University of Alberta

Improved Spatial and Transform Domain Compression Schemes

by

Jason Alan Knipe

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

Department of Computing Science

Edmonton, Alberta
Fall 1996

ISBN 0-612-18285-1

University of Alberta

Library Release Form

Name of Author: Jason Alan Knipe

Title of Thesis: Improved Spatial and Transform Domain Compression Schemes

Degree: Master of Science

Year this Degree Granted: 1996

Jason Knipe

Jason Alan Knipe
Box 181
Bawlf, Alberta
Canada, T0B 0J0

Date: August 6, 1996

The hardest thing in the world to understand is the income tax.
Albert Einstein

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Improved Spatial and Transform Domain Compression Schemes** submitted by Jason Alan Knipe in partial fulfillment of the requirements for the degree of Masters of Science.

Xiaobo Li
. . . .
Dr. X. Li (Supervisor)

. . . .
Dr. N. Durdle (External)

. . . .
Dr. J. Buchanan

Date: 96/07/26

to my family

Abstract

Many lossy compression algorithms have been proposed that perform well for high compression ratios, but are computaticnally intense. On the other hand, there are many simple algorithms that perform poorly at high compression ratios. An algorithm that incorporates both computational simplicity and acceptable performance at low bit rates would have many potential applications. In wireless communications, for example, where computing power and transmission speed are important factors, the time required to compress and send image data would need to be reduced significantly. Two new methods are proposed in this thesis, one spatial domain based and the other transform domain based, which make improvements over existing compression technology in their respective domains. Both of the new schemes can be shown to have advantages over the methods upon which they were based, and both are theoretically simple computationally, making them ideal for use in many circumstances.

Acknowledgements

I would like to thank Dr. Xiaobo Li for his help and guidance over the course of my research, and for helping to make this project a very enjoyable one for me.

Additional thanks are owed to the members of the research team who provided me with ideas and encouragement: Nicholas Reeder for the tension breaking squash games and help with the vectors, Howard Cheng for the insight and ideas, and Bin Han for his help in understanding wavelets.

Special thanks to my fiancee Nicole, who helped me by making me take breaks when I needed them, listened to me complain when I was stuck, and played games of network Warcraft with me to cheer me up when I was feeling down.

This research is supported in part by Motorola Wireless Data Group and the Canadian Natural Sciences and Engineering Research Council under Grant OGP9198.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The compression of digitized images is a prominent part of current computing science. Modern computing power and storage has reached the point where the displaying of large, detailed images is not only possible, but in high demand for almost any conceivable application. One need look no further than the increasing popularity of the world wide web to realize that the storage and transmission of images is and will continue to be an important aspect of computing.

Image compression can take two forms: lossless compression, where the compression and decompression process results in a reconstructed image which is identical to the original, and lossy compression, where the reconstructed image differs from the original. In general, lossless image compression will not result in large compression ratios. Lossy image compression, on the other hand, can result in very impressive compression, often with very little loss to the reconstructed image. Depending on the application a lossy compression algorithm may be more desirable than lossless for this reason.

1.1 Redundancy in Images

Any compression scheme, whether it is specific to images or some other form of data, relies on redundancies in the data in order to achieve compression. A completely random data stream will not contain many redundancies, and thus will not compress well. A digitized image, which we can generally assume will be representative of some visual phenomenon and not random intensities, will contain much redundancy. In image compression, there are three basic types of redundancy that need to be considered in order to effectively reduce the space required by an image: coding redundancy, interpixel redundancy, and psychovisual redundancy.

1.1.1 Coding Redundancy

When symbols from a data stream are being coded, certain symbols will occur with higher or lower frequency than others, assuming that the data stream is not random. For a given data stream with N symbols, where each symbol r_k occurs n_k times, we can calculate the probability of each symbol occurring as:

$$p(r_k) = \frac{n_k}{N}, \qquad k = 0, 1, \ldots, N-1 \tag{1.1}$$

In order to minimize the space required in the coding of the symbols, we must minimize the equation:

$$L_{avg} = \sum_{k=0}^{N-1} l(r_k)p(r_k) \tag{1.2}$$

where $l(r_k)$ is the number of bits used to store symbol r_k.

In the general case, using a non-variable coding scheme to code the symbols (m bits per symbol, where $m = \lceil \log_2 N \rceil$) will not minimize the equation. Clearly, less bits need to be allocated to more common symbols, which implies that more bits will be needed for less common symbols.

Using this principle, the *Huffman coding* scheme generates a variable length code which effectively minimizes equation 1.2 for a fixed value of N, subject to the constraint that the symbols are coded one at a time[9, 13]. The constraint allows for the fact that there can be advantages to coding symbols in groups, which may lead to further redundancies in certain situations.

Another popular coding scheme, *arithmetic coding*[9, 1, 33], does not operate on the same principle of mapping each input symbol to a code symbol on a one to one basis, as is done by Huffman coding. Rather, the entire data stream is converted into a single arithmetic codeword. Theoretically, arithmetic coding can code an input stream in the lowest possible number of symbols as dictated by information theory.

1.1.2 Interpixel Redundancy

Interpixel redundancy is based on the observation that in an image there exist similarities between the intensity values of neighboring pixels. This type of redundancy in two dimensions is one of the major differences in the compression of image data and one dimensional data streams.

One of the simplest methods for eliminating interpixel redundancy is to use what is known as *predictive coding*. In this scheme, each pixel value is "predicted" from the values of a neighboring pixels in such a way that only the difference between the predicted and actual value need be stored. Because of interpixel redundancy, the difference values are generally small, thus creating a low entropy source which will be more effectively coded.

Other means of reducing interpixel redundancy include transform coding, which is frequency based rather than spatially based like the predictive coding method. Instead of considering the direct redundancies between the pixel intensities themselves, which is done in spatial domain methods, transform coding converts the image into a set of frequency coefficients using a reversible transform. The advantage of such transforms is the generation of transform coefficients which are inherently easy to quantize (resulting in lossy compression). The discrete cosine transform (DCT) used in JPEG compression[20, 34] is an example of transform coding.

1.1.3 Psychovisual Redundancy

A final type of redundancy is that which is observed by the human eye. In an image, certain pieces of visual information may be more or less important than others.

Psychovisual redundancies are often eliminated in a compression scheme by the quantization of pixel or transform values. By selectively quantizing pixel values in image regions with little detail, for example, the amount of space required to store an image can be reduced without losing any important information. Often the results of quantization are not apparent to the human eye except on close inspection.

1.2 An Image Compression Model

A lossy image compression scheme will generally have components which deal with each of the three types of image redundancies. A generic lossy image compression model will consist of the following components:

- *Mapper*: deals with interpixel redundancies by transforming the image into a nonvisual form.
- *Quantizer*: eliminates psychovisual redundancies by quantizing the output from the mapper. The mapper and quantizer can often be thought of as a single block.
- *Symbol Encoder*: handles coding redundancy by assigning a code (generally variable length) to the output from the mapper/quantizer.

Similarly, the decompression model would consist of the following:

- *Symbol Decoder*: decodes the symbols which have been coded by the symbol coder.
- *Inverse Mapper*: converts the decoded nonvisual information back to visual information.

The quantizer module will, of course, be the only module which will not be present in any lossless compression scheme. Figures 1.1 and 1.2 show the encoding and decoding process as described in[9].

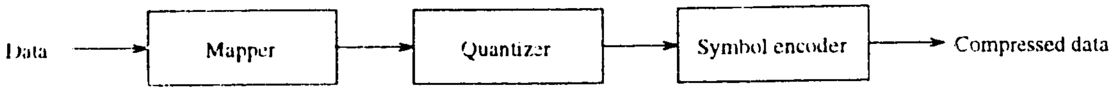


Figure 1.1: Source encoder

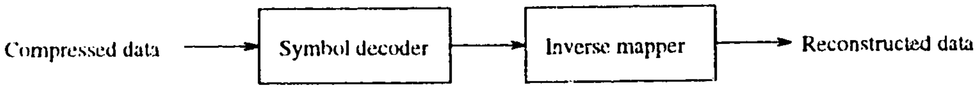


Figure 1.2: Source decoder

1.3 New Approaches to Image Compression

In this thesis two compression methods will be presented which often result in significant improvements over previous methods. Both new algorithms are based on previously existing schemes, but have been modified from their original form resulting in some of the following improvements:

- Improved image quality, as measured mathematically.
- Improved visual image quality. It will be shown that visual quality and mathematical quality (different measures of reconstructed image error) do not necessarily correlate.
- Minimized computational complexity

The first of the proposed schemes is based on the spatial domain quadtree compression method developed by Shusterman and Feder[28]. By making what would seem to be very simple and almost obvious modifications to the reconstruction process, the image quality is significantly improved, and at the same time the algorithm complexity is reduced.

The second proposed scheme is a transform domain method based on a variation of the wavelet transform method developed by J. Shapiro[27]. By adding vector quantization and several optimization routines, the quality of the reconstructed images can increase for busy images (images containing an abundance of detail), at low cost to the computational complexity.

Many publications describe compression results in terms of very few images and bit rates. By doing so, the reader is often not necessarily convinced of the algorithm's success. The methods presented herein, however, have been tested on a large image set which includes a variety of images and different image types. Consequently, the effectiveness of the new schemes is more clear.

1.4 Outline of Thesis

This thesis is devoted to two new lossy compression schemes, one spatial domain based, the other transform domain based. Chapter 2 is devoted to the first, which is based on quadtree compression similar to that described by Shusterman and Feder[28]. Chapter 3 explores a wavelet compression algorithm based on the Embedded Zerotree Wavelets (EZW)[27] and SPIHT[21] algorithms. Chapter 4 summarizes the findings, comparing the execution and results of the two algorithms.

Chapter 2

Quadtree Compression

2.1 Introduction

Many algorithms have been developed which perform well for low bit rate applications (less than 0.3 bits per pixel). One need look no further than Model-based image coding [2] or any recent wavelet transform methods [27, 24] to see that such algorithms exist and give good results. However, the higher conceptual and computational complexity of such algorithms encourages the further examination of some of the simpler spatial domain compression methods. Of these, one of the most common is the quadtree decomposition method.

Conceptually, quadtrees may be one of the simplest means of compression available. It is this simplicity, as well as the potential for relatively good performance for low bit rates that make quadtrees a desirable choice for a compression scheme. Quadtrees are used for a variety of applications. Many papers have been written on their usefulness, some of which explore various properties and applications of quadtrees and quadtree-like structures [14, 32, 23, 35, 11, 18], while others concentrate on efficiency and different methods of storage [15, 7, 8, 22, 10, 26, 12]. The quadtree methods that are of particular relevance to this research are those which deal directly with the compression of grayscale images [28, 29, 30, 31].

Strobach describes a very successful compression method based on the quadtree[30] which is based on "describing" quadtree leaves in terms of a linear model: $f(x,y) = a + bx + cy$. Given the plane parameters $[a, b, c]$ for a block of size N x N, it is possible to efficiently compute the same parameters for a block of size $2N$ x $2N$. This method reports a PSNR (Peak Signal to Noise Ratio) of greater than 32dB at 0.5 bpp, and is relatively simple computationally when compared to other current methods.

Another method, developed by Shusterman and Feder[28], was particularly appealing because of the apparent ease with which it could be modified to test possible combinations with other compression methods, as well as its apparent computational simplicity. In this chapter, we will outline some improvements made to this algorithm, analyze its complexity, and make comparisons to other methods. Much of the content will be devoted to the discussion of the new algorithm and differences between it and that of Shusterman and Feder.

2.2 The Quadtree

The concept of a quadtree is quite simple. It involves the construction of a quadtree structure (each node has either no children or four children) in which each leaf node represents a homogeneous region. The definition of "homogeneous" in this context depends upon the type of compression which is desired. In the lossless compression of a binary image, for instance, a homogeneous region would be defined as a region that is either all black or all white.

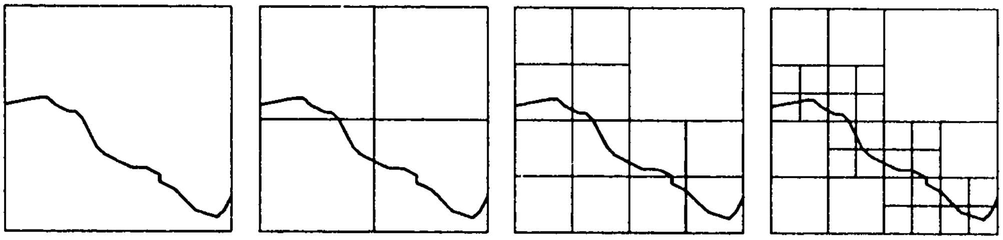


Figure 2.1: Top-down quadtree decomposition

There are two ways in which quadtree decomposition can be done: top-down or bottom-up. In the top-down procedure each sub-image (beginning with the entire

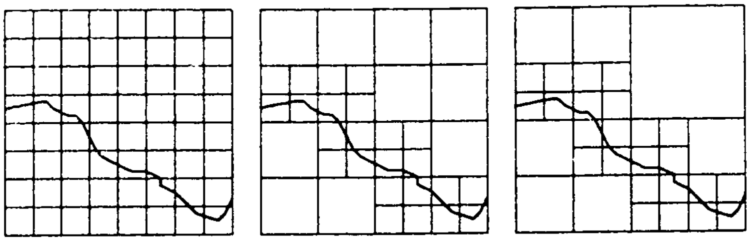


Figure 2.2: Bottom-up quadtree decomposition

image) is examined to determine if it meets whatever homogeneity criterion is appropriate. If the criterion is not met, four new nodes (each representing one quadrant of the sub-image) are created. Otherwise, the node representing the sub-image remains as a leaf containing information about the region it represents. The bottom-up procedure considers first the pixel level of the image, which requires that a complete quadtree structure exist. If any four "neighbor" pixels meet an appropriate merging criterion, their corresponding nodes in the quadtree are eliminated, and a resulting average value is stored in the parent node which becomes a leaf. It is well known and easy to see that the bottom-up procedure is superior. Figures 2.1 and 2.2 illustrate the two alternatives.

The actual coding of the tree can be done by storing the quadtree structure as a string of 0's and 1's, and then storing separately the values of the leaves. The quadtree structures from the previous two figures is as follows:

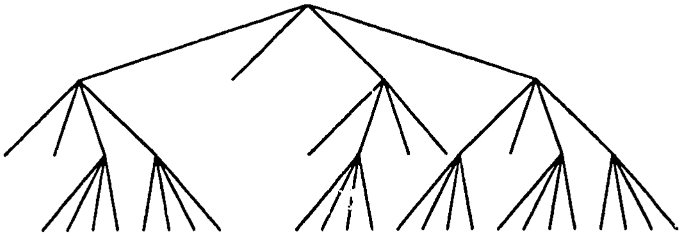


Figure 2.3: Quadtree Structure

The branches in the above figure (from left to right) indicate the north-west, north-east, south-east and south-west children respectively. The actual order is unimportant. The resulting structure code, if we use NW-NE-SE-SW DFS traversal using a 1 to represent a node and a 0 for a leaf would be: 11001101010011011. A breadth first traversal would yield the code: 1-1011-0011 0100 1011. Any method of storage which allows the quadtree structure to be reconstructed is acceptable. Note that the bottom level need not be stored since everything at the pixel level is implicitly a leaf.

2.3 Shusterman and Feder's Algorithm

Since the proposed algorithm is based primarily on that presented by Shusterman and Feder[28], we will briefly outline their basic compression algorithm in this section.

Before proceeding to the algorithm itself, the notation that will be used to describe the quadtree structure will be explained. First of all, in a quadtree representing an $N * N$ image (where $N = 2^n$)[1], the tree itself will contain $n + 1$ levels, which will be numbered starting from the bottom. The quadtree elements will be referred to in the following manner: $x_i(j, k)$ will represent the quadtree node at level i with coordinates (j, k). The children of such a node (if it is not at the bottom level of the tree), in the order NW-NE-SE-SW, would be: $x_{i-1}(2j, 2k)$, $x_{i-1}(2j + 1, 2k)$, $x_{i-1}(2j + 1, 2k + 1)$ and $x_{i-1}(2j, 2k + 1)$.

2.3.1 Quadtree Decomposition and Bit Allocation

The "value" of any internal node in the tree is calculated to be the average of the values of its children. That is, for all levels except the lowest level (which contains the values of the image pixels):

$$x_i(j, k) = \frac{1}{4} \sum_{l=0}^{1} \sum_{m=0}^{1} x_{i-1}(2j + l, 2k + m) \qquad (2.1)$$

Note that the division at this point can be done as a bit shift operation.

The basic algorithm is based on the idea that the merging of four siblings occurs when their values differ from their parent's value by less than a certain threshold T,

[1] Images of any dimension can be compressed using this method and that presented in the next chapter, but must be padded in some way to fit the dimension requirement.

which will hereafter be referred to as the **decomposition threshold**. Thus, we only merge siblings if the following condition is true:

$$\bigcap_{l,m=0}^{1} |x_i(j,k) - x_{i-1}(2j+l, 2k+m)| \leq T \tag{2.2}$$

The value of the threshold, however, is not static throughout the quadtree. Because merging siblings at higher levels in the tree can potentially cause greater distortion in the resulting image as a consequence of the increasing size of the area represented by higher level nodes, it seems reasonable that the threshold should become smaller at higher levels. An optimal solution described by Shusterman and Feder [28] is very expensive, but an acceptable suboptimal solution is presented, in which an **initial decomposition threshold** T_1 is given for the first level, and the thresholds at the other levels can be calculated by:

$$2 \leq i \leq n \qquad T_i = \frac{T_1}{2^{i-1}} \tag{2.3}$$

Thus, the decomposition threshold decreases by a factor of two each time the level is incremented, and formula (2) becomes:

$$\bigcap_{l,m=0}^{1} |x_i(j,k) - x_{i-1}(2j+l, 2k+m)| \leq T_i \tag{2.4}$$

The bit allocation is also found to vary by level. The near optimal bit allocation for level i, B_i, is found using the formula[28]

$$B_i = \frac{1}{2}\log\frac{\sigma_i^2 L_{qt}}{4^{n-i}D} \tag{2.5}$$

where σ_i^2 is the variance of the leaf values at level i, L_{qt} is the leaf count of the quadtree, and D is a distortion constant. The larger the value of D, the fewer bits will be used at each level to represent the leaf values.

The compression algorithm[28] using the above information is:

Step 1: Choose T_1. Let $i = 1; N = 2^{n-1}$;

```
Step 2: For $j, k = 0, \ldots, N - 1$
          IF for $l, m = 0, 1$ all $x_{i-1}(2j + l, 2k + m)$ are leaves
            calculate $x_i$ according to (2.1)
            perform the test according to (2.4)
            IF the test is true, $x_i(j, k)$ is a leaf.
            ELSE $x_i(j, k)$ is a node.
Step 3: IF no leaves were produced by Step 2 goto Step 4,
          ELSE $N = \frac{N}{2}; i = i + 1; T_i = \frac{T_{i-1}}{2}$; goto Step 2.
Step 4: Code the tree structure information.
Step 5: Calculate $L_{qt}$; Choose a desired distortion level $D$; $i = 0$;
Step 6: Calculate leaves variance $\sigma_i^2$.
Step 7: Allocate bits for level $i$ leaves according to (2.5).
Step 8: Quantize level $i$ leaves.
Step 9: $i = i + 1$; IF $i > n$ STOP.
          ELSE goto Step 6.
```

Using this algorithm, the size of the tree structure is relatively small when compared to the size of the pixel data. Table 2.1 shows the averages of several test runs, varying the bit rate[2].

Bits per Pixel	Structure Size (%)
1.0	17
0.5	21
0.2	22

Table 2.1: Quadtree structure size as a percentage of total compressed data size

[2] Here as well as in the rest of the thesis, it is assumed that the original images require 8 bits per pixel in uncompressed form

2.3.2 Image Reconstruction

Once the quadtree is reconstructed from the coded data it becomes a simple matter to convert the tree back into image form. However, unless some form of smoothing is done the resulting image will be blocky, as each leaf in the quadtree will represent a square region with a single intensity value. The use of a reconstruction filter which propagates the values of the quadtree leaves down to the pixel level in a manner which reduces blockiness is described by Strobach[29]. Shusterman and Feder[28] recommend the use of this reconstruction method. The filter itself is the following 2-d mask:

$$F = \begin{pmatrix} 0.0285 & 0.1519 & -0.0285 \\ 0.1519 & 0.9787 & -0.1357 \\ -0.0285 & -0.1357 & 0.0159 \end{pmatrix} \tag{2.6}$$

The reconstruction of leaf values at level $i-1$ from level i is then done as follows:

$$X_i(j,k) = \begin{pmatrix} x_i(j-1,k-1) & x_i(j,k-1) & x_i(j+1)(k-1) \\ x_i(j-1,k) & x_i(j,k) & x_i(j+1)(k) \\ x_i(j-i,k+1) & x_i(j,k+1) & x_i(j+1)(k+1) \end{pmatrix} \tag{2.7}$$

$$R_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, R_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

for $l, m = 0, 1$

$$x_{i-1}(2j+l, 2k+m) = R_l F R_m X_i(j,k) \tag{2.8}$$

What the above essentially amounts to is placing a (rotated) version of the filter F over the leaf value at $x_i(j,k)$ and its 8-neighbors, and calculating the "value" of its children. Suppose we have the following values in the X matrix (2.7) for some values of i, j, k:

$$X = \begin{pmatrix} 200 & 195 & 195 \\ 190 & 185 & 160 \\ 190 & 80 & 50 \end{pmatrix} \qquad (2.9)$$

The resulting values for the four children of the leaf $x_i(j,k)$will be:

$$\begin{pmatrix} 202 & 200 \\ 175 & 158 \end{pmatrix} \qquad (2.10)$$

The numbers which have been calculated above illustrate what was consider to be one of the problems with this reconstruction filter. For instance, the value of $x_{i-1}(2j,2k)$ is 202. It seems reasonable, however, that the value should be somewhere in the range of (185, ..., 200) as indicated by the values in the upper left portion of the X matrix (2.9).

The filter given by Strobach[29] seems to over compensate edges, making the bright side of an edge brighter than it should be and the dark side darker than it should be, creating ghost edges that disappear a few pixels away from the actual edge. In a relatively smooth area, it performs well. It would seem that the problem is caused by an assumption that the filter must be 3x3 and include the values of neighbors that perhaps should not influence the reconstructed value of the child.

2.4 Modifications and Improvements

The reconstruction phase of the algorithm is changed significantly from Shusterman and Feder's method. Different sized filters as well as the introduction of a `reconstruction threshold`, which removes the smoothing of sharp edges, have contributed to give better results to the output. This section will be devoted to explaining these changes, and documenting the improvements that result.

2.4.1 An Optimal 2x2 Filter

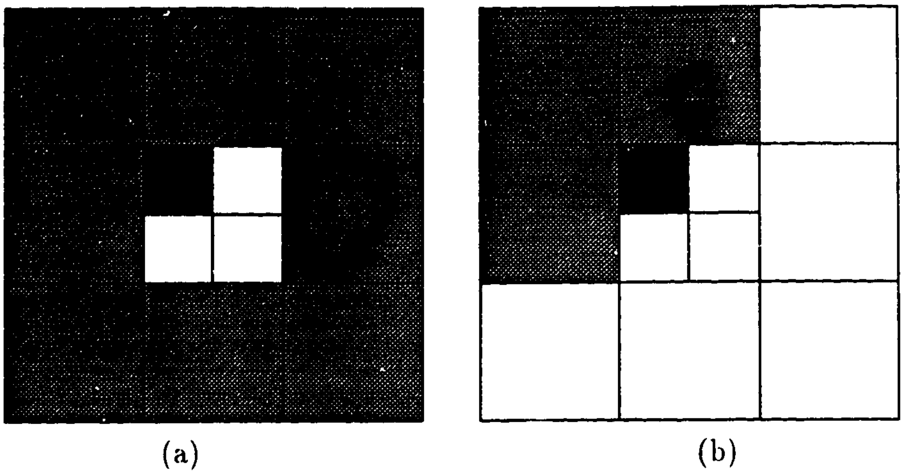


Figure 2.4: An illustration of 3x3 and 2x2 filter reconstruction. The black box shows the child leaf which is to be reconstructed, and the grey shaded boxes show the neighbor leaves whose values will influence the final value of the child. (a) 3x3 filter; (b) 2x2 filter

Consider Figure 2.4. It should be clear that generating an intensity value for a child leaf by using the intensity values of neighbor leaves which do not border on the child does not make intuitive sense. Instead of using a 3x3 filter in this way, a 2x2 filter seems to be a more reasonable choice. A a 2x2 filter which considers only those neighbors at level i that border on the child to be reconstructed at level $i - 1$ may outperform a 3x3 filter, and be simpler computationally.

The process of finding an "optimal" filter that minimized MSE (Mean Squared Error)[3] as was done by Strobach to calculate the values of his 3x3 mask, was designed to generate a different filter for each level of the quadtree. The results of running the search process, which attempted to find 2x2 filter values that minimized the MSE on a set of 15 different images, showed that there were significant differences in the "optimum" filter values at the different levels. Thus, a search was made for each quadtree level, and the result was that improvements were made in the MSE values

[3] MSE=$\frac{1}{N}\sum_{i=1}^{N}(c_i - \hat{c}_i)^2$, where c and $\hat{c}$ are the original and quantized signals respectively

of the resulting images. The values of the filters are shown below:

$$F_1 = \begin{pmatrix} 0.0133 & 0.0887 \\ 0.0887 & 0.8093 \end{pmatrix} F_2 = \begin{pmatrix} 0.0040 & 0.0784 \\ 0.0784 & 0.8391 \end{pmatrix}$$

$$F_3 = \begin{pmatrix} 0.0186 & 0.0375 \\ 0.0375 & 0.9064 \end{pmatrix} F_{4+} = \begin{pmatrix} 0.0000 & 0.0000 \\ 0.0000 & 1.0000 \end{pmatrix}$$

Where F_i is the filter to be used at level i to generate values at level $i-1$. For any level i such that $i \geq 4$, the filter F_{4+} is used, as it was found to be the case that the MSE was minimized using this filter at all levels above 3.

Thus, the reconstruction of the leaves would be computed as before (2.8), but with the reconstruction filter itself taking a different form:

$$F_i = \begin{pmatrix} A1_i & A2_i & 0 \\ A2_i & A3_i & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.11}$$

Note that because of the reduced number of coefficients in the filter, the computational complexity of the algoritl.m is reduced.

The equation for the reconstruction process then becomes:

$$x_{i-1}(2j+l, 2k+m) = R_l F_i R_m X_i(j,k) \tag{2.12}$$

The results of using this filter as opposed to Strobach's can be seen in the images shown in Figure 2.5, which use the same initial values of T_1 and D, with the resulting images being compressed to about 0.42 bpp. The original image is *Lena* (see Figure 2.9), which has been cropped to show more detail. The PSNR of the image shown in Figure 2.5(a) is 31.02, and that of the image in Figure 2.5(b) is 31.77. These results differ from those given in a paper accepted for publication at ICPR'96[16] which I co-authored, since a different version of the *Lena* test image was used. The PSNR is calculated according to the following formula [30]:

$$PSNR = 10 \log \frac{255^2}{MSE} dB \tag{2.13}$$

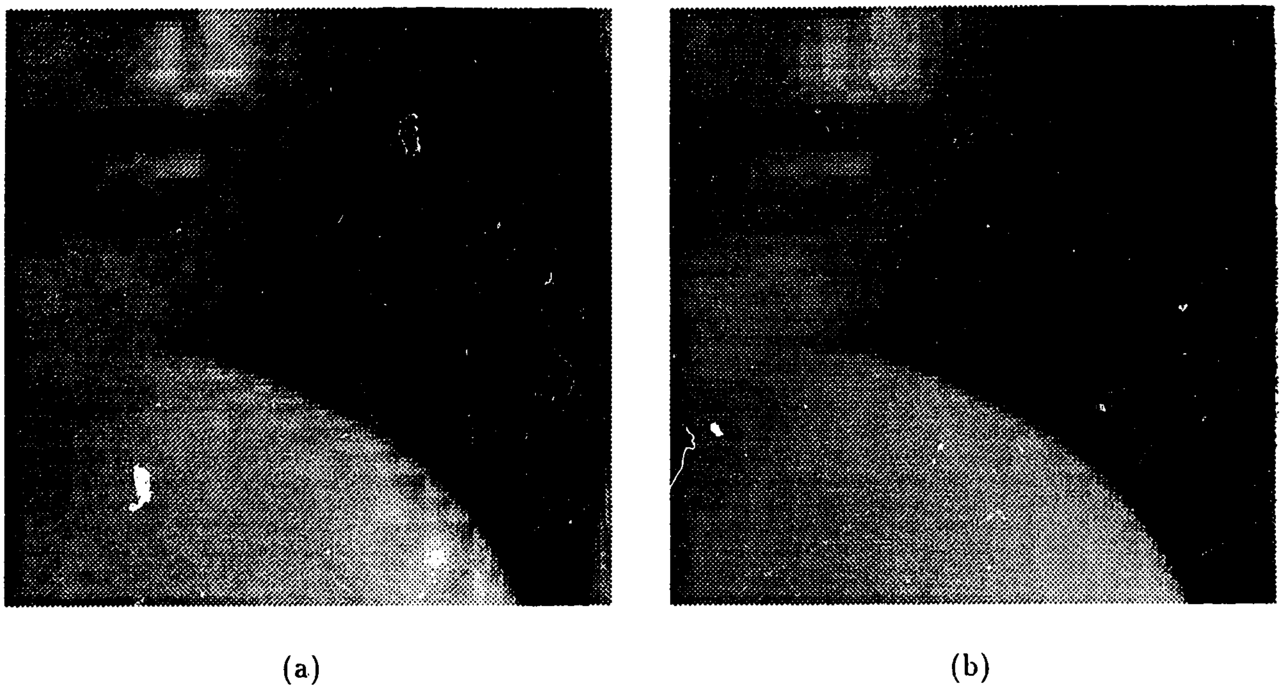


Figure 2.5: Comparison of Methods: (a) results using Strobach's single filter; (b) results using multiple filters.

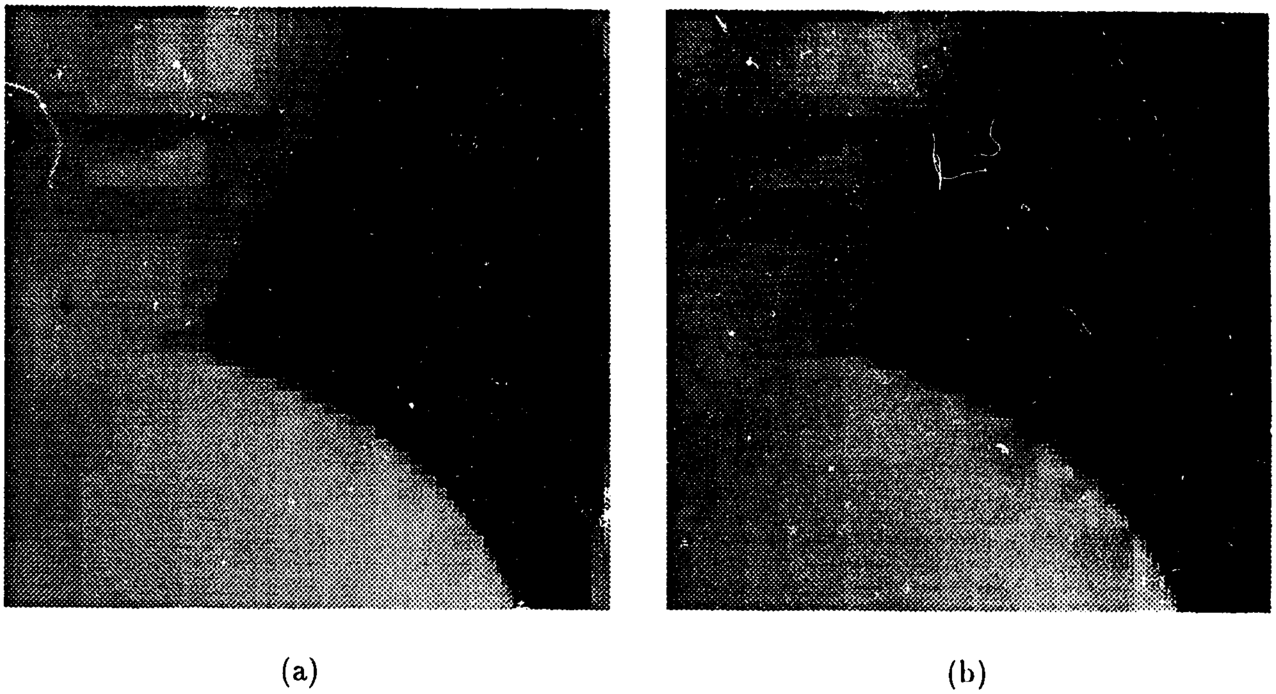
A higher PSNR value indicates a smaller mean squared error, and generally (although not always) a better visual quality image.

Aside from the increased PSNR in the right image, there are some differences worth noting. First, the right image appears to be more blocky. This is caused by the final filter F_{4+} which does not contribute to the smoothing of the final image. Second, the left image seems to contain "ghost" edges beside each sharp edge in the image, which would seem to be similar in nature to the results seen in the previous example (2.10). It would appear that a smooth gradient, present in the left image is sacrificed for better edge handling in the right image.

2.4.2 Smoothing Alternatives

Because of the potential blockiness caused by the use of multiple filters (which decrease the MSE of the image), it may be desirable to use a single smoothing filter in place of the multiple filters. While such filters do not in general generate better MSE results than multiple filters, they are often able to smooth out all of the resulting blockiness. A filter which seems to be well suited to this purpose (but not necessarily to the optimization of MSE) is:

$$F' = \begin{pmatrix} 0.00 & 0.13 & 0 \\ 0.13 & 0.74 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$


(a)

(b)

Figure 2.6: A comparison of the two new filter schemes, using the Lena image compressed to 0.2 bpp: (a) result of using the multiple filters, PSNR=29.17; (b) result of using the new single filter, PSNR=28.63.

Consider the images in Figure 2.6, both compressed to approximately 0.2 bpp. Figure 2.6(a) has a PSNR of 29.17, Figure 2.6(b) 28.63. Note the relative smoothness which results from the use of the single filter.

It should be noted that Figure 2.6(b) contains some noticeable "bleeding" artifacts in the shoulder area which are a result of the reconstruction process' inability to distinguish between regions. This shortcoming will be discussed and eliminated in the following section.

2.4.3 Reconstruction Threshold

A problem with the reconstruction algorithm is that it ignores edges. When an edge is encountered in an image, the reconstruction process will attempt to smooth the areas around it, which may lead to "bleeding" of dark regions onto light regions, and vice versa. A simple solution is to introduce a *reconstruction threshold*, which prevents this from occurring. When considering whether or not to use the value of a leaf's neighbor, the difference between the two would be compared to the reconstruction threshold. If the value is within the threshold, we proceed with the reconstruction as previously described. Otherwise, we ignore the value of the neighbor.

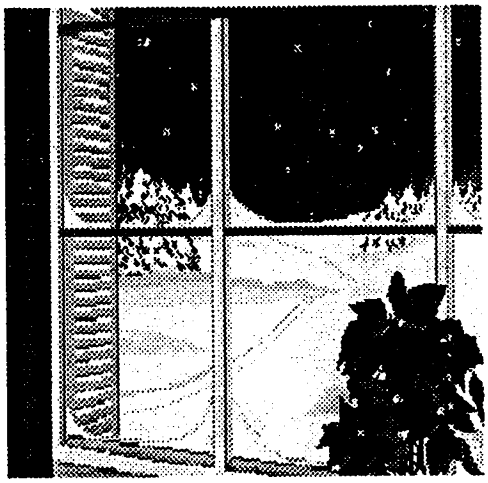


Figure 2.7: Window Image

Figure 2.7 is used as an example for the reconstruction threshold. This image. which contains many sharp edges, will be used to illustrate the more extreme effects of the bleeding.

Thus, the $X_i(j,k)$ matrix (2.7) would change in the following manner:

$$\text{for } l,m{=}-1,0,1;\ (l,m) \neq (0,0)$$

IF $|x_i(j+l,k+m) - x_i(j,k)| < RT$ then $x_i'(j+l,k+m) = x_i(j+l,k+m)$
ELSE $x_i'(j+l,k+m) = x_i(j,k)$

$$X_i'(j,k) = \begin{pmatrix} x_i'(j-1,k-1) & x_i'(j,k-1) & x_i'(j+1)(k-1) \\ x_i'(j-1,k) & x_i(j,k) & x_i'(j+1)(k) \\ x_i'(j-1,k+1) & x_i'(j,k+1) & x_i'(j+1)(k+1) \end{pmatrix} \tag{2.14}$$

The equation for the reconstruction process then becomes:

$$x_{i-1}(2j+l,2k+m) = R_l F' R_m X_i'(j,k) \tag{2.15}$$

In the general case, image quality can improve with the use of a reconstruction threshold. However, it is difficult to determine what value should be given to the threshold and in which cases the threshold is not required. Images which are compressed at higher bit rates, as well as those which contain many sharp edges between contrasting regions would appear to benefit the most.

Interestingly enough, although the reconstruction threshold gives improved results when the single filter F' is used, its use does not contribute visually or numerically (in terms of MSE) for the reconstructed image when filters $F_1 - F_{4+}$ are used for the different levels.

A good example of how the reconstruction threshold can be used to improve image quality can be seen in the images in Figure 2.8. Figure 2.8(a) does not use a reconstruction filter, while Figure 2.8(b) does. The corresponding error images are shown in Figure 2.8(c) and (d). Note the superiority of the error image in (d).

Because of the increased visual image quality resulting from using the single filter in combination with a reconstruction threshold, it would appear to be a better alternative than the use of the multiple filters described earlier. When referring to the improvements made to Shusterman and Feder's quadtree algorithm, the single filter and reconstruction threshold method (or SR quadtree compression, as it will be called hereafter) will be implied.

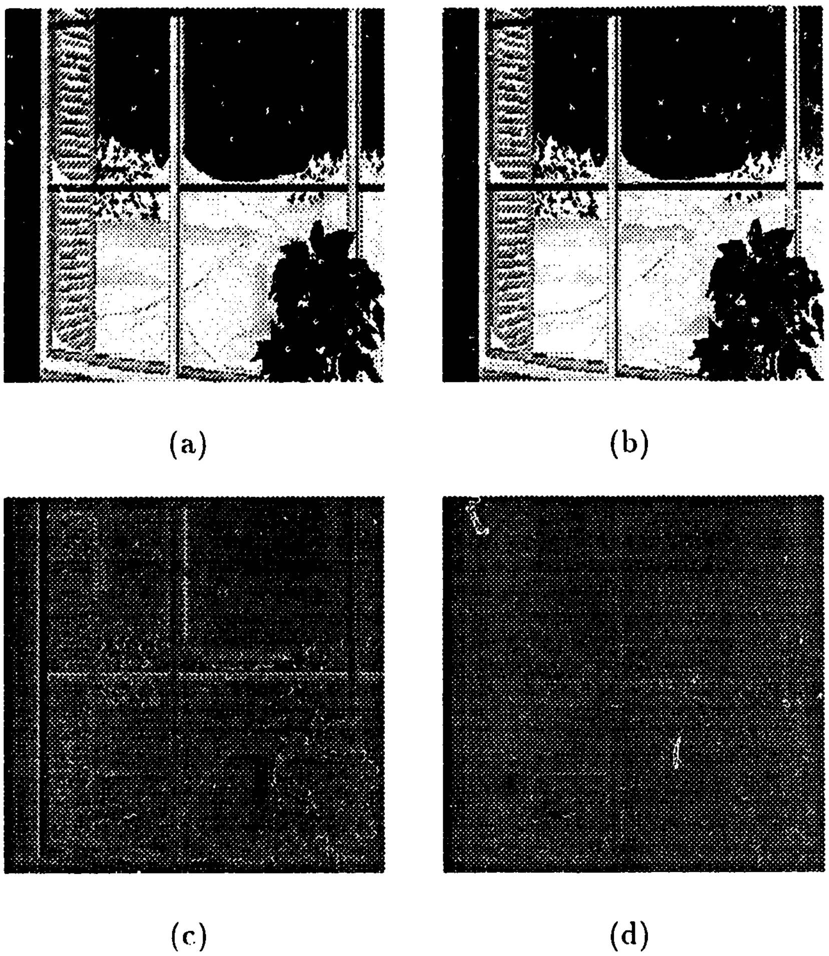


Figure 2.8: Reconstruction and Error Images for Figure 2.7

2.4.4 Coding Alternatives

Rather than simply quantizing the leaves and packing bits into an array for output, comparisons were done with Huffman coding and Arithmetic coding schemes to determine which would generate the best results. The input to the Huffman and Arithmetic coding routines was the quantized leaf values. In general, Arithmetic coding performed best, although the margin of improvement over bit packing is quite small, indicating that the bit allocation suggested by Shusterman and Feder is indeed very near to optimal. All of the results reported in this thesis, however, are based on the use of the Arithmetic coding scheme, since it does give a slight improvement without adding a significant amount of complexity.

Table 2.2 gives a comparison of the three alternatives that were considered, using the 512x512x8 image **Lena** and varying values of the initial decomposition threshold, T_1.

Sizes of compressed files (in bytes)			
Lena			
T_1	Bit Packing	Huffman Coding	Arithmetic Coding
10	41458	41356	38989
20	14199	15335	12443
30	9413	10247	8513
40	6396	6748	6128
50	4485	4961	4318
60	3613	4037	3544
75	2897	3242	2803

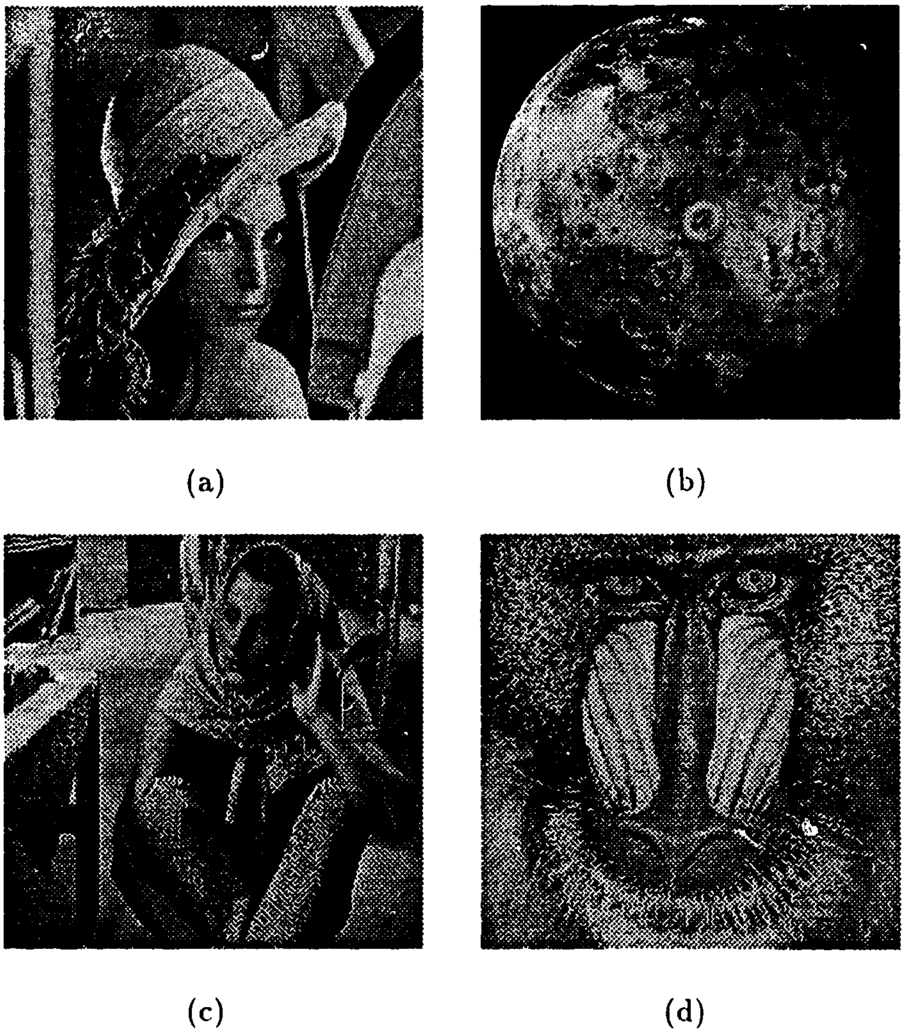
Table 2.2: Comparison of coding alternatives

2.4.5 Results

In this section, we will use several graphs and tables in order to illustrate the results of the algorithm. Several Rate-Distortion curves (hereafter referred to as R-D curves) will be presented for the methods described (as well as JPEG compression). The methods graphed will be the original presented by Shusterman and Feder, the modified version including different filters for each level, and the version which includes a single 2x2 filter and a reconstruction threshold. The above methods will be referred to as "SF" (Shusterman and Feder), "MF" (Multiple Filters) and "SR" (Single filter and Reconstruction threshold). In addition to the R-D curves, a table containing information relating the results of the new algorithm to JPEG compression will be shown.

Test Image Set

In order to adequately determine the success of any compression algorithm, the algorithm must be tested on a wide range of test images. Many compression schemes are compared to previous methods based on their performance with respect to very few images - in fact, often only one test image is used, usually the *Lena* image. Four test images that will be used more commonly in this thesis are shown in Figure 2.9.



(a)

(b)

(c)

(d)

Figure 2.9: Commonly used test images: (a) Lena; (b) Io; (c) Barbara; (d) Mandrill.

Since many different versions of the popular test image *Lena* are in existence, it should be noted for those interested in making comparisons that the image used in this research was obtained from ftp://ipl.rpi.edu, and is the same version of *Lena* used in the wavelet compression paper by Said and Pearlman[21].

There are several different types of test images which should be used in testing a compression scheme. The major groups of images which will be utilized for testing in this thesis are the following common and not necessarily mutually exclusive sets:

- **Busy images:** plenty of detail. e.g. *Barbara*
- **Smooth images:** low detail. e.g. *Lena*
- **Portrait images:** common images of human faces
- **Nature images:** NASA images, outdoor scenes
- **Computer generated images**

Numerical Results

Since the implementation of the quadtree methods does not provide for the specifying of an exact bit rate (the T parameter will vary the degree of compression, similar to the *quality* parameter in JPEG compression), tablular summarizations of the results are not useful. Thus, graphical representations of the results, with the bit rate on the horizontal axis and (inverse) PSNR on the vertical axis (known as a Rate-Distortion or R-D curve) will be used instead.

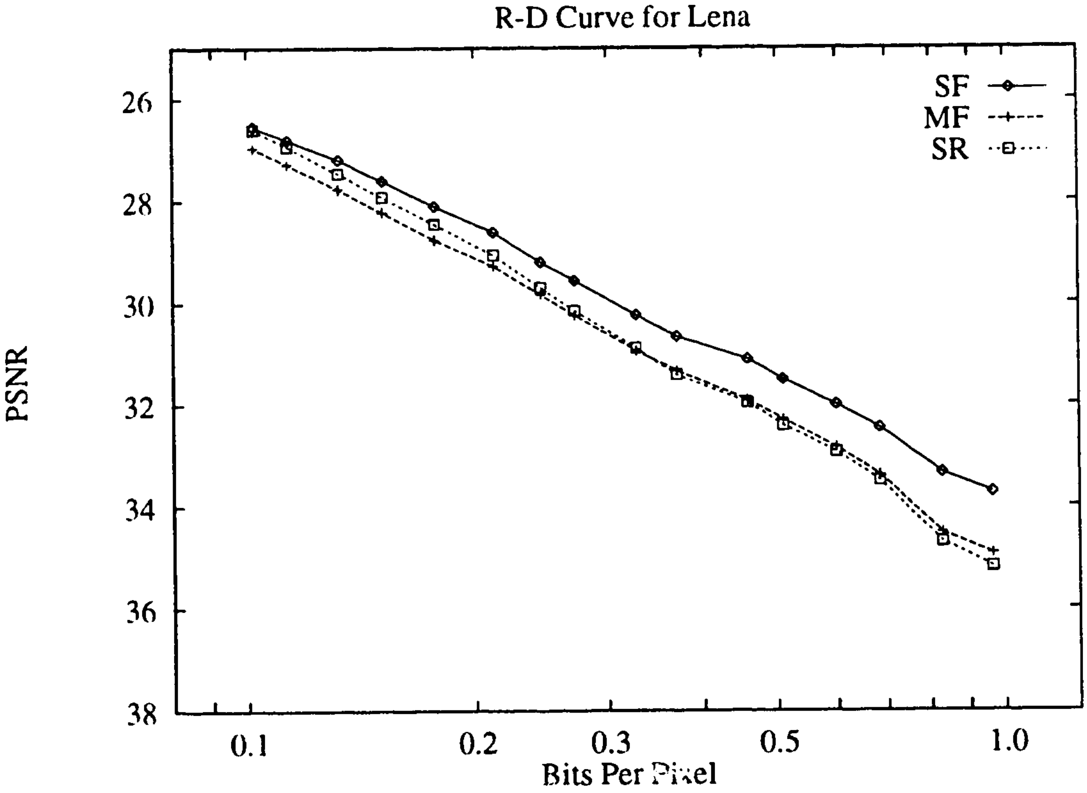


Figure 2.10: PSNR results f. · *Lena*

Figure 2.10 clearly shows that the modifications made to Shusterman and Feder's algorithm result in a noticeable improvement in the image quality for *Lena* as measured by the PSNR. Note that in many cases, especially at low bit rates, the multiple filters (*MF*) give the best results. However, the PSNR values for the *SR* are very similar, and are consistently better than the original single filtering method.

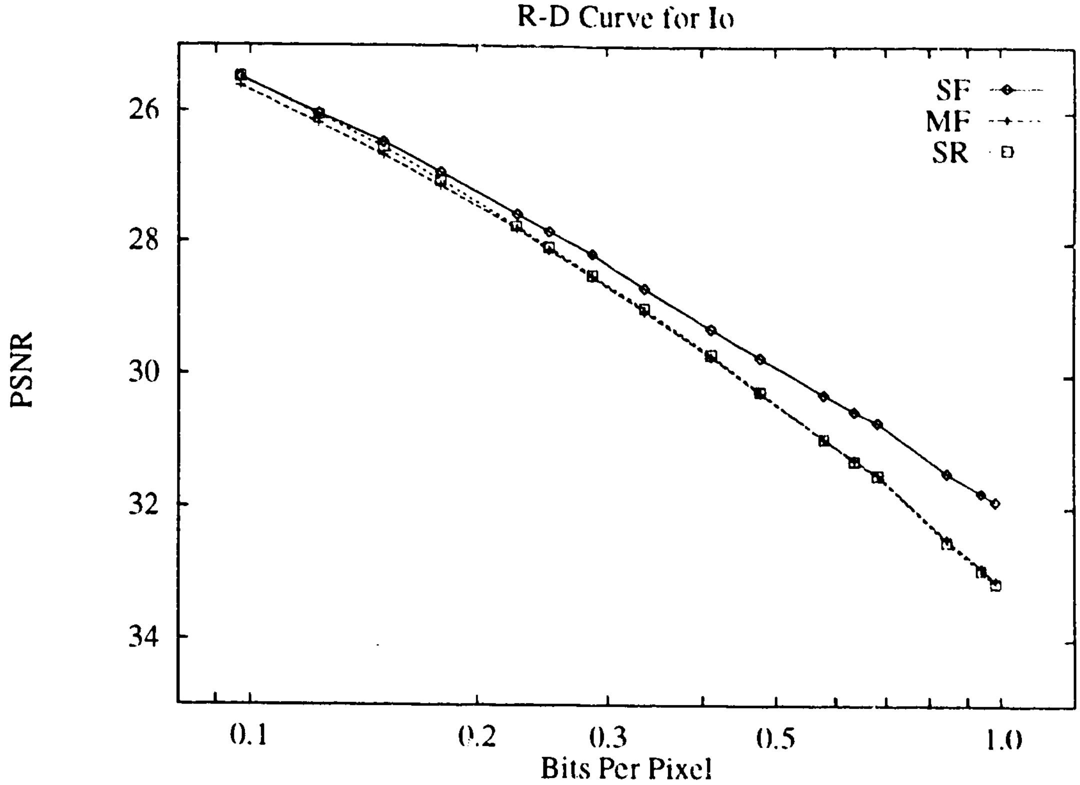


Figure 2.11: PSNR results for *Io*

Similar results are observable in the "Io" image when the results from the algorithms are compared in Figure 2.11. Again, we can see a improvement in results at every bit rate.

In order to make a complete comparison, the results should be compared to JPEG compression. In the following comparisons, only JPEG compression and *SR* results will be compared for two reasons: first, the *SR* scheme gives the best *visual* results, regardless of the fact that *MR* often gives better PSNR values; second, the resulting graph will be less cluttered than if all the filtering methods were included.

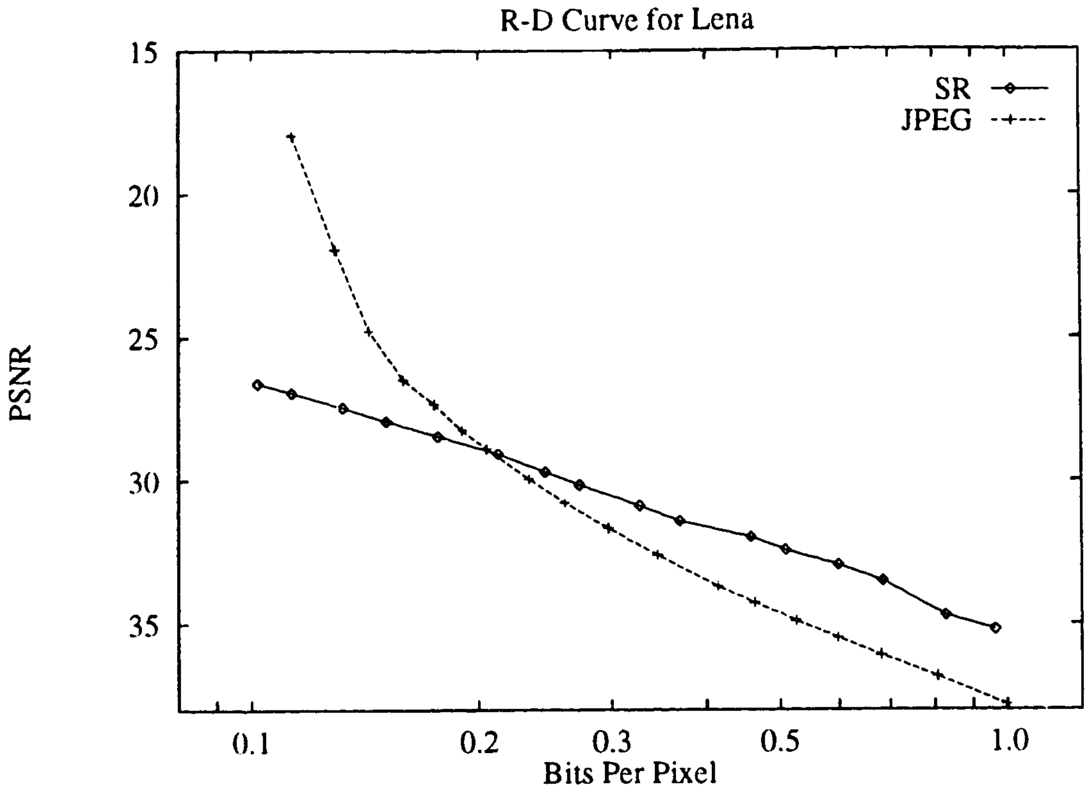


Figure 2.12: Comparison to JPEG compression using Lena

Figure 2.12 shows that the modified algorithm performs better than JPEG up to approximately 0.2 bpp. It may also be noted that visually, the JPEG results are quite noticeably blocky for bit rates lower than 0.4 bpp, which may make it less visually appealing than the SR results, as can be seen in Figure 2.16.

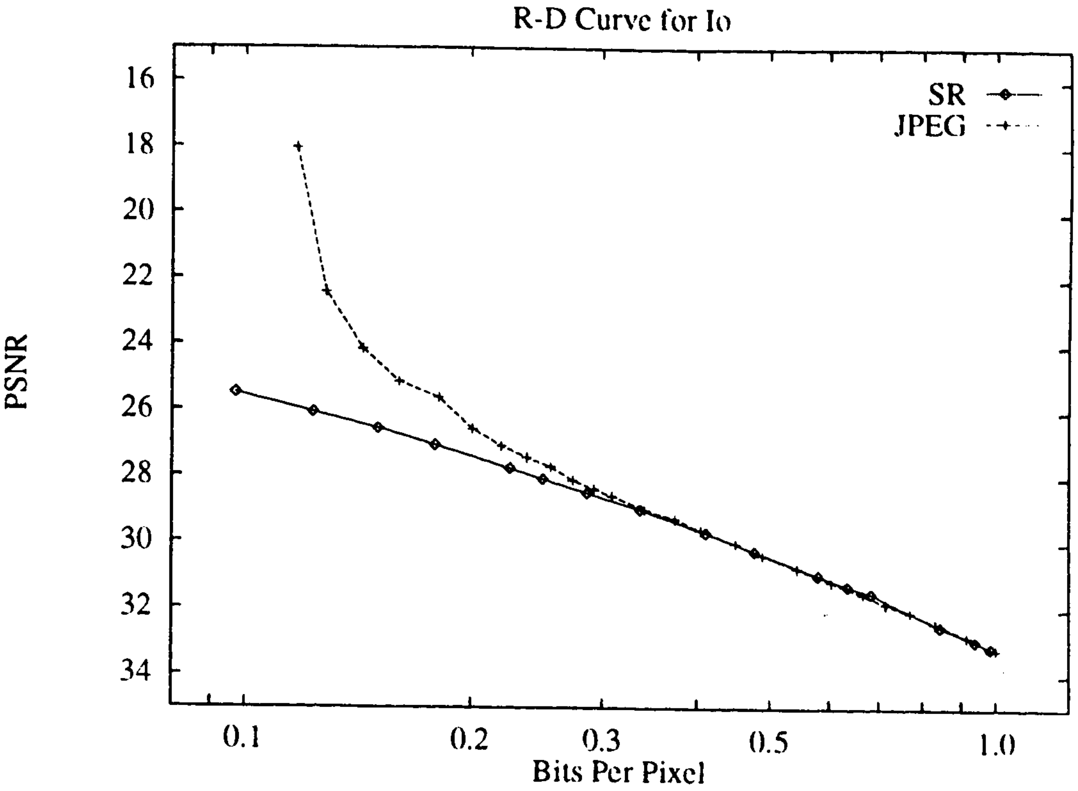


Figure 2.13: Comparison to JPEG compression using Io

Again as can be seen in Figure 2.13, the modified algorithm performs better than JPEG, in this case at higher bit rates than the previous example, up to about 0.35 bpp. At higher bit rates, the algorithm gives almost identical PSNR results to JPEG.

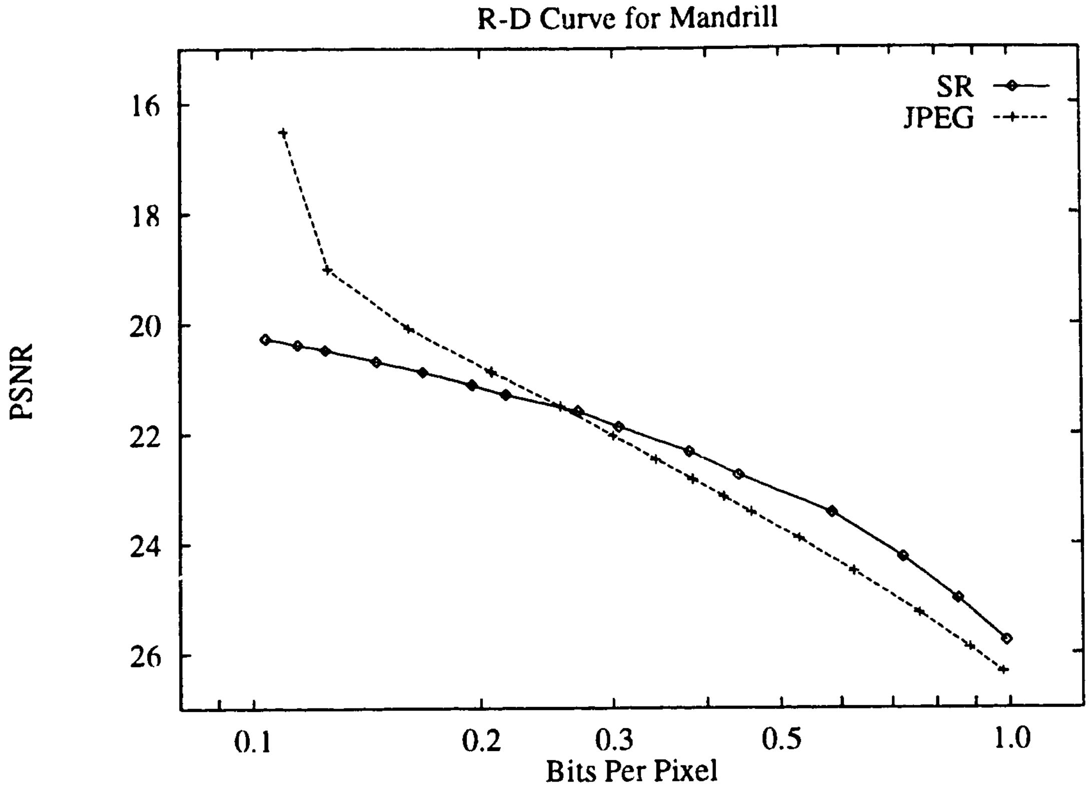


Figure 2.14: Comparison to JPEG compression using Barbara

Figures 2.14 and 2.15, showing results for *Barbara* and *Mandrill* respectively, indicate that the R-D curves for these two images are similar to that of *Lena*. The *SR* method wins handily at the lower bit rates, but is surpassed by JPEG between 0.2 and 0.3 bits per pixel. Again, emphasis should be given to the fact that JPEG's blockiness often makes the *SR* algorithm a desirable alternative at bit rates higher than what the graphs would seem to indicate.

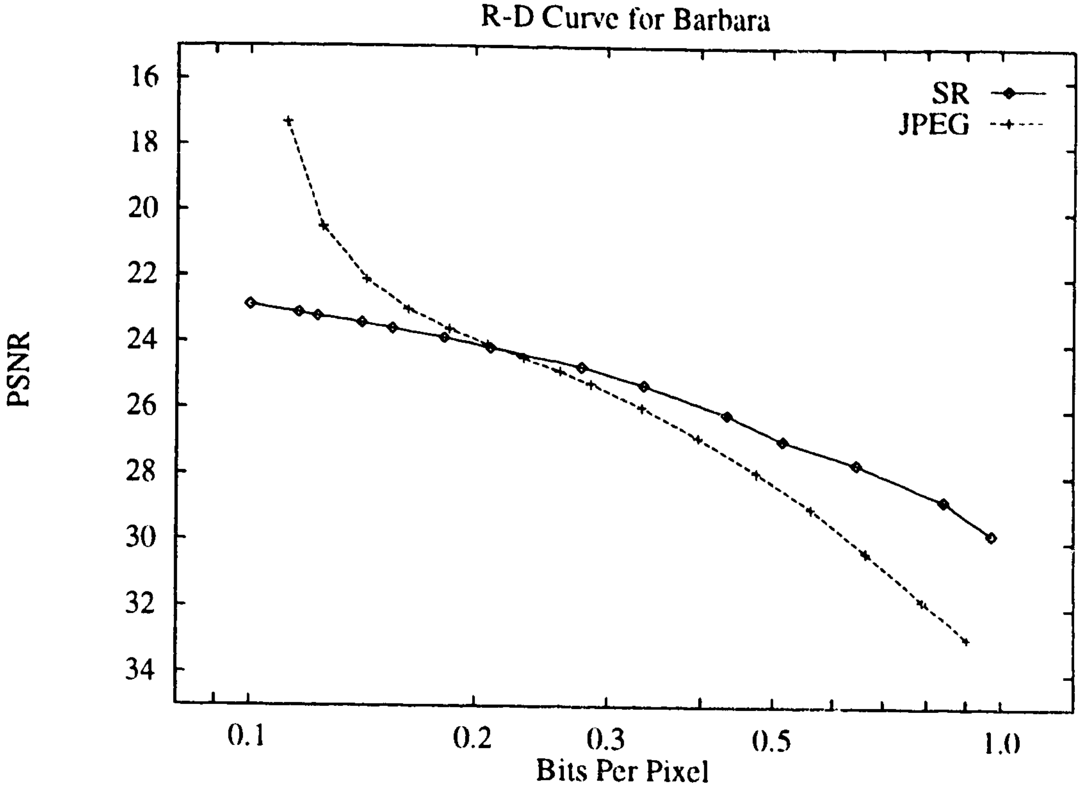


Figure 2.15: Comparison to JPEG compression using Mandrill

Since the results described above consider only four distinct images and thus may not present a complete picture of the results, some additional data has been recorded in Table 2.3. The results were compiled using 14 of the 256x256 images from the test image set, again using the *SR* reconstruction scheme to obtain the quadtree results. Instead of showing the R-D graph for each of the 14 images, two pieces of information are recorded from the R-D curves of each test image:

1. The point at which the R-D curves cross (in bits per pixel). At bit rates lower than the given points, *SR* outperforms JPEG.

2. The positive improvement in PSNR given by using *SR* as opposed to JPEG at approximately 0.2 bits per pixel (the R-D intersection is at a bit rate greater than 0.2 bpp in each case, so ΔPSNR will always be positive) is also given. A bit rate of 0.2 bpp is used

Image	R-D Intersection	ΔPSNR at 0.2 bpp
Bay	2.26	0.9
Build	3.20	0.8
Car	0.25	0.9
Carrier	0.27	0.9
Earth	0.26	0.9
Flower	0.28	1.8
Fruit	0.27	1.0
Lena	0.31	1.5
Model	0.25	0.7
Monalisa	0.29	1.0
Pfeifer	0.26	1.2
Potala	0.28	0.9
Street	0.30	0.5
Sunset	0.25	1.6

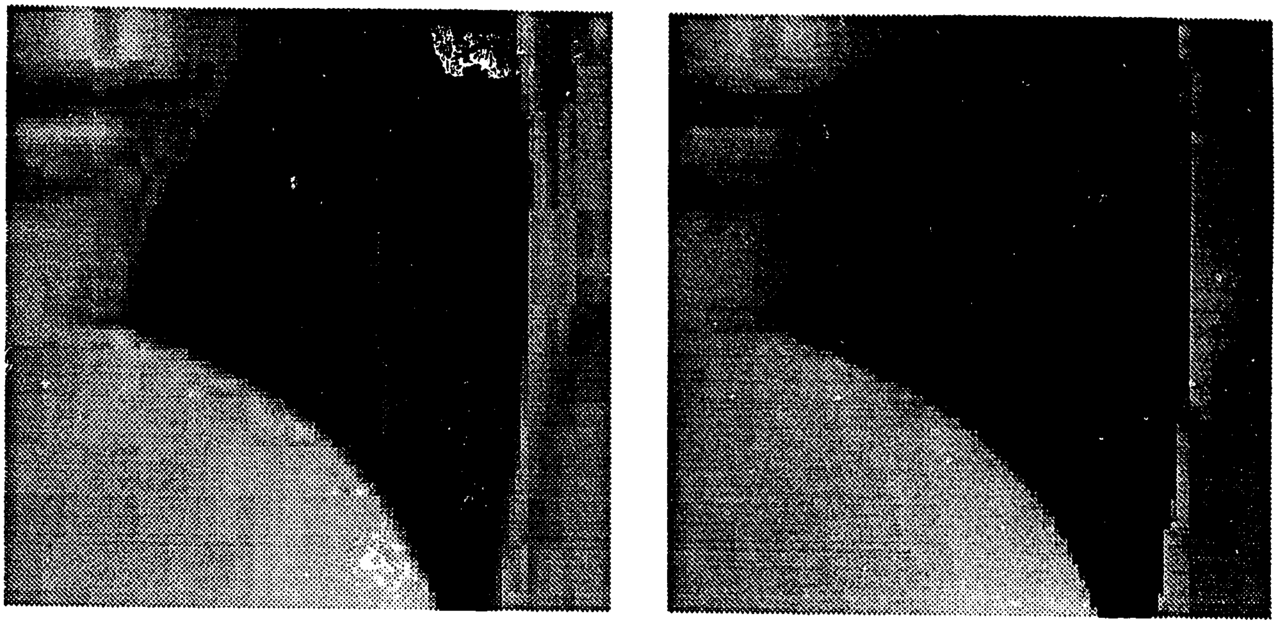
Table 2.3: Comparison to JPEG compression

Note that in each case there is a positive result in favour of the SR reconstruction scheme. The first two images show very impressive results similar to those which can be seen in the R-D curve for Io 2.13

2.5 Complexity

In this section, the theoretical complexity of the proposed algorithm will be examined and compared to the complexities of Shusterman and Feder's method[28], Strobach's RPD coding[30] and JPEG compression[20]. Computationally, the improved reconstruction process performs better than the original process suggested by Shusterman and Feder, and the decomposition process can theoretically outperform both RPD coding and JPEG.

It is clear that at low bit rates, the leaf terms become less significant, and the number of required operations decreases. The approximate numbers of operations required for images compressed to 0.80 bpp and 0.16 bpp are shown in Table 2.4.



(a) JPEG

(b) SR

Figure 2.16: Lena compressed to 0.3 bpp with JPEG and SR

Strobach's RPD compression complexity is given in his paper[30], and is shown in Table 2.5.

The decompression complexity is not given. However, it is clear that the compression complexity of the proposed algorithm is somewhat better than that of Strobach's RPD method which, as has already been claimed, is a fairly simple method relative to other common compression schemes.

The complexity of the JPEG algorithm, without the symbol encoder, is based on the fast DCT algorithm described by Arai *et al.*[20, 34], which requires only 5 multiplications and 29 adds for a 1-D DCT of dimension 8, which must be performed 16 times for each 8x8 block. The quantization of the DCT coefficients will also require 1 division for each pixel. Thus, the operations count will be as shown in Table 2.6: Note that the decompression phase of JPEG will require the same operations count, except that the divisions become multiplications.

In theory, the proposed algorithm is clearly superior in terms of operations in the compression phase. The analysis of the decompression phase shows it to be similar

Operations per Pixel (0.80 bpp)		Operations per Pixel (0.16 bpp)	
Compression:			
Shifts	0.50	Shifts	0.36
Additions	3.13	Additions	2.48
Multiplications	0.16	Multiplications	0.03
Decompression:			
Shifts	0.16	Shifts	0.03
Additions	6.82	Additions	6.69
Multiplications	3.00	Multiplications	3.00

Table 2.4: Operation count for the *SR* algorithm

Compression:	
Additions	8.00
Multiplications	3.33

Table 2.5: RPD operation count

in operation count to JPEG compression.

2.6 Future Work

Although the algorithm performs well, the only changes made to the original algorithm by Shusterman and Feder were made to the reconstruction process. There were,

Operations per Pixel	
Compression:	
Additions	7.25
Multiplications	1.25
Divisions	1.00
Decompression:	
Additions	7.25
Multiplications	2.25

Table 2.6: JPEG operation count

in fact, several successful modifications to the decomposition phase which were not incorporated because the magnitude of their improvements did not justify the added increases in computational complexity. In this section, these modifications, which may present alternatives for future work, will be described briefly.

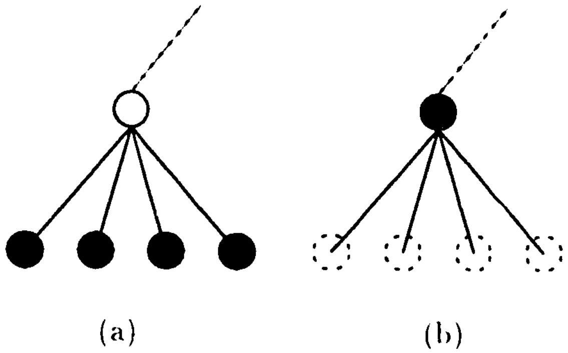


Figure 2.17: Storage of vectors in quadtree nodes: (a) the current scheme, with leaf nodes shaded black; (b) a potential vector quantization scheme, using the next-to-leaf node (shaded black) to store a vector which would be used to reconstruct the leaf nodes

Because of the nature of the decomposition process, the leaves in the quadtree structure have children whose values vary little from each other, the magnitude of the variance being dependent on the decomposition threshold T. This fact could simplify the construction of a 2x2 vector codebook which could be used to code each leaf which was not at the pixel level. Such a process could likely also benefit if a different codebook were used for each level, since the variance in the values of a leaf's children decreases at the higher levels of the tree. Figure 2.17 shows how the next-to-leaf quadtree nodes rather than leaf nodes could be used to store pixel intensity information. Variations of such a scheme could include:

1. Coding a single vector containing an approximation of the child values

2. Coding an error vector and an average value

A brief experiment into the use of LGB vector quantization in the next-to-leaf level of the quadtree structure was attempted, with promising initial results. The image quality (visually and as measured by PSNR) was generally found to be very close to that generated by *SR* quadtree compression.

The downside to the addition of such a vector quantization scheme is the obvious increase in complexity. One of the major strengths of the *SR* quadtree algorithm is its low complexity as compared to existing methods, which would be eliminated with the addition of LGB vector quantization. Justification for the addition would need to be present in the form of much improved image quality, or the use of a simpler vector quantization scheme such as lattice vector quantization, which will be examined in the next chapter.

2.7 Conclusion

As compared to Shusterman and Feder's method, the proposed algorithm performs slightly better computationally in the reconstruction of the image, and also generates better results with the additions of the improved 2x2 filters and reconstruction threshold. It also outperforms Strobach's method both in quality and complexity, making it currently the best performing quadtree decomposition compression method.

The *SR* reconstruction algorithm has several advantages over previous methods in terms of computational complexity and compressed image quality at low bit rates. Its performance is comparable to JPEG compression in the range of less than 1.0 bpp, and is in fact able to generate higher quality results than JPEG at lower bit rates, and better looking images at some higher bit rates.

It has been shown that the described algorithm is simpler than JPEG computationally, using significantly fewer operations for the compression phase. This, combined with the fact that it can outperform JPEG at low bit rates, makes it a desirable alternative.

There is a good deal of potential for future work in the combining of this method with other compression schemes such as vector quantization, as such a combination has already been successfully attempted with some improvement in the resulting quality.

In conclusion, this algorithm would appear to be suitable for use with a portable or wireless system which would depend on speed and low bit rate transmissions. It has been shown that the presented algorithm performs well in both areas.

Chapter 3

Wavelet Compression

The use of the wavelet transform in image compression has become very popular as of late, and has been used in many very successful algorithms. Much current image compression literature is devoted to the applications and advantages of the wavelet transform. Wavelet compression techniques have evolved to the point where they can be shown to be very computationally efficient, as well as generating exceptional compression results.

A basic introduction to the mechanics of the wavelet transform will be given, followed by a literature survey of some recent, successful wavelet compression algorithms. The proposed wavelet algorithm, which is similar in many aspects to those in the literature survey, will then be described and analyzed.

3.1 Introduction to Wavelet Compression

3.1.1 Basic Wavelet Concepts

Conceptually, the wavelet transform can be broken into two separate transforms which perform lowpass and highpass filtering on the input data stream. In theory, the wavelet transform is lossless in that a data stream which has been transformed using a forward wavelet transform can be completely reconstructed using the corresponding inverse transform. Many if not most compression algorithms based on wavelet decomposition, however, allow for lossy compression.

The forward wavelet operators can be described as a pair of convolution operations with the forward lowpass and highpass filters L and H, performed on the original data stream C^0:

$$C^1 = L * C^0$$

$$D^1 = H * C^0$$

where $*$ is the convolution operation. Similarly, the reverse wavelet transform can be described using the inverse lowpass and highpass filters L^{-1} and H^{-1}:

$$C^0 = L^{-1} * C^1 + H^{-1} * D^1$$

Because quantization is usually performed on the wavelet coefficients before the reconstruction of the data stream is attempted, it may be more accurate to express the previous equation as follows:

$$\hat{C}^0 = L^{-1}\hat{C}^1 + H^{-1} * \hat{D}^1$$

where $\hat{C}^1$, $\hat{D}^1$ and $\hat{C}^0$ are the quantized versions of C^1, D^1 and C^0 respectively.

$$\begin{tikzcd}
C^0 \arrow[r, "L"] \arrow[dr, "H"'] & C^1 \arrow[r, "L"] \arrow[dr, "H"'] & C^2 \\
& D^1 & D^2
\end{tikzcd}$$

Figure 3.1: Wavelet decomposition

The space requirement for the wavelet transform operations is *often* the same as that required by the original data stream, the low and high frequency segments each requiring half the space of the original signal. This work will consider only those wavelet filters which meet this criterion. Consider Figure 3.1 in which the original data stream is shown as C^0 and the wavelet decomposition is indicated by the

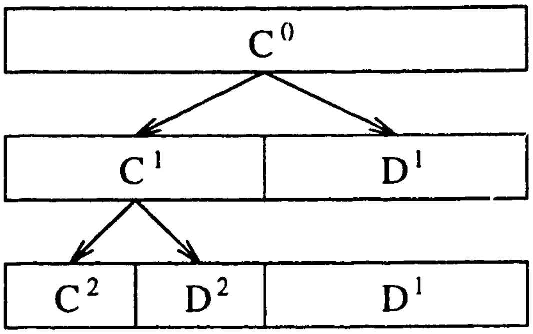


Figure 3.2: Space taken by transform coefficients

arrows labeled L and H. Because the original stream can be reconstructed by its two transformed components (which are both half the size of the original), once a transform is performed the original stream can be overwritten by its components. Thus, at the end of the process illustrated above, the transformed stream would not require any additional space other than the original data stream storage (see Figure 3.2).

Thus, the nature of the wavelet transform allows for the splitting of a data stream into two separate components with each coefficient generated by this operation representative of the spatial area of approximately two of the original data elements. The low frequency component can be thought of as a quantized version of the original data stream. Because of this, we know that an additional wavelet transform performed on this low frequency component will generate another set of two components, one of which will be a further quantized version of the original data stream, and another high frequency component. Note that in Figure 3.1, as a result of the similarity between C^0 and C^1, the high frequency components D^1 and D^2 will share similar properties. Although D^2 will be coarser than D^1 (spatially, each D^2 coefficient will be representative of two coefficients in D^1), similar high frequency phenomenon can and will be observed in both. It is this redundancy, when exploited, along with the compaction of the low frequency information, which proves to be very useful for compression purposes.

3.1.2 Distribution of Coefficient Magnitudes

The nature of the coefficient distribution is of some interest since, as was emphasized previously, an ideal transform compression method will generate much redundancy. Generally, like the DCT example, a given transform method will generate coefficients which are concentrated around zero, a phenomenon which is often modeled using modifications of the Gaussian curve[3]. The high frequency wavelet transform coefficients also have these characteristics. A typical distribution of high frequency wavelet coefficients, using more than 65000 coefficients generated from a single test image, is shown in Figure 3.3.

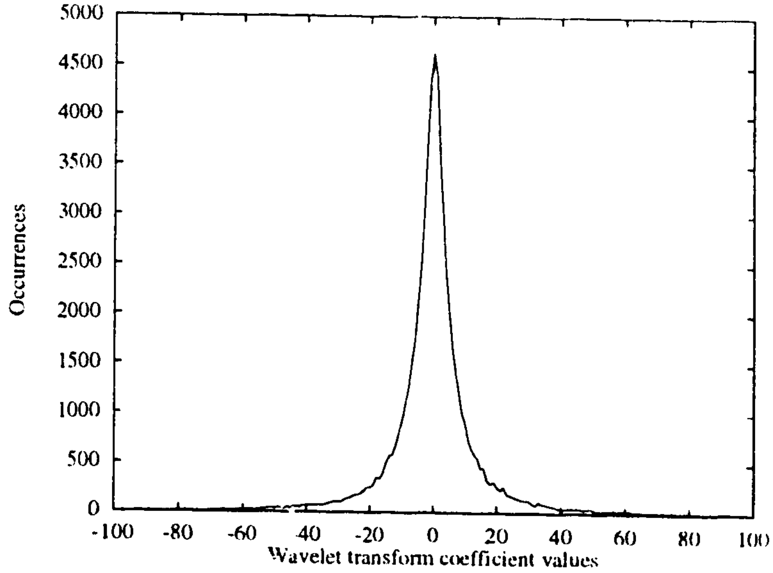


Figure 3.3: High frequency coefficient distribution

Obviously, the low frequency portion of the wavelet transform will be distributed rather randomly, depending on the intensities of the original data stream. Some compression schemes attempt to manipulate the low frequency information by subtracting the mean so that it becomes more similar to the high frequency information. As will be shown later, however, the low frequency components make up a very small proportion of the final coefficients and their effect on the distribution is minimal.

3.1.3 Relative Importance of Wavelet Coefficients

The relative importance of a given wavelet transform coefficient to the quality of the reconstructed signal is a very significant issue. Without knowing which of the coefficients are important to the reconstruction process, an effective quantizer cannot be easily constructed or understood.

Mathematically, the following equation relates the original data stream to the transformed coefficients:

$$(1-\epsilon)(\|C^1\|^2 + \|D^1\|^2) \leq \|C^0\|^2 \leq (1+\epsilon)(\|C^1\|^2 + \|D^1\|^2) \tag{3.1}$$

For most filters, the value of ϵ can be assumed to be small. In fact, for *unitary filters* $\epsilon = 0$, and equation 3.1 becomes

$$\|C^0\|^2 = \|C^1\|^2 + \|D^1\|^2 \tag{3.2}$$

It then follows for unitary filters that the error caused by a particular quantization scheme can be described as

$$(\|C^1\|^2 - \|\hat{C}^1\|^2) + (\|D^1\|^2 - \|\hat{D}^1\|^2) \tag{3.3}$$

Clearly then, the mean-squared error of a reconstructed image will be identical to the mean-squared error in the coefficients as a result of quantization, assuming that a unitary transform is used. While this is not the case in general, since most common wavelet filters are non-unitary, the indication is still that *larger wavelet coefficients are more important than smaller ones.*

3.2 Literature Survey in Current Wavelet Techniques

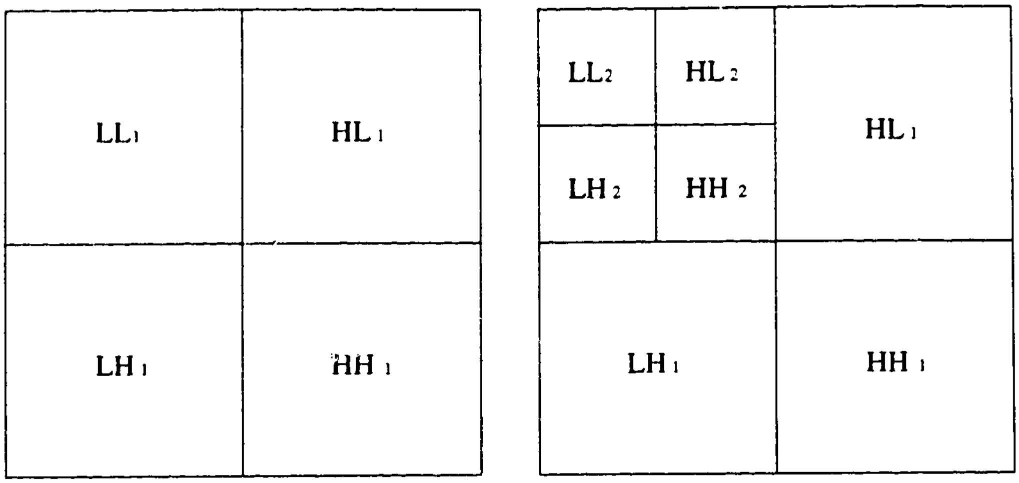
Many wavelet transform based compression methods have been developed recently which exhibit impressive compression performance, low complexity, or both. This section examines several of these schemes which are of particular interest because of their influence on and/or similarity to the proposed wavelet algorithm. Before going into the details of the methods, a summary of *multiresolution decomposition* will be provided. since this forms the basis for all of the background methods which will be discussed.

3.2.1 Multiresolution Decomposition in Wavelets

The main purpose of any transform domain compression technique is to localize redundancy, making it simpler to develop effective coding schemes. The DCT as applied in JPEG compression[20] for example, will often generate coefficient blocks which contain large numbers of small coefficients which can be safely quantized to zero without any significant loss of detail in the reconstructed block. The JPEG algorithm codes the transform coefficients as a one dimensional stream using a variable length code which takes advantage of large strings of zero coefficients. Similarly, for the wavelet transform to be applied effectively, an element of redundancy which does not exist between the pixels themselves must be introduced.

Although one application of a wavelet filter will result in coefficients which are distributed about zero and hence easily exploitable for redundancy, several iterations of the filter on the low resolution components of the previously transformed coefficients results in even more easily exploitable redundancy, as was discussed briefly in the previous section. The two dimensional case, being more complex, warrants a more complete investigation into the advantages of iterative application of the wavelet transform.

In practice, wavelet filters may be applied to a particular image many times. Multiresolution wavelet decomposition is a common application of the wavelet transform which involves splitting the image into "subbands". A wavelet filter is iteratively applied to the low frequency component of the previous transform coefficients, creating smaller, coarser subbands. This type of decomposition initially involves using both the lowpass and highpass filters first on the rows of an image and then on the columns. The resulting coefficient array will then be broken into four subbands of the form shown in Figure 3.4a. This process continues, with the transformations always



(a)

(b)

Figure 3.4: Wavelet decomposition of an image: (a) after one iteration of the filter; (b) after two iterations

being applied to the most recently generated low frequency (LL) subband (see Figure 3.4b).

Because of the order of the application of the transforms, the high frequency subbands contain different types of high frequency information. The LH subbands contain horizontal high frequency information because the high frequency filter transforms the columns, HL subbands contain vertical high frequency information since the high frequency filter transforms the rows, and the HH subbands contain diagonal high frequency information as a result of the high frequency filter being applied to both the rows and columns.

One of the current conventions for the identification of the different subbands is to describe them as resolution-orientation pairs. The exact convention which will be used in this work will be as shown in Figure 3.5. The low frequency subband will be known as resolution 0 (with no orientation). The coarsest high frequency subbands will have resolution 1 and orientation 0, 1 or 2 corresponding to horizontal, diagonal and vertical subbands respectively. The next coarsest subbands will have resolution 2, and so on. The entire subband structure is often referred to as a *pyramid* ; in order

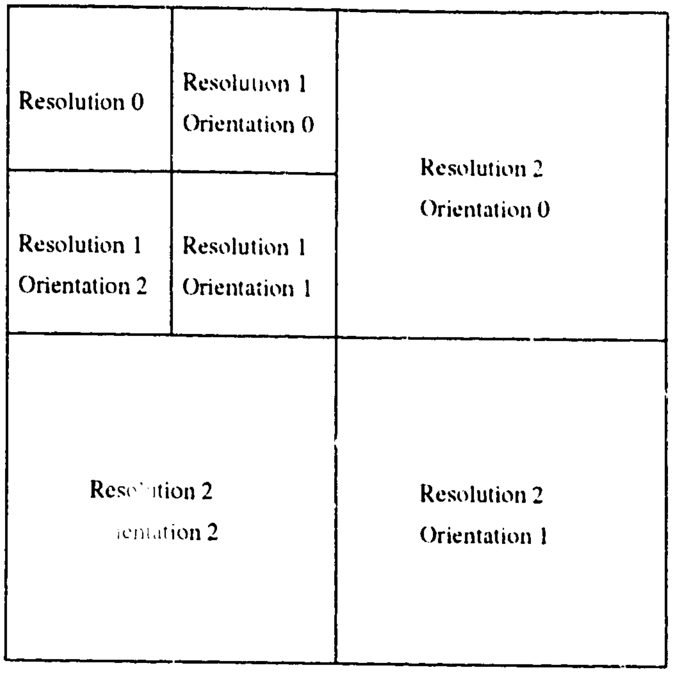


Figure 3.5: Naming of subbands

to make the subject clear, it will be hereafter referred to as a *wavelet pyramid*.

As a result of the similarity in the low frequency subbands at different iterations, there will exist additional similarities between corresponding high frequency subbands at different iterations. Figure 3.6 illustrates how similar features in high resolution subbands are propagated throughout the subband structure. Each coefficient in a high frequency subband (with the exception of those contained in the finest resolution subbands) will be roughly equivalent to four coefficients in the next finest resolution. It seems reasonable to assume then that guesses or predictions can be made about the values of the "descendants" of a particular coefficient given its value.

Because the low frequency subband (which is in the upper left of the previous figures) will not contain any high frequency features, its coefficients cannot be used to predict the values of other coefficients in the same way that can be done with coefficients in high frequency subbands.

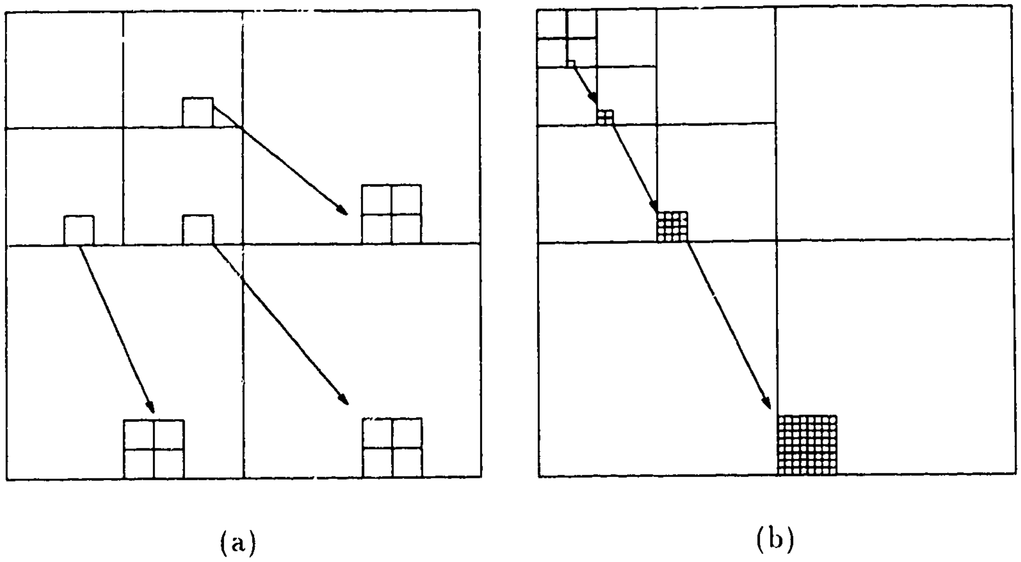


Figure 3.6: Dependencies in image subbands: (a) illustration of the immediate dependencies of three coarse coefficients; (b) all of the descendants of a coarse resolution coefficient

3.2.2 Zerotrees of Wavelet Coefficients

In previous sections, two characteristics of wavelet coefficients were clearly identified as being important to achieving superior compression results. Those characteristics were:

1. **Subband redundancies** - potential for predictive coding

2. **Importance of large coefficients** - essential for minimizing error

One particular compression method which takes advantage of both points, as well as being computationally efficient and generating an embedded code, is Shapiro's *Embedded Zerotree Wavelet*[27] algorithm, hereafter referred to as EZW.

While other wavelet algorithms based on multiresolution decomposition such as that proposed by Antonini *et al.*[3] generate good results by concentrating on generating a vector codebook and optimizing bit allocation for the different subbands, the EZW algorithm bases coding on a hierarchical structure which groups coefficients in a manner which is similar to the parent-child structure discussed previously with

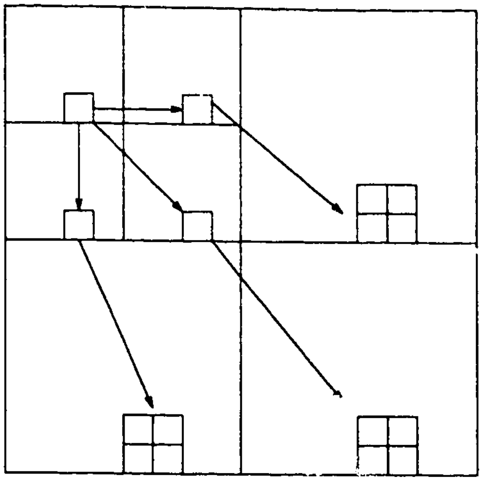


Figure 3.7: Parent-child dependencies in the EZW algorithm

a minor difference. This structure includes the low frequency subband as well to preserve the embedded nature of the final code (this will be made clear later) in the manner shown in Figure 3.7.

Possibly the most important contribution of Shapiro's paper to wavelet compression is the zerotree assumption, which takes full advantage of the aforementioned subband redundancies. To quote Shapiro[27],

> A wavelet coefficient x is said to be *insignificant* with respect to a given threshold T if $|x| < T$. The zerotree is based on the hypothesis that if a wavelet coefficient at a coarse scale is insignificant with respect to a given threshold T, then *all* wavelet coefficients of the same orientation in the same spatial location at finer scales are likely to be insignificant with respect to T.

Logically, the zerotree idea makes sense. An insignificant coefficient at a coarse resolution will likely have descendants which are also insignificant. While this is not true in all instances, it is usually the case that the zerotree hypothesis is correct approximately 10 times more often than not (this varies depending on the input image)

The next logical step might seem to be to make another similar assumption, that a significant coefficient has significant descendants; however, this is not as certain. It is

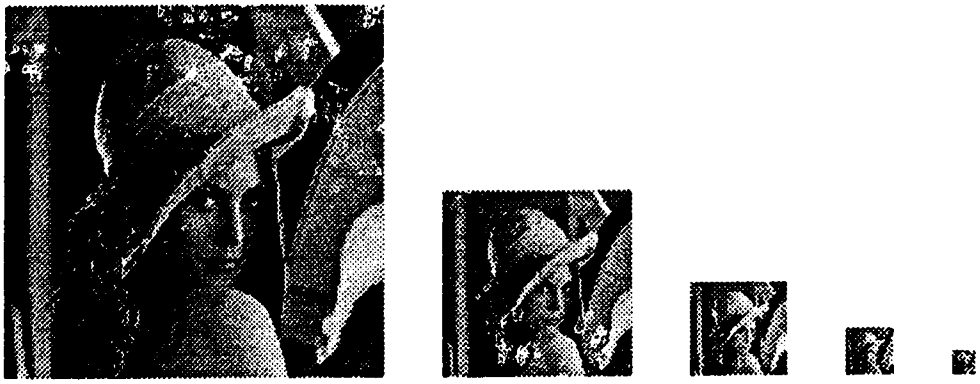


Figure 3.8: Quantized versions of the Lena image

likely that in the quantization of the original signal (iterative application of the lowpass wavelet filters), higher frequency information may be obtained from the quantized signal than were obtained from the original. Consider, for example, the sequence of quantized images shown in Figure 3.8. The higher quantized images show an abundance of high frequency information per unit area. Relatively smooth areas can become less frequent, and it is easily observable that high frequency information in the highest quantized of the images in Figure 3.8 will often correspond spatially with large portions of low frequency information in the less quantized images.

The zerotree assumption is not true in all cases. While the probability of finding a significant coefficient as the descendant of an insignificant coefficient is not high, it will happen many times in a typical wavelet pyramid. Because larger coefficient values should always be coded because of their relative importance (see equations 3.1, 3.2, 3.3), accommodations for these instances need to be made.

It should be apparent by now that the zerotree assumption can be effectively applied to a wavelet coefficient array. Given a particular threshold, we need only code those coefficients which are significant, using the zerotree criterion to eliminate the need for coding large numbers of descendants of insignificant coefficients. The EZW algorithm does exactly this in a very simple fashion.

Basics of the Algorithm

The only input to the algorithm, besides the image itself, is the exact number of bytes to which the image should be compressed. This *byte budget* can be matched exactly by the algorithm, giving it an advantage over algorithms such as JPEG and the quadtree algorithm discussed in a previous chapter.

Shapiro suggests that the wavelet transform be applied five times to the image, resulting in a 6-level wavelet pyramid. The first step in the processing of the pyramid is to find the initial threshold which will determine a given coefficient's significance. Given the maximum magnitude threshold in the wavelet pyramid, x_{max}, the initial threshold T_0 is set to the largest power of 2 less than x_{max}.

$$|x| \leq x_{max}$$

$$|x| < 2T_0$$

$$T_0 = 2^n, |x| < 2^{n+1} = 2T_0$$

for any x in the wavelet pyramid. Since the algorithm requires many passes of the coefficients and almost all coefficients would be insignificant with respect to T_0 (see Figure 3.3), many passes are made in which the value of the threshold is reduced: $T_i = T_{i-1}/2, i > 0$.

There are two main modules in the algorithm which are executed until the byte budget has been met: the *dominant pass* and *subordinate pass*, each dealing with a corresponding *dominant list* and *subordinate list* respectively. The dominant pass looks at all coefficients which have not yet been found to be *significant* with respect to the current threshold while the subordinate pass deals with refining the values of coefficients which have already been found to be significant with respect to the current (or a previous) threshold.

While the dominant list elements are simple (a reference to a single wavelet coefficient), the subordinate list is more complex. It contains the magnitude of the original coefficient as well as a *reconstructed magnitude*. The reconstructed magnitude is the component which is refined during each subordinate pass. I will refer to the two fields as *actual_magnitude* and *reconstructed_magnitude*.

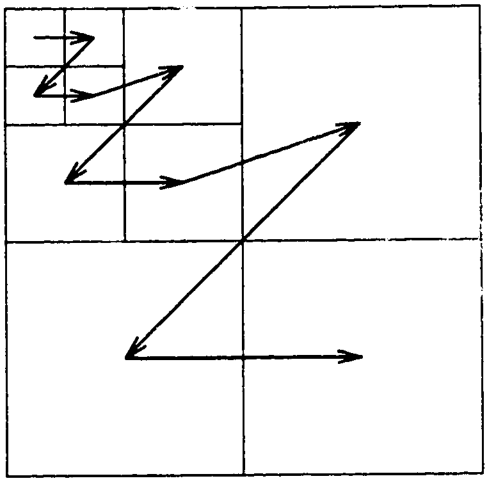


Figure 3.9: Scan order of wavelet subbands

The dominant list is initialized to contain one element corresponding to each coefficient in the wavelet pyramid while the subordinate list is initially empty. The order of elements in the dominant list is important since the coefficients will be scanned and handled in their order of appearance in this list. In order to effectively apply the zerotree idea, no coefficient should be handled before all of its ancestors have been handled first. In this way, we can make use of any zerotrees to ignore the values of zerotree descendants. The suggested ordering by resolution/orientation pair is shown by the Z-scan in Figure 3.9 with the scan starting in resolution 0 and the coefficients within each subband handled in scan-line order. The scan-line order appears to be an arbitrary but reasonable choice since all we wish to do is be assured that the lower resolution coefficients are handled before their higher resolution counterparts. Note that all of the coefficients within a particular subband are handled before any coefficients in the next subband of the Z-scan order.

A high-level outline of the algorithm could be described in the following steps:

1. **Get the image and byte budget**

2. **Transform the image**

3. **Initialize dominant and subordinate lists**

4. **Find the initial threshold** T

5. **WHILE** $(T > 1)$

   - **Perform dominant pass**
   - **Perform subordinate pass**
   - $T \leftarrow T/2$

The Dominant Pass

The dominant pass simply traverses and updates the dominant list, coding symbols where appropriate. There are four distinct symbols which are coded using an arithmetic coder during a dominant pass. They are:

1. ZT - indicates a zerotree root

   If an element of the dominant list is found to be a zerotree root, the ZT symbol is coded, and all descendants of the element are marked so that they will not be coded during this pass.

2. IZ - indicates an isolated zero

   This symbol is coded when we encounter an insignificant coefficient with at least one significant descendant. Nothing further is done besides the coding of the IZ symbol.

3. POS and NEG - indicate significant coefficients

   These symbols are coded when a significant coefficient (positive or negative) is encountered. When this happens, the wavelet coefficient corresponding to the list element is set to zero (so as to not prevent the possibility of zerotrees in subsequent passes), and an element is added to the subordinate list with the fields set as follows:

```
• $actual\_magnitude \leftarrow coefficient\_magnitude$
• $reconstructed\_magnitude \leftarrow \frac{T+2T}{2}$
```

Note that as soon as an element is added to the subordinate list, it is given a "best guess" length (*reconstructed_magnitude*) of the midpoint of the uncertainty interval associated with the the current threshold. That is, since we know that a symbol is never coded when its magnitude is less than T and it will always be coded when its magnitude is greater than T, the best guess as to its actual value is the midpoint of the current threshold and previous threshold values.

The Subordinate Pass

After each dominant pass, a subordinate pass is performed to refine the magnitude of each element in the subordinate list. As was mentioned previously, an element which is inserted into the subordinate list has its reconstructed magnitude set to $\frac{T+2T}{2}$, the "best guess" that can be made with the given threshold T. The subordinate pass refines this "guess" by coding (again with an arithmetic coder) one of two symbols for each subordinate pass element:

```
• IF ($actual\_magnitude > reconstructed\_magnitude$)
    – Code 1
    – $reconstructed\_magnitude \leftarrow reconstructed\_magnitude + \frac{T}{4}$
• ELSE
    Code 0
    – $reconstructed\_magnitude \leftarrow reconstructed\_magnitude - \frac{T}{4}$
```

After all of the subordinate list elements are refined in this fashion, which amounts to reducing the size of the uncertainty interval to half of its previous size at each pass (see Figure 3.10), the list is sorted, with the elements with the largest *reconstructed_magnitudes* at the front of the list. This ensures that the larger magnitude coefficient values will be refined first.

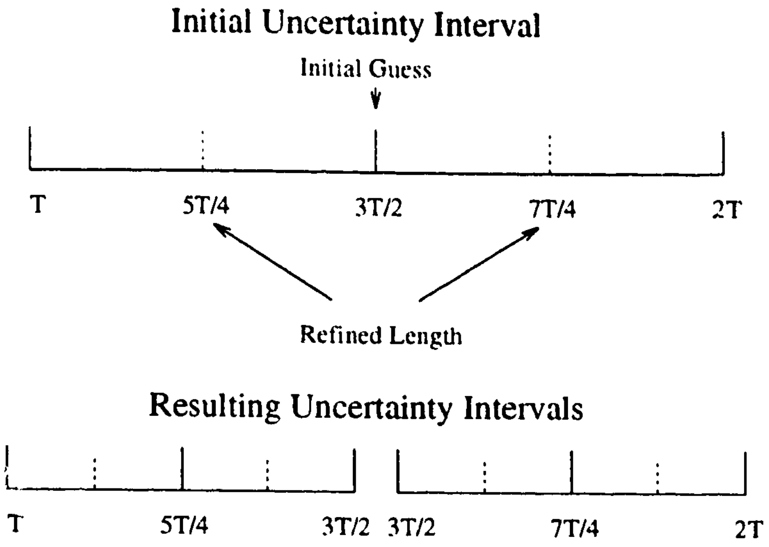


Figure 3.10: Uncertainty interval in the dominant pass and first subordinate pass for a wavelet coefficient.

Decompression of the Data

The decompression of the compressed data is very simple. The steps used in the decompression phase are almost identical to those used in the compression phase. Note that with each symbol that is decoded it is *implicitly known* where changes should be made in the wavelet pyramid given the original *Z*-scan ordering.

The embedded nature of the code should also be obvious at this point. Each additional decoded symbol simply adds another piece of information to the wavelet pyramid. By terminating the data stream before any of the symbols can be read, the result is a *NULL image* containing no information (all coefficients in the wavelet pyramid have magnitude 0). Terminating the data stream after 100 bytes, say, will result in the wavelet pyramid containing 100 bytes worth of coded information. With each additional symbol, the reconstructed signal becomes more complete (with the exception of the IZ and ZT symbols, which serve to reduce the required number of symbols in total, but do not by themselves add directly to the completeness of an image). Consider Table 3.1, which indicates the effect that the decoding of each symbol has on the reconstructed image.

Dominant Pass	
POS/NEG	Coefficient value to within $\frac{T}{2}$
IZ	No effect
ZT	No effect
Subordinate Pass	
0	Coefficient value to within $\frac{T}{4}$
1	Coefficient value to within $\frac{T}{4}$

Table 3.1: Effects of coding each symbol on its corresponding coefficient

Conclusion

The EZW algorithm very effectively deals with the problem of quantizing the wavelet coefficients. The algorithm is very efficient, generates exceptional results, and because of its embedded code, allows for the user to compress an image once in order to test many bit rates.

3.2.3 Set Partitioning in Hierarchical Trees

Another algorithm based on the same basic principles as the EZW algorithm is *Set Partitioning in Hierarchical Trees* (hereafter referred to as SPIHT), proposed by Said and Pearlman[21]. Essentially, SPIHT is a variation of the EZW algorithm which approaches the quantization of the coefficients from another perspective: rather than considering each coefficient individually, coefficients are grouped together into sets based on the same parent-child dependencies used in EZW. In this way, it is possible to code the (in)significance of many coefficients with a single symbol.

The parent-child dependencies suggested by Said and Pearlman are essentially the same as those used by Shapiro with the only difference being the handling of the resolution 0 components (a minor difference since, as was mentioned before, resolution 0 coefficients do not relate to higher resolution coefficients and are included in the structure to preserve continuity in the algorithms). Instead of each resolution 0 coefficient having three descendants as in Figure 3.7, three of every four coefficients has four descendants, as shown in Figure 3.11.

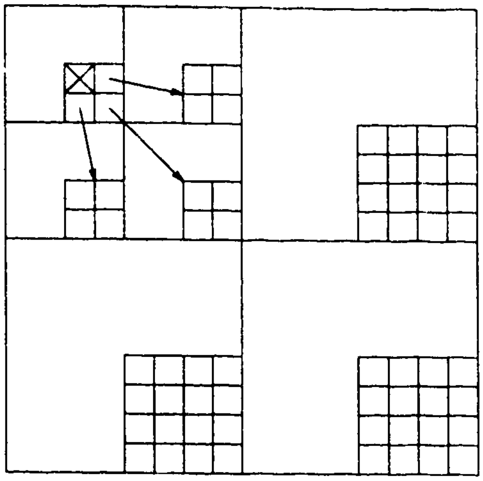


Figure 3.11: Resolution 0 dependencies in SPIHT

The SPIHT algorithm makes use of four types of sets as listed in Table 3.2 below. A graphical illustration of the sets can be seen in Figure 3.12. Each individual coefficient within a set is referred to by its coordinates: $c_{i,j}$. A given set S is said to be *insignificant* with respect to a threshold T if all of the elements in the set are less than T in magnitude. The following function is defined to determine set (in)significance:

$O(i,j)$	contains *all* offspring of node (i,j)
$D(i,j)$	contains all direct descendants of node (i,j)
$L(i,j)$	$D(i,j) - O(i,j)$
H	contains all resolution 0 coefficients

Table 3.2: Sets in SPIHT

$$S_n(S) = \begin{cases} 1 & max\{S\} \leq T \\ 0 & otherwise \end{cases} \tag{3.4}$$

Three lists are used in the algorithm to keep track of the various sets used. They are:

- **LIS** - list of insignificant sets
- **LIP** - list of insignificant pixels

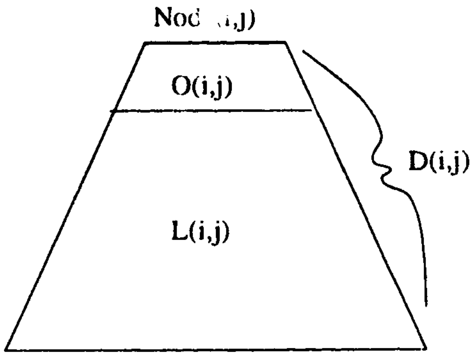


Figure 3.12: Dependency Sets in SPIHT

- **LSP** - list of significant pixels

The elements of both the LIP and LSP are individual pixels while the LIS contains sets of type $D(i,j)$ (type A) or $L(i,j)$ (type B).

The steps in the SPIHT algorithm as described by Said and Pearlman[21] are listed below:

1. **Initialization:** output $\lfloor n = \log_2(max\{\text{coefficients}\}\rfloor$; set the LSP as an empty list and add the coordinates $(i,j) \in H$ to both the LIP and the LIS as type A entries.

2. **Sorting Pass:** (Similar to EZW dominant pass)

   (a) for each entry (i,j) in the LIP do:

      i. output $S_n(i,j)$;

      ii. If $S_n(i,j) = 1$ then move (i,j) to the LSP and output the sign of $c_{i,j}$

   (b) for each entry (i,j) in the LIS do:

      i. if the entry is of type A then

         - output $S_n(D(i,j))$;
         - if $S_n(D(i,j)) = 1$ then
           - for each $(k,l) \in O(i,j)$ do:
             - $*$ output $S_n(k,l)$;

        * if $S_n(k,l) = 1$ then add (k,l) to the LSP and output the sign of $c_{k,l}$;
        * if $S_n(k,l) = 0$ then add (k,l) to the end of the LIP;
      - if $L(i,j) \neq \emptyset$ then move (i,j) to the end of the LIS, as an entry of type B;

   ii. if the entry is of type B then

      - output $S_n(L(i,j))$;
      - if $S_n(L(i,j)) = 1$ then
        - add each $(k,l) \in O(i,j)$ to the end of the LIS as an entry of type A;
        - remove (i,j) from the LIS;

3. **Refinement pass:** (similar to EZW subordinate pass)
   For each entry (i,j) in the LSP, **except** those included in the last sorting pass, output the n-th most significant bit of $|c_{i,j}|$;

4. **Quantization-step update:** decrement n by 1 and go to step 2.

It should be clear that the SPIHT algorithm is very similar to EZW in many respects. In fact, it would appear to be a more compact version of the same algorithm with a few differences. Because of the set structures which are utilized, all of the descendants of a single coefficient can be identified as significant or insignificant with a single symbol. The EZW algorithm requires that *each* immediate descendant of a coefficient be coded in this way. In addition to this, SPIHT requires that the coder require only two symbols at *any* point in the algorithm (note that only 0's and 1's are coded). The refinement pass in SPIHT and subordinate pass in EZW are slightly different as well, but accomplish the same goal: the refinement of coefficients which have already been found to be significant. As with EZW, the decompression algorithm is virtually identical to the compression algorithm, and the spatial location of the effect of each symbol being decoded is implicitly known.

Since the SPIHT algorithm requires only two symbols, it is possible to use straight binary coding rather than arithmetic coding. While the binary coding results are not as good as when arithmetic coding is used, they are still better than the EZW results and require less computational effort as a result.

It should be noted that the SPIHT algorithm does *not* code coefficients which have been included in the last sorting pass. Resources are instead allocated to coding the next sorting pass, with the said coefficients being refined in the next refinement pass. The EZW algorithm, on the other hand, refines the coefficients immediately. This results in the refined coefficients being approximated to within $T/4$ of their actual values, while the remaining insignificant coefficients are guessed to have a value of 0, within T of their actual values. It is much more efficient to code the remaining insignificant coefficients *first* to within $T/2$ (the value of the threshold at the next pass) given that the MSE of the coefficients is closely related to the MSE of the reconstructed image. This is one of the more important improvements of SPIHT over the EZW algorithm.

Because of the similarity in the EZW and SPIHT algorithms and the fact that SPIHT deals with the quantization in a more efficient manner, SPIHT is almost guaranteed to generate better results than EZW for any image without any increase in complexity. Some sample PSNR results for two test images are shown in Table 3.3.

	Lena image		Barbara image	
Bits per pixel	EZW	SPIHT	EZW	SPIHT
1.0	39.55	40.42	35.14	37.45
0.5	36.28	37.22	30.53	32.10
0.25	33.17	34.12	26.77	28.13
0.125	30.23	31.10	24.03	25.37
0.0625	27.54	28.38	23.10	23.77
0.03125	25.38	25.97	21.94	22.71
0.015625	23.63	23.97	20.75	21.79
0.0078125	21.69	22.08	19.54	20.62

Table 3.3: EZW and SPIHT results

3.2.4 Wavelet Combined with Vector Quantization

Because of the advantages of vector based quantization over scalar based quantization, it is natural to extend wavelet compression routines to include vector quantization. In this section, two methods which combine these two approaches will be examined: one which uses the codebook based LGB quantization and the other which uses lattice vector quantization.

LGB Based Vector Quantization

Several methods have been proposed which combine wavelets and LGB based vector quantization. Although there are examples of this type of combination which make use of the zerotree concept[6], the most recent (and most successful) method, which will be described in this section, does not.

A very successful method which combines multiresolution decomposition wavelets with codebook based LGB vector quantization[17] is proposed by Averbuch *et al.*[4]. Their approach to wavelet compression is quite different from that taken by the EZW and SPIHT algorithms in that the parent-child hierarchy and zerotree concepts are not used in any way. Instead, the wavelet subbands are all coded using vectors from codebooks that have been constructed using test images. Two passes are made (the second to code the residual from the first pass) in the quantization as shown in Figure 3.13, each pass requiring a separate codebook.

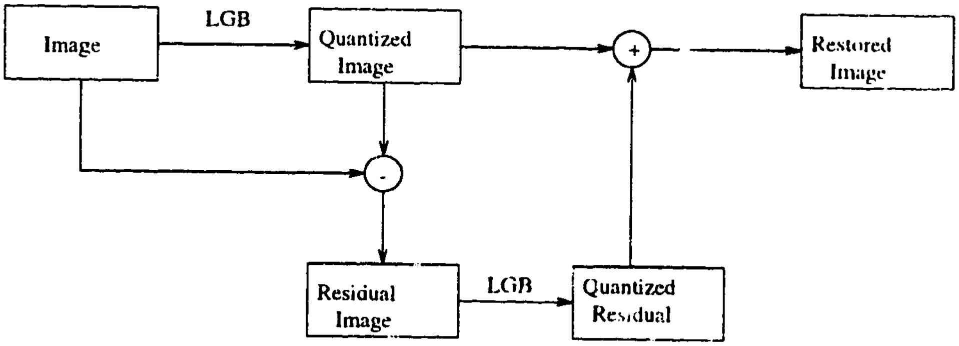


Figure 3.13: Two pass LGB coding

An interesting point is also made in the paper, specifically that using such a scheme will require that 75% of the required space will be taken by the finest three resolutions. In order to take advantage of the importance of the coefficients, their algorithm makes allowances for these three resolutions to be *completely ignored* if lower bit rates are requested. Note the the EZW and SPIHT algorithms already make implicit allowances for the small magnitudes of the finest resolution coefficients and such a scheme would be ineffective if applied to them.

Some drawbacks to this method are obvious when comparing it to EZW/SPIHT. Although the reported results are excellent (comparable to those given by SPIHT), other aspects of the algorithm would seem to make it an undesirable choice. Some of these aspects are:

- Vector codebooks need to be generated on a sample set of images before run time.
- Codebooks must be accessed and searched during run time thus increasing the time complexity of the method to a level which may not be practical if computing speed is an issue (e.g. remote wireless systems).
- Although a terminated data stream could be used to regenerate parts of the original image, the embedded nature of EZW/SPIHT is not supported.
- There is no easy way to set an exact bit rate for a compressed image.

Lattice Vector Quantization

It is desirable to apply a vector quantization scheme to the EZW/SPIHT algorithm in such a way that the advantages of the algorithm would not be lost. This brings us to the question of whether or not codebook based vector quantization would work well when combined with such a scheme. It is possible to use vector quantization in tandem with EZW/SPIHT by grouping coefficients into blocks of, say, 2x2 or 4x4, and treating each block as a single unit. In this way, residual vectors would need to be refined in the refinement pass instead of the coefficient magnitudes. The question remains: *would such a scheme work with codebook based vectors?*

One of the major problems associated with combining wavelet compression with codebook based vector quantization is the generation of acceptable codebooks. Two of the major contributions to wavelet compression techniques shown in combination in the EZW/SPIHT algorithm, is the zerotree concept with iterative refinement of coefficients. These concepts will not work well with codebook-based vector quantization for the following reasons:

1. Residual vectors will result which may not match well with any initial codebooks. Potential (but costly and incomplete) solutions to this could be:
   - The construction of additional codebooks designed with residuals in mind
   - The use of a limited number of passes as in Averbuch at al.[4] This is not particularly desirable.
2. The clustering of vectors as a result of the LGB algorithm may leave *holes* in n-dimensional space which result in poor vector matches. Because of the nature of the iterative refinements, a poor match may result in a block of residual coefficients which may *never* converge

An alternative to LGB vector quantization (and other codebook based variations) is *lattice based* vector quantization. A lattice is a regular, symmetric ordering of points in n-dimensional space. Lattices are often associated with the *sphere packing* problem which asks *"How densely can n dimensional spheres be packed together?"* or the *covering problem* which asks *"What is the most economical way to cover n-dimensional Euclidean space with equal overlapping spheres?"*[5]. An example solution to the first question in 2-dimensional space is shown in Figure 3.14. The centers of the circles form the coordinates of lattice points in what is known as the *hexagonal lattice*.

Because of the regular nature of lattice vectors, lattice vector quantization can be performed very efficiently. Given a particular input vector, the closest lattice "codebook" vector can often be found by simple rounding operations. Sampson and Ghanbari describe simple schemes for using vector quantization with several specific

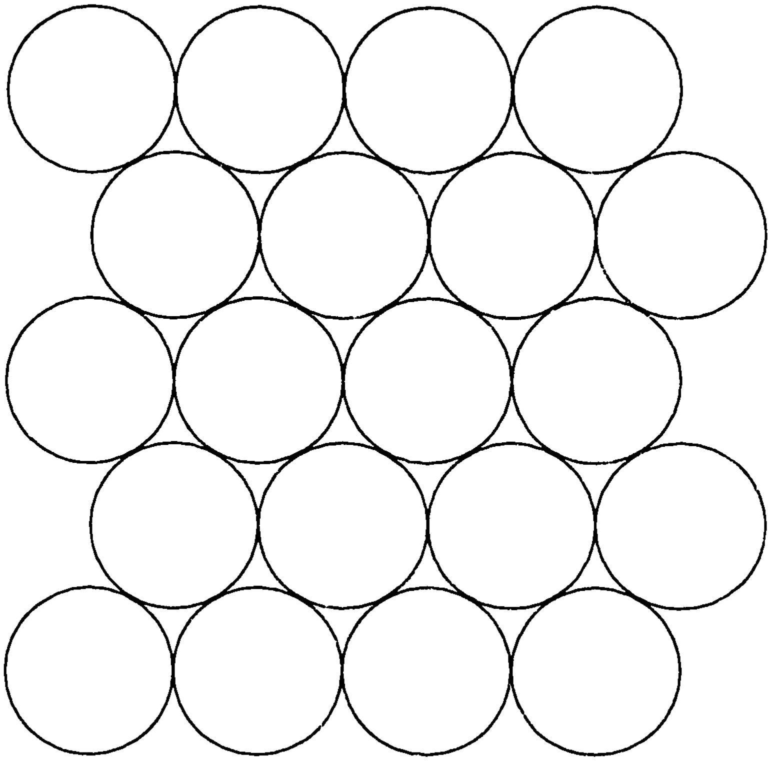


Figure 3.14: Sphere packing in 2-dimensions

lattices[25], and state that the resulting computational complexity of using lattice vectors in their implementation of an image sequence coder is approximately 4% of that of a similar LGB coder.

An important set of lattices which turns out to be very useful to wavelet compression is the set of *root lattices*. These types of lattices result in vector points being regularly distributed on the surfaces of concentric hyperspheres or *shells*[25]. The result of taking one or more of these shells to be a vector "codebook" is a regular, (hyper)spherical arrangement of vectors.

would seem then that the characteristics of lattice vector quantization will solve at least some of the problems associated with LGB vector quantization. Residual vectors should match just as well as original vectors since the vector distribution is regular and the regularity of the lattice should reduce the number of "holes" in the codebook.

A variation of the EZW algorithm described by Sampson *et al.*[24] makes use of lattice vectors to further quantize wavelet coefficients in the proposed manner. Wavelet coefficients are clustered into 4, 8 and 16 dimensional vectors and then quantized using lattice vector quantization. The quantization makes use of an implicit threshold and unit vectors from a *lattice codebook*, the actual code vector being the normalized lattice vector scaled by the current threshold. Thus, the shape, not the length, of each lattice vector is important.

Only vectors in the first shell of the the three root lattices described below (although in the case of D_4, the second shell will be of some interest later on) are used by Sampson *et al.*[24]:

1. **D_4 lattice: 4-dimensional**

   - Vectors in this lattice have the property that their components add up to an even number. Thus: $\langle 1,0,0,0\rangle \notin D_4, \langle 1,1,0,0\rangle \in D_4$
   - The first (and second) shells contain 24 vectors
   - First shell vectors have a length of $\sqrt{2}$, second shell vectors have a length of 2.

2. **E_8 lattice: 8-dimensional**

   - Vectors in this lattice have the following property:

   $$\vec{v} \in E_8 \Leftrightarrow \begin{cases} \vec{v} \in D_8 & or \\ \vec{v} \in (D_8 + \vec{\tfrac{1}{2}}) \end{cases}$$

   where D_8 has exactly the same properties as D_4 except that it is 8-dimensional, and $\vec{\frac{1}{2}} = \langle \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \ldots\rangle$.

   - The first shell of this lattice contains 240 vectors
   - First shell vectors have length $\sqrt{2}$

3. **Λ_{16} lattice: 16-dimensional**

   - Known as the *Barnes-Wall lattice*, the first shell vectors take the form $\frac{1}{\sqrt{2}}\langle \pm 2^2, 0^{14}\rangle$ and all permutations (480 vectors), and $\langle \pm 1^8, 0^8\rangle$ where the positions of the nonzero elements correspond to the nonzero elements of the first-order weight 8 Reed-Muller code [5, 19] in addition to having an even number of negatives. The basis vectors for the Reed-Muller codes of interest are as follows:

$$\langle 0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1 \rangle$$
$$\langle 0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1 \rangle$$
$$\langle 0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1 \rangle$$
$$\langle 0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1 \rangle$$

Note that in order to obtain all of the 30 possible codewords, cyclic shifts must be performed on the above vectors.

- The first shell of this lattice contains 4320 vectors

In addition to using the above lattice vectors, the algorithm[24] contains one notable difference to the EZW algorithm: it does not divide the threshold by 2 at each pass; rather, it is multiplied by a real constant $a < 1$. Thus, at step n of the refinement process, the approximation $\hat{\vec{x}}$ to an original vector $\vec{x}$ can be described as follows:

$$\hat{\vec{x}} = a\vec{x}_1 + a^2\vec{x}_2 + \ldots + a^n\vec{x}_n$$

where x_i is the codevector used at refinement i. The reason for the use of this constant is simply to ensure the convergence of the vector refinements as shown in Figure 3.15.

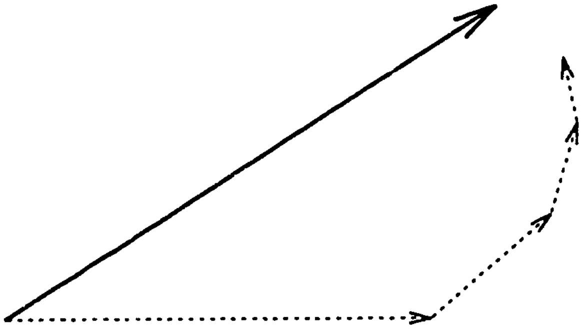


Figure 3.15: Convergence of vectors

The steps in the modified algorithm are described below:

```
1. Initialization: $T \leftarrow a * \text{maximum\_vector\_magnitude}$

2. Dominant pass equivalent:

    (a) While (bit budget not exceeded) Scan vectors for zerotree generation:
        if ($||\vec{x}|| < T$) code IZ or ZT as appropriate
        else replace with closest orientation codevector multiplied by $T$

3. Subordinate pass equivalent:

    (a) $T \leftarrow aT$

    (b) Refine nonzero vectors
```

This again brings up an important point about using vector quantization of any form that differentiates it from the scalar quantization methods: a vector which results in a poor match will often never converge especially if the a constant is too small. Consider the following example where we have a 4-dimensional input vector of the form $\vec{v} = \langle 1, 1, 1, 0.9\rangle$, we are using the D_4 lattice, $a = \frac{1}{2}$ and $T = 1$. Since $||\vec{v}|| \geq T$, the vector will be coded at this refinement (and will not have been coded in the previous refinement, since $||\vec{v}|| < 2T$). The closest normalized first shell D_4 lattice vector (and since $T = 1$ the codevector which will be used) is one of the permutations of $\vec{\hat{v}} = \langle \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0\rangle$. The residual vector will then be $\vec{r} = \langle 1 - \frac{1}{\sqrt{2}}, 1 - \frac{1}{\sqrt{2}}, 1, 0.9\rangle$. Since $||\vec{r}|| \approx 1.408$ is already greater than the *current* value of T (which, with $a = \frac{1}{2}$ will be the limit of the remaining values of T), the residual will never converge to zero.

The results of the EZW/lattice combination can be seen in the table below taken directly from the paper by Sampson *et al.*[24]. The results in many cases appear to be superior to those of EZW especially for the Λ_{16} lattice. Clearly, the higher dimensional lattice vectors yield significantly better results than the lower resolution vectors.

Test image	D_4	E_8	Λ_{16}	EZW
Barbara	29.36	30.60	30.90	29.03
Boats	34.19	34.78	35.24	34.29
Girl	35.27	35.91	36.12	35.14
Gold	31.01	32.76	32.61	32.48
Zelda	38.43	39.36	39.44	39.08
Lena (256x256)	30.13	30.15	30.29	30.06

Table 3.4: EZW/lattice results for images at 0.4 bpp

3.3 A New Wavelet/LVQ Algorithm: ModLVQ

In this section, a new wavelet compression algorithm which makes use of a modified form of lattice vector quantization will be described. Since the most successful scalar based multiresolution wavelet quantization algorithm is the SPIHT algorithm[21], this will be used as a basis for the new algorithm. In addition to combining this algorithm with a modified form of lattice vector quantization, the new method will look at minimizing the residuals of the vectors as well as eliminating additional redundancies within SPIHT. In total, three changes will be made to the existing SPIHT algorithm:

1. The inclusion of vector quantization in a manner similar to that presented by Sampson *et al.*[24]

2. The further addition of a vector residual minimization scheme

3. The elimination of existing redundancies in the SPIHT algorithm

The new algorithm will hereafter be called *ModLVQ* as it will include a modified lattice vector quantization scheme as its major component.

The resulting algorithm can be shown to be significantly numerically superior (in terms of PSNR) to the SPIHT algorithm for detailed images and comparable to SPIHT for smooth images. In addition to this, there is often a significant visual improvement in reconstructed versions of high detail images over SPIHT.

3.3.1 Modified Lattice Vector Quantization

The SPIHT algorithm is based on a hierarchical structure which considers single coefficient elements. This structure can be easily extended to handle multidimensional elements, assuming that the elements are *square*. That is, the algorithm can accommodate the use of 2x2 or 4x4 blocks (or vectors) of coefficients. Figure 3.16 shows how the structure would appear with 2x2 blocks.

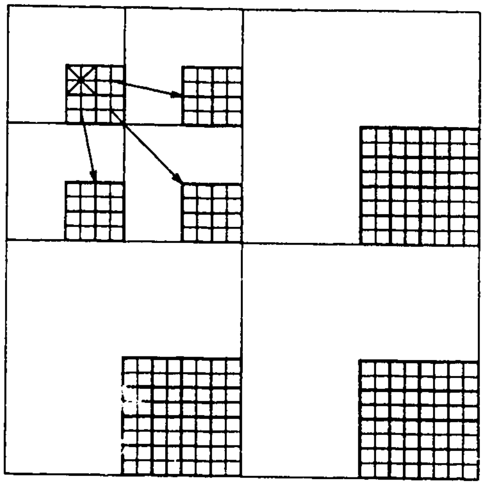


Figure 3.16: Organization of 2x2 vectors in the SPIHT hierarchy

Using the E_8 lattice vectors is not quite as simple. It is not immediately clear how 8-dimensional vectors could be incorporated into the hierarchical structure without substantial changes to parts of the algorithm. For this reason, only the D_4 and Λ_{16} (in modified form) lattices were used in the *ModLVQ* algorithm.

Lattice vector quantization, as mentioned previously, can be a very efficient and effective quantization method and has been shown to work well when combined with wavelet quantization. However, there are some problems associated with lattice vector quantization in general, and also in the way in which it has been applied to wavelet compression specifically:

1. The distribution of the lattice vectors makes no allowances for the nature of the input vectors themselves.

   - Although the distribution of lattice vectors is regular, it is by no means even. There are *holes* in lattice structures. If a certain vector orientation resulting from the wavelet coefficients happens to regularly fall into one of these holes, large residuals will result.

- This is perhaps the most undesirable characteristic of lattice vector quantization. By holding to the strict rules of the lattice, certain vector types will not be approximated very well by their lattice codevectors.

2. Some "holes" in the first shell lattice structures may be filled in. Some suggestions include:

    - The D_4 lattice could be expanded to include second shell lattice vectors, doubling the number of codevectors from 24 to 48.
    - The Λ_{16} lattice could be expanded to include the Reed-Muller codeword structures with even *and* odd numbers of negatives. This would almost double the number of codewords from 4320 to 8160.

One potential solution to the problem of the lattice vectors not accommodating the input vectors very well might be a combination with LGB vector quantization or one of its variations. This would, however, increase the complexity of the algorithm significantly and require training sets to be tested in advance of compression. One of the major advantages of lattice vector quantization techniques, as was previously discussed, is that they do not require this additional complexity.

Another option is to add additional lattice or non-lattice vectors to the lattice codebook. Such vectors, if they were non-lattice, would need to have the following characteristics:

- The vectors' nature must be such that they can be efficiently tested for matches with input vectors.
- The vectors must be designed to match input vector forms that appear consistently within the wavelet coefficients but are not well handled by the original lattice.

Another consideration is that the number of additional vectors cannot be too high: a relatively low number of symbols is required in order for an arithmetic coder to effectively compress the information.

The nature of the coefficient distribution (Figure 3.3) would seem to indicate that, in many cases, vectors will consist of a single *dominant component* which causes the vector to become significant. Observation of the coefficient vectors confirms this hypothesis. In such a case, the best vector form which would match this would be $\langle \pm 1^1, 0^3 \rangle$ for 4-dimensional vectors and $\langle \pm 1^1, 0^{15} \rangle$ for 16-dimensional vectors.

By simply including the second shell D_4 vectors to the 2x2 block coding, this would be taken care of - $\langle \pm 2^1, 0^3 \rangle \in D_4^{shell2}$, and since the vector quantization would only consider the shapes of the vectors, this is equivalent to $\langle \pm 1^1, 0^3 \rangle$. Because they are still D_4 lattice vectors, this addition will have no significant adverse effects on the complexity of the quantization.

The Λ_{16} lattice is somewhat different. It has no form in its first shell which matches $\langle \pm 1^1, 0^{15} \rangle$ - the closest form being $\langle \pm 1^2, 0^{14} \rangle$, which is not a very good approximation. In addition to this, the second shell of the lattice contains 61440 vectors. This quantity of vectors is too many for an arithmetic coder, eliminating the possibility of using the same idea as was used for D_4. In this case, the most sensible course of action is to simply add the 32 non-lattice vectors of the form $\langle \pm 1^1, 0^{15} \rangle$ to the coding algorithm. Since there are only a small number of additional vectors which are very easy to compare with input vectors for potential matches, this addition has no adverse effect on complexity.

The final form of the new Λ'_{16} vectors including the non-even form of the Reed-Muller shape vectors and the 32 *dominant component* vectors will have the following forms:

1. 32 vectors: $\langle \pm 1^1, 0^{15} \rangle$

2. 480 vectors: $\frac{1}{\sqrt{2}} \langle \pm 2^2, 0^{14} \rangle$

3. 7680 vectors: $\langle \pm 1^8, 0^8 \rangle$, where the positions of the nonzero elements correspond

to the nonzero elements of the first-order weight 8 Reed-Muller code and there are no restrictions on the number of negatives.

In total, using this new scheme, Λ'_{16} contains 8192 vectors.

Because the addition of more vectors implies that more space, on average, will be taken by the arithmetic coder to encode each vector index, there will be two shifts on the R-D curve as a result of this modification to the vectors. The first shift will come as a result of the increased space required (see Figure 3.17), which gives an increased rate. The second will come as a result of better matches between input vectors and codevectors, which results in decreased distortion (see figure 3.18). In mathematical terms, we have an increase in rate, ΔR, and a decrease in distortion ΔD. In order for this modification to be successful, $-\frac{\Delta D}{\Delta R}$ must be sufficiently large to lead to an overall downward shift in the R-D curve as shown in Figure 3.19. By choosing "better" vectors (vectors that will match better with the coefficient vectors), the magnitude of ΔD increases and the R-D curve will shift more downward.

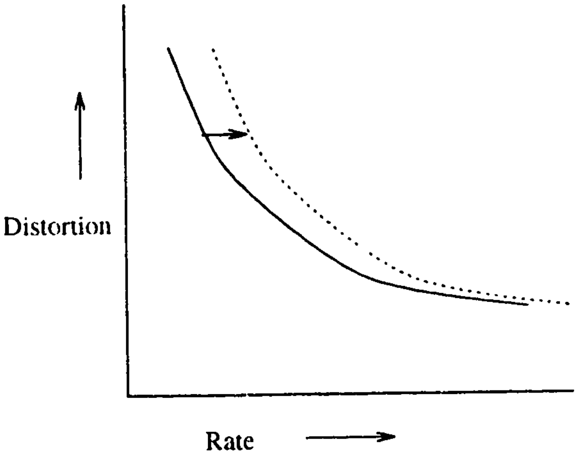


Figure 3.17: R-D curve shift: increase in ΔR

3.3.2 Clarification of Vector Refinements

It is not entirely clear in the EZW/lattice combination by Sampson *et al.* that any allowances are made for "insignificant" residual vectors in the subordinate (or refinement) pass, although it seems likely that this was implemented. Such a case may

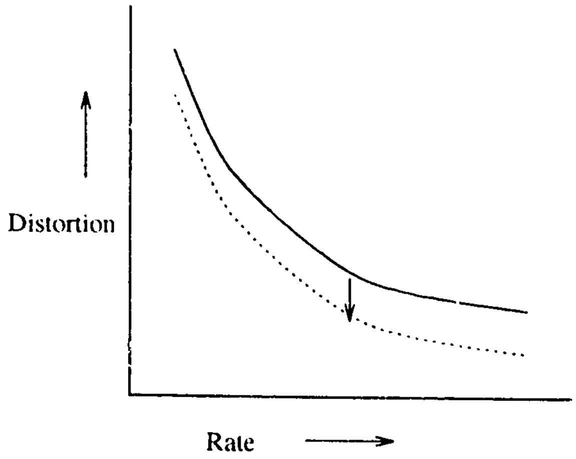


Figure 3.18: R-D curve shift: decrease in ΔD

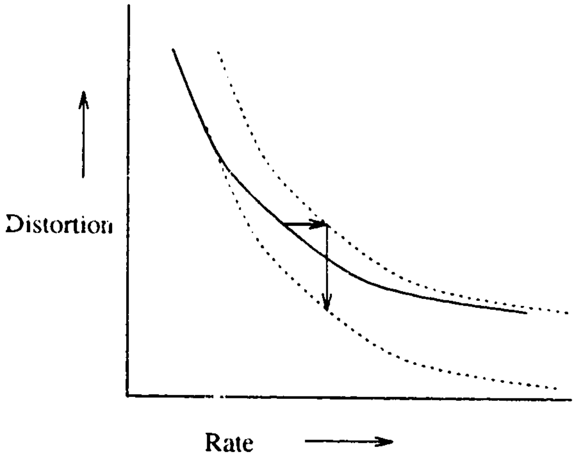


Figure 3.19: R-D curve shift: favourable $\frac{\Delta D}{\Delta R}$

arise, for example, when a vector quantization step results in a very small residual (smaller in magnitude that the current threshold). Such an insignificant vector should NOT be coded with a codevector until a later pass as the residual may increase in size as a result. (see Figure 3.20).

Thus, during the refinement pass, one of two symbols would need to be coded in addition to the vector indices:

1. **Significant**

   Indicates that the current residual is significant. This symbol will be followed by a codevector index.

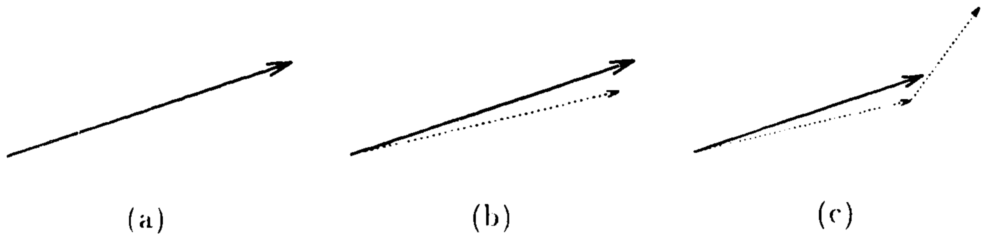


Figure 3.20: Consequence of coding insignificant vectors: (a) The original vector; (b) The original is matched well by a codevector leaving a small residual; (c) The small residual is vector quantized even though it is insignificant, resulting in a larger residual

2. **Insignificant**

   Indicates that the current residual is not significant. The codevector index will not be coded.

3.3.3 Minimizing the Lattice Vector Residuals

One aspect of using vectors to refine wavelet coefficients that has been overlooked is the simple issue of minimizing the residual by coding a vectr r at the correct time. The residual of a vector can be theoretically minimized using basic linear algebra which tells us that, given an initial vector and an arbitrary length codevector, the optimal length of the codevector for the minimization of the residual will be the projection of the initial vector onto the codevector as in Figure 3.21.

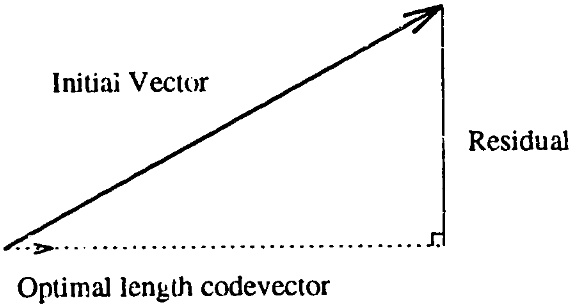


Figure 3.21: Residual Optimization

Using an algorithm similar to that suggested by Sampson *et al.*[24] , however, we do not have the option of setting the codevector to the exact length which would minimize the residual. To do so would be very costly in terms of space as the algorithm handles the length of each vector implicitly which requires no storage. However, using a similar scheme to the one described in the previous section which either codes each vector as significant or insignificant, we can choose the best time to quantize a given input vector.

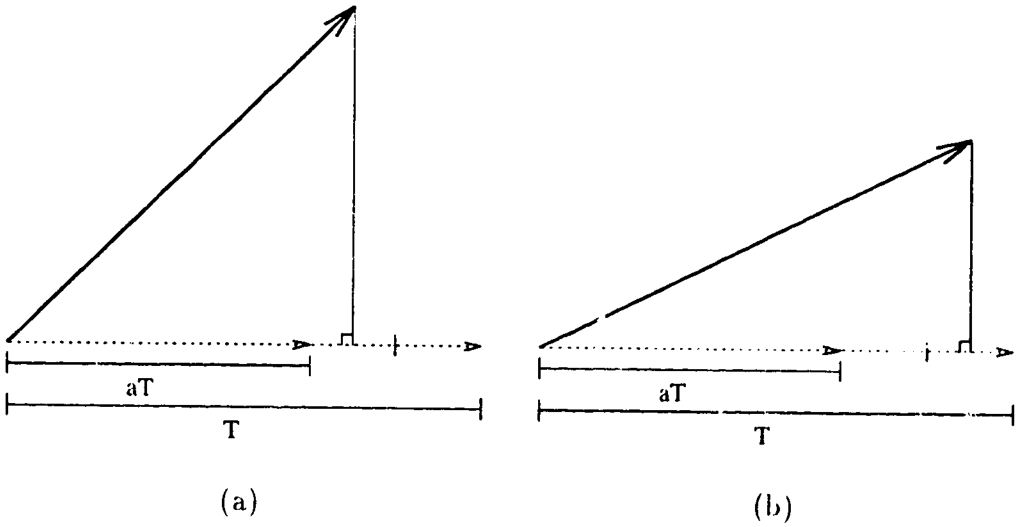
There are two cases where it might be desirable to code a given vector at a different pass than it otherwise would be coded. The two cases are:

1. **Short projection**

   - Can occur when an input vector is poorly matched by a codevector
   - The residual would become smaller if the vector were coded using the same codevector in the *next* refinement. That is, given the current threshold T, the input vector $\vec{x}$ and the normalized codevector $\hat{\vec{c}}$,

$$\vec{x} - aT\hat{\vec{c}} < \vec{x} - T\hat{\vec{c}} \qquad (3.5)$$

   See Figure 3.22.



(a)

(b)

Figure 3.22: Poorly matched vector: (a) a short projection which would be better matched in the next pass; (b) a normal projection which is optimally matched in the current pass

2. **Long projection**

- Can occur if an input vector is very well matched by a codevector
- The residual would become smaller if the vector were coded using the same codevector in the *previous* refinement. That is, given the current threshold T, the input vector $\vec{x}$ and the normalized codevector $\hat{\vec{c}}$,

$$\vec{x} - \frac{T}{a}\hat{\vec{c}} < \vec{x} - T\hat{\vec{c}} \tag{3.6}$$

See Figure 3.23.

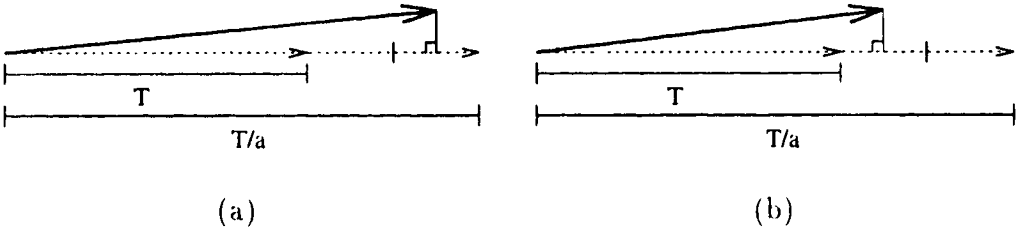


Figure 3.23: Well matched vector: (a) a long projection, which would have been better coded in the previous pass; (b) a normal projection which is optimally coded in the current pass

It should be clear that the factor which determines when to optimally code a vector is the projection length. If the projection of the initial vector onto the codevector is greater in length than the midpoint of aT and T, it should be coded at the current threshold. Otherwise, the coding is postponed until a subsequent pass with a smaller threshold.

A simple refinement to the algorithm can be made to accommodate the optimization of the residuals. Instead of considering the vector length as the criterion for "significance" (which might be better termed *optimal significance* in light of the proposed modifications), the projection length would be considered. As before, two symbols in addition to the codevector indices will be required to indicate whether or not to code the vector at this pass. They will be renamed from *significant* and *insignificant* to *optimally significant* and *optimally insignificant*:

- IF $(T\frac{a+1}{2} < projection\_length)$

  Vector is *optimally significant*

- ELSE

  Vector is *optimally insignificant*

There are some potential problems associated with this modification however. While each vector will be coded with an optimal length codevector, the potential changes have to be considered within the framework of the entire algorithm. The following problems could arise:

1. Coding of significant vectors could be left to a subsequent pass which might never occur (it could be the final pass).

   Smaller vectors will often be coded before much larger ones.

3. The coding of additional *optimally insignificant* symbols will now take place *before* the coding of many vectors which would otherwise require only a single *significant* symbol. That is, the following steps in coding a single vector could change from the non residual-optimized:

   (a) Code *significant*

   (b) Code vector index

   (c) ... *Next pass* ...

   to the residual-optimized:

   (a) Code *optimally insignificant*

   (b) ... *Next pass* ...

   (c) Code *optimally significant*

   (d) Code vector index

Because of this, there will be an increase in the rate as well as a decrease in distortion $(-\frac{\Delta D}{\Delta R})$, and similar shifts to those described for the modified lattice vector quantization will occur on the R-D curve. Again, the ratio of ΔD to ΔR needs to be favourable in order for this modification to be successful.

3.3.4 Further Minimization of Redundancy

The SPIHT algorithm is in many ways a replica of the EZW algorithm with several improvements which reduce redundancies in the quantization resulting in a more effective compression scheme. Similarly, by eliminating remaining redundancies in the SPIHT algorithm, superior results can be achieved. One such existing redundancy in the algorithm will be eliminated.

Consider steps (b)i and (b)ii in the SPIHT algorithm. When an entry $D(i,j)$ of type A is found to be significant, each $(k,l) \in O(i,j)$ is tested for significance. Note that $D(i,j) = O(i,j) \bigcup L(i,j)$, and $O(i,j) \bigcap L(i,j) = \emptyset$. From this, it should be clear that:

$$\forall x \in O(i,j) S_n(x) = 0 \Longrightarrow \exists y \in L(i,j) \ s.t. \ S_n(y) = 1$$

From this, we know implicitly that $S_n(L(i,j)) = 1$, and hence there is no need to code any symbol for $S_n(L(i,j))$ since it will be implicitly known.

In such a case, the steps associated with (b)i and (b)ii in the SPIHT algorithm need to be modified slightly. To deal with the situation, an attribute will be added to each set element which will be set as follows:

$$NoCode(i,j) = \begin{cases} 1 & \forall x \in O(i,j), S_n(x) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

This attribute, of course, will indicate when the value of $S_n(L(i,j))$ is implicitly known to be 1.

Obviously, this redundancy elimination is minor compared to the changes which mark the differences between the EZW and SPIHT algorithms. However, such an improvement is guaranteed to reduce the space required to code an image to a given quality. That is, there is a reduction in the rate (a negative ΔR) without any associated increase in distortion which is equivalent to a simple leftward shift of the R-D curve. In addition to this, checking for this redundancy requires a minimum of computational effort.

3.3.5 A Natural Extension to SPIHT

Given the modifications described above, the combination of lattice vector quantization and SPIHT can be described in steps similar to the original SPIHT algorithm except that vectors are considered instead of single coefficients. The significance function will need to be redefined in terms of the threshold T and the new significance rules:

$$S_T(S) = \begin{cases} 1 & \exists s \in S \text{ s.t. } s \text{ is optimally significant} \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

The set notation would remain basically the same except that there is no distinction between significant and insignificant vectors since the refinement of residual vectors will be handled in the same way as the initial coding of a coefficient vector.

- **LIS** - list of insignificant sets
- **LV** - list of potentially significant vectors

Thus, the modified aigorithm would have the following form:

1. **Initialization:** output $T = a * max\{coefficient\_vector\_norms\}$; add the coordinates of the vectors $(i,j) \in H$ to both the LV and the LIS as type A entries.

2. **Sorting Pass:**

   (a) for each entry (i,j) in the LV do:

      i. output $S_T(i,j)$;

      ii. If $S_T(i,j) = 1$ then code the codevector index and put the residual vector back into the coefficient array;

```
(b) for each entry $(i,j)$ in the LIS do:
    i. if the entry is of type A then
        • output $S_T(D(i,j))$;
        • if $S_T(D(i,j)) = 1$ then
            – for each $(k,l) \in O(i,j)$ do:
                * output $S_T(k,l)$;
                * if $S_T(k,l) = 1$ then add $(k,l)$ to the end of the LV, code the
                  codevector index, and put the residual vector back into the
                  coefficient array;
                * if $S_T(k,l) = 0$ then add $(k,l)$ to the end of the LV;
            – if $L(i,j) \neq \emptyset$ then move $(i,j)$ to the end of the LIS as an entry
              of type B;
            – if $(\forall (k,l) \in O(i,j), S_n(k,l) = 0), NoCode(k,l) \leftarrow 1$
              else $NoCode(k,l) \leftarrow 0$
    ii. if the entry is of type B then
        • if $(NoCode(k,l) = 0)$ output $S_T(L(i,j))$;
        • if $(NoCode(k,l) = 1$ or $S_T(L(i,j)) = 1)$ then
            – add each $(k,l) \in O(i,j)$ to the end of the LIS as an entry of type
              A;
            – remove $(i,j)$ from the LIS;
```

```
3. Quantization-step update:
    • $T \leftarrow aT$;
    • go to step 2.
```

Notice that the refinement pass has been eliminated from the algorithm. The refinement of the vectors is done in the sorting pass since the residuals of each quantized vector are placed back into the coefficient array. This does not adversely affect the results since each vector which is quantized never needs to be checked for inclusion in a zerotree structure again. The only time a residual vector is coded is when it is refined and checked for *optimal significance*.

3.4 ModLVQ Compression Results

The suggested modifications which were outlined in the previous section result in substantial changes to the quality of the reconstructed images. In this section, the impact of the modifications will be measured mathematically (in terms of PSNR) and visually, image classes on which the *ModLVQ* algorithm performs well will be identified, and the relevance of the res'lts will be discussed.

3.4.1 Test Image Set

In order to make an adequate assessment of the performance of a compression algorithm, as was mentioned in the previous chapter, a wide range of test images and test image types must be used. The same test image set as was used for the quadtree compression algorithm is used to test the *ModLVQ* method. The four images shown in Figure 2.9 will be used often in the analysis of the *ModLVQ* algorithm.

3.4.2 Modification to Short Projection Optimization

Tradeoffs are an important part of image compression. As was discussed several times previously, there are places in the suggested modifications where there is a tradeoff between resulting reductions in distortion and increases in rate ($-\frac{\Delta D}{\Delta R}$). In order for the modification to be successful, the magnitude of the ratio of ΔD to ΔR must be sufficiently large (see Figure 3.19).

In almc^t all of the cases, the tradeoff came out in favour of the proposed modifications. In one particular case, that of the short projection *optimal significance*, the ratio $-\frac{\Delta D}{\Delta R}$ was not favourable, and caused a slight overall decrease in image quality for a given bit rate. The decrease in quality was not large (in general,

$\Delta PSNR \in \{-0.2, \ldots, -0.1\}$), but was consistent throughout the test image set. Because of this, the short projection vectors must be removed from the definition of *optimally significant* by assuring that if a vector magnitude is greater than the current threshold, it will always be *optimally significant*:

- IF ($T\frac{a+1}{2} < \mathit{projection\_length}$ OR $T \leq \mathit{vector\_length}$)
  Vector is *optimally significant*
- ELSE
  Vector is *optimally insignificant*

The improvement in image quality as a result of this addition as compared to some of the others is small, and thus the calculation of optimal significance is removed everywhere in the algorithm *except* in the steps shown in 2(a) of the algorithm dealing with the potentially significant vectors themselves. This eliminates the need for additional processing during the algorithm with imperceptible loss to the improvements in the resulting image quality.

3.4.3 Numerical Results

In this section, the numerical results of the *ModLVQ* algorithm will be discussed in detail. The fidelity criterion used to determine image quality will, as before, be the PSNR. The overall changes will be discussed first with concentration on the successes and limitations of the algorithm. Later in the section, the various modifications to SPIHT will be analyzed separately as to their impact on the algorithm as a whole.

Overall Results

The single largest change to the SPIHT algorithm is, of course, the addition of the vector quantization scheme. Although the addition of lattice vectors to the EZW algorithm produces better quality reconstructed images, there was no guarantee that the same combination with SPIHT would also result in an improvement, even with the knowledge that SPIHT and EZW are closely related.

Consider the gap between the EZW and SPIHT results which is often greater than 1 dB PSNR in favour of SPIHT. The addition of lattice vector quantization to the EZW algorithm, especially in the case of the Λ_{16} lattice, is very effective as its use in place of the regular scalar quantization results in an advantage. However, with the more optimized SPIHT algorithm it is possible that the scalar quantization algorithm in place could already be generating a more efficient code than would result with the addition of lattice vectors.

As it turns out, the *ModLVQ* algorithm can result in higher quality images *for a certain image class* when the Λ_{16} vectors (and the modified combinations) are added. The D_4 lattice vectors generally reduced the quality of the resulting images although it was easily verified that the use of the first two shells produced better results than using only the first shell. In general, image quality for busy images increases with the addition of Λ_{16}' vectors while there is a general decrease in quality for smooth images. How can this phenomenon be described mathematically?

Given that high frequency wavelet coefficients are distributed about zero, the mean absolute norm (MAN) of the coefficients is easily calculated as the average absolute coefficient value,

$$MAN = \frac{1}{N}\sum_{i=1}^{N}|c_i| \qquad (3.9)$$

where N is the number of coefficients, and c_i are the coefficient values. Since only high frequency coefficients are used in the analysis, it makes sense that the higher the value of the MAN, the larger (on average) the high frequency wavelet coefficients are, and hence the more detail exists in the original image. Another method, of course, is to calculate the mean-squared norm (MSN) of the high frequency coefficients:

$$MSN = \frac{1}{N}\sum_{i=1}^{N}c_i^2 \qquad (3.10)$$

Using the MSN as a measure of the amount of high frequency information in the coefficients can be criticized since a few very large coefficients can result in a larger MSN than many moderately-sized coefficients. For this reason, I will use the MAN in my analysis.

Consider the graph in Figure 3.24. The horizontal axis shows increasing MAN of the high frequency wavelet coefficients and the vertical axis indicates the change in PSNR which occurs as a result of using *ModLVQ* as opposed to SPIHT. The $\Delta PSNR$ measure is taken at 0.125 bpp on a set of 28 test images. Visually, there appears to be a loose correlation between the MAN and $\Delta PSNR$ values.

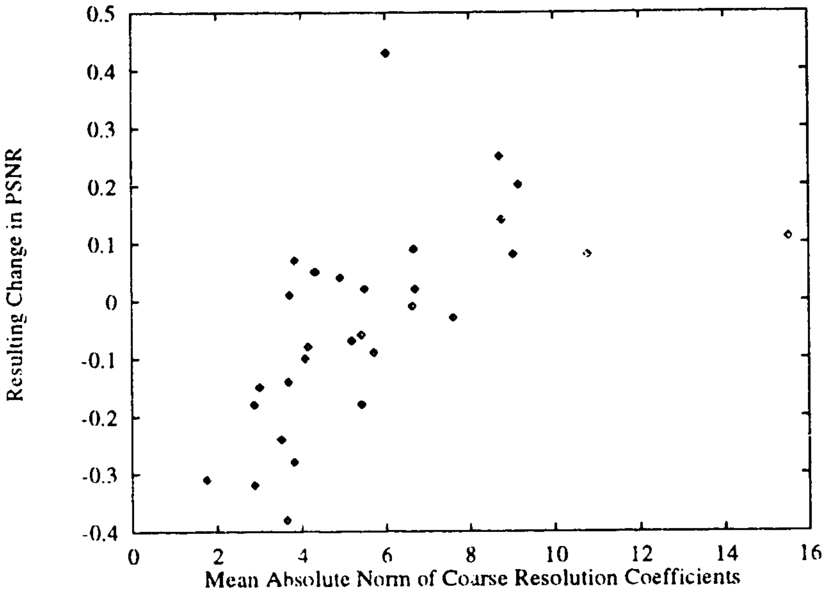


Figure 3.24: Coefficient Mean Absolute Norm vs PSNR correlation

A visual test of the data, however, does not prove the correlation. Statistically, the correlation, ρ, between two variables can be calculated in the following way:

$$\rho = \frac{Cov(Y_1, Y_2)}{\sigma_1 \sigma_2} \tag{3.11}$$

$$Cov(Y_1, Y_2) = E(Y_1 - \mu_1)E(Y_2 - \mu_2) \tag{3.12}$$

The resulting correlation will always satisfy the inequality $-1 \leq \rho \leq 1$, with a 1 (or -1) implying perfect correlation, and 0 implying no correlation In this case, the two sample populations Y_1 and Y_2 are the MAN and $\Delta PSNR$. Since the data points seem to be related in a positive fashion in Figure 3.24, the expected value of ρ should be positive. Using 168 (MAN, $\Delta PSNR$) pairs generated from 28 test images, the statistical correlation works out to be close to 0.45 showing a loose correlation between the two variables. By removing the two images which correspond to the obvious outlying points in Figure 3.24 from the test set, the correlation becomes 0.51

A comparison of the reconstructed image quality of EZW, SPIHT, and *ModLVQ* for the Lena and Barbara images is shown below in Table 3.5 where the values in the table are in dB PSNR. Note that the Lena image is relatively smooth while the Barbara image contains considerably more detail and is more difficult to compress. In addition to this, the SPIHT and *ModLVQ* results are shown for the Io and Mandrill images in Table 3.6.

	Lena image			Barbara image		
Bits per pixel	EZW	SPIHT	ModLVQ	EZW	SPIHT	ModLVQ
1.0	39.55	40.42	40.47	35.14	37.45	37.52
0.5	36.28	37.22	37.02	30.53	32.10	32.50
0.25	33.17	34.12	33.99	26.77	28.13	28.95
0.125	30.23	31.10	30.81	24.03	25.37	26.16
0.0625	27.54	28.38	28.26	23.10	23.77	24.20
0.03125	25.38	25.97	25.92	21.94	22.71	22.79
0.015625	23.63	23.97	24.01	20.75	21.79	21.67
0.0078125	21.69	22.08	22.17	19.54	20.62	20.62

Table 3.5: EZW, SPIHT and *ModLVQ* results

	Io image		Mandrill image	
	SPIHT	ModLVQ	SPIHT	ModLVQ
[illegible]	[illegible]40	36.12	29.17	29.46
[illegible]	32.90	32.61	25.64	25.78
[illegible]	30.36	30.08	23.27	23.44
0.125	28.21	28.07	21.72	21.91
0.0625	26.42	26.36	20.74	20.82
0.03125	25.01	24.95	19.58	20.08
0.015625	23.67	23.68	19.57	19.59
0.0078125	22.41	22.48	19.18	19.18

Table 3.6: SPIHT and *ModLVQ* results

Because of the increased number of large high frequency coefficients, busy images generally give lower quality (in terms of PSNR) reconstructed images when *any* compression scheme is applied to them. Notice the general trend in the above tables, which shows that the busier images tend to be better compressed with *ModLVQ* than SPIHT, which is in agreement with the statistical data presented earlier.

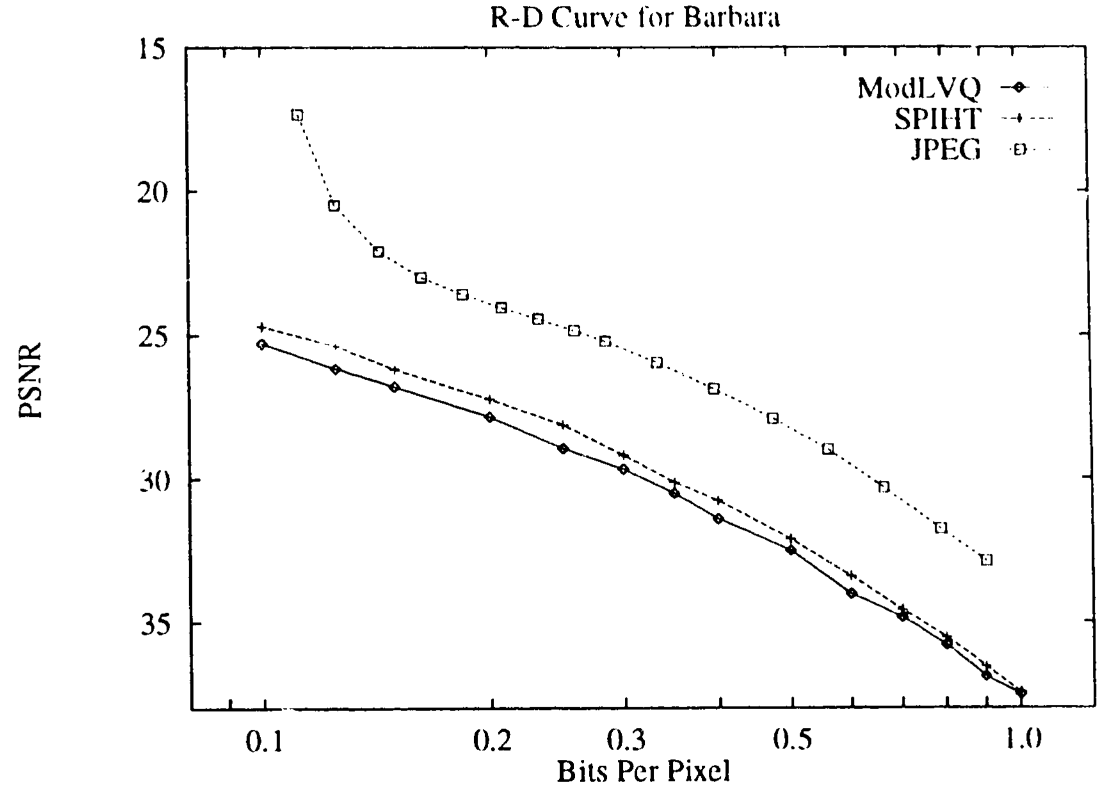


Figure 3.25: A comparison of SPIHT, *ModLVQ* and JPEG

A comparison to JPEG compression is given in the graph shown in Figure 3.25 rather than in table form since JPEG cannot be used to exactly specify a bit rate. Only one graph comparing *ModLVQ* and SPIHT to JPEG will be given as it should be obvious that JPEG cannot compete with these methods at any bit rate.

Effects of the Modifications

There were several modifications which were made to the SPIHT algorithm which resulted in the *ModLVQ* algorithm. It is clear that the most prominent change was the addition of Λ_{16} (or in the final case, Λ'_{16}) vectors. In order to determine the exact impact of each modification, a baseline must be established with the results of the addition of each modification compared to the results of using the baseline method. I will use two separate baselines for two different modification classes:

1. **Vector Modifications:** The baseline will be the addition of the **unmodified** Λ_{16} lattice vectors. The modifications which will be compared to this baseline will be:

   (a) The addition of vectors with one dominant component of the form $\langle \pm 1^1, 0^{15} \rangle$

   (b) The addition of Reed-Muller vector form codewords with *odd* numbers of negatives

   (c) The combination of the above two modifications into the Λ'_{16} codebook

2. **Optimization and Redundancy Elimination:** The baseline will be the combination of SPIHT and Λ'_{16} vectors which will be compared to versions with the following additions:

   (a) Optimal significance

   (b) Redundancy Elimination

For each modification, a single table will be given illustrating the *improvement* in dB PSNR over the baseline method using the *Barbara* image. In addition to this, a short summary will be given which describes the effectiveness of the modification over a data set of 182 runs on 28 images (using different bit-rates on the image set).

Although the addition of the "one dominant component vectors" does not make a tremendous difference in most cases, and in the case of the *Barbara* image actually loses at two bit rates (see Table 3.7), it is generally successful. On 182 test runs on the 28 test images, the modification results in an improvement 115 times, ties the baseline 50 times, and loses 17 times.

The addition of vectors with the Reed-Muller form and odd numbers of negatives gives a more pronounced improvement as can be seen in Table 3.8 above. Using the same 182 test runs, this modification wins 124 times, ties 10 times, and loses 48 times.

Bits per pixel	dB PSNR Improvement
1.0	0.01
0.5	0.01
0.25	0.01
0.125	-0.01
0.0625	0.00
0.03125	0.00
0.015625	-0.01
0.0078125	0.00

Table 3.7: Effects of "one dominant component" vectors

Bits per pixel	dB PSNR Improvement
1.0	0.31
0.5	0.10
0.25	0.10
0.125	-0.04
0.0625	0.00
0.03125	0.03
0.015625	0.02
0.0078125	0.06

Table 3.8: Effects of including the odd negative count in the Reed-Muller forms

It should be clear that the combination of the previous two modifications do not necessarily result in perfectly accumulated results as can be observed in Table 3.9. The use of Λ'_{16} vectors generally beat the original Λ_{16} lattice vectors: 127 wins, 13 ties, and 42 losses as compared to the baseline.

The addition of optimal significance does not change the results much. In fact, the addition mostly results in ties with the baseline, although wins are more common than losses: 64 wins, 99 ties, and 19 losses.

The redundancy elimination modification was, as was mentioned before, guaranteed not to lose. Its addition does not result in tremendous gains and often results in ties, but it is a clear improvement: 71 wins, 111 ties, and 0 losses.

Bits per pixel	dB PSNR Improvement
1.0	0.29
0.5	0.08
0.25	0.06
0.125	-0.07
0.0625	-0.03
0.03125	0.02
0.015625	0.01
0.0078125	0.06

Table 3.9: Effects of the use of Λ'_{16} vectors

Bits per pixel	dB PSNR Improvement
1.0	0.01
0.5	0.00
0.25	0.01
0.125	-0.01
0.0625	0.00
0.03125	0.00
0.015625	0.00
0.0078125	0.00

Table 3.10: Effects of the addition of optimal significance

3.4.4 Visual Results

Because the strictly mathematical nature of the PSNR measurements do not necessarily imply that a reconstructed image with a higher PSNR is more visually appealing than another reconstructed image, this section will look at the impact of the modifications from a visual standpoint. Improvements in some specific images will be shown and further proof that the *ModLVQ* method performs well on high detailed images/regions will be given.

Unlike the quadtree method presented in the previous chapter, it is difficult to design a wavelet compression algorithm which directly benefits the visual quality of the image (which, as we have seen, can sometimes result in a lower PSNR). The best that a wavelet compression method can do is to minimize the error in the quantized

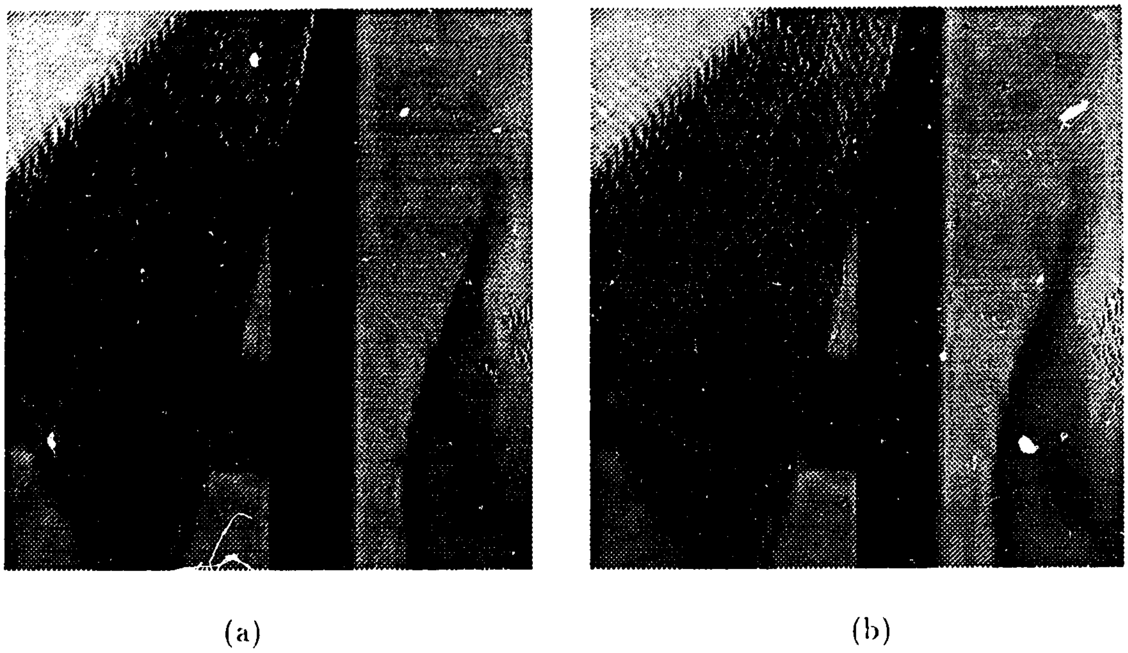
Bits per pixel	dB PSNR Improvement
1.0	0.02
0.5	0.04
0.25	0.03
0.125	0.05
0.0625	0.04
0.03125	0.02
0.015625	0.00
0.0078125	0.00

Table 3.11: Effects of redundancy elimination

wavelet coefficients, thus indirectly reducing the *error* in the reconstructed image. Since the error can only be calculated mathematically and not visually, no guarantees can be given as to the visual quality.

An important question as to the effectiveness of *ModLVQ* is: "Are the results really superior to SPIHT even when the PSNR indicates that this is the case?". In the case of the *Barbara* image, on which the *ModLVQ* algorithm generates impressive results, the visual results are much more visually appealing. Consider Figure 3.26, which shows a tablecloth portion of the Barbara image when compressed to 0.25 bpp using both the SPIHT and *ModLVQ* methods. Clearly, the *ModLVQ* retains much more of the detail. In Figure 3.27, which shows the facial portion of the same 0.25 bpp *Barbara* images, it is more difficult to see the differences. However, upon close inspection, the *ModLVQ* algorithm does a better job of reconstructing the rightmost eye.

Another question which might be asked is: "Are the visual results of the *ModLVQ* algorithm better than those of SPIHT even if the PSNR is lower"? This is difficult to determine. Generally, for smooth images such as *Lena* which do not give the same improved results when the *ModLVQ* algorithm is applied, the visual differences are minimal. However, it should be noted that positive differences can often be picked out of *busy regions* of just about any image. Consider the image segments shown in Figure 3.28, which show the portion of the *Lena* image containing part of a hat, just



(a)

(b)

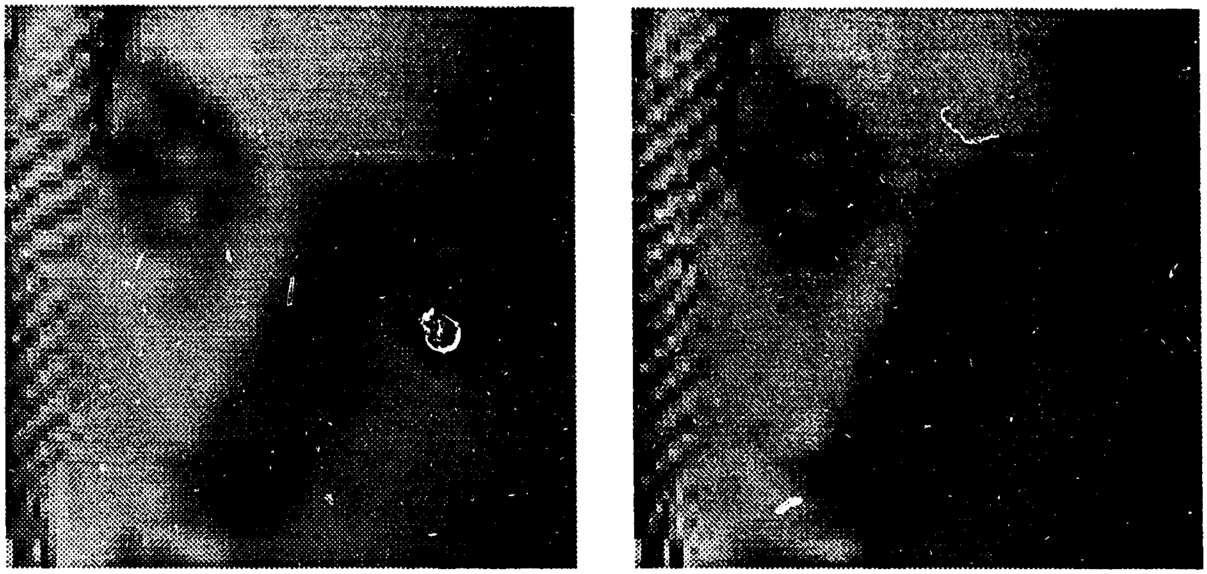
Figure 3.26: Tablecloth portion of Barbara image: (a) compressed using SPIHT; (b) compressed using *ModLVQ*

above the rim. The compression ratio was set in both cases to 0.25 bpp. The SPIHT compression results show a relatively smooth area containing little detail, while the *ModLVQ* results show the fine texture in the hat.

3.4.5 Analysis

An exploration into the reasons for the *ModLVQ*'s enhanced performance for busy images (and busy portions of images) needs to be performed in order to explain the phenomenon. While perhaps not being initially obvious, the major reason for the success is very simple and involves primarily the variation in the distribution of high frequency wavelet coefficients which result from busy as opposed to smooth images or image segments (hereafter, the terms ***smooth image*** and ***busy image*** will be used for both entire images and image segments).

The distribution of high frequency wavelet coefficients will quite clearly be different for a busy image than it would be for a smooth image. The general shape of the



(a) (b)

Figure 3.27: Facial portion of Barbara image: (a) compressed using SPIHT; (b) compressed using *ModLVQ*

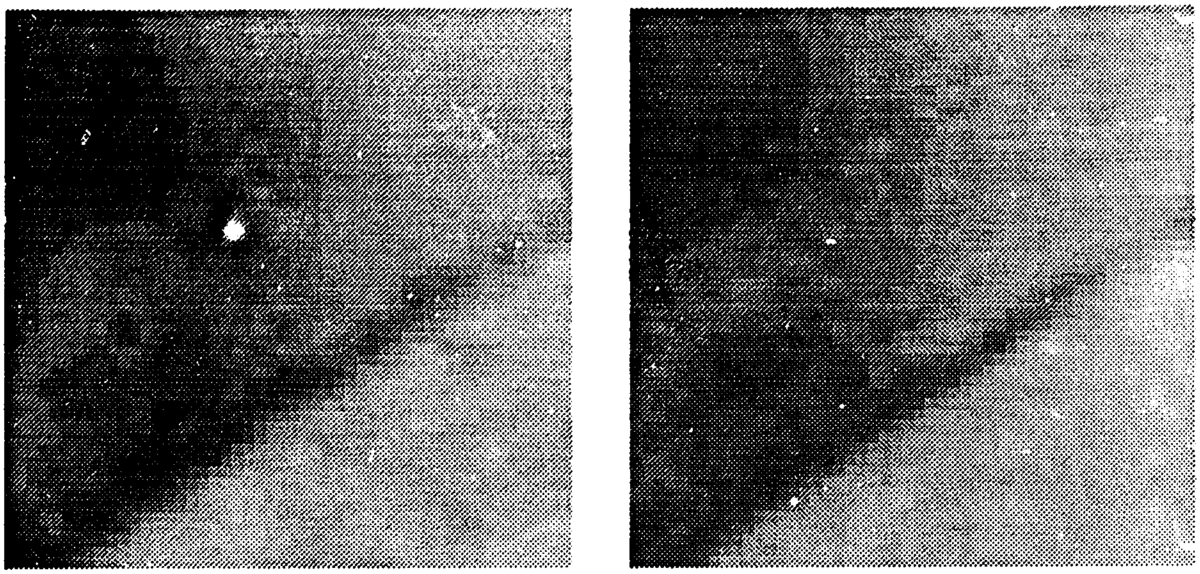
distribution curve will be similar for most images, but a *wider* distribution should be expected for busier images, indicating that there are higher numbers of large magnitude high frequency coefficients. Consider the distributions shown in Figure 3.29. Figure 3.29(a) shows the distribution resulting from a low detail, smooth image, while Figure 3.29(b) shows the distribution resulting from a busy image. The MAN values of the images also indicate the differences in coefficient distributions.

Let us now consider two different types of 4x4 blocks of coefficients that will occur in the coefficient array at any point in the compression:

1. **Smooth blocks:** these will occur with more frequency in smooth images. An example is shown in Figure 3.30(a).

2. **Busy blocks:** these will occur with more frequency in busy images. An example is shown in Figure 3.30(b).

Consider the space which will be required by the coding of the two example blocks by the competing algorithms, SPIHT and *ModLVQ*. A quick summary of the space



(a) (b)

Figure 3.28: Hat portion of the Lena image (a) compressed using SPIHT; (b) compressed using *ModLVQ*

required by each for a generic 4x4 block (without considering refinement passes) is given below:

1. **SPIHT:**
   - For each 2x2 block that has not already been coded as significant, one symbol (from a 2-symbol alphabet) to code the block's (in)significance
   - For each significant 2x2 block, one symbol (from a 2-symbol alphabet) to code the significance of each symbol *which has not already been coded as significant*
2. **ModLVQ:**
   - One symbol (from a 2-symbol alphabet) to code the (in)significance of the 4x4 block
   - One symbol (from a 8192-symbol alphabet) to code the Λ'_{16} codevector

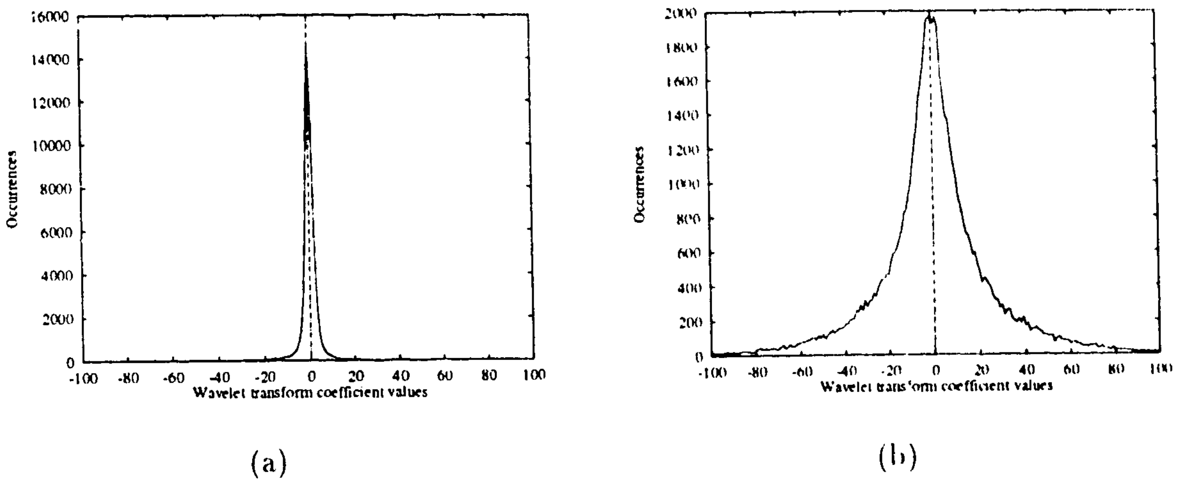


Figure 3.29: Coefficient distributions for two different images: (a) smooth image, MAN=1.75; (b) busy image, MAN=15.52

Clearly, since each significant 2x2 block requires more coding by the SPIHT algorithm, it is able to code a block of the form shown in Figure 3.30(a) more easily than a block of the form shown in Figure 3.30(b). The *ModLVQ* algorithm treats the two blocks in exactly the same manner.

In addition to the above analysis which considers only the sorting pass, the refinement pass will also turn in favour of SPIHT for smooth images. The SPIHT refinement pass codes a single symbol from a 2-symbol alphabet for each coefficient that has been found to be significant. *ModLVQ*, on the other hand, treats refinements in the same way as the initial vector encoding where each block will be coded using a codevector. When fewer significant coefficients are involved, the SPIHT algorithm will perform better than it will when many significant coefficients are involved.

In conclusion, the SPIHT algorithm has more difficulty with coding busy blocks than smooth blocks while *ModLVQ* treats all blocks in the same fashion. As a result, it is not surprising that *ModLVQ* is able to perform well for busy images while SPIHT has more difficulty.

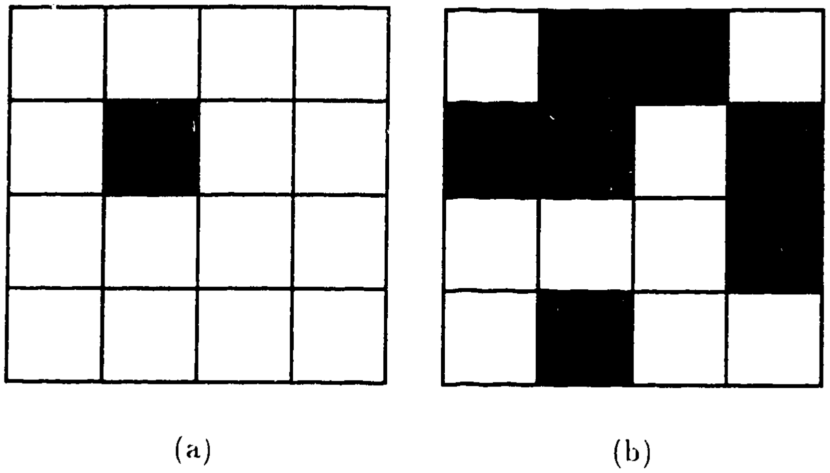


Figure 3.30: 4x4 coefficient blocks, showing large (significant) coefficients shaded black: (a) smooth block; (b) busy block

3.5 Complexity

An exhaustive complexity analysis of the *ModLVQ* (as well as EZW and SPIHT) algorithm is [illegible] simple as it is for the *SR* quadtree algorithm for several reasons:

1. SPIHT requres very few mathematical operations aside from the wavelet transform itself which varies only with the size of the input image.
2. The *ModLVQ* algorithm requires additional mathematical operations that are not required in SPIHT, but it is hard to give a good comparison of the complexities of the two when the mathematical complexity of one of the algorithms is nearly nonexistent.
3. The current implementation has not been optimized to the point where a reasonable comparison of execution time can be performed.

The one thing which seems certain is that the *ModLVQ* algorithm is indeed more complex than its predecessor. The addition of the Λ'_{16} vector quantization requires that additional computations and comparisons be done. The complexity of the algorithm is, like SPIHT, very dependent upon the user selected bit rate for both the compression and decompression modules.

Given the low complexity of lattice vector quantization and the efficiencies of the SPIHT algorithm, the *ModLVQ* algorithm requires a minimal amount of computation power. Although it is not a computational improvement over SPIHT as the improved *SR* quadtree compression was over Shusterman and Feder's quadtree decomposition, it remains an efficient algorithm.

3.6 Future Work

The *ModLVQ* compression method not only can result in superior image quality when compared to existing compression schemes, but it presents several opportunities for future work for further improved compression. Two potential areas for continued research are the vector quantization scheme and the possibly exploitable relationship between image complexity and *ModLVQ* success.

A discussion into the successes of the wavelet/LGB vector quantization combination of Averbuch *et al.*[4] has been given, briefly describing the algorithm to be comparable to SPIHT in image quality, but requiring additional computational complexity. The higher complexity was cited as the main reason for choosing a lattice vector quantization scheme over a similar LGB method. However, the desire for a computationally simple algorithm need not prevent further research into the area. Some possibilities for continued research which include LGB compression could be:

1. A combination of *ModLVQ* and the method described by Averbuch *et al.*, in which an initial vector quantization pass is performed using the suggested LGB method and subsequent refinements to the residuals are performed using *ModLVQ*.

2. The use of lattice vectors in combination with LGB generated vectors in the *ModLVQ* codebook. An entire section in this chapter has been devoted to a modified form of lattice vector quantization designed to improve the results. It may be worthwhile to consider using LGB vector quantization in the design of a modified lattice vector codebook.

A second suggestion for continued work is to find a stronger correlation between the ΔPSNR given by *ModLVQ* over SPIHT and image complexity. Currently, a relatively weak correlation of approximately 0.45 has been established between the MAN and ΔPSNR. If a stronger correlation were to be established using a measure other than the MAN, it might be possible to use a combined *ModLVQ* and SPIHT algorithm which would first calculate some measure of the image's complexity and then decide which of the two algorithms would be most likely to generate the best PSNR results. For example, consider Figure 3.24. A point which divides positive from negative ΔPSNR could be established at an MAN of about 6. Any image which was found to have an MAN greater than 6 could be compressed using *ModLVQ*, otherwise it would be coded using SPIHT. Using the MAN, of course, does not provide the best correlation and many images would be sent to the wrong algorithm.

In addition to the above possibilities for improving PSNR results, further consideration needs to be given to the optimization of the existing implementation in order to make a fair comparison between the complexity differences of *ModLVQ* and SPIHT, and to make the *ModLVQ* algorithm more attractive for general use. Currently, the SPIHT implementation is impressively efficient, requiring less than four seconds of CPU time on an AMD 486DX/4-100 processor to compress a 512x512 image to 1 bit per pixel with similar decompression times. Given the relative theoretical simplicity (requiring only rounding operations) of lattice vector quantization, an efficient implementation which rivals that of SPIHT in execution time is a possibility.

3.7 Conclusion

One of the most powerful wavelet compression algorithms in existence is the SPIHT algorithm. By eliminating many of the redundancies which the EZW algorithm failed to handle effectively and using many of the same efficient structures within EZW, the SPIHT algorithm generates superior results in addition to requiring very little computational effort.

The *ModLVQ* algorithm, based on several modifications to SPIHT including the addition of a modified lattice vector codebook, can generate better PSNR results for busy images. A correlation has been shown to exist between the improvement in PSNR and a measure of image complexity (MAN). In addition, *ModLVQ* generates better quality visual results in areas of images that contain plenty of detail whether or not the entire image is generally busy or smooth.

Aside from the main modification to SPIHT, which was the addition of lattice codevectors, there were several additional modifications which were all shown to be successful in the majority of cases. At least one of these modifications, the reduction in redundancy, need not be applied only to *ModLVQ*. This modification would result in a similar improvement, as has been shown already, if it were applied to SPIHT.

Chapter 4

Conclusion

4.1 Relative Effectiveness of the Algorithms

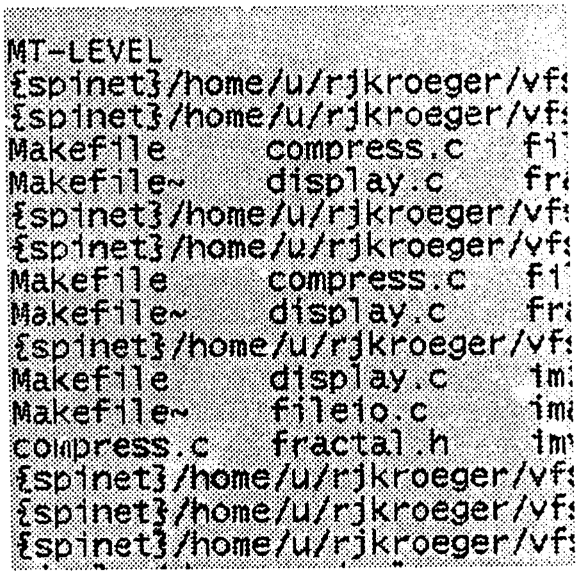
Both the *SR Quadtree Compression* and *ModLVQ* algorithms give impressive results for spatial and transform domain compression respectively. In spite of the improvements made in the quadtree compression, however, its final results are poor in comparison to *ModLVQ*'s. An example comparison is given in Table 4.1, which uses unusual bit rates which result from different thresholds in the quadtree algorithm (since the *SR* compression cannot be given an exact bit rate): In addition to these results, it should be noted that of the entire image set of 28 images described in chapter 2 section 4.5, the *ModLVQ* algorithm performs better on all of them except one (the *window*

Bits per pixel	ModLVQ	SR
0.964	40.30	35.16
0.501	37.03	32.14
0.271	34.26	29.79
0.154	31.88	27.78
0.102	30.04	26.41
0.074	28.74	25.18
0.036	26.29	23.37
0.017	24.23	21.75

Table 4.1: Comparison of *ModLVQ* and *SR* using *Lena*

image - see Figure 2.7), at all bit rates. The wavelet algorithm has several advantages over the quadtree algorithm, which are listed below:

- Embedded code
- Exact compression ratio can be selected by the user
- Superior compression results, even at low bit rates, which are the strongest point of the *SR* quadtree coding



```
MT-LEVEL
{spinet}/home/u/rjkroeger/vf
{spinet}/home/u/rjkroeger/vf
Makefile     compress.c   fi
Makefile~    display.c    fr
{spinet}/home/u/rjkroeger/vf
{spinet}/home/u/rjkroeger/vf
Makefile     compress.c   fi
Makefile~    display.c    fr
{spinet}/home/u/rjkroeger/vf
Makefile     display.c    im
Makefile~    fileio.c     im
compress.c   fractal.h    im
{spinet}/home/u/rjkroeger/vf
{spinet}/home/u/rjkroeger/vf
{spinet}/home/u/rjkroeger/vf
```

Figure 4.1: Text image

While the wavelet based *ModLVQ* compression will give better results than *SR* in general, there are cases where spatial domain methods will still have an edge for some bit rates. One such case is the image shown in Figure 4.1 which is composed of text. A comparison of the results is shown in Table 4.2. The only bit rate in the above table at which the reconstructed image contains reasonably well formed, readable text is the first (0.675 bpp), and then only for *SR* quadtree compression. In images such as the above (and the ***window*** image), quadtree compression algorithms will almost certainly have an advantage over the various transform domain based compression schemes including ***ModLVQ***. Some characteristics which could define images which are

Bits per pixel	ModLVQ	SR
0.675	18.34	48.09
0.464	16.33	15.39
0.120	13.59	12.98

Table 4.2: Successful *SR* compression result

very easily compressed using *SR* quadtree compression (or another form of quadtree or spatial domain compression) are:

- Contains large homogeneous regions
- Edges are sharp and well defined
- Edges are oriented horizontally or vertically
- Few grayscale values are used

Clearly, the above characteristics define a set of images which is quite restricted. However, this does not mean that quadtree compression schemes such as *SR* quadtree compression are not useful. Images with the characteristics described above do exist, for which a spatial domain method such as that presented in Chapter 2 will perform very well.

SR Quadtree Compression

The *SR* quadtree compression method, which is essentially an improved reconstruction method for the decompression phase of the quadtree decomposition method suggested by Shusterman and Feder[28], has been shown to be an effective and efficient quadtree compression algorithm. It has resulted in improvements in complexity as well as the image quality (both mathematically and visually) over previous quadtree compression schemes.

Computationally, the algorithm is very simple. This, combined with its success at lower bit rates, makes it an excellent candidate for use with applications requiring efficient compression, low storage space and good results. In wireless communications, for example, where computing power and transmission speed are important factors, the time required to compress and send image data would need to be reduced significantly. The *SR* algorithm is well suited to such a purpose.

While being unable to compete with the *ModLVQ* algorithm in all but a limited class of images, *SR* compression has been shown to generate higher quality results when compared to its predecessor both mathematically and visually. In addition to this, it is simpler computationally, making it a very attractive spatial domain method.

4.3 ModLVQ Wavelet Compression

Although the modifications to the existing SPIHT algorithm do not generate universal improvements in reconstructed image quality, the *ModLVQ* algorithm can be shown to generate impressive compression results (both mathematically and visually) for busy images and/or busy image segments. Regardless, the *ModLVQ* scheme must be considered as one of the most powerful wavelet compression algorithms that has been developed to date.

Compared to *SR* quadtree compression, *ModLVQ* is the obvious choice when better compression results are desired unless the input image possesses the qualities discussed previously which would make it a good candidate for a spatial domain technique. The efficiency with which *ModLVQ* is able to choose and encode the wavelet coefficients with the greatest impact on the resulting image quality and the additional advantage over SPIHT which allows for superior coding of busy images is difficult for any spatial domain compression scheme to match.

Bibliography

[1] N. Abramson. *Information Theory and Coding.* McGraw-Hill, 1963.

[2] K. Aizawa and T.S. Huang. Model-based image coding: Advanced video techniques for very low bit-rate applications. *Proc. IEEE Special Issue on Advances in Image and Video Comp.*, pages 259–271, 1995.

[3] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Trans. Image Proc.*, 1(2):205-220, 1992.

[4] A. Averbuch, D. Lazar, and M. Israeli. Image compression using wavelet transform and multiresolution decomposition. *IEEE Trans. Image Proc.*, 5(1):4–15, 1996.

[5] J.H. Conway and N.J.A. Sloane. *Sphere Packings, Lattices and Groups.* Springer-Verlag, 1988.

[6] P.C. Cosman, S.M. Perlmutter, and K.O. Perlmutter. Tree-structured vector quantization with significance map for wavelet image coding. *Proceedings Data Compression Conference*, pages 33–41, March 1995.

[7] D. Fuhrmann. Quadtree traversal algorithms for pointer-based and depth first representations. *IEEE Trans. Pattern Anal. Mach. Intel. PAMI-10.*, pages 955–960, 1988.

[8] I Gargantini. An effective way to represent quadtrees. *Commun. ACM.*, 25:905–910, 1982.

[9] R. C. Gonzalez and R.E. Woods. *Digital Image Processing*. Addison-Wesley, 1992.

[10] W.I. Grosky and R. Jain. Optimal quadtrees for image segments. *IEEE Trans. Pattern Anal. Mach. Intel. PAMI-5.*, pages 77–83, 1983.

[11] M. Tamminen H. Samet. Efficient component labelling of images of arbitrary dimension represented by linear bintrees. *IEEE Trans. Pattern Anal. Mach. Intel. PAMI-10*, pages 579–586, 1988.

[12] F.C. Holroyd and D.C. Mason. Efficient linear quadtree construction algorithm. *Image and Vision Computing*, pages 218–224, 1990.

[13] D.A. Huffman. A method for the construction of minimum redundancy codes. *Proc. IRE*, 40(2):1098–1101, 1952.

[14] G.M. Hunter and K. Steiglitz. Operations on images using quad trees. *IEEE Trans. Pattern Anal. Mach. Intel. PAMI-1.*, pages 145–153, 1979.

[15] L. Jones and S. Iyengar. Space and time efficient virtual quadtrees. *IEEE Trans. Pattern Anal. Mach. Intel. PAMI-6*, pages 244–247, 1984.

[16] J. Knipe and X. Li. A new quadtree decomposition reconstruction method. *Accepted for publication at the International Conference on Pattern Recognition*, 1996.

[17] Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer design. *IEEE Trans. Commun.*, COM-28(1):84–95, 1980.

[18] P. Sudarsana M. Manohar and S. S. Iyengar. Template quadtrees for representing region and line data present in binary images. *Comput. Vision, Graphics, and Image Proc.*, 51:338–354, 1990.

[19] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error Correcting Codes*. North-Holland, 1978.

[20] W.B. Pennebaker and J.L. Mitchell. *JPEG Still Image Data Compression Standard.* Van Nostrand Reinhold, 1993.

[21] A. Said and W.A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *to appear in IEEE Trans. Circuits and Systems for Video Tech.*

[22] H. Samet. An algorithm for converting rasters to quadtrees. *IEEE Trans. Pattern Anal. Mach. Intel. PAMI-3.*, pages 93–95, 1981.

[23] H. Samet and R.E. Webber. On encoding boundaries with quadtrees. *IEEE Trans. Pattern Anal. Mach. Intel. PAMI-6*, pages 365–369, 1984.

[24] D.G. Sampson, E.A.B. da Silva, and M. Ghanbari. Wavelet transform image coding using lattice vector quantization. *Electronics Letters*, 30(18):1477–1478, 1994.

[25] D.G. Sampson and M. Ghanbari. Fast lattice-based gain-shape vector quantisation for image-sequence coding. *IEE Proceedings-I*, 140(1):56–66, 1993.

[26] C.A. Shaffer and H. Samet. Optimal quadtree construction algorithms. *Comput. Vision, Graphics and Image Proc.*, 37:402–419, 1987.

[27] J.M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. Signal Proc.*, 41(12):3445–3462, 1993.

[28] E. Shusterman and M. Feder. Image compression via improved quadtree decomposition. *IEEE Trans. Image Proc.*, 3(2):207–215, 1994.

[29] Peter Strobach. Tree-structured scene adaptive coder. *IEEE Trans. Commun.*, 38(4):477–486, 1990.

[30] Peter Strobach. Quadtree-structured recursive plane decomposition coding of images. *IEEE Trans. Sig. Proc.*, 39(6):1380–1397, 1991.

[31] G.J. Sullivan and R.L. Baker. Efficient quadtree coding of images and video. *IEEE Trans. Image Proc.*, 3(3):327–331, 1994.

[32] R. Wilson. Quad-tree predictive coding: A new class of image data compression algorithms. *Proc. Int. Conf. Acoustics, Speech and Signal Proc., 29.3.1*, 1984.

[33] I.H. Witten, R.M. Neal, and J.G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30:520–540, 1987.

[34] T. Agui Y. Arai and M. Nakajima. A fast dct-sq scheme for images. *Trans. of the IEICE*, 71(11):1095, 1988.

[35] M. Landy Y. Cohen and M. Pavel. Hierarchical coding of binary images. *IEEE Trans. Pattern Anal. Mach. Intel. PAMI-7.*, pages 284–298, 1985.