# SELECTING DON'T CARE BITS IN JPEG2000 ROI CODING

by

Jamshid Ameli
B.Sc. Sharif University of Technology, 1998

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

In the School
of
Engineering Science

SIMON FRASER UNIVERSITY

February 2004

# Approval

| | |
|---|---|
| **Name:** | Jamshid Ameli |
| **Degree:** | Master of Applied Science |
| **Title of Thesis:** | Selecting "Don't Care" Bits in JPEG2000 ROI Coding |

**Examining Committee:**

Chair: Dr. Mirza Faisal Beg

Dr. Rodney Vaughan
Senior Supervisor
Professor of Electrical Engineering
School of Engineering Science, SFU

Dr. Mark Drew
Supervisor
Associate Professor of Computing Science, SFU

Dr. Torsten Moeller
Examiner
Assistant Professor of Computing Science, SFU

**Date Approved:**

# ABSTRACT

At the time of this writing, JPEG2000 is the most recent standard for still image compression. One of the important features of the standard is region-of-interest (ROI) coding, which allows user-defined parts of an image to be coded with higher quality than parts in the "background" (BG). In fact, the ROI feature enables a general non-uniform distribution of quality for different parts of an image.

There are two main types of applications for ROI coding. In many applications such as in medical images, a high quality or even lossless quality is needed for only a small part of an image. In these cases, with using ROI coding technique, there is no need to code the whole image with high quality. The image background is coded with moderate quality and as a result a higher compression ratio is achieved. The second type of application is in progressive transmission over low-capacity channels. In these cases, the ROI is transmitted before the background, and at each point during the transmission, the quality of the ROI is better than the BG. Therefore, once the desired image fidelity is provided at the receiver, transmission can be terminated. Also the relative quality between the ROI and BG regions can be selected during the coding process.

The JPEG2000 standard only exists for the decoder side. Any algorithm can be used at the encoder as long as it is compatible with the standard's decoder. In this thesis, we propose a modification for the current JPEG2000 encoders in order to improve the compression performance of the ROI mode of the standard. ROI coding is accomplished in JPEG 2000 by de-emphasizing the wavelet coefficients associated with the non-ROI regions of the image. A number of extra bits appear below the least significant bit of the ROI samples after the shifting process. These bits need to be coded at the time of bit-plane coding, but they are discarded by the decoder. The usual procedure is for the current encoders to set these "don't care" bits to zero. In this thesis, we examine the possible strategies for setting these “don’t care” bits and then propose a method that exploits the state of the JPEG2000 entropy coder to set the values of these bits in a more intelligent way. The method has been observed to reduce the number of bits required to represent ROI code-block by up to 7%, with the bit-stream remaining JPEG2000 compliant.

# DEDICATION

To my family

## ACKNOWLEDGMENT

I would like to express my gratitude to my late senior supervisor Dr. Jacques Vaisey who, unfortunately, passed away two months before my defence. He was a great professor, a great teacher and, above all, a great human. This work would not have been possible without his guidance.

Special thanks to my supervisory committee members, Dr. Rodney Vaughan, Dr. Mark Drew, Dr. Torsten Moeller and Dr. Faisal Beg for their comments on my thesis. Also thanks to Dr. Kamal Gupta for his help at SFU.

Thanks to Vahid Shahrezaei for his idea about the Simulated Annealing method and thanks to Tong Jin who always answered my questions patiently. Also thanks to my friends for their friendship and support during the period of my studies at SFU.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1
# INTRODUCTION

## *Introduction*

In the last couple of decades, we have been witnessing a revolution in the information technology, and this revolution has changed the way we communicate. The emergence and development of the internet and the growing number of mobile phones and digital TV users are part of this revolution. Data compression has definitely had a very important role in the development of the multimedia revolution [27]. In fact, without the current data compression techniques, the internet could not have the size and shape of what it has today, and the mobile phones and digital TV's could not be as widespread as what we see these days. The music stored in CD's, the movies stored in DVD's and the images stored in digital cameras are all compressed with a type of data compression standard/algorithm.

Data in the shape of an image is widely used in our daily communications. One can hardly find a page in the internet that does not contain at least one image. Also, if we could search the information in the computers around the world, maybe it would be difficult to find a computer without image files stored in it.

Storing an uncompressed digital image needs a large memory space. As an example, a typical 500x500 pixel color image requires 750kB of memory (24bits/pixel), however, the image file can be compressed with a ratio of, say, 1:10, and the reconstructed image cannot probably be distinguished from the original image by a human eye. Note, however, that in some applications such as medical images or preservation of artworks [28] the above-mentioned compression ratios cannot be achieved as in these cases, no distortion is tolerated in the reconstructed image. Furthermore, the image compression techniques are also used in most of the video compression algorithms and standards. In fact, in many video compression standards, an image compression technique is used to code some non-consecutive video frames and the frames between them are interpolated using motion compensation techniques [27] to exploit the dependency between the frames.

The latest image compression standard is the JPEG2000, which compared to the previous standards, allows for both increased flexibility and performance. The main contribution of this thesis is a modification for the ROI mode of JPEG2000 coder to improve the compression performance.

## 1.1 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 provides background information for the main building blocks of a very large group of image compression algorithms/standards including JPEG2000. Transform coding, quantization and entropy coding techniques are the main subjects discussed in this chapter.

Chapter 3 starts with introducing the JPEG2000 standard features and the state-of-the-art compression technique utilized in this standard. The problem that is dealt with in this thesis is defined at the end of this chapter.

Chapter 4 explains the complete details of the problem and continues with the way we approached it. Our suggested method for dealing with the problem wraps up this chapter.

Chapter 5 reports the experimental results. Our designing experiment to test the suggested method is explained and the improvements possible with our method over the usual procedure used in the current JPEG2000 coders are provided.

Chapter 6 concludes the thesis and suggests future works.

And finally the appendix reports more experimental results and reproduces the test images that are used for experiments throughout the thesis.

# CHAPTER 2
# IMAGE PROCESSING

## *Introduction*

In this chapter we will briefly review the literature behind the current image compression algorithms and standards, and we will keep our focus on JPEG2000. Our goal here is to provide just enough information for a person with electrical engineering background to understand the research work explained in this thesis. For more in depth information, interested readers are referred to the references introduced throughout the chapter for each subject.

A digital image is represented by a two dimensional array. Each member of the array is called a pixel, with the value of each pixel representing the intensity or brightness of the image at that location. In most images, the values of the neighboring pixels are highly dependent, and almost all of the common image compression standards and algorithms exploit this fact to compress the bitstream needed to store an image.

The basic structure of many image compression algorithms and standards, including JPEG2000, is illustrated in Figure 2.1. The major building blocks are transform coding, quantization and entropy coding. In this chapter each of these techniques will be discussed in detail. Note that this block diagram belongs to the encoder part of a codec. In the decoder, these steps are simply reversed in order to construct a lossy or lossless version of the encoded image, depending upon the type of the compression technique. It is also worth mentioning that there exist other image compression methods that work based on different structures. However, as the focus of this thesis is the JPEG2000 standard, we only explain the theory related to this standard.

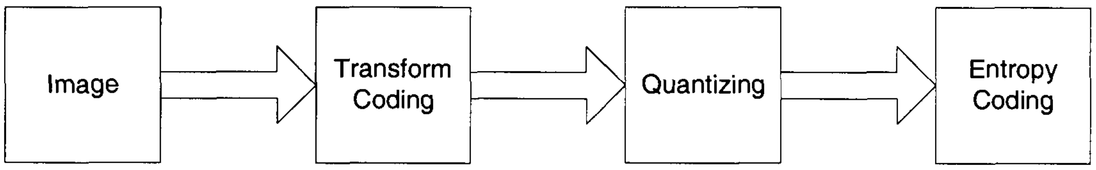


**Figure 2.1. Building blocks of one category of image compression algorithms (coder)**

## 2.1 Entropy coding

Entropy, first introduced by Shannon [29], is a measure for information. There is high information in an event if the probability of occurring of the event is low and vice versa. As an example, suppose you receive a phone call in January from a friend of yours in the Northern Territories. He says the weather there is very cold. This sentence really does not carry much information, as at that time of the year you do expect the weather to be cold there. However, if a top seed tennis player is beaten by a Wimbledon wildcard in the first round of tournament, all the sports channels will talk about the unexpected news, as there is high information in it.

Regarding the mathematical measurement of entropy, suppose we have a source, $X$, that generates a set of independent symbols $x_i$, with $p_i$ the probability of occurring each symbol. The zeroth-order entropy associated with the source is then computed as follow,

$$H(X) = -\sum_i p(x_i)\log_b p(x_i) \tag{2.1}$$

The parameter *b*, determines the unit of information. With $b = 2$ the unit is bit.

Shannon showed that, theoretically, an ideal coding algorithm can code the independent symbols generated by a source, with an average number of symbols equal to the entropy of the source. The above argument holds when the source symbols are completely independent. In cases where there is dependency between different symbols, as in image coding, the dependency can be exploited in the coding process, and the resulting bit-stream can be lower than the zeroth-order entropy. The usual procedure utilized in image coding, and also in JPEG2000, is that the probability of each sample is estimated according to the neighboring samples or the sample's context [34] and during the coding process. This process will be discussed in the next section.

The term entropy coding refers to the techniques used to losslessly encode the symbols generated by a source. As previously mentioned, the lower bound or the minimum achievable average rate for each memoryless source, is the entropy of the source. Several entropy coding techniques are available in the data coding literature. The main idea behind all of the techniques is to code the more popular symbols in a sequence with shorter codes and vice versa. This manipulation will decrease the length of the codeword needed to represent the sequence in question. The entropy coding technique used in the JPEG2000 standard is called arithmetic coding and it will be discussed here in details.

### 2.1.1 Arithmetic coding

In arithmetic coding [19][34][27], every sequence of symbols is uniquely assigned a tag, which is a number in the interval [0,1). Infinite number of numbers exist in this unit interval, thus it is possible to generate a distinct tag for every possible sequence.

The usual procedure is to use the cumulative distribution function (cdf), in order to map a sequence into the unit interval. Here the coding procedure is explained through an example. Suppose a source randomly generates sequences comprising the three symbols $s_1$, $s_2$, and $s_3$, with probabilities equal to 0.6, 0.3 and 0.1 respectively. Now consider we want to code the sequence $s_3\ s_1\ s_2$. Table 2.1 can be drawn for this example.

**Table 2.1. Probabilities and intervals associated to three symbols of a source**

| Symbol | Probability | Interval |
|---|---|---|
| $s_1$ | 0.6 | [0,0.6) |
| $s_2$ | 0.3 | [0.6,0.9) |
| $s_3$ | 0.1 | [0.9,1) |

The tag lies in one of the above intervals depending upon the first symbol in the sequence, which is $s_3$ in our example. Therefore, the tag resides in the interval [0.9,1), as shown in the Figure 2.2. Now, the interval [0.9,1) is divided into subintervals exactly in the same manner as the unit interval, and the rest of the unit interval is discarded. The resulting subintervals will be [0.9,0.96), [0.96,0.99) and [0.99,1). Now since the second symbol in the sequence is $s_1$, this time the tag resides in the first subinterval, [0.9,0.96), and the other subintervals are discarded. This procedure is performed once again, as there is still another symbol in the sequence. This time the resulting subintervals will be [0.9,0.936), [0.936,0.954), [0.954,0.96), and the first and third subintervals will be discarded since the third symbol is $s_2$. The tag is therefore restricted to the subinterval [0.936,0.954). Any number in this interval, say the middle point 0.945, combined with the probabilities of the source symbols uniquely represent the sequence.

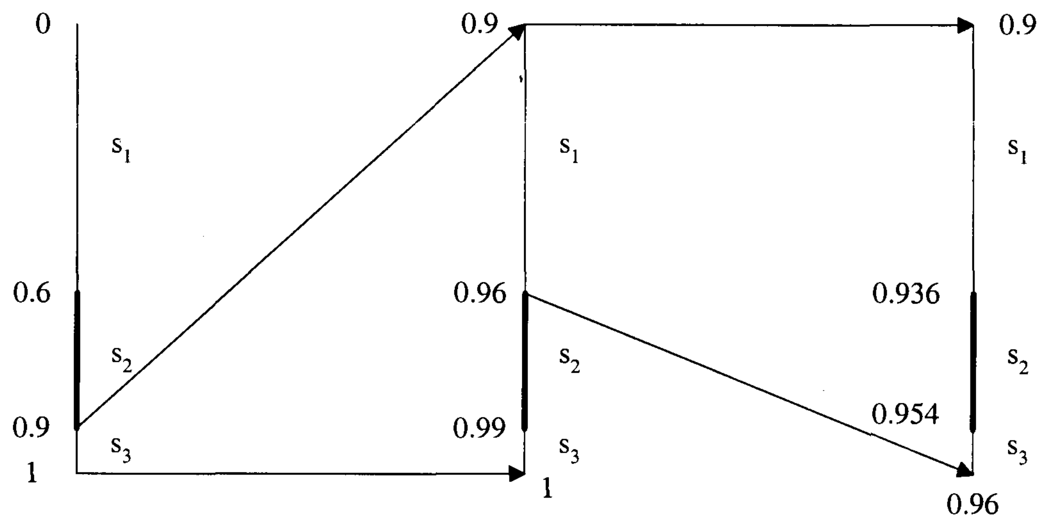


**Figure 2.2. Formation of tag in coding the sequence $s_3s_1s_2$**

In order to decode the tag, we simply reverse the above procedure. First, among the intervals [0,0.6), [0.6,0.9) and [0.9,1), we find the interval that contains the tag. Since the third interval [0.9,1) contains the tag, the first symbol is decoded as $s_3$. Now further division of this interval results in the three subintervals [0.9,0.96), [0.96,0.99), [0.99,1). This time the tag lies in the first subinterval, hence the second symbol decoded as $s_1$. This time, the first subinterval is divided into three subintervals [0.9,0.936), [0.936,0.954), [0.954,1), and as the tag belongs to the second subinterval, the third symbol will be $s_2$.

Theoretically, arithmetic coding is performed based on the above procedure, however, for most practical purposes, the tag should be efficiently represented by a binary code. The binary code can be the binary representation of the tag, truncated in a way so that it still represents the tag uniquely. An algorithm [27] will be described here that incrementally computes a unique binary representation of the tag during the coding process.

A key part of this algorithm is the rescaling strategy. When arithmetic coding is used to code long sequences of symbols, the tag interval shrinks to very small intervals. Representing this small interval requires higher precision as more number of symbols are coded. In order to avoid this problem, we rescale the interval during the coding process. If the interval is entirely in the lower half of the unit interval, as it will definitely stay there during the coding process, a 0 is sent to the decoder and the resulting interval is rescaled as follow,

$$E(x) = 2x \tag{2.2}$$

If the interval lies completely in the upper half of the unit interval, the binary bit 1 is sent and the interval is rescaled as,

$$E(x) = 2(x - 0.5) \tag{2.3}$$

We can construct Table 2.2 for the above example.

**Table 2.2. Arithmetic coding the sequence $s_3s_1s_2$**

| symbol | Operation | interval | Coded bit |
|---|---|---|---|
| $s_3$ | choosing the third interval | [0.9,1) | 1 |
|  | rescaling | [0.8,1) | 1 |
|  | rescaling | [0.6,1) | 1 |
|  | rescaling | [0.2,1) | - |
| $s_1$ | choosing the first subinterval | [0.2,0.68) | - |
| $s_2$ | choosing second subinterval | [0.488,0.632) | 1 |

The final coding step is termination in which an arbitrary value inside the tag interval is added to the coded bitstream. In our example a straight forward choice is 0.5 as its binary equivalent is 0.1, which can be represented by one bit. Thus a 1 is coded in the last step. The final coded bitstream is therefore 0.1111 which is equal to 0.9375 and lies in the subinterval [0.936,0.954) computed in Figure 2.2.

In order to decode the bitstream, we should select a window of length, *n*, and start decoding with the first *n* bits of the coded bitstream. At each step the window is moved through the bitstream, the MSB bit is shifted out and a new bit is shifted in from the LSB side. The size of the window should be long enough to be able to represent the smallest tag interval belonging to the symbol with the lowest probability, in this case 0.1. Thus we need to use a window length four. We start decoding by reading the first four symbols 1111, which corresponds to a value of 0.9375. This value lies in the interval [0.9,1) which shows that the first symbol of the sequence is $s_3$. The interval [0.9,1) resides in the upper half of the unit interval, so we shift the window one bit to the left. Using (2.2), we need to map the tag interval to [0.8,1). Now we are dealing with the code 1110, which corresponds to a tag value of 0.875. We repeat the last step twice more, as the interval is still in the upper half of the unit interval. The resulting intervals will be [0.6,1) and [0.2,1) and the tag value 1100 and 1000. Now the tag value is 0.5 that lies in the first 60% of the interval [0.2,1). So the second symbol is decoded as $s_1$, and the updated interval is [0.2,0.68) which is the first 60% portion of the [0.2,1). The tag, 0.5, is now between the 60%-90% portion of the interval [0.2,0.68), which implies that the third and final symbol should be $s_2$. The information about the number of the encoded symbols need be provided to the decoder so that the decoding procedure terminates at the proper point. Here after decoding the third element, the decoder should stop decoding. The decoding procedure is summarized in Table 2.3.

**Table 2.3. Decoding the sequence $s_3s_1s_2$**

| interval | Windowed bitstream | tag | decoded symbol |
|---|---|---|---|
| [0.9,1) | 1111 | 0.9375 | $s_3$ |
| [0.8,1) | 1110 | 0.875 | - |
| [0.6,1) | 1100 | 0.75 | - |
| [0.2,1) | 1000 | 0.5 | - |
| [0.2,0.68) | 1000 | 0.5 | $s_1$ |
| [0.488,0.632) | 1000 | 0.5 | $s_2$ |

## 2.2 Quantization

Quantization [10] is the act of mapping a large set of different values to a smaller set, which is one of the basic ideas of lossy data compression. Figure 2.3 is an example of a scalar quantizer, where all the real values in the $x$ axis are mapped into only six values in the $y$ axis. In this example the values that reside in the range $[0, \Delta)$ are mapped into $\Delta/2$, etc.

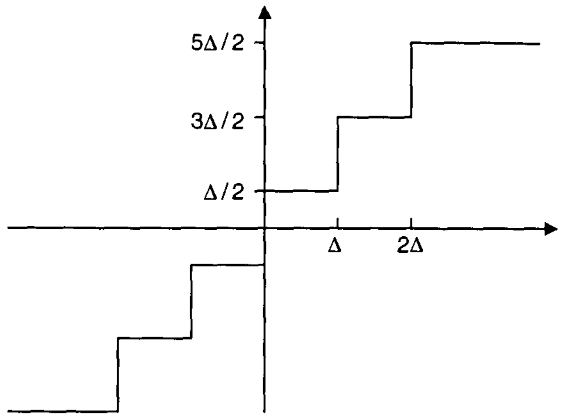


**Figure 2.3. Example of a uniform scalar quantizer**

Usually, each of the values on the *y* axis is assigned a quantization index, and the indexes are entropy coded at the coder side. At the decoder, first the indexes are entropy decoded, and then since the exact values of the coded samples are not known, each index is mapped into a reconstruction value that in some sense, optimally represents the samples in that interval. This mapping of index to value is usually predetermined and the encoder uses this to decide on the quantization index. Also since the exact value of the sample cannot be recovered at the decoder, the resulting compression will be lossy.

The design parameters in every scalar quantizer include the sizes of each interval in both of the *x* and *y* axis, the number of the quantization levels and the reconstruction values. The design, in turn, depends upon the statistics of the source samples, and the conditions and constraints that exist in each practical problem. In the end we mention that the quantizer utilized in the JPEG2000 standard is a scalar quantizer with deadzone [34], which will be explained in the next chapter.

## 2.3 Transform coding

In a typical image, the values of neighboring samples are highly correlated. Appropriate transforms can be applied to an image in order to reduce or ideally remove the correlation among the samples and come up with coefficients with higher energy compaction. The resulting signal consists of a large group of coefficients with a small variance, and only a small group of coefficients with large variance. Most of the energy of the original image lies in the group with large variance. A larger portion of the bit budget can be allocated to this group, and the group with small variance can be coded with a lower budget. This operation results in compression [5].

The concept of a useful transform for image compression can be better understood through a simple example [34]. In this example, image samples are partitioned into blocks of $2 \times 2$. The four samples in each block are represented with the average of the four samples in the block, and the differences between three of the samples with the average value. As shown in Figure 2.4, the four transformed samples consist of the average of the four image samples, 27, and the differences of the three of the image samples 28, 23 and 32 from the average value which will be 1, -4 and 5 respectively. One of the transformed coefficients has a large value, while the other three have small values close to zero.

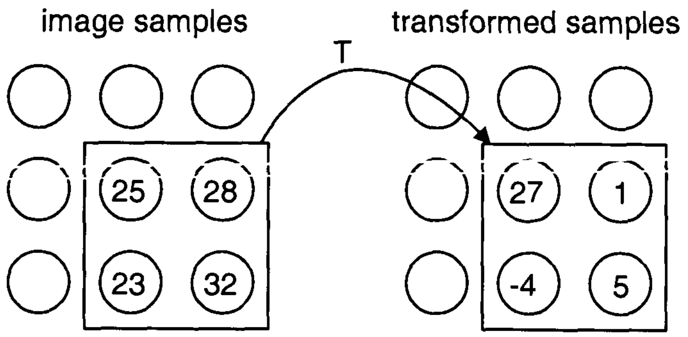


**Figure 2.4. A simple image transform example**

Mathematically, the transform for each block can be represented through the following equation,

$$y(m,n) = \begin{cases} 0.25 \sum_{i,j \in \{0,1\}} x(m+i, n+j) & \text{if } m = n = 0 \\ x(m,n) - y(0,0) & \text{otherwise} \end{cases} \tag{2.4}$$

and the inverse transform through,

$$\hat{x}(m,n) = \begin{cases} y(0,0) + y(m,n) & \text{if } m \vee n \neq 0 \\ 4y(0,0) - \hat{x}(0,1) - \hat{x}(1,0) - \hat{x}(1,1) & \text{otherwise} \end{cases} \quad (2.5)$$

When this transform is applied to all of the blocks, the resulting average values, which consist of 25% of the total coefficients, will have a large variance while the other 75% of the samples, the difference values, will have small values close to zero. Note that even if we discard the three difference values, we have compressed the image with almost a degree of four, and the reconstructed image samples will all be 27, which is relatively close to the original values. Note that in this case the compression type will be lossy.

Most of the transforms that are used in image compression are linear transforms. A 1-D linear transform is a transform that maps a vector of dimension $n$ to a another vector of dimension $m$ according to the following equation,

$$\mathbf{y} = \mathbf{\bar{A}x} \quad (2.6)$$

where $A$ is an $n \times m$ matrix. As we will discuss in the next chapter, most of the transforms used for image compression are non-expensive transforms, that is the dimensions of the input and output vectors are equal, or the transform will not produce more number of coefficients than the image samples. In this case, the inverse of the transform matrix is unique [34] and is equal to

$$\mathbf{B} = \mathbf{A}^{-1} \quad (2.7)$$

and the inverse transform can be written as,

$$\mathbf{x} = \mathbf{By} \quad (2.8)$$

Note that the $p$th component of the output vector can be computed as,

$$y_p = \sum_{i=0}^{n-1} x_i a_{p,i} \tag{2.9}$$

and the $q$th component of the input vector is related to the output through,

$$x_q = \sum_{i=0}^{n-1} y_i b_{q,i} \tag{2.10}$$

where $a_{p,i}$ and $b_{q,i}$ are the elements of the **A** and **B** matrices.

The Discreet Fourier Transform (DFT) [20] is an example of a linear and non-expensive transform. In an $n$ point DFT the transform matrix can be defined as,

$$A = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ e^{-2\pi/n} & e^{-2\times 2\pi/n} & \cdots & e^{-(n-1)\times 2\pi/n} \\ e^{-2\times 2\pi/n} & e^{-4\times 2\pi/n} & \cdots & e^{-2(n-1)\times 2\pi/n} \\ \vdots & \vdots & \ddots & \vdots \\ e^{-(n-1)\times 2\pi/n} & e^{-2(n-1)\times 2\pi/n} & \cdots & e^{-(n-1)(n-1)\times 2\pi/n} \end{bmatrix} \tag{2.11}$$

and the inverse transform matrix is,

$$B = A^{-1} = \frac{1}{n}\begin{bmatrix} 1 & 1 & \cdots & 1 \\ e^{2\pi/n} & e^{2\times 2\pi/n} & \cdots & e^{(n-1)\times 2\pi/n} \\ e^{2\times 2\pi/n} & e^{4\times 2\pi/n} & \cdots & e^{2(n-1)\times 2\pi/n} \\ \vdots & \vdots & \ddots & \vdots \\ e^{(n-1)\times 2\pi/n} & e^{2(n-1)\times 2\pi/n} & \cdots & e^{(n-1)(n-1)\times 2\pi/n} \end{bmatrix} \tag{2.12}$$

A transform can be better understood by looking at the inverse matrix. Any input signal is decomposed as a linear combination of the rows of the matrix or the synthesis vectors. In the case of the DFT, the input is represented as a linear combination of complex exponential waveforms with frequencies equal to the multiples of $2\pi / n$. This transform is one of the important transforms in the DSP area, however, it is not common in image compression applications. The reason, as we will discuss in the next chapter, is that in DFT the input signal is extended periodically and this kind of extension might result in abrupt discontinuities in the image boundaries. In the Discrete Cosine Transform, DCT [27], the input sequence is extended symmetrically, which is more appropriate for image compression applications. As a result, the DCT is widely used in the image compression area, for example in the JPEG standard [21]. Furthermore, there is a fast algorithm to compute the DFT (and the DCT) which has helped the popularity of the transforms.

The transforms utilized for image compression can be divided into two categories. In the first type, the block transform, the image is partitioned into different blocks and the transform is applied to each block separately. As an example, in the JPEG standard the image is partitioned into blocks of 8x8 samples, and the DCT is applied to each block. The advantage of the block transforms is that the amount of memory needed for implementing the transform is low. On the downside however, adjacent correlated image samples might lie in different blocks and as a result the transformed coefficients from adjacent blocks might still be highly correlated. Also partitioning an image into small blocks and treating each block independently causes artifacts at the block edges.

The second transform type is the subband transform. As we mentioned earlier, signals with non-overlapping frequency responses are not correlated. In subband transforms [34][27], the whole image is transformed into different bands of frequency and the process results in decorrelation. Note that in practice, as the filters are of finite dimensions the subbands do overlap, so the utilized transforms should have orthogonal basis functions in order to remove the correlation.

The procedure in a subband transform is that the image signal is filtered into components with separate bands of frequency. The coefficients belonging to each band are down-sampled and then grouped together into so called subbands. Down-sampling results in aliasing, which will be cancelled during the reconstruction process (in the absence of quantization). The advantage of down-sampling is that it avoids an increase in the total number of resulting coefficients in all of the subbands, comparing to the number of image samples. In the case of lossy compression, the coefficients are quantized and then entropy coded in order to achieve a higher compression, while in the lossless compression, the quantization part is omitted.

Although digital images are represented by two-dimensional matrices, usually separable 1-D transforms are applied to the image samples, first on each row and then on each column or vice versa, and in order to reduce the complexity. In JPEG2000 the frequency decomposition is performed using wavelet transform, as discussed below.

### 2.3.1 Wavelet transform

Wavelet means “small wave”. The advantage of wavelets is that they have their energy concentrated in time, and this property makes them a good choice for the analysis of non-stationary and time-varying signals [16]. Maybe the most important property of the wavelet transform [4][32][39] in the image compression applications, as we will see later, is the multi-resolution property where a signal is represented by a coarse approximate version followed by a set of details.

In the multi-resolution analysis in the 1-D transform, the scaling function, $\varphi(t)$, and the mother wavelet, $\psi(t)$ correspond to the low- and high-pass synthesis filters respectively. We will talk about the relationship between these two functions later in this section. The wavelet basis functions are obtained by just shifting and scaling the mother wavelet function as follows [27],

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}.\psi(\frac{t-b}{a}) \qquad \textbf{(2.13)}$$

where *a* and *b* are the scaling and shifting parameters. A popular choice for these two parameters is,

$$a = 2^{-m},\ b = n.2^{-m} \qquad \textbf{(2.14)}$$

where *m* and *n* are integers. Substituting these values, the wavelet basis functions can be expressed as,

$$\psi_{m,n}(t) = 2^{m/2}\psi(2^m t - n) \qquad \textbf{(2.15)}$$

The one-dimensional scaling function can also be obtained from the scaling function in a similar way and through the following equation,

$$\varphi_{m,n}(t) = 2^{m/2}\varphi(2^m t - n) \qquad \textbf{(2.16)}$$

As an example the Haar wavelet, which is a very simple wavelet, is defined by,

$$\psi(t) = \begin{cases} 1 & 0 \le t < 0.5 \\ -1 & 0.5 \le t < 1 \\ 0 & otherwise \end{cases} \qquad \textbf{(2.17)}$$

and the Haar scaling function as,

$$\varphi(t) = \begin{cases} 1 & 0 \le t < 1 \\ 0 & \text{, otherwise} \end{cases} \tag{2.18}$$

The idea behind the multi-resolution property of wavelets can be explained as follow. The scaling function has the property that if a function can be represented by the shifted versions of the scaling function at one resolution, it can also be represented by the shifted versions of the scaling function at a higher resolution. That is, if we can find $a_n$'s so that,

$$f(t) = \sum_n a_n \varphi_{m_0,n}(t) \tag{2.19}$$

then we can also find $b_n$'s such that,

$$f(t) = \sum_n b_n \varphi_{m_0+1,n}(t) \tag{2.20}$$

In other words, if $\gamma_{m_0} = \overline{\underset{n}{Span}\{\varphi_{m_0,n}\}}$ and $\gamma_{m_0+1} = \overline{\underset{n}{Span}\{\varphi_{m_0+1,n}\}}$ then $\gamma_{m_0} \subset \gamma_{m_0+1}$. This property can be generalized to all scales of the scaling function, and thus by considering the higher resolutions of the scaling function, a wider range of signals are spanned. In order to widen the range of the spanned signals, instead of just using higher resolutions of the scaling function, a better way is to use also a different set of functions, $\psi_{m,n}(t)$, that spans the differences between the spaces spanned by the scaling function [4]. These functions are the wavelet functions and they are orthogonal to the scaling functions. The orthogonal property of a transform, as we will see in the next chapter, is very important in image compression applications, as it results in the Parseval's property that results in an easy computation of the signal's energy in the transform domain.

The orthogonal complement of $\gamma_m$ in $\gamma_{m+1}$ is defined as $w_m$ [4]. If we start from $m = 0$, then we can write,

$$\gamma_1 = \gamma_0 \oplus w_0 \tag{2.21}$$

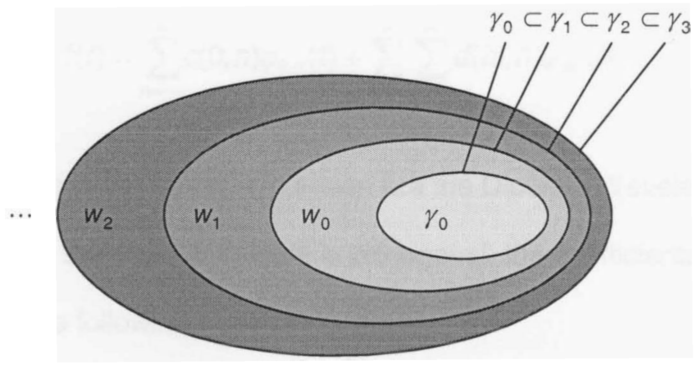
where $\oplus$ is the summation of two spaces. Therefore, we can further write the above equation as,

$$\gamma_2 = \gamma_0 \oplus w_0 \oplus w_1 \tag{2.22}$$

and

$$\gamma_m = \gamma_0 \oplus w_0 \oplus w_1 \cdots \oplus w_{m-1} \tag{2.23}$$

This property is illustrated in Figure 2.5.



$\gamma_0 \subset \gamma_1 \subset \gamma_2 \subset \gamma_3$
...
$w_2$
$w_1$
$w_0$
$\gamma_0$

**Figure 2.5. The relationship of the spanned spaces (taken from [4])**

Since $\gamma_0 \subset \gamma_1$, any function belonging to $\gamma_0$ also belongs to $\gamma_1$. A direct result of this statement is that $\varphi(t)$ which belongs to $\gamma_0$ can be expressed as a function of the shifted versions of the $\varphi(2t)$,

$$\varphi(t) = \sum_j h(j)\sqrt{2}\varphi(2t - j) \tag{2.24}$$

Also as $w_0 \subset \gamma_1$, again $\psi(t)$ can be represented by a weighted sum of the shifted versions of $\varphi(2t)$,

$$\psi(t) = \sum_j h_1(j)\sqrt{2}\varphi(2t - j) \tag{2.25}$$

With a change of variable, it is easy to verify that [4],

$$\varphi(2^m t - n) = \sum_j h(j - 2n)\sqrt{2}\varphi(2^{m+1} t - j) \tag{2.26}$$

Based on the above arguments, any signal, $f(t)$, belonging to the spanning set of a wavelet function, $\psi(t)$, and the corresponding scaling function, $\varphi(t)$, can be expanded as follow,

$$f(t) = \sum_{n=-\infty}^{\infty} c(0,n)\varphi_{0,n}(t) + \sum_{m=0}^{\infty}\sum_{n=-\infty}^{\infty} d(m,n)\psi_{m,n}(t) \tag{2.27}$$

where the coefficients in the above expansion are the Discrete Wavelet Transform (DWT) coefficients. If the wavelet system is orthogonal, the coefficients can be calculated through the following two inner products,

$$\begin{aligned} c(0,n) &= \langle f(t), \varphi_{0,n}(t) \rangle = \int f(t)\varphi_{0,n}(t)dt \\ d(m,n) &= \langle f(t), \psi_{m,n}(t) \rangle = \int f(t)\psi_{m,n}(t)dt \end{aligned} \tag{2.28}$$

Since $\gamma_{m+1} = \gamma_m \oplus w_m$, then if $f(t) \in \gamma_{m+1}$, we can write,

$$f(t)=\sum_j c(m+1,j)2^{(m+1)/2}\varphi(2^{m+1}t-j) \tag{2.29}$$

and since also $f(t)\in \gamma_m \oplus w_m$ then,

$$f(t)=\sum_j c(m,j)2^{m/2}\varphi(2^m t-j)+\sum_j d(m,j)2^{m/2}\psi(2^m t-j) \tag{2.30}$$

where $c(m,n)$ can be computed with the following inner product,

$$c(m,n)=\langle f(t),\varphi_{m,n}(t)\rangle = \int f(t)2^{m/2}\varphi(2^m t-n)dt \tag{2.31}$$

We can easily find the relationship between the coefficients in equations (2.29) and (2.30). Using equation (2.26), we can write,

$$c(m,n)=\sum_j h(j-2n)\int f(t)2^{(m+1)/2}\varphi(2^{m+1}t-j)dt \tag{2.32}$$

Comparing equations (2.31) and (2.32) we will have,

$$c(m,n)=\sum_j h(j-2n)c(m+1,j) \tag{2.33}$$

And similarly,

$$d(m,n)=\sum_j h_1(j-2n)c(m+1,j) \tag{2.34}$$

Therefore, if we know the wavelet transform coefficients in one resolution, we can easily compute the coefficients of a lower wavelet resolution.

If we compare equations (2.33) and (2.34) with (2.35)

$$y(n) = \sum_j h(j)x(n-j) \quad (2.35)$$

we can see that in order to compute the coefficients in a lower resolution, the coefficients in the current resolution are filtered by $h(-n)$ or $h_1(-n)$, and then down-sampled by a factor of two as illustrated in Figure 2.6. It can be shown [4] that the filters represented by $h(-n)$ and $h_1(-n)$ are FIR low- and high-pass filters respectively.

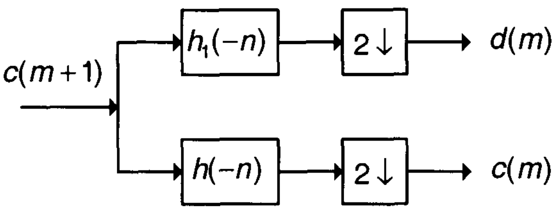


**Figure 2.6. Analysis filter bank**

Therefore, if we know the input set of coefficients, $c(m)$, for a signal in any resolution, we can easily compute the low- and high-passed filtered versions of those coefficients. Also the above analysis can be further continued, and the resulting low frequency coefficients can be further filtered into low and high frequency bands as shown in Figure 2.7.

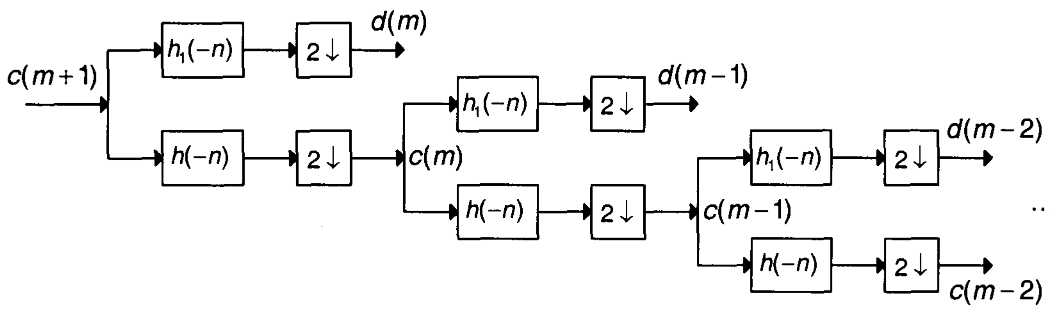


**Figure 2.7. A three level decomposition**

The frequency bands of the resulting signals are illustrated in Figure 2.7. Note that the transition between each two set of coefficients is not so sharp, as the filters $h(-n)$ and $h_1(-n)$ are not ideal filters. So each signal has some energy in its neighboring bands too, however, as the basis vectors in a wavelet transform are orthogonal, the transform removes the correlation among the input samples.

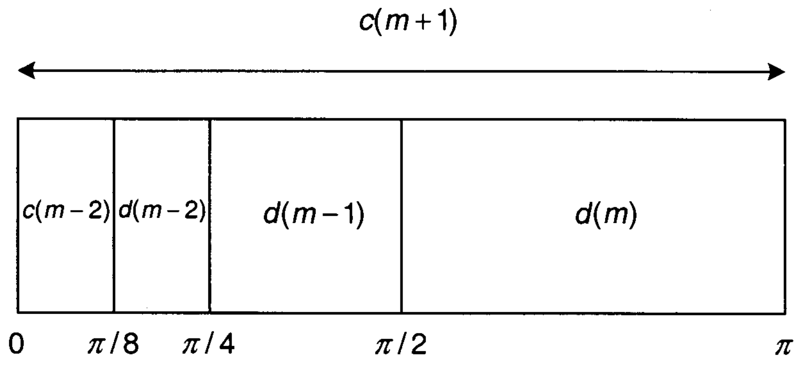


**Figure 2.8. Frequency ranges of the coefficients in a three level decomposition**

The number of input coefficients is equal to that of the output at each level of decomposition because of down-sampling. This property is particularly useful for data compression applications. However, down-sampling results in aliasing and might result in loss of information. Fortunately, there exist methods [38] to undo the aliasing resulting from the low-pass filter in the high-pass filter and vice versa, and thus, by applying the inverse transform, we can have a perfect reconstruction of the original signal (in the absence of quantization). At the end, note that in order to apply a 1-D transform to a 2-D image signal, the transform is first applied to the rows and then to the columns of the image or vice-versa.

In this chapter we briefly introduced the main building blocks of a wide range of image compression algorithms. We will talk about the compression method used in JPEG2000 and the region of interest feature of the standard.

# CHAPTER 3
# AN OVERVIEW OF THE JPEG2000 STANDARD

## *Introduction*

This chapter provides a brief overview of the JPEG2000 [33][34][35][13][14] standard, which is the latest image compression standard at the time of this writing. The standard is mainly based on an algorithm called independent Embedded Block Coding with Optimized Truncation of the embedded bit-streams (EBCOT) [35], introduced by D. Taubman. The acronym stands for “Joint Photographic Experts Group”.

Previous commercial image compression standards are generally used as an input/output filter [28]. Decisions on the quality, compression ratio and resolution (image dimensions) are made at the encoder side, and at the decoder, only a single image with a particular quality, compression ratio and resolution is recovered. There are exceptions to the above argument. As an example, the JPEG standard has a progressive mode in which the image can be decoded with different qualities, also the lossless mode of the JPEG standard exist, however, these modes have different structures and work with different algorithms. When an image is coded in each of the modes, the resulting bit-stream has only one particular property and cannot be decoded by another mode.

Unlike these previous standards, JPEG2000 works more like an image processing system and it allows for both increased flexibility and performance. A single coded bitstream can be used to reconstruct many different versions of the coded image with different dimensions, qualities and colors, and the decisions on the characteristics of the image can be delayed until the decoding time.

## *3.1 JPEG2000 Features*

**Compression performance**

For lossy compression and in bit-rates lower than 0.25 bpp and higher than 1.5 bpp, JPEG2000 performs significantly better than JPEG, and in moderate bit-rates from 0.5 to 1.0 bpp, JPEG2000 compression performance is usually a few dB better in peak signal to noise ratio (PSNR) than JPEG, however,

In the lossless case, JPEG2000 performs much better than the lossless mode of JPEG, but somewhat worse than JPEG-LS [15], which is a new image compression standard for only lossless and nearly lossless compressions. A quantitative comparison for the JPEG2000 standard and different modes of the JPEG standard and JPEG-LS is provided in section 3.2.6

**Compress once, decompress many ways**

At the time of compression, only a few coding parameters are chosen such as the maximum image quality and the maximum resolution. Any image quality, size and resolution up to these maximums can be reconstructed from a single coded bit-stream.

As an example, suppose an image is coded losslessly and with full resolution, resulting in a 100kb bit-stream. A lossy image can be constructed by locating and decoding, say, 10% of the coded bit-stream. Also by locating and decoding 15% of the coded bits, it's possible to reconstruct a smaller resolution version of the image. Finally this random access feature is also available for color components as well. Note that in each of the above cases, it is not necessary to decode the whole bit-stream. Also accessing arbitrary parts of an image is possible through the encoded bit-stream, and it is only a matter of locating and decoding the proper packets of the coded bit-stream.

## Progression

JPEG2000 supports four types of progression: quality, resolution, spatial location and component, all in one single bit-stream [28].

- ***Quality***

The first kind of progression is in quality. As more data is decoded, the image quality is enhanced up to the maximum that was initially chosen at the time of coding. The maximum quality can be chosen as lossless. The progression property is most useful when data is transmitted through a low capacity channel.

- ***Resolution***

The second type of progression is in resolution or image dimensions. By decoding the first few bytes, a thumbnail size of the picture is constructed, as the decoding process continues, the dimensions of the resulting image are doubled until the full size image is obtained. The number of these smaller size images is always one more than the number of the levels of wavelet transform performed at the time of compression. Note that the only available resolutions are $(1/2)^k$ times the original image dimensions, where $k$ is an integer in the range $[0, m)$ and $m$ is the number of the stages of the wavelet decomposition performed during the compression process.

- ***Spatial Location***

The third type of progression supported is in spatial location. In this type, the image can be decoded in, say, a stripe-based fashion starting from top of the image. This property has applications in low-memory printers and scanners.

- ***Component***

The final type of progression is in imàge components. Most digital images are represented by either one, three or four components, like grayscale, RGB and CMYK images respectively. Images with more than four components are usually from scientific instruments. JPEG2000 supports up to 16384 components. Data representing each component can be extracted and decoded separately from the coded bitstream. As an example, an image can be decoded to its grayscale version followed by its color components.

### Region-of-Interest (ROI) coding

Another feature of the standard is the possibility to code some specific parts of an image with higher quality than the other parts or the background, at the compression time. For images that are coded with a region of interest, the ROI will be decoded before the background in the progressive decoding process. Thus, if the decoding process is terminated at any point, the ROI will have a better quality than the background.

The relative quality between the ROI and the background, and choosing the region of interest are the parameters that can be picked by the user at the time of compression. This feature is useful for storage purposes as well as for data transmission over low-capacity channels. This property is the subject of this thesis and will be discussed in detail later in this chapter.

## *3.2 The JPEG2000 algorithm*

The JPEG2000 standard only specifies the decoder's algorithm; however, for a better understanding of how the algorithm works, we will go through the coding procedure in this section.

### 3.2.1 Pre-processing

#### *Tiles*

The first step in the coding procedure is to divide the image samples into non-overlapping rectangular tiles [34]. The samples from each component that lie in a tile are grouped as a tile-component. The main advantage of tiling is to reduce the amount of memory needed for implementing the encoder, as each of the tile-components are treated independently. The disadvantage of tiling is the boundary breaks that are the visually visible artifacts that appear on the boundaries of the tiles. These artifacts result from compressing each tile independently when the qualities of the adjacent samples reconstructed in two different tiles are not exactly the same. The effects of these breaks are alleviated in high bit-rates (high qualities) and large tiles. In situations where memory is not a big concern and the image dimensions are not too large, the tile size can be chosen to be equal to the image dimensions.

#### *Components*

Unlike grayscale images, color images are represented with multiple components. Each component is represented by a separate $n_1 \times n_2$ matrix where $n_1$ and $n_2$ are the dimensions of the image. As an example, in a very well-known type of image, RGB, the three components that represent the image, correspond to the intensity of the colors red, green and blue respectively. In JPEG2000, a component transform is applied to the first three image components in order to remove the correlation between them and hence to improve the compression efficiency. After that, each transformed component is coded independently.

Two types of transforms are supported by the JPEG2000. One is the RGB to YCbCr transform, where Y is the luminance component and can be decoded as the grayscale version of the image, and Cb and Cr are the blue and red color difference components. This transform is used for lossy compression applications, as it is an irreversible transform and is called the irreversible component transform (ICT). The forward and inverse transforms can be written as (3.1) and (3.2) respectively [30]. Note that the transform matrix has an inverse; however, it cannot be practically implemented as its elements are generally of infinite precision.

$$\begin{bmatrix} Y \\ C_r \\ C_b \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ -0.16875 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{bmatrix} . \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.1}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34413 & -0.71414 \\ 1 & 1.772 & 0 \end{bmatrix} . \begin{bmatrix} Y \\ C_r \\ C_b \end{bmatrix} \tag{3.2}$$

The other transform is the reversible color transform (RCT) [11], which has similar properties to the YCbCr transform. The only important difference between the two transforms is that the RCT results in components with integer values, making it appropriate for lossless compression. The amount of improvement in the compression efficiency is different for the two types of transforms and for different images. Table 3.1 provides the compression performances for coding a test image with and without performing the component transform. The ICT and the RCT are employed for the lossy and lossless cases respectively.

**Table 3.1. The effect of component transform on compression performance of the ski image (taken from [30])**

| | Without component | With component |
|---|---|---|
| **Lossless compression** | 16.88 b/p | 14.78 b/p |
| **Lossy compression at** | 25.67 dB | 26.49 dB |

Hereafter, we will focus on coding a tile-component, as the tile-components in an image are literally coded independently and separately, and the bit-streams are put together only after each of them is completely coded. From this section, when we mention "tile", we actually mean "tile-component".

### 3.2.2 Wavelet Transform

The wavelet transform is utilized to decorrelate the image samples. One other advantage of the wavelet transform is that it results in energy compaction among the image samples. It can be shown that signals with non-overlapping frequency components are uncorrelated [36]. Therefore, the approach in JPEG2000 is to decompose the image samples into subbands with approximately separate bands of frequency. Although images are represented by two dimensional signals, the usual approach in the image compression literature, and also JPEG2000 is to implement a 2-D transform by applying a separable 1-D transform to the rows and then to the columns of the image [27]. Thus, we will only discuss 1-D wavelet transforms in this section.

The general form of a 1-D wavelet transform is illustrated in Figure 3.1. In each transform level, the input is decomposed into two signals with equal frequency bands using a low-pass and a high-pass filter, and then each signal is decimated by a factor of two in order to maintain the critical sampling and to avoid an increase in the number of resulting coefficients.

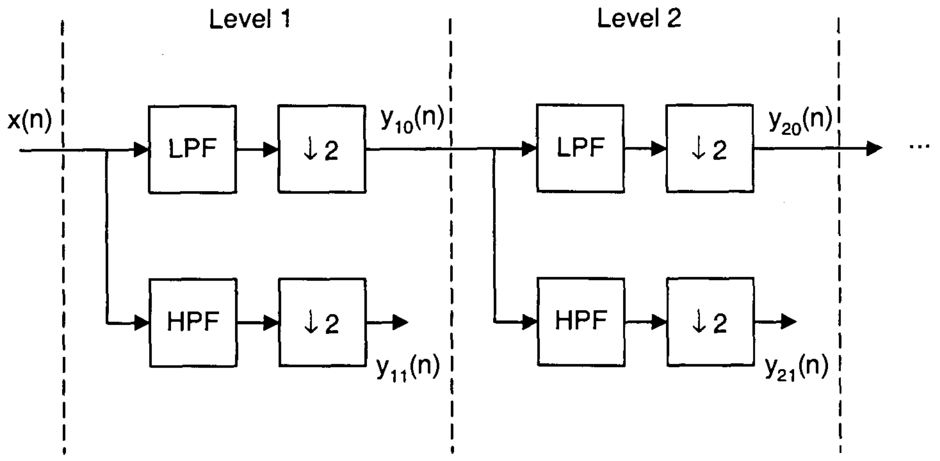
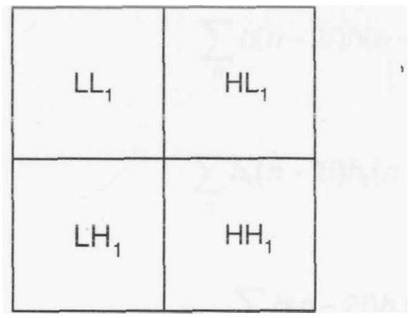


**Figure 3.1. Illustration of a multi-level 1-D wavelet decomposition**

Applying a one level decomposition to the 2-D image signal, results in four subbands: LL, LH, HL and HH. The first letter represents the frequency band of the subband row and the second letter represents the frequency of the subband column. As an example, the HL subband contains the high-pass filtered samples of the image rows and the low-pass filtered samples of the image columns. The decomposition can be further continued through decomposing the $LL_1$ subband as shown in Figure 3.2. Note that for a third level decomposition the $LL_2$ subband should be decomposed.



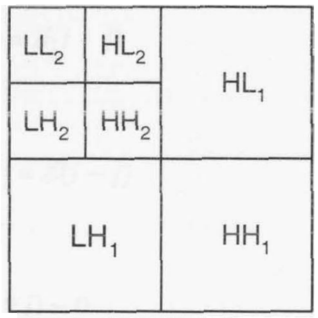


**Figure 3.2. A one-level (left) and a two-level (right) 2-D wavelet transform of the image signal**

In Figure 3.3, a two level 2-D wavelet transform is applied to the Lena image.

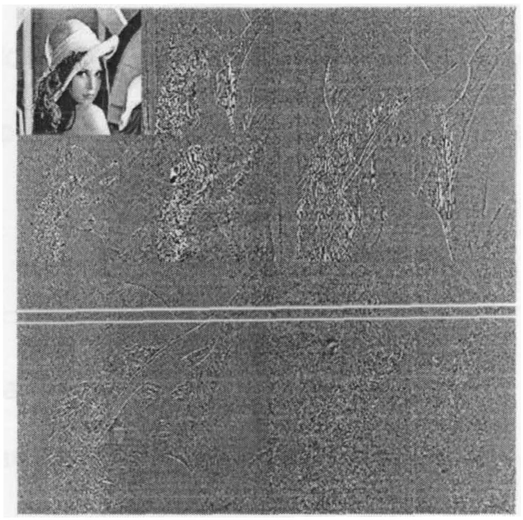


**Figure 3.3. A two-level wavelet transform of the Lena image**

The original wavelet transforms were implemented using orthogonal wavelets [37]. Orthogonal transforms project the input signal to orthogonal basis vectors. Using orthogonal transforms in image compression applications is important as it results in decorrelation between the transformed samples despite the overlap in the frequency bands of different subbands. The orthogonal wavelets satisfy the following conditions:

$$\sum_n h(n-2i)h(n-2j) = \delta(i-j) \tag{3.3}$$

$$\sum_n h_1(n-2i)h_1(n-2j) = \delta(i-j) \tag{3.4}$$

$$\sum_n h(n-2i)h_1(n-2j) = 0 \tag{3.5}$$

where $h(n)$ and $h_1(n)$ are the impulse responses of the low-pass and high-pass filter respectively. Note that the above wavelet filters are also normalized, which yield a transform that is energy preserving. Mathematically speaking, for a signal $x(n)$ of length $N$, and its wavelet transform $w(n)$ of length $L$, this property, which is the equivalent version of the Parseval property [20] in the Fourier transform, can be written as,

$$\sum_{n=0}^{N-1} x^2(n) = \sum_{l=0}^{L-1} w^2(l) \tag{3.6}$$

The energy preserving property is important because for such transforms, the distortion caused by quantizing the transformed coefficients is equal to the distortion of the reconstructed coefficients after the inverse transform. As a result, the JPEG2000 bit allocation can be done in the transform domain, a matter of great practical importance.

The filters used in the image compression literature are somewhat different from filters used in other areas of signal processing. Actually, the image compression filters with the expansion method utilized in them are used exclusively in this area [31]. In the practical filters used in other signal processing areas, the number of the coefficients of the output is generally larger than the number of the coefficients of the input signal. The amount of the increase in the number of the coefficients at the output depends on the number of the filter taps. The reason comes from the fact that convolving two signals with lengths $n_1$ and $n_2$, results in a signal with length $n_1 + n_2 - 1$, in a linear convolution. This expansion is not a problem for most other applications, like speech signals, where the signal length is practically infinite compared to the filter taps; but this is not the case for image signals, which typically have modest dimensions.

One solution to this problem is to design filters that perform circular convolution with periodically extending the input signal rather than performing linear convolution. The circular convolution results in periodic outputs with the same period as the input; however, there are two drawbacks for this type of filters [34]. One problem is that there will be sudden jumps at the two boundaries of the signal when the input signal is periodically extended. These jumps correspond to large high frequency coefficients in the frequency domain and, in case of an image signal, the artifacts resulting from the quantization errors at one side is correlated to those of the other side. The result is disturbing especially for images where the pixel intensities at the opposite boundaries differ largely. The other practical drawback is that circular convolution requires extra memory to buffer the first few coefficients of each sequence in order to filter the samples at the end of the sequence. One solution to these two drawbacks is to extend the input signal symmetrically. In this case the jumps will be eased and the extra buffer is no longer needed; however, in order to come up with periodic outputs, we will have to use filters with linear phase [31]. There are other reasons for liking linear phase filters, specifically, the distortion is less visible. We know from the signal processing theory that linear phase filters have impulse responses that are either symmetric or anti-symmetric [20].

From the above argument, we would conclude that one good choice for implementing the wavelet transform is to use the symmetric extension method combined with linear phase filters. In the case of wavelet transforms, there is again a problem here, and that is the only orthogonal filter to have linear phase is the Haar filter, which is not a useful wavelet for the purpose of image compression [37]. For this reason, a set of more general type of wavelets has been developed that are called biorthogonal wavelets. This group of wavelets are different from the orthogonal filters in a sense that although their filters have some sort of orthogonality between them, they actually project the input signal to the basis vectors that are not orthogonal and as a result, the transform will not be energy preserving. Fortunately, this type of wavelet can be designed so that the filter coefficients are very close to being orthogonal, and two sets of these biorthogonal filters are utilized in the JPEG2000 standard.

The first type is the KDF 9/7 wavelet which is used for lossy compression. This irreversible type of wavelet filter has a floating point impulse response of length 9 and 7 for the low-pass and high-pass filters respectively. The analysis low- and high-pass filters are given by,

$$\begin{aligned} h_0(z) &= 0.602949018236 + 0.266864118443(z^1 + z^{-1}) \\ &- 0.078223266529(z^2 + z^{-2}) - 0.016864118443(z^3 + z^{-3}) \\ &+ 0.026748757411(z^4 + z^{-4}) \end{aligned} \tag{3.7}$$

$$\begin{aligned} h_1(z) &= 0.557543526229 - 0.295635881557(z^1 + z^{-1}) \\ &- 0.028771763114(z^2 + z^{-2}) + 0.045635881557(z^3 + z^{-3}) \end{aligned} \tag{3.8}$$

And the low- and high-pass synthesis filters relate to the analysis filters through the following equations [34],

$$g_0[n] = \alpha^{-1}(-1)^n h_1[n] \tag{3.9}$$

$$g_1[n] = \alpha^{-1}(-1)^n h_0[n] \tag{3.10}$$

Where $\alpha$ is a factor to compensate for the gain of the analysis filters, and its value is implementation dependent. The subband analysis and synthesis, are performed through the following time-varying convolutions,

$$y(n) = \sum_{p} x(p) h_{n \bmod 2}(n-p) \tag{3.11}$$

$$x(n) = \sum_{p} y(p) g_{p \bmod 2}(n-p) \tag{3.12}$$

The reversible (5,3) DWT has the following low- and high-pass filters. This transform maps integer image samples to integer wavelet coefficients and is used for lossless compression. The implementation of this type of filter, however, is performed through non-linear equations that are approximations of the main linear input/output filter equations, in order to efficiently map integers to integers.

$$h_0(z) = 0.75 + 0.25(z^1 + z^{-1}) - 0.125(z^2 + z^{-2}) \tag{3.13}$$

$$h_1(z) = 0.5 - 0.25(z^1 + z^{-1}) \tag{3.14}$$

In the end, we mention that the performance of the KDF 9/7 wavelet is better than the 5/3 wavelet when they are both used to code an image with a particular quality.

### 3.2.3 Code-blocks and Quality Layers

After the wavelet transform on the tile, the subbands are partitioned into so called code-blocks. Typical dimensions of the code-blocks are 32x32 or 64x64, except possibly for the code-blocks adjacent to the subband borders. Each of these code-blocks is quantized and then entropy coded independently. After the entropy coding, each code-block is represented by a quality-progressive embedded bit-stream. The entropy coding method will be discussed in details in the next section, but for now, assume that all the code-blocks in the entire image are coded, each with an embedded bit-stream as in Figure 3.4.

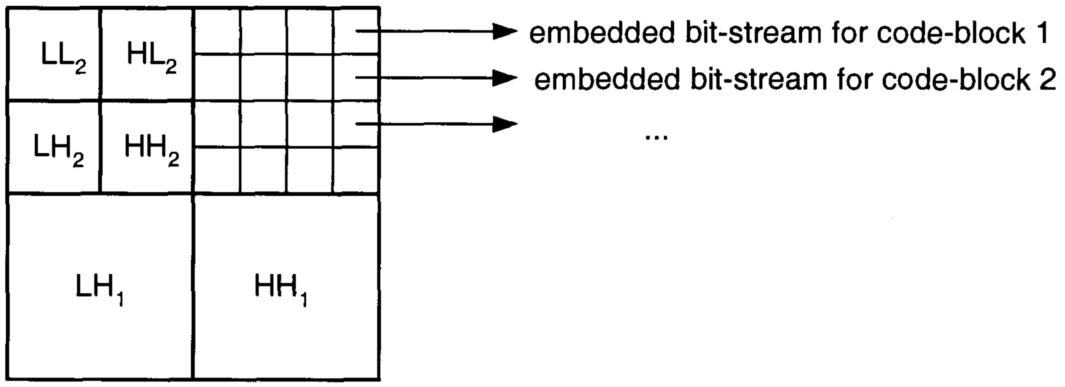

LL2
HL2
LH2
HH2
LH1
HH1
embedded bit-stream for code-block 1
embedded bit-stream for code-block 2
...

**Figure 3.4. Representation of code-blocks with embedded bit-streams, note that all the subbands are partitioned into code-blocks and each will be represented by a separate embedded bitstream**

Although each code-block is represented by an embedded bit-stream, it does not guarantee that the whole image is quality progressive. In order to come up with an efficiently scalable bit-stream, we need to determine the optimum point to truncate each code-block's bit-stream in relation to the others, in order to code the image with a specific quality. This manipulation is performed by introducing the concept of quality layers. As shown in Figure 3.5, each layer contains an incremental contribution from all of the code-blocks in the image. The vertical bars in this figure represent the embedded bitstream corresponding to each code-block of the tile. Thus the first layer contains those parts of each code-block that are shaded in the dark color. In the second layer, the parts in the light shade are also added to the data from the first layer and so on.

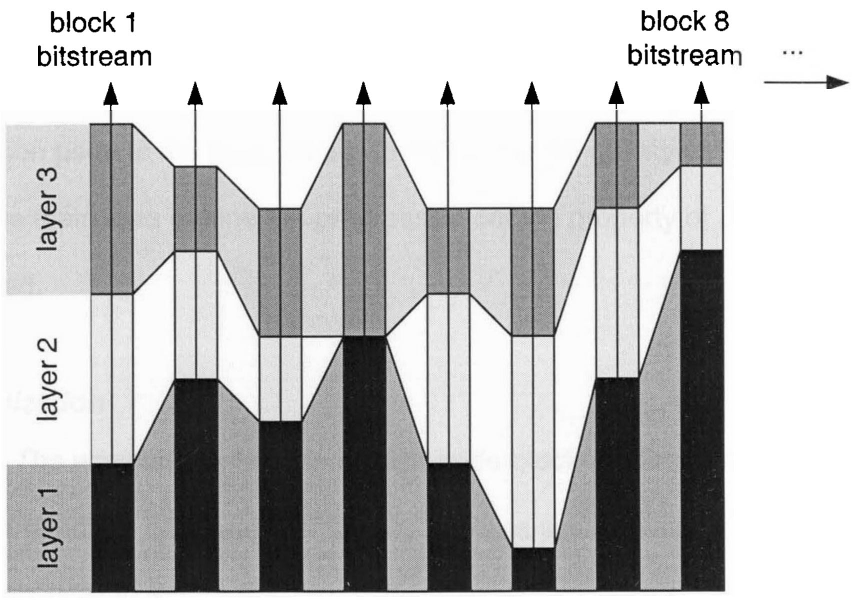


**Figure 3.5. Formation of the quality layers (taken from [33])**

Basically, the important code-blocks in specific ranges of frequency will have more contributions in the final bit-stream. Hence, even empty contribution from one or more code-blocks is permitted in a quality layer. The truncation point for each code-block is determined during the compression process, and in a manner that the bit-stream representing the quality layers from 1 to *n*, form an optimal rate-distortion representation of the image. The term "optimum" here means that the rate-distortion characteristics of the embedded bit-stream are very close and comparable to the situation where the image at the first place is coded with quantization step-sizes associated with an efficient non-embedded bit-stream. This manipulation is performed using the post-compression rate-distortion (PCRD) optimization algorithm. Explanation of this method is beyond the scope of this thesis and will not be discussed here, refer to [34] for details.

### 3.2.4 Embedded Block Coding

As mentioned above, the coefficients in each code-block are quantized and coded independently and are represented by an embedded bit-stream. That is, there are a large number of optimum truncation points available in the coded bit-stream at which the decoding process can be stopped. Continuing the decoding process from each truncation point to the next, results in improving the quality of the decoded image. In this part, the main idea behind the progressive coding property of JPEG2000 will be reviewed.

### *Quantization*

The wavelet coefficients in each code-block are first quantized and then coded as an embedded bit-stream. JPEG2000 utilizes scalar quantization with a deadzone at zero as shown in Figure 3.6.

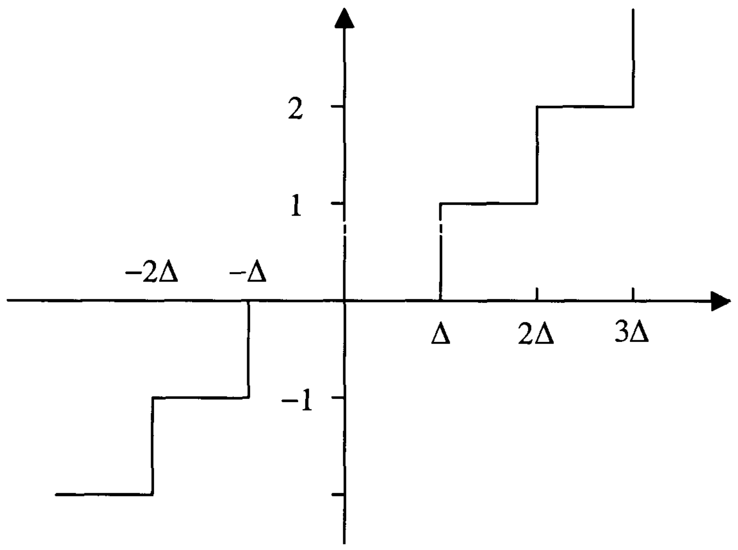


**Figure 3.6. Structure of a scalar quantizer with deadzone**

Deadzone quantization has an important role in the progression property of the standard. The central quantization bin is twice as large as the other bins and the input values that lie in that range are quantized to zero. For an input sample, *z*, and a quantizer with step size, $\Delta$ , the output, *q*, is computed as,

$$q = sign(z).\left\lfloor \frac{|z|}{\Delta} \right\rfloor \tag{3.15}$$

and at the decoder, the decoded sample $q$ is estimated as,

$$\hat{z} = \begin{cases} 0 & q = 0 \\ sign(q)(|q| + \delta)\Delta & q \neq 0 \end{cases} \tag{3.16}$$

where $\delta$ is a selectable parameter, usually chosen as 1/2.

Although the deadzone increases distortion compared to simple uniform scalar quantizers, it has two main advantages. The first important advantage is that the quantization interval is embedded within a coarser interval. That is, if $q$ is the output of a quantizer with step-size $\Delta$, and if $p$ of the least significant bits of the output $q$ are not yet known to the receiver, then the output of the quantizer is the same as if in the first place, quantization was performed with the step size $2^p.\Delta$. As an example, suppose in Figure 3.7 the input value lies in the interval $[2\Delta, 3\Delta)$, resulting in the output of the quantizer to be equal to two, or equivalently 010 in a three bit binary quantization. Now if we decode only the first MSB bit of the quantizer output, then $p = 2$ and it is the same as if at the first place quantization was performed with the step size $4\Delta$ and hence the quantizer output is 0. Now assume we receive the next MSB bit of the quantizer output, which is 1. Then $p$ is one, the virtual quantization step size is $2\Delta$ and the quantization output is equal to one.

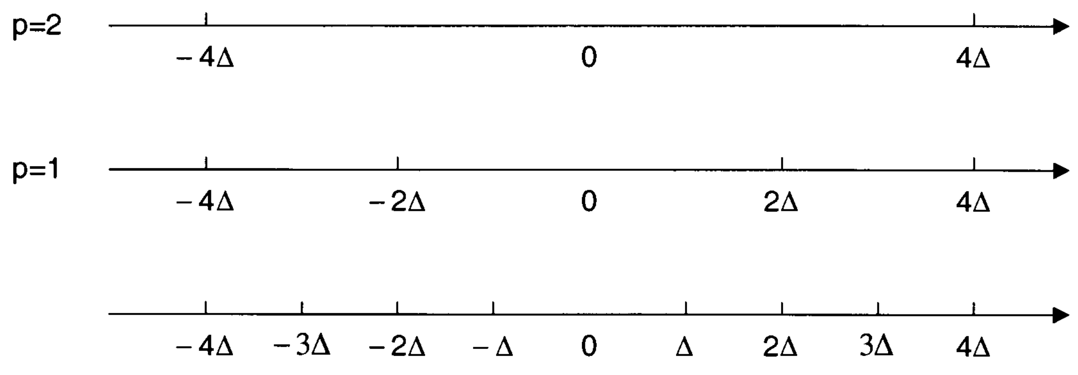


**Figure 3.7. Illustration of the embedded property of a deadzone quantizer**

Therefore at the decoder side, receiving the MSB bit of each wavelet coefficient is sufficient for reconstructing that coefficient with a coarse approximation. Receiving and decoding more of the LSB bits results in a better approximation of the original value.

Regarding the second advantage of deadzone quantizers, we note that the wavelet coefficients in the high-frequency subbands of most of the usual images tend to have small values close to zero, except the samples in the neighborhood of sharp edges. Therefore, quantizing the samples whose values are mostly close to zero with a deadzone quantizer, results in more zero samples, which can improve the compression performance.

***Bit-plane coding***

In order to achieve an efficient embedded bitstream, the coding process is performed bit-plane by bit-plane, starting from the most significant plane for each sample. A bit-plane is a binary array consisting of one bit from each sample. The MSB bits of all the samples in one code-block are coded first, followed by the second and third MSB bits etc., with the procedure continuing until the last bit-plane is reached. This situation is illustrated in Figure 3.8, where $n_1$ and $n_2$ are the image dimensions and each small cube represents a single bit. All the vertical bits in one location $(n_1, n_2)$ represent a single image pixel, with the top bit representing the MSB bit. A bit-plane includes all the bits in a horizontal plane.

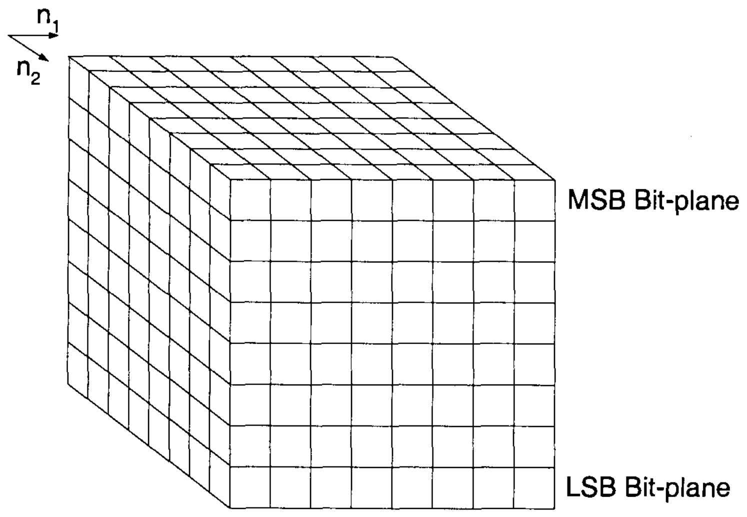


**Figure 3.8. Formation of bit-planes for the entropy coding in JPEG2000**

After decoding only a small portion of the encoded code-block bitstream, i.e. at least one bit-plane, we will be able to reconstruct all the wavelet coefficients in the code-block with a rough approximation. Then, by decoding more number of the bit-planes, the quantization step size becomes finer and the decoded values get closer to the original values. The ability to truncate the resulting code-block bit-streams part way through coding is the key to the quality progressive feature of the JPEG2000 standard.

The wavelet coefficients in each code-block are coded utilizing a context-dependent binary arithmetic coder. The coefficients are coded using a sign magnitude format. That is, for a wavelet coefficient, $z$ and quantizer step size $\Delta$ we have,

$$q = sign(z)\left\lfloor \frac{|z|}{\Delta} \right\rfloor \qquad \textbf{(3.17)}$$

Therefore, the signs are represented as one separate bit-plane. The scanning pattern of the coefficients in each bit-plane is illustrated in Figure 3.9.

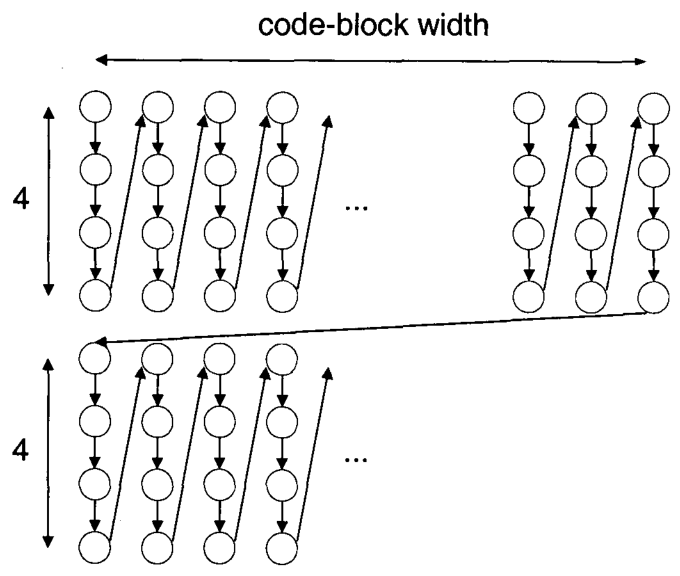


**Figure 3.9. The scanning pattern used for bit-plane coding in JPEG2000**

The coefficients in each code-block are divided into stripes of four rows, with the possible exception of the last stripe in the block, and the samples are scanned column by column from left to right. Also note that each stripe contains all of the columns in a code-block. The MSB bit-plane is coded first, and after coding the last bit in the bottom right corner of a bit-plane, the process proceeds from the top left bit in the next MSB bit-plane.

***Conditional entropy coding***

Image subband samples show common statistical properties that are exploited in the JPEG2000 entropy coding process in order to increase the coding performance. We will discuss these properties later in this section. Here, we will introduce some notation and define some important parameters, which will be used to explain the entropy coding procedure in JPEG2000.

Let $q[\mathbf{n}] = q[n_1, n_2]$ represent the quantized wavelet coefficient in column $n_1$ and row $n_2$ of a given code-block, also $\chi[\mathbf{n}] = \text{sign}(q[\mathbf{n}])$ and $v[\mathbf{n}] = |q[\mathbf{n}]|$. Also let $v^p[\mathbf{n}]$ denote the binary bits of the coefficient $q[\mathbf{n}]$, with $p$=0 representing the LSB bit, $p$=1 the next LSB bit, etc. Furthermore, an important parameter, the "significance state", is defined for each coefficient, which is denoted by $\sigma[\mathbf{n}]$. This parameter is initialized to zero at the beginning of the coding process for all the quantized coefficients, and is set to one right after the first non-zero magnitude bit is encountered for coefficient $q[\mathbf{n}]$. When the significance state of a sample is set to one, it remains unchanged until the last bit-plane is coded.

Each bit is coded with an arithmetic coder using a conditional probability histogram that is estimated using the previous coded bits. Ideally, all of the previously coded bits need be considered to estimate the new histogram; however, the sizes and the number of the distinct contexts are usually limited because of practical restrictions.

The coding process is performed in three different stages based on the significance state of each sample. If $\sigma[\mathbf{n}] = 0$, then the bit is coded in the significance coding stage. If the sample becomes significant in this stage, the sign coding stage is utilized immediately. For those samples that have already become significant in the previous bit-planes, the magnitude refinement stage is invoked in order to code the next MSB bit of the non-zero sample, which is equivalent to refining the quantization step-size. Estimates for the probability distributions associated with the possible contexts are updated after each coding operation.

## Significance coding

As mentioned above, a bit is initially coded in the significance coding stage. This coding stage works in two separate modes: normal mode and run mode.

*Normal mode*

Experimental results [33] show that the significance of a sample at any bit-plane depends heavily on the significance of the eight immediate neighboring samples. As a result, the bits coded in this mode are processed based on the significance of their eight neighboring samples, as shown in Figure 3.10.

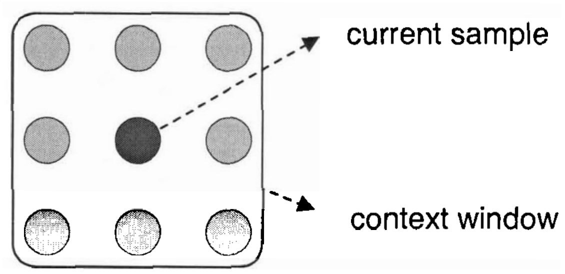


**Figure 3.10. Context for significance coding**

For coding a sample located at $\mathbf{n} = (n_1, n_2)$ in this mode, the nine contexts are selected according to the values of three secondary parameters:

$$\kappa^h[\mathbf{n}] = \sigma[n_1, n_2 - 1] + \sigma[n_1, n_2 + 1] \tag{3.18}$$

$$\kappa^v[\mathbf{n}] = \sigma[n_1 - 1, n_2] + \sigma[n_1 + 1, n_2] \tag{3.19}$$

$$\kappa^d[\mathbf{n}] = \sum_{k_1=\pm1} \sum_{k_2=\pm1} \sigma[n_1 + k_1, n_2 + k_2] \tag{3.20}$$

Any neighboring sample that lies outside of the code-block is considered insignificant for the purpose of computing the above parameters. The context number, $\kappa^{sig}[\mathbf{n}]$, is selected according to Table 3.2.

**Table 3.2. Context selection in significance coding**

| $\kappa^{sig}[\mathbf{n}]$ | LL and LH subbands | | | HL subbands | | | HH subbands | |
|---|---|---|---|---|---|---|---|---|
| | $\kappa^h[\mathbf{n}]$ | $\kappa^v[\mathbf{n}]$ | $\kappa^d[\mathbf{n}]$ | $\kappa^h[\mathbf{n}]$ | $\kappa^v[\mathbf{n}]$ | $\kappa^d[\mathbf{n}]$ | $\kappa^d[\mathbf{n}]$ | $\kappa^h[\mathbf{n}]+\kappa^v[\mathbf{n}]$ |
| 8 | 2 | x | x | x | 2 | x | $\geq 3$ | x |
| 7 | 1 | $\geq 1$ | x | $\geq 1$ | 1 | x | 2 | $\geq 1$ |
| 6 | 1 | 0 | $\geq 1$ | 0 | 1 | $\geq 1$ | 2 | 0 |
| 5 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | $\geq 2$ |
| 4 | 0 | 2 | x | 2 | 0 | x | 1 | 1 |
| 3 | 0 | 1 | x | 1 | 0 | x | 1 | 0 |
| 2 | 0 | 0 | $\geq 2$ | 0 | 0 | $\geq 2$ | 0 | $\geq 2$ |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

These contexts have been designed based upon extensive experiments performed on various types of images; although, some modifications have been done in order to make the implementation procedure more efficient, both in terms of hardware and software. We mention here that, as soon as a symbol 1 is coded in this coding stage, i.e. the sample becomes significant, then the sign of the sample is immediately coded. After the sign coding, the next symbol in the scanning pattern will be coded.

*Run mode*

When a run of successive insignificant samples are encountered, all the insignificant samples are coded with a single binary bit. This situation happens more often during the early stages of the bit-plane coding process, when the MSB bit-planes are coded. The purpose of designing this mode is mainly to reduce the number of binary bits which need be coded by the arithmetic coder. This mode also results in improvement in compression performance. More specifically, in order to enter the run-coding mode, all these three conditions must hold:

1. Four vertically consecutive samples should be insignificant in the current bit-plane, i.e. $\sigma[n_1+r, n_2]=0$ for $0 \le r < 4$.

2. All four vertically consecutive samples should have insignificant neighbors in the current bit-plane, or $\kappa^{sig}[n_1+r, n_2]=0$ for $0 \le r < 4$.

3. The four vertically consecutive insignificant samples should all reside in a single stripe of the scanning pattern shown in Figure 3.9.

In run mode, a single binary "run interruption" bit is coded to identify whether the group of four insignificant samples all remain insignificant in the current bit-plane or not. A "0" symbol indicates that they remain insignificant, while 1 shows that at least one of the four samples becomes significant. A separate context, $\kappa^{run}=9$, is allocated to code the run interruption bit. If at least one of the four insignificant samples become significant in the current bit-plane, say the sample in location $[n_1+r, n_2]$, the "run length", $r$, is also coded using two bits. Right after that, the sign of the first significant bit is also coded. The coding process is then resumed to code the significance of the next sample in the scanning pattern, i.e. location $[n_1+r+1, n_2]$ or $[n_1, n_2+1]$ if $r=3$, in the normal significance coding mode. The run length parameter, $r$, is coded in two bits starting with the MSB, using a non-adaptive uniform context, that is, zero and one have the same probability in this context throughout the coding process.

## Sign coding

The sign coding stage is utilized once for each sample as soon as it becomes significant. The exception happens when the bits in a sample location are zero in all of the bit-planes, meaning that the sample never becomes significant at any bit-plane. The sign information is coded in five different contexts according to a few intermediate parameters called "net sign bias" that are defined as follow:

$$\chi^{h}[\mathbf{n}] = \chi[n_1, n_2 - 1]\sigma[n_1, n_2 - 1] + \chi[n_1, n_2 + 1]\sigma[n_1, n_2 + 1] \quad \textbf{(3.21)}$$

$$\chi^{v}[\mathbf{n}] = \chi[n_1 - 1, n_2]\sigma[n_1 - 1, n_2] + \chi[n_1 + 1, n_2]\sigma[n_1 + 1, n_2] \quad \textbf{(3.22)}$$

These two parameters lie in the range [-2,2]. Using the following equations, the values of the parameters are truncated in order to change their range to [-1,1]:

$$\chi^{-h}[n] = \text{sign}(\chi^{h}[\mathbf{n}]).\min\{1, |\chi^{h}[\mathbf{n}]|\} \quad \textbf{(3.23)}$$

$$\chi^{-v}[n] = \text{sign}(\chi^{v}[\mathbf{n}]).\min\{1, |\chi^{v}[\mathbf{n}]|\} \quad \textbf{(3.24)}$$

The context number, $\kappa^{sign}[\mathbf{n}]$, and sign-flipping factor, $\chi^{flip}[\mathbf{n}]$, are both determined using the information shown in Table 3.3. The binary symbol which is finally coded will be 0 if $\chi[\mathbf{n}].\chi^{flip}[\mathbf{n}] = 1$ and 1 if $\chi[\mathbf{n}].\chi^{flip}[\mathbf{n}] = -1$.

**Table 3.3. Context selection for the sign coding stage**

| $\chi^{-h}[\mathbf{n}]$ | $\chi^{-v}[\mathbf{n}]$ | $\kappa^{sign}[\mathbf{n}]$ | $\chi^{flip}[\mathbf{n}]$ |
|---|---|---|---|
| 1 | 1 | 14 | 1 |
| 1 | 0 | 13 | 1 |
| 1 | -1 | 12 | 1 |
| 0 | 1 | 11 | 1 |
| 0 | 0 | 10 | 1 |
| 0 | -1 | 11 | -1 |
| -1 | 1 | 12 | -1 |
| -1 | 0 | 13 | -1 |
| -1 | -1 | 14 | -1 |

**Magnitude refinement coding**

The magnitude refinement, MR, coding stage is performed for samples that have already become significant in one of the previous bit-planes, i.e. when $\sigma[\mathbf{n}] = 1$. This information is used at the decoder to refine the quantization index, producing a decoded value closer to the encoded sample.

The symbols are coded in one of the three different contexts allocated to this coding stage. The MR coding context, $\kappa^{mag}[\mathbf{n}]$, is chosen as shown in Table 3.4. The parameter $\tilde{\sigma}[\mathbf{n}]$ is the significance state of the symbol being coded delayed by one bit. This quantity is initialized to zero for each symbol and remains zero until one bit-plane after the first non-zero magnitude bit is coded, at which point it is set to one. Thus, it is set to one, one bit-plane after the significance state. The $\kappa^{sig}[\mathbf{n}]$ parameter is derived from the significance state of the neighboring eight samples: it is zero if all the eight neighboring symbols are insignificant in the current bit-plane, and greater than zero otherwise (the actual values of $\kappa^{sig}[\mathbf{n}]$ are not needed here).

**Table 3.4. Context selection for the magnitude refinement stage**

| $\tilde{\sigma}[\mathbf{n}]$ | $\kappa^{sig}[\mathbf{n}]$ | $\kappa^{mag}[\mathbf{n}]$ |
|---|---|---|
| 0 | 0 | 15 |
| 0 | $>0$ | 16 |
| 1 | $\times^{*}$ | 17 |

***Coding passes and fractional bit-planes***

With the coding scheme explained in the last sections, the only natural truncation points in the coded bit-stream are the bit-planes end-points. These points correspond to the encoded coefficients quantized with step-sizes equal to $2^{p}.\Delta$, where $p$ is the bit-plane number at which the bit-stream is truncated. If the bit-stream is truncated at a point other than the bit-planes end-points, a portion of the samples will have larger quantization step-sizes than the rest, and this is not good in terms of rate-distortion characteristics [34].

In JPEG2000, the coded bit-stream of a code-block has many more useful truncation points than just bit-planes endpoints. This property is achieved by coding the symbols in each code-block in three consecutive passes. In each pass, the entire bit-plane is scanned with the pattern shown in Figure 3.9, however, only a portion of the symbols are coded; and at the end of the third pass, all of the bits in the code-block are coded. The first pass only codes the significance of the samples that are likely to become significant in the current bit-plane, and this information is likely to result in the most reduction in the distortion relative to the coding cost. Also in the third pass, those symbols are coded that are likely to have the least effect in the overall distortion relative to coding cost. Each pass chooses its members dynamically, and according to the significant state of each symbol's neighboring coefficients, as explained below.

Figure 3.11 shows the rate-distortion characteristics of the bit-streams resulting from the two situations; once a typical code-block is coded in one pass, and the other time in three passes. As shown in the figure, the distortion corresponding to the bit-stream coded in three passes is almost always lower than the bit-stream coded in one pass. The exceptions are, of course, the code-block end-points.

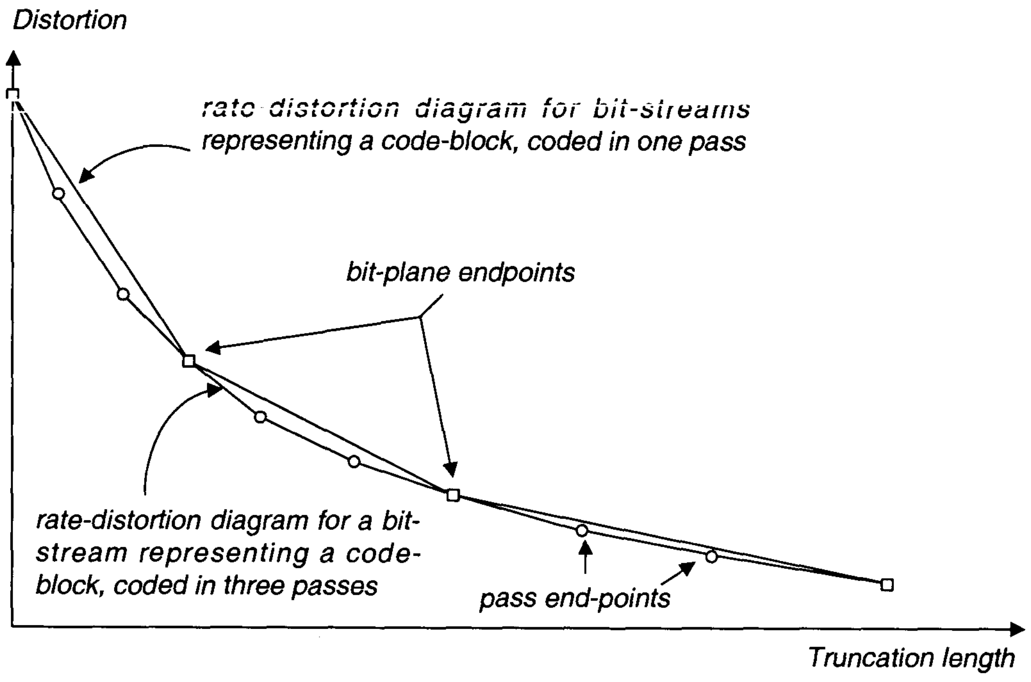


**Figure 3.11. Rate-distortion characteristics of the bit-streams resulting from two situations; once a code-block is coded in one pass, and the other time in three passes**

**Significance propagation pass**

A sample is a member of this pass if it is insignificant and has at least one significant neighbour; i.e., $\sigma[\mathbf{n}] = 0$ and $\kappa^{sig}[\mathbf{n}] > 0$. The name of this pass comes from the fact that when a sample is coded in this pass, it causes its four neighboring samples to be coded in this pass if they are insignificant.

**Magnitude refinement pass**

A sample belongs to this pass, if it has become significant in a previous bit-plane. In this pass the next magnitude bit of the sample is coded.

**Cleanup pass**

Those samples that have not been included in the first two passes are coded in the cleanup pass. Consequently, a sample belongs to this pass if it is insignificant, and has no significant neighbors. Also note that the conditions for run length coding, can only hold in this pass.

### 3.2.5 Region of Interest coding

Region-of-interest (ROI) coding allows user-defined parts of an image to be coded with a higher quality than other regions, and the subject of this thesis is related to this feature. In JPEG2000, ROI coding can be implemented by de-emphasizing wavelet coefficients not associated with the ROI reconstruction, based on the bit-plane by bit-plane coding scheme discussed in the last section. In other words, the "background" coefficients are shifted down towards the least significant bits, prior to the bit-plane coding utilized in the JPEG2000 coding scheme. This manipulation results in the ROI coefficients being coded before those of the background. The decoder can easily reverse the scaling provided that it knows which coefficients are in the ROI and the amount of scaling that has been done.

If the coded bit-stream is truncated, or the decoding process is terminated at any point before the decoding process is complete, then the ROI will have a better quality than the background, as the ROI will be reconstructed with a greater number of bit-planes. If the entire bit-stream is decoded, the whole image can be reconstructed in full quality. The full quality is actually the maximum quality that is chosen at the coding time. Two types of ROI coding methods are supported by the JPEG2000 standard: the Scaling-Based method and the Maxshift method.

***Scaling-Based method***

In the Scaling-Based (SB) method [14], any scaling value, *s*, is permitted, since the ROI shape information is included in the coded bit-stream for the decoder. The scaling value controls the relative quality between the ROI and background qualities during the progressive decoding, and it is added as side information to the coded bit-stream. A mask is generated both at the encoder and decoder specifying the wavelet coefficients that belong to the ROI. The coding process is performed through the following five steps:

1. Apply wavelet transform and compute wavelet coefficients of the image

2. Derive the ROI mask. The mask specifies which wavelet coefficients in each subband participate in the ROI reconstruction

3. Quantize the wavelet coefficients

4. Downshift the background coefficients by *s* bit-planes

5. Perform the bit-plane by bit-plane entropy coding

At the decoder, the samples are entropy decoded, the mask is derived and the background coefficients determined by the mask are shifted up. The “don't care” bits are removed and finally the dequantization and the inverse wavelet transform steps are applied.

As illustrated in Figure 3.12, scaling results in extra "don't care" bits appearing in the LSB region of the ROI coefficients. These bits are marked with an "x" in the figure and can be set arbitrarily by the encoder: they are simply discarded after the background samples are shifted back to their original places. Despite their “don't care” status, these bits must be encoded for compatibility with the standard coder, and their values affect the bit rate; setting them appropriately in order to reduce the coding cost of the code-block is the problem that is dealt with in this thesis.

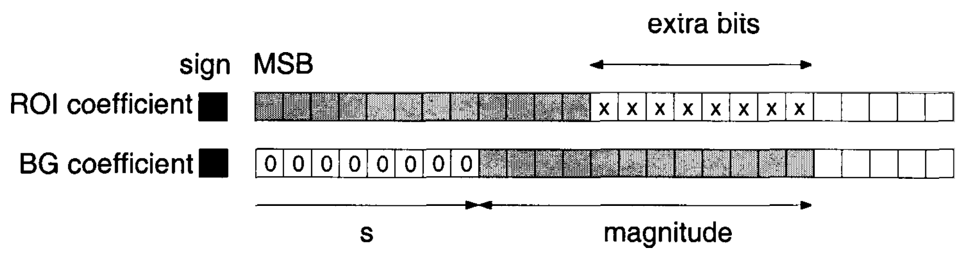


**Figure 3.12. Illustration of the Scaling-Based method**

Mask generation is necessary in both encoder and decoder side. In order to generate the mask, we introduce the following function,

$$M(\mathbf{j}) = \begin{cases} 1 & \text{if} \quad (j_1, j_2) \in ROI \\ 0 & \text{otherwise} \end{cases} \tag{3.25}$$

where $\mathbf{j} = [j_1, j_2]$ is a vector that indexes samples in the image. Now, we start performing the wavelet transform. After doing the first step, which is decomposition of the image into two subbands, $M(j_1, j_2)$ is updated line by line and column by column for all **j**'s belonging to the image dimensions. If the sample in location $(j_1, j_2)$ is needed to reproduce one of the ROI coefficients in the composition process of two subbands into one, then $M(j_1, j_2)$ is set to one, and it is set to zero otherwise. The next step is the decomposition of two subbands into four. Once again the function is updated, and this time the values of those locations that participate in reproducing the ROI sample in the composition of two subbands onto four, are set to one. This process is repeated for all of the subband decomposition stages, as illustrated in Figure 3.13. Also note that mask derivation is a function of ROI shape, length of the wavelet filters and also the number of decomposition levels in the wavelet transform.

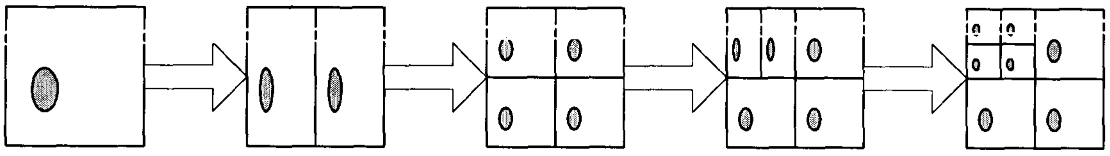


**Figure 3.13. Illustration of mask generation**

As mentioned earlier, the ROI shape information must be added to the coded bit-stream; however, adding shape information results in two major issues. Firstly, it increases the bit-rate and thus reduces compression performance, and secondly, it increases the coder and decoder complexities as a result of the need for deriving a mask. Both of these problems are alleviated by restricting the shape of the ROI to specific geometrical shapes such as rectangles and ellipses. In these cases, only the locations and dimensions of the ROI, which can be coded with a very small number of bits, are included in the coded bit-stream. Also a fast algorithm [7] has been introduced for deriving the mask for these shapes, which significantly reduces the coder and decoder complexities.

***Maxshift method***

The Maxshift (MS) method [13] is a special case of the SB method, where the scaling value, *s*, is so large that there is no overlap between the ROI and any of the background bit-planes in all of the code-blocks. This situation is illustrated in Figure 3.14. In this case, there is no need to send the shape information to the decoder, since the ROI samples can be recognized at the decoder via comparing to the threshold $2^s$. If the sample is larger than the threshold it belongs to ROI, otherwise it is a background sample and should be shifted up after decoding. As a result, there is no need for mask generation at the decoder. As shown in Figure 3.14, again the shifting process results in extra bits in the LSB side of the ROI samples. These bits also need be coded at the coder side, and are discarded by the decoder. As we will see in the next chapter, there are restrictions in choosing the values of these bits and these bits are better set to zero.

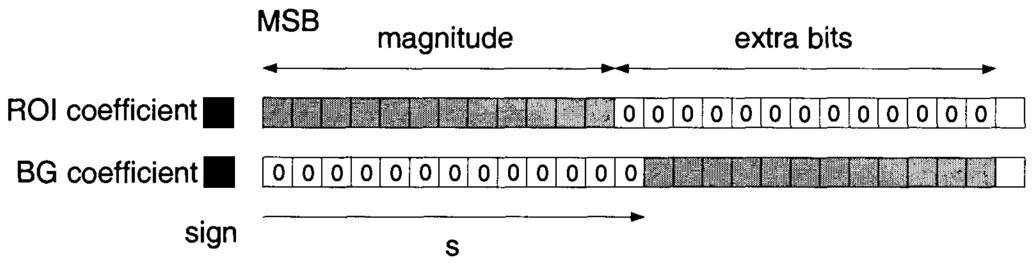


**Figure 3.14. The shifting process in the Maxshift method**

Both of these two ROI coding methods have advantages and disadvantages. In the SB method, any scaling value is permitted, and any relative quality between the ROI and background is feasible. However, adding the shape information to the coded bitstream reduces the compression performance. We saw that the problem could be alleviated by restricting the ROI to particular shapes like rectangles or ellipses; however, in some cases we might be interested in ROI's associated with specific objects in an image, which generally have arbitrary shapes. On the other hand, in the MS method no shape information is needed at the decoder, thus any arbitrary ROI shape is permitted; on the downside however, there is no control over the relative quality between the ROI and background, as the scaling value is fixed to the extreme case. Hence, in the progressive decoding process, there will be no information from the background before the ROI is decoded in full quality. The techniques used in the two ROI methods are illustrated in Figure 3.15.

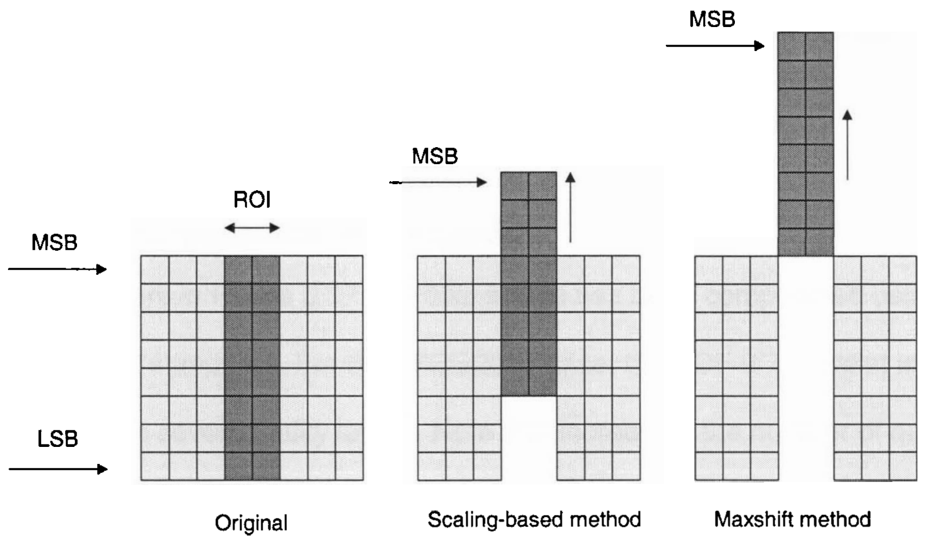


**Figure 3.15. ROI coding in JPEG2000**

Finally it is worth mentioning that, coding an image with an ROI increases the file size since there is a greater number of bit-planes to be coded in this case compared to the image without an ROI. The amount of increase depends on the size of the ROI, the coding method that is used and also the scaling factor. Experiments [6] show that, in lossless coding of an image with a rectangular ROI of the size 25% of the original uncompressed image, the bit-rate increases for about 1-8% in the MS method and 0.5-

4% in the SB method. Figure 3.16 shows an example of ROI coding applied to the Barbara image.

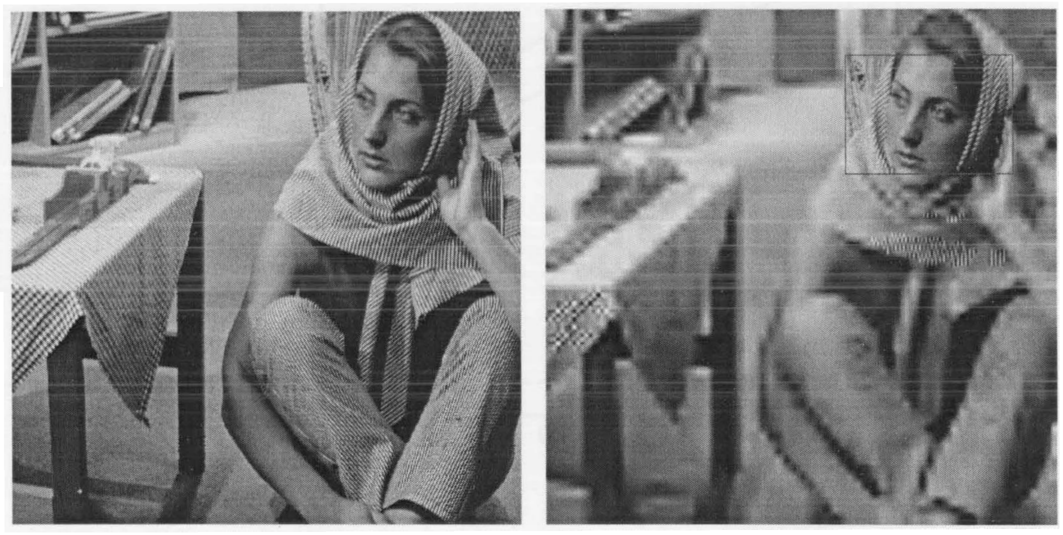


**Figure 3.16. ROI coding implemented on Barbara image**

### 3.2.6 JPEG2000 compression performance

In an experiment in Figure 3.17, the bike image has been compressed using the JPEG2000 and JPEG standards. For the JPEG2000 case, the KDF 9/7 wavelet is used, and the bit-stream has seven quality layers. Note that increasing the number of quality layers degrades the compression performance.

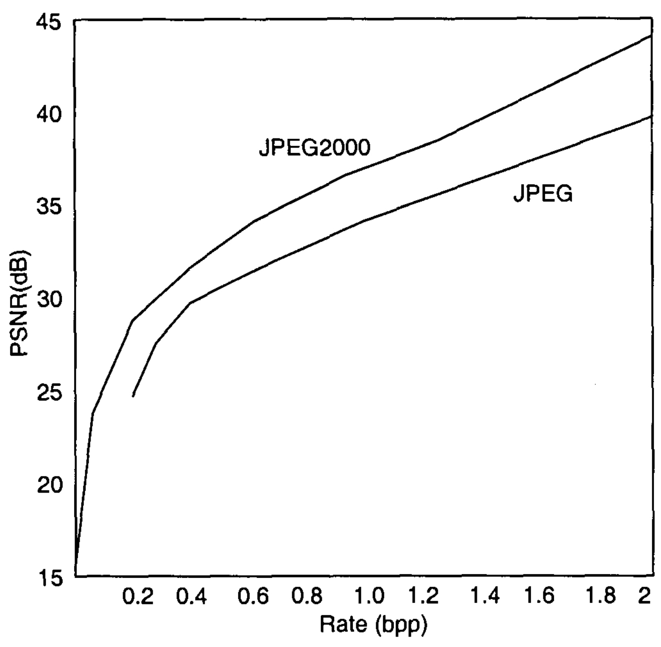


**Figure 3.17. Compression performances of JPEG2000 and JPEG standards on a test image, bike. (part of a graph in [28])**

In the lossless case as shown in Table 3.5, JPEG2000 performs much better than JPEG, but somewhat worse than JPEG-LS.

**Table 3.5. Lossless compression performance of three standards**

| Method | Images | | |
|---|---|---|---|
| | Aerial2 | Bike | Barbara |
| JPEG | 5.589 bpp | 4.980 bpp | 5.663 bpp |
| JPEG-LS | 5.286 bpp | 4.356 bpp | 4.863 bpp |
| JPEG2000 | 5.467 bpp | 4.562 bpp | 4.823 bpp |

In this chapter we introduced the features and a brief overview of the JPEG2000 standard. Also, the problem that will be dealt with in this thesis was defined at the last part of the chapter. In the next chapter, the problem will be completely defined in complete details and our approach to choose the values of the “don’t care” bits in both ROI methods will be discussed.

# CHAPTER 4
# SELECTING THE DON'T CARE BITS

## *Introduction*

Two types of ROI coding methods are supported by JPEG2000, the Scaling-Based method and the Maxshift method. As illustrated in Figure 3.12 and Figure 3.14, the shifting process in the scaling-based method results in extra "don't care" bits appearing in the LSB region of the ROI coefficients. These bits are shown in white background in the figures and can be set arbitrarily by the encoder. Despite their "don't care" status, these bits must be encoded and their values affect the bit-rate; setting them appropriately is the subject that will be discussed in this chapter.

## *4.1 Don't care bits in the Maxshift method*

In the Maxshift method, there are restrictions in choosing the extra bits values. In fact, if the value of an ROI sample is zero and one of the extra bits is set to one, then that coefficient will be wrongly considered as a background coefficient and will be shifted up by the decoder [12]. Thus, the extra bits should always be set to zero with this ROI strategy.

## *4.2 Don't care bits in the Scaling-Based method*

Unlike in the Maxshift method, in the scaling-based method the ROI samples are distinguished from the BG coefficients using the encoded shape information. Thus, there are no restrictions in choosing the values of the "don't care" bits. In the current JPEG2000 coders, the don't care bits are all set to zero for simplicity [12]. Our goal is to select the values of the "don't care" bits in a way that results in the smallest possible coded bit-stream representing code-blocks containing ROI's. To start with, it is important to note that the "don't care" bits in the scaling-based method can be divided into two categories:

*Category #1.* The "don't care" bits belonging to the ROI symbols with non-zero values;

*Category #2.* The "don't care" bits that belong to the zero-valued ROI coefficients.

Note that we are interested in the values of the symbols after they are quantized.

As we shall see, the frequency of zero-valued coefficients in a code-block will affect the gains possible with our approach. This frequency depends on factors such as the quantization step size, the subband level, the subband type, the image characteristics and the type of the wavelet transform. As an example, however, the number of these symbols in a "typical" $32 \times 32$ code-block drawn from a three-level subband decomposition of the Barbara image is shown in Table 4.1. The wavelet transform utilized is the KDF 9/7 kernel, which is supported in part 1 of the JPEG2000 standard, and the quantization step size is 1/256, which is the default value in the JPEG2000 implementation in [34] when the wavelet coefficients are normalized to the range -0.5 to 0.5. Based on many similar experiments, we expect around 25% of the symbols in a code-block to be zero, unless the block is from the low-pass subband. One of the reasons that results in high number of zero-valued samples is the specific type of quantizer that is used in JPEG2000, which has a deadzone on the zero area.

**Table 4.1. Popularity of the zero-valued symbols in a sample 32x32 code-block (Barbara). And the percentage of the zero valued samples of the total 1024 samples in a code-block**

| **subband** | **$HH_1$** | **$LH_1$** | **$HL_1$** | **$HH_2$** | **$LH_2$** |
|---|---|---|---|---|---|
| **number of zero samples** | 207 | 296 | 261 | 253 | 407 |
| **percentage** | 20.2% | 28.9% | 25.5% | 24.7% | 39.7% |
| **subband** | **$HL_2$** | **$HH_3$** | **$LH_3$** | **$HL_3$** | **$LL_3$** |
| **Number of zero samples** | 330 | 165 | 281 | 158 | 2 |
| **subband** | 32.2% | 16.1% | 27.4% | 15.4% | 0.2% |

The don't care bits in category #1 belong to the samples that have already become significant in the previous bit-planes and are therefore coded in the three contexts allocated to the MR coding; i.e. contexts 15, 16 and 17 as shown in Table 3.4. The majority of the bits in category #1 are coded in context 17, owing to the fact that most ROI coefficients become significant in the earlier bit planes. Figure 4.1 gives an example, where the sample becomes significant in the third bit-plane and thus the bits in the 4th-15th bit-planes are coded in MR contexts. $\overset{>}{s}$**[n]** toggles to one after the 4th bit is coded and so this parameter is equal to one for the bits in the bit-planes 5 to 15. As a result, only the bit located in the fourth bit-plane is coded in either context 15 or 16 depending on the significance of the neighboring samples. All the bits located in the 5th-15th bit-planes are then coded in context 17 and there is no dependence on the neighboring samples in this context situation.

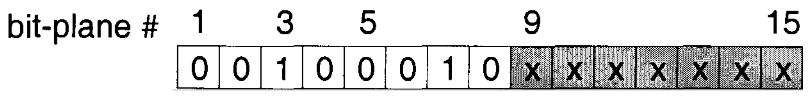


**Figure 4.1. An example of a non-zero ROI sample after the scaling process**

The only exception to the above happens when the absolute value of an ROI sample is equal to one, in which case $\overset{\frown}{s}[\mathbf{n}]$ is zero for the first don't care bit, and only this bit is coded in either context 15 or 16. The other don't care bits are all coded in context 17.

In order to select the values with the lowest coding cost for the "don't care" bits in category #1, first we look at the pdf function of the wavelet coefficients. According to our experiments, also mentioned in [34], the pdf function of the wavelet coefficients in a typical image is heavily skewed toward zero. The histogram of the wavelet coefficients found in a typical 64×64 code-block of the Barbara image is shown in Figure 4.2. As we can see, the histogram is almost symmetric and hence independent of the sign of the sample. In addition, the histogram is monotonically decreasing in either sides of zero, therefore, given that the value of a coefficient lies in a specific range, say between four and seven, there is a better chance that the sample has a smaller value in that range; i.e., the value is more likely to be close to four. Equivalently, if a sample becomes significant at any bit-plane, then it is more likely that each of the following bits are "0"s, rather than "1"s. This property results in the histograms of the three magnitude refinement contexts, i.e. 15, 16 and 17 being biased toward zero. Recall that all the magnitude bits of an already significant sample are coded using the MR contexts, which implies that the best way of choosing the values of the category #1 "don't care" bits is to set them to zero.

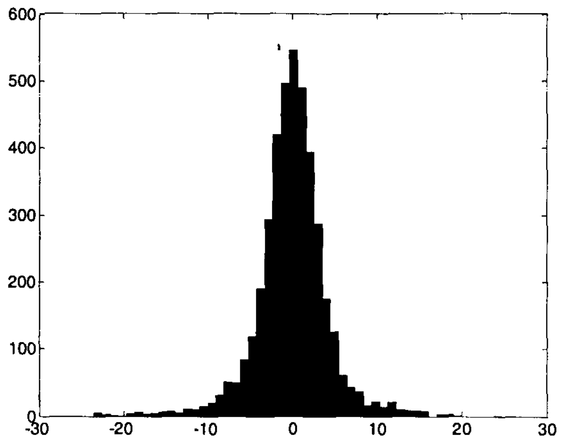


**Figure 4.2. Histogram of a 64x64 code-block in Barbara**

Regarding the bits in category #2, the zero-valued symbols never become significant in any bit-plane prior to the first extra bit's bit-plane, and the first extra bit is always coded using one of the first 10 contexts reserved for significance information. Subsequent "don't care" bits are also coded using these same 10 contexts unless a previous "don't care" bit for this coefficient has been set to one.

Taking a closer look at how category #2 bits are coded, we recall from the last chapter that the run significance coding mode is used when there are four consecutive insignificant samples in the scanning pattern, each with insignificant neighbors. We have observed that this condition rarely happens for the "don't care" bits, once the BG coefficients begin to be coded in the middle stages of the coding process. This observation can be explained by noting the fact that the run mode is usually utilized for coding significance information of the samples in the early bit-planes when significance is not common among the samples. Furthermore, the ROI samples tend to cluster together and an ROI sample is unlikely to be isolated among eight background samples. Hence, there are at least a few ROI samples in the neighborhood of an ROI sample and some of those ROI samples are likely to be significant at that stage. For these reasons, almost all of the “don't care” bits that belong to a zero-valued ROI symbol are coded using the normal significance coding mode, or contexts 0-8.

The contexts in this mode are selected according to Table 3.2, where it can be observed that the higher index contexts occur when there are a higher number of significant neighbors and thus we normally expect that the “don't care” bits in the second category tend to be processed using contexts with indices in the upper level of the range 0-8.

As discussed in [34], the histograms of these nine contexts are skewed toward zero. However, as shown in Figure 4.3, our experiments show that the skew reduces as the context index increased. This effect is expected since the symbols coded in these contexts come from neighborhoods where significance is more common and symbols are more likely to be significant. In these cases, the two possibilities are almost equiprobable and setting the first “don't care” bit to “1” will not increase total coding cost significantly. The advantage of this strategy is that the subsequent “don't care” bits will now be coded in the three MR contexts, and as we will discuss later, these contexts are more skewed than those used for significance coding in the code-blocks with ROI scaling. The result is an overall improvement in the performance.

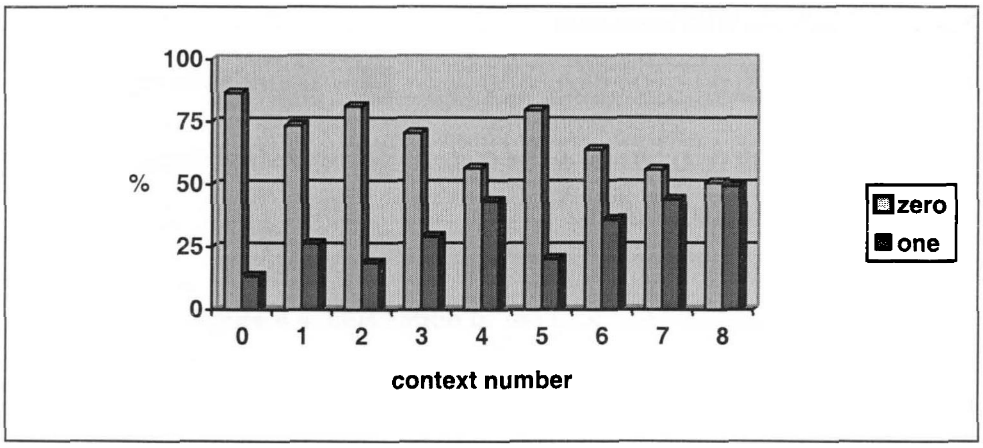


**Figure 4.3. Histograms of the first nine contexts of a 32x32 code-block in Barbara. The skew toward zero decreases in higher-level contexts.**

## *4.3 The proposed method*

In this section, we discuss the technique that we propose for setting the "don't care" bits for the category #2 symbols, those whose value is zero after quantization. As discussed above, the "don't care" bits of the category #1 are simply set to zero.

We start by setting the first "don't care" bit of each of the zero-valued ROI symbols to "1" and the remainder to "0", as shown in Figure 4.4. In this case, the first extra bit is coded using one of the significance coding contexts and, as a result, the sample becomes significant. The following "don't care" bits are then arithmetically coded using the MR contexts, which are coded more efficiently than the significance coding contexts due to their high skew. Indeed, as illustrated in Figure 4.5, for the code-blocks that include ROI samples, the MR context histograms, and especially the histogram for context 17, become hugely skewed toward zero and a zero is coded with very little cost. This result is because the "don't care" bits in the first category have been set to zero, thus increasing the skew of these histograms even further.

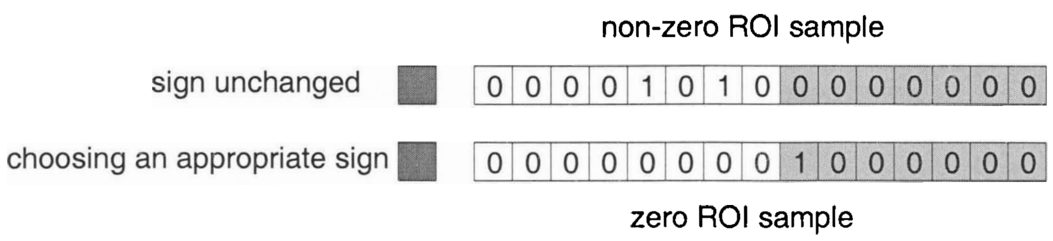


**Figure 4.4. Illustration of the proposed method**

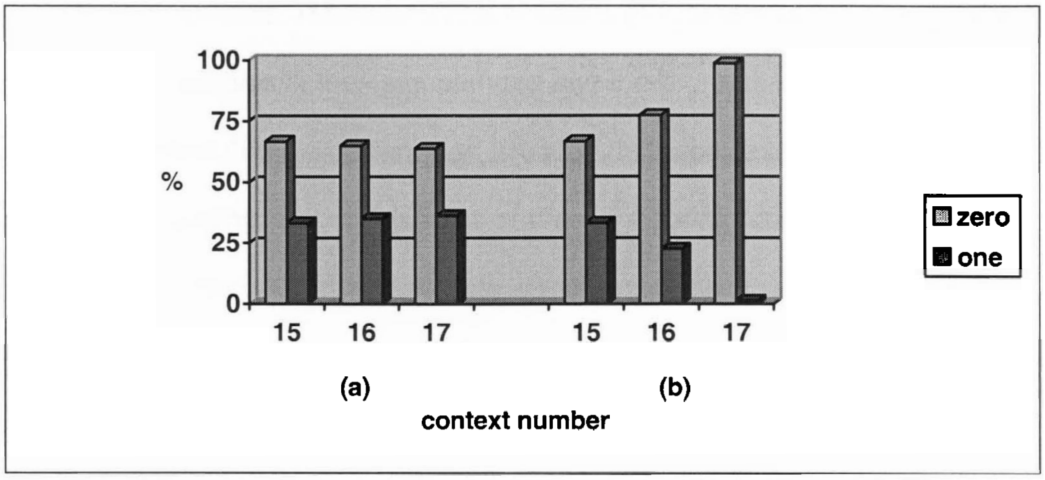


**Figure 4.5. Histograms of the MR contexts in a sample 32x32 code-block in the $HL_1$ subband in Lena. Part (a) shows the histograms without ROI scaling. In part (b) all of the samples in the same code-block as part (a) are considered to belong to the ROI. The scaling factor is six and the samples are represented by seven magnitude bits**

The sign information of a coefficient is coded immediately after it becomes significant – and this is new information that doesn't need to be coded when both a coefficient and its "don't care" values are zero. However, although the cost of sign coding slightly increases the total bit-rate, the improvement resulting from coding the "don't care" bits using the MR contexts usually well-compensates for this extra cost, as shown in Table 5.1. The exception is the case where the scaling factor is very small, as illustrated in Table 5.2. In addition, this sign is actually an additional "don't care" quantity, since the real value of the sample is zero. As a result, the sign coding cost can be alleviated by choosing appropriate values for the signs, as discussed below.

***Choosing a set of low-cost signs***

As mentioned in the previous chapter, the signs of the samples are coded using five contexts according to Table 3.3. In order to choose a low cost sign for the newly significant sample, we are actually interested in the sign that is more popular in the "don't care" sign's context. However, in some cases the current sample has one or more category #2 ROI samples in the neighborhood and therefore, there is flexibility in choosing the context that the current sign is coded. In these cases, it might be useful in terms of the total coding cost, to flip some of the neighboring "don't care" signs so that the current sample is coded in a more skewed context. Also we should keep in mind that each sample might lie in the context of some more samples and thus the minimization task should be performed globally.

The skew of the sign contexts are different in the low-pass and high-pass subbands. According to [35], in LH (vertically high-pass) subband of a typical image, horizontally adjacent samples tend to have the same signs; while vertically adjacent samples tend to have opposite signs. In HL subbands, the skew is the other way around. In these subbands, the horizontally adjacent samples tend to have opposite signs and the vertically adjacent samples tend to have the same signs. The histograms of the sign contexts in a sample code-block in the HL subband of Barbara are shown in Figure 4.6.

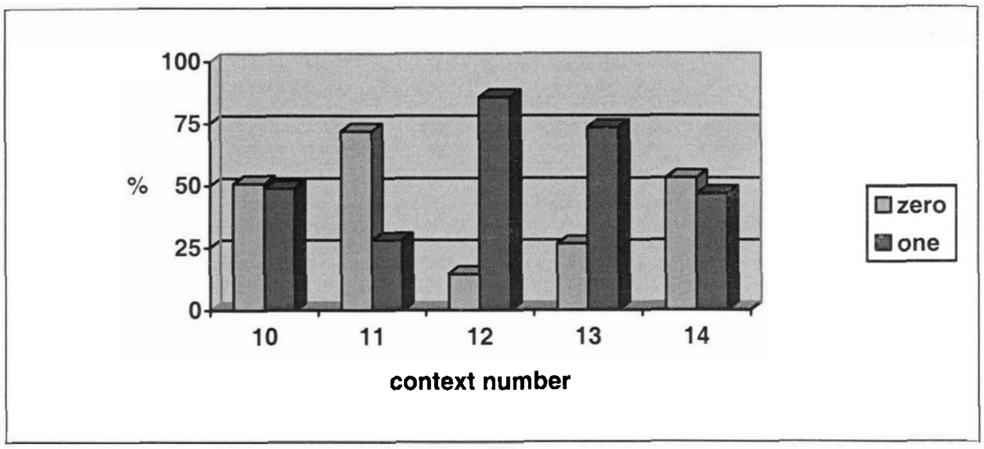


**Figure 4.6. Histograms of the sign contexts in a sample code-block from an HL subband in Barbara. The histogram of each context is provided independently and based on the signs that are coded using that context**

The above argument is verified by these histograms as the context 12 has the most skewed histogram. Based on Table 3.3, this sign context has one of the formations shown in Figure 4.7. In part (a) of the figure, both the horizontal and vertical neighbors act in favor of a negative sign for the current symbol, while in part (b) the horizontal and vertical neighbors are in favor of a positive sign. Note that in part (a), $\chi^{-h}[\mathbf{n}] = 1$, $\chi^{-v}[\mathbf{n}] = -1$ and $\chi^{flip}[\mathbf{n}] = 1$ and thus a negative sign, which is the more probable sign, is coded with "1", while in part (b) a positive sign, or the more probable sign, is coded with "1" since in this case $\chi^{-h}[\mathbf{n}] = -1$, $\chi^{-v}[\mathbf{n}] = 1$ and $\chi^{flip}[\mathbf{n}] = -1$. Also, other configurations of this context are also possible, i.e. one of the vertical (or horizontal) neighbors can be insignificant.

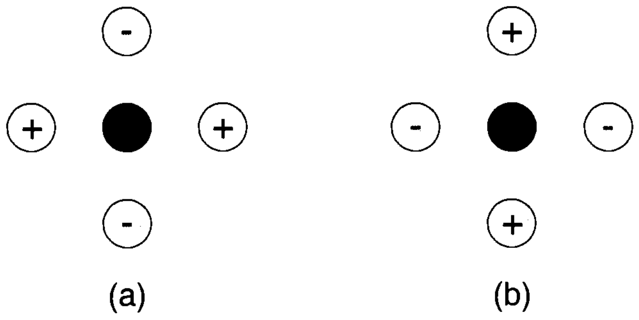


**Figure 4.7. Two possible formations of the context #12**

The histograms of the other sign contexts are usually less skewed and more data dependent since in some cases the contexts contain non-significant samples and some cases the vertical and horizontal neighbors are in favor of opposite signs, thus making the histograms less skewed. The above argument implies that if we have the choice to choose the signs of the surrounding samples, we should try to code as many signs as possible using the context 12, which is more skewed. Also note that if non of the four neighboring symbols of the current “don't care” sample are zero-valued ROI sample and thus have fixed signs, then the context number cannot be changed. In this case the choice of the “don't care” sign is straightforward, as we need to choose the sign that is more popular in the current context.

As mentioned before, changing the sign of each “don't care” symbol affects the sign coding cost of its four neighboring samples, as it actually lies in their context windows. Some of those neighboring samples might in turn, be category #2 ROI samples and will thus also have “don't care” signs. As a result, signs of the “don't care” samples should be chosen globally and in a way to minimize the overall coding cost of the sign bit-plane. Unfortunately, it is not feasible to examine the cost of all of the possible sign configurations in a code-block. Recall that there are roughly ~250 zero-valued wavelet coefficients in a 32×32 code-block in a typical image subband and thus we need to compute the coding cost of $2^{250}$ possible sign patterns if doing a full search. Alternative methods are clearly needed.

In order to find a reasonable set of sign values, we used simulated annealing (also called the Metropolis algorithm [17]) for the search method. In this approach, we start with an initial set of signs for the "don't care" samples and compute the coding cost of the code-block. In the next step, each "don't care" coefficient is assigned an equal probability and one of these coefficients chosen by random selection. The sign of the chosen coefficient is flipped and the coding cost is computed again. If the second cost, $C_2$, is lower than the initial cost, $C_1$, then the change is accepted. Otherwise, if the change results in an increase in the total cost, then the change is only accepted with a probability equal to

$$p = e^{-(C_2 - C_1)/T} \tag{4.1}$$

where $T$ is temperature in the Metropolis method. Also in order to reduce complexity and computation, it suffices to compute only the sign bit-plane cost of the code-block rather than the entire code-block with magnitude bit-planes, since all of the samples become significant in some point and therefore all of the signs are eventually coded.

In high temperatures, the exponential probability increases and thus more changes will be accepted. In lower temperatures, however, few of the changes that increase the cost will be accepted. Working in high temperatures is useful in a sense that it avoids being trapped in a local minimum and results in jumping over the local minima. The best strategy for our problem is to start with a high temperature, say 0.5-1, and decrease it linearly to a value of around 0.01 with every sign flip attempt during the search to a minimum. We observed that the cost converges to a reasonable minimum if we run the algorithm for about 1000 to 5000 trials as illustrated in Figure 5.1. The temperature values have mostly been picked through trial and error, and considering the fact that each sign flip attempt results in the cost function to fluctuate 0.5-2 bits on average.

In order to further reduce the computations in our search for a set of low-cost signs, we note that since in each iteration in the Metropolis method, only the sign of one ROI sample is flipped, this change only affects the sign coding cost of just a few samples in the neighboring area of the current sample. More accurately, as shown in Figure 4.8, the change only affects the coding cost of the current sample as well as the horizontally and vertically adjacent samples to the current sample. Suppose in this figure the sign of the sample at location (3,4) is flipped. In this case, the coding cost of the sample (3,4) is obviously changed. Also, the coding cost of the samples at locations (3,3), (3,5), (2,4) and (4,4) will also change as the sample at (3,4) is within the context windows that are used to code these four samples.

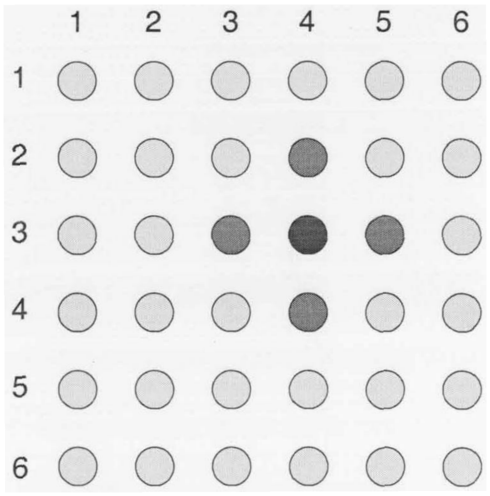


**Figure 4.8. Flipping the sign of each sample affects the coding cost of only five samples**

Thus, in each step in the Metropolis method and in order to compute the new cost, we need to flip one sign and compute the sign coding cost of only the current sample and its four neighbors and compare the result to the case where the sign is not flipped. This operation reduces the computation and the complexity to a very reasonable amount.

In this chapter, we examined the possible strategies for setting the "don't care" bits in both of the scaling-based the Maxshift ROI coding methods. We saw that the best choice for the "don't care" bits in the Maxshift method is to set them to zero. We also proposed a method that exploits the state of the JPEG2000 entropy coder to set the values of the "don't care" bits in the scaling-based method. In the next chapter, we will show the improvements possible with our proposed method versus the default all-zero method.

# CHAPTER 5
# RESULTS

## *Introduction*

The goal here is to identify the improvements possible with our proposed method for choosing the "don't care" bits compared to the all-zero method used in the current JPEG2000 coders. The bit-rate improvement associated with our technique is a function of several parameters. More specifically, it depends upon the size of the region of interest, the scaling factor, as well as the number of zero symbols in the quantized "detail" subbands. The number of zeros, in turn, is affected by the image itself. In this chapter we will provide some results to demonstrate the effectiveness of our method.

## *5.1 Codec design*

The JPEG2000 standard was first introduced as the JPEG2000 part 1 [13], and four extension parts were added to it afterwards. The ROI scaling-based method belongs to the second part of the standard. Unfortunately, as far as the author is aware, only the source code of the first part of the standard has been released to the public, and therefore, we had to implement our own codec in order to test the method.

As the image code-blocks are quantized and entropy coded independently, we only implemented a codec to code and decode an image code-block. In our codec, the image is first partitioned into different subbands using a wavelet transform. We used the KDF 9/7 wavelet in our implementation, which is one of the two types of wavelets supported [34] by the JPEG2000 standard. In the next step, the subbands are partitioned into code-blocks of 32x32 samples and then the codec processes one code-block at a time.

The entropy coding method of the codec, including the quantization method, formation of the contexts, coding passes, etc., is exactly the same as what has been explained in the third chapter. The arithmetic coder that is used in the codec is the famous escape coder implemented in [9]. The quantization step size is 1/256, which is the default value in the JPEG2000 implementation in [34] when the wavelet coefficients are normalized to the range -0.5 to 0.5.

We ran some experiments on the 512×512 Barbara image by selecting 32×32 code-blocks from different parts of the nine "detail" subbands after a 3-level subband decomposition. There were seven magnitude bit-planes and, for simplicity, all bits in the test code-blocks were declared to be in the ROI, which used a scaling factor of six. The number of bits needed by the arithmetic coder is given in Table 5.1, in comparison with the default strategy of always setting the "don't care" bits to "0". The results for all of the code-blocks in Barbara are provided in the appendix. Note that if some of the samples in a code-block belong to the BG, then the improvements with our method should be less, since, statistically, some of the zero-valued samples will belong to the BG in this case.

**Table 5.1. Number of bits to code sample 32×32 code-blocks in Barbara**

| **method** | **subband (subscript indicates the subband level)** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $HH_1$ | $LH_1$ | $HL_1$ | $HH_2$ | $LH_2$ | $HL_2$ | $HH_3$ | $LH_3$ | $HL_3$ |
| always zero | 5,899 | 4,611 | 4,507 | 4,098 | 4,442 | 4,565 | 7,478 | 6,715 | 7,387 |
| proposed | 5,781 | 4,270 | 4,436 | 4,059 | 4,287 | 4,422 | 7,411 | 6,588 | 7,262 |
| improvement | 2% | 7.4% | 1.6% | 1% | 3.6% | 3.2% | 0.9% | 2% | 1.7% |

As mentioned previously, our method works better with higher scaling factors. This situation results from the cost of coding the signs of the "don't care" samples. As shown in Table 5.2, the coding cost for our method is higher than the "always zero" strategy for scaling factors of one and two. However, after the third extra bit-plane is coded, the increase in the cost will be alleviated in our method because of the more skewed MR contexts.

**Table 5.2. Coding cost of a sample 32×32 code-block with different scaling factors and seven magnitude bit-plane (Barbara)**

| method | scaling factor | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| always zero | 3,730 | 4,015 | 4,213 | 4,369 | 4,499 | 4,611 |
| proposed | 3,887 | 4,066 | 4,152 | 4,206 | 4,245 | 4,275 |
| improvement | -4% | -1.3% | 1.5% | 3.8% | 5.8% | 7.4% |

Setting the first don't care bits belonging to the category #2 samples to one, usually results in a small decrease in the bit-rate cost of the code-block. Then after each sign flip in the Metropolis method, the cost decreases to a smaller value, and after a few thousand iterations it will reach to an almost constant value as shown in Figure 5.1. The coding cost of this code-block in the Barbara image with the always zero method is 5531. After setting the MSB of the category #2 don't care samples to one, the cost reduces to 5433. Note that in the Metropolis method, we need to start with an arbitrary set of signs for the "don't care" samples and we used all positive signs in this experiment. The cost value decreases to 5200 after 1300 iterations and to 5185 after

4500 iterations. The initial value for the temperature here is 0.5 and it is decreased linearly to 0.01 with every sign flip attempt.

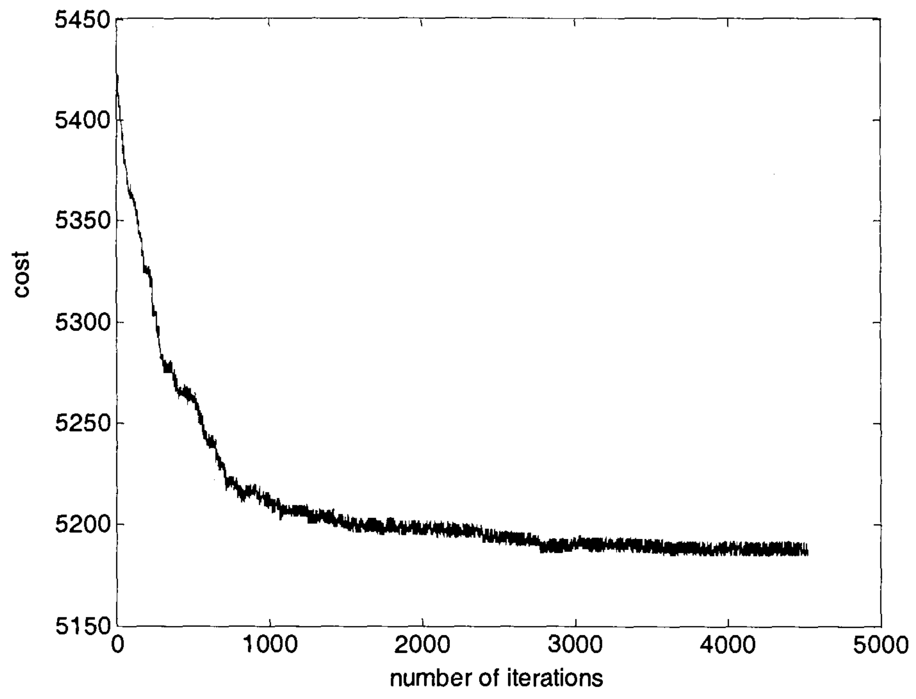


**Figure 5.1. The coding cost reduction with the metropolis method (a sample code-block in Barbara)**

In this chapter, we ran some experiments on different code-blocks of the 512x512 Barbara image as a demonstration of the effectiveness of our method. A more comprehensive set of results is provided in the appendix of the thesis. In the next chapter, we will wrap up the thesis and suggest some future work.

# CHAPTER 6
# CONCLUSIONS

In this thesis we examined the possible strategies to choose the “don’t care” bits in JPEG2000 ROI coding. We showed that in the scaling-based method, the all-zero method which is used in the current JPEG2000 coders is not the best choice in terms of the coding cost of the image code-blocks. We proposed a method to exploit the state of the JPEG2000 entropy coder to choose the values of the bits in a more intelligent way.

The benefits of our method are two fold: the bit-rate is reduced for a given set of block truncation points and, more subtlety, a rate-distortion optimized search for the best set of code-block truncation points may now be able to find a better operating point.

## *6.1 Future work*

As we did not have access to the part 2 of the JPEG2000 source code, we had restrictions in testing the exact improvements possible with our method for different images and ROI shapes. Apart from the ROI shape and image type there are other parameters that affect the performance of our method. These parameters include the type of the wavelet transform, the value of the scaling factor and the quantization step-size. The effects of all these parameters can be measured by implementing the method on the part 2 of the JPEG2000 source code.

The other important future work is to try to reduce the amount of computation needed to find the set of optimum signs for the “don’t care” samples. We proposed the simulated annealing, also known as the Metropolis method for this purpose. We saw in the last chapter that the coding cost of five signs needs to be computed a few thousand times so that the overall cost converges to a reasonably low value. For some applications such as in battery-powered devices like digital cameras, this increase in the complexity and computation might be undesirable. There might be some ways to reduce this added complexity. One interesting possibility is to use the deterministic annealing [23] for choosing the signs.

# BIBLIOGRAPHY

[1] Ameli, J., Vaisey, J. and Jin, T., "Selecting Don't Care Bits in JPEG2000 ROI Coding", *SPIE Conference on Electronic Imaging"*, San Jose, 2004.

[2] Antonini, M., Barlaud, M., Mathieu, P. and Daubechies, I., "Image coding using wavelet transform", *IEEE Trans. Signal Processing*, vol.1, pp.205-221, Apr.1992.

[3] Askelof, J., Larsson, M. and Christopoulos, C., "Region of interest coding in JPEG 2000", *Signal Processing: Image Compression*, vol. 17, pp. 105–111, 2002.

[4] Burrus, C.S., Gopinath, R.A., Guo, H., *Introduction to Wavelets and Wavelet Transforms*, Prentic Hall, New Jersey 1998.

[5] Chiu, E., "Lossy Compression of Medical Ultrasound Images Using Space-Frequency Segmentation" *School of Engineering Science M.A.Sc. Thesis*: Simon Fraser University, 1999, http://www.ensc.sfu.ca/grad/theses

[6] Christopoulos, C., Askelof, J. and Larsson, M., "Efficient methods for encoding regions of interest in the upcoming JPEG2000 still image coding standard" *IEEE Signal Processing Letters*, vol. 7, pp.247-249, Sep. 2000.

[7] Christopoulos, C., Askelof, J. and Larsson, M., "Efficient region of interest coding techniques in the upcoming JPEG2000 still image coding standard," *Proc. IEEE Int. Conf. Image Processing*, vol. 2, pp. 41–44, Sep. 2000.

[8] Christopoulos, C., Skodras, A. and Ebrahimi, T., "The JPEG2000 still image coding system: An overview," *IEEE Trans. Consumer Electron.*, vol. 46, pp. 1103–1127, Nov. 2000.

[9] Davis, G., Baseline Wavelet Transform Coder Construction Kit, Version 0.3, http://www.cs.dartmouth.edu/~gdavis

[10] Gersho, A. and Gray R., *Vector Quantization and Signal Compression*. Norwell, MA: Kluwer, 1992.

[11] Gormish, M., Schwartz, E, Keith, A., Boliek, M. and Zandi, A., "Lossless and nearly lossless compression of high-quality images," *IEEE Trans. Signal Processing*, vol. 45, pp. 62–70, Mar. 1997.

[12] Grosbois, R., Santa Cruz, D. and Ebrahimi, T., "New approach to JPEG2000 complaint region-of-interest coding" in *Proc. SPIE 46th Annu. Meeting, Applications of Digital Image Processing*, vol.. XXIV, San Diego, CA, July 29-Aug.3 2001.

[13] ISO/IEC JTC 1/SC 29/WG 1 (ITU-T SG8) JPEG2000 Part I Final Committee Draft Version 1.0, Mar. 2000.

[14] ISO/IEC JTC 1/SC 29/WG 1 (ITU-T SG8) JPEG2000 Part II Final Committee Draft, Dec.2000.

[15] ISO/IEC, *ISO/IEC 14495-1:1999: Information technology— Lossless and near-lossless compression of continuous-tone still images: Baseline*, Dec. 1999.

[16] Ji, E., "Context-Based Entropy Coding with Spacefrequency Segmentation in Ultrasound Image Compression" *School of Engineering Science M.A.Sc. Thesis*: Simon Fraser University, 2001, http://www.ensc.sfu.ca/grad/theses

[17] Landau, D. and Binder, K., *A guide to Monte Carlo simulations in statistical Physics,* Cambridge University Press, 2000.

[18] Marcellin, M. Gormish, M., Bilgin, A. and Boliek, M., "An overview of JPEG2000," in *Proc. IEEE Data Compression Conf.*, Snowbird, UT, 2000.

[19] Moffat, A., "Arithmetic Coding Revisited.", *ACM Transactions on Information System*, vol.16,No.3, Jul. 1998.

[20] Oppenheim, A. and Schafer, R., *Discrete-Time Signal Processing,* Prentice Hall, 1989.

[21] Pennebaker, W. and Mitchell, J., *JPEG Still Image Data Compression Standard.* New York: Van Nostrand Reinhold, 1994

[22] Rabbani, M. and Joshi, R., "An overview of the JPEG2000 still image compression standard," *Signal Process.: Image Communications.*, vol. 17, pp. 3–48, Jan. 2002.

[23] Rose, K., "Deterministic Annealing for Clustering, Compression, Classification, Regression, and Related Optimization Problems," *Proceedings of the IEEE*, vol. 80, pp. 2210-2239, Nov. 1998.

[24] Santa Cruz, D., Ebrahimi, T., "An analytical study of JPEG2000 functionalities," *Proceedings ICIP-2000*, Sep. 2000.

[25] Santa Cruz, D., Grosbois, R. and Ebrahimi, T., "JPEG 2000 performance evaluation and assessment," *Signal Process.: Image Communications*, vol. 17, pp. 113–130, Jan. 2002.

[26] Santa-Cruz, D., Ebrahimi, T., Askelof, J., Larsson, M. and Christopoulos, C., *JPEG 2000 still image coding versus other standards*, Proceedings of the SPIE's 45th annual meeting, Applications of Digital Image Processing XXIII , vol. 4115, pp: 446-454, San Diego, California, Jul. 2000.

[27] Sayood, K., *Introduction to Data Compression.* Morgan Kauffman Publishers Second edition, 2000.

[28] Sayood, K., *Lossless Compression Handbook*, Academic Press, pp250-270, 2003.

[29] Shannon, C.E., "A Mathematical Theory of Communication", *Bell System Technical Journal*, vol.27, no.3, pp.379-423,1948.

[30] Skodras, A., Christopoulos, C. and Ebrahimi, T., "The JPEG2000 still image compression standard" *IEEE Signal Processing Magazine*, vol. 18, pp. 36-58, Sep. 2001.

[31] Smith, M. and Eddins, S., "Analysis/synthesis techniques for subband image coding," *IEEE Trans. Acoust. Speech Signal Processing*, vol. 38, pp. 1446-1456, Aug. 1990.

[32] Strang, G., Nguyen, T., *Wavelets and Filger Banks*, Wellesley-Cambridge Press, 1996.

[33] Taubman, D. and Marcellin, M., "JPEG2000: Standard for Interactive Imaging", *Proceedings of the IEEE*, Volume: 90 Issue: 8 , pp: 1336 -1357, Aug. 2002.

[34] Taubman, D. and Marcellin, M., *JPEG2000: Image Compression Fundamentals, Standards and Practice.* Norwell, MA: Kluwer, 2002.

[35] Taubman, D., "High Performance Scalable Image Compression with EBCOT" *IEEE Transactions on Image Processing* , vol. 9, No.7, pp: 1158-1171, Jul.2000.

[36] Trees, H.V., *Detection, Estimation, and Modulation Theory.* New York: Wiley, 1968.

[37] Usevitch, B., "A tutorial on modern lossy wavelet image compression: Foundations of JPEG 2000", *IEEE Signal Processing Magazine*, pp22-34, Sep. 2001.

[38] Vaidyanathan, P.P., *Multirate Systems and Filter Banks*, Prentice-Hall, Englewood Cliffs, NJ, 1992.

[39] Vetterli M. and Kovacevic, J., *Wavelets and Subband Coding.* Englewood Cliffs, N.J., Pretice-Hall, 1995.

# APPENDIX A
# EXPERIMENTAL RESULTS

In the tables in this section, the code-blocks in each image subband are numbered like matrix coefficients as shown in Figure A.1.

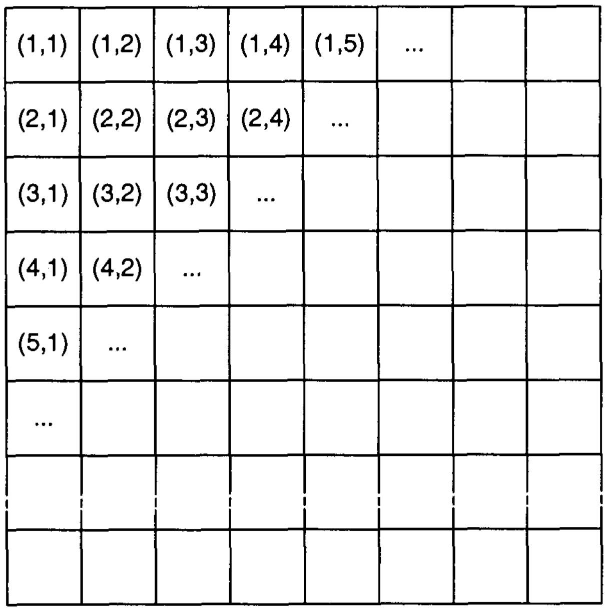


**Figure A.1. Numbering method of code-blocks in an image subband in this section**

**a.z.**: always zero method

**prop.**: proposed method

**imp.**: improvement

**Table A.1. Coding cost of 32x32 code-blocks in the $LH_1$ subband of Barbara**

| subband | a.z. | prop. | imp. | subband | a.z. | prop. | imp. |
|---|---|---|---|---|---|---|---|
| (1,1) | 4557 | 4375 | 4.1% | (1,2) | 4611 | 4275 | 7.4% |
| (1,3) | 3901 | 3816 | 2.2% | (1,4) | 3114 | 3120 | -0.2% |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| (1,5) | 4012 | 3821 | 4.9% | (1,6) | 4636 | 4433 | 4.5% |
| (1,7) | 4783 | 4632 | 3.2% | (1,8) | 5135 | 4993 | 2.8% |
| (2,1) | 4014 | 4080 | -1.6% | (2,2) | 4894 | 4722 | 3.6% |
| (2,3) | 4022 | 3805 | 5.6% | (2,4) | 3440 | 3328 | 3.3% |
| (2,5) | 4405 | 4171 | 5.6% | (2,6) | 5497 | 5302 | 3.6% |
| (2,7) | 5172 | 4946 | 4.4% | (2,8) | 6180 | 5907 | 4.5% |
| (3,1) | 5531 | 5185 | 6.6% | (3,2) | 5022 | 4809 | 4.3% |
| (3,3) | 5119 | 4866 | 5.1% | (3,4) | 2978 | 2943 | 1.2% |
| (3,5) | 5911 | 5581 | 5.8% | (3,6) | 5920 | 5725 | 3.3% |
| (3,7) | 5095 | 4953 | 2.8% | (3,8) | 5821 | 5562 | 4.5% |
| (4,1) | 6107 | 5919 | 3.1% | (4,2) | 6179 | 5922 | 4.2% |
| (4,3) | 4880 | 4764 | 2.4% | (4,4) | 3186 | 3198 | -0.4% |
| (4,5) | 5727 | 5444 | 5.1% | (4,6) | 6390 | 6120 | 4.3% |
| (4,7) | 5928 | 5687 | 4.2% | (4,8) | 5202 | 5025 | 3.5% |
| (5,1) | 7008 | 6909 | 1.4% | (5,2) | 5674 | 5483 | 3.4% |
| (5,3) | 4903 | 4736 | 3.5% | (5,4) | 3877 | 3895 | -0.5% |
| (5,5) | 5277 | 5078 | 3.8% | (5,6) | 5853 | 5671 | 3.2% |
| (5,7) | 6834 | 6620 | 3.2% | (5,8) | 5683 | 5559 | 2.2% |
| (6,1) | 5449 | 5278 | 3.2% | (6,2) | 4604 | 4382 | 4.9% |
| (6,3) | 4515 | 4397 | 2.6% | (6,4) | 5150 | 4993 | 3.1% |
| (6,5) | 6875 | 6721 | 2.3% | (6,6) | 6564 | 6342 | 3.4% |
| (6,7) | 6206 | 5998 | 3.4% | (6,8) | 6037 | 5887 | 2.5% |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| (7,1) | 4250 | 4129 | 2.9% | (7,2) | 4566 | 4355 | 4.7% |
| (7,3) | 4626 | 4481 | 3.2% | (7,4) | 4428 | 4344 | 1.9% |
| (7,5) | 6231 | 5999 | 3.8% | (7,6) | 8066 | 7813 | 3.2% |
| (7,7) | 6178 | 5994 | 3.0% | (7,8) | 5613 | 5437 | 3.2% |
| (8,1) | 4766 | 4560 | 4.4% | (8,2) | 5243 | 5062 | 3.5% |
| (8,3) | 5205 | 5046 | 3.1% | (8,4) | 5406 | 5305 | 1.9% |
| (8,5) | 4672 | 4616 | 1.2% | (8,6) | 7021 | 6821 | 2.9% |
| (8,7) | 6568 | 6423 | 2.2% | (8,8) | 4936 | 4902 | 0.7% |

average improvement: 3.3%

# APPENDIX B
# IMAGES IN THE EXPERIMENTS

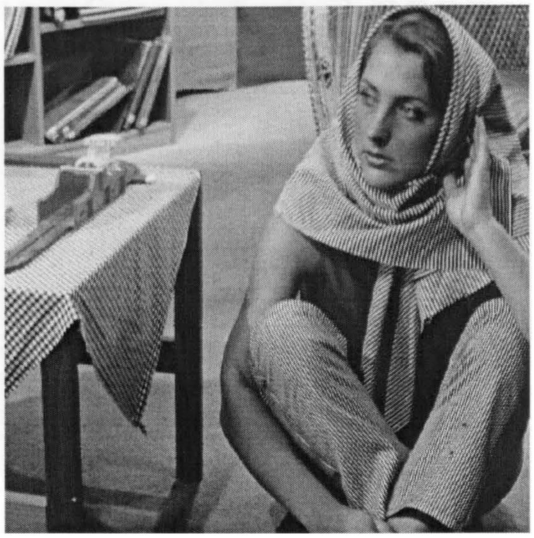


Figure B.1. Barbara 512x512

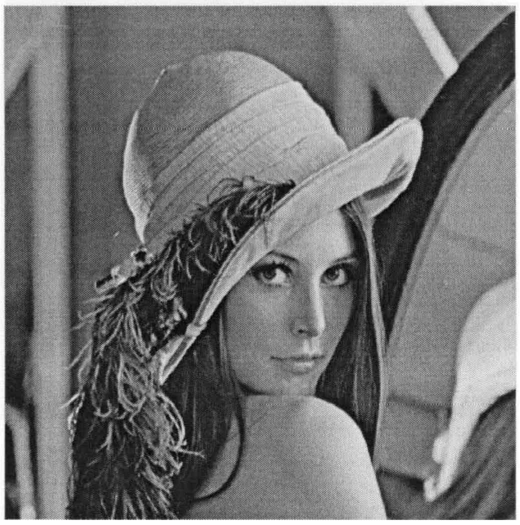


Figure B.2. Lena 512x512

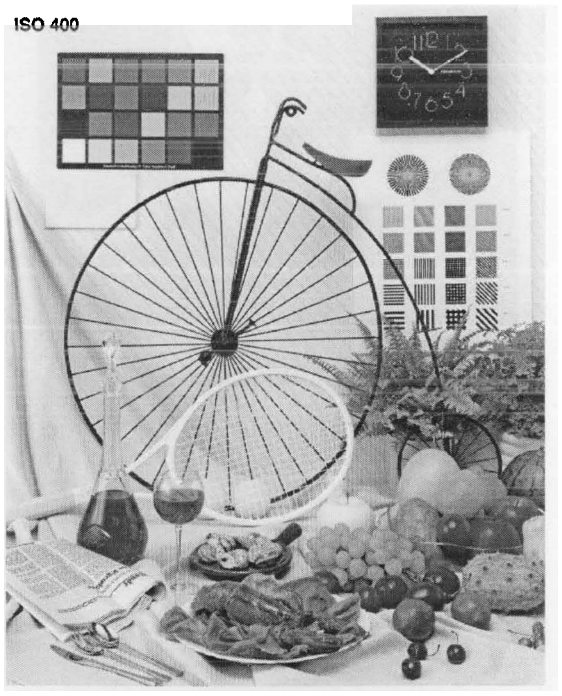


Figure B.3. Bike (2048X2560)

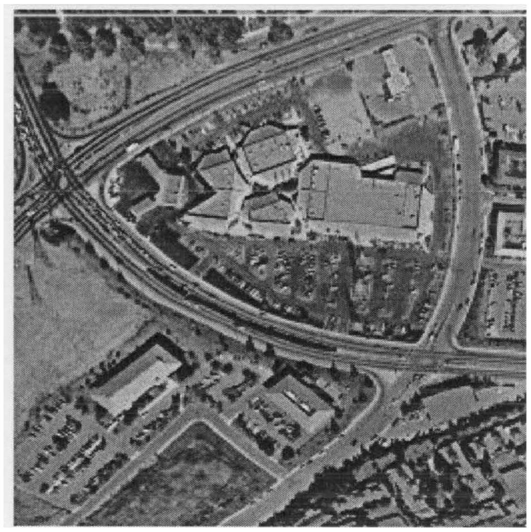


Figure B.4. Aerial (256X256)

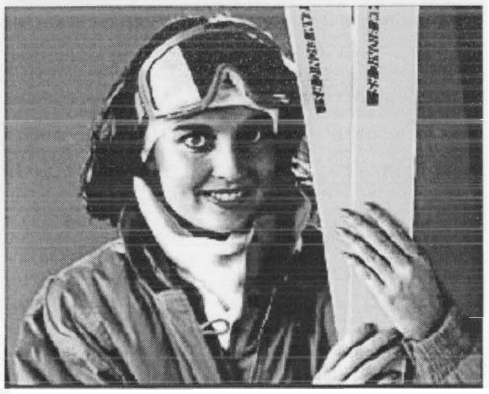


**Figure B.5. Ski (720x576)**