Image Compression using Run Length and Variable Length Coding

A dissertation submitted in partial fulfillment for the award of the degree of

Master of Engineering

in

Computer Science

Submitted by
Rajesh Kumar

Department of Computer Science & Engineering
Thapar Institute of Engineering & Technology
(Deemed University)
PATIALA-147004.

CERTIFICATE

I hereby certify that the work presented in the thesis entitled "***Image Compression using Run Length and Variable Length Coding***" in fulfillment of the requirement for the award of the degree of the master of engineering in Computer Science is being submitted in the Department of Computer Science and Engineering. It is further certified that the work carried out for the thesis is an authentic record done under the supervision of Dr. P. K. Bansal.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other university.

(Rajesh Kumar)

This to certify that the above statement made by the candidate is correct to the best of our knowledge.

21.3.2009

(Dr. P. K. Bansal)

Professor
Department of Computer Science and Engineering
TIET, Patiala.

RK Sharma

Head
Department of Computer Science and Engineering
TIET, Patiala.

Dean
Academic Affairs
TIET, Patiala.

ACKNOWLEDGEMENT

I would like to thank Thapar Institute of Engineering and Technology, Patiala for providing me an opportunity to carry out the thesis on **"*Image Compression using Run Length and Variable Length Coding*"** as a part of ME (Computer Science) degree programme.

I consider it a great privilege to express my deep sense of gratitude to Dr. P. K. Bansal, for his invaluable guidance, constant inspiration and critical review throughout the course of this work. He has always been more than generous in sparing his very valuable time.

I am also thankful to all my friends for their time to time suggestions and cooperation.

There are times in such projects when the clock beats you time again and again and you just run out of energy and just want to finish it off once and for all. My sincere appreciation goes to my wife Reena for her cooperation and untiring patience in many odd hours during the thesis work. And finally thanks to my parents for their encouragement and moral support.

Rajesh Kumar

ABSTRACT

A revolution in information and entertainment has been widespread across the world. Information contents in digital form are pulsing through cables, telecommunication networks and direct satellites delivery system viewed on PC, stored on RAM, disk, tape, CD-ROM etc. The digital information contains text, audio and video. Among these, audio and video pose real challenges because both of these require real time operation and produce large data files when they are digitized. To overcome these problems either cables with high bandwidth can be used or data compression technique can be used. Compression of data is a better alternative.

A foregoing research and many implementations using software, hardware or both for, two algorithms, one for compression of data at source encoding and another for decompression at its destination decoding, has established that transformation coding achieves larger compression than predictive coding. In transformation coding, compression is achieved by transforming the given image into another array such that a large amount of information is packed into small number of samples. While predictive coding involves predicting subsequent values by observing previous ones and then transmitting only the small difference between actual and predicted data. Any distortion due to quantization and channel errors gets distributed during inverse transformations. Predictive coding is quite sensitive to changes in the statistics of the data and in two dimensions, finite order casual predictors may never achieve compression ability close to transform coding. Discrete Cosine Transformation is used in this work, as information capability is superior to other transformations. The choice of coding is situation dependent. In general a single coding is employed that may be variable length coding or run length coding. This gives a satisfactory compression ratio up to 50% but still is not very good. The present work employs the combination of both the coding techniques i.e. image is first coded as run length of levels and then run lengths are again coded using variable codes. The outcome of this work is a software system for gray scale image compression which is based on lossy compression. This technique has the advantage of providing better compression ratio i.e. 10 to 95 times over the other existing techniques which compress images up to 50 times. Further this technique is very flexible as it gives user the choice for selecting compression ratio and retrieved image quality.

CONTENTS

CHAPTER – 1

INTRODUCTION

INTRODUCTION

The term *digital image processing* generally refers to processing of a two dimensional picture by a digital computer. A digital image can be considered as finite array of real or complex numbers represented by a finite numbers of bits. Figure 1.1 shows the steps in a typical image processing sequence.

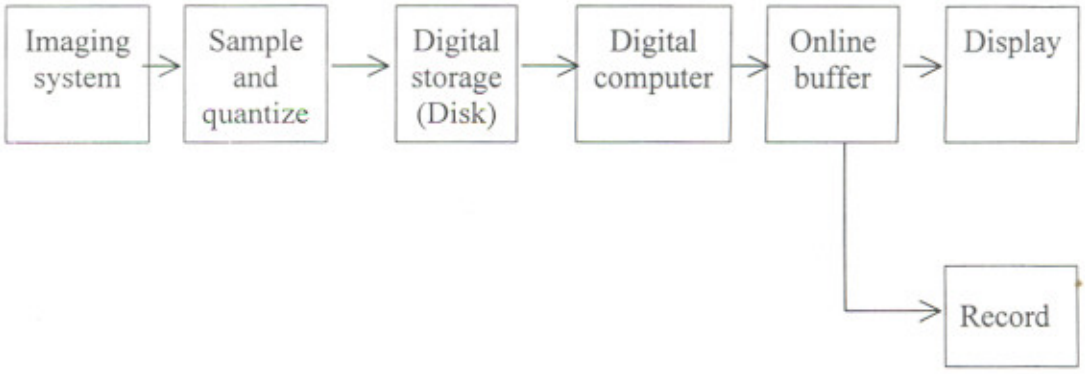

Fig 1.1 Image Processing System

1.1 ELEMENTS OF DIGITAL IMAGE PROCESSING SYSTEM

Major elements in digital image processing and their inter communications are shown in fig 1.2. The elements of a general purpose digital signal processing system should be capable of performing the image processing operations:

- Acquisition
- Storage
- Processing
- Communication
- Display

1.1.1 Image Acquisition

Two elements are required to acquire a digital image. The first is the device that is sensitive to a band in the electromagnetic energy spectrum such as x-ray, ultraviolet, visible or infrared bands. The second is a digitizer, which converts the electrical output of sensing device into digital form. Sensor can be a monochrome or color TV Camera that produces the entire image. The imaging sensor could also be a line scan camera that produces a single image line at a time. In this case object's motion past the line scanner produces a two-dimensional image. If the output of the imaging sensor is not already in digital form, an analog to digital converter digitizes it.

1.1.2 Storage

An 8-bit image of size 1024 X 1024 pixels requires one million bytes of storage. Thus providing adequate storage is a challenge in the design of image processing systems.

The VGA display is the current lowest common denominator for high quality color graphics. VGA displays 256 simultaneous colors. This lets the VGA display continuous tone images, such as color photographs, with a reasonable amount of quality. The problem with using images of photographic quality is the amount of storage required to use in a program. With VGA 256 color screen image has 200 rows of 320 pixels each consuming a single byte of storage. This means that a single screen image consumes a minimum of 64 k. It is not hard to imagine applications that would require hundreds of these images to be accessed. An on line catalog of a retail sales outlet for example could have easily images stored for immediate access. The problem is 1000 images of this quality would consume 64 MB of storage. And this is not an unreasonable number. Now games programs are being distributed on CD-ROMS due to the enormous amount of storage required by screen images [28].

1.1.3 Processing

Most image processing functions are implemented in Software. Only in image acquisition and display hardware processes are involved. The only reason for specialized image processing hardware is the need for speed in some applications. Although large-scale

image processing systems are still being sold for massive imaging applications, such as processing of Satellite images, the trend continues toward miniaturizing and merging general purpose small computers equipped with image processing hardware. The imaging

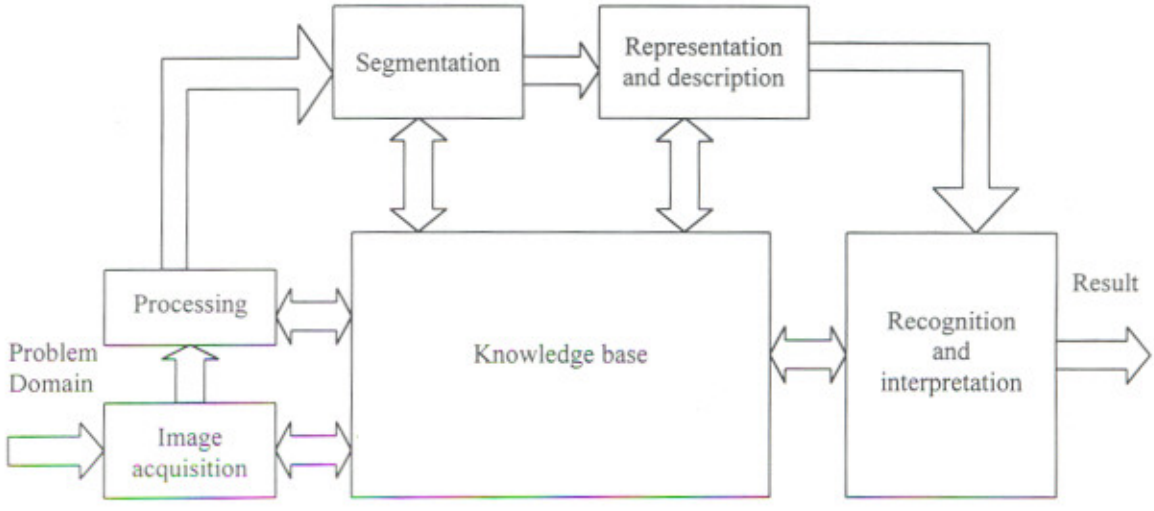

Fig 1.2 Functional units of Image Processing System

hardware being added to these computers consists of a digitizer and frame buffer combination for image digitization and temporary storage and so called mathematics co-processor for performing arithmetic and logic operations at frame rates.

1.1.4 Image Communication

Communication in digital image processing primarily involves local communication between image processing systems and remote communication from one point to another. Hardware and Software for local communication are available for most computers. Communication across vast distances presents a serious problem. As digital images contain a significant amount of data. A voice grade telephone line can transmit at a maximum rate of 9600 bits/sec. Thus to transmit 512 x 512, 8 bit image would require nearly four minutes. Wireless links such as satellites are much faster, but they cost considerably high.

The point is that transmission of entire images over long distances is far from trivial. Data compression and decompression techniques play a central role in addressing this problem.

1.1.5 Image Display

Monochrome and color TV monitors are the principal display devices used in modern image processing systems. Monitors are driven by outputs of a hardware image display module in the host computer or as part of the hardware associated with an image processor. The signals at the output of the display module can also be fed into an image recording device that produces a hard copy, slides, photographs or transparencies, of the image being viewed on the monitor screen. Other common means of recording an image directly on paper include laser printers, heat sensitive paper devices and ink-spray systems.

1.2 APPLICATIONS

In evolving scenario of Internet, digital image processing has a broad range of applications such as:

- Remote sensing via satellites and other spacecrafts.
- Image transmission and storage for business applications.
- Medical processing.
- Radar, sonar and acoustic image processing.
- Robotics and automated inspection of industrial parts.
- E-business.

Images acquired by satellites are useful in tracking of earth resources; geographical mapping, prediction of agriculture crops, urban growth, and weather; flood and fire control; and many other environmental applications. Space image applications include recognition and analysis of objects contained in images obtained from deep space probe missions.

Transmission and storage applications occur in broadcast television, teleconferencing, transmission of facsimile images (printed documents and graphics) for office automation, communication over computer networks, closed-circuit television based security monitoring systems, and in military communication systems. In medical applications one is concerned with processing of chest X-rays, cinengiograms, projection images of transaxial tomography, and other medical images that occur in radiology nuclear magnetic resonance (NMR), and ultrasonic scanning. These images may be used for patient screening and monitoring or for detection of tumors or other disease in patients. Radar and sonar images are used for detection and recognition of various types of targets or in guidance and maneuvering of missile or aircraft systems. There are many other applications ranging from robot vision for industrial automation to image synthesis for cartoon making or fashion design. An on-line catalog for a retail sales outlet can have images stored for immediate access, we can see few images of the product on web itself.

1.3 IMAGE PROCESSING PROBLEMS AND TECHNIQUES

Major topics of digital image processing are representation, processing techniques and communication. Image representation includes tasks ranging from acquisition, digitization and display to mathematical characterization of images for subsequent processing. Often a proper representation is a prerequisite to an efficient processing technique such as enhancement, restoration, analysis, reconstruction from projections and image communication. Although there are many image processing applications and problems, the basic classes of problems are:

- Image representation and modeling.
- Image enhancement
- Image restoration
- Image analysis
- Image reconstruction
- Image data compression

1.3.1 Image Representation and Modeling

In image representation one is concerned with characterization of the quality that each pixel represents. In general, any two dimensional function that bears information can be considered an image. Image models give a logical or quantitative description of the properties of this function. An important consideration in image representation in the fidelity or intelligibility criteria for measuring the quality of an image or the performance of a processing technique. The fundamental requirement of digital processing is that image must be sampled and quantized. Digitization of spatial coordinates (x, y) is called image sampling and sampling rate should be large enough to preserve the useful information in an image. Analog to digital conversion of a sampled image to a finite number of gray levels is called Image quantization.

1.3.2 Image Enhancement

In image enhancement, the goal is to accentuate certain image features for subsequent analysis or for image display. Examples include contrast and edge enhancement, pseudo coloring, noise filtering, shapening and magnifying. Image enhancement is useful in feature extraction, image analysis and visual information display. The enhancement process itself does not increase the inherent information content in the data. It simply emphasizes certain specified image characteristics. Enhancement algorithms are generally interactive and application dependent.

1.3.3 Image Restoration

Image restoration refers to removal or minimization of known degradations in an image. This includes deblurring of images degraded by the limitations of a sensor or its environment, noise filtering and correction of geometric distortion or non-linearities due to sensors.

1.3.4 Image Analysis

Image analysis is concerned with making quantitative measurements from an image to produce a description of it. In the simplest form, this task could be reading a lable on a grocery item, sorting different parts on an assembly line, or measuring the size and

orientation of blood cells in a medical image. More advance image analysis systems measure quantitative information and use it to make a sophisticated decision, such as controlling the arm of a robot move an object after identifying it or navigating an aircraft the aid of images acquired along its trajectory. Image analysis techniques require extraction of certain features that aid in the identification of the object. Quantitative measurements of object features allow classification and description of the image.

1.3.5 Image Reconstruction From Projections

Image reconstruction from projections is a special class of image restoration problems where a two or higher dimensional object is reconstructed from several one dimension projections. Reconstruction algorithms drive an image of a thin axial slice of the object, giving an inside view otherwise unobtainable without performing extensive surgery. Such techniques are important in medical imaging (CT scanners), astronomy, radar imaging, geological exploration, and non-destruction testing of assemblies.

1.3.6 Image Data Compression

An enormous amount of data is produced when a 2-D light intensity function is sampled and quantized to create a digital image. In fact the amount of data generated may be so great that it results in impractical storage, processing and communications requirements. In such cases, representation beyond the simple 2-D sampling and gray level quantization are needed. The term *data compression* refers to the process of reducing the amount of data required to represent a given quantity of information. A clear distinction must be made between data and information. In fact, data are the means by which information is conveyed. Various amounts of data may be used to represent the same amount of information. Some of these methods of representing the information include non-essential data. That is, they contain data that either provide no relevant information or simply restate that which is already known. It is thus said to contain data redundancy. Image data compression techniques are concerned with reduction of the number of bits required to store or transmit images without any appreciable loss of information.

1.4 Problem Formulation

A revolution in information and entertainment has been widespread across the world. Information contents in digital form are pulsing through cables, telecommunication networks and direct satellites delivery system viewed on PC, stored on RAM, disk, tape, CD-ROM etc. The digital data following around these networks contains text, audio and video. Among these, audio and video are real challenges. This is because both of these produce large data files when they are digitized. To overcome this we can either use the cables with high bandwidth or to compress the data. Compression of data would be good option as it reduces the cable cost. Smooth motion video requires 25 frames per second. The resolution factor plays an important role. The common configurations are 640x480(VGA), 800x600(SVGA) and 1024x786 (XGA). An XGA display with 24 bits per pixel i.e. 16 million color pattern and 25 frames per second need to be fed at 472 Mbps. Clearly compression is essential for images in digital multimedia applications. A megabyte of space will be filled by roughly 6 seconds of video or by a single 640x480 resolution color image stored using 24 bits per pixel. Digital images contain large amount of data, which serve as strong motivation for seeking methods capable of achieving data reduction. It is clear that transmitting material should be compressed. All compression systems require two algorithms: one for compression of data at source end and another for decompression at destination end.

There are basically two types of compression techniques Lossless compression and Lossy compression. In lossless compression technique the original representation can be perfectly recovered and the input patterns and output patterns are identical. Lossy compression technique involves encoding into a form that takes up relatively small amount of space but which can be decoded to obtain a representation that human being finds similar to the original. Different approaches involved in these are Prediction, Frequency oriented compression and Importance oriented compression. In frequency oriented compression transformation is another approach. An especially important two dimension spatial transform is Discrete Cosine Transform (DCT), which is related to fast Fourier transforms. This transformation has the property that for images without sharp discontinuities most of the spectral power is in the first term allowing later ones to be ignored without much information loss.

The transformation used for mapping in this thesis is Discrete Cosine Transformation (DCT). DCT is chosen, as DCT is the standard in the present days and is superior to others such as Karhunen-Loeve (KLT), Walsh-Harmard (WHT) and Discrete Fornier (DFT) transformation due to following reasons:

- Its basis images are fixed, thus it is input independent.
- It provides good compromise between information packing and computational complexity.
- DCT minimizes the block like appearance, called blocking artifact.

1.5 Organisation of Thesis

This thesis contains five chapters, which are organized as follows:

The first chapter covers introduction and background of the subject.

Digital images contain large amount of data, which serve as strong motivation for seeking methods capable of achieving data reduction. Different types of data redundancies are discussed in chapter two.

General compression model is considered in chapter three.

Chapter four extensively treats image compression using DCT model.

Finally conclusions and future scope of work is discussed in chapter five.

1.6 CONCLUSION

Multimedia revolution has become possible due to digital signal processing. Digital Image Processing is a rapidly evolving field with growing applications in science and engineering. Image processing holds the possibility of developing the ultimate machine that could perform the visual functions of all living beings. In this chapter the background work for image compression has been covered. Problem formulation and organization of the thesis are also discussed. Many theoretical as well as technological breakthroughs in processing techniques and problems have been considered. A few applications viz. remote sensing, image storage for business applications, Medical processing, Robotics and automated inspection etc. are identified. Next chapter focuses on the important issue of data redundancy in image compression.

CHAPTER – 2

DATA REDUNDANCY AND INFORMATION THEORY

INTRODUCTION

Various amounts of data, the means by which information is conveyed, may be used to represent the same amount of information. The nonessential i.e. provide no relevant information or simply restate that is already known is called redundant data.

Data redundancy, a central issue in digital image compression is not an abstract concept but a mathematically quantifiable quantity. If n_1 and n_2 denote the number of information carrying units in two data sets that represent the same information, the *relative data redundancy* R_d of the first data set can be defined as:

$$R_d = 1 - \frac{1}{C_r}$$

Where C_r commonly called the *compression ratio*, is

$$C_r = \frac{n_1}{n_2}$$

For the case $n_2 = n_1$, $C_r = 1$ and $R_d = 0$ indicating that (relative to the second data set) the first representation of the information contains no redundant data.

When $n_2 << n_1$, $C_r \rightarrow \infty$ and $R_d \rightarrow 1$, implying significant compression and highly redundant data. In the final case, $n_2 >> n_1$, $C_r \rightarrow 0$, and $R_d \rightarrow -\infty$, indicating that the second data set contains much more data than the original representation. This of course, is the normally undesirable case of data expansion. In general C_r and R_d lie in the open intervals $(0, \infty)$ and $(-\infty, 1)$ respectively. A practical compression ratio, such as 10 means that the first data set has the 10 information carrying units say bits for every 1 unit in the second or compressed data set and the corresponding redundancy of 0.9 implies that 90 percent of data in the first data set is redundant [1].

2.1 VARIOUS TYPES OF DATA REDUNDANCIES

In digital image compression, three basic data redundancies can be identified and exploited:

- Coding redundancy
- Interpixel redundancy
- Psychovisual redundancy

Data compression is achieved when one or more of these redundancies are reduced or eliminated [6,8].

2.1.1 Coding Redundancy

Let us assume that a discrete random variable R_k in the interval (0,1) represents the gray levels of an image and that each R_k occurs with probability $p(R_k)$. That is

$$p(R_k) = \frac{N_k}{n} \qquad k = 0,1,2,3.....,L-1$$

Where L is the number of gray levels, N_k is the number of times that the kth gray level appears in the image, and n is the total number of pixels in the image. If the number of bits used to represent each value of R_k is $\ell(R_k)$ the average number of bits required to represent each pixel is

$$L^{avg} = \sum_{k=0}^{L-1} \ell(R_k) \times p(R_k)$$

Thus the total number of bits required to code an M X N image is M * N * L_{avg}. Representing the gray levels of an image with a natural m-bit binary code then $L_{avg} = m$. If instead of natural coding image is represented by assigning fewer bits to more probable gray levels than to the less probable ones than $L_{avg} < m$. This process commonly is referred to as variable length coding. If the gray levels of an image are coded in a way that uses

more code symbols than absolutely necessary to present each gray level, the resulting image is said to contain coding redundancy.

2.1.2 Interpixel Redundancy

Another important form of data redundancy is interpixel redundancy, which is directly related to the interpixel correlations within an image. Because the value of any given pixel can be reasonable predicted from the value of its neighbours, the information carried by individual pixels is relatively small. Much of the visual contribution of a single pixel to an image is redundant; it could have been guessed on the basis of its neighbour's values. A variety of names, including spatial redundancy, geometric redundancy, and interframe redundancy, have been coined to refer to these interpixel dependencies. In order to reduce the interpixel redundancies in an image, the 2-D pixel array normally used for human viewing and interpretation must be transformed into a more efficient but usually non visual format. For example, the differences between adjacent pixels can be used to represent an image. Transformations of this type are referred to as mappings. They are called reversible if the original image elements can be reconstructed from the transformed data set.

2.1.3 Psychovisual Redundancy

Human perception of the information in an image normally does not involve quantitative analysis of every pixel or luminance value in the image. In general, an observer searches for distinguishing features such as edges or textural regions and mentally combines them into recognisable groupings. The brain then correlates these groupings with prior knowledge in order to complete the image interpretation process. Thus eye does not respond with equal sensitivity to all visual information. Certain information simply has less relative importance than other information in normal visual processing. This information is said to be *psychovisually redundant.* It can be eliminated without significantly impairing the quality of image perception. Psychovisual redundancy is fundamentally different from the coding redundancy and interpixel redundancy. Unlike coding redundancy and interpixel redundancy, psychovisual redundancy is associated with real or quantifiable visual information. Its elimination is possible only because the information itself is not essential for normal visual processing. Since the elimination of psychovisual redundant data results in a loss of quantitative information. Thus it is an irreversible process.

2.2 INFORMATION THEORY

The term image refers to a two-dimensional light intensity function, denoted by f(x,y) where the value or amplitude of 'f ' at spatial coordinates (x,y) gives the intensity or brightness of the image at that point. The images people perceive in everyday visual activities normally consist of light reflected from objects. The basic nature of f(x,y) may be characterized by the two components :

1. The amount of source light incident on the scene being viewed and

2. The amount of light reflected by the objects in the scene.

They are called the illumination and reflectance components and are denoted by i(x,y) and r(x,y) respectively.

$$f(x,y) = i(x,y) * r(x,y)$$

We call the intensity of a monochrome image f at coordinates (x,y) the gray level (l) of the image at that point [1,4].

To be suitable for computer processing an image function f(x,y) must be digitized both spatially and in amplitude. Digitization of the spatial coordinates (x,y) is called image *sampling* and amplitude digitization is called gray level *quantization*. A continuous image f(x,y) is approximated by equally spaced samples arranged in the form of an N X M array. Where each element of the array is a discrete quantity.

$$f(x,y) = \begin{pmatrix} f(0,0) & f(0,1) & \cdots\cdots & f(0,M-1) \\ f(1,0) & f(1,1) & \cdots\cdots & f(1,M-1) \\ . & & & \\ . & & & \\ . & & & \\ f(N-1,0) & f(N-1,1) & & f(N-1,M-1) \end{pmatrix}$$

The right hand side is commonly called a digital image. Each element of the array is referred to as an image element or pixel.

Digitization process requires decisions about values for N, M and the number of discrete gray levels allowed for each pixel. Common practice in digital image processing is to let these quantities be integer powers of two. If m is bits required for G number of gray levels.

Then

$$b = M \times N \times m$$

If M = N

$$b = N^2 m$$

b bits required to store digitized image. For example a 128 X 128 image with 64 gray levels requires 98,304 bits of storage. This equation clearly points out the unfortunate fact that storage and consequently processing requirements increase rapidly as a function of N, M and m. Fig. 2.1 & Fig. 2.2 shows a general impression of how a digital image degrades as its spatial resolution and gray level quantiziation are decreased. Fig. 2.1(a) shows a 1024 X 1024, 256 gray level digital image of a rose. Figs. 2.1(b) - 2.1(f) shows the results of reducing the spatial resolution from N = 1024 to N = 512, 256, 128, 64 and 32 respectively. In all cases the maximum gray levels allowed are 256.

Fig. 2.2 illustrates the effects produced by decreasing the number of bits represent the number of gray levels in an image. Fig. 2.2(a) is 1024 X 1024, with 8 bits for gray levels. Figs. 2.2 (b) - 2.2 (h) were obtained by reducing the gray levels to 128, 64, 32, 16, 8, 4 & 2 levels [11,12].

2.3 MEASUREMENT OF INFORMATION

When we consider various methods to reduce the amount of data used to represent an image, we need to know how few data actually are needed to represent the image and minimum amount of data that is sufficient to describe completely the image without loss of information. Information theory provides the mathematical framework to measure the information content in an image [18].

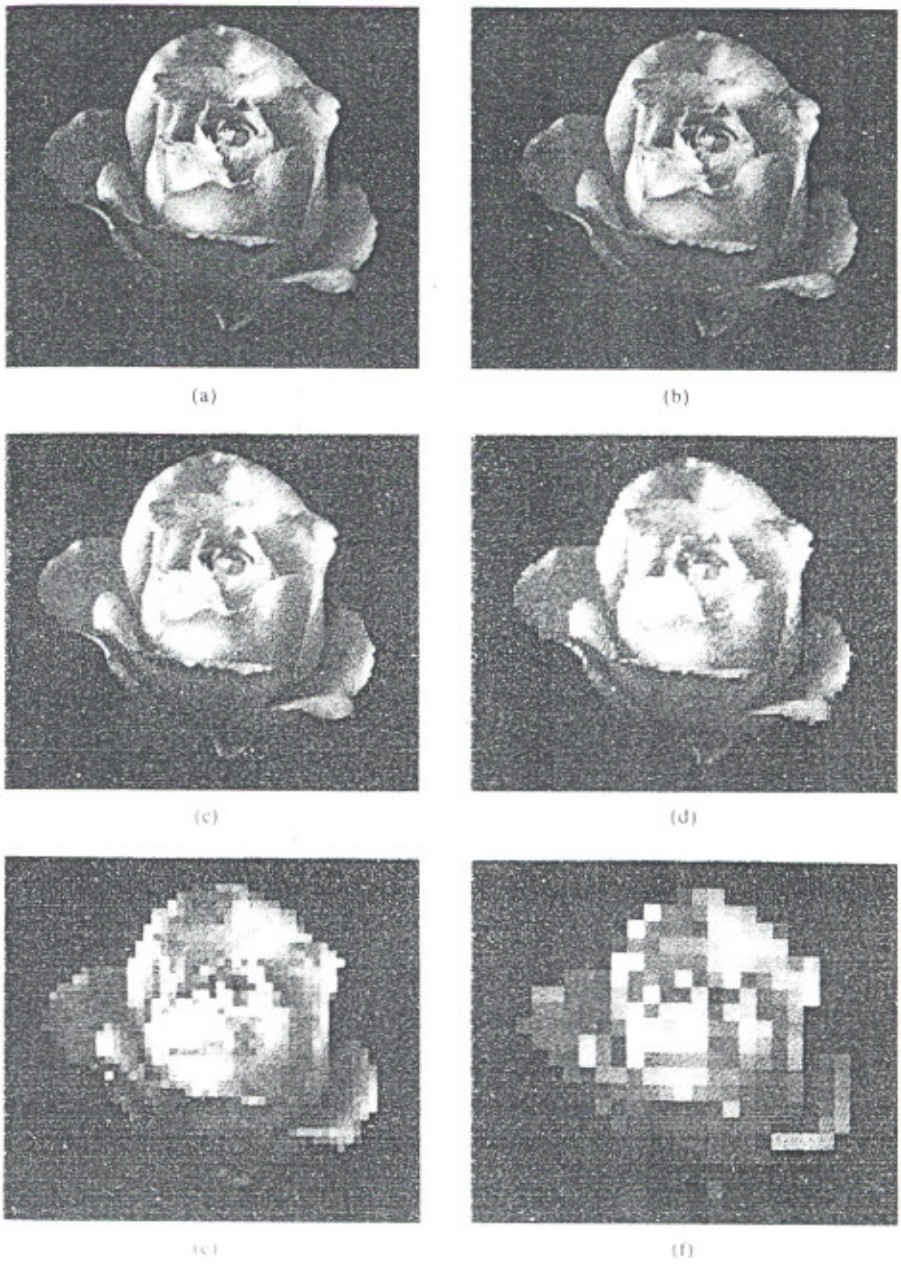

Fig 2.1 Effects of reducing spatial resolution.

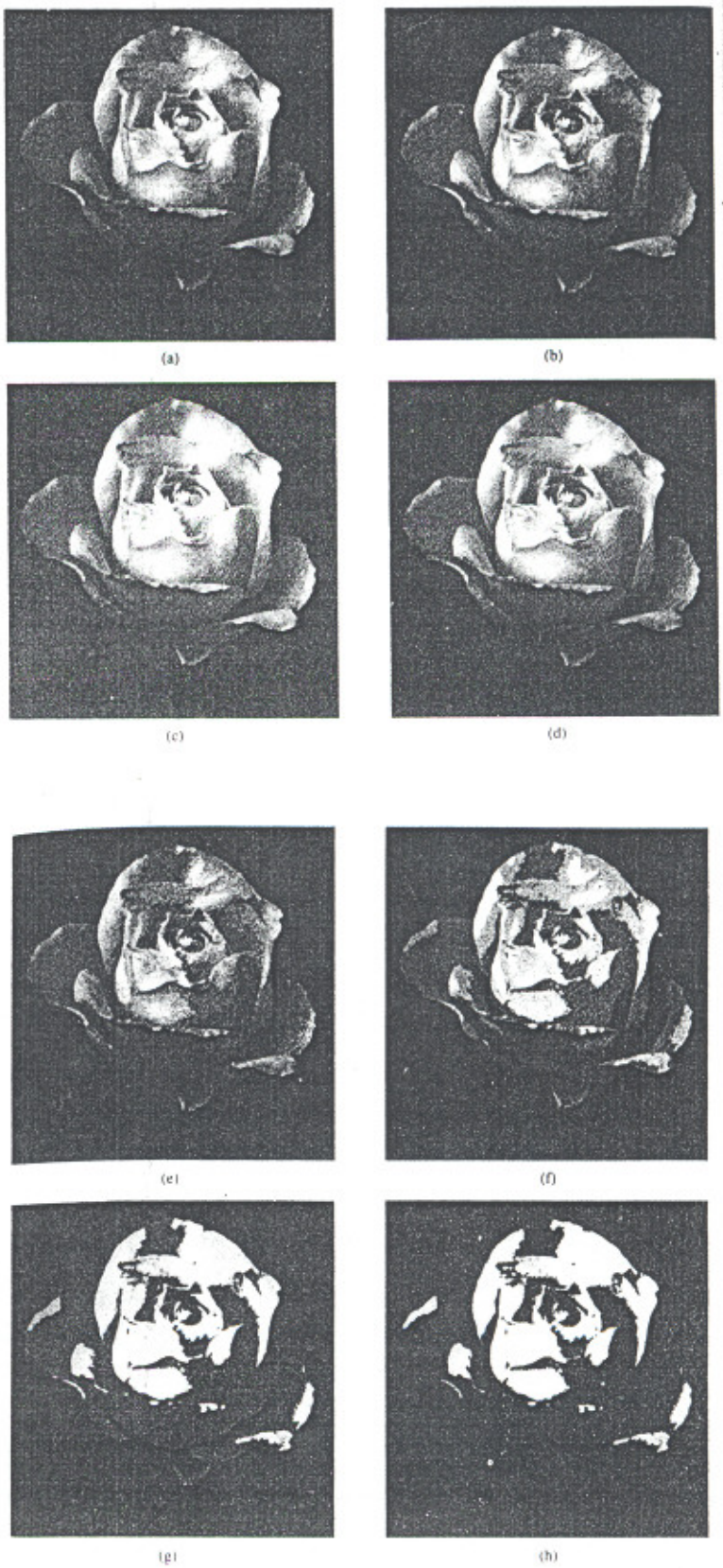

Fig 2.2 Effects of reducing gray levels of image.

2.3.1 Information

The fundamental premise of information theory is that the generation of information can be modeled as a probabilistic process that can be measured in a manner that agrees with intution. In accordance with this supposition, a random event (gray-level) that occurs with probability P (E) is said to contain

$$I(E) = \log\left(\frac{1}{P(E)}\right) = -\log P(E)$$

units of information. The base of logarithm in above equation determines the unit used to measure information. If the base r logarithm is used, the measurement is said to be in r-ary units. If the base 2 is selected, the resulting unit of information is called a *bit*.

2.3.2 Entropy of the Source

Lets assume that the information source generates a random sequence of symbols from a finite or countable infinite set of possible symbols.

The set of source symbols (a_1, a_2, a_3 ..... a_j) is referred to as the source alphabet. A, and the elements of the set, are called symbols or letters. The probability of the event that the source will produce symbol a_j is $P(a_j)$ and thus

$$\sum_{j=1}^{J} P(a_j) = 1$$

The J x 1 vector z = [$P(a_1)$, $P(a_2)$, ..., $P(a_j)$] t customarily represents the set of all source symbol probabilities {$P(a_1)$, $P(a_2)$, ..., $P(a_j)$}. The finite ensemble (A,z) describes the information source completely. If k source symbols are generated, the law of large number stipulates that, for a sufficiently large value of k, symbol a_j will on average be output $kP(a_j)$ times. Thus the average self information obtained from k outputs is

$$-kP(a_1)\log P(a_1) - kP(a_2)\log P(a_2) - \ldots\ldots\ldots\ldots - kP(a_j)\log P(a_j)$$

or

$$-k\sum_{j=1}^{J} P(a_j)\ \log P(a_j)$$

The average information per source output, denoted H(z), is

$$H(z) = -\sum_{j=1}^{J} P(a_j)\ \log P(a_j)$$

and is called the *uncertainty* or *entropy* of the source. It defines the average amount of information (in r-ary units per symbol) obtained by observing a single source output. Thus H(z) is the theoretical lower bound on storage required to represent the complete information.

2.4 CONCLUSION

The underlying basis of compression process is the removal of redundant data. From the mathematical viewpoint, this amounts to transforming a 2-D pixel array into a statistically uncorrelated data set. The transformation is applied prior to storage or transmission of the image. At some later time, the compressed image is decompressed to reconstruct the original image or an approximation to it. The foundation of digital image compression and the most common redundancies that form the core of the existing technology has been described in this chapter. Further measurement of the minimum amount of data actually needed to represent the image has also been discussed. The fundamental premise of information theory is that the generation of information can be modeled as a probabilistic process. If the probability of symbols in a message were known, there ought to be a way to code the symbols so that the message would take up less space. Codes for symbols with low probabilities have more bits and codes for symbols with high probabilities have less bits. The ideas of entropy, information contents and redundancies are also explored in this chapter. General compression model and different coding schemes are considered in the next chapter.

CHAPTER – 3

GENERAL COMPRESSION MODEL

INTRODUCTION

Three general techniques for reducing the amount of data required to represent an image are discussed in previous chapter. However, these techniques are combined to form a practical image compression system.

Generally a compression system consists of two distinct structural blocks: an encoder and a decoder. An input image f(x,y) is fed into the encoder, which create a set of symbols from the input data. After transmission over the channel, the encoded representation is fed to the decoder, where a restructured output image f'(x,y) is generated. In general f' (x,y) may or may not be an exact replica of f(x,y).

3.1 A GENERAL COMPRESSION MODEL

A general compression model as shown in fig 3.1, shows that encoder and decoder consist of two relatively independent functions or sub blocks. The encoder is made up of source encoder, which removes input redundancies, and a channel encoder, which increases the noise immunity of the source encoder's output. Similarly, the decoder includes a channel decoder followed by a source decoder. If the channel between the encoder and decoder is noise free, the channel encoder and decoder are omitted, and the general encoder and decoder become the source encoder and decoder, respectively [4].

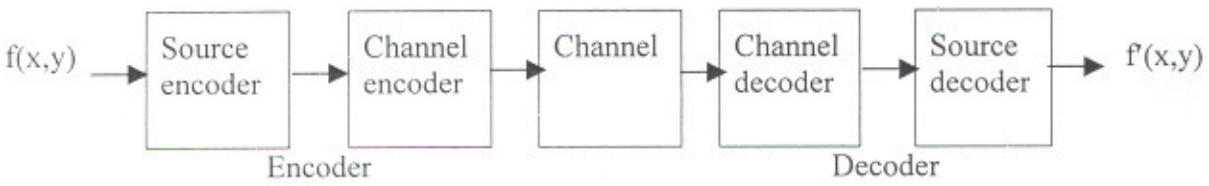

Fig 3.1 General Compression Model

3.1.1 The Source Encoder

The source encoder is responsible for reducing or eliminating any coding, interpixel, or psychovisiual redundancies in the input image. The specific application dictates the best encoding approach. Normally, the approach can be modeled by a series of three independent operations. Operation is designed to reduce one of the three redundancies discussed earlier.

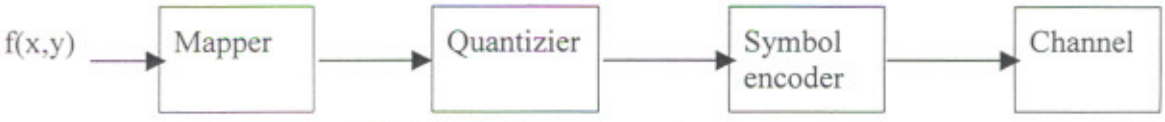

Fig 3.2(a) Source encoder

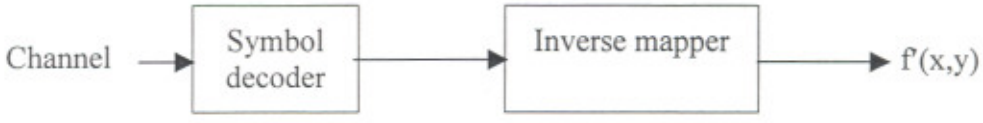

Fig 3.2(b) Source decoder

3.1.2 Mapper

In the first stage of the source encoding processes, the mapper transforms the input data into a (usually nonvisual) format designed to reduce interpixel redundancies in the input image. This operation generally is reversible and may or may not reduce directly the amount of data required to represent the image.

3.1.3 Quantizer

The second stage, or quantizer block reduces the accuracy of the mapper's output in accordance with some pre-established fidelity criterion. This stage reduces the psychovisual redundancies of the input image. This operation is irreversible. Thus, it must be omitted when error-free compression is desired.

3.1.4 Symbol Encoder

In the third and final stage of source encoding processes, the symbol coder creates a fixed or variable-length code to represent the quantizer output and maps the output in accordance with the code. The term symbol coder distinguishes this coding operation from the overall source encoding processes. In most cases, a variable length code is used to represent the mapped and quantized data set. It assigns the shortest code words to the most, frequently occurring output values and thus reduces coding redundancy. The operation is completely reversible. Upon completion of symbol coding step, the input image has been processed to remove each of the three redundancies discussed earlier. Here we have shown the source encoding process as three successive operations, but all three operations are not necessarily included in every compression. For example, the quantizer must be omitted when error-free compression is desired. In addition, some compression techniques normally are modeled by merging blocks that are physically separate in figure 3.2(a).

3.1.5 Source Decoder

The source decoder shown in figure contains only two components: a symbol decoder and an inverse mapper. These blocks perform, in reverse order, the inverse operations of the source encoder's symbol encoder and mapper blocks. Because quantization results in irreversible information loss, an inverse quantizer block is not included in the general source decoder model shown in the figure 3.2(b).

3.2 SOURCE CODING

A conversion of the output of a discrete memory less source into a sequence of binary symbols (binary code word) called source coding. The device, which performs this operation, is called source encoder. An objective of source coding is to minimize the average bit rate required for representation of the source by reducing the redundancy of information source [16,17,25].

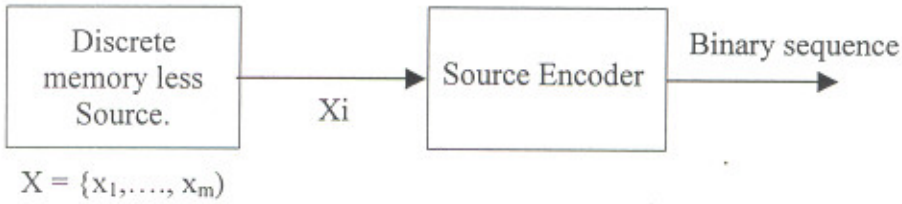

Fig 3.3 SOURCE CODING

3.2.1 Code length and Code Efficiency

As shown in fig 3.3, let X be a Discrete Memory less Source with finite entropy H(X) and an alphabet $\{x_1, ..., x_m\}$ with corresponding probabilities of occurrence $p(x_i\}$ (I = 1,….m). Let the binary code word assigned to symbol x_i by the encoder have length n_i measured in bits. The length of a code word is the number of binary digits in the code word. The average code word length L per source symbol is

$$L = \sum p(x_i)\, n_i$$

The parameter L represents the average number of bits per symbol used in source coding process.

The code efficiency η is defined as

$$\eta = L_{min}./L$$

Where L_{min} is the minimum possible value of L. When approaches unity, the code is said to be efficient.

The code redundancy γ is defined as

$$\gamma = 1 - \eta$$

Source Coding Theorem

The source coding theorem states that for a Discrete memory less source with entropy H(X), the average code word length L per symbol is bounded as

$$L > H(X)$$

And further L can be as close to H(X) as desired for suitably chosen code.

Thus with $L_{min} = H(X)$ the code efficiency can be rewritten as

$$\eta = H(X) / L$$

3.2.2 Classification of Codes

Classification of codes is best illustrated by an example. Consider table where a source of size 4 has been in binary codes with symbols 0 & 1 [20].

BINARY CODES

x_I	Code 1	Code 2	Code 3	Code 4	Code 5	Code 6
X1	00	00	0	0	0	1
X2	01	01	1	10	01	01
X3	00	10	00	110	011	001
X4	11	11	11	111	0111	0001

1. Fixed Length Codes

A fixed length code is one whose code word length is fixed. Code 1 and Code 2 of table are fixed length code with length 2.

2. Variable Length Codes

A variable length code is one whose code word is not fixed. All codes of the table except 1 & 2 are variable length codes [21].

3. Distinct Codes

A code of distinct if each code word is distinguishable from other code words. All codes of table except code 1 are distinct codes, notice that for x_1 and x_3.

4. Prefix-Free Codes

A code in which no code word can be formed by adding code symbols to another code word is called a prefix-free code. Thus, in a prefix free code no code word is a prefix of another. Codes 2,4 and 6 of table are prefix free codes.

In general the prefix condition requires that for a given code word C_k of length k having elements (b_1, b_2,..... b_k) there is no other code word of length $1 < k$ with elements (b_1, b_2,..... b_i) for $1 <= 1 <= k-1$. In other words there is no code word of length $1 < k$ with elements (b_1, b_2, ... b_1) for $1 <= 1 <= k-1$. In other words there is no code word of length $1 < k$ that is identical to first 1 binary digits of another code word of the length $k < 1$. This property makes the code words instantaneously decodable.

5. Uniquely Decodable Codes

A distinct code is uniquely decodable if the original source sequence can be reconstructed perfectly from the binary sequence. Note that code 3 of the table is not a uniquely decodable code. For example, the binary sequence 1001 may correspond to the source sequences $X_2X_3X_1$ or $X_2X_1X_1X_2$. A sufficient condition to ensure that a code is uniquely decodable is that no code word is prefix of other. Thus, the prefix free codes 2,4 & 6 are uniquely decodable codes. Note that the prefix free condition is not a necessary condition for unique decodability. For example code 5 of the table does not satisfy the prefix free condition, and yet it is uniquely decodable since the bit 0 indicates the beginning of each code word of the code [31].

6. Instantaneous Codes

A uniquely decodable code is called an instantaneous code if the end of any code word is recognisable without examining subsequent code symbols. The instantaneous codes have the property previously mentioned that no code word is a prefix of another code word. For this reason, prefix free codes are sometimes called instantaneous codes.

7. Optimal Codes

A code is said to be optimal if it is instantaneous and has minimum average length L for a given source with a given probability assignment for the source symbols.

8. Entropy Coding

The design of a variable length code such that its average length approaches the entropy of the discrete memory less source is often referred to as entropy coding. In this section two examples of entropy encoding are presented.

1. Shannon Fano Coding

2. Huffman Encoding

Huffman coding results in an optimum code. This algorithm is optimum in the sense that average number of binary digits required to represent the source symbols is a minimum subject to the constraint that the code words satisfy the prefix condition which allows the received sequence to be uniquely & instantaneously decodable. This is the code that has the highest efficiency.

3.3 ERROR-FREE COMPRESSION

If the image retrieved by decompressing the compressed image is the exact as the original image, the compression is said to be error-free. In numerous applications error-free compression is the only acceptable means of data reduction. One such application is the archival of medical or business documents, where lossy compression usually is prohibited

for legal reasons. In error-free compressions, quantization process is omitted, because it is the only irreversible process in image data compression model. Error-free compression techniques usually provide compression ratio of 2 to 10. Thus where image quality is more important than storage saving, Error-free compression is applied.

3.4 LOSSY COMPRESSION

Unlike the error-free approaches, lossy encoding is based on the concept of compromising the accuracy of the reconstructed image in exchange for increased compression. If the resulting distortion (which may or may not be visually apparent) can be tolerated, the increase in compression can be significant. In fact, many lossy encoding techniques are capable to reproducing recognisable monochrome images from data that have been compressed by more than 30 : 1 and images that are virtually distinguishable from the originals at 10 : 1 and 20 : 1. The main part of the lossy image compression models is the quantizer, where actual compression of image data takes place[26].

3.5 CONCLUSION

Various general compression model blocks like source encoder, mapper, quantizer and decoder has been covered in this chapter. Two types of compression models i.e. Lossy and lossless have also been discussed. Lossy compression accepts a slight loss of data to facilitate compression which makes two passes; first pass over the data performs a high level signal processing function which transforms the data into the frequency domain. Transformed data in second pass is *smoothed,* rounding off high and low points contributing to loss of signal. The more the data is massaged, the greater the signal loss and more compression will occur. Lossless compression is applied where image quality is more important than storage saving. In the next chapter, Image compression by implementing lossy DCT model has been discussed.

CHAPTER – 4

DCT COMPRESSION MODEL

INTRODUCTION

In transform coding a reversible, linear transform is used to map the image into a set of transformed co-efficients, which are then quantized and coded. Various transformation techniques are Karhunen-Loeve (KLT), Discrete Cosine (DCT), Walsh-Hadamard (WHT) etc. In this chapter Discrete Cosine Transformations have have been chosen among the various other transformations have been chosen as its information packing ability is superior to others and it provides a good compromise between information packing and computational complexity[27].

4.1 DCT ENCODER AND DECODER

As shown in fig 4.1(a), encoder performs four relatively straightforward operations i.e. Subimage decomposition, Transformation, Quantization and Coding. The decoder implements the inverse sequence of steps with the exception of the quantization function of the encoder shown in fig 4.1(b).

An N x N input image is first subdivided into subimages of size n x n, which are then transformed to generate $(N / n)^2$, n x n subimage transform arrays. The goal of transformation process is to decorrelate the pixels of each subimage, or to pack as much information as possible into smallest number of transform co-efficients. The quantization stage selectively eliminates or more coarsely quantizes the co-efficients that carry the least information. These co-efficients have the smallest impact on reconstructed subimage quality. The encoding process terminates by coding the quantized co-efficients.

Any or all of the transform coding steps can be adapted in local image content, called adaptive transform coding, or fixed for all subimages, called nonadptive transform coding. In the present implementation nonadaptive transform coding have been chosen.

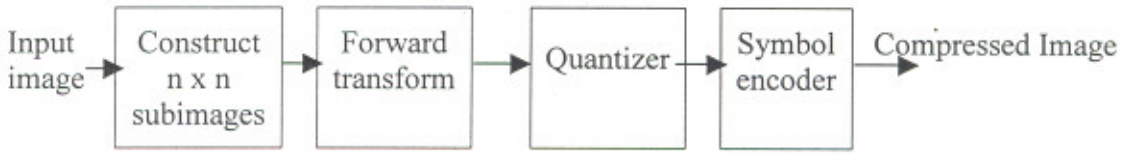

Fig 4.1(a) Encoding process of transform system.

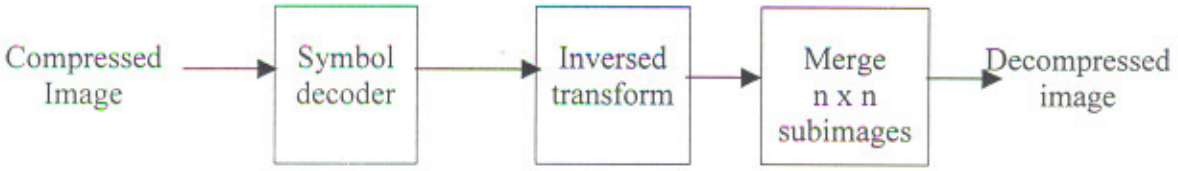

Fig 4.1(b) Decoder of transform coding system

4.2 SELECTION OF THE TRANSFORMATION

In transform coding system any kind of transformation can be chosen, such as Karhunen-Loeve (KLT), Discrete Cosine (DCT), Walsh-Hadamard (WHT) etc. The choice of a particular transformation in a given application depends on the amount of reconstruction error that can be tolerated and the computational resources available. Compression is achieved during the quantization of the transformed co-efficients and not during the transformation step.

In the present implementation Discrete Cosine Transformation (DCT) have been chosen. Because its basic images are fixed so it is input independent. It provides a good compromise between information packing ability and computational complexity. Compared to the other input independent transformations, it has the advantages of packing the more information into the fewest co-efficients for most natural images, and minimizing the block like appearance, called *blocking artifact*, that results when the boundaries between subimages become visible [29,32,34].

In encoding stage DCT is used to get mapped image coordinates and to reproduce. The image inverse DCT is used. Two dimensional DCT and Inverse DCT are as follows:

$$DCT(i,j) = 1/(2N)^{1/2} * c(i) * c(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x,y) \cos\frac{(2x+1)i\pi}{2N} \cos\frac{(2y+1)j\pi}{2N}$$

$$c(x) = 1/(2)^{1/2} \text{ if } x \text{ is } 0, \text{ else } 1 \text{ if } x > 0$$

Discrete Cosine Transform

$$Pixel(x,y) = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} c(i)c(j)DCT(i,j) \cos\frac{(2x+1)i\pi}{2N} \cos\frac{(2y+1)j\pi}{2N}$$

$$c(x) = \frac{1}{\sqrt{2N}} \text{ if } x \text{ is } 0, \text{ else } 1 \text{ if } x > 0$$

The Inverse Discrete Cosine Transform

DCT(i, j) is mapped coordinates and pixel(x, y) are actual coordinates in sub image. A C program based upon these two equations has been developed for the transformations required for image compression.

4.2.1 Subimage Size Selection

A significant factor affecting transform coding, error and computational complexity is function of subimage size. In most application images are subdivided so that the correlation (redundancy) between adjacent subimages is reduced to some acceptable level so that n can be expressed as an integral power of 2, where n is the subimage dimension. The latter condition simplifies the computation of the subimage transformation. In general, both the levels of compression and computational complexity increase as the subimage size increases. The most popular subimage sizes are 8 x 8 and 16 x 16 and subimage size of 8 x 8 have been chosen in the present work for analysis of compression algorithm.

4.2.2 Bit Allocation

The reconstruction error associated with quantization is a function of number and relative importance of the transformed co-efficient of the discarded, as well as precision of the retained co-efficients. In transformed coding systems, the retained co-efficients can be

selected on the basis of maximum variance, called *zonal coding*, or on the basis of maximum magnitude, called *threshold coding*. The overall process of truncating, quantizing and coding the co-efficients of a transformed subimage is commonly called *bit allocation*.

Zonal coding is based on the information theory. The zonal sampling process can be viewed, as multiplying each DCT (i,j) by the corresponding element in a zonal mask, which is constructed by placing 1 in the locations of maximum variance and 0 in all other locations. The co-efficients retained during the zonal sampling process must be quantized and coded. Zonal coding is usually implemented by using a single fixed mask for all subimages[17].

Threshold coding, however, is inherently adaptive in the sense that the location of the transform co-efficients retained for each subimage varies from one subimage to another. There are three basic ways to threshold a transformed subimage:

(1) A single global threshold can be applied to all subimages.

(2) A different threshold can be used for each subimage.

(3) The threshold can be varied as a function of the location of each co-efficient within the subimage.

In the first approach, the level of compression differs from image to image, depending on the number of the co-efficients that exceed the global threshold. In the second, called N-largest coding, the same number of co-efficients is discarded for each subimage. As a result, the code rate is constant and is known in advance. The third technique, like the first, results in a variable code rate, but offers the advantage that thresholding and quantization can be combined.

In the present implementation threshold coding using quantization matrix have been chosen. For every elemental position in the DCT matrix, a corresponding value in the quantization matrix produces a quantum value. The quantum value indicates what the step size is going to be, which ranges from 1 to 255 for the element in the compressed form of

the picture. The value that matters most to the picture will be encoded with the small step size, size 1 offering the most precision. The values can become higher as we move away from the origin. From the formula, it becomes clear that virtually all higher-frequency components will be rounded off to zero. The actual formula used for quantization is:

$$\text{Quantized Value (i, j)} = \frac{\text{DCT (i, j)}}{\text{Quantum (i, j)}}$$

Rounded to the nearest integer

During decoding, the dequantization formula operates in reverse:

$$\text{DCT (i,j)} = \text{Quantized value (i,j)} * \text{Quantum (i,j)}$$

From this it can be seen that when large quantum values are used, the risk of generating large error in the DCT output during dequantization exists. But fortunately, errors generated in the high-frequency components during dequantization normally don't have a serious effect on picture quality.

4.2.3 Quantization Matrix

Here a particular quantization matrix have not been chosen, instead it can be defined at runtime when compression takes place. Selecting quantization matrix at run time is just like to dial in a picture quality value when compressing graphics. By choosing extraordinarily high step sizes for most DCT co-efficients, we get excellent compression ratios and pure picture quality. By choosing cautiously low step sizes, compression level degrades gracefully, but picture quality will be excellent. This allows for a great deal of

flexibility for the users, choosing picture quality based on both imaging requirements and storage capacity.

To determine the value of the quantum step sizes, the user inputs a single quality factor, which should range from 1 to about 25. Values larger than 25 should also work, but with degraded picture reproduction quality. The quantization tables has been created using following algorithm.

```
For (i = 0; i < N; i ++)
For (j = 0; j < N; j ++)
Quantum [i] [j] = 1 + ( ( 1+ i + j)* quality);
```

The quality level sets the difference between adjoining bands of the same quantization level. These bands are originated on diagonal lines across the matrix, so quantization levels of the same value are all roughly the same distance from the origin. This set the threshold for the value of an element before it is going to contribute any meaningful information to the picture. Any contribution under the value of this threshold is simply unacceptable and is the critical point in the algorithm where the lossy effect takes place[25,33].

4.2.4 Coding

The final step in the compression process is coding the quantized image. First the co-efficients of the image are arranged in the zig-zag sequence. Then they have been encoded using run-length coding of zero values and variable-length integer coding.

4.2.4.1 The Zig-Zag Sequence

A simple code has been developed that gives the count of consecutive zero values in the quantized image, this gives a measure for outstanding compression. Because quantum bands of some values are arranged along the diagonals of the pixel matrix, so as to increase the length of run levels of quantized co-efficients arranged in the zig-zag sequence.

4.2.4.2 Run Length and Variable-length Integer Coding

After reordering the quantized DCT block in the zig-zag sequence. Basically, the output of the run-length and variable-length integer encoding consists of a sequence of three tokens, repeated until the block is complete. The three tokens are like this:

Run Length The number of consecutive zeros preceded the current element in the DCT output matrix.

Bit Count The number of bits to follow in the amplitude number.

Amplitude The amplitude of the DCT co-efficients.

The variable length integer coding schemes takes advantage of the fact that the DCT output should consist of mostly smaller numbers, which are required to encode with smaller number of bits.

4.3 CONCLUSION

The Discrete Cosine Transformation, the key to the compression, considered in this chapter is a mathematical transformation that includes the well known Fourier transformations. The basic operation performed by these transforms is to take a signal and transform it from one type of representation to another. Selection of transformation, coding scheme has been discussed in this chapter. In this implementation double coding scheme is used first image is coded as run length then run lengths are coded as variable length. Combined coding technique gives enormous amount of compression without much degradation in quality. With this compression system in most cases images can be compressed up to 85% without much loss in picture quality.

CHAPTER – 5

CONCLUSIONS AND FUTURE SCOPE

5.1 CONCLUSIONS

To save storage space and communication channel bandwidth, compression of large information files is a feasible approach. The Compression technique developed in the present work is based on transformation coding. Discrete Cosine transformation is used as it is found to be superior to other transformation techniques such as Walsh-Hadamard (WHT), Karhunen-Loeve (KLT) etc. Combination of run length and variable length coding gives a good amount of compression without much degradation in picture quality. The technique developed is very flexible as it gives user the choice for selecting the compression ratio and retrieved image quality. Present technique for compression has been tested and found successful to compress gray scale images. For most of the images quality factor of 5 or less produce a slight loss resolution, but without significant loss of picture quality. Once the quality factor gets above 15 blocking affect of compression starts and become visible. The images with quality factor of 20 or higher show graceful degradation in the image quality. In most cases images can be compressed up to 85% without much loss in picture quality.

5.2 FUTURE SCOPE OF WORK

In the present work, images with one colour component have been considered. But compression of coloured images can also be analyzed using this technique.

Here only still images are analyzed. Further this technique can be explored for moving images.

This technique further can be explored to compress audio files.

Source Code

```
/*************** START OF BITIO.C *************/

#include<stdio.h>
#include<stdlib.h>
#include "d:\bhatia\rkb\comp\BITIO.H"
#define PACIFIER_COUNT 2047

BIT_FILE *OpenOutputBitFile(char *name)
{
BIT_FILE *bit_file;
bit_file=(BIT_FILE *)calloc(1,sizeof(BIT_FILE));

/*
calloc provides access to the C memory heap,
which is available for dynamic allocation of
variable-sized blocks of memory.
Many data structures, such as trees and lists,
naturally employ heap memory allocation.
calloc allocates a block (nitems * size) bytes
and clears it to 0. To allocate a block larger
than 64K, use farcalloc.
*/

if(bit_file==NULL)
   return(bit_file);
bit_file->file=fopen(name,"wb");
bit_file->rack=0;
bit_file->mask=0x80;
bit_file->pacifier_counter=0;
return(bit_file);
}   //END OF OPENOUTPUTBITFILE FUNCTION

//START OF OPENOUTPUTBITFILE
BIT_FILE *OpenInputBitFile(char *name)
{
BIT_FILE *bit_file;
bit_file=(BIT_FILE *)calloc(1,sizeof(BIT_FILE));
if(bit_file==NULL)
   return(bit_file);
bit_file->file=fopen(name,"rb");
bit_file->rack=0;
bit_file->mask=0x80;
bit_file->pacifier_counter=0;
return(bit_file);
}  //END OF OPENINPUTBITFILE FUNCTION
```

```
//START OF CLOSEOUTPUTBITFILE
void CloseOutputBitFile(BIT_FILE *bit_file)
{
  if(bit_file->mask!=0x80)
     if(putc(bit_file->rack,bit_file->file)!=bit_file->rack)
       printf("Fatal error in CloseBitFile!");
     fclose(bit_file->file);
     free((char *)bit_file);
}    //END OF CLOSEOUTPUTBITFILE


//START OF CLOSEINPUTBITFILE
void CloseInputBitFile(BIT_FILE *bit_file)
{
  fclose(bit_file->file);
  free((char *)bit_file);
}  //END OF CLOSEINPUTBITFILE

//START OF OUTPUTBIT
void OutputBit(BIT_FILE *bit_file,int bit)
{
 if(bit)
   bit_file->rack|=bit_file->mask;
 bit_file->mask>>=1;
 if(bit_file->mask==0)
  {
   if(putc(bit_file->rack,bit_file->file)!=bit_file->rack)
       printf("Fatal error in OutputBit!\n");
   else
          if((bit_file->pacifier_counter++ &
          PACIFIER_COUNT/*4095*/)==0)//check it
           putc('.',stdout);
   bit_file->rack=0;
   bit_file->mask=0x80;
  }
}    //END OF OUTPUTBIT


//START OF OUTPUTBITS
void OutputBits(BIT_FILE *bit_file, unsigned long code,int
count)
{
  unsigned long mask;
  mask=1L<<(count-1);
```

```
while(mask!=0)
{
if(mask & code)
  bit_file->rack|=bit_file->mask;
bit_file->mask>>=1;
if(bit_file->mask==0)
 {
 if(putc(bit_file->rack,bit_file->file)!=bit_file->rack)
            printf("Fatal error in OutputBits!\n");
 else
     if((bit_file->pacifier_counter++ &
     PACIFIER_COUNT/*2047*/)==0)//check it
            putc('.',stdout);
 bit_file->rack=0;
 bit_file->mask=0x80;
 }     //end of if()
mask>>=1;
 }    //end of while
}     //End of outputbits

//start of InputBit
int InputBit(BIT_FILE *bit_file)
{
int value;
if(bit_file->mask==0x80)
  {
    if(feof(bit_file->file))
      {
      printf("Error ");
      getch();
      return;
      }
   bit_file->rack=getc(bit_file->file);
     if((bit_file->pacifier_counter++ &
     PACIFIER_COUNT/*2047*/)==0)//check it
           putc('.',stdout);
   }
value=bit_file->rack & bit_file->mask;
bit_file->mask>>=1;
if(bit_file->mask==0)
  bit_file->mask=0x80;
return(value?1:0);
}    //endof inpubit
```

```
//start of inputbits
unsigned long InputBits(BIT_FILE *bit_file,int bit_count)
{
  unsigned long mask, return_value;
  mask=1L<<(bit_count-1);
  return_value=0;
  while(mask!=0)
   {
     if(bit_file->mask==0x80)
      {
        bit_file->rack=getc(bit_file->file);
        if(feof(bit_file->file))
           {
           printf("Error ");
           getch();
           return;
           }
       if((bit_file->pacifier_counter++ & 2047)==0)
            putc('.',stdout);
      }
      if(bit_file->rack & bit_file->mask)
        return_value|=mask;
      mask>>=1;
      bit_file->mask>>=1;
      if(bit_file->mask==0)
        bit_file->mask=0x80;
      }   //end of while
     return(return_value);
   }      //End of outputbits

//Start of fileprintbinary
void FilePrintBinary(FILE *file,unsigned int code,int bits)
{
unsigned int mask;
mask=1<<(bits-1);
while(mask!=0)
  {
   if(code & mask)
     fputc('1',file);
   else
     fputc('0',file);
   mask>>=1;
   }
}    //end of fileprintbinary
/*************END OF BTIO.C*****************/
```

```
/*********** START OF COMPRESS.C************/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include"d:\bhatia\rkb\comp\BITIO.c"

/*the ROWS and COLS define the number of rows and columns in
   the grey scale image respectively .the last one ,'N'
,defines the DCT block size*/

#define ROWS 200
#define COLS 320
#define N 8

/*this macro is used to ensure correct rounding of integer
values*/

#define ROUND(a) (((a)<0)?(int)((a)-0.5):(int)((a)+0.5))

/*FUNCTION PROTOTYPES*/

void usage_exit(char *prog_name);
void print_ratios(char *input,char *output);
long file_size(char *name);
void initialize(int quality);
void ReadPixelStrip(FILE *input,unsigned char
strip[N][COLS]);
void OutputCode(BIT_FILE *output_file,int code);
void ForwardDCT(unsigned char *input[N],int output[N][N]);
void CompressFile(FILE *input,BIT_FILE *output,int argc,char
*argv[]);

/*global data used at various places*/
char intermediate;
unsigned char PixelStrip[N][COLS];
double C[N][N];
double Ct[N][N];

int InputRunLength;
int OutputRunLength;
int Quantum[N][N];
```

```
struct zigzag{
          int row;
          int col;
          }ZigZag[N*N]=
          {
           {0,0},
           {0,1},{1,0},
           {2,0},{1,1},{0,2},
           {0,3},{1,2},{2,1},{3,0},
           {4,0},{3,1},{2,2},{1,3},{0,4},
           {0,5},{1,4},{2,3},{3,2},{4,1},{5,0},
           {6,0},{5,1},{4,2},{3,3},{2,4},{1,5},{0,6},
           {0,7},{1,6},{2,5},{3,4},{4,3},{5,2},{6,1},{7,0},
           {7,1},{6,2},{5,3},{4,4},{3,5},{2,6},{1,7},
           {2,7},{3,6},{4,5},{5,4},{6,3},{7,2},
            {7,3},{6,4},{5,5},{4,6},{3,7},
           {4,7},{5,6},{6,5},{7,4},
           {7,5},{6,6},{5,7},
           {6,7},{7,6},
           {7,7}
          };

void Initialize(int quality)
{
int i,j;
double pi=atan(1.0)*4.0;
if(toupper(intermediate)=='Y')
   printf("Quantum Matrix is :");

for(i=0;i<N;i++)
{
  if(toupper(intermediate)=='Y')

  printf("\n");
  for(j=0;j<N;j++)
     {
     Quantum[i][j]=1+((1+i+j)*quality);
     if(j%N==0)
     {
     if(toupper(intermediate)=='Y')
     printf("\n");
     }
     if(toupper(intermediate)=='Y')
     printf("  Q[%d][%d] %d",i,j,Quantum[i][j]);

     }
}
```

```
OutputRunLength=0;
InputRunLength=0;
if(toupper(intermediate)=='Y')
printf("\n\nC and Ct matrix are :");

for(j=0;j<N;j++)
  {
  C[0][j]=1.0/sqrt((double)N);
  if(toupper(intermediate)=='Y')
     printf("C[0][%d] %d",j,C[0][j]);

  Ct[j][0]=C[0][j];
     if(toupper(intermediate)=='Y')

  printf("Ct[%d][0] %d\n",j,Ct[j][0]);
}

for(i=1;i<N;i++)
  {
  for(j=0;j<N;j++)
     {
      C[i][j]=sqrt(2.0/N)*cos(pi*(2*j+1)*i/(2.0*N));
     if(toupper(intermediate)=='Y')

      printf("C[%d][%d] %d",i,j,C[i][j]);

      Ct[j][i]=C[i][j];
     if(toupper(intermediate)=='Y')

      printf("Ct[%d][%d] %d\n",j,i,C[j][i]);

       //  getch();
      }

   }
//getch();
}
```

```
//MAIN FUNCTION STARTS HERE
int main(int argc,char *argv[])
 {
  BIT_FILE *output;
  FILE *input;
  setbuf(stdout,NULL);
  clrscr();
  if(argc<3)    usage_exit(argv[0]);
  input=fopen(argv[1],"rb");
  if(input==NULL)
   {
    printf("Error opening %s for input %s \n",argv[1]);
    exit(1);
   }
   output=OpenOutputBitFile(argv[2]);
   printf("\n Want to see intermediate results (y/n) ? ");
   scanf("%c",&intermediate);
   if(output==NULL)
     printf("Error opening %s for output %s \n",argv[2]);
    printf("\nCompressing %s to %s \n",argv[1],argv[2]);
    printf("Using DCT compression\n");
    CompressFile(input,output,argc-3,argv+3);
    CloseOutputBitFile(output);
    fclose(input);
    print_ratios(argv[1],argv[2]);
    return(0);
   }

/*this routine prints out the usage message that is*/
void usage_exit(char *prog_name)
{
char *short_name, *extension;
short_name=strrchr(prog_name,'\\');
if(short_name==NULL)  short_name=strrchr(prog_name,'/');
if(short_name==NULL)  short_name=strrchr(prog_name,':');
if(short_name!=NULL)  short_name++;
else
   short_name=prog_name;
extension=strrchr(short_name,'.');
if(extension!=NULL)
  *extension='\0';
printf("\nUsage:%s infile outfile[quality]\nquality from 0-
25",short_name);
exit(0);
}
```

```
void intialize(int quality)
{
int i,j;
double pi=atan(1.0)*4.0;
for(i=0;i<N;i++)
  for(j=0;j<N;j++)
     Quantum[i][j]=1+((1+i+j)*quality);
OutputRunLength=0;
InputRunLength=0;
for(j=0;j<N;j++)
  {
  C[0][j]=1.0/sqrt((double)N);
  Ct[j][0]=C[0][j];
  }
for(i=1;i<N;i++)
  {
  for(j=0;j<N;j++)
    {
      C[i][j]=sqrt(2.0/N)*(cos(pi*(2*j+1)*i/(2.0*N)));
     Ct[j][i]=C[i][j];
     }
   }
}

void ReadPixelStrip(FILE *input,unsigned char
strip[N][COLS])
{
int row,col,c,f=0;
for(row=0;row<N;row++)
 {
  for(col=0;col<COLS;col++)
     {
     if(!feof(input))
     {
     c=getc(input);
     }
     else
     {
      f=1; break;
     }
     if(f==1)  break;
   }
 strip[row][col]=(unsigned char)c;
 }
}
```

```
void OutputCode(BIT_FILE *output_file,int code)
{
int top_of_range,abs_code,bit_count;
if(code==0)
  {
   OutputRunLength++;   return;
  }
if(OutputRunLength!=0)
 {
 while(OutputRunLength>0)
  {
   OutputBits(output_file,0L,2);
   if(OutputRunLength<=16)
    {
     OutputBits(output_file,(unsigned long)(OutputRunLength-
1),4);
     OutputRunLength=0;
     }
      else
     {
      OutputBits(output_file,15L,4);
      OutputRunLength=16;
      }
       }/*while*/
    }/*if    */
if(code<0)
 abs_code=-code;
else
 abs_code=code;
top_of_range=1;
bit_count=1;
while(abs_code>top_of_range)
  {
   bit_count++;
   top_of_range=((top_of_range+1)*2)-1;
   }
if(bit_count<3)
  OutputBits(output_file,(unsigned long)(bit_count+1),3);
else
  OutputBits(output_file,(unsigned long)(bit_count+5),4);

if(code>0)
  OutputBits(output_file,(unsigned long)code,bit_count);
else
  OutputBits(output_file,(unsigned
long)(code+top_of_range),bit_count);
}
```

```
void ForwardDCT(unsigned char *input[N],int output[N][N])
{
double temp[N][N];
double temp1;
int i,j,k;

/*matrix multiply(temp,input,Ct)*/
for(i=0;i<N;i++)
  for(j=0;j<N;j++)
    {
     temp[i][j]=0.0;
     for(k=0;k<N;k++)
        temp[i][j]+=((int)input[i][k]-128)*Ct[k][j];
     }
/*matrix multiply(output,C,temp)*/
for(i=0;i<N;i++)
  for(j=0;j<N;j++)
    {
     temp1=0.0;
     for(k=0;k<N;k++)
       temp1+=C[i][k]*temp[k][j];
     output[i][j]=ROUND(temp1);
     }
}
void WriteDCTData(BIT_FILE *output_file,int
output_data[N][N])
{
int i;
int row,col;
double result;

for(i=0;i<(N*N);i++)
  {
   row=ZigZag[i].row;
   col=ZigZag[i].col;
   result=output_data[row][col]/Quantum[row][col];
   OutputCode(output_file,ROUND(result));
   }
}

void CompressFile(FILE *input,BIT_FILE *output,int argc,char
*argv[])
{
int row,col,i,output_array[N][N],quality;
unsigned char *input_array[N];
if(argc-->0)
  quality=atoi(argv[0]);
else
  quality=3;
```

```
if(quality<0 || quality>50)
    {
    printf("Illegal quality factor of %d\n",quality);
    exit(1);
    }
printf("Using quality factor of %d\n",quality);
Initialize(quality);
OutputBits(output,(unsigned long)quality,8);
for(row=0;row<ROWS;row+=N)
    {
    ReadPixelStrip(input,PixelStrip);
    for(col=0;col<COLS;col+=N)
     {
     for(i=0;i<N;i++)
        input_array[i]=PixelStrip[i]+col;
      ForwardDCT(input_array,output_array);
      WriteDCTData(output,output_array);
     }
    }
OutputCode(output,1);
while(argc-->0)
   printf("Unused argument:%s\n",*argv++);
}

#ifndef SEEK_END
#define SEEK_END 2
#endif

long file_size(char *name)
{
 long eof_ftell;
 FILE *file;
 file=fopen(name,"r");
 if(file==NULL)   return(0L);
 fseek(file,0L,SEEK_END);
 eof_ftell=ftell(file);
 fclose(file);
 return(eof_ftell);
}


void print_ratios(char *input,char *output)
{
long input_size,output_size;
int ratio;

input_size=file_size(input);
if(input_size==0)
  input_size=1;
```

```
output_size=file_size(output);
ratio=100-(int)(output_size*100L/input_size);
printf("\nInput bytes:   %ld\n",input_size);
printf("\nOutput bytes:   %ld\n",output_size);
if(output_size==0)
  output_size=1;
printf("Compression ratio:%d%%\n",ratio);
getch();
}


/************* END OF COMPRESS.C ************/
```

```
/******************START OF EXPAND.C*****************/
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include"d:\bhatia\rkb\comp\BITIO.c"

/*
*THE ROWS and COLS define the number of rows and columns in
*the grey scale image respectively.The last one,'N',defines
the DCT block
* size.
*/

#define ROWS    200
#define COLS  320
#define N      8

/*This macro is used to ensure correct rounding of integer
values.*/
#define ROUND(a)   (((a)<0)?(int)((a)-0.5):(int)((a)+0.5))
/* Function prototypes */
void Initialize(int quality);
int Inputcode(BIT_FILE *input);
void ReadDCTData(BIT_FILE *input,int input_data[N][N]);
void WritePixelStrip(FILE *output,unsigned char
strip[N][COLS]);
void InverseDCT(int input[N][N],unsigned char *output[N]);
void ExpandFile(BIT_FILE *input,FILE *output,int argc,char
*argv[]);

unsigned char PixelStrip[N][COLS];
double C[N][N];
double Ct[N][N];

int InputRunLength;
int OutputRunLength;
int Quantum[N][N];
```

```
struct zigzag{
          int row;
          int col;
          }ZigZag[N*N]=
          {
           {0,0},
           {0,1},{1,0},
           {2,0},{1,1},{0,2},
           {0,3},{1,2},{2,1},{3,0},
           {4,0},{3,1},{2,2},{1,3},{0,4},
           {0,5},{1,4},{2,3},{3,2},{4,1},{5,0},
           {6,0},{5,1},{4,2},{3,3},{2,4},{1,5},{0,6},
           {0,7},{1,6},{2,5},{3,4},{4,3},{5,2},{6,1},{7,0},
           {7,1},{6,2},{5,3},{4,4},{3,5},{2,6},{1,7},
           {2,7},{3,6},{4,5},{5,4},{6,3},{7,2},
           {7,3},{6,4},{5,5},{4,6},{3,7},
           {4,7},{5,6},{6,5},{7,4},
           {7,5},{6,6},{5,7},
           {6,7},{7,6},
           {7,7}
           };

void Initialize(int quality)
{
int i,j;
double pi=atan(1.0)*4.0;
printf("Quantum Matrix is :");
for(i=0;i<N;i++)
{
  printf("\n");
  for(j=0;j<N;j++)
     {
     Quantum[i][j]=1+((1+i+j)*quality);
     if(j%N==0)
     {
     printf("\n");
     }
     printf("  Q[%d][%d] %d",i,j,Quantum[i][j]);

     }
}
OutputRunLength=0;
InputRunLength=0;
printf("\n\nC and Ct matrix are :");
```

```
for(j=0;j<N;j++)
  {
  C[0][j]=1.0/sqrt((double)N);
  printf("C[0][%d] %d",j,C[0][j]);
  Ct[j][0]=C[0][j];
  printf("\tCt[%d][0] %d\n",j,Ct[j][0]);
}
for(i=1;i<N;i++)
  {
  for(j=0;j<N;j++)
     {
      C[i][j]=sqrt(2.0/N)*cos(pi*(2*j+1)*i/(2.0*N));
      printf("C[%d][%d] %d",i,j,C[i][j]);
      Ct[j][i]=C[i][j];
      printf("\tCt[%d][%d] %d\n",j,i,C[j][i]);
      }
   }
}

int InputCode(BIT_FILE* input_file)
{
int bit_count, result;
if(InputRunLength > 0)
{
InputRunLength--;
return(0);
}
bit_count=(int) InputBits(input_file,2);
if(bit_count==0)
{
InputRunLength=(int) InputBits(input_file,4);
return(0);
}
if(bit_count==1)
{
InputRunLength=(int) InputBits(input_file,4);
bit_count=(int) InputBits(input_file,1)+1;
}
else
{
bit_count=(int) InputBits(input_file,2)+(bit_count<<2)-5;
result=(int) InputBits(input_file,bit_count);
}
if(result & (1<<(bit_count-1)))return(result);
return(result-(1<<bit_count)+1);
}
```

```
void ReadDCTData(BIT_FILE *input,int input_data[N][N])
{
int I,row,col;
for(i=0;i<(N*N);i++)
{
row=ZigZag[i].row;
col=ZigZag[i].col;
input_data[row][col]=InputCode(input)*Quantum[row][col];
}
}

void WritePixelStrip(FILE *output,unsigned char
strip[N][COLS])
{
int row,col;
for(row=0;row<N;row++)
for(col=0;col<COLS;col++)
putc(strip[row][col],output);
}

void InverseDCT(int input[N][N],unsigned char *output[N])
{
double temp[N][N], temp1;
int I,j,k;
/* MatrixMultiply(temp,input,C); */
for(i=0;i<N;i++)
{
for(j=0;j<N;j++)
temp[i][j]=0.0;
for(k=0;k<N;k++)
temp[i][j]+=input[i][j]*C[k][j];
}
/* MatrixMultiply(output,Ct,temp); */
for(i=0;i<N;i++)
{
for(j=0;j<N;j++)
{
temp1=0.0;
for(k=0;k<N;k++)
temp1+=Ct[i][k]*temp[k][j];
temp1+=128.0;
if(temp1<0) output[i][j]=0;
else if(temp1>255) output[i][j]=255;
else output[i][j]=(unsigned char)ROUND(temp1);
}
}
}
```

```
void ExpandFile(BIT_FILE *input,FILE *output,int argc,char
*argv[])
{
int row,col,I, quality;
int input_array[N][N];
unsigned char *output_array[N];
quality=(int)InputBits(input,8);
printf("\rUsing quality factor of %d\n",quality);
Initialize(quality);
for(row=0;row<ROWS;row+=N)
{
for(col=0;col<COLS;col+=N)
{
for(i=0;i<N;i++)
output_array[i]=PixelStrip[i]+col;
ReadDCTData(input,input_array);
InverseDCT(input_array,output_array);
}
WritePixelStrip(output,PixelStrip);
}
while(argc-->0) printf("Unused argument:%s\n",*argv++);
}
long file_size(char *name)
{
 long eof_ftell;
 FILE *file;
 file=fopen(name,"rb");
 if(file==NULL)    return(0L);
 lseek(file,0L,SEEK_END);
 eof_ftell=ftell(file);
 fclose(file);
 return(eof_ftell);
}
void print_ratios(char *input,char *output)
{
long input_size,output_size;
int ratio;
input_size=file_size(input);
if(input_size==0)   input_size=1;
output_size=file_size(output);
ratio=100-(int)(output_size*100L/input_size);
printf("\n Input bytes:   %ld\n",input_size);
printf("\n Output bytes:   %ld\n",output_size);
if(output_size==0)  output_size=1;
printf(" Expansion ratio:%d% \n",ratio);
}
```

```
int main(int argc,char *argv[])
  {
   BIT_FILE *output;
   FILE *input;

   setbuf(stdout,NULL);
   if(argc<3)
    exit(1);
   input=fopen(argv[1],"rb");
   if(input==NULL)
    printf("Error opening %s for input %s\n",argv[1]);
   output=OpenOutputBitFile(argv[2]);
   if(output==NULL)
    printf("Error opening %s for output %s\n",argv[2]);
   printf("\nExpanding %s to %s \n",argv[1],argv[2]);
   printf("Using DCT compression\n");
   ExpandFile(input,output,argc-3,argv+3);
   CloseOutputBitFile(output);
   fclose(input);
   print_ratios(argv[1],argv[2]);
   getch();
   return(0);
  }

/***************End of EXPAND.C***************/
```

```
/************* START OF MOVING.C ***************/
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>

void move(int x, int y)
{
int size;
void *arrow;
if(y<=150)
 {
   /* calculate the size of the image */
   size = imagesize(x, y, x+250, y+40);
   /* allocate memory to hold the image */
   arrow = malloc(size);
   /* grab the image */
   getimage(x, y, x+200, y+40, arrow);
   }
  else if(y>150 && y<=230)
  {
     size = imagesize(x, y, x+350, y+40);
   /* allocate memory to hold the image */
     row = malloc(size);
   /* grab the image */
     getimage(x, y, x+350, y+40, arrow);
  }
 else if(y>230 && y<270)
    {
      size = imagesize(x, y, x+200, y+40);
      /* allocate memory to hold the image */
      arrow = malloc(size);
     /*grab the image */
      getimage(x, y, x+200, y+40, arrow);
  }
if(y==50)
  {
   while(x>190)
     {  /* erase old image */
      putimage(x, y, arrow, XOR_PUT);
       x -= 10;
```

```
        /* plot new image */
        putimage(x, y, arrow, XOR_PUT);
        if(y==50)
           delay(200);
        else
           delay(100);
        if(y==100 && x<250)
                break;

        }
       }
    else if(y<=150 && y>50)
    {
      while(x>230 )
       {  /* erase old image */
        putimage(x, y, arrow, XOR_PUT);
        x -= 10;
        /* plot new image */
        putimage(x, y, arrow, XOR_PUT);
        if(y==50)
           delay(200);
        else
           delay(100);
        if(y==100 && x<250)
                break;
        }
       }
       else
        {
        while(x>50)
         {  /* erase old image */
             putimage(x, y, arrow, XOR_PUT);
             x -= 10;
            /* plot new image */
            putimage(x, y, arrow, XOR_PUT);
            delay(100);
         }
        }
}

int main(void)
{
   /* request autodetection */
   int gdriver = DETECT, gmode, errorcode;
   void *arrow;
   int x, y, maxx;
   unsigned int size;
```

```
    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "d:\\software\\tc");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk)  /* an error occurred */
    {
        printf("Graphics error: %s\n",
        grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error code */
    }
    setbkcolor(1);
    setcolor(2);
    settextstyle(DEFAULT_FONT,HORIZ_DIR,3);
    moveto(150,10);
    outtext("DCT IMPLEMENTATION");
    settextstyle(GOTHIC_FONT,HORIZ_DIR,2);
    moveto(10,50);
    outtext("Submitted By: ");
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
    x=350;
    y=50;
    moveto(x,y);
    outtext("Mr. Rajesh Kumar");
    move(x,y);
    delay(1000);
    x=350;
    y=100;
    moveto(x,y);
    outtext("(ME-16/97)");
    move(x,y);
    settextstyle(GOTHIC_FONT,HORIZ_DIR,2);
    moveto(10,150);
outtext("Under Supervision Of :");
settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
x=400;
y=150;
moveto(x,y);
outtext("Dr. P.K. Bansal");
move(x,y);
x=270;  y=200;
moveto(x,y);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
outtext("Deptt. Of Computer Sci. & Engg,");
move(x,y);
x=270;   y=230;
moveto(x,y);
```

```
outtext("Thapar Institute Of Engg. & Tech.");
move(x,y);
x=400;   y=260;
moveto(x,y);
outtext("Deemed University");
move(x,y);
int   midx = getmaxx() / 2;
int  midy = getmaxy() / 2;
void *pt;
int d=0;
int j=SOLID_FILL;
int  mx = getmaxx() / 2;
int  my = getmaxy() / 2;
int tmx=mx,tmy=my;
void *ptr;

/* request auto detection */
int style=DEFAULT_FONT;
int siz=1;
int f=0,color;

while(!kbhit())//for(int z=0;z<20;z++)
 {
  /* loop through the fill patterns */

  for (int i=EMPTY_FILL; i<USER_FILL; i++)
   {
     /* set the fill style */
       f++;
     setfillstyle(i, getmaxcolor());
      /* draw the 3-d bar */
     bar3d(midx-50, midy+50, midx+50,
     midy+150, 10, 1);

     if(j>USER_FILL || mx>550)
        {
        j=SOLID_FILL;
        mx=midx;
        }

       else   j++;
       setfillstyle(j, getmaxcolor());
       bar(mx+60, my+50, mx+75, my+70);
     size=imagesize(mx+60,my+50,mx+75,my+70);
     if(size==0xFFFF)
        {
        outtext("Image is too large");  exit(-1);
        }
     ptr=malloc(size);
```

```
        getimage(mx+60,my+50,mx+75,my+70,ptr);
        putimage(mx+60,my+50,ptr,XOR_PUT);
        mx+=20;
        putimage(mx+60,my+50,ptr,XOR_PUT);
        if(mx>550)
          {
          delay(100);
          putimage(mx+60,my+50,ptr,XOR_PUT);
          }
        delay(100);
        setcolor(2);
        if(i==USER_FILL)    i=EMPTY_FILL;
        if(f%10==0)
         {
          f++;
          setcolor(1);
          outtextxy(250, 400,"Data Compression");
          setcolor(2);
          if (style == TRIPLEX_FONT) siz = 2;
          if(style>4)    style=0;
          /* select the text style */
          settextstyle(style, HORIZ_DIR, siz);
           style++;
          /* output a message */
          outtextxy(250, 400,"Data Compression");
         }
      }
  }

/* clean up */
free(arrow);
closegraph();
return 0;
}

/**************END OF MOVING.C*************/
```

REFERENCES

1. Abramson N. "Information Theory & Coding" Mc Graw Hill Co. 1963.
2. Ahmed ,N. , Natrajan,T.,and Rao, K.R. Discrete Cosine Tranform, IEEE Trans. Computers, Vol. C-23, Jan. 1974, pp. 90-93.
3. Ahmed N. and Rao K.R. , "Orthogonal Transforms For Digital Signal Processing." New York : Springes – Verlog 1975.
4. Arvind R. and Gersho A., "Image Compression Based On Quantization With Finite Memory," Opt. Engg. Vol. 26, pp. 570-580, 1987.
5. Boliek , M., Gormish, M.J., Schwartz, E.L., and Keith, A. Next Generation Image Compression and Manipulation , Ptoc. IEEE ICIP, 1997.
6. Bryan G. Talbot, Lisa M. Talbot "Source Coding With a Permutation Based Reversible Memory-Binding Transformation For Data Compression in Categorical Data Domains" IEEE Transactions on image processing, vol. 7, No. 8, Sept-1998.
7. C. and Gibson J.D. "Distribution Of Two Dimensional DCT Coefficients For Images," IEEE Trans. Commn. Vol. Com-31, pp. 835-839, June, 1983.
8. Calderbank, R.C., Daubechies,I., Sweldens, W., and Yeo, B.L. Wavelet Transform that Map Integers to Integers, Vol. 5,No.3,pp. 332-369 , 1998.
9. Coifman, R.R. and Wickerhauser, M.V. Entropy Based Alogrithm for Best Basis Selection , IEEE Trans. Information Theory , Vol. 38, No. 2, Mar.1992, pp. 713-718.
10. Computer Graphics Proceedings 1997 ACM SIGGRAPH.
11. David F. Roger, Procedural Elements for Computer Graphics, International students edition, Mc Graw-hill Book company 1985.
12. Donald Hearn and M. Pauline Backer, Computer Graphics, PHI 1996.
13. Froment, J., and Mallat,S. Second generation Compact Image Coating With wavelets, in C.K. Chui, editor, wavelets: A Tutorial in Theory and Applications, Vol.2, Academic Press, NY 1992.

14. Gersho, A. and Grey, R.M. Vector Quantization and Signal Compression, Kluwer Academic Publishers,1991.

15. Hilton, M.L. , Jawerth, B.D. , and Sengupta, A. Compressing Still and Moving Images with Wavelets, Multimedia Systems, Vol.2, No.3, April,1994.

16. ISO/IEC/JTC1/SC29/WG1 N390R, JPEG 2000 Image Coding System, Mar. 1997, http://www.jpeg.org/public/wgln505.pdf.

17. ITU-T Recommmendation T-82 Information Technology-Coded representation of picture & audio information progressive bi level image compression helsinki, Finland 1993.

18. Johnsen O., "On The Redundancy Of Binary Huffman Codes" IEEE Trans. Inform. Theory, Vol. IT-26, No.-2, pp. 220-223, Mar. 1980.

19. Lewis, A.S. and Knowles, G. Image Compression Using the 2-D Wavelets Transform, IEEE Trans. IP, Vol.1 No.2 ,April 1992, pp. 244-250.

20. Maxted J. and Robinson J.P., "Error Recovery For Variable Length Codes," IEEE Trans. IT. 31.6, pp. 794-801 (Nov. 1985).

21. Murakarni H., Mastsumoto S. & Yamamoto H., "Algorithm For Construction Of Variable Length Codes With Limited Maximum Word Length," IEEE Trans. Com-31.10, pp. 1157-1159 (Oct-1984).

22. Newman and Sproull, Principles of Interactive Computer Graphics, Mc Graw-Hill 1988.

23. PC Quest November 1998 pp. 51, Understanding graphics file formats.

24. Pennebaker, W.B. and Mitchell, J.L. JPEG - Still Image Data Compression Standards, Van Norstand Reinhold, 1993.

25. Perlman W.A. and Chekima A. "Source Coding Bounds Using Quantizer Reproduction Levels," IEEE Trans. Inform. Theory Vol. IT-30, pp. 559-567, May-1984.

26. Rabbani M. and Jones P.W., "Digital Image Compression Techniques," Bellingham., Washington : SPIE Optical Engg. Press 1991.

27. Rao,K.R. and Yip, P. Discrete Cosine Transforms - Alogrithms, Advantages, Applications, Academic press, 1990.

28. Roger T. Stevens and Christopher D., Advanced Graphics Programming in C and C++, BPB Publications 1993.

29. Said, A. and Pearlman, W.A. a New , fast, and Efficient Image Code Based on Set Partitioning in Hierarchical Trees, IEEE Trans. CSVT, Vol.6, NO.3, June 1996, pp. 243-250, ftp:// ipl.rpi.edu/pub/EW_Code/SPIHT.ps.gz.

30. Steven Harriington, Computer Graphics a Programming Approach, Mc Graw-Hill Book Company, 1987.

31. Takishima Y., Wada M. and Murakami H., "Variable Length Codes With High Resynehronization," The Journal Of Institute of Television Engineers Of Japan Vol, 43, No. 8, (1989), pp. 844-846.

32. Tanaka Z. Ji K. and Kitamura S. "Block Permutation Coding of Images Using Cosine Transforma," IEEE Trans. Commn. Vol. 43, pp. 2233-2246.

33. Vishwanath M. and Chou P. "An Efficient Algorithm For hierarchical Compression of Video" in 1994 int conf. Image Processing Austin, Tx, Nov. 1994, vol. 3, pp. 275-279.

34. Wallace, G.K. The JPEG Still Picture Compression Standard, Comm.ACM, Vol.34, No.4, April 1991, pp. 30-34.