AN EFFICIENT CODING METHOD FOR SPATIAL DATA: THE ROTATING, HIERARCHICAL, OVERLAPPING REPRESENTATION

Yuan Zhang

Department of Electrical & Computer Engineering
McGill University, Montréal

September 2002

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of
Master of Engineering

National Library of Canada

Acquisitions and Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada

Acquisisitons et services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 0-612-85903-7
Our file Notre référence
ISBN: 0-612-85903-7

ABSTRACT

A new, efficient image compression algorithm based on a rotating, overlapping, hierarchical representation (ROHR) is presented. This algorithm first decomposes the spatial image data into a hierarchical coordinate space, composed of interconnected nodes. The *surprise* of each node represents the entropy of the corresponding image area, and determines its significance in reconstructing the image. *Surprises* are ordered and transmitted in decreasing order of magnitude with a bucket encoding method, yielding an embedded code. ROHR allows for either lossless compression, or for termination of the encoding or decoding at any point to meet a target compression ratio or distortion metric. In the latter, image reconstruction from the transmitted data occurs without the introduction of artifacts that would indicate where the truncation occurred. Moreover, the procedures allow for real-time video encoding or decoding operations on color images of 256×192 pixels on current PC architectures, considerably faster than equivalent software JPEG codecs.

RÉSUMÉ

Cette thése présente un nouvel algorithme efficace de compression basé sur une représentation hiérarchique composée de noeuds pivotés et superposés (Rotating, Overlapping, Hierarchical Representation - ROHR.) Cet algorithme décompose d'abord les données spatiales d'une image en un espace de coordonnées hiérarchiques composé de noeuds interconnectés. La surprise de chaque noeud représente l'entropie de la région correspondante de l'image et détermine son importance dans la reconstruction de l'image. Les surprises sont ordonnées et transmises par ordre de grandeur décroissant avec une méthode d'encodage par casier, produisant un code intégré. ROHR permet une compression sans perte ou l'interruption de l'encodage ou du décodage à n'importe quel point pour respecter une mesure de distorsion ou un taux de compression désiré. Dans ce dernier cas, l'image est reconstruite à partir des données transmises sans l'introduction d'artefacts qui indiqueraient où la troncation s'est produite. De plus, les procédures permettent l'encodage ou le décodage vidéo en temps réel d'images de 256×192 pixels sur des architectures PC actuelles considérablement plus rapidement que les codecs logiciels JPEG équivalents.

ACKNOWLEDGEMENTS

I am indebted to my thesis supervisor, Professor Jeremy Cooperstock, for his continued supervision and guidance throughout the course of this research. I would like to sincerely thank Stephen Spackman for his suggestions and invaluable support for this thesis. I admire his deep insight into many scientific subjects and his enthusiasm for research. He has played an important role in my research work. I truly appreciate his time and effort.

I would like to thank Xiaohui Song of the Department of Mechanical Engineering for his friendly help in editing this thesis, and Vincent Levesque of the Department of Electrical & Computer Engineering of his kind help in translating the abstract.

Last but not least, I would like to express my deepest gratitude to the persons who directly and indirectly contributed to my thesis, but are not mentioned here.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Image compression is an intensively studied research area in signal and image processing. It plays an important role in applications such as television transmission and image databases. Images have been used increasingly as a supplement to traditional communication media such as text, or in many cases, taken their places. However, uncompressed images usually require considerable storage capacity and transmission bandwidth. Despite rapid progress in storage density, processor speeds, and digital communication system performance, data storage capacity and data transmission bandwidth remain bottlenecks for many applications.

Image compression is the process of converting an input image into another data stream of smaller size by removing the redundancy from the original image. The generated data stream is either a file or bit stream over communication lines. Compression should preserve the *information* in the original image, where information theory presents a mathematical definition of *information* and *redundancy*, and provides basic theories for image compression as well.

1. Information Theory

Information is the knowledge of a specific event or situation; it is a collection of facts or data. Quantifying information is based on the observation that the information content of a message is equivalent to the amount of *surprise* in a message. Suppose an event E will occur with probability 0.01, and will not occur with probability 0.99. We expect that E will not occur, but if it does, we will be greatly surprised and receive a lot of information. The following definition relates the information of an event to its probability [**1**]:

> Let E be some event that occurs with probability *P(E)*. If we are told that event E has occurred, then we say we have received
>
> $$I(E) = log_2 \frac{1}{P(E)}$$
>
> units of information.

The choice of the logarithm to the base 2 indicates that this amount of information could be expressed as *I(E)* binary bits, that is, *I(E)* yes or no questions. We note, also, that if $P(E) = \frac{1}{2}$, then $I(E) = 1$ bit. That is, we obtain one bit of information when one of two equally likely alternatives is specified.

Now consider a sequence of symbols from a fixed finite alphabet $S = \{s_1, s_2, ..., s_n\}$. We assume that symbol s_i occurs with probability P_i. The sum of probabilities equals unity: $P_1 + P_2 + ... + P_n = 1$. If symbol s_i occurs, we obtain information:

$$I(s_i) = log \frac{1}{P_i}$$

The average amount of information per symbol is:

$$H(S) = \sum_{i=1}^{n} P_i \, log_2 \frac{1}{P_i}$$

$H(S)$ is called the *entropy* of a symbol. In the special case where all the symbols are equiprobable, $P_1 = P_2 = ... = P_n = P = \frac{1}{n}$, the entropy achieves the maximum value:

$$H(S)_{max} = \sum_{i=1}^{n} P\, log_2 P = log_2 n$$

This fact is used to define the *redundancy* R in the data, as the difference between a symbol set's largest possible entropy and its actual entropy:

$$R = -log_2(1/P) - (-\sum_{i=1}^{n} P_i\, log_2 P_i) = -log_2 n + \sum_{i=1}^{n} P_i\, log_2 P_i$$

Generally speaking, data can be compressed if it is redundant.

2. Why Image Compression Works

A digital image is an array of pixels. A common characteristic of natural images is that neighboring pixels are correlated and therefore contain redundant information. "If we select a pixel in the image at random, there is a good chance that its neighbors will have the same color or very similar colors."[2] The central task of image compression is therefore to remove the redundancy from the original data and create a less correlated representation of the image. In general, two types of redundancy can be identified:

1. *Spatial Redundancy* or correlation between neighboring pixel values.
2. *Spectral Redundancy* or correlation between different color planes or spectral bands.

Image compression research aims to reduce the number of bits needed to represent an image by removing the spatial and spectral redundancies as much as possible. This suggests the general law of data compression: assign short codes to common events

(or symbols) and long codes to rare events.

3. Approaches to Image Compression

According to whether or not distortion is introduced into the reconstructed images, image compression methods are classified as lossy or lossless. Lossless compression methods ensure that all the information in the original image is preserved after compression, thus maintaining the full quality of the original. Lossy methods, however, discard some information during compression, typically high frequency features to which the human eye is insensitive. Lossy compression often results in much smaller file sizes than lossless compression.

A further classification of compression methods divides them into statistical, predictive and transform techniques.

Statistical methods are straightforward when we consider the general law of data compression: assign short codes to common symbols and long codes to rare ones. The compression quality depends on the accuracy of the statistical model. Variable-size codes are assigned to certain symbols according to their frequencies in the input stream. *Huffman Coding* is an example of such methods. Under certain statistical models, it can generate an optimal coding. Other commonly used statistical compression schemes are *Shannon-Fano Coding* and *Arithmetic Coding.*

Predictive methods use the information already coded to predict future values and encode the difference. When applied to the image or spatial domain, it is relatively simple to implement and is readily adapted to local image characteristics. *Differential Pulse Code Modulation* (DPCM) is an example of predictive coding.

Transform methods are powerful tools in image compression. They first transform the image from its spatial domain representation to a different representation using some well-known mapping and then code the transformed coefficients. Compression is achieved in two ways. First, the transform coefficients are usually smaller, on average, than the original spatial values because the high frequency components tend to be of small magnitudes. Lossy compression, therefore, can be achieved by quantizing these coefficients. Second, the transform coefficients are generally less correlated with each other than the original image pixels, and can thus be encoded more compactly. Transform methods often provide better compression than predictive methods, although at the expense of greater computational requirements. Commonly used transforms are the *Fourier Transform* (FT), *Discrete Cosine Transform* (DCT) and the *Wavelet Transform* (WT).

Other image compression approaches include the *Run Length Encoding* (RLE), the *Gray Codes*, and the *Vector Quantization* (VQ). Practical image compression algorithms often comprise a combination of basic methods in order to achieve better compression than a single method alone. An example is JPEG (sequential mode). The original image is first organized into groups of 8×8 pixels called data units. The DCT transform is then applied to each data unit to create an 8×8 block of frequency coefficients. Each coefficient is quantized and statistically coded.

4. Thesis Roadmap

This thesis presents a new image compression algorithm, whose overall structure is shown in Figure 1.1. The original image is first decomposed into a hierarchical *coordinate space*, which is composed of interconnected nodes. The *surprise* of each node represents the entropy of the corresponding image portion, and determines its significance in reconstructing the image. Since most images are highly spatially redundant,

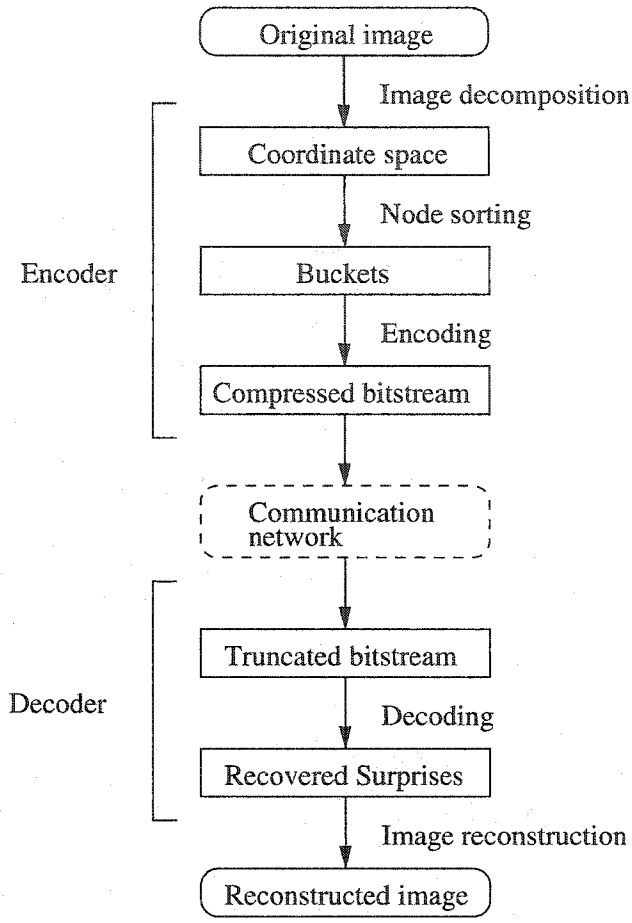


FIGURE 1.1. Structure of our image compression algorithm

many nodes have trivial surprises. Compression, therefore, is achieved by neglecting nodes with small surprises. One important characteristic of this method is that when the encoding or the decoding is prematurely terminated, the image can still be reconstructed from the remaining data, without introduction of artifacts that would indicate where the truncation occurred.

The image decomposition step is discussed in Chapter 2. We begin with a simple example and present the one-dimensional decomposition. Its relationship to wavelet transform is discussed. We then continue with the two-dimensional case and present two decomposition schemes – pyramid decomposition and quincunx decomposition.

In Chapter 3 and Chapter 4, we discuss the encoding methods in detail. We introduce the philosophy of *Embedded Coding*, and describe some successful examples. The *bucket* is our implementation of embedded coding. We discuss the prioritization principles, bucket labeling and coalescing, and efficient encoding of the bucket contents. In Chapter 5, we expand the approach to color image compression. Experimental results are presented and discussed in Chapter 6. Finally, in Chapter 7, we summarize the improvements obtained by the proposed algorithm and discuss future work.

CHAPTER 2

Image Decomposition

A natural image is highly spatially correlated, with most of the area typically representing spatial *trend* [4], i.e. uniform or slightly varied regions. However, the *anomalies*, such as edges or object boundaries, which take on perceptual significance, contribute little to the numerical energy of the image as a whole. Image compression is achieved by detecting these *anomalies* and allocating more bits for coding them than the *trends.*

Some traditional frequency transforms, such as the Fourier Transform, are used for this purpose. They transform the whole image into the frequency domain, in which the *trends* are expressed as low frequency components and *anomalies* as high frequency components. This type of transformation is computationally expensive. Moreover, since it expresses the image as a sum of periodic waves that are well localized in the frequency domain but not in the time domain, they cannot represent the local properties of an image efficiently.

Some other traditional image compression algorithms, such as JPEG, first decompose the image into a number of macroblocks, and then compute the DCT over these, such that each coefficients corresponds to a fixed sized area and a fixed frequency bandwidth, where the bandwidth and spatial extent are effectively the same for all

coefficients in the representation. This method is also known as windowed frequency analysis. However, at low bit rate, blocking artifacts at the macroblock level are often evident due to quantization effects.

In order to achieve an efficient compression at both high and low bit rates, we need a framework that gives the *anomalies* and *trends* the same weighting in the analysis. In addition, it should analyze the images at different *scales* and provide a multiresolution representation in which some of the coefficients represent long data lags corresponding to a narrow band, low frequency range, and some of the coefficients represent short data lags corresponding to a wide band, high frequency range. A representative of this multiresolution transform is the Wavelet Transform, which achieves excellent results at low bit rate compression. In addition, some spatial image decomposition schemes can also achieve good multiresolution transformation.

In this chapter, we discuss a new spatial decomposition method - the rotating, overlapping, hierarchical representation (ROHR). It is effective for image compression and easy to implement. The process of decomposition is presented in detail, and its properties are discussed. We also compare it to the wavelet transform and investigate their relationship.

1. One-dimensional Decomposition

The decomposition of a one-dimensional array into a hierarchical network is shown in Figure 2.1. The input image signal contains 16 individual pixels, numbered from 16 to 31. They constitute the bottom layer of the hierarchical structure. Layer 4 is constructed from layer 5 by combining every three nodes together. For example, node 8 is constructed by combining nodes 16, 17 and 18, and is considered as their parent, since it represents the image area covered by its three children. Node 9 is constructed from nodes 18, 19 and 20. Nodes 8 and 9 share a common child, node

18, as the two image areas they represent overlap. Similarly, other nodes in layer 4 are constructed from three consecutive children nodes in layer 5, and share children with their immediate neighbors in the same layer. In some cases, nodes falling at the right border, for example node 15 in this figure, may have only two children.

This process is repeated until the top layer is created, at which point we obtain a hierarchical structure called the *coordinate space*. Each node in this coordinate space is related to a piece of image area, with its scale, or resolution, depending on the layer in which it resides. Nodes in the bottom layer, which correspond to real image pixels, are the highest resolution block. Their parents have larger scales and lower resolutions. The first layer consists of a single uniform area, with the size of the entire image. This coordinate space, therefore, is regarded as a multiresolution representation of the original image.

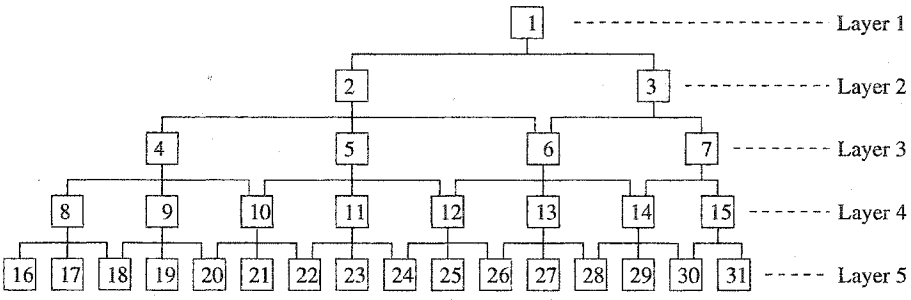


FIGURE 2.1. One-dimensional decomposition into a coordinate space

We designate the nodes without children as leaf nodes, and those having two or three children as non-leaf nodes. The top node, called *root*, is the entry point to the coordinate space. There are two kinds of *typical* nodes in the space. One is like node 10, which has two parents (nodes 4 and 5) and three children (nodes 20, 21 and 22). We call it an N-node. The other is like node 11, which has one parent (node 5) and three children (nodes 22, 23 and 24). Since it is the middle child of its parent, we call it an M-node. If we traverse the coordinate space top down and number the nodes in breadth-first sequence, as shown in Figure 2.1, the structure of N-nodes and

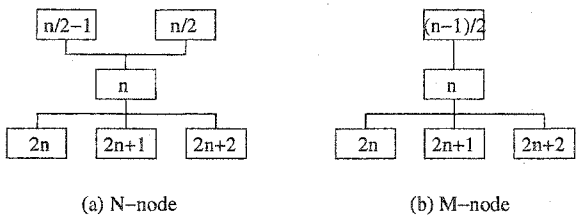


FIGURE 2.2. Structure of N-nodes and M-nodes

M-nodes, including the index number relationships, can be summarized in Figure 2.2. Note that these index number relationships are also suitable for the root and leaf nodes.

Each node in the coordinate space is associated with a (*value*, *surprise*) pair. The value, denoted as *val*, indicates the *trend* within the corresponding image area. It should be the representative of all its children, therefore similar to an average. We define the value as the weighted sum of its children's values:

$$val(n) = \frac{1}{2}val(left\ \ child) + val(middle\ \ child) + \frac{1}{2}val(right\ \ child) \qquad (2.1)$$

For those nodes having only two children, the value is defined as:

$$val(n) = val(left\ \ child) + val(right\ \ child)$$

The value of a leaf node is the intensity of the corresponding image pixel.

The surprise of a node, on the other hand, reflects *anomalies*. Consider an N-node; if it resides in a uniform, or nearly uniform region, its two parents probably have the same, or nearly the same values. Hence, its value can be well predicted from its parents' values. On the contrary, if it happens to cross an edge or an object boundary, its two parents' values reflect the different *trends* on the two sides of the edge. As a result, there is a significant difference between the node's value and the predicted value from its two parents. Therefore, we measure image details by evaluating how accurately a node's value can be predicted from its parents'. This is called the *surprise*

of a node, which is defined as:

$$surprise(n) = val(n) - (val(left\ \ parent) + val(right\ \ parent))/4 \qquad (2.2)$$

For those nodes having only two children, the surprise is defined as:

$$surprise(n) = val(n) - val(parent)/2$$

The surprise of an M-node does not contain any information about *anomalies* because the image area it represents is completely covered by its parent's. Ignoring surprises of M-nodes does not influence the extraction of *anomalies*, so we only compute surprises of N-nodes as the multiresolutional details of the original image. Moreover, values of M-nodes are *redundant* because they can be inferred completely from N-nodes' values.

Because of general image spatial correlation in images, the surprises, which represent possible *anomalies*, are decorrelated and often small. Consequently, an arbitrary level of compression can be achieved by neglecting surprises below a given threshold during the encoding.

The decomposition method described above can be implemented by the following algorithm:

> Copy the image data to the bottom layer of the coordinate space
>
> Build the coordinate space from bottom up, computing the value of every node according to Equation 2.1.
>
> Traverse the coordinate space from top down, computing the surprise of every N-node (except the root) according to Equation 2.2

The decomposition is reversible, that is, we can reconstruct the entire coordinate space from the root's value and surprises of N-nodes. Given the surprise of an N-node n

and its parents' values, we can recover the value of n as:

$$val(n) = surprise(n) + (val(left\ parent) + val(right\ parent))/4 \qquad (2.3)$$

Whereas the value of an M-node m is recovered by:

$$val(m) = val(parent) - \frac{1}{2}val(left\ neighbour) - \frac{1}{2}val(right\ neighbour) \qquad (2.4)$$

The reconstruction process can be implemented with the following algorithm:

Copy the value of root.
layer $L \leftarrow 0$
while L is not the bottom layer do:
    $L \leftarrow L + 1$
    Compute values of N-nodes in L according to Equation 2.3
    Compute values of M-nodes in L according to Equation 2.4

Values in the reconstructed bottom layer correspond to the recovered image.

2. Properties of the Image Decomposition

This decomposition is independent of image content. The depth and breath of coordinate space is determined only by the size of the image. If there are k nodes in one layer, then there are

$$\begin{cases} \frac{k}{2} & if\ k\ is\ even \\ \frac{k-1}{2} & if\ k\ is\ odd \end{cases} \simeq \frac{k}{2}$$

nodes in its parent layer. Supposing the original image contains p pixels, the depth of the coordinate space is $\lfloor log_2 p + 1 \rfloor$, where $\lfloor x \rfloor$ indicates the largest integer not

exceeding x, and the total number of nodes in the coordinate space is:

$$p + \frac{1}{2^1}p + \frac{1}{2^2}p + \frac{1}{2^3}p + ... + 1 \simeq 2 \times p$$

If we ignore all the M-nodes as they contain no additional information, the number of nodes encoded is p, equal to the original image size.

From Equation 2.1 we also notice that the parent's value is *statistically* twice those of its children. Since the image area covered by the parent is twice those covered by its children, the value of a node, in some sense, is ranked by the size of the image portion it represents. As a consequence, the surprises in the entire coordinate space are statistically grouped with respect to their layers. In other words, if we consider the surprise in a certain layer as a random variable, its expected value is higher than those in any of its descendant layers. This property is important for the encoder design.

Another interesting and appealing property of this decomposition is that it produces integer values and surprises. Moreover, since the decomposition and reconstruction are symmetric, every value can be completely recovered during reconstruction. Hence we can avoid any deviation caused by floating point arithmetic operations and obtain a simple and lossless recovery calculation.

3. Wavelet Transform Interpretation

3.1. Wavelet Transform. The continuous wavelet transform (CWT) of a function $f(t)$ involves a mother wavelet $\psi(t)$. The mother wavelet must be integrable and square integrable. The CWT of a square integrable function $f(t)$ is defined as:

$$W(a,b) = \int_{-\infty}^{\infty} f(t)\, \psi_{a,b}(t)\, dt$$

where

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\psi(\frac{t-b}{a})$$

For any a, $\psi_{a,b}(t)$ is a copy of $\psi_{a,0}$ shifted b units along the time axis. Thus, b is a translation parameter and a is a scaling, or dilation parameter. Values $a > 1$ stretch the wavelet, while $0 < a < 1$ shrink it.

The CWT is best thought of as an array of numbers that are inner products of $f(t)$ and $\psi_{a,b}(t)$, which is a series of mother wavelets with different translation and scaling parameters. The result $W(a,b)$ shows the match between $f(t)$ and the wavelet at different frequencies and at different times. Wavelet transform, therefore, provides us the analysis of $f(t)$ at different scales. When we change the scale of the wavelet, we obtain new information about the function being analyzed. The quality of the transform depends on the choice of scale factors and time shifts, as well as the choice of wavelet.

The inverse CWT is defined by:

$$f(t) = \frac{1}{C}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{1}{|a|^2}W(a,b)\psi_{a,b}(t)\,da\,db$$

where the quantity C is defined as:

$$C = \int_{-\infty}^{\infty}\frac{|\Psi(\omega)|^2}{|\omega|}\,d\omega$$

and $\Psi(\omega)$ is the Fourier transform of $\psi(t)$:

$$\Psi(\omega) = \int_{-\infty}^{\infty}\psi(t)e^{-i\omega t}\,dt$$

The discrete wavelet transform is used in practical applications. Consider a 1-D discrete image signal $x[n]$, where n is the pixel index. The wavelet transform expresses

$x[n]$ as a sum of basis functions $f_i[n]$ weighted by the coefficients p_i:

$$x[n] = \sum_i p_i f_i[n]$$

The coefficients p_i can be derived by taking the inner product of the image with a set of wavelets $g_i[n]$:

$$p_i = \sum_n g_i[n]x[n]$$

This may be expressed in matrix notation as follows [8]: Let the image signal be a column vector $\mathbf{x}$. Define the kernel matrix to be composed of columns f_i:

$$\mathbf{F} = \begin{vmatrix} \vdots & \vdots & \vdots & \\ f_0 & f_1 & f_2 & \cdots \\ \vdots & \vdots & \vdots & \end{vmatrix}$$

Then we have:

$$\mathbf{x} = \mathbf{F}\mathbf{p}$$

where $\mathbf{p}$ is the column vector of coefficients. These coefficients are found by multiplying the image with the transform matrix $\mathbf{G}$, where $\mathbf{G}$ is composed of rows g_i. Thus

$$\mathbf{p} = \mathbf{G}\mathbf{x}$$

and

$$\mathbf{x} = \mathbf{F}\mathbf{G}\mathbf{x}$$

so we have:

$$\mathbf{G} = \mathbf{F}^{-1}$$

We are particularly interested in the situation where the basis functions can be partitioned into a few classes, where the functions within a given class are shifted versions of each other. That is, the kernel matrix has the form:

$$\mathbf{F} = \begin{vmatrix} f_a & & f_b & & \\ \vdots & f_a & \vdots & f_b & \cdots \\ & \vdots & f_a & \vdots & f_b \\ & & \vdots & & \vdots \end{vmatrix}$$

Then the transform matrix $\mathbf{G}$ has the form:

$$\mathbf{G} = \begin{vmatrix} g_a & \cdots & & \\ g_b & \cdots & & \\ & g_a & \cdots & \\ & g_b & \cdots & \\ & & g_a & \cdots \\ & & g_b & \cdots \\ & & & \vdots \end{vmatrix}$$

As an example, consider the *Daubechies D4*. It is based on four coefficients c_1, c_2, c_3 and c_4. The transform matrix $\mathbf{G}$ is:

$$\mathbf{G} = \begin{vmatrix} c_1 & c_2 & c_3 & c_4 & 0 & 0 & \cdots & 0 \\ c_4 & -c_3 & c_2 & -c_1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & c_1 & c_2 & c_3 & c_4 & \cdots & 0 \\ 0 & 0 & c_4 & -c_3 & c_2 & -c_1 & \cdots & 0 \\ \vdots & \vdots & & & & \ddots & & \\ 0 & 0 & \cdots & 0 & c_1 & c_2 & c_3 & c_4 \\ 0 & 0 & \cdots & 0 & c_4 & -c_3 & c_2 & -c_1 \\ c_3 & c_4 & 0 & \cdots & 0 & 0 & c_1 & c_2 \\ c_2 & -c_1 & 0 & \cdots & 0 & 0 & c_4 & -c_3 \end{vmatrix}$$

When this matrix is applied to the column vector $\mathbf{x}$, its top row generates the weighted sum $s_1 = c_1x_1 + c_2x_2 + c_3x_3 + c_4x_4$, its third row generates the weighted sum $s_2 = c_1x_3 + c_2x_4 + c_3x_5 + c_4x_6$, and the other odd-numbered rows generate similar weighted sums s_i. Each of s_i is called a *smooth coefficient*, and together they are called an L smoothing filter.

In a similar manner, the second row of the matrix generates $d_1 = c_4x_1 - c_3x_2 + c_2x_3 - c_1x_4$, and the other even-numbered rows generate similar inner products. Each d_i is called a *detail coefficient* and together they are called an H filter. The H filter generates small values when the image data x_i are correlated. L and H are called *Quadrature Mirror Filters* (QMF).

The discrete wavelet transform of an image can therefore be viewed as passing the original image through a QMF that consists of a pair of low-pass (L) and high-pass (H) filters. If $\mathbf{G}$ is an $n \times n$ matrix, it generates $\frac{n}{2}$ smooth coefficients s_i and $\frac{n}{2}$ detail coefficients d_i.

An alternate description of this wavelet transform is shown in Figure 2.3. The image signal $x(n)$ is fed into the filters one by one, and each filter computes and outputs one number $y(n)$ in response to $x(n)$. The number of response is therefore double the number of inputs. To correct this situation, each filter is followed by a downsampling process. One downsampling throws away the odd-numbered outputs, the other throws away the even ones. When they are synthesized, they are first upsampled, and passed through the inverse filters, and finally combined to form the signal $x'(n)$. This process is also called a *subband transform.*

Note that the transform matrix $\mathbf{G}$ in the *D4* example shows how to generate the finest resolution coefficients. The scaling parameter of the wavelet transform can be illustrated in Figure 2.4. The outputs of the low-pass filter L are normally passed

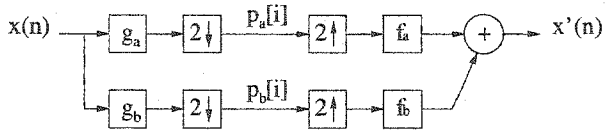


FIGURE 2.3. A two-channel filter bank

through the analysis filter several times, creating shorter and shorter outputs. Since each node of this tree produces half the number of outputs as its predecessor, the tree is called a *logarithmic tree* [2]. Each level of the tree corresponds to twice the frequency of the preceding level, so this tree is also called a *multiresolution tree.* Successive filtering through the tree separates lower and lower frequencies.

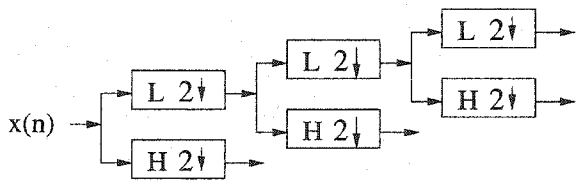


FIGURE 2.4. Illustration of the scaling parameter

3.2. Relevance to Image Decomposition. Consider the one-dimensional decomposition example in Figure 2.1. The input image data vector **x** consists of the layer 5. When we build up its parent layer, we compute the surprises in layer 5 by Equation 2.2 and values in layer 4 by Equation 2.1. These two equations can be described by a more general model, as shown in Figure 2.5.

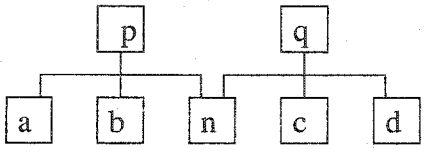


FIGURE 2.5. A general model for calculating the values and surprises

For the value calculation, we have:

$$\begin{aligned} p &= \frac{1}{2}a + b + \frac{1}{2}n \\ q &= \frac{1}{2}n + c + \frac{1}{2}d \end{aligned}$$

and for surprise, we have:

$$\begin{aligned} surprise(n) &= n - \frac{1}{4}(p+q) \\ &= n - \frac{1}{4}(\frac{1}{2}a + b + \frac{1}{2}n + \frac{1}{2}n + c + \frac{1}{2}d) \\ &= -\frac{1}{8}a - \frac{1}{4}b + \frac{3}{4}n - \frac{1}{4}c - \frac{1}{8}d \end{aligned}$$

Effectively, calculating the values is equivalent to passing the data through a 5-tap low-pass filter L:

$$\{0, \frac{1}{2}, 1, \frac{1}{2}, 0\}$$

whereas calculating surprises is equivalent to passing the data through a 5-tap high-pass filter H:

$$\{-\frac{1}{8}, -\frac{1}{4}, \frac{3}{4}, -\frac{1}{4}, -\frac{1}{8}\}$$

By ignoring the middle children, half the number of values and half the number of surprises are generated. This process, therefore, is a 5-tap kernel wavelet transform, which splits the frequency band by a low-pass and a high-pass filter. The transform matrix for 16-point image data (including the border cases) is:

$$\mathbf{W}_{16\times16} = \begin{vmatrix} \frac{3}{4} & -\frac{1}{2} & -\frac{1}{4} & 0 & 0 & 0 & 0 & \cdots & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 & \cdots & 0 \\ -\frac{1}{8} & -\frac{1}{4} & \frac{3}{4} & -\frac{1}{4} & -\frac{1}{8} & 0 & 0 & \cdots & 0 \\ 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & \cdots & 0 \\ 0 & 0 & -\frac{1}{8} & -\frac{1}{4} & \frac{3}{4} & -\frac{1}{4} & -\frac{1}{8} & \cdots & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & \cdots & 0 \\ \vdots & \vdots & & & & \ddots & & & \\ 0 & 0 & \cdots & -\frac{1}{8} & -\frac{1}{4} & \frac{3}{4} & -\frac{1}{4} & -\frac{1}{8} & 0 \\ 0 & 0 & \cdots & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 \\ 0 & 0 & \cdots & 0 & 0 & -\frac{1}{8} & -\frac{1}{4} & \frac{3}{4} & -\frac{1}{8} \\ 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{vmatrix}$$

The outputs of the low-pass filters are transformed several times, so the wavelet kernels are cascaded hierarchically to create a multiscale pyramid, i.e. the coordinate space. Also, by numbering this pyramid from top down, surprises are always put on the right half of the output and the values on the left half. Thus the process of building the coordinate space can be described by Figure 2.6.

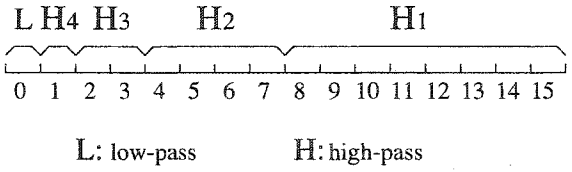


FIGURE 2.6. Band-splitting description of 1-D decomposition. The labels, H_1 throught H_4 correspond to high-pass subbands of decreasing spatial frequency, respectively.

Further investigation of this wavelet transform indicates that (1) the transform kernels are not orthonormal and (2) the high-pass kernel has a zero DC gain. As noted by Adelson et al [8], "In transform based on QMF kernels, the basis set is orthonormal and so the transform functions are identical to the basis functions. But orthogonality is not strictly required for image compression. It is only necessary that the transform

be invertible." In our case, the matrix of $\mathbf{W}_{16\times16}$ is non-singular, and the inverse matrix is:

$$\mathbf{W}_{16\times16}^{-1} = \begin{vmatrix} 1 & -\frac{1}{2} & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \frac{1}{2} & \frac{5}{8} & \frac{1}{4} & -\frac{1}{8} & 0 & 0 & 0 & \cdots & 0 \\ 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 & 0 & \cdots & 0 \\ 0 & -\frac{1}{8} & \frac{1}{4} & \frac{3}{4} & \frac{1}{4} & -\frac{1}{8} & 0 & \cdots & 0 \\ \vdots & \vdots & & & & \ddots & & & \\ 0 & 0 & \cdots & 0 & -\frac{1}{2} & 1 & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \cdots & 0 & -\frac{1}{8} & \frac{1}{4} & \frac{3}{4} & \frac{1}{4} & -\frac{1}{8} \\ 0 & 0 & \cdots & 0 & 0 & 0 & -\frac{1}{2} & 1 & -1 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \frac{1}{8} & \frac{1}{4} & \frac{7}{4} \end{vmatrix}$$

Both $\mathbf{W}$ and $\mathbf{W}^{-1}$ are extremely simple and easy to compute. Many multiplications can be simply implemented by shifts and adds. It is thus more computationally efficient than most other wavelet transforms.

4. Two-dimensional Decomposition

In order to decompose two-dimensional images into several subbands, the most straightforward approach is to apply one-dimensional transforms to rows and columns separately, in a *pyramid decomposition*, illustrated in Figure 2.7. The first step calculates the values and surprises for all the rows. This creates values in the left half of the image and surprises in the right half. The second step calculates values and surprises for all the columns, which creates values in the top-left quadrant and surprises in the remaining three quadrants. These two steps generate a new layer in the coordinate space, in which each node has nine children, and shares one child with each of its four immediate neighbors. This pair of steps is repeated on the top-left subquadrant, until the top layer is created. Pyramid decomposition results in an average subband (LL), as well as horizontal (LH), vertical (HL) and diagonal (HH) details.

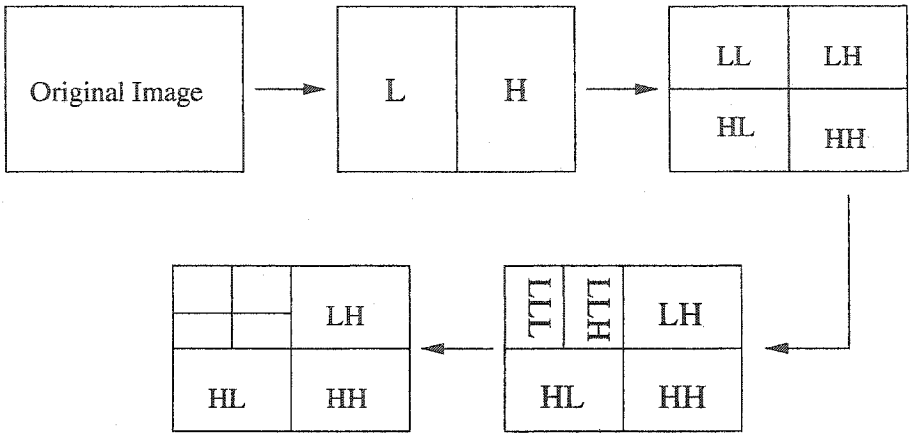


FIGURE 2.7. Pyramid decomposition (taken from [2])

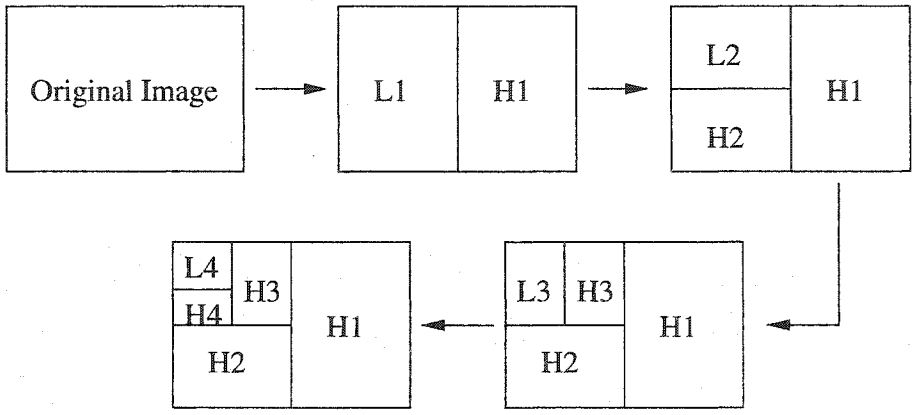


FIGURE 2.8. Quincunx decomposition (taken from [2])

Another commonly used method is *quincunx decomposition*, shown in Figure 2.8. It computes the wavelet transform by alternating between rows and columns. It first computes the transform for all the rows, and generates a low-pass subband $L1$ in the left half and high-pass subband $H1$ in the right half. Each subband images has a vertical resolution twice that of the horizontal resolution. A new layer whose nodes have three children is generated in the coordinate space, just as for the one-dimensional case. The wavelet transform is then applied to the low-pass subband $L1$ only, which generates another pair of subbands $L2$ and $H2$. This process is repeated until the top layer is created. Compared with the pyramid decomposition, this method is more efficient and computationally simple, so it is chosen for our

algorithm. However, quincunx decomposition results in fewer subbands, which may lead to reconstructed images with lower visual quality. As a supplement, Figure 2.9 illustrates the decomposition steps from a spatial perspective.

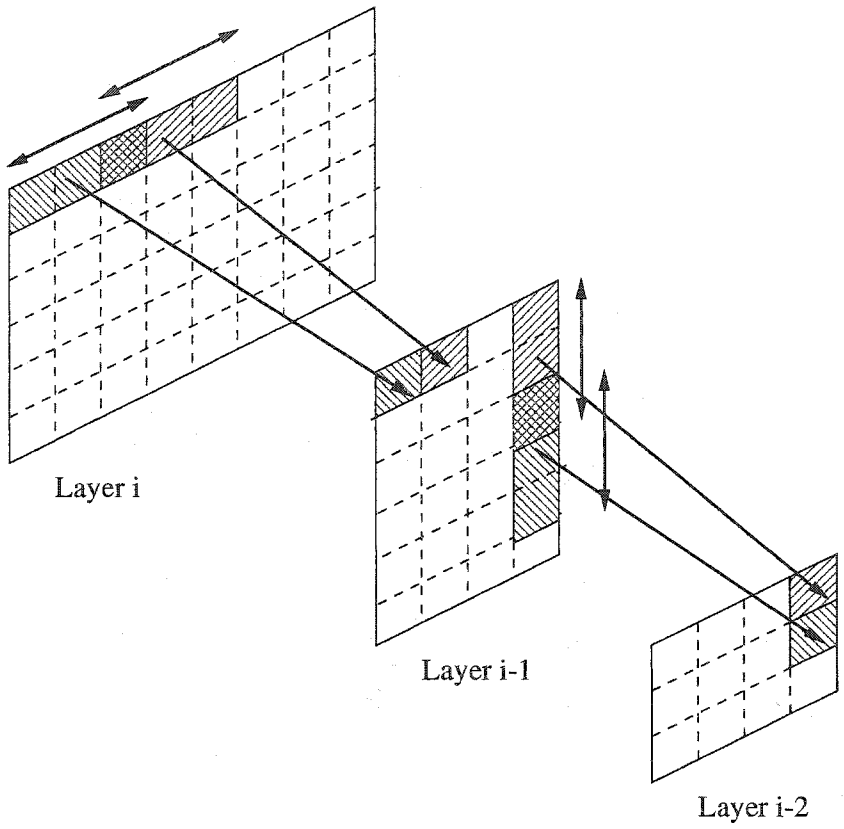
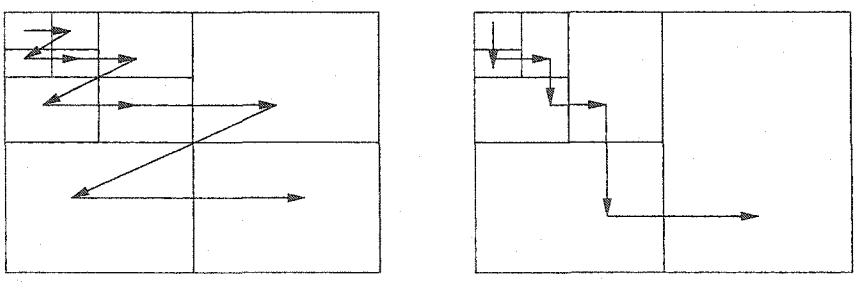


FIGURE 2.9. Two-dimensional decomposition

The numbering of the surprises is performed in such a manner that no child node is scanned before its parent. Figure 2.10 shows the numberings for the pyramid and the quincunx decompositions. In the former case, the scan begins at the lowest frequency subband, denoted as LL_n, and scans subbands HL_n, LH_n and HH_n successively, then moves on to scale $n-1$, and so on. In the latter case, the scan begins at L_n, progresses to H_n, and then moves on to the next scale. Note that each coefficient within a given subband is scanned before any coefficient in the next subband.



(a)Pyramid decomposition (b)Quincunx decomposition

FIGURE 2.10. Scanning order of the subbands

5. Summary

This chapter surveyed a new method, ROHR, to decompose an image into a hierarchical coordinate space. Each node in this structure is associated with a (*value*, *surprise*) pair, which represent the *trends* and *anomalies* of the original image at different scales. The purpose of this decomposition is to eliminate spatial correlation within the image. The generated surprises, from which the coordinate space will be recovered, are encoded in the next step.

We further demonstrated that ROHR is a computationally efficient wavelet transform, involving a simple transform matrix.

The two-dimensional decomposition is an extension of the one-dimensional case. Two methods were concerned, with the quincunx approach favored due to its simplicity and consistency with the one-dimensional case.

CHAPTER 3

Bucket Coding

In this and the following chapters, we discuss methods to encode the surprises and generate the compressed bitstream, as suitable for transmission over a communication network.

Generally speaking, encoding can proceed either sequentially or progressively. Sequential coding, as its name implies, scans the image and encodes the data by rows. During the decoding, the resolution of the recovered image is a constant and later data only adds more area. This coding mode, however, is not suitable for transmission and real-time decoding because users would need to wait for the entire data stream in order to view the full image.

Progressive coding, on the contrary, compresses and encodes the image in multiple passes so that more important information is encoded first. In the decoder, later data adds progressively greater resolution to an already full-sized image, allowing for incremental refinement. This mode is attractive because a user viewing the image can normally recognize most of the image features after only 10-15% of it has been decoded. Moreover, users can effectively control the bit rate. For these reasons, most modern image compression methods are either progressive or optionally so.

Embedded coding is a kind of progressive coding that provides the useful property of allowing truncation of the encoding or decoding at an arbitrary point while still displaying the full image, albeit at less than full quality. In this chapter, we present a novel method, *bucket coding*, which exhibits this desirable property. We first focus on how to sort the data and implement the truncatable progressive coding. The following chapter will discuss methods for the efficient coding of this data.

1. Embedded Coding

A strict definition of embedded coding is:

> If two files produced by the encoder have size M and N bits, with $M > N$, then the file of size N is identical to the first N bits of the file of size M.

In his EZW algorithm [4], Shapiro implements progressive coding in an embedded fashion, so that "all encodings of the same image at lower bit rates are embedded in the beginning of the bit stream for the target bit rate, effectively, the bits are ordered in importance." As discussed earlier, this allows for termination of either the encoding or decoding "at any point, thereby allowing a target rate or distortion metric to be met exactly."

EZW first uses a QMF wavelet transform to obtain a multiresolution representation of the original image. It then provides a compact *significance map*, which indicates the positions of significant coefficients. The embedded coding is then achieved by ordering the wavelet coefficients according to their precision magnitudes, scales and spatial locations, and transmitting the most significant bits of coefficients first. Zerotrees allow the prediction of insignificant coefficients across multiple scales.

Figure 3.1 [6] illustrates the basic idea of transmitting the most significant bits first. Each column contains the bits of a number c. The bits are ordered from top down. The top two rows represent the sign bit and the most significant bit of each number, respectively, while the bottom row is the least significant bit. We assume that the *significance* of a number is determined merely by its magnitude. The coding method adopted by EZW and SPIHT [6] (an extension of EZW) is to transmit the most significant bits first, that is, the MSB (bit 5) of all numbers, then the second most significant bit (bit 4) of all numbers, and so on, until the LSB. Although the practical coding methods of EZW and SPIHT are in fact much more complicated and efficient, in which only the bits corresponding to the arrow sweeps need to be transmitted, the basic logic is similar. When the encoding or decoding is prematurely terminated, all the data can be approximated by as many of the most significant bits that were decoded, without producing artifacts that would indicate where the termination occurred.

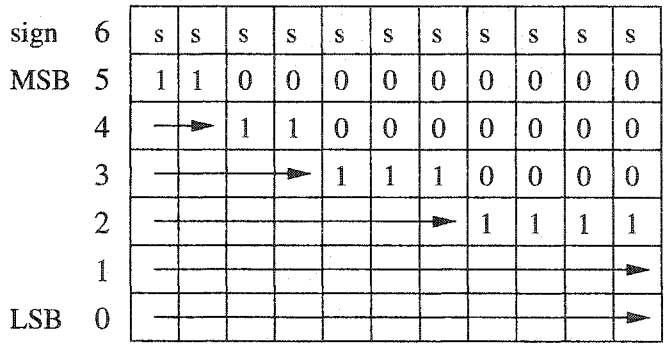


FIGURE 3.1. Simple example of embedded coding taken from SPIHT

2. The Bucket Data Structure

The coding method used in our algorithm is based on the same philosophy as EZW and SPIHT. A major objective in progressive transmission is to select the most important information, which yields the largest distortion reduction, to be transmitted first. Recalling the image decomposition phase in Chapter 2, the surprise of a node

indicates its significance in the reconstruction of the original image. Larger surprise contains more information about the image. By transmitting them first, the generated bit stream is ordered in importance. If truncated at any point, we lose the least significant data and can still reconstruct the image with the most significant data.

As opposed to the embedded coding used by EZW, which sends the most significant *bits* first, our algorithm first sends the most significant *data* in its full precision. Effectively, this assigns a greater significance to the least significant bit of a larger surprise than the most significant bit of a smaller surprise. We are presently considering an improvement to this approach, motivated by the prioritized ordering of SPIHT.

Since we order the surprises according to their magnitudes, regardless of scale and spatial location, we must transmit their positions in the coordinate space together with their magnitudes and signs because the coding order is unknown to the receiver. For this purpose, we traverse the coordinate space and encode the nodes as we number them in Chapter 2, in which nodes at higher scales have smaller numbers than those in lower scales.

The ordered sequence of (*surprise*, *index*) pairs contain all the information of the original image. However, this leads to a decrease in coding efficiency since a naive representation of these pairs is extremely costly. In order to reduce the overhead, we introduce a *bucket* data structure, which contains a set of node index numbers, all of which have the same surprise, s. The bucket index number is determined directly by s. For example, let us consider the following ordered (*surprise*, *index*) sequence:

(100,1), (100,5), (100,6), (-100,7), (48,34), (48,56), (48,90), (48,101), (48,108), (48,110), (-10,20) ...

Collecting these into buckets, we have:

Bucket 100 : 1, 5, 6
Bucket -100 : 7
Bucket 48 : 34, 56, 90, 101, 108, 110
Bucket -10 : 20
⋮

where the bucket's index number represents the surprise of all contained nodes within it. Now instead of coding the ordered (*surprise*, *index*) pairs , we can code the bucket index number followed by an ordered sequence of node numbers. While this step decreases the coding efficiency, we can compress the data even further.

Recalling that the surprise of a node in the coordinate space depends in part on the size of the image region it represents. As a consequence, nodes in one layer are likely to have larger surprises than those in the child layer. To state this more concretely, let $p_x(x)$ be the probability density function (PDF) of surprises in layer L_k, and $p_y(y)$ be the PDF of surprises in layer L_{k+1}, where the root of the coordinate space is in layer L_0. We then have the inequality:

$$E[p_x(x)] \geq E[p_y(y)]$$

which suggests that nodes in one layer generally have greater surprises than their children. In practice, buckets tend to collect nodes from the same layer, or several adjacent layers. Consequently, the index numbers in one bucket are often correlated, and the difference between two consecutive numbers is generally small. This property suggests encoding the *differences* between node indices. Consider the previous example in which Bucket 48 contains the following six indices:

34, 56, 90, 101, 108, 110

The average of these indices is 83.17. Subtracting adjacent pairs (with an implicit 0 in the leftmost position) results in the sequence:

34, 22, 34, 11, 7, 2

The average is these differences 18.33, considerably less than the original values, thus allowing for a significant data reduction. Note that the differences are all positive since the indices are generated in ascending order.

This *differential encoding* is most effective for the small-indexed buckets, which contain many nodes. For example, Figure 3.2 shows the histogram of index differences in Bucket 1 of the 256×256 *Lena* image. The skewed distribution indicates that differences between raw indices are generally small. This figure shows only differences within the range $[0, 100]$. In fact, for differences between 101 and 675 (675 is the maximum difference in this bucket), the histogram contains almost no entry.

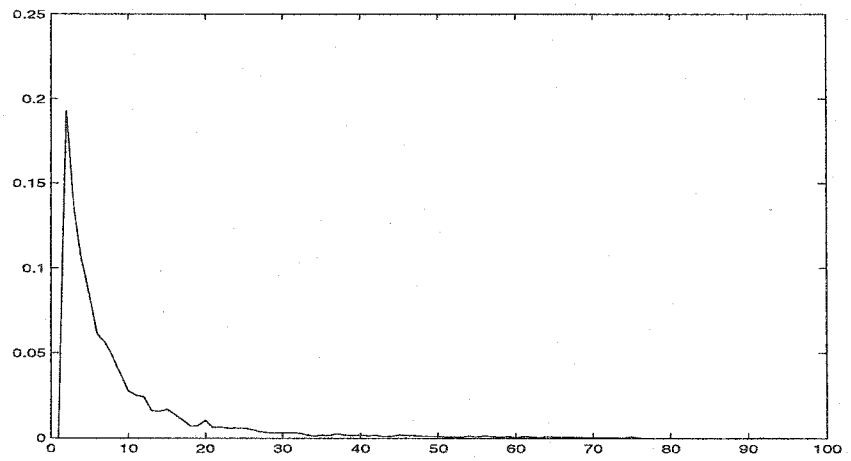


FIGURE 3.2. Histogram of differences in Bucket 1 of 256×256 graylevel *Lena*

3. Bucket Coalescing

The total number of buckets depends on the complexity of the original image, typically ranging from several hundred to several thousand. For each bucket, we first code its header, which contains the surprise value, the number of nodes in the bucket, and

some additional parameters to be described later, and then code the differences one by one. The bucket headers introduce coding overhead, so it is important that these remain small compared to the actual bucket data. Unfortunately, complex images require a large number of buckets, with many containing only a handful of nodes. For such buckets, the headers outweigh the bucket data thus greatly reducing efficiency. To address this problem, we consider coalescing buckets together to generate *super buckets.*

A naive method is to coalesce those whose labels (surprises) have the same k most significant bits. We can group nodes with positive surprises into even-numbered buckets, and those with negative surprises into odd buckets, using the sign bit to choose between even and odd buckets, and the k most significant bits of the magnitude to determine the super bucket label. Suppose the magnitude is expressed as a 16-bit number. Then k most significant bits of surprise are used as an index, effectively, combining every 2^{16-k} buckets into a super bucket according to:

$$\textit{super bucket number} = \begin{cases} (2 \times |\textit{surprise}|) \gg (16-k) & \textit{if surprise} \geq 0 \\ \{(2 \times |\textit{surprise}|) \gg (16-k)\} + 1 & \textit{otherwise} \end{cases}$$

The example for $k = 12$ is summarized in Table 3.1.

Surprises	Super Bucket Number
0, 1, 2, $\cdots$, 15	0
-1, -2, -3, $\cdots$, -15	1
16, 17, 18, $\cdots$, 31	2
-16, -17, -18, $\cdots$, -31	3

TABLE 3.1. A naive method for bucket coalescing

In order to recover completely each surprise in the decoder, we must store the remaining least significant bits as an amendment to the node number.

For example, node 62 has a surprise of -120 ($-$ 0000 0000 0111 1000_b), so for $k = 12$, it belongs to super bucket 15. The four least significant bits (1000_b) are stored with the node number, 62. Suppose, prior to the insertion of node 62, super bucket 15 contains the following:

Super bucket 15	
info	-7
number of data	7
last node	43
data	8(12) , 1(9) , 4(3) , 12(0) , 9(6) , 5(0) , 4(3)

TABLE 3.2. Super bucket 15 before adding node 62

The *info* field expresses both the sign and 12 most significant bits of surprise, and the *last node* field records the full data of the latest inserted node index. After adding node 62, we have:

Super bucket 15	
info	-7
number of data	8
last node	62
data	8(12) , 1(9) , 4(3) , 12(0) , 9(6) , 5(0) , 4(3) , 19(8)

TABLE 3.3. Bucket 15 after adding node 62

Note that the final data element contains 19 (the difference between 62 and the *last node* as well as 8 (1000_b), the four least significant bits of surprise as an amendment. And the new *last node* of super bucket 15 is 62.

When decoding this bucket, each node index can be obtained by accumulating differences, and the surprise is recovered by:

$$surprise = \begin{cases} bucket\ number \times 2^{16-k} + amendment\ bits \\ \qquad\qquad\qquad\qquad (if\ bucket\ number\ is\ even) \\ -[(bucket\ number - 1) \times 2^{16-k} + amendment\ bits] \\ \qquad\qquad\qquad\qquad (if\ bucket\ number\ is\ odd) \end{cases}$$

We call this method *fixed coalescing* because it uses a fixed number k to coalesce the buckets. Although the k amendment bits for each node require extra space, the differences within a bucket become smaller and can thus be coded more efficiently. However, this naive method has a significant disadvantage. We know that a natural image is highly spatially correlated because adjacent pixels are likely to have the same or similar intensities (or colors). As a result, a node's surprise is likely to be small, with most nodes being clustered in the lower labeled buckets, while the remaining sparsely scattered across the higher ones, as shown in Figure 3.3. This is problematic because we cannot code buckets with few nodes efficiently. Because of the low density of nodes in the higher numbered buckets, the fixed coalescing strategy has little ameliorating effect.

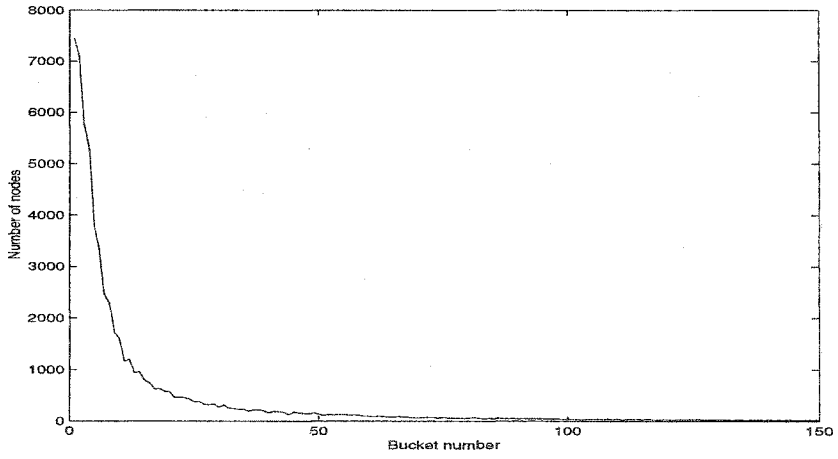


FIGURE 3.3. Nodes distribution among buckets for 256 × 256 grayscale *Lena*

To address the problem, we introduce *logarithmic coalescing*, which coalesces higher numbered buckets more aggressively. As for the fixed coalescing, nodes of positive surprises are assigned directly to even buckets, and those of negative surprises to odd buckets. A factor w is used to control the coalescing, dividing $abs(i)$, $i = 0, \pm1, \pm2, \cdots$, into ranges $[0, 2^w)$, $[2^w, 2^{w+1})$, $\cdots$, $[2^{w+n-1}, 2^{w+n})$. Buckets in the first range $r = 0$ already contain a large number of nodes, so they need not coalesced. In the second range $r = 1$, we coalesce every two adjacent even or odd buckets, and thus generate 2^{w-1} new buckets, whose nodes include one amendment bit. In the third range $r = 3$, we coalesce every four even or odd buckets and generate 2^{w-2} new ones, whose nodes include two amendment bits, and so forth.

Figure 3.4 represents the bitmap of a bucket label after coalescing and illustrates how the bucket is indexed using the surprise value. Let r denote the range of the coalesced bucket, the $\lfloor log_2 r \rfloor + 1$ ($\lfloor x \rfloor$ indicates the largest integer not exceeding x) most significant bits contain the width of the amendment, followed by the w most significant bits of $abs(surprise)$, counted from the first set bit. The least significant bit records the surprise's sign, 0 for positive and 1 for negative.

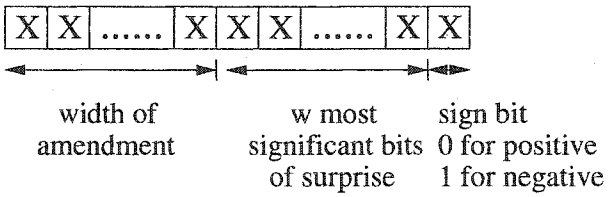


FIGURE 3.4. Bucket label bitmap

For example, consider a node with surprise -307 ($-$ 1 0011 0011_b), and assume the choice of $w = 8$. Because $2^8 < 307 < 2^{8+1}$, it falls into the second range, $r = 1$, in which one amendment bit is recorded. The w most significant bits are 1001 1001, and the sign bit is 1, so the bucket number is (1, 1001 1001, 1_b), or 0x333. Further examples are listed in Table 3.4, with numbers expressed in hexadecimal format for

clarity.

Original Bucket	Coalesced Bucket	Amendment width
0	0	0
+1	0x02	0
-1	0x03	0
+2	0x04	0
-2	0x05	0
...	...	...
+0xff	0x1fe	0
-0xff	0x1ff	0
+0x100, +0x101	0x300	1
-0x100, -0x101	0x301	1
+0x102, +0x103	0x302	1
-0x102, -0x103	0x303	1
...	...	...
+0x1fe, +0x1ff	0x3fe	1
-0x1fe, -0x1ff	0x3ff	1
+0x200, +0x201, +0x202, +0x203	0x500	2
-0x200, -0x201, -0x202, -0x203	0x501	2
...	...	...

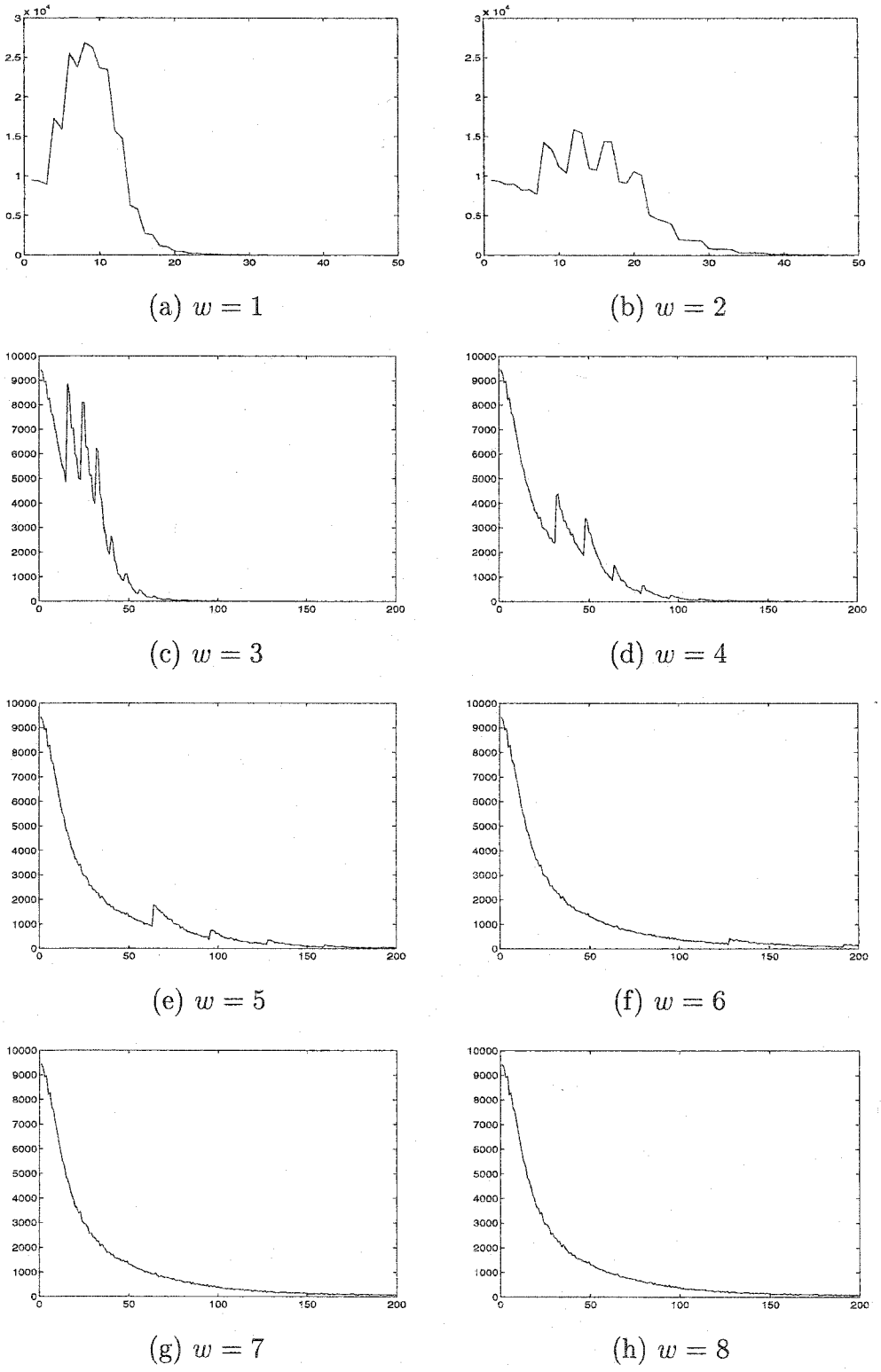
TABLE 3.4. Logarithmic bucket coalescing with $w = 8$

The factor w is an essential parameter that directly influences the performance of coalescing and encoding. Figure 3.5 shows the node distribution among buckets with different coalescing factors for 512×512 grayscale *Mandrill* image. With lower w, the node distribution is more uniform, but more amendment bits are required. Good coding performance is achieved by making a trade-off between an even distribution and fewer amendment bits. Generally speaking, a factor between [3,5] is appropriate for most practical applications. In the remainder of this thesis, we use $w = 4$.

4. Summary

Bucket encoding is a kind of embedded coding that contains all lower rate codes embedded at the beginning of the bitstream. It allows the encoding or decoding to be terminated in the middle of a pass, without introducing artifacts that would indicate where the termination occurred.

Bucket encoding is achieved by ordering nodes according to their surprises' magnitudes and transmitting the most significant nodes first. *Buckets* collect nodes with the same surprise (or wavelet coefficient). The differences between two adjacent nodes within a bucket are computed in order to remove correlation further. In order to improve the encoding efficiency, logarithmic coalescing is used to combine buckets.



(a) $w = 1$ (b) $w = 2$

(c) $w = 3$ (d) $w = 4$

(e) $w = 5$ (f) $w = 6$

(g) $w = 7$ (h) $w = 8$

FIGURE 3.5. Node distribution for different coalescing factor w (Note that the change of scale between (b) and (c) is solely for clarity of illustration)

CHAPTER 4

Encoding

1. Prefix Coding

According to information theory, an efficient variable-size code assigns short code words to common symbols and long code words to rare ones. To avoid ambiguity, such a code must obey the prefix property, that is, no code word is a prefix of any other code word. To understand this property, consider the following code book, which violates the prefix property:

Symbol	a_1	a_2	a_3	a_4
Code	1	01	001	010

If we receive the 15-bit code string:

$$101001001010011$$

we can either decode it as $a_1\, a_2\, a_3\, a_3\, a_2\, a_3\, a_1$, or $a_1\, a_4\, a_2\, a_3\, a_4\, a_2\, a_1$, hence this code is ambiguous. We can remedy this problem by inserting special *break* symbols, for example, 0000, into the code string to separate different code words, but this is obviously inefficient. Alternatively, we can replace the code for a_4 by 000, which restores the prefix property and thus allows for immediate and unambiguous decodability.

If a statistical model of data is known a priori, we can use a statistical coding method, such as Huffman coding or arithmetic coding, to compress data. However, generating such a model can be expensive, and furthermore, may not result in an optimally efficient representation. In the following discussion, we examine alternative prefix codings for the bucket data.

A simple prefix code is the *unary prefix code*, illustrated in Table 4.1. Four code word groups are defined, each of which is comprised of an unambiguous prefix followed by a number of data bits. The code words are designed in such a way that every code, or every set of two codes from the first group, ends on a byte boundary. This method, although simple and fast, is inefficient, because it often expands the data rather than compress it.

	Codeword format	Number of codewords	Range of integers
1	0xxx	8	[0,8)
2	10xx,xxxx	2^6	$[8, 8+2^6)$
3	$110\underbrace{x,x...xx}_{13}$	2^{13}	$[8+2^6, 8+2^6+2^{13})$
4	$111\underbrace{x,x...xx}_{21}$	2^{21}	$[8+2^6+2^{13}, 8+2^6+2^{13}+2^{21})$

TABLE 4.1. General unary prefix coding

2. Multi-range Encoding Methods

Unary prefix coding uses a global, predetermined code book for all the data. Such a method works best when the data is of a single type or is evenly distributed. However, as discussed in Chapter 3, image nodes tend to be distributed in an irregular manner, both across and within buckets, thus suggesting the use of an adaptive method instead.

For each bucket, we can find the maximum difference d_{max}, which determines the data range as well as the the maximum number of bits needed for every element. Since the distribution of node differences within a bucket is skewed toward small values, we can exploit this property by selecting a coarse threshold, d_{mean}, which requires fewer bits than d_{max}, and then encoding each node according to its relationship to this threshold adaptively. For maximum efficiency, d_{mean} is updated for each element based on the running average of the last N nodes, where N is typically chosen from the range $[16, 64]$. For the first N nodes who have fewer predecessors, we pad enough zeros. Given d_{mean} and d_{max}, we encode a node d_i as:

$$\begin{cases} 0\underbrace{xx...x}_{m} & : \quad d_i \leq d_{mean} \\ 1\underbrace{xx...x}_{n} & : \quad d_i > d_{mean} \end{cases}$$

where $n \leq \lfloor 1 + log_2 d_{max} \rfloor$ and $m \leq \lfloor 1 + log_2 d_{mean} \rfloor$ ($\lfloor x \rfloor$ indicates the largest integer not exceeding x). d_{max} is encoded in the bucket header. d_{mean} for each data can be recovered by the decoder since the encoder and decoder are symmetric, and data is received and decoded in the same order as it is sent.

2.1. Encoding Values $\leq$ d_{mean}. As a first approach to encoding data $d_i \leq d_{mean}$, we consider the naive method of representing such values using $k = \lfloor 1 + log_2 d_{mean} \rfloor$ bits, as described above. However, it is possible to do considerably better by exploiting a statistically based adaptive method.

Suppose we are trying to encode a number x in $[0, n)$, where n represents d_{mean}. Any such n can be expressed as the sum of powers-of-two, according to the set bits in its binary representation, for example, $53(110101_b) = 32 + 16 + 4 + 1$. We denote the powers-of-two components of n as $s_1, s_2, \cdots, s_k$, where s_1 is the largest, and introduce $s_0 = 0$, such that $s_0 + s_1 + s_2 + \cdots + s_k = n$. With s_i $(i = 0, 1, 2, ..., k)$, we divide

$[0, n)$ into k subranges, denoted as r_i $(i = 1, 2, ..., k)$:

$$r_i = \left[\sum_{j=0}^{i-1} s_j \, , \; \sum_{j=0}^{i} s_j \right)$$

The length of r_i is s_i, and $r_i > r_j$ $(i < j)$. Using *powers-of-two breakdown* coding, we can express the value of a sample falling in range r_i with s_i bits, preceded by a prefix indicating the range, as shown in Table 4.2.

	Subrange	Codewords	Number of codewords
1	$[0 \, , \, s_1)$	$0 \; \underbrace{xx...x}_{log_2(s_1)}$	s_1
2	$[s_1 \, , \, s_1 + s_2)$	$10 \; \underbrace{xx...x}_{log_2(s_2)}$	s_2
...	...	...	...
k-1	$[\sum_{i=0}^{k-2} s_i \, , \, \sum_{i=0}^{k-1} s_i)$	$\underbrace{11..1}_{k-2}0 \; \underbrace{xx...x}_{log_2(s_{k-1})}$	s_{k-1}
k	$[\sum_{i=0}^{k-1} s_i \, , \, n)$	$\underbrace{11...1}_{k-1} \; \underbrace{xx...x}_{log_2(s_k)}$	s_k

TABLE 4.2. Powers-of-two breakdown coding

For example, [0,53) is divided into four subranges [0,32), [32,48), [48,52) and [52, 53), with associated codewords are listed in Table 4.3. Note that the final codeword for 53 is represented entirely by its prefix 111.

Since the encoding of small numbers is less efficient than large ones, and larger ones occur with lower frequency, we code $d_{mean} - d_i$ instead of d_i. For large buckets, the probability density function of node differences can be approximated by:

$$p(x) = \alpha e^{-\alpha x} (\alpha > 0)$$

where α is usually less than one. In order to obtain this approximation, different regression functions were tested on node distributions of populated buckets and the exponential function was found to have the least deviation in regression. When

	Subrange	Codewords	Code length
1	[0,32)	0: 0,00000 1: 0,00001 2: 0,00010 ... 30: 0,11110 31: 0,11111	6
2	[32,48)	32: 10,0000 33: 10,0001 ... 47: 10,1111	6
3	[48,52)	48: 110,00 49: 110,01 50: 110,10 51: 110,11	5
4	[52,53)	52: 111,	3

TABLE 4.3. Example of powers-of-two breakdown coding with $d_{mean} = 53$

$d_{mean} = 53$, the average code length of $d_{mean} - d_i$ is:

$$3\,\alpha e^{-\alpha 0} + 5\sum_{i=1}^{4} \alpha e^{-\alpha i} + 6\sum_{i=5}^{20} \alpha e^{-\alpha i} + 6\sum_{i=21}^{52} \alpha e^{-\alpha i}\ \ bits$$

Empirically, we find that α is approximately 0.2, yielding an average code length of 5.52 bits, which is significantly better than the 8 bits for unary prefix coding, following the specification of Table 4.1.

2.2. Encoding Values $>$ d_{mean}. When data d_i is larger than d_{mean}, we encode $d_i - d_{mean}$ within the range $d_{max} - d_{mean}$. If d_{mean} is relatively small compared to d_{max}, the data range $d_{max} - d_{mean}$ is relatively large, we found that the powers-of-two breakdown method is not efficient enough. For example, consider the codebook for the powers-of-two breakdown method when $d_{mean} = 1000\ \ (11, 1110, 1000_b)$, as shown in Table 4.4.

	subrange	Codeword	code width
1	[0, 512)	0x,xxxx,xxxx	10
2	[512, 768)	10,xxxx,xxxx	10
3	[768, 896)	11,0xxx,xxxx	10
4	[896, 960)	11,10xx,xxxx	10
5	[960, 992)	11,110x,xxxx	10
6	[992, 1000)	1111,1xxx	8

TABLE 4.4. Powers-of-two breakdown coding with $d_{mean} = 1000$

n	nth subrange	Codeword	Code width	Range of integers
1	$s_1 = \lfloor \frac{3}{6} n \rfloor$	$0 \underbrace{xx...x}_{\lfloor \frac{3}{6} n \rfloor}$	$1 + \lfloor \frac{3}{6} n \rfloor$	$[0, 2^{s_1})$
2	$s_2 = \lfloor \frac{4}{6} n \rfloor$	$10 \underbrace{xx...x}_{\lfloor \frac{4}{6} n \rfloor}$	$2 + \lfloor \frac{4}{6} n \rfloor$	$[2^{s_1}, 2^{s_1} + 2^{s_2})$
3	$s_3 = \lfloor \frac{5}{6} n \rfloor$	$110 \underbrace{xx...x}_{\lfloor \frac{5}{6} n \rfloor}$	$3 + \lfloor \frac{5}{6} n \rfloor$	$[2^{s_1} + 2^{s_2}, 2^{s_1} + 2^{s_2} + 2^{s_3})$
4	$s_4 = n$	$111 \underbrace{xx...x}_{n}$	$3 + n$	$[2^{s_1} + 2^{s_2} + 2^{s_3}, 2^{s_1} + 2^{s_2} + 2^{s_3} + 2^{s_4})$

TABLE 4.5. Quadrange coding

If $\alpha = 0.2$, the average code width is:

$$8 \sum_{i=0}^{i<8} \alpha e^{-\alpha i} + 10 \sum_{i=8}^{i<1000} \alpha e^{-\alpha i} = 9.27 \ bits$$

there is little saving of bit budget compared with the 10 bits native coding. Since it is typically the case that $d_{max} \gg d_{mean}$, we make use of another method, referred to as *quadrange coding*, for any $d_i > d_{mean}$.

Suppose $(d_{max} - d_{mean})$ is an n-bit number. We first divide $(0, d_{max} - d_{mean}]$ into four ranges $\{s_1, s_2, s3, s4\}$, which are $\lfloor \frac{3}{6} n \rfloor$, $\lfloor \frac{4}{6} n \rfloor$, $\lfloor \frac{5}{6} n \rfloor$, and n bits long, respectively, and then code the data $d_i - d_{mean}$ as shown in Table 4.5.

From the previous example, 1000 $(11, 1110, 1000_b)$ is 10 bits long, so for *quadrange coding*, $s_1 = 5, s_2 = 6, s_3 = 8, s_4 = 10$, the average code width is:

$$6 \sum_{i=0}^{i<32} \alpha e^{-\alpha i} + 8 \sum_{i=32}^{i<96} \alpha e^{-\alpha i} + 11 \sum_{i=96}^{i<352} \alpha e^{-\alpha i} + 13 \times \sum_{i=352}^{i<1000} \alpha e^{-\alpha i} = 6.62 \ bits$$

which is much better than either powers-of-two breakdown encoding or unary prefix encoding. Therefore, we explore the *quadrange* coding method for data d_i in $(d_{mean}, d_{max}]$.

3. Summary

An efficient, adaptive multi-range encoding of bucket contents is described. This method relates to two numbers; d_{max}, which is calculated for each bucket, and d_{mean}, which is adaptive to individual data element within a bucket. According to its numerical relationship to d_{mean} and d_{max}, data d_i is encoded by the *powers-of-two breakdown* method or, the *quadrange* method. Both of these satisfy the prefix property so the data can be decoded unambiguously.

Using the notation that $f(a, b)$ describes an encoding function in which a denotes the data being coded and b denotes the coding range, our multi-range encoding algorithm can be summarized as follows:

code the header for bucket k with fixed-length codes

$d_{max} \leftarrow$ maximum difference of nodes in bucket k

$d_{mean} \leftarrow 0$

$w \leftarrow$ amendment width of bucket k

for each data d_i in bucket k:

$d_{mean} \leftarrow$ running average of the last N predecessors of d_i

(i.e. $\frac{1}{N}[\, d_{i-N} + d_{i-N+1} + \cdots + d_{i-2} + d_{i-1}\,]$)

if $d_i \leq d_{mean}$ then

code the prefix 0

code *powers-of-two breakdown* $(d_{mean} - d_i\ ,\ d_{mean}\ )$

else

code the prefix 1

code *quadrange* $(d_i - d_{mean}\ ,\ d_{max} - d_{mean} + 1\ )$

code amendment of d_i in w bits

CHAPTER 5

Color Image Compression

1. Color Spaces

Color is the perceptual result of light in the wavelength range of 400 to 700 nm. Physically, the *intensity* of a color is "a measure over some interval of the electromagnetic spectrum of the flow of power that is radiated from, or incident on, a surface. [15]" *Brightness* is defined as "the attribute of a visual sensation according to which an area appears to exhibit more or less light." The Commission Internationale de L'Éclairage (CIE) defines the *luminance* as the radiant power weighted by a spectral sensitivity function that is characteristic of vision. *Hue* is the attribute of a color perception influenced by its dominant wavelength, for example, blue, green, yellow or red. *Saturation*, or *purity*, is the degree of color concentration at any one wavelength.

There are three types of color photoreceptor cone cells in the human retina, which respond to light and affect perception. Thus, color may be described by the triplet (x, y, z), denoting a point in three-dimensional space; hence the term *color space.*

Color images are generally represented as an array of pixels, where each pixel contains the numerical components that define its color. A practical image coding system must be computationally efficient, cannot afford unlimited precision, and needs to cover a reasonably wide range of colors. In the remainder of this section, we discuss two

common color systems in use today, RGB and YUV.

The RGB model, shown in Figure 5.1, is the color space favored by CRT monitors and raster graphics display. RGB is composed of the additive primary colors, red, green, and blue, in which the desired color can be produced by an addition of appropriate quantities of each primary component. The diagonal from black (0,0,0) to white (1,1,1) represents varying intensity of grayscale.

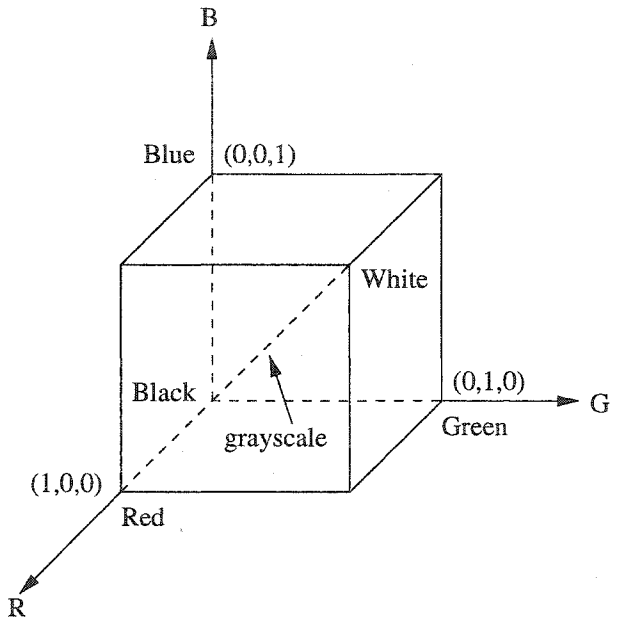


FIGURE 5.1. RGB color model

The YUV model, commonly used in video transmission, was invented for backward compatibility of color television broadcasts with black and white televisions. YUV can be derived from RGB through the following matrixed operation:

$$\begin{vmatrix} Y \\ U \\ V \end{vmatrix} = \begin{vmatrix} 0.30 & 0.59 & 0.11 \\ 0.70 & -0.59 & -0.11 \\ 0.30 & -0.59 & 0.89 \end{vmatrix} \begin{vmatrix} R \\ G \\ B \end{vmatrix}$$

Y refers to the luminance of the color, in other words, its grayscale value, while U and V refer to the differences between R and Y, and B and Y, respectively.

The inverse transformation matrix is:

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} 1 & 1 & 0 \\ 1 & \frac{-0.3}{0.59} & \frac{-0.11}{0.59} \\ 1 & 0 & 1 \end{vmatrix} \begin{vmatrix} Y \\ U \\ V \end{vmatrix}$$

2. Structure of Color Image Compression

Since an RGB image can be regarded as three separate color planes, the most straightforward compression algorithm is to transform and encode the three color planes independently, as shown in Fig 5.2.

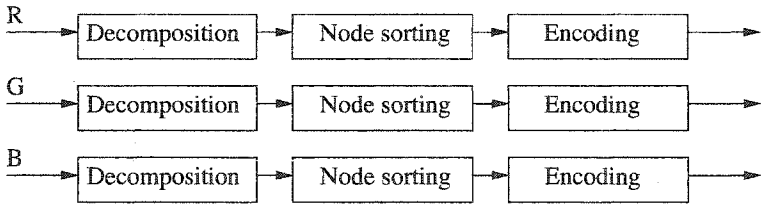


FIGURE 5.2. Naive color image compression in RGB

A more efficient method is shown as Fig 5.3, in which the image is first transformed from RGB to its YUV representation which is then encoded in separate channels. Since the human eye is sensitive to small changes in luminance (brightness) but less so to small changes in chrominance (color), the U and V components may be compressed somewhat without introducing distortions to which the eye is sensitive. Many lossy compression methods, such as JPEG [**10**], take advantage of this observation, compressing U and V more aggressively than Y, or even downsampling the

two chrominance components in order to reduce bandwidth.

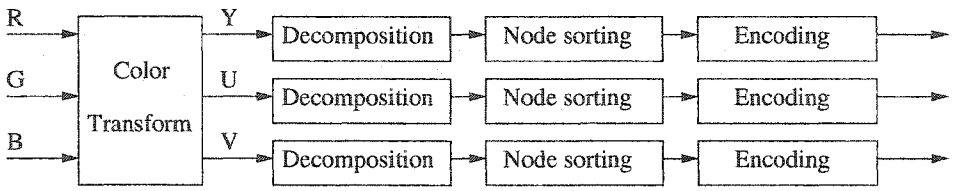


FIGURE 5.3. Color image compression through transformation to YUV

We may also compress the three color components, either in RGB and YUV space, together as a vector, as shown in Fig 5.4. In the decomposition step, each node is associated with a three-dimensional surprise vector, representing the contribution of each color component. In bucket encoding, the significance of node is determined by some *matrixed* quantity. This method, however, risks affecting the hue of the image because one color component may lose more information than the others.

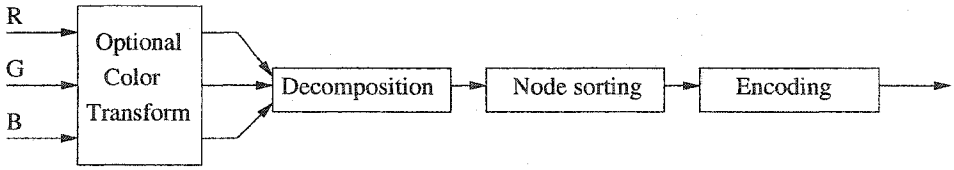


FIGURE 5.4. Color image compression by vector

CHAPTER 6

Experimental Results

In this chapter, we review our experimental results of encoding and decoding images with the ROHR algorithm. Results for grayscale and color image encoding are described. We discuss the quality of the recovered image and compare the results with other relevant compression algorithms, from the aspects of both compression quality and computational efficiency. Advantages and disadvantages of our algorithm are presented and analyzed.

First, we briefly summarize the encoding steps involved in ROHR, as follows: Images first undergo a wavelet transform and quincunx decomposition into a two-dimensional coordinate space. The transform coefficients or surprises are then sorted into buckets according to their magnitudes, and coded using a multi-range encoding method, which requires a mean estimate of the difference between successive values. This estimate is based on a running average of N samples; empirically we find $N = 32$ provides good results. Buckets are coalesced using a coalescing factor w; we use $w = 4$. Finally, in order to satisfy the embedded coding property, we transmit buckets in descending order of their label magnitudes.

1. Grayscale Image Compression

The encoder was applied to the standard black and white 256×256 *Lena* image and the 512×362 *Peppers* image. Coding results for *Lena* are summarized in Table 6.1 and Figure 6.3. Similar results for *Peppers* are shown in Table 6.2 and Figure 6.4.

The compression ratio is defined as:

$$Compression\ ratio = \frac{size\ of\ output\ stream}{size\ of\ input\ stream}$$

Note that our bit rates are not entropy estimates, but are calculated from the actual size of the compressed files, and expressed in bits per pixel (bpp), since they represent the average number of bits needed to express one pixel in the original image.

Error of the reconstructed image is measured by its Peak Signal to Noise Ratio (PSNR). Denoting the pixels of the original image by P_i and the pixels of the reconstructed image by Q_i (where $1 \leq i \leq n$), we define the Mean Square Error (MSE) between two images as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(P_i - Q_i)^2$$

The PSNR is defined as:

$$PSNR = 20log_{10}\frac{max_i|P_i|}{\sqrt{MSE}}$$

PSNR is dimensionless and expressed in *decibels* (db). Greater resemblance between images implies a smaller MSE and thus a higher PSNR.

Different bit rates are achieved by terminating either the encoding or decoding at some desired point. No artifacts are produced in the recovered image that would indicate where the termination occurred. This is consistent with our claim that transmitting the transform coefficients in descending order of their surprise magnitudes offers an

effective implementation of embedded coding. This further demonstrates that our image decomposition method, described in Chapter 2, can successfully detect *anomalies* in the original image, and use the magnitudes of these surprises to reconstruct the image.

ROHR shows good performance at high bit rates, providing nearly indistinguishable reconstructed images from the originals. Lossless compression is achieved by transmitting all non-zero nodes, and performance drops linearly with the decrease of bit rate, as details are gradually lost. At very low bit rate, e.g. 0.1 bpp, almost all the image details are lost, and artifacts become obvious.

Bitrate(bpp)	Bytes	Compression Ratio	Number of Nodes	PSNR(db)
6.804	55,742	1.2:1	65,535	∞
4.064	33,290	2:1	26,214	42.396
2.027	16,605	4:1	10,485	33.008
1.023	8,379	8:1	4,587	27.394
0.509	4,173	16:1	1,966	23.696
0.102	842	80:1	249	18.062

TABLE 6.1. Coding results for 256 $\times$ 256 grayscale *Lena*

Bitrate(bpp)	Bytes	Compression Ratio	Number of Nodes	PSNR(db)
6.531	151,314	1.2:1	185,343	∞
4.038	93,558	2:1	83,404	43.169
2.024	46,886	4:1	31,508	35.791
1.018	23,588	8:1	13,900	30.922
0.510	11,822	16:1	6,116	26.592
0.105	2,434	80:1	926	19.912

TABLE 6.2. Coding results for 512 $\times$ 362 grayscale *Peppers*

The performance of ROHR was also compared to the method of Set Partitioning in Hierarchical Trees (SPIHT) [6][7] and the JPEG standard [10] [11]. SPIHT is an extension of EZW [4]. It first performs a wavelet transform with 9-tap symmetric

quadrature mirror filters (QMF) [8], then sorts the coefficients according to their magnitudes and transmits the most significant bits first. The spatial orientation tree explores the self-similarity between subbands and is used to optimize the ordering principle. It allows the user to determine the bit rate, and can achieve lossless compression at the highest rate. JPEG does not allow the users to truncate the bit stream, but permits a choice of *quality factor*. The performance of these three methods is compared in the plots of Figure 6.1, and the reconstructed images of Figure 6.5. Generally, the PSNR of ROHR is 5db lower than JPEG and 10db lower than SPIHT. At very low bit rates, both ROHR and JPEG exhibit prevalent blocking effects, while SPIHT preserves an acceptable level of visual quality.

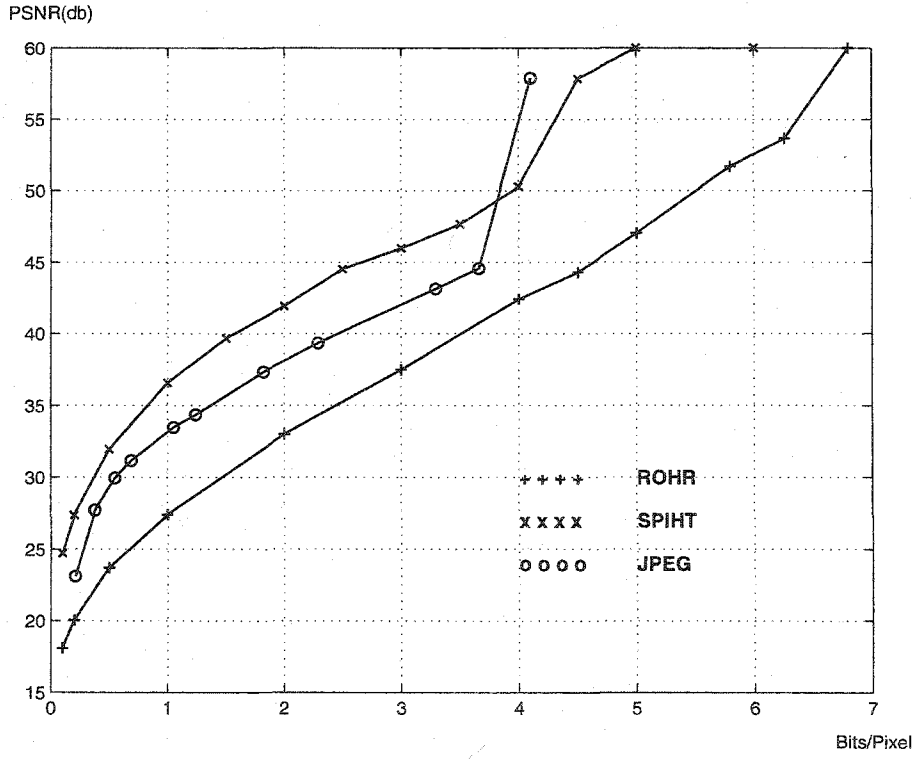


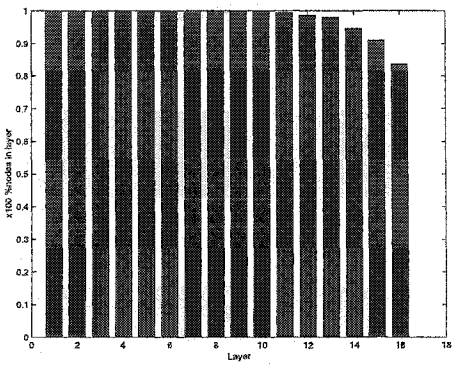
FIGURE 6.1. Comparative performance evaluation of ROHR

Further analysis reveals several reasons for the inferior performance of ROHR as compared to SPIHT and JPEG. First, as discussed in Chapter 2, the wavelet transform

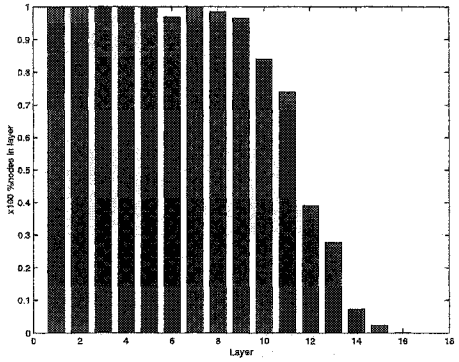
bases are not orthogonal, thus producing correlated coefficients, which results in a larger compressed file because of redundancy.

Second, as discussed in Chapter 3, since surprises in the parent layer are statistically twice as large as those in the child layer, surprises from a given layer are often grouped together. While this property helps us achieve an efficient bucket coding, it "implicitly de-emphasizes the outliers, which represent the most significant anomalies or edges."[4] Because the significance of a node is determined by its surprise magnitude, a decrease in bit rate usually leads to the discarding of more nodes from finer, higher-resolution layers, than from coarser, lower-resolution layers. Thus, image details, or edges, will be lost earlier than *trends*. This point is made more explicit in Figure 6.2, which indicates the percentage of nodes in each layer that are transmitted at various bit rates.

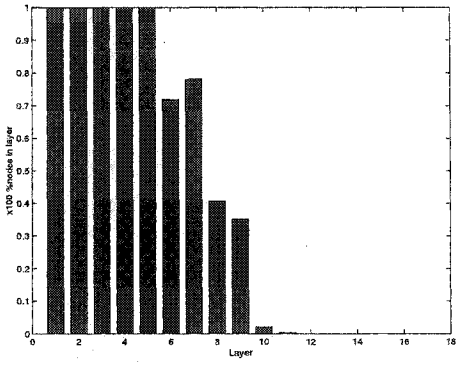
Solving this problem involves two aspects. First, the transform matrix should be normalized, so that the high-pass filter, which computes the surprises has zero gain, and the low-pass filter, which computes the trends, has a unity, or near-unity gain. Second, we should take into consideration the effect of inter-layer correlation. It is important to realize that in a hierarchical subband decomposition system, the wavelet filters and the decomposition method are independent. In a pyramid decomposition, for example, with the exception of the highest frequency subbands, every coefficient at a given layer is related to four coefficients at the next finer scale of similar orientation. We could abstract this decomposition method as a *quadtree* structure, in which every non-leaf node has four children in the same orientation. Note that the meaning of *children* in this context is different from that of image decomposition into a coordinate space. Similarly, the quincunx decomposition could be regarded as a *bintree* structure, with every non-leaf node in this structure having two children. Based on these models, Shapiro [4] presented the hypothesis in his EZW algorithm and then



(a) 8 bits/pixel



(b) 1 bit/pixel



(c) 0.1 bit/pixel

FIGURE 6.2. Nodes being sent in each layer for 256 × 256 grayscale *Lena*

proved that "if a coefficient at a coarser scale is insignificant with respect to a threshold then all of its descendants are also insignificant." This is also stated in SPIHT [**6**] as "a spatial self-similarity between subbands," and "the coefficients are expected to be better magnitude-ordered if we move downward in the pyramid following the same spatial orientation." To take advantage of this property, SPIHT checks every coefficient recursively. If it and any of its direct descendants are significant to this threshold, these are coded first. This scheme sorts the coefficients according not only to their magnitudes, but also to their scales and spatial locations. It has proved to be effective in EZW and SPIHT, and could be used to improve the performance of

ROHR. This remains a subject of ongoing research.

Despite its inferior compression performance, ROHR surpasses both JPEG and SPIHT in execution speed, which is our major concern. Most importantly, encoding time must be minimized because this stage provides uniform data to all clients, while different users are free to configure their decoders as desired. Table 6.3 lists the encoding times (including CPU and I/O time) for ROHR, SPIHT and JPEG, all running on a dual-processor Intel i686 workstation, for the 256×256 *Lena* image. Although the execution speeds of them depend on the size of the compressed file, the execution speed of ROHR is superior to both SPIHT and JPEG, and demonstrates better scaling performance with increased bit rate.

Bit Rate	8	4	2	1	0.5	0.1
ROHR (s)	0.041	0.034	0.027	0.025	0.023	0.023
SPIHT (s)	0.15	0.14	0.09	0.05	0.04	0.03
JPEG (s)	0.317	0.233	1.167	0.150	0.133	NA

TABLE 6.3. Comparative execution times of ROHR, SPIHT and JPEG for encoding 256×256 grayscale *Lena* at various bit rates

2. Color Image Compression

The ROHR color image encoder was applied to the 512×512 pixels YUV 4:2:0 *House* image in which there are four parts of luminance, and one part each of the two chrominances. The results were summaried in Table 6.4.

We also compare the performance of this compression method with SPIHT, with the results plotted in Figure 6.6, and reconstructed images shown in Figure 6.8. Their execution speed comparison is shown in Table 6.5. While SPIHT provides superior

Bit Rate (bpp)	Bytes	Compression Ratio	Total Bits	PSNR (db)
8.49	278,200	1.9:1	2,225,602	inf.
7.00	229,376	2.3:1	1,835,008	49.79
3.00	98,304	5.3:1	786,432	35.89
1.00	32,768	16:1	262,144	28.96
0.50	16,384	32:1	131,072	26.11

TABLE 6.4. Compression results of 512 × 512 YUV 4:2:0 *House* image

Bit Rate	8.49	7	5.44	4	3	2	1	0.5
ROHR (s)	0.16	0.15	0.14	0.13	0.12	0.12	0.11	0.11
SPIHT (s)	.	.	0.40	0.33	0.26	0.20	0.14	0.11

TABLE 6.5. Comparative execution times of ROHR and SPIHT for encoding 512 × 512 YUV 4:2:0 *house* image at various bit rates

quality decodings than ROHR, the computational efficiency of the latter make it better suited to the needs of real-time bandwidth-limited channels.

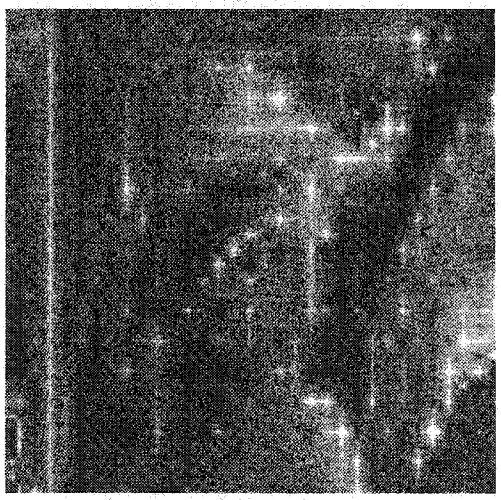
(a) Original image at 8 bits/pixel

(b) Lossless compression at 6.8 bits/pixel

(c) 4 bits/pixel

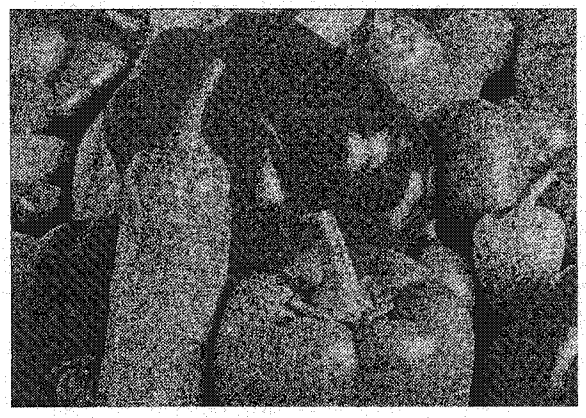
(d) 1 bit/pixel

(e) 0.5 bit/pixel



(f) 0.1 bit/pixel

FIGURE 6.3. Recovered images of 256 × 256 grayscale *Lena*



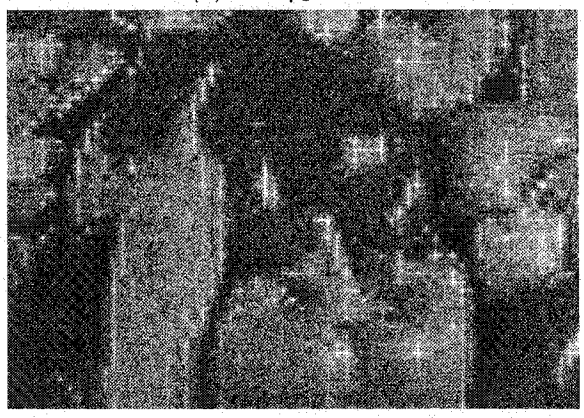
(a) Original image at 8 bits/pixel

(b) Lossless compression at 6.5 bits/pixel

(c) 4 bits/pixel

(d) 1 bit/pixel

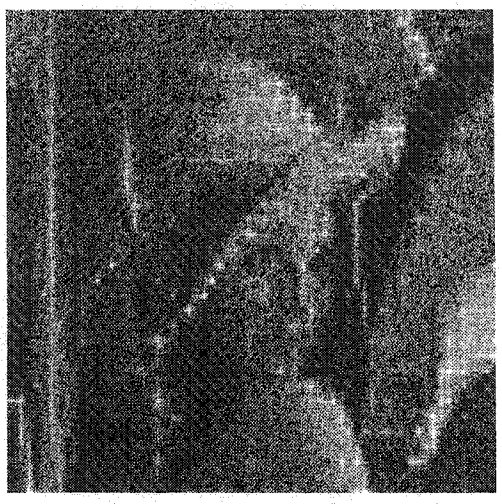
(e) 0.5 bit/pixel



(f) 0.1 bit/pixel

FIGURE 6.4. Recovered Images of 512 × 362 grayscale *Peppers*

(a) ROHR at 1 bit/pixel



(b) ROHR at 0.2 bit/pixel

(c) JPEG at 1 bit/pixel

(d) JPEG at 0.2 bit/pixel

(e) SPIHT at 1 bit/pixel

(f) SPIHT at 0.2 bit/pixel

FIGURE 6.5. Performance comparison of ROHR, JPEG and SPIHT operating on 256 × 256 grayscale *Lena*

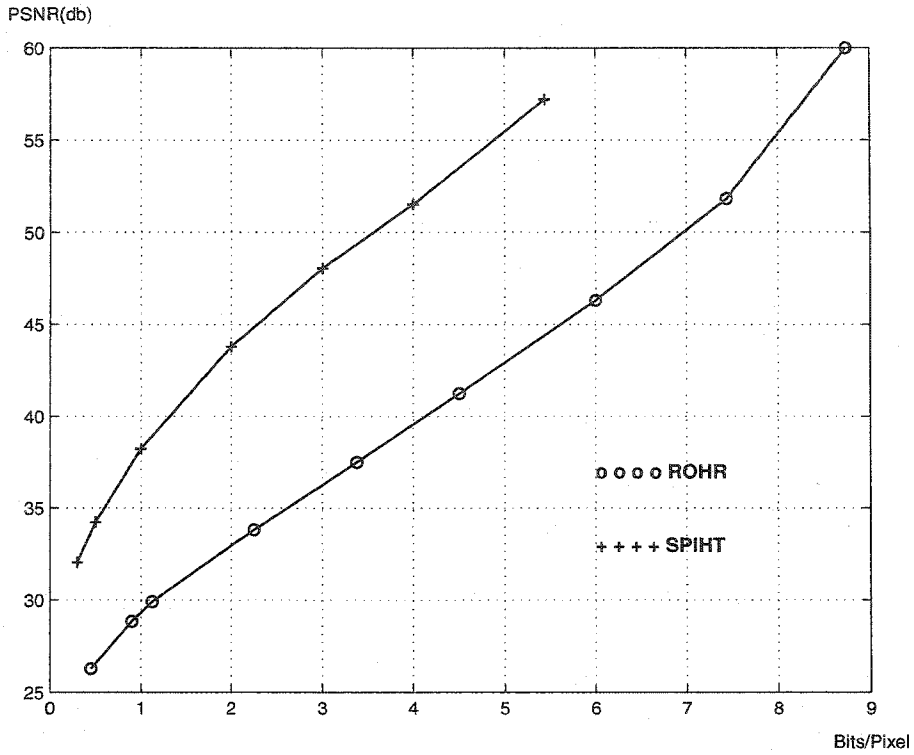


FIGURE 6.6. Comparative performance of color image compression

(a) Original image at 24 bits/pixel

(b) 8 bits/pixel

(c) 2 bits/pixel

(d) 0.5 bit/pixel

FIGURE 6.7. Recovered images of 512 × 512 YUV 4:2:0 *House*

(a) ROHR at 3 bit/pixel

(b) ROHR at 0.5 bit/pixel

(c) SPIHT at 3 bit/pixel

(d) SPIHT at 0.5 bit/pixel

FIGURE 6.8. Performance comparison of ROHR and SPIHT operating on 512×512 YUV 4:2:0 *House* image

CHAPTER 7

Conclusions

1. Summary of Work

In this thesis, we have presented a new image compression algorithm that performs spatial image decomposition and bucket coding in an embedded format. The main features of this algorithm are:

1. Spatial decomposition decorrelates the image by expressing the data as a series of multiresolution *surprises* in a hierarchical structure, reflecting the *anomalies* at different scales. This decomposition is similar to a wavelet transform in that it divides the image into several subbands, but differs by using integers, rather than floating point operations, for accuracy and computational efficiency.

2. Buckets collect nodes with the same surprises, and in effect, order the nodes according to the magnitude of their surprises. Transmitting nodes with higher surprises first satisfies the requirements of embedded coding, thus allowing the encoding or decoding to be stopped at any point. Bucket coalescing balances the distribution of nodes, and increases coding efficiency by decreasing the frequency of sparsely populated buckets.

3. Adaptive multi-range coding of the bucket contents further capitalizes on intra-bucket correlation. Since no statistical model or training process is required, this offers a general-purpose approach that performs well with many data type.

4. This algorithm is computationally efficient, and can be applied for live video rates at resolution of 256×192.

2. Future Work

The design of wavelet transforms remains an essential problem for image compression, directly influencing the image representation and compression performance. The wavelet transform (or spatial image decomposition) used in this thesis is not orthonormal and thus produces correlated coefficients, which results in a larger compressed file and decreases the compression efficiency. A future improvement of the ROHR algorithm therefore involves designing an orthonormal and computationally efficient wavelet transform kernel. Furthermore, since the surprise of every M-node is ignored during the encoding and recovered from its neighboring N-nodes during the decoding, its accuracy is completely dependent on its neighbors. Any distortion in the values of its neighbors will cause an opposing error in its value. This error will be inherited by the M-node's children, propagated to a wider area, and accumulated layer by layer until the bottom of the coordinate space, which represents the reconstructed image. Consequently, any deviation in a coarser-resolution layer will result in a larger deviation over a wider area in the finer-resolution layers. An important factor to increase compression performance is thus the control and remedy of this kind of deviation.

Another important problem is the exploration of the inter-scale correlation based on self-similarity in bucket encoding. Predicting a transitive insignificance across scales could provide substantial coding gains.

Data reliability and robustness is another problem deserving further investigation. ROHR packs data bit by bit into the output stream in order to increase compression efficiency. However, the bucket format and multi-range coding method make the bits highly correlated. If only one bit is distorted during transmission, the entire decoding process could collapse. Unfortunately, image compression in removing redundancy also decreases reliability. In certain applications such as narrow-bandwidth video transmission, data robustness is as important as compression performance. Therefore, the generation of reliable and efficiently compressed data is not only a concern of ROHR, but also problem of image compression in general.

REFERENCES

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

[2] D. Salomon, Data Compression, the Complete Reference, Springer, 2000

[3] J. M. Shapiro, "An embedded wavelet hierarchical image coder", *1992 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 92)*, San Francisco, CA, Mar. 1992

[4] J. M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients", *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445-3462, Dec. 1993

[5] A. Said and W. A. Pearlman, "Image compression using the spatial-orientation tree", *1993 IEEE International Symposium on Circuits and Systems (ISCAS 93)*, Chicago, IL, pp. 279-282, May 1993

[6] A. Said, A. Pearlman, "A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 243-250, June 1996

[7] SPIHT free software, http://www.cipr.rpi.edu/research/SPIHT/

[8] E. H. Adelson, E. Simoncelli, and R. Hingorani, "Orthogonal pyramid transforms for image coding", *Proceedings SPIE, Visual Communication and Image Processing II*, Cambridge, MA, vol. 845, pp. 50-58, Oct 1987

[9] Y. Huang, H. M. Driezen, and N. P. Galatsanos, "Prioritized DCT for Compression and Progressive Transmission of Images", *IEEE Transactions on Image Processing*, vol. 1, pp. 477-487, Oct. 1992

[10] W. B. Pennebaker and J. L. Mitchell, JPEG still image data compression standard, Van Nostrand Reinhold, 1993

[11] Independent JPEG Group's free JPEG software, http://www.ijg.org, 1998

[12] S. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 674-693, July 1989

[13] S.Mallat, "Multifrequency channel decompositions of images and wavelet models", *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol.37, pp.2091-2110, Dec.1990

[14] R. A. DeVore, B. Jawerth and B. J. Lucier, "Image compression through wavelet transform coding", *IEEE Transactions on Information Theory*, vol. 38, pp. 719-746, Mar. 1992

[15] Charles Poynton, "A guided tour of color space,"
http://www.inforamp.net/~poynton/papers/Guided_tour/abstract.html

[16] T. Nakachi, T. Fujii and J. Suzuki, "Lossless and near-lossless compression of still color images", *Proceedings. 1999 International Conference on Image Processing (ICIP 99)*, vol. 1, pp. 453 -457, 1999

Document Log:

Manuscript Version 0

Typeset by $\mathcal{AMS}$-LaTeX — 29 September 2002

YUAN ZHANG

Typeset by $\mathcal{AMS}$-LaTeX