Transform-Based Medical Image Compression

by

Katarzyna Ania Muldner

B.Sc., Acadia University, 1995

A Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming

to the required standard

[redacted]

Dr. R. N. Horspool, Supervisor (Department of Computer Science)

[redacted]

Dr. M. Serra, Departmental Member (Department of Computer Science)

[redacted]

Dr. R. Illner, Outside Member (Department of Mathematics and Statistics)

[redacted]

Dr. M.L. Lesperance, External Examiner (Department of Mathematics and Statistics)

Supervisor: Dr. R. N. Horspool

Abstract

The wavelet transform has received much attention in the signal processing and data compression fields. A popular way of achieving image compression is by using the Embedded Zero Tree approach to code the wavelet-transformed data. This is primarily a lossy technique, but has been adapted to lossless compression applications. We propose a new approach to compress the wavelet transformed data, which allows for the exact recovery of the original image. The approach is based on exploiting the characteristics of the wavelet coefficients through the careful construction of a model. The lossless compression ratios obtained with our approach surpass several of the lossless compressors applied to image files in the majority of cases.

Examiners:

[redacted]

Dr. R. N. Horspool, Supervisor (Department of Computer Science)

[redacted]

Dr. M. Serra, Departmental Member (Department of Computer Science)

[redacted]

Dr. R. Illner, Outside Member (Department of Mathematics and Statistics)

[redacted]

Dr. M. L. Lesperance, External Examiner (Department of Mathematics and Statistics)

Table of Contents

List of Tables

List of Figures

Acknowledgments

The research and final thesis could not have been completed without the guidance and wisdom of Dr. N. Horspool. His ability to suggest a solution when one didn't seem possible has helped me greatly to finish this work.

As always, I would like to thank my parents for their never-ending support and encouragement. Their advice, and positive outlook helped me greatly in numerous occasions throughout the last two years of doing research and writing.

The above applies equally to Ben. I also thank him for putting up with my ramblings, mood swings, and long absences all of which seem to inevitably accompany a thesis.

1 Introduction

All information stored, manipulated and transmitted by a computer is represented as bit strings. Bit strings may be used to encode any information we may need to process, including text, source code, executable code, digitized sound files, digitized image files and moving picture files. These representations of data almost always contain redundancy. If we can find some alternate representation for the data which eliminates the source redundancies, we achieve data compression. In general, data compression techniques consist of taking a stream of symbols and translating them into a sequence of codes. These codes are usually not readable as ASCII text; we must recover the source by applying an appropriate decompression program.

The field of data compression may be divided into two major families: lossless and lossy. Lossless compression techniques allow exact recovery of the original source file. They are used for text, executable code, and in applications requiring exact recovery of the input file. Lossy techniques recreate a file which is an approximation to the source. They are used to compress images, sound files and moving picture files, if the application tolerates approximate recovery of the original. If the compression program is a good one, an observer can not distinguish between the original file and one that has been compressed and then decompressed using a lossy method. The advantage is that lossy compression techniques usually boast much higher compression ratios then lossless ones (up to 100:1 [17]).

In this thesis, we will be dealing with images. Image files typically require a great deal of storage space. A relatively small grey scale image file (256 rows by 256 columns) takes 65 356 bytes of storage space. Many applications require hundreds of images, and so use huge amounts of storage. As a result, there has been an explosion of research into graphics storage during the last decade [13].

Conventional data compression techniques do not tend to work very well on images [13]. The main reason for this is that image files tend to have their pixel values spread out over the entire range of possible values. As a result, there appears to be less redundant information in the file, making it more difficult to compress. Several different approaches have been proposed for image compression, including predictive coding methods such as PMC, transform coding methods such as JPEG's DCT based coder, and other methods such as Vector Quantization. Many of the approaches discussed in the literature are lossy methods, since the compression ratios achievable with them are potentially much higher than lossless ones.

Lossy compression techniques typically use a transform approach. The input source file is filtered using some kind of mathematical transform. The role of the transform is to remove inter-pixel correlations by packing them into as few coefficients as possible. The transform coefficients are then coded using some hybrid approach. If the energy-packing ability of the transform is reasonable, it is possible to transmit only the transform coefficients with the largest magnitudes (and their positions) to the compressed file, and still recover a reasonable approximation of the original image. In recent years, the Zero-Tree approach applied to wavelet transformed data has proved to be very effective technique for lossy image compression [14], [17], [22]. More will be said on wavelet transforms shortly.

One of the problems associated with many of the traditional transforms, such as the Fourier transform, is that they are well suited for images which are very "regular", but do not perform well with more anomalous ones. The bases of Fourier transforms are sine and cosine functions; as a result, they do a very poor job approximating choppy signals. Wavelets give us a basis which is more suited to a wide range of signals.

As already pointed out, a wavelet transform has often been followed by the Embedded Zero Tree (EZW) coding method to compress image files. We would like to take a different approach to compressing the wavelet coefficients, one that exploits certain characteristics and correlations present in the wavelet transformed data.

For our experiments, we were given a series of ultrasound medical images that were being compressed using the TIFF lossless compression scheme. TIFF embodies Lempel-Ziv-Welch (LZW) coding, which is a dictionary based method that is not particularly well suited for images. However, it does perform quite well on very smooth images; a category many of the ultrasound files fall under.

The main goal of this thesis was to develop and implement a lossless wavelet based compression/decompression coder that performed on a par (or better) with the compression scheme that was currently being used with the files (namely TIFF). As already pointed out, LZW compression is not always particularly suitable with image files. For this reason we also chose to try and match (or beat) the results obtained using the SPIHT algorithm, which was developed specifically for lossless image compression [14].

The result of the thesis is a new lossless compression algorithm, which is based on the wavelet transform.

2 Data Compression Overview

Data compression consists of taking a stream of symbols and transforming them into codes [13]. Both symbols and codes are bit strings. If these codes require fewer bits than the original symbols, then compression is achieved.

Data compression is inextricably tied up with prediction [2]. If one could predict exactly what appears next in a stream of symbols, there would be no need to transmit the actual stream, and perfect compression would be achieved. This is obviously not realistic; however, even approximate predictions will usually lead to some compression. One could achieve this compression by assigning shorter codes to symbols with high prediction values and longer codes to symbol with low prediction values. For example, it is known that on average, the letter 'e' in English text has a much higher rate of occurrence than other letters. Therefore, one could exploit this fact by assigning a shorter code to this letter.

2.1 Fundamentals

Information stored on a computer is quantified in terms of bits, bytes, kilobytes and so on. However, these notions are not a measure of the information content of the data stored; otherwise data compression would not be possible. Some data may contain large amounts of redundant information. Any such situation provides an opportunity for compression - if the redundant data can be represented in some other form, then the size of the original message is reduced.

A stream of symbols stored in a file may be referred to as the *message*.

Compression ratio C_r refers to the amount of compression achieved and is defined as follows:

$$C_r = \frac{m_1}{m_2}$$

where m_2 is the size of the original message and m_1 is the size of the compressed message.

The relative redundancy R_d of a message can be defined as follows [7]:

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

where C_r is the compression ratio.

2.2 Measuring Information

The term entropy is used to refer to a measure of how much information is contained in a message, and so consequently, a measure of the number of bits needed to represent that message [13]. In other terms, it can be viewed as the amount of redundancy in the message. The higher the entropy, the more information is contained in the message and the more difficult it is to compress it.

More formally defined, suppose we have some symbol that has a probability P_i of occurring. According to Shannon [2], its entropy or information content is defined as the negative of the logarithm of its probability:

$$E_i = -\log_2 P_i \qquad [1].$$

This means that more likely symbols (ones with higher probabilities) will have lower entropy values.

The definition of entropy in equation [1] corresponds to the information content of an individual event. We can use this to define the information content of a decision between events. Suppose we have n symbols, each of which have associated probabilities (p_1, p_2,... p_n) of occurring and respective entropy values (E_1, E_2,... E_n). Given an n-way decision between these symbols, the average entropy of the individual decision is [2]:

$$E = \sum_i p_i E_i \qquad [2].$$

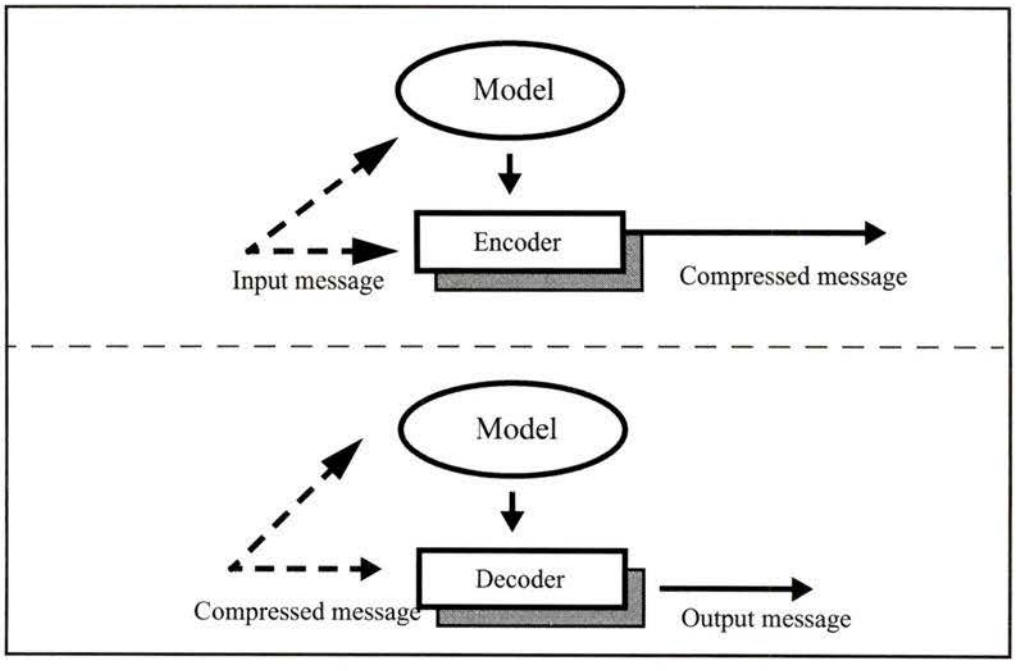
If we plug in the above formula for the individual entropies E_i, we obtain the overall entropy:

$$E = -\sum_i p_i \log_2 p_i \qquad [3].$$

2.3 Compression Systems and Models

A general compression system consists of an encoder, a decoder, and a model, as shown in Figure 2.1. The encoder takes the input message and creates the compressed file by generating new symbol codes based on predictions made by the model. The decoder takes that compressed file and recreates the original message, using the same model to interpret the codes. The model used by the decompressor must be identical to the one used by the compressor in order to recover the original message. Every compression/decompression scheme may be cast into this model / encoder, model / decoder form.

Figure 2.1 Compression System



At the beginning of this chapter we observed that compression is tied up with prediction, and that it is achieved through exploiting redundancies. In order to compress data well, we need a good model of the data being compressed, one that will capture its notable features. The fundamental role of the model is to supply probabilities for the symbols [2]. It makes its predictions by exploiting the redundancies in the input data. Accurate predictions allow the encoder to assign shorter codes to symbols that occur more frequently, and compression is achieved. Models may be classified as static, semi-adaptive and adaptive.

2.3.1 Static Models

Static modeling consists of building a model based on some statistical information gathered from the type of data we expect to be compressing. The model is created once, and the encoder uses this model to compress all files. Since this model never changes, it may be transmitted once and used by the decoder to decompress all files sent by the encoder in the future, or alternatively, the model may be built into the programs responsible for compressing and decompressing the data. The major disadvantage of this type of model is that its performance is very dependent on the statistical similarity of the file being compressed to the type of files that were used to create the model.

2.3.2 Semi-Adaptive Models

Semi-adaptive schemes build a model based only on the file that is currently being compressed. In order for the decoder to decompress the data, the model must be transmitted along with the compressed data to the decoder. The compression achieved by such a scheme may be better than with a static model since the model is more specific to the file being compressed. Unfortunately, the transmission of the model often adds a significant component to the compressed file and so some of the advantage is cancelled out.

2.3.3 Adaptive Models

Adaptive modelling schemes never explicitly transmit the actual model along with the compressed data. Instead, both the coder and the decoder will start off with an initial model. As the data is being compressed, the coder will update the model after each symbol is added to the compressed file. The updating is necessary so that the model will predict symbols according to their frequency of occurrence in the message thus far. The decoder will extract each symbol and then add it to its model. In this manner, the model is continually being adapted to the data within the file to try and suit it better.

For example, suppose we wanted to encode the very simple message "aii" using an adaptive scheme. Our coder starts off with an empty model. It tries to code the symbol 'a' using the

model, and obviously does not find it within the model. There are several options available which may be used to encode the symbol in this case; they are explained later. For now we assume that we somehow encoded the symbol. We add it to the model, along with some non-zero probability associated with it. We now try to encode the symbol 'i', and again find that it is not present in the model. We again encode it using some alternate method, and add it to the model; it now contains the symbols 'a' and 'i', each with an associated probability. Finally, we encode the last symbol in the message, 'i'. We find that it does occur in the model, and so we use the associated probability to encode it. We then increment the probability of the symbol 'i' found in the model to reflect that it has occurred. In this way, the model is continually being updated and so adapted to the input data. The decoder starts off with the same empty model. As it decodes each new symbol, it updates the model in the same way as the encoder.

2.3.3.1 The Zero Frequency Problem

A problem arises with using an adaptive model when a symbol occurs in the input message which is currently not predicted by the model. For example, suppose the model makes predictions for the symbols 'a' and 'b', but the symbol 'c' actually occurs. Since the model does not currently predict the symbol 'c', we have no way of encoding it.

One way of overcoming this is to initialize the model to contain all of the possible symbols in the input alphabet, and assign each symbol some non-zero probability. Thus, all of the symbols are initially equally likely to occur; as symbols are coded, their probabilities are incremented by the model to gradually reflect the data in the file. The problem is "start up" time [2]. The symbols which have occurred in the file and thus have higher probabilities are outweighed by the other symbols which have not yet occurred but to which we were forced to assign a fixed probability. If the input alphabet is large, this may cause significant loss of compression ability.

A second solution is to use a second alphabet (often referred to as the *escape alphabet*), which is initialized to predict all symbols in the input message. Prior to beginning compression, the model is initialized to predict one symbol, referred to as the *escape* symbol. This symbol must be one that never occurs in the actual input file. Whenever the symbol we are currently trying to encode is not predicted by the model, the encoder generates the escape symbol to let the decoder know that the symbol could not be predicted, and then codes the symbol using the prediction generated by the escape alphabet. The symbol is then added to the model and assigned some non-zero probability.

The question that naturally arises is which type of model is the best. It is not possible to say that adaptive models will always achieve better compression ratios than static or semi-adaptive ones. However, it is known that adaptive models can only do slightly worse than the best non-adaptive ones [2].

2.4 Encoding Probabilities

It has been stated several times that the encoder will translate the prediction probabilities generated by the model into codes that will be transmitted to the compressed file. But how do we translate these probabilities to the bits contained in the compressed file? The problem may be stated in another way: given a stream of symbols with associated probabilities and the symbol that actually occurs, how do we translate this information into a code that can later be interpreted by the decoder? There are many methods for encoding with respect to a given model. Two of the better known ones are referred to as Huffman coding and Arithmetic coding.

2.4.1 Huffman Coding

Huffman coding assigns shorter codes to the symbols of the input file that have a higher probability of occurring. These codes are characterized by the unique prefix property, which makes them decodable in a left-to-right scan. Huffman codes may be represented a binary tree. The symbols make up the leaves of the tree; the unique code for each symbol is found by following the path from the root of the tree to a leaf associated with that symbol. The procedure can be elegantly described in five steps:

1) Initialization: a list containing all the symbols is created; each symbol has an associated weight, which is simply the probability assigned to it by the model. These symbols will make up the leaf nodes of the tree.

2) The two symbol nodes with the lowest weights are located.

3) A parent node is created for the two nodes, and assigned a weight equal to the sum of its two children nodes.

4) The two children nodes are removed from the list, and the newly created parent node is added.

5) The procedure is repeated at step 2 until only one node remains in the list; this is the designated root of the tree.

Symbol codes can now easily be created by following a path from the root of the tree to each of the leaves; if the path follows a right branch from a parent node to a child node, a 1 bit

is appended to the code, otherwise a 0 bit is appended to the code. (This is a matter of convention.)

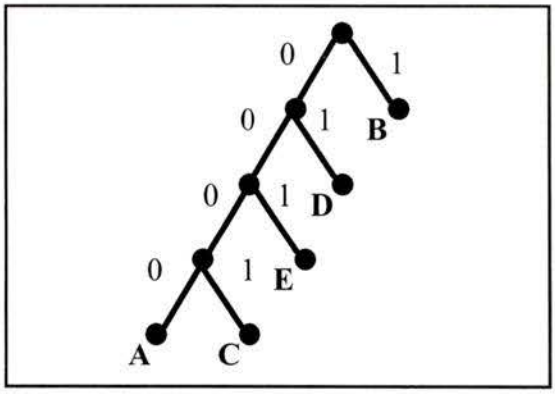
Example:

Suppose we have a list of symbols and their associated frequencies counts (we may translate a frequency count into our estimate of its probability simply by dividing it by the total frequency count), shown below.

A	B	C	D	E
5	20	2	15	6

The two symbols with the lowest weights are located; in this example, the symbols *C* and *A*. A new node is created which is the parent node for these two symbols, and assigned a weight equaling the sum of the weights of its two children: 7. The two symbol nodes are removed from the list and the newly created parent node is added. We proceed by locating the next two nodes with the smallest probabilities: the symbol node E and the node created in the last step, and add this new node to the list, removing its two children from the list. The process continues until only one node is left, generating the tree shown in Figure 2.2.

Figure 2.2 Huffman Tree



Below are listed the codes generated for each of the symbols using our tree of Figure 2.2.

A	B	C	D	E
0000	1	0001	01	001

The example demonstrates that shorter codes are assigned to symbols with higher frequencies.

One of the shortcomings of Huffman coding is that it is only optimal if all symbol probabilities are powers of $\frac{1}{2}$. This is because Huffman codes have to be an integral number of bits long.

The complexity of a static coding scheme (i.e. the Huffman tree is built before any compression takes place and is not updated during the compression stage) is $O(n \log n)$, where n is the number of symbols to be compressed.

2.4.2 Arithmetic Coding

Conceptually, arithmetic coding represents the entire input stream by an interval of real numbers between 0 and 1 [2]. The longer the message, the smaller the interval needed to represent it, and the greater the number of bits needed to specify this interval. This concept was known theoretically for quite some time but it was only recently that a practical implementation was suggested.

In order to construct the code, probabilities must first be assigned by the model to the input symbols. Once we know what the symbol probabilities are, we need to assign each symbol a portion of the interval that is reflective of the symbol probability. In other words, we will assign greater portions of the interval to symbols with higher probabilities and smaller portions to symbols with lower probabilities. For ease of explanation, a simple example will accompany the description of the algorithm.

Suppose we have the following probabilities,

Symbol	Probability
a	2/5
i	2/5
e	1/5

and the message "iaiea". The ranges that the interval would initially be divided into is shown below.

Symbol	Probability	Range
a	2/5	(0.0, 0.4)
i	2/5	[0.4, 0.8)
e	1/5	[0.8, 1.0]

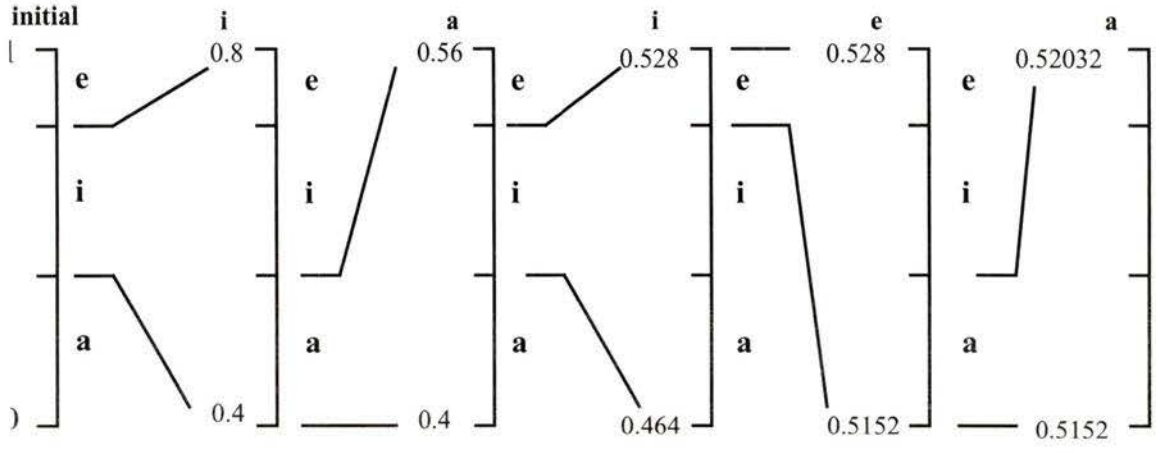
The probability of occurrence determines the sub-range of the 0 to 1 interval assigned to the symbol. The order in which symbols are assigned portions of the interval is irrelevant as long as the same order is used by coder and the decoder. Now the actual encoding process can start. The first symbol in the message is 'i'. The encoder narrows the interval to [0.4, 0.8), from the previous [0,1) which is exactly the range allocated to the symbol by the model. The sub-range [0.4, 0.8) is sub-divided into three sub-intervals; [0.4, 0.56), [0.56, 0.72) and [0.72, 0.8) according to the probabilities of the three symbols. The encoding of the second symbol 'a' will then further narrow the previous range [0.4, 0.8) to produce a new interval that is now [0.4, 0.56). This process continues until the entire message has been processed, as shown in Table 2.1.

Table 2.1 : Range Sub-Division

		Range
Initially		[0, 1)
After seeing:	i	[0.4, 0.8)
	a	[0.4, 0.56)
	i	[0.464, 0.528)
	e	[0.5152, 0.528)
	a	[0.5152, 0.52032)

A perhaps more intuitive way of viewing this process is shown in Figure 2.3. Symbol probabilities are represented by the vertical bars. The first symbol 'i' is processed and the range is scaled into [0.4, 0.8). For ease of presentation, the figure shows the sub-ranges expanded to full height at every stage, as opposed to showing the entire 0 to 1 range.

Figure 2.3 The Arithmetic Coding Process



For this example, the decoder receives the final range interval, [0.5152,0.52032). It sets up the initial range intervals based on symbol probabilities given to it by the model (it must have the same model as the encoder). From the final interval, it can immediately decode the

first symbol 'i', since the sub-range for 'i' [0.4, 0.8) entirely encloses it. The decoder then simulates the action of the encoder, by breaking up its range intervals in the same manner. The initial interval was between 0 and 1, it now becomes [0.4, 0.8). The decoder can then decode the second symbol 'a' since this produces the new range [0.4, 0.56) which entirely encloses the given interval [0.5152,0.52032). None of the other symbols would produce a new interval which would completely cover this interval. The decoder proceeds like this until all of the symbols have been decoded.

The algorithm presented above is overly simplistic in several respects. The most obvious problem that comes to mind is that of floating point arithmetic. Most computers cannot store floating point numbers with more than 80 bits for the mantissa. In our example, the encoder does not transmit anything to the decoder until the entire message has been coded, which means that a great deal of precision is required. This seems to render the algorithm useless since it could only encode the shortest of messages. In reality, floating point arithmetic is replaced with integer arithmetic. To further solve the precision problem, the algorithm above is replaced with an incremental one. In the new version of the algorithm, bits are sent to the encoder as soon as the most significant bits of the low and high ends of the range are equal. In the previous example (Figure 2.3), after the fourth symbol 'e' has been coded, the range is [0.5152, 0.528). Since the final code must fall within this range, we know it begins with "0.5...", and the most significant bit of the binary fraction may be transmitted.

Unlike Huffman coding, Arithmetic coding does not require that each source symbol be translated into an integral number of code symbols. Arithmetic coding generates the same number of bits as the entropy of the message with respect to the model used. However, the use of finite precision arithmetic may hinder its performance [7].

Arithmetic coding is easily combined with an adaptive model, since the probabilities may be changed at each coding / decoding step. In principle, Huffman coding may also be adaptive. We start off with an empty Huffman tree, and at each step modify it to account for new symbols being encoded. However, repeatedly modifying the tree is computationally intensive.

In our work, we chose to use Arithmetic Coding, since as already pointed out, it is an optimal scheme.

A general background for data compression has been presented. We will now move on to the topic of image compression, since that is the type of compression relevant to the work

in this thesis. In the following chapter, various image compression techniques will be introduced.

3 Digital Images

The second chapter provided some fundamental background for data compression. Since we are interested in compressing images, it is necessary to explore the characteristics of digital images, and some of the existing compression techniques. The compression techniques used with images tend to differ greatly from the ones used for text compression; in fact, conventional data compression techniques typically do not do well with images [13]. The reason is that conventional techniques rely on the input file to have some symbols (i.e. pixel values) occur much more frequently than others. Images typically have their symbols more spread out over the entire range.

From this point on, we will only deal with gray-scale images.

A two-dimensional image may be treated as a light intensity function *f(x,y)*, where *x* and *y* refer to the spatial coordinates within the image, and the value of the function at this position corresponds to the brightness. When we are dealing with black-and-white images in which there can only be different shades of gray, this brightness is referred to as the *gray level* of the image; the image files we work with have 256 gray levels associated with them. Another way of thinking about digital images is as of matrices whose row and column indices identify a point in the image, and the matrix value at that row and column position identify the gray level [7]. This matrix value is commonly referred to as a *pixel* value.

3.1 Image Mean, Variance and Energy

We will begin by introducing several terms.

The mean of an image is defined as:

$$\bar{x} = \frac{1}{N \times M} \sum_{n=1}^{N} \sum_{m=1}^{M} x_{nm}$$

where x_{nm} is some pixel value, and *N* and *M* are the numbers of image rows and columns, respectively (where x_{nm}, and *f(x, y)* are equivalent).

The variance of an image is then defined as:

$$\vartheta^2 = \frac{1}{N \times M} \sum_{n=1}^{N} \sum_{m=1}^{M} (x_{nm} - \bar{x})^2$$

The variance and the mean of an image do give us a little information about the image. To a certain degree, the variance tells us how much the image pixels vary in value. There is also somewhat of a connection between the two. For example, if the mean of the image is around the middle of the range of possible image values, we may expect the variance to be quite high [3]. Alternately, if the mean is near the upper or lower limit of the possible range, it is not as likely for large positive differences between element values and the mean to exist, and so the variance is typically lower [3].

In order to compress anything well, a good understanding of the properties of the type of data we are trying to compress is needed. As already mentioned, the removal of redundancy results in data compression. Therefore, we will begin by describing the types of redundancies present in digital images.

3.2 Redundancy

In digital image compression, three basic data redundancies can be identified and exploited: coding redundancy, inter-pixel redundancy and psychovisual redundancy [7].

3.2.1 Coding Redundancy

Image pixels are stored on a computer using some fixed number of bits. For example, gray-scale images are typically represented by eight bits per pixel. However, this coding scheme does not take into account the relative probabilities of the various pixels. A pixel is represented by a fixed number of bits regardless of whether it occurs once or three hundred times in the message. Most representations of images will contain at least some redundancy since it is generally the case that some pixels are more probable than others.

Example:

Suppose we have an input signal = [3 2 2 1 0]. If we represented each value in the signal in a fixed field, say 3 bits, the total number of bits for the signal would be 15.

To calculate zero-order entropy, we use the Equation 3 from Section 2.2, and assign a fixed probability to each symbol in the set of input symbols (which is {3, 2, 1, 0}).

We assign the symbol 3 the probability $\frac{1}{5}$ (since it appears once), the symbol 2 the probability of $\frac{2}{5}$ (since it appears twice), the symbol 1 the probability $\frac{1}{5}$ (since it appears once), and the symbol 0 the probability $\frac{1}{5}$ (since it appears once).

This gives us:

$$\left(-\frac{1}{5}\right)\log_2 \frac{1}{5} + \left(-\frac{2}{5}\right)\log_2 \frac{2(1)}{5} + \left(-\frac{1}{5}\right)\log_2 \frac{1}{5} + \left(-\frac{1}{5}\right)\log_2 \frac{1}{5} = 1.92$$

bits per signal; for a total of 9.6 bits for the entire signal.

As demonstrated by the example, assigning fewer bits to pixels that occur more frequently (i.e have higher probabilities) and more bits to less probable pixels removes at least some of the redundancy in the original message, and thus achieves compression. This process is referred to as variable length coding. (Two variable length coding techniques have already been mentioned: Huffman and Arithmetic coding.) Whenever the various frequency levels are not equally probable, variable length coding can be used to achieve at least some compression.

3.2.2 Inter-Pixel Redundancy

Simply assigning shorter codes to pixels with a higher frequency count does not exploit inter-pixel redundancy. There is a high probability that the neighbors of a pixel will have values similar to the value of the current pixel [7]. Thus, a pixel's value can often be predicted from the values of the surrounding neighbor pixels. There are many ways that the inter-pixel redundancies can be exploited. Typically, the message is transformed into a more efficient format that is non-visual.

Example:

One simple way to exploit inter-pixel redundancies is by coding the differences between adjacent pixels instead of the pixels themselves. Suppose we again have the one dimensional signal = [3 2 2 1 0]. The zero-order entropy of this message was calculated above to be 1.92. Now if we code only the differences between adjacent pixels we get the following message: [3 1 0 1 1] (the first difference is simply the first symbol in the original message). The entropy of this new message is 1.37, a small improvement over the original.

3.2.3 Psychovisual Redundancy

Our eyes are more sensitive to certain visual information; the less important information which is not perceived by our visual system is referred to as psychovisually redundant. For example, we notice changes in edges and sharp lines more than minute changes in pixel intensities. The psychovisually redundant information may be removed, often without damaging the perceived quality of the image.

3.3 Lossless versus Lossy compression

Lossless data compression (also referred to as exact, or reversible compression) means that the reconstructed data is identical to the original data. Some applications require this type of compression; for example, text, and images used by the medical community.

Lossy compression introduces some degree of loss of the original information. The reconstructed signal is an approximation of the original. However, if one is trying to compress speech or certain types of image data, this loss may be imperceptible to the human ear or eye and is quite acceptable. One of the goals in the development of lossy compression techniques is to determine the point or threshold where the distortion introduced by the compression stage is not high enough to seriously damage the quality of the reconstructed signal, while still achieving good compression ratios.

The type of distortion errors introduced by lossy compression may be classified to be one of five types: blockiness, jaggedness, blurring, ghost figures, translation distortion, and quantization errors [9]. Blockiness refers to artificial edges (resembling blocks) that appear in the decompressed image. Jaggedness in the decompressed image means that the edges are fragmented. Blurring occurs when the sharp edges in the original image become less defined in the decompressed one. Ghost figures are structures that arise near sharp edges. Translation distortion occurs when objects in the image are shifted from their original positions. Finally, quantization means that the number of grey levels has been reduced, leading to loss of overall "smoothness" of the image.

3.3.1 Fidelity Criteria

The key issue in lossy compression is how to measure the distortion caused by the compression [9]. There are two categories of techniques used to determine this loss: subjective and objective ones. When the level of information loss can be expressed as a function of the original and reconstructed image (i.e. compressed and subsequently decompressed image), it is said to be based on an objective fidelity criterion [7]. There are

several such techniques currently being used, including the *rms* error and *psnr* error between an input and output image.

Suppose we have an input image represented by the function *f(x, y)* and an output image (a lossy version of the input) represented by a function *g(x,y)*. The rms error e_{rms} between the two images is defined as follows:

$$e_{rms} = \left[\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [g(x,y) - f(x,y)]^2 \right]^{1/2} .$$

Another objective fidelity criterion is the peak signal to noise ratio (*psnr*), defined as follows:

$$PSNR = 10 \times \log_{10} \left(\frac{MAX^2}{e_{rms}^2} \right)$$

where MAX is the upper range of pixels values. For example, if our pixels are represented by 8 bits, MAX = 255.

These measures are convenient because they allow the quality of the lossy image to be determined without even looking at it. Their major drawback is that the image will eventually be looked at by humans who often have different quality criteria than computers. Consequently, measuring image quality by using the subjective evaluations of a human observer is often more appropriate [7]. For example, a subjective evaluation may include showing the original image and the decompressed one and asking the viewers to judge the quality based on a scale {0 = completely useless, 1, 2, 3 = very annoying errors, 4, 5 = annoying errors, 6, 7 = slightly annoying errors, 8, 9 = very good, 10 = perfect: no annoying errors} [9].

3.4 Image Compression Techniques

We will now present a general overview of image compression techniques, which are separated into the following categories: *predictive*, *transform*, and *other*. Although some of the techniques mentioned are strictly used for lossless compression and some for lossy, there tends to be a great deal of overlap between the two areas. For example, transform techniques are traditionally used for lossy compression, but have been shown to be useful

for lossless compression as well [15], while predictive techniques are applied widely to both types of compression. After generally introducing these three categories, we will describe transform and predictive techniques in some detail, as they are relevant to our work.

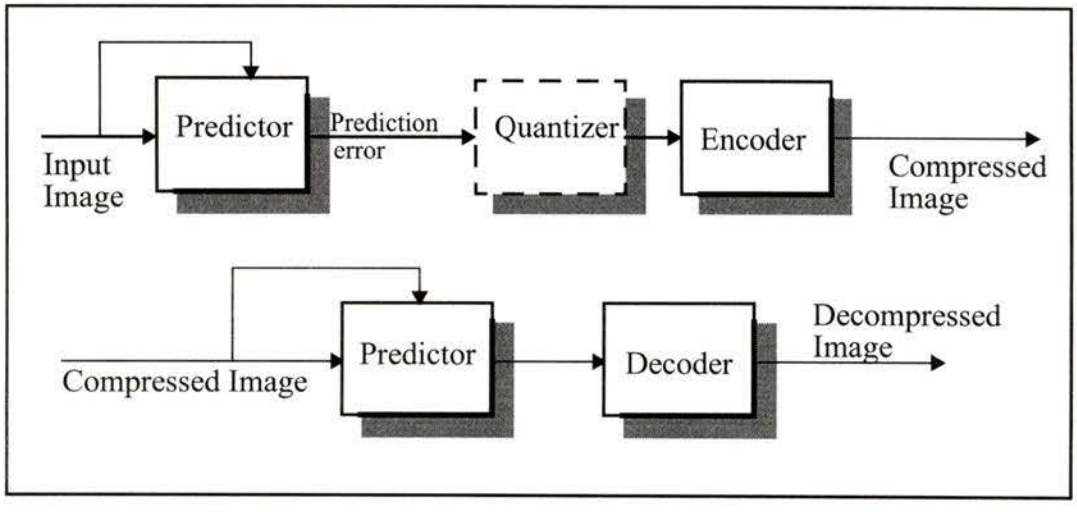
3.4.1 Predictive Techniques

Predictive techniques are based on removing inter-pixel redundancies of neighboring pixels. It is often the case that the current pixel will be very similar to its neighbors, especially if the image is highly correlated. Thus, we can predict with some degree of accuracy the value of the new pixel using the knowledge we have about its neighbors. We code only the difference between the actual value of the pixel and our predicted value.

The predictor generates a prediction for the current pixel based on previously processed neighboring pixels. The difference between the prediction and the actual value is referred to as the *prediction error*. There are two types of predictive coders: lossless and lossy ones. A lossy coder sends the prediction error to a quantizer, which typically rounds the prediction by scaling it. This is the irreversible phase omitted by lossless coders. The encoder takes care of translating the prediction error to bits and transmitting it to the compressed file.

The system may be generally illustrated in Figure 3.1.

Figure 3.1 Predictive Coding System



3.4.2 Transform Coding

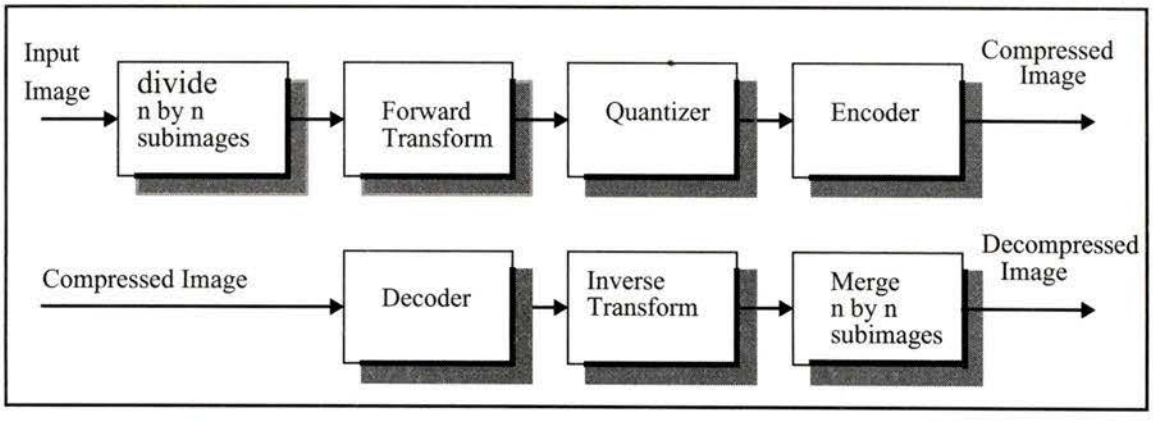
Transform coding techniques are based on applying some mathematical transform to the original input signal. The purpose of the transform is to reduce inter-pixel correlations and pack as much of the image energy into as few of the transform coefficients as possible. Most of the work done with transform coding has been applied to lossy compression for reasons that will soon become clear. However, since the transform operation removes some of the inter-pixel correlations, it is quite applicable to lossless compression techniques.

Four steps are performed. First, in traditional coding techniques, the image is decomposed into *n* by *n* sub-images. This step allows for faster computation of the transform coefficients and has traditionally been necessary for practical implementation. The wavelet transform (Chapter 4) typically does not require this step. Next, the forward transform is applied and, in the case of lossy compression, the transformed coefficients are quantized. A good quantization scheme balances high compression ratios with acceptable image quality. We must omit the quantization step in order to avoid loss of information. Finally, the coefficients are encoded using some variable-length coding scheme.

The decoder recovers the transform coefficients (or the approximate transform coefficients in the case of lossy compression) and applies the inverse transform to recover the original image.

Figure 3.2 shows a typical transform coding system.

Figure 3.2 Transform Coding System



3.4.3 Vector Quantization (VQ)

Theoretically, better data compression performance may be achieved by coding vectors instead of scalars (individual values) [8]. In that sense, even transform coding is sub-optimal, since the quantization stage is applied to scalar values. Vector Quantization considers whole vectors and so theoretically should perform better than transform techniques. In reality, its performance is comparable [3].

VQ begins by segmenting the input image into *n* by *n* blocks, which are treated as vectors containing n^2 elements, and processed. The basic idea behind the algorithm is this: for each such *n* by *n* block, the encoder finds a block that best matches it in a *codebook*. The codebook index where the match was found is then transmitted to the compressed file. The decoder has an identical codebook. It reconstructs the data by replacing an index from the compressed file by the vector found in the codebook at that index position. Obviously, this is a lossy technique, since the vectors found in the codebook are generally not identical to the input vectors.

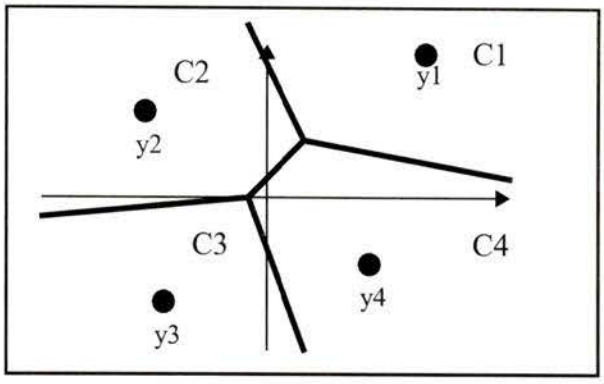
The ideas behind VQ will now be discussed in more detail. In the following discussion, an input block of pixels is referred to as a vector.

3.4.3.1 Approach

Suppose we have an *N*-dimensional vector $\mathbf{x} = [x_1, x_2, x_3, \ldots x_N]$. This vector is mapped onto another *N* dimensional vector **y**. We can say that **x** is quantized to **y**, or that **y** is the reconstruction value of **x**. The set of **y** vectors is referred to as the *codebook* [8].

This technique extends the idea of quantization to more than one dimension. Figure 3.3 shows the partitioning of a two dimensional space for VQ (the principle applies to higher dimensions but is difficult to draw).

Figure 3.3 Partitioning of 2-D Space into Five Cells



The positions of code vectors are depicted as dots in the diagram; here, the total number of these vectors is 4. Any vector that lies within the cell C_i is quantized as $\mathbf{y_i}$. In general, a vector $\mathbf{v_i}$ is quantized to a region C_i if it falls within the boundaries of that region.

The quantization of a vector **x** to a vector **y** introduces a quantization error. Therefore, the coder has to choose a set of vectors **y** such that this distortion is minimized. There are many distortion measures listed in literature [11]. One widely used one is referred to as the *square error distribution measure* [8]. It is simply the square of the Euclidean distance between vectors.

3.4.4 Fractal Compression

Fractal image compression uses sets of transformations to represent an image. The original image is broken down into manageable regions. Each region is covered with a set of transformed versions of itself, and the coefficients of each transformation are coded. These transformations are referred to as IFS codes.

IFS codes approximate a natural image by using affine transformations. These are combinations of rotations, scalings and translations. Fractal compression models are based only on affine transformations which have the property of being contractive - they move the input image points closer together. These affine transformations may be represented by coefficients which are transmitted to the compressed file [1]. Finding affine transformations which represent the input image is a matter of solving for values of the coefficients representing the transformation. IFS functions are then simply collections of contractive affine transformations. Each transformation also has associated with it a probability which determines its importance relative to the other transformations.

Fractal image compression generated much excitement in the field by claiming compression ratios as high as a million to one [1]. There are several factors which have prevented widespread use of this technique. The first is that the method is extremely computationally intensive: some complex color images have been reported to take as many as 100 hours to compress [1]. Another criticism of the technique involves the need to sub-divide the original image into sub-regions (this is done because original images are too complex and make it impossible to find a set of transformations which will represent them accurately). This technique therefore suffers from the same blocking artifact problems as do traditional transform techniques. Another drawback is that the technique only tends to work well on graphically generated scenes and not on naturally occurring ones. More realistically, the compression ratios which are achieved by this method are similar to those of transform techniques.

3.4.5 Bitplane Coding

Bitplane coding is a technique that decomposes a multilevel gray scale or color image into a series of binary images. Each binary image (which consists of 0's and 1's) is compressed separately. In this section we will consider only gray scale images for ease of illustration; the principles may be easily extended to color images. A pixel of an *m*-bit gray-scale image is represented in the computer in the following form:

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + a_{m-3}2^{m-3} + \ldots + a_0 2^0$$

where each a_i is either a 0 or a 1. Using this representation, it is possible to separate the image into *m* bit planes, and obtain *m* binary images. For example, the zeroth order bit plane is generated by collecting the a_0 bits from all the pixels in the original image. Since a binary image consists of only black and white pixels, a pixel in this zeroth order plane is defined to be white if the a_0 bit in the original image is 1, and black if the bit is 0.

Example:

Figure 3.5 show the first, fifth and seventh binary bit planes of the image shown in Figure 3.4, in a left-to-right order.

Figure 3.4 "baboon" Image Fille

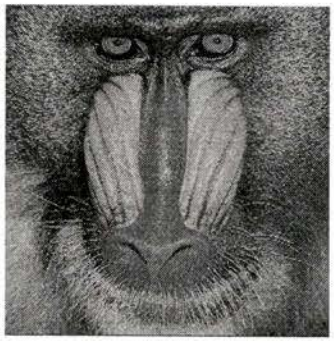


Figure 3.5 The First, Fifth and Seventh Bit Planes of the Image in Figure 3.4

Once the image has been decomposed into bit-planes, each bit-plane must be coded separately using some compression technique. One method is to decompose the bit-plane being coded into blocks of size m by n pixels, and classify these blocks into three categories: all white, all black, or mixed (i.e. black and white). The most frequently occurring category can then be assigned a bit code of length 1, while the other two can be represented by bit codes of length 2. In order to achieve lossless compression, all the bit intensities must be transmitted after sending a mixed code.

Another method that is used to code the bit planes is run-length encoding. This method replaces a sequence of black or white pixels by a count of the length of the sequence. The run-lengths must be prefixed by some code which indicates whether the run is a black or a white one. The number of bits used to represent each run may be determined before hand

and therefore fixed for the entire image. Alternatively, the run-lengths themselves may be variable length coded using some kind of a model; a technique which often improves compression.

3.4.6 Predictive and Transform Techniques in Detail

We will now describe predictive and transform techniques in some detail, as they are relevant to our work.

3.4.6.1 Predictive Techniques

There are two types of predictive coders: lossless and lossy ones. The basic components of a lossless system will be described first.

The function of the predictor is to generate an approximation of the value of the next pixel, based on previously processed neighbouring pixels. The difference e_n between the actual pixel value f_n and the predicted pixel value f^l_n (which is simply the prediction error) is calculated as follows:

$$e_n = f_n - f^l_n \quad .$$

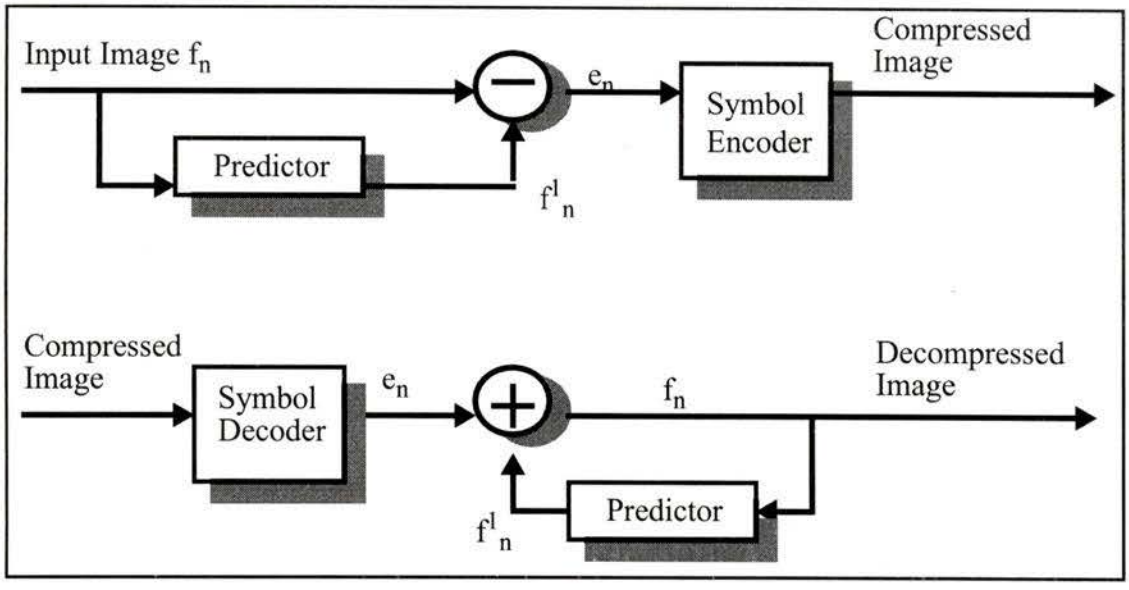
This prediction error is then coded using some type of a variable length coding scheme. The decoder can fully recover the original pixel, since it has the same predictor as the encoder. It makes the same prediction f^l_n as the encoder, based on the current pixel's neighbors, and recovers the original pixel by adding the prediction error e_n (given to it by the encoder) to f^l_n:

$$f_n = e_n + f^l_n \quad .$$

The lossless predictive coding system is shown in Figure 3.6. This type of a model is also referred to as Open-Loop Differential Pulse Code Modulation. These systems are characterized by the predictor predicting the next value based on past history of pixel values [5].

A question that arises is how to start prediction; in other words, how we predict the symbols occurring at the beginning of the file. One solution is to transmit the necessary symbols uncoded to the compressed file.

Figure 3.6 A Lossless Predictive Coding System



Predictors

There are many methods that can be used to generate the predicted value f^l_n. A simple approach is to use a linear combination of some *m* preceding pixels:

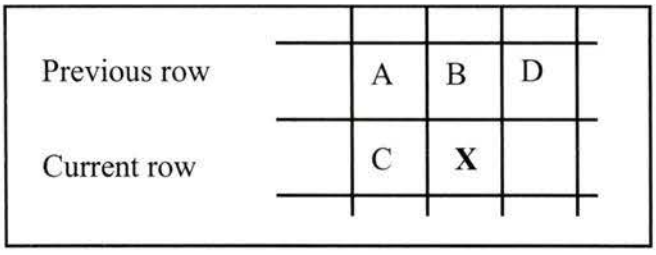
$$f^l_n = \left\lfloor \sum_{i=1}^{m} \alpha_i \cdot f_{n-i} \right\rfloor$$

where α_i is called a predictor coefficient, and f_{n-i} is some preceding pixel value in the input signal. The formula applies to signals of different dimensions [7].

Note that in the case of a 1-D signal, the predictor is a function of some previous pixels on the current line. For a 2-D image, it is a function of some preceding pixels in a left-to-right, top-to-bottom scan of the image.

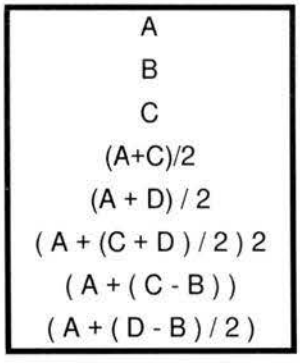
As an example, Figure 3.7 shows the pixel *X* we are currently trying to encode and the four neighboring pixels for a two-dimensional signal.

Figure 3.7 Pixel X and Neighboring Pixels



Several possible combinations of neighboring pixels that are commonly used to make a prediction [13] are listed in Figure 3.8.

Figure 3.8 Combinations of Pixels Used to Generate a Prediction



Example of Lossless Predictive Coding:

Suppose we encode the gray-scale image of Figure 3.4 using a first-order predictor function. The predictor function is defined:

$$f^{1}_{n} = \lfloor \alpha f(x, y-1) \rfloor$$

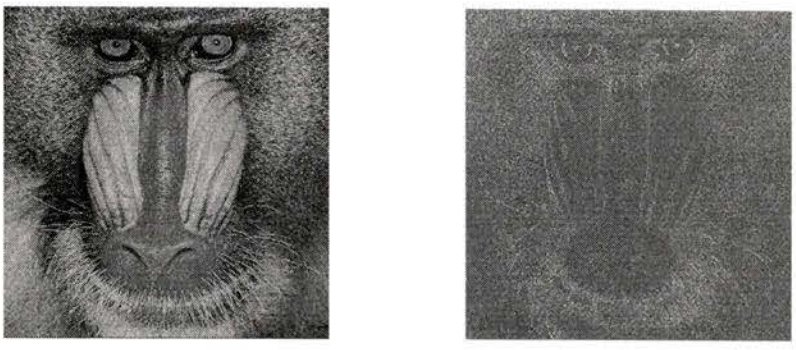
with $\alpha_1 = 1$. This type of predictor is referred to as a previous pixel predictor.

The *error image* is defined as the image consisting of the prediction errors. The more uniformly dark the error image is, the better the predictions generated by the predictor. (The

ideal predictor would always predict the value that actually occurred and so the error image would consist only of 0-value pixels; i.e. it would be black.)

On the left, Figure 3.9 shows original image. The error image generated with our first-order predictor is shown on the right.

Figure 3.9 Original "baboon" and Error Image



The lighter regions in the error image indicate larger prediction errors, which usually result from the presence of edges.

The entropy of the error image is smaller than that of the original image. This drop in entropy value is a consequence of the predictive technique removing some of the redundancy found in the original image.

We are now ready to discuss lossy predictive techniques.

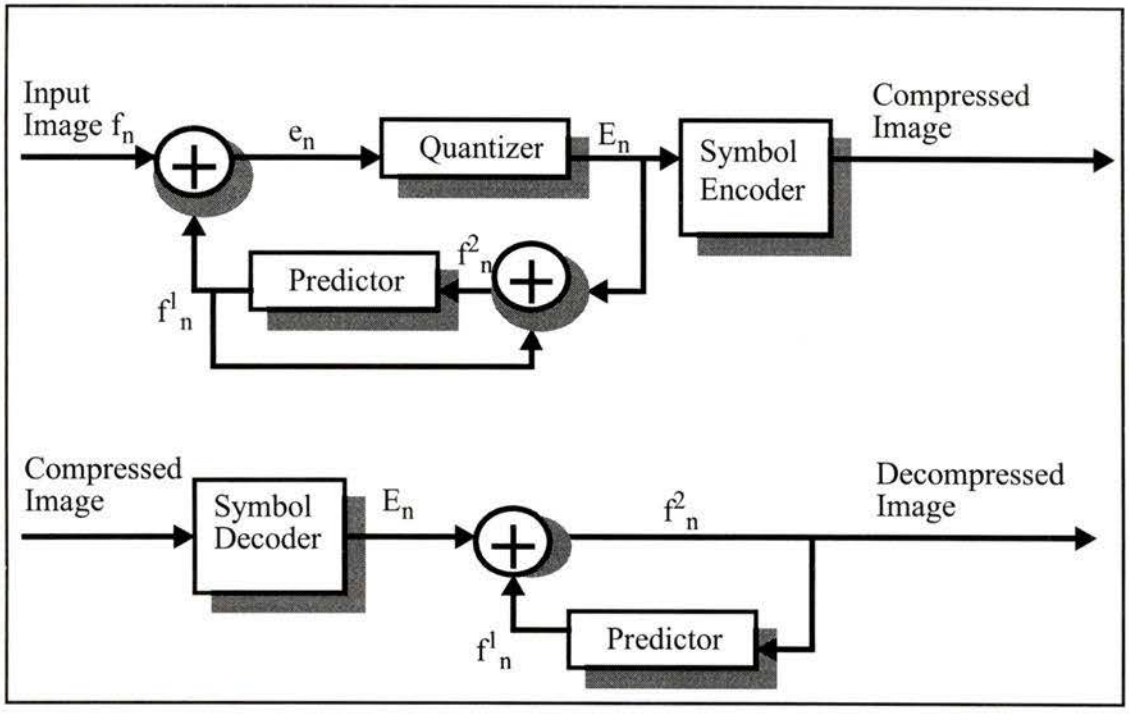
A lossy predictive coding model adds a quantizer to the model discussed in Figure 3.6. It is illustrated in Figure 3.10. This type of a system is also referred to as Differential Pulse Code Modulation (DPMC) [5]. The role of the quantizer in the system is to further reduce the storage needed for the prediction error by mapping it to a limited range of outputs. This step is irreversible and introduces distortion to the decompressed image.

In order for the encoder and decoder to be synchronized, the encoder's predictor must now be altered to make its output value be based on a sum of past predictions and corresponding quantized errors as follows:

$$f^2_n = E_n + f^1_n$$

where f^2_n is the new prediction function and E_n is the output of the quantizer - see Figure 3.10. Note that the decoder does not have a quantizer. We will not go into further detail describing this method since we are primarily interested in lossless compression techniques.

Figure 3.10 Lossy Predictive Coding System



3.4.6.2 Transform Techniques

The transform operation will now be described for the general one dimensional case. The coefficients resulting from the convolution of the basis vectors with the input vector are referred to as *transform coefficients*. The operation is a successive multiplication of the input vector elements by a set of basis vectors. Suppose we have the input vector with four elements [3]:

$$f^T = [f_1\ f_2\ f_3\ f_4] \quad .$$

The vector is written as f^T to indicate it has been transposed since convention dictates that all vectors be written as column vectors. If we define the first basis vector as:

$$t_1{}^T = [t_{11}\ t_{12}\ t_{13}\ t_{14}] \quad .$$

then we can obtain the first transform coefficient C_1 by using matrix multiplication of the basis vector with the input vector:

$$C_1 = f_1t_{11} + f_2t_{12} + f_3t_{13} + f_4t_{14} \quad .$$

Similarly, if we define the second basis vector $t_2{}^T$ as:

$$t_2{}^T = [t_{21}\ t_{22}\ t_{23}\ t_{24}] \quad .$$

then the second transform coefficient is calculated:

$$C_2 = f_1t_{21} + f_2t_{22} + f_3t_{23} + f_4t_{24} \quad .$$

The operation for an input vector of length *N* may therefore be summarized as:

$$C_p = \sum_{k=1}^{N} f_k t_{pk} \qquad p = 1, 2, \ \ldots \ N.$$

The calculation may be expressed in matrix form as:

$$C = [T]f \quad .$$

where **C** is the vector containing transform coefficients, and **f** is the vector containing data elements and **T** is the matrix containing the individual basis vectors in row format. For example, in the case where N=4, the transform operation is expressed in matrix form as:

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

Using this definition, for an input signal of size N, the practical implementation requires N^2 multiplications and $N(N-1)$ additions. If N is sufficiently large, the computational cost of this is very high. This is the reason behind the image subdivision into n by n blocks mentioned at the beginning of this section. As already mentioned, this step is usually not required for a wavelet transform (see Chapter 4).

Example:

Suppose we have an input vector = [5 6 4 8] and the 4 by 4 Walsh-Hadamard matrix of basis vectors:

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 5 \\ 6 \\ 4 \\ 8 \end{bmatrix}$$

The $\frac{1}{2}$ term in the equation is referred to as the weighing coefficient of the basis function. We get the following vector of transform coefficients [3]:

$$C^T = [11.5 \quad -0.5 \quad 1.5 \quad -2.5] \quad .$$

Notice that only one of the transformed coefficients is large. If we choose to transmit only the two dominant (highest magnitude) coefficients, and their positions, we can recover an approximate input signal which would be [4.5 7.0 4.5 7.0]. We do so by convolving the transpose of the Walsh-Hadamard matrix with the vector [11.5 0 0 -2.5]. In the case of lossy compression, a fraction of the original coefficients may often be transmitted and high compression ratios achieved. Even in the case of lossless compression, the transform operation is desirable, since the transform removes inter-pixel correlations.

The Transform Operation in Two Dimensions

Any input signal representing an image may be represented as a two dimensional matrix. The principles of the one dimensional transform may be easily applied to the two dimensional case: the transform matrix is applied to each row of the input signal matrix, and then applied to each column of the signal matrix. This reduces data redundancy in both the horizontal and vertical direction.

Types of Transforms

In order to achieve maximum compression, we should choose a transform that can pack the most information into the least number of coefficients. From this perspective, the optimal transform is the Karhunen-Loeve (KL) transform; however, it is computationally expensive and thus is not used often in practise [3]. The Discrete Cosine Transform (DCT) provides a good compromise between information packing ability and speed, and has traditionally been frequently used [3]. Recently, the wavelet transform (see Chapter 4) has gained much attention, due to its ability to realize extremely high, but lossy, compression ratios with an appropriate coefficient coding algorithm.

3.5 Existing Standards

3.5.1 The JPEG Standard

JPEG (Joint Photographic Experts Group) was an international effort that worked toward a digital compression standard for color and grayscale still images [21]. Such a standard facilitates the exchange of images across different applications, an aspect that is becoming increasingly important as the number of image-dependent applications increases. The JPEG group had several requirements:

1) To meet current state of the art system compression ratios; also, in the case of lossy compression to meet visual fidelity accompanying those compression ratios, while giving the user the option of setting the compression/image quality trade-off.

2) That the compression be applicable to all digital images.

3) That the system be computationally practical in terms of software implementations

4) That the system have several modes of operation:

- sequential coding (a left to right, top to bottom scan);
- progressive encoding, where the image is successively build up in several coarse-to clear passes;
- a lossless encoding, guaranteeing exact recovery of the original image;
- Hierarchical encoding, which allows the lower-resolution image to be accessed before the entire image is decoded.

The four modes of operation can be seen as a 'generic toolkit', from which the user can choose the one most suitable to the application being used.

3.5.2 The TIFF Standard

Tiff is a file format that gives the user the option of compressing the file being processed. The compression scheme used is a dictionary-based one: LZW. LZW belongs to the LZ78 subset of the Ziv-Lempel family of algorithms [2]. These were first introduced in the late 70's; many variations have sprung up since then.

LZW maintains a dictionary of sub-strings which have occurred in the data thus far. The dictionary is first pre-loaded to contain the set of all possible strings of length one. We also initialize a special string, referred to as the current string, to be the empty string. As each symbol is read in from the input file, it is appended to the current string. We then try to match the current string to a string in the dictionary. As long as the current string has a matching string somewhere in the dictionary, we read in new symbols from the input file and append them to it. At the point where the current string does not have a corresponding string in the dictionary, we output the dictionary index where the last matching string was found. The current string, consisting of this matching string plus one new symbol, is then added to the dictionary at the next available slot. Following this, the current string is then initialized to the last symbol read from the input file. The idea is to build up longer and longer strings in the dictionary, so that we can replace longer input strings with a single dictionary index [2].

The scheme is a lossless one. It is not optimal in the sense that it ignores inter-pixel redundancies in the vertical direction. (Typically, we scan the image in a left-to-right, top-to-bottom scan. This exploits inter-pixel redundancies in the horizontal direction, but not in the vertical.)

Example:

Suppose we have an input string: "bbcbbc". The steps performed by the algorithm are listed below.

Step 1: found "b" at index 98;

Step2: did not find "bb"; output index 98, added string "bb" at index 257;

Step 3: found "b" at index 98;

Step 4: did not find "bc"; output index 98, added string "bc" at index 258;

Step 5: found "c" at index 99;

Step 6: did not find "cb"; output index 99, added string "cb" at index 259;

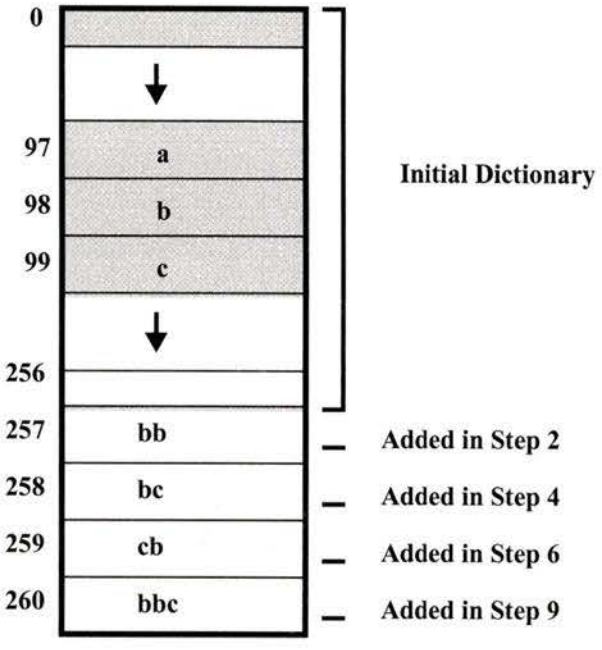
Step 7: found "b" at index 98;

Step 8: found "bb" at index 257;

Step 9: did not find "bbc"; output index 257, added string "bbc" at index 260.

The sequence of dictionary updates is shown in Figure 3.11.

Figure 3.11 The LZW Dictionary



4 Wavelets Overview

Wavelets are a current topic of active research in the signal processing and data compression fields. Ground breaking compression ratios have been achieved using wavelets as part of lossy compression scheme [17], [14], inspiring a great deal of further research in the topic. In this introduction to wavelets, we will first attempt to answer the following three questions:

1. what is the goal of wavelets?
2. what are wavelets?
3. how is the wavelet transform achieved?

We will first define the *basis* of a function space. A basis of some function space is defined as the set of independent functions that may be combined to produce all functions from this space [18]. For example, suppose we have a function f which is a musical tone. We can construct f by using sine and cosine functions. These are the basis functions in our example.

The goal of wavelets is to represent functions in new ways [18]. Traditional Fourier methods have sine and cosine functions as their basis. This means that they are suited for signals which are very regular, but do not perform as well on choppy ones [18]. Wavelets strive to choose a basis which overcomes this problem.

A wavelet transform is achieved by the application of two filters to the input image: a low and a high pass filter. The low pass filter produces smoothed data by using a low pass filter; the high pass filter produces the detail data that was lost in the smoothing operation. This application is recursive, resulting in levels of different resolutions.

This chapter will review some of the extensive mathematical theory present in literature and provide examples of transforms. The overview presented here is neither meant to be complete nor involved, for details see [18].

4.1 Filters and Filter Banks

Both the scaling and wavelet basis functions are characterized by a set of coefficients referred to as filter coefficients [9]. These filter coefficients may be represented by some vector $\mathbf{h} = [h(0), h(1), \ldots, h(N)]$. The filter acts on an input signal by convolving it with the filter coefficients.

The operation may be described in terms of the input vector **x** and the output vector **y** as follows [6]:

$$y(n) = \sum_{k} h(k)\, x(n-k)$$

where **x** = [... , x(-1), x(0), x(1), ...] is the input signal, *x(n)* is the input symbol at time *n*, *y(n)* is the output at time *n*, and *k* is simply the *k*-th filter coefficient. (For the discussion of input signals at negative time, see Example 4.1.1.)

We will begin by introducing two very important types of filters: the low and high pass filters.

4.1.1 Low Pass Filter

The role of a low pass filter is to smooth the input signal by removing the bumps [18], which correspond to the high frequencies.

As an example, if we consider the simplest low pass filter with only two filter coefficients $[\frac{1}{2}, \frac{1}{2}]$, and use the Equation of Section 4.1.1, we obtain the following low pass filter:

$$L(n) = \frac{1}{2}x(n) + \frac{1}{2}x(n-1)$$

where *x(n)* is the input signal [18], [12]. We use the letter **L** to denote the low pass filter. This filter is then simply a moving average of the current component of the input signal *x(n)* with the previous component.

Example 4.1.1:

Suppose we have the input vector $\mathbf{x}^T$ = [10 4 6 12 4 2 4 4]. The result of applying the low pass filter presented above to this input results in the output vector **L** = [7 7 5 9 8 3 3 4]. The reader may notice that l(0) is the average of x(0) with the input x(7). The equation dictates that $l(0) = \frac{1}{2}x(0) + \frac{1}{2}x(-1)$. Obviously, x(-1) does not exist in the input signal; there are several ways of dealing with this problem. We "wrap around" to the end of the input; this is referred to as periodic extension. For now, we chose this method, as it simplifies the explanation of the recovery of the original signal (see Section 4.1.3).

4.1.2 Highpass Filter

The high pass filter is a moving difference. While the low pass filter performs smoothing to the input signal, the high pass filter picks out the irregularities in it and it removes the lowest frequencies.

The companion high pass filter to the low pass one presented in Section 4.1.2 is defined simply by taking h = [$\frac{1}{2}$, -$\frac{1}{2}$], i.e.:

$$H(n) = \frac{1}{2}x(n) - \frac{1}{2}x(n-1)$$

where **H** is the high pass filter, *x(n)* is the input signal [12], [18]. This filter is then simply a moving difference between adjacent input signals.

Example 4.1.2:

If we consider the same input vector **x** = [10 4 6 12 4 2 4 4], and apply the above high pass filter to it, we obtain the output vector **H** = [3 -3 1 3 -4 -1 1 0].

4.1.3 Filter Banks

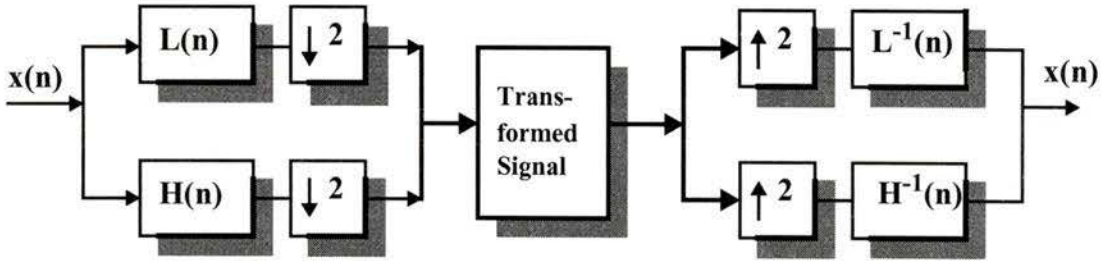
The high pass and lowpass filters are not invertible by themselves [18]. In the examples 4.1.1-4.1.2, the average values of adjacent function inputs are not sufficient in themselves to recover the original signal; neither are the differences. However, together these two filters do pass on enough information to recover the original signal fully. They also do something desirable: the separation of the input signal into low and high frequency bands. There is only one problem: the signal length has now doubled since the length of both the high pass and low pass filter is equal to the length of the original signal. *Downsampling* is the simple solution.

Downsampling means that we keep exactly half of the output of each of the filters; specifically, all the components in odd numbered positions are discarded and the ones in even-numbered positions are retained. Traditionally, the even components of the low pass filter make up the first half of the transformed signal, and the even components of the high pass filter make up its second half. The original signal is still recoverable [18].

Together, the low and high pass filters make up the analysis filter bank. The two inverse filters that recover the original signal make up the synthesis filter bank. The analysis and synthesis filter banks are depicted in Figure 4.1.

The low pass filter of the analysis filter bank is labelled L(n), the high pass filter is labelled H(n), and downsampling is depicted as $\downarrow 2$. Recovery of the input signal $x(n)$ is accomplished by first upsampling the output of the analysis filter by 2, and then applying the inverse low and high pass filters, depicted as $L^{-1}(n)$ and $H^{-1}(n)$ respectively.

Figure 4.1 The Analysis and Synthesis Filter Banks



4.1.4 Matrices

The entire operation described thus far may be accomplished by using simple matrix operations. This is because every linear operator, such as the filters presented at the beginning of the chapter, may be represented by a matrix [18]. For example, given the input vector **x** and the filters presented in Sections 4.1.1 - 4.1.2, the output vector **y** may be obtained by carrying out the following matrix operation:

$$\begin{bmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ -1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1/2 & 1/2 \end{bmatrix} \begin{bmatrix} 4 \\ 10 \\ 4 \\ 6 \\ 12 \\ 4 \\ 2 \\ 4 \end{bmatrix}.$$

The columns of the matrix contain the filter coefficients of the low or the high pass filters. Matrix multiplication of this matrix with the input vector **x** results in the output vector **y** which contains sums and differences obtained from the low and high pass filters. In order to achieve the downsampling, we removed the odd numbered columns of the matrix. In order to achieve the wrap-around, we moved the coefficient 4 to the beginning of the input vector x.

This matrix represents the analysis filter bank. The inverse of this matrix is then the synthesis filter bank which allows us to recover the original data (provided that a normalization factor is included, see [17]).

There are many types of filter banks in literature. One of these is the orthogonal filter bank. In this type of a filter bank, the analysis and synthesis banks are transposes and inverses of one another. Finding the inverse matrix for an orthogonal filter bank then becomes trivial - we simply transpose the columns of the analysis bank. When the two banks are inverses of one another but not necessarily transposes, the filter bank is said to be biorthogonal. The coefficients of the analysis filter bank are different from the coefficients of the synthesis filter bank for a biorthogonal transform.

Thus far we have only mentioned how one may transform an input signal by applying a low and a high pass filter, and discarding half the elements. We will now introduce the scaling function and the wavelet function, which are the crucial connection between wavelets and filters [18]. The low pass filter determines the scaling function, the high pass coefficients produce the wavelets.

4.2 The Scaling Function

Recall the low pass filter of Section 4.1.1. Corresponding to any low pass filter there is a scaling function ϕ. This function is constrained to obey a *dilation* equation of the following form:

$$\phi(t) = \sum_{k=0}^{N} L_k \phi(2t-k) \quad .$$

Dilation may be thought of as scaling [6]. The L_k in the equation are the coefficients of the low pass filter, where N denotes the number of such coefficients. The scaling function thus specifies a way of convolving the input signal with the filter coefficients. (In our case, we

multiply the above sum by 2.) This equation produces a set of basis scaling functions with time scales.

The solution (if it exists) to this function is constrained to be zero outside some interval $0 \le t < N$.

For our example from Section 4.1.1 with $2c_0 = 1$ and $2c_1 = 1$, the dilation equation becomes

$$\phi(t) = \phi(2t) + \phi(2t-1)$$

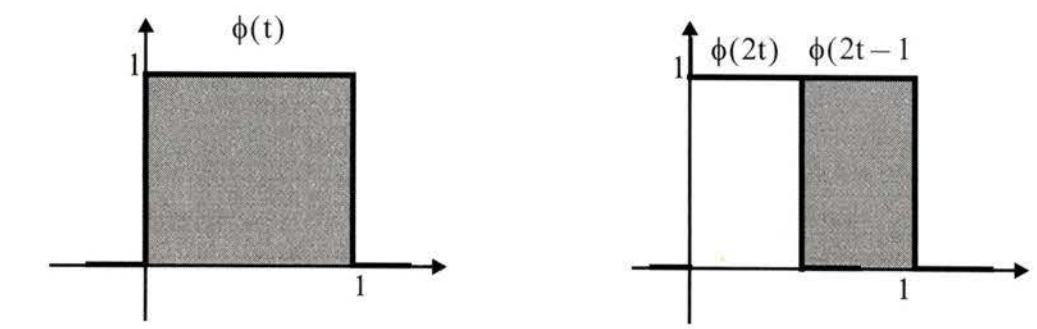
An example of a function which satisfies this equation is:

$$\phi(t) = \begin{cases} 1 & \text{on } [0, 1) \\ 0 & \text{otherwise} \end{cases}$$

This function turns out to be the Haar box function [18].

Yet another way of looking at this equation is graphically. The graph of $\phi(t)$ is shown on the left of Figure 4.2.

Figure 4.2 The Haar Scaling Function, and its Scaled and Translated Versions



The graphs of $\phi(2t)$ and $\phi(2t-1)$ are depicted on the right of the above Figure 4.2. The function $\phi(t)$ is compressed by 2 to obtain the function $\phi(2t)$; this is translated to obtain $\phi(2t-1)$.

4.3 The Wavelet Function

The wavelet equation is associated with the coefficients from the high pass filter. A function w is called a wavelet if it satisfies the following equation:

$$w(t) = \sum_{k=0}^{N} H_k \phi(2t-k)$$

where $\phi(t)$ is the scaling function defined in Section 4.2. The wavelet coefficients H_k are the high pass filter coefficients.

If we again refer to the example of Section 4.1.2, the above wavelet equation has an extra factor of 2 associated with it.

Alternately, we can think of the equation as:

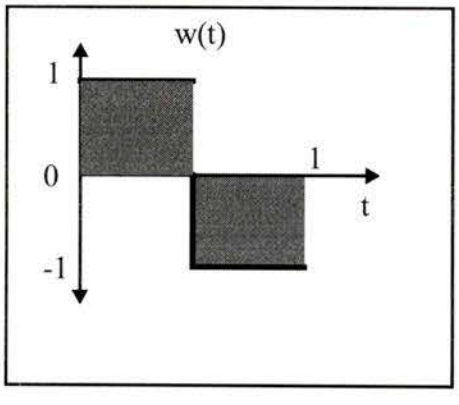
$$w(t) = \phi(2t) - \phi(2t-1)$$

with a high pass filter [1, -1].

Using the Haar scaling function defined in Section 4.2, we obtain the Haar mother wavelet, defined by w(t) = 1 on [0,1/2), w(t) = -1 on [1/2, 1) and w(t) = 0 elsewhere [12].

Again, we can think of this graphically [18], as shown in Figure 4.3.

Figure 4.3 The Haar Mother Wavelet Function



To summarize what has taken place, we have introduced two functions, the wavelet and the scaling function. The construction of wavelets begins with the solution of the dilation equation. The wavelet function comes from the scaling function, and the basis comes from the translation and the scaling of the wavelet function. We will now move on to the implementation of the algorithm; for full details and mathematical theory behind wavelets, the reader is referred to [18].

4.4 The Wavelet Transform

The wavelet transform may be implemented through the convolution of the low and the high pass filters with an input signal.

Given input signal *f*, we convolve it with the low and the high pass filter. The convolution with the low pass filter smooths the data, producing *scaling coefficients*. The convolution with the highpass filter produces the *difference coefficients*. Typically, the scaled coefficients are stored in the first half portion of the original signal (the low pass component); the difference coefficients in the second half (the high pass component).

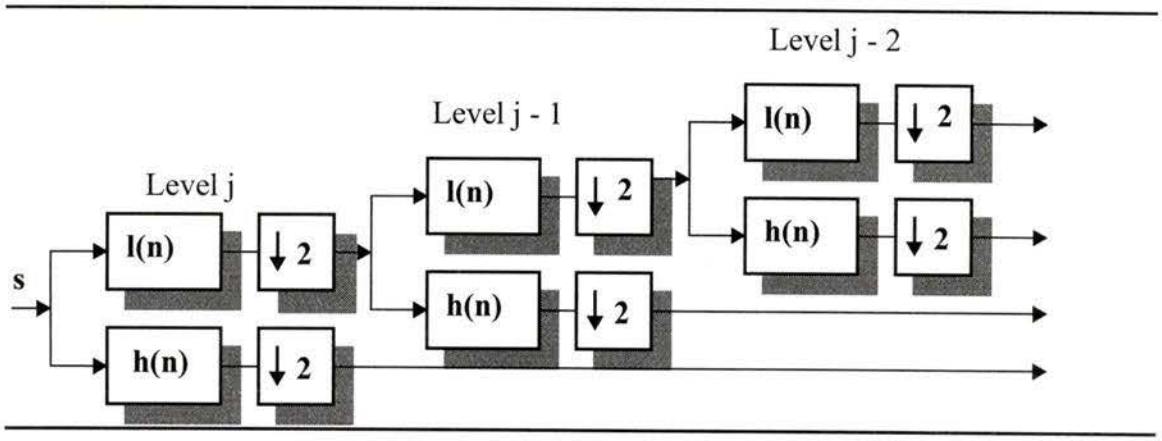
Most of the original signal information exists in the scaled coefficients. If we take only the first half of the signal corresponding to the scaled coefficients, and apply the low and the high pass filters again, we get two new sets of data in the place of the low pass components, each one a quarter of the size of the original signal. This is the basic idea behind the wavelet transform. Since the lowpass portion of the signal is continually decomposed, the low frequency content of the data is being represented by a smaller and smaller subset of the original signal.

Each such decomposition produces a new coarser level of resolution.

The maximum number of such decompositions or levels is $\log_2(\min(nr, nc))$, where *nr* is the number of input image rows and *nc* is the number of input image columns.

The wavelet transform is depicted in Figure 4.4. The low pass filter is labelled l(n), the high pass filter h(n); *s* is the input signal.

Figure 4.4 The Wavelet Transform



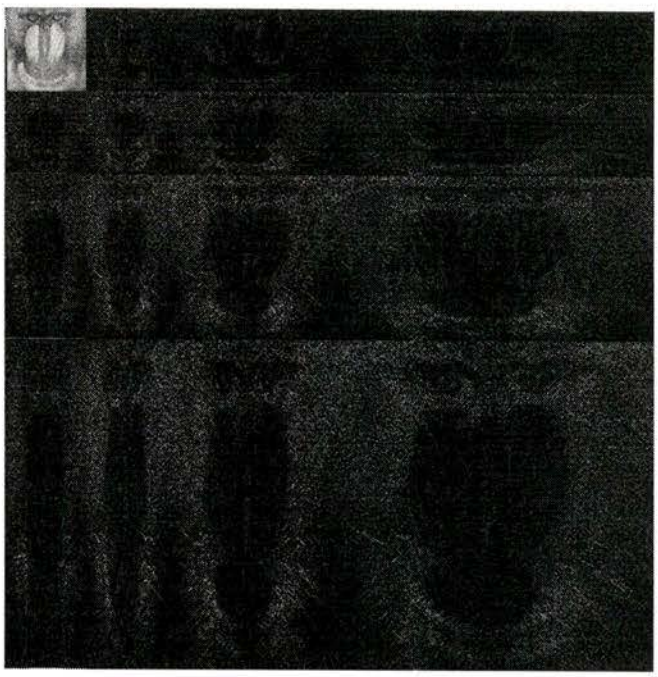
4.5 Multi-Dimensional Wavelet Transform

So far we have generally described how a transform may be applied to a one dimensional signal. The transition to two dimensions (i.e. an image) can be easily made. There are two methods of generalizing the transform to two dimensions, referred to as standard and non-standard decomposition [6].

4.5.1 Standard Decomposition

The one dimensional wavelet transform described previously is applied *recursively* to each of the rows of the image. For each row, this gives as an average value and the wavelet (or detail) coefficients (assuming that we performed all of the transformations). The wavelet transform is then applied *recursively* to each column of the image. The process generates the wavelet transformed image shown in Figure 4.5.

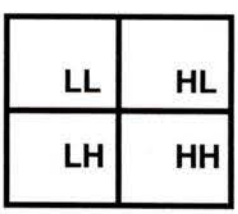
Figure 4.5 Three Level Standard Decomposition



4.5.2 Non-Standard Decomposition

In this type of wavelet decomposition, we first apply one step of the wavelet transform to each row of the input image, and then to each column. The result breaks the image into four regions referred to as subbands shown in Figure 4.6.

Figure 4.6 Subband Structure



The LL subband contains the smoothed input signal (the averages in both the vertical and horizontal directions), the HL subband emphasizes the horizontal image features, the HL

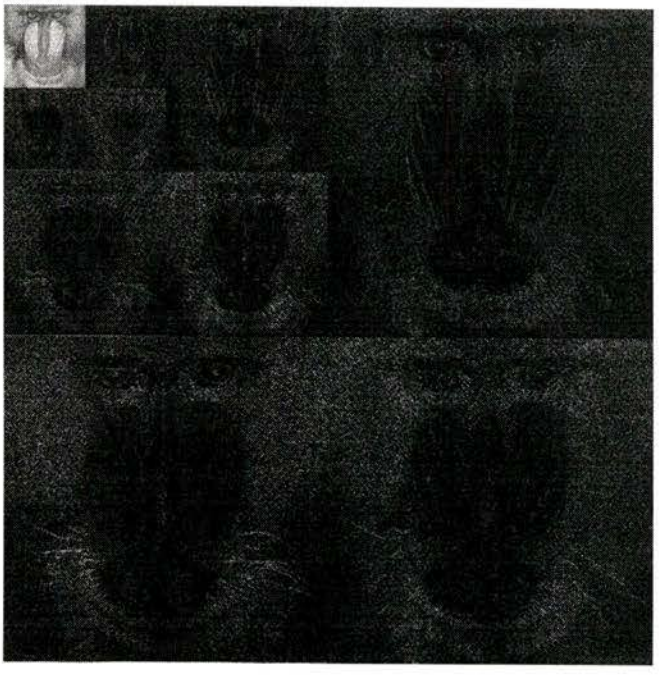
subband emphasizes the vertical features and finally the HH subband emphasizes the diagonal features. These result from the combinations of the low and high pass filters that are applied. For example, the LL subband is generated from the application of the low pass filter in the horizontal and the vertical direction. The low pass filter is the "averaging" filter; so the signal contained in this subband is smoothed.

We then apply the wavelet transform to the rows and columns of the LL subband only. This process is continued until the desired number of transformation steps has been achieved.

One of the advantages of using this decomposition as opposed to the standard one is that each step of the transform needs to compute only one quarter of the coefficients of the previous step; as opposed to one half in the standard case [6]. It is therefore more efficient.

The wavelet transformed image is shown in Figure 4.7.

Figure 4.7 Three Level Non-Standard Decomposition



4.6 Multi-Resolution Reversible Transforms

We will now define exact reconstruction systems. Most of the literature deals with these types of systems. Suppose we have an input signal x, and apply the forward and inverse

wavelet transform to yield the reconstructed signal x^1. The system is defined to be an exact reconstruction system if x and x^1 are identical up to a multiplicative constant and a delay term [22]. In practice the original and reconstructed systems will differ slightly due to the lack of precision and the inefficiency of storing floating point values; a difference that should not be perceivable by the human eye. Unfortunately, in some applications, even this minute loss of information is not acceptable.

A reversible system is defined as an implementation of an exact reconstruction system in integer arithmetic so that a signal with integer coefficients may be losslessly recovered [22].

Reversible transforms are defined in terms of low and high pass filters. However, they are generally not linear [22], and so are approximations to the wavelet transform. Therefore, they are defined as expressions instead of in terms of filter coefficients, as is typical for wavelet transforms. Several such transforms will now be presented.

4.6.1 S Transform

The S transform is similar to the Haar multi-resolution image representation [15]. Given an input signal x(n), n = 0,1,...N-1, where N is even, the low and high pass filters are defined as follows:

$$l(n) = \lfloor (x(2n) + x(2n + 1)) / 2 \rfloor \quad , \quad n = 0, \ldots, N / 2 - 1$$

$$h(n) = x(2n) - x(2n + 1) \quad , \quad n = 0, \ldots, N / 2 - 1$$

The $\lfloor x \rfloor$ function means downward truncation of x to the closest *smaller* integer. Knowledge of the average and the difference of two numbers is sufficient to recover the original numbers; thus, the transform is fully reversible.

This transform is very simple to implement; however, it leaves a residual correlation between the high pass components [15].

4.6.2 S+P Transform

The S+P (S-transform + Prediction) estimates the high order filter by subtracting a prediction value from the high pass filter of the S transform [15]. We borrow it from [15]. Suppose we define:

$$\Delta l(n) = l(n-1) - l(n)$$

where l is the low pass filter defined for the S transform in Section 4.6.1. The estimated prediction $\tilde{h}(n)$ can now be defined:

$$\tilde{h}(n) = \frac{2}{8} \times \Delta l(n) + \frac{3}{8} \times \Delta l(n+1) - \frac{2}{8} \times \Delta h(n+1) \quad .$$

The high pass filter h^1 then becomes (here $h(n)$ is the high pass filter defined for the S transform in Section 4.6.1):

$$h'(n) = h(n) - \left\lfloor \tilde{h}(n) + \frac{1}{2} \right\rfloor \quad .$$

4.6.3 Predictive Transform

Given an input signal $x(n)$, n = 0,1,...N-1, where N is even, the low and high pass filters are defined as follows [16]:

$$l(n) = x(2n) \quad , \quad n = 0, \ldots , N/2$$

$$h(n) = x(2n+1) - \text{TRUNC}\left(\frac{x(2n) - x(2n+2)}{2} \right) \quad , \quad n = 0, \ldots , N/2.$$

TRUNC(x) in the equation refers to downward truncation of x, which is slightly different from the $\lfloor x \rfloor$ defined above. The definition is identical for positive x to the one for $\lfloor x \rfloor$; for negative x, TRUNC sets x to the closest largest number. For example, TRUNC(-3.5) = -3.0.

The Predictive transform uses a simplistic lowpass filter - one that simply discards every odd input element and retains every even one. As a result, the high pass filter can make a very accurate prediction of the difference signal. Specifically, it predicts the detail signal as the difference between the discarded odd elements and the average of the two neighboring even values. The signal is still fully reversible due to the fact that we retain the exact value of the even coefficients.

4.7 SPIHT Example

SPIHT is a codec (compression / decompression system) introduced by Said and Pearlman [14]. The algorithm is based on first performing a lossless (reversible) wavelet transform, and then using a coding scheme to compress the wavelet coefficients. The approach used to compress the wavelet data is similar to the EZW algorithm [17]. The main principles of the

EZW operation include ordering the bits of the wavelet coefficients by magnitude, transmitting these ordered bit planes, while exploiting self similarity of the wavelet transform across different bit planes. We mention the algorithm here since it is designed specifically for lossless compression, and has been used for medical compression purposes [14].

The input image is first transformed using some wavelet transformation. The wavelet transform coefficients are referred to as *x(n)*, where n = 0,... *N* (the total number of transformed coefficients). In the following discussion, we will assume that the coefficients are stored in some sort of a list structure, and that *coefficient coordinates* define the position of a particular coefficient in that list.

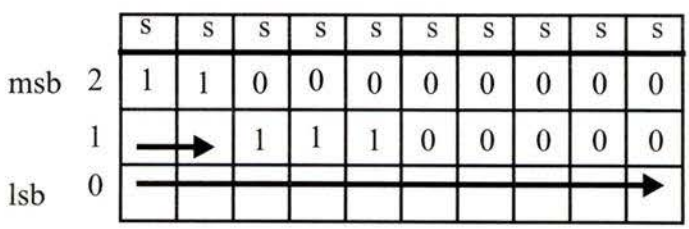
4.7.1 Ordering Strategy

We want to transmit the highest-magnitude coefficients first. We order the wavelet-transformed coefficients according to the number of bits that is required for a magnitude binary representation as follows:

$$\lfloor \log_2 |x_n| \rfloor \geq \lfloor \log_2 |x_{n+1}| \rfloor \qquad , n = 1, \ldots, N .$$

This gives the ordered coefficients shown in Figure 4.8.

Figure 4.8 Ordering Strategy



For our example presented above, we are using coefficients consisting of only 3 bits, where msb refers to the most significant bit and lsb refers to the least significant bit. The bits of some coefficient x(n) are contained in a particular column of Figure 4.8. The top-most row

contains the sign of the coefficient in that column. The rows are numbered from bottom up; the bits in the lowest row are the least significant.

The most effective transmission to the compressed file is to sequentially send the bits in each row to the decoder, as is indicated by the arrows in the diagram. For the moment we ignore the fact that the decoder needs to know the actual pixel coordinates prior to the ordering. We need to transmit two things in order to recover the data: the number of coefficients in each row of the table, and the actual coefficient bits. Note that we only need to transmit the 1 bits.

We have transmitted sorted data to the decoder. However, we have for the moment omitted the sending of the original coordinates of the coefficients prior to the sort. The decoder can recover the sorted data quite easily, but it cannot recreate the original order of the pixels prior to the sort. If we explicitly send the pixel coordinates in the original ordering (i.e. prior to the sort), we waste a significant number of bits, thus increasing the size of our compressed file.

The authors propose a sorting algorithm which eliminates the need to send the coefficient coordinates explicitly. The sorting algorithm selects the coefficients to be transmitted along a particular execution path that is shared by the coder and the decoder, so that the ordering information is explicit.

4.7.2 Sorting Algorithm

The sorting algorithm does not explicitly sort the coefficients; in the n-th pass it selects the only those coefficients x(k) in the range $[2^n, 2^{n+1})$. A coefficient is said to be *significant* if it falls in this range, and *insignificant* otherwise. The primary idea behind the algorithm is to partition the wavelet data into subsets (and not just individual pixels). Each time a subset is found insignificant, we know that all of the pixels in it are also insignificant. Conversely, each time a subset is found to be significant, it is further partitioned using the rules described in the next section, and the test for significance is performed again.

In order to show how the coefficients are partitioned into significant and insignificant sets, we need to introduce some terminology. We will then show how the partitioning is performed.

4.7.3 Spatial Orientation Trees and Set Partitioning Algorithm

The spatial orientation tree is defined such that each node corresponds to a pixel. The pixels at the highest levels of the wavelet pyramid are the roots of the trees. These roots contain some offspring. The tree is used to define some sets as follows:

- **O(i, j)** -the set of coordinates of all offspring of a node (i, j);
- **D(i, j)** - the set of coordinates of all descendants of a node (i, j);
- **H** - the set of spatial tree roots;
- **L(i, j)** - the set difference of D (i, j) - O (i, j).

We partition these sets using the following rules:

1. Initially, we have the partition corresponding to the sets {(i, j)} and D(i, j) for all (i, j) in H.

2. If D(i, j) is significant then it is partitioned into L(i, j), plus the four single element sets which make up the set O(i, j).

3. If L (i,j) is significant then it is partitioned into the four sets D (k,l) which are the descendants of each of the offspring contained in O(i, j).

4.7.4 Coding Algorithm

The coding algorithm is necessary so that the execution path of the encoder is implicit, and may be mirrored by the decoder. The description provided here is quite general, the reader is referred to [14] for details. There are three separate sets maintained: list of significant pixels (LSP), list of insignificant sets (LIS), and list of insignificant pixels (LIP). The algorithm is composed of four steps:

1. Initialization: coordinates in **H** are added to the LIP, and those with descendants to the LIS

2. Sorting Pass

3. Refinement Pass

4. Quantization Step Update (skipped if lossless compression is indicated).

During the sorting pass, the pixels in the LSP are tested for significance and moved to the LSP if found to be significant. Sets in the LIS are tested similarly; if a set is found to be significant, it is partitioned according to the set partitioning rules. Any subsets obtained

from the partition that contain more than one pixel are added to the end of the LIS, while any singleton sets are added to either the LSP or the LIP depending on their significance.

The algorithm makes a small adaptation if lossless compression is indicated. The algorithm becomes inefficient when the least significant coefficients are being coded. The modification codes the coefficients as described above up to the third least significant bit, and then uses a predictive approach similar to JPEG's lossless technique to code the remaining three bit planes.

5 The New MICS Approach

We have three things to address when describing the approach we took to achieve the compression results: the wavelet transform, the model and the arithmetic encoder.

A wavelet transform is applied to the entire input image. Wavelets have been extensively used in conjunction with compression systems [22], [14], [17]. The majority of the techniques we found in the literature use some variation of the EZW algorithm [14] (see Section 4.9). This algorithm replaces whole sets of wavelet coefficients with one symbol (the Zero Root) to achieve very good compression ratios. An alternate approach, and one we decided to use, exploits the properties of the wavelet coefficients by building a model. These properties include the frequency distribution of the wavelet coefficients, and spatial correlations that exist between the wavelet coefficients (Section 5.1.2).

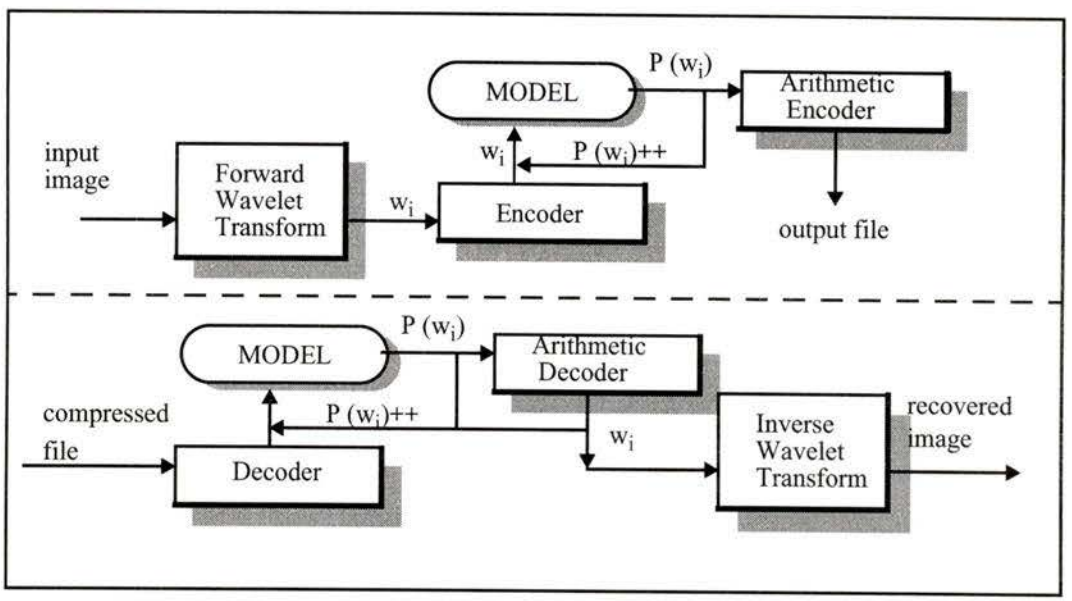
The choice of the wavelet transform itself is also important. The main reason behind using a non-linear integer transform over the more widely used linear ones is that lossless image compression is required. (The transform is non-linear precisely because integer arithmetic is used.) As pointed out in section 4.6, linear transforms use floating point arithmetic, and are exactly recoverable given enough precision. We do not wish to transmit floating point data to our compressed file, as it requires much more storage than integer data. Thus, we require integer transforms. In addition to being perfectly recoverable, an integer transform has the advantage of being very fast to execute, since many of the divisions and multiplications may be done by bit shifts. These transforms are not wavelet transforms in a strict mathematical sense; however, they are referred to as such in literature [22], and we do so as well in this work.

Finally, our approach took into consideration the choice of low and high pass filter coefficients, and their effect on compression ratios.

The encoder is responsible for generating the model. The model built is adaptive, since this type tends to work well with large files where it has time to adapt to the data being compressed [2]. The encoder passes each wavelet coefficient w_i to the model. The model generates a probability $P(w_i)$ based on information that it has about coefficients correlated to w_i, and passes it to the arithmetic encoder. The arithmetic encoder translates this probability into bits and sends them to the output file. Following the transmission, the model is updated by the encoder (i.e. the probability of coefficient w_i occurring is increased) to reflect the occurrence of the coefficient. The encoder and decoder are illustrated in Figure

5.1. This codec (compression / decompression system) developed here is referred to as MICS (Medical Image Compression System).

Figure 5.1 The MICS Codec



The wavelet transform, the model and the arithmetic encoder will now be described in detail.

5.1 Wavelet Transform

We will discuss the frequency distribution of the wavelet transformed data, and the spatial correlations which exist in the wavelet hierarchy. The properties of the wavelet coefficients should be well understood in order to build a model which is representative of them. Before we address these, the choice of the low and high pass filter coefficients will be addressed.

5.1.1 Filter Coefficients

It is well known that the choice of filter banks in wavelet compression is a critical issue that effects compression [20]. However, the issue of deciding which is the "ideal" filter bank remains unresolved. Several standards have been proposed, relating to such issues as filter regularity, orthogonality and others [20], [4]. However, these measures are typically involved with not just finding the best data compression ratios, but image quality as well,

an issue which is not relevant to this work. In general, it has been noted that the characteristics of the image determine the best choice of transform; specifically, smooth and noiseless images require attenuation at the low frequencies, while noisy images require a filter with a small gain at the high frequencies [15]. In order to simplify matters, we have chosen to use zero-order entropy as the criterion to choose the best transform.

Zero-order entropy uses the frequency of occurrence of each coefficient as an estimate probability of that coefficient. We use that probability in the entropy equation defined in section 2.2. The entropy value is a measure of the information content of the data; specifically, the number of bits need to represent that symbol, and so can be used to calculate the compression ratio. It is important to keep in mind that zero order entropy uses a simplistic and fixed (i.e non adaptive) probability calculation, which is usually not the best estimate possible (by best we mean that better probability estimates will generate better compression ratios). However, it can be satisfactory for comparison purposes.

Table 5.1 depicts zero order entropy values obtained with different wavelet transforms over a variety of images. The number of transformation levels performed in each case is the maximum number possible $\lfloor \log_2 \langle \min \langle nr, nc \rangle \rangle \rfloor$ where *nr* is the number of image rows and *nc* is the number of image columns). We compare the S transform, the S+P transform [15], and the predictive transform (see section 4.6 for transform details) in table 5.1. Even though the focus of this work is medical image files, we have also chosen to show the results using two other files, namely the widely used *lena* and *barb* files. The reason for this is that we wanted to show the entropy results from the different transforms applied to a variety of input files; the medical image files supplied did not have entropies higher than 7; the *lena* and *barb* files did. The transform obtaining the best entropy appears in bold.

Table 5.1 Zero-Order Entropies Obtained Using Different Wavelet Transforms

Image	0 order entropy	S Trans.	S+P Trans.	Predic-tive Trans.
UL1	4.98	4.24	4.32	**4.17**
UL2	6.33	5.23	5.24	**5.14**
UL3	2.69	3.05	3.17	**2.87**
UL4	5.39	4.12	4.15	**4.04**
UL5	3.76	3.59	3.43	**2.29**
UL6	1.58	2.09	2.27	**1.83**
UL7	4.21	3.77	3.83	**3.47**
UL8	3.18	3.30	3.22	**2.99**
lena	7.57	5.38	**4.95**	5.09
barb	7.46	5.60	**5.13**	5.38

It is claimed that good general predictors exists which perform well over a broad range of images [15]; in fact, the S+P transform found in table 5.1 is claimed to be such a transform. This claim does not appear to hold based on our results found in Table 5.1. In fact, the S+P transform appears to clearly outperform the other two transforms given an image with a very high zero-order entropy (such as lena). On the other hand, the S+P transform does not appear suited for images possessing a lower entropy. In fact, the difference between the entropy obtained using the S+P transform and the predictive transform is significant.

The approach we have chosen to resolve the issue of choosing the filters for the transform is adaptive, but rather ad hoc. Prior to wavelet transforming the image, we calculate its zero order entropy. If the entropy is above a threshold value, we use the S+P transform, which based on our tests does improve the compression ratio (for example, see lena and barb results in Table 5.1). If the zero order entropy is below that value, we use the predictive transform in the wavelet stage of the compression process. This small step, along with the performance of the predictive transform on smooth images, is important to the compression ability of MICS.

5.1.2 Properties of the Wavelet Coefficients

The wavelet transformed coefficients have a distinct frequency distribution, which should be taken into account when building the model. In addition, the wavelet transform leaves

some residual correlations between the different coefficients [17] which allows to build a model which makes predictions more accurately. These two properties will now be addressed.

5.1.2.1 Frequency Distribution of Wavelet Coefficients

The wavelet transform has the following advantage: it decorrelates the input image data, packing the image energy into fewer coefficients. As a consequence, it often decreases the range of the frequency distribution of the original data. The wavelet transformed data typically has a frequency distribution of its coefficients centered closely around zero. To illustrate this point, Figure 5.2 shows the frequency distribution of a sample medical image file. A corresponding Figure 5.3 shows the frequency distribution of the absolute value of the coefficients obtained by applying the wavelet transform to the data.

Figure 5.2 Frequency Distribution of Image Pixel Coefficients

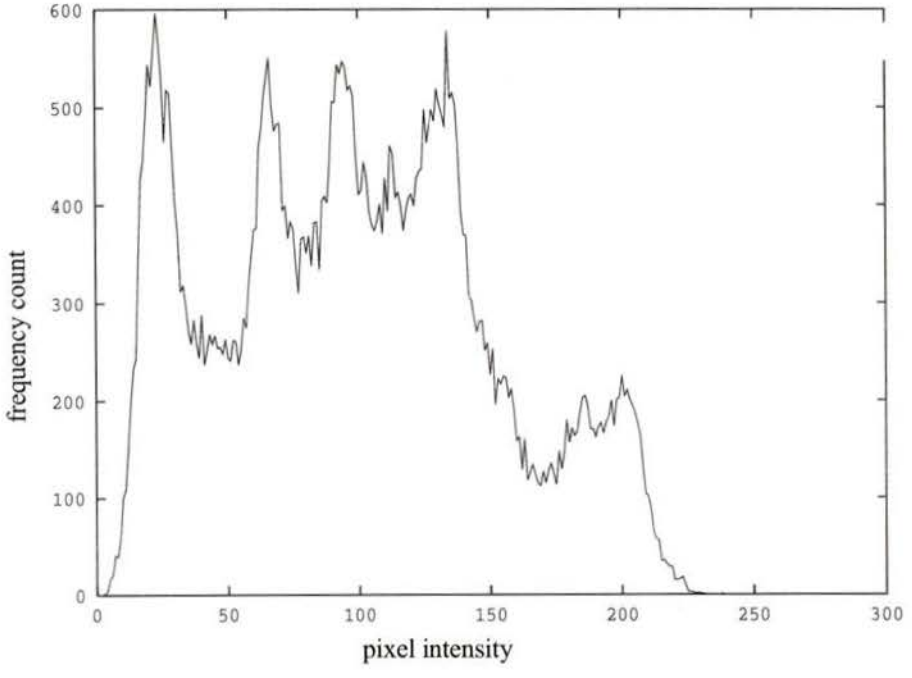
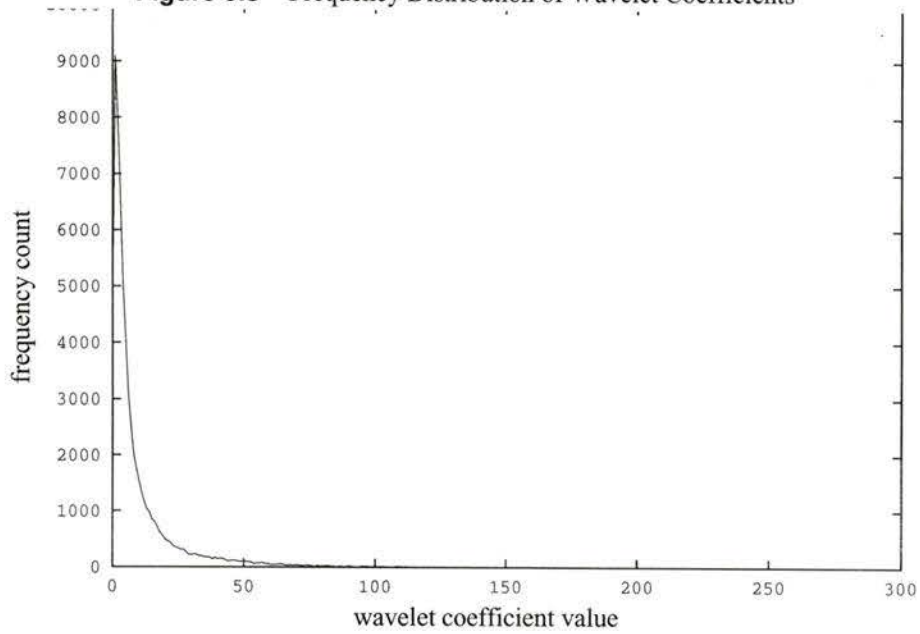


Figure 5.3 Frequency Distribution of Wavelet Coefficients

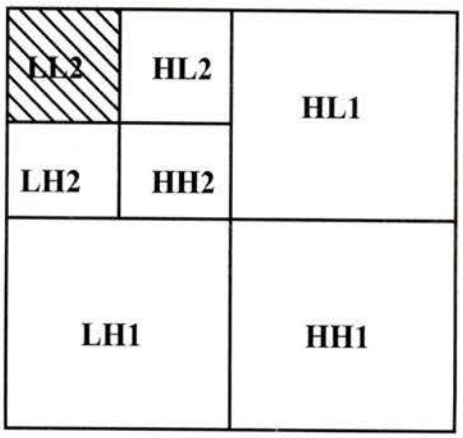


The wavelet transformed data has a much smoother curve, with most coefficients close to or at zero. This property is especially appealing to lossy compression techniques, since only a small percentage of the higher-magnitude coefficients need to be transmitted (along with their positions!) in order to recover a good approximation to the original. Lossless techniques stand to also benefit through the construction of careful models which exploit these characteristics.

5.1.2.2 Spatial Correlations in Wavelet Hierarchy

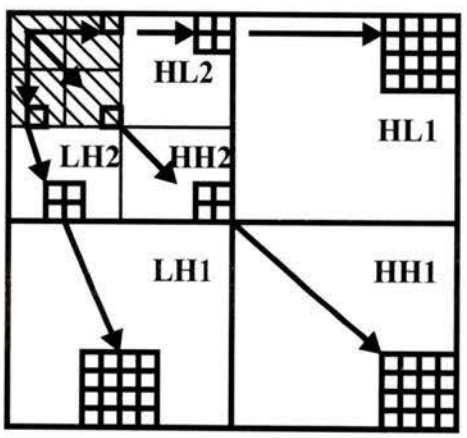
As already described in Section 4.7, the non-standard decomposition begins by decomposing the image into four subbands, LL, HL, LH, HH. These subbands arise from the application of the low and high pass filters to the image rows and columns. The next coarser level of wavelet coefficients is obtained by decomposing the upper left subband. This gives us a two scale wavelet decomposition, shown in Figure 5.4.

Figure 5.4 Two-scale Wavelet Decomposition



In such a system, every coefficient at a particular level can be related to a set of coefficients at the next finer level (i.e. the next higher level) [17]. The coefficient at the coarse scale is referred to as the parent, and the coefficients at the corresponding finer scale are referred to as its children. This parent child dependency is shown in Figure 5.5.

Figure 5.5 : Parent-Child Dependencies



In an image where the number of rows and the number of columns are a perfect power of 2, each parent always has four children, and every pixel has a parent (except the pixels in the highest level of the LL subband). The four children are referred to as siblings.

The goal of this transform is to produce coefficients which are decorrelated. However, in general most transforms leave a residual correlation between the parents and the children [17], and a residual correlation between the four siblings. The presence of these correlations allows for the generation of more accurate prediction values. The approach taken by the model to exploit these correlations will be described shortly in Section 5.2.

5.1.3 Non-standard Images

Implementations of wavelet transforms often assume that images have dimensions which are perfect powers of 2 [10], [14]. If an image does not satisfy this criterion, it must be padded with some values. This means that new values are introduced and must be coded by the compressor in order for the original to be fully recovered. The padding therefore increases the size of the compressed file unnecessarily. The approach we adopted was to implement our own lossless integer library which deals with images of all dimensions without padding.

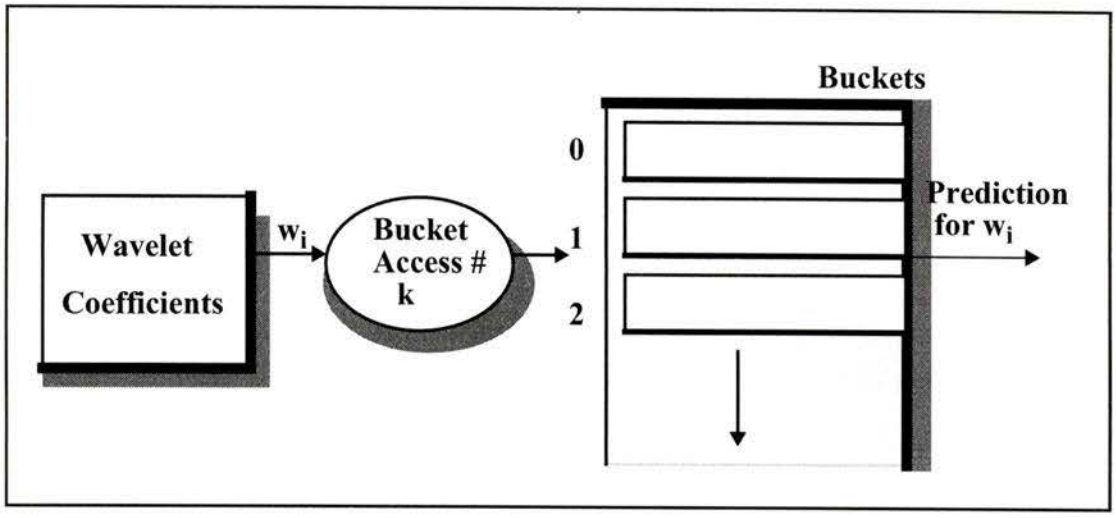
5.2 The MICS Model

The concept of a model was generally introduced in Section 2.3, where we stressed that the most important function of the model is to supply probabilities for input symbols. The simplest model is one which allocates a fixed probability for each of the symbols in the input file. Our zero-order entropy calculation above used such a model; the probability of each symbol was simply the percentage of the number of occurrences of each symbol. Such a model does not take context into account. For example, if we are compressing a text file, we can make our prediction of the current character based on the character directly before it. The number n of preceding characters that we take into account when making the current prediction is referred to as the order of the model. This type of a model exploits correlations between characters in a text file. If we were to use it for compressing an image file, we would ignore a second correlation: pixels in an image are correlated vertically as well as horizontally. By applying the wavelet transform, we also introduce new spatial correlations (see Section 5.1.2.2), which should be taken into account when designing the model in order to achieve the maximum possible compression.

Before we get more specific, it would be helpful to generally describe the model used in this compression scheme. We want the model to reflect the spatial correlations between the

parent and children wavelet coefficients. The most natural way to accomplish this is by treating the parent value of the current wavelet coefficient as the context. There will be a set of such contexts, which may be represented by a list of buckets. A bucket is simply a structure which holds some probability tables. These probability tables hold the set of coefficient values which have occurred in the bucket along with associated probability values. It is the responsibility of the bucket to generate a prediction for the current coefficient. Figure 5.6 illustrates the basic idea.

Figure 5.6 Buckets



We must be able to calculate which bucket we will use for the current symbol (referred to as the *bucket access* number). Once this is known, the probability for our current coefficient may be generated by that bucket. The arithmetic encoder will then take care of translating this probability into bits, and transmitting it to the compressed file. Following the transmission of each symbol, its probability is incremented to account for the fact that our estimate of its "chance of occurring" has just increased.

Before we describe the model more specifically, we will address the zero frequency problem (see Section 2.3.3.1). Following this, we will describe the strategy adopted for numbering the buckets, the calculation for deriving the bucket access number, and finally, the contents (i.e. probability tables) of each bucket.

5.2.1 The Zero Frequency Problem

Our model starts off not predicting any coefficients (i.e. all coefficients have a probability of zero). In order to encode each new coefficient, the bucket defined by the bucket access number must generate a probability value for that coefficient. We must address the problem of what to do in the case where the coefficient we are trying to encode is not predicted by the current bucket.

The solution we have chosen is simple: every probability table, predicts one additional symbol, the *escape* symbol, (we refer to its probability as the *escape probability*). In addition, we construct an *escape alphabet* which makes a non-zero prediction for every coefficient which occurs in the wavelet transform. Each time we encounter a coefficient which is not predicted by the current bucket, we transmit the escape code, followed by the code for the prediction made for that coefficient by the escape alphabet.

The other option for solving the zero frequency problem was to initialize all the tables in the model to hold every coefficient in the alphabet, with some fixed initial probability. The overwhelming negative side of this solution is its memory requirements. The initial model would be huge, and so impractical. The other problem with this solution is deciding what probabilities to initially assign to the coefficients. If we assign all coefficients equal probabilities, the start-up time of the model is too long. By this we mean that a significant number of coefficients will have to be encoded for the model to start reflecting the characteristics of the file. If we assign non-fixed probabilities, we must transmit them to the compressed file, in order to tell the decoder to do the same; a wastage of bits.

5.2.2 Numbering of Buckets

Before we discuss the calculation determining the bucket number, we will mention how the actual buckets are numbered. As we showed in Figure 5.3, most of the wavelet coefficients are close to zero. We will now show how we can take advantage of this property.

The few larger coefficients will be difficult to compress, since they usually occur only once or twice. Therefore, instead of ordering the buckets sequentially, we chose a representation (borrowed from [15]) which was more appropriate to the wavelet transformed coefficients. This method represents a range of numbers by a value referred to as the magnitude set (MS) value. The magnitude set representation is shown in Table 5.2.

Table 5.2 The MS Number Representation

Magnitude Set (MS)	Amplitude Intervals
0	[0]
1	[-1], [1]
2	[-2], [2]
3	[-3], [3]
4	[-5,4], [4,5]
5	[-7, -6], [6, 7]
6	[-11, -8], [8, 11]
7	[-15,-12], [12,15]
8	[-23, -16], [16,23]
9	[-31,24], [24,31]

The representation takes into account that the wavelet transform produces numbers which are very close to zero. As the MS value increases, so does the range it represents. This accounts for the fact that we do not expect to see very many large coefficients, and so it is possible to be less accurate when representing them by the model. It is important to keep in mind that we are only using this representation to label the buckets. For all other calculations, the standard integer representation is used.

It is worthwhile pointing out that this step is relevant to the compression capability of MICS. Table 5.3 shows the compression ratios of five files compressed using (1) MICS with the MS integer representation to number the buckets, and (2) MICS with the standard integer representation to number the buckets (referred to as *Coding method 1* and *Coding method 2* in Table 5.3). The MS representation is clearly superior to the standard one; an advantage especially apparent as the entropy of the input file increases. (The remainder of MICS is described in following sections; here, it is relevant to note that the numbering scheme is significant to obtaining better compression ratios.)

Table 5.3 Coding Results Using the MS, and the Standard Representation

Image	Zero-order Entropy	Coding Method 1	Coding Method 2
UL5	4.97	**3.76**	3.94
UL1	6.32	**4.60**	4.96
UL8	2.69	**1.50**	2.11
UL4	5.39	**3.16**	3.62
UL6	3.76	**2.10**	2.66

We will try to show why this representation is superior through a simple example. Suppose we are currently trying to encode the wavelet coefficient w_i. We will ignore for the moment how we calculate the bucket access number (this is described in the following section), and describe the general procedure for encoding a coefficient. We go to the appropriate bucket, and use a probability table found inside it to try and generate $P(w_i)$, the probability for our coefficient w_i. In the case that we do not find the coefficient w_i inside the bucket, we will generate the escape symbol, encode w_i using the escape alphabet, and add w_i to our current probability table.

Now consider the way the buckets are numbered.

If we simply number the buckets using the standard integer representation, we make each bucket very specific. This means that the case where we do not find the coefficient inside the bucket's probability table will come up more often. As already pointed out, the escape symbol will be generated in each such case, followed by the probability obtained using the escape symbol. This causes a wastage of bits for two reasons. One, the escape symbol must be transmitted to the compressed file, and two, the probability generated by the escape alphabet is not as accurate a prediction as the one generated by a bucket in the model.

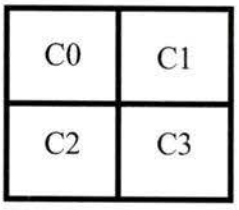
The MS scheme represents each bucket as a range of values which reflect the frequency distribution of the wavelet coefficients. This and the fact that this is a more general numbering scheme which does not generate the escape symbol as frequently improves compression.

5.2.3 Bucket Access Number

In order to determine which bucket will be used to generate a probability for our coefficient, we take advantage of the previously mentioned correlation between parent and children

wavelet coefficients as well as the correlation between the siblings. This correlation is used as the context which determines which set of probability tables will be used to encode the coefficient. We will restrict our discussion to images which have dimensions that are perfect powers of two (even though the codec is quite capable of handling all types of images) for ease of discussion. For this type of an image, each child has a parent. For each parent in a wavelet hierarchy, there are four children to transmit to the compressed file, as shown in Figure 5.7.

Figure 5.7 The Spatial Relationship of the Four Children



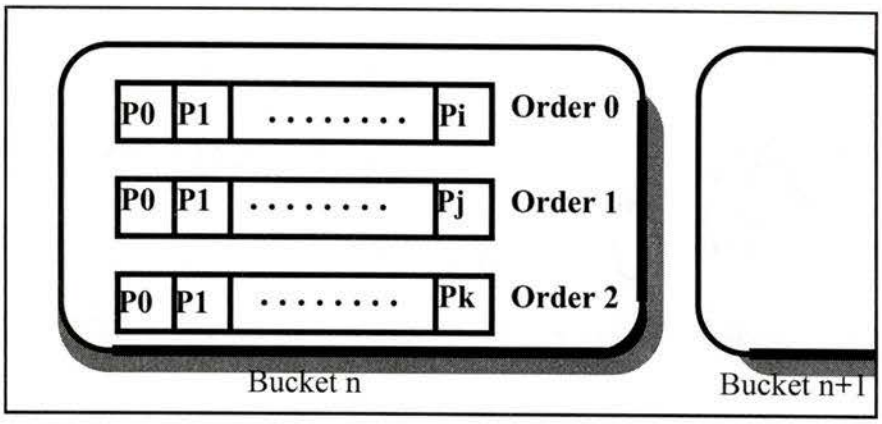
The parent-child dependencies were discussed in Section 5.1.2.2. The bucket for the first child c_1 is determined directly by the value of the parent. The bucket for the second child c_2 is calculated by computing the average of the parent and its first sibling c_1, the bucket for the third child is calculated by computing the average of the parent, c_1 and c_2, and similarly for the fourth child:

We are exploiting the correlation between the parents and the children, and between the siblings, by using it as a context to determine which probability table to use from the set of buckets. We could have used the exact value of the parent, and later the siblings, as the context determining the bucket, but we have found that the average was a better context estimate.

5.2.4 Bucket Probability Tables

To review what has taken place, we have described how to determine the bucket within a model, given a wavelet coefficient, and armed with the knowledge of that coefficient's parent and sibling values. We will now describe the internal bucket structure. Figure 5.8 illustrates the basic idea.

Figure 5.8 Internal Bucket Structure



Each bucket in the model contains three probability tables, labeled *order0*, *order1*, and *order2*. These probability tables hold probabilities and associated coefficient values for each of the wavelet coefficients which have thus far appeared in the file. The idea behind the tables is simple: we always look for our current coefficient in the highest order table; here order2. If we find it, we use this set of probabilities to transmit to the arithmetic encoder. If we do not find the coefficient in this order, we look in the next lower order table; here order1. If we find the coefficient and its associated probability, we transmit the probability, otherwise we are left with one order, order0. If we do not find the symbol here, we use the escape alphabet to generate the probability and add it to the probability table of order0 only. Each time we find a symbol in a particular order, and transmit the probability, we increment the probability count of that symbol to reflect its increased possibility of occurrence the next time around. This adapts the model to the input file.

Note that we only add coefficients to the order0 table (never to order1 or order2). Instead of being added to the order1 and order2 tables, they are moved. Specifically, if the current order is not order2, we check to see what percentage of the total a coefficient's probability occupies in this order. If that percentage is above some constant amount, we move it up to the next order.

In order to demonstrate the advantage of using three probability buckets in each bucket over just using one, we ran a series of tests. Table 5.4 shows the results of five ultrasound files compressed with two versions of MICS. Compression method (1) uses the bucket scheme with three probability tables per bucket; compression method (2) uses one probability table per bucket. All other factors of the algorithm are the same. We include both compression ratio C_r and compression time results from each compression method.

Table 5.4 Method (1) and (2) Compression Ratio and Time Results

Image	Zero-Order Entropy	Method (1) C_R	Method (1) Time	Method (2) C_R	Method (2) Time
UL1	6.33	4.60	**11.85**	4.60	19.27
UL2	5.56	4.04	**12.91**	4.05	20.30
UL3	5.39	3.16	**11.06**	3.16	14.48
UL5	4.98	3.76	**11.85**	3.77	19.27
UL11	1.76	0.73	**6.04**	0.73	9.70
UL12	1.58	0.67	**5.83**	0.68	9.61

The table shows that there is very little difference in the compression ratio obtained by the two methods (the reason for this is explained below). There is however a significant difference in compression time; method (1) is clearly superior.

The philosophy behind method (1) was that the coefficients which occur the most frequently will eventually be moved up to the highest order (order2), and compressed using probability tables more specific to those values. Thus the probability estimate would be more accurate and the compression ratios would improve. We also speculated that the probability tables would remain smaller, and so execution time is decreased. In reality, compression ratios are little affected by using three probability tables instead of one. Most of the advantage of doing so is often cancelled out by the necessity of generating the escape symbol every time we do not find our current symbol in the highest order table. This is the reason we chose to use three buckets; any more would not be practical in terms of compression ratio. (The more tables we have, the more often we generate the escape symbol, because the case where the current probability table does not predict the symbol we are trying to encode occurs more often.) However, the second advantage, namely speed of execution, does come through.

5.3 The Arithmetic Coder

The arithmetic coder used to translate the symbol probabilities generated by the model into bits is borrowed from Mark Nelson [13]. There was one significant change made to it. Specifically, the Nelson coder represented probabilities as frequency counts, which were at a later point translated to probability values by dividing them by total frequency count. Each time a probability table was passed to the coder for encoding, all the frequency counts were scaled to fit one byte. This was done for storage reasons, but had two serious drawbacks. First, the scaling often hurt compression; a fact acknowledged by the author (" ... every time

we lose some accuracy in our counts by scaling, we also lose a certain amount of compression..."). The second drawback was that scaling the counts every time a symbol had to be arithmetic encoded slowed down the speed of execution quite significantly. We changed the implementation so that the scaling was performed only when necessary for the correct operation of the arithmetic coder (i.e. in order to prevent overflow).

5.4 MICS Coding Algorithm

The application of a Multi-Resolution transform produces a matrix of transform coefficients. Before we present the MICS algorithm for coding these coefficients, we define some terms.

Each coefficient c_{ij} (where i, j are the row and column numbers of the matrix respectively) has a parent coefficient *p* in the matrix:

$$c_{ij} = p_{\lfloor i/2 \rfloor \lfloor j/2 \rfloor} ,$$

(with the exception of coefficient at position (0,0).) For a coefficient c_{ij}, we refer to the value of the parent coefficient as Parent(c). The siblings of a coefficient *c* are those coefficients which have a parent in the same position in the matrix of transform coefficients as *c*.

We also define MS(K), where K is some integer, to be the MS value for K defined in Table 5.2.

The model is an array. Each array slot holds three tables: *Order2*, *Order1*, and *Order0*. Each of these tables hold coefficients, along with associated frequency counts, and are responsible for generating predictions (the predictions are approximated by the frequency counts). Each table also holds an additional coefficient, referred to as the *Escape_Symbol,* and an associated count. Each *order* table in the model is initialed to contain only the *Escape_Symbol* and its associated frequency count.

We also have a second array, referred to as the *EscapeAlphabet*, which holds the set of all the coefficients which occur in the input file, along with associated frequency counts.

The algorithm makes use of two constants: *MaxTotal* and *MovingConstant*. Whenever the total frequency count for some table reaches *MaxTotal*, and the current table is not *order3*, we try moving coefficients up to a higher table. For each coefficient, we want to calculate *Percentage*(coefficient), which is its frequency count divided by total frequency count.

Each coefficient for which that value is greater than the *MovingConstant* value is then moved up to the next higher table.

We traverse the coefficients by always scanning the parent coefficients before the children.

MICS Algorithm:

```
for each coefficient

    K = average( Parent(coefficient), siblings)
    call TryEncoding(K, 2, coefficient)

    if coefficient has not been coded then

        Send (EscapeAlphabet, coefficient) pair to Arithmetic Coder
        Add coefficient to Model[ MS(K), 0 ]

    fi
endfor



proc Try_Encoding(K, OrderNumber, coefficient)

    if OrderNumber < 0 return fi

    if Model[ MS(K), OrderNumber ] predicts coefficient then
        Send ( Model[ MS(K), OrderNumber ], coefficient) pair to Arithmetic Coder
        increment frequency count of coeffficent
        if totalFrequencyCount for Model[ MS(K), OrderNumber ] > MaxTotal then
            for each coefficient c in table Model[ MS(K), OrderNumber ]
            if Percentage(coefficient) > MovingConstant and OrderNumber < 2 then
                move coefficient to Model[ MS(K), OrderNumber+1 ]
            fi
        fi
    else
        Send (Model[ MS(K), OrderNumber ], escape_Symbol) pair to Arithmetic Coder
        call Try_Encoding( K, OrderNumber-1, coefficient)
    fi
endproc
```

5.5 Results

We will compare MICS with two other compression systems: SPHINT, and the TIFF compression scheme. Table 5.4 shows the entropy values of 12 compressed files using the three codecs. The second column contains the zero-order entropy of each file, before it is wavelet transformed and compressed.

The best compression ratios appear in bold.

Table 5.5 Results

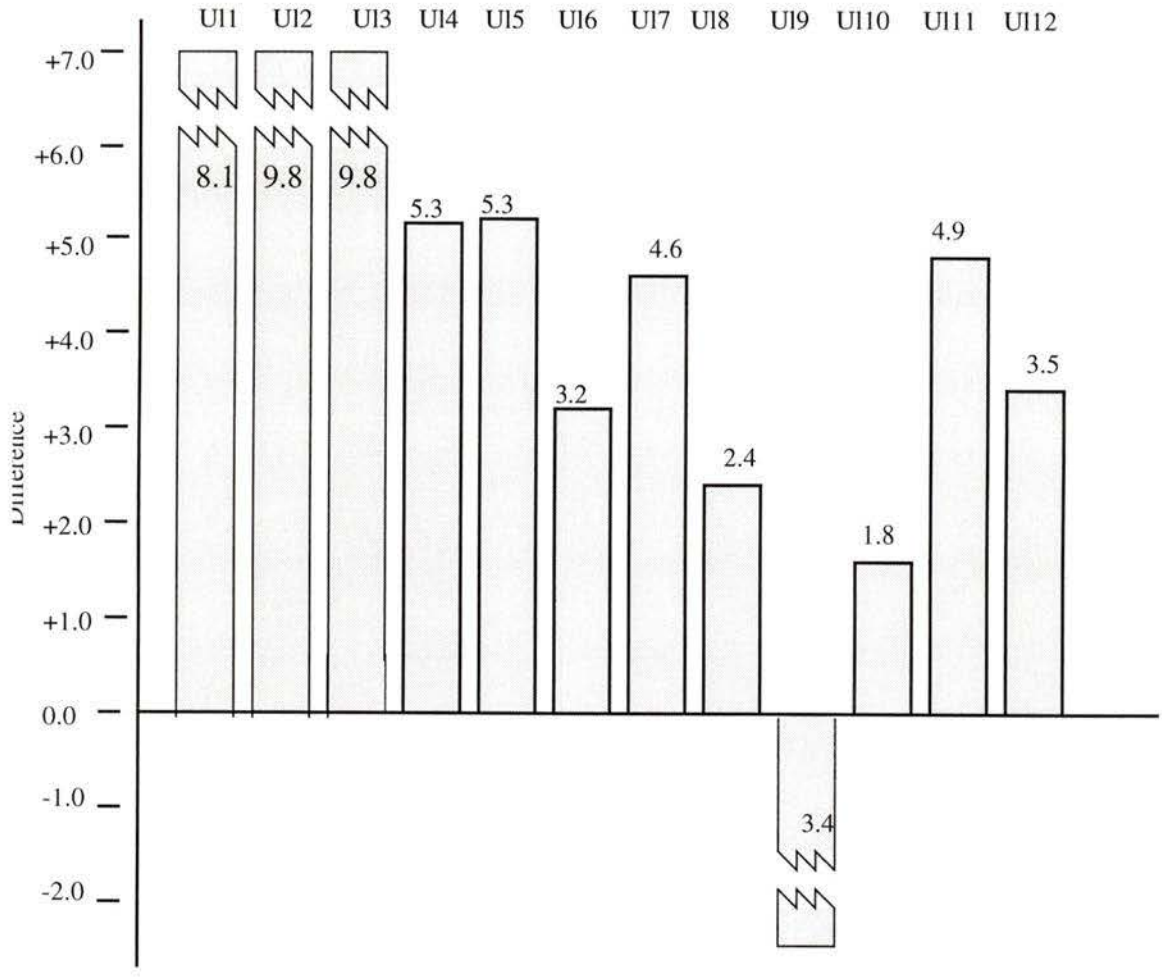
Image	Zero-Order Entropy	MICS	SPIHT	TIFF
UL1	6.33	4.60	**4.58**	5.25
UL2	5.56	**4.04**	4.06	4.80
UL3	5.39	**3.16**	3.22	3.62
UL4	5.39	**3.16**	3.20	3.59
UL5	4.98	**3.76**	3.80	4.25
UL6	3.76	**2.10**	2.19	2.35
UL7	3.18	**1.66**	1.87	2.0 3
UL8	2.69	**1.50**	1.78	1.98
UL9	2.60	1.87	2.06	**1.60**
UL10	2.24	**1.60**	1.93	1.75
UL11	1.76	**0.732**	1.14	1.12
UL12	1.58	**0.671**	1.06	0.95

It is interesting to note that MICS obtains similar results on higher-entropy files (UL1, UL2) to SPIHT, but does significantly better than TIFF on these. On the other hand, its performance is better than that of the SPIHT codec and the TIFF codec on the average entropy files (for example, UL6, U74, UL8). Only in a few select cases is MICS outperformed by TIFF, and the SHIHT coder performs even worse in these cases. These cases are isolated to files having a great deal of "patterns" which are exploited by the dictionary structure of the TIFF coder (See section 3.5.2).

Figure 5.9 shows the differences in compression ratios between TIFF and MICS for the files in Table 5.4 (i.e. (TIFF:compression ratio) - (MICS:compression ratio)). A positive difference indicates that MICS is obtaining better compression ratios than TIFF; a negative difference means that TIFF is outperforming MICS. The corresponding differences are labelled on the graph. The files in the graph are ordered by zero-order entropy; from highest

to lowest. Several of the differences were too high or too low to be shown entirely in the figure. These have the jagged edges as the tops of their graphs to indicate this. In these cases, it is necessary to read the labelled difference.

Figure 5.9 Differences in Compression Ratios Between TIFF and MICS (Bigger is Better)

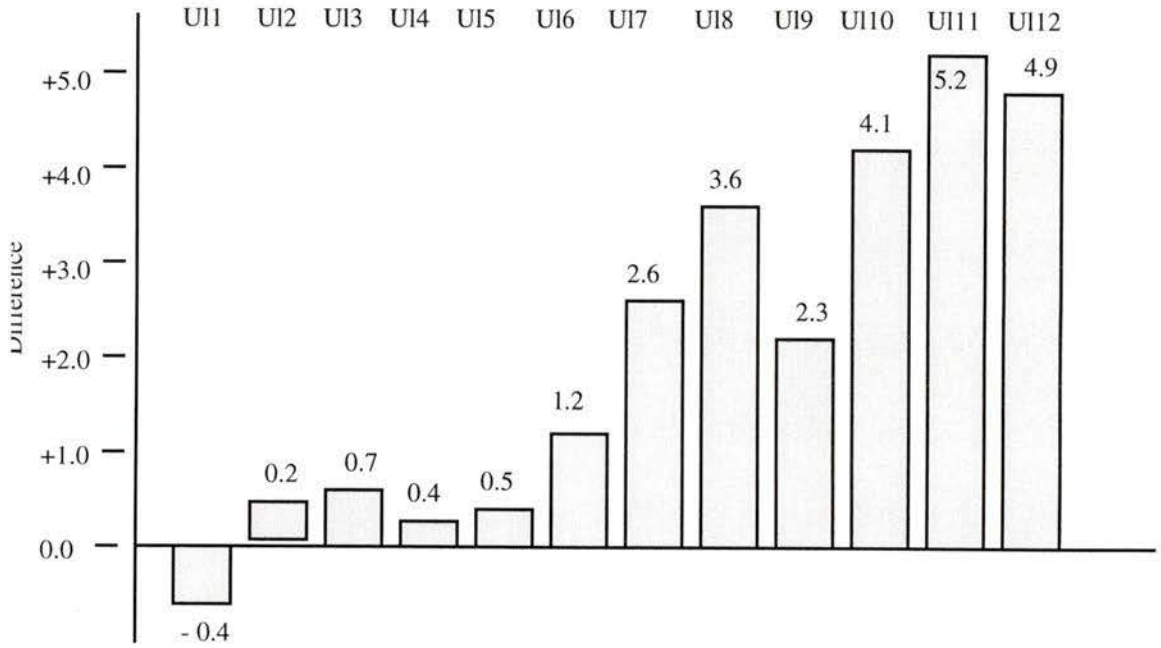


From the figure, it is clear that MICS easily outperforms TIFF on higher entropy files.

Figure 5.10 shows the differences in compression ratios between SPIHT and MICS for the files in Table 5.4 (i.e. (SPIHT:compression ratio) - (MICS:compression ratio)). A positive difference indicates that MICS is obtaining better compression ratios than TIFF; a negative

difference means that TIFF is outperforming MICS. The corresponding differences are labelled on the graph. The files are ordered by entropy; from highest to lowest.

Figure 5.10 Differences in Compression Ratios Between SPIHT and MICS (Bigger is Better)



The figure shows that MICS does about the same (or slightly worse in the case of the UL1 file) than SPIHT on the higher entropy files, and significantly better on the lower entropy ones.

5.6 CPU Requirements

The Tables 5.6 and 5.7 show the compression and decompression times for the three algorithms: MICS, SPIHT and TIFF. These were calculated using the UNIX function *getUsage*.

Table 5.6 Compression Times

Image	MICS	SPIHT	TIFF
UL1	14.64	8.88	3.08
UL2	12.91	8.01	3.49
UL3	11.06	6.92	2.01
UL4	11.09	6.82	1.16
UL5	11.85	7.44	4.15
UL6	9.35	5.93	1.00
UL7	9.50	5.78	0.97
UL8	8.46	5.89	0.86
UL9	8.93	6.02	1.05
UL10	8.92	6.09	0.98
UL11	6.04	4.41	0.63
UL12	5.83	4.35	0.57

Table 5.7 Decompression Times

Image	MICS	SPIHT	TIFF
UL1	12.43	7.63	0.3
UL2	13.31	8.24	0.32
UL3	11.24	6.87	0.23
UL4	11.36	7.04	0.26
UL5	12.43	7.63	0.30
UL6	9.59	6.20	0.24
UL7	9.99	5.74	0.24
UL8	8.54	5.96	0.22
UL9	9.27	6.11	0.17
UL10	9.53	6.67	0.17
UL11	9.36	6.39	0.24
UL12	6.45	4.51	0.12

Both the MICS and SPIHT compressor and decompressor are slower than TIFF. Both of these coders/ decoders use arithmetic coding, while TIFF does not. It has been noted that arithmetic coding is typically slower than dictionary based methods [2]. Decoding for dictionary methods is especially fast, as is evident from Table 5.7. MICS is slower than both

TIFF and SPIHT in both the compression and decompression stages; however, both of these algorithms are commercial ones that have been optimized for speed. MICS on the other hand has not. There are several areas which could be optimized for speed. For example, MICS makes use of the *C* function "realloc", which makes memory allocation simpler, but also slower.

6 Conclusions and Future Work

We have presented a wavelet based codec for medical image compression. There are several contributions of this work.

The wavelet transform library included with the codec is designed to handle all types of images. In the previous wavelet libraries we encountered, images which did not conform to dimensions specified by the library (specifically, that the image dimensions be perfect powers of 2) were padded with values [10], [14]. This impaired compression ability, especially for files with dimensions like 640:480. A number of the image files we dealt with fell into this category. The new library redesigned the wavelet algorithm to avoid padding; a step which decreases output file size significantly in some cases. The forward and reverse transforms are also implemented to be computationally efficient, with the majority of the multiplications/divisions done with bit shifts.

The effects that different filters have on the compression ratios are evident in the results. We have not been able to find a satisfactory "universal" filter that performs well over a range of images. A major advantage of this codec is its ability to pick the 'better' transform at run time.

The model developed uses a different and novel approach, rather than a variation of the popular EZW traversal. It is built adaptively by the codec, and reflects the spatial correlations and frequency distribution of the wavelet transformed data.

It is important to keep in mind that differences in compression ratios between lossless compression systems are typically not very high. The performance of MICS surpasses SPIHT on all but the highest entropy files. In fact, SPIHT does not appear to perform very well on the smoother files; a category under which many of the medical files fall under. MICS also surpasses the current compressor being used, TIFF, in the majority of cases. The only cases where this does not happen is when the input file exhibits much repetition, which is exploited by the LZW dictionary. In these cases, the medical coder SPIHT performs even more poorly than MICS. MICS does significantly better than TIFF on higher entropy files (with an improvement in compression ratio of up to 9.8 percent).

Both TIFF and SPIHT are commercial systems that have patents associated with them (SPIHT is pending), which renders them unavailable for unrestricted use. MICS is freely available.

6.1 Future Work

More research needs to be done in the area of wavelet transforms. Factors affecting which transforms perform better, and why so, should be explored. We believe that a better transform (i.e. one that would reduce the entropy of the original data) would improve compression capabilities of the current system.

The arithmetic coder presently used is not the most efficient one possible. One which is more efficient (both in terms of speed and ability to translate probabilities into codes) would improve the performance of the coder.

The coder currently only accepts grey-scale sun raster files. The ability to read and code different image formats would make the system more versatile. In order to incorporate the capability to code color files, more research would have to be done in the area of color transformations. For example, it is standard to transform the RGB color plane into the YUV color plane. This makes the coding of the color coefficients much more efficient in terms of compression capability. The only drawback of doing so is the introduction of floating point values, which are truncated to integer values in the compressed file. This means that the color transformation is not exactly recoverable. Since the compression must be lossless, this introduces the need to compensate for the lost data in some way. The model would also have to reconstructed. Specifically, the color planes are correlated; a property which should be reflected by the model.

References

[1] Michael F. Barnsley and Alan D. Sloan, "A Better Way to Compress Images", BYTE, January 1988, 215-224.

[2] Timothy Bell, John Cleary and Ian Witten, *Text Compression*, Prentice Hall, New Jersey, 1990.

[3] R. J. Clark, *Transform Coding Of Images*, Academic Press, 1985.

[4] Oliver Egger and Wei Li, "Subband Coding of Images Using Asymmetrical Filter Banks", IEEE Transactions on Image Processing, Vol. 4, no. 4, April 1995, 478-485.

[5] Michael P. Ekstrom, *Digital Image Processing Techniques*, Academic Press, 1984.

[6] Alan Fournier (Organizer), "Wavelets and their Applications in Computer Graphics", SIGGRAPH '95 Course Notes.

[7] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Addison-Wesley, 1993.

[8] Robert Gray, "Vector Quantization", IEEE ASSP Magazine, April 1994, 4-29.

[9] Timo Kaukoranta, Pasi Franti and Olli Nevalainen, "Empirical Study on Subjective Quality Evaluation of Compressed Images", University of Turku Research Report.

[10] Bob Lewis, "The UBC Imager Wavelet Package Release 2.1", May, 1995.

[11] J. Makhoul, S. Roucos and H. Gish, "Vector Quantization in Speech Coding", Proceedings of the IEEE, Vol. 73, November 1985, 1551-1588.

[12] Colm Mulcahy, "Plotting and Scheming with Wavelets", Mathematics Magazine, Vol. 69, no. 5, December 1996, 323-343.

[13] Mark Nelson, *The Data Compression Book*, M&T Publishing, New York, 1992.

[14] A. Said and W. Pearlman, "A New and Efficient Image Codec Based on Set Partitioning in Hierarchial Trees", IEEE Transactions on Circuits and Systems for Video Technology, Vol6, 243-250, June 1996.

[15] A. Said and W. Pearlman, "An Image Multi-Resolution Representation for Lossless and Lossy Compression", IEEE Transactions on Image Processing, Vol. 5, 1303-1310, Sept. 1996.

[16] Ken Savage, "The Predictive Transform", presonal communication, May, 1997.

[17] Jerome Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coeffi-cients", IEEE Transactions on Signal Processing, Vol. 41, Dec 1993, 3445-3462.

[18] Gilbert Strang and Truong Nguyen, *Wavelets and Filter Banks,* Wellesley-Cambridge Press, 1996.

[19] John W.Woods, *Subband Image Coding*, Kluwer Academic Publishers, 1991.

[20] Hohn D. Villasenor, Benjamin Belzer and Judy Liao, "Wavelet Filter Evaluation for Image Compression", IEEE Transactions on Image Processing, Vol. 4, no. 8, August 1995, 1053-1060.

[21] G. K. Wallace, "The JPEG Still Picture Compression Standard", Communications of the ACM, Vol. 34, no 4. April 1991, 30- 44.

[22] Ahmad Zandi, James D. Allen, Edward L. Schwartz and Martin Boliek, "CREW: Compression with Reversible Embedded Wavelets", Proceedings of the Data Compression Conference, Snowbird, Utah, March 1995, IEEE Computer Society Press, 212-321.

Vita

Surname: Muldner Given Names: Katarzyna Ania

Place of Birth: Warsaw, Poland

Education Institutions Attended:

University of Victoria 1995 - 1997

Acadia University 1991-1995

Degrees Awarded:

B. Sc. (Honors) Acadia University 1995

Honors and Awards:

The ASI Scholarship 1995

President's SCholarship 1995, 1996

NSERC PGS A Research Award 1995-1997

Computer Science Acadia University gold medal 1994

NSERC Undergraduate Student Research Award

Sandra Pineo Memorial Scholarship in Computer Science, 1993-1994

Digital Equipment of Canada Limited Award of Merit, 1993-1994

Hazel Duncanson Vaughan Memorial Scholarship in Art, 1992-1993

Carl J. Sanders Memorial Scholarship, 1991-1992

Publications:

T. Muldner, C. VanVeen, K. Muldner, "Experience from the Design of an Authoring Environment", Journal of Educational Multimedia and Hypermedia (JEHM).Volume 6, Number 1, 1997.

T. Muldner, K. Muldner, "Computer Based Presentations to teach Computer Science: Past and Current Experience", Accepted for WCCE'97

T. Muldner, C. VanVeen, K. Muldner, "The Design and Evolution of an Authoring Environment and its Applications", EDMEDIA'96, Boston, June, 1996

Partial Copyright Notice

I hereby grant the right to lend my thesis (or dissertation) to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of thisthesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my writtTitle of Thesis/Dissertation:

Title of Thesis:

Transform-Based Medical Image Compression

Author 

Katarzyna Ania Muldner

October 6, 1997