CHAPTER 1

INTRODUCTION OF THE PROJECT

1.1 Introduction

Image compression is very important for efficient transmission and storage of images . Demand for communication of multimedia data through the telecommunications network and accessing the multimedia data through Internet is growing explosively. With the use of digital cameras, requirements for storage, manipulation, and transfer of digital images, has grown explosively . These image files can be very large and can occupy a lot of memory. A gray scale image that is 256 x 256 pixels has 65, 536 elements to store, and a a typical 640 x 480 color image has nearly a million. Downloading of these files from internet can be very time consuming task. Image data comprise of a significant portion of the multimedia data and they occupy the major portion of the communication bandwidth for multimedia communication. Therefore development of efficient techniques for image compression has become quite necessary. A common characteristic of most images is that the neighbouring pixels are highly correlated and therefore contain highly redundant information. The basic objective of image compression is to find an image representation in which pixels are less correlated. The two fundamental principles used in image compression are redundancy and irrelevancy. Redundancy removes redundancy from the signal source and irrelevancy omits pixel values which are not noticeable by human eye. JPEG and JPEG 2000 are two important techniques used for image compression.

Work on international standards for image compression started in the late 1970s with the CCITT (currently ITU-T) need to standardize binary image compression algorithms for Group 3 facsimile communications. Since then, many other committees and standards have been formed to produce de jure standards (such as JPEG), while several commercially successful initiatives have effectively become de facto standards (such as GIF).

1.2 Hardware and Software Requirements

Hardware Requirements

1. A Personal Computer/Laptop
2. 30 Gb Disk Space
3. 4 Gb or greater RAM
4. 1 Gb or greater GPU

Software Requirements

5. Mat Lab Software
6. Guide
7. Text Editor
8. Image Viewer

CHAPTER-2

IMAGE COMPRESSION

2..1 Introduction

The Main purpose of Image compression is to reduce the size of image without effecting its ability and clarity.

An image is an artifact that depicts visual perception, such as a photograph or other two-dimensional picture, that resembles a subject—usually a physical object—and thus provides a depiction of it. In the context of signal processing, an image is a distributed amplitude of color

Images may be two-dimensional, such as a photograph or screen display, or three-dimensional, such as a statue or hologram. They may be captured by optical devices – such as cameras, mirrors, lenses, telescopes, microscopes, etc. and natural objects and phenomena, such as the human eye or water.

Image compression is an application of data compression that encodes the original image with few bits. The objective of image compression is to reduce the redundancy of the image and to store or transmit data in an efficient form. Fig 1.1 shows the block diagram of the general image storage system. The main goal of such system is to reduce the storage quantity as much as possible, and the decoded image displayed in the monitor can be similar to the original image as much as can be. The essence of each block will be introduced in the following sections.

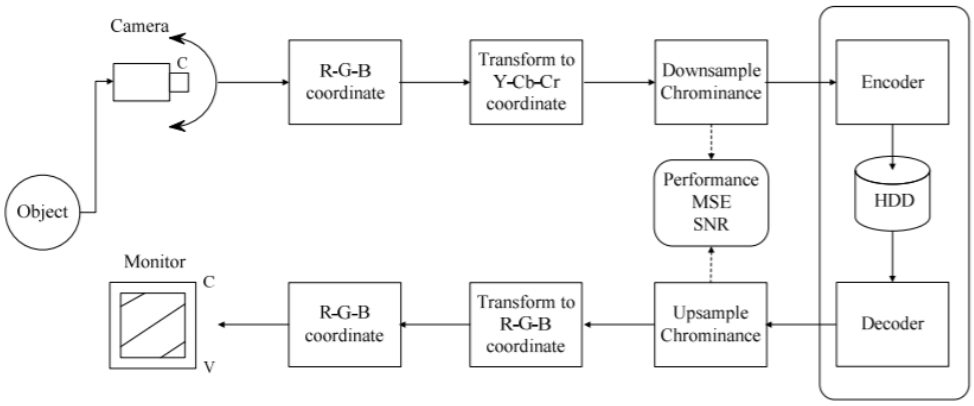


Fig 2 .1 General Image Storage System

2.2 Color Specification

The Y, Cb, and Cr components of one color image are defined in YUV color coordinate, where Y is commonly called the luminance and Cb, Cr are commonly called the chrominance. The meaning of luminance and chrominance is described as follows

- Luminance: received brightness of the light, which is proportional to the total energy in the visible band.
- Chrominance: describe the perceived color tone of a light, which depends on the wavelength composition of light chrominance is in turn characterized by two attributes – hue and saturation.
    1. **hue**: Specify the colour tone, which depends on the peak wavelength of the light
    2. **saturation**: Describe how pure the color is, which depends on the spread or bandwidth of the light spectrum

The RGB primary commonly used for color display mixes the luminance and chrominance attributes of a light. In many applications, it is desirable to describe a color in terms of its luminance and chrominance content separately, to enable more efficient processing and transmission of color signals. Towards this goal, various three-component color coordinates have been developed, in which one component reflects the luminance and the other two collectively characterize hue and saturation. One such coordinate is the YUV color space. The [Y Cb Cr]T values in the YUV coordinate are related to the [R G B]T values in the RGB coordinate by

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.334 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

Similarly, if we would like to transform the YUV coordinate back to RGB coordinate, the inverse matrix can be calculated from (1.1), and the inverse transform is taken to obtain the corresponding RGB components.

2.3 Spatial Sampling of Color Component

Because the eyes of human are more sensitive to the luminance than the chrominance, the sampling rate of chrominance components is half that of the luminance component. This will result in good performance in image compression with almost no loss of characteristics in visual perception of the new up sampled image. There are three color formats in the baseline system:

- **4:4:4 format:** The sampling rate of the luminance component is the same as those of the chrominance.
- **4:2:2 format:** There are 2 Cb samples and 2 Cr samples for every 4 Y samples. This leads to half number of pixels in each line, but the same number of lines per frame.
- **4:2:0 format:** Sample the Cb and Cr components by half in both the horizontal and vertical directions. In this format, there are also 1 Cb sample and 1 Cr sample for every 4 Y samples.

At the decoder, the down sampled chrominance components of 4:2:2 and 4:2:0 formats should be up sampled back to 4:4:4 format.

2.4 The Flow of Image Compression Coding

What is the so-called image compression coding? Image compression coding is to store the image into bit-stream as compact as possible and to display the decoded image in the monitor as exact as possible. Now consider an encoder and a decoder as shown in Fig. 1.3. When the encoder receives the original image file, the image file will be converted into a series of binary data, which is called

the bit-stream. The decoder then receives the encoded bit-stream and decodes it to form the decoded image. If the total data quantity of the bit-stream is less than the total data quantity of the original image, then this is called image compression. The full compression flow is as shown in Fig. 1.3.

Fig. 2.2 The basic flow of image compression coding

The compression ratio is defined as follows:

$$Cr = \frac{n1}{n2},$$

where n1 is the data rate of original image and n2 is that of the encoded bit-stream. In order to evaluate the performance of the image compression coding, it is necessary to define a measurement that can estimate the difference between the original image and the decoded image. Two common used measurements are the Mean Square Error (MSE) and the Peak Signal to Noise Ratio (PSNR), which are defined in (1.3) and (1.4), respectively. f(x,y) is the pixel value of the original image, and f'(x,y)is the pixel value of the decoded image. Most image compression systems are designed to minimize the MSE and maximize the PSNR.

$$MSE = \sqrt{\frac{\sum_{x=0}^{W-1}\sum_{y=0}^{H-1}[f(x,y) - f'(x,y)]^2}{WH}}$$

$$PSNR = 20\log_{10}\frac{255}{MSE}$$

The general encoding architecture of image compression system is shown is Fig. 1.4. The fundamental theory and concept of each functional block will be introduced in the following sections.

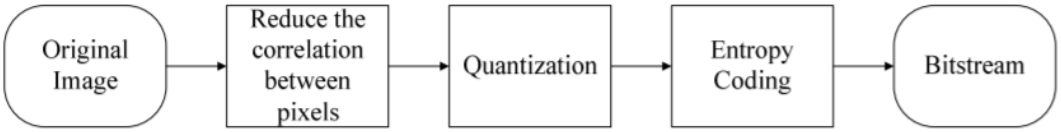


Fig 2.3 The general encoding flow of image compression

2.5 Reduce the Correlation between Pixels

Why an image can be compressed? The reason is that the correlation between one pixel and its neighbor pixels is very high, or we can say that the values of one pixel and its adjacent pixels are very similar. Once the correlation between the pixels is reduced, we can take advantage of the statistical characteristics and the variable length coding theory to reduce the storage quantity. This is the most important part of the image compression algorithm; there are a lot of relevant processing methods being proposed. The best-known methods are as follows:

- **Predictive Coding**: Predictive Coding such as DPCM (Differential Pulse Code Modulation) is a lossless coding method, which means that the decoded image and the original image have the same value for every corresponding element.
- **Orthogonal Transform**: Karhunen-Loeve Transform (KLT) and Discrete Cosine Transform (DCT) are the two most well-known orthogonal transforms. The DCT-based image compression standard such as JPEG is a lossy coding method that will result in some loss of details and unrecoverable distortion.
- **Subband Coding**: Subband Coding such as Discrete Wavelet Transform (DWT) is also a lossy coding method. The objective of subband coding is to divide the spectrum of one image into the lowpass and the highpass components. JPEG 2000 is a 2-dimension DWT based image compression standard.

2.6 Quantization

The objective of quantization is to reduce the precision and to achieve higher compression ratio. For instance, the original image uses 8 bits to store one element for every pixel; if we use less bits such as 6 bits to save the information of the image, then the storage quantity will be reduced, and the image can be compressed. The shortcoming of quantization is that it is a lossy operation, which will result into loss of precision and unrecoverable distortion.

CHAPTER 3

IMAGE COMPRESSION METHODS

Image compression can be further classified or divided in two separate types such as lossy compression and loss-less compression. In the lossy compression as its name indicated that it results in the loss of little information. In this technique the compressed image is same as to actual/original uncompressed image yet not exact to the previous one as within the compression process littler information related to the image has been lost. So they are normally applied for the photographs. The very natural example of the lossy compression is a JPEG.

Where are in Lossless compression, it compresses an image by encoding it's all information from the actual file, so in case if the image is get decompressed again, then it will be the exactly same as the actual image. For examples of the lossless technique of image compression are PNG and GIF i.e., GIF only provides 8-bit images. At the time of using a specific format of image compression that basically based on what is being get compressed.

3.1 LOSSLESS

Lossless In the technique of Lossless compression with the compressing of data that is when get decompressed, will be the same replica of actual data. In this case, when the binary data like the documents, executable etc. are get compressed. This required to be reproduced exactly when get decompressed again. On the contrary, the images and the music also required not to be generated 'exactly'. A resemblance of the actual image is sufficient for the most objective, as far as the error or problems between the actual and compressed image is avoidable or tolerable.

These types of compression are also known as noiseless as they never add noise to signal or image. It is also termed as the entropy coding as it uses the techniques of decomposition/statistics to remove/reduce the redundancy. It is also used only for the some specific applications along with the rigid needs like a medical-imaging. Below mentioned techniques consists in the lossless compression:

1. Huffman encoding
2. Run length encoding
3. Arithmetic coding

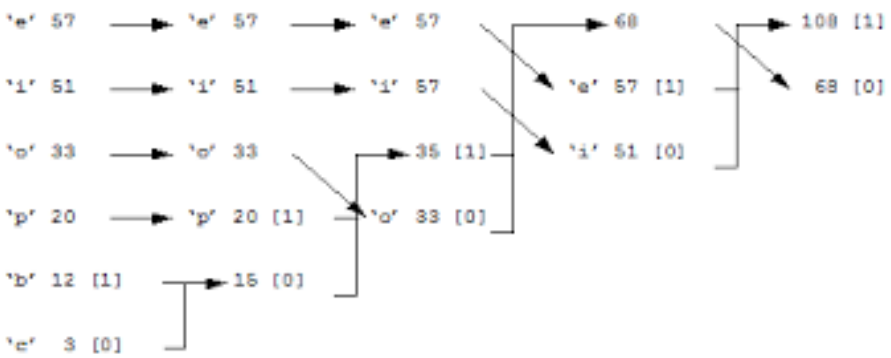
3.1.1 Huffman Encoding:

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.The variable-length codes assigned to input characters are prefix codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bit stream.Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be “cccd” or “ccb” or “acd” or “ab”.

In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding and/or using such a code proceeds by means of Huffman coding, an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".

The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (weight) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted.[2] However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods

We give an example of the result of Huffman coding for a code with five characters and given weights. We will not verify that it minimizes L over all codes, but we will compute L and compare it to the Shannon entropy H of the given set of weights; the result is nearly optimal.



For any code that is biunique, meaning that the code is uniquely decodeable, the sum of the probability budgets across all symbols is always less than or equal to one. In this example, the sum is strictly equal to one; as a result, the code is termed a complete code. If this is not the case, you can always derive an equivalent code by adding extra symbols (with associated null probabilities), to make the code complete while keeping it biunique.

As defined by Shannon (1948), the information content h (in bits) of each symbol ai with non-null probability is

$$h(a_i) = \log_2 \frac{1}{w_i}.$$

The entropy H (in bits) is the weighted sum, across all symbols ai with non-zero probability with of the information content of each symbol:

$$H(A) = \sum_{w_i > 0} w_i h(a_i) = \sum_{w_i > 0} w_i \log_2 \frac{1}{w_i} = -\sum_{w_i > 0} w_i \log_2 w_i.$$

As a consequence of Shannon's source coding theorem, the entropy is a measure of the smallest codeword length that is theoretically possible for the given alphabet with associated weights. In this example, the weighted average codeword length is 2.25 bits per symbol, only slightly larger than the calculated entropy of 2.205 bits per symbol. So not only is this code optimal in the sense that no other feasible code performs better, but it is very close to the theoretical limit established by Shannon.

In general, a Huffman code need not be unique. Thus the set of Huffman codes for a given probability distribution is a non-empty subset of the codes minimizing L(C) for that probability distribution. (However, for each minimizing codeword length assignment, there exists at least one Huffman code with those lengths.)

3.1.2 RUN LENGTH CODING

Run-length encoding (RLE) is a very simple form of lossless data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs. Consider, for example, simple graphic images such as icons, line drawings, Conway's Game of Life, and animations. It is not useful with files that don't have many runs as it could greatly increase the file size.

RLE may also be used to refer to an early graphics file format supported by CompuServe for compressing black and white images, but was widely supplanted by their later Graphics Interchange Format. RLE also refers to a little-used image format in Windows 3.x, with the extension rle, which is a Run Length Encoded Bitmap, used to compress the Windows 3.x startup screen.

For example, consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text. A hypothetical scan line, with B representing a black pixel and W representing white, might read as follows:

WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWWWW
WWWWWWWWWWBWWWWWWWWWWWWW

With a run-length encoding (RLE) data compression algorithm applied to the above hypothetical scan line, it can be rendered as follows:

12W1B12W3B24W1B14W

This can be interpreted as a sequence of twelve Ws, one B, twelve Ws, three Bs, etc..,

The run-length code represents the original 67 characters in only 18. While the actual format used for the storage of images is generally binary rather than ASCII characters like this, the principle remains the same. Even binary data files can be compressed with this method; file format specifications often dictate repeated bytes in files as padding space. However, newer compression methods such as DEFLATE often use LZ77-based algorithms, a generalization of

run-length encoding that can take advantage of runs of strings of characters (such as BWWBWWBWWBWW).

Run-length encoding can be expressed in multiple ways to accommodate data properties as well as additional compression algorithms. For instance, one popular method encodes run lengths for runs of two or more characters only, using an "escape" symbol to identify runs, or using the character itself as the escape, so that any time a character appears twice it denotes a run. On the previous example, this would give the following:

WW12BWW12BB3WW24BWW14

This would be interpreted as a run of twelve Ws, a B, a run of twelve Ws, a run of three Bs, etc. In data where runs are less frequent, this can significantly improve the compression rate.

One other matter is the application of additional compression algorithms. Even with the runs extracted, the frequencies of different characters may be large, allowing for further compression; however, if the run lengths are written in the file in the locations where the runs occurred, the presence of these numbers interrupts the normal flow and makes it harder to compress. To overcome this, some run-length encoders separate the data and escape symbols from the run lengths, so that the two can be handled independently. For the example data, this would result in two outputs, the string "WWBWWBBWWBWW" and the numbers

Run-length encoding schemes were employed in the transmission of television signals as far back as 1967.It is particularly well suited to palette-based bitmapped images such as computer icons, and was a popular image compression method on early online services such as CompuServe before the advent of more sophisticated formats such as GIF. It does not work well at all on continuous-tone images such as photographs, although JPEG uses it quite effectively on the coefficients that remain after transforming and quantizing image blocks.

Common formats for run-length encoded data include Truevision TGA, PackBits, PCX and ILBM. The ITU also describes a standard to encode run-length-colour for fax machines, known as T.45. The standard, which is combined with other techniques into Modified Huffman coding,[citation needed] is relatively efficient because most faxed documents are generally white space, with occasional interruptions of black.

3.1.3 ARTHIMETIC CODING

Arithmetic coding is a form of entropy encoding used in lossless data compression. Normally, a string of characters such as the words "hello there" is represented using a fixed number of bits per character, as in the ASCII code. When a string is converted to arithmetic encoding, frequently used characters will be stored with fewer bits and not-so-frequently occurring characters will be stored with more bits, resulting in fewer bits used in total. Arithmetic coding differs from other forms of entropy encoding, such as Huffman coding, in that rather than separating the input into component symbols and replacing each with a code, arithmetic coding encodes the entire message into a single number, an arbitrary-precision fraction q where $0.0 \leq q < 1.0$. It represents the current information as a range, defined by two numbers. Recent family of entropy coders called asymmetric numeral systems allows for faster implementations thanks to directly operating on a single natural number representing the current information.

In the simplest case, the probability of each symbol occurring is equal. For example, consider a set of three symbols, A, B, and C, each equally likely to occur. Simple block encoding would require 2 bits per symbol, which is wasteful: one of the bit variations is never used. That is to say, A=00, B=01, and C=10, but 11 is unused.

A more efficient solution is to represent a sequence of these three symbols as a rational number in base 3 where each digit represents a symbol. For example, the sequence "ABBCAB" could become 0.0112013, in arithmetic coding as a value in the interval [0, 1). The next step is to encode this ternary number using a fixed-point binary number of sufficient precision to recover it, such as 0.00101100102 — this is only 10 bits; 2 bits are saved in comparison with naïve block encoding. This is feasible for long sequences because there are efficient, in-place algorithms for converting the base of arbitrarily precise numbers.

To decode the value, knowing the original string had length 6, one can simply convert back to base 3, round to 6 digits, and recover the string.

The first thing to understand about arithmetic coding is what it produces. Arithmetic coding takes a message (often a file) composed of symbols (nearly always eight-bit characters), and converts it to a floating-point number greater than or equal to zero and less than one. This floating-point number can be quite long — effectively your entire output file is one long number — which means it is not a normal data type that you are accustomed to using in conventional programming languages. My implementation of the algorithm will have to create this floating-point number from

scratch, bit by bit, and likewise read it in and decode it bit by bit. This encoding process is done incrementally. As each character in a file is encoded, a few bits will be added to the encoded message, so it is built up over time as the algorithm proceeds.

The second thing to understand about arithmetic coding is that it relies on a model to characterize the symbols it is processing. The job of the model is to tell the encoder what the probability of a character is in a given message. If the model gives an accurate probability of the characters in the message, they will be encoded very close to optimally. If the model misrepresents the probabilities of symbols, the encoder may actually expand a message instead of compressing it!

The term arithmetic coding covers two separate processes: encoding messages and decoding them. I'll start by looking at the encoding process with sample C++ code that implements the algorithm in a very limited form using C++ double data. The code in this first section is only useful for exposition. That is, don't try to do any real compression with it.

To perform arithmetic encoding, we first need to define a proper model. Remember that the function of the model is to provide probabilities of a given character in a message. The conceptual idea of an arithmetic coding model is that each symbol will own its own unique segment of the number line of real numbers between 0 and 1. It's important to note that there are many different ways to model character probabilities. Some models are static, never changing. Others are updated after every character is processed. The only two things that matter to us are that the model attempts to accurately predict the probability a character will appear, and that the encoder and decoder have identical models at all times.

As an example, we can start with an encoder that can encode only an alphabet of 100 different characters. In a simple static model, we will start with capital letters, then move to the lower case letters. This means that the first symbol, 'A', will own the number line from 0 to .01, 'B' will own .01 to .02, and so on. (In all cases, this is strictly a half-closed interval, so the probability range for 'A' is actually >= 0 and < .01.)

With this model, my encoder can represent the single letter 'B' by outputting a floating-point number that is less than .02 and greater than or equal to .01. So for example, an arithmetic encoder that wanted to create that single letter could output .15 and be done.

Obviously, an encoder that just outputs single characters is not much use. To encode a string of symbols involves a slightly more complicated process. In this process, the first character defines a range of the number line that corresponds to the section assigned to it by the model. For the character 'B', that means the message is between .01 and .02.

3.1.4 LZ77

LZ77 algorithms achieve compression by replacing repeated occurrences of data with references to a single copy of that data existing earlier in the uncompressed data stream. A match is encoded by a pair of numbers called a length-distance pair, which is equivalent to the statement "each of the next length characters is equal to the characters exactly distance characters behind it in the uncompressed stream". (The "distance" is sometimes called the "offset" instead.)

To spot matches, the encoder must keep track of some amount of the most recent data, such as the last 2 kB, 4 kB, or 32 kB. The structure in which this data is held is called a sliding window, which is why LZ77 is sometimes called sliding-window compression. The encoder needs to keep this data to look for matches, and the decoder needs to keep this data to interpret the matches the encoder refers to. The larger the sliding window is, the longer back the encoder may search for creating references.

It is not only acceptable but frequently useful to allow length-distance pairs to specify a length that actually exceeds the distance. As a copy command, this is puzzling: "Go back four characters and copy ten characters from that position into the current position". How can ten characters be copied over when only four of them are actually in the buffer? Tackling one byte at a time, there is no problem serving this request, because as a byte is copied over, it may be fed again as input to the copy command. When the copy-from position makes it to the initial destination position, it is consequently fed data that was pasted from the beginning of the copy-from position. The operation is thus equivalent to the statement "copy the data you were given and repetitively paste it until it fits". As this type of pair repeats a single copy of data multiple times, it can be used to incorporate a flexible and easy form of run-length encoding.

Another way to see things is as follows: While encoding, for the search pointer to continue finding matched pairs past the end of the search window, all characters from the first match at offset D and forward to the end of the search window must have matched input, and these are the

(previously seen) characters that comprise a single run unit of length LR, which must equal D. Then as the search pointer proceeds past the search window and forward, as far as the run pattern repeats in the input, the search and input pointers will be in sync and match characters until the run pattern is interrupted. Then L characters have been matched in total, L > D, and the code is [D, L, c].

Upon decoding [D, L, c], again, D = LR. When the first LR characters are read to the output, this corresponds to a single run unit appended to the output buffer. At this point, the read pointer could be thought of as only needing to return int(L/LR) + (1 if L mod LR ≠ 0) times to the start of that single buffered run unit, read LR characters (or maybe fewer on the last return), and repeat until a total of L characters are read. But mirroring the encoding process, since the pattern is repetitive, the read pointer need only trail in sync with the write pointer by a fixed distance equal to the run length LR until L characters have been copied to output in total.

The pseudocode is a reproduction of the LZ77 compression algorithm sliding window.

```
while input is not empty do
    prefix := longest prefix of input that begins in window

    if prefix exists then
        i := distance to start of prefix
        l := length of prefix
        c := char following prefix in input
    else
        i := 0
        l := 0
        c := first char of input
    end if

    output (i, l, c)

    s := pop l+1 chars from front of input
    discard l+1 chars from front of window
    append s to back of window
repeat
```

Even though all LZ77 algorithms work by definition on the same basic principle, they can vary widely in how they encode their compressed data to vary the numerical ranges of a length–distance pair, alter the number of bits consumed for a length–distance pair, and distinguish their length–distance pairs from literals (raw data encoded as itself, rather than as part of a length–distance pair). A few examples:

- The algorithm illustrated in Lempel and Ziv's original 1977 article outputs all its data three values at a time: the length and distance of the longest match found in the buffer, and the literal that followed that match. If two successive characters in the input stream could be encoded only as literals, the length of the length–distance pair would be 0.

- LZSS improves on LZ77 by using a 1-bit flag to indicate whether the next chunk of data is a literal or a length–distance pair, and using literals if a length–distance pair would be longer.

- In the PalmDoc format, a length–distance pair is always encoded by a two-byte sequence. Of the 16 bits that make up these two bytes, 11 bits go to encoding the distance, 3 go to encoding the length, and the remaining two are used to make sure the decoder can identify the first byte as the beginning of such a two-byte sequence.

- In the implementation used for many games by Electronic Arts,[7] the size in bytes of a length–distance pair can be specified inside the first byte of the length–distance pair itself; depending on whether the first byte begins with a 0, 10, 110, or 111 (when read in big-endian bit orientation), the length of the entire length–distance pair can be 1 to 4 bytes large.

- As of 2008, the most popular LZ77-based compression method is DEFLATE; it combines LZ77 with Huffman coding.[8] Literals, lengths, and a symbol to indicate the end of the current block of data are all placed together into one alphabet. Distances can be safely placed into a separate alphabet; because a distance only occurs just after a length, it cannot be mistaken for another kind of symbol or vice versa.

3.1.5 LZ78

LZ78 algorithms achieve compression by replacing repeated occurrences of data with references to a dictionary that is built based on the input data stream. Each dictionary entry is of the form dictionary[...] = {index, character}, where index is the index to a previous dictionary entry, and character is appended to the string represented by dictionary[index]. For example, "abc" would be stored (in reverse order) as follows: dictionary[k] = {j, 'c'}, dictionary[j] = {i, 'b'}, dictionary[i] = {0, 'a'}, where an index of 0 specifies the first character of a string. The algorithm initializes last matching index = 0 and next available index = 1. For each character of the input stream, the dictionary is searched for a match: {last matching index, character}. If a match is found, then last matching index is set to the index of the matching entry, and nothing is output. If a match is not found, then a new dictionary entry is created: dictionary[next available index] = {last matching index, character}, and the algorithm outputs last matching index, followed by character, then resets last matching index = 0 and increments next available index. Once the dictionary is full, no more entries are added. When the end of the input stream is reached, the algorithm outputs last matching index. Note that strings are stored in the dictionary in reverse order, which an LZ78 decoder will have to deal with. LZW is an LZ78-based algorithm that uses a dictionary pre-initialized with all possible characters (symbols) or emulation of a pre-initialized dictionary. The main improvement of LZW is that when a match is not found, the current input stream character is assumed to be the first character of an existing string in the dictionary (since the dictionary is initialized with all possible characters), so only the last matching index is output (which may be the pre-initialized dictionary index corresponding to the previous (or the initial) input character). Refer to the LZW article for implementation details.

BTLZ is an LZ78-based algorithm that was developed for use in real-time communications systems (originally modems) and standardized by CCITT/ITU as V.42bis. When the trie-structured dictionary is full, a simple re-use/recovery algorithm is used to ensure that the dictionary can keep adapting to changing data. A counter cycles through the dictionary. When a new entry is needed, the counter steps through the dictionary until a leaf node is found (a node with no dependents). This is deleted and the space re-used for the new entry. This is simpler to implement than LRU or LFU and achieves equivalent performance.

LZ77 and LZ78 are the two lossless data compression algorithms published in papers by Abraham Lempel and Jacob Ziv in 1977 and 1978. They are also known as LZ1 and LZ2 respectively. These two algorithms form the basis for many variations including LZW, LZSS, LZMA and others. Besides their academic influence, these algorithms formed the basis of several ubiquitous compression schemes, including GIF and the DEFLATE algorithm used in PNG and ZIP.

They are both theoretically dictionary coders. LZ77 maintains a sliding window during compression. This was later shown to be equivalent to the explicit dictionary constructed by LZ78—however, they are only equivalent when the entire data is intended to be decompressed.

Since LZ77 encodes and decodes from a sliding window over previously seen characters, decompression must always start at the beginning of the input. Conceptually, LZ78 decompression could allow random access to the input if the entire dictionary were known in advance. However, in practice the dictionary is created during encoding and decoding by creating a new phrase whenever a token is output

In the technique of Lossy compression, it decreases the bits by recognizing the not required information and by eliminating it. The system of decreasing the size of the file of data is commonly termed as the data-compression, though its formal name is the source-coding that is coding get done at source of data before it gets stored or sent. In these methods few loss of the information is acceptable. Dropping non-essential information from the source of data can save the storage area. The Lossy data-compression methods are aware by the researches on how the people anticipate data in the question. As an example, the human eye is very ensitive to slight variations in the luminance as compare that there are so many variations in the color. The Lossy image compression

technique is used in the digital cameras, to raise the storage ability with the minimal decline of the quality of picture. Similarly in the DVDs which uses the lossy MPEG-2 Video codec technique for the compression of the video. In the lossy audio compression, the techniques of psycho acoustics have been used to eliminate the non-audible or less audible components of signal.

In information technology, lossy compression or irreversible compression is the class of data encoding methods that uses inexact approximations and partial data discarding to represent the content. These techniques are used to reduce data size for storing, handling, and transmitting content. The different versions of the photo of the cat to the right show how higher degrees of approximation create coarser images as more details are removed. This is opposed to lossless data compression (reversible data compression) which does not degrade the data. The amount of data reduction possible using lossy compression is much higher than through lossless techniques.

Well-designed lossy compression technology often reduces file sizes significantly before degradation is noticed by the end-user. Even when noticeable by the user, further data reduction may be desirable (e.g., for real-time communication, to reduce transmission times, or to reduce storage needs).

Lossy compression is most commonly used to compress multimedia data (audio, video, and images), especially in applications such as streaming media and internet telephony. By contrast, lossless compression is typically required for text and data files, such as bank records and text articles. It can be advantageous to make a master lossless file which can then be used to produce additional copies from. This allows one to avoid basing new compressed copies off of a lossy source file, which would yield additional artifacts and further unnecessary information loss.

It is possible to compress many types of digital data in a way that reduces the size of a computer file needed to store it, or the bandwidth needed to transmit it, with no loss of the full information contained in the original file. A picture, for example, is converted to a digital file by considering it to be an array of dots and specifying the color and brightness of each dot. If the picture contains an area of the same color, it can be compressed without loss by saying "200 red dots" instead of "red dot, red dot, ...(197 more times)..., red dot."

The original data contains a certain amount of information, and there is a lower limit to the size of file that can carry all the information. Basic information theory says that there is an absolute limit in reducing the size of this data. When data is compressed, its entropy increases, and it

cannot increase indefinitely. As an intuitive example, most people know that a compressed ZIP file is smaller than the original file, but repeatedly compressing the same file will not reduce the size to nothing. Most compression algorithms can recognize when further compression would be pointless and would in fact increase the size of the data.

In many cases, files or data streams contain more information than is needed for a particular purpose. For example, a picture may have more detail than the eye can distinguish when reproduced at the largest size intended; likewise, an audio file does not need a lot of fine detail during a very loud passage. Developing lossy compression techniques as closely matched to human perception as possible is a complex task. Sometimes the ideal is a file that provides exactly the same perception as the original, with as much digital information as possible removed; other times, perceptible loss of quality is considered a valid trade-off for the reduced data.

The terms 'irreversible' and 'reversible' are preferred over 'lossy' and 'lossless' respectively for some applications, such as medical image compression, to circumvent the negative implications of 'loss'. The type and amount of loss can affect the utility of the images. Artifacts or undesirable effects of compression may be clearly discernible yet the result still useful for the intended purpose. Or lossy compressed images may be 'visually lossless', or in the case of medical images, so-called Diagnostically Acceptable Irreversible Compression (DAIC) may have been applied.

More generally, some forms of lossy compression can be thought of as an application of transform coding – in the case of multimedia data, perceptual coding: it transforms the raw data to a domain that more accurately reflects the information content. For example, rather than expressing a sound file as the amplitude levels over time, one may express it as the frequency spectrum over time, which corresponds more accurately to human audio perception. While data reduction (compression, be it lossy or lossless) is a main goal of transform coding, it also allows other goals: one may represent data more accurately for the original amount of space – for example, in principle, if one starts with an analog or high-resolution digital master, an MP3 file of a given size should provide a better representation than a raw uncompressed audio in WAV or AIFF file of the same size. This is because uncompressed audio can only reduce file size by lowering bit rate or depth, whereas compressing audio can reduce size while maintaining bit rate and depth. This compression becomes a selective loss of the least significant data, rather than losing data across the board. Further, a transform coding may provide a better domain for manipulating or otherwise

editing the data – for example, equalization of audio is most naturally expressed in the frequency domain (boost the bass, for instance) rather than in the raw time domain.

From this point of view, perceptual encoding is not essentially about discarding data, but rather about a better representation of data. Another use is for backward compatibility and graceful degradation: in color television, encoding color via a luminance-chrominance transform domain (such as YUV) means that black-and-white sets display the luminance, while ignoring the color information. Another example is chroma subsampling: the use of color spaces such as YIQ, used in NTSC, allow one to reduce the resolution on the components to accord with human perception – humans have highest resolution for black-and-white (luma), lower resolution for mid-spectrum colors like yellow and green, and lowest for red and blues – thus NTSC displays approximately 350 pixels of luma per scanline, 150 pixels of yellow vs. green, and 50 pixels of blue vs. red, which are proportional to human sensitivity to each component.

Lossy compression formats suffer from generation loss: repeatedly compressing and decompressing the file will cause it to progressively lose quality. This is in contrast with lossless data compression, where data will not be lost via the use of such a procedure. Information-theoretical foundations for lossy data compression are provided by rate-distortion theory. Much like the use of probability in optimal coding theory, rate-distortion theory heavily draws on Bayesian estimation and decision theory in order to model perceptual distortion and even aesthetic judgment.

There are two basic lossy compression schemes:

- In lossy transform codecs, samples of picture or sound are taken, chopped into small segments, transformed into a new basis space, and quantized. The resulting quantized values are then entropy coded.
- In lossy predictive codecs, previous and/or subsequent decoded data is used to predict the current sound sample or image frame. The error between the predicted data and the real data, together with any extra information needed to reproduce the prediction, is then quantized and coded.

3.2 METHODS OF LOSSY IMAGE COMPRESSION

3.2.1 BETTER PORTABLE GRAPHICS

Better Portable Graphics (BPG) is a file format for coding digital images, which was created by programmer Fabrice Bellard in 2014. He has proposed it as a replacement for the JPEG image format as the more compression-efficient alternative in terms of image quality or file size.It is based on the intra-frame encoding of the High Efficiency Video Coding (HEVC) video compression standard.[2] Tests on photographic images in July 2014 found that BPG produced smaller files for a given quality than JPEG, JPEG XR and WebP. its portability, high quality, and low memory requirements, BPG has potential applications in portable handheld and IoT devices, where those properties are particularly important. Current research works on designing and developing more energy-efficient BPG hardware which can then be integrated in portable devices such as digital cameras.

BPG's container format is intended to be more suited to a generic image format than the raw bitstream format used in HEVC (which is otherwise ordinarily used within some other wrapper format, such as the .mp4 file format).BPG supports the color formats known as 4:4:4, 4:2:2, and 4:2:0.[2] Support for a separately coded extra channel is also included for an alpha channel or the fourth channel of a CMYK image. Metadata support is included for Exif, ICC profiles, and XMP.

Color space support is included for YCbCr with ITU-R BT.601, BT.709, and BT.2020 (non-constant luminance) definitions, YCgCo, RGB, CMYK, and grayscale. Support for HEVC's lossy and lossless data compression is included. BPG supports animation

While there is no built-in native support for BPG in any mainstream browsers, websites can still deliver BPG images to all browsers by including a JavaScript library written by Bellard.

3.2.2 CARTESIAN PERCEPTUAL COMPRESSION

Cartesian Perceptual Compression (abbreviated CPC, with filename extension .cpc) is a proprietary image file format. It was designed for high compression of black-and-white raster Document Imaging for archival scans.

CPC is lossy, has no lossless mode, and is restricted to bi-tonal images. The company which controls the patented format claims it is highly effective in the compression of text, black-and-white (halftone) photographs, and line art. The format is intended for use in the web distribution of legal documents, design plans, and geographical plot maps.

Viewing and converting documents in the CPC format currently requires the download of proprietary software. Although viewing CPC documents is free, as is converting CPC images to other formats, conversion to CPC format requires a purchase.

JSTOR, a United States-based online system for archiving academic journals, converted its online archives to CPC in 1997. The CPC files are used to reduce storage requirements for its online collection, but are temporarily converted on their servers to GIF for display, and to PDF for printing. JSTOR still scans to TIFF G4 and considers those files its preservation masters.

3.2.3 DjVu

DjVu is a computer file format designed primarily to store scanned documents, especially those containing a combination of text, line drawings, indexed color images, and photographs. It uses technologies such as image layer separation of text and background/images, progressive loading, arithmetic coding, and lossy compression for bitonal (monochrome) images. This allows high-quality, readable images to be stored in a minimum of space, so that they can be made available on the web.

DjVu has been promoted as providing smaller files than PDF for most scanned documents.[4] The DjVu developers report that color magazine pages compress to 40–70 kB, black-and-white technical papers compress to 15–40 kB, and ancient manuscripts compress to around 100 kB; a satisfactory JPEG image typically requires 500 kB.[5] Like PDF, DjVu can contain an OCR text layer, making it easy to perform copy and paste and text search operations.

Free creators, manipulators, converters, browser plug-ins, and desktop viewers are available.DjVu is supported by a number of multi-format document viewers and e-book reader software on Linux (Okular, Evince) and Windows (SumatraPDF).

The primary usage of the DjVu format has been the electronic distribution of documents with a quality comparable to that of printed documents. As that niche is also the primary usage for PDF, it was inevitable that the two formats would become competitors. It should however be observed that the two formats approach the problem of delivering high resolution documents in very different ways: PDF primarily encodes graphics and text as vectorised data, whereas DjVu primarily encodes them as pixmap images. This means PDF places the burden of rendering the document on the reader, whereas DjVu places that burden on the creator.

During a number of years, significantly overlapping with the period when DjVu was being developed, there were no PDF viewers for free operating systems — a particular stumbling block was the rendering of vectorised fonts, which are essential for combining small file size with high resolution in PDF. Since displaying DjVu was a simpler problem for which free software was available, there were suggestions that the free software movement should employ DjVu instead of PDF for distributing documentation; rendering for creating DjVu is in principle not much different from rendering for a device-specific printer driver, and DjVu can as a last resort be generated from scans of paper media. However when FreeType 2.0 in 2000 began provide rendering of all major vectorised font formats, that specific advantage of DjVu began to erode.

In the 2000s, with the growth of the world wide web and before widespread adoption of broadband, DjVu was often adopted by digital libraries as their format of choice, thanks to its integration with software like Greenstone[10] and the Internet Archive,[11] browser plugins which allowed advanced online browsing, smaller file size for comparable quality of book scans and other image-heavy documents[12] and support for embedding and searching full text from OCR.[13] Some features such as the thumbnail previews were later integrated in the Internet Archive's BookReader[14] and DjVu browsing was deprecated in its favour as around 2015 some major browsers stopped supporting Java applets and DjVu plugins with them.

DjVu divides a single image into many different images, then compresses them separately. To create a DjVu file, the initial image is first separated into three images: a background image, a foreground image, and a mask image. The background and foreground images are typically lower-resolution color images (e.g., 100 dpi); the mask image is a high-resolution bilevel image (e.g., 300 dpi) and is typically where the text is stored. The background and foreground images are then compressed using a wavelet-based compression algorithm named IW44. The mask image is compressed using a method called JB2 (similar to JBIG2). The JB2 encoding method identifies

nearly identical shapes on the page, such as multiple occurrences of a particular character in a given font, style, and size. It compresses the bitmap of each unique shape separately, and then encodes the locations where each shape appears on the page. Thus, instead of compressing a letter "e" in a given font multiple times, it compresses the letter "e" once (as a compressed bit image) and then records every place on the page it occurs.

Optionally, these shapes may be mapped to UTF-8 codes (either by hand or potentially by a text recognition system) and stored in the DjVu file. If this mapping exists, it is possible to select and copy text.Since JBIG2 was based on JB2, both compression methods have the same problems when performing lossy compression. Numbers may be substituted with similarly looking numbers (such as replacing 6 with 8) if the text was scanned at a low resolution prior to lossy compression.

Despite its advantages, DjVu is not widely supported by scanning and viewing software.[23] While viewers can be downloaded, opening DjVu files is not implemented in most operating systems by default. In 2002, the DjVu file format was chosen by the Internet Archive as a format in which its Million Book Project provides scanned public-domain books online (along with TIFF and PDF). In February 2016, the IA announced that DjVu would no longer be used for new uploads.

3.2.4 FRACTAL COMPRESSION

Fractal compression is a lossy compression method for digital images, based on fractals. The method is best suited for textures and natural images, relying on the fact that parts of an image often resemble other parts of the same image.[citation needed] Fractal algorithms convert these parts into mathematical data called "fractal codes" which are used to recreate the encoded image.

Fractal image representation may be described mathematically as an iterated function system (IFS).We begin with the representation of a binary image, where the image may be thought of as a subset of . An IFS is a set of contraction mappings $f1,...,fN$,

A challenging problem of ongoing research in fractal image representation is how to choose the $f1,...,fN$ such that its fixed point approximates the input image, and how to do this efficiently.

A simple approach[1] for doing so is the following partitioned iterated function system (PIFS):

1. Partition the image domain into range blocks Ri of size s×s.
2. For each Ri, search the image to find a block Di of size 2s×2s that is very similar to Ri.
3. Select the mapping functions such that H(Di) = Ri for each i.

In the second step, it is important to find a similar block so that the IFS accurately represents the input image, so a sufficient number of candidate blocks for Di need to be considered. On the other hand, a large search considering many blocks is computationally costly. This bottleneck of searching for similar blocks is why PIFS fractal encoding is much slower than for example DCT and wavelet based image representation.

The initial square partitioning and brute-force search algorithm presented by Jacquin provides a starting point for further research and extensions in many possible directions -- different ways of partitioning the image into range blocks of various sizes and shapes; fast techniques for quickly finding a close-enough matching domain block for each range block rather than brute-force searching, such as fast motion estimation algorithms; different ways of encoding the mapping from the domain block to the range block; etc.

Other researchers attempt to find algorithms to automatically encode an arbitrary image as RIFS (recurrent iterated function systems) or global IFS, rather than PIFS; and algorithms for fractal video compression including motion compensation and three dimensional iterated function systems. Fractal image compression has many similarities to vector quantization image compression

With fractal compression, encoding is extremely computationally expensive because of the search used to find the self-similarities. Decoding, however, is quite fast. While this asymmetry has so far made it impractical for real time applications, when video is archived for distribution from disk storage or file downloads fractal compression becomes more competitive.

At common compression ratios, up to about 50:1, Fractal compression provides similar results to DCT-based algorithms such as JPEG At high compression ratios fractal compression may offer superior quality. For satellite imagery, ratios of over 170:1have been achieved with acceptable results. Fractal video compression ratios of 25:1–244:1 have been achieved in

reasonable compression times (2.4 to 66 sec/frame).Compression efficiency increases with higher image complexity and color depth, compared to simple grayscale images.

An inherent feature of fractal compression is that images become resolution independent after being converted to fractal code. This is because the iterated function systems in the compressed file scale indefinitely. This indefinite scaling property of a fractal is known as "fractal scaling". Michael Barnsley led development of fractal compression in 1987, and was granted several patents on the technology.[16] The most widely known practical fractal compression algorithm was invented by Barnsley and Alan Sloan. Barnsley's graduate student Arnaud Jacquin implemented the first automatic algorithm in software in 1992.[17][18] All methods are based on the fractal transform using iterated function systems. Michael Barnsley and Alan Sloan formed Iterated Systems Inc.in 1987 which was granted over 20 additional patents related to fractal compression.

A major breakthrough for Iterated Systems Inc. was the automatic fractal transform process which eliminated the need for human intervention during compression as was the case in early experimentation with fractal compression technology. In 1992, Iterated Systems Inc. received a US$2.1 million government grant[20] to develop a prototype digital image storage and decompression chip using fractal transform image compression technology.Fractal image compression has been used in a number of commercial applications: on One Software, developed under license from Iterated Systems Inc., Genuine Fractals 5[21] which is a Photoshop plugin capable of saving files in compressed FIF (Fractal Image Format). To date the most successful use of still fractal image compression is by Microsoft in its Encarta multimedia encyclopedia,[22] also under license.

During the 1990s Iterated Systems Inc. and its partners expended considerable resources to bring fractal compression to video. While compression results were promising, computer hardware of that time lacked the processing power for fractal video compression to be practical beyond a few select usages. Up to 15 hours were required to compress a single minute of video. ClearVideo – also known as RealVideo (Fractal) – and SoftVideo were early fractal video compression products. ClearFusion was Iterated's freely distributed streaming video plugin for web browsers. In 1994 SoftVideo was licensed to Spectrum Holobyte for use in its CD-ROM games including Falcon Gold and Star Trek: The Next Generation A Final Unity.

ICER is a wavelet-based image compression file format used by the NASA Mars Rovers. ICER has both lossy and lossless compression modes. The Mars Exploration Rovers “Spirit” (MER-A) and “Opportunity” (MER-B) both use ICER. Onboard image compression is used extensively to make best use of the downlink resources. The Mars Science Lab supports the use of ICER for its navigation cameras (but all other cameras use other file formats).

Most of the MER images are compressed with the ICER image compression software. The remaining MER images that are compressed make use of modified Low Complexity Lossless Compression (LOCO) software, a lossless sub mode of ICER.ICER is a wavelet-based image compressor that allows for a graceful trade-off between the amount of compression (expressed in terms of compressed data volume in bits/pixel) and the resulting degradation in image quality (distortion). ICER has some similarities to JPEG2000, with respect to select wavelet operations. The development of ICER was driven by the desire to achieve high compression performance while meeting the specialized needs of deep space applications.

To control the image quality and amount of compression in ICER, the user specifies a byte quota (the nominal number of bytes to be used to store the compressed image) and a quality level parameter (which is essentially a quality goal).ICER attempts to produce a compressed image that meets the quality level using as few compressed bytes as possible. ICER stops producing compressed bytes once the quality level or byte quota is met, whichever comes first. This arrangement provides added flexibility compared to compressors (like the JPEG compressor used on Mars Pathfinder) that provide only a single parameter to control image quality. Using ICER, when the primary concern is the bandwidth available to transmit the compressed image, one can set the quality goal to lossless and the given byte quota will determine the amount of compression obtained.At the other extreme—when the only important consideration is a minimum acceptable image quality it is possible to specify sufficiently large byte quota and the amount of compression will be determined by the quality level specified.

To achieve error containment, ICER produces the compressed bitstream in separate pieces or segments that can be decoded independently. These segments represent rectangular regions of the original image, but are defined in the transform domain. If the image were partitioned directly and the wavelet transform separately applied to each segment, under lossy compression the boundaries between segments would tend to be noticeable in the reconstructed image even when no compressed data is lost. Since ICER provides a facility for automated flexibility in choosing

the number of segments, compression effectiveness can be traded against packet loss protection, thereby accommodating different channel error rates .Note also that more segments are not always bad for compression effectiveness: many images are most effectively compressed using 4 to 6 segments (for megapixel images) because disparate regions of the image end up in different segments.

JPEG2000 and ICER have many important internal differences

- JPEG 2000 uses floating point math, where ICER uses only integer math. Thus ICER will have good performance on integer only CPUs like the T414 Transputer, whereas JPEG 2000 will not perform as well as it is forced into floating point emulation.
- ICER reverts to a separate internal LOCO (Low Complexity Lossless Compression) compressor for lossless image compression.
- JPEG 2000 implements a low complexity symmetrical wavelet lossless compressor, but ICER uses an integer only non-wavelet lossless compressor.
- ICER and JPEG 2000 encode color spaces differently.
- ICER in its current form does compress monochrome images better than colour images due to its origins as an internal NASA Deep Space Network file format.
- ICER is subject to less than 1% overshoot when byte and quality quotas are in effect. On the other hand, JPEG2000 codecs are typically designed never to overshoot their byte quotas.

ICER was created for low end 32 bit CPUs (essentially embedded computers) on spacecraft. It was finally used for the Mars Exploration Rovers. It has never been used for any real time application, only near real time.

JPEG2000 has been used by many image processing applications in near real time and real-time (Digital Cinema, Broadcast). Main advantages of the codec is that it is License free (JPEG2000 PART1). The JPEG committee has stated: “It has always been a strong goal of the JPEG committee that its standards should be implementable in their baseline form without payment of royalty and license fees. Agreements have been reached with over 20 large organizations holding many patents in this area to allow use of their intellectual property in connection with the standard without payment of license fees or royalties”. Hewlett-Packard's Remote Graphics Software uses a video codec called HP3 (codec) which claims to derive from Mars Rover compression - this could be a real-time implementation of ICER.

ICER offers a new mode called Spectral + ICER that makes possible lower rate distortion levels (aka grey level errors) with ICER images. This mode is only so far being used with the Mars Pathfinders, but may see wider implementation in the ICER standard
Error-containment segments in ICER-3D are defined spatially (in the wavelet transform domain) similarly to JPEG 2000. The wavelet-transformed data are partitioned in much the same way as in ICER, except that in ICER-3D the segments extend through all spectral bands. Error-containment segments in ICER and ICER-3D are defined using an unmodified form of the ICER rectangle partitioning algorithm.

In ICER-3D, contexts are defined based on two neighboring coefficients in the spectral dimension and no neighboring coefficients in the same spatial plane. This contrasts with the context modeling scheme used by ICER, which makes use of previously encoded information from spatially neighboring coefficients.ICER-3D exploits 3D data dependencies in part by using a 3-D wavelet decomposition. The particular decomposition used by ICER-3D includes additional spatial decomposition steps compared to a 3-D Mallat decomposition. This modified decomposition provides benefits in the form of quantitatively improved rate-distortion performance and in the elimination of spectral ringing artifacts.

ICER-3D takes advantage of the correlation properties of wavelet-transformed hyperspectral data by using a context modeling procedure that emphasizes spectral (rather than spatial) dependencies in the wavelet-transformed data. This provides a significant gain over the

alternative spatial context modeler considered .ICER-3D also inherits most of the important features of ICER, including progressive compression, the ability to perform lossless and lossy compression, and an effective error-containment scheme to limit the effects of data loss on the deep-space channel..

3.2.5 PROGRESSIVE GRAPHICS FILE

PGF (Progressive Graphics File) is a wavelet-based bitmapped image format that employs lossless and lossy data compression. PGF was created to improve upon and replace the JPEG format. It was developed at the same time as JPEG 2000 but with a focus on speed over compression ratio. PGF can operate at higher compression ratios without taking more encoding/decoding time and without generating the characteristic "blocky and blurry" artifacts of the original DCT-based JPEG standard.[2] It also allows more sophisticated progressive downloads.

PGF claims to achieve an improved compression quality over JPEG adding or improving features such as scalability. Its compression performance is similar to the original JPEG standard. Very low and very high compression rates (including lossless compression) are also supported in PGF. The ability of the design to handle a very large range of effective bit rates is one of the strengths of PGF. For example, to reduce the number of bits for a picture below a certain amount, the advisable thing to do with the first JPEG standard is to reduce the resolution of the input image before encoding it — something that is ordinarily not necessary for that purpose when using PGF because of its wavelet scalability properties.

The PGF process chain contains the following four steps:

1. Color space transform (in case of color images)
2. Discrete Wavelet Transform
3. Quantization (in case of lossy data compression)
4. Hierarchical bit-plane run-length encoding

3.2.6 S3 TEXTURE COMPRESSION

S3 Texture Compression (S3TC) (sometimes also called DXTn or DXTC) is a group of related lossy texture compression algorithms originally developed by Iourcha et al. of S3 Graphics, Ltd. for use in their Savage 3D computer graphics accelerator. The method of compression is strikingly similar to the previously published Color Cell Compression,which is in turn an adaptation of Block Truncation Coding published in the late 1970s. Unlike some image compression algorithms (e.g. JPEG), S3TC's fixed-rate data compression coupled with the single memory access (cf. Color Cell Compression and some VQ-based schemes) made it well-suited for use in compressing textures in hardware-accelerated 3D computer graphics. Its subsequent inclusion in Microsoft's DirectX 6.0 and OpenGL 1.3 (via the GL_EXT_texture_compression_s3tc extension) led to widespread adoption of the technology among hardware and software makers. While S3 Graphics is no longer a competitor in the graphics accelerator market, license fees have been levied and collected for the use of S3TC technology until October 2017, for example in game consoles and graphics cards. The wide use of S3TC has led to a de facto requirement for OpenGL drivers to support it, but the patent-encumbered status of S3TC presented a major obstacle to open source implementations,while implementation approaches which tried to avoid the patented parts existed.

Like many modern image compression algorithms, S3TC only specifies the method used to decompress images, allowing implementers to design the compression algorithm to suit their specific needs, although the patent still covers compression algorithms. The nVidia GeForce 256 through to GeForce 4 cards also used 16-bit interpolation to render DXT1 textures, which resulted in banding when unpacking textures with color gradients.

There are five variations of the S3TC algorithm (named DXT1 through DXT5, referring to the FourCC code assigned by Microsoft to each format), each designed for specific types of image data. All convert a 4×4 block of pixels to a 64-bit or 128-bit quantity, resulting in compression ratios of 6:1 with 24-bit RGB input data or 4:1 with 32-bit RGBA input data. S3TC is a lossy compression algorithm, resulting in image quality degradation, an effect which is minimized by the ability to increase texture resolutions while maintaining the same memory requirements. Hand-drawn cartoon-like images do not compress well, nor do normal map data, both of which usually generate artifacts. ATI's 3Dc compression algorithm is a modification of DXT5 designed to

overcome S3TC's shortcomings with regard to normal maps. id Software worked around the normalmap compression issues in Doom 3 by moving the red component into the alpha channel before compression and moving it back during rendering in the pixel shader.

3.2.7 WebP

WebP is an image format employing both lossy[6] and lossless compression. It is currently developed by Google, based on technology acquired with the purchase of On2 Technologies.As a derivative of the VP8 video format, it is a sister project to the WebM multimedia container format.WebP-related software is released under a BSD license.The format was first announced on 30 September 2010 as a new open standard for lossy compressed true-color graphics on the web, producing smaller files of comparable image quality to the older JPEG scheme. On October 3, 2011 Google announced WebP support for animation, ICC profile, XMP metadata, and tiling (compositing very large images from maximum 16384×16384 tiles).

On 18 November 2011 Google began to experiment with lossless compression and support for transparency (alpha channel) in both lossless and lossy modes; support has been enabled by default in libwebp 0.2.0 (16 August 2012).[12][13] According to Google's measurements, a conversion from PNG to WebP results in a 45% reduction in file size when starting with PNGs found on the web, and a 28% reduction compared to PNGs that are recompressed with pngcrush and PNGOUT.

WebP's lossy compression algorithm is based on the intra-frame coding of the VP8 video format and the Resource Interchange File Format (RIFF) as a container format. As such, it is a block-based transformation scheme with eight bits of color depth and a luminance-chrominance model with chroma subsampling by a ratio of 1:2 (YCbCr 4:2:0) Without further content, the mandatory RIFF container has an overhead of only twenty bytes, though it can also hold additional metadata. The side length of WebP images is limited to 16383 pixels WebP is based on block prediction. Each block is predicted on the values from the three blocks above it and from one block to the left of it (block decoding is done in raster-scan order: left to right and top to bottom). There are four basic modes of block prediction: horizontal, vertical, DC (one color), and TrueMotion. Mispredicted data and non-predicted blocks are compressed in a 4×4 pixel sub-block with a discrete cosine transform or a Walsh–Hadamard transform. Both transforms are done with fixed-point arithmetic to avoid rounding errors. The output is compressed with entropy encoding.WebP also has explicit support for parallel decoding.

The reference implementation consists of converter software in the form of a command-line program for Linux (cwebp) and a programming library for the decoding, the same as for WebM. The open source community quickly managed to port the converter to other platforms, such as Windows. WebP's lossless compression uses advanced techniques such as dedicated entropy codes for different color channels, exploiting 2D locality of backward reference distances and a color cache of recently used colors. This complements basic techniques such as dictionary coding, Huffman coding and color indexing transform.

Google has proposed using WebP for animated images as an alternative to the popular GIF format, citing the advantages of 24-bit color with transparency, combining frames with lossy and lossless compression in the same animation, and as well as support for seeking to specific frames.[19] Google reports a 64% reduction in file size for images converted from animated GIFs to lossy WebP, and a 19% reduction when converted to lossless WebP.

CHAPTER 4

IMAGE COMPRESSION USING DISCRETE COSINE TRANSFORMATION

A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. DCTs are important to numerous applications in science and engineering, from lossy compression of audio (e.g. MP3) and images (e.g. JPEG) (where small high-frequency components can be discarded), to spectral methods for the numerical solution of partial differential equations. The use of cosine rather than sine functions is critical for compression, since it turns out (as described below) that fewer cosine functions are needed to approximate a typical signal, whereas for differential equations the cosines express a particular choice of boundary conditions

In particular, a DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. The DCTs are generally related to Fourier Series coefficients of a periodically and symmetrically extended sequence whereas DFTs are related to Fourier Series coefficients of a periodically extended sequence. DCTs are equivalent to DFTs of roughly twice the length, operating on real data with even symmetry (since the Fourier transform of a real and even function is real and even), whereas in some variants the input and/or output data are shifted by half a sample.

The most common variant of discrete cosine transform is the type-II DCT, which is often called simply the DCT.Its inverse, the type-III DCT, is correspondingly often called simply "the inverse DCT" or "the IDCT". Two related transforms are the discrete sine transform (DST), which is equivalent to a DFT of real and odd functions, These are developed to reduce the computational complexity of implementing

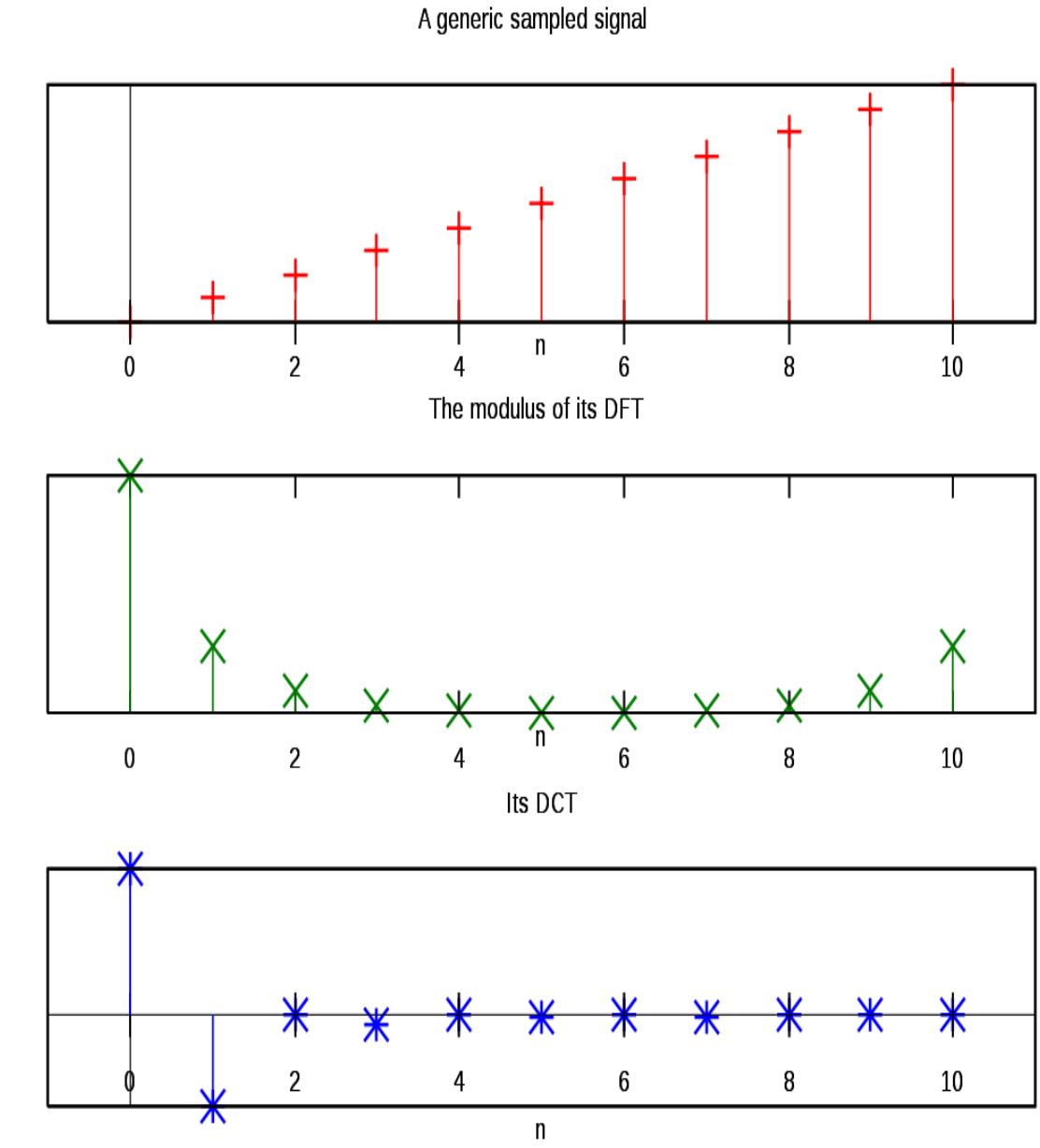


Fig (4.1.1) DCT COMPARED TO DFT OF A INPUT SIGNAL

4.1 APPLICATION OF DCT

The DCT, and in particular the DCT-II, is often used in signal and image processing, especially for lossy compression, because it has a strong "energy compaction" property: in typical applications, most of the signal information tends to be concentrated in a few low-frequency components of the DCT. For strongly correlated Markov processes, the DCT can approach the compaction efficiency of the Karhunen-Loève transform (which is optimal in the decorrelation sense). As explained below, this stems from the boundary conditions implicit in the cosine functions.A related transform, the modified discrete cosine transform, or MDCT (based on the DCT-IV), is used in AAC, Vorbis, WMA, and MP3 audio compression.DCTs are also widely employed in solving partial differential equations by spectral methods, where the different variants of the DCT correspond to slightly different even/odd boundary conditions at the two ends of the array.DCTs are also closely related to Chebyshev polynomials, and fast DCT algorithms (below) are used in Chebyshev approximation of arbitrary functions by series of Chebyshev polynomials, for example in Clenshaw–Curtis quadrature.

4.2 JPEG

JPEG stands for the Joint Photographic Experts Group, a standards committee that had its origins within the International Standard Organization (ISO).JPEG provides a compression method that is capable of compressing continuous-tone image data with a pixel depth of 6 to 24 bits with reasonable speed and efficiency.JPEG may be adjusted to produce very small, compressed images that are of relatively poor quality in appearance but still suitable for many applications. Conversely, JPEG is capable of producing very high-quality compressed images that are still far smaller than the original uncompressed data.

JPEG is primarily a lossy method of compression.JPEG was designed specifically to discard information that the human eye cannot easily see. Slight changes in color are not perceived well by the human eye, while slight changes in intensity (light and dark) are. Therefore JPEG's lossy encoding tends to be more frugal with the gray-scale part of an image and to be more frivolous with the color[21].DCT separates images into parts of different frequencies where less important frequencies are discarded through quantization and important frequencies are used to retrieve the image during decompression. Compared to other input dependent transforms, DCT has many advantages: (1) It has been implemented in single integrated circuit; (2) It has the ability

The forward 2D_DCT transformation is given by the following equation:

$$C(u,v)=D(u)D(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos[(2x+1)u\pi/2N]\cos[(2y+1)v\pi/2N]$$

Where,u,v=0,1,2,3,…….,N-1

The inverse 2D-DCT transformation is given by the following equation:

$$f(x,y)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} D(u)D(v)D(u,v)\cos[(2x+1)u\pi/2N]x\cos(2y+1)v\pi/2N]$$

where

D(u)=(1/N) ^1/2

for u=0

D(u)=2(/N)^1/2 for u=1,2,3……..,(N-1)

4.3 JPEG Process:

- Original image is divided into blocks of 8 x 8.
- Pixel values of a black and white image range from 0-255 but DCT is designed to work on pixel values ranging from -128 to 127 .Therefore each block is modified to work in the range.
- Equation(1) is used to calculate DCT matrix.
- DCT is applied to each block by multiplying the modified block with DCT matrix on the left and transpose of DCT matrix on its right.
- Each block is then compressed through quantization.
- Quantized matrix is then entropy encoded.
- Compressed image is reconstructed through reverse process.
- Inverse DCT is used for decompression[11].

4.4 Quantization

Quantization is achieved by compressing a range of values to a single quantum value. When the number of discrete symbols in a given stream is reduced, the stream becomes more compressible .A quantization matrix is used in combination with a DCT coefficient matrix to carry out transformation. Quantization is the step where most of the compression takes place. DCT really does not compress the image because it is almost lossless. Quantization makes use of the fact that higher frequency components are less important than low frequency components. It allows varying levels of image compression and quality through selection of specific quantization matrices. Thus quality levels ranging from 1 to 100 can be selected, where 1 gives the poorest image quality and highest compression, while 100 gives the best quality and lowest compression. As a result quality to compression ratio can be selected to meet different needs.

JPEG committee suggests matrix with quality level 50 as standard matrix. For obtaining quantization matrices with other quality levels, scalar multiplications of standard quantization matrix are used. Quantization is achieved by dividing transformed image matrix by the quantization matrix used. Values of the resultant matrix are then rounded off. In the resultant matrix coefficients situated near the upper left corner have lower frequencies .Human eye is more sensitive to lower frequencies .Higher frequencies are discarded. Lower frequencies are used to reconstruct the image.

4.5 Entropy Encoding

After quantization, most of the high frequency coefficients are zeros. To exploit the number of zeros, a zig-zag scan of the matrix is used yielding to long string of zeros. Once a block has been converted to a spectrum and quantized, the JPEG compression algorithm then takes the result and converts it into a one dimensional linear array, or vector of 64 values, performing a zig-zag scan by selecting the elements in the numerical order indicated by the numbers in the grid below:

	0	1	2	3	4	5	6	7
0:	0	1	5	6	14	15	27	28
1:	2	4	7	13	16	26	29	42
2:	3	8	12	17	25	30	41	43
3:	9	11	18	24	31	40	44	53
4:	10	19	23	32	39	45	52	5
5:	20	22	33	38	46	51	55	60

This places the elements of the coefficient block in a reasonable order of increasing frequency. Since the higher frequencies are more likely to be zero after quantization, this tends to group zero values in the high end of the vector

Huffman coding:The basic idea in Huffman coding is to assign short codewords to those input blocks with high probabilities and long codewords to those with low probabilities.A Huffman code is designed by merging together the two least probable characters, and repeating this process until there is only one character remaining. A code tree is thus generated and the Huffman code is obtained from the labeling of the code tree.

4.6 Results and Discussions:

Results obtained after performing DCT of various orders on original images are shown.Fig(3.4.1) shows original images.Images obtained after applying 8 x 8 DCT are as shown in Fig(3.4.2) whereas Fig(3.4.4) shows image obtained for same original image after applying 4 x 4 DCT.Similarly Fig(3.4.6) and Fig(3.4.8) are obtained after applying 8 x 8 DCT and 4 x 4 DCT of the image shown in Fig(3.4.5).Fig(3.4.9) shows the original Lena image.Fig(3.4.10) to Fig(3.4.14) show compressed images for the original Lena image after taking various number of coefficients for quantization.As the number of coefficients increases quality of the image decreases whereas compression ratio continues to increase.Fig(3.4.15) shows that SNR value increases with number of coefficients.

4.7 DCT RESULTS:

Fig (4.1) DCT INPUT FILE

Fig (4.2) DCT COMPRESSED OUTPUT FILE

I/P SIZE	592 KB
O/P SIZE	26 KB

CHAPTER 5

IMAGE COMPRESSION USING DISCRETE WAVELET TRANSFORMATION

Wavelet Transform has become an important method for image compression. Wavelet based coding provides substantial improvement in picture quality at high compression ratios mainly due to better energy compaction property of wavelet transforms. Wavelet transform partitions a signal into a set of functions called wavelets. Wavelets are obtained from a single prototype wavelet called mother wavelet by dilations and shifting. The wavelet transform is computed separately for different segments of the time-domain signal at different frequencies.

5.1 SUBBAND CODING:

A signal is passed through a series of filters to calculate DWT. Procedure starts by passing this signal sequence through a half band digital low pass filter with impulse response h(n).Filtering of a signal is numerically equal to convolution of the tile signal with impulse response of the filter.

$$x[n]*h[n]= \sum_{=-\infty}^{\infty} x[k].h[n-k]$$

A half band low pass filter removes all frequencies that are above half of the highest frequency in the tile signal. Then the signal is passed through high pass filter. The two filters are related to each other as

$$h[L-1-n]=(-1)^n g(n)$$

Filters satisfying this condition are known as quadrature mirror filters. After filtering half of the samples can be eliminated since the signal now has the highest frequency as half of the original frequency. The signal can therefore be subsampled by 2,simply by discarding every other sample. This constitutes 1 level of decomposition and can mathematically be expressed as

$$Y1[n]=\sum_{=-\infty}^{\infty} x[k]h[2n-k]$$

$$Y2[n]=\sum_{=-\infty}^{\infty} x[k]g[2n+1-k]$$

where y1[n] and y2[n] are the outputs of low pass and high pass filters,respectively after subsampling by 2.

This decomposition halves the time resolution since only half the number of sample now characterizes the whole signal.Frequency resolution has doubled because each output has half the frequency band of the input.This process is called as sub band coding.It can be repeated further to increase the frequency resolution as shown by the filter bank.

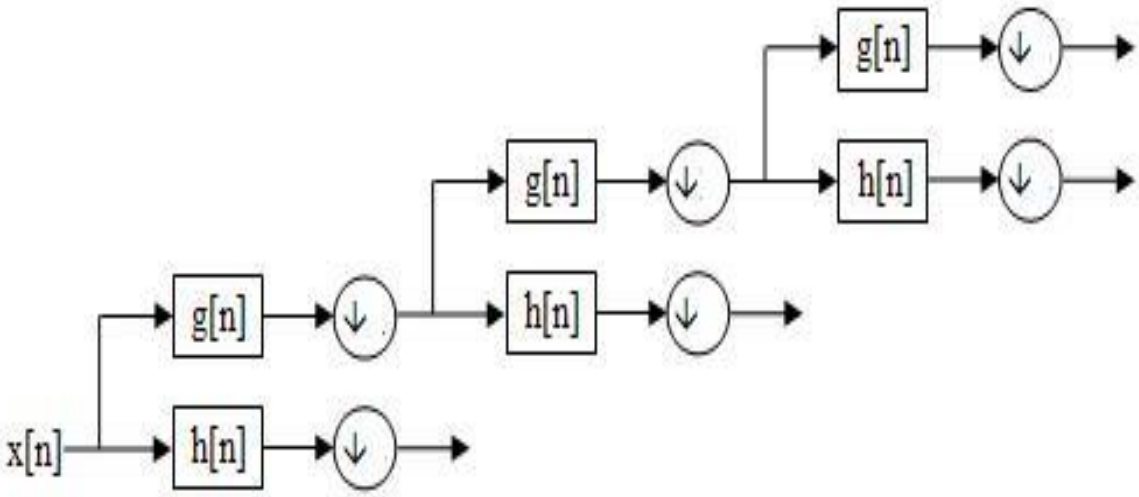


Fig (5.1) Block Diagram of Discrete Wavelet Transform

5.2 Compression steps:

1. .Digitize the source image into a signal s, which is a string of numbers.
2. .Decompose the signal into a sequence of wavelet coefficients w.
3. .Use threshold to modify the wavelet coefficients from w to w'.
4. .Use quantization to convert w' to a sequence q.
5. .Entropy encoding is applied to convert q into a sequence e.

5.3 Digitation

The image is digitized first. The digitized image can be characterized by its intensity levels, or scales of gray which range from 0(black) to 255(white), and its resolution, or how many pixels per square inch

5.4 Thresholding

In certain signals, many of the wavelet coefficients are close or equal to zero. Through threshold these coefficients are modified so that the sequence of wavelet coefficients contains long strings of zeros.

In hard threshold ,a threshold is selected. Any wavelet whose absolute value falls below the tolerance is set to zero with the goal to introduce many zeros without losing a great amount of detail.

5.5 Quantization

Quantization converts a sequence of floating numbers w' to a sequence of integers q. The simplest form is to round to the nearest integer. Another method is to multiply each number in w' by a constant k, and then round to the nearest integer. Quantization is called lossy because it introduces error into the process, since the conversion of w' to q is not one to one function

5.6 Entropy encoding

With this method, a integer sequence q is changed into a shorter sequence with the numbers in e being 8 bit integers The conversion is made by an entropy encoding table. Strings of zeros are coded by numbers 1 through 100,105 and 106,while the non-zero integers in q are coded by 101 through 104 and 107 through 254.

5.7 DWT Results:

Results obtained with the Mat Lab code[20]are shown below. Fig (4.3.1) shows original Lena image. Fig(4.3.2) to Fig(4.3.4) show compressed images for various threshold values. As threshold value increases blurring of image continues to increase.

Fig (5.2) Original Input Image

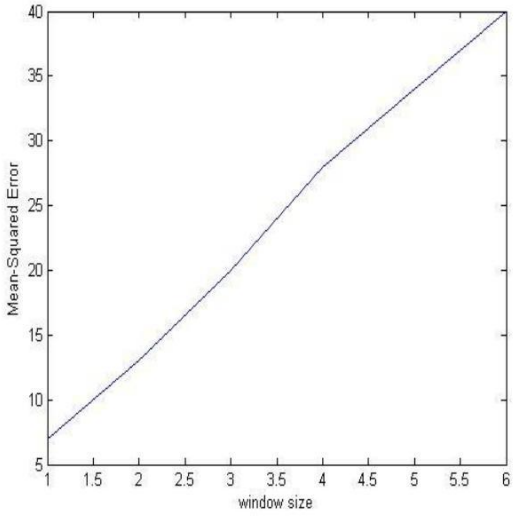
Fig (5.3) DWT Compressed Output Image

PARAMETERS OF INPUT AND OUTPUT FILES

PARAMETER	VALUE
I/P SIZE	592 KB
O/P SIZE	33.89 KB
COMPRESSION RATIO	1
REDUNANCY	-6.66134 e
PSNR	40.32

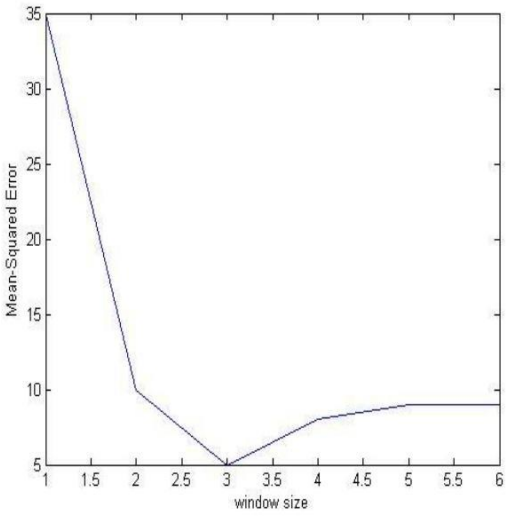
5.8 Comparisons of results for DCT and DWT based on various performance parameters :

Mean Squared Error (MSE) is defined as the square of differences in the pixel values between the corresponding pixels of the two images. Graph of Fig(4.4.1) shows that for DCT based image compression ,as the window size increases MSE increases proportionately whereas for DWT based image compression Fig(4.4.2) shows that MSE first decreases with increase in window size and then starts to increase slowly with finally attaining a constant value. Fig(4.4.3) and Fig(4.4.4) plot show required for compressing image with change in window size for DCT and DWT respectively. Fig(4.4.5) and Fig(4.4.6) indicate compression ratio with change in window size for DCT and DWT based image compression techniques respectively. Compression increases with increase in window size for DCT and decreases with increase in window size for DWT



Fig(5.3)

Mean Squared Error vs. window size for DCT



Fig(5.4)

Mean Squared Error vs. window size for DWT

CHAPTER 6

MATLAB AND GUIDE (Visuals)

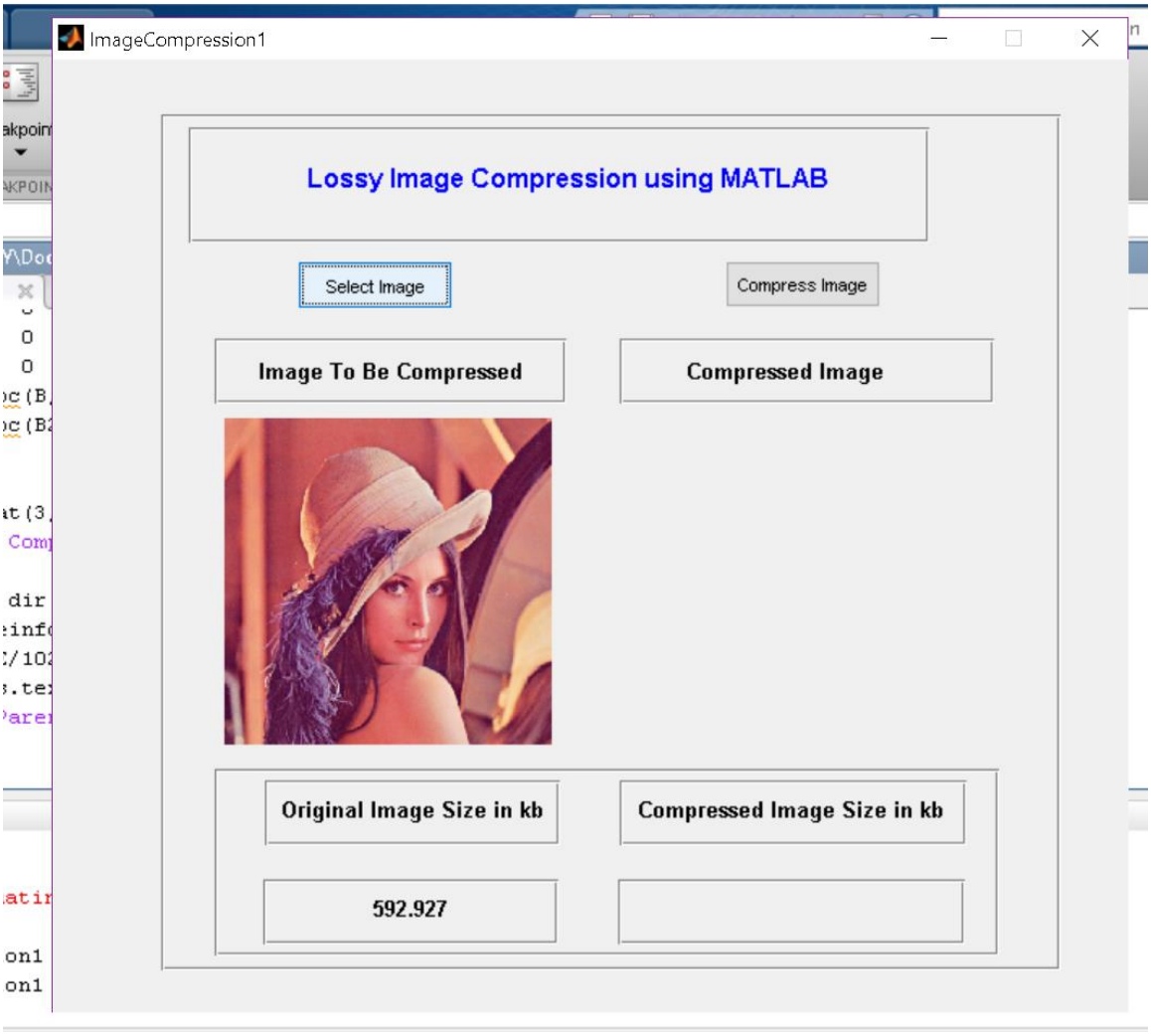


Fig (6.1) DCT GUIDE USER INTERFACE

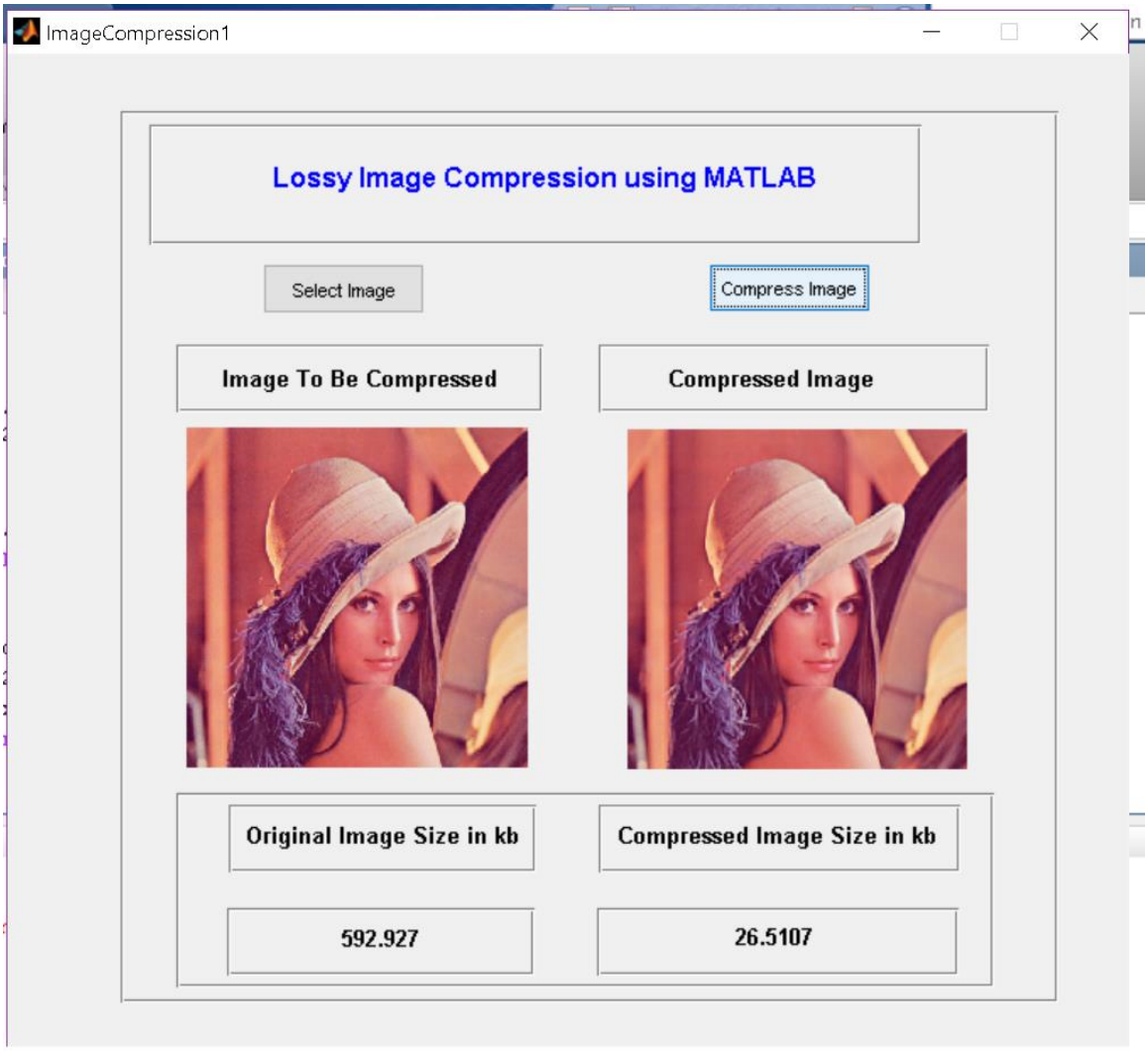


Fig (6.2) DCT I/P AND O/P COMPARISION

Conclusions:

In the thesis image compression techniques using DCT and DWT were implemented.

DCT is used for transformation in JPEG standard. DCT performs efficiently at medium bit rates .Disadvantage with DCT is that only spatial correlation of the pixels inside the single 2-D block is considered and the correlation from the pixels of the neighboring blocks is neglected .Blocks cannot be decorrelated at their boundaries using DCT.

DWT is used as basis for transformation in JPEG 2000 standard. DWT provides high quality compression at low bit rates. The use of larger DWT basis functions or wavelet filters produces blurring near edges in images.

DWT performs better than DCT in the context that it avoids blocking artifacts which degrade reconstructed images. However DWT provides lower quality than JPEG at low compression rates. DWT requires longer compression time.

CODE

DCT in Mat Lab and Guide

```
function varargout = ImageCompression1(varargin)
% IMAGECOMPRESSION1 MATLAB code for ImageCompression1.fig
%      IMAGECOMPRESSION1, by itself, creates a new
IMAGECOMPRESSION1 or raises the existing
%      singleton*.
%
%      H = IMAGECOMPRESSION1 returns the handle to a new
IMAGECOMPRESSION1 or the handle to
%      the existing singleton*.
%
%      IMAGECOMPRESSION1('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in IMAGECOMPRESSION1.M with the
given input arguments.
%
%      IMAGECOMPRESSION1('Property','Value',...) creates a new
IMAGECOMPRESSION1 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before ImageCompression1_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to ImageCompression1_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
```

```
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ImageCompression1

% Last Modified by GUIDE v2.5 15-Oct-2014 22:20:56

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ImageCompression1_OpeningFcn, ...
                   'gui_OutputFcn',  @ImageCompression1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before ImageCompression1 is made visible.
function ImageCompression1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ImageCompression1 (see VARARGIN)

% Choose default command line output for ImageCompression1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
guidata(hObject, handles);
set(handles.axes1,'visible','off')
set(handles.axes2,'visible','off')
axis off
axis off
% UIWAIT makes ImageCompression1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = ImageCompression1_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global file_name;
%guidata(hObject,handles)
file_name=uigetfile({'*.bmp;*.jpg;*.png;*.tiff;';'*.*'},'Select an Image File');
fileinfo = dir(file_name);
SIZE = fileinfo.bytes;
Size = SIZE/1024;
set(handles.text7,'string',Size);
imshow(file_name,'Parent', handles.axes1)

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global file_name;
if(~ischar(file_name))
  errordlg('Please select Images first');
else
   I1 = imread(file_name);
I = I1(:,:,1);

I = im2double(I);
T = dctmtx(8);
alfa=I1(1:8,1:8,1);
disp(alfa);
```

```
B = blkproc(I,[8 8],'P1*x*P2',T,T');

mask  = [1   1   1   1   0   0   0   0
      1   1   1   0   0   0   0   0
      1   1   0   0   0   0   0   0
      1   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0];
B2 = blkproc(B,[8 8],'P1.*x',mask);
I2 = blkproc(B2,[8 8],'P1*x*P2',T',T);

I = I1(:,:,2);
I = im2double(I);
T = dctmtx(8);
B = blkproc(I,[8 8],'P1*x*P2',T,T');
mask = [1   1   1   1   0   0   0   0
      1   1   1     0   0   0   0
      1   1   0   0   0   0   0   0
      1   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0
      0   0   0   0   0   0   0   0];
B2 = blkproc(B,[8 8],'P1.*x',mask);
I3 = blkproc(B2,[8 8],'P1*x*P2',T',T);


I = I1(:,:,3);
```

```
I = im2double(I);
T = dctmtx(8);
B = blkproc(I,[8 8],'P1*x*P2',T,T');
mask = [1   1   1   1   0   0   0   0
        1   1   1   0   0   0   0   0
        1   1   0   0   0   0   0   0
        1   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0
        0   0   0   0   0   0   0   0];
B2 = blkproc(B,[8 8],'P1.*x',mask);
I4 = blkproc(B2,[8 8],'P1*x*P2',T',T);


L(:,:,:)=cat(3,I2, I3, I4);
imwrite(L,'CompressedColourImage.jpg');

fileinfo = dir('CompressedColourImage.jpg');
SIZE = fileinfo.bytes;
Size = SIZE/1024;
set(handles.text8,'string',Size);
imshow(L,'Parent', handles.axes2)
end
```

DWT in Mat Lab and Guide :

```
function varargout = alfa(varargin)
% ALFA MATLAB code for alfa.fig
%      ALFA, by itself, creates a new ALFA or raises the existing
%      singleton*.
%
%      H = ALFA returns the handle to a new ALFA or the handle to
%      the existing singleton*.
%
%      ALFA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ALFA.M with the given input arguments.
%
%      ALFA('Property','Value',...) creates a new ALFA or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before alfa_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to alfa_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help alfa

% Last Modified by GUIDE v2.5 12-Mar-2019 22:02:56

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
```

```
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @alfa_OpeningFcn, ...
                   'gui_OutputFcn',  @alfa_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before alfa is made visible.
function alfa_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to alfa (see VARARGIN)

% Choose default command line output for alfa
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);
```

```
% UIWAIT makes alfa wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = alfa_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
global file_name;
%guidata(hObject,handles)
file_name=uigetfile({'*.bmp;*.jpg;*.png;*.tiff;';'*.*'},'Select an Image File');
fileinfo = dir(file_name);
SIZE = fileinfo.bytes;
zSize = SIZE/1024;
imshow(file_name,'Parent', handles.axes2)
set(handles.text18,'string',zSize);
set(handles.text2,'string','Image Loaded');
set(handles.text2,'string','Conversion In Progress');
f=msgbox('Compression Is In Process','Batch 14 ECE');
I=imread(file_name);
[m,n]=size(I);
%Wavelet transform
```

```
[cA1,cH1,cV1,cD1] = dwt2(I,'db2');
dec2d = [cA1,cH1;cV1,cD1];

%Inverse Wavelet transform

IA=idwt2(cA1,[],[],[],'db2');
fna=strcat('.\Compressed\',file_name);
newf3=strcat(fna,'CompressDWT');
seem=randi(100);
disp(seem);
newf2=strcat(newf3,'');
newf=strcat(newf2,'.jpg');
imwrite(uint8(IA),newf);




set(handles.text2,'string',strcat('Image Written to',newf ));
IH=idwt2([],cH1,[],[],'db2');

IV=idwt2([],[],cV1,[],'db2');

ID=idwt2([],[],[],cD1,'db2');

%Compression ratio
I=double(I);
sumI=0;
sumIA=0;
sumIH=0;
sumIV=0;
```

```
sumID=0;
for i=1:m
   for j=1:n
      sumI=sumI+I(i,j);
      sumIA=sumIA+IA(i,j);
      sumIH=sumIH+IH(i,j);
      sumID=sumID+ID(i,j);
      sumIV=sumIV+IV(i,j);
   end
end
cr=(sumIA+sumIH+sumID+sumIV)/(sumI);
display('compression ratio is:');
disp(cr);

%relative data redundancy
red=(1)-(1/cr);
display('relative redundancy is:');

disp(red);
%Calculation of PSNR and compression ratio
squaredErrorImage = (double(I) - double(IA)) .^ 2;
mse = sum(sum(squaredErrorImage)) / (m*n);
PSNR = 10 * log10( 255^2 / mse);
display('PSNR for LL band');
display(mse);
display(PSNR);


fileinfo = dir(newf);
OSIZE = fileinfo.bytes;
OSize = OSIZE/1024;
```

```
set(handles.text19,'string',OSize);
set(handles.text20,'string',cr);
set(handles.text21,'string',red);
set(handles.text22,'string',PSNR);

figure;
subplot(1,2,1),imshow(file_name),title('Original Image');
subplot(1,2,2),imshow(newf),title('Compressed Image');
imshow(newf,'Parent', handles.axes3)
f=msgbox('Compression Successful   \n ....\n ....','Batch 14 ECE');
% --- Executes on button press in togglebutton1.
function togglebutton1_Callback(hObject, eventdata, handles)
% hObject    handle to togglebutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of togglebutton1
```

BIBILOGRAPHY :

[1] Aldroubi, Akram and Unser, Michael (editors), Wavelets in Medicine and Biology, CRC Press, Boca Raton FL, 1996.
[2] Benedetto, John J. and Frazier, Michael (editors), Wavelets; Mathematics and Applications, CRC Press, Boca Raton
FL, 1996.
[3] Brislawn, Christopher M., \Fingerprints go digital," AMS Notices 42(1995), 1278{1283.
[4] Chui, Charles, An Introduction to Wavelets, Academic Press, San Diego CA, 1992.
[5] Daubechies, Ingrid, Ten Lectures on Wavelets, CBMS 61, SIAM Press, Philadelphia PA, 1992.
[6] Glassner, Andrew S., Principles of Digital Image Synthesis, Morgan Kaufmann, San Francisco CA, 1995.

[1.] R. C. Gonzalez and R. E. Woods, “Digital Image Processing”,Second edition,pp. 411-514,2004.

[2.] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," IEEE Trans. on Computers, vol. C-23, pp. 90-93,1974.

[3.] A. S. Lewis and G. Knowles, "Image Compression Using the 2-D Wavelet Transform" IEEE Trans. on Image Processing, Vol. I . NO. 2, PP. 244 - 250, APRIL 1992.

[4.] Amir Averbuch, Danny Lazar, and Moshe Israeli,"Image Compression Using Wavelet Transform and Multiresolution Decomposition"IEEE Trans. on Image Processing, Vol. 5, No. 1, JANUARY 1996.

[5.] M. Antonini , M. Barlaud and I. Daubechies, "Image Coding using Wavelet Transform”,IEEE Trans. On Image Processing Vol.1, No.2, pp. 205 – 220, APRIL 1992.

[6.] Robert M. Gray,IEEE, and David L. Neuhoff, IEEE"Quantization", IEEE Trans. on Information Theory, Vol. 44, NO. 6,pp. 2325-2383, OCTOBER 1998.(invited paper).

[7.] Ronald A. DeVore, Bjorn Jawerth, and Bradley J. Lucier, Member,"Image Compression Through Wavelet Transform Coding" IEEE Trans. on Information Theory, Vol. 38. NO. 2, pp. 719-746, MARCH 1992.

[8.] http://en.wikipedia.org/Image_compression.

[9.] Greg Ames,"Image Compression", Dec 07, 2002.

[10.] S. Mallat, "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 11, No.7, pp. 674-693, July 1989.

[11.] ken cabeen and Peter Gent,"Image Compression and the Descrete Cosine Transform"Math 45,College of the Redwoods

[12.] J. Ziv and A. Lempel, ``A Universal Algorithm for Sequential Data Compression," IEEE Trans. on Information Theory, Vol. 23, pp. 337--342, 1977.

[13.] src-http://searchcio-midmarket.techtarget.com/sDefinition/0,,sid183_gci212327,00.html

[14.] http://www.3.interscience.wiley.com

[15.] http://en.wikipedia.org/wiki/Discrete_wavelet_transform.

[16.] http://www.vectorsite.net/ttdcmp 2.html.

[17.] src-http://searchcio midmarket.techtarget.com/sDefinition/0,,sid183_gci212327,00.html.

[18.] http://encyclopedia.jrank.org/articles/pages/6760/Image-Compression-and-Coding.html

[19.] . http://www.fileformat.info/mirror/egff/ch09_06.html

[20.] http://www.spelman.edu/%7Ecolm/wav.html