Title: Investigation of compression algorithms for the purposes of medical archiving and remote diagnostics

Candidate: Raymond Rochford DipTechEng DipEE BSc(Eng)

MEng thesis

University Name: Dublin City University

Supervisor: Robert Lawlor DipEE BSc(Eng) MSc(Eng)

School: School of Electronic Engineering

Month and Year of Submission: March 1999

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Masters Degree in Engineering is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Ray Rochford

Raymond Rochford

Date: 01/03/22

ACKNOWLEDGEMENTS

I would like to thank all those people who have helped with my research and the writing of this thesis especially Frank Duignan and Eamon Maher.

I would like to especially thank my supervisor Bob Lawlor for his help and guidance in this project.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

LIST OF TABLES

Chapter 4

Chapter 5

Appendix D

Appendix F

Appendix H

Title: Investigation of Compression Algorithms for the Purpose of Medical Archiving and Remote Diagnostics.

Author: Raymond Rochford

Abstract

The objective of this research is to investigate compression algorithms for the purposes of medical archiving and remote diagnostics. A single X-ray of size 1024 by 1024 pixels at 8 bit resolution occupies approximately 1 Mbyte of storage space and takes approximately 8 minutes to transmit over a standard telephone line. There is a need to reduce both required storage space and transmission time. This is achieved by compressing the image. For archiving, a lossless compression algorithm is used and for remote diagnostics a lossy compression algorithm is used. The development of an application for the transmission of images from the server application to a client application using the Internet is also investigated.

The performances of three lossless compression algorithms are investigated: Huffman Coding, Arithmetic Coding and Huffman Coding using Splay Trees. These algorithms are written in C, transformed into (Dynamic Linked Library) DLLs using Visual C++ for use in a Visual Basic application. The algorithms are tested on five images, three X-rays and two standard images, and compression ratio, compression time and decompression time are recorded for each.

The lossy algorithm investigated is Transform Image coding using the Discrete Cosine Transform (DCT). This algorithm is written in C, transformed into a DLL using Visual C++ for use in a Visual Basic application. The algorithm is tested on five images, three X-rays and two standard images, and compression ratio, compression time, decompression time and mean square error for different quality factors are recorded for each image.

The application is developed with a user-friendly Graphic User Interface (GUI) using Visual Basic. The client could choose an image from the server and then zoom in on any section of it. This can be used for remote diagnostics or as a reference tool. The application could also determine which DCT method to use to optimise the bandwidth, depending on the speed of the medium used.

CHAPTER 1
INTRODUCTION

1.1 Objective

The objective of this project is to investigate lossless and lossy compression algorithms for the archiving and transmission of images respectively and to determine the algorithms to be used in a software application which utilises the Internet. The software application should incorporate a user-friendly graphic interface for the archiving and transmission of images over the Internet.

The lossless algorithms tested are Huffman coding, Huffman coding using splay trees and arithmetic coding. They are chosen for their speed and compression ratio with the main emphasis on a fast and simple algorithm. The lossy algorithm tested is the Discrete Cosine Transform (DCT) and is chosen because of the need for speed and the optimisation of the bandwidth used.

The software application is developed to optimise the bandwidth used where the original image is compressed at a default resolution at the server application using the lossy algorithm. The compressed image is then transmitted to the client application where it is viewed at this default resolution. The client can then select a section of the image at a new resolution and only this section would be transmitted at the new resolution from the server application to the client application, thus optimising the bandwidth used.

1.2 Digital image processing

When discussing the digital processing of images, where an image is a two dimensional (2-D) array of values that specifies the intensity of the image at a particular spatial position, the study of the human visual system is important because this system is the most sophisticated image detection, formation and analysis system known [1]. The proverb 'A picture tells a thousand words' expresses the idea of the amount of information contained in a single picture.

The large volume of optical information in many disciplines and the need for its processing and transmission led to the development of image processing by digital

computers. The relevant efforts started around 1964 at the Jet Propulsion Laboratory in Pasadena, California and concerned the digital processing of satellite photographic images coming from the moon. A new branch of science called digital image processing emerged and has in the past three decades exhibited an impressive growth in terms of both theoretical development and applications. It constitutes a leading technology in a number of important areas (e.g. digital communications, broadcasting and medical imaging). The evaluation of the results of image processing operations is affected by how the human visual system responds to image data with the eye acting as a camera and the visual portions of the brain as a complex image processing system [2].

Digital image processing concerns the transformation of an image to a digital format and its processing by digital computers where both the input and output of this transformation are digital images [3]. The first step in any digital image processing application is the digital image formation system.

This system basically consists of an optical system, a sensor and a digitiser. The optical signal is transformed to an analogue electrical signal by using a sensing device (e.g. CCD (Charged Coupled Device) sensor). The analogue signal is transformed to a digital one by using a video digitiser (frame grabber). Thus, the optical image is transformed to a digital one.

The analogue image must be sampled and digitised (by the use of a frame grabber) before it can be processed by the computer. Sampling is the process by which an image is broken down into a set of pixels (picture element) and the grey level in each pixel digitised. A pixel is the basic spatial element of a computer image and a digital image is composed of these pixels distributed in a rectangular array. Each pixel in the digital image takes on a value that indicates the intensity of the analogue image at its location. Shown schematically in Figure 1.1 is an analogue image and its corresponding digital image. The analogue image is sampled at discrete locations and assigned a value in the digital image which corresponds to the intensity at that location in the analogue image. The digital image is thus a representation of the analogue image. Sampling and digitisation are performed by an A/D (analogue to digital) converter.

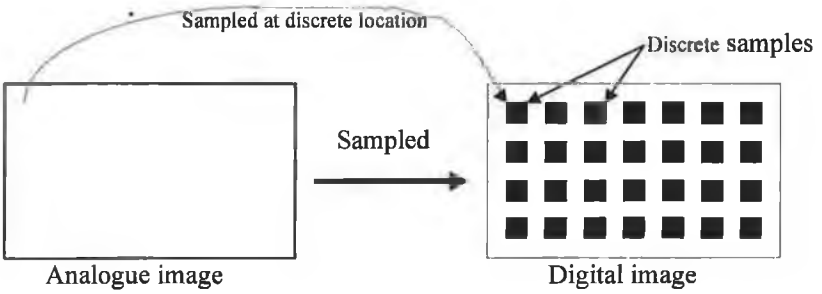


Figure 1.1: Sampling example

The pixels are also quantised by the A/D converter. The process of quantisation is the assignment of a meaningful range of values to individual pixels according to the intensity at a location in the image. An image with pixel values of only 0 or 1 is called a binary image. In a monochrome image, the range of quantisation values is called the grey scale. The range begins with black 0, and increases to white 255, with lighter and lighter shades of grey in between. The individual quantisation values are called grey levels. The use of two hundred and fifty six levels of grey to describe a pixel's quantisation value corresponds to an eight bit binary number and so is important because the basic unit of computer memory storage is the byte (eight bits). Shown in Figure 1.2 is a discrete sample from the digitised image and its corresponding value. This is a pixel in the digital image. Since the image is viewed in grey scale then the quantisation range is from 0 to 255 where each value is an integer. So the discrete sample will be quantised to 253.

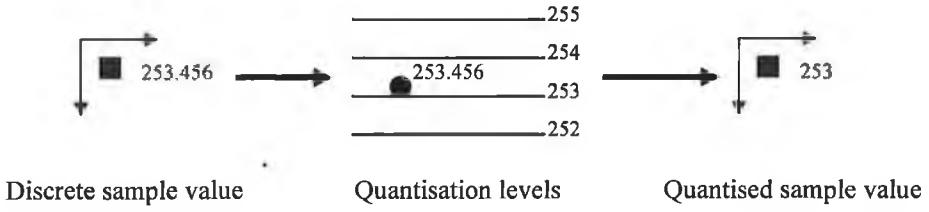


Figure 1.2: Quantisation example

1.3 Digital image compression

Digital images require a large amount of disk space for their storage. A grey scale image of size 1024 x 1024 x 8 bits occupies approximately 1Mb of disk space. Thus, the reduction of disk space requirements is very important in applications such as image storage or transmission.

There are two types of compression. Lossless compression is where the reconstructed image (after it has been compressed and decompressed) is identical to the original. There is no loss of information in the compression-decompression process. Lossless compression is used for archiving since medical images have to be stored without any loss of information. Lossy compression is where the reconstructed image, after it has been compressed and decompressed, is not identical to the original. There is a loss of information in this case but this loss is not generally visible. Visually the reconstructed image looks identical to the original image. Lossy compression produces higher compression ratios than lossless compression so it is used to compress an image before it is to be transmitted as in the case of remote diagnostics.

Digital image coding and compression take advantage of the information redundancy existing in the image, by coding the information content more efficiently [4]. Redundancy relates to how data is distributed throughout the image. Shown in Figure 1.3 is a section of a digital image. The information content is not coded efficiently as the value 152 appears several times occupying 8 bits each time. This can be coded more efficiently by reducing the number of bits required to represent some of the 152 values and can be achieved simply by using a code , & , which indicates that 152 will appear a number of times, dependent on the value immediately following the code, in the file, thus reducing the number of bits required.

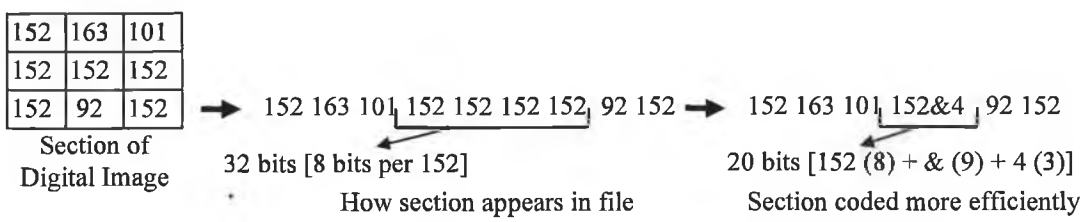


Figure 1.3: Example of redundancy

Large compression ratios (e.g. 1:10) can be obtained by thorough exploitation of the information redundancy. Excessive image compression however results in image degradation after decompression. A good compromise between image degradation and compression ratio must be found. Intensive research over recent years has led to a large number of digital image compression techniques. Some of these are already CCITT (**C**onsultative **C**ommittee for **I**nternational **T**elephony and **T**elegraphy) standards [5].

Digital image frequency content plays an important role in digital image compression. Discrete Cosine Transform (DCT) is used to obtain the digital image frequency content, shown schematically in Figure 1.4. So transform theory is an integral part of digital image processing. The transforms used are 2-dimensional (2-D), because the digital image itself is a 2-D signal. The computation of this transform requires a large number of numerical operations (multiplications and additions). Therefore, the development of fast transform algorithms has been a very important advance in this work.

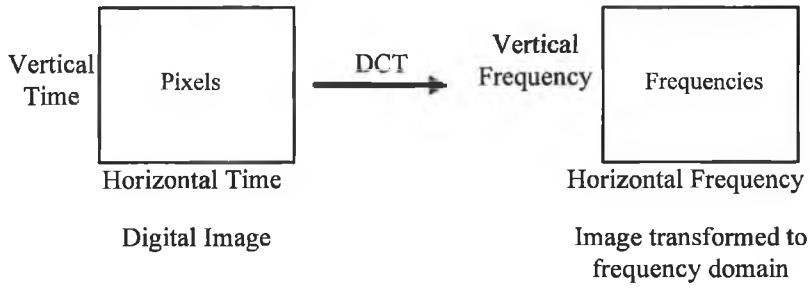


Figure 1.4: Transform using DCT

1.4 Application to the medical environment

An average X-ray department takes an X-ray approximately every six minutes during the working day. If each X-ray is digitised to a 1024 x 1024 x 8 bits digital image, over one week a substantial amount of memory space would be required. If each image could be compressed to one quarter of the above size using lossless compression, there would be a corresponding reduction in the memory space required. Due to the nature of medical images they must be stored (archived) without

loss of information and lossless compression achieves this [6]. An example would be a compact disk of size 800 Mb which can now hold 800 X-ray images. If the images were compressed, this compact disk could hold 3200 X-ray images and each X-ray image could be faithfully decompressed when required.

If an image is to be transmitted for diagnosis or as a reference image it takes approximately 300 s using a standard 14.4 Kb/s modem. This time required for the image to download is not acceptable and adds to the telephony costs. If the image could be compressed to one tenth its size, using lossy compression without any visible loss of information, the transmission time would be reduced to approximately 30s. Lossy compression achieves this level of compression and yields an acceptable decompressed image since the final receiver is the human eye which can compensate for a certain level of image imperfection.

Therefore the implementation of an appropriate lossless or lossy algorithm is very significant in the medical imaging field for archiving and also for the transmission of medical images.

1.5 Summary of work

This **Chapter** introduces the objective of this project and gives a background to image processing and image compression. It also introduces some of the principles and techniques involved in image compression.

In **Chapter 2,** the subject of image compression is discussed in some detail. The two compression techniques, lossless and lossy, are introduced. The lossless algorithms investigated are static Huffman coding, adaptive Huffman coding, Huffman coding using splay trees and arithmetic coding. These algorithms are discussed and so also is run length encoding as used in the lossy compression algorithm. The lossy compression algorithm investigated is the Discrete Cosine Transform (DCT) compression algorithm.

In **Chapter 3,** the use of the Internet for transmitting and receiving data is introduced. The client-server principle is described, as is also the requirement to

change from OSI (Open System Interconnection) to TCP/IP (Transport Control Protocol / Internet Protocol) protocols. Next the TCP/IP addressing scheme is introduced and the principle of data transmission using the Internet is discussed.

In **Chapter 4,** the implementation of the overall objective is discussed. The software application for the transmission of the images and the functionality of this application is described. The choice of the language environment is also discussed. The test images which are to be used, as well as their relevant histograms, are introduced.

The implementation of each lossless compression algorithm is discussed, as well as the results of these implementations and the appropriate lossless algorithm is chosen for the specific software application.

The implementation of the lossy algorithm is investigated using different size quantisation matrices and different lossless encoding algorithms. The results of these implementations are shown and the optimal method is chosen for the software application.

Finally, the Internet software application is discussed and the transmission times are shown for each method evaluated using an Internet LAN (Local Area Network) and a 14.4 Kb/s modem. The method of sending only the difference instead of the image when zooming in is investigated and the resulting transmission times are shown over an Internet LAN and a 14.4 Kb/s modem.

In **Chapter 5** the application-specific use of lossless and lossy algorithms is discussed. Future development of the work is introduced. The extension to colour medical images is discussed. Also discussed briefly is a lossy algorithms which could be incorporated into the application, wavelet transforms. Finally the introduction of encryption into the application is discussed.

Appendix A contains an example of Huffman coding using splay trees and shows how this can result in the attainment of higher image compression ratios.

Appendix B contains the derivation of the 1-dimensional DCT and the 1-dimensional inverse DCT. The 2-dimensional DCT and the 2-dimensional inverse DCT are then derived from the 1-dimensional equations.

Appendix C shows the headers for the bitmap format which is the format for viewing images in a Visual Basic environment. The process of transformation of colour images from the RGB plane to the YUV plane is also shown.

In **Appendix D**, the lossless compression results for the test images, a chest X-ray image, a pelvis X-ray image, a mandrill image and a Lenna image, are shown.

In **Appendix E** the quantisation matrices are shown.

Appendix F shows the lossy compression results for the four test images, for different quantisation sizes, 8 x 8, 16 x 16 and 32 x 32.

In **Appendix G** screenshots from the Visual Basic software application are shown and the use of the client software to view an archived image on the server is discussed.

In **Appendix H** the results for transmitting only the difference between the four test images, at different resolutions, are shown.

Appendix I shows the abstract of a presentation of some of this work given at the first annual scientific meeting of the Biomedical Engineering Association of Ireland on 9 March 1996.

In **Appendix J** the paper on this work presented at the IDSPCC in Trinity College, Dublin on 24 June is shown.

CHAPTER 2
DIGITAL IMAGE COMPRESSION

2.1 Image compression

Digital image compression is concerned with the minimisation of the number of bytes needed to represent a digital image. Compression therefore reduces the amount of disk space required for image archiving in a variety of applications, e.g. medical imaging and remote sensing. Digital image compression techniques can be divided into two classes, lossless and lossy [7].

Lossless compression is used in applications where raw image data contain vital information that must not be altered in any way by the process e.g. medical diagnostic imaging.

Lossy compression can be used when raw image data can be easily reproduced or when the information loss can be tolerated at the receiver site. Typical cases are in video conferencing and digital TV applications where the final receiver is the human eye.

All digital image compression techniques are based on the exploitation of information redundancy in the digital images. The redundancy stems from the statistics of the image data (e.g. strong spatial correlation). The aim of compression algorithms is to represent the image, using a lower number of bits per pixel without losing the ability to reconstruct the image. No non-redundant image data may be lost in the data compression process, otherwise error-free reconstruction is impossible. Image redundancy can be described in various ways and each of these leads to a particular class of digital image compression algorithms.

Statistical redundancy is directly related to the image data probability distribution and can be treated by information theory techniques using image entropy concepts. Its removal results in lossless image compression techniques (Huffman coding, run-length coding). Predictability in a local image neighbourhood is another way of describing image redundancy. Local prediction models take advantage of the strong spatial correlation and try to decorrelate the pixels in a local neighbourhood. The transmission or storage of the decorrelated (error) image results in image

compression. Most predictive compression schemes, e.g. Linear Predictive Coding (LPC), result in lossy compression [8].

Image compression can also be achieved by information packing through image transforms. This is obtained by exploiting the fact that certain transforms (e.g. Discrete Cosine Transform) can concentrate image energy in a few transform coefficients. Thus, coding of transform coefficients can lead to considerable data compression [9].

2.2 Lossless image compression

2.2.1 Huffman coding

Binary coding is a process by which numbers are represented by corresponding strings of 0s and 1s called code words. This form of coding is ideally suited to meet the requirements for the storage or transmission of data. The inverse process, the reconstruction of the data is called decoding.

The single feature assuring unambiguous decoding is called the prefix condition. No word in the code book should appear as the first bits, or prefix, in any other word. When a codeword, codes representing each number, has been received, there should be no possibility that it is part of a longer word. That means that, the next bit must be the beginning of a new word [10].

An image is sampled to a size $N \times M$ pixel matrix and each pixel quantised to B bits. Each of the 2^B image intensity levels is transmitted by using B bits. The average number of bits per pixel is reduced by assigning binary codes of different bit lengths to the various image intensities.

Let p(i) be the probability density function (pdf) of the image intensities i, for $0 \le i \le 2^B$. Once the pdf is known, short codewords can be assigned to intensities having a high probability of occurrence and larger codewords can be assigned to less frequent intensity levels. This coding scheme is called entropy coding. Suppose a code word of length L(i) is assigned to intensity levels i, $0 \le i \le 2^B$. Average code word length is then given by

$$\bar{L} = \sum_{i=0}^{2^B-1} L(i)p(i) \quad \text{.....2.1}$$

Length L(i) must be chosen in such a way that $\bar{L}$ is minimised. Information theory gives a lower bound H(B) on the average codeword length

$$\bar{L} \geq H(B) \text{ where } H(B) = -\sum_{i=0}^{2^B-1} p(i)\log_2(p(i)), \text{ the image entropy} \quad \ldots\ldots\ldots\ldots 2.2$$

If the image intensity levels are coded by using variable length codewords, then the codewords are joined to form a binary data stream. This stream must be decoded at the receiver site. Therefore the combinations of the joined code words must be decipherable. The Huffman code possesses this property [11]. Its average code word length is very close to the image entropy value H(B).

In the Huffman code, no codeword can be the prefix of another code word. This guarantees that the encoded binary data stream is decipherable. The resulting code has a tree form. The Huffman code can be constructed by using a tree.

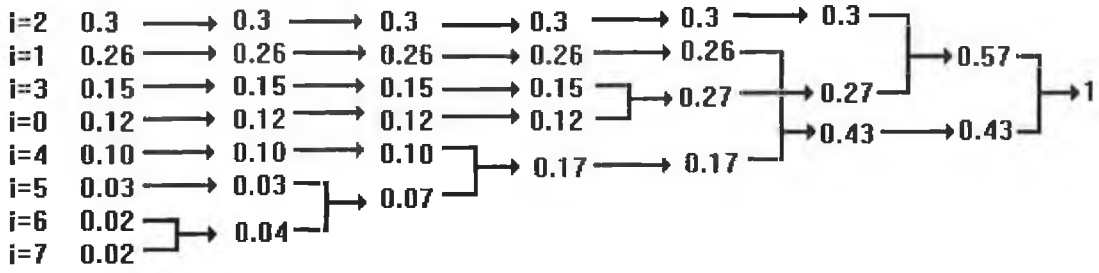
The Huffman algorithm [12] builds up a weighted binary tree according to the rate of occurrence of the intensities. Each element of this tree is assigned a new code, where the length of the codeword is determined by its position in the tree. The intensity which has the highest rate of occurrence then becomes the root of the tree and is assigned the shortest code. Each less frequent intensity is assigned a longer code word. Shown is an example to demonstrate the method of Huffman coding

$$p(i) = [.12 \quad .26 \quad .3 \quad .15 \quad .1 \quad .03 \quad .02 \quad .02]$$

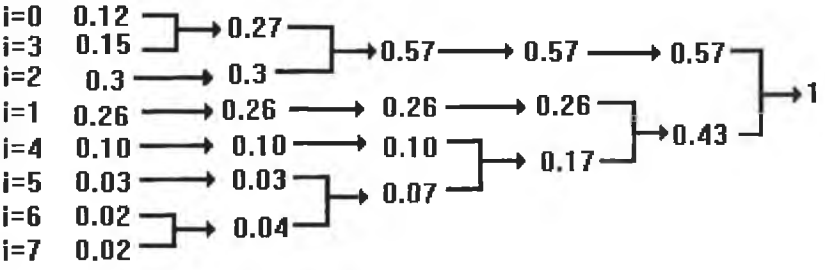
Suppose that the image has eight intensity levels i=0---7. We assume that the probabilities p(i) are known and a column of intensity levels with descending probabilities can be used.

$$\begin{aligned} i &= 2 \quad .3 \\ i &= 1 \quad .26 \\ i &= 3 \quad .15 \\ i &= 0 \quad .12 \\ i &= 4 \quad .1 \\ i &= 5 \quad .03 \\ i &= 6 \quad .02 \\ i &= 7 \quad .02 \end{aligned}$$

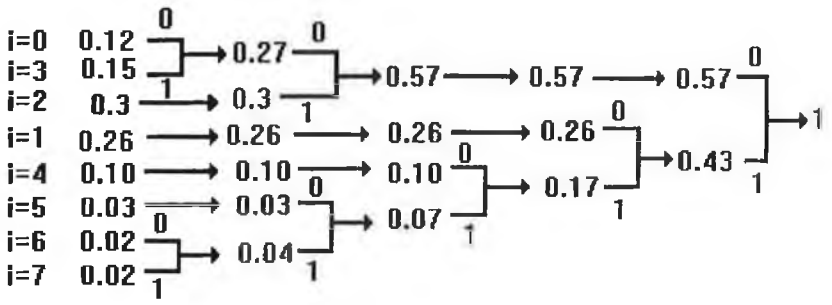
The intensities of this column constitute the so-called leaves of the Huffman code tree. The tree is constructed in individual steps. At each step the two tree nodes having minimal probabilities are connected to form an intermediate node. The probability assigned to this node is the sum of the probabilities of the two branches. This procedure is repeated until all branches (intensity levels) are used and the probability sum is 1.



The code tree is unscrambled to eliminate branch crossovers.



The code words are constructed by travelling through the decoding tree from its root to its leaves. At each level a 0 is assigned to the top branch and a 1 to the bottom branch. The procedure is repeated until all tree leaves are reached. Each leaf corresponds to a unique intensity level.



The code word for each intensity consists of the zeros and ones that exist in the path from the root to this specific leaf.

i	p(i)	Code word
2	0.3	01
1	0.26	10
3	0.15	001
0	0.12	000
4	0.10	110
5	0.03	1110
6	0.02	11110
7	0.02	11111

The intensity levels i=1,2 are assigned small code words since they have a high probability density function while the intensities with low probability density functions have longer code words.

As these intensities appear in the image they are assigned their code words to form a binary data stream and are decoded at the receiving end to reconstruct these intensities.

2.2.2 Arithmetic coding

Huffman codes clearly have to be an integral number of bits long. If the probability of an intensity occurring is 0.33, then the optimum number of bits to code that intensity is around 1.6. Huffman coding has to assign either one or two bits to the code and this leads to a longer compressed code.

The Huffman coding method is optimal when and only when the intensity probabilities are integral powers of ½. The method of arithmetic coding [13] does not have this restriction. It treats the stream of intensities as a single unit, and thus attains the theoretical entropy bound to compression entropy.

Instead of representing each input intensity with a specific code, arithmetic coding represents a stream of input intensities with a floating-point number. Arithmetic coding works by representing each intensity by an interval of real numbers between 0 and 1, therefore the output from arithmetic coding is a single number less than 1 and greater than or equal to 0 [14], a floating-point number. As the input stream becomes longer the interval needed to represent it becomes smaller and smaller. This floating-point number can be uniquely decoded to create the exact stream of intensities that went into its construction.

Consider an example stream of input intensities [34 83 89 89 11 73 21 121 56 110] to demonstrate the method of arithmetic coding.

Intensity	Probability
11	0.1
21	0.1
34	0.1
56	0.1
73	0.1
83	0.1
89	0.2
110	0.1
121	0.1

Once intensity probabilities are known individual intensities are assigned a range along a "probability line", nominally 0 to 1. Each intensity is assigned the portion of the 0 to 1 range that corresponds to its probability of occurrence.

Intensity	Probability	Range		
		Low Range		High Range
11	0.1	0.00	to	0.1
21	0.1	0.1	to	0.2
34	0.1	0.2	to	0.3
56	0.1	0.3	to	0.4
73	0.1	0.4	to	0.5
83	0.1	0.5	to	0.6
89	0.2	0.6	to	0.8
110	0.1	0.8	to	0.9
121	0.1	0.9	to	1.0

Intensity Table

Intensity 121 has the range 0.9 to 0.999. The most significant intensity is 34 which has the range 0.20 to 0.30, and so all other subranges are calculated from this range.

Intensity	Low Value	High Value
34	0.20	0.30

The following transforms are used to calculate the subranges

range = high value - low value
high value = low value + range * low range
low value = low value + range * high range

From intensity 83: range = 0.3-0.2 = 0.10
low value = 0.2 + 0.1*0.5 = 0.25
high value = 0.2 + 0.1*0.6 = 0.26

Intensity	Low Value	High Value
34	0.20	0.30
83	0.25	0.26

From intensity 89: range = 0.26-0.25 = 0.01
low value = 0.25 + 0.01*0.6 = 0.256
high value = 0.25 + 0.01*0.8 = 0.258

Intensity	Low Value	High Value
34	0.20	0.30
83	0.25	0.26
89	0.256	0.258

From intensity 89: range = 0.258-0.256 = 0.002
low value = 0.256 + 0.002*0.6 = 0.2572
high value = 0.256 + 0.002*0.8 = 0.2576

Intensity	Low Value	High Value
34	0.20	0.30
83	0.25	0.26
89	0.256	0.258
89	0.2572	0.2576

From intensity 11: range = 0.2576-0.2572 = 0.0004
low value = 0.2572 + 0.0004*0.00 = 0.25720
high value = 0.2572 + 0.0004*0.01 = 0.25724

Intensity	Low Value	High Value
34	0.20	0.30
83	0.25	0.26
89	0.256	0.258
89	0.2572	0.2576
11	0.25720	0.25724

From intensity 73: range = 0.25724-0.25720 = 0.00004
low value = 0.25720 + 0.00004*0.4 = 0.257216
high value = 0.25720 + 0.00004*0.5 = 0.257220

Intensity	Low Value	High Value
34	0.20	0.30
83	0.25	0.26
89	0.256	0.258
89	0.2572	0.2576
11	0.25720	0.25724
73	0.257216	0.257220

and so on for other intensities until a final low number is obtained

Intensity	Low Value	High Value
34	0.20	0.30
83	0.25	0.26
89	0.256	0.258
89	0.2572	0.2576
11	0.25720	0.25724
73	0.257216	0.257220
21	0.2572164	0.2572168
121	0.25721676	0.25721680
56	0.257216772	0.257216776
110	0.2572167752	0.2572167756

Thus 0.2572167752 uniquely encodes the stream of intensities.

To decode the floating point number, the number 0.257168852 has a low value 0.2

and when compared to the intensity table gives an intensity of 34.

Transform used to calculate the individual intensities

range = high range - low range
number = number - low range
number = number / range

Therefore range = 0.3 - 0.2 = 0.1
number = 0.2572167752 - 0.2 = 0.0572167752
number = 0.0572167752 / 0.1 = 0.572167752

Low value is 0.5 and when compared to intensity table gives an intensity of 83

Therefore range = 0.6 - 0.5 = 0.1
number = 0.572167752 - 0.5 = 0.072167752
number = 0.072167752 / 0.1 = 0.72167752

Low value is 0.6 and when compared to intensity table gives an intensity of 89

Therefore range = 0.8 - 0.6 = 0.2
number = 0.72167752 - 0.6 = 0.12167752
number = 0.12167752 / 0.2 = 0.6083876

Low value is 0.6 and when compared to intensity table gives an intensity of 89

Therefore range = 0.8 - 0.6 = 0.2
number = 0.6083876 - 0.6 = 0.0083876
number = 0.0083876 / 0.2 = 0.041938

Low value is 0.0 and when compared to intensity table gives an intensity of 11

Therefore range = 0.1 - 0.0 = 0.1
number = 0.041938 - 0.0 = 0.041938
number = 0.041938 / 0.1 = 0.0041938

Low value is 0.4 and when compared to intensity table gives an intensity of 73

And so on to reconstruct other intensities to obtain

[34 83 89 89 11 73 21 121 56 110]

2.2.3 Splay trees

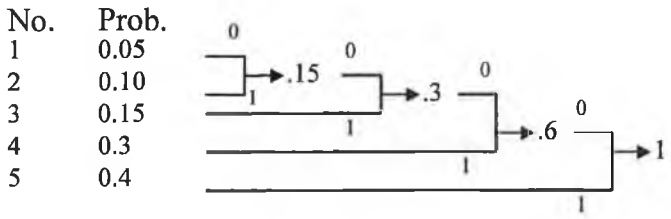
Huffman compression algorithms require the use of a tree balancing scheme. Splay trees are ordered binary search trees, which when applied to Huffman coding, lead to a locally adaptive compression algorithm.

A binary code used in the compressed data file may not be the prefix of any other code. Prefix codes are part of a binary tree and can be read by following the path from the root of the tree to the intensity leaf and associating a 0 with each left branch followed and a 1 with each right branch followed. The code tree for Huffman coding is a weight balanced tree where each leaf is weighted with an intensity frequency value and internal nodes have no weight. Huffman coding requires two passes through the data to be compressed. First pass is to obtain the intensity frequencies and the second pass is to perform the actual compression.

When a node in a tree is accessed the tree is splayed [15]. That means that the accessed node becomes the root and all nodes to the left of it form a new left subtree while all roots to the right form a new right subtree. Splaying is accomplished by following the path from the old root to the target root, making only local changes along the way

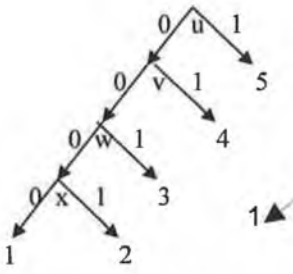
Splaying applies to the trees where there are data stored in the internal roots and not in the leaves. Semi-splaying is applied to prefix code trees. The target node is not saved to the root and the path from the root to the target is reduced by a factor of two.

Since a Huffman tree is a statically balanced tree, splaying is applicable to data compression. Shown in Figure 2.1 is an example of splaying. Figure 2.1 (a) shows a Huffman code tree and its corresponding Huffman codes assigned to the values. The total number of bits is 14. Shown in Figure 2.1 (b) is the Huffman tree from the top down. The length from root to node 1 is 4. Figure 2.1 (c) shows the tree splayed around node 1 so that new subtrees have been created. The length from the root to node 1 has been halved, giving the length now 2 and the total number of bits required to represent the numbers now 12. See Appendix A for a larger example of splaying when applied to Huffman coding.

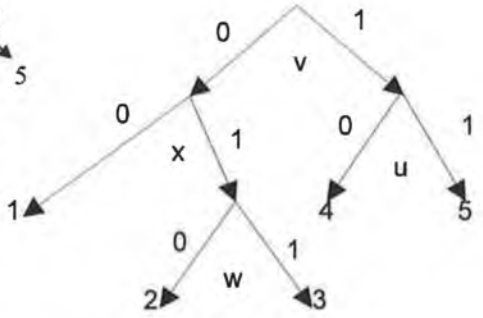


1	0000
2	0001
3	001
4	01
5	1

(a) Huffman tree and corresponding Huffman codes



(b) Tree from top down



1	00
2	010
3	011
4	10
5	11

(c) Tree after splaying and corresponding new codes

Figure 2.1: Example of splaying

2.2.4 Run length encoding

In an input intensity stream, intensity values are often identical. Run Length Encoding (RLE) [16,17] checks the stream of intensity values for this and inserts a code each time a chain of more than two equal intensities are found. This code tells the decoder to insert the following intensity n times into the output stream. Figure 2.2 is an example of RLE. Figure 2.2 (a) shows a section of an image and how it would appear in a file, scanned from left to right. Also shown is the run length encoding of this data. Figure 2.2 (b) shows the section when it has been zig-zag coded and then run length encoded. As can be seen, when zig-zag coded longer run lengths are produced.

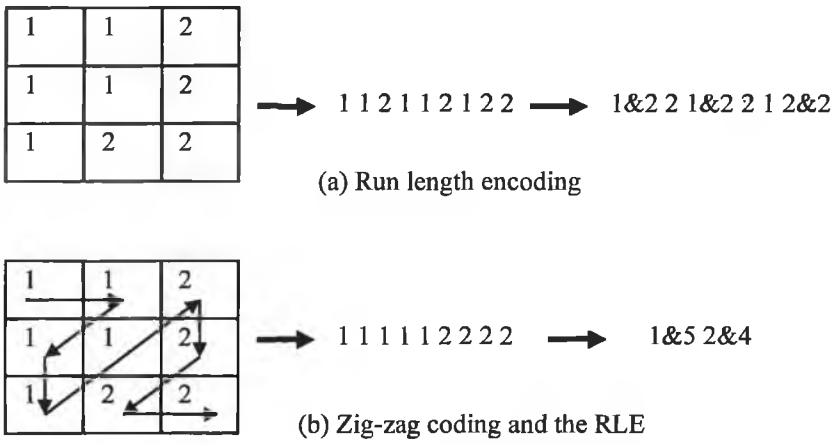


Figure 2.2 Example of run length encoding

Show is another example to demonstrate the method of RLE.

A selection of input intensities is shown.

[147 134 140 140 140 140 140 140 67 56]

The output from the run length coder would be

[147 134 %6140 67 56]

Thus the amount of information to be stored or transmitted has been reduced and the coded intensities can be easily decoded

2.3 Lossy compression

2.3.1 Transform image coding

Another approach to digital image coding is to use image transforms to concentrate the image energy in a few transform coefficients [18]. In transform image coding, an image is transformed to a domain different from the image intensity domain and the transform coefficients are then coded. If energy packing is obtained, a large number of transform coefficients can be discarded and the rest coded with variable length codewords, thereby giving data compression. Let f represent an image of size

$L = N \times M$. The transform vector is given by F=Af where A is the transform matrix and the inverse transform is defined as $f = A^{-1}F$.

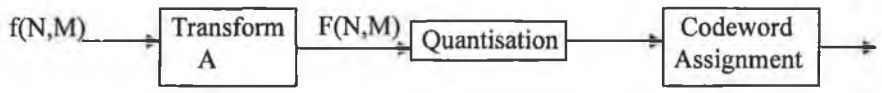


Figure 2.3: Transform image coding block diagram

Transform image coding techniques attempt to reduce the correlation that exists among image pixel intensities. The energy compaction property arises from the fact that a large amount of energy is concentrated in a small fraction of the transform coefficients. In most practical cases, the signal energy is unevenly distributed in the transform coefficients. The d.c. coefficient and some other low frequency coefficients F(k), $1 \le k \le K \le L$ tend to concentrate most of the signal energy [19].

Many transform coefficients (e.g. F(k), k>K) may be discarded without significant loss of information. A variable number of bits n_k, $1 \le k \le K$, may be allocated to each coefficient, so that the average number of bits per pixel is equal to a predefined number B.

Once the number of bits allocated is determined, the transform coefficients F(k), $1 \le k \le K$, may be quantised to produce the encoded image. To obtain a good transform coding algorithm several problems must be solved [20].

1. The choice of the transform to be used, e.g. Discrete Fourier Transform, Discrete Cosine Transform etc. The DCT is used in the JPEG (Joint Photographic Experts Group) compression standard of CCITT (see section 2.3.2).
2. The choice of image block size. It is not advisable to apply one transform to the entire image because of the changing image statistics in the various image regions. The image is split into a number of non-overlapping blocks that are coded independently. A typical block size in this application $N \times M$, is 8×8 or 16×16.
3. Determination of bits allocation. Suppose n_k, k=1----L, is the number of bits allocated to each transform coefficient F(k), k=1----L. If n_k=0, it means that the

corresponding coefficient F(k) is discarded. The average number of bits per pixel is

$$\frac{1}{L}\sum_{k=1}^{L} n_k = B \quad \text{.....2.3}$$

2.3.2 JPEG (Joint Photographic Experts Group)

The primary example of the transform coding of still pictures is the JPEG international standard prepared by the ISO/IEC [21] JTC1/SC2/WG10 photographic committee. This group collaborated informally with a special rapporteur committee of CCITT SGVIII. In this joint work the ISO/IEC selected, developed and tested coding techniques, whereas CCITT SGVIII provided coding specifications for image communication applications. The JPEG standard provides algorithms for lossless and lossy compression [22].

2.3.3 Discrete Cosine Transform (DCT)

The Discrete Fourier Transform (DFT) [23] is a transform which has a fixed set of basis functions, an efficient algorithm for its computation and good energy compaction. The transform of $x(n_1, n_2)$ is $X(\omega_1, \omega_2)$. The DFT of typical images have most of their energy concentrated in a small region in the frequency domain near the origin and along the ω_1 and ω_2 axes. Energy concentration occurs near the origin because most images have large regions where the intensities change slowly. Sharp discontinuities contribute to high frequency components. The energy concentration along the ω_1 and ω_2 axes is due to the rectangular window used to obtain a finite extent image. The rectangular window creates artificial sharp discontinuities at the four boundaries. Discontinuities at the top and bottom of the image contribute energy along the ω_2 (vertical) axis and discontinuities at the two sides contribute energy along the ω_1 (horizontal) axis [24].

Since most of the signal energy is concentrated in a small region an image may be reconstructed without significant loss of quality and intelligibility from a small percentage of transform coefficients. It is possible to improve the energy

compaction property of the DFT without sacrificing other qualities such as the existence of a computationally efficient algorithm, this is because most of the energy is packed into the fewest coefficients. In transform coding the transform coefficients of an image rather than its intensities are coded. The DCT [25] can be derived from the DFT since it is so closely related to it. Appendix B gives the derivations of the Discrete Cosine Transform.

The DCT is used instead of other transforms such as the Haar and the Hadamaard even though they require fewer computations than the DCT, because their energy compaction properties are not as good as that of the DCT for typical images.

The Discrete Cosine Transform was introduced by Ahmed and his colleagues in 1974 [26, pp 1-25]. Since then it has been extensively used in various image coding schemes, because of its excellent performance in typical images exhibiting high spatial correlation. In most practical cases, the use of a DCT-based transform scheme has a computational advantage over other transform schemes due to its use of the FFT [26, pp 123-135]. The DCT forms the basis of the JPEG standard for still image compression and also the standard for moving image compression standard, MPEG. For images the 2-D DCT is used. Shown is an example of the DCT.

Case 1. The image in the intensity domain is a white vertical stripe. When transformed to the frequency domain all the energy is contained in the low horizontal frequencies.

Intensity Domain							Frequency Domain				
0	0	1	0	0			20	0	-20	0	20
0	0	1	0	0		DCT	0	0	0	0	0
0	0	1	0	0	.	→	0	0	0	0	0
0	0	1	0	0			0	0	0	0	0
0	0	1	0	0			0	0	0	0	0

Case 1

Case 2. The image in the intensity domain is a white horizontal stripe. When transformed to the frequency domain all the energy is contained in the low vertical frequencies.

Intensity Domain						Frequency Domain				
0	0	0	0	0		20	0	0	0	0
0	0	0	0	0	DCT	0	0	0	0	0
1	1	1	1	1	→	-20	0	0	0	0
0	0	0	0	0		0	0	0	0	0
0	0	0	0	0		20	0	0	0	0

Case 2

Case 3. The image is a white plus sign. When transformed to the frequency domain most of the energy is concentrated in the DC coefficient which is 36.

Intensity Domain						Frequency Domain				
0	0	1	0	0		36	0	16	0	-16
0	0	1	0	0	DCT	0	0	0	0	0
1	1	1	1	1	→	-16	0	-4	0	4
0	0	1	0	0		0	0	0	0	0
0	0	1	0	0		16	0	-4	0	-4

Case 3

2.3.4 Sub-image by sub-image coding

In transform image coding an image is divided into many sub-images, each of which is transformed and coded separately [26, pp 166]. By coding each sub-image separately the coder can be made adaptive to local image characteristics. Quantisation and bit allocation methods may differ between uniform background regions and edge regions. Sub-image by sub-image coding reduces storage and computational requirements. Since one sub-image is processed at a time, it is not necessary to store the entire image.

To illustrate the resulting reduction in computational requirements, consider an image $f(n_1, n_2)$ with $N \times N$ pixels where N can be expressed as a power of 2. $f(n_1, n_2)$ is divided into sub-images. Each sub-image is $M \times M$ where M can be

expressed as a power of 2. The number of sub-images in the image is $\frac{N^2}{M^2}$. The number of arithmetic operations in computing an $M \times M$ point transform is $M^2 \log_2 M^2$. The total number of arithmetic operations required in computing all the transforms in an image is

$$\left(\frac{N^2}{M^2}\right) M^2 \log_2 M^2 = N^2 \log_2 M^2 \quad \text{.....2.4}$$

Computing the transform of one $N \times N$ point image requires $N^2 \log_2 N^2$ arithmetic operations. Since M<<N then there are less arithmetic operations in the sub-image by sub-image approach.

Although a smaller sized sub-image is more efficient computationally and allows a coder to be more adaptive to local image characteristics, the sub-image size cannot be reduced indefinitely. As the image is divided into smaller segments, transform coding exploits less of the correlation present among image pixel intensities. As sub-image size decreases the correlation among neighbouring subimages increases. Since each sub-image is coded independently, possible correlation among neighbouring sub-images is not exploited. This decreases the performance of transform image coding and imposes a limit on the sub-image size. Typical sub-image sizes are 8×8 and 6×16.

2.3.5 Bit allocation

Transform image coding seeks to exploit the energy compaction property of the transform. Selection procedures are required since only a small percentage of the transform coefficients are typically coded. Two approaches which are used to determine which transform coefficients are to be coded are zonal and threshold coding [26, pp 167-175].

In zonal coding, only the coefficients within a specified region are coded. The zone shapes are consistent with the observation that most of the energy in typical images is concentrated in the low frequency region.

In threshold coding, transform coefficients are compared with a given threshold and those above the threshold are coded. From the energy compaction point of view threshold coding is preferable to zonal coding. In zonal coding some transform coefficients with small magnitudes may be coded while those with large magnitudes are discarded since zones are pre-specified. In threshold coding only the coefficients with large magnitudes are selected. Choice of which transform coefficients are to be coded depends on the local image characteristics.

It is beneficial to allocate more bits to a coefficient with a large expected variance. For the DCT, the expected variance is much larger for low frequency coefficients than for high frequency coefficients.

Quantisation of one transform coefficient affects all the image intensities within the sub-image. Several types of image degradation result from quantisation noise in transform image coding. One type is loss of spatial resolution. In DCT coding of images, the discarded transform coefficients are typically high frequency components. The result is a loss of detail in the image. Another type of degradation results from quantisation of the retained transform coefficients. Degradation in this case appears as graininess in the image. A third type of degradation arises from sub-image by sub-image coding. Since each sub-image is coded independently, the pixels at the sub-image boundaries may have artificial intensity discontinuities. This is known as the blocking effect and becomes more pronounced as the compression ratio increases.

The DCT is used in the algorithm to compress the image before it has been transmitted to decrease the bandwidth required to transmit the image. Chapter 3 provides a discussion of the Internet principles for the transmission of images, which illustrate the value of compression in this application.

CHAPTER 3
INTERNET COMMUNICATIONS

3.1 The Internet

The Internet consists of thousands of networks that use the TCP/IP (Transport Control Protocol / Internet Protocol) suite [27]. Protocols are rules that define how the software application must work to handle the flow of information that passes from application to application. The TCP/IP protocol suite manages all information that moves across the Internet.

3.2 Network programming models

Network programming can be thought of in two primary contexts, client/server and distributed. For the purposes of this project a client/server model is used [28].

An application is split into two parts. A front-end client that presents information to the user and collects information from the user, and a back-end server that stores, retrieves and manipulates data, and generally handles the bulk of the computing tasks for the client.

A server is any program that runs on a networked computer and can provide a service. On receipt of a request via the network, a server performs the necessary processing to service the request and returns the result to the requester. The client is the program that sends a request to a server and waits for a response. For a client and server to communicate and co-ordinate their work, an interprocess communication (IPC) facility is needed.

In the client/server environment data are sent in their raw format from the server to the client. The specific application running on the client computer determines how the data is displayed.

3.3 TCP/IP

The experimental network called ARPANET gave rise to the TCP/IP protocol [29]. The TCP/IP protocol includes a set of standards that specify how networked

computers communicate and how data is routed through the interconnected computers. TCP/IP provides two primary services: connectionless packet delivery and reliable stream transport.

The International Standards Organisation (ISO) introduced the Open System Interconnection (OSI) reference model, a layered network architecture. The OSI model is said to be an open systems architecture because it connects computer systems that are open for communication with other systems. Connected computer systems do not have to run the same operating system [30].

The OSI model is composed of seven layers as shown in Figure 3.1. These layers define the function of data communication protocols. Each layer of the OSI model represents a function performed when data is transferred between co-operating applications across a connecting network. A layer does not have to define a single protocol, but rather a function that is performed by any number of protocols.

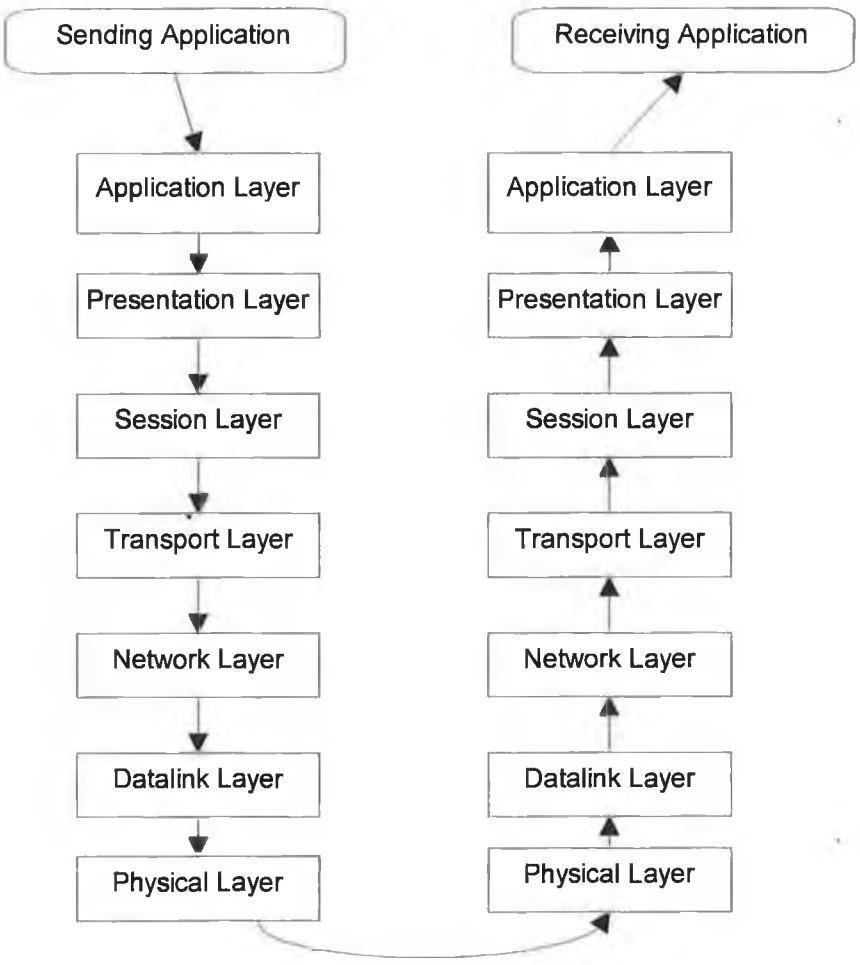


Figure 3.1: OSI model

The Application Layer provides end-user services. This is the layer closest to what the user of the computers sees and manipulates. The Presentation Layer controls how data is represented. Data compression and decompression may take place at this point. The Session Layer manages process-to-process communication sessions between hosts and is responsible for establishing and terminating connections between co-operating applications. The Transport Layer performs end-to-end error detection and correction. This layer guarantees that the receiving application receives the data exactly as it was sent. The Network Layer manages the network connection. This layer takes care of data packet routing between source and destination computers as well as network congestion. The Datalink Layer provides reliable data delivery across the physical network and does not assume that the physical network is necessarily reliable. The Physical Layer is concerned with transmitting and receiving raw bits over a physical communication channel. The Ethernet is an example of a channel. This layer defines voltage levels and connection points appropriate to the physical hardware [31].

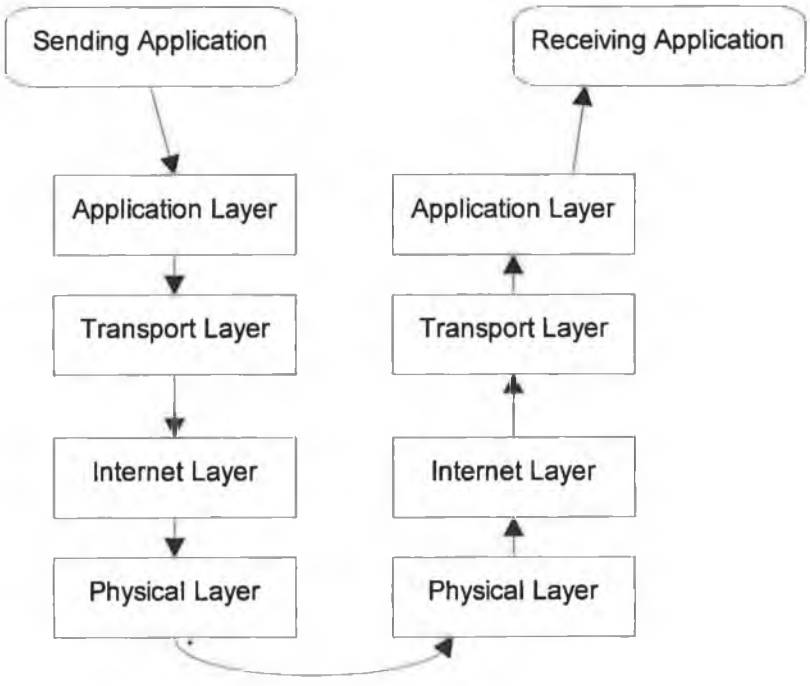


Figure 3.2: TCP/IP model

TCP/IP as shown in Figure 3.2 does not directly follow the OSI model. The Application Layer consists of applications that make use of the network application and presentation layers of the OSI model e.g. if data transferred between two programs is going to be compressed the Application Layer is responsible for compression and decompression. The Transport Layer provides end-to-end data delivery. OSI's Session and Transport Layers fit into this layer. OSI's session connection can be compared to TCP/IP's socket mechanism.

A socket is one end of two way communications link between two programs running on a network and is a low level connection. The client and server both communicate through a stream of bytes written to their sockets. Before communication the client and server must agree on a protocol, that is, they must agree on the language of the information transferred back and forth through the socket. In this case the protocol is TCP which inherits the behaviour of the Transport layer from the OSI model.

TCP/IP socket is an end-point of communications composed of a computers address and a specific port on that computer. TCP provides for reliable data delivery and guarantees that packets of data will arrive in the order they were sent, with no duplication and with no data corruption. The Internet Layer defines datagrams and handles their routing. A datagram is the packet of data manipulated by the IP protocol. It contains the source address, destination address and data as well as other control fields. This is equivalent to OSI's Network and Datalink Layers. IP is analogous to the network layer. It is responsible for encapsulating the underlying network from the upper layers. It also handles addresses and delivery of datagrams. In the Physical Layer TCP/IP makes no effort to define the underlying network physical connectivity. Instead, it makes use of existing standards provided by IEEE, which defines RS232, Ethernet and other electronic interfaces used in data communications.

When a packet is sent, it travels to the Transport Layer where the transport header is added. The Internet Layer then adds its header, and finally the Physical Layer attaches its header. When a packet is received, the process is reversed [32].

3.4 TCP/IP addressing schemes

TCP/IP has a universal addressing scheme [33]. Each computer on a TCP/IP network has an address that uniquely identifies it. Its IP's responsibility is to deliver datagrams among the TCP/IP network's computers. Each computer has an unique IP address composed of a 32 bit number. The IP address contains enough information to uniquely identify a network and a specific computer on the network.

A computer's IP address must uniquely identify not only the computer but also the network the computer is attached to, the IP address is split between a network identifier (net id) and a host identifier (host id) part, as shown in Figure 3.3. The split between these two identifiers is not the same for all addresses. The class of address determines how many bits of IP address are reserved for net id and how may are reserved for host id.

Figure 3.3: IP Address

There are three classes and each class has its own class id. Class A has a 0 in bit 31. The net id is from bit 24 to 30, and the host id is from bit 0 to 23. Class B has a 1 in bit 31, and a 0 in bit 30. The net id is from bit 24 to 29, and the host id is from bit 0 to 23. Class C has a 1 in bit 31, and a 1 in bit 30. The net id is from bit 24 to 28, and the host id is from bit 0 to 23.

An IP address is represented by four decimal numbers from 0 to 255 separated by a (.) e.g. 166.78.4.139 which is 10100110 0100110 00000100 10001011 so that the net id is 166.78 and host id 4.139. This is a class B addressing model.

A Name Server is a computer which provides a name to an IP address. When a request to translate a name to its IP address arrives at the name server, it checks the database to see if this information is there.

Network hardware does not understand IP addresses and so Address Resolution Protocol (ARP) is used to map IP address and host names into a physical address that the network hardware understands. A message is transmitted to check if the computer with the IP address exists and if it is listening. If so, it will return a message with its physical hardware address to the source. Any other computer

ignores the request message. This protocol only works on the local network because the format of the physical address is dependent on the hardware used in the network [31] e.g. Ethernet.

3.5 Data transmission

TCP verifies that data is delivered in order and without corruption. There is extra overhead and maintenance involved in this connection. Reliability comes from inclusion of a checksum with each packet of data transmitted. When the packet is received the checksum is generated and compared to the checksum included in the header of the data packet. If checksums do not match, this is communicated to the sender and the data is re-sent. We do not have to be concerned with this function because lower layers mask it. TCP is connection-oriented because two end-points of communications exchange a handshaking dialogue before data transmission can begin. This handshake guarantees that the receiver is ready to accept data [34].

A socket is simply an end-point of communication. A TCP/IP socket is comprised of an IP address and a port. Some ports are reserved for well-known services and others for use by applications. Sockets may be set up to provide either a reliable connection-oriented stream service or an unreliable connectionless datagram service.

The most reliable stream socket is based on TCP. This requires that a connection be established before two processes can send or receive data. Data is then sent in a stream of bytes. A connection-oriented stream service is best suited to client/server architecture. In client/server interaction, the server creates a socket, gives the socket a name and waits for clients to connect to the socket. The client creates a socket and connects to the named socket on the server. When the server detects a connection to the named socket, it creates a new socket and uses that new socket for communication with the client. The servers named socket waits for connections from other clients.

CHAPTER 4
IMPLEMENTATION OF COMPRESSION ALGORITHMS

4.1 Objectives

This chapter first discusses the implementation of the objective of the project. Next the language that the software application is written in is chosen. The lossless algorithms are then described: Huffman coding, arithmetic coding and Huffman coding using splay trees. These algorithms are investigated using the test images and an appropriate algorithm is chosen. Next the lossy algorithm implemented, transform image coding using the DCT, is discussed. Several modifications to this algorithm are investigated and the results are discussed. Finally the Internet software application is then discussed using these algorithms and their transmission over a LAN are investigated.

The objective of this project is to develop a software application using appropriate lossless and lossy algorithms for archiving or transmitting medical images over the Internet. The application should have a user friendly Graphic User Interface (GUI) and so the application should be preferably Windows-based and not DOS-based, since Windows is more user-friendly than DOS. Also the compression algorithms should be invisible to the user. The user should not need to have knowledge of how the actual algorithms work

Figure 4.1 shows the flow chart for the application. The algorithm for archiving is lossless and for transmission is lossy. X-ray images are archived on the server PC and when a client request is received, an X-ray image is chosen, decompressed from the server database using the lossless algorithm, compressed using the lossy algorithm at a default resolution and then transmitted to the client over the Internet where it is decompressed and viewed on the monitor.

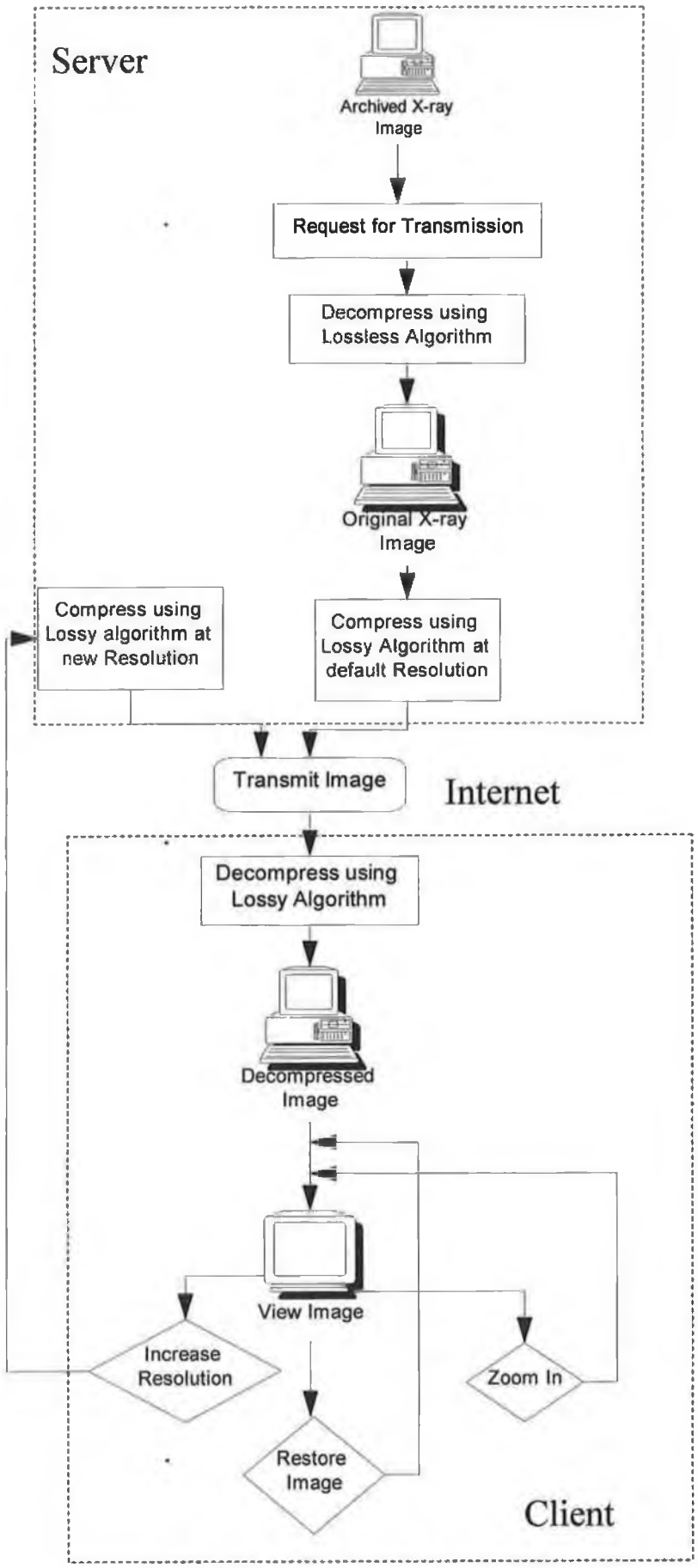


Figure 4.1: Flow chart of application

The user can then zoom in on a section of the image and increase the resolution of that area. This section only of the image is transmitted at the new resolution to the client where it is viewed. The user can then restore to the full X-ray at default resolution and view another section of the image at increased resolution. This is to ensure that only the section that is of interest is sent to the user at a required resolution, so that time is not wasted in sending the whole image at a high resolution when only a small section is required.

4.2 Choice of language environment

An early decision that had to be made was the language in which the application was developed. Initially MATLAB® [35] was investigated because it is a high level modular language. This language had the advantage that image processing functions were already available and that a GUI could be developed in a Windows environment.

A GUI was developed in MATLAB® to compress and decompress an image using a DCT lossy algorithm for different image resolutions. It was seen to be computationally very slow in compressing or decompressing and it was concluded that developing an application using MATLAB® would not be practical. For example, the compression of a 512 x 512 x 8 bit image took approximately 35 min on a 66 MHz 486 PC with 8 Mb of RAM.

C was then considered as an environment in which to develop the compression algorithms. The advantage of C is that algorithms developed in this environment are computationally fast, but, since C is a DOS-based language and part of the overall objective is to develop a user friendly GUI, the actual front end interface could not be developed in C. Visual C++ was the obvious environment in which to develop a fast Windows-based application. The algorithms developed in C could be expanded and developed in Visual C++. This route was investigated but Visual C++ was seen to be too complex to develop a simple front end interface when Visual Basic could achieve this with less complexity.

Consequently the algorithms for lossless and lossy compression were written in C, imported into Visual C++ and developed into a Dynamic Linked Library

(DLL). The front end GUI was then developed in Visual Basic and the functions for lossy and lossless compression were declared in Visual Basic and are transparent to the user.

4.3 Test images

The five images shown in Figures 4.2 to 4.6 are the images on which the lossless and lossy algorithms were tested. The three X-ray images, wrist, chest and pelvis were obtained from the Medical Imaging Department in St. James' Hospital, Dublin. The final two images, Mandrill and Lenna, are standard test images. The test images are 1024 x 1024 x 8 bits.

The standard format for storing and viewing bitmap images in Windows applications is the bitmap (bmp) file format [36]. To view an image in Visual Basic the image intensity data needs to be transformed to a bmp file. So programs for reading and writing bitmaps were written in C and developed into DLLs where they could then be used in Visual Basic. Appendix C gives the header structures for the bitmap format and a description of the transform from a single intensity matrix to the bmp RGB format.

Figure 4.2: Wrist X-ray image

Figure 4.3: Pelvis X-ray image

Figure 4.4: Chest X-ray image

Figure 4.5: Standard Mandrill image

Figure 4.6: Standard Lenna image

Figures 4.7 to 4.11 are the histograms of the test images. The histograms show the number of pixels per intensity in an image. These demonstrate the frequency of intensities appearing in an image. The probability distribution can be interpreted from the histograms. The higher the number of pixels per intensity, the greater the probability. The lossless algorithms are based on this probability, where the codes generated by the lossless algorithm depend on each probability. In Figure 4.7 it can be seen that two intensities have a high probability since they appear more often than others. This makes the image easier to compress. Compare this to Figure 4.11 where the intensities are distributed more evenly making this image harder to compress.

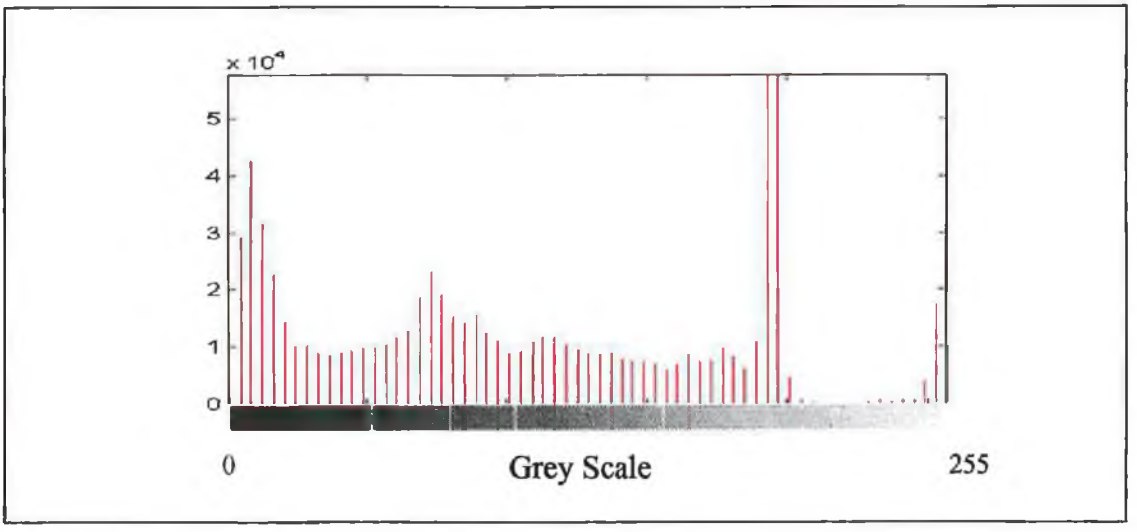


Figure 4.7: Histogram of wrist X-ray image

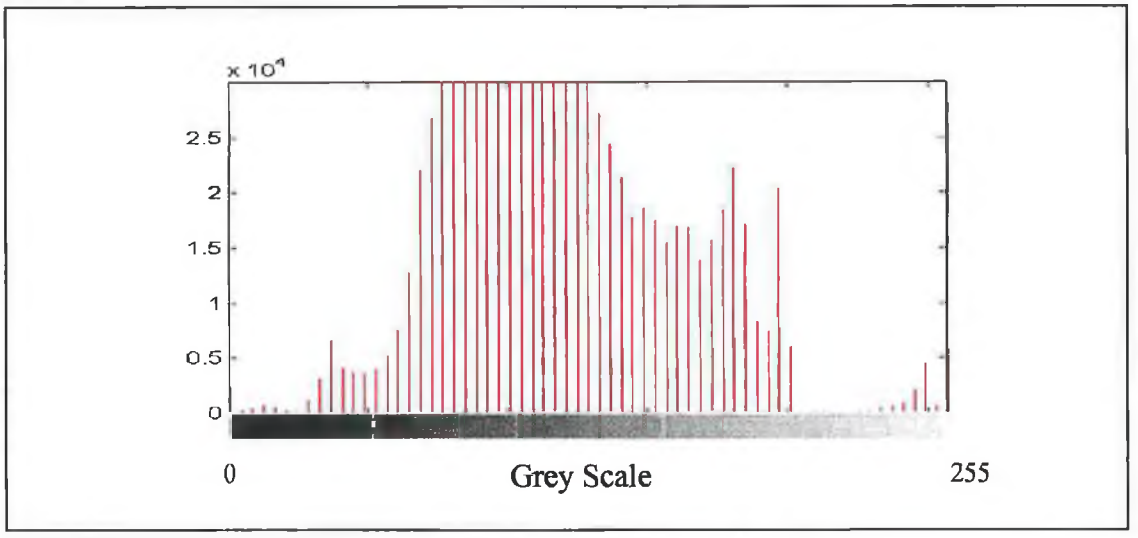


Figure 4.8: Histogram of chest X-ray image

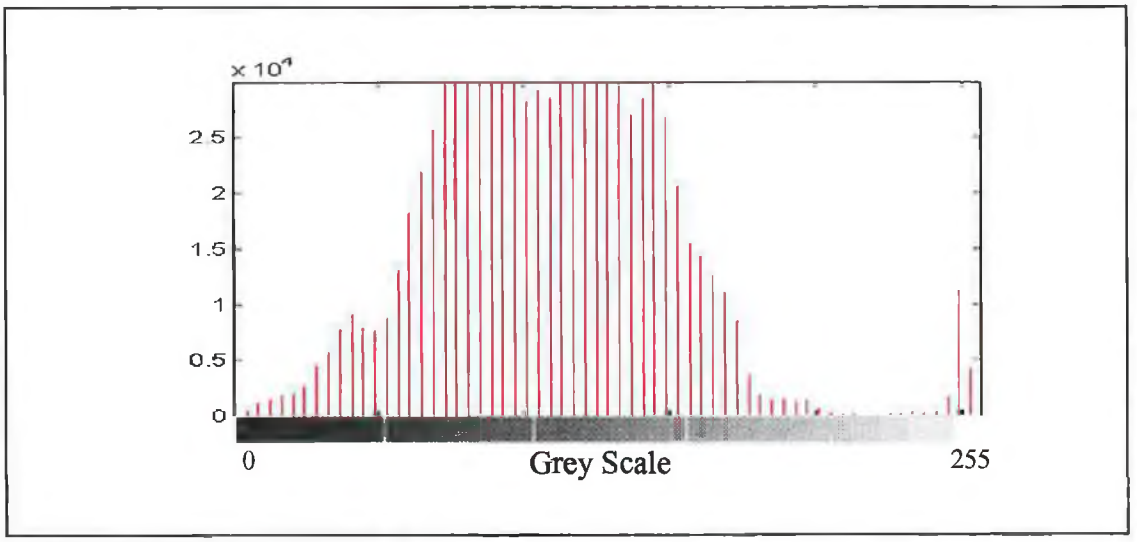


Figure 4.9: Histogram of pelvis X-ray image

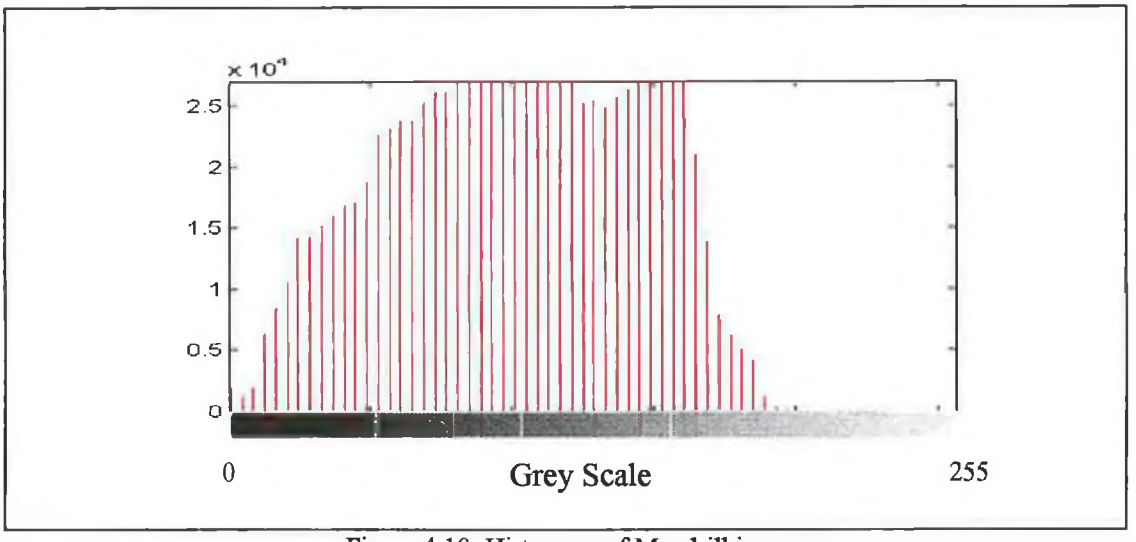


Figure 4.10: Histogram of Mandrill image

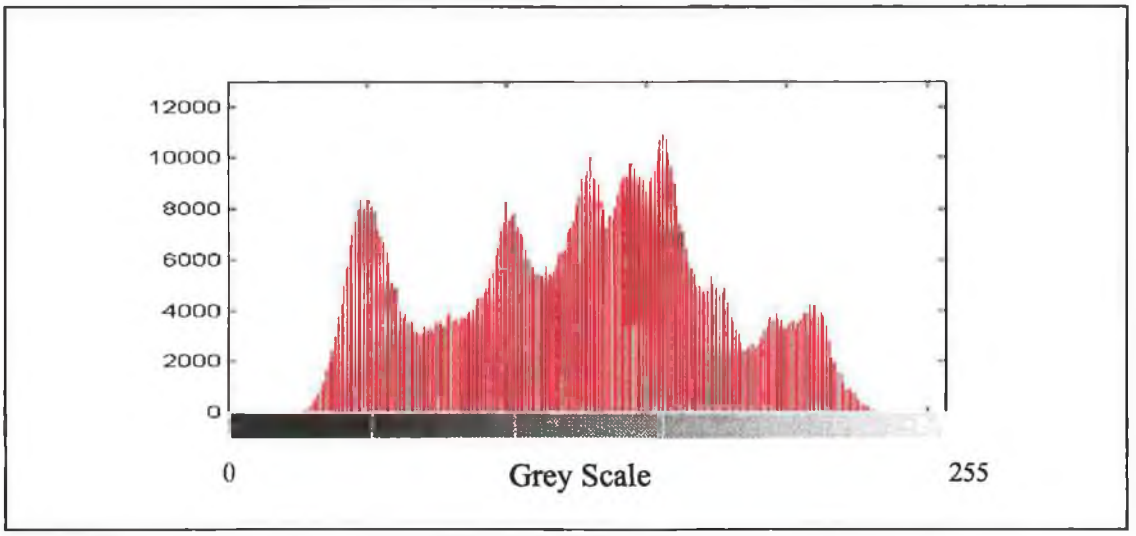


Figure 4.11: Histogram of Lenna image

4.4 Implementation of lossless compression algorithms

Lossless algorithms investigated were static Huffman coding, adaptive huffman coding, arithmetic coding and Huffman coding using splay trees. These algorithms were investigated due to their computational speed and simplicity [37].

4.4.1 Implementation of static Huffman coding

Figure 4.12 shows the flowchart for the static Huffman encoding algorithm [38, 39, pp. 186-195]. Section 2.2.1 gives an explanation of Huffman coding. The probability of each intensity in the image is determined and this probability is the weight at each node, where the intensity is the node. Leaves in the Huffman tree are the starting nodes.

To decode, the decoder needs a copy of the Huffman tree identical to the one used by the encoder. Therefore a header is passed to the output file. The header contains intensity counts, which consists of runs of counts. A count-run consists of values of the first intensity in the run, followed by the value of the last intensity in the run, followed by the counts for all the intensities in the run for first and last. The output file is a stream of bits which has been created from the intensities corresponding Huffman codes.

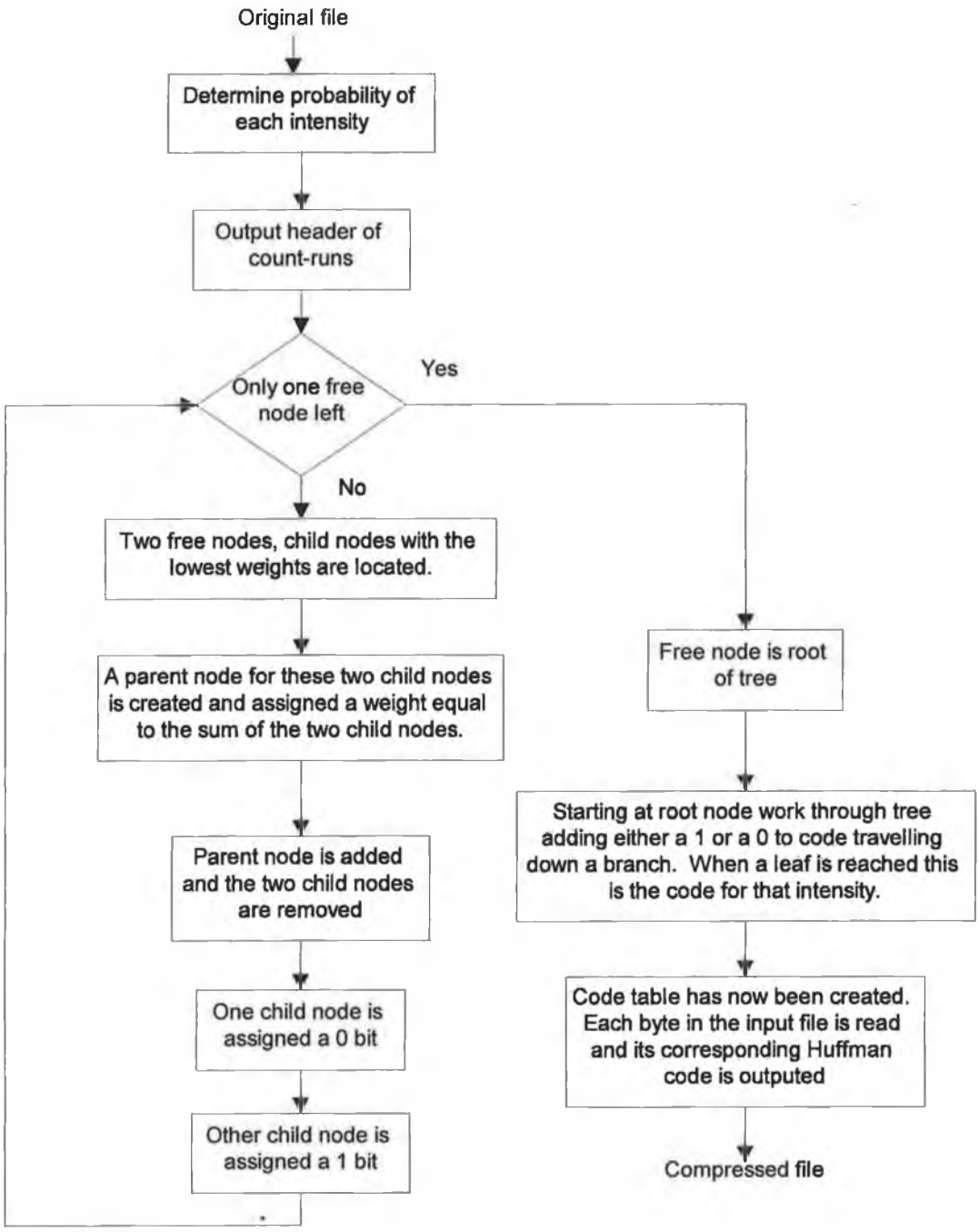


Figure 4.12: Flowchart of static Huffman encoding algorithm

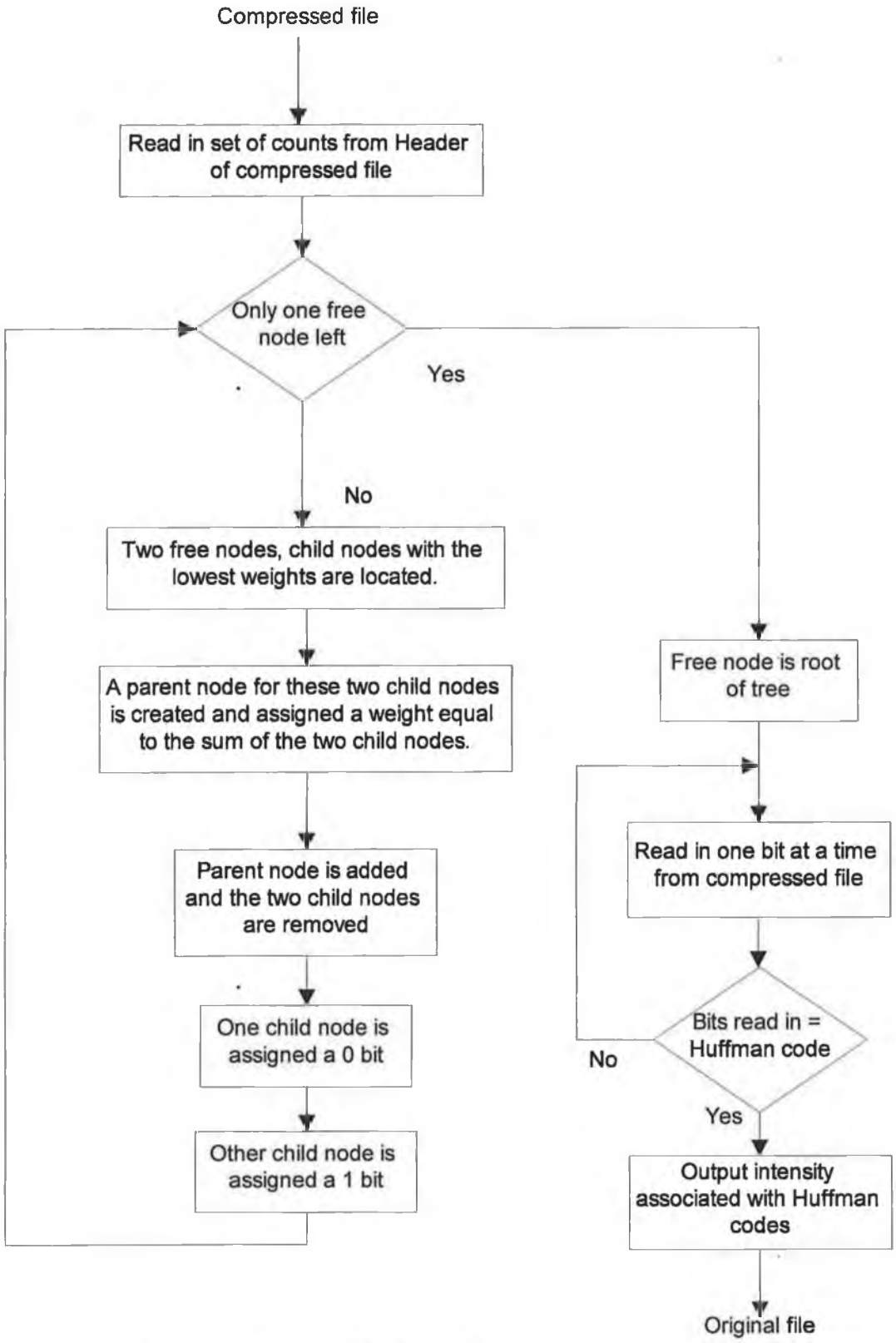


Figure 4.13: Flowchart of static Huffman decoding algorithm

Figure 4.13 shows the flowchart for the static Huffman decoding algorithm. The Huffman tree is built from the header read in from the compressed file. The original file can be reconstructed from the Huffman codes read in from the compressed file.

For the static Huffman encoding algorithm the Huffman tree has to be passed to the decoder ahead of the compressed code stream. In order for the decoder to function correctly it requires foreknowledge of the probability table code assignments.

4.4.2 Implementation of adaptive Huffman coding

For adaptive Huffman coding, statistics are continually modified as new intensities are read in and coded [39, pp. 217-241]. The Huffman tree is adjusted continuously based on data previously seen. The adaptive Huffman encoder and decoder start with identical models so when the encoder puts out the first encoded intensity, the decoder will be able to interpret it.

Figure 4.14 gives the adaptive Huffman encoding algorithm. Each intensity is read in and if the intensity is not already found in the Huffman tree the unencoded intensity is output together with an escape code so that the decoder will know what intensity to add to the table. The escape code is a code to inform the decoder that an intensity to be used to reconstruct the Huffman tree is being transmitted. The intensity is then added to the tree as a leaf node. Encoding of the intensity is performed by starting at the leaf node and moving through the parent nodes to the root node assigning a 0 bit to one child node and a 1 bit to the other child node. Starting at the leaf node each bit is added to the Huffman codeword.

The Huffman tree is reconstructed at the decoder so that the Huffman codes can be associated with the corresponding intensities. Figure 4.15 shows the adaptive Huffman decoder algorithm.

If the escape code is read in and decoded, the escape code is rejected and the unencoded intensity is read in and added to the Huffman tree as a leaf node. Then the update model is reconstructed as the compressed file is read. As the Huffman codes are read in the corresponding intensity is output.

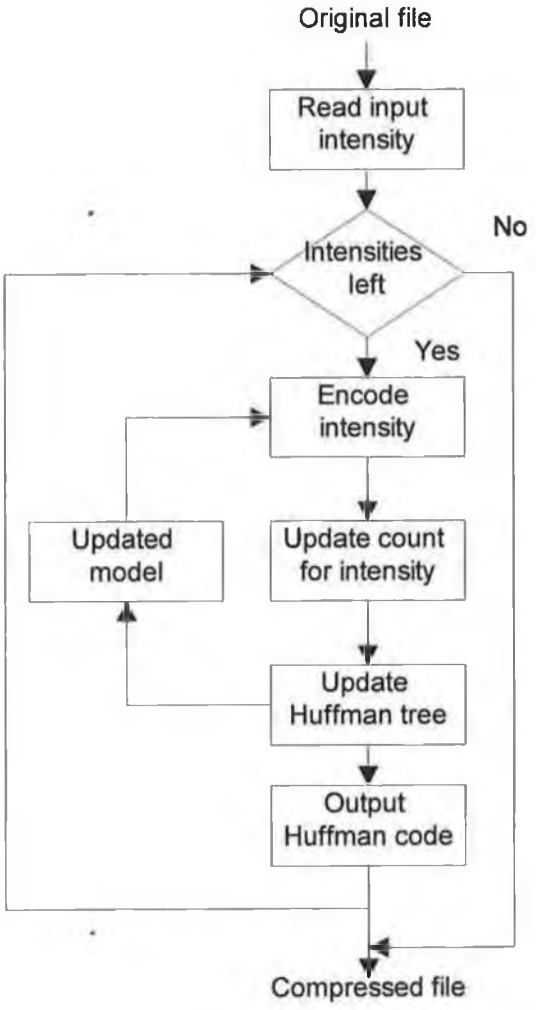


Figure 4.14: Flowchart of adaptive Huffman encoding algorithm

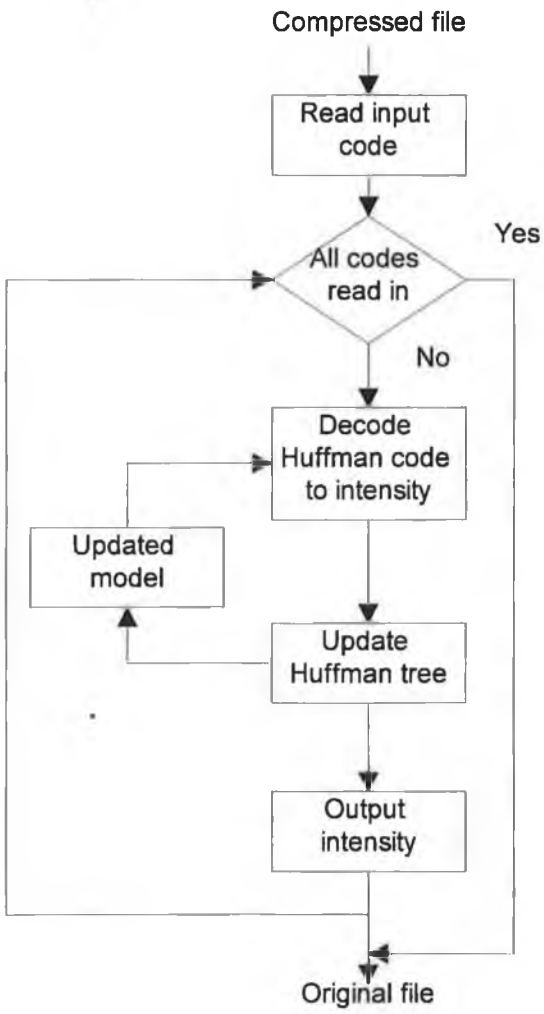


Figure 4.15: Flowchart of adaptive decoding algorithm

4.4.3 Implementation of Huffman coding using splay trees

Figure 4.16 shows the Huffman coding using splay trees encoding algorithm [40]. Section 2.2.3 gives an explanation of splay trees and shown in Appendix A is an example of splay trees when applied to Huffman coding. An initial balanced code tree is generated which is used for the encoding and decoding algorithm. The intensity read in is encoded using the current Huffman code tree and then output to the compression file. Encoding is performed by following the path from the leaf to the root of the tree. The code bits are then reversed since they are in the reverse order to which they will be transmitted. After each intensity is encoded using the current version of the code tree, the tree is then splayed around the code for that intensity. If an intensity is read in that has not be seen before, it will be sent to the compression

file with an escape code so that the decoder can build up the Huffman code tree as each new intensity is read in.

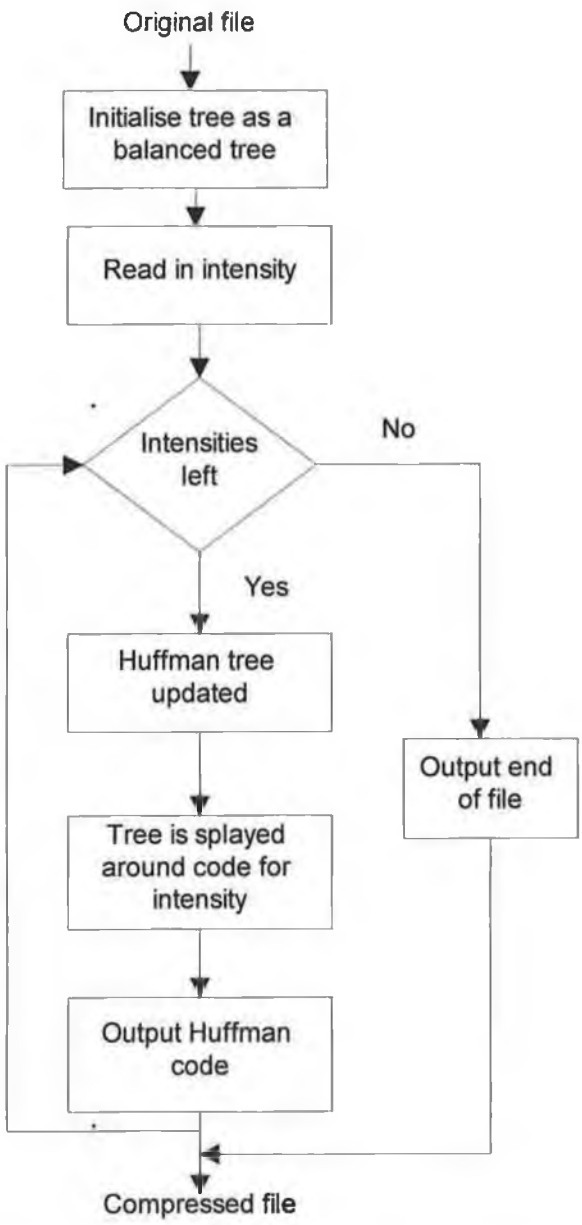


Figure 4.16: Flowchart of Huffman coding using splay trees encoding algorithm

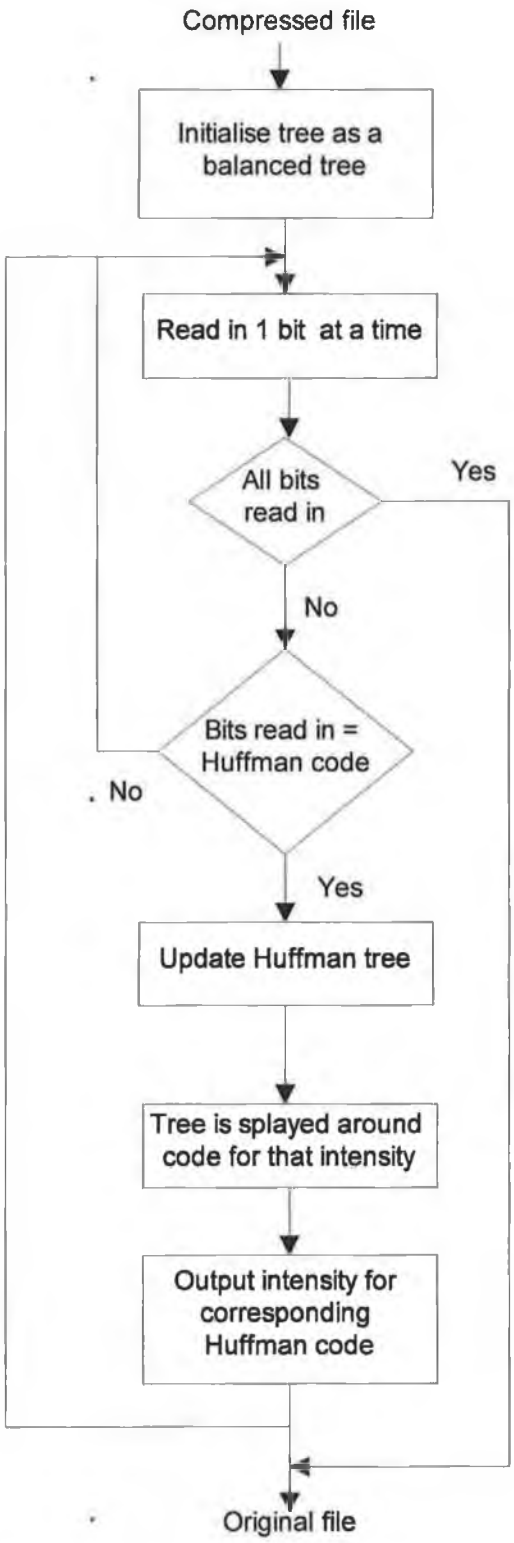


Figure 4.17: Flowchart of Huffman coding using splay trees decoding algorithm

Figure 4.17 shows the Huffman coding using splay trees decoding algorithm. As the codes are read in, if an escape code is read in then the next code is an intensity which is to be added to the Huffman tree. As each code which is not new, the

corresponding intensity is output and the tree is splayed to reconstruct the Huffman tree that had been generated at the encoding end.

4.4.4 Implementation of arithmetic coding

Figure 4.18 gives the algorithm for arithmetic encoding [41]. Section 2.2.2 gives an explanation of arithmetic coding.

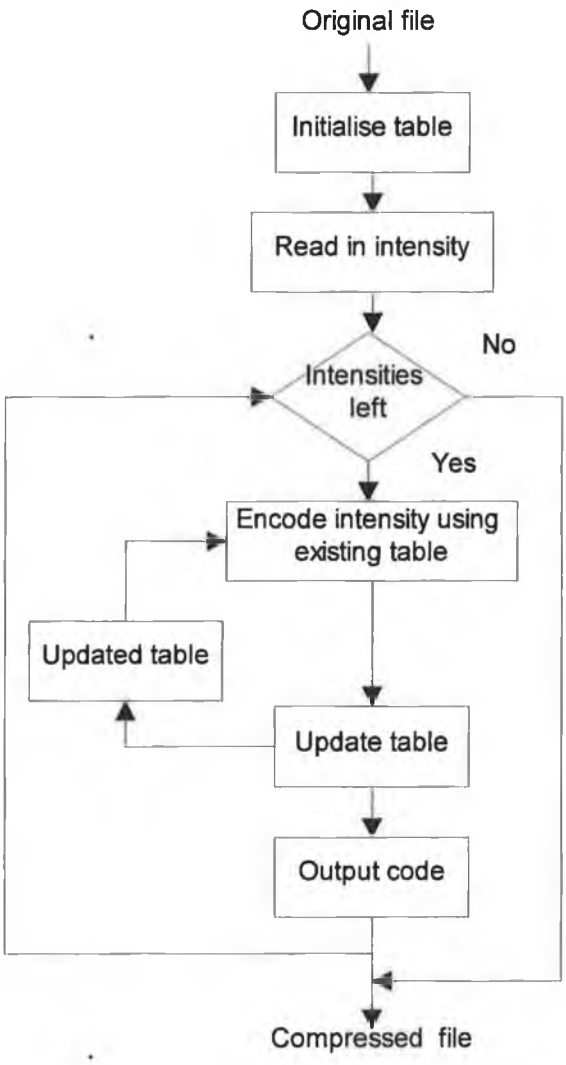


Figure 4.18: Flowchart of arithmetic encoding algorithm

The table which contains the ranges are only added to the table as they appear. The encoder and decoder start with the same model, where the model is the table of ranges. Each intensity is read in and if the intensity is not already found in

the table then an escape code and a predefined code for the intensity is output. This is so that the decoder can build the table at the decoding end. This is to ensure that the intensity is added to the table.

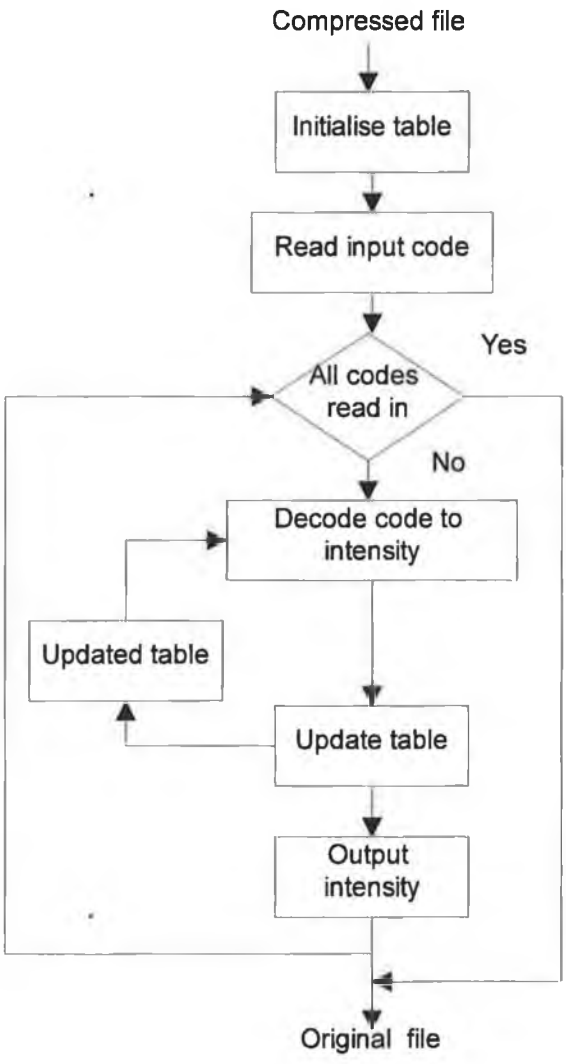


Figure 4.19: Flowchart of arithmetic decoding algorithm

The escape code is a code to inform the decoder that an intensity to be used to reconstruct the table is being output. The intensity is then encoded using the existing table, that is, the high and low ranges. The table is then updated to account for the new intensity.

Figure 4.19 shows the flowchart for the arithmetic decoding algorithm. The table is reconstructed at the decoder so that the codes read in can be associated with the corresponding intensities.

If the escape code is read in and decoded then the corresponding intensity is added to the table. The update table is reconstructed as the compressed file is read, so that as the codes are read in the corresponding intensity is output.

4.5 Lossless compression results

The performance of a number of lossless algorithms were tested: static Huffman coding, adaptive Huffman coding, arithmetic coding and Huffman coding using splay trees. The algorithms were applied to each of the test images and in each case the compression ratio, compression time and decompression time were recorded. The mean square error was recorded for each algorithm in each case and was found to be zero. This was done to verify that the algorithms are indeed lossless, i.e. that no information was lost.

4.5.1 Lossless compression results for 1024 x 1024 images

Table 4.1 gives the results for the lossless compression algorithms when applied to the wrist X-ray image. Appendix D shows the tabulated results for the other test images.

	Static Huffman	Adaptive Huffman	Splay Trees	Arithmetic
Compression ratio	1:1.69	1:1.93	1:3.40	1:8.63
486, 66 MHz				
Compression Time /s	25.46	40.51	19.68	297.45
Decompression Time /s	24.31	35.88	15.05	333.33
Pentium, 90 MHz				
Compression Time /s	9.26	12.73	4.63	76.39
Decompression Time /s	9.26	12.73	4.63	78.52

Table 4.1: Lossless compression results for wrist X-ray (1024 x 1024) image

The Decompression Rate, defined by

$$\text{Decompression Rate} = \frac{\text{Compression Ratio}}{\text{Decompression Time}} \quad \ldots\ldots\ldots\ldots\ldots\ldots 4.1$$

is a figure of merit for the compression / decompression process. Since the image must be decompressed from the archive before it is to be transmitted the trade-off

between decompression time and compression ratio is crucial. The decompression time is approximately equal to the compression time for algorithms tested as can be seen in the tables.

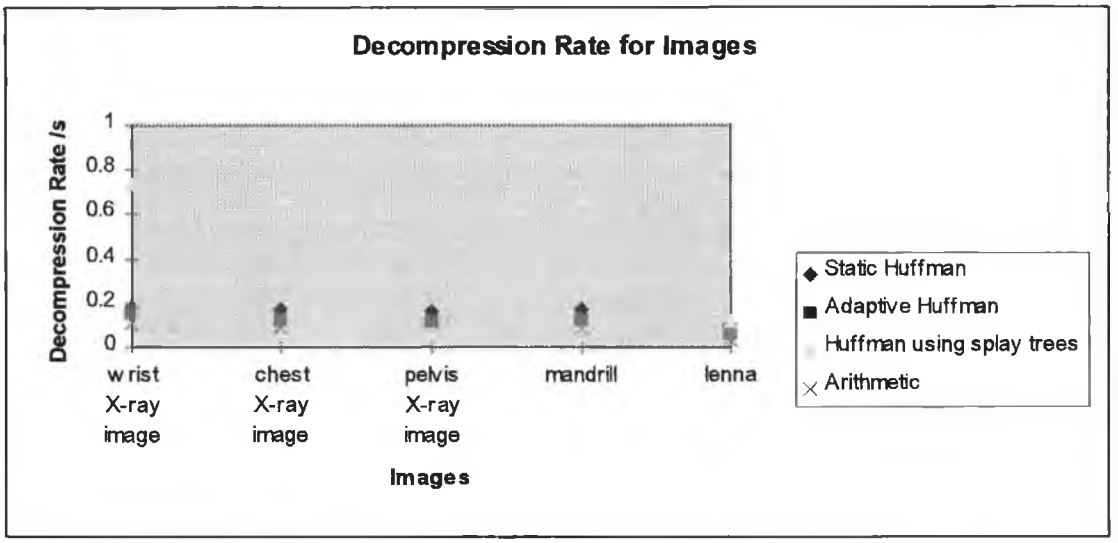


Figure 4.20: Decompression Rate for 1024 X 1024 images

Figure 4.20 gives the decompression rate for each image tested. From this figure it is clear that the decompression rate for Huffman coding using splay trees is high compared to the other algorithms tested for all images tested. So this algorithm has a high compression ratio with a short decompression and compression time.

Figure 4.21 shows the compression ratio plotted against compression time for the test images, on a 66 MHz 486 and 90 MHz Pentium. It can be seen that as the processing power of the PC increases the decompression and compression times decrease. The arithmetic coding algorithm produces the highest compression ratio but with a long compression and decompression time. The Huffman coding algorithm using splay trees produces a high compression ratio with a relatively short compression and decompression time.

The Huffman coding using splay trees algorithm performs better than the static Huffman coding because it does not have to scan the data twice in order to build the tree. Its performance is superior to adaptive Huffman coding algorithm since by splaying it reduces the distance from the root to the leaves, and this increases the compression ratio and decreases the compression time. It performs

better than arithmetic coding since there are no floating point numbers involved. Therefore the Huffman coding using splay trees produces a high compression ratio with short compression and decompression time compared to the other algorithms. This algorithm was therefore selected as the lossless algorithm for use in the Visual Basic archiving and remote diagnostics software application.

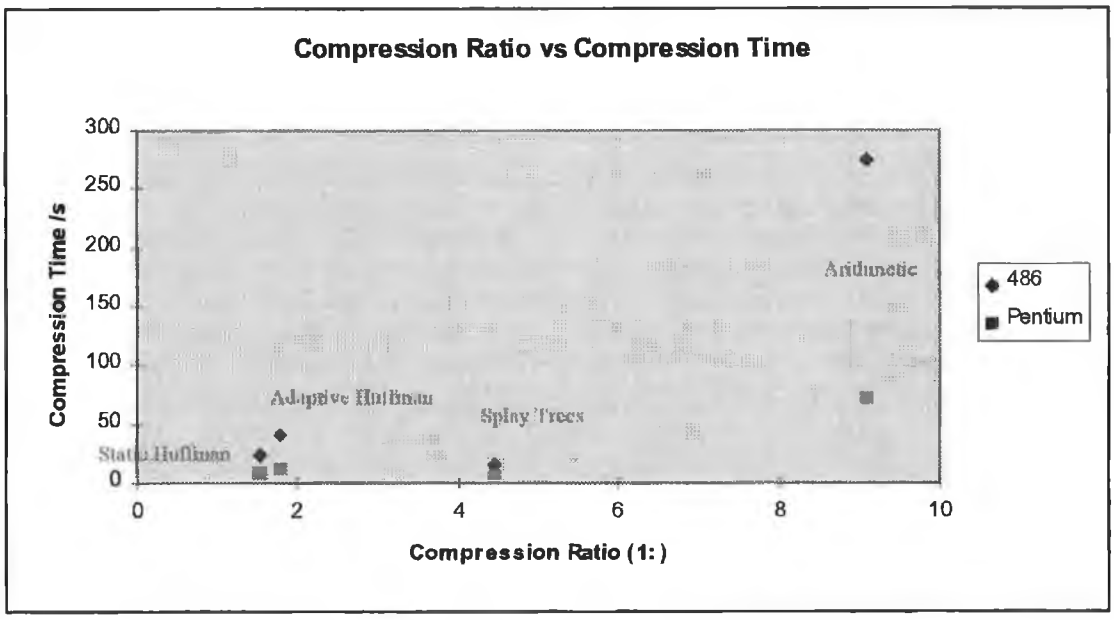


Figure 4.21: Compression Ratio vs. Compression Time for 1024 x 1024 images

4.5.2 Lossless compression results for 512 x 512 images

The lossless compression algorithms were tested on a 512 x 512 x 8 bit wrist X-ray image to investigate if the compression time, decompression time and compression ratio were adequate for smaller images. The results are shown in Table 4.2. Appendix D shows the corresponding results for the other test images.

	Static Huffman	Adaptive Huffman	Splay Trees	Arithmetic
Compression ratio	1:1.68	1:1.84	1:2.66	1:6.03
486, 66 MHz				
Compression Time /s	9.26	13.89	5.79	90.28
Decompression Time /s	9.26	13.89	5.79	100.69
Pentium, 90 MHz				
Compression Time /s	4.79	5.79	1.16	25.46
Decompression Time /s	4.79	5.79	1.16	26.79

Table 4.2: Lossless compression results for wrist X-ray (512 x 512) image

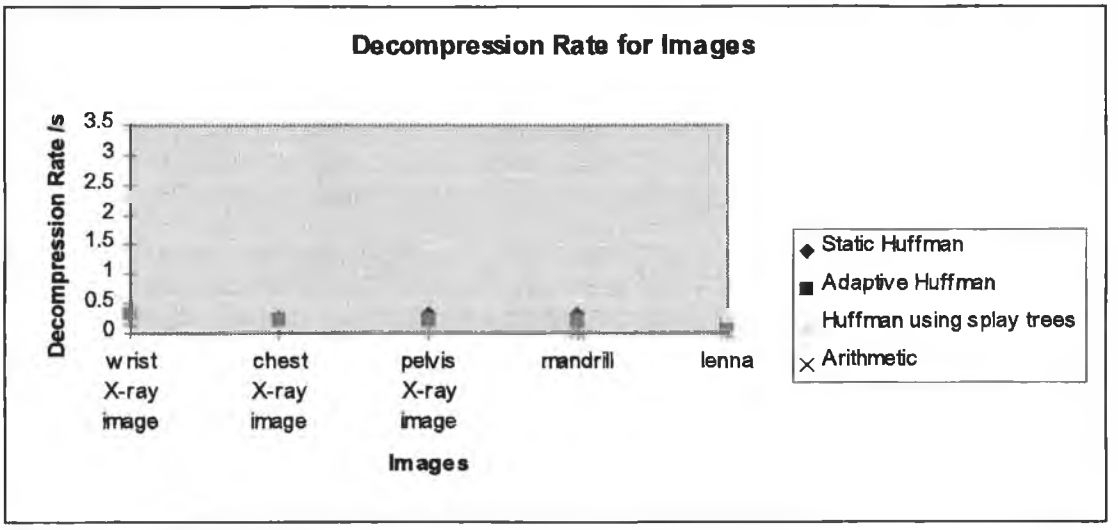


Figure 4.22: Decompression rate for 512 x 512 images

Again decompression rate was calculated for each image and is shown in Figure 4.22. As can be seen Huffman coding using splay trees gives the highest decompression rate for each image compared to the other algorithms in the case of smaller images.

Figure 4.23 shows the compression ratio versus compression time for the test images on average. As can be seen arithmetic coding gives the best compression ratio but with a long compression and decompression time. Again Huffman coding using splay trees gives a high compression ratio with short compression and decompression times and is therefore the lossless algorithm to be used even with smaller images.

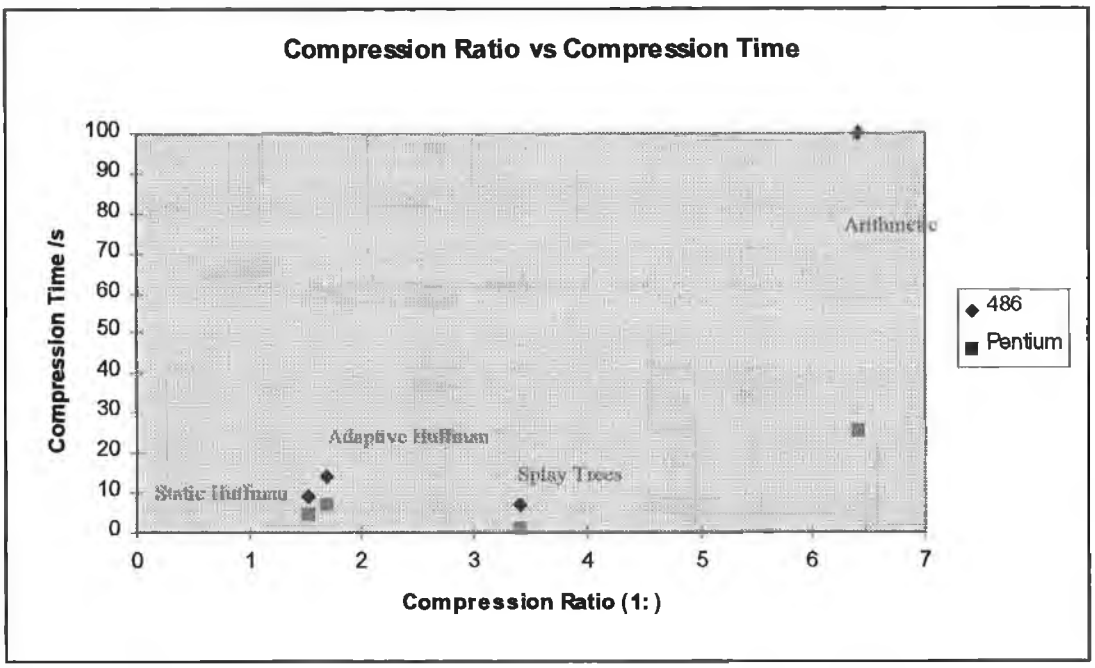


Figure 4.23: Compression ratio vs. compression time for 512 x 512 images

4.6 Implementation of lossy compression algorithm

The lossy algorithm chosen because of its simplicity and computational speed was transform image coding using the DCT [42]. Figure 4.24 shows the flowchart for the DCT encoding algorithm. Section 2.3.1 provides an explanation of transform image coding using the DCT. When using the DCT as a transform the calculation time has to be considered. The calculation time required for each element in the DCT is heavily dependent on the size of the matrix. Therefore the image that is to be transformed is divided into a number of sub-images of size N x N [43]. The sizes of the sub-image investigated were 8 x 8, 16 x 16 and 32 x 32. As the size of the sub-image increases so too does the time required to calculate the DCT of the image. Depending on the degree of correlation present between the pixels, the compression ratio may increase.

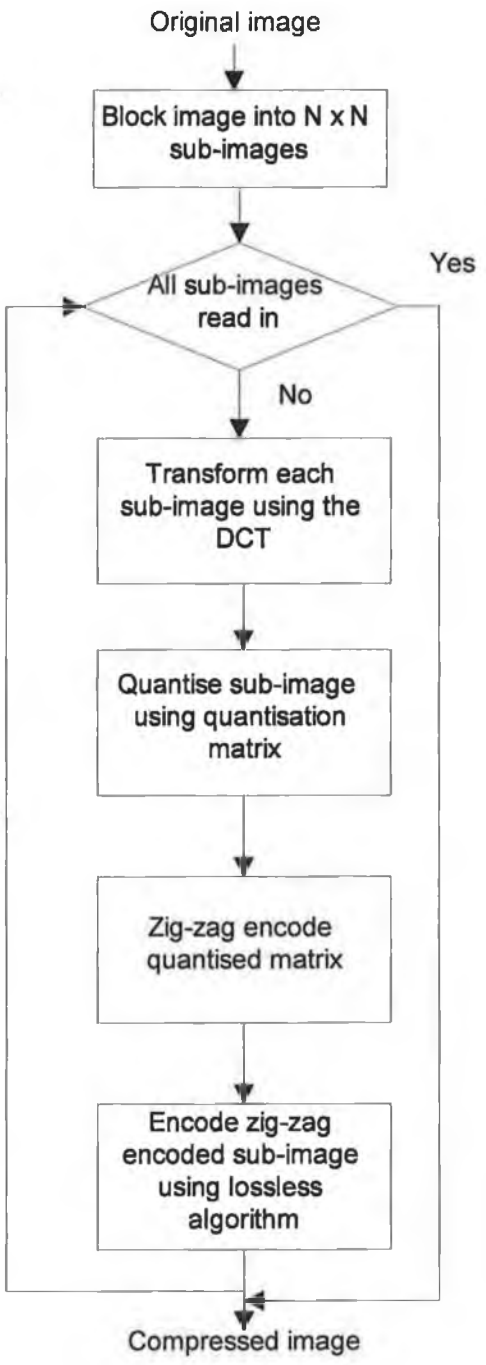


Figure 4.24: Flowchart of DCT encoding algorithm

The image has therefore been broken into smaller, more manageable blocks. Figure 4.25 shows an 8 x 8 sub-image from the wrist X-ray image.

99	99	115	115	115	115	99	99
99	99	115	115	115	99	99	115
99	115	115	115	115	99	99	99
99	115	115	115	115	99	115	115
99	115	115	115	115	99	99	115
115	115	115	115	99	99	99	99
99	115	115	115	99	99	99	115
99	115	99	99	99	99	99	99

Figure 4.25: 8 x 8 sub-image from wrist X-ray image

Before transforming the sub-image it is scaled between -128 and 127 by subtracting 128. This sub-image is then transformed to the frequency domain using the DCT [44] and Figure 4.26 shows the DCT transform of the 8 x 8 sub-image from the wrist X-ray image.

-168	14	-19	-30	4	-11	-5	-1
9	-10	-20	3	7	2	7	9
-16	-3	-2	10	-3	2	9	-5
5	4	-3	1	3	6	11	-4
0	-9	-2	7	-4	-5	-4	2
7	-2	-2	-2	-6	4	-1	-2
-6	15	-3	14	-7	4	-2	-2
1	1	-7	-3	0	-8	1	3

Figure 4.26: DCT of 8 x 8 sub-image from wrist X-ray image

As can be seen in Figure 4.26, most of the energy is contained in the d.c. and other low frequency coefficients. The DCT output takes more space to store than the original matrix of pixels, since the largest coefficient value in the DCT output requires a precision of more than 8 bits, and these coefficients must be quantised.

Quantisation is the process of reducing the number of bits needed to store a value by reducing the precision of the value. The precision of the coefficients is reduced as one moves away from the d.c. coefficient at the origin. The further one moves from the d.c. coefficient, the less contribution is made to the image, and so the less care needs to be taken in maintaining rigorous precision in its value. Quantisation is implemented using a quantisation matrix, using zonal coding.

$$\text{Quantised Coefficient} = \text{ROUND}\left(\frac{\text{DCT Coefficient}}{\text{Quantisation Matrix Coefficient}}\right) \quad \ldots\ldots\ldots\ldots 4.2$$

Quantisation matrices allow for run-time selection of quality as shown in Appendix E. This allows the user to select the quality required which in turn will select the quantisation matrix.

Figure 4.27 shows a quantisation matrix used to quantise the DCT transformed coefficients. This quantisation matrix is based on the human visual system's sensitivity. The quantisation levels are the same value all ranging the same distance from the origin [45]. Figure 4.28 shows the quantised transform coefficients.

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

Figure 4.27: 8 x 8 quantisation matrix

-56	2	-2	-3	0	0	0	0
1	-1	-2	0	0	0	0	0
-2	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
0	0	0	0	0	0	0	0

Figure 4.28: Quantised 8 x 8 transform coefficients

Instead of encoding the sub-image from left to right the sub-image is encoded in a zig-zag fashion. This creates longer run lengths when the sub-image is to be encoded. Shown in Figure 4.29 is zig zag encoding of an 8 x 8 size sub-image.

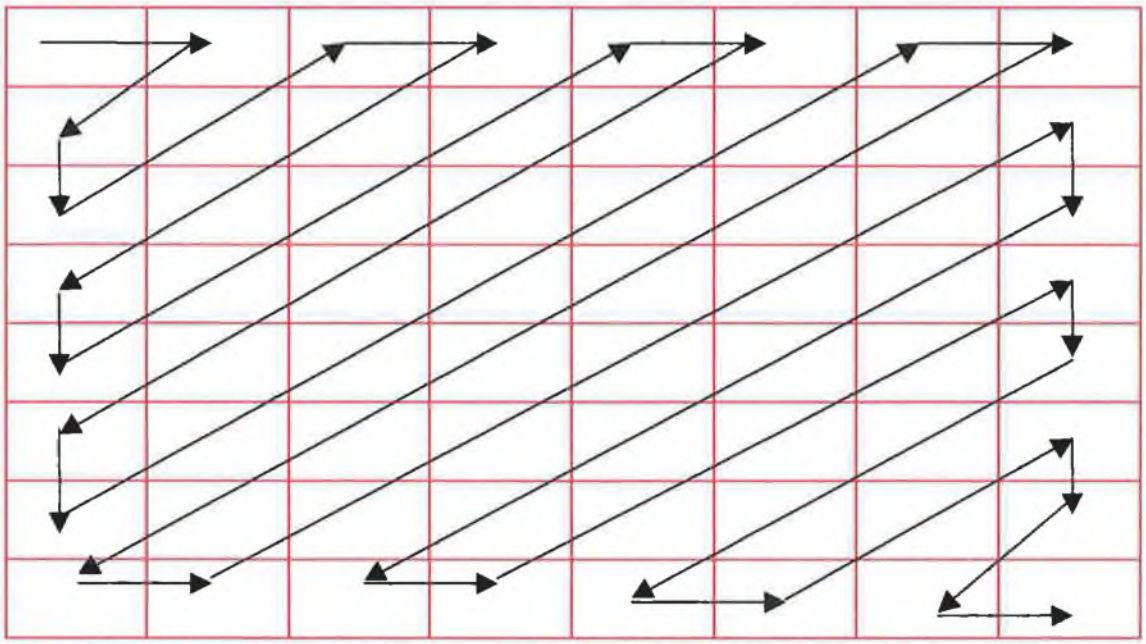


Figure 4.29: Zig-zag encoding of an 8 x 8 sub-image

Shown in Figure 4.30 is the quantised transform coefficients zig-zag encoded

	-56	2	1	-2	-1	-2	-3	2	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	0	0	0	0 →
→	0	0	0	0	0	0	0	0	0	0	0	0	0				

Figure 4.30: 8 x 8 Quantised coefficients zig-zag encoded

Once the transform coefficients have been zig zag encoded the coefficients can be encoded using a lossless compression algorithm. The lossless algorithms tested were run length encoding and Huffman coding using splay trees.

Figure 4.31 gives the flowchart for the DCT decoding algorithm. The codes are read in, and decoded into transform coefficients using the lossless algorithm, for an N x N number of coefficients. In this example when 64 coefficients have been decoded the coefficients were zig-zag decoded to form the 8 x 8 transform coefficients. These coefficients were then dequantised using

DCT Value = Coefficient read in * Quantisation Matrix Coefficient 4.3

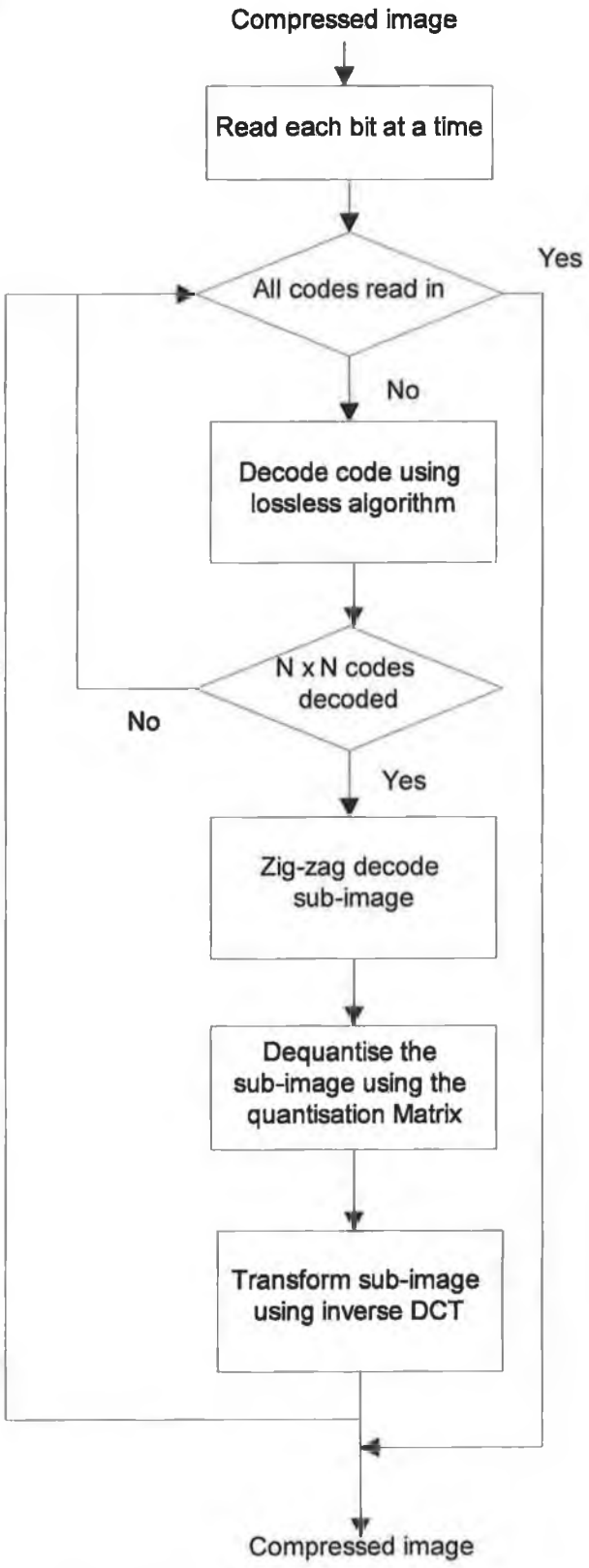


Figure 4.31: Flowchart of DCT decoding algorithm

The quantisation matrix coefficients are given in the quantisation matrix shown in Figure 4.27. When the coefficients have been dequantised the sub-image is transformed to the intensity domain using the inverse DCT transform.. Shown in Figure 4.32 is the reconstructed intensity sub-image.

95	104	113	115	109	104	102	103
97	106	114	115	110	104	103	105
101	108	115	116	110	105	105	107
104	110	116	115	110	105	106	109
106	111	116	114	108	104	106	110
106	111	114	111	104	101	104	109
106	109	111	107	101	98	102	107
105	108	110	105	99	96	101	106

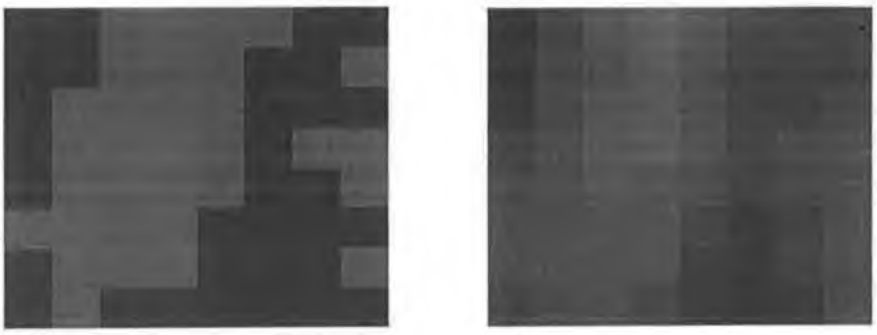
Figure 4.32: Reconstructed 8 x 8 sub-image of wrist X-ray image

As can be seen the reconstructed image is not identical to the original. The loss or error is shown in Figure 4.33 but the difference between the original and reconstructed sub-image is not large.

4	-5	2	0	6	11	-3	-4
2	-7	1	0	5	-5	-4	10
-2	7	0	-1	5	-6	-6	-8
-5	5	-1	0	5	-6	9	6
-7	4	-1	1	7	-5	-7	5
9	4	1	4	-5	-2	-5	-10
-7	6	4	8	-2	1	-3	8
-6	7	-11	-6	0	3	-2	-7

Figure 4.33: Error between original and reconstructed sub-image

Shown in Figure 4.34 is the 8 x 8 section of the wrist X-ray image before it was compressed and after it had been reconstructed from compression. As can be seen there is not a great difference between the original and reconstructed image, but the difference is still visible. If one now looks at the wrist X-ray image, before and after compression one can see that the difference is not apparent when the whole image is viewed in Figure 4.35. Nevertheless the size of the original image was 263,222 bytes and that of the compressed image is 27,789 bytes.



Original sub-image Reconstructed sub-image

Figure 4.34: 8 x 8 before and after compression

Original image

Reconstructed image

Figure 4.35: Wrist X-ray image before and after compression

4.7 Lossy compression results

The lossy algorithm was also developed as a DLL and called in Visual Basic. The algorithm was tested on a 66 MHz 486 and a 90 MHz Pentium PCs. Compression ratio, mean square error, compression time and decompression time were recorded for each image using different quality factors. As the quality factor increases the resolution decreases so that a quality factor of 1 would have the highest resolution. The DCT algorithm was tested using two different encoding schemes, that is, run length encoding and Huffman coding using splay trees, and also using different size quantisation matrices. Appendix E shows the 8 x 8 quantisation matrices used to compress the images. The 16 x 16 and 32 x 32 quantisation matrices have the same shape as the 8 x 8 quantisation matrix.

The application developed uses zonal coding to allow the client the facility to choose the resolution of the image to be viewed. Thresholding was investigated and was seen to give an optimal compression ratio but did not have the flexibility to allow the user to choose differing resolutions. The amount of bandwidth required is decreased since the client can choose a low resolution on a large image and a high resolution on only a small section of that image.

There are two ways of assessing image quality, subjectively and objectively [46]. Subjective assessment is when the image is viewed and any noticeable distortion or degradation compared to the original is perceived as a level of degradation. These subjective levels are shown in Table 4.3. The original and reconstructed images were viewed by six people and a value of degradation was assigned to each reconstructed image from the degradation levels.

An objective measure is a quantitative evaluation which produces a single number. A standard method is the mean square error between the reconstructed image and original image. Quantitative methods take no account of factors such as distribution of the error throughout the image and aspects of human visual perception.

1	No Visual Degradation
2	Low Level of Degradation
3	Visually Degraded
4	Significant Degradation
5	Highly Degraded

Table 4.3: Subjective levels of degradation

4.7.1 8 x 8 quantisation matrix results for 1024 x 1024 images

Table 4.4 shows the results for the lossy compression algorithm for the wrist X-ray image of size 1024 x 1024 pixels using a quantisation matrix of size 8 x 8. The results for the other test images are shown in Appendix F.

	486 66 MHz	Pentium 90 MHz
Compression Time /s	82.69	18.52
Decompression Time /s	83.84 s	17.36

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	1.42	2.82	4.15	5.53	6.88	8.26	20.4
Visual Degradation	1	2	2	2	3	4	5
Using Run Length Encoding							
Compression Ratio	1:9.39	1:11.77	1:12.69	1:13.24	1:13.74	1:14.03	1:15.20
Using Splay Trees							
Compression Ratio	1:8.90	1:12.75	1:13.31	1:14.24	1:14.24	1:16.55	1:18.45

Table 4.4: Lossy compression of wrist X-ray image (1024 x 1024) using 8 x 8 matrix

4.7.2 8 x 8 quantisation matrix results for 512 x 512 images

The lossy algorithm was tested on a 512 x 512 x 8 bit wrist X-ray image to investigate if the compression time, decompression time and compression ratio were adequate for smaller images. The results are shown in Table 4.5. Appendix F shows the results for the other test images.

	486 66 MHz	Pentium 90 MHz
Compression Time /s	22.41	4.63
Decompression Time /s	24.72	4.63

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	2.23	5.26	8.24	11.3	14.1	16.82	36.4
Visual Degradation	1	2	4	4	5	5	5
Using Run Length Encoding							
Compression Ratio	1:7.94	1:10.32	1:11.48	1:12.21	1:12.82	1:13.18	1:14.68
Using Splay Trees							
Compression Ratio	1:7.22	1:10.31	1:11.29	1:12.47	1:13.83	1:14.40	1:16.95

Table 4.5: Lossy compression of wrist X-ray image (512 x 512) using 8 x 8 matrix

4.7.3 16 x 16 quantisation matrix results for 1024 x 1024 images

An investigation using a quantisation matrix of size 16 x 16 on 1024 x 1024 size images to see if there is a substantial increase in compression ratio, while not trading off too much degradation in the reconstructed image produced the results shown in Table 4.6. The results for the other test images are shown in Appendix F. The test images were compressed and decompressed on the Pentium PC. It has already been seen that there is a great reduction in processing time using a Pentium PC, since this processor's architecture is faster when calculating floating point arithmetic.

	Pentium 90 MHz
Compression Time /s	28.94
Decompression Time /s	27.78

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	1.99	3.96	5.72	7.46	9.11	10.57	20.13
Visual Degradation	1	2	2	2	3	4	5
Using Run Length Encoding							
Compression Ratio	1:13.1	1:15.53	1:16.44	1:16.98	1:17.37	1:17.61	1:18.54
Using Splay Trees							
Compression Ratio	1:15.77	1:21.03	1:24.51	1:26.18	1:26.44	1:27.26	1:32.33

Table 4.6: Lossy compression of wrist X-ray image (1024 x 1024) using 16 x 16 matrix

4.7.4 16 x 16 quantisation matrix results for 512 x 512 images

The lossy compression algorithm was tested on a 512 x 512 x 8 bit wrist X-ray image to investigate the use of the 16 x 16 matrix on a smaller image. Results are shown in Table 4.7. Appendix F shows the results for the other test images.

	Pentium 90 MHz
Compression Time /s	6.94
Decompression Time /s	6.94

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	3.71	9.42	13.97	18.28	21.81	26.18	51.19
Visual Degradation	1	2	3	4	5	5	5
Using Run Length Encoding							
Compression Ratio	1:10.79	1:13.71	1:14.87	1:15.61	1:16.13	1:16.46	1:17.82
Using Splay Trees							
Compression Ratio	1:11.44	1:16.62	1:19.70	1:21.58	1:22.43	1:23.46	1:28.36

Table 4.7: Lossy compression of wrist X-ray image (512 x 512) using 16 x 16 matrix

4.7.5 32 x 32 quantisation matrix results for 1024 x 1024 images

Now using a quantisation matrix of 32 x 32 on 1024 x 1024 images to again determine if an increase in the size of the quantisation matrix will increase the compression ratio while not substantially increasing the degradation yielded the results are shown in Table 4.8. Appendix F shows the results for the other test images.

	Pentium 90 MHz
Compression Time /s	53.24
Decompression Time /s	52.08

Quality Factor	1	3	5	7	9	25
Mean Square Error	6.45	9.41	11.95	14.48	16.46	31.01
Visual Degradation	2	3	3	4	4	5
Using Run Length Encoding						
Compression Ratio	1:17.65	1:18.42	1:18.81	1:19.08	1:19.27	1:19.93
Using Splay Trees						
Compression Ratio	1:30.99	1:34.02	1:36.13	1:38.00	1:39.24	1:42.90

Table 4.8: Lossy compression of wrist X-ray image (1024 x 1024) using 32 x 32 matrix

4.7.6 32 x 32 quantisation matrix results for 512 x 512 images

The lossy algorithm was tested on a 512 x 512 x 8 bit wrist X-ray image to investigate the use of a 32 x 32 quantisation matrix on a smaller image. The results are shown in Table 4.9. Appendix F shows the results for the other test images.

	Pentium 90 MHz
Compression Time /s	12.73
Decompression Time /s	13.81

Quality Factor	1	3	5	7	9	25
Mean Square Error	16.14	23.94	29.8	36.48	42.8	79.63
Visual Degradation	4	5	5	5	5	5
Using Run Length Encoding						
Compression Ratio	1:15.92	1:17.01	1:17.62	1:18.05	1:18.38	1:19.37
Using Splay Trees						
Compression Ratio	1:24.26	1:27.86	1:30.44	1:32.40	1:33.99	1:39.03

Table 4.9: Lossy compression of wrist X-ray image (512 x 512) using 32 x 32 matrix

4.7.7 Comparison of results using different size quantisation matrices

The DCT may give rise to distortion in the reconstructed image, typically causing rings to be observed at sharp boundaries in the reconstructed image. These rings are attributable to Gibb's phenomenon, an effect produced by the inability of the DCT Cosine basis functions to exactly reproduce square wave functions. Also high compression rates a block pattern is produced in images. It is caused by the fact that sub-image blocks are compressed independently and so the distortion introduced is discontinuous between blocks [47].

If the quantisation of high frequencies is too severe, a visible degradation in fine detail can result. As the size of the sub-image increases the visibility of distortions such as blocking and ringing increases. As the size of the quantisation matrix, which is the size of the sub-image, increases from 8 x 8 through 16 x 16 to 32 x 32 it can be seen that blocking becomes more noticeable as the sub-image size increases. Shown in Figure 4.36 is the original Lenna image and the effect of blocking when using a 32 x 32 quantisation matrix

Original image Reconstructed image

Figure 4.36: Lenna image before and after compression

From the results it can be seen that as the sub-image increases there is an increase in compression ratio, but the visual degradation and mean square error also increase. There is also a substantial increase in processing time. It can be seen in the tables that at a low quality factor, which produces the best resolution, the run length encoding gives a slightly higher compression ratio. This is because the correlation between coefficients is not as high as in the case when a high quality factor is used. The Huffman coding using splay trees algorithm does perform as well as run length encoding for the low quality factor because it is an adaptive algorithm, as described in Section 4.4.3, so that initially there is an overhead which may decrease the compression ratio, but as the correlation in the sub-image increases the compression ratio increases. Also as the size of the sub-image increases so too does the correlation, and therefore the Huffman coding using splay trees algorithm delivers the best compression ratio.

In some cases the reconstructed image was seen to have a sharper appearance. This is the result of ringing increasing the local contrast of edges thus giving the image a sharper appearance In some cases the reconstructed image was also observed to contain more detail than the original. If the image contains large areas of near constant tone, as in some of the X-ray images, then a slight blocking effect introduces noise which could be interpreted as detail. Shown in Figure 4.37 is the

compression/decompression rate for a quality factor of 25, for the test images where

$$\text{Compression / Decompression Rate} = \frac{\text{Compression Ratio}}{\text{Compression Time} + \text{Decompression Time}}$$

The trade-off between the compression time and decompression time and the compression ratio is crucial, since the image is compressed at the server, transmitted and then decompressed at the client, so that the method which produces the highest compression/decompression rate is most desirable.

As can be seen in Figure 4.37, for quality factor 25, the DCT method using an 8 x 8 sub-image and Huffman coding using splay trees gives the highest compression/decompression rate, while the method using 32 x 32 sub-image with run length encoding performs the worst. Also shown in Figure 4.38 is the mean square error for the X-ray images. The DCT method using the 8 x 8 sub-image also has the lowest mean square error in these cases.

As can be seen in Figure 4.39, for quality factor 1, the method using a 32 x 32 sub-image with Huffman coding using splay trees performs the best. This is because there is a higher correlation in each sub-image compared to using a quality factor of 25. Again the mean square error is shown in Figure 4.40 for the X-ray images. The DCT method using a 32 x 32 sub-image also has the highest mean square error which means it would look visually degraded compared to using the other methods. The compression/decompression rate for the 512 x 512 images is comparable to that for the 1024 x 1024 images.

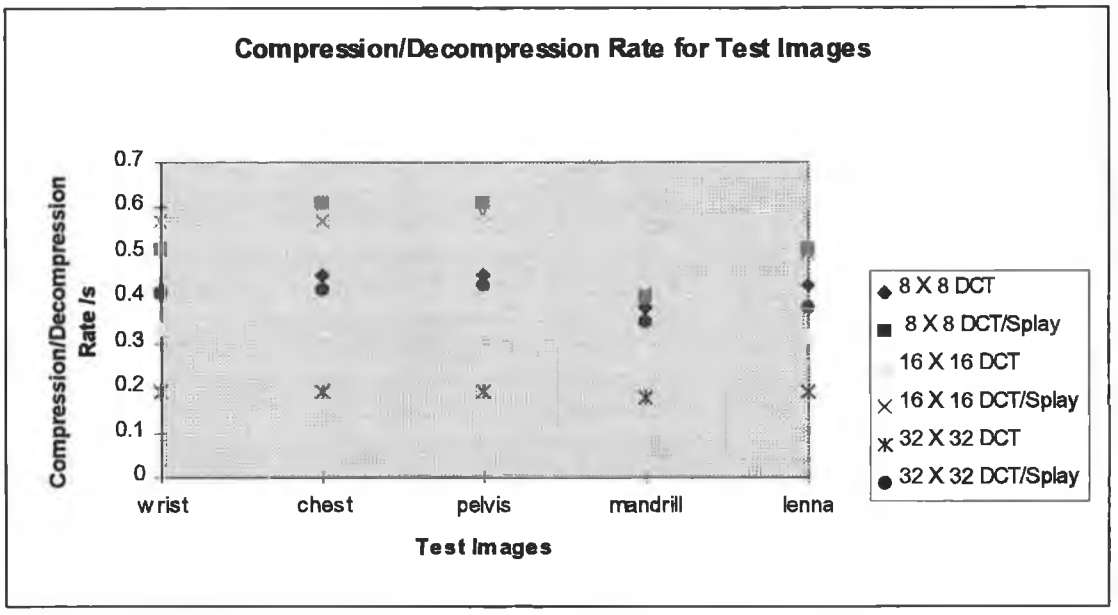


Figure 4.37: Decompression rate for 1024 x 1024 images at a quality factor of 25

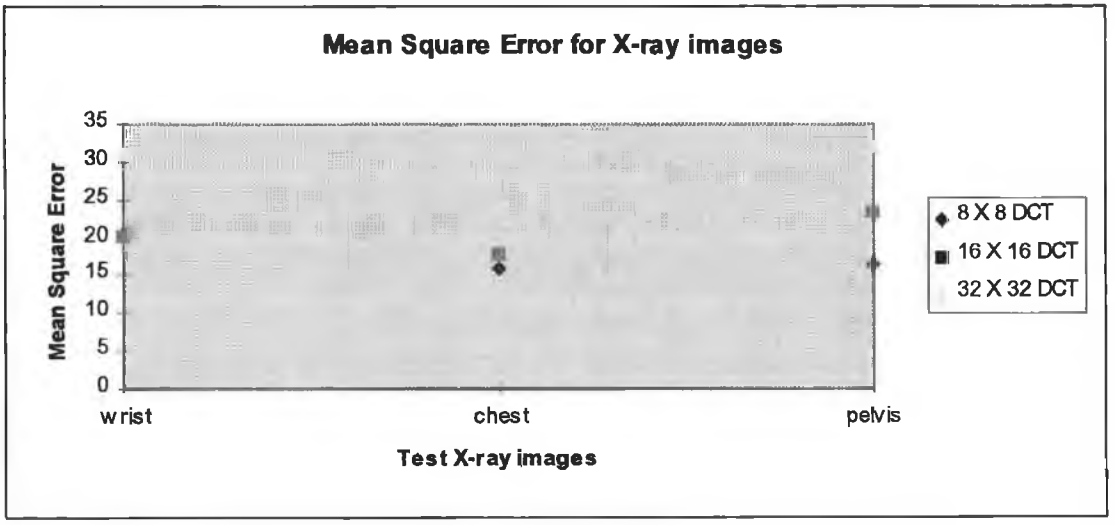


Figure 4.38: Mean square error of 1024 x 1024 X-rays at a quality factor of 25

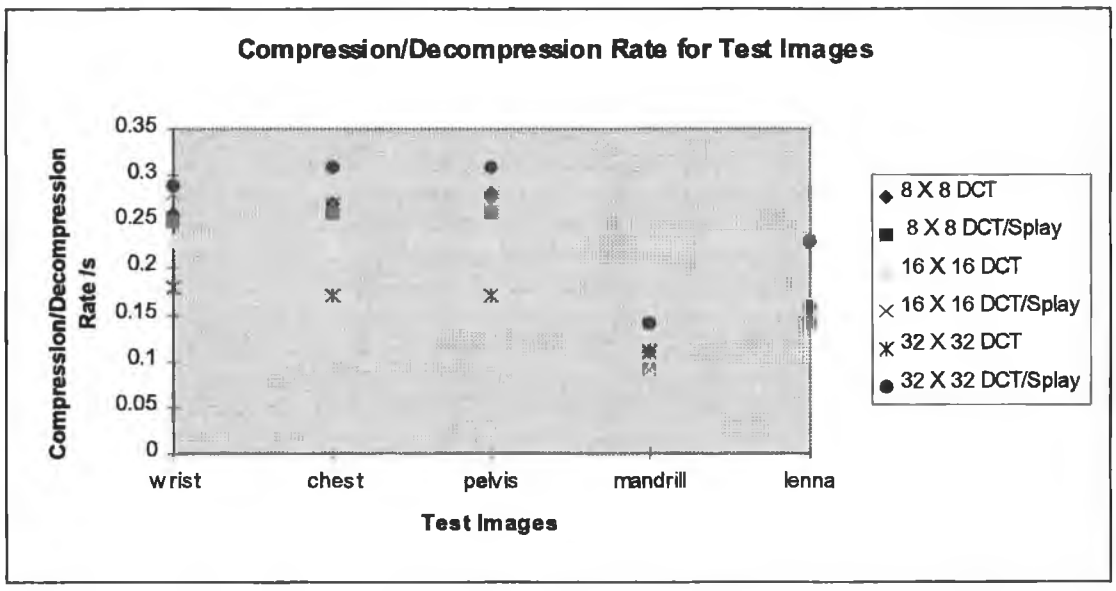


Figure 4.39: Decompression rate for 1024 x 1024 images at a quality factor of 1

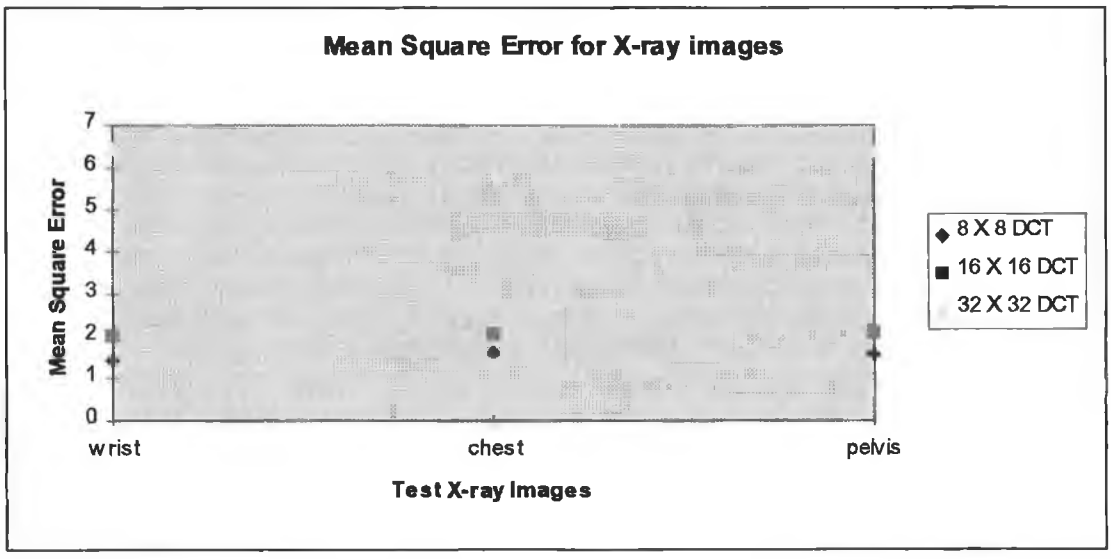


Figure 4.40: Mean square error for 1024 x 1024 X-rays at a quality factor of 1

Before choosing a method that is to be used in the archiving and remote diagnostics software application, transmission times of compressed images using the different methods must be considered and not just the compression/decompression rate.

4.8 Internet software application

An application was developed in Visual Basic to connect to a remote system and view an image from that remote system. Figure 4.41 shows the functions available for the client application. The "zoom", "increase resolution", and "restore image" are only available when "view remote image" has been selected and the images has been transmitted and received. Shown in Figure 4.42 is the flowchart for the software applications. The socket (remote connection) can be closed at any stage after the connection has been established. Appendix G provides screenshots on how to use the application.

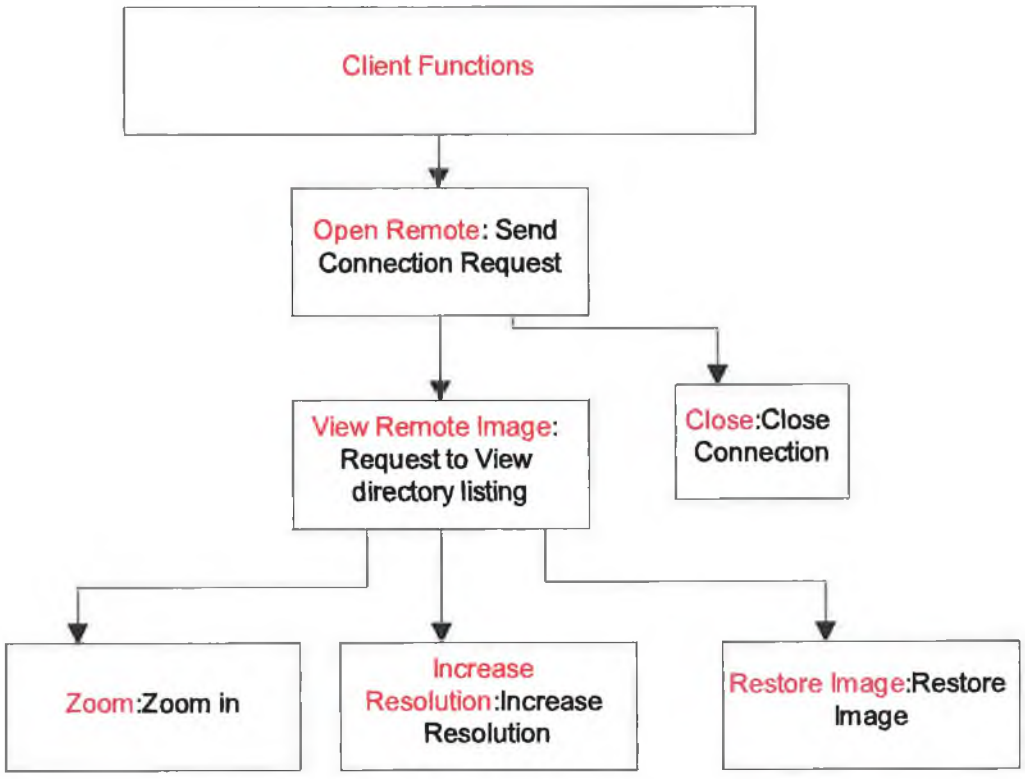


Figure 4.41: Client software application functions

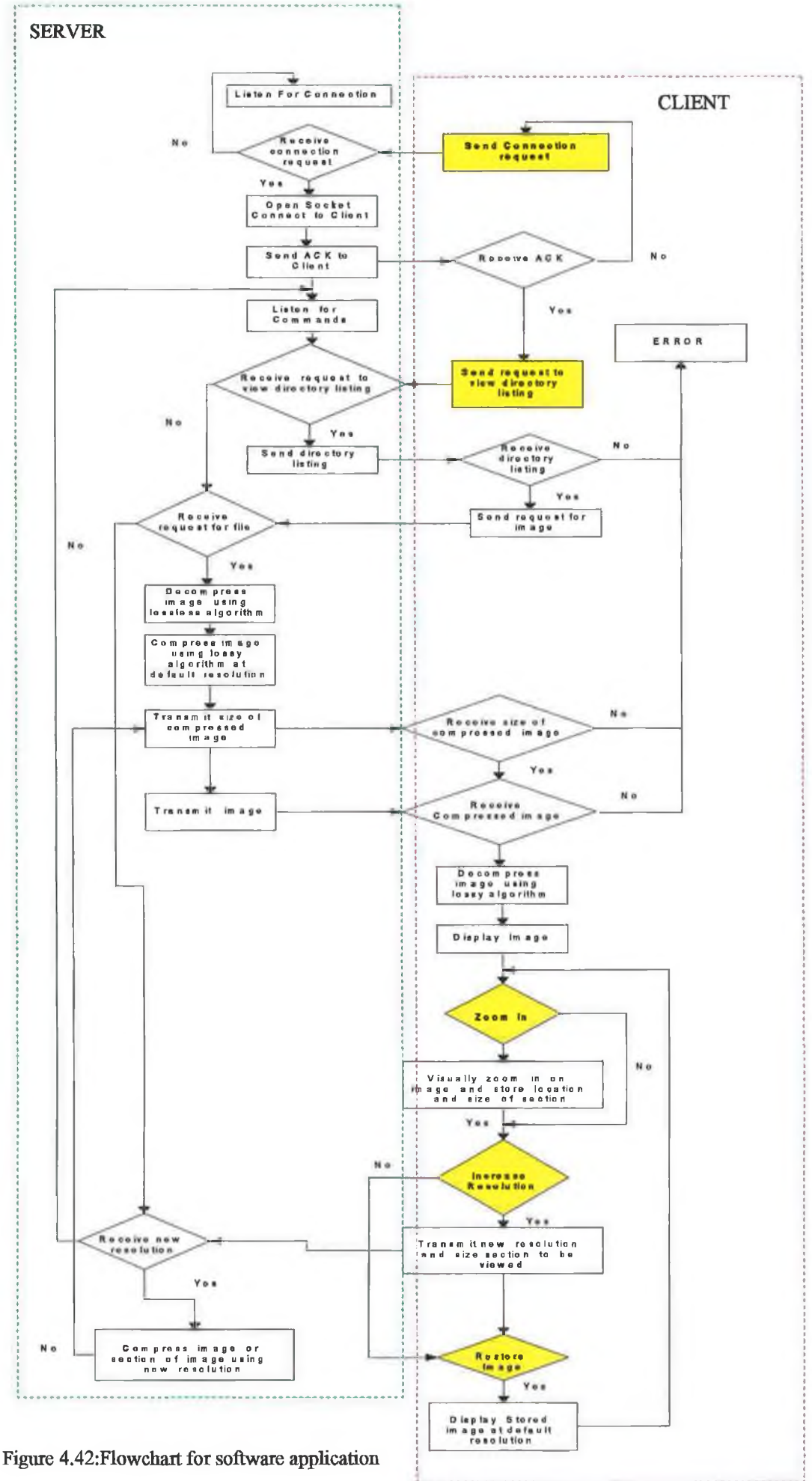


Figure 4.42:Flowchart for software application

In Figure 4.42 as a function is called in the client application, a request is made to the server application. For example, to view an image a request is made by the client application to the server application to view a directory listing. The server will then send the directory listing to the client application and an image can be chosen from this listing and sent from the server application to the client application.

The opening, closing, reading and writing of the TCP/IP sockets were again developed as DLLs and called in Visual Basic. All code in the use of the TCP/IP sockets were developed in the Application Layer of the TCP/IP model. All of the compression and decompression algorithms were also developed in the Application Layer. The other layers of the TCP/IP model, as described in Section 3.3, take care of the reliability of packet delivery and the headers associated with the transmission of the packets.

4.8.1 Comparison of transmission times

Once the application was developed the different variations on the DCT compression method were used to transmit a wrist X-ray image of size 1024 x 1024 pixels, at an intensity resolution of 8 bits/pixel and the results were compared to find which method was the fastest overall to compress, transmit, and decompress. The methods investigated are shown in Table 4.10. Figure 4.43 shows the comparison of these methods.

	METHOD
1	Image Transmitted without Compression
2	8 x 8 DCT
3	8 x 8 DCT with Splay
4	16 x 16 DCT
5	16 x 16 DCT with Splay
6	32 x 32 DCT
7	32 x 32 DCT with splay

Table 4.10: Compression methods for transmission comparison

As can be seen in this figure method 3 performs the best over a LAN system. Figure 4.44 shows the estimated transmission times using a 14.4 Kb/s modem, in this system method 5 performs the best. Thus, as the transmission time increases there is

more emphasis on the compression ratio but when there is a short transmission time there is more emphasis on the client and server processing times.

Therefore using the Internet, as the traffic increases the application could determine which method to use to optimise the bandwidth and decrease the amount of time waiting to view the image. A packet would be transmitted from the server to the client once the connection had been made. Depending how long it takes the client to receive the packet will determine which method is to be used. If there is a lot of traffic on the Internet or using a 14.4 Kb/s or slower modem, the client will send a corresponding signal to the server. The signal will inform the server and client which method to use. If there is a slow medium the emphasis will be placed on compression ratio, and if there is a fast medium the emphasis will be placed on compression and decompression time.

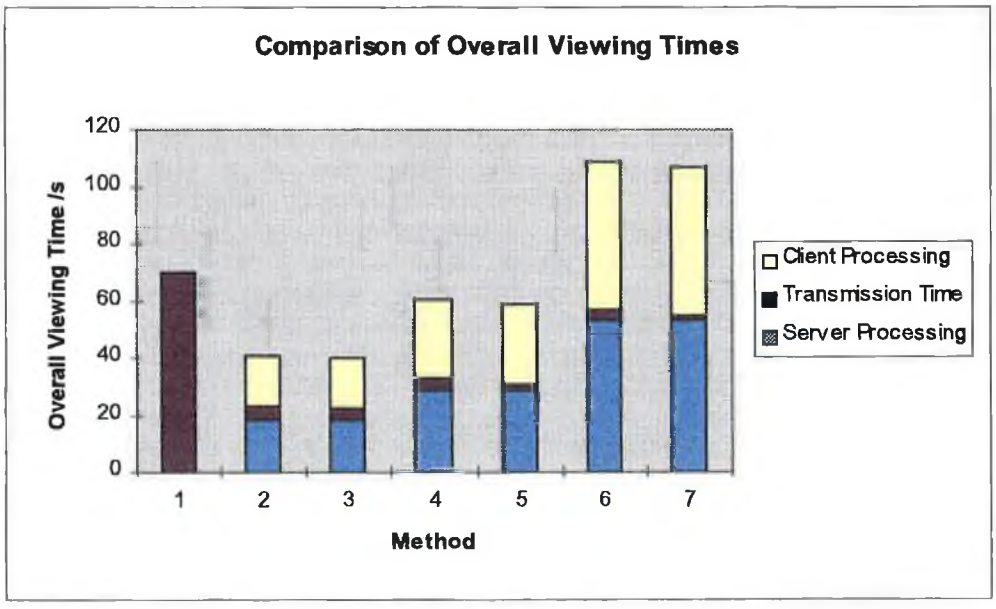


Figure 4.43: Comparison of overall viewing times over a LAN

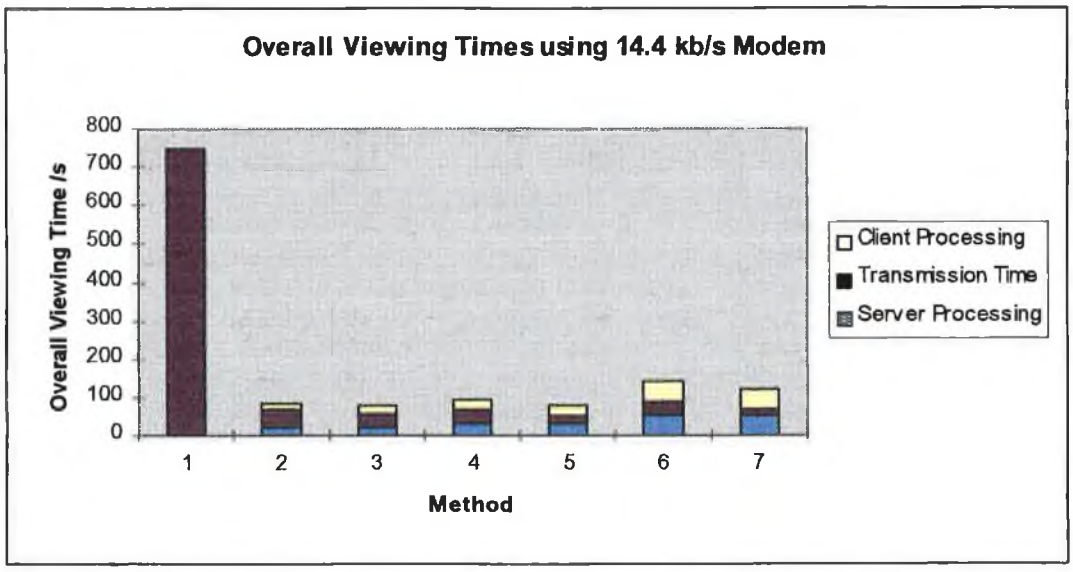


Figure 4.44: Estimated transmission times using a 14.4 kb/sec modem with different size quantisation matrices.

4.8.2 Comparison of approaches when transmitting only difference

The difference when transmitting the image means that instead of transmitting the image at a new resolution, only the difference between the image at original and new resolution is compressed and transmitted. The comparison was made of transmitting only the difference by different methods when zooming in on an image at the client application instead of transmitting the entire section. Table 4.11 lists the methods investigated. Table 4.12 shows the results when viewing the 1024 x 1024 wrist X-ray image at default resolution 25, and then requesting increased resolution of 1 for the entire image. Appendix H gives the corresponding results for the other test images. All times in the tables are measured in seconds.

	METHOD
1	8 x 8 DCT
2	8 x 8 DCT of difference
3	8 x 8 DCT with splay
4	8 x 8 DCT with splay of difference
5	16 x 16 DCT
6	16 x 16 DCT of difference
7	16 x 16 DCT with Splay
8	16 x 16 DCT with splay of difference

Table 4.11: Methods to compare transmission of difference

	1	2	3	4	5	6	7	8
Compression Ratio	1:9.37	1:10.56	1:8.8	1:11.02	1:13.09	1:13.87	1:15.75	1:17.44
Compression Time /s	18.52	18.52	19.68	18.68	28.94	28.94	30.09	30.09
Decompression Time /s	17.36	17.36	17.36	17.36	27.78	27.78	27.778	26.62
Time For Difference /s	----------	8.10	-------	6.94	--------	8.10	--------	6.94
Time for Addition /s	----------	6.94	-------	6.94	--------	6.94	--------	8.10
Transmission Time /s	7.51	6.66	7.92	6.37	5.37	5.06	4.46	4.03
Overall Time /s	43.39	57.58	44.96	57.29	62.09	76.82	62.33	75.78

Table 4.12 : Comparison of difference methods for wrist X-ray image

Figure 4.45 gives the comparison of the results for the eight methods investigated over a LAN. As can be seen when using a LAN method 1 performs the best but when using a 14.4 Kb/sec modem (shown in Figure 4.46) method 7 performs the best. Transmitting the difference between the image at default resolution and the new resolution does not necessarily decrease the time waiting to view the image. This is because even though the compression ratio increases it does not increase enough, because there is additional processing at the client and the server in determining the difference and then adding the transmitted difference to the image at default resolution at the client. As can be seen in the tables the transmission time does decrease but the processing time at the client and server increases also, thus decreasing the overall effectiveness.

When transmission time is short the processing times are crucial and when there is a long transmission time the compression ratio of the compressed image is crucial. Thus the software application could determine the method to use to optimise the bandwidth used depending on the medium used, modem, LAN, WAN.

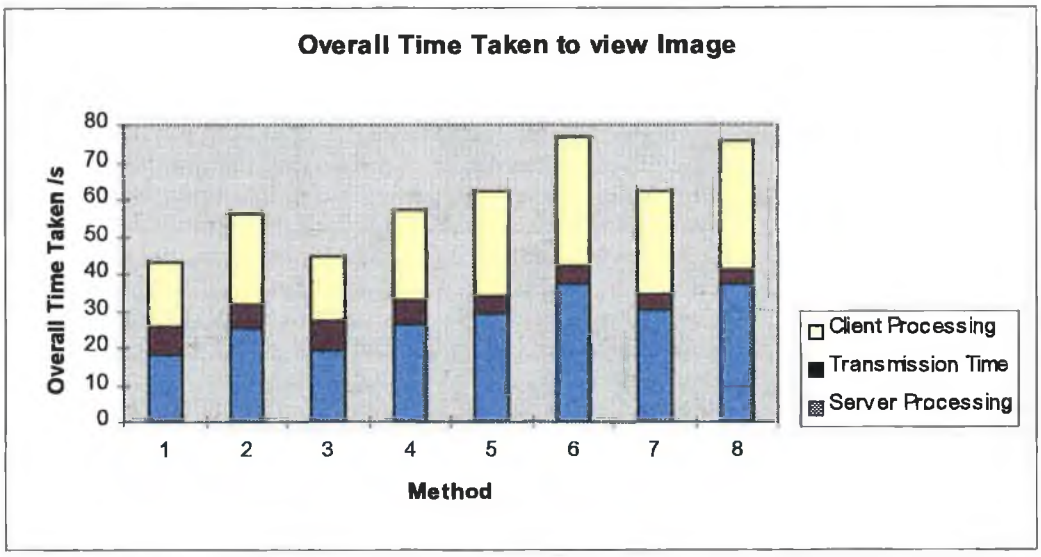


Figure 4.45: Comparison of transmitting images using difference methods over a LAN

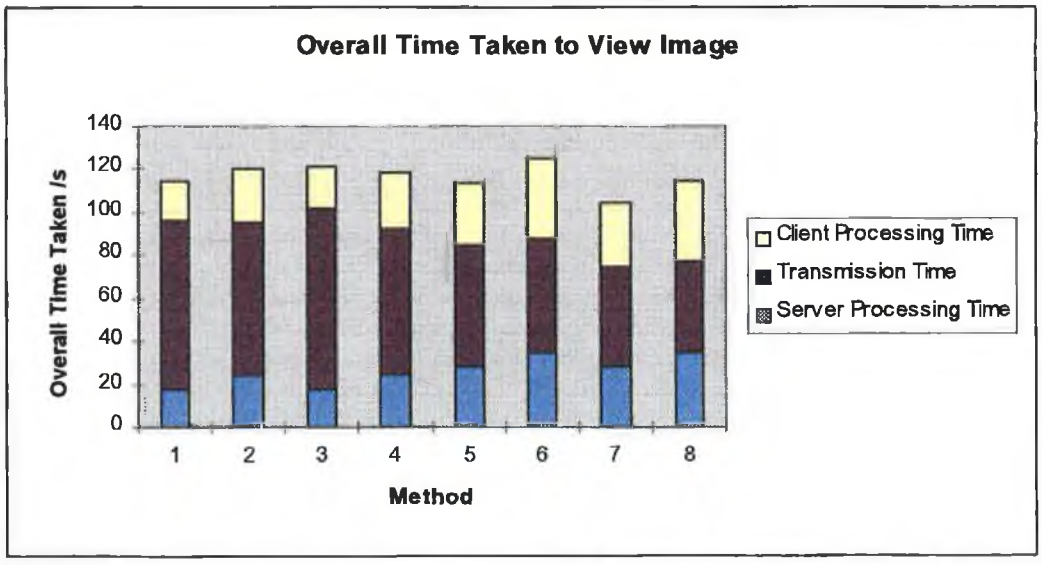


Figure 4.46: Estimated times for transmitting images using difference methods over a 14.4 Kb/s modem

Chapter 5 discusses the software application and the implementation of the algorithms. Future developments are also discussed.

CHAPTER 5
FUTURE DEVELOPMENT AND DISCUSSION OF THE SOFTWARE APPLICATION

5.1 Extension to Colour Images

Thus for all the compression algorithms were tested on grey scale images. The lossless and lossy algorithms developed for grey scale images can be however extended to colour images. The lossless algorithm, Huffman coding using splay trees, can easily be extended since the algorithm works on an input stream and does not care what type of file it is whereas the lossy algorithm, Transform image coding using the DCT, operates on a matrix.

The DCT algorithm can be extended by transforming the RGB value to another set of tristimulus values. RGB is transformed to the luminance-chrominance YIQ set, as described in appendix G. The Y component is the luminance component and is responsible for the perception of brightness of a colour image. Most high frequency components of an image are primarily in the Y component. The I and Q components are the chrominance components and are responsible for the perception of hue and saturation of a colour image. Once the RGB has been transformed to YIQ each matrix, Y, I and Q, can be compressed individually using the DCT algorithm.

The lossless and lossy compression algorithms therefore can be applied to colour images. Figure 5.1 displays the 24 bit standard Lenna image and in Figures 5.2 and 5.3 are two 24 bit images of blood samples obtained from "The Bristol Biomedical Image Archive" from the University of Bristol [48].

Figure 5.1: Standard 24 bit Lenna image (1024 X 1024)

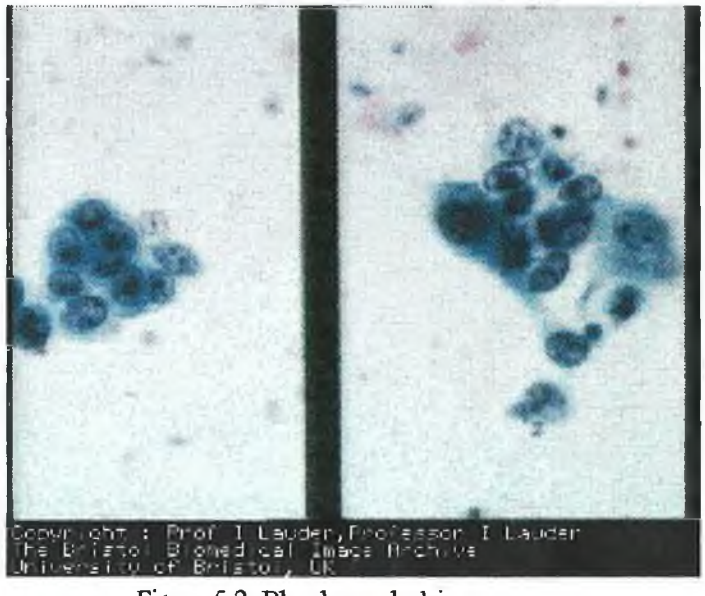


Figure 5.2: Blood sample 1 image

Figure 5.3: Blood sample 2 image

Shown in Table 5.1 are the results for compressing and decompressing the colour 24 bit Lenna image. The compression and decompression time are displayed. So also is the compression ratio and the visual degradation. The levels of degradation are as shown in Table 4.3. The results for the blood samples 1 and 2 images are shown in Tables 5.2 and 5.3 respectively.

	Pentium 90 MHz
Compression Time /s	55.56
Decompression Time /s	38.19

Quality factor	1	3	5	7	9	11	25
Visual Degradation	1	1	1	1	1	2	5
Compression Ratio	1:12.39	1:13.0	1:13.98	1:14.88	1:16.34	1:17.88	1:73.35

Table 5.1: Results for colour 24 bit Lenna image

	Pentium 90 MHz
Compression Time /s	11.57
Decompression Time /s	6.94

Quality factor	1	3	5	7	9	11	25
Visual Degradation	1	1	1	2	2	2	5
Compression Ratio	1:9.32	1:9.85	1:10.56	1:11.28	1:12.38	1:13.59	1:47.48

Table 5.2: Results for colour 24 bit blood sample 1

	Pentium 90 MHz
Compression Time /s	11.57
Decompression Time /s	8.10

Quality factor	1	3	5	7	9	11	25
Visual Degradation	1	1	1	2	2	2	5
Compression Ratio	1:7.48	1:7.87	1:8.41	1:8.95	1:9.72	1:10.59	1:43.70

Table 5.3: Results for colour 24 bit blood sample 2

Figure 5.4 shows a comparison of times for transmitting the uncompressed image over the LAN and transmitting the image at different compressed quality factors. As can be seen there is a 50% saving in time waiting to view the image. If a slower medium is used, for example a 14.4 kb/sec modem, there would be an increase in the time saved.

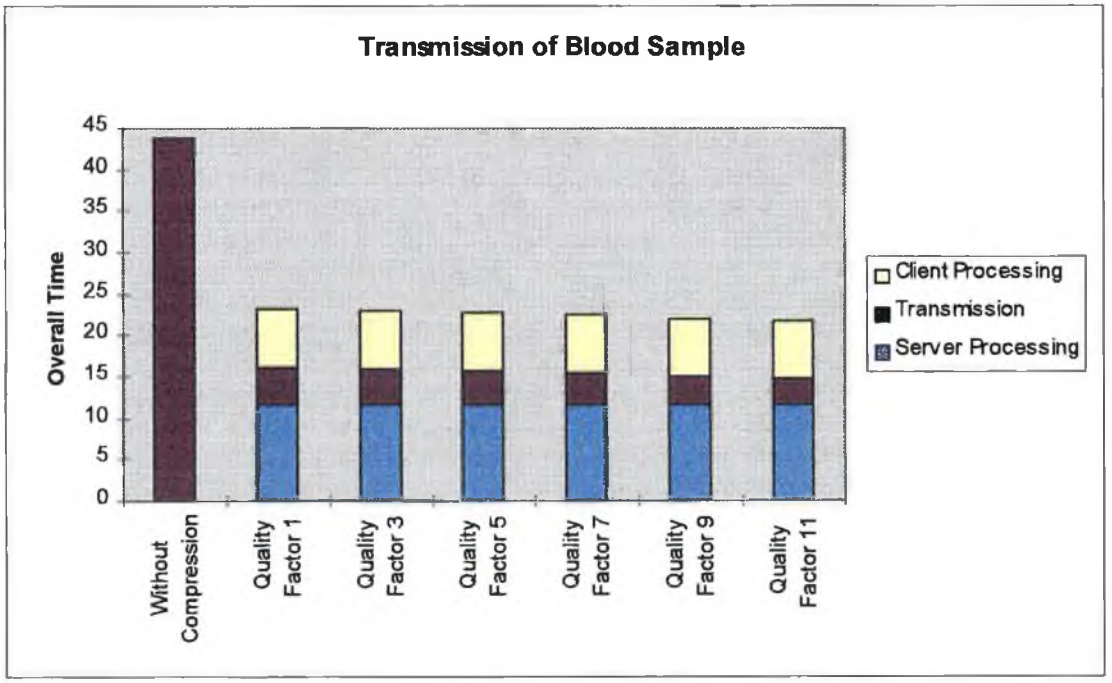


Figure 5.4: Transmission of blood sample 1 using a LAN

The algorithm for compressing colour images could be incorporated into the software application. From the header of the image the application could determine whether to use the grey scale or colour image compression algorithm. The colour algorithm could be developed as a DLL and used in the Visual Basic software application.

5.2 Lossy Algorithms

Other lossy algorithms can be investigated to determine if they would produce a high compression ratio with a short compression and decompression time and also with a low level of visual degradation. These algorithms could be developed into DLLs and inserted into the software application instead of the Transform Image coding using the DCT. The present application has been developed so that the compression algorithms can be exchanged very easily using other DLLs. One other lossy algorithm is Wavelet Transform compression. This algorithm could be investigated as part of future research. Shown in the next section is an introduction to Wavelet transforms.

5.2.1 Wavelet transform compression

Compression is accomplished in this case by applying a wavelet transform [49] to decorrelate the image data, quantising the resulting transform coefficients and coding the quantised values as shown in Figure 5.5 [50]. Image reconstruction is accomplished by inverting the compression operation as shown in Figure 5.6.

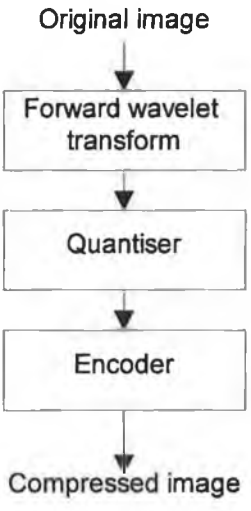


Figure 5.5: Forward wavelet compression

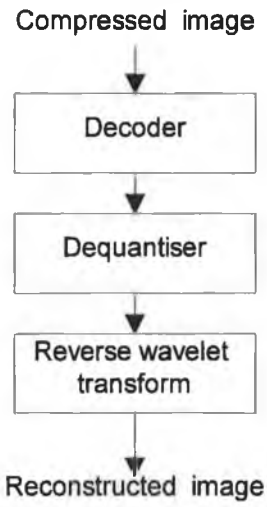


Figure 5.6: Reverse wavelet compression

The forward and inverse wavelet transforms can be implemented by a pair of appropriately designed quadrature mirror filters [51]. Wavelet based image compression can be viewed as a form of sub-band coding. Each quadrature mirror filter pair consists of a lowpass filter and a highpass filter which split an image's bandwidth in half.

Figure 5.7 shows the flowchart for a single 2-D forward wavelet transform of an image. This is accomplished by two separate 1-D transforms. Since the bandwidth of the lowpass and highpass images are half that of the original image it is possible to downsample by a factor of two without any loss of information. The image has been decomposed into an average image and three detail images. Sub-image 1 is the average image and sub-image 2 emphasises the horizontal image features, sub-image 3 emphasises the vertical image features and sub-image 4 the diagonal features.

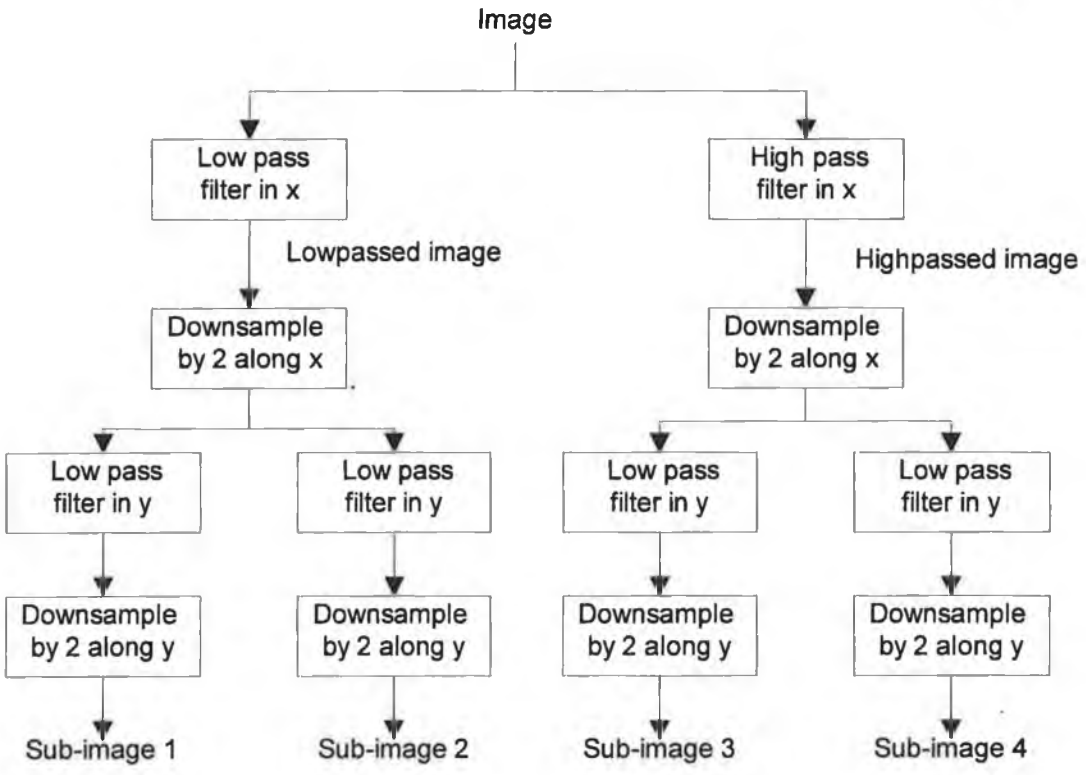


Figure 5.7: 2-D Forward wavelet transform

For compression the image is recursively transformed. For a high compression ratio the transform is performed several times. After the forward wavelet transform has been performed, a matrix of coefficients is left that contains the average image and the detail images at each scale. Compression is achieved by quantising and encoding coefficients. To achieve high compression ratios a separate quantiser should be designed for each scale. Quantisation and encoding have already been discussed in section 2.3.5.

Shown in Figure 5.8 is the flowchart of the 2-D inverse wavelet transform. However sub-image 1 is the average image and sub-images 2 to 4 are the detail images. Upsampling is accomplished by inserting a zero between each pair of values in the y or x dimension. Upsampling is necessary to recover the proper bandwidth required to add the sub-images back together.

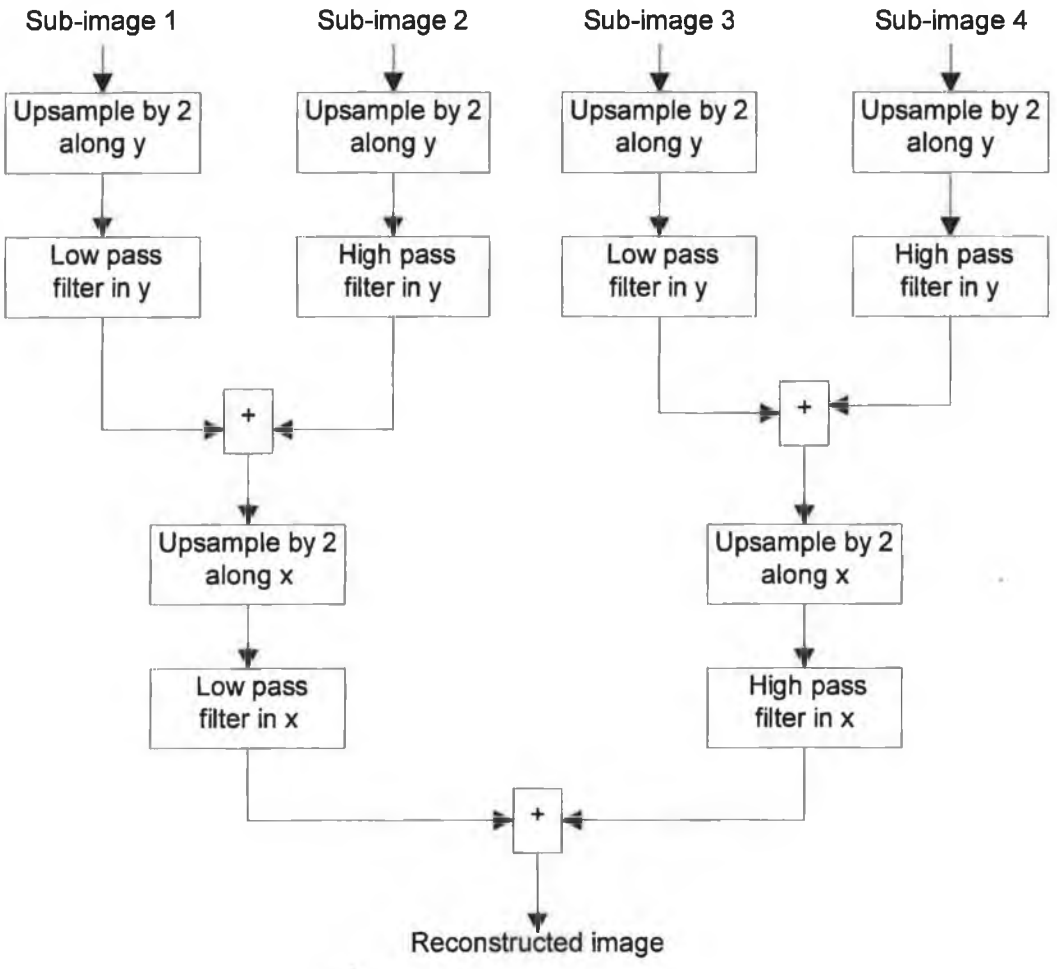


Figure 5.8: Reverse wavelet transform

There are differences between using Fourier analysis (DCT) and wavelets. Fourier basis functions are localised in frequency but not in time. Small frequency changes in the Fourier transform produce changes everywhere in the time domain. Wavelets are local both in frequency/scale (via dilation) and in time (via translations). In wavelet transforms there is a tradeoff between frequency and time.

Many images can be represented by wavelets in a more compact way. For example, images with discontinuities and images with sharp spikes usually take substantially fewer wavelet basis functions to achieve the same precision. Thus the wavelet based method has the potential to achieve a higher image compression ratio while decreasing the visual degradation at higher compression ratios [52].

5.3 Encryption

Due to the nature of medical images security is very important. Security can be implemented by introducing password protection of different levels into the software application. The two levels would include an administration level where the database could be modified and the user level where the information could be viewed only. Another form of security is the encryption of the compressed images as they are transmitted over a network such as the Internet. Encryption could be inserted into the Software Application as part of future research. An introduction to encryption follows.

Encryption [53] is the transformation of data into a form unreadable by anyone without a secret decryption key. Its purpose is to ensure privacy by keeping the information hidden from anyone for whom it is not intended, even those who can see the encrypted data. Encryption allows secure communication over an insecure channel.

Encryption works as follows. Bob wishes to view an image from the server. The server encrypts the image, so that no one can view the image except Bob, with an encryption key. The image is called plaintext and the encrypted image is called ciphertext. The encrypted image is now sent to Bob. Bob decrypts the ciphertext with the decryption key and views the image. In a secure cryptosystem, the plaintext cannot be recovered from the ciphertext without using the decryption key. In a symmetric cryptosystem a single key serves as both the encryption and decryption keys. Figure 5.9 shows the addition of encryption to the software application. The encryption and decryption algorithms can be developed as DLLs and are called in the software application to encrypt and decrypt the compressed images respectively.

An example of an encryption algorithm is PGP (Pretty Good Privacy) [54]. PGP is a high security cryptographic software application that allows people to exchange messages with both privacy and authentication. It provides privacy in that only those intended to receive a message can read it. By providing the ability to encrypt messages, PGP provides protection against anyone eavesdropping on the Internet. Even if the packet is intercepted it will be unreadable by the eavesdropper. It provides authentication because PGP ensures that a message appearing to be from a

particular location can have originated from that location only and that the message has not been altered.

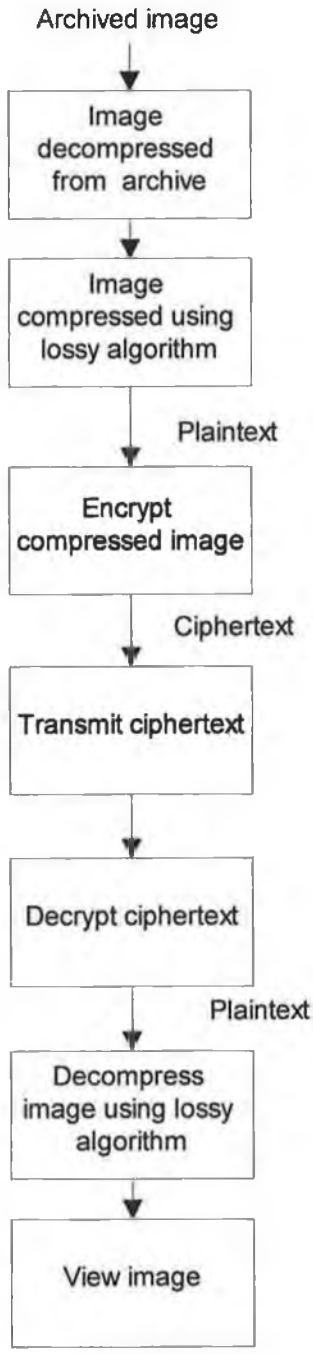


Figure 5.9: Addition of encryption to software application

5.4 Discussion of Software Application

The purpose of this project was to develop a software application using an appropriate lossless algorithm and lossy algorithm for archiving and transmission over the Internet respectively. The objective was to choose a fast algorithm which would optimise the use of bandwidth over the Internet and reduce the amount of time waiting for the image to be transmitted. The application was developed with a user friendly Graphic User Interface (GUI) front-end in the Windows environment using Visual Basic.

The application allowed the user to connect from a remote PC to the server using the Internet. They could then choose an image to be transmitted from the server database. The user could then zoom in on a section of the image at a new resolution, download the entire image at a new resolution or choose another image. The application was designed so that this procedure could be accomplished quite easily. The user does not have to have any knowledge of the algorithms for compression or transmission of the images.

Several languages were investigated in which to develop the application, MATLAB®, C, Visual C++ and Visual Basic. A decision was made that the compression algorithms would be developed in C due to its speed and versatility and then coded as DLLs in Visual C++ so that they could be used in the Windows environment. Visual Basic was used to develop the Windows front end since it is quite simple to develop a GUI in this environment. The compression DLLs were declared and called in the Visual Basic application. The opening and closing of sockets and also the reading and writing of data using TCP/IP protocols over the Internet were also developed as a DLL and called in the Visual Basic application.

The lossless and lossy algorithms were tested on five images, three X-ray images obtained from St. James' Hospital and two standard test images. The lossless algorithms investigated were static Huffman coding, adaptive Huffman coding, Huffman coding using splay trees and arithmetic coding. The algorithms were applied to each test image and in each case compression ratio, compression time and decompression time were recorded. The mean square error was also recorded for each algorithm in each case and was seen to be zero. This was done to verify that the algorithms were lossless. From these results the decompression rate was calculated,

where Decompression Rate $= \frac{\text{Compression Ratio}}{\text{Decompression Time}}$. This parameter is important since the image must be decompressed from the archive before it is transmitted and the trade-off between the decompression time and the compression ratio is crucial.

The decompression rate for Huffman coding using splay trees was seen to be high compared to that of the other algorithms tested. This algorithm has a high compression ratio with short decompression and compression time. It was demonstrated that Huffman coding using splay trees performed the best for each test image. The arithmetic coding algorithm produced the highest compression ratio but had long compression and decompression times. The Huffman coding using splay trees algorithm performed better than the static Huffman coding algorithm since it is adaptive and does not have to scan through the data twice to build the Huffman code tree and then encode the leaves. Splaying reduces the distance from the root to the leaves of the tree and so this increases the compression ratio and decreases the compression time compared to the adaptive Huffman coding algorithm. Since the Huffman coding using splay trees algorithm does not involve any floating point calculations it is faster than the arithmetic coding algorithm. Therefore the Huffman coding using splay trees was selected as the preferred lossless algorithm for use in the Visual Basic application.

The lossy algorithm chosen was transform image coding using the Discrete Cosine Transform. Compression ratio, mean square error, compression time and decompression time were recorded for each test image using different quality factors. The quality factor distinguishes the different resolutions available. As the quality factor increases the resolution decreases. The DCT algorithm was tested using different encoding algorithms, that is, run length encoding and Huffman coding using splay trees and also using different size quantisation matrices. Sub-image sizes investigated were 8 x 8, 16 x 16 and 32 x 32.

The use of the DCT transform can introduce a number of distortions into the reconstructed image. High compression ratios can cause a block pattern to appear in the images and also a visible degradation in fine detail if the high frequencies are quantised too severely. As the size of the sub-image increases the

blocking becomes more noticeable. There is also a substantial increase in processing time.

At the lowest quality factor (highest resolution) run length encoding produces a higher compression ratio than does Huffman coding using splay trees. This is because the correlation between coefficients is not as high as in the case when a high quality factor is used. The Huffman coding using splay trees algorithm does not perform as well as run length encoding for the low quality factors because it is an adaptive algorithm and initially there is an overhead which may decrease the compression ratio but as the correlation in the sub-image increases, either by increasing the quality factor or the size of the sub-image, the compression ratio increases.

From the results recorded the compression/decompression rate was calculated where

$$\text{Compression / Decompression Rate} = \frac{\text{Compression Ratio}}{\text{Compression Time} + \text{Decompression Time}}$$

The trade-off between the compression time and decompression time and the compression ratio is crucial since the image will be compressed at the server, transmitted and then decompressed at the client. The lossy method which produces the highest compression/decompression rate is optimal. For a quality factor of 25, 8 x 8 DCT using Huffman coding with splay trees produces the highest compression/decompression rate and the lowest mean square error. For a quality factor of the method using 32 x 32 sub-image with Huffman coding using splay trees performs the best but has the highest mean square error. This is because there is a higher correlation in each sub-image compared that when using a quality factor of 25 but the image will look blocky due to the size of the sub-image used.

The transmission times of compressed test images using the different methods were considered. The times considered were the server processing time, transmission time and the client processing time. As the size of the sub-image increases the compression ratio decreases which decreases the transmission time but increases the compression and decompression time.

Depending on which medium is used, for example LAN or 14.4 kb/sec modem, different methods of compression produce optimum results. So as the

transmission time increases more emphasis is placed on the compression ratio and the method using 16 x 16 DCT with Huffman coding using splay trees would be used, but when there is a short transmission time there is more emphasis on the client and server processing time the 8 x 8 DCT with Huffman coding using splay trees would be used.

Using the Internet, as the traffic increases the application could determine which method to use to optimise the bandwidth and decrease the amount of time waiting to view the image. This would be accomplished by transmitting a packet from the server to the client once the remote connection had been made. The transmission time of the packet would determine which packet to use depending on the speed of the transmission medium.

The advantages of transmitting only the difference when zooming in on an image at the client application instead of transmitting the entire section was investigated. It was seen that transmitting the difference didn't necessarily decrease the time waiting to view the image. This is because even though the compression ratio increases, thereby decreasing the transmission time there is also additional processing at the server and client applications. Again depending on the speed of the medium the appropriate method is selected to optimise the bandwidth.

The Visual Basic application could also be developed to archive and transmit colour images over the Internet. Another lossy algorithms which could be investigated to compare to the DCT is wavelet compression. Again the trade-off between compression and decompression time and the compression ratio is crucial. The inclusion of security into the application could be achieved by using password protection and encryption of the compressed images before they are transmitted.

REFERENCES

[1] Gonalez R.C, *Digital Image Processing: Past, Present and Future*, International Electronic Imaging Expo. and Conference, M.A. USA, March 1988, Vol. 1, pp. 30-34

[2] Jähne, Bernd, *Digital Image Processing: Concepts, Algorithms and Scientific Applications*, Springer, 1991, pp. 1-8

[3] Marion, André, *An Introduction to Image Processing*, Chapman and Hall 1991

[4] Myler, Harley R.,Weeks, Arthur R, *The Pocket Handbook of Image Processing Algorithms in C*, Prentice Hall, 1993, pp. 47

[5] CCITT Standards: *http://cuiwww.unige.ch/OSG/MultimrdiaInfo/mmsurvey/Standards.html*

[6] Redfern, A.J., Hines, Dr. E. L, *Medical Image Data Compression Techniques*, Third International Conference on Image Processing and its Applications, Conf. Publ. No. 307, University of Warwick, pp. 558-562

[7] Definition of Lossless and Lossy Compression: *http://www.ccnet.ab.ca/dcp/ratiocmp.html*

[8] Storer, J.A., *Data Compression*, Computer Science Press, Rockerville, MD, 1984

[9] Jain, Anil K, Farrelle, Paul M, and Algazi, V. Ralph, *Image Data Compression, Digital Image Processing Techniques*, Academic Press, San Diego, 1984, pp. 171-221

[10] Held, Gilbert, *Data and Image Compression: Tools and Techniques*, Wiley, New York, 1996, pp. 176-184

[11] Huffman, D.A., *A Method for the Construction of Minimum Redundancy Codes*, Proceedings of the IRE, Vol. 40, No. 9, September 1952, pp. 1098-1101

[12] Compression Frequently Asked Questions: *http://www.cis.ohio-state.edu/hypertext/faq/usenet/compression-faq/part2/faq-doc-1.html*

[13] Langdon, G, *An Introduction to Arithmetic Coding*, IBM J. Res. Develop, 28, March 1984

[14] Compression Frequently Asked Questions: *http://www.cis.ohio-state.edu/hypertext/faq/usenet/compression-faq/part2/faq-doc-1.html*

[15] Jones, Douglas W., *Application of Splay Trees to Data Compression*, Commun. ACM, August 1988, Vol. 31, No. 8, pp. 996-1007

[16] Jayant, N. S., Noll, Peter, *Digital Coding of Waveforms*, Prentice Hall, Englewood Cliffs NJ, pp. 465-479

[17] Myler, Harley R., Weeks, Arthur R., *The Pocket Handbook of Image Processing Algorithms in C*, Prentice Hall, 1993, pp. 200-201

[18] Jayant, N. S., Noll, Peter, *Digital Coding of Waveforms*, Prentice Hall, Englewood Cliffs NJ, pp. 510-563

[19] Rao, K.R., Yip, J., *Discrete Cosine Transform: Algorithms, Advantages and Applications*, Academic Press Inc., San Diego, 1990, pp. 168

[20] Kagina, K., Overview of JPEG: Is it acceptable in Medical Fields?, Medinfo 92, Elsevir Science Publishers B.V., North Holland, pp. 762-768

[21] CCITT JPEG Standard: http://cuiwww.unige.ch/OSG/MultimediaInfo/mmsurvey/standards.html#JPEG

[22] JPEG Frequently Asked Questions: http:ww.cis.ohio-state.edu/hypertext/faq/usenet/jpeg-faq/faq.html

[23] Jähne, Bernd, *Digital Image Processing: Concepts, Algorithms and Scientific Applications*, Springer, 1991, pp. 346-348

[24] Karl, John H., *An Introduction to Digital Signal Processing*, Academic Press, San Diego, 1989

[25] Narasimha, M. J., Peterson, A. M., *On the Computation of the Discrete Cosine Transform*, IEEE Trans. Comm. Vol. 26, pp. 934-936

[26] Rao, K.R., Yip, J., *Discrete Cosine Transform: Algorithms, Advantages and Applications*, Academic Press Inc., San Diego, 1990

[27] TCP/IP: *gopher://gopher-chem.ucdavis.edu/11/index/Internet_aw/intro_the_Internet/intro.to.ip*

[28] Hakall, Fred, *Data Communications, Computer Networks and Open Systems*, Addison-Wesley, 1996, Reading MA

[29] Jamsa, Kris, Cope, Ken, *Internet Programming*, Jamsa, 1996

[30] Tanenbaum, A.S., *Modern Operating Systems*, Prentice Hall International, Englewood Cliffs, N.J. 1992, pp. 396-456

[31] Day, J.D., The OSI Reference Model, Proc. of the IEEE, Vol. 71, Dec. 1983, pp. 1334-1340

[31] Tanenbaum, A.S, *Computer Networks*, 3rd. Ed., Englewood Cliffs, N.J., Prentice Hall, 1988

[33] Internet Addressing: *gopher://gopher-chem.ucdavis.edu:70/00/Index/Internet_aw/intro_the_Internet/intro.to.ip/07_Internet_addresses*

[34] Coularis, G., Dollimore, J., Kindberg, T., *Distributed Systems, Concepts and Design*, Addison-Wesley, Reading MA, 1994

[35] MATLAB Reference Guide, The Math Works Inc.

[36] Charlap, D., *The BMP File Format Part 2*, Dr. Dobbs Journal, April 1995

[37] Myler, Harley R., Weeks, Arthur R., *The Pocket Handbook of Image Processing Algorithms in C*, Prentice Hall, 1993, pp. 125-127

[38] W. Press et al., *Numerical Recipes in C*, Cambridge University Press, New York, 1992, pp. 896-902

[39] Held, Gilbert, *Data and Image Compression: Tools and Techniques*, Wiley, New York, 1996

[40] Jones, Douglas W., *Application of Splay Trees to Data Compression*, Commun. ACM, August 1988, Vol. 31, No. 8, pp. 996-1007

[41] Witten, Ian H., Neal, Radford M., Clery, John G., *Arithmetic Coding for Data Compression*, Communications of the ACM, Vol. 30, No. 6, June 1987, pp. 520-540

[42] W. Press et al., *Numerical Recipes in C*, Cambridge University Press, New York, 1992, pp. 902-906

[43] Held, Gilbert, *Data and Image Compression: Tools and Techniques*, Wiley, New York, pp. 208-217

[44] W. Press et al., *Numerical Recipes in C*, Cambridge University Press, New York, 1992, pp. 67-69

[45] Gregory, E. J., Wallace, K., *The JPEG Still Image Compression Standard*, Communication of the ACM, 34(4), April 1991, pp. 30-44

[46] Rao, K.R., Yip, J., *Discrete Cosine Transform: Algorithms, Advantages and Applications*, Academic Press Inc., San Diego, 1990, pp. 164-166

[47] Ford, A. M., Jacobson, R. E., Attridge, G. G., Pointer, M. R., *Effects of Image Compression on Image Quality*, Royal Photographic Society Symposium on Photographic Image Quality, School of Communication, University of Westminster, 15th September 1994

[48] The Bristol Biomedical Image Archive, University of Bristol, *http://www/bris.ac.uk*

[49] Graps, Amara, *An Introduction to Wavelets*, IEEE Computational Science and Engineering, Vol. 2, No. 2, IEEE Computer Society, CA USA, Summer 1995, pp. 1-18

[50] Hilton, Michael L., Jawerth, Björn A., Sengupta, Ayan, *Compressing Still and Moving Images with Wavelets*, Multimedia Systems, Vol. 2, No. 3, April 18 1994, pp. 1-19

[51] W. Press et al., *Numerical Recipes in C*, Cambridge University Press, New York, 1992, pp. 584-600

[52] Advantages of Wavelets: *http://www.summus.com/wavelets.htm*

[53] Myer, Peter, An Introduction to the use of Encryption, *http:// www.virtualschool.edu/mon/Crypto/IntroEncryption.html*

[54] PGP: http://rschp2.anu.edu.au:8080/crypt.html#Pgp

[55] Charlap, D., *The BMP File Format Part 1*, Dr. Dobbs Journal, March1995

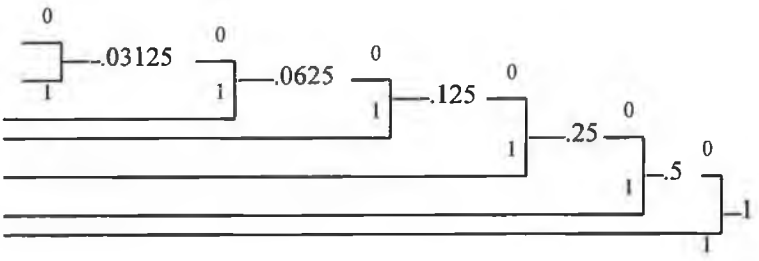
Appendix A
Example of Splay Trees

Seven intensities and their probabilities are shown below.

Intensity	Probability
50	0.015625
75	0.015625
100	0.03125
125	0.0625
140	0.125
197	0.25
250	0.5

A Huffman tree can be developed from these probabilities.

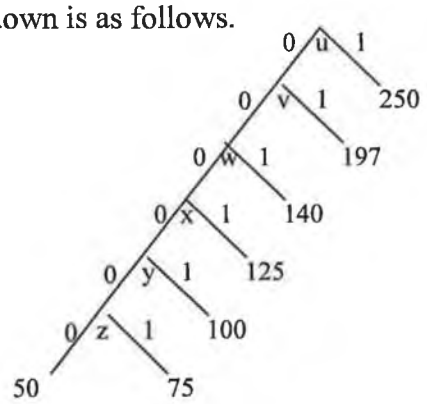
Intensity	Probability
50	0.015625
75	0.015625
100	0.03125
125	0.0625
140	0.125
197	0.25
250	0.5

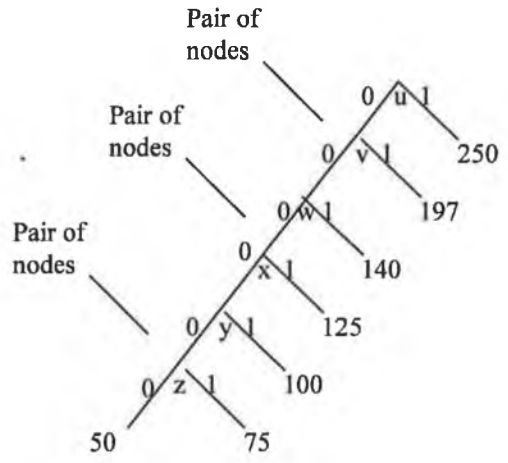


The intensities and their corresponding Huffman codes are shown.

Intensity	Huffman Code
50	000000
75	000001
100	00001
125	0001
140	001
197	01
250	1

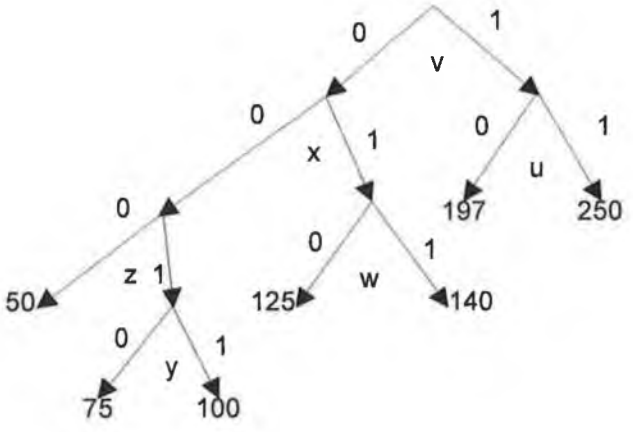
The Huffman tree from the top down is as follows.





Moving along the path from the root, node u, to leaf node 50. Each pair of successive internal nodes (node v to z) are rotated so that path length from root to the leaf node is halved. The nodes in each pair that are farthest from the root stay on the new path (nodes v, x, z) while those that are closest move off the path (nodes u, w, y).

So after splaying, the tree is as follows.



Intensity	Splay Code
50	000
75	0010
100	0011
125	010
140	011
197	10
250	11

Thus instead of coding these intensities using Huffman coding which requires twenty seven bits these intensities can be coded using splaying which requires twenty one bits.

Appendix B
Derivation of Discrete Cosine Transform

To derive one dimensional (1-D) DCT

Let x(n) denote an N point sequence that is zero outside $0 \le n \le N-1$.

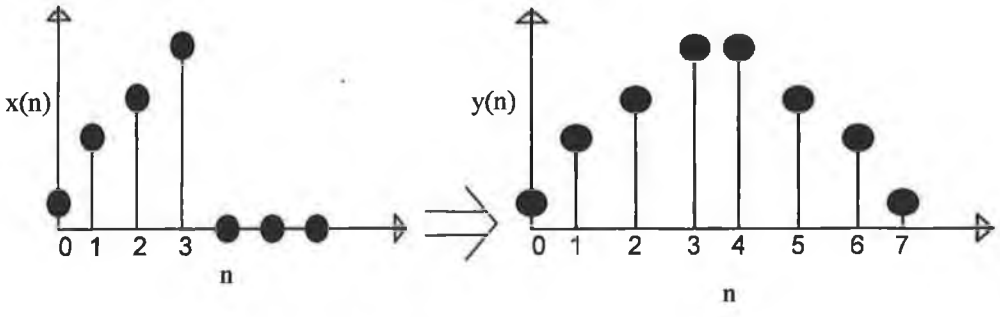
Consider one variation known as the even symmetrical DCT which is most often used in signal coding applications. To derive the 1-dimensional DCT, one relates the N point sequence x(n) to a new 2N point sequence y(n) which is then related to its 2N point DFT Y(k). Then one relates Y(k) to $C_x(k)$ which is the 2N point DCT of x(n). Thus

$$x(n) \leftrightarrow y(n) \leftrightarrow Y(k) \leftrightarrow C_x(k)$$

Relation between x(n) and y(n) is

$$y(n) = x(n) + 2(2N-1-n)$$

$$y(n) = \begin{cases} x(n) & 0 \le n \le N-1 \\ x(2N-1-n) & N \le n \le 2N-1 \end{cases}$$



Relation between y(n) and Y(k) is

$$Y(k) = \sum_{n=0}^{2N-1} y(n) W_{2N}^{kn} \qquad 0 \le k \le 2N-1 \text{ where } W_{2N} = e^{-j\left(\frac{2\pi}{2N}\right)}$$

$$Y(k) = \sum_{n=0}^{N-1} x(n) W_{2N}^{kn} + \sum_{n=N}^{2N-1} x(2N-1-n) W_{2N}^{kn} \qquad 0 \le k \le 2N-1$$

$$Y(k) = W_{2N}^{\frac{-k}{2}} \sum_{n=0}^{N-1} 2x(n)\cos\left(\frac{\pi}{2N}k(2n+1)\right)$$

Relation between Y(k) and $C_x(k)$ is

$$C_x(k) = \begin{cases} W^{\frac{k}{2}} Y(k) & 0 \le k \le N-1 \\ 0 & \text{otherwise} \end{cases}$$

The DCT of x(n) is shown below

$$C_x(k) = \sum_{n=0}^{N-1} 2x(n)\cos\left(\left(\frac{\pi}{2N}\right)k(2n+1)\right) \qquad 0 \le k \le N-1$$
$$= 0 \qquad \text{otherwise}$$

where $C_x(k)$ is a N point sequence. If x(n) is real $C_x(k)$ is real.

An algorithm for the fast calculation of the DCT is

step 1. $y(n) = x(n) + x(2N-1-n)$

step 2. $Y(k) = DFT[y(n)]$ (2N point DFT computation)

step 3. $C_x(k) = \begin{cases} W_{2N}^{\frac{k}{2}} Y(k) & 0 \le k \le N-1 \\ 0 & \text{otherwise} \end{cases}$

To derive one dimensional (1-D) Inverse DCT (IDCT)

To derive the IDCT relate $C_x(k)$ to Y(k), Y(k) to y(n) and then y(n) to x(n) which is,

$C_x(k) \leftrightarrow Y(k) \leftrightarrow y(n) \leftrightarrow x(n)$

Although Y(k) is a 2N point sequence and $C_x(k)$ is an N point sequence, redundancy in Y(k) due to the symmetry of y(n), allows Y(k) to be determined from $C_x(k)$.

So $Y(k) = \begin{cases} W_{2N}^{-k} Y(2N-k) & 0 \le k \le 2N-1 \\ 0 & k = N \end{cases}$

Now the relation between Y(k) and y(n) through the 2N point inverse DFT is give by

$$y(n) = \frac{1}{2N}\sum_{k=0}^{2N-1} Y(k) W_{2N}^{-kn} \qquad 0 \le n \le 2N-1$$

x(n) can be related to y(n) by

$$x(n) = \begin{cases} y(n) & 0 \le n \le N-1 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{therefore } x(n) = \begin{cases} \frac{1}{N}\left[\frac{C_x(o)}{2} + \sum_{k=1}^{N-1} C_x(k)\cos\left(\frac{\pi}{2N}k(2n+1)\right)\right] & 0 \le n \le N-1 \\ 0 & \text{otherwise} \end{cases}$$

x(n) can then be expressed as the IDCT of $C_x(k)$, shown below

$$x(n) = \begin{cases} \frac{1}{N}\sum_{k=0}^{N-1} w(k) C_x(k)\cos\left(\frac{\pi}{2N}k(2n+1)\right) & 0 \le n \le N-1 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{where } w(k) = \begin{cases} \frac{1}{2} & k = 0 \\ 1 & 1 \le k \le N-1 \end{cases}$$

An algorithm for the fast computation of the IDCT is

$$\text{step 1. } Y(k) = \begin{cases} W_{2N}^{\frac{-k}{2}} C_x(k) & 0 \le k \le N-1 \\ 0 & k = N \\ -W_{2N}^{\frac{-k}{2}} C_x(2N-k) & N+1 \le k \le 2N-1 \end{cases}$$

step 2. $y(n) = IDFT[Y(k)]$ (2N point Inverse DFT computation)

$$\text{step 3. } x(n) = \begin{cases} y(n) & 0 \le n \le N-1 \\ 0 & \text{otherwise} \end{cases}$$

The even symmetrical DCT is more commonly used since the odd symmetrical DCT involves computing an odd length DFT, which is not very convenient when one is using FFT algorithms.

To derive the two dimensional (2-D) DCT

The one dimensional DCT can be extended to the two dimensional DCT. Let $x(n_1, n_2)$ denote a 2-D sequence of $N_1 \times N_2$ points that is zero outside $0 \le n_1 \le N_1 - 1,\ 0 \le n_2 \le N_2 - 1$. The 2-D DCT pair can be derived by relating $x(n_1, n_2)$ to a new $2N_1 \times 2N_2$ point sequence $y(n_1, n_2)$ which is then related to its $2N_1 \times 2N_2$ DFT $Y(k_1, k_2)$. Then relate $Y(k_1, k_2)$ to $C_x(k_1, k_2)$ the $N_1 \times N_2$ DCT

$$\begin{array}{cccc} N_1 \times N_2 & 2N_1 \times 2N_2 & 2N_1 \times 2N_2 & N_1 \times N_2 \\ \Downarrow & \Downarrow & \Downarrow & \Downarrow \\ x(n_1, n_2) & \leftrightarrow y(n_1, n_2) & \leftrightarrow Y(k_1, k_2) & \leftrightarrow C_x(k_1, k_2) \end{array}$$

$x(n_1, n_2)$ is related to $y(n_1, n_2)$ by

$$y(n_1, n_2) = x(n_1, n_2) + x(2N_1 - 1 - n_1, n_2) + x(n_1, 2N_2 - 1 - n_2) + x(2N_1 - 1 - n_1, 2N_2 - 1 - n_2)$$

The sequence $y(n_1, n_2)$ is related to $Y(k_1, k_2)$ by

$$Y(k_1, k_2) = DFT[y(n_1, n_2)]$$

The $N_1 \times N_2$ point DCT of $x(n_1, n_2)$, $C_x(k_1, k_2)$ is obtained from $Y(k_1, k_2)$ by

$$C_x(k_1, k_2) = \begin{cases} W_{2N_1}^{\frac{k_1}{2}} W_{2N_2}^{\frac{k_2}{2}} Y(k_1, k_2) \\ 0 \end{cases}$$

therefore the 2-D DCT is given by

$$C_x(k_1, k_2) = \begin{cases} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} 4x(n_1, n_2) \cos\left(\frac{\pi}{2N_1} k_1 (2n_1 + 1)\right) \cos\left(\frac{\pi}{2N_2} k_2 (2n_2 + 1)\right) & 0 \le k_1 \le N_1 - 1 \\ & 0 \le k_2 \le N_2 - 1 \\ 0 & \text{otherwise} \end{cases}$$

A Fast Algorithm for the computation of the 2-D Discrete Cosine Transform is shown

step 1. $y(n_1,n_2) = x(n_1,n_2) + x(2N_1 - 1 - n_1, n_2) + x(n_1, 2N_2 - 1 - n_2) + x(2N_1 - 1 - n_1, 2N_2 - 1 - n_2)$

step 2. $Y(k_1,k_2) = DFT[y(n_1,n_2)]$ ($2N_1 \times 2N_2$ point DFT computation)

step 3. $C_x(k_1,k_2) = W_{2N_1}^{\frac{k_1}{2}} W_{2N_2}^{\frac{k_2}{2}} Y(k_1,k_2)$ $0 \le k_1 \le N_1 - 1$, $0 \le k_2 \le N_2 - 1$

To derive the two dimensional (2-D) Inverse DCT (IDCT)

The inverse DCT can be derived by relating $C_x(k_1,k_2)$ to $Y(k_1,k_2)$, exploiting the redundancy in $Y(k_1,k_2)$ due to the symmetry of $y(n_1,n_2)$, $Y(k_1,k_2)$ can be related to $y(n_1,n_2)$ through the Inverse DFT relationship and then relating $y(n_1,n_2)$ to $x(n_1,n_2)$.

Therefore the 2-D IDCT is given by

$$x(n_1,n_2) = \begin{cases} \dfrac{1}{N_1 N_2} \displaystyle\sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N-1} w_1(k_1) w_2(k_2) C_x(k_1,k_2) \cos\left(\frac{\pi}{2N_1} k_1(2n_1+1)\right) \cos\left(\frac{\pi}{2N_2} k_2(2n_2+1)\right) \\ \qquad \text{for} \quad 0 \le n_1 \le N_1 - 1 \text{ and } 0 \le n_2 \le N_2 - 1 \\ 0 \qquad \text{otherwise} \end{cases}$$

where $w_1(k_1) = \begin{cases} \frac{1}{2} & k_1 = 0 \\ 1 & 1 \le k_1 \le N_1 - 1 \end{cases}$ and $w_2(k_2) = \begin{cases} \frac{1}{2} & k_2 = 0 \\ 1 & 1 \le k_2 \le N_2 - 1 \end{cases}$

An Algorithm for the fast computation of the IDCT is

step 1. $$Y(k_1,k_2) = \begin{cases} W_{2N_1}^{\frac{-k_1}{2}} W_{2N_2}^{\frac{-k_2}{2}} C_x(k_1,k_2) & 0 \le k_1 \le N_1 - 1,\ 0 \le k_2 \le N_2 - 1 \\ -W_{2N_1}^{\frac{-k_1}{2}} W_{2N_2}^{\frac{-k_2}{2}} C_x(2N_1 - k_1, k_2) & N_1 + 1 \le k_1 \le 2N_1 - 1,\ 0 \le k_2 \le N_2 - 1 \\ -W_{2N_1}^{\frac{-k_1}{2}} W_{2N_2}^{\frac{-k_2}{2}} C_x(k_1, 2N_2 - k_2) & 0 \le k_1 \le N_1 - 1,\ N_2 + 1 \le k_2 \le 2N_2 - 1 \\ W_{2N_1}^{\frac{-k_1}{2}} W_{2N_2}^{\frac{-k_2}{2}} C_x(2N_1 - k_1, 2N_2 - k_2) & N_1 + 1 \le k_1 \le 2N_1 - 1,\ N_2 + 1 \le k_2 \le 2N_2 - 1 \\ 0 \end{cases}$$

step 2. $y(n_1,n_2) = IDFT[Y(k_1,k_2)]$ ($2N_1 \times 2N_2$ point IDFT computation)

step 3. $x(n_1,n_2) = \begin{cases} y(n_1,n_2) & 0 \le n_1 \le N_1 - 1,\ 0 \le n_2 \le N_2 - 1 \\ 0 & \text{otherwise} \end{cases}$

Appendix C
BMP Format

Bitmap (BMP) File Format

BMP was designed to work with systems based on the Intel processors with their little-endian byte ordering scheme [55]. Whenever a multiple field is read, 16 or 32 bit integer, the first byte read will be the least significant and the last, the most significant.

There are three structures to describe the Windows bitmap file. These are

1. Bitmapfileheader
2. Bitmapheader
3. RGB

The **Bitmapfileheader** structure (14 Bytes) :

Type: Unsigned Integer 16 Bits: 2 Bytes: Type of image. The character BM indicates the bitmap mnemonic describing each type.

Size: Unsigned Integer 32 Bits: 4 Bytes: Size of entire bitmap file.

xHotspot: Integer 16 Bits: 2 Bytes: Not used for Windows Bitmaps.

yHotspot: Integer 16 Bits: 2 Bytes: Not used for Windows Bitmaps.

offsetToBits: Unsigned Integer 32 Bits: 4 Bytes: Byte offset from start of file to the first pixel data. This is 1078 Bytes for windows bitmaps 8 bits deep.

The **Bitmapheader** structure (40 Bytes) :

This can be a maximum of 64 Bytes long. Since Windows bitmaps do not use all the fields the Bitmapheader is 40 Bytes long.

Size: Unsigned Integer 32 Bits: 4 Bytes: Size of structure on disk. This contains the length in Bytes of the Bitmapheader structure. For windows bitmaps the value is 40 Bytes.

Width: Integer 32 Bits: 4 Bytes: Width in pixels of the image.

Height: Integer 32 Bits: 4 Bytes: Height in pixels of the image

numBitPlanes: Unsigned Integer 16 Bits: 2 Bytes: Indicates colour depth. The value is 1 since Windows only supports single plane BMPs.

numBitsPerPlane: Unsigned Integer 16 Bits: 2 Bytes: Indicates colour depth. Windows only supports 1,4 and 8 bit depths.

compressionscheme: Unsigned Integer 32 Bits: 4 Bytes: 0 indicates no compression.

sizeOfImageData: Unsigned Integer 32 Bits: 4 Bytes: Number of bytes an images pixel data consumes. Pointed to by offsetToBits. Usually 0 since size calculated using width, height and bit depth. For 256 x 256 size image the value is 65536.

xResolution: Unsigned Integer 32 Bits: 4 Bytes: Resolution in pixels. If non-zero, then can be used to generate a scaling factor to print the image at the proper size.

yResolution: Unsigned Integer 32 Bits.: 4 Bytes: Resolution in pixels. If non-zero, then can be used to generate a scaling factor to print the image at the proper size.

numColorsUsed: Unsigned Integer 32 Bits: 4 Bytes: Number of colours in colour table. 0 indicates all of them.

numImportantColors: Unsigned Integer 32 Bits: 4 Bytes: Number of colours required for proper rendering of the image. 0 indicates all of them.

The **RGB** structure:

Immediately following the Bitmapheader is a colour table. For windows bitmaps it is an array of RGB structures. When reading the pixel values of such a bitmap, each pixel's value will be an index into this array indicating the colour their value represents. For bitmaps with normal colour tables the length of the colour table is a function of the number of colours the image can have. The number of possible colours is $2^{bitdepth}$ where bit depth is the number of colour planes multiplied by the number of bits per plane. Each entry in the normal colour table is one RGB structure . First byte is the blue value, second green and the third red. There is also a further additional byte of padding. For windows bitmap at 8 bits deep therefore $2^8 = 256$ there 256 * 4 bytes gives 1024 bytes for RGB and padding.

Reading the Bits:

Once the headers and RGB have been read, the offsettobits field points to the start of the pixel data. Each row contains a multiple of 4 bytes. Each row is a packed array of pixel values. Each pixels bit width is the image's bit depth. For a bit depth of 1, each byte represents 8 pixels, the most significant bits within a byte are the left-most pixels. For bit depth of 4 each byte represents 2 pixels, the most significant 4 bits representing the left side pixel and the least significant 4 bits, the right side pixel. For a bit depth of 8, each byte represents a pixel.

Intensity Levels:

Human perception of light is generally described in terms of Brightness, Hue and Saturation. Brightness refers to how bright the light is. Hue refers to the colour, an attribute which allows the colours red and blue to be distinguished. Saturation, sometimes called chroma, refers to how vivid or dull a colour is.

For a black and white image, it can be represented by one number I where I is the luminance or intensity. A colour image can be viewed as three monochrome images (tristimulus). For a colour image this is represented by three numbers. The numbers used in practice are the RGB numbers.

These three values, RGB, can be transformed into another set of tristimulus values. One particular set is known as luminance-chrominance. The corresponding luminance-chrominance values YIQ are related to RGB by

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.583 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

and

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0.956 & 0.621 \\ 1 & -0.273 & -0.647 \\ 1 & -1.104 & 1.701 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

The Y component is called the luminance component and is primarily responsible for the perception of brightness of a colour image and can be used for black and white images. The I and Q components are the chrominance components and are responsible for the perception of hue and saturation of a colour image. One advantage of the YIQ set of tristimulus values over the RGB set of tristimulus values is that the Y component can be processed individually . Most high frequency components of an image are primarily in the Y component.

Appendix D
Lossless Compression Results for Test Images

D.1 Lossless Compression Results for 1024 x 1024 Images

The results when the lossless algorithms were applied to the 1024 x 1024 test images are shown in the tables D.1 to D.4.

	Static Huffman	Adaptive Huffman	Splay Trees	Arithmetic
Compression ratio	1:1.54	1:1.79	1:4.74	1:9.75
486 66 MHz				
Compression Time /s	24.31	40.51	15.05	273.15
Decompression Time /s	24.31	35.88	13.89	307.87
Pentium 90 MHz				
Compression Time /s	9.26	13.89	8.10	103.10
Decompression Time /s	9.26	13.89	6.94	105.79

Table D.1: Lossless compression results for chest x-ray (1024 X 1024) image

	Static Huffman	Adaptive Huffman	Splay Trees	Arithmetic
Compression ratio	1:1.53	1:1.72	1:4.46	1:9.09
486 66 MHz				
Compression Time /s	26.62	45.14	13.89	275.46
Decompression Time /s	27.78	39.35	13.89	311.34
Pentium 90 MHz				
Compression Time /s	9.26	13.89	3.47	72.92
Decompression Time /s	9.26	13.89	4.63	74.35

Table D.2: Lossless compression results for pelvis X-ray (1024 x 1024) image

	Static Huffman	Adaptive Huffman	Splay Trees	Arithmetic
Compression ratio	1:1.54	1:1.79	1:4.74	1:9.75
486 66 MHz				
Compression Time /s	24.31	40.51	15.05	273.15
Decompression Time /s	24.31	35.88	13.89	307.87
Pentium 90 MHz				
Compression Time /s	9.26	12.73	8.10	103.10
Decompression Time /s	9.26	13.89	6.94	105.79

Table D.3: Lossless compression results for mandrill Image (1024 x 1024)

	Static Huffman	Adaptive Huffman	Splay Trees	Arithmetic
Compression ratio	1:1.07	1:1.10	1:1.29	1:4.03
486 66 MHz				
Compression Time /s	32.4	69.44	32.4	307.87
Decompression Time /s	37.04	64.97	32.4	321.15
Pentium 90 MHz				
Compression Time /s	10.42	19.68	10.42	113.10
Decompression Time /s	10.42	19.68	10.42	115.87

Table D.4: Lossless compression results for Lenna image (1024 x 1024)

D.2 Lossless Compression Results for 512 x 512 Images

The results when the lossless algorithms are applied to the 512 x 512 test images are shown in the tables D.5 to D.8.

	Static Huffman	Adaptive Huffman	Splay Trees	Arithmetic
Compression ratio	1:1.53	1:1.68	1:3.71	1:6.96
486 66 MHz				
Compression Time /s	9.26	13.89	6.94	69.44
Decompression Time /s	9.26	12.73	5.79	79.86
Pentium 90 MHz				
Compression Time /s	5.79	6.94	1.16	23.15
Decompression Time /s	5.79	6.94	1.16	25.35

Table D.5: Lossless compression results for chest X-ray (512 x 512) image

	Static Huffman	Adaptive Huffman	Splay Trees	Arithmetic
Compression ratio	1:1.53	1:1.64	1:3.41	1:6.41
486 66 MHz				
Compression Time /s	11.57	18.52	8.10	79.86
Decompression Time /s	11.57	17.36	5.79	89.12
Pentium 90 MHz				
Compression Time /s	4.63	6.94	2.31	23.15
Decompression Time /s	4.63	6.94	2.31	25.46

Table D.6: Lossless compression results for pelvis X-ray (512 x 512) image

	Static Huffman	Adaptive Huffman	Splay Trees	Arithmetic
Compression ratio	1:1.48	1:1.52	1:1.55	1:2.04
486 66 MHz				
Compression Time /s	11.57	16.20	10.42	130.79
Decompression Time /s	11.57	15.05	9.26	140.05
Pentium 90 MHz				
Compression Time /s	4.63	6.94	2.31	38.19
Decompression Time /s	4.63	6.94	2.31	40.32

Table D.7: Lossless compression results for Mandrill image (512 x 512)

	Static Huffman	Adaptive Huffman	Splay Trees	Arithmetic
Compression ratio	1:1.07	1:1.09	1:1.04	1:2.05
486 66 MHz				
Compression Time /s	11.57	24.31	13.42	134.79
Decompression Time /s	13.89	24.31	12.46	145.05
Pentium 90 MHz				
Compression Time /s	5.79	8.10	3.47	39.29
Decompression Time /s	5.79	8.10	3.47	41.32

Table D.8: Lossless compression results for Lenna image (512 x 512)

Appendix E
QUANTISATION MATRICES

Shown below are the 8 x 8 quantisation matrices used in the DCT algorithm to compress the test images. The quantisation matrices are based on the human visual system's sensitivity. The quantisation levels are the same value all ranging the same distance from the origin.

Quality = 1

2	3	4	5	6	7	8	9
3	4	5	6	7	8	9	10
4	5	6	7	8	9	10	11
5	6	7	8	9	10	11	12
6	7	8	9	10	11	12	13
7	8	9	10	11	12	13	14
8	9	10	11	12	13	14	15
9	10	11	12	13	14	15	16

Quality = 3

4	7	10	13	16	19	22	25
7	10	13	16	19	22	25	28
10	13	16	19	22	25	28	31
13	16	19	22	25	28	31	34
16	19	22	25	28	31	34	37
19	22	25	28	31	34	37	40
22	25	28	31	34	37	40	43
25	28	31	34	37	40	43	46

Quality = 5

6	11	16	21	26	31	36	41
11	16	21	26	31	36	41	46
16	21	26	31	36	41	46	51
21	26	31	36	41	46	51	56
26	31	36	41	46	51	56	61
31	36	41	46	51	56	61	66
36	41	46	51	56	61	66	71
41	46	51	56	61	66	71	76

Quality = 7

8	15	22	29	36	43	50	57
15	22	29	36	43	50	57	64
22	29	36	43	50	57	64	71
29	36	43	50	57	64	71	78
36	43	50	57	64	71	78	85
43	50	57	64	71	78	85	92
50	57	64	71	78	85	92	99
57	64	71	78	85	92	99	106

Quality = 9

10	19	28	37	46	55	64	73
19	28	37	46	55	64	73	82
28	37	46	55	64	73	82	91
37	46	55	64	73	82	91	100
46	55	64	73	82	91	100	109
55	64	73	82	91	100	109	118
64	73	82	91	100	109	118	127
73	82	91	100	109	118	127	136

Quality = 11

12	33	44	55	66	77	88	99
33	44	55	66	77	88	99	1101
44	55	66	77	88	99	110	121
55	66	77	88	99	110	121	132
66	77	88	99	110	121	132	143
77	88	99	110	121	132	143	154
88	99	110	121	132	143	154	165
99	110.	121	132	143	154	165	176

Quality = 25

26	51	76	101	126	151	176	201
51	76	101	126	151	176	201	226
76	101	126	151	176	201	226	251
101	126	151	176	201	226	251	276
126	151	176	201	226	251	276	301
151	176	201	226	251	276	301	326
176	201	226	251	276	301	326	351
201	226	251	276	301	326	351	376

Appendix F
Lossy Compression Results for Test Images

F.1 8 x 8 quantisation matrix results for 1024 x 1024 images

The results when the lossy algorithm is applied to the 1024 x 1024 test images using a 8 x 8 quantisation matrix are shown in the tables F.1 to F.4.

	486 66 MHz	Pentium
Compression Time /s	78.06	18.52
Decompression Time /s	83.84	17.36

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	1.62	2.67	3.77	5.36	6.48	7.47	16.00
Visual Degradation	1	2	2	2	3	5	5
Using Run Length Encoding							
Compression Ratio	1:9.88	1:12.51	1:13.53	1:14.29	1:14.7	1:15.01	1:16.10
Using Splay Trees							
Compression Ratio	1:9.33	1:12.79	1:14.54	1:16.38	1:17.21	1:17.76	1:22.79

Table F.1: Lossy compression of chest X-ray image (1024 x 1024) using 8 x 8 matrix

	486 66 MHz	Pentium
Compression Time /s	78.06	18.52
Decompression Time /s	83.84	17.36

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	1.55	2.63	3.76	5.02	6.53	7.59	16.47
Visual Degradation	1	2	2	3	4	4	5
Using Run Length Encoding							
Compression Ratio	1:9.90	1:12.71	1:13.77	1:14.81	1:14.9	1:15.19	1:16.20
Using Splay Trees							
Compression Ratio	1:9.34	1:13.00	1:14.88	1:16.70	1:17.46	1:18.14	1:22.04

Table J.2: Lossy compression of pelvis X-ray image (1024 x 1024) using 8 x 8 matrix

	486 66 MHz	Pentium
Compression Time /s	78.06	19.68
Decompression Time /s	83.10 s	16.20

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	6.1	20.2	33.4	46.2	58.1	60.21	137.62
Visual Degradation	1	2	2	2	2	3	3
Using Run Length Encoding							
Compression Ratio	1:3.82	1:6.07	1:7.53	1:8.64	1:9.54	1:10.29	1:13.54
Using Splay Trees							
Compression Ratio	1:3.29	1:5.36	1:6.81	1:7.96	1:8.94	1:9.81	1:14.51

Table F.3: Lossy compression of mandrill image (1024 x 1024) using 8 x 8 matrix

	486 66 MHz	Pentium
Compression Time /s	78.06	19.68
Decompression Time /s	83.10	18.52

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	4.02	11.00	16.26	20.25	23.66	26.88	45.93
Visual Degradation	1	1	1	1	2	3	3
Using Run Length Encoding							
Compression Ratio	1:5.69	1:9.38	1:11.14	1:12.20	1:12.93	1:13.48	1:15.39
Using Splay Trees							
Compression Ratio	1:5.01	1:8.72	1:10.79	1:12.14	1:13.29	1:14.31	1:18.27

Table F.4: Lossy compression of Lenna image (1024 x 1024) using 8 x 8 matrix

F.2 8 x 8 quantisation matrix results for 512 x 512 images

The results when the lossy algorithm is applied to the 512 x 512 test images using a 8 x 8 quantisation matrix are shown in the tables J.5 to J.8.

	486 66 MHz	Pentium
Compression Time /s	27.04	4.63
Decompression Time /s	30.51	4.63

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	2.26	4.1	6.25	8.47	11.2	13.27	28.95
Visual Degradation	1	2	4	4	4	5	5
Using Run Length Encoding							
Compression Ratio	1:8.52	1:11.32	1:12.49	1:13.28	1:13.84	1:14.27	1:15.78
Using Splay Trees							
Compression Ratio	1:7.75	1:11.05	1:12.62	1:14.07	1:15.00	1:15.78	1:20.68

Table F.5: Lossy compression of chest X-ray image (512 x 512) using 8 x 8 matrix

	486 66 MHz	Pentium
Compression Time /s	22.41	5.79
Decompression Time /s	24.72	3.47

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	2.31	4.37	6.82	9.44	11.39	15.4	32.4
Visual Degradation	1	2	4	4	4	4	5
Using Run Length Encoding							
Compression Ratio	1:8.46	1:11.34	1:12.54	1:13.4	1:14	1:14.45	1:15.82
Using Splay Trees							
Compression Ratio	1:7.69	1:11.06	1:12.69	1:14.2	1:15.3	1:16.13	1:20.16

Table F.6: Lossy compression of pelvis X-ray image (512 x 512) using 8 x 8 matrix

	486 66 MHz	Pentium
Compression Time /s	22.41	4.63
Decompression Time /s	24.72	4.63

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	14.9	54.2	88.4	118	145	169.5	289
Visual Degradation	1	2	3	3	3	4	5
Using Run Length Encoding							
Compression Ratio	1:2.64	1:4.8	1:6.32	1:7.53	1:8.55	1:9.41	1:13.26
Using Splay Trees							
Compression Ratio	1:2.29	1:4.2	1:5.63	1:6.81	1:7.86	1:8.80	1:13.87

Table F.7: Lossy compression of mandrill image (512 x 512) using 8 x 8 matrix

	486 66 MHz	Pentium
Compression Time /s	22.41	4.63
Decompression Time /s	24.72	4.63

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	7.8	18.56	27.64	35.69	43.45	50.12	92.24
Visual Degradation	1	2	2	3	4	4	5
Using Run Length Encoding							
Compression Ratio	1:4.88	1:7.97	1:9.6	1:10.66	1:11.51	1:12.15	1:14.72
Using Splay Trees							
Compression Ratio	1:4.25	1:7.2	1:8.92	1:10.17	1:11.25	1:12.20	1:16.50

Table F.8: Lossy compression of Lenna image (512 x 512) using 8 x 8 matrix

F.3 16 x 16 quantisation matrix results for 1024 x 1024 images

The results when the lossy algorithm is applied to the 1024 x 1024 test images using a 16 x 16 quantisation matrix are shown in the tables F.9 to F.12.

	Pentium
Compression Time /s	30.09
Decompression Time /s	28.94

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	2.05	3.78	5.49	7.06	8.50	9.97	17.89
Visual Degradation	1	2	2	3	4	4	5
Using Run Length Encoding							
Compression Ratio	1:13.30	1:16.11	1:17.05	1:17.57	1:17.91	1:18.19	1:19.07
Using Splay Trees							
Compression Ratio	1:15.89	1:22.42	1:25.25	1:26.95	1:28.23	1:29.46	1:33.70

Table F.9: Lossy compression of chest X-ray image (1024 x 1024) using 16 x 16 matrix

	Pentium
Compression Time /s	30.09
Decompression Time /s	27.78

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	2.1	3.88	5.79	7.66	9.61	11.24	23.23
Visual Degradation	1	2	2	3	4	4	5
Using Run Length Encoding							
Compression Ratio	1:13.18	1:16.07	1:17.04	1:17.58	1:17.93	1:18.21	1:19.18
Using Splay Trees							
Compression Ratio	1:15.71	1:22.23	1:25.30	1:26.96	1:28.23	1:29.46	1:34.24

Table F.10: Lossy compression of pelvis X-ray image (1024 x 1024) using 16 x 16 matrix

	Pentium
Compression Time /s	30.09
Decompression Time /s	27.78

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	11.46	34.45	54.47	72.92	89.59	104.95	193.85
Visual Degradation	1	1	1	2	2	3	4
Using Run Length Encoding							
Compression Ratio	1:5.51	1:8.81	1:10.69	1:11.99	1:12.96	1:13.70	1:16.61
Using Splay Trees							
Compression Ratio	1:5.02	1:8.92	1:11.57	1:13.68	1:15.43	1:16.88	1:24.01

Table F.11: Lossy compression of mandrill image (1024 x 1024) using 16 x 16 matrix

	Pentium
Compression Time /s	30.09
Decompression Time /s	28.94

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	8.06	18.05	24.20	29.02	33.06	36.83	60.55
Visual Degradation	1	1	1	2	2	3	4
Using Run Length Encoding							
Compression Ratio	1:8.70	1:12.92	1:14.54	1:15.46	1:16.10	1:16.56	1:18.16
Using Splay Trees							
Compression Ratio	1:8.58	1:15.20	1:18.92	1:20.91	1:22.60	1:23.80	1:29.50

Table F.12: Lossy compression of Lenna image (1024 x 1024) using 16 x 16 matrix

F.4 16 x 16 quantisation matrix results for 512 x 512 images

The results when the lossy algorithm is applied to the 512 x 512 test images using a 16 x 16 quantisation matrix are shown in the tables F13 to F.16.

	Pentium
Compression Time /s	6.94
Decompression Time /s	8.10

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	3.10	6.44	10.48	14.13	18.70	21.32	42.54
Visual Degradation	1	2	3	4	4	5	5
Using Run Length Encoding							
Compression Ratio	1:11.62	1:14.62	1:15.83	1:16.5	1:16.99	1:17.29	1:18.50
Using Splay Trees							
Compression Ratio	1:12.74	1:18.38	1:21.50	1:23.31	1:24.89	1:25.94	1:30.49

Table F.13: Lossy compression of chest X-ray image (512 x 512) using 16 x 16 matrix

	Pentium
Compression Time /s	6.94
Decompression Time /s	6.94

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	3.38	7.49	11.45	16.83	21.48	25.21	49.16
Visual Degradation	1	2	3	4	4	5	5
Using Run Length Encoding							
Compression Ratio	1:11.43	1:14.43	1:15.64	1:16.45	1:16.95	1:17.33	1:18.55
Using Splay Trees							
Compression Ratio	1:12.47	1:18.08	1:20.90	1:23.34	1:24.81	1:26.10	1:30.78

Table F.14: Lossy compression of pelvis X-ray image (512 x 512) using 16 x 16 matrix

	Pentium
Compression Time /s	8.10
Decompression Time /s	8.10

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	33.4	96.28	145.24	185.39	218.05	248.28	373.73
Visual Degradation	1	1	2	3	4	5	5
Using Run Length Encoding							
Compression Ratio	1:3.99	1:7.36	1:9.52	1:11.09	1:12.27	1:13.26	1:16.74
Using Splay Trees							
Compression Ratio	1:3.94	1:7.09	1:9.80	1:12.13	1:14.11	1:15.89	1:24.27

Table F.15: Lossy compression of mandrill image (512 x 512) using 16 x 16 matrix

	Pentium
Compression Time /s	6.94
Decompression Time /s	6.94

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	13.12	29.5	43.21	55.03	65.11	75.08	127.12
Visual Degradation	1	2	3	4	4	4	5
Using Run Length Encoding							
Compression Ratio	1:7.24	1:10.94	1:12.69	1:13.84	1:14.62	1:15.24	1:17.37
Using Splay Trees							
Compression Ratio	1:6.89	1:11.86	1:14.76	1:17.03	1:18.70	1:20.17	1:26.31

Table F.16: Lossy compression of Lenna image (512 x 512) using 16 x 16 matrix

F.5 32 x 32 quantisation matrix results for 1024 x 1024 images

The results when the lossy algorithm is applied to the 1024 x 1024 test images using a 32 x 32 quantisation matrix are shown in the tables F.17 to F.20.

	Pentium
Compression Time /s	53.24
Decompression Time /s	52.08

Quality Factor	1	3	5	7	9	25
Mean Square Error	5.84	8.36	10.7	12.78	14.02	23.78
Visual Degradation	2	3	3	4	4	5
Using Run Length Encoding						
Compression Ratio	1:18.15	1:18.89	1:19.29	1:19.53	1:19.69	1:20.21
Using Splay Trees						
Compression Ratio	1:32.8	1:36.57	1:38.20	1:40.21	1:41.4	1:44.86

Table F.17: Lossy compression of chest X-ray image (1024 x 1024) using 32 x 32 matrix

	Pentium
Compression Time /s	53.24
Decompression Time /s	52.08

Quality Factor	1	3	5	7	9	25
Mean Square Error	6.39	9.93	12.96	15.66	18.04	31.94
Visual Degradation	2	3	3	4	4	5
Using Run Length Encoding						
Compression Ratio	1:18.01	1:18.79	1:19.21	1:19.47	1:19.15	1:20.15
Using Splay Trees						
Compression Ratio	1:32.26	1:36.08	1:38.24	1:39.75	1:41.05	1:45.01

Table F.18: Lossy compression of pelvis X-ray image (1024 x 1024) using 32 x 32 matrix

	Pentium
Compression Time /s	53.24
Decompression Time /s	52.08

Quality Factor	1	3	5	7	9	25
Mean Square Error	60.4	91.33	117.99	141.71	163.98	273.89
Visual Degradation	1	2	2	3	4	5
Using Run Length Encoding						
Compression Ratio	1:11.72	1:13.73	1:15.02	1:15.90	1:16.58	1:18.82
Using Splay Trees						
Compression Ratio	1:14.58	1:18.92	1:22.26	1:24.89	1:27.09	1:36.44

Table F.19: Lossy compression of mandrill image (1024 x 1024) using 32 x 32 matrix

	Pentium
Compression Time /s	53.24
Decompression Time /s	52.08

Quality Factor	1	3	5	7	9	25
Mean Square Error	26.58	34.17	40.48	46.69	52.70	85.83
Visual Degradation	1	2	3	4	4	5
Using Run Length Encoding						
Compression Ratio	1:15.61	1:16.89	1:17.57	1:18.06	1:18.40	1:19.54
Using Splay Trees						
Compression Ratio	1:23.80	1:28.07	1:30.74	1:32.81	1:34.31	1:40.35

Table F.20: Lossy compression of Lenna image (1024 x 1024) using 32 x 32 matrix

F.6 32 x 32 quantisation matrix results for 512 x 512 images

The results when the lossy algorithm is applied to the 512 x 512 test images using a 32 x 32 quantisation matrix are shown in the tables F.21 to F.24.

	Pentium
Compression Time /s	12.73
Decompression Time /s	12.73

Quality Factor	1	3	5	7	9	25
Mean Square Error	12.01	18.89	24.05	28.86	34.48	65.15
Visual Degradation	5	5	5	5	5	5
Using Run Length Encoding						
Compression Ratio	1:16.87	1:17.81	1:18.35	1:18.71	1:19.01	1:19.86
Using Splay Trees						
Compression Ratio	1:27.45	1:31.21	1:33.61	1:35.38	1:36.86	1:41.73

Table F.21: Lossy compression of chest X-ray image (512 X 512) using 32 x 32 matrix

	Pentium
Compression Time /s	12.73
Decompression Time /s	12.73

Quality Factor	1	3	5	7	9	25
Mean Square Error	13.56	22.09	28.29	34.13	38.99	76.91
Visual Degradation	5	5	5	5	5	5
Using Run Length Encoding						
Compression Ratio	1:16.70	1:17.73	1:18.27	1:18.65	1:18.92	1:19.84
Using Splay Trees						
Compression Ratio	1:26.84	1:30.94	1:33.37	1:35.18	1:36.34	1:41.87

Table F.22: Lossy compression of pelvis X-ray image (512 x 512) using 32 x 32 matrix

	Pentium
Compression Time /s	13.89
Decompression Time /s	12.73

Quality Factor	1	3	5	7	9	25
Mean Square Error	166.66	230.54	278.33	315.37	344.24	459.37
Visual Degradation	3	4	5	5	5	5
Using Run Length Encoding						
Compression Ratio	1:10.53	1:13.06	1:14.68	1:15.81	1:16.62	1:19.04
Using Splay Trees						
Compression Ratio	1:12.26	1:17.24	1:21.20	1:24.44	1:27.08	1:37.34

Table F.23: Lossy compression of mandrill image (512 x 512) using 32 x 32 matrix

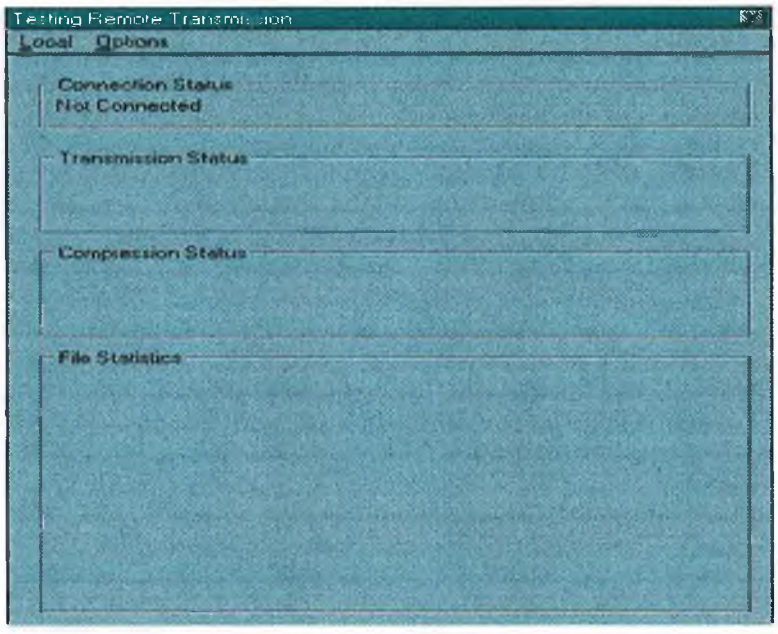
	Pentium
Compression Time /s	12.73
Decompression Time /s	13.81

Quality Factor	1	3	5	7	9	25
Mean Square Error	48.70	69.17	85.83	100.46	113.78	181.53
Visual Degradation	3	4	5	5	5	5
Using Run Length Encoding						
Compression Ratio	1:13.54	1:15.28	1:16.26	1:16.93	1:17.42	1:19.01
Using Splay Trees						
Compression Ratio	1:18.24	1:22.67	1:25.68	1:28.05	1:29.86	1:37.14

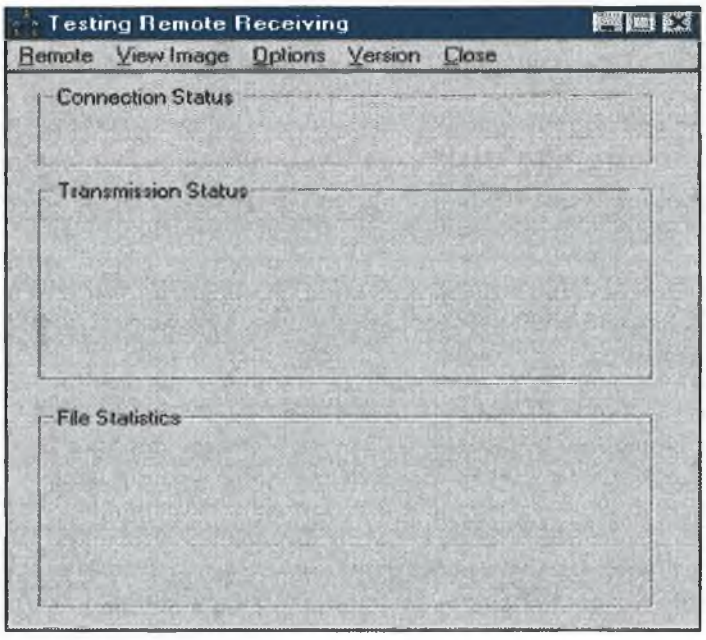
Table F.24: Lossy compression of Lenna image (512 x 512) using 32 x 32 matrix

Appendix G
Visual Basic Software Application

There are two applications, the server application and the client application. The server application, shown below, is situated on the remote system where the images are archived. The server application is concerned with decompressing the image from the archive using the lossless algorithm, compressing the image for transmission using the lossy algorithm and then transmitting the image to the client.



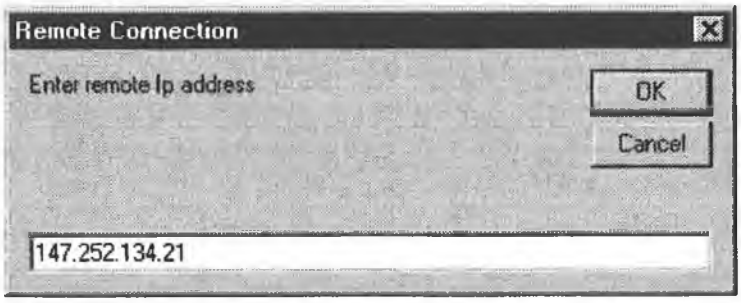
Shown below is the client application. The client is concerned with receiving the image, decompressing the image using a lossy algorithm and displaying the image on the screen.



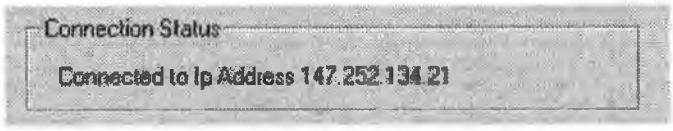
To View a remote image.
Step 1:Set up the IP address of the local PC (client) using the options menu choice.

Step 2:Connect to remote system by choosing the open remote on the menu.

Another window will open allowing the user to enter the IP address of the remote system (server)

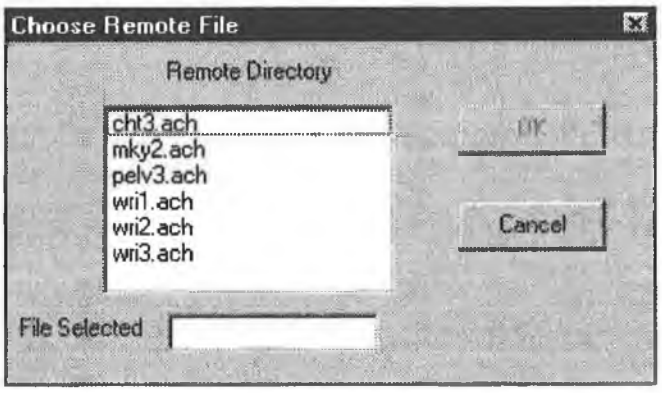


The client will now try to connect to the server and if successful will change the connection status to display where the client has been connected, thus.



Step 3:Once the client has connected to the server, more menu options will be available. To view a remote image choose the menu option View Remote Image.

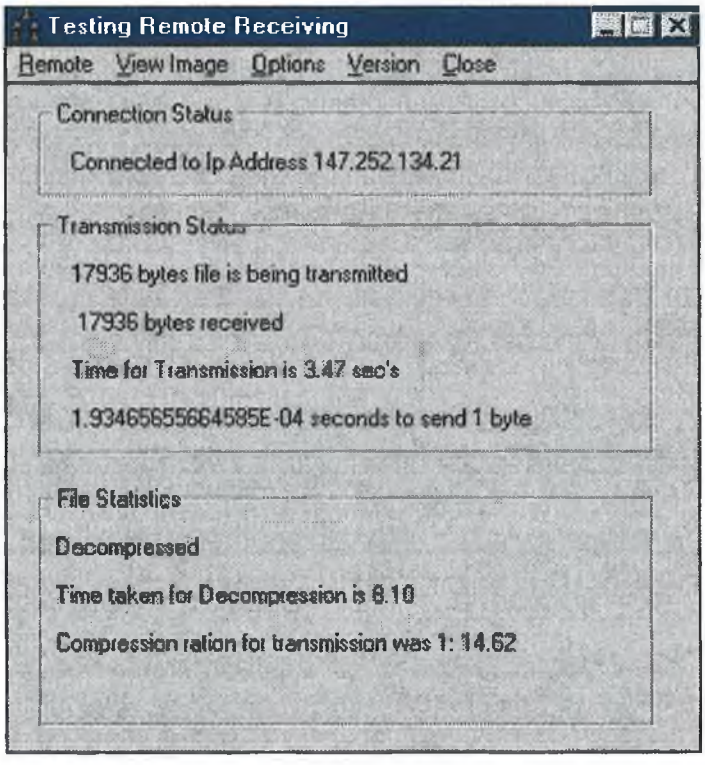
The server transmits to the client a directory listing of the archived images on the server.



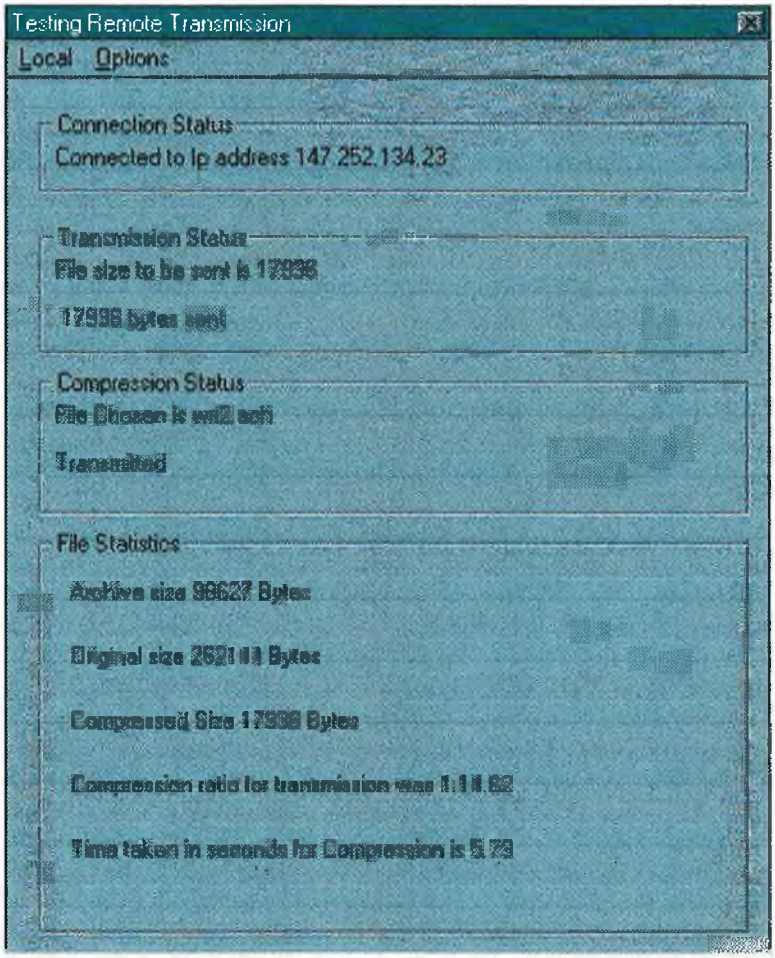
The client can now choose an image to be viewed and the name of this file will be transmitted to the server and the server will transmit a lossy compressed image at a default resolution to the client. The file chosen in this case is wri2.ach, an archived wrist x-ray image of size 512 x 512 pixels at an intensity resolution of 8 bits/pixel.

Step 4: While the image is being transmitted the client application will inform the user how much of the image is left to be transmitted. This is shown in the Transmission Status section. This also displays the time the image took to be transmitted and how fast the transfer system is.

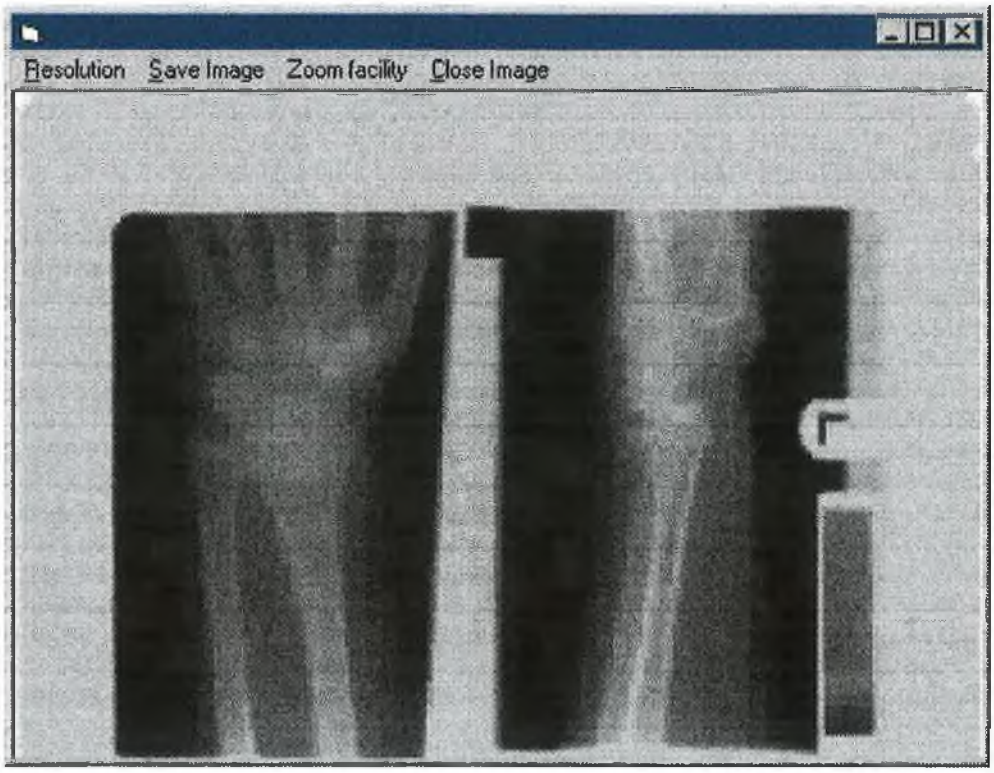
The file statistics section displays what the actual client application is doing. e.g. transmitting or decompressing. This section also displays information on decompression time and compression ratio. All the sections can be turned on or off using the options menu selection.



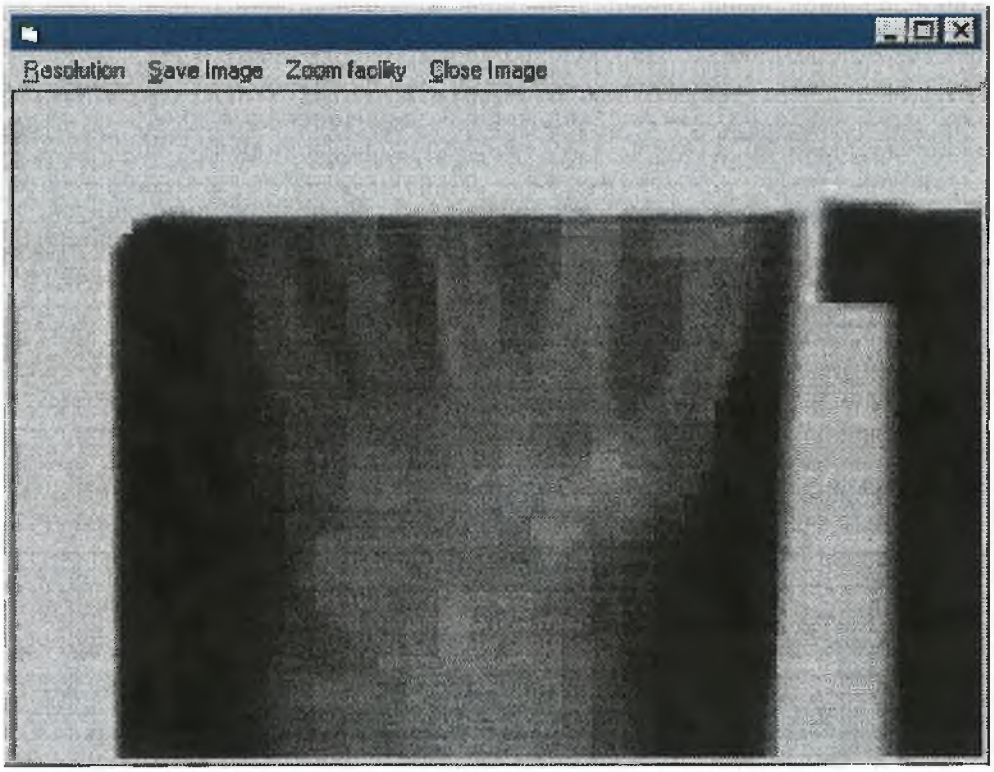
The server application also displays information about the transmission. The connection status section displays to which client it has currently transmitted an image Transmission status section displays how much of the image has been transmitted The file statistics section displays information about the size of the archived image, the size of the original image, the size of the image to be transmitted, compression ratio and the time taken to compress the image. The sections can be turned on or off using in the options menu selection.



When the image has been transmitted and decompressed another window opens and displays the image. This image is at a default resolution.

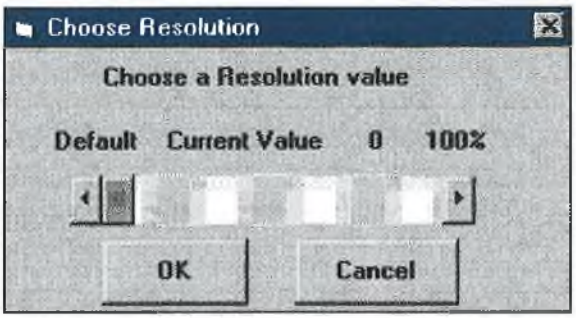


The user can now choose to either increase resolution on the entire image or choose a section of the image by using the zoom menu

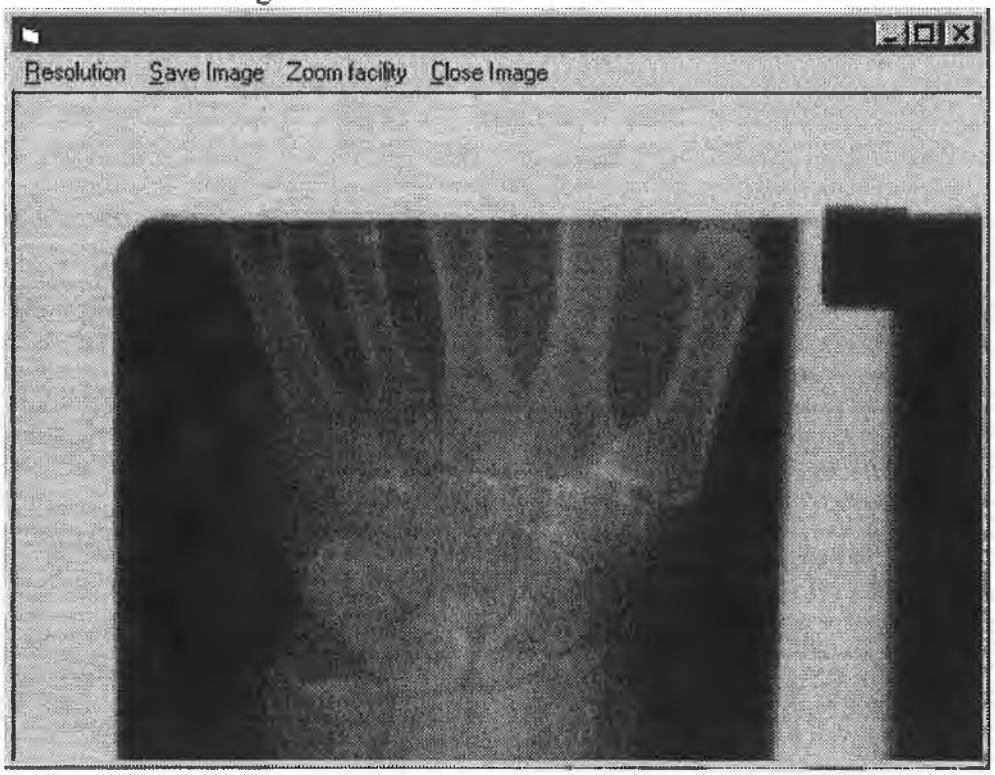


Once the section has been zoomed in, the client can choose to increase the resolution of this section by choosing the increase resolution on the menu

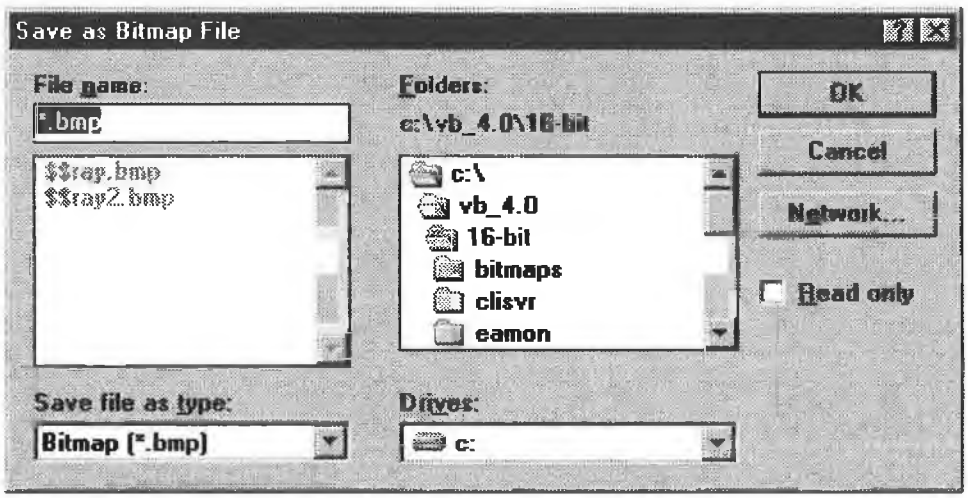
Another window will open allowing the client to choose a new resolution



Once the resolution has been chosen, information about the size and location of the section to be transmitted and also the new resolution is transmitted to the server. Only the section of the image chosen will be transmitted from the server to the client.



This image can now be saved to the local hard disk using the save image menu



Appendix H
Comparison of Transmitting only Difference

The results when transmitting the difference are shown in the tables H.1 to H.4.

	1	2	3	4	5	6	7	8
Compression Ratio	1:9.87	1:10.99	1:9.32	1:10.75	1:13.28	1:13.91	1:15.87	1:17.27
Compression Time /s	19.68	18.52	19.68	19.68	28.94	28.94	30.09	28.94
Decompression Time /s	16.20	17.36	17.36	17.36	27.78	27.78	26.62	27.78
Time For Difference /s	--------	8.10	-------	6.94	--------	8.10	--------	6.94
Time for Addition /s	----------	6.94	-------	6.94	--------	6.94	--------	8.10
Transmission Time /s	7.12	6.39	7.54	6.53	5.29	5.05	4.43	4.07
Overall Time /s	43	57.31	44.58	57.48	62.01	76.81	61.14	75.83

Table H.1 : Comparison of difference methods for chest X-ray image

	1	2	3	4	5	6	7	8
Compression Ratio	1:9.88	1:10.97	1:9.33	1:10.71	1:13.16	1:13.77	1:15.69	1:17.03
Compression Time /s	18.52	18.52	19.68	18.52	28.94	28.94	30.09	30.09
Decompression Time /s	17.36	17.36	16.20	17.36	27.78	27.78	27.78	27.78
Time For Difference /s	----------	6.94	-------	8.10	--------	8.10	--------	6.94
Time for Addition /s	----------	6.94	-------	8.10	--------	6.94	--------	6.94
Transmission Time /s	7.11	6.41	7.53	6.56	5.34	5.10	4.48	4.13
Overall Time /s	42.99	56.17	43.41	58.64	62.06	76.86	62.35	75.68

Table H.2 : Comparison of difference methods for pelvis X-ray image

	1	2	3	4	5	6	7	8
Compression Ratio	1:3.82	1:4.05	1:3.29	1:3.53	1:5.50	1:5.71	1:5.01	1:5.28
Compression Time /s	19.68	20.83	23.15	21.99	30.09	30.09	31.25	31.25
Decompression Time /s	18.52	17.36	18.52	18.52	28.94	28.94	28.94	28.94
Time For Difference /s	----------	6.94	-------	6.94	--------	6.94	--------	6.94
Time for Addition /s	----------	8.10	-------	6.94	--------	6.94	--------	6.94
Transmission Time /s	18.41	17.35	21.39	19.91	12.77	12.30	14.01	13.30
Overall Time /s	56.61	70.58	63.06	74.3	71.8	85.21	74.2	87.37

Table H.3 : Comparison of difference methods for mandrill image

	1	2	3	4	5	6	7	8
Compression Ratio	1:5.69	1:6.10	1:5.00	1:5.47	1:8.69	1:9.06	1:8.57	1:9.14
Compression Time /s	19.68	19.68	20.83	20.83	30.09	28.94	30.09	31.25
Decompression Time /s	17.36	17.36	17.36	17.36	27.78	27.78	27.78	27.78
Time For Difference /s	----------	6.94	-------	6.94	--------	8.10	--------	6.94
Time for Addition /s	----------	6.94	-------	8.10	--------	8.10	--------	6.94
Transmission Time /s	12.35	11.52	14.04	12.85	8.08	7.75	8.20	7.69
Overall Time /s	49.39	62.44	52.23	66.08	65.95	80.67	66.07	80.6

Table H.4: Comparison of difference methods for Lenna image

Appendix I
Abstract of Presentation given at the First Annual Scientific Meeting of the Biomedical Engineering Association of Ireland on 9th March 1996

Medical Image Compression for Interactive Remote Diagnostics over the Internet using a PC.

Raymond Rochford, Eamon Maher, Bob Lawlor, Frank Duignan

Abstract

This paper discusses a software application for the use of lossless algorithms in archiving and lossy algorithms for the transmission of medical images over the Internet.

A grey scale image 1024 x 1024 pixels 8 bits deep occupies approximately 1 M Byte of disk space. There is a great necessity to reduce this excessive use of storage, and also retrieval and transmission times.

Due to the nature of x-rays and legal considerations the images must be archived without any loss of information, therefore for medical archiving the images must be stored using a lossless algorithm

The lossless algorithms investigated are Huffman coding, arithmetic coding and the use of splay trees. Each concept will compress and decompress a number of different test images and the compression ratio, compression time and decompression time will be recorded for each concept on each image. The objective is to find the optimal method to compress an x-ray

For a clinician to remotely view an image all that is required is a PC with a network card installed. The network card can then be connected to a LAN, a WAN or a telephone line depending on the location of the clinician. Methods to ensure the complete and accurate transmission of an image will be addressed, as will the contingency for an interrupted connection.

To remotely view an image the image is compressed lossy and then transmitted to the clinician. Lossy compression is acceptable since the final receiver is the human eye and the eye can tolerate certain image imperfections. As the clinician requires increased resolution of all or a section of the image, additional information is transmitted accordingly.

The Discrete Cosine Transform (DCT) is used extensively in image coding since it is a lossy algorithm. Algorithms using the DCT will be investigated and compression ratio, compression time, decompression time and mean square error will be calculated for each image.

Appendix J
Paper Presented at the IDSPCC, Trinity College Dublin, 24-25 June 1996, pp. 181-188

Medical Image Compression for Interactive Remote Diagnostics Over the Internet Using a PC

Raymond Rochford, Eamon Maher, Bob Lawlor, Frank Duignan

Department of Control Systems and Electrical Engineering
DIT Kevin St.
Dublin 8

Abstract

This paper discusses a software application for the use of lossless algorithms in archiving and lossy algorithms in the transmission of medical images. The lossless algorithms investigated were Huffman coding, Arithmetic coding and Huffman coding using Splay Trees. The algorithms were tested on four images and in each case the compression ratio, compression time and decompression time were recorded for each.

The lossy algorithm investigated was transform coding using the Discrete Cosine Transform. Again this algorithm was tested on four images and compression ratio, compression time, decompression time and mean square error was recorded for different quality factors.

The development of a graphic user interface for the transmission of images from a server to a client was investigated. The algorithms were written in C, developed into DLLs using Visual C++ and called in Visual Basic where the interface was developed.

Keywords

Lossless compression, Lossy compression, Internet, Huffman Coding, Arithmetic Coding, Splay Trees, Discrete Cosine Transform (DCT)

1. Introduction

This paper discusses a software application for the use of lossless algorithms in archiving and lossy algorithms in the transmission of medical images over the Internet. All software is designed for use on any Windows based platform such as the 486 or Pentium PC as found in numerous medical environments.

A grey scale image comprising of 1024 X 1024 pixels occupies approximately 1M byte of disk space. There is a great necessity to reduce the excessive use of storage, and also small retrieval times. There is also a necessity to reduce the transmission time for an image.

Topics tackled in this paper are

- The use of a fast and effective lossless algorithm which would compress an image by approximately 2:1 with a corresponding decrease in retrieval time for archiving.
- A lossy algorithm with an approximate compression of 1:10 to reduce transmission time in remote diagnostics.
- The development, using Visual Basic and Visual C++, of a user friendly graphic interface so that a clinician can perform interactive remote diagnosis of an image.

2. Lossless Compression For Archiving

Due to the nature of x-rays and legal considerations the images must be archived without any loss of information, therefore for medical archiving the images must be stored using a lossless algorithm. Lossless coding guarantees that the decompressed image is absolutely identical to the image before compression.

All compression techniques are based on the exploitation of information redundancy that exists in digital images. The redundancy stems from the statistics of the image data (e.g. strong spatial correlation). The aim is to represent the image using a lower number of bits per pixel without losing the ability to reconstruct the image. No non-redundant image data may be lost in the data compression process, otherwise error free reconstruction is impossible.

Statistical redundancy is directly related to the image data probability distribution and can be treated by information theory techniques using image entropy algorithms. Its removal results in lossless image compression techniques such as Huffman coding and Arithmetic coding.

2.1 Huffman Coding

Huffman coding [1] is based on the fact that certain symbols, in this case intensities, appear more often than others in an input stream. The average number of bits per pixel can be reduced by assigning binary codes of different bit lengths to the various image intensities.

Once the probability density function of the image intensities is known, short codewords can be assigned to intensities having a high probability of occurrence and larger codewords can be assigned to less frequent image intensity levels. The codewords are then joined to form a binary data stream. This stream must be decoded at the receiver end and therefore the combination of the joined codewords must be decipherable, that is, no codeword can be the same as the first bits, the prefix, of another codeword. This ensures that when a codeword has been received, there should be no possibility that it is part of a longer word.

2.2 Arithmetic Coding

Arithmetic coding [2] is based on the same principle as Huffman coding but instead of replacing an input symbol with a specific code, it replaces a stream of input symbols with a single floating point number. Once the symbol probabilities are known, each symbol is assigned a range along a "probability line" from 0 to 1 which corresponds to its occurrence. As each symbol is encoded it creates a subrange and the entire input stream becomes a single floating point number.

2.3 Splay Tree

A Huffman tree is a weight balanced tree where each leaf is weighted with the intensity frequency and the internal nodes have no weighting. When splaying [3] is applied to prefix code trees, the path from the root to the target is reduced by a factor of two thus reducing the number of bits required in the input stream.

The Huffman compression algorithm requires the use of a tree balancing scheme and since splay trees are ordered binary search trees, when applied to Huffman coding compression algorithm leads to a locally adaptive compression algorithm.

3. Lossy Compression For Transmission

Lossy compression can be used when the information loss can be tolerated at the receiving end. For a clinician to remotely view an image all that is required is a PC with a network card installed. The network card can be connected to a local area, LAN (within a medical site), a wide area network, WAN (between medical sites), or a telephone line (for remote users) depending on the location of the clinician.

If the remote computer (client) connects to the server over a telephone line the cost of connection time needs to be considered. To minimise the time (and hence the cost) of communicating over the specific connection, the archived image is decompressed and compressed in a lossy format and then transmitted to the clinician. Lossy compression is acceptable since the final receiver is the human eye and the eye can tolerate a certain level of image imperfection. As the clinician requires increased resolution of all or a section of the image, the additional information is transmitted accordingly.

3.1 Transform Image Coding

One approach is to use image transforms [4] to concentrate the image energy in a few transform coefficients. An image is transformed to a domain different from the intensity domain and the transform coefficients are then coded. If the energy packing is obtained, a large number of transform coefficients can be discarded and the rest coded with variable length codewords thereby resulting in data compression.

The energy packing property relies on the fact that a large amount of energy is concentrated in a small fraction of the transform coefficients. The DC coefficients and some other low frequency coefficients tend to concentrate most of the signal energy. This is because images have large regions where the intensities change slowly. High frequency components are mainly associated with sharp discontinuities in the image.

Many transform coefficients can be discarded without much loss of information. Since most of the signal energy is concentrated in the DC coefficient and some other low frequencies an image can be reconstructed without significant loss of quality and intelligibility from a small percentage of transform coefficients. A varying number of bits can be allocated to the remaining coefficients and the result is the encoded image.

Transform coding can be generalised into four stages.

- Image Subdivision
- Image Transformation
- Coefficient Quantisation
- Encoding

3.1.1 Image Subdivision

It is not advisable to apply one transform to the entire image because of the changing image statistics in the various image regions. The image is split into a number of non-overlapping blocks that are coded independently. Subdivision reduces storage and computational requirements. Since one subdivision is processed at a time, it is not necessary to store the entire image in memory. As the image is divided into smaller segments, transform coding exploits less of the correlation among image pixel intensities and the correlation among neighbouring subdivisions increases. There is a limit on the subdivision size.

3.1.2 Image Transformation

The Discrete Cosine Transform (DCT) [5] is a transform which has energy packing properties. The DCT is used because of its nearly optimal performance in typical images having high correlation in adjacent image pixels.

3.1.3 Coefficient Quantisation)

The sub-image is transformed from intensity level to the transform domain and quantised. The zone shapes (which defines quantisation resolution for each coefficient) are consistent with the observation that most of the energy in typical images is concentrated in the low frequency region (zonal coding).

Another form of coefficient quantisation is threshold coding. Transform coefficients are compared with a threshold and those above the threshold are coded.

It is beneficial to allocate more bits to a coefficient with a large expected variance. For the DCT the expected variance is much larger for low frequency coefficients

than for high frequency coefficients. Quantisation is not a reversible operation making this a lossy method.

3.1.4 Encoding

The resulting remaining coefficients can then be encoded using a lossless coding scheme such as Huffman coding or Run Length encoding.

4. Results

Shown in figure 1 is the flow chart for the application. The algorithm for archiving is lossless and for transmission is lossy. The x-ray is archived on the server archive database and when there was a request for transmission from the client, an x-ray would be chosen, decompressed from the archive using the lossless algorithm, compressed at a default resolution using the lossy algorithm and then transmitted over the Internet to the client where it is decompressed and viewed on the monitor.

The user can then zoom in on a section of the x-ray and then request further resolution of that area. Only this section of the x-ray is transmitted at the new resolution to the client where it can be viewed. The user can then restore to the full x-ray at default resolution and view another section of the x-ray at increased resolution. This is to ensure that only the section that is of interest is sent to the user at the required resolution so that bandwidth is not wasted in sending the whole image at a high resolution when only a small section is required.

Test images are 1024 X 1024 pixels at an intensity resolution of 8 bits/pixel as shown in figures 2 through 5. The lossless algorithms tested were static Huffman coding, adaptive Huffman coding, Arithmetic coding and Huffman coding using Splay Trees. These algorithms were written in C and transformed into Dynamic Linked Libraries using Visual C++ and called in Visual Basic. Algorithms were tested on a 486 66 MHz and Pentium 90 MHz PC. The four algorithms were applied to each test image and in each case compression ratio, compression and decompression times recorded. Figure 6 shows the average compression time vs. average compression ratio for the four algorithms. The mean square error was also recorded for each algorithm in each case and was seen to be zero. This was done to verify that the algorithms used were lossless.

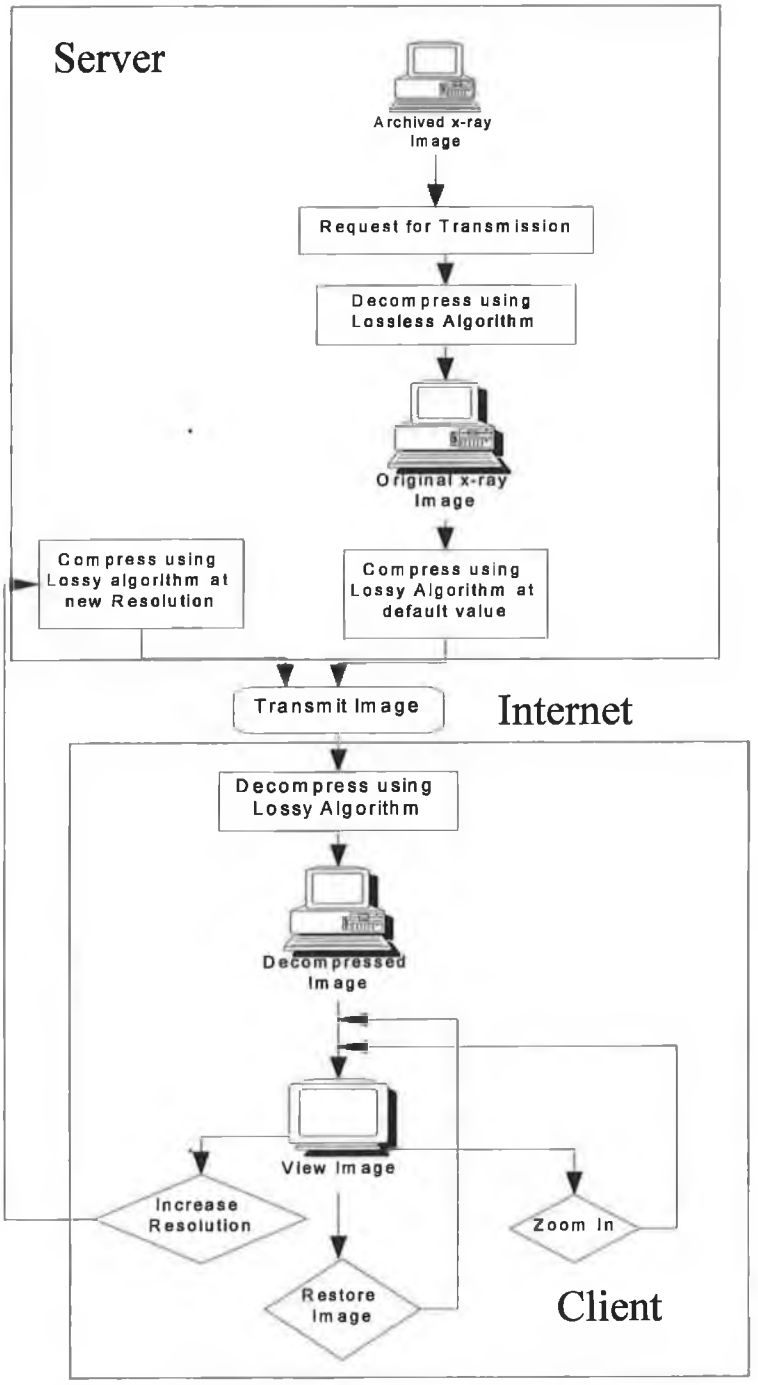


Figure1: Flow Chart of Application

Figure 2: Wrist x-ray

Figure 3: Mandrill

Figure 4: Pelvis x-ray

Figure 5: Chest x-ray

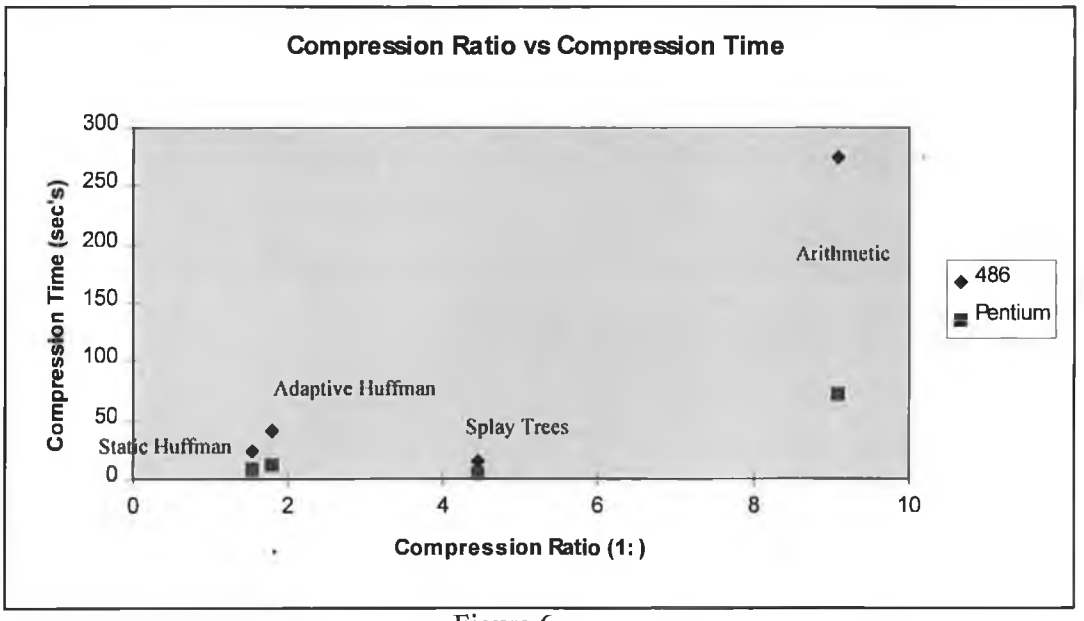


Figure 6

As can be seen Huffman coding schemes do not achieve high compression ratios. Arithmetic coding achieves high compression ratio but this requires an increase in compression and decompression times. The algorithm using splay tree performs the best when a high compression ratio is required with a fast compression and decompression time. The Huffman coding algorithm using splay trees will be used as the lossless algorithm in the software application.

The lossy algorithm was also developed as a DLL and used in Visual Basic and tested on a 486 66 MHz and Pentium 90 MHz PC. Compression ratio, mean square error, compression time and decompression time were recorded for each image using different quality factors. As the quality factor increases the resolution decreases. So a quality factor of 1 would have the highest resolution. Shown below in table 1 are the results recorded for the wrist x-ray. Results recorded for the other images are similar. The DCT algorithm was tested using different encoding schemes as seen in table 1 and also using different size quantisation matrices.

This application uses zonal coding to allow the client the ability to choose the resolution of the image to be viewed. Threshold coding as investigated and was seen to give an optimal compression ratio but didn't have the flexibility to allow the user to choose differing resolutions. This also decreases the amount of bandwidth required since the client can choose a low resolution on a large image and a high resolution on only a small section of that image.

	486 33 MHz	Pentium
Compression Time	82.69 sec's	18.52 sec's
Decompression Time	83.84 sec's	17.36 sec's

1	No Visual Degradation
2	Low Level of Degradation
3	Visually Degraded
4	Significant Degradation
5	Highly Degraded

Quality Factor	1	3	5	7	9	11	25
Mean Square Error	1.42	2.82	4.15	5.53	6.88	8.26	20.4
Visual Degradation	1	2	2	2	3	4	5
Using Run Length Encoding							
Compression Ratio	1:9.39	1:11.77	1:12.69	1:13.24	1:13.74	1:14.03	1:15.20
Using Splay Trees							
Compression Ratio	1:8.90	1:12.75	1:13.31	1:14.24	1:14.24	1:16.55	1:18.45

Table 1: Lossy compression of Wrist x-ray

Figure 7 shows a comparison between overall viewing times using different methods when using a quality factor of 25 on a Pentium. As the size of the quantisation matrix increases the compression ratio increases but so also does the compression time, so there is a tradeoff for optimal overall performance. This was tested on a LAN so the transmission times are short. As can be seen for such a short transmission time, method 2 or 3 performs the best but when using a different transmission medium, for example a modem where the cost of transmission would be vital, the transmission time will increase so another method may be applicable since it would have a greater compression ratio. Also as the traffic on a LAN increases the transmission time will increase so again another method may be applicable.

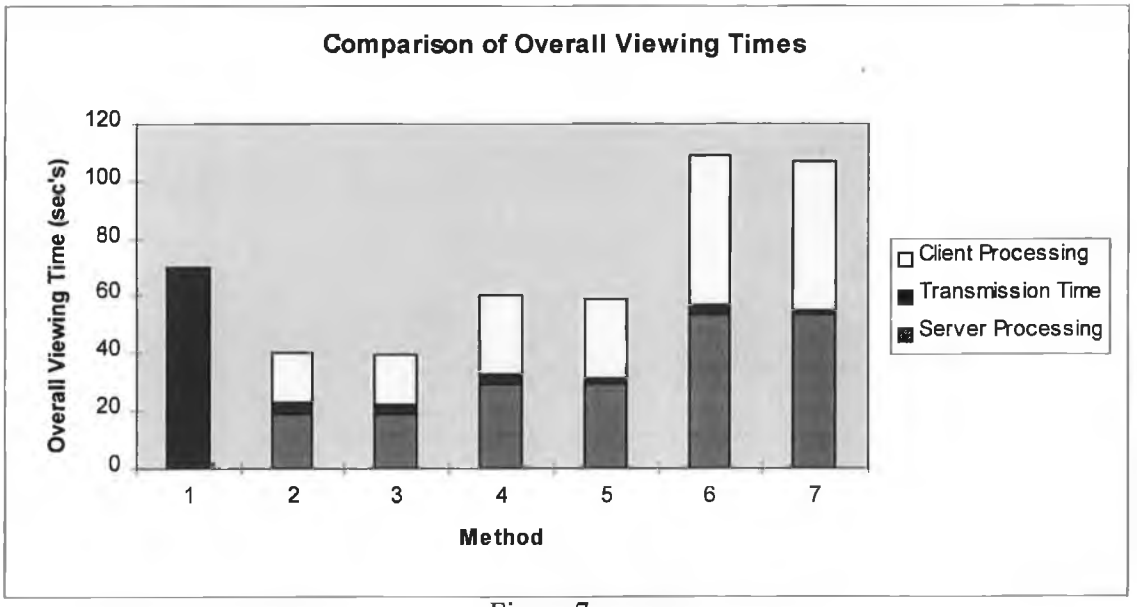


Figure 7

	METHOD
1	Image Transmitted without Compression
2	8 x 8 DCT
3	8 x 8 DCT with Splay
4	16 x 16 DCT
5	16 x 16 DCT with Splay
6	32 x 32 DCT
7	32 x 32 DCT with splay

5 Discussion

The lossless algorithm implemented was Huffman coding using Splay trees. This provides the optimal tradeoff between compression ratio and compression time. The lossy method employed was Transform Image coding using DCT with a quantisation matrix of size 8 X 8 and Huffman coding using Splay trees as the final lossless encoding scheme. The image will be transmitted from server to client using this lossy method. The client can calculate the transmission time and decide which method would be preferable to use to transmit any other images or sections of images. If the transmission rate was slow so that a method incorporating a higher compression ratio would be optimal. Again there is a tradeoff between transmission time and processing time. So if a modem at 14.4 Kb/sec is being used a method with a higher compression ratio would be required to reduce transmission time.

When the client requests further resolution of all the image or a section of the image only the difference is transmitted. This again gives the user control over the selection of resolution thus compression ratio thus optimising the use of bandwidth.

With the latest generation of Web Browsers and the coming of age of Java Applets the inclusion of interactive remote diagnostics using the World Wide Web could

Future developments of the software application would include the investigation of wavelet compression and fractal compression as means of a higher compressed lossy image. Again there would be a tradeoff between transmission time and processing time. Also the inclusion of some spatial filtering in the client software.

References

[1] Gilbert Held, *Data Compression Techniques and Applications*, Wiley
[2] Mark Nelson, *The Data Compression Book*, M&T Books
[3] Douglas W. Jones, *Application of Splay Trees to Data Compression*, Commun. ACM August 1988 Vol. 31 No. 8
[4] N.S Jayant, Peter Noll, *Digital Coding of Waveforms*, Prentice Hall.
[5] K.R. Rao, J. Yip, *Discrete Cosine Transform: Algorithms, Advantages and Application*, Academic Press Inc., San. Diego, 1990