WAVELET IMAGE COMPRESSION RATE DISTORTION OPTIMIZATIONS AND COMPLEXITY REDUCTIONS

by

Christos Chrysafis

- -

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
in Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(ELECTRICAL ENGINEERING)

February 2000

Christos Chrysafis

Abstract

Compression of digital images has been a topic of research for many years and a number of image compression standards has been created for different applications. The role of compression is to reduce bandwidth requirements for transmission and memory requirements for storage of all forms of data. While today more than ever before new technologies provide high speed digital communications and large memories, image compression is still of major importance, because along with the advances in technologies there is increasing demand for image communications, as well as demand for higher quality image printing and display.

In this work we focus on some key new technologies for image compression, namely wavelet based image coders. Wavelet coders apart from offering superior compression ratios have also very useful features such as resolution scalability, i.e. they allow decoding a given image at a number of different resolutions depending on the application. We start by presenting in a simple manner a collection of tools and techniques to apply wavelet filtering in images, ranging from boundary extension to fast implementations. We continue by exploiting the use of rate distortion theory in trying to achieve very high compression ratios for a wavelet coder. The results we report are among the best in the literature. We apply rate distortion theory on a per coefficient basis combining a theoretical analysis with an online probability estimation.

After presenting the rate distortion algorithm we focus on techniques to reduce the complexity of generic wavelet coders. One of the main contributions of this work is the ability to compress an image with a wavelet coder without the need to buffer the complete image. The memory requirements of the proposed approach are

orders of magnitude lower than other algorithms proposed up to date, which would require buffering of the entire image. This limited low memory implementation is key in enabling widespread commercial use of wavelet image coding and has been incorporated in the informative part of the upcoming JPEG2000 standard.

Acknowledgments

I would like to thank my advisor Antonio Ortega for the invaluable guidance he provided me during the whole process of my graduate studies, our conversations and the feedback I got from Antonio were determining factors in completing this work.

I would like to thank my undergraduate thesis advisor Michael Strintzis for introducing me to the area of Image compression and the theory behind the wavelet transforms. He was the one to motivate my research interests in the area, I will always be grateful to him.

I would like to thank my colleagues in Hewlett Packard Laboratories Amir Said and David Taubman for sharing their expertise with me on a lot of the practical issues on Image Compression, as well as Alex Drukarev for providing the environment and the means for me to work on my thesis while at Hewlett Packard Laboratories.

I would like to thank C.C Jay Kuo, Richard Leahy and Larry Goldstein for serving as committee members in my defense examination.

Last, but not least, I would like to thank my family. Their support has always been unlimited through out the years. They have given me much more than the luxury to write this thesis. They have provided me with tools and have taught me how to approach problems, how to deal with people and respect hard work. They have taught me how to adapt myself to the challenges of life and be focused.

Contents

List of Tables

List of Figures

List of Abbreviations

DCT - Discrete Cosine Transform

DFT - Discrete Fourier Transform

DPCM - Differential Pulse Code Modulation

DSP - Digital Signal Processor

DVD - Digital Versatile Disc

FBI - Federal Bureau of Investigation

FFT - Fast Fourier Transform

FIR - Finite Impulse Response filter

IDCT - Inverse Discrete Cosine Transform

IDFT - Inverse Discrete Fourier Transform

IIR - Infinite Impulse Response filter

ISO - International Standardization Organization

JBIG - Joint Bi-level Image experts Group

JPEG - Joint Pictures Experts Group

IWT - Inverse Wavelet Transform

KLT - Karhunen Loève Transform

LOT - Lapped Orthogonal Transforms

LPTV - Linear Periodically Time Variant System

LSI - Linear Shift Invariant System

LTV - Linear Time Variant System

MPEG - Moving Pictures Experts Group

MSE - Mean Square Error

PCM - Pulse Code Modulation

PSNR - Peak Signal to Noise Ratio

SIMD - Single Instruction Multiple Data

QMF - Quadrature Mirror Filter

SNR - Signal to Noise Ratio

WT - Wavelet Transform

Chapter 1

Introduction to Image Compression

Contents

Compression of digital images plays a key role in image storage and transmission. In this chapter we will give a brief introduction to general image compression, and wavelet image compression in particular.

1.1 Bandwidth, Memory, Computations and Compression

The advances in technology the last decades have made the use of *digital images* a commonality in everyday life. While the usefulness of digital images in communicating information is not questionable, the cost of storing and transmitting images is much larger compared to storage and transmission of text, so that for example image databases require more storage than document archives. While series of still images can form a video sequence, still images are treated in a different way than video images. In video we are dealing with a three dimensional space, with the third dimension being time. Video images are usually of limited size compared to still images which

might be orders of magnitude larger, but video images need to be transmitted at a certain rate for real time display.

The amount of data transmitted through the Internet doubles every year, and a large portion of that data is images. Reducing the bandwidth needs of any given device will result in significant cost reductions and will make use of the device more affordable. Magnetic hard discs (HD), CD's, DVD's of larger capacity are released every year, in response to greater demand for storage of digital data. The use of digital cameras, both still and video, is becoming more widely accepted. Image compression offers ways to represent an image in a more compact way, so that one can store more images and transmit images faster. The advantages of image compression come at the expense of numerical computations, and therefore we can trade off computations for storage or bandwidth. Before storing or transmitting an image we process it in such a way that will require fewer bits for its representation.

Technologies addressing increase of bandwidth of a given channel, increase of the size of a memory device and increase in the number of computations per second performed by a CPU, have been developing simultaneously without any sign of saturation. Memory sizes are only limited by the physical size of the device, if we place two discs one next to the other we double the capacity of the system. The same is true for computations, as two processors have twice the computational power of a single processor. However the problem of just duplicating a processor or a memory unit is bandwidth. If we duplicate a processor we need to make sure that the two processors can communicate with each other at high enough rates and we can access all memory units through the same bus. It seems that bandwidth, memory and computations are closely interrelated, and which one is more important depends on the specific application. Compression algorithms can be thought of as means to *trade off computation, memory and bandwidth* as seen in Figure 1.1. A compression algorithm tries to offer the best trade off among the three factors, for a given application. For example if we are limited in terms of memory we can spend more computational time to compress the image and make sure it fits into the given memory size. If we are computation limited we can store the image as is with no compression or with limited

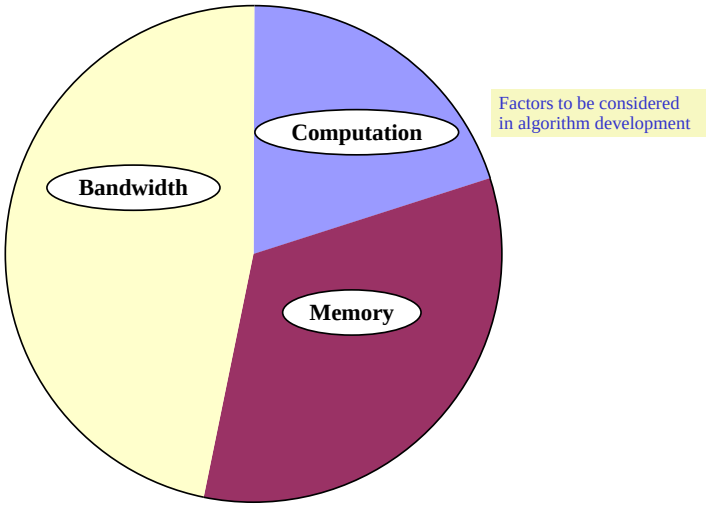


Figure 1.1: Complexity trade offs. Memory, computations and bandwidth are of major importance on algorithm development.

compression with a simple compression algorithm.

1.2 Historical background on Image Compression and Compression Standards

Image compression algorithms have been the subject of research both in academia and industry for many years. Today, while significantly improved algorithms have been achieved and compression performance is better than a decade ago, there is still room for new technologies. The first widely adopted international image compression standard was JPEG [49, 71] which was introduced in the late eighties. JPEG is based on DCT followed by entropy coding based on either Huffman coding [46, 84, 90] or binary arithmetic coding [58, 80, 90, 105]. It has been widely used from the printing industry to Internet applications. For example all high-end printers compress the image to be printed before they actually send it to the print engine, and most images transmitted through the internet are JPEG compressed. JPEG is intended for continuous tone images of more than one bit depth. Algorithms for binary images

work in a different way, JBIG-1 and JBIG-2 are the standards covering this area. JPEG and JBIG are part of other standards, such as facsimile transmission standards [48], the FlashPix file format [42], the TIFF file format [1], and page description languages like PDF.

In recent years researchers have been using the *discrete wavelet transform* in compression systems. In 1983 Burt and Anderson [7] were the first to introduce multiresolutional analysis in image compression. While their approach seemed counter intuitive at the first glance, given that it increased the number of samples to be coded, their results were promising. Mallat [63] was the first to point out the connection between multiresolutional analysis and the wavelet transform. Daubechies [31] has studied the discrete wavelet transform and has made it a popular tool in the scientific community. Some of the first papers on wavelet image compression [3,83,85,93] presented excellent compression performance results and gave a lot of intuition behind the use of the wavelet transform in image compression. A number of researchers [107] have described the same principles of wavelet image compression by looking at it from a system perspective, using filter banks, and subband decomposition, and refer to wavelet coding as subband coding. Subband coding and wavelet coding essentially refer to the same system, the description of the system is from a slightly different point of view. In subband coding the emphasis is in the frequency domain unlike wavelet coding where the emphasis is in the space domain [98].

Numerous organizations have been using wavelet compression algorithms as their own, internal compression standards. An example is the FBI [44] where there was a need for storing large data-bases of finger-prints and JPEG did not satisfy their requirements. Only more recently was there a decision by the ISO to standardize a wavelet coder in JPEG2000 [59]. Until recently [17] all proposed wavelet coders would require buffering the whole image, computing the wavelet transform in a frame buffer, application of a quantization on the wavelet coefficients and entropy coding of the generated indexes. Wavelet coders could indeed perform very well, but their complexity was well above the complexity of the current JPEG standard. Complexity issues on wavelet coders have only been emphasized by researchers in the last few

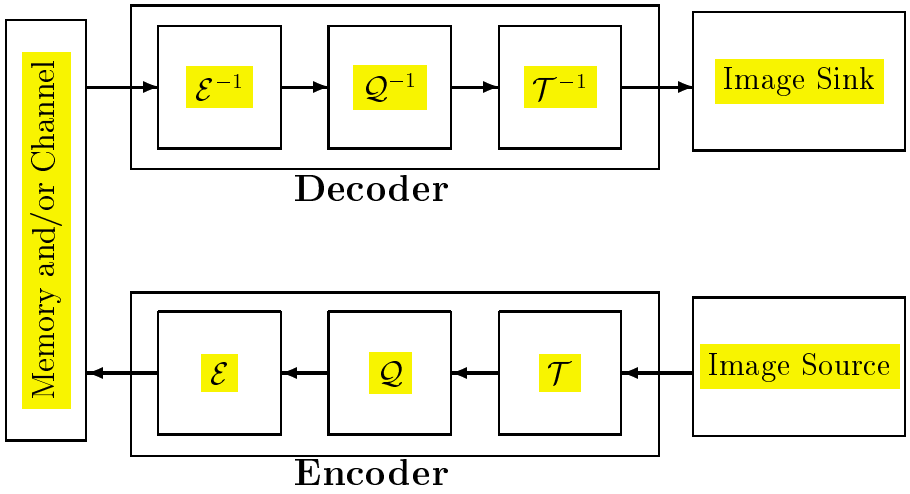


Figure 1.2: Generic Image Encoder and Decoder. $\mathcal{E} \rightarrow$ Entropy Encoder, $\mathcal{Q} \rightarrow$ Quantizer, $\mathcal{T} \rightarrow$ Transform. $\mathcal{E}^{-1} \rightarrow$ Entropy Decoder, $\mathcal{Q}^{-1} \rightarrow$ Inverse Quantizer, $\mathcal{T}^{-1} \rightarrow$ Inverse Transform

years, as the JPEG2000 standard process has exposed the complexity of wavelet coders. Our work [17] was among the first to address low memory wavelet coding.

There are two major classes of image compression algorithms, namely *lossy* and *lossless* algorithms. Lossless algorithms preserve the image data, i.e. original and reconstructed images are exactly the same. In *lossy* image compression original and reconstructed images may or may not be identical in a strict mathematical sense, but to a human observer they may look the same, so the goal is to achieve compression that is *visually lossless*. Both *lossy* and *lossless* compression algorithms are used today in a broad range of applications, from transmitting satellite images, to web browsing to image printing and scanning. With lossy compression algorithms we can achieve significantly larger compression ratios compared to lossless algorithms.

1.3 A Generic Image Compression System

Most image coders consist of *transform*, *quantization* and *entropy coding*, as seen in Figure 1.2. The transform block is in general a reversible operation, i.e. a cascade of a forward and an inverse transform blocks is the identity operation. $\mathcal{T}\mathcal{T}^{-1}\{arg\} =$

$\mathcal{T}^{-1}\mathcal{T}\{arg\} = arg$. Quantization on the other hand introduces some loss. The quantizer usually maps an interval of real numbers to a single index, constituting the only lossy part of the coding system i.e., $\mathcal{Q}^{-1}\mathcal{Q}\{arg\} \neq arg$. It is lossy because the knowledge of an index is only enough to give us the corresponding interval in the real line but not the exact number in the real line. The entropy coder is the building block responsible for compression, it maps more frequent indexes to small codewords and less frequent indexes to larger codewords. It is also a reversible operation i.e., $\mathcal{E}^{-1}\mathcal{E}\{arg\} = arg$. A large portion of the computational complexity of a compression system is due to the entropy coding part of the system. More compression usually translates to higher computational complexity. In general arithmetic [58] and Huffman coding [46] are the most common choices. Arithmetic coding is intended for *high-end* applications where complexity is not a concern, but compression performance is, while Huffman coding is intended for *low-end* applications where simplicity is more important. Typically the most memory intensive element is the transform, as will be explained in chapter 5. Quantization on the other hand is a much simpler process than the transform or the entropy coder. For the JPEG decompressor for example with optimized C code, the Huffman decoding contributed to 60% of the execution time with DCT contributing 30% and quantization only 6% [1] for a Pentium-II processor. When talking about quantization we refer to scalar quantization. While we could also use vector quantization the complexity would be much higher and therefore we only focus on scalar quantization.

1.4 The Discrete Wavelet Transform

The one-dimensional wavelet transform is a particular linear transformation of a signal. Let $\mathbf{x}$ be our input signal and $\mathbf{y}$ be the output signal. Then the input output

[1] The remaining 4% was spend on function calls and marker parsing.

relation for a generic linear transform is:

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{M}_0 \\ \mathbf{M}_1 \end{bmatrix} \mathbf{x} = \mathbf{M}\mathbf{x}, \tag{1.1}$$

where, in the case of two channel wavelet transform, both matrices $\mathbf{M}_0, \mathbf{M}_1$ are *sparse* with non-zero elements only across the diagonal region. Moreover each row of $\mathbf{M}_0, \mathbf{M}_1$ is a shifted version of an other:

$$\mathbf{M}_i = \begin{bmatrix} .. & .. & .. & .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \\ .. & 0 & a_{k-1}^{(i)} & a_{k-2}^{(i)} & .. & a_1^{(i)} & a_0^{(i)} & 0 & 0 & .. & .. & .. & .. \\ .. & .. & 0 & 0 & a_{k-1}^{(i)} & a_{k-2}^{(i)} & .. & a_1^{(i)} & a_0^{(i)} & 0 & 0 & .. & \\ .. & .. & .. & .. & 0 & 0 & a_{k-1}^{(i)} & a_{k-2}^{(i)} & .. & a_1^{(i)} & a_0^{(i)} & 0 & .. \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \end{bmatrix}, i = 0, 1 \tag{1.2}$$

Two channel wavelet transforms generate two subsequences $\mathbf{y}_0$, $\mathbf{y}_1$ from any one given sequence $\mathbf{x}$. The wavelet transform is a *critically sampled* transform, i.e., the number of samples at the input is the same as the number of samples at the output. The size of the output vector $\mathbf{y}$ is equal to the size of the input vector $\mathbf{x}$, and the size of each one of $\mathbf{y}_0$ and $\mathbf{y}_1$ is half the size of $\mathbf{x}$. The wavelet transform is a reversible transform and the inverse transform usually has the exact same structure as the forward transform. Some wavelet transforms are *orthogonal*, but the most useful class of wavelet transforms for image compression is the class of so called *bi-orthogonal* wavelet transforms [88]. Bi-orthogonal transforms are the only transforms that can have the *linear phase* property [97], which turns out to be very useful as explained in chapter 2. There are two reasons for the popularity of these transforms, one is the *fast implementations* possible by using filtering operations as seen in Figure 1.3, the other is the *multiresolutional representations* they allow. An interesting property of the wavelet transform is the following: assume that $\mathbf{x}$ is a signal that contains mostly low pass information, then $\mathbf{y}_0$ will also have low pass information and the "shape" of $\mathbf{y}_0$ will resemble that of $\mathbf{x}$, unlike the $\mathbf{y}_1$ samples which will capture the detail

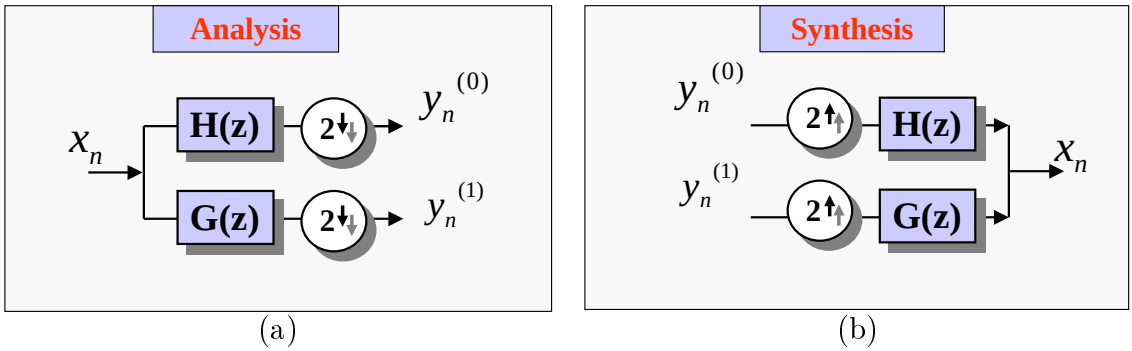
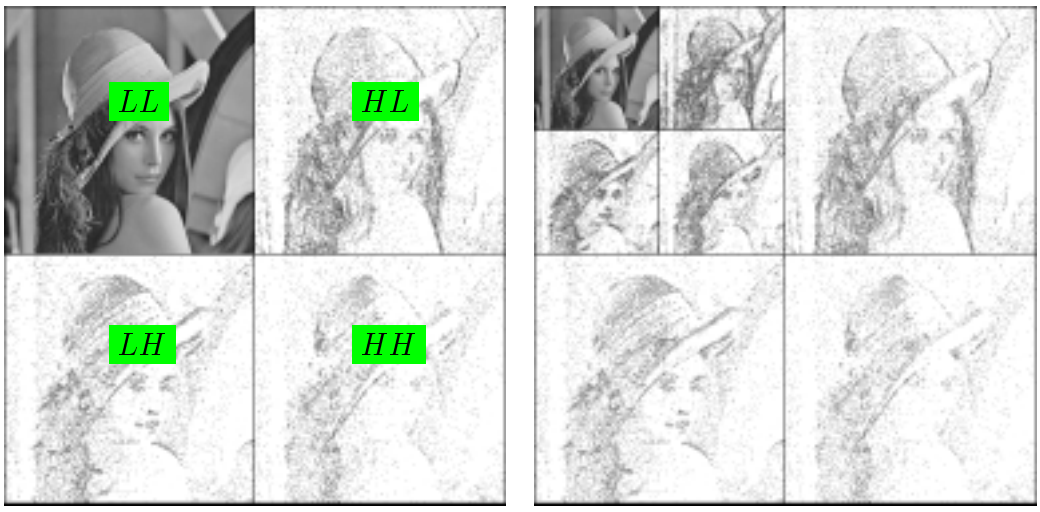


Figure 1.3: (a) Two channel analysis filter bank, (b) two channel synthesis filter bank.

information of the signal $\mathbf{x}$. Referring to 1.3, half of the output ($\mathbf{y}_0$) is obtained by filtering the input with filter $H(z)$ and down-sampling by a factor of two, while the other half of the output ($\mathbf{y}_1$) is obtained by filtering the input with filter $G(z)$ and down-sampling by a factor of two again. $H(z)$ is a low pass filter, while filter $G(z)$ is a high pass filter.

We will not discuss the 1D wavelet transform in detail, and refer to standard texts [88, 97, 98] for more information. Instead we will mention briefly a few interesting properties of the 2D wavelet transform. In two dimensions we usually apply filtering both horizontally and vertically. Filtering in one direction results in decomposing the image in two "components". The total number of "components" we have after vertical and horizontal decompositions is four. From now on we will refer to these "components" as *image subbands*, LL,HL,LH,HH. One of the four subbands, the LL subband, will contain low pass information, which is essentially a low resolution version of the image. Subband HL will contain low pass information vertically and high pass information horizontally, and subband LH will contain low pass information horizontally and high pass information vertically. Finally, subband HH will contain high pass information in both directions. From Figure 1.4(a) we see that subband LL is more important than the other 3 subbands, as it represents a coarse version of the original image. The *multiresolutional* features of the wavelet transform have



(a) (b)

Figure 1.4: (a) One level wavelet transform in both directions of a 2D signal. The LL subband contains the most information about the image, while all other subbands constitute the high resolution information for the image. (b) Two levels of wavelet transform in both directions.

contributed to its popularity. We can encode subband LL alone and add to the encoded bit stream information about the subbands HL,LH,HH. Then the decoder is responsible to decide whether to decode only LL or all subbands.

1.4.1 Compressing Wavelet Coefficients

Subband data are usually real numbers, therefore they require a significant number of bits in their representation, and do not directly offer any form of compression. However by looking at the subband data we find out that the majority of the coefficients in the high frequency subbands are very small. Moreover there is little if any correlation between coefficients in the subband data[2].

The first approach towards compressing wavelet data is to apply a quantizer in all the coefficients, where *uniform* and *dead-zone* quantizers are the most common

[2]An exception is the top level in the decomposition, i.e. the low pass (LL) band.

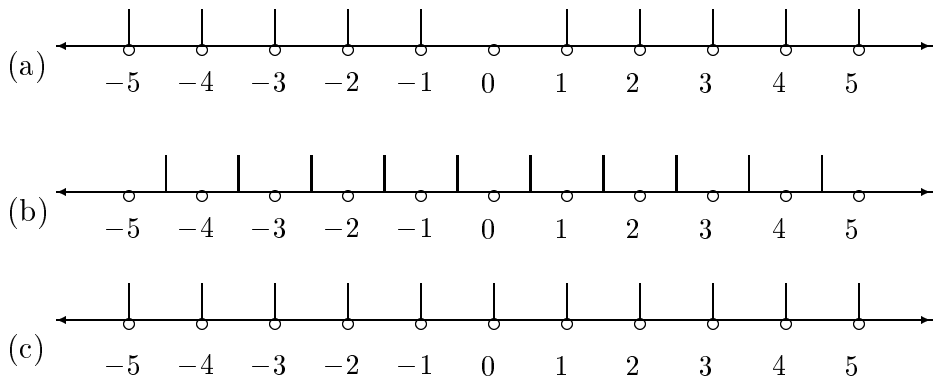


Figure 1.5: (a) Dead-zone quantizer, (b) Uniform quantizer with symmetry around zero, (c) Uniform quantizer centered around zero. The circles "∘" represent numbers on the real axis, while the small vertical lines "|" denote partitions for quantization bins.

choices, as seen in Figure 1.5. After quantization we select an entropy coder to compress the quantization indexes. Little or no gain can be achieved by using linear prediction on the subband data. However context based classification is always possible even if prediction is not. Almost all popular coders [3, 16, 17, 83, 85, 93] define techniques to classify the subband data into different *sets* or *classes* based on past encoded information. Then it is possible to take advantage of the different statistics in each class. There exists some cross-correlation between the same spatial locations in subbands LH,HL,HH. While exploiting this for of correlation usually costs a lot in terms of complexity it can be used to achieve better compression performance, as we will discuss later in the thesis in chapter 4.

1.5 Image Quality Metrics

As discussed above, in lossy compression the reconstructed image is not the same as the original image, so that individual pixels values may be different from the values in the original image. In order to measure the quality of the reconstructed image we need to introduce a quality metric, ideally one that is correlated to visual appearance. Clearly if original and reconstructed images cannot be distinguished by the human eye

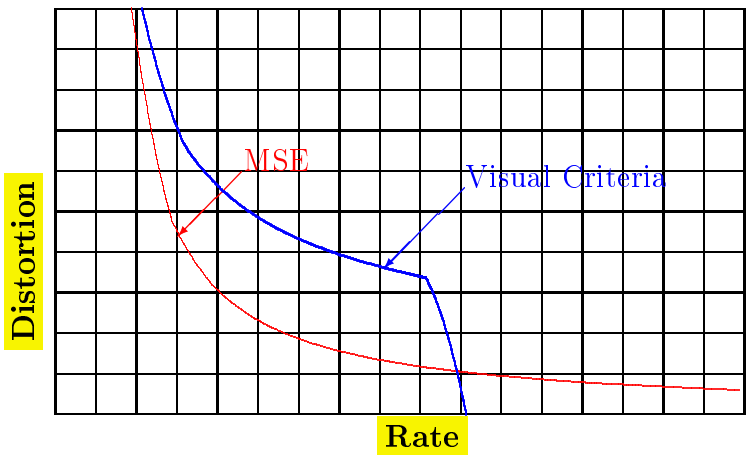


Figure 1.6: Rate distortion curves for MSE and visual appearance metrics. For high enough rates there is no visual distortion. When MSE is sufficiently small the original and decoded images are perceptually indistinguishable.

then our compression algorithm performs very well. Definition of perceptual metrics is a very hard problem and it is difficult to find the point in the rate distortion curve in Figure 1.6 for an image where we transition from visually lossless to visually lossy. This transition point differs, depending up on the viewing angle, the viewing distance the display medium and the viewer himself. Instead, the peak signal to noise ratio (PSNR) between the original and the reconstructed image is a common choice. PSNR is a function of the MSE (Mean Square Error) and is defined as follows:

$$PSNR \triangleq 10 \log_{10} \frac{MAX^2}{MSE}, \tag{1.3}$$

Where MAX is the maximum possible intensity value in the image. For example for eight bit depth images $MAX = 2^8 - 1 = 255$, and for images with d bits depth $MAX_d = 2^d - 1$.

The PSNR metric is the most popular metric of image quality used in compression. Occasionally pure MSE or SNR numbers have been used in testing algorithms, and in some cases the L_∞ norm has also been used. That is, the maximum error between original and reconstructed image is used as a distortion metric. However for convenience PSNR will be the only metric used in the thesis.

1.6 Bit stream features

There are certain features in an image coder that are of interest, such as for example the ability of the coder to decode different resolutions of the image, a property defined as *resolution scalability*. Here by resolution we mean the size of the reconstructed image. We can classify all resolution scalable coders into two different classes, namely *resolution progressive* coders and *resolution parsable* coders. A resolution progressive coder is a system where the decoder can reconstruct a lower resolution of the image by just decoding the first portion of the bit stream, the resolution parsable term refers to the system that can reconstruct a lower resolution of the image by going through the bit stream and extracting only certain portions of it. In both cases no complete decoding of the whole bit stream is needed.

Image coders can be further classified into *SNR progressive* and *SNR parsable* coders. Instead of an intermediate resolution we reconstruct the full resolution image, but we are interested on image quality not image resolution. By decoding more and more bits we can get a higher quality image (higher SNR.) Another very interesting property of some coders is the ability to decode only certain regions of an image, the so called *random access* property. A coder can support decoding of arbitrary shaped regions or of any rectangular shaped region, like for example the coder in [9]. Region of interest encoding and decoding is also related to random access. For region of interest encoding we only encode an arbitrary shaped region of the input image, and discard the rest of the image, that does not fall into the region of interest.

Resolution scalability can be achieved by encoding whole subbands one after the other, without interleaving bits from different subbands [16,19]. SNR scalability can be achieved by distributing encoded bits from one subband into the whole bit stream in an optimal way as done for example in [9,83,92]. Random access can be achieved by independently encoding portions of the whole image [42]. Wavelet tree coders like [83,85] can be modified to support random access. All image coders can support random access by tiling the image and encoding each tile independently of the rest.

1.7 Rate Control

Since image compression is usually a non real time operation, it would appear that compression does not have the rate control problems of video compression. In video we need to transmit image frames in real time. For example we might need to compress at the constant rate of 30 frames per second and thus we need to design a rate control algorithm to make sure that output buffers do not overflow. In still image compression rate control is not needed when dealing with small images. But when compressing larger images rate control might be as important as in video applications. Many devices, such as high quality printers and scanners, are required to compress an image with a certain compression ratio. An input image is fed into the compression algorithm and the compressed bit stream is meant to be stored in a fixed size memory. We need to make sure that under the worst case scenario the compressed bit stream will fit into the fixed size memory. In practice the compressed bit stream may occupy only a portion of the physical memory. By aiming at utilizing the whole memory, we would have the highest quality images, but we would run the risk of buffer overflow, in which case we would lose part of the image. The best solution to the problem is to be able to control our rate as we encode the image. There are many different solutions to the problem (see for example [92]). Most approaches are based on *bit-plane* coding as will be discussed later

Rate control does not apply to lossless compression scenarios, where we cannot adjust in any way the size of the compressed bit stream. The compression ratio will depend upon the image itself and the algorithm we are using and we cannot force a minimum compression ratio. For example for lossless compression of white noise we cannot achieve any compression at all.

1.8 Color Spaces and Digital Imaging

Color spaces are very important in digital imaging and have different effects in image compression algorithms [38, 53, 71]. All natural images can be represented by three

components. Some of the most common color spaces are RGB, Lab, YC_bC_r, YUV, CMY, $CMYK$[3]. Color spaces can be separated into two different categories, *device dependent* and *device independent* spaces. Device dependent spaces are such that the values of all the components are set with respect to a given device, while device independent color spaces refer to color spaces where the values in all components are set with respect to a given standard [38,53,66] and do not depend on the characteristics of any specific input or output device. All input devices like scanners and digital cameras provide color data in a device dependent color space, and the data is then converted to a device independent color space by a *device dependent transformation*. Device independent data can go through any image processing algorithm, including compression and decompression. When data has to be sent to an output device like a printer or a monitor, device independent data is converted to a device dependent color space. It is alway preferable to compress data in a device independent color space, so that we can output the compressed bit stream to any output device without the need to know where the data came from. The most common technique for compression of color images is to use a color space where there is not much correlation between the three components, as for example Lab. A common choice is to also sub-sample the chrominance components by a factor of two in each dimension and compress each component independently of the others. This approach has been used within the JPEG standard and will also be used in the upcoming JPEG2000 standard. There is some correlation between different components in an image and it is possible to get better compression results by jointly compressing all three components, but such an approach has not been proved useful in practice for two reasons:

1. We do not expect significant gains by joint coding of all components. When encoding each color component separately, the luminance component requires many more bits to encode than the two chrominance components. A color transform decorrelates the three color components, so there is almost no linear

[3]Color spaces with more than three components are mainly used for printing.

correlation among the three components. If we try to jointly encode all components we can only reduce the number of bits slightly for the two chrominance components, which already represent a very small portion of the bit stream.

2. In some cases, e.g., block and white printing and display it is desirable to decode only the luminance component of an image and thus it is convenient to be able to decode luminance independently of chrominance.

When evaluating compression algorithms we will concentrate on gray scale images because it is much easier to compare results and draw conclusions. MSE is the most common metric of image quality when working with gray scale images. Dealing with color images MSE is not so informative of the image quality since it depends on the color space and a lot of color spaces are *non linear*[4]. Compression algorithms for color images need to be compared based on visual criteria. We will not discuss color spaces any more in this thesis, instead we will concentrate on gray scale images and compare results based on MSE.

1.9 Overview and contributions of the thesis

In chapters 2 and 3 we introduce the reader to the problems that will discuss later in the thesis. They are of fundamental importance to understanding the material in chapters 4 and 5. The material we present in chapters 2 and 3 applies to any wavelet coder. In chapter 2 we discuss the implementation of the wavelet transform, on finite support signals as those encountered in image compression. The wavelet transform is just a mathematical tool and is intended for infinite support signals. Even though special treatment at the borders of the image has been studied by a numbers of researchers, there is no comprehensive reference for practical implementations. We provide a complete analysis of all possible cases when dealing with the image boundaries. In chapter 3 we discuss entropy coding techniques used in wavelet based

[4] *Nonlinear color spaces* are color spaces that are derived from standard RGB with a non linear mapping.

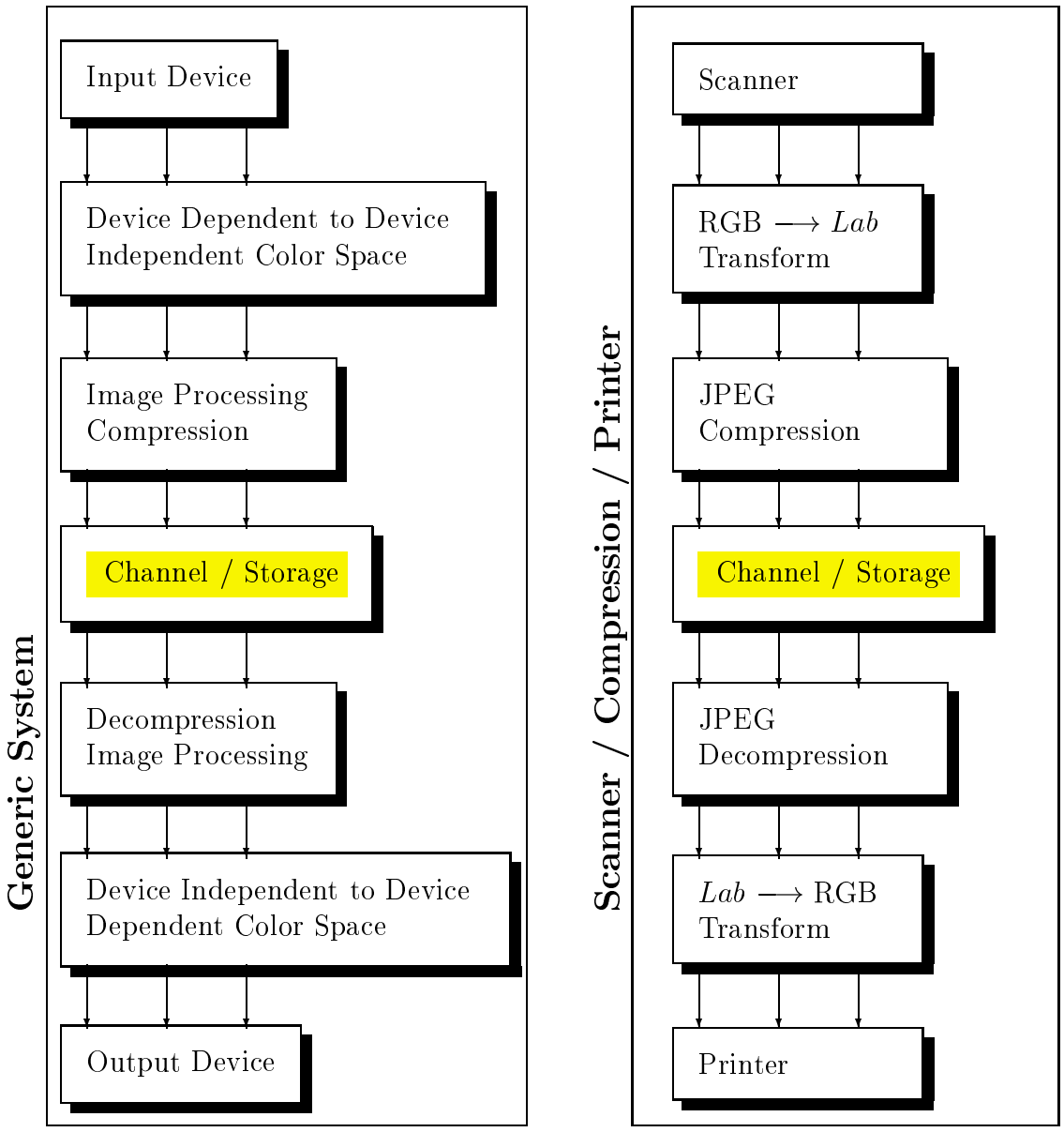


Figure 1.7: Digital Imaging path. On the left we see a generic system for digital image acquisition and consumption, on the right we see the same path for the transmission of an image from the scanner to the printer.

image coding with emphasis on arithmetic coding, and the role of adaptivity.

The key contributions of this work are in chapters 4, 5 and 6 . In chapter 4 we focus on rate distortion performance, with out purpose being to achieve as good quality as possible for a given bit rate. This will serve as a benchmark for other coders we consider in this dissertation. The key contribution of this work is the use of *rate distortion theory* to jointly quantize and entropy encode a subband of wavelet coefficients. In chapter 5 we propose a novel way of applying the wavelet transform in image compression applications where we *minimize the memory needed for wavelet transform and compression*. This low memory wavelet transform has been adopted by the JPEG2000 committee and it is highly likely that it will become part of the evolving standard. A key disadvantage of past wavelet coders was their excessive memory requirements. In order to compress an image these algorithms would always need to buffer the whole input image in some form, either the original image or the wavelet coefficients of the whole image. In chapter 5 we present a complete solution to the problem of excessive memory requirements. The memory needs with our proposed approach are proportional to the width of the image and the filter length, unlike other approaches that need to buffer the whole image. Moreover we combine the low memory wavelet transform with a simple entropy coder in order to demonstrate that such and approach can indeed be useful in a practical compression scheme. In chapter 6 we refine our results from chapter 5 by using lifting structures. Lifting structures lead to lower memory as well as to lower computational complexity as compared to convolution implementations. We also elaborate on the advantages of lifting structures when dealing with boundary signal extensions. Lifting implementations have been studied only recently and more research in the area is needed.

Chapter 2

Finite Support Wavelet Transforms

Contents

The discrete wavelet transform in equation (1.2) can only be applied directly to infinite support signals, as the matrix in equation (1.2) is *banded* [88], that is it has infinite support. However in all imaging applications our input signal has finite support. Since *causality* is usually not an issue in image processing, special treatment of the image boundaries can be used in dealing with the finite support. Several researchers in the signal processing community have addressed processing at the boundaries when filtering finite length signals [4, 6, 64, 65, 87].

There are two ways to deal with a finite support signal, we can *extend* the signal so that it has infinite support [87] or we can modify the matrices in (1.2), to have *finite dimensions* [43]. While both approaches have their merits, for image compression the best choice is the one that decorrelates the input signal in the best possible way, while preserving features such as orthogonality and perfect reconstruction.

2.1 Boundary Treatment

2.1.1 Zero padding

When filtering we can assume that the pixel values of an image outside its bounds are *zero*. This will cause two major problems. If we assume critical sampling[1], that is the number of input samples is equal to the number of output samples, the wavelet transform will not be *perfect reconstruction*. The lower resolution subbands very often happen to be of size less than 16×16, in this case a small error in the boundaries will translate to a much larger error in the full resolution image. If we can afford not to have critical sampling, nonzero wavelet coefficients will appear outside the image boundary. Perfect reconstruction will be possible by using those additional samples, that is the number of output coefficients will be larger than the number of input coefficients.

A second problem is that by assuming a *zero padding* at the boundaries we are introducing a lot of high frequencies that do not exist in the original image. Image compression algorithms are based on the assumption that most of the energy in an image is at the low frequencies. By introducing high frequencies we make it more difficult to compress an image and the compression ratio suffers. Therefore zero padding is not an acceptable solution for image compression.

2.1.2 Circular convolution

An alternative to zero padding is *circular filtering*, where we assume the image repeats itself infinitely many times in the 2D plane. The output of the WT, i.e., the WT coefficients, will retain the same periodic structure. This approach does solve the problem of requiring more wavelet coefficients than image pixels, since in this case the wavelet coefficients form a two dimensional periodic signal with period the size of the image. However this approach introduces extra high frequencies, since what appears in the left hand side of an image is not the same as the right hand side of the

[1] Another term for critically sampled transforms is *non-expansive*

image. Periodic extension offers perfect reconstruction and can be acceptable in some cases, but the compression performance of any given algorithm will tend to suffer.

2.1.3 Boundary filters

Zero padding and circular filtering are approaches that will work on any signal and any wavelet kernel, but are not suitable for image compression. Another approach that will work with any filter and any signal would be to design *boundary filters*, so that the matrices in equation 1.2 would have finite support. This approach for finite signal wavelet decomposition is explained in detail in [43]. The basic idea is to change a few entries in the matrices $\mathbf{M}_i$ and reduce them to a block triangular form:

$$\mathbf{M}'_i = \begin{bmatrix} \mathbf{J}_i^u & 0 & .. \\ 0 & \mathbf{B}_i & 0 \\ .. & 0 & \mathbf{J}_i^l \end{bmatrix} \tag{2.1}$$

Where the matrices $\mathbf{B}_i$ are of finite dimensions, the matrices $\mathbf{J}_i^u$ and $\mathbf{J}_i^l$ are of infinite dimensions in one direction and $\mathbf{M}'_i \approx \mathbf{M}_i$. Most entries of $\mathbf{M}'_i$ are the same as the corresponding entries of $\mathbf{M}_i$ with only few exceptions around the bounds of $\mathbf{B}_i$, $\mathbf{J}_i^u$ and $\mathbf{J}_i^l$. If matrices $\mathbf{B}_i$ instead of $\mathbf{M}_i$ are used for filtering, we can work with finite length signals, and the whole approach leads to *time varying filter banks*. The *time varying* term refers to that not all input pixels are treated in the same way, the pixels on the borders are transformed in a slightly different manner, which is equivalent to changing the wavelet transform as a function of the location we are at. Most elements of matrices $\mathbf{B}_i$ are the same as the ones in $\mathbf{M}_i$, the only differences are around the matrix bounds. We change the least number of coefficients from the original matrices $\mathbf{M}_i$ in order to reduce them to a block triangular form, while at the same time preserving the low pass or high pass nature of the convolution kernels corresponding the the matrices. The design involves selection of few parameters in a way that around the bounds the filters corresponding to the transform are close to the original filter bank. The drawback is that we need to design boundary filters for each

Filter Length (L)	Signal Length (τ)	Analysis	Synthesis Lowpass left	Synthesis Lowpass right	Synthesis Highpass left	Synthesis Highpass right
Odd	Even	$\mathcal{W}$	$\mathcal{W}$	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{W}$
Odd	Odd	$\mathcal{W}$	$\mathcal{W}^*$	$\mathcal{W}^*$	$\mathcal{H}^*$	$\mathcal{H}^*$
Even	Even	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{H}$	-$\mathcal{H}$	-$\mathcal{H}$
Even	Odd	$\mathcal{H}$	$\mathcal{H}$	$\mathcal{W}$	-$\mathcal{H}$	+(0)-$\mathcal{W}$

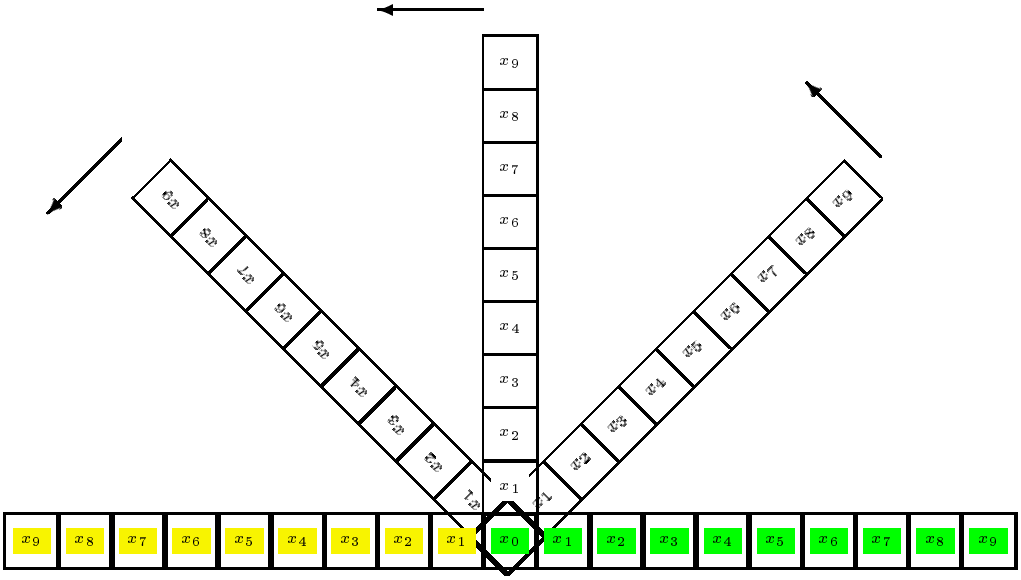
Table 2.1: Type of extension for analysis and synthesis filtering to the left and to the right bound of the signal. The * indicates that $\mathcal{W}, \mathcal{H}$ can be interchanged while $-\mathcal{H}$ denotes that apart from extension we also multiply the extended signal by -1. The (0) denotes that we also need to account for the appearance of zeros as seen later on this chapter.

individual filter bank; changing the coefficients for the wavelet kernel would require designing new boundary filters. Moreover all pixel values around the boundary of the image would be treated in a *special* way. If the boundary filters are designed carefully we will not be introducing high frequencies, but we will still be having some different behavior around the bounds of the image.

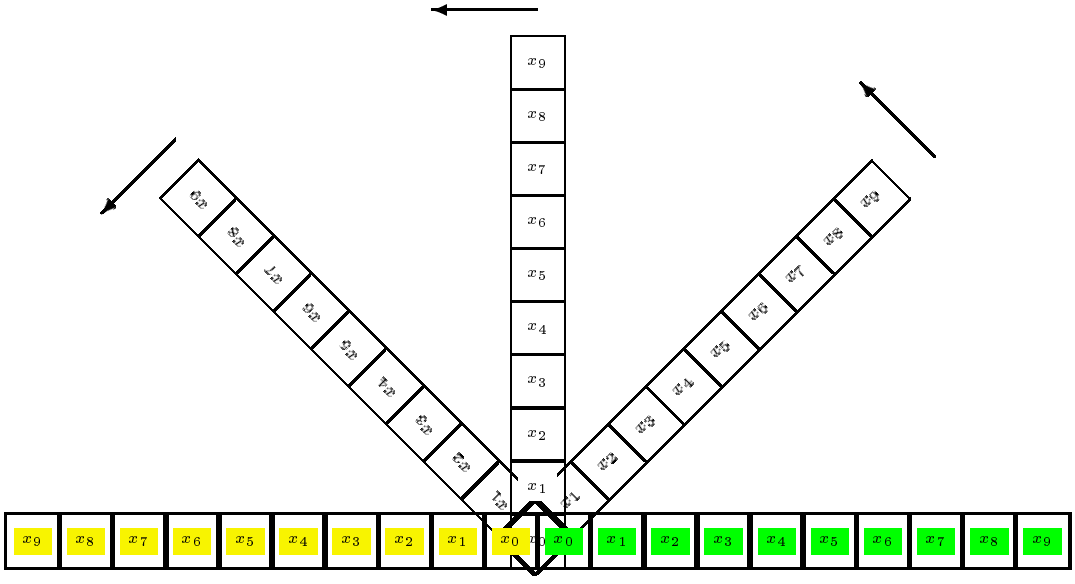
2.1.4 Symmetric extension

As an alternative to the above approaches, we can use *symmetric extension* of the signal at the boundaries, as long as the filter kernel is also symmetric [6]. Symmetric extension works very well since it does not introduce more high frequencies than already exist in an image and "treats" all image pixels in the exact same way, in the sense that all pixels are filtered with the same kernel. Symmetric extension is of such fundamental importance that most wavelet coders are restricted to using symmetric filters in order to allow the use of symmetric extensions[2]. The only orthogonal filter that can also be symmetric is the Haar filter [88]. If we want to have symmetric kernels we need to use bi-orthogonal rather than orthogonal filters. The exact type of symmetric extension depends on the *filter length*, the *signal length* and also whether we

[2]Symmetric filters are also referred to as linear phase filters



$\mathcal{W}$



$\mathcal{H}$

Figure 2.1: Whole point ($\mathcal{W}$) and half point ($\mathcal{H}$) extension

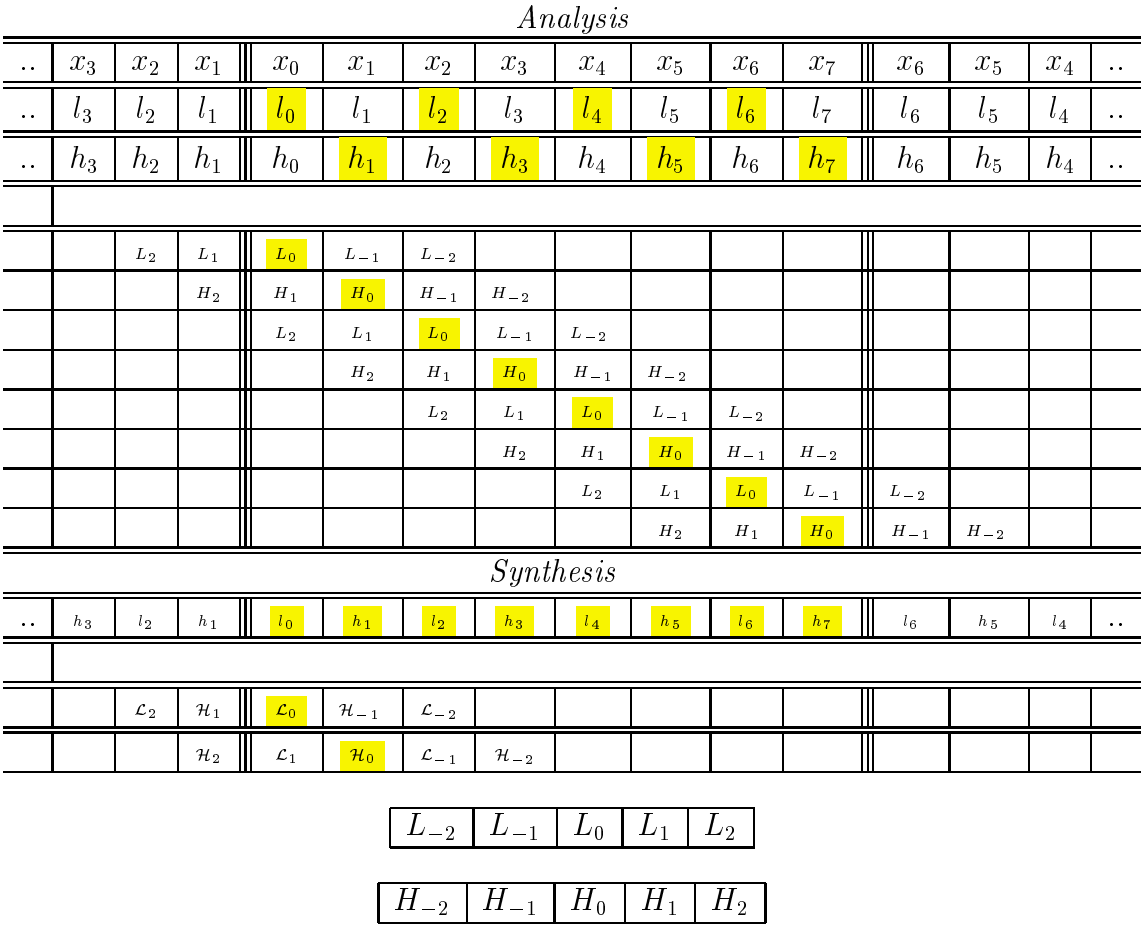


Figure 2.2: Even length signal, Odd length filters, highlighted coefficients are sampling points. $L_{-i} = L_i$, $H_{-i} = H_i$

are considering the *analysis* or the *synthesis* filter banks. In the following paragraphs we cover all the different cases of symmetric extension in the most general settings providing a valuable reference to anyone who wants to implement any specific wavelet kernel.

2.1.5 Whole point/ half point extension

Consider a signal $x_n, n = 0, 1, 2, 3 \ldots$, we can have two types of symmetric extension at the boundary of the signal, *whole point* extension ($\mathcal{W}$) as seen in Figure 2.1($\mathcal{W}$)

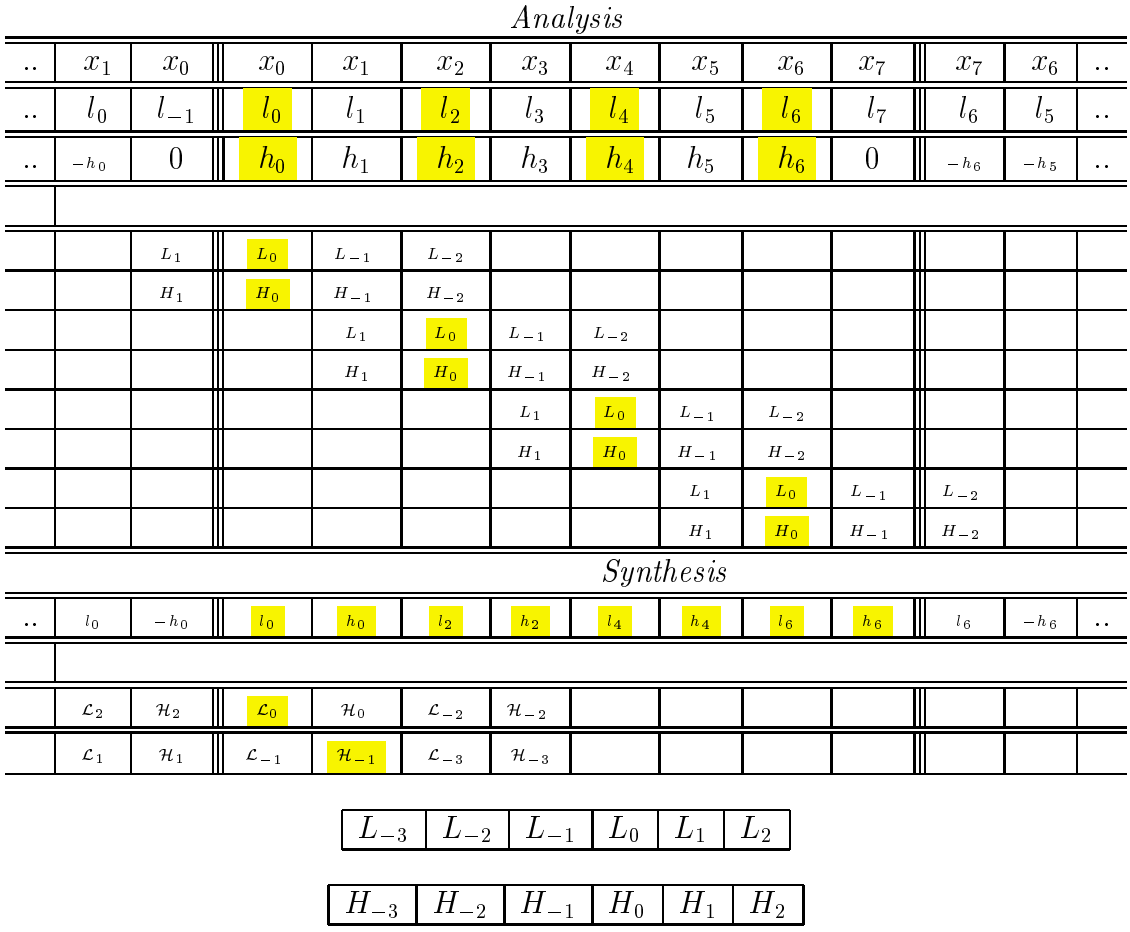


Figure 2.3: Even length signal, even length filters, highlighted coefficients are sampling points. $L_{-i} = L_{i-1}$, $H_{-i} = -H_{i-1}$

and *half point* ($\mathcal{H}$) extension as seen in Figure 2.1($\mathcal{H}$). In the $\mathcal{H}$ extension case the first sample x_0 is repeated, while the $\mathcal{W}$ extension case x_0 is the only sample not to repeat itself. In wavelet filtering we use symmetric extension in such a way that *after filtering* the wavelet coefficients still *possess a symmetry* around the boundaries. We need to be able to find the coefficients outside the boundaries based only on coefficients inside the image. It turns out that we need to consider four different cases, for odd or even signal length τ and odd or even filter length L. In table 2.1 we present the type of extension that we use for the analysis signal, the synthesis high-pass and synthesis low-pass signals, at the beginning and at the end of the signal.

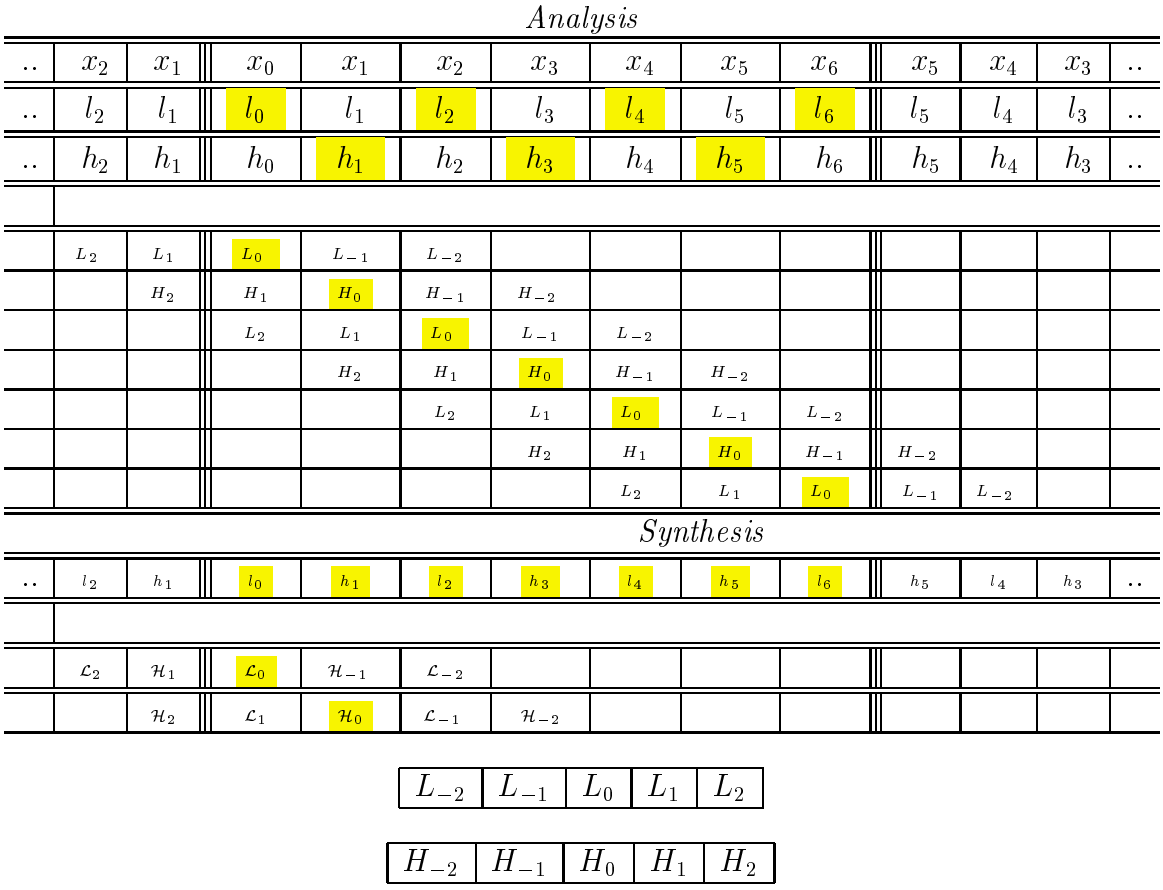


Figure 2.4: Odd length signal, Odd length filters, highlighted coefficients are sampling points. $L_{-i} = L_i$, $H_{-i} = H_i$

We refer to the beginning of the signal as the left bound and the end of the signal as the right bound. Another issue when using symmetric extension on the boundaries is how to center the analysis and synthesis filters around the signal to be filtered. The details of the filtering operations can be seen in Figures 2.2, 2.3, 2.5 and 2.4. Where x_i is the input signal $i = 0, 1, \ldots N-1$, L_i, H_i are the low pass and high pass filters in the filter bank, l_i and h_i are the low pass and high pass filtered versions of the input before sub-sampling.

For even length signals, low-pass and high-pass channels have the exact same number of samples. For odd length signals we have the freedom to assign more coefficients to the low-pass or to the high pass channels. For convenience, in this

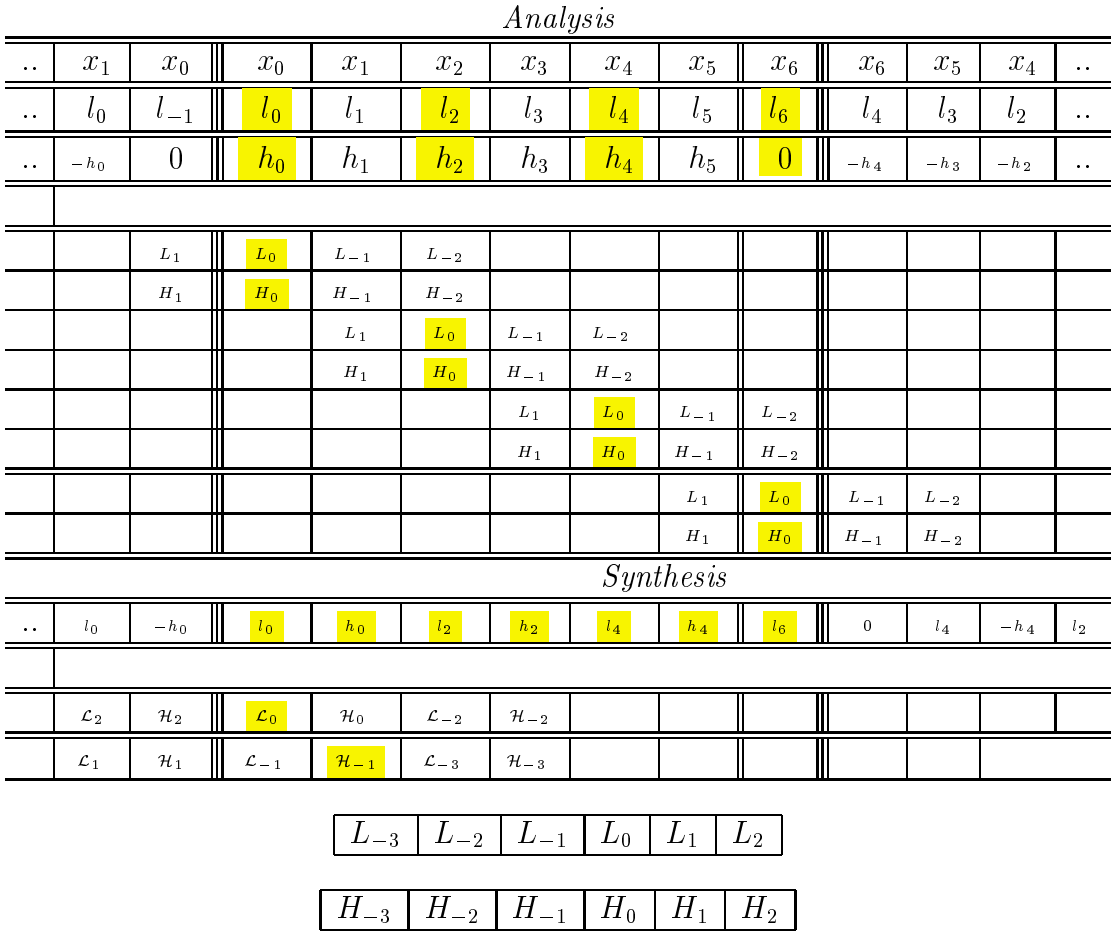


Figure 2.5: Odd length signal, even length filters, highlighted coefficients are sampling points. $L_{-i} = L_{i-1}$, $H_{-i} = -H_{i-1}$

thesis in the case of odd length signals we are assigning more samples to the low-pass channel than the high pass channel. The only reason for such a choice is that we can thus achieve more levels of decomposition.

2.1.6 Point Symmetric Extension

A less well known variation on the above presented symmetric extension type is the *point symmetric extension* [54]. The policy for boundary extension is the same as the one in the previous section, the difference is that the extension is around a point

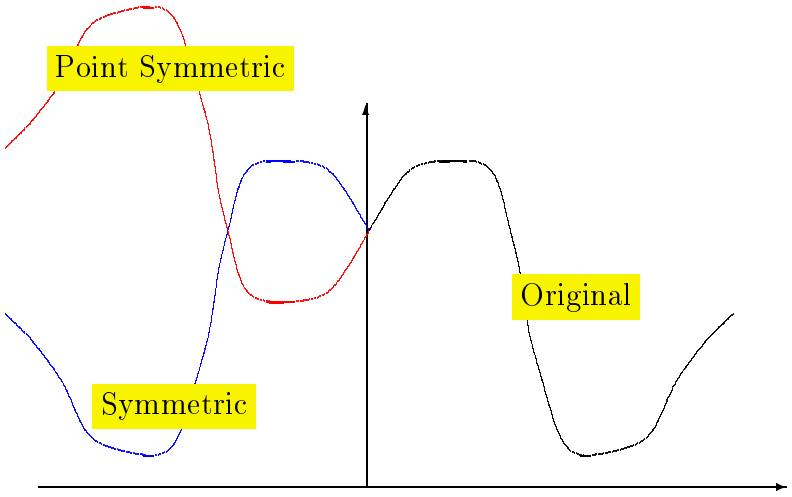


Figure 2.6: Symmetric and point symmetric boundary extension.

instead of a line as seen in Figure 2.6.

There are certain advantages and disadvantages of such a policy for extending a signal at the boundary. Point symmetric extension preserves first and second[3] order derivatives of the signal instead of setting them to zero. This fact alone is extremely helpful in reducing boundary artifacts when blocks of the image are transformed independently. The disadvantage is that the extended signal depends on the last signal sample which might be quantized at the decoder side, in which case we have some precision errors.

The above presented point symmetric extension has not been as widely accepted as the symmetric extension of section 2.1.5.

2.2 Interleaved filtering on the synthesis filter bank

In the analysis filter bank the conventional way to reconstruct a given signal is to up-sample the low pass, up-sample the high pass, filter each one of them separately and add the two results as seen in Figure 1.3b. It is possible to avoid the two different filtering operations if we interleave low pass and high pass signals to create one signal

[3]Second order derivative is set to zero

that is twice as long as the two individual signals. Doing so we can skip the up-sampling process. In this case when we apply filtering we need to also interleave the wavelet filter coefficients. Detailed examples can be seen in Figures 2.2, 2.3, 2.5 and 2.4. L_i, H_i coefficients correspond to the analysis low pass and high pass filters while $\mathcal{L}_i, \mathcal{H}_i$ correspond to the synthesis interleaved filters. Interleaved filtering usually leads to faster and simpler implementations, since we only need to go through the output data once. Interleaved filtering was first used in an implementation of the SPIHT coder in [83].

2.3 Variable complexity inverse WT

When performing the inverse wavelet transform all coefficients usually come from an entropy decoder or a dequantizer. Thus we expect to have a lot of zeros among the coefficients, and it is possible to take advantage of the large number of zero coefficients and avoid filtering altogether. Therefore in the inverse wavelet transform, when we implement the convolution of a filter with a signal we can check if certain portions of our signal are zero. If that is the case we can avoid filtering and instead set the output to zero. The main problem with such approaches is that the number of comparisons with zero involves some computational cost. Also if after comparing with zero a given region is not zero, we will still need to go ahead with the filtering operation. Consider an input signal $x_n, n = 0, 1, 2, \ldots$ a filter $h_0, h_1, \ldots h_{L-1}$ and an output signal $y_n, n = 0, 1, 2, \ldots$ the input output relation is:

$$y_n = \sum_{k=0}^{L-1} x_{n-k} h_k \tag{2.2}$$

If we want to speed up the implementation of the convolution for each output sample we need L comparisons per sample. We compare $x_{n-k}, k = 0, 1, \ldots L-1$ to zero, and if all those numbers are zero we do not continue the convolution. Instead we set the output to zero. If at least one of those values is nonzero we need to multiply the non-zero coefficients with the corresponding filter coefficients. Computationally

testing for zeros in a window of the input signal turns out to be complex. A much better approach is to check each input sample x_n. If it is zero we move on to the next sample, if it is nonzero we add its contribution to the output sequence y_n. The algorithm is as follows:

1. if x_n is zero $\{\ n \leftarrow n+1$, go to step 1 $\}$ else go to step 2
2. for $k = 0, 1, 2, \ldots, L-1, \quad y_{n+k} \leftarrow y_{n+k} + x_n h_k$
3. $n \leftarrow n+1$ go to step 1

Even though theoretically the above approach may offer some benefits in terms of complexity reductions, in practice those benefits can rarely be of any use. The reason is that the savings are highly irregular as the processor cannot predict when a coefficient is significant or not, and as a result the processor pipeline is disturbed, because of the different paths we need to follow depending on the coefficients that we observe. In modern DSP's multiplications can be performed in one cycle, so the above analysis may not even apply at all. Another interesting direction of research is the use SIMD type of instructions on a general purpose processor, in this case we can apply the same operation on multiple data at the same time. Future computers will provide more and more SIMD type of instructions so we will need to revisit the complexity issues based on the latest computer architectures.

A different approach has been proposed in [35], where the IWT is combined with the inverse decoding in a wavelet tree across different levels of decomposition. This approach can indeed speed up computations and works best when used along with a tree based coder.

2.4 Bit reversal and interleaving

We call interleaved, data structures such that wavelet coefficients from different decomposition bands are clustered together, thus allowing spatial localization to be preserved. Instead, in non-interleaved data structures all coefficients corresponding

to the same frequency band are stored together. Wavelet transforms have certain similarities to the FFT, given that data interleaving (bit reversal) is needed in both cases in order to get the correct order of transformed coefficients. Interleaved data in the wavelet transform lead to tree representations, while non-interleaved data lead to subband representations. It is not clear whether it is preferable to encode spatially clustered coefficients or frequency clustered coefficients. This will depend on the application.

Consider the case of a one dimensional two channel filter bank. The input to the filter bank is a signal $x_n, n = 0, 1, 2, \ldots, 2N-1$, the output is comprised of two signals $y_0^0, y_1^0, \ldots, y_{N-1}^0$ and $y_0^1, y_1^1, \ldots, y_{N-1}^1$. We can place those two signals in different memory locations, so that we will have two different frequency bands separated from each other. We can also interleave the two signals into one signal $\mathbf{y} = \{y_0^0, y_0^1, y_1^0, y_1^1 \ldots, y_{N-1}^0, y_{N-1}^1\}$. The same principle applies for an arbitrary number of decomposition levels, where we can have the samples from different bands, located in different memory locations or we can interleave them. Interleaving leads to spatial localization of all coefficients, while separate positioning leads to frequency localization. Tree coders work better with interleaved data while subband coders work better with non-interleaved data, by tree coder we mean a coder that works with trees of wavelet coefficients from different decomposition levels.

From a computational complexity point of view when working with a large number of decomposition levels it is preferable to have non-interleaved data. The reason is that, for interleaved data, samples from the same band are located further apart, so processing of a given band requires more data to go through the cache. The complexity here depends on the cache hits, since the number of operations to be performed will be the same. For a small number of decomposition levels the opposite is true, that is, interleaved data is preferred, since the distance between data from the same subband is upper bounded to a small number.

Interleaved data can be put in non interleaved form by using a bit reversal table and vice versa. Bit reversal is used when implementing the FFT. Wavelet transforms

can be implemented in-place using similar structures to the butterflies in FFT, followed by bit reversal in a similar manner to FFT. Lifting implementations are very well suited for such tasks, as will be seen in chapter 6.

2.5 Issues in Hardware Designs

The majority of new compression schemes based on wavelets and arithmetic coding are implemented in software, and therefore their complexity analysis is based on software. Hardware implementations of individual components of the system such as arithmetic coding or wavelet transforms have been studied independently by a number of authors [11], but there has been no comprehensive study of the hardware architectures for a complete compression system, since such systems are not widely used, if used at all, as of today. Designs for individual systems have been performed but no complete compression system design is known to exist. Something that has been greatly overlooked is the fact that practical devices such as printers need to compress images of hundreds of millions of pixels and it is not feasible to buffer the complete image. For large images the bottleneck in the whole design is the bandwidth needs of any algorithm. It is extremely difficult to pass the data around all the different components of a system for the whole image, in a reasonable time. There have been designs addressing video images of sizes no more that 1K by 1K, but to the best of our knowledge there is no work addressing large size images. It will be of great interest to investigate some possible hardware architectures and see what are the complexity tread-offs compared to software implementations. In hardware, numerical operations are not as time consuming as in software, but on the other hand memory accesses are much more expensive.

The objective of the hardware designer is to design a system that can encode wavelet coefficients at the rate the wavelet transform engine generates them. Thus the entropy coding and the transform engine need to be balanced. In this way we can fully utilize all components of the system without one component being idle, while another is working full speed. It is of great importance that all components work

in parallel, in this way we get the best performance out of the hardware or the best performance for the minimum number of gates.

We will not elaborate more on hardware design issues in the rest of the thesis, since that requires detailed knowledge of the specific system, but we will always try to take into consideration the basic principles of hardware designs as explained in this section.

Chapter 3

Entropy Coding

Contents

In this chapter we will discuss briefly a few issues related to entropy coding. We will review the basics of entropy coding and emphasize that code design problems translate to probability estimation. We will also discuss entropy coding and probability estimation techniques that are used in image compression and for non-stationary data.

3.1 Introduction

Entropy coding is of fundamental importance in any compression system. The term coding refers to a mapping C from a discrete random variable $x \in \mathcal{X}$ to a finite length string $C(x)$ from an alphabet $\mathcal{B}$. This finite length string is called a *codeword*. The most common alphabet for the representation of codewords is the binary alphabet $\mathcal{B} = \{0, 1\}$. The purpose of entropy coding is to map the most probable symbols to shorter codewords and the less probable symbols to longer codewords. In this way the expected length for the representation of a sequence can be minimized. Entropy coding has been studied extensively in information theory [90]. In image compression applications the most common entropy coding techniques are Huffman coding and

arithmetic coding. The *expected length* $L(C)$ of a *source code* $C(x)$ for a random variable X with probability mass function $p(x)$ is given by:

$$L(C) = \sum_{x \in \mathcal{X}} p(x) l(x) \text{ bits}, \tag{3.1}$$

where $l(x)$ is the length of the codeword for x.

3.1.1 Definition of Entropy

Consider a discrete random variable X with values from set $\mathcal{X}$, and probability mass function $p(x) = Pr(X = x), x \in \mathcal{X}$, the entropy $H(X)$ of the random variable X is defined as follows:

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log_2 p(x) \text{ bits} \tag{3.2}$$

Entropy denotes the minimum average number of bits needed to represent an instance of the random variable X. Entropy of a random variable represents the lower bound of the expected length for any code for that variable. That is, no code for a given variable X can achieve a bit rate lower than the entropy.

Entropy can defined only if we have a well defined probability model for the random variable X, without a probability model $p(x)$ entropy cannot be defined and used [70]. Moreover it has to be emphasized that the bound $L(C) \geq H(X)$ applies only given that the probabilistic model and the coding technique are based on the same model of the source. Thus if $H(X)$ is based on a first order model $P(X = x)$ but we use a vector coder $x_1, x_2 \rightarrow C(x_1, x_2)$ the average length of the code can be less than $H(X)$, when there is some dependence between $x1$ and x_2.

3.1.2 Huffman Codes

Within the class of codes that map one input alphabet symbol into one codeword, Huffman codes are optimal for any given source. That is, there is no other code with

smaller average expected length. It can be proved [90] that the average length L of Huffman codes is within one bit from the entropy, that is:

$$H(X) \leq L < H(X) + 1 \tag{3.3}$$

Based on the above equation we can see that for sequences with entropy much larger than one bit per pixel, $H(X) >> 1$, Huffman codes are very efficient, but for sources with entropy less than one, $H(X) < 1$, the efficiency of Huffman codes is not guaranteed. Huffman codes are optimal codes that map one symbol to one codeword, but if we consider mapping 2 symbols to one codeword we can achieve better codes and the bounds in 3.3 change dramatically. In the case where we encode N symbols in one codeword using Huffman codes, the bounds in 3.3 become [90]:

$$H(X) \leq L < H(X) + \frac{1}{N} \tag{3.4}$$

That is, asymptotically as we encode larger and larger blocks of data with one code we can achieve performance arbitrary close to the entropy of the source (we assume and i.i.d. source). The design of Huffman codes for encoding vectors of symbols becomes complicated. Moreover the size of the codebook becomes prohibitively large. A better alternative to using higher dimension Huffman codes is to employ arithmetic coding.

3.1.3 Arithmetic Coding

The principle behind arithmetic coding is the encoding of a sequence of values $x_0, x_1, \ldots x_n$ with one codeword. Arithmetic codes can asymptotically achieve the entropy of a source as the size of the encoded sequence becomes larger and larger, provided that the source is stationary and the statistics of the source are known. Arithmetic coding involves computing the probability of the sequence $x_0, x_1, \ldots, x_{n-1}$ as a product of the individual probabilities, $p = p_0 p_1 p_2 \ldots p_{n-1}$ and mapping the probability p to a codeword. The encoder does not need to scan the whole sequence in order to generate the codeword. Instead, it can start generating bits of the codeword after going

through the first symbols in the sequence. Arithmetic coding is essentially a way of encoding a sequence without direct use of a codebook.

Arithmetic coding performs very well for sequences with very low entropy where Huffman codes lose their efficiency. They are always asymptotically optimal for stationary sources. But none of the above features of arithmetic coding has made it as popular as its ability to work with different probability models p_i. Huffman codes can also be made adaptive but the complexity of such a task is higher as compared to arithmetic coding. Thus arithmetic coding owes its popularity to a large degree to their ability to adapt to changing input statistics.

3.1.4 Huffman versus Arithmetic Coding

In many compression standards like JPEG, entropy coding takes more than 60% of the time for compressions ratios of about $8:1$. Techniques such as Huffman coding, or even Golomb [41] codes, are usually faster than arithmetic coding. However the performance will suffer if we directly use a Huffman code instead of arithmetic coding. Moreover adaptivity for non-arithmetic coders is usually more complex. But there are a few alternatives techniques where one can avoid using arithmetic coding altogether as for example [104]. This will be of great interest for any practical compression system.

3.2 Adaptation in Entropy Coding

In arithmetic coding we can change our estimate of the probability mass function of the source symbols. As long as both encoder and decoder use the exact same probabilities, the code will be *uniquely decodable*. For sources for which the distribution is not known a priory we can start with a uniform distribution and adapt the distribution based on the statistics of the encoded data. For sources such that the first order statistics change over time we can keep our estimated model of the source as we move along, so that we can have a very efficient code for a source with varying statistics.

Arithmetic coding separates coding from modeling, and is close to optimal as long as our probability model reflects the statistics of our data. We translate the code design problem to a probability estimation problem; the *more accurate* our estimates the *better the performance* of our code.

3.2.1 Probability estimation

Consider a sequence of symbols x from the set $\mathcal{X} = \{\omega_0, \omega_1, \omega_2, \ldots \omega_{N-1}\}$, let $T[]$ be a table, with entries denoting the number of times each symbol ω_k occurred in the sequence x. That is $T[k] = q$, if ω_k occurred q times. Let $s = \sum_{i=0}^{N-1} T[i]$. An estimate of the probability of symbol ω_k is $p^{(est)}[k] = T[k]/s$. That is in s tries ω_k occurred $T[k]$ times. When we start counting we set all values in the table $T[]$ equal to one. For a stationary source the number $p^{(est)}[k]$ will converge to the actual probability $p[k]$ of ω_k as the number of observations goes to infinity. In a computer, the entries of the table $T[]$ have finite precision and we cannot store an infinitely large number there. For practical reasons we usually restrict the maximum number that we can store in the table to $2^{16} - 1$, that is each entry in the table occupies two bytes. In order to avoid overflow each time we update an entry to the table we check if the number $s = \sum_{i=0}^{N-1} T[i]$ is larger than a threshold τ. If it is larger than τ we divide all entries of the table by two, that is $T[k] \leftarrow (T[k] + 1)/2, k = 0, 1, \ldots, N - 1$.

For stationary sources, during the adaptation process all the entries of the table $T[]$ move from a *transient state* to a *steady state*. In the steady state, there is some fluctuation around a convergence point, due to the finite precision of the implementation. The initialization of the entries in the table $T[]$ will have an effect on the speed of convergence. It was found based on experiments that having all entries equal to one as the initial value makes the algorithm adapt faster. It is also worth pointing out that the state of our table will never go back to all entries equal to one. Normalization will only occur when $s > \tau$. Starting with another state other than all entries equal to one will make it harder to adapt if the initial state is not the steady state. For example if the actual probability of one entry is equal to zero and we have started

with an initial entry equal to a large number it will take much longer to reach convergence. The above described procedure of estimating a probability of a source is very useful for non-stationary data. For non-stationary data the re-normalization of the entries in the table $T[]$ gives us a powerful tool to estimate adaptively the probability mass function of continuously changing data. The greater the non-stationary nature of our data the smaller the parameter τ needs to be. This form of re-normalization is equivalent to a "forget factor" used in a lot of adaptive algorithms. Also the above mentioned techniques have certain similarities to *simulated annealing* used in function minimization [73].

3.3 Bit plane coding versus whole word coding

There are two different approaches to entropy encoding a sequence of numbers x from a set $\mathcal{X}$. The first approach is to use an M-ary code to encode each sample from the sequence, where $M = |\mathcal{X}|$. A second approach is to take the binary representation of each number x and encode each bit of x using a binary entropy coder. From now on we will call the first approach *whole word coding* and the later *binary coding*. In general, for pure lossless compression, binary coding does not offer any advantages and is rarely used. The benefits of binary coding appear when we combine entropy coding with quantization. In this case we can use bitplane coding , i.e., send the most significant bits of all inputs first, then the second most significant, etc. Bitplane encoding enables a successive approximation to quantization and is used in progressive encoding and lossy to lossless compression.

Even if progressive encoding is not desired we may still prefer to implement a binary coder instead of a whole word coder. In hardware implementations of arithmetic coding it is more complex to implement an M-ary coder than to implement M binary arithmetic coders. This is because in hardware memory accesses are more expensive than addition or shift operations.

In many applications it is desired to be able to progressively decode a given image, to do so we need to use bit plane coding of the wavelet coefficients. This is a

requirement in the new JPEG2000 standard and has been addressed by a number of researchers. What is missing in this area is an analysis of how much more progressive transmission costs in terms of memory. In the case of progressive encoding/decoding the encoder usually needs to buffer the whole bit stream or at least parts of the bit stream if not the complete image in some cases. Some insight on the subject will be given in chapter 5. We will not consider bit plane binary coding in the rest of the thesis in great detail since our focus in this work is not on progressive encoding, or other features of interest but rather on compression performance along with memory utilization.

3.4 Fast Huffman Decoding, Canonical Huffman codes

Huffman encoding is a relatively simple operation, since it is a one to one mapping from an index to a codeword. Computational complexity is some of the lowest of any coding algorithm. One cannot say the same for Huffman decoding. The typical way to decode a bit stream is to parse through it and check bit after bit. Each time we read a bit from the bit stream we travel through the nodes of the Huffman tree [55]. Once we have reached a leaf node we have decoded a whole codeword and we can start over again to decode the next symbol.

We cannot decode two codewords in parallel and the decoding process is strictly sequential. Also within a codeword we need to decode bit after bit. A somewhat better approach to the problem is to use a look-up table for decoding. If we know that the largest codeword is of length L, we can use a look-up table of length 2^L to decode the bit stream. We pass L bits of the bit stream to the lookup table and the entry to the table gives us the length of the codeword, since we now know the length of the codeword we can find the codeword itself. Working in this fashion we can decode the whole bit stream. The problem with this approach is that it does not scale very well as the maximum codeword length L increases. For example for 16 bit

codewords we need $65536 = 2^{16}$ bytes to store the look up table.

A different approach is to use normalized Huffman codes as described in [55,84]. For a given source of symbols there is a family of Huffman codes that gives the same compression performance. Within this family of Huffman codes there is only one with the following two properties.

- Shorter codes have a numerically (if filled with zeros to the right) higher value than longer codes.
- Within the same length, numerical values of Huffman coders increase with alphabet. That is, the Huffman codes for each length are sorted in an ascending order.

There are some advantages of using these (or similar) rules and produce a canonical Huffman code. The first rule guarantees that no more than the $\lceil(\log_2(alphabet\_size))\rceil$ rightmost bits of the code can differ from zero. The first rule also allows an efficient decoding. Both rules together allow a complete reconstruction of the code knowing only the code lengths for each symbol. Canonical Huffman codes are used within the JPEG [71] baseline standard. They allow a concrete representation of the code as well as fast decoding.

Even though canonical Huffman codes allow fast decoding, look-up table decoding is usually faster for small alphabet sizes. So in practice a combination of the two is used. We first try to decode with a small look-up table, if decoding is possible we continue with the next symbol, if decoding is not possible we go ahead with the standard procedure to decode canonical Huffman codes. More details can be found in [84].

Chapter 4

Rate Distortion Optimal Adaptive Encoding

Contents

In this chapter we present an adaptive image coding algorithm based on novel backward-adaptive quantization, classification and rate distortion techniques. The rate distortion selection is done in one pass with minor complexity overhead and without the need for extensive search or multiple passes. Moreover the probability models for each context are adapted to past encoded data and no overhead information has to be sent to the decoder. Our results are comparable or in most cases better than the recent state of the art. The main contribution of this work is the use of rate distortion techniques in adaptive encoding applications, jointly optimizing quantizer and entropy coder in one system.

4.1 Background

Over the past few years adaptivity has become an essential component of state of the art image coders, in particular those based on wavelets. Several researchers have advocated making adaptive in various ways the basic components in a wavelet-based image coder, namely, the tree-structured filter bank [114], the filters themselves, the

quantizers [9, 115] and the entropy coders [83, 85, 93]. In this work we concentrate on the issue of adaptive quantization and entropy coding for a fixed filter bank. The novelty comes from the *joint* consideration of quantizer and entropy coder, where we use *rate distortion criteria at the encoder* in order to make optimal decisions, about both entropy coder and quantizer on each individual pixel. We use context modeling and classification and model the distribution of our source as a *mixture of distributions*. Context modeling was directly used in [93] while in [83, 85] context information was taken into account by using a zero-tree data structure which enables the joint transmission of zero-valued coefficients present at the same spatial location across several frequency bands.

We are considering a context-based adaptive arithmetic coder similar to that proposed in [103] with two major differences being (i) we operate in the subband domain, rather than the image domain and (ii) our contexts are determined based on past *quantized* data rather than the original data as in the lossless compression scheme of [103]. Our approach is simpler than adaptive quantization methods and may also be better suited to high rates where the layered coding approaches lose some of their benefits. A similar context based adaptation is presented in [62], our main difference with the approach in [62] is the use of online joint rate distortion optimization on each individual pixel, unlike [62] where there is no such consideration. We also differ from the coder used in the JPEG2000 verification model in a few respects. For example we use a *non binary* arithmetic coder, we use rate distortion criteria to encode wavelet coefficients and unlike JPEG2000 we consider quantization and coding jointly on each individual coefficient. As will be seen our coder can be thought of as a lossy encoding of fine quantized indices.

4.1.1 Algorithm Basics

We first provide a brief outline of our algorithm. Given an image we apply a wavelet transform and weight each subband by an appropriate factor. We scan the subbands in a predetermined order. Within a subband we classify each coefficient to be encoded

based on past encoded coefficients. To each class corresponds a probability model for the distribution of the wavelet coefficients in the class and based on this model we decide on how to encode and quantize a coefficient, using rate distortion theory. After encoding each coefficient we update the probability model corresponding to its class. In the next section we give a theoretical background for quantization, rate distortion, entropy coding and control of image quality. The introductory material is necessary to understand our choices later in the chapter.

4.2 Quantization and Optimal Quantizer Design

A quantizer Q is a mapping from the real line $\mathcal{R}$ to a K-point set $\mathcal{Q}$, $Q : \mathcal{R} \rightarrow \mathcal{C}$, where $\mathcal{Q} = \{q_0, q_1, q_2, \ldots, q_{K-1}\} \subset \mathcal{R}$. Input images are stored in digital form and after application of the wavelet transform the wavelet coefficients require more precision than the original pixel values and are usually stored in floating point format. Quantization is used to reduce the number of bits needed for their representation. Strictly speaking wavelet coefficients are not continuous but this is a very good approximation.

The *quantizer resolution* r is defined as $r = \log_2 K$. Each quantizer is defined by its *output levels* $q_i, i = 0, 1, \ldots, K-1$ and its *partition cells* $R_i = (x_{i-1}, x_i)$. An input value x is mapped to q_i if and only if $x \in R_i$. A measure of performance for a quantizer is the distortion D it induces on the input signal x. A very common distortion measure is the L_α norm:

$$D_\alpha = (\sum_{i=0}^{K} \int_{R_i} f(x)(x - q_i)^\alpha dx)^{\frac{1}{\alpha}} \tag{4.1}$$

Where $f(.)$ is the probability density function (pdf) of x. Our objective when designing a quantizer is to minimize the distortion measure D_α. There is a general procedure to design such a quantizer known as Lloyd-Max iteration [40]. This procedure is not guaranteed to achieve a global optimum but is at least guaranteed to achieve a local optimal solution that depends on the initial conditions. The design is an iterative process based on two conditions. Consider that we are at iteration n and

we are going to the next iteration $n+1$. The first condition is the *centroid condition*:

$$q_i^{(n+1)} = E[X \Big| X \in R_i^{(n)}] \tag{4.2}$$

and the second condition is the *midpoint condition*:

$$x_i^{(n+1)} = \frac{q_{i-1}^{(n)} + q_i^{(n)}}{2} \tag{4.3}$$

Where $q_i^{(n)}, i = 0, 1, \ldots$ are the output levels, $R_i^{(n)}, i = 0, 1, 2, 3 \ldots$ are the partition cells, and $x_i^{(n)}, i = 0, 1, 2, 3 \ldots$ are the bounds between partition cells at the n^{th} iteration. Optimal quantizers depend heavily upon the input distribution and they need to be designed for each input source. Even though they are guaranteed to minimize an error metric, they have not been proved to be particularly useful as a stand alone system in image compression, because:

1. The optimal quantizer depends upon the input source and different images have different statistics.

2. Design of an optimal quantizer is an iterative procedure and computationally expensive

3. The quantizer is only a part of a much larger system and needs to be considered as such so that optimal quantization does not imply that an optimal compression system will be achieved

4. The structure of the quantizer will affect the entropy coder and the entropy coder needs to be designed based on the quantizer output

4.2.1 Jointly Optimal Quantizer and Entropy Coder

In order to specify the quantized value q_i of a sample x we only need an index i. This index will be further entropy coded and mapped to a codeword $\lambda(i)$. Our objective in image compression is to minimize distortion for a given rate, rather than to minimize

distortion for a given quantizer, so we need to consider design of a quantizer and entropy coder jointly. There exists a procedure to design an optimal[1] entropy coder for a given source. This process leads to *Huffman* codes [90]. There is also a whole theoretical framework for rate distortion in information theory [90]. The objective is to minimize a distortion metric D subject to a maximum possible bit rate, $R \leq R_0$. This objective if formulated as a constrained minimization problem, as explained in [12, 61], will lead to an unconstrained minimization of a metric of the form:

$$J = D + \mu R \tag{4.4}$$

where D is the distortion, R is the rate and $\mu \geq 0$ the Lagrange multiplier. The parameter μ needs to be adjusted so that the target bit rate of R_0 is achieved. For each value of μ the system will be optimal for a bit rate $R = R(\mu)$ and the goal will be to find μ such that $R(\mu) = R_0$ The Lloyd-Max design for a quantizer can be generalized [12] to include rate distortion measures as in (4.4). In this case the centroid condition remains the same:

$$q_i^{(n+1)} = E[x \Big| x \in R_i] \tag{4.5}$$

and the midpoint condition becomes the *generalized* midpoint condition.

$$R_i^{(n+1)} = \{x : J(x, q_i^{(n)}) < J(x, q_j^{(n)}), i \neq j\} \tag{4.6}$$

At the end of each step we also need to redesign our entropy coder to reflect the new statistics for the distribution of the quantized indexes. The whole Lloyd-Max design is based on training, and thus we need to know our source and design the quantizer based on the source. The design is computationally intensive and needs to be applied to every source sequence to be compressed. After the design, quantizer and entropy coder parameters need to be sent as overhead to the decoder along with the actual compressed bit stream.

[1] Optimal under the assumption of a one to one mapping from a source symbol to a codeword

Since the quantizer is considered jointly with the entropy coder, it is not clear what the respective contributions of quantizer and entropy coder are to the coder performance. One might claim that a "bad" quantizer if combined with an appropriate entropy coder might be as good as the above mentioned optimal design. Moreover locally optimal solutions might hinder the global optimum. These are some of the reasons for not using the above mentioned algorithm in practical image compression systems.

Instead it is preferable to use a fixed uniform quantizer throughout the whole image. The entropy coder can be adapted on the fly to the input statistics and in this way it is not necessary to redesign the quantizer. The quantizer remains fixed, and the output of the quantizer is used to collect statistics that are used by the entropy coder. Overall this leads to a much simpler system, both computationally and conceptually. Also uniform quantizers are much simpler to implement than quantizers with variable bin size; The adaptation on the fly is essential in all image compression algorithms since image data are not stationary and we need to be able to adapt our models in order to take full advantage of a coder. Uniform quantization, followed by optimal entropy coding has been proved to be very close to the rate distortion curves under relatively non restrictive conditions [34, 72], and thus in what follows we will use uniform quantizers.

4.2.2 High Resolution Quantizers

The term *high resolution* quantizer [61] refers to quantizers where the quantization intervals R_i are small compared to the variation of the input signal [40]. Assume a random variable x with pdf $f(.)$ and let $M = \max\{x\}$, $m = \min\{x\}$, where M and m can be finite or infinite. Consider a uniform quantizer with quantization step size Δ and assume Δ is small enough such that within an interval of width Δ the function $f(\rho)$ is *practically* constant. Let the reconstruction point for the inverse quantizer be exactly in the middle of the interval $[n\Delta, n\Delta + \Delta)$, that is $n\Delta + \Delta/2$. Consider the L_α norm, $L_\alpha\{x\} = (Ex^\alpha)^{\frac{1}{\alpha}}$, where E denotes expectation, and let us calculate the

quantization error:

$$(L_\alpha\{x-\hat{x}\})^\alpha = \sum_{n=-\infty}^{\infty}\int_{n\Delta}^{n\Delta+\Delta}(\rho - n\Delta + \frac{\Delta}{2})^\alpha f(\rho)d\rho \approx$$

$$\sum_{n=-\infty}^{\infty}\int_{n\Delta}^{n\Delta+\Delta}(\rho - n\Delta + \frac{\Delta}{2})^\alpha f(n\Delta)d\rho = \sum_{n=-\infty}^{\infty}\frac{\Delta^{\alpha+1}}{2^\alpha(\alpha+1)}f(n\Delta) = \frac{\Delta^\alpha}{2^\alpha(\alpha+1)}$$

That is:

$$D_\alpha = L_\alpha\{x-\hat{x}\} = \frac{\Delta}{2(\alpha+1)^{\frac{1}{\alpha}}}. \tag{4.7}$$

We consider the general L_α norm here but later we will concentrate on the L_1 norm for simplicity. The reason for taking a generic approach is that we are trying to cover both L_1 and L_2 norms at once. Consider the rate distortion measure $J_\alpha = D_\alpha^\alpha + \mu R$ where R is the rate and D_α is the distortion. We want to minimize J_α subject to a constraint $R \leq R_0$. We know the distortion D_α as a function of the quantization step size Δ, but we also need to evaluate the entropy $H(\hat{x})$ as a function of the same quantity Δ. We will consider a few different particular cases before we give a more general estimate of the entropy.

1. Assume that x has a uniform distribution, that is, $f(\rho)$ is constant. Since Δ is relatively small, we can assume that $\hat{x}$ has a discrete uniform distribution. In this case the entropy will be:

$$H(\hat{x}) = \log_2 \frac{M-m}{\Delta} = \log_2(M-m) - \log_2\Delta(\text{bits}) \tag{4.8}$$

2. Assume that x has Laplacian distribution, that is, $f(\rho) = \frac{1}{2}\lambda e^{-\lambda|\rho|}$. In this case $\hat{x}$ has a 'two sided' geometric distribution. The differential entropy of x is:

$$h(x) = -\int_{-\infty}^{\infty} f(\rho)\ln f(\rho)d\rho = -\ln\frac{\lambda}{2} + 1 \text{ (nats)} = -\log_2\lambda + \frac{1}{\ln 2} + 1 \text{ (bits)} \tag{4.9}$$

Define θ as:

$$\theta = e^{-\lambda\Delta}, \tag{4.10}$$

The probability that a sample falls into the n^{th} bin $[n\Delta, n\Delta + \Delta)$ is:

$$p_n = \frac{1}{2}\theta^{|n|}(1-\theta) \tag{4.11}$$

The entropy of the quantized variable $\hat{x}$ is:

$$H(\hat{x}) = \sum_{n=-\infty}^{\infty} p_n \ln p_n = \frac{\theta}{1-\theta}\ln\theta + \ln\frac{1-\theta}{2} \text{ (nats)} \tag{4.12}$$

If we consider a high resolution quantizer, that is for $\lambda\Delta \to 0$ we have $\theta \approx 1-\lambda\Delta$

$$H(\hat{x}) = -\ln\frac{\Delta\lambda}{2} - \frac{1-\Delta\lambda}{\Delta\lambda}\ln(1-\Delta\lambda) \approx h(x) - \ln\Delta \text{ (nats)} \tag{4.13}$$

the rate needed for encoding of the quantizer indexes is: $R = h(x) - \log_2\Delta$ bits.

In both cases 1 and 2 the rate depends on the logarithm of the quantization step size $\log_2\Delta$. This observation can be generalized by the following theorem.

Theorem 1 *Let $H(\hat{x})$ be the entropy of the quantized variable $\hat{x}$ obtained from a continuous variable x using a uniform quantizer with step size Δ. Then:*

$$H(\hat{x}) \approx h(x) - \log_2\Delta \text{ bits} \tag{4.14}$$

The conditions for the theorem to hold is that Δ is sufficiently small compared to the variation in the pdf of the random variable x. $h(.)$ stands for differential entropy while $H(.)$ stands for discrete entropy.

Proof: *See Appendix A.*

The above theorem will be very useful when trying to use rate distortion theory in making decisions on how to quantize wavelet coefficients.

4.2.3 Control of image quality

If we use a uniform quantizer throughout the whole image, the quantizer step size will control the image quality. For small enough Δ, based on equation (4.7) the MSE in the transform domain will be $MSE = \Delta^2/12$. If we assume an orthonormal transform the MSE will be the same in the image domain and the $PSNR$ will be:

$$PSNR \approx 10 \log_{10} \frac{12MAX^2}{\Delta^2} \tag{4.15}$$

In this way we can control the image quality by varying the quantization step size Δ, for example we can achieve a target $PSNR$ by selecting the appropriate step size Δ. In our algorithm the only parameter we can vary is Δ. Before starting to compress an image we can set a target quality ($PSNR$) and choose the appropriate step size Δ to achieve that.

4.3 Proposed Rate Distortion Encoding Scheme

We consider coding and quantization jointly according to Figure 4.1. The quantization cannot be considered as a separate process as explained earlier. Our reconstruction levels are integer multiples of Δ. We thus approximate the real line with the set of integers, where each integer n represents the real number $n\Delta$. The only reason we selected such a representation is simplicity. For each wavelet coefficient x we transmit the index n that is optimal in the rate distortion sense, i.e. the one that minimizes:

$$J_\alpha(x, n\Delta) = |x - n\Delta|^\alpha + \mu R(n) \tag{4.16}$$

From the set of available integers n we can easily select the one that minimizes (4.16) for a given μ. The rate $R(n)$ can be found directly from the probability table used in conjunction with the arithmetic coder. However, the Lagrange multiplier μ, which will be fixed for all coefficients, will have to be determined . We now provide a derivation of the optimal value for μ, under relatively non restrictive conditions.

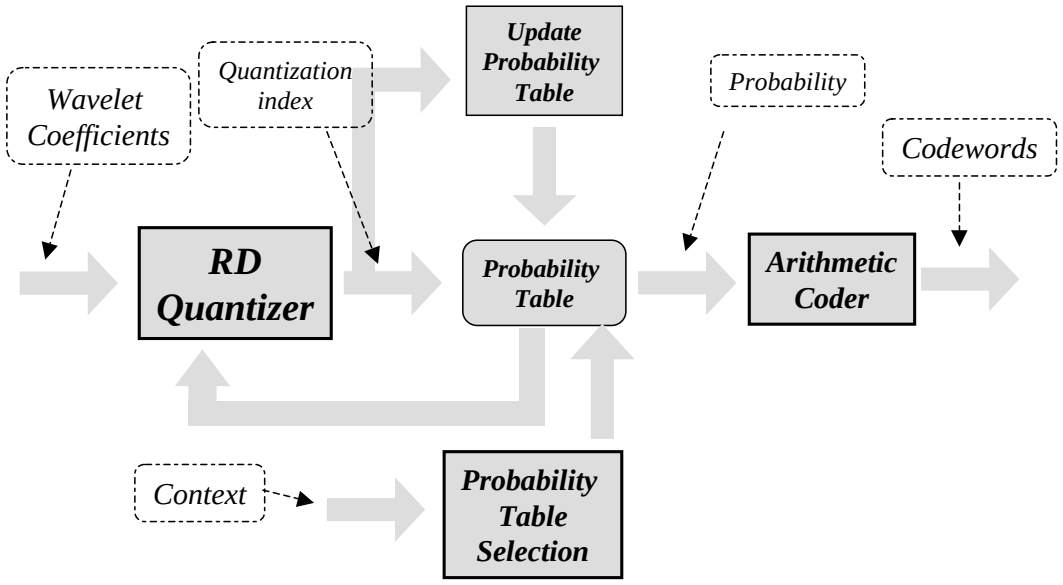


Figure 4.1: Block diagram of the quantization and entropy coding parts of the system. There is a closed loop formed by the rate distortion optimized quantizer and the probability table estimation. The quantizer makes decisions based on the given probability model and the probability model is adapted based on the output of the quantizer.

4.3.1 Optimal choice for the Lagrange Multiplier

The parameter μ will depend on the desired bit rate R_0 and the parameters of the data to be encoded. It needs to be estimated before we start encoding wavelet coefficients. As it turns out the choice of μ does not depend on the input distribution but only on the quantization step size Δ. If we take expectations in (4.16) we have:

$$J_\alpha(\Delta) = E\{J_\alpha(x, n\Delta)\} = E\{|x - n\Delta|^\alpha + \mu R(n)\} = \frac{\Delta^\alpha}{2^\alpha(\alpha+1)} + \mu(h(x) - \log_2 \Delta) \tag{4.17}$$

We then substitute equations (4.7) and (4.14) in (4.17) and minimize J_α subject to the constraint $R \leq R_0$,

$$\left.\begin{array}{c} \min\limits_{\mu}\{J_\alpha\} \\ \\ R \leq R_0 \end{array}\right\} \Rightarrow \left.\begin{array}{c} \min\limits_{\mu}\{\dfrac{\Delta^\alpha}{2^\alpha(\alpha+1)} + \mu(h(x) - \log_2 \Delta)\} \\ \\ h(x) - \log_2 \Delta \leq h(x) - \log_2 \Delta_0 \end{array}\right\} \Rightarrow$$

$$\left.\begin{array}{c} \dfrac{\partial}{\partial \Delta}\{\dfrac{\Delta^\alpha}{2^\alpha(\alpha+1)} + \mu(h(x) - \log_2 \Delta)\} = 0 \\ \\ \Delta \geq \Delta_0 \end{array}\right\} \Rightarrow \left.\begin{array}{c} \dfrac{\alpha\Delta^{\alpha-1}}{2^\alpha(\alpha+1)} - \mu\dfrac{1}{\Delta \ln 2} = 0 \\ \\ \Delta \geq \Delta_0 \end{array}\right\} \Rightarrow$$

$$\mu = \mu_\alpha \Delta_0^\alpha \quad , \quad \mu_\alpha = \frac{\alpha \ln 2}{2^\alpha(1+\alpha)}$$

So in this case:

$$J_\alpha(x, \hat{x}) = |x - n\Delta_0|^\alpha + \Delta_0^\alpha \mu_\alpha R(n) \tag{4.18}$$

or we can normalize $J_\alpha(x, n\Delta)$ as:

$$J'_\alpha(\tilde{x}, n) \triangleq \frac{J_\alpha(x, n\Delta)}{\Delta^\alpha} = |\frac{x - n\Delta}{\Delta}|^\alpha + \mu_\alpha R(n) = |\tilde{x} - n|^\alpha + \mu_\alpha R(n) \tag{4.19}$$

where $\tilde{x} = x/\Delta$. Thus the coding procedure would be to compute $\tilde{x}$, then find n that minimizes (4.19). It is very common, for the set of integers n we consider in equation (4.19), the solution to be a relatively large number, e.g., ranging from -255 to 255. In this case a full search might be very expensive, so we now propose a fast way of searching for the optimal solution.

4.3.2 Fast search for the optimal solution

Up to now we considered a general metric L_α for the quantization error. From now on we will concentrate on the L_1 norm. The reason for such a choice is simplicity. When we use RD allocation decisions we need to evaluate a number ρ^α as in (4.19). Obviously, the simplest choice is $\alpha = 1$ which leads to performing a simple absolute value operation $|\rho|$. In the case of the L_1 norm equation (4.19) becomes:

$$J_1'(\tilde{x}, n) = |\tilde{x} - n| + \mu_1 R(n) \tag{4.20}$$

Let $n_0 = \lfloor x/\Delta \rfloor = \lfloor \tilde{x} \rfloor$ and let n_{opt} be the optimal choice in the rate distortion sense, i.e. the one that minimizes 4.20. Then, by definition of optimality we have that $J_1'(\tilde{x}, n_{opt}) = |\tilde{x} - n_{opt}| + \mu_1 R(n_{opt}) \leq J_1'(\tilde{x}, n_0)$, leading to $|\tilde{x} - n_{opt}| \leq J_1'(\tilde{x}, n_0)$, so that we can bound n_{opt} as:

$$\tilde{x} - J_1'(\tilde{x}, n_0) \leq n_{opt} \leq \tilde{x} + J_1'(\tilde{x}, n_0) \tag{4.21}$$

The above equation states that the optimal solution is in the interval $\mathcal{S} = [\tilde{x} - J_1'(\tilde{x}, n_0), \tilde{x} + J_1'(\tilde{x}, n_0)] \cap [n_{min}, n_{max}]$. Moreover, there are only a finite and small number of solutions in the interval, since the solution needs to be a multiple of the step size.

Let the possible set of solutions be $\mathcal{W} = \{\rho_0, \rho_1, \ldots, \rho_{|\mathcal{S}|-1}\}$, we need to go through all the elements of the set $\mathcal{W}$ and choose the one that minimizes the cost metric J_1'. Evaluation of the rate $R(n)$ involves taking the logarithm of the probability p, i.e., $R = -\log_2 p$. The probability p is the ratio of an element $q = T[n]$ to the sum s of all the elements of $T[]$, $s = \sum_j T[j]$, i.e., $p = q/s$. Then we can write:

$$J_1'(\tilde{x}, n) = |\tilde{x} - n| - \mu_1 \log_2 \frac{q(n)}{s(n)} = |\tilde{x} - n| - \mu_1 \log_2 q(n) + \mu_1 \log_2 s(n) \tag{4.22}$$

Since the term $\mu_1 \log_2 s(n)$ is common to all possible solutions we only consider the

other two terms, that is, we define:

$$J_1''(\tilde{x}, n) \triangleq |\tilde{x} - n| - \mu_1 \log_2 q(n) \tag{4.23}$$

Computationally, the evaluation of the logarithm might turn out to be very expensive, but we can use an approximation to the logarithm, e.g., we can evaluate only the integer part of the logarithm. An even better approach is to use a lookup table $\Psi[]$, where the size of the lookup table will be 2^δ and δ is the number of bits used to store the frequency counts in the probability estimation. Equation (4.23) then becomes:

$$J_1''(\tilde{x}, n) = |\tilde{x} - n| + \Psi[T[n]] \tag{4.24}$$

We can use equation 4.23, restrict our search to the set $\mathcal{S}$ and use a lookup table for the logarithm. Using all these approaches we can significantly speed up our rate distortion selection. For example, for typical images the reduction in complexity when using these techniques instead of full search with the complete cost function is of the order of at least 3 times.

4.4 Exploiting Local Energy Clustering

Energy in frequency subbands after wavelet transform tends to be clustered, so that large coefficients tend to occur in groups. This can be exploited to increase coding efficiency. Suppose that we have transmitted a number of coefficients $x_k, k = 0 \ldots n$ of the wavelet representation of our image. Then based on the past we can try to estimate the next coefficient that we need to transmit. Many experiments have shown that traditional linear prediction is not a very efficient predictor in image subbands. In a compression scheme using linear prediction, the difference between the current coefficient and a predictor obtained from previously quantized coefficients is sent. Since linear correlation of wavelet coefficients tends to be close to zero and prediction results in doubling of the dynamic range, little gain is in general achieved with this method. However context information is useful when it comes to adapting the entropy

coder, as was demonstrated in [103] in a lossless image coding scenario. In this work we use a neighborhood of previously quantized coefficients to determine, from a finite set of choices, which probability model to use for the entropy coder. The motivation is simple; when surrounding coefficients are close to zero it is more likely that the next coefficient will also be zero. The coefficient in the same position in the previous band also offers some information about the value of the current coefficient. Our proposed context formation and classification is performed by computing:

$$\hat{y} = \sum_i w_i |\psi_i| \tag{4.25}$$

where ψ_i is a previously quantized coefficient and i covers all the previously quantized coefficients in the neighborhood of our current coefficient. The weights w_i are chosen to be inversely proportional to the distance from our current coefficient to the position corresponding to ψ_i. The coefficients ψ_i used for context are depicted in Figure 4.2. The class γ to which each coefficient is assigned is:

$$\gamma = \max\{c, \lceil \log_2(\hat{y}+1) \rceil\}, \quad c = number\_of\_classes - 1 \tag{4.26}$$

The total number of classes used in the final experiments was 13 as explained in Section 4.5.5. We do not use an actual $\log_2$ function on floating point data, instead we convert $\hat{y}$ to an integer where $\lceil \log_2 \rceil$ is trivially implemented. There is no particular optimization in the design of the context formation. The operator $|.|$ introduces the non linearity needed to exploit the magnitude correlation. The weights w_i are chosen in a way such that closer coefficients have larger contribution to the context formation than coefficients that are further away from the current pixel location. The $\log_2()$ operator was also chosen since it was observed that the distribution of the values of $\hat{y}$ was approximately geometric. That is, most values of $\hat{y}$ were close to zero with very few values of $\hat{y}$ much larger than zero. The $\log_2()$ operator ensures that we get a roughly uniform distribution of the points in all classes. In this way, we use small length intervals to define classes in the low magnitude case where most classes occur,

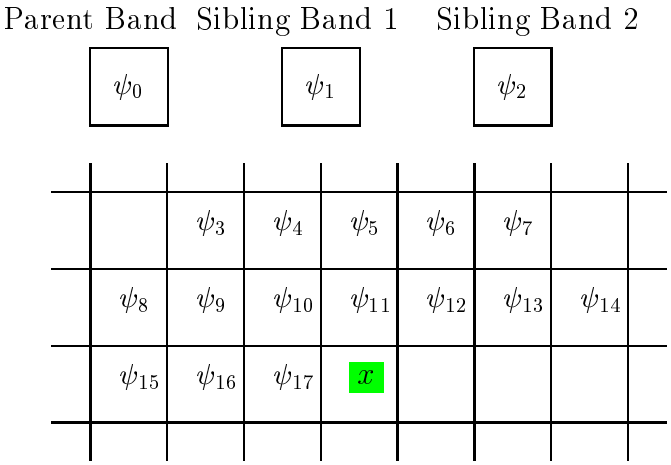


Figure 4.2: Context used for the computation of $\hat{y}$. The parent band and up to two sibling bands may be used in context formation. The parent band will be used if it is not the DC band. When encoding the 3 subbands in one level, the first has no sibling, the second has one sibling and the third has two siblings.

while we use long length intervals for the classes in the high magnitude case where fewer samples occur.

4.4.1 Recursive Context Formation

Our proposed context formation as seen in (4.25) requires some computation to be performed each time we consider a new coefficient. This computation can be seen to be equivalent to an FIR filter. In Chapter 5 we use a much simpler technique to compute the context with very good results. In fact, the later approach carries information about the context from farther away than the approach in this section. When forming the context we do not need to restrict ourselves to any form of linear filtering and we can use any linear or non-linear technique. In any case it might be preferable to recursively evaluate the context information and use the previously computed context in order to evaluate the next one. Such techniques are of great interest since they will speed up the coder significantly [111]. Certain similarities to IIR filtering are also present in the later formulation (Chapter 5).

4.5 Description of the Algorithm

The proposed algorithm can be summarized as follows:

Step 1 Compute the wavelet transform for the whole image and weight all subbands appropriately as described in Section 4.5.4

Step 2 Initialize all probability models to a uniform distribution.

Step 3 Start scanning all the bands from low to high resolution in a predetermined order.

Step 4 When a band is first visited send the *maximum* and *minimum* of its quantized coefficients (normalized with quantization step size Δ)

Step 5 Scan each subband, line by line, left to right top to bottom

Step 6 Classify each new coefficient x into a class γ based on equation (4.25)

Step 7 For the class γ chosen in the previous step, transmit the codeword closest to x in the rate distortion sense, according to equation (4.23).

Step 8 Update the statistics for class γ

Step 9 Continue until all the coefficients in a subband have been scanned. (go to step 6)

Step 10 Continue until all subbands have been scanned. (go to step 4)

Notice that the whole algorithm is simple, as no explicit training is required, the bulk of the complexity comes from computing the wavelet transform and the arithmetic coder, rather than from the quantization itself. The rate distortion formulation makes our encoder/decoder asymmetric, i.e. all the search complexity is at the encoder while the decoder only needs to use the reproduction level chosen by the encoder.

4.5.1 Adaptation Strategies

The issue of initialization of the entropy coders is non-trivial since different images have different characteristics and explicit initialization may require a significant amount of side information.

It is useful to observe that on the top levels of our wavelet decomposition the wavelet coefficient distribution is almost uniform, but as we move towards the bottom levels this distribution gets more and more peaked around zero. We can use the same probability models for the entropy coders throughout the whole pyramidal structure, starting with a uniform distribution on the top level. The distributions "learnt" at a higher level are used to initialize the distribution at lower levels. Thus the statistics are learnt on the fly as we move towards to bottom of our pyramid and no initialization is required for each subband. Whenever a new band is visited we transmit explicitly the *minimum* and *maximum* coefficient in this band. While this is not necessary it is useful because by knowing the dynamic range of our data we reduce the size of our probability tables.

4.5.2 Subband Skipping and Resolution scalability

In the case where in a given band both minimum and maximum are equal to zero we do not transmit any coefficient from this band, we just move on to the next subband. It is worth pointing out that this fact alone gives us a resolution scalable coder. We can skip the subbands corresponding to higher resolution in order to encode/decode a smaller resolution image. This also speeds up our coder, since at low rates some subbands may turn out to have no nonzero value coefficients, and in this case we do not encode any coefficient for the whole subband.

4.5.3 Inverse quantization

The inverse quantization, unlike the widely used dead-zone quantizer, is just a multiplication operation. Let v be the quantized index, the reconstructed value $\hat{\alpha}$ will

be:

$$\hat{\alpha} = v \cdot \Delta \tag{4.27}$$

It is not possible to further decrease the MSE by optimizing the inverse quantization, as is the case for example when using a uniform quantizer. The encoder already optimized the bit stream for the optimal reconstruction points and no better reconstruction points can be found. This is a fundamental difference between our approach and most techniques used up to now in different image coders.

4.5.4 Subband Weighting

Wavelet data in different subbands contribute to the quality of the reconstructed image in a different way. We need to assign a certain number of bits to each subband in a way that an error metric in the reconstructed image is minimized. This corresponds to an *optimal bit allocation* among the subbands. Since we do not have any direct way of controlling the exact number of bits assigned to each subband, we can instead adjust the quantization step size to each subband. We mentioned in the previous sections that we are using a uniform quantizer for all the data, but the size of the quantizer needs to be adjusted properly to each subband. A better solution to adjusting the quantization step size to each subband is to use the same step size through out all the subbands, but weight each subband appropriately. The effect of both methods is the same, but the latter approach is computationally simpler, since the weighting factors can be incorporated into the filter coefficients.

Bit allocation is very important in image compression. Consider for example the case where the coefficients in one subband are multiplied by a large number. Then, our algorithm will spend a significant amount of bits on this subband which is undesirable. We need to introduce appropriate normalization factors for each subband to avoid those situations. Assume that subband k was obtained from the original image after filtering with filters $f_0, f_1, \ldots f_{V(k)}$. These filters could have been applied either vertically or horizontally. Let, ξ_k be the filter corresponding to the convolution

of all the above filters.

$$\xi_k(n) = f_1(n) * f_2(n) \cdots * f_{V(k)}(n) \tag{4.28}$$

The input output relation for the subband data can be modeled as: $y(n) = \xi_k(n) * x(n)$, or in the frequency domain $Y(\omega) = \Xi_k(\omega)X(\omega)$. We want to keep the energy constant through out the entire transform i.e., quantitatively we would like to have $\int_{-\pi}^{\pi} |\Xi_k(\omega)|^2 d\omega = \sum_n \xi_k[n]^2 = 1$. This will keep the mean square errors in the transform domain the same as in the image domain. The normalization factor σ_k to achieve that corresponds to the L_2 norm of ξ_k which is:

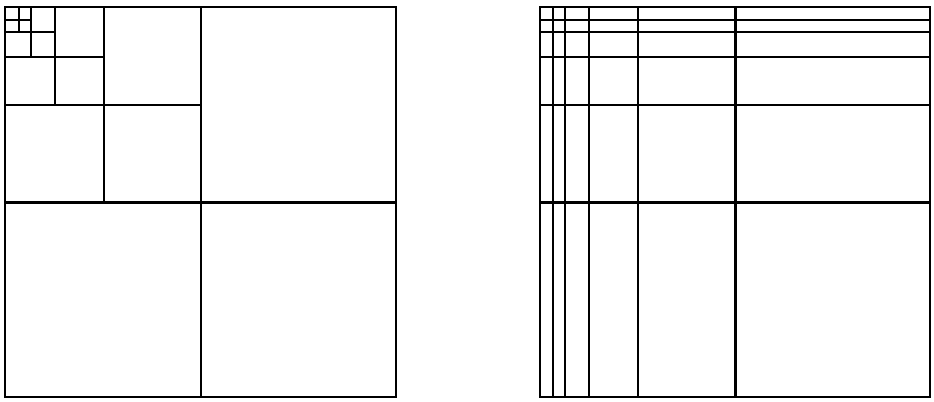
$$\sigma_k = (\sum_{n=-\infty}^{\infty} |\xi_k(n)|^2)^{\frac{1}{2}} \tag{4.29}$$

Thus before applying any form of rate distortion optimization or any quantization we normalize each subband by the corresponding factor σ_k. The normalization factors can also be incorporated into the filter coefficients, in this way we save one multiplication per pixel for the whole image. The normalization gives up the equivalent effect of using an orthogonal filter bank even when that is not the case, more detailed analysis can be seen in [96] .

4.5.5 Effect of the number of classes on performance

In this section we will examine the effect the number of classes in the classification scheme has on the rate distortion curves for different images. Theoretically, for large enough data sets the larger the number of classes the more accurate the statistics derived from the data. The key point is that there should be enough data to learn the statistics for each class. If the number of classes is large the number of samples in each class will be very small and every effort to learn statistics will fail. We thus need to keep a balance between number of classes and number of samples in each class [2]. Another issue of major importance is that data is not stationary. Probability model

[2] The log() operator is particularly useful for this purpose



(a) Dyadic split decomposition (b) Separable decomposition

Figure 4.3: (a) Five level *dyadic* decomposition, or the so called *mallat* decomposition. (b) Five level *separable* decomposition.

estimation tries to keep track of characteristics of the data. In the degenerate case of a large number of classes, statistic estimation may harm performance, as too many classes result in samples in the same class being located far apart in the image, so there might not be much correlation between data in the same class. From a purely complexity point of view we want to restrict ourselves to a small number of classes. The fewer the classes the smaller the amount of memory needed to store the model parameters, which will also translates to speed advantages. We evaluated the PSNR for five different rates for four images, for different number of classes. The results are depicted in Figure 4.4. We observe that the optimal number of classes is slightly larger than ten. For more than 13 classes the results did not change significantly, so in our experiments we used 13 classes.

For images with a lot of high frequencies the PSNR gains obtained with classification are larger when compared to smooth images. Higher order correlation is present in images with a lot of texture, so classification is more likely to work better in regions with a lot of texture. Also note that the gains are higher at high rates. This is due in part to the fact that at low rates the high magnitude classes contain very few coefficients.

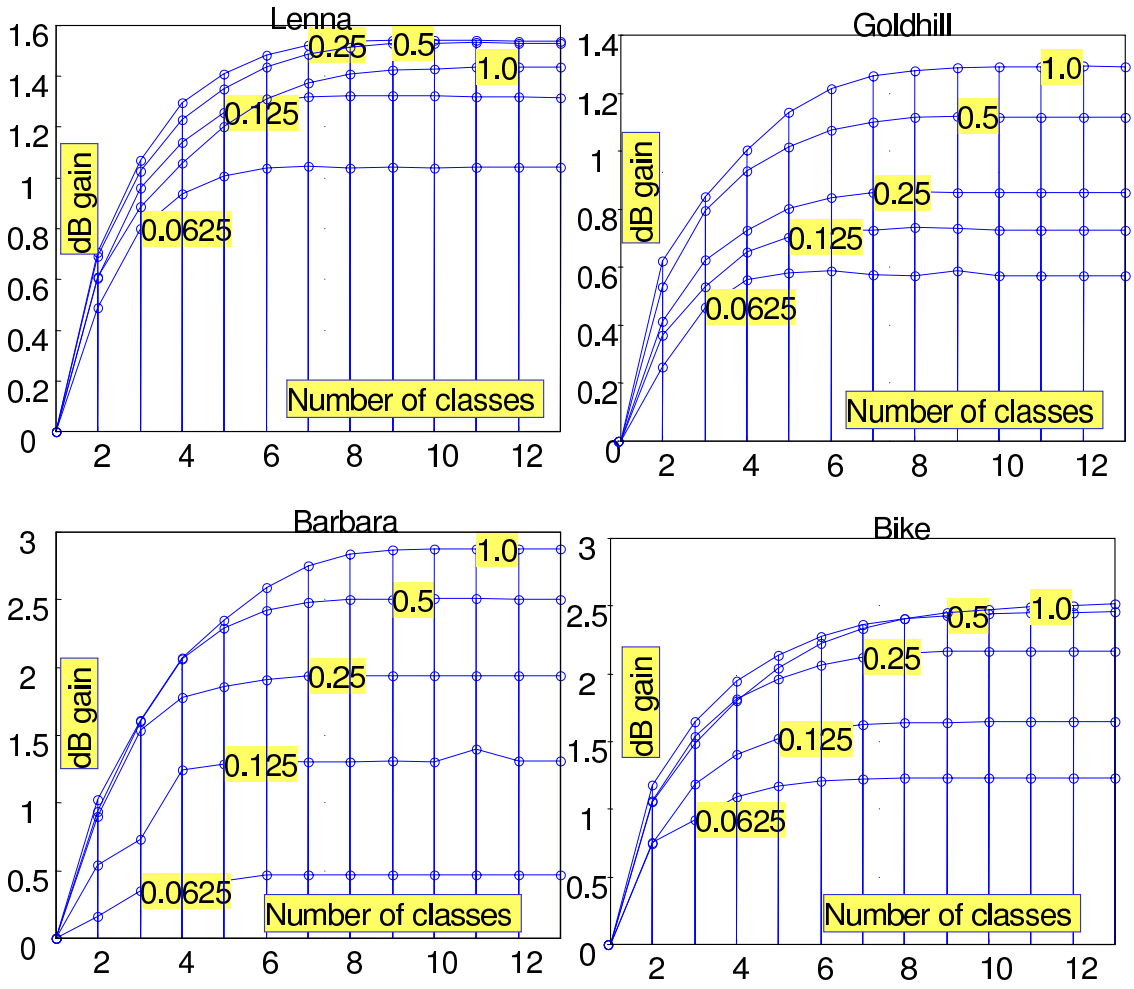


Figure 4.4: Gain in PSNR with respect to the single class as a function of the number of classes. For each image we can see 5 curves corresponding to 5 different rates. The PSNR gain is larger for higher rates. Moreover the gain achieved by increasing the number of classes varies from image to image, from 1dB to up to 3dB.

4.6 Experimental Results

In table 4.1 we present results for two different filter banks and for two different decompositions, as seen in Figure 4.3. For images with a lot of high frequency content the separable decomposition in Figure 4.3(b) is preferable, unlike smooth images where the decomposition in Figure 4.3(a) is better. Generally speaking, for very low rates the decomposition in Figure 4.3(b) is also preferable. We have used two sets of filters the Daubechies 9-7 tap filter, and a 28-28 tap filter bank designed with a program down-loaded from `http://saigon.ece.wisc.edu/~waveweb/QMF/software.html` . The last column in table 4.1 corresponds to the case of forming our context without any information from previous bands. The results in this case are on the average about 0.1dB worse than the case where we use information from previous bands. Our results on average are some of the best reported in the literature. The bit stream is not embedded, so we need several iterations in order to achieve the exact bit rate. The bit stream of the current coder does not have all the features present in a coder like [92], though is conceptually simpler to understand and does not use any form of bit plane coding.

4.7 Future Work

In this chapter we presented a novel approach for wavelet image coding. Based on theoretical analysis of rate distortion theory we developed an encoder with excellent performance. One of the key points of this work is that we do not use a bit plane coder but instead a whole coefficient coder. We used arithmetic coding through out our work because of the ease of adaptivity. Non adaptive entropy coders would significantly increase the speed of our system. The number of classes was limited because of the context dilution phenomenon. If we do not adapt our entropy coder we can increase the number of classes to virtually any number. Such a system would be of great interest both from theoretical point of view, given the challenge of designing or modeling our fixed probability models, but also from practical point of view, since

it would offer an extremely simple solution to wavelet image coding.

	Rate	SPIHT [83]	VM 3.2A [92]	RD 9-7 (4.3a)	RD 28-28 (4.3a)	RD 9-7 (4.3b)	RD 28-28 (4.3b)	RD 9-7 (4.3a*)
Lena	0.125	31.10	31.22	31.32	**31.44**	30.89	31.07	31.25
512 × 512	0.25	34.13	34.28	34.45	**34.58**	33.96	34.12	34.38
	0.50	37.24	37.43	37.60	**37.69**	37.11	37.31	37.53
	1.00	40.45	40.61	40.86	**40.91**	40.34	40.30	40.78
Barbara	0.125	24.84	25.55	25.39	25.75	26.45	**26.91**	25.41
512 × 512	0.25	27.57	28.55	28.32	28.87	29.40	**30.00**	28.37
	0.50	31.39	32.48	32.29	33.01	33.17	**33.80**	32.25
	1.00	36.41	37.37	37.40	37.98	37.82	**38.39**	37.21
Goldhill	0.125	28.47	28.49	28.61	28.63	28.69	**28.73**	28.60
512 × 512	0.25	30.55	30.71	30.75	30.77	30.77	**30.86**	30.73
	0.50	33.12	33.34	33.45	**33.50**	33.35	33.48	33.43
	1.00	36.54	36.72	36.95	**37.03**	36.77	36.94	36.93
Bike	0.125	25.82	26.49	26.16	26.20	26.39	**26.62**	26.03
2560 × 2048	0.25	29.12	**29.76**	29.43	29.57	29.39	29.61	29.39
	0.50	33.00	**33.68**	33.47	33.46	33.02	33.29	33.37
	1.00	37.69	**38.29**	38.27	38.19	37.45	37.72	38.16
Woman	0.125	27.27	27.46	27.67	**27.78**	27.60	**27.78**	27.63
2560 × 2048	0.25	29.89	30.15	30.36	**30.49**	30.16	30.35	30.33
	0.50	33.54	33.81	34.12	**34.22**	33.76	33.94	34.04
	1.00	38.24	38.67	38.92	**38.97**	38.46	38.65	38.85

Table 4.1: Comparison between our method and [83, 92] for images: Barbara, Lena, Goldhill, Bike and Woman, the last two images are part of the test images for JPEG2000. We used five levels of vertical decomposition with the two decompositions in figure 4.3. We have also used two different filter banks, the 9-7 Daubechies and a 28-28 tap filter bank. In the last column (*) we did not use the parent bands to form context information. Moreover we used a recursive context formation that will be described in Chapter 5.

Chapter 5

Line Based Wavelet Transform and Coding

Contents

This chapter addresses the problem of low memory wavelet image compression. While wavelet or subband coding of images has been shown to be superior to more traditional transform coding techniques, little attention has been paid until recently to the important issue of whether both the wavelet transforms and the subsequent coding can be implemented in low memory without significant loss in performance. We present a complete system to perform low memory wavelet image coding. Our approach is "line-based" in that the images are read line by line and only the minimum required number of lines is kept in memory. The main contributions of our work are two. First, we introduce a line-based approach for the implementation of the wavelet transform, *which yields the same results as a "normal" implementation*, but where, unlike prior work, we address memory issues arising from the need to synchronize encoder and decoder. Second, we propose a novel context-based encoder which requires no global information and stores only a local set of wavelet coefficients. This low memory coder achieves performance comparable to state of the art coders at a fraction of their memory utilization.

5.1 Introduction

Memory is an important constraint in many image compression applications. In some cases, especially for mass market consumer products such as printers or digital cameras, this is due to the need to maintain low costs. In other cases, even if sufficient memory is available (e.g., image encoding/decoding in a PC or workstation), inefficient memory utilization may limit scalability and hinder overall performance. For example, if for a given algorithm doubling of the image size results in doubling of the memory requirements, the practicality of using the algorithm over a wide range of systems may be questionable.

Existing Discrete Cosine Transform (DCT) based compression algorithms such as those defined under the JPEG standard [71] are very efficient in their memory utilization because, if needed, they can operate on individual image blocks and thus the minimum amount of memory they require is low indeed (e.g., a system could conceivably be implemented so as to manipulate a single image block at a time). Wavelet based coders have been shown to outperform DCT based coders in terms of compression efficiency, but their implementations have not yet reached the stage of maturity of DCT based approaches [71]. Memory efficiency is in fact one of the key issues to be addressed before a widespread deployment of wavelet based techniques takes place and it is currently one area of major research activity within the JPEG2000 standardization process [59].

Algorithms such as those in [8, 16, 52, 62, 83, 85, 112, 115], are representative of the state of the art in wavelet coders. All of these algorithms assume that the wavelet transform (WT) for the whole image has been computed so that all the corresponding coefficients are available in the coding process. Global image information[1] is used in various ways including, among others, classification (e.g., [52]), initialization of the probability models used by a quantizer or arithmetic coder [16, 62, 115] or selection of specific decompositions of the signal [112]. It is also worth noting that even if no

[1] i.e., information that can only be obtained after the whole image has been transformed. Examples of global information include the maximum and minimum coefficient values in one subband, the energy per subband, histograms of coefficient values in a subband, etc.

global information has to be measured, algorithms that provide progressive transmission [83,85] might require to store the complete set of wavelet coefficients. The above mentioned algorithms were developed with the goal of achieving competitive compression performance and thus memory utilization was not a major consideration in their development. These algorithms are typically required to buffer the whole image at the encoder, so that memory usage increases proportionally to the image size, without such factors as filter length, or the number levels of the wavelet decomposition affecting significantly the memory utilization.

Low memory implementations of wavelet transforms were first addressed in [102], which only considered one-dimensional (1D) transforms and did not consider the synchronization issues that arise when both forward and inverse transform memory requirements are considered. Interest in memory issues has recently increased as memory needs for the WT have been found to be the one of the main bottlenecks for wavelet-based image compression. There have been several recent studies of hardware issues in the implementation of the WT [11,33]. Many of these studies consider an in-depth analysis of the whole wavelet transform system, including architecture level optimizations and memory usage as in [11], but do not consider the joint design of transform and compression algorithm to guarantee low memory operation. In addition, much of this work has focused on video compression implementations, where images can be orders of magnitude smaller than some of those processed in the hard-copy industry, and thus the proposed approaches might not scale well to large image sizes. For example, typical designs consider an on chip memory to handle the filtering operations [2] and such memory can become prohibitively large when images of hundreds of millions of pixels or more have to be processed.

Assume that a particular WT implementation can handle small images efficiently. Obviously there are approaches to use such an implementation for wavelet coding of large images. The most immediate approach is to tile the large image and encode each tile independently of the others, i.e., as if each tile were a separate image. While tiling is a simple approach it can present some serious drawbacks, especially if the tile size is small with respect to the image size. For example as compression is

performed independently, blocking artifacts may appear at the boundaries between tiles. Moreover, since it is not easy to allocate bits among the tiles (since the wavelet coefficients of all the tiles are not known and each tile is treated independently) the performance degradation due to tiling may be severe.

An alternative and more efficient approach can be found in the recent work of Cosman and Zeger [28,29]. Here, the memory utilization of the encoder is left unchanged and a standard algorithm (e.g. [83]) can be used to compress the image. The whole image is buffered at the encoder, but the order of transmission of the bit-stream is altered from that used in a normal implementation so that the memory utilization at the decoder is reduced. The basic idea is to have the decoder receive "local" sets of encoded wavelet coefficients so that the inverse wavelet transform can be started without having to wait for all the coefficients to be decoded. Performance can be further improved by selecting filters so that the number of coefficients required at the decoder remains small (this can be achieved for example by choosing shorter filters for filtering along the vertical direction.) We refer the reader to [28, 29] for further details.

In summary, the recent work on memory efficient wavelet image coding does not consider a complete coding system but concentrates instead on the WT or the compressor alone, or only considers either encoder or decoder, but not both.

Our proposed approach differs from earlier work in several ways. First, we consider the overall memory utilization and propose a system with reduced memory *at both encoder and decoder*, whereas [102] and [28, 29] addressed only the memory usage at encoder and decoder, respectively. We thus consider the memory needed for synchronization between encoder and decoder (e.g. a minimum memory forward wavelet transform may require a high memory inverse wavelet transform). Second, we consider a complete coding system, i.e., including both WT and quantization plus entropy coding, and propose a novel context-based compression approach to provide high compression performance with reduced memory utilization. With respect to existing coders, the degradation in performance is modest (e.g., less than $0.5dB$ in average with respect to [16]) although we do not support progressive transmission

as [83, 85]. In terms of memory utilization our results show reductions of almost *two orders of magnitude* with respect to widely available implementations of wavelet image coders. In fact, our proposed low memory WT implementation has been adopted within the latest version of the JPEG 2000 verification model.

Our proposed system includes a line-based implementation of the WT, where we assume the image data is available one image line at a time. We begin by analyzing the minimum memory requirements to compute the wavelet transform in Section 5.2. Analyzing the 1D WT allows us to discuss the various ways in which memory is utilized, including filtering but also synchronization between encoder and decoder. We extend these analysis to the two-dimensional (2D) WT and propose an implementation that requires the minimum number of image lines to be stored for a given filter length and number of levels of decomposition. With this approach the memory needs of the encoder and decoder depend only on the width of the image (rather than the total size as in a traditional row column filtering implementation) which significantly enhances the scalability of the system. Moreover, appropriate choices of filters (e.g., short filters for the vertical filtering) can be used as in [28, 29] to further reduce the memory requirements.

In Section 5.3 we then propose a backward adaptive context-based coding scheme which utilizes *only* a reduced number of coefficients stored at the encoder or decoder at a given time. While this approach precludes the use of any global information we show that competitive performance can be achieved because, as has been shown in [16, 52, 62, 115], there exists significant localization of energy within wavelet subbands, which can be efficiently exploited with context-based methods. We provide a complete description of our algorithm and highlight the modifications that had to be undertaken with respect to our earlier context-based coding approaches [16] in order to compensate for the lack of global information.

Our experimental results are presented in Section 5.4, where we include comparisons with several algorithms [8, 16, 71, 83], which all have significantly larger memory requirements. Our results indicate that the low memory approach we propose can

achieve excellent compression performance with significantly lower memory utilization. In section 5.5 we revise the main contributions of our work.

5.2 Line-based 2D wavelet transform

Image data is usually acquired in a serial manner. For example, a very common way to acquire image data is to scan an image one line at a time. Throughout this chapter we will assume our system operates with this line-by-line acquisition. Given this, our objective in this section will be to design a 2D, WT that requires *storing a minimum total number of lines.* The assumption is that images are stored in memory only while they are used to generate output coefficients, and they are released from memory when no longer needed.

Obviously, performing a 1D WT on a single line can be done without significant memory. However, in order to implement the separable 2D transform the next step is to perform column filtering and here memory utilization can become a concern. For example, a completely separable implementation would require that all the lines be filtered before column filtering starts and thus memory sizes of the order of the image size will be required. The obvious alternative is to start column filtering as soon as a sufficient number of lines, as determined by the filter length, has been horizontally filtered. For example for a one level decomposition if we use 9-7 tap filters we only need 9 lines of the image in order to start the column filtering and generate the first line of output wavelet coefficients.

This online computation approach, which is described in more detail in [102] for the 1D case, will form the basis of our wavelet filtering. This will allow us to store in memory only a reduced number of input lines. The memory needs, as will be discussed in what follows, depend not only on the filter length but also on the number of levels of decomposition and the type of decomposition. For example generic *wavelet packet* [77] decompositions require different structures than a *dyadic tree* decomposition and need to be considered separately. In addition, the order in which lines of wavelet coefficients are generated at the analysis filter bank is not the same order the synthesis filter-bank

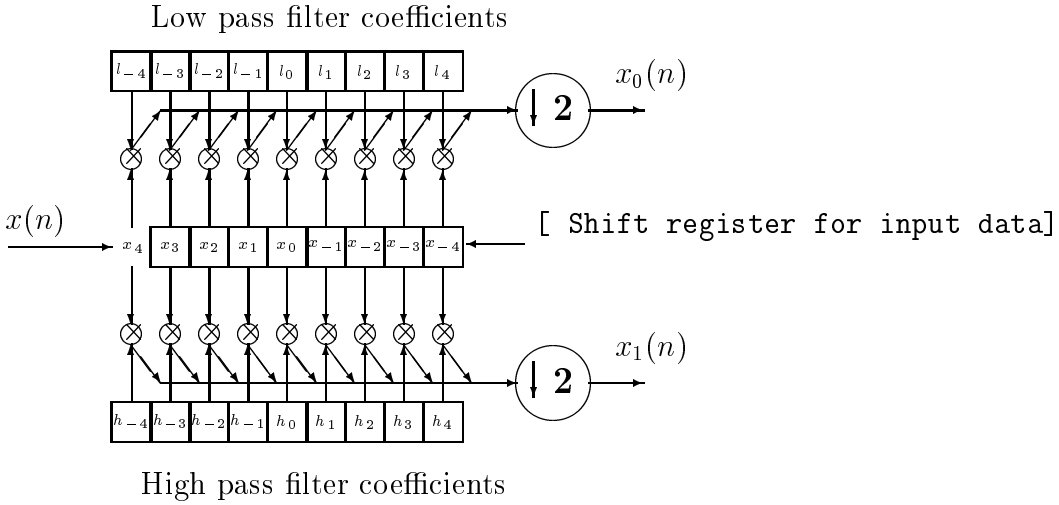


Figure 5.1: One level decomposition for filters of length $L = 2S + 1 = 9$. We need eight memory elements, which are represented by a shift register. The delay between input and output is $S = 4$ samples.

expects them and we will thus need to introduce some additional "line management" functionality to synchronize encoder and decoder.

5.2.1 One dimensional wavelet transform

Let us consider first an implementation of a 1D WT, where the system receives data sequentially, one pixel at a time. Let $L = 2S + \phi$ be the maximum length of the filters used in the analysis filter-bank, which can be either odd or even, for $\phi = 1$ and $\phi = 0$, respectively. In the next sections, without loss of generality, we will only consider the odd length filter case. The even length filter case can be treated in a similar way. For compression efficiency we use symmetric extensions throughout this chapter. Similar delay analysis can be found in [102], although synchronization delays were not considered there.

Consider first a single stage of the WT. At time zero we start receiving data and store it in a shift register as seen in Figure 5.1. At time S we have received enough data to fill the entire input buffer, i.e., we have received $S+1$ samples and after symmetric extension we can have the total of $L = S+1+S$ samples we need for filtering. Thus,

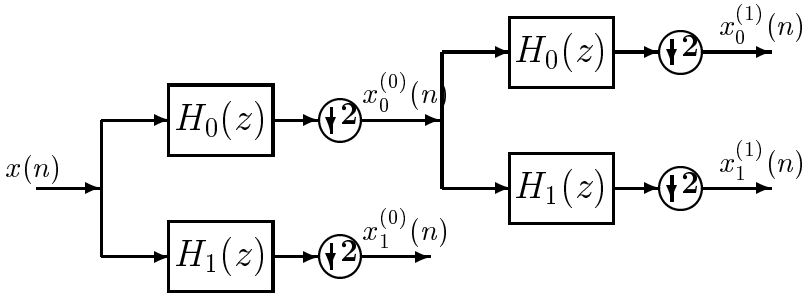


Figure 5.2: Cascade of two levels of wavelet decomposition. The delay for the first level is S, the delay for the second level is $2S$ and the total delay for both levels is $S + 2S$. The time units for the delay are considered in the input sampling rate.

the *delay* for generating output coefficients for one level of decomposition is S. Note that it is more efficient to generate two coefficients at a time as shown in Figure 5.1, i.e., we read two input samples at a time and generate both a low-pass and a high pass coefficient.

Consider now a two-level decomposition, as shown in Figure 5.2. Let ρ be the sample rate (in samples per second) at the input of the filter-bank. Then the sample rate at the output of the first level will be $2^{-1}\rho$. Let us consider the interval between input samples (i.e., $1/\rho$ seconds) as our basic time unit. Each individual filter-bank introduces an S sample delay. However the input to the second filter-bank arrives at a lower rate $2^{-1}\rho$, due to down-sampling. Thus to begin filtering and see the first outputs of the second filter-bank we will have to wait (i) S time units for the first output of the first level filter-bank to be generated, and then (ii) another $2S$ time units until sufficient samples have been generated at rate $2^{-1}\rho$. Thus, the total delay from input to output of the second level in the system of Figure 5.2 is the sum of those individual delays, i.e., $S + 2S$ input samples.

This analysis can be easily extended to the the case where an N-level decomposition is used. Assume that the levels in this decomposition are indexed from 0 to $N - 1$, where 0 corresponds to the first level, 1 corresponds to the second level, and so on. It will take $2^n S$ time intervals for S samples to be loaded in the n^{th} level filters and this will happen only after outputs have been generated by levels 0 to $n-1$. Thus

the total delay from the input to the output of an N level filter-bank will be the sum of all the individual delays for each level, $D_N = S + 2S + 2^2S + 2^3S + \cdots + 2^{N-1}S = \sum_{k=0}^{N-1} 2^k S = (2^N - 1)S$. The delay from the n^{th} level to the output will be the same as the delay from the input to the output of an $N - n$ level decomposition, i.e., $D_{n,N} = D_{N-n} = (2^{N-n} - 1)S$.

The memory needed for *filtering* will be L samples[2] for each level of decomposition, i.e., the total will be $L \cdot N$ if we use a dyadic tree decomposition. In general we will just need an additional memory of size L for each additional 2-channel filter-bank added to our wavelet tree.

In the synthesis filter-bank the delays from input to output are the same as in the analysis filter-bank and are thus a function of the number of levels of decomposition. Note that, referring to Figs. 5.2 and 5.4, the synthesis filter-bank will not be able to process $x_1^{(0)}$ until it has processed a sufficient number of $x_0^{(1)}, x_1^{(1)}$ coefficients to generate $x_0^{(0)}$. However the analysis bank generates $x_1^{(0)}$ with less delay than $x_0^{(1)}, x_1^{(1)}$. Thus we will need to store a certain number of $x_1^{(0)}$ samples while the $x_0^{(1)}, x_1^{(1)}$ samples are being generated. We will call the required memory to store these samples *synchronization* buffers.

Because $2S$ samples at level 1 are produced before the first sample at level 2 is produced, we will need a synchronization memory of $2S$ samples (see Figure 5.4). The required memory can be split into two buffers of size S pixels, with one buffer assigned to the analysis filter-bank and the other to the synthesis filter-bank.

In the more general N-level case the delay for samples to move from level n to level $N - 1$ is D_{N-n}. The synchronization buffer for level n is equal to the delay for data to move from level n to level $N - 1$ which is also D_{N-n}, thus the total buffer size needed for *synchronization* is $T_N = \sum_{k=0}^{N-1} D_{N-k} = \sum_{k=1}^{N} D_k = (2^N - N - 1)S$.

[2]Implementations with memory sizes of $S + 1$ samples (or lines) are also possible, but here we assume storage of L lines to facilitate the description of the synchronization problems. More efficient approaches based on lifting implementations or lattice structures, can asymptotically bring the memory needs from L down to $S + 1$. For example the lattice structure used in [33] can help in reduce both complexity and memory. These structures will not be considered here since the memory savings are filter dependent and do not significantly affect our proposed solution. Lifting or lattice structures can be used within our framework to provide additional memory reductions.

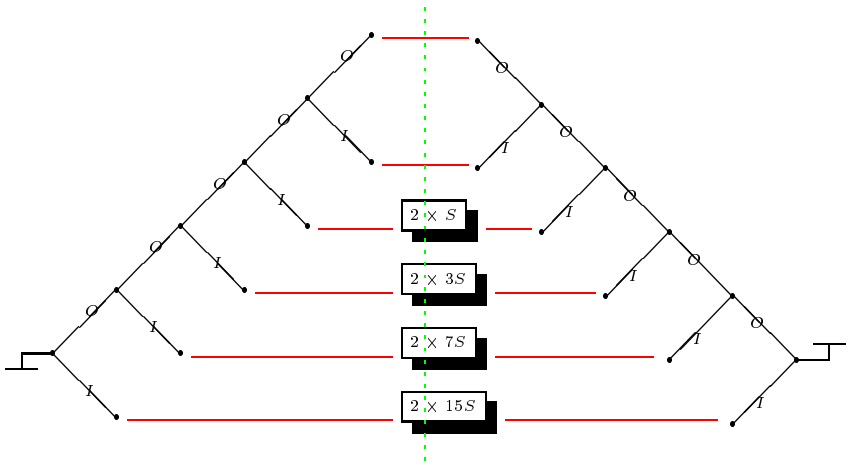


Figure 5.3: One dimensional analysis and synthesis decomposition trees, the delay involved in the analysis/synthesis filter banks is depicted, the memory needed increases exponentially with the number of levels. The major factor for memory is the syncronization buffers and not the filtering buffers.

For a five level decomposition the size of the synchronization buffers can be seen in Figure 5.3.

As a conclusion, for the 1D case when considering both analysis and synthesis banks, our design will have to optimize memory for both *filtering* and *synchronization*. In the above analysis we have kept analysis and synthesis filter-banks symmetric in terms of memory needs. However, synchronization buffers can be easily assigned to either the analysis or the synthesis filter-banks if it is necessary to make one more memory-efficient than the other. In the rest of the chapter we will only consider symmetric systems. In summary, for 1D signals and N levels of decomposition, the memory needs for the n^{th} level will consist of

- a filtering buffer of size L, and
- a synchronization buffer of size $D_{N-n} = (2^{N-n} - 1)S$.

Therefore the total memory size needed for N levels of decomposition in a symmetric system is: $T_{total,N} = (2^N - N - 1)S + NL$. Note that as the number of levels becomes large synchronization buffers become a major concern.

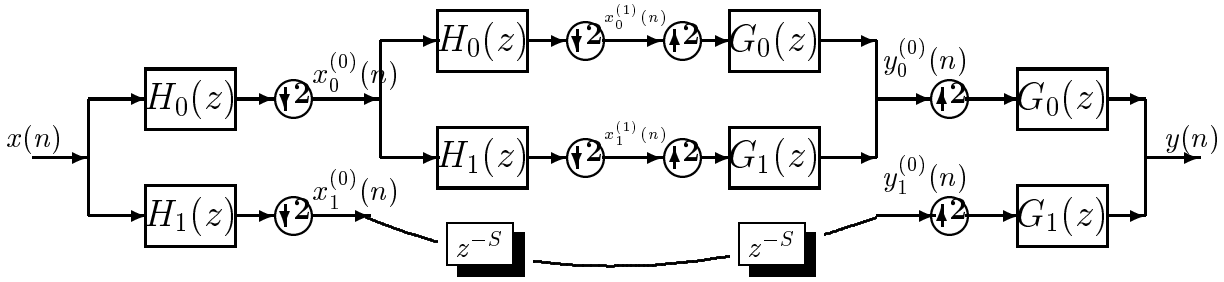


Figure 5.4: Consideration of both synthesis and analysis filter banks reveals the need for syncronization buffers. Two buffers z^{-S} are needed to form a queue for the $x^{(0)}(n)$ samples.

5.2.2 Two dimensional wavelet transform

Let us now generalize our memory analysis to two dimensions. As a simplifying assumption we assume that horizontal filtering is performed in the usual manner, i.e., our memory budget allows us to store complete lines of output coefficients after horizontal filtering. Thus, after each line is received all the corresponding filter outputs are generated and stored in memory, requiring a memory of size X for each line, where X is the width of the image. Thus we can now apply the above analysis to the vertical filtering operation, except that the input to the WT is now comprised of *lines* of output coefficients generated by horizontal filtering and thus the memory sizes shown above have to be adjusted to account for line buffering requirements.

The exact memory requirements depend on the structure of decomposition. Figures 5.5(a) and (b) depict the common *dyadic tree* decomposition and a "hybrid" decomposition, which have the same memory requirements. Let us concentrate on the decomposition of Figure 5.5(a). In order to implement a one level decomposition vertically we need to buffer L lines[3]. At the second level of the decomposition again we will need to buffer L lines, but the length of each line will be $X/2$ coefficients, because in the dyadic composition the second level decomposition is only applied to

[3]As indicated before, implementations with only $S+1$ lines are possible. They are not considered here since the memory savings in this case come at the cost of a more complicated system.

the low pass coefficients generated by the first level. Thus the width of our image is reduced by two each time we move up one level in the decomposition, and, correspondingly, the memory needs are reduced by half each time. For N levels we will need $\sum_{k=0}^{N-1} 2^{-k}L = 2(1-2^{-N})L$ "equivalent" image lines for filtering (refer to Figure 5.6). As N grows the required number of lines tends to $2L$, and the corresponding memory becomes $2LX$ i.e., asymptotically we only need a number of lines equal to twice the filter length. The above analysis is valid for both encoder and decoder as long as we use the decompositions in Fig. 5.5.

As in the 1D case, synchronization issues have to be addressed because coefficients from the first level of decomposition are generated by the analysis bank before coefficients from higher levels, while the synthesis filter-bank requires the higher level coefficients before it can use the coefficients from the first level. For example in a two-level decomposition we will need to store the $\mathrm{HL}_0, \mathrm{LH}_0, \mathrm{HH}_0$ bands[4] since these bands become available before the $\mathrm{HL}_1, \mathrm{LH}_1, \mathrm{HH}_1$ bands, and the synthesis filter-bank has to start processing data from the second level before it can process data from the first level.

Thus, as in the 1D case, in an N-level decomposition the synchronization delay required for data at level n is $D_{N-n} = (2^{N-n}-1)S$ lines[5], where the width of each line is the width of a subband at a particular level, e.g., the width of one line at level n will be $2^{-n-1}X$, due to down-sampling and the use of a dyadic decomposition. Because 4 bands are generated at each level, but only one (i.e. the LL band) is decomposed further, we will need synchronization buffers for the remaining three subbands. Thus the synchronization buffers for level n will have a total size of $3 \cdot (2^{N-n}-1)S \cdot 2^{-n-1}X$ pixels and the total memory needed for synchronization will be:

$$T_N^{(2d)} = 3\sum_{k=0}^{N-1}(2^{N-k}-1)SX2^{-k-1} = (2 \cdot 2^N + 2^{-N} - 3)XS \tag{5.1}$$

From (5.1) we see that the size of the delay buffer increases exponentially with the

[4]Index 0 corresponds to the first level of decomposition

[5]Note that we express this delay in terms of number of input lines, instead of pixels

number of levels, while as discussed before the memory needed for filtering is upper bounded by $2LX$ pixels. Thus, as the number of decomposition levels grows, the filtering requirements remain relatively modest, while the size of the synchronization buffers tends to grow fast. However, memory-efficient implementations are still possible because the filtering buffers hold data that is accessed multiple times, while the synchronization buffers are only delay lines (FIFO queues). This is a key distinction because, as will be described later, the data in the synchronization buffers will only be used by the decoder and therefore it can be *quantized and entropy coded* so that the actual memory requirements are much lower.

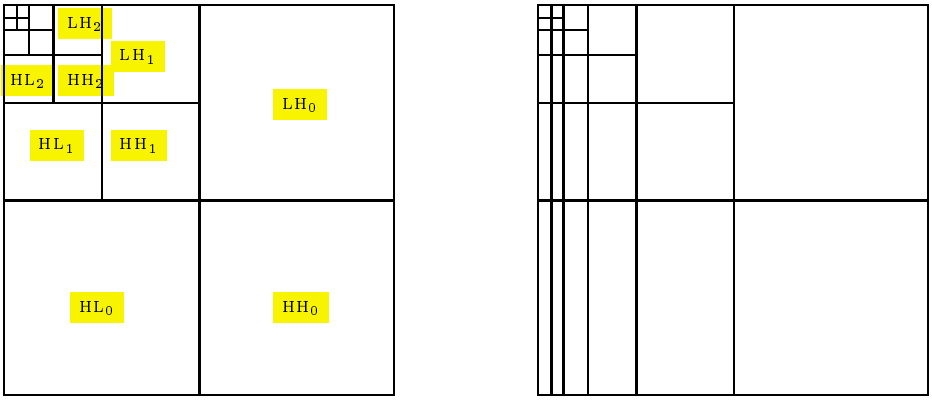
In summary, for an N-level dyadic decomposition, we will need

- filtering buffers for up to $2LX$ pixels, and
- synchronization buffers for $T_N^{(2d)} = (2 \cdot 2^N + 2^{-N} - 3)XS$ pixels.

The decomposition of Fig. 5.5(b) can be implemented with the same memory as that of Fig. 5.5(a). For the case of Fig. 5.5(b), we perform five levels of horizontal decomposition when we first read a line and we skip horizontal filtering in all the subsequent decomposition levels in Figure 5.6. This decomposition gives better compression results for images having a significant part of their energy in the high frequencies. For a specific wavelet packet decompositions the exact structure of a line-based implementation would have to be determined on a case-by-case basis, and thus the above formulas do not apply. However, special decompositions such as the one in Figure 5.5(b), having the same memory requirements as the simple "dyadic tree" decomposition, can provide some of the gains of a wavelet packet decomposition, while being memory efficient.

5.2.3 Example

To illustrate our WT algorithm let us consider an example, with $N = 5$ levels of decomposition. Refer to Figure 5.6. After filtering, data from decomposition level n are passed on to decomposition level $n + 1$. Assume we are at level 2 and we



(a) Dyadic tree decomposition (b) "Hybrid" decomposition

Figure 5.5: (a) Five level dyadic tree decomposition. The multiplications needed for the whole decomposition are $\frac{8}{3}XYL$. (b) Five levels decomposition in the horizontal direction for all lines, followed by a dyadic decomposition in the vertical direction. The memory needs are the same as in the previous case, the multiplications needed are $\frac{10}{3}XYL$

are receiving data from level 1. Each time we receive a line we perform horizontal filtering and we store the data into a circular buffer that can hold L lines. After we have received $S+1$ lines we can perform vertical symmetric extension inside the buffer. We will end up with L lines that have already gone through a horizontal decomposition and we are then ready to perform vertical filtering. We can generate 2 output lines at once since we have all the necessary input lines. One line will have vertical low pass information and the other will have vertical high pass information. Moreover, half the coefficients in each line will contain horizontal low pass information and the other half will contain horizontal high pass information. As output we have four lines $\mathrm{LL}_2, \mathrm{LH}_2, \mathrm{HL}_2, \mathrm{HH}_2$ of length half the length of our vertical buffer at this level. The LL_2 line needs to go to the next decomposition level, i.e., level 3, while the lines $\mathrm{LH}_2, \mathrm{HL}_2, \mathrm{HH}_2$ need to go through the synchronization buffer FIFO_2. The width of each input line for level n is $2^{-n}X$, while the width for each output line is $2^{-n-1}X$. This process continues for each level of decomposition, up to the last level (level $N-1$).

5.2.4 Speed advantages

The WT implementation described in the previous sections provides significant memory savings, as compared to a "naive" row column filtering implementation, which will require a memory size of the order of the image size. This memory efficiency is advantageous also in terms of computation speed. Obviously, the number additions and multiplications is exactly the same in our line-based implementation as in the row column filtering implementation. However in image compression applications we frequently deal with large images such that the whole image does not fit into the processor cache memory. If the cache memory is insufficient all the image pixels will have to be loaded into cache several times in the course of the WT computation. For example, in a software environment the operating system and the processor are responsible for loading and unloading certain parts of the whole image into cache, and a memory inefficient approach will result in increased memory management overhead. In the naive approach, output coefficients generated by horizontal filtering will have to be removed from the cache, then reloaded when column filtering is performed. Instead, in our proposed system, the enforced "locality" of the filtering operations makes it more likely that strips of the image get loaded into the cache only once [6]. This fact alone reduces the traffic through the buses of the processor and cuts the bandwidth for memory access by orders of magnitude. Cache memory is around 5 times faster than main memory, so we expect speed ups of around 5 times by using our approach. In software implementations these were indeed our observations in simulations. Also the larger the size of the image the greater the speed advantages offered by our algorithm. No optimization was performed in our code and yet to the best of our knowledge our algorithm provides the fastest currently available software implementation.

[6] Of course, depending of the size of the cache relative to the image size, we might need to load them more than once.

5.3 Low memory entropy coding

5.3.1 Desired characteristics of a low memory compression scheme

In order to use our low-memory WT for an image compression application, and still keep the memory needs low, we need to ensure that wavelet coefficients are compressed soon after they have been generated. Wavelet coefficients are generated line by line in an interleaved fashion (i.e., as mentioned earlier, filtering operations generated both high-pass and low-pass data for a given input line), and therefore it will be useful to be able to encode the data in the order it is generated. Obviously, buffering all the coefficients from a certain band before they are coded increases memory requirements and should be avoided. Thus a low-memory encoder should be able to code data as soon as it becomes available, buffering up only a few lines before encoding (rather than entire subbands) and avoiding having to go through the wavelet coefficients more than once.

It should be noted that if providing an embedded bit stream is required it will be necessary to perform several passes through the data and thus the memory requirements will be larger. An embedded coder will typically send the most significant bits of *all* the wavelet coefficients, whereas a memory efficient approach would tend to transmit coefficients (down to maximum level of significance) as they are produced. If an embedded bit-stream is desired then it will be necessary to store all the wavelet coefficients (so that the most significant bits of all coefficients can be sent first). Alternatively, with the appropriate bit-stream syntax, it may be possible to generate an embedded bit-stream by first storing a compressed image and then reordering the bit-stream before transmission (as in [92]). In either case, an embedded output requires more buffering than our proposed approach.

As indicated earlier, to reduce the synchronization memory requirements, it will be preferable to store compressed data in those buffers *after* compression. This is clearly advantageous since wavelet coefficients are usually kept in floating point format (32

bits) while after compression they can be stored with about one bit per coefficient on average. Thus one should keep in compressed form as much data as possible and avoid buffering uncompressed wavelet data.

5.3.2 Line based entropy coder

We now describe our proposed low-memory coding strategy, based on context modeling and classification along with arithmetic coding and probability estimation. Processing is based on the lines of wavelet coefficients produced by the WT. Each band is encoded *separately*, i.e. we do not use *any* kind of information from one band in order to encode another. Moreover no global information is required. Our purpose is to demonstrate that a line-based transform combined with a line-based coder can be competitive in terms of compression performance, at a fraction of the memory requirements of a more general algorithm like [8, 16, 62, 83, 85].

In the rest of the chapter we will assume that all subband data are quantized with the same *dead-zone quantizer*, that is the step size δ of the quantizer is the same for all subbands. The quantization operation is a mapping from a wavelet coefficient α to the index $v = \lfloor \frac{\alpha}{\delta} \rfloor$. The inverse quantization is a mapping from the index v to an estimate $\hat{\alpha}$ of the original wavelet coefficient α.

$$\hat{\alpha} = \begin{cases} (v + 1/2) \cdot \delta & \text{if} \quad v > 0 \\ (v - 1/2) \cdot \delta & \text{if} \quad v < 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

We assume appropriate normalization of the filter coefficients, as discussed in [16], in order to compensate for the fact that the biorthogonal filter-banks we use here (to take advantage of the symmetry properties) do not have norm one. This normalization allows us to use the same quantization step size for each band.

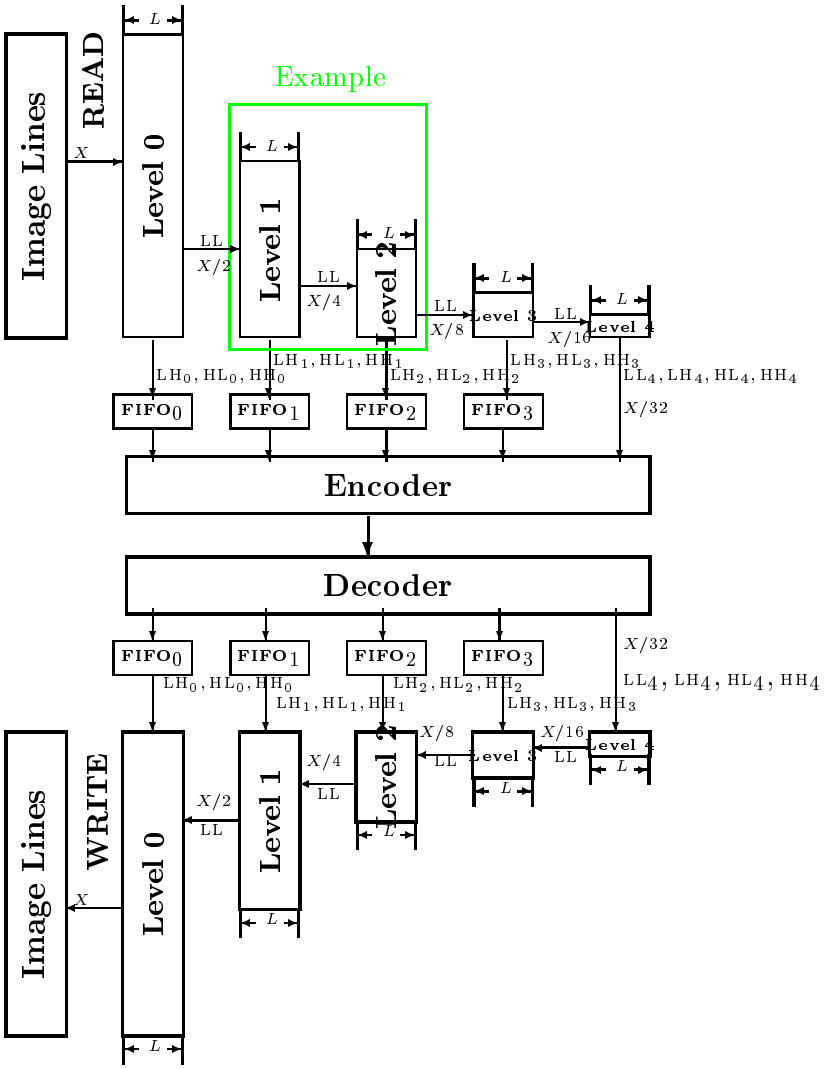


Figure 5.6: Full system with analysis filter bank, encoder, decoder and synthesis filter bank, we consider 5 levels of decomposition. The FIFO buffers can become part of the encoder and decoder, in order to reduce the total memory size. Encoder and decoder need to be able to work with each band independent of the others. The filtering buffer for level n consists of L lines of length $2^{-n}X$. The data flow is as follows: we read image data, pass through the filtering blocks for the appropriate levels, send data to the encoder and inverse the process in order to reconstruct the image.

Introduction to Context Modeling

Context modeling is used in various forms in many compression algorithms. It has been the central component of lossless compression algorithms like [47, 104, 109], and is also widely applied in lossy compression environments [16, 62, 115]. We define the context information for a coefficient as the information we can obtain from the neighboring previously quantized and encoded coefficients[7]. Obviously context information is only useful if the data exhibits some sort of correlation, as is the case when considering wavelet coefficients obtained from filtering natural images. Consider for example a one dimensional signal $x_0, x_1, x_2, \ldots$. Context information corresponding to a coefficient x_n can be any function of the form $\sigma_n = f(x_{n-1}, x_{n-2}, x_{n-3}, \ldots)$. Usually we only consider a small finite window, that is, the number of arguments for the function $f()$ is small. The context information σ_n can be useful as a general form of prediction for x_n, where our objective is not to predict the value of x_n itself, but rather to characterize the distribution of x_n given σ_n. For example, if $\sigma_n = |x_{n-1}| + |x_{n-2}| + |x_{n-3}|$, i.e, is the sum of the magnitudes of three previously quantized coefficients, then small values of σ_n may correspond to a probability function for x_n that is highly peaked around zero (i.e. another small coefficient is likely), while larger values of σ_n may correspond to an almost uniform distribution for x_n (i.e. large magnitude wavelet coefficients will tend to appear in clusters). Note, that if each value of σ_n had to be assigned a different probability model, the system would quickly become impractical (excessive number of models as compared to the amount of data would result in context dilution.) Thus, in practice, a finite number of *classes* or *probability models* $\mathcal{N}$ is used so that a given x_n is assigned to one of these $\mathcal{N}$ classes depending on the context. Since the possible values of σ_n may be very large, selecting the $\mathcal{N}$ classes is similar to quantizing σ_n into $\mathcal{N}$ discrete values.

[7] We assume that only causal contexts are used to avoid having to send any side information to the decoder.

Context Modeling and low memory

We now describe a low memory context modeling approach that is well suited for our line-based transform, refer to Fig. 5.7. For each subband we keep only one line of context information, where the context contains both sign and magnitude information from previous lines. As a context we will use a *weighted average* of all previous lines of wavelet coefficients. This approach is a good compromise that allow us to maintain low memory while at the same time being able to include in the context more information than that corresponding to the previous line. Let $\alpha_i, i = 0, \ldots, K-1$ be the *quantized* wavelet coefficients in one line of a certain subband, where K is the width of a line. Let $c_i, i = 0, \ldots, K-1$ be the context information for magnitude in this subband. We start at the top of the subband with all c_i equal to zero and update each c_i after we encode a pixel of value α_i as follows:

$$c_i = \begin{cases} |\alpha_i| & \alpha_i \neq 0 \\ c_i/2 & \text{otherwise} \end{cases} \tag{5.3}$$

The scanning within a subband is, left to right, top to bottom. If a coefficient is found to be non-zero (at the given quantization level), we keep its absolute value as the context information in the current position in a line. However if the current coefficient is zero we divide the previous context information by 2, so as to lower (but not set to zero) the neighborhood magnitude. The factor of 2 is chosen for simplicity of implementation, and there was no particular optimization involved. The above way of forming context information is equivalent to looking to several previous lines at a time and using the information from the nearest nonzero coefficient in the vertical direction. The advantage of our approach is that we are doing it in a computationally much more efficient way since we accumulate all context information in one line. Context information c_i is kept in fixed point format, so we expect to have $c_i = 0$ in areas having many zeros, i.e., a non zero coefficient does not "propagate" more than few lines or samples in the context information. Apart from the context related to the magnitude of past encoded coefficients we also keep context information

for the sign of the coefficients from the previous line, a coefficient can either be positive "+", negative "-", or zero "0". Note that the sign we store here is that of the actual coefficient in the line in that position. We need 2 bits for each coefficient in order to keep this information, the context information for the sign is:

$$s_i = \text{sign}\{\alpha_i\} = \begin{cases} 0 & \text{if } \alpha_i = 0 \\ 1 & \text{if } \alpha_i > 0 \\ -1 & \text{if } \alpha_i < 0 \end{cases} \tag{5.4}$$

The total memory needed for storing the context information for all subbands is three equivalent image lines. We need 3 lines of length $X/2$ for the first level of decomposition, 3 lines of length $X/8$ for the second level, and in general 3 lines of length 2^{-n-1} for the n^{th} level. The total buffer size is $T_C = \sum_{i=0}^{N-1} 3 \cdot 2^{-n-1} X = 3(1 - 2^{-N})X$, which tends to $3X$ as N grows.

Classification and Encoding

In the rest of the chapter we will use a function $\texttt{encode}\{v \big| \zeta\}$, to represent encoding a number v given a discrete probability model ζ. $\texttt{encode}\{\}$ will use arithmetic coding and $\texttt{encode-raw}\{v^{[\xi]}\}$ will represent the function of sending the ξ least significant bits of v, without any form of entropy coding. For the cases where we are encoding a single bit we use the function $\texttt{encode-bit}\{\mu \big| \zeta\}$, in order to emphasize that we are encoding a single bit.

Based on the context information c_i (see Figure 5.7) we classify each new coefficient α_i into a class γ_i, among 15 possible classes, as follows:

$$\beta_i = c_i + 2c_{i-1} + c_{i+1} + (c_{i-3}||c_{i-2}||c_{i+2}||c_{i+3}) \tag{5.5}$$

$$\gamma_i = \begin{cases} 14 & \text{if} \quad \beta_i > 2^{14} - 1 \\ 0 & \text{if} \quad \beta_i = 0 \\ 1 + \lfloor \log_2 \beta_i \rfloor & \text{otherwise} \end{cases} \tag{5.6}$$

				c_i	c_{i+1}	c_{i+2}	c_{i+3}
	c_{i-3}	c_{i-2}	c_{i-1}	α_i			

Figure 5.7: Context information for magnitude encoding. For each subband we keep a line of context information. In this Figure we see the relative position of the wavelet coefficient α_i to be encoded and the context information around it.

Where || stands for logical "or",

$$\alpha||\beta = \begin{cases} 0 & \text{if} \quad \alpha = 0 \text{ and } \beta = 0 \\ 1 & \text{otherwise} \end{cases}$$

The motivation behind this scheme is to keep the contribution of $c_{i-3}, c_{i-2}, c_{i+2}, c_{i+3}$ to the context formation to a minimum. The selection of the $1 + \lfloor \log_2 . \rfloor$ operator, is mostly for simplicity since it represents the number of significant bits. Moreover using a logarithmic rule for quantization into classes also accounts for the fact that in typical images neighborhoods with small context magnitude (β_i small) are more likely than those with high magnitude. Thus a logarithmic rule allows us to have more classes at low magnitude than at high magnitude.

Class $\gamma_i = 0$ corresponds to a coefficient where all its neighbors are zero, so that we expect $|\alpha_i|$ to be very close to zero, for values of γ_i further away from zero the distribution of $|\alpha_i|$ is much less peaked around zero. Up to 15 classes can occur, but in practice the number of classes that are used depends up on the bit rate. If we happen to have large enough wavelet coefficients, or high bit rate, all 15 classes might be used, but if we are encoding at low bit rates only a portion of the 15 classes might occur.

After classification we encode a bit denoting if our current coefficient α_i is significant or not, the encoding of this bit is conditioned upon the class γ_i, sending information denoting if our current coefficient α_i is significant or not corresponds to

class	sign flip	g_1	g_0
0	NO	0	0
1	NO	+	0
	YES	-	0
2	NO	0	-
	YES	0	+
3	NO	-	-
	YES	+	+
4	NO	+	-
	YES	-	+

		g_0
g_1	x	

Table 5.1: Context formation for sign encoding/decoding. We only use the sign from the two nearest neighbors for context formation. We exploit symmetry by using a sign flipping technique and we thus reduce the number of classes from nine to five. g_0, g_1 are the signs of the wavelet coefficients at the corresponding locations.

`encode-bit`$\{|\alpha_i| > 0 \big| \gamma_i\}$. If a given coefficient is found to be significant, we also encode a parameter l_i:

$$l_i = \lfloor \log_2 |\alpha_i| \rfloor \tag{5.7}$$

This parameter denotes the extra number of LSBs[8] needed to represent the magnitude of $|\alpha_i|$, given that $|\alpha_i| > 0$. We follow by a raw encoding of the l_i LSBs of $|\alpha_i|$. As an example, assume[9] $|\alpha_i| = 0000\overline{1}\underline{1101}$, $l_i = 4$. If the decoder knows the value of l_i it can then find the highest order nonzero bit of α_i, and with the transmission of l_i more bits, $|\alpha_i|$ will have been completely transmitted.

The sign of α_i is subsequently encoded using a separate context modeling, based on the sign of the two nearest neighbors. Each neighbor can be either zero, positive or negative so that we have a total of nine different combinations. By using a technique known as *sign flipping* [109] we can reduce the number of classes from nine to five. In sign flipping we take advantage of certain symmetries present in the context formation,

[8] Least significant bits

[9] The over-lined bit is the highest order nonzero bit, while the underlined bits are the additional l_i LSBs that will be sent to the decoder.

as seen in Table 5.1. Let us consider an example. Assume that both neighbors g_0, g_1 are positive and let $p, 1-p$ be the probability that a new coefficient x has the same sign or not, respectively, as its neighbors. We can assume that the probability of "same sign" will be the same if both g_0, g_1 are negative and thus we only characterize the probability of having a sign change and assume these are roughly the same regardless of the sign g_0, g_1 have, as long as their sign is the same.

State information/ Arithmetic Coder

We use an arithmetic coder that operates with different probability models where the probability model depends on the class γ_i. For each class, we keep track of symbol occurrences so as to update the probability model. The models needed for each band are:

- Five binary models for sign encoding
- 15 binary models for encoding significance information,where each model corresponds to a class γ
- One model $\mathcal{B}$ of up to 14 symbols for encoding the number l_i from (5.7). This model will be used as an M-ary model with $0 < M \leq 14$ by considering the first M symbols. As explained in section 5.3.2, l_i will always be bounded by a number M for each line.

Therefore we will need $5 + 15 + 14 = 34$ words per subband in order to store the probability model information. As can be deduced from the foregoing discussion, context modeling does not represent a significant memory overhead to our system as compared to the memory needed for filtering.

Algorithm

After wavelet transform and quantization, with the same dead-zone quantizer for all the coefficients, the algorithm for encoding a line corresponding to one subband can be described as follows:

1. For each new line $\texttt{encode-raw}\{L^{[15]}\}$, where $L = \lfloor \log_2(1 + \max_i |\alpha_i|) \rfloor$ is the number of bits needed to represent the largest coefficient in a row line. If $L > 0$ go to step 2 else return to step 1. This allows us to determine the maximum value for l_i and to skip lines that are all zero.

2. Classify each new wavelet coefficient α_i into a class γ_i according to equations (5.5) and (5.6).

3. $\texttt{encode-bit}\{|\alpha_i| > 0 \Big| \gamma_i\}$ (Encode whether α_i is zero or not, by using the corresponding model γ_i) and update the statistics for model γ_i.

4. if $|\alpha_i| > 0$ $\Big\{$

   - $\texttt{encode}\{l_i \Big| \mathcal{B}^{(M)}\}$ where $l_i = \lfloor \log_2 |\alpha_i| \rfloor$, $M = \lceil \log_2 L \rceil$, $\mathcal{B}^{(M)}$ means that we are using model $\mathcal{B}$ as an $M-$ary model, since we know that $l_i < M$.
   - form a class w_i for the encoding of the sign according to table 5.1
   - $\texttt{encode-bit}\{\alpha_i > 0 \Big| w_i\}$ and update the statistics for model w_i.
   - $\texttt{encode-raw}\{|\alpha_i|^{[l_i]}\}$, that is we encode the l_i LSBs of $|\alpha_i|$.

   $\Big\}$ else go to step 5

5. update the context information according to equation (5.3)

6. if not end of line go to the next pixel (step 2)

7. if not end of image go to the next line (step 1)

5.4 Experimental Results and Memory Analysis

In Table 5.2 we present PSNR results for five different images along with comparisons with algorithms in [8, 16, 83] and also JPEG [71] with arithmetic coding[10]. Our

[10]In order to provide a fair comparison with a DCT based technique we select the arithmetic coding based JPEG (JPEG-AR) rather than baseline JPEG. The memory requirements are very similar and the compression performance better for JPEG-AR. All the wavelet coders we consider use arithmetic coding.

results are not always the best but are competitive at a fraction of the complexity and memory utilization. It is worth repeating that our algorithm is one pass and there are no rate distortion optimization decisions made at any stage. The memory requirements depend upon the filter length in our filter-bank, the number of levels in the decomposition, the type of decomposition itself, and the width of the image. The height of the image does not affect the memory needs. The transform can be implemented in only $2LX$ pixels of memory independently of the number of levels of decomposition, the encoder and decoder are responsible for handling the synchronization buffers. Even though the synchronization buffers grow exponentially with the number of decomposition levels, we can bring their size down by orders of magnitude if we keep them in compressed form. The memory needed for context modeling is $3X$ pixels. The overall memory needs for our algorithm are the lowest reported in the literature for wavelet coders.

In Table 5.3 we present the exact memory usage for all algorithms [8, 16, 71, 83], as measured in an HP-Kayak workstation running windows NT, the memory needs of our algorithm are much closer to JPEG than any of the above mentioned algorithms. The scaling problems of wavelet coders can be seen clearly by observing the numbers in table 5.3. In many practical applications images of hundreds of millions of pixels need to be compressed, but in many cases it is impossible to buffer the whole image. Tiling the image causes blocking artifacts and degrades the performance. In table 5.2 for the column corresponding to [8] we forced the algorithm to work with tiles of size 128×128, this configuration corresponds to memory needs slightly above our current algorithm, but the performance of the wavelet coder in this case falls below that of JPEG-AR.

It is worth pointing out that we do not use a bit plane coder for the refinement bits as for example [8]. Instead we are encoding each quantized wavelet coefficient at once, without the need for multiple passes. The results, as compared to bit plane coders [8, 83, 85], are still very competitive. We compensate for not entropy coding some bits by the use of a higher order arithmetic coder to encode the number l_i of those bits.

	Rate	SPIHT [83]	C/B [16]	JPEG-AR	VM2.0 [8]	This work
Lena	0.125	31.10	31.32	28.45	30.93,(27.96)	31.05
512 × 512	0.25	34.13	34.45	31.96	34.03,(31.36)	34.20
	0.50	37.24	37.60	35.51	37.16,(34.75)	37.35
	1.00	40.45	40.86	38.78	40.36,(38.60)	40.47
Barbara	0.125	24.84	25.39	23.69	24.87,(23.27)	25.20
512 × 512	0.25	27.57	28.32	26.42	28.17,(25.38)	28.18
	0.50	31.39	32.29	30.53	31.82,(29.20)	31.87
	1.00	36.41	37.40	35.60	36.96,(33.79)	36.68
Goldhill	0.125	28.47	28.61	27.25	28.48,(26.84)	28.49
512 × 512	0.25	30.55	30.75	29.47	30.58,(29.21)	30.64
	0.50	33.12	33.45	32.12	33.27,(31.88)	33.27
	1.00	36.54	36.95	35.57	36.81,(35.47)	36.66
Bike	0.125	25.82	26.16	24.88	25.75,(21.89)	25.92
2560 × 2048	0.25	29.12	29.43	28.20	29.30,(24.83)	29.17
	0.50	33.00	33.47	32.11	33.28,(29.30)	33.04
	1.00	37.69	38.27	36.39	38.08,(34.39)	37.66
Woman	0.125	27.27	27.67	26.05	27.23,(24.09)	27.51
2560 × 2048	0.25	29.89	30.36	28.83	29.79,(26.12)	30.14
	0.50	33.54	34.12	32.47	33.54,(28.80)	33.74
	1.00	38.24	38.92	37.11	38.30,(32.96)	38.47

Table 5.2: Comparison between our method and [8, 16, 71, 83] for images: Barbara, Lena, Goldhill, Bike and Woman, the last two images are part of the test images for JPEG2000. We used five levels dyadic decomposition with 9-7 tap filters.(JPEG-AR stands for JPEG compression with the addition of arithmetic coding.) For algorithm [8] the numbers in parenthesis correspond to tiles of size 128×128.

5.5 Conclusions and Future Work

In this chapter we have developed a technique for line based wavelet transforms, analyzed the memory needs of the transform and separated the memory needed in two different categories, namely *filtering* memory and *synchronization* memory. We pointed out that the synchronization memory can be assigned to the encoder or the decoder and that it can hold compressed data. We provided an analysis for the case where both encoder and decoder are symmetric in terms of memory needs and complexity. We described a novel entropy coding algorithm that can work with

very low memory in combination with the line-based transform, and showed that its performance can be competitive with state of the art image coders, at a fraction of their memory utilization. The whole system can also be used as a tool for WT implementation in low memory, and in fact line based transforms as implemented in this work have been incorporated into the JPEG 2000 verification model. It is also worth mentioning that we can use compression to deal with the memory growth of the synchronization buffers, even in applications where compression is not the main objective, but where memory is an issue.

To the best of our knowledge, our work is the first to propose a detailed implementation of a low memory wavelet image coder. It offers a significant advantage by making a wavelet coder attractive both in terms of speed and memory needs. Further improvements of our system especially in terms of speed can be achieved by introducing a lattice factorization of the wavelet kernel or by using the lifting steps. This will reduce the computational complexity and complement the memory reductions mentioned in this work.

Image Size	2560×2048	3312×5120	6624×5120
Compressed	650K	2.1M	4.2M
SPIHT [83]	27M	81M	*
C/B [16]	21M	67M	92M
JPEG [71]	688K	720K	720K
VM2.0 [8]	51M	97M	*
This work	850K (1.5M)	1.3M (3.4M)	1.3M (5.5M)

Table 5.3: Memory usage for algorithms [8, 16, 71, 83] for tree different image sizes 5.2, 16.9 and 33.9 Mbytes. All images were compressed at 1b/p. The numbers were obtained using a personal computer, with 128M of memory. Numbers in parenthesis for the line based algorithm correspond to the memory needed for the algorithm plus memory for buffering of the complete bit stream. Memory usage was measured for the decoder but for all the above algorithms encoder and decoder are symmetric. The "*" corresponds to cases where the memory needs exceeded the machines limitations .

Chapter 6

Lifting, Memory and Complexity

Contents

All publications on the lifting scheme up to now [10, 89] consider non-causal systems, where the assumption is that the whole input signal is buffered. This is problematic if we want to use lifting in a low memory scenario. In this chapter we present an analysis for making a *lifting implementation of a filter bank causal*, while at the same time *reducing the amount of delay* (or memory) needed for the whole system. The amount of memory needed for the lifting implementation of any filter bank can be shown to be always smaller than the corresponding convolution implementation. The amount of memory savings is filter bank dependent, it ranges from 0% for the Haar transform to 40% for a $2-10$ filter bank. The amount of savings depends on the number of lifting steps as well as the length of the lifting steps used.

We will also elaborate on the use of *boundary extensions* on each lifting step instead of the whole signal. This leads to lower memory requirements as well as simpler implementations.

6.1 Lifting Structures

There have been a number of publications that propose constructing wavelet transforms as a series of *lifting* and *dual lifting* steps [10,89], or *prediction* and *update* steps.

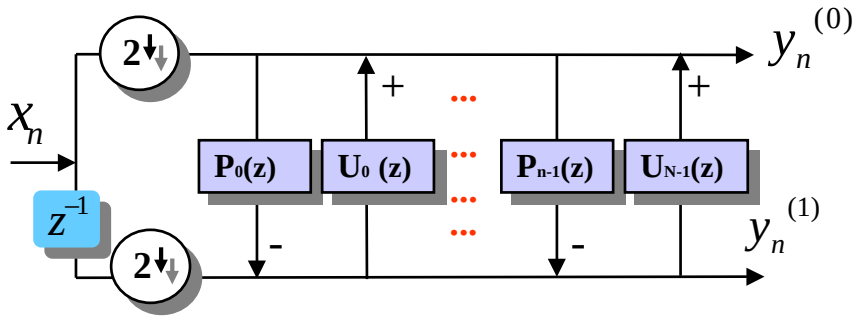


Figure 6.1: Lifting structure for a two channel analysis filter bank.

The lifting approach to WT allows for arbitrary decompositions including nonlinear filters. Complexity trade-offs for this approach have not yet been studied in depth. There has been no general theoretical formulation of the memory reduction potential of using lifting implementation of wavelets. Consider the lifting structure in Figures 6.1 and 6.2, where filters $P_k(z)$ and $U_k(z)$ may or may not be causal. The structure provides an easy way to understand the lifting implementation for one dimensional signals.

In the Figures we do not take into account scaling of the two channel outputs for normalization of the wavelet coefficients. That is, we can multiply the two outputs in Figure 6.1 by constant numbers and then divide the inputs in Figure 6.2 by the same numbers in order to have for example DC gain of one. These normalization factors are not considered since they do not have any impact on our analysis.

The assumption is that we are applying the one filter step in the whole signal and we continue with the next step. We follow the update/predict steps in the sequence they appear for the whole signal length, one step at a time. A "naive" implementation would consist of applying each filter, for example $P_0(z)$, to the whole signal, storing the result, then applying $U_0(z)$ to the output that was stored. Memory of the order of the signal length is needed in this case. In this chapter we will study how to implement such systems with minimal memory, i.e., such that outputs can be produced as soon as sufficient inputs are available.

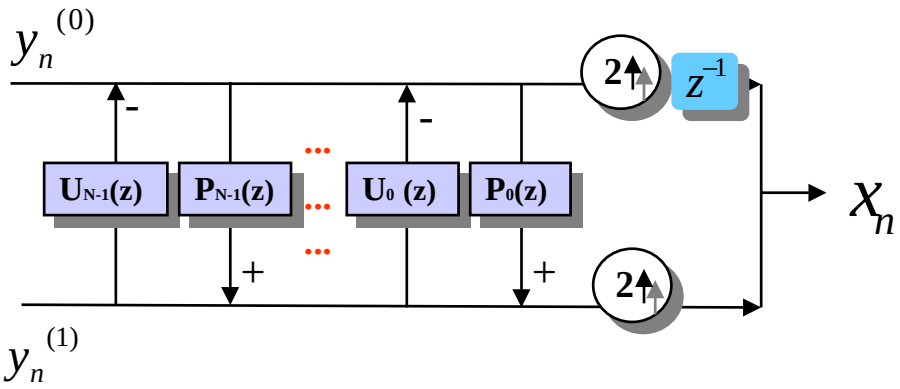


Figure 6.2: Lifting structure for a two channel synthesis filter bank.

6.1.1 Nonlinear perfect reconstruction systems

The system in Figure 6.2 is the exact inverse of the one in Figure 6.1, so that the cascade of the two systems will be the identity operator. The basic idea in this lifting structure is to partition the original signal x_n into two sets; one set with even samples and the other set with odd samples. In the analysis filter bank we try to predict the values in one set based on values from the other set. In the synthesis filter bank we perform the exact opposite operations, we remove the prediction added in the analysis.

Prediction and update operators can have any form. For example we can apply linear or non linear operator and we can even quantize the different prediction values. The filter bank implemented using a lifting structure will always be perfect reconstruction. It may not posses some energy preserving properties[1], but even after the introduction of any non-linearities perfect reconstruction will be guaranteed, because all prediction is done within closed loops. Perfect reconstruction is especially useful when we want to use a filter-bank in a lossless compression system. In this case we usually cannot afford to encode real numbers, but by using a lifting structure we can have a system that operates on integers, without the need for increased precision on the input data [10]. Moreover, with lifting structures fixed point implementations are

[1]The energy of the image is not the same as the energy of the transformed coefficients

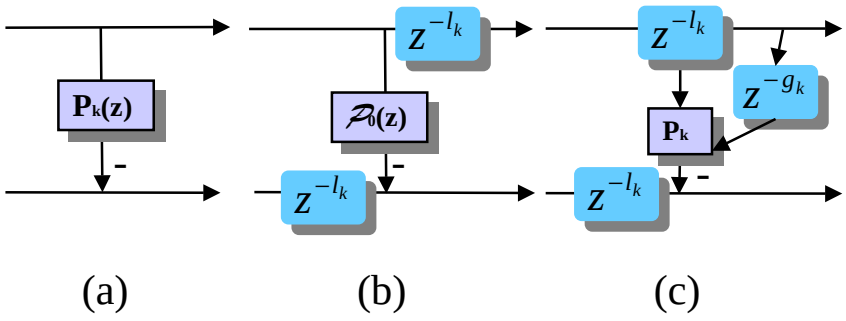


Figure 6.3: (a) Non causal (b) causal (c) modified causal lifting steps.

made possible. We do not have to keep the least significant bits in the predict and update stages. This is important because hardware fixed point implementations are substantially cheaper than floating point equivalents.

6.2 Lifting Implementations in Low Memory

The system in Figure 6.1 may have to be used as causal system, for example for audio processing applications or for a low memory wavelet transform. That is, we may not have access to the whole signal at once and, instead we may receive input samples one sample at a time. We would like to process data as we receive them and therefore we need to have a causal system. Our goal is given that samples are received one at a time to generate outputs while buffering the minimum number of data. In this case the systems in Figures 6.1 and 6.2 need to be modified and some delay lines need to be inserted into the system.

Assume that we have a series of N_p prediction steps and N_u update steps, where each prediction step is followed by an update step. If we have an odd total number of lifting steps the last one will be a prediction step and will not be followed by an update step. We assume that we start the lifting decomposition by first applying a prediction step and then an update step. If we happen to have two or more prediction steps following each other then we can combine them in a single prediction step. So, based on the above observations, without loss of generality we can assume that $N_p = N_u$

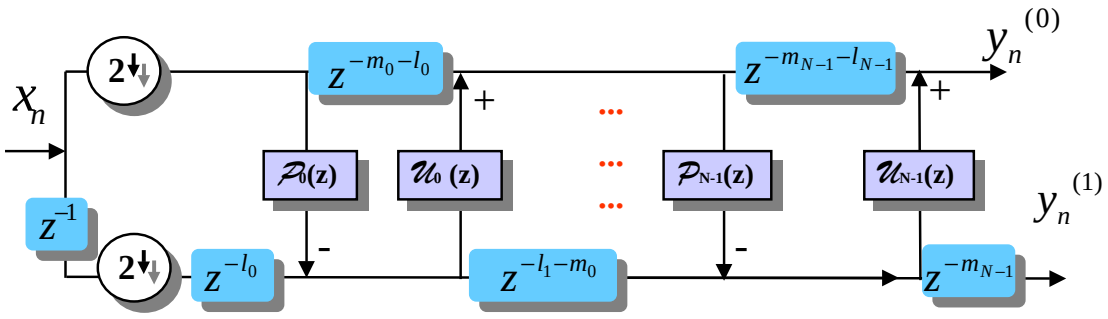


Figure 6.4: Analysis lifting structure with causal lifting steps $\mathcal{P}(z), \mathcal{U}(z)$, the synthesis structure can be derived in a similar manner.

or $N_p = N_u + 1$, which are the only two possible situations for any two channel filter bank. That is, the number of prediction steps is either equal to the number of update steps or one more than the number of update steps.

Let the k^{th} prediction $P_k(z)$ and update steps $U_k(z)$ steps be:

$$P_k(z) = p_{-l_k} z^{l_k} + p_{-l_k+1} z^{l_k-1} + \cdots + p_0 z^0 + \cdots + p_{g_k-1} z^{-g_k+1} + p_{g_k} z^{-g_k} \tag{6.1}$$

and

$$U_k(z) = u_{-m_k} z^{m_k} + u_{-m_k+1} z^{m_k-1} + \cdots + u_0 z^0 + \cdots + u_{f_k-1} z^{-f_k+1} + u_{f_k} z^{-f_k} \tag{6.2}$$

Let $\mathcal{P}(z)$ be a causal version of $P(z)$ obtained by shifting by z^{-l_k}, and $\mathcal{U}(z)$ be a causal version of $U(z)$ obtained by shifting by z^{-m_k}:

$$\mathcal{P}_k(z) = p_{-l_k} + p_{-l_k+1} z^1 \cdots + p_{g_k-1-l_k} z^{-g_k+1-l_k} + p_{g_k-l_k} z^{-g_k-l_k} \tag{6.3}$$

$$\mathcal{U}_k(z) = u_{-m_k} + u_{-m_k+1} z^1 \cdots + u_{-m_k+f_k-1} z^{-f_k-m_k+1} + u_{f_k-m_k} z^{-f_k-m_k} \tag{6.4}$$

Let us introduce the operators $\mathcal{C}$ and $\mathcal{A}$ denoting the causal and anti-causal parts of a signal, that is:

$$\mathcal{C}\{\sum_{i=-\infty}^{\infty} a_i z^{-i}\} = \sum_{i=0}^{\infty} a_i z^{-i} \tag{6.5}$$

and

$$\mathcal{A}\{\sum_{i=-\infty}^{\infty} a_i z^{-i}\} = \sum_{i=1}^{\infty} a_i z^{i} \tag{6.6}$$

The anti-causal parts of $P(z)$ and $U(z)$ are:

$$\mathcal{A}\{P_k(z)\} = p_{-l_k} z^{l_k} + p_{-l_k-1} z^{l_k-1} + \cdots + p_1 z^1, \tag{6.7}$$

and

$$\mathcal{A}\{U_k(z)\} = u_{-m_k} z^{m_k} + u_{-m_k-1} z^{m_k-1} + \cdots + u_{-1} z^1. \tag{6.8}$$

While the causal parts of $P(z)$ and $U(z)$ are:

$$\mathcal{C}\{P_k(z)\} = p_0 + p_0 z^1 + \cdots + p_{g_k-1} z^{-g_k+1} + p_{g_k} z^{-g_k}, \tag{6.9}$$

and

$$\mathcal{C}\{U_k(z)\} = u_0 + u_1 z^1 + \cdots + u_{f_k-1} z^{-f_k+1} + u_{f_k} z^{-f_k}. \tag{6.10}$$

The orders of the causal and anti-causal parts of $P_k(z)$ and $U_k(z)$ are:

$$\text{order}\{\mathcal{C}\{P_k(z)\}\} = g_0, \qquad \text{order}\{\mathcal{A}\{P_k(z)\}\} = l_0 \tag{6.11}$$

$$\text{order}\{\mathcal{C}\{U_k(z)\}\} = f_0, \qquad \text{order}\{\mathcal{A}\{U_k(z)\}\} = m_0 \tag{6.12}$$

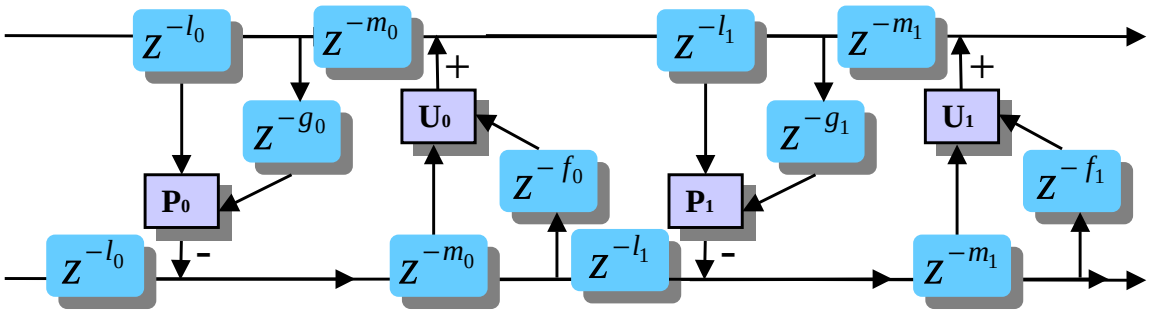


Figure 6.5: Modified lifting representation analysis filter bank, with filtering disjointed from delay units. Blocks $[P_i]$ do not include any delay, instead they reads multiple input data from the delay lines z^{-l_0} and z^{-g_0}.

The lengths of the predict and update steps $P_k(z)$ and $U_k(z)$ are:

$$\text{Length}\{P_k(z)\} = 1 + \text{order}\{\mathcal{C}\{P_k(z)\}\} + \text{order}\{\mathcal{A}\{P_k(z)\}\} = 1 + g_0 + l_0 \qquad (6.13)$$

$$\text{Length}\{U_k(z)\} = 1 + \text{order}\{\mathcal{C}\{U_k(z)\}\} + \text{order}\{\mathcal{A}\{U_k(z)\}\} = 1 + f_0 + m_0 \qquad (6.14)$$

We can force all filtering steps in Figure 6.1 to be causal by introducing appropriate delays. In Figures 6.3a,b we see a single lifting step implementation with a non causal (a) or a causal (b) structure. The complete modified analysis system is depicted in Figure 6.4.

The main problem in forcing the system to be causal is the increased amount of memory or delay needed. The system in Figure 6.4 requires more delay than the one in Figure 6.3, due to the z^{-l_k} and z^{-m_k} delays that were introduced to produce a causal system. We can combine the memory needed for filtering $P_k(z)$ and the delay line z^{-l_k} in between consecutive lifting steps by using the single modified block in Figure 6.3c. The complete analysis system with the use of the cell[2] in Figure 6.3c is seen in Figure 6.5. If we combine adjacent delays from two cells into a single delay element we get Figure 6.6. The quantities γ_k, ϕ_k are defined as:

[2]By cell we refer to either a prediction or an update step seen in Figure 6.3

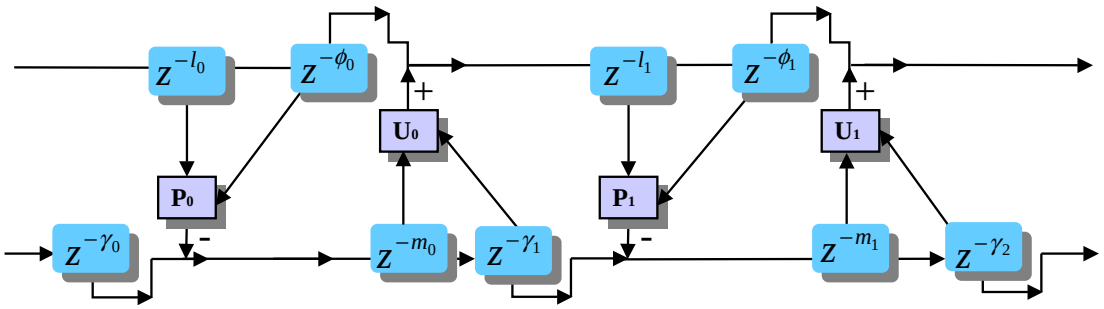


Figure 6.6: Analysis lifting structure with causal lifting steps and minimum memory requirements.

$$\gamma_k \triangleq \max\{l_k, f_{k-1}\}, k = 0, 1, 2 \ldots, N-1 \tag{6.15}$$

$$\phi_k \triangleq \max\{g_k, m_k\}, k = 0, 1, 2 \ldots, N-1 \tag{6.16}$$

In the above equations we assume that:

$$l_k = g_k = 0, \quad \text{for} \quad k \neq 0, 1, 2 \ldots, N_p - 1, \tag{6.17}$$

and

$$m_k = f_k = 0, \quad \text{for} \quad k \neq 0, 1, 2 \ldots, N_u - 1 \tag{6.18}$$

The above two conditions are used to avoid special cases in the definitions (6.15) and (6.16). The total memory T_S needed for the lifting system in figure 6.6 is:

$$T_S = \sum_{k=0}^{N_p-1} \gamma_k + \sum_{k=0}^{N_u-1} \phi_k + \sum_{k=0}^{N_p-1} l_k + \sum_{k=0}^{N_u-1} m_k \tag{6.19}$$

The above memory does not include the two samples for input and output. The lifting system described in this section is a two input two output system and as such there

is also need to buffer the two input samples. In this case the total memory needed to implement the filter bank is $T_A = T_S + 2$. For the case where $l_k = l, g_k = g, k = 0, 1, 2, \ldots, N_p - 1$, $m_k = m, f_k = f, k = 0, 1, 2, \ldots, N_u - 1$ and $N_u = N_p = N$.

$$T_S = N(l + m + f + g) \tag{6.20}$$

For example for the $9 - 7$ tap filters:

$$T_S^{(9-7)} = 1 + 1 + 1 + 1 = 4 \tag{6.21}$$

6.3 Lifting implementations of filters

In Appendix B we give several filter coefficients for convolution and lifting implementations as presented in the JPEG2000 standardization meetings [24]. In all cases the memory needed for the lifting structure implementation is significantly smaller than that needed for the straight forward convolution implementation. The total memory needed for the system is denoted by T_S. To this memory we need to add the memory from the two input samples, since the system is a two input two output operator. However, in many cases, depending on our system design, we may not have to account for this memory as part of the WT system memory. Since the input and output memory locations will be provided by another system, for example the image reader or writer. This is the reason we decided not to include the two additional samples of delay in T_S. Still we do include values of T_A.

In Table 6.1 we present data for nine different filters, the length of the low pass and the high pass filters l_{low}, l_{high}, the memory T_S and the memory T_A, along with the percentage of memory saving for each filter.

6.3.1 Lifting steps, filter symmetry and fast implementations

For a lifting structure to achieve perfect reconstruction no symmetry is needed. Because of the structure, arbitrary prediction/update filters can be used and perfect

Filter	l_{low}	l_{high}	T_S	$T_A = T_S + 2$	memory savings
$9-7$	9	7	4	6	33.33%
$13-7$	13	7	6	8	38.46%
$9-3$	9	3	4	6	33.33%
$5-3$	5	3	2	4	20.00%
$13-11$	13	11	8	10	9.09%
$5-11$	5	11	7	9	18.18%
$2-6$	2	6	2	4	33.33%
$2-10$	2	10	4	6	40.00%
$2-2$	2	2	0	2	0.00%

Table 6.1: Memory savings for different filters according to appendix B

reconstruction will still be guaranteed. However, intuitively it is reasonable to use symmetric filters, since there is no reason why a coefficient to the left of our current location should be less significant than a coefficient to the right and vice versa. Imposing symmetry helps in reducing numerical operations; as symmetric filters have approximately half the complexity of the equivalent non-symmetric filters of same length.

Consider a symmetric filter $P(z)$:

$$P(z) = \sum_{j=0}^{N-1} p_j (z^{-j} + z^j) \tag{6.22}$$

a convolution of a signal x_n with the filter will be implemented as follows:

$$y_n = \sum_{i=-\infty}^{\infty} (x_{n-i} + x_{n+i}) p_i \tag{6.23}$$

when we are performing filtering along the vertical direction in two dimensional signals we need to perform all operations in lines instead of samples. In this case we will need to add two lines and multiply the result with the corresponding filter coefficient. Symmetry is a key point to fast implementations of a wavelet transform in software.

Moreover we should scan along the image in the horizontal direction as much

as possible and try to avoid vertical scanning. The reason is that image data are stored line by line and therefore by scanning in the same order we reduce the memory accesses to a minimum. The key operation needed is the ability to add two lines together and multiply the result by the appropriate coefficient. Adding two lines and multiplying the result by a single number is a highly regular operation, it is highly predictable and does not cause and disturbances in the pipeline of a processor.

The issues of scanning the data in different directions for complexity reductions did not arise in the JPEG standard since the number of coefficients in a DCT block is very small, instead the main focus was on reducing the number of additions and multiplications. In wavelet transforms it is more beneficial to concentrate on memory patterns than on multiplication reductions. For example the 9-7 Daubechies filter bank requires more multiplications per pixel than the DCT transform. Moreover the data manipulations in both cases follow completely different patterns, and more savings are possible by memory re-arrangements than by reducing the number of additions and multiplications.

In all of the above examples the lifting steps were symmetric and were derived from symmetric filters. For symmetric filter banks it is always possible to design lifting structures with symmetric lifting steps [32]. Moreover even length filters correspond to odd length lifting steps, while odd length filters correspond to even length lifting steps.

Theorem 2 *Let* $h(z) = (-1)^{\tau_h} h(z^{-1}) z^{\phi_h}$ *and* $g(z) = (-1)^{\tau_g} g(z^{-1}) z^{\phi_g}$ *be the two low pass and high pass filters of an analysis, symmetric, bi-orthogonal filter bank. Every lifting step of the form* $p(z) = (-1)^{\tau_h + \tau_g} p(z^{-1}) z^{\frac{\phi_h - \phi_g}{2}}$ *applied to the original pair of filters will lead to a new symmetric, bi-orthogonal filter bank.*

Proof: *See Appendix C.*

Theorem 3 *Every two channel filter bank can be implemented with only symmetric and antisymmetric set of lifting steps.*

Proof: *See Appendix D.*

The above theorems are very important since symmetric systems reduce the implementation complexity of a given filter bank by a factor of two. Theorem 2 gives us a way to design a symmetric filter bank based on lifting steps, we can start with a small number of lifting steps and keep increasing the number of lifting steps while at the same time preserving the linear phase properties.

6.3.2 Boundary extensions and lifting steps

When working with lifting steps at the boundaries of a signal and wanting to take advantage of symmetric extensions, we can extend the original signal in the same way we do for the convolution implementation. Even though this will work and will give the exact same result as the convolution implementation it is preferable not to do so. The reason is that in order to perform symmetric extension along the vertical direction we need to buffer a number of lines equal to half the filter length in a buffer outside our system. This increases significantly our memory requirements and also does not allow us to take advantage of the powerful lifting structures.

A much better solution to the problem is to perform symmetric extension, as we go through each individual lifting step. In this way we can utilize the memory allocated for lifting without a need for any additional memory. For example at the end of the image when we reach a boundary we do not have any lines in the input, so we recycle the lines already stored inside the buffer. We just need to know we have reached the end of the image. The same applies to the beginning of the image, in the beginning of the image we read a line and we copy this line to a location in an internal buffer line. When filtering if we need to go beyond the beginning of the image we just reuse the lines we already have in the buffer. This offers a systematic way of dealing with the boundaries since each individual lifting step is responsible for it's own symmetric extension. No secondary system is needed to feed the whole lifting structure.

A simpler way of looking on this approach is by considering each lifting step as a stand alone system. Consequently, the amount of symmetric extension needed at the boundaries is much smaller than by considering the complete filter bank. Each

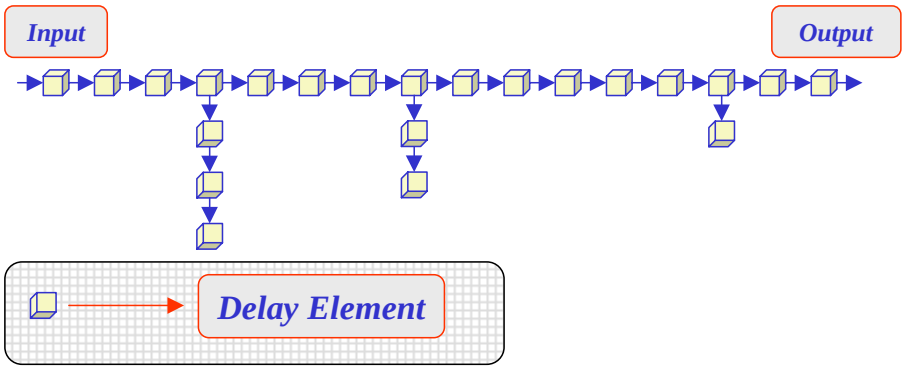


Figure 6.7: Delay line for one band of an analysis filter bank.

lifting step performs symmetric extension based on the data it receives from the previous lifting step. Related work has also been independently performed in [50, 51] for different purposes.

When working with lifting structures symmetric extension is not the only choice. In chapter 2 we described the various choices with convolution implementation of a filter bank. When working with lifting we can have almost arbitrary boundary conditions without compromising on perfect reconstruction as can be deduced from [89]. We can for example use repetition on each lifting step, so instead of using symmetric extension we can instead just repeat the last sample of our signal. Lifting implementations will always give us perfect reconstruction. The reason we will not elaborate more on this issue is that repetition does not allow us a unified approach in both convolution and lifting implementations of wavelets.

6.3.3 Software/Hardware Architectures For Lifting Implementations on Images

The direct lifting algorithm is probably the most efficient approach for filtering in the horizontal direction when working with images. The direct algorithm works in place as described in [89], i.e. we read one line of the original image and we process everything in place. Even symmetric extensions can be handled in place. In the case of symmetric extensions we do not need to extend the input signal instead we can

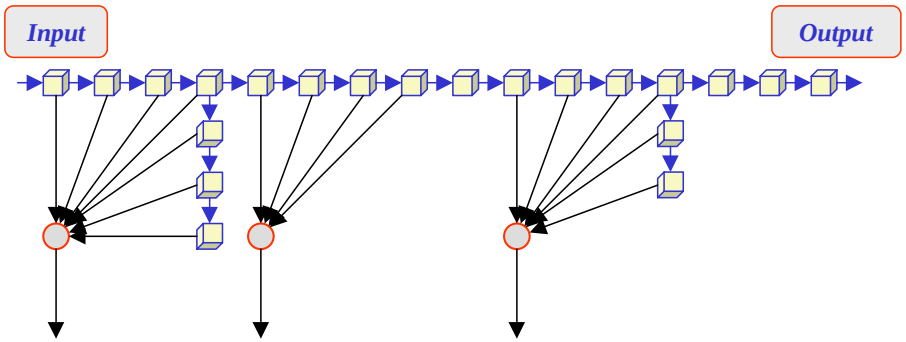


Figure 6.8: Delay line along with the filtering elements for one band of an analysis filter bank.

achieve the same effect by using boundary filters as explained in Chapter 2. The design of boundary filters is trivial in the case of lifting structures.

In the vertical direction special treatment is needed as mentioned above. The best way to implement the lifting structure is to allocate two delay lines one for the even samples and another for the odd samples. Even samples will become low pass coefficients while odd samples will become high pass coefficients. The length of each delay line can be derived from Figure 6.6. The length for the even samples will be $\sum_i \phi_i + \sum_i l_i$ elements while the delay for the odd samples will be $\sum \gamma_i + \sum_i m_i$. Those will be the total lengths of the delay lines, but the lines will not be "straight", as it can be seen from Figure 6.7. Each line might have a number of branches equal to the number of lifting steps originating from this line. An example of the whole system with the delay line and filtering can be seem in Figure 6.8. From the latest Figure 6.8 one can see that our proposed approach not only minimizes the total memory needed for filtering but also minimizes the delay between input and output. The approach is not only beneficial for software it might also be extremely useful in hardware, since by using this structure we do not need to have multiple accesses to our data, the algorithm becomes one pass unlike the direct implementation which requires a number of passes equal to the number of lifting steps.

6.3.4 Asymptotic memory savings

Asymptotically, as the filter length grows the amount of memory savings will tend to 50%, this is the exact same savings that we are getting in terms of numerical computations as explained in [32]. The proof of the amount of memory savings is essentially the same as the one presented in [32] for the number of numerical operations.

6.4 Conclusions and future work

In this chapter we described how to use a lifting structure as a causal system. We explained why lifting implementations proposed up to now cannot be causal. By forcing a lifting implementation to be causal we achieved a substantial amount of memory savings. Memory savings also translate to speed advantages on a general purpose processor. We gave a procedure to move from a lifting structure of a given length to a lifting structure of larger length, while preserving symmetry. Symmetry is of fundamental importance in reducing computational complexity. We described a process to apply symmetric extension on the bounds of a signal for the case of lifting filtering without the need for additional memory, we achieved this by extending the signal on each individual lifting step instead of the whole original signal. The effect is the same as the extension of the whole signal.

There are a lot of open problems in this area. It will be of great practical interest to investigate design algorithms to derive a symmetric lifting structure from a symmetric filter bank. It would also be of interest to design filter banks directly in the lifting form. An even more detailed description of lifting implementations in causal form, in a step-by-step approach would be of interest along with description of computational tricks. This could serve as a reference to people implementing lifting in software or hardware.

Bibliography

[1] Adobe. "TIFF, Tag based Image File Format, revision 6.0". http://www.adobe.com/Support/TechNotes.html/, June 1992.

[2] Analog Devices. *"Low Cost Multiformat Video Codec, ADC601"*, 1997. Manual.

[3] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. "Image Coding Using Wavelet Transform". *IEEE Trans. Image Processing*, 1(2):205–220, April 1992.

[4] R. H. Bamberger, S. L. Eddins, and V. Nuri. "Generalized Symmetric Extension for Size-Limited Multirate Filter Banks". *IEEE Trans. Image Processing*, 3(1):82–87, January 1994.

[5] A. Bookstein and S. T. Klein. " Is Huffman coding dead?". *Computing*, 50:279–296, 1993.

[6] C. M. Brislawn. "Preservation of Subband Symmetry in Multirate Signal Coding". *IEEE Trans. Signal Processing*, 43(12):3046–3050, December 1995.

[7] P. J. Burt and E. H. Adelson. "The Laplacian Pyramid as a Compact Image Code". *IEEE Trans. Computers*, 31(4):532–540, April 1983.

[8] C. Christopoulos (Editor). "JPEG2000 Verification model Version 2.1". ISO/IEC JTC/SC29/WG1, June 1998.

[9] C. Christopoulos (Editor). "JPEG2000 Verification model Version 5.3". ISO/IEC JTC/SC29/WG1, November 1999.

[10] R. Calderbank, I. Daubechies, W. Sweldens, and B. L. Yeo. "Losless Image Compression using Integer to Integer Wavelet Transforms". In *International Conference on Image Processing (ICIP), Vol. I*, pages 596–599. IEEE Press, 1997.

[11] C. Chakrabarti and C. Mumford. "Efficient realizations of encoders and decoders based on the 2-D Discrete Wavelet Transform". *Accepted for publication in the IEEE Transactions on VLSI Systems*, 1998.

[12] P. A. Chou, T. Lookabaugh, and R. M. Gray. "Entropy-Constrained Vector Quantization". *IEEE Trans. Acoustics, Speech and Signal Processing*, ASSP-37(1):31–42, 1989.

[13] C. Chrysafis. "Low Memory High Speed Line Based Wavelet Transforms". *Hewlett Packard, ISO/IEC JTC/SC29/WG1N978 Document, Los Angeles*, November 1998.

[14] C. Chrysafis, A. Drukarev, S. Liu, and N. Memon. "Some results on DCT-based JPEG experiments". *Hewlett Packard, ISO/IEC JTC/SC29/WG1N748 Document, Geneva*, March 1998.

[15] C. Chrysafis, L. Kondis, H. Sahinoglou, and M. G. Strintzis. "3D Medical Data Compression". In *European Symposium on Image Processing*, Boudapest,Hungary, April 1994.

[16] C. Chrysafis and A. Ortega. "Efficient Context-based Entropy Coding for Lossy Wavelet Image Compression". In *Proc. IEEE Data Compression Conference*, pages 241–250, Snowbird, Utah, 1997.

[17] C. Chrysafis and A. Ortega. "Line Based Reduced Memory Wavelet Image Compression". In *Proc. IEEE Data Compression Conference*, pages 398–407, Snowbird, Utah, 1998.

[18] C. Chrysafis and A. Ortega. "An Algorithm for Low Memory Wavelet Image Compression". In *Proc. of the Intl. Conf. on Image Proc., ICIP99*, Kobe, Japan, October 1999.

[19] C. Chrysafis and A. Ortega. "Line Based, Reduced Memory, Wavelet Image Compression". *IEEE Transactions on Image Processing*, March 2000. To appear.

[20] C. Chrysafis, H. Sahinoglou, and M. G. Strintzis. "A Nonseparable Wavelet Method for the Compression of 3D Data". In *Proc. 4th European Workshop on 3D TV*, Rome, Italy, October 1993.

[21] C. Chrysafis, A. Said, A. Drukarev, W.A Pearlman, A. Islam, and F. Wheeler. "Low complexity entropy coding with set partitioning". ISO/IEC JTC1/SC29/WG1/N1313 , June 1999.

[22] C. Chrysafis and M. G. Strintzis. "Design of Filters for Minimum Variance Multiresolution Analysis and Subband Decomposition". *" Presented at SPIE Symposium on Visual Communication and Image Processing."*, September 1994.

[23] C. Chui, J. Spring, and L. Zhong. "Complexity and memory-usage analysis of quadtree-based entropy coding (QEC)". *Teralogic Inc., ISO/IEC JTC/SC29/WG1N770 Document, Geneva*, March 1998.

[24] C. Chui, J. Spring, and L. Zhong. "Integer Wavelet Transforms". *Teralogic Inc., ISO/IEC JTC/SC29/WG1N769 Document, Geneva*, March 1998.

[25] R. Claypoole, G. Davis, W. Sweldens, and R. Baraniuk. "Nonlinear Wavelet Transforms for Image Coding". In *Proc. of 31th Asilomar Conf. on Signals, Systems and Computers*, 1997.

[26] R. L. Claypoole, R. G. Baraniuk, and R. D. Nowak. "Adaptive Wavelet Transforms via Lifting". In *Proc. IEEE Int. Conf. Acoust., Speech and Signal Processing*, Seattle, WA, 1998.

[27] A. Cohen, I. Daubechies, and J. C. Feauveau. "Biorthonormal Bases of Compactly Supported Wavelets". *Communications on Pure and Applied Mathematics*, XLV:485–560, 1992.

[28] P. C. Cosman and K. Zeger. "Memory Constrained Wavelet-Based Image Coding". In *Presented at the First Annual UCSD Conference on Wireless Communications*, March 1998.

[29] P. C. Cosman and K. Zeger. "Memory Constrained Wavelet-Based Image Coding". *Signal Processing Letters*, 5(9):221–223, September 1998.

[30] R. E. Crochiere and L.R. Rabiner. *Multirate Digital Signal Processing*. Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[31] I. Daubechies. "Orthonormal Bases of Compactly Supported Wavelets". *Comm. Pure Appl. Math.*, 41:909–996, November 1988.

[32] I. Daubechies and W. Sweldens. "Factoring Wavelet Transforms into Lifting Steps". *J. Fourier Anal. Appl.*, 4(3):245–267, 1998.

[33] T. C. Denk and K. K. Parhi. "LSI Architectures for Lattice Structure Based Orthonormal Discrete Wavelet Transforms". *IEEE Trans. Circuits and Systems*, 44(2):129–132, February 1997.

[34] N. Farvardin and J. W. Modestino. "Optimum Quantizer Performance for a class of Non-Gaussian Memoryless Sources". *IEEE Transactions on Information Theory*, IT-30(3):485–497, May 1984.

[35] P. Fernandez and A. Ortega. "An Input Dependent Algorithm for the Inverse Discrete Wavelet Transform". In *Proc. of 32th Asilomar Conf. on Signals, Systems and Computers*, November 1998.

[36] A. S. Fraenkel and S. T. Klein. "Bidirectional Huffman Coding". *The Computer journal*, 33:296–307, 1990.

[37] J. Fridman and E. S. Manolakos. "1-D Discrete Wavelet Transform: Data Dependence Analysis and Synthesis of Distributed Memory and Control Array Architectures". *IEEE Trans. Signal Processing*, 45(5):1291–1308, May 1997.

[38] G. Sharma and J. Trussull. "Digital Color Imaging". *IEEE Trans. Image Processing*, 6(7):901–932, July 1997.

[39] R. G. Gallager and D. V. Voothis. "Optimal Source Codes for Geometrically Distributed Integer Alphabets". *IEEE Trans. Information Theory*, 21:228–230, March 1975.

[40] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic publishers, Boston, MA, 1992.

[41] S. W. Golomb. "Run Length Encodings". *IEEE Trans. Information Theory*, 12:399–401, July 1966.

[42] C. R. Hauf and J. C. Houchin. "The FlashPix(TM) image file format". In *Proceedings of the Fourth Color Imaging Conference: Color Science, Systems and Applications*, pages 234–238, November 1996.

[43] C. Herley. "Boundary filters for finite length signals and time varying filter banks". *IEEE Transactions on Circuits and Systems II*, 42(2):102–114, February 1995.

[44] T. Hopper, C. M. Brislawn, and J. N. Bradley. " WSQ Gray-Scale Fingerprint Image Compression Specification". Technical Report IAFIS-IC-0110-v2, Fed. Bureau of Investig., February 1993.

[45] Y. Huang, N.P. Galatsanos, and H.M. Driezen. "Priority DCT Coding for Image Sequences". In *Proc. IEEE Int. Conf. Acoust., Speech and Signal Processing*, pages 2629–2632, Toronto, Canada, April 1991.

[46] D. A. Huffman. " A method for the construction of minimum redundancy codes". *Proc IRE 40:*, pages 1098–1101, 1952.

[47] "ISO/IEC JTC1/SC29/WG1 (ITU-T SG8)". "Coding of Still Pictures, JBIG, JPEG. Lossless, nearly lossless Compression, WD14495". Technical report, Working Draft, June 1996.

[48] ITU-T.4. *Standardization of Group 3 Fascimile Apparatus for Document Transmission, ITU-T Recommendation T.4.* ITU, 1993.

[49] ITU-T.81. *Information Technology Digital Compression and Coding of Continuous-Tone Still Images Requirements and Guidelines. Recommendation T.81.* ITU, 1992.

[50] W. Jiang and A. Ortega. "Efficient Discrete Wavelet Transform Architectures Based on Filterbank Factorizations". In *in Intl. Conf. on Image Processing*, Kobe, Japan, October 1999.

[51] W. Jiang and A. Ortega. "Parallel Architecture for the Discrete Wavelet Transform based on the Lifting Factorization". In *in Proc of SPIE in Parallel and Distributed Methods for Image Processing III*, Denver, CO, July 1999.

[52] R. L. Joshi, H. Jafarkhani, J. H. Kasner, T. R. Fischer, N. Farvardin, M. W. Marcellin, and R. H. Bamberger. "Comparison of different methods of classification in subband coding of images". *IEEE Trans. on Image Proc.*, 6:1473–1486, Nov. 1997.

[53] H. Kang. *Color Technology for Electronic Imaging Devices.* SPIE press, 1997.

[54] I. Kharitonenko and J. Liang. "Low complexity SSWT". ISO/IEC JTC1/SC29/WG1 N1160, 1999.

[55] S. T. Klein. " Space and time-efficient decoding with canonical Huffman trees". In *Proc. 8th Symp. on Combinatorial Pattern Matching*, volume 1264, pages 65–75, Aarhus, Denmark, 1997. Lecture Notes in Computer Science 1264, Springer Verlag,Berlin.

[56] R. Klette and P. Zamperoni. *Handbook of Image Processing Operators*. John Wiley & Sons, 1997.

[57] G. Lafruit, M. Peon, B. Vanhoof, and J. Bormans. "Complexity Analysis of FCD still texture coding". *IMEC, Leuven, Belgium, ISO/IEC JTC/SC29/WG11 MPEG98/M3568 Dublin*, July 1998.

[58] G. G. Langdon. "An Introduction to Arithmetic Coding". *IBM J. Res. Develop.*, 28:135–149, 1984.

[59] D. Lee. "New work item proposal: JPEG2000 image coding system". ISO/IEC JTC1/SC29/WG1 N390, 1996.

[60] J. Liang, C. Chrysafis, A. Ortega, Y. Yoo, K. Ramchandran, and X. Yang. "The Predictive Embedded Zerotree Wavelet (PEZW) Coder, a Highly Scalable Image Coder for Multimedia Applications, Proposal for JPEG 2000". *ISO/IEC JTC/SC29/WG1N680 Document, Sydney*, November 1997.

[61] T. D. Lookabaugh and R. M. Gray. "High-resolution quantization theory and vector quantizer advantage". *IEEE Trans. Information Theory*, 35:1020–1033, September 1989.

[62] S. M. LoPresto, K. Ramchandran, and M. T. Orchard. "Image Coding based on Mixture Modeling of Wavelet Coefficients and a Fast Estimation-Quantization Framework". In *Proc. IEEE Data Compression Conference*, pages 221–230, Snowbird, Utah, 1997.

[63] S. G. Mallat. "A Theory for Multiresolution Signal Decomposition. The Wavelet Representation". *IEEE Trans. Pattern Anal. and Mach. Intell.*, 1(7):674–693, October 1989.

[64] S. A. Martucci. "Symmetric Convolution and the Discrete Cosine Transforms". *IEEE Trans. Signal Processing*, 42(5):1038–1051, May 1994.

[65] S. A. Martucci and R. M. Mersereau. "The Symmetric Convolution Approach to the Nonexpansive Implementation of the FIR Filter Banks for Images". In *Proc. IEEE Int. Conf. Acoust., Speech and Signal Processing*, volume 5, pages 65–68, Atlanta, GA, April 1993.

[66] R. G. Matteson. *Introduction to document Image Processing Techniques.* Artech House, 1995.

[67] P. Moulin. "A New look at Signal Adapted QMF Bank Design". *Proceedings of ICASSP*, pages 1312–1315, April 1995.

[68] A. N. Netravali and B. G. Haskell. *Digital Pictures: Representation and Compression.* Plenum, New Jersey, 1988.

[69] E. Ordentlich, M. Weinberger, and G. Seroussi. "A Low Complexity Modeling Approach for Embedded Coding of Wavelet Coefficients". In *Proc. IEEE Data Compression Conference*, pages 408–417, Snowbird, Utah, 1998.

[70] A. Ortega and K. Ramchandran. "Rate-Distortion Techniques in Image and Video Compression". *Signal Processing*, 15(6):23–50, November 1998.

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

[72] A. S. Popat. *"Scalar Quantization with Arithmetic Coding".* Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1990.

[73] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C.* Cambridge University Press., New York., 1988.

[74] M. Rabbani and P. W. Jones. *Digital Image Conpression Techniques.* SPIE Press, Bellingham WA, 1991.

[75] M. Rabbani, J. Liang, and M. Charrier. "Coding Efficiency Sub-Group Report of Copenhagen 14th WG1 Meeting". *ISO/IEC JTC/SC29/WG1N974 Document, Copenhagen*, June 1998.

[76] L. Rade and B. Westergren. *Beta Mathematics Handbook*. CRC press, Sweden, 1992.

[77] K. Ramchandran and M. Vetterli. "Best Wavelet Packet Bases in a Rate–Distortion Sense". *IEEE Trans. Image Processing*, 2(2):160–175, April 1993.

[78] J. Rissanen. "A Universal Data Compression System". *IEEE Trans. Information Theory*, 29(5):656–664, September 1983.

[79] J. Rissanen. "Universal Coding, Information, Prediction, and Estimation". *IEEE Trans. Information Theory*, 30(4):629–636, July 1984.

[80] J. Rissanen and G. G. Langdon. "Arithmetic Coding". *IBM J. Res. Develop.*, 23:149–162, March 1979.

[81] H. Sahinoglou. *Multiresolutional Image Encoding Methods Leading to Efficient Algorithms and Implementations*. PhD thesis, Pennsylvania State University, 1992.

[82] H. Sahinoglou and S. D. Cabrera. " A High-Speed Pyramid Image Coding Algorithm for a VLSI Implementation". *IEEE Trans. Circuits and Systems for Video Technology*, 1(4):369–374, December 1991.

[83] A. Said and W. Pearlman. "A New Fast and Efficient Image Coder Based on Set Partitioning on Hierarchical Trees". *IEEE Trans. Circuits and Systems for Video Technology*, 6(3):243–250, June 1996.

[84] M. Schindler. "Practical Huffman Coding". http:// www.compressconsult.com/ huffman/, October 1998.

[85] J. M. Shapiro. "Embedded Image Coding Using Zerotrees of Wavelet Coefficients". *IEEE Trans. Signal Processing*, 41(12):3445–3462, December 1993.

[86] J. Shen and G. Strang. "The zeros of the Daubechies polynomials". In *Proc. Amer. Math. Soc.*, 1996.

[87] M. J. Smith and S. L. Eddins. "Analysis/Synthesis Techniques for Subband Image Coding". *IEEE Trans. Acoust., Speech and Signal Processing*, 38(8):1446–1456, August 1990.

[88] G. Strang and T. Nguyen. *Wavelets and Filter Banks.* Wellsley Cambridge Press, MA, 1996.

[89] W. Sweldens. "The Lifting Scheme: A custom-design construction of biorthogonal wavelets". *Appl. Comput. Harmon. Anal.*, 3(2):186–200, 1996.

[90] T. M. Cover and J. A. Thomas . *Elements of information theory.* Wiley Series in Communications, 1991.

[91] T. N. Pappas. "Model Based Halftoming of Color Images". *IEEE Trans. Image Processing*, 6(7):1014–1024, July 1997.

[92] D. Taubman. "Embedded independent block-based coding of subband data". *Hewlett Packard, ISO/IEC JTC/SC29/WG1N871 Document, Copenhagen*, June 1998.

[93] D. Taubman and A. Zakhor. "Multirate 3-D Subband Coding of Video". *IEEE Trans. Image Processing*, 3(5):572–588, Sept. 1994.

[94] R. Ulichney. *Digital Halftoning.* MIT Press, 1995.

[95] M. Unser. "On the Optimality of Ideal Filters for Pyramid and Wavelet Signal Approximation". *IEEE Trans. Signal Processing*, 41(12):3591–3596, December 1993.

[96] B. Usevitch. "Optimal Bit Allocation for Biorthogonal Wavelet Coding". In J.A. Storer and M. Cohn, editors, *Proc. IEEE Data Compression Conference*, pages 387–395, Snowbird, Utah, April 1996.

[97] P. P. Vaidyanathan. *Multirate Systems and Filter Banks.* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[98] M. Vetterli and J. Kovacevic. *Wavelets and Subband Coding*. Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[99] M.A. Viergever and P.Roos. "Hierarchical Interpolation". *IEEE Engineering in Medicine and Biology*, pages 48–55, March 1993.

[100] J. Villasenor, B. Belzer, and J. Liao. "Wavelet Filter Evaluation for Image Compression". *IEEE Trans. Image Processing*, 2:1053–1060, August 1995.

[101] E. Viscito and J. P. Allebach. "The Analysis and Design of Multidimensional FIR Perfect Reconstruction Filter Banks for Arbitrary Sampling Lattices". *IEEE Trans. Circuits and Systems*, 38(1):29–41, January 1991.

[102] M. Vishwanath. "The recursive pyramid algorithm for the discrete wavelet transform". *IEEE Trans. Signal Processing*, 42(3):673–676, March 1994.

[103] M. J. Weinberger, J. J. Rissanen, and R. B. Arps. "Applications of Universal Context Modeling to Lossless Compression of Gray-Scale Images". *IEEE Trans. Image Processing*, 5(4):575–586, Apr. 1996.

[104] M. J. Weinberger, G. Seroussi, and G. Sapiro. "Loco-I: A Low Complexity, Context-Based, Lossless Image Compression Algorithm". In *Proc. IEEE Data Compression Conference*, pages 140–149, Snowbird, Utah, 1996.

[105] R. M. Witten, I. H. Neal, and J. G. Cleary. "Arithmetic Coding for Data Compression". *Communications of the ACM*, 30(6):520–540, June 1987.

[106] J. W. Woods, editor. *Subband Image Coding*. Kluwer Academic publisher, New York, 1991.

[107] J. W. Woods and S. D. O'Neil. "Subband Coding of Images". *IEEE Trans. Acoust., Speech and Signal Processing*, 34(5):1278–1288, October 1986.

[108] X. Wu. "An Algorithmic study on lossless image compression". In *Proc. IEEE Data Compression Conference*, pages 150–159, Snowbird, Utah, March 1996.

[109] X. Wu. "Lossless Compression of Continuous-tone Images via Contex Selection, Quantization and Modeling". *IEEE Trans. Image Processing*, 6(5):656–664, May 1997.

[110] X. Wu. "Context Quantization with Fisher Discriminant for Adaptive Embedded Wavelet Image Coding". In *DCC: Data Compression Conference*. IEEE Computer Society, 1999.

[111] X. Wu. "Low Complexity High-Order Context Modeling of Embedded Wavelet Bit Streams". In *DCC: Data Compression Conference*. IEEE Computer Society, 1999.

[112] Z. Xiong, K. Ramchandran, and M. Orchard. "Space-frequency quantization for wavelet image coding". *IEEE Trans. Image Processing*, 6:677–693, May 1997.

[113] Z. Xiong, K. Ramchandran, and M. T. Orchard. "Space-frequency Quantization for Wavelet Image Coding". *IEEE Trans. Image Processing*, 6(5):677–693, May 1997.

[114] Z. Xiong, K. Ramchandran, and M. T. Orchard. "Wavelet Packet Image Coding Using Space-frequency Quantization". *IEEE Trans. Image Processing*, 7(6):892–898, June 1998.

[115] Y. Yoo, A. Ortega, and B. Yu. "Image Subband Coding Using Progressive classification and Adaptive Quantization". *IEEE Trans. Image Processing*, 8(12):1702–1715, December 1999.

[116] J. Ziv and A. Lempel. "A universal algorithm for sequential data compression". *IEEE Trans Inform Theory 23:*, pages 337–343, 1977.

Appendix A

Proof of Theorem 1

Assume that $f(x)$ is a continuous function, or has a finite number of discontinuities, using Riemann sums [76] we have.

$$H(\hat{x}) = -\sum_{i=-\infty}^{\infty} f(\hat{x}_i)\Delta \log_2(f(\hat{x}_i)\Delta) = \tag{A.1}$$

$$-\sum_{i=-\infty}^{\infty} f(\hat{x}_i)\Delta \log_2 f(\hat{x}_i) - \sum_{i=-\infty}^{\infty} f(\hat{x}_i)\Delta \log_2 \Delta = \tag{A.2}$$

$$-\Delta \sum_{i=-\infty}^{\infty} f(\hat{x}_i) \log_2 f(\hat{x}_i) - \Delta \log_2 \Delta \sum_{i=-\infty}^{\infty} f(\hat{x}_i) \underset{\Delta \to 0}{\longrightarrow} \tag{A.3}$$

$$-\int_{-\infty}^{\infty} f(x) \log_2 f(x) dx - \log_2 \Delta \int_{-\infty}^{\infty} f(x) dx = \tag{A.4}$$

$$h(x) - \log_2 \Delta \tag{A.5}$$

We thus have a relation between differential entropy $h(x)$ and discrete entropy $H(\hat{x})$ for a quantized continuous variable.

Appendix B

Example lifting structures

13-7 Filter Bank

$$L_{(13,7)} = \frac{1}{2^9}\begin{bmatrix} -1 & 0 & 18 & -16 & -63 & 144 & 348 & 144 & -63 & -16 & 18 & 0 & -1 \end{bmatrix}$$

$$H_{(13,7)} = \frac{1}{2^4}\begin{bmatrix} 1 & 0 & -9 & 16 & -9 & 0 & 1 \end{bmatrix} \tag{B.1}$$

$$P_0^{(13,7)} = \frac{1}{2^4}\begin{bmatrix} -1 & 9 & 9 & -1 \end{bmatrix}, \qquad U_0^{(13,7)} = \frac{1}{2^5}\begin{bmatrix} -1 & 9 & 9 & -1 \end{bmatrix} \tag{B.2}$$

$$N_u = N_p = 1, \quad l_0 = 2, \quad g_0 = 1, \quad m_0 = 1, \quad f_0 = 2, \quad \gamma_0 = 2, \quad \phi_0 = 1 \tag{B.3}$$

$$T_S = l_0 + m_0 + \gamma_0 + \phi_0 = 6 \tag{B.4}$$

$$T_A = T_S + 2 = 8 \tag{B.5}$$

9-7 Filter Bank

$$L_{(9,7)} = \frac{1}{2^6}\begin{bmatrix} 1 & 0 & -8 & 16 & 46 & 16 & -8 & 0 \end{bmatrix}$$
$$H_{(9,7)} = \frac{1}{2^4}\begin{bmatrix} 1 & 0 & -9 & 16 & -9 & 0 & 1 \end{bmatrix} \tag{B.6}$$

$$P_0^{(9,7)} = \frac{1}{2^2}\begin{bmatrix} -1 & 9 & 9 & -1 \end{bmatrix}, \qquad U_0^{(9,7)} = \frac{1}{2^2}\begin{bmatrix} 1 & 1 \end{bmatrix} \tag{B.7}$$

$$N_u = N_p = 1, \quad l_0 = 2, \quad g_0 = 1, \quad m_0 = 0, \quad f_0 = 1, \quad \gamma_0 = 1, \quad \phi_0 = 1 \tag{B.8}$$
$$T_S = l_0 + m_0 + \gamma_0 + \phi_0 = 4 \tag{B.9}$$
$$T_A = T_S + 2 = 6 \tag{B.10}$$

9-3 Filter Bank

$$L_{(9,3)} = \frac{1}{2^7}\begin{bmatrix} 3 & -6 & -16 & 38 & 90 & 38 & -16 & -6 & 3 \end{bmatrix}$$
$$H_{(9,3)} = \frac{1}{2}\begin{bmatrix} -1 & 2 & -1 \end{bmatrix} \tag{B.11}$$

$$P_0^{(9,3)} = \frac{1}{2}\begin{bmatrix} 1 & 1 \end{bmatrix}, \qquad U_0^{(9,3)} = \frac{1}{2^6}\begin{bmatrix} -3 & 19 & 19 & -3 \end{bmatrix} \tag{B.12}$$

$$N_u = N_p = 1, \quad l_0 = 1, \quad g_0 = 0, \quad m_0 = 1, \quad f_0 = 2, \quad \gamma_0 = 1, \quad \phi_0 = 1 \tag{B.13}$$
$$T_S = l_0 + m_0 + \gamma_0 + \phi_0 = 4 \tag{B.14}$$
$$T_A = T_S + 2 = 6 \tag{B.15}$$

5-3 Filter Bank

$$L_{(5,3)} = \frac{1}{2^3}\left[\begin{array}{ccccc} -1 & 2 & 6 & 2 & -1) \end{array}\right], \qquad H_{(5,3)} = \frac{1}{2}\left[\begin{array}{ccc} -1 & 2 & -1 \end{array}\right] \tag{B.16}$$

$$P_0^{(5,3)} = \frac{1}{2}\left[\begin{array}{cc} 1 & 1 \end{array}\right], \qquad U_0^{(5,3)} = \frac{1}{2^2}\left[\begin{array}{cc} 1 & 1 \end{array}\right] \tag{B.17}$$

$$N_u = N_p = 1, \quad l_0 = 1, \quad g_0 = 0, \quad m_0 = 0, \quad f_0 = 1, \quad \gamma_0 = 1, \quad \phi_0 = 0 \tag{B.18}$$

$$T_S = l_0 + m_0 + \gamma_0 + \phi_0 = 2 \tag{B.19}$$

$$T_A = T_S + 2 = 4 \tag{B.20}$$

13-11 Filter Bank

$$L_{(13,11)} = \frac{1}{2^{10}}\left[\begin{array}{cccccccccccc} -3 & 0 & 22 & 0 & -125 & 256 & 724 & 256 & -125 & 0 & 22 & 0 & -3 \end{array}\right]$$

$$H_{(13,11)} = \frac{1}{2^8}\left[\begin{array}{ccccccccccc} -3 & 0 & 25 & 0 & -150 & 256 & -150 & 0 & 25 & 0 & -3 \end{array}\right] \tag{B.21}$$

$$P_0^{(13,11)} = \frac{1}{2^8}\left[\begin{array}{cccccc} 3 & -25 & 150 & 150 & -25 & 3 \end{array}\right], \qquad U_0^{(13,11)} = \frac{1}{2^2}\left[\begin{array}{cc} 1 & 1 \end{array}\right] \tag{B.22}$$

$$N_u = N_p = 1, \quad l_0 = 3, \quad g_0 = 2, \quad m_0 = 0, \quad f_0 = 1, \quad \gamma_0 = 3, \quad \phi_0 = 2 \tag{B.23}$$

$$T_S = l_0 + m_0 + \gamma_0 + \phi_0 = 8 \tag{B.24}$$

$$T_A = T_S + 2 = 10 \tag{B.25}$$

5-11 Filter Bank

$$L_{(5,11)} = \frac{1}{2^3}\left[\begin{array}{ccccc} -1 & 2 & 6 & 2 & -1) \end{array}\right]$$
$$H_{(5,11)} = \frac{1}{2^7}\left[\begin{array}{ccccccccccc} -1 & 2 & 7 & 0 & -70 & 124 & -70 & 0 & 7 & 2 & -1 \end{array}\right] \tag{B.26}$$

$$P_0^{(5,11)} = \frac{1}{2}\left[\begin{array}{cc} 1 & 1 \end{array}\right], \quad U_0^{(5,11)} = \frac{1}{2}\left[\begin{array}{cc} 1 & 1 \end{array}\right], \quad P_1^{(5,11)} = \frac{1}{2^4}\left[\begin{array}{cccc} -1 & 1 & 1 & -1 \end{array}\right] \tag{B.27}$$

$$N_u = 2, \quad N_p = 1, \quad l_0 = 1, \quad g_0 = 1, \quad m_0 = 0, \quad f_0 = 0, \quad l_1 = 2, \quad g_1 = 1,$$
$$\gamma_0 = 1, \quad \phi_0 = 1, \quad \gamma_1 = 2 \tag{B.28}$$

$$T_S = l_0 + l_1 + m_0 + \gamma_0 + \phi_0 + \gamma_1 = 7 \tag{B.29}$$

$$T_A = T_S + 2 = 9 \tag{B.30}$$

2-6 Filter Bank

$$L_{(2,6)} = \frac{1}{2}\left[\begin{array}{cc} 1 & 1 \end{array}\right], \qquad H_{(2,6)} = \frac{1}{2^3}\left[\begin{array}{cccccc} 1 & 1 & -8 & 8 & -1 & -1 \end{array}\right] \tag{B.31}$$

$$P_0^{(2,6)} = \begin{array}{c} 1 \end{array}, \quad U_0^{(2,6)} = \frac{1}{2}, \quad P_1^{(2,6)} = \frac{1}{2^2}\left[\begin{array}{ccc} -1 & 0 & 1 \end{array}\right], \tag{B.32}$$

$$N_u = 2, \quad N_p = 1, \quad l_0 = 0, \quad g_0 = 0, \quad m_0 = 0, \quad f_0 = 0, \quad l_1 = 1, \quad g_1 = 1,$$
$$\gamma_0 = 0, \quad \phi_0 = 0, \quad \gamma_1 = 1 \tag{B.33}$$

$$T_S = l_0 + l_1 + m_0 + \gamma_0 + \phi_0 + \gamma_1 = 2 \tag{B.34}$$

$$T_A = T_S + 2 = 4 \tag{B.35}$$

2-10 Filter Bank

$$L_{(2,10)} = \frac{1}{2}\left[\begin{array}{cc} 1 & 1 \end{array}\right]$$
$$H_{(2,10)} = \frac{1}{2^7}\left[\begin{array}{cccccccccc} -3 & -3 & 22 & 22 & -128 & 128 & -22 & -22 & 3 & 3 \end{array}\right] \tag{B.36}$$

$$P_0^{(2,10)} = \begin{array}{c} 1 \end{array}, \quad U_0^{(2,10)} = \frac{1}{2}, \quad P_1^{(2,10)} = \frac{1}{2^6}\left[\begin{array}{ccccc} -3 & 22 & 0 & -22 & 3 \end{array}\right] \tag{B.37}$$

$$N_u = 2, \quad N_p = 1, \quad l_0 = 0, \quad g_0 = 0, \quad m_0 = 0, \quad f_0 = 0, \quad l_1 = 2, \quad g_1 = 2,$$
$$\gamma_0 = 0, \quad \phi_0 = 0, \quad \gamma_1 = 2 \tag{B.38}$$

$$T_S = l_0 + l_1 + m_0 + \gamma_0 + \phi_0 + \gamma_1 = 4 \tag{B.39}$$

$$T_A = T_S + 2 = 6 \tag{B.40}$$

2-2 Filter Bank (Haar)

$$L_{(2,2)} = \frac{1}{2}\left[\begin{array}{cc} 1 & 1 \end{array}\right], \qquad H_{(2,2)} = \left[\begin{array}{cc} -1 & 1 \end{array}\right] \tag{B.41}$$

$$P_0^{(2,2)} = \begin{array}{c} 1 \end{array}, \qquad U_0^{(2,2)} = \frac{1}{2} \tag{B.42}$$

$$N_u = N_p = 1, \quad l_0 = 0, \quad g_0 = 0, \quad m_0 = 0, \quad f_0 = 0, \quad \gamma_0 = 0, \quad \phi_0 = 0 \tag{B.43}$$

$$T_S = l_0 + l_1 + m_0 + \gamma_0 + \phi_0 = 0 \tag{B.44}$$

$$T_A = T_S + 2 = 2 \tag{B.45}$$

9-7 Filter Bank Daubechies (floating point)

This is the popular Daubechies 9-7 filter bank:

$$L_{(9,7)} = \left[\begin{array}{ccccc} 0.026749 & -0.016864 & -0.078223 & 0.266864 & 0.602949 \\ 0.266864 & -0.078223 & -0.016864 & 0.026749 & \end{array} \right]$$

$$H_{(9,7)} = \left[\begin{array}{ccccccc} -0.045636 & 0.028772 & 0.295636 & -0.557543 & 0.295636 & 0.028772 & -0.045636 \end{array} \right] \tag{B.46}$$

$$\begin{aligned} P_0^{(9,7)D} &= -1.586134342 \left[\begin{array}{cc} 1 & 1 \end{array} \right], \quad U_0^{(9,7)D} = -0.05298011854 \left[\begin{array}{cc} 1 & 1 \end{array} \right] \\ P_1^{(9,7)D} &= 0.8819110762 \left[\begin{array}{cc} 1 & 1 \end{array} \right], \quad U_1^{(9,7)D} = 0.4435068522 \left[\begin{array}{cc} 1 & 1 \end{array} \right] \end{aligned} \tag{B.47}$$

$$N_u = N_p = 1, \quad l_0 = 2, \quad g_0 = 1, \quad m_0 = 0, \quad f_0 = 1, \quad \gamma_0 = 1, \quad \phi_0 = 1 \tag{B.48}$$

$$T_S = l_0 + m_0 + \gamma_0 + \phi_0 = 4 \tag{B.49}$$

$$T_A = T_S + 2 = 6 \tag{B.50}$$

13-7 Filter Bank CRF

$$L_{(13,7)} = \frac{1}{2^8} \left[\begin{array}{ccccccccccccc} -1 & 0 & 14 & 16 & -31 & -80 & 164 & -80 & -31 & 16 & 14 & 0 & -1 \end{array} \right]$$

$$H_{(13,7)} = \frac{1}{2^4} \left[\begin{array}{ccccccc} -1 & 0 & 9 & 16 & 9 & 0 & -1 \end{array} \right] \tag{B.51}$$

$$P_0^{(13,7)CRF} = \frac{1}{2^4} \left[\begin{array}{cccc} -1 & 9 & 9 & -1 \end{array} \right], \quad U_0^{(13,7)CRF} = \frac{1}{2^4} \left[\begin{array}{cccc} -1 & 5 & 5 & -1 \end{array} \right] \tag{B.52}$$

$$N_u = N_p = 1, \quad l_0 = 2, \quad g_0 = 1, \quad m_0 = 1, \quad f_0 = 2, \quad \gamma_0 = 2, \quad \phi_0 = 1 \tag{B.53}$$

$$T_S = l_0 + m_0 + \gamma_0 + \phi_0 = 6 \tag{B.54}$$

$$T_A = T_S + 2 = 8 \tag{B.55}$$

Appendix C

Proof of Theorem 2

$$h(z) = (-1)^{\tau_h} h(z^{-1}) z^{\phi_h} \tag{C.1}$$

$$g(z) = (-1)^{\tau_g} g(z^{-1}) z^{\phi_g} \tag{C.2}$$

After application of one additional lifting step $p(z)$ the new high pass filter $h_{new}(z)$ will be:

$$h_{new}(z) = h(z) - p(z^2) g(z) \tag{C.3}$$

The new high pass filter $h_{new}(z)$ will be symmetric if and only if:

$$h_{new}(z) = (-1)^{\tau} z^{\phi} h_{new}(z^{-1}) \Leftrightarrow \tag{C.4}$$

$$h(z) - p(z^2) g(z) = (-1)^{\tau} z^{\phi} (h(z^{-1}) - p(z^{-2}) g(z^{-1})) \Leftrightarrow \tag{C.5}$$

$$h(z) \underbrace{(1 - (-1)^{\tau_h + \tau} z^{-\phi_h + \phi})}_{} - g(z) \underbrace{(p(z^2) - p(z^{-2})(-1)^{\tau_g + \tau} z^{-\phi_g + \phi})}_{} = 0 \tag{C.6}$$

if we set the two terms multiplying $h(z)$ and $g(z)$ equal to zero we will achieve the desired symmetry properties, therefore:

$$1-(-1)^{\tau_h+\tau}z^{-\phi_h+\phi}=0, \quad p(z^2)-p(z^{-2})(-1)^{\tau_g+\tau}z^{-\phi_g+\phi}=0 \Leftrightarrow \tag{C.7}$$

$$\phi=\phi_n, \quad \tau+\tau_h=even, \quad p(z)=(-1)^{\tau_g+\tau_h}z^{\frac{-\phi_g+\phi_h}{2}}p(z^{-1}) \tag{C.8}$$

so we have theorem 2.

Appendix D

Proof of Theorem 3

Every function $f(z)$ can be written in the form

$$f(z) = f_e(z) + f_o(z) \tag{D.1}$$

where

$$f_e(z) = \frac{1}{2}(f(z) + f(z^{-1})) \tag{D.2}$$

and

$$f_o(z) = \frac{1}{2}(f(z) - f(z^{-1})) \tag{D.3}$$

are the odd and even parts, and $f_e(z^{-1}) = f_e(z)$, $f_o(z^{-1}) = -f(z)$.

So after decomposition of a filter bank into lifting steps we can always split a single step into two lifting steps one symmetric and the other antisymmetric.