)+x);
            z++;
          }
         sum=0;
         ave=0;
         for(z=0;z<=15;z++)
          sum+=b[z];
          ave=sum/16;
          med=0;
          median=0;
          for(z=0;z<=15;z++)
          med=med+abs(b[z]-ave);
          median=med/16;
          putc(ave,fp2);
          putc(median,fp2);
          t=0;
          for(x=0;x<=7;x++)
           if(b[x]>ave)
            t=t|check[x];
          putc(t,fp2);
          t=0;
          for(x=8;x<=15;x++)
          if(b[x]>ave)
           t=t|check[x];
          putc(t,fp2);
          ct+=16;
          p+=4;
          if(p==320)
           {
             l+=4;
             p=0;
           }
         }
         time(&time2);
         fclose(fp2);
         fclose(fp1);
         farfree(a);
         printf("the image has been compressed.\n");
         [illegible] for compression is %lf
        deco();
        getch();
}       /* Program for decompression for AMBTC coding */


/* AMBTCH.C */

#include<graphics.h>
void ambtch1()
{
int gd=DETECT,gm=DETECT;
initgraph(&gd,&gm,"");

setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(160,60,DARKGRAY);

setcolor(WHITE);
rectangle(5,35,635,445);
setfillstyle(SOLID_FILL,BLUE);
floodfill(160,60,WHITE);

setcolor(WHITE);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
outtextxy(180,57,"THE ABSOLUTE MOMENT BTC ALGORITHM");
setcolor(RED);
outtextxy(182,57,"THE ABSOLUTE MOMENT BTC ALGORITHM");
setcolor(LIGHTGRAY);
settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
outtextxy(40,95,"    In AMBTC the given image is divided into
blocks of size 4x4.");
outtextxy(40,120,"  Then mean, absolute moment of each  block
is found.The mean value");
outtextxy(40,145,"gives   central  tendency  and   dispersion
about the mean is given by");
outtextxy(40,170,"absolute moment.");
outtextxy(40,210,"   Then each block is represented by a  two
level one bit quantizer");
outtextxy(40,235,"(0  or  1) based on whether  the  pixel  is
above or below the mean of ");
outtextxy(40,260,"the  block. But the  resultant  compressed
image contains each block's");
outtextxy(40,285,"information mean and absolute moment.");
outtextxy(40,325,"    In the decompression process  the  mean
and absolute moment are used");
outtextxy(40,350,"to  calculate the values of A and  B  which
are used to replace the pixel");
outtextxy(40,375,"values  0 and 1.In this  method  compressed
image size is one fourth of");
outtextxy(40,400,"the original image.");
getch();
closegraph();
}
```

```
#include<stdio.h>
#include<process.h>
#include<conio.h>
#include<dos.h>
#include<alloc.h>
#include<process.h>
#include<math.h>
#include<time.h>
#define MAXCOLS 320
void decomp();
void btc1()
{
  int check[]={0x01,0x02,0x04,0x08,0x010,0x020,0x040,0x080};
  unsigned int x,y,l,t,b[16],p,z;
  unsigned int  sum,ave,median;
  long int ct,med;
  double diff;
  time_t time1;
  time_t time2;
   int(huge *a)[MAXCOLS];
  FILE *fp1,*fp2;
  char fn[15],fn1[15];
  a=farcalloc(sizeof(*a),200);
  clrscr();
  gotoxy(5,10);
  printf("ENTER THE IMAGE FILE NAME : ");
  scanf("%s",fn);
  printf("\nENTER THE IMAGE NAME IN WHICH COMPRESSED IMAGE  TO
 BE STORED: ");
  scanf("%s",fn1);
  printf("\nPLEASE NOTE DOWN COMPRESSED FILE NAME FOR DECODING
 PURPOSE.\n");
  if((fp1=fopen(fn,"rb"))==NULL)
  {
   printf("file cannot be opened.\n");
   exit(0);
  }
  printf("\nPlease wait...,Compression is going on.\n");
  time(&time1);
  fp2=fopen(fn1,"wb");
  for(y=0;y<200;y++)
   {
    for(x=0;x<320;x++)
     {
       *(*(a+y)+x)=getc(fp1);
     }
   }
  ct=0;
  l=0;
  p=0;

  while(ct<64000)
```

```
        for(y=l;y<l+4;++y)
        for(x=p;x<p+4;x++)
         {
           b[z]=*(*(a+y)+x);
             z++;
          }
         sum=0;
         ave=0;
         for(z=0;z<=15;z++)
          sum+=b[z];
          ave=sum/16;
          med=0;
          median=0;
          for(z=0;z<=15;z++)
          med=med+(b[z]*b[z]);
          median=med/16;
          diff=median-(ave*ave);
          diff=sqrt((double)diff);
          putc(ave,fp2);
          putc(diff,fp2);
          t=0;
          for(x=0;x<=7;x++)
           if(b[x]>ave)
            t=t|check[x];
          putc(t,fp2);
          t=0;
          for(x=8;x<=15;x++)
          if(b[x]>ave)
           t=t|check[x-8];
          putc(t,fp2);
          ct+=16;
          p=p+4;
          if(p==320)
           {
             l+=4;
             p=0;
           }
         }
         time(&time2);
         fclose(fp2);
         fclose(fp1);
         farfree(a);
         printf("the image has been compressed.\n");
         printf("compression              time           is              %lf
seconds.\n",difftime(time2,time1));
         decomp();
         getch();
}         /* Program for decompression for BTC coding */


void decomp()
{
 unsigned int b,c,ch,ch1,ch2,ch3,ch4,inc,temp;
```

```
char fn[15],fn1[15];
double temp1,temp2;
time_t time1;
time_t time2;
unsigned int (huge *a)[320];

FILE *fp1,*fp2;
clrscr();
gotoxy(3,10);
time(&time1);
printf("ENTER THE COMPRESSED FILE NAME : ");
scanf("%s",fn);
printf("\nENTER THE DECOMPRESSED FILE NAME : ");
scanf("%s",fn1);
printf("\nPlese   note  down  decompressed  file  name   for
display\n");
printf("   Press   ENTER   for   start   of   decompression
process.\n");
getch();
printf("Please wait...,decompression is going on.\n");
fp1=fopen(fn,"rb");
fp2=fopen(fn1,"wb");
a=farcalloc(sizeof(*a),200);
i=0;
j=0;
row=0;
col=0;
count=0;
while(count<16000)
{
   ch=getc(fp1);
   ch1=getc(fp1);
   ch2=getc(fp1);
   ch4=getc(fp1);
   count+=4;
   inc=0;
   ch3=ch2& 0x080;
   if(ch3!=0) inc++;
   ch3=ch2& 0x040;
   if(ch3!=0) inc++;
   ch3=ch2& 0x020;
   if(ch3!=0) inc++;
   ch3=ch2& 0x010;
   if(ch3!=0) inc++;
   ch3=ch2& 0x08;
   if(ch3!=0) inc++;
   ch3=ch2& 0x04;
   if(ch3!=0) inc++;
   ch3=ch2& 0x02;
   if(ch3!=0) inc++;
   ch3=ch2& 0x01;
   if(ch3!=0) inc++;
   ch3=ch4& 0x080;
```

```
if(ch3!=0) inc++;
ch3=ch4& 0x020;
if(ch3!=0) inc++;
ch3=ch4& 0x010;
if(ch3!=0) inc++;
ch3=ch4& 0x08;
if(ch3!=0) inc++;
ch3=ch4& 0x04;
if(ch3!=0) inc++;
ch3=ch4& 0x02;
if(ch3!=0) inc++;
ch3=ch4& 0x01;
if(ch3!=0) inc++;
temp=16-inc;
if(temp!=0)
  {
      temp1=sqrt((double)(inc/temp));
      if(inc!=0)
      temp2=sqrt((double)(temp/inc));
      else temp2=sqrt((double)(temp));
      c=ch-(ch1*(int)temp1);
      b=ch+(ch1*(int)temp2);
   }
  else
  {
   c=ch-(ch1*(int)sqrt(inc));
   b=ch;
  }
  i=row;
  j=col;
  ch3=ch2&0x0001;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0002;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0004;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0008;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j=col;
  i++;
  ch3=ch2&0x0010;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0020;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0040;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0080;
```

```
    i++;
    ch3=ch4&0x0001;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=ch4&0x0002;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=ch4&0x0004;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=ch4&0x0008;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j=col;
    i++;
    ch3=ch4&0x0010;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=ch4&0x0020;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=ch4&0x0040;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=ch4&0x0080;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    col+=4;
    if(col==320)
     {
       row+=4;
       col=0;
      }
    }
    fclose(fp1);
    for( i=0;i<200;i++)
     for(j=0;j<320;j++)
      putc(*(*(a+i)+j),fp2);
      fclose(fp2);
    time(&time2);
    printf("decompression is over.\n");
    printf("\nTIME    TAKEN    FOR    DECOMPRESSION    IS    %lf
seconds.\n",difftime(time2,time1));
    getch();
 }

/* BTCH1.C */

#include<graphics.h>
void btch1()
{
int gd=DETECT,gm=DETECT;
initgraph(&gd,&gm,"");

setfillstyle(SOLID_FILL,LIGHTGRAY);
```

```
setcolor(WHITE);
rectangle(5,35,635,445);
setfillstyle(SOLID_FILL,BLUE);
floodfill(160,60,WHITE);

setcolor(WHITE);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
outtextxy(180,57,"THE BASIC BTC ALGORITHM");
setcolor(RED);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
outtextxy(182,57,"THE BASIC BTC ALGORITHM");

setcolor(LIGHTGRAY);
settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
outtextxy(40,95,"    In basic BTC the given image is  divided
into blocks of size 4x4.");
outtextxy(40,120,"Then mean,standard deviation of each  block
is found.The mean value ");
outtextxy(40,145,"gives    central  tendency   and   standard
deviation gives the dispersion");
outtextxy(40,170,"about the central tendency");
outtextxy(40,210,"   Then each block is represented by a  two
level one bit quantizer");
outtextxy(40,235,"(0  or  1) based on whether  the  pixel  is
above or below the mean of ");
outtextxy(40,260,"the  block. But the  resultant  compressed
image contains each block's");
outtextxy(40,285,"information mean and standard deviation.");
outtextxy(40,325,"    In decompression process the  mean  and
standard deviation are ");
outtextxy(40,350,"used  to  calculate the values of A  and  B
which are used to replace");
outtextxy(40,375,"pixel  values  0 and 1.In  this  compressed
image size is one fourth of");
outtextxy(40,400,"the original image.");
getch();
closegraph();
}

/* DISPLAY.C */

#include<graphics.h>
void btch1()
{
int gd=DETECT,gm=DETECT;
initgraph(&gd,&gm,"");

setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(160,60,DARKGRAY);

setcolor(WHITE);
rectangle(5,35,635,445);
setfillstyle(SOLID_FILL,BLUE);
```

```
setcolor(WHITE);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
outtextxy(180,57,"THE BASIC BTC ALGORITHM");
setcolor(RED);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
outtextxy(182,57,"THE BASIC BTC ALGORITHM");

setcolor(LIGHTGRAY);
settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
outtextxy(40,95,"    In basic BTC the given image is  divided
into blocks of size 4x4.");
outtextxy(40,120,"Then mean,standard deviation of each  block
is found.The mean value ");
outtextxy(40,145,"gives    central  tendency   and   standard
deviation gives the dispersion");
outtextxy(40,170,"about the central tendency");
outtextxy(40,210,"   Then each block is represented by a  two
level one bit quantizer");
outtextxy(40,235,"(0  or  1) based on whether  the  pixel  is
above or below the mean of ");
outtextxy(40,260,"the  block. But the  resultant  compressed
image contains each block's");
outtextxy(40,285,"information mean and standard deviation.");
outtextxy(40,325,"    In decompression process the  mean  and
standard deviation are ");
outtextxy(40,350,"used  to  calculate the values of A  and  B
which are used to replace");
outtextxy(40,375,"pixel  values  0 and 1.In  this  compressed
image size is one fourth of");
outtextxy(40,400,"the original image.");
getch();
closegraph();
}

/* LOOKUP.C */
       /*  Program  for compression using  AMBTC  with  table
Lookup */

#include<stdio.h>
#include<time.h>
#include<alloc.h>
#include<conio.h>
#include<math.h>
#include<dos.h>
#define MAXCOLS 320
void decom(void)
{
 unsigned int b,c,ch,ch1,ch2,ch3,ch4,ch5,ch6;
 float inc,temp;
 unsigned int row,col,i,j;
 int count;
 double temp1,temp2;
 unsigned int (huge *a)[320];
 char fn1[15],fn2[15];
 time(&time1);
 clrscr();
 printf("enter the compressed image file name : ");
 scanf("%s",fn1);
 printf("enter  the file name in which decompressed image  is
to be stored : ");
 scanf("%s",fn2);
 printf("please   note   down   decompressed   file   name   for
display.\n");
 printf("press <ENTER> for starting decompression.\n");
 getch();
 printf("please wait...,decompression is going on.\n");
 fp1=fopen(fn1,"rb");
 fp2=fopen(fn2,"wb");
 a=farcalloc(sizeof(*a),200);
 i=0;
 j=0;
 row=0;
 col=0;
 count=0;
 while(count<12000)
 {
    ch=getc(fp1);
    ch1=getc(fp1);
    ch2=getc(fp1);
    count+=3;
    inc=0.0;
    ch3=ch2& 0x080;
    if(ch3!=0) inc++;
    ch3=ch2& 0x040;
    if(ch3!=0) inc++;
    ch3=ch2& 0x020;
    if(ch3!=0) inc++;
    ch3=ch2& 0x010;
    if(ch3!=0) inc++;
    ch3=ch2& 0x08;
    if(ch3!=0) inc++;
    ch3=ch2& 0x04;
    if(ch3!=0) inc++;
    ch3=ch2& 0x02;
    if(ch3!=0) inc++;
    ch3=ch2& 0x01;
    if(ch3!=0) inc++;
    ch3=(ch2&0x01);
    ch4=ch2&0x02;
    ch5=(ch2&0x04);
    ch6=ch2&0x10;
    if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0)) inc++;
    ch3=(ch2& 0x01);
    ch4=ch2&0x02;
    ch5=(ch2&0x08);
    ch6=ch2&0x20;
```

```
ch4=ch2&0x40;
ch5=(ch2&0x04);
ch6=ch2&0x08;
if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0)) inc++;
ch3=(ch2& 0x02);
ch4=ch2&0x80;
ch5=(ch2&0x04);
ch6=ch2&0x08;
if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0)) inc++;
ch3=(ch2& 0x01);
ch4=ch2&0x40;
ch5=(ch2&0x10);
ch6=ch2&0x20;
if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0)) inc++;
ch3=(ch2& 0x02);
ch4=ch2&0x80;
ch5=(ch2&0x10);
ch6=ch2&0x20;
if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0)) inc++;
ch3=(ch2& 0x40);
ch4=ch2&0x80;
ch5=(ch2&0x10);
ch6=ch2&0x04;
if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0)) inc++;
ch3=(ch2& 0x40);
ch4=ch2&0x80;
ch5=(ch2&0x08);
ch6=ch2&0x20;
if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0)) inc++;
temp=16-inc;
if(temp!=0)
 if(inc!=0)
  {
     temp1=(8*ch1)/inc;
     temp2=(8*ch1)/temp;
   }
else
{
 temp1=0.0;
 temp2=0.0;
}
 c=ch-(int)temp2;
 b=ch+(int)temp1;
 i=row;
 j=col;
 ch3=ch2&0x0001;
 if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
 j++;
 ch3=(ch2&0x01);
 ch4=ch2&0x02;
 ch5=(ch2&0x04);
 ch6=ch2&0x10;
 if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0))       *(*(a+i)+j)=b;
```

```
    ch3=(ch2& 0x01);
    ch4=ch2&0x02;
    ch5=(ch2&0x08);
    ch6=ch2&0x20;
    if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0))      *(*(a+i)+j)=b;
else *(*(a+i)+j)=c;
    j++;
    ch3=ch2&0x0002;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j=col;
    i++;
    ch3=(ch2& 0x01);
    ch4=ch2&0x40;
    ch5=(ch2&0x04);
    ch6=ch2&0x08;
    if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0))      *(*(a+i)+j)=b;
else *(*(a+i)+j)=c;
    j++;
    ch3=ch2&0x0004;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=ch2&0x0008;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=(ch2& 0x02);
    ch4=ch2&0x80;
    ch5=(ch2&0x04);
    ch6=ch2&0x08;
    if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0))      *(*(a+i)+j)=b;
else *(*(a+i)+j)=c;
    j=col;
    i++;
    ch3=(ch2& 0x01);
    ch4=ch2&0x40;
    ch5=(ch2&0x10);
    ch6=ch2&0x20;
    if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0))      *(*(a+i)+j)=b;
else *(*(a+i)+j)=c;
    j++;
    ch3=ch2&0x0010;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=ch2&0x0020;
    if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
    j++;
    ch3=(ch2& 0x02);
    ch4=ch2&0x80;
    ch5=(ch2&0x10);
    ch6=ch2&0x20;
    if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0))      *(*(a+i)+j)=b;
else *(*(a+i)+j)=c;
    j=col;
    i++;
    ch3=ch2&0x40;
```

```
     j++;
     ch3=(ch2& 0x40);
     ch4=ch2&0x80;
     ch5=(ch2&0x10);
     ch6=ch2&0x04;
     if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0))       *(*(a+i)+j)=b;
else *(*(a+i)+j)=c;
     j++;
     ch3=(ch2& 0x40);
     ch4=ch2&0x80;
     ch5=(ch2&0x08);
     ch6=ch2&0x20;
     if((ch3!=0&&ch4!=0)||(ch5!=0&&ch6!=0))       *(*(a+i)+j)=b;
else *(*(a+i)+j)=c;
     j++;
     ch3=ch2&0x0080;
     if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
     col+=4;
     if(col==320)
      {
        row+=4;
        col=0;
       }
     }
     fclose(fp1);
     for( i=0;i<200;i++)
      for(j=0;j<320;j++)
       putc(*(*(a+i)+j),fp2);
       fclose(fp2);
       farfree(a);
       time(&time2);
       printf("the image has been reconstructed.\n");
       printf("time    taken    for    decompression    is     %lf
 seconds.\n",difftime(time2,time1));
       getch();
  }
 void lookup1()
 {
  int check[]={0x01,0x02,0x04,0x08,0x010,0x020,0x040,0x080};
  unsigned int x,y,l,t,b[16],p,z,i,j,temp,m,n;
  unsigned int  sum,ave,med;
  long int ct;
  time_t time1;
  time_t time2;
   int(huge *a)[MAXCOLS];
  FILE *fp1,*fp2;
  char fn1[15],fn2[15];
  a=farcalloc(sizeof(*a),200);
  time(&time1);
  clrscr();
  gotoxy(5,10);
  printf("enter the original file name: ");
  scanf("%s",fn1);
 scanf("%s",fn2);
 printf("please  note  down compressed image  file  name  for
decompression process.\n");
 printf("press <ENTER> for starting compression.\n");
 getch();
 printf("please wait...,compression is going on.\n");
 if((fp1=fopen(fn1,"rb"))==NULL)
 {
  printf("file cannot be opened.\n");
  exit(0);
 }
 fp2=fopen(fn2,"wb");

 for(y=0;y<200;y++)
  for(x=0;x<320;x++)
    *(*(a+y)+x)=getc(fp1);
 ct=0;
 l=0;
 p=0;
 while(ct<64000)
  {
          z=0;
        for(y=l;y<l+4;++y)
        for(x=p;x<p+4;x++)
         {
           b[z]=*(*(a+y)+x);
            z++;
          }
         sum=0;
         ave=0;
         for(z=0;z<=15;z++)
          sum+=b[z];
          ave=sum/16;
          putc(ave,fp2);
           for(z=0;z<=15;z++)
             med+=abs(b[z]-ave);
             med=med/16;
             putc(med,fp2);
             t=0;
             if(b[0]>ave)
             t=t|check[0];
             if(b[3]>ave)
             t=t|check[1];
             if(b[5]>ave)
             t=t|check[2];
             if(b[6]>ave)
             t=t|check[3];
             if(b[9]>ave)
             t=t|check[4];
             if(b[10]>ave)
             t=t|check[5];
             if(b[12]>ave)
             t=t|check[6];
          putc(t,fp2);
            ct+=16;
            p=p+4;
       if(p==320)
        {
          l+=4;
          p=0;
        }
     }
     time(&time2);
     fclose(fp2);
     fclose(fp1);
     farfree(a);
     printf("the image has been compressed.\n");
     printf("time    taken    for    compression    is    %lf
seconds.\n",difftime(time2,time1));
     getch();
     decom();
}
/*  Program  for decompression for AMBTC  with  Table  Lookup
coding */

/* LOOKUPH.C */

#include<graphics.h>
void lookuph1()
{
int gd=DETECT,gm=DETECT;
initgraph(&gd,&gm,"");

setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(160,60,DARKGRAY);

setcolor(WHITE);
rectangle(5,35,635,445);
setfillstyle(SOLID_FILL,BLUE);
floodfill(160,60,WHITE);

setcolor(WHITE);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
outtextxy(110,57,"THE AMBTC TECHNIQUE USING LOOKUP TABLE");
setcolor(RED);
outtextxy(111,57,"THE AMBTC TECHNIQUE USING LOOKUP TABLE");
setcolor(LIGHTGRAY);
settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
outtextxy(30,95,"     In the table lookup method only 8  bits
are coded instead of 16 bits");
outtextxy(30,120,"of  the binary bit plane.This is  based  on
the occurence  probability of");
outtextxy(30,145,"certain bit plane patterns.The 8 bits  that
are coded will  specify the");
outtextxy(30,170,"value of the other bits in the plane.");
outtextxy(30,210,"     In the decompression process  the  mean
are used to replace the pixel");
outtextxy(30,260,"values 0 and 1.");
getch();
closegraph();
}

/* MBTC.C */

          /* Program for compression using Modified BTC */
#include<stdio.h>
#include<process.h>
#include<conio.h>
#include<dos.h>
#include<alloc.h>
#include<math.h>
#include<time.h>
#define MAXCOLS 320
void compress();
void decompress();
void mbtc1()
{
compress();
decompress();
}
void compress()
{
 int
check[]={0x01,0x02,0x04,0x08,0x010,0x020,0x040,0x080,0x01,0x02
 unsigned int x,y,l,t,b[16],p,z,lower_mean;
 unsigned int  sum,ave,median,inc,temp,total,total1;
 long int ct,med;
 double diff;
 time_t time1;
 time_t time2;
  int(huge *a)[MAXCOLS];
 FILE *fp1,*fp2;
 char fn[15],fn1[15];
 a=farcalloc(sizeof(*a),200);
 clrscr();
 gotoxy(5,10);
 time(&time1);
 printf("ENTER THE IMAGE FILE NAME : ");
 scanf("%s",fn);
 printf("ENTER THE IMAGE NAME IN WHICH COMPRESSED IMAGE TO BE
STORED: ");
 scanf("%s",fn1);
 if((fp1=fopen(fn,"rb"))==NULL)
 {
  printf("file cannot be opened.\n");
  exit(0);
 }
 fp2=fopen(fn1,"wb");
 for(y=0;y<200;y++)
```

```
        {
          *(*(a+y)+x)=getc(fp1);
        }
     }
   ct=0;
   l=0;
   p=0;
   while(ct<64000)
    {
             z=0;
           for(y=l;y<l+4;++y)
           for(x=p;x<p+4;x++)
            {
              b[z]=*(*(a+y)+x);
               z++;
             }
            sum=0;
            ave=0;
            for(z=0;z<=15;z++)
             sum+=b[z];
             ave=sum/16;
/*           med=0;
             median=0;
             for(z=0;z<=15;z++)
             med=med+(b[z]*b[z]);
             median=med/16;
             diff=median-(ave*ave);
             diff=sqrt((double)diff);
             putc(ave,fp2);
             putc(diff,fp2);   */
             inc=0;
             total=0;
             total1=0;
             temp=0;
             for(z=0;z<16;z++)
             {
              if(b[z]<ave)
               {
                inc++;
                total+=b[z];
               }
               else
                {
                 temp++;
                 total1+=b[z];
                }
             }
             if(inc!=0)
              lower_mean=total/inc;
             putc((int)ave,fp2);
             putc(lower_mean,fp2);
             t=0;
             for(x=0;x<=7;x++)
          putc(t,fp2);
          t=0;
          for(x=8;x<=15;x++)
          if(b[x]>ave)
           t=t|check[x];
          putc(t,fp2);
          ct+=16;
          p=p+4;
          if(p==320)
            {
              l+=4;
              p=0;
            }
        }
        time(&time2);
        fclose(fp2);
        fclose(fp1);
        farfree(a);
        printf("the image has been compressed.\n");
        printf("diff time is %lf\n",difftime(time2,time1));
}

void decompress()
{
 unsigned int b,c,ch,ch1,ch2,ch3,ch4,inc,temp;
 unsigned int row,col,i,j;
 int count;
 char fn[15],fn1[15];
 double temp1,temp2;
 time_t time1;
 time_t time2;
 unsigned int (huge *a)[320];

 FILE *fp1,*fp2;
 clrscr();
 gotoxy(3,10);
 time(&time1);
 printf("ENTER THE COMPRESSED FILE NAME : ");
 scanf("%s",fn);
 printf("\nENTER THE DECOMPRESSED FILE NAME : ");
 scanf("%s",fn1);
 fp1=fopen(fn,"rb");
 fp2=fopen(fn1,"wb");
 a=farcalloc(sizeof(*a),200);
 i=0;
 j=0;
 row=0;
 col=0;
 count=0;
 while(count<16000)
 {
    ch=getc(fp1);
```

```
count+=4;
inc=0;
ch3=ch2& 0x080;
if(ch3!=0) inc++;
ch3=ch2& 0x040;
if(ch3!=0) inc++;
ch3=ch2& 0x020;
if(ch3!=0) inc++;
ch3=ch2& 0x010;
if(ch3!=0) inc++;
ch3=ch2& 0x08;
if(ch3!=0) inc++;
ch3=ch2& 0x04;
if(ch3!=0) inc++;
ch3=ch2& 0x02;
if(ch3!=0) inc++;
ch3=ch2& 0x01;
if(ch3!=0) inc++;
ch3=ch4& 0x080;
if(ch3!=0) inc++;
ch3=ch4& 0x040;
if(ch3!=0) inc++;
ch3=ch4& 0x020;
if(ch3!=0) inc++;
ch3=ch4& 0x010;
if(ch3!=0) inc++;
ch3=ch4& 0x08;
if(ch3!=0) inc++;
ch3=ch4& 0x04;
if(ch3!=0) inc++;
ch3=ch4& 0x02;
if(ch3!=0) inc++;
ch3=ch4& 0x01;
if(ch3!=0) inc++;
  if(inc!=0)
  b=ch+(ch-ch1)/inc;
  c=ch1;
  i=row;
  j=col;
  ch3=ch2&0x0001;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0002;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0004;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j++;
  ch3=ch2&0x0008;
  if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
  j=col;
  i++;
  ch3=ch2&0x0010;
```

```
   ch3=ch2&0x0020;
   if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
   j++;
   ch3=ch2&0x0040;
   if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
   j++;
   ch3=ch2&0x0080;
   if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
   j=col;
   i++;
   ch3=ch4&0x0001;
   if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
   j++;
   ch3=ch4&0x0002;
   if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
   j++;
   ch3=ch4&0x0004;
   if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
   j++;
   ch3=ch4&0x0008;
   if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
   j=col;
   i++;
   ch3=ch4&0x0010;
   if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
   j++;
   ch3=ch4&0x0020;
   if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
   j++;
   ch3=ch4&0x0040;
   if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
   j++;
   ch3=ch4&0x0080;
   if(ch3!=0) *(*(a+i)+j)=b; else *(*(a+i)+j)=c;
   col+=4;
   if(col==320)
    {
      row+=4;
      col=0;
     }
   }
   fclose(fp1);
   for( i=0;i<200;i++)
    for(j=0;j<320;j++)
     putc(*(*(a+i)+j),fp2);
     fclose(fp2);
   time(&time2);
   printf("\n\nTIME     TAKEN     FOR     DECOMPRESSION      IS
%lf\n",difftime(time2,time1));
   farfree(a);
 }

/* MBTCH.C */
```

```
void mbtch1()
{
int gd=DETECT,gm=DETECT;
initgraph(&gd,&gm,"");

setfillstyle(SOLID_FILL,LIGHTGRAY);
floodfill(160,60,DARKGRAY);

setcolor(WHITE);
rectangle(5,35,635,445);
setfillstyle(SOLID_FILL,BLUE);
floodfill(160,60,WHITE);

setcolor(WHITE);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
outtextxy(180,57,"THE MODIFIED BTC ALGORITHM");
setcolor(RED);
outtextxy(182,57,"THE MODIFIED BTC ALGORITHM");
setcolor(LIGHTGRAY);
settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
outtextxy(40,95,"     In  modified  BTC the  given  image  is
divided into blocks of size ");
outtextxy(40,120,"4x4. Then mean, lower mean of each block is
found. The mean value ");
outtextxy(40,145,"gives   central tendency and lower mean  is
mean of the pixels whose ");
outtextxy(40,170,"value is lesser than mean.");
outtextxy(40,210,"   Then each block is represented by a  two
level one bit quantizer");
outtextxy(40,235,"(0  or  1) based on whether  the  pixel  is
above or below the mean of ");
outtextxy(40,260,"the  block. But the  resultant  compressed
image contains each block's");
outtextxy(40,285,"information mean and lower mean.");
outtextxy(40,325,"    In the decompression process  the  mean
and lower mean are used to");
outtextxy(40,350,"calculate  the values of A and B which  are
used to replace the pixel");
outtextxy(40,375,"values  0 and 1.In this  method  compressed
image size is one fourth of");
outtextxy(40,400,"the original image.");
getch();
closegraph();
}

/* TEST1.C */

#include<graphics.h>
#include<dos.h>
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include "btc.c"
#include "mbtc.c"
```

```
#include "ambtc.c"
#include "lookup.c"
#include "aclookup.c"
#include "abouth.c"
#include "btch.c"
#include "mbtch.c"
#include "acch.c"
#include "ambtch.c"
#include "lookuph.c"
#include "aclookh.c"
#include "display.c"
#define POPUP_BORDER   GREEN
#define POPUP_COLOR    MAGENTA
#define TEXT_COLOR     WHITE
#define HIGH_TEXT_COLOR LIGHTRED

#define HIGH   1
#define LOW    0

#define LEFT          0x4b00
#define RIGHT         0x4d00
#define UP            0x4800
#define DOWN          0x5000
#define ENTER         0x000d
#define ESC           0x001b

                                                    mmenu[4][17]={
    char
 "COMPRESSION","DISPLAY","HELP","QUIT"};
    char popup[4][7][17]=
         {
            {"BTC","MBTC","AMBTC","ACC","LOOKUP","ACLOOKUP"},
            {""},
            {"ABOUT","BTC","MBTC","AMBTC","ACC","LOOKUP","ACL
            {""}};

    int count[4]={6,0,7,0};
  char mess[4][7][55]=
     {
      {
       "BLOCK TRUNCATION CODING",
       "MODIFIED BLOCK TRUNCATION CODING",
       "ABSOLUTE MOMENT BLOCK TRUNCATION CODING",
       "ADAPTIVE COMPRESSION CODING",
       "BLOCK TRUNCATION CODING USING LOOKUP TABLE",
       "ADAPTIVE COMPRESSION CODING USING LOOKUP TABLE"
      },
      {
       "DISPLAY THE IMAGE FILE"
       },
      {
       "ABOUT PROJECT",
       "ABOUT BLOCK TRUNCATION CODING",
      "ABOUT ADAPTIVE COMPRESSION CODING",
      "ABOUT BLOCK TRUNCATION CODING USING LOOKUP TABLE",
      "ABOUT ADAPTIVE COMPRESSION CODING USING LOOKUP TABLE"
     },
     {
      "QUIT TO DOS ENVIRONMENT"
     }
    };
    int pop_opt[4];
    int main_opt;



union REGS in_reg,out_reg;
int GetKey();
void Block(int,int,int,int,int,int);
void drawbar();
void drawpopup(int);
void highlight(int);
int trace();
void menuscreen();

int GetKey()
{
 in_reg.h.ah=8;
 int86(0x21,&in_reg,&out_reg);
 out_reg.h.ah=0;
 if(out_reg.h.al!=0) return(out_reg.x.ax);
 int86(0x21,&in_reg,&out_reg);
 out_reg.h.ah=out_reg.h.al;
 out_reg.h.al=0;
 return(out_reg.x.ax);
}

void Block(int left,int top,int len,int ht,int col,int bcol)
 {
   int rec[4][2];
   /*To set fill style & color*/
   if(bcol==17)
   bcol=col;
  setcolor(bcol);
  setfillstyle(SOLID_FILL,col);
 /* To set up rectangle coordinates*/
  rec[0][0]=rec[3][0]=left;
  rec[0][1]=rec[1][1]=top;
  rec[1][0]=rec[2][0]=left+len;
  rec[2][1]=rec[3][1]=top+ht;
  fillpoly(4,&(rec[0][0]));
 }
void drawbar()
{
 int i;
 settextstyle(1,HORIZ_DIR,1);
 for(i=0;i<4;i++)
   outtextxy(i*140+20,13,mmenu[i]);      /*displays the  mmenu
elements*/
}
void drawpopup(int pop_num)
{
 int i,x,y;
 if((pop_num==0)||(pop_num==2))
{
 y=44;
 x=pop_num*120+25;
 setcolor(POPUP_BORDER);
 /*setlinestyle(1,1,2);*/
 rectangle(x,y,x+120,y+count[pop_num]*12+5);
 setfillstyle(SOLID_FILL,POPUP_COLOR);
 floodfill(x+1,y+1,POPUP_BORDER);
 settextstyle(0,HORIZ_DIR,1);
 setcolor(TEXT_COLOR);
 for(i=0;i<count[pop_num];i++)
  outtextxy(x+4,y+i*11+3,popup[pop_num][i]);
  }
}
void highlight(int high)
{
 int col;
 col=high?HIGH_TEXT_COLOR:TEXT_COLOR;
 setcolor(col);
 settextstyle(DEFAULT_FONT,HORIZ_DIR,1);    /*MENU   ELEMENTS&
HIGHLIGHTED ONE*/
 if(count[main_opt])
 outtextxy(main_opt*120+29,44+pop_opt[main_opt]*11+3,
            popup[main_opt][pop_opt[main_opt]]);
 settextstyle(1,HORIZ_DIR,1);
 outtextxy(main_opt*140+20,13,mmenu[main_opt]);
 Block(0,441,640,40,BLUE,BLUE);
 setcolor(WHITE);
 settextstyle(TRIPLEX_FONT,HORIZ_DIR,1);  /*MESSAGE*/
 outtextxy(50,443,mess[main_opt][pop_opt[main_opt]]);
}
int trace()
{
 while(1)
 {
  switch(GetKey())
  {
  case RIGHT:
          highlight(LOW);
          main_opt++;
          main_opt%=4;
          Block(0,43,640,120,BLACK,BLACK);
          if(count[main_opt]) drawpopup(main_opt);
          highlight(HIGH);
```

```
            highlight(LOW);
            if(!main_opt)
                 main_opt=3;
            else
               main_opt--;
            Block(0,43,640,120,BLACK,BLACK);
            if(count[main_opt]) drawpopup(main_opt);
            highlight(HIGH);
            break;
  case DOWN:if(!count[main_opt]) break;
            highlight(LOW);
            pop_opt[main_opt]++;
            pop_opt[main_opt]%=count[main_opt];
            highlight(HIGH);
            break;
  case UP   :if(!count[main_opt]) break;
            highlight(LOW);
            if(pop_opt[main_opt])
                pop_opt[main_opt]--;
            else
               pop_opt[main_opt]=count[main_opt]-1;
            highlight(HIGH);
            break;
  case ENTER:return(main_opt*100+pop_opt[main_opt]+1);
  case ESC : closegraph();exit(0);return(0);
  default: continue;
  }
 }
}

void menuscreen()
{
      int gd=DETECT,gm=DETECT;
      initgraph(&gd,&gm,"v:\\software\\tc\\bgi");
      setbkcolor(YELLOW);
      drawbar();
      drawpopup(main_opt);
      highlight(HIGH);
}

void main()
{
 int i,opt;
 char s[5];
 main_opt=0;

 for(i=0;i<4;i++)
           pop_opt[i]=0;
 menuscreen();

 while(1)
 {
  opt=trace();
```

```
    case 1:closegraph();
           btc1();
           menuscreen();
           break;
    case 2:closegraph();
           mbtc1();
           menuscreen();
           break;
    case 3:closegraph();
           ambtc1();
           menuscreen();
           break;
    case 4:closegraph();
           acc1();
           menuscreen();
           break;                 /* 1 to 6 are  the  compression
methods*/
    case 5:closegraph();
           lookup1();
           menuscreen();
           break;

    case 6:closegraph();
           aclook1();
           menuscreen();
           break;
    case 101:closegraph();
             display1();            /* 101 is the display*/
             menuscreen();
             break;
    case 201:closegraph();
             abouth1();
           menuscreen();
           break;
    case 202:closegraph();
             btch1();
           menuscreen();
           break;
    case 203:closegraph();
             mbtch1();
           menuscreen();
           break;
    case 204:closegraph();
             ambtch1();
           menuscreen();
           break;
    case 205:closegraph();
           acch1();
           menuscreen();
           break;
    case 206:closegraph();
           lookuph1();
           menuscreen();
```

```
          aclookh1();
          menuscreen();
          break;
    case 301:closegraph();
            outtextxy(200,200,"thanks      for      using       BTC
techniques");
            exit(0);

   default: sprintf(s,"%d",opt);
            Block(200,200,40,40,LIGHTGRAY,LIGHTGRAY);
            setcolor(WHITE);
            outtextxy(200,200,s);
   }
  }
 }
```

REFERENCES

1. Rafeez C.Gonzalez and paul wintz, Digital image processing-Addision wesley publishing company.

2. P.Nasiopolos,rabab k.ward, and Daryl, J.Morse,"Adaptive compression coding" IEEE transactions on COMMUNICATIONS VOL 39,AUGUST 1991.

3. E.J.Delp and O.R.Mitchell,"Multilevel graphics representation using block Truncation coding", Proceedings IEEE Vol 68,July 1980.

4. E.J.Delp and O.R.Mitchell, "Image compression using block trunction coding, IEEE transactions on COMMUNICATIONS VOL 27, sept 1979.

6. O.R.Mitchell, E.J.Delp and S.G.Carlton "block truncation coding A new approach to image compression", Conference record, 1978 IEEE International conference on communications (ICC'78), VOL1,June 1978.

7. M.D.Lemma and O.R.Mitchell,"Absolute moment block truncation coding and its application to color images," IEEE transactions on COMMUNICATIONS VOL32,October 1984.

8. D.R.Halverson ,N.C.Griswold, and G.L.Wise, "A generalized block trunction coding algorithm for image compression" IEEE transactions on COMMUNICATIONS VOL20,June 1984.

9. A.N.Netravali and J.O.Limb, "Picture coding: A review". Proceedings of IEEE, VOL 68, March 1981.

10. V.Udpikar and J.P.raina, "A modified algorithm for block trunction coding of monochrome images," Electron.Lett., VOL 21, September1985.