Entropy Coding for Image Compression Based on Generalized Lifting and SPECK

Xavier Alameda i Pineda

Technical University of Catalonia
School of Telecommunications

Entropy Coding for Image Compression Based on Generalized Lifting and SPECK

Author: Xavier Alameda i Pineda

Director: Philippe Salembier Clairon

Techincal University of Catalonia

School of Telecommunications

September 2009

En primer lloc m'agradaria agrair molt especialment al Dr. Philippe Salembier la dedicació, les crítiques constructives i les discussions dels resultats. Tot plegat ha estat la base del procés de recerca que presento en les següents planes.

En segon lloc m'agradaria donar-li les gràcies a en Julio Rolon. Sense la seva ajuda a molts nivells no hagués pogut implementar els algorismes ni hagués arribat als resultats que exposo en aquesta memòria.

També voldria agrair molt especialment a l'Ariadna la seva capacitat de treure forces d'on a mi ja no me'n quedaven. Gràcies per ser-hi en moments de desesperació i haver-me arrancat de davant de l'ordinador quan ho necessitava.

Em resta donar les gràcies als meus pares, la tenacitat dels quals m'ha ajudat a seguir constant en la meva fita: "amb temps i palla, maduren les nespres".

Finalment voldria manifertar el meu agraïment al Centre de Formació Interdisciplinària Superior de la UPC, organisme que m'ha permès compaginar les carreres de matemàtiques i telecomunicacions, un factor clau en el meu fonament teòric i en la realització d'aquest projecte.

Xavier Alameda i Pineda

Contents

List of Figures . . . vii
List of Tables . . . ix
List of Algorithms . . . xi

Abstract and Aims **xiii**

Notation **xv**

I Background **1**

1 Wavelets **3**
1.1 Introduction . . . 3
1.2 Vetterli Condition and Biorthogonal wavelets . . . 4
1.3 Lifting scheme . . . 7

2 Entropy Coding **9**
2.1 Introduction to Entropy Coding . . . 9
2.2 Arithmetic Coding . . . 10
2.2.1 Basics . . . 10
2.2.2 Improvements . . . 11
2.2.3 MQ-coder . . . 12
2.3 Wavelet based image entropy coders history . . . 12
2.4 Entropy codec evaluation . . . 13

3 Generalized Lifting **15**
3.1 History . . . 15
3.2 Definition . . . 15
3.3 Energy Minimizing Design Criteria . . . 16
3.4 Latest Results . . . 17
3.5 Our Choice . . . 18

II Image Entropy Coders **23**

4 Methodology **25**

5 SPECK **27**
5.1 Introduction . . . 27
5.2 SPECK main features . . . 27
5.3 Pseudocode . . . 28

5.4 Implementation issues . . . 30
5.5 Performance . . . 33

6 MQ Arithmetic Coder **35**
6.1 Introduction . . . 35
6.2 MQ-coder main features . . . 35
6.3 Performing adaptation . . . 35
6.4 Pseudocode . . . 36
6.5 Performance . . . 41

7 Experiments and results **47**
7.1 Introduction . . . 47
7.2 Taking benefit of zero blocks independently of the SPECK partition . . . 48
7.3 Arithmetic coding after the wavelet-based SPECK algorithm . . . 50
7.4 Splitting the significance label: the JPEG2000, the Bit Plane and the Size label sets . . . 52
7.5 Increasing the number of labels: combination of two label sets . . . 55
7.6 Taking into account the block occupancy instead of the block significance . . . 56
7.7 JPEG2000 fine conditioning . . . 58

8 Main conclusions and future research lines **63**

Bibliography **65**

List of Figures

1 Image Source Coding/Decoding procedures block diagram. . . . xiii

1.1 Two channel multirate filter bank. . . . 4
1.2 Two-dimensional wavelet transform (forward and inverse) block diagram. . . . 6
1.3 Example of a one level wavelet transform output. . . . 7
1.4 Pyramidal or hierarchical structure. . . . 7
1.5 Example of a Lifting Transform Block Diagram. . . . 8
1.6 Example of an inverse Lifting Transform Block Diagram. . . . 8

2.1 Arithmetic code example. . . . 11
2.2 Rate distortion curves example. . . . 14

3.1 Classical Lifting Scheme basic structure. . . . 15
3.2 Generalized Lifting Scheme block diagram. . . . 16
3.3 Spatial "GLazy" decomposition over the original image. . . . 19
3.4 "GL full" decomposition block diagram. . . . 20
3.5 "GLazy" decomposition block diagram. . . . 20
3.6 Context spatial distribution. . . . 21
3.7 Schematic search window. . . . 21
3.8 Transform outputs of Barbara . . . 22

5.1 SPECK I set splitting scheme. . . . 28
5.2 Wavelet image transform example. . . . 28
5.4 SPECK partition rule for eccentric sets. . . . 32
5.3 Example no how SPECK splits an odd sized set. . . . 32
5.5 SPECK rate-distortion curves for Goldhill. . . . 33

6.1 MQ coder state diagram. . . . 40
6.2 Histogram of block output byte length relative to the reference software in %. . . . 45

7.1 Spatial "GLazy" decomposition over the original image. . . . 47
7.2 Image Barbara with the three regions highlighted. . . . 49
7.3 Horizontal DWT subbands for Barbara (top-left), Bike (top-right), Cameraman (bottom-left) and Lena (bottom-right). . . . 60
7.4 Horizontal GL-co subbands for Barbara (top-left), Bike (top-right), Cameraman (bottom-left) and Lena (bottom-right). . . . 61

List of Tables

3.1 Example: mapping construction. . . . 17
3.2 CDF 9/7 tap filter coefficients 18

5.1 An example of the application of the SPECK algorithm. . . . 29
5.2 SPECK performance benchmark for Goldhill. . . . 33

6.1 Example of a Dynamic Table for 5 contexts. . . . 36
6.2 Static Table of the MQ-coder. . . . 37
6.3 MQ-coder performance example A. . . . 38
6.4 MQ-coder performance example B. . . . 39

7.1 Bit budget savings for different regions. Black region technique evaluation. . . . 49
7.2 SPECK's output bit stream length for horizontal DWT subbands of Barbara, Cameraman, Bike and Lena. . . . 51
7.3 Results for the "SPECK + Arithmetic Coder" experiment for co-located techniques. . . . 51
7.4 Results for the "SPECK + Arithmetic Coder" experiment for pyramidal techniques. . . . 52
7.5 JPEG2000 significance labels defined from significance functions σ_h, σ_v and σ_d. . . . 53
7.6 Bit stream length variation (in % with respect to the bit stream length of the original image) for the JPEG2000, the Bit Plane and the Size conditioning label sets when using a co-located technique. . . . 53
7.7 Bit stream length variation (in % with respect to the bit stream length of the original image) for the JPEG2000, the Bit Plane and the Size conditioning label sets when using a pyramidal technique. . . . 54
7.8 Bit stream length variation (in %) for JPEG200 & Bit Plane, JPEG2000 & Size and Bit Plane & Size label sets for co-located techniques. . . . 55
7.9 Bit stream length variation (in %) for JPEG200 & Bit Plane, JPEG2000 & Size and Bit Plane & Size label sets for hierarchical decompositions. . . . 55
7.10 Bit stream variation (in %) for block occupancy conditioning (BO and SBO) for co-located techniques. . . . 57
7.11 Bit stream variation (in %) for block occupancy conditioning (BO and SBO) for pyramidal techniques. . . . 58
7.12 Bit stream variation (in %) for JPEG2000 fine labels for co-located techniques. . . . 59
7.13 Bit stream variation (in %) for JPEG2000 fine labels for hierarchical techniques. . . . 59

List of Algorithms

5.1 SPECK algorithm pseudocode . . . 30
5.2 Procedure **ProcessS**. . . . 31
5.3 Procedure **CodeS**. . . . 31
5.4 Procedure **ProcessI** . . . 31
5.5 Procedure **CodeI** . . . 32
6.1 Procedure **MQ Encoder Initialization** . . . 41
6.2 Procedure **MQ Encode** . . . 42
6.3 Procedure **Transfer-Byte** (encoder) . . . 42
6.4 Procedure **Put-Byte** (encoder) . . . 43
6.5 Procedure **MQ Codeword Termination** (encoder) . . . 43
6.6 Procedure **MQ Decoder Initialization** . . . 43
6.7 Procedure **Fill-LSBs** (decoder) . . . 43
6.8 Procedure **MQ Decode** (returns x) . . . 44
6.9 Procedure **Renormalize-Once** (decoder) . . . 44

Image source coding became very important in the 80's. There were some key factors that helped to that data (image) compression revolution. First, the idea of digital image: an image composed by a set of finite-precision coefficients placed over a finite and discrete grid. To represent the image one could scan its coefficients and concatenate the coefficients binary representation. This strategy does not take into account any of the characteristics of the input signal (the image). Moreover it is not an efficient representation of the image and compression techniques are therefore necessary. Second, researchers had started to develop new techniques and softwares in the field of text compression and generic data coding some years ago (see [10, 32]). Third, the presence of digital image processing in the market increased due to its simplicity, performance and results. Those are, under my opinion, the key factors that promoted image entropy coding at the research level in the 80's.
On the other hand, this data (image) compression revolution helped to trigger another spectacular growth: the number of plain Internet users. A large amount of people started to share information: plain and formatted text, photo, music, video,... These information exchanges required better-performed techniques in terms of rate-distortion characteristics. User's needs increased and researchers, developers and producers had to upgrade the system's quality and capacity. This Internet growth, in the 1990s [1], would not have been possible without the previous image compression systems quality rise.
Image source coding/decoding systems are very diverse. However, they usually have three main blocks: the transform step, the quantization step and the entropy coder step[1]. Diagram in 1 shows the generic scheme of an image source CODEC (COding/DEConding). Each block has a different role:

Transform This step has the aim to decorrelate the signal coefficients and to concentrate the image information into a few coefficients. In short, the goal of this step is to reorganize the information in such a way that the coder step can work properly.

Quantizer Its task is to select the relevant information and to remove the remaining part. Usually, it is the only step in which the procedure loses some information. The inverse step, the Dequantizer, has the aim to minimize the effect of this information loss over the image quality.

Entropy coder This step aims to reduce the number of bits used to represent the image without any loss of information. The shortest codewords correspond to the most probable symbols and the largest codewords correspond to the least probable ones. The two previous steps (information reorganizer and information selection) prepare the signal for that step, in which the signal information is transformed into a bit stream sent to the decoder procedure through a channel, storage media, ...

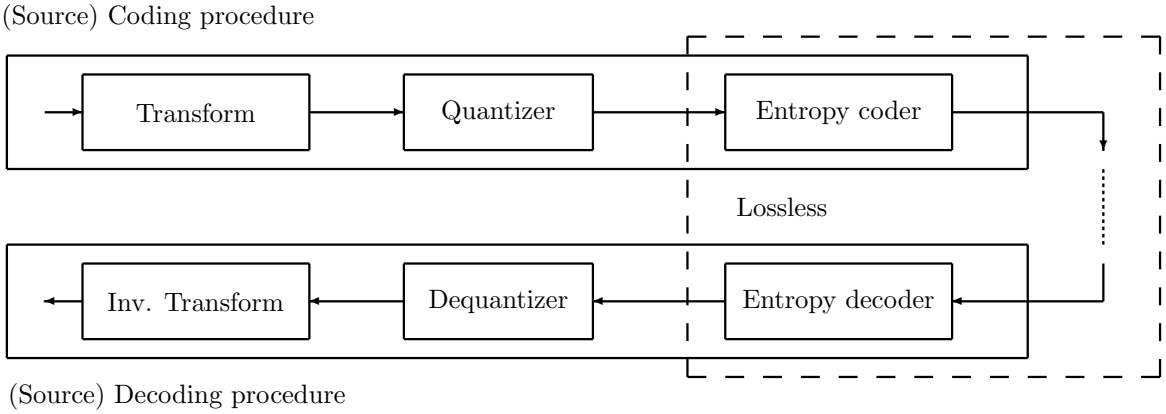

Figure 1: Image Source Coding/Decoding procedures block diagram.

[1] All three steps have the forward algorithm in the coder and the inverse algorithm in the decoder.

In order to improve the image source coding state-of-the-art, we tried to use a recent Transform named Generalized Lifting [26] for image coding. Some interesting research work have been done (see [17–20]) on this area. But, up to now, no in-depth analysis or experiment have been done on how to define an efficient entropy coder for images which have been transformed by the Generalized Lifting. Precisely, in the following report, we study several strategies based on two different entropy encoders: Arithmetic coding and the SPECK.
To end this introduction, we present the document's structure, which has two parts[2]. The first part, called **Background**, is a summary on what the reader should know to understand this research work. It has tree chapters:

Wavelets This first background chapter deals with wavelet theory. More precisely, it is an introduction to perfect reconstruction filter banks, that lead to biorthogonal wavelets. Moreover, there is an explanation about the Lifting Scheme and the relationship between wavelet decomposition and this technique.

Entropy Coding This chapter gives some notions about image entropy coding. After an introduction and some basic concepts, a description of the principles and improvements of arithmetic coding is given. Finally, some basic concepts on wavelet-based image entropy coders (like EZW, SPIHT, SPECK, ...) are given.

Generalized Lifting This one describes the Generalized Lifting Scheme technique and demonstrates its performance in an image compression scenario.

The second part, called **Image Entropy Coders: Implementation, Modification and Evaluation**, is the report of our research work. Indeed, the encoders, the tests we implemented and the results we obtained are described. It has five chapters described below.

Methodology In this chapter we describe the methodology we used for this project.

SPECK That part of the report corresponds to the description of the SPECK algorithm, our implementation and its performance.

MQ-coder This chapter has the same structure as the previous one, but in this case, the algorithm implemented and checked is the MQ-coder: the adaptive arithmetic coding used in JPEG2000.

Evaluation tasks and Results In this chapter we describe all evaluation tasks we implemented: why did we choose each task. We also discuss the difficulties and the problems we have encountered. A part from all these issues, we also show and analyze the results we obtained.

Conclusions This chapter deals with our conclusions on this research work. Moreover, we will also give some interesting lines of future research.

At the end of the report, we present the references we used.

[2]Before these two parts, there is a list on the notation used along the whole document.

h^* Complex conjugate.

H Fourier transform of a discrete signal: $H(\omega) = \sum_{n\in\mathbb{Z}} h[n]\mathrm{e}^{-in\omega}$.

$\overline{h}$ Discrete signal reversed in time: $\overline{h}[n] = h[-n]$.

$\check{h}$ Zero expanded signal (by 2): $\check{h}[n] = \begin{cases} h[n/2] & \text{if n is even} \\ 0 & \text{otherwise} \end{cases}$.

Part I

Background: Wavelets, Entropy Coding and Generalized Lifting

Chapter 1

Wavelets

1.1 Introduction

§ History Since 1807, when *Joseph Baptiste Fourier* in [5] demonstrated that every periodic function could be expressed by an infinite sum of sine and cosine functions, to the **JPEG2000** Standard[3], there is a two centuries interval of time in which science, technology and research had a key role to develop wavelet theory and applications. In 1909, in [8], the Hungarian mathematician *Alfred Haar* showed a function basis that would be the first known wavelet. 37 years later the physicist *Dennis Gabor* decomposed a signal into time-frequency packages [7]. In 1981 *Jean Morlet* founded out a way to decompose a seismic signal into a kind of "wavelets". Three years later, jointly with the quantum physicist *Alex Grossman*, they developed Morlet's model. The word wavelet appeared in 1984, for the first time. In 1986, *Stéphane Mallat* showed that those methods developed by *Haar*, *Gabor*, *Morlet...* are variations of the same algorithm (see [11]). In 1987 *Ingrid Daubechies* designed and constructed the first orthogonal compactly supported wavelet family in [3]. As a result of this research work, wavelets became a really useful tool for calculus and signal processing.

§ Motivation Some natural questions arise: why wavelets?, which is their main characteristic?, why are they so useful and widely used? As Heisenberg proposed in his famous principle, the position and momentum of a particle cannot be estimated at the same time with arbitrary precision. Translated into the signal processing field: it is not possible to have arbitrary high time and frequency resolutions at the same time. In this sense, there are two extreme (and dual) representations: the representation in time (that has the highest resolution in time and the smallest resolution in frequency) and the representation in frequency, the Fourier Transform, (that has the smallest resolution in time and the highest resolution in frequency). But, is there any representation that allow a compromise between this two ones? The well-known Short Time Fourier Transform and Wavelets are examples of transforms that represent the signal in a time-frequency or space-frequency domain. Wavelet became a really good framework for digital signal processing, because it is a tool used to represent how the energy of a signal is distributed in time and frequency (scale).

§ But, what are wavelets? From signal processing point of view, this tool (wavelet) splits each signal into two new ones: the first one is a coarser approximation of the original, and the second one, the details removed from the original to create the coarser approximation. In order to generate the coarser [detail] signal the impulse response of a low-pass [high-pass] filter is convolved with the input signal. Moreover, a subsampling by two is applied to both convolved signals. Thereby, we get an invertible non-redundant transform, that is, the transform's output is exactly what the inverse procedure needs to recover the input signal.

Usually that splitting task is iterated over the approximation coefficients: this is the idea of multiresolution. The input signal is decomposed in successive approximations. At each level, the wavelet decomposition generates an approximation signal and a detail signal. Starting from the coarsest approximation, we can recover each level representation by using the details generated from the coarsest approximation up to the recovering level. This multiresolution scheme is a key factor in the interpretation of the wavelet transform as a tool to analyze the energy distribution along time and frequency.

§ Wavelet uses Wavelets are not applicable to everything, but they are widely used. Among all wavelet applications, there is one that is particularly important: image source coding. A lot of research work has been done in order to design new wavelet-based algorithms[4] to encode and decode. Some of this algorithms have demonstrated excellent performance and very competitive rate-distortion curves.

§ Continuous vs. Discrete Wavelet theory is really large. One could find several approximations to that theory depending on the point of view. A really good reference for this subject is [12], where the reader could

[3] JPEG2000 official website

[4] Wavelet-based algorithms for image source coding are image entropy coders that expect an image wavelet decomposition as input. See section 2.3 for a detailed explanation and a few selected examples.

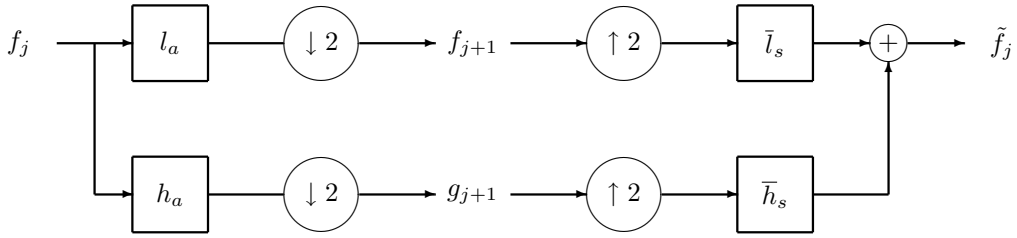

Figure 1.1: Two channel multirate filter bank.

find an exposition from several points of view between Real Analysis (Mathematics) and Filter Banks (Signal Processing). However, the following section focusses on the discrete wavelet transform from a strictly signal processing point of view, due to the purposes of this report.

1.2 Vetterli Condition and Biorthogonal wavelets

§ VETTERLI PERFECT RECONSTRUCTION CONDITION A two channel multirate filter bank is described by the block diagram in Figure 1.1. It convolves the input signal with l_a and h_a[5] and subsamples by two the output ($\downarrow 2$). The reconstruction steps are zero expansion ($\uparrow 2$), convolution with two impulse responses ($\bar{l}_s, \bar{h}_s$), and addition. Immediately after defining this structure, a question arises: is that scheme invertible? In other words: is that a perfect reconstruction transform? The aim of this section is to determine which are the restrictions on the filters to guarantee perfect reconstruction.

In order to find some conditions on the filters, we have to know which is the relationship between $\tilde{f}_j$ and f_j. It's well known that the convolution with l_a in the time domain is equivalent to multiply by L_a in the Fourier domain. But what about subsampling and zero expansion? We want an expression for the concatenation of these two systems (subsampling and zero adding):

Proposition 1.1. *Let be* $c[n] = a[\check{2}n] = \begin{cases} a[n] & \text{if } n \text{ is even} \\ 0 & \text{otherwise} \end{cases}$, *then*

$$C(\omega) = \frac{A(\omega) + A(\omega + \pi)}{2}. \tag{1.1}$$

Proof. First, note that

$$c[n] = a[\check{2}n] = \frac{a[n] + (-1)^n a[n]}{2}. \tag{1.2}$$

Let us compute the Fourier transform of $(-1)^n a[n]$[6]:

$$\begin{aligned} \sum_n (-1)^n a[n] \mathrm{e}^{-j\omega n} &= \sum_n a[n] \cos(\pi n) \mathrm{e}^{-j\omega n} \\ &= \sum_n a[n] \frac{\mathrm{e}^{j\pi n} + \mathrm{e}^{-j\pi n}}{2} \mathrm{e}^{-j\omega n} \\ &= \frac{1}{2} \sum_n a[n] \mathrm{e}^{-j(\omega-\pi)n} + \frac{1}{2} \sum_n a[n] \mathrm{e}^{-j(\omega+\pi)n} \\ &= A(\omega + \pi), \end{aligned}$$

due to the periodicity of the complex exponential.
Clearly we get the result:

$$C(\omega) = \frac{A(\omega) + A(\omega + \pi)}{2}. \tag{1.3}$$

■

[5]Notation: l (h) comes from low-pass (high-pass) filter. Subscript a (s) comes from analysis (synthesis) step.
[6]Recall that $(-1)^n = \cos(\pi n)$.

The input signal Fourier transform is modified as follows along the first channel of that filter bank:

$$\begin{aligned} F_j(\omega) \quad \rightarrow \quad & F_j(\omega)L_a(\omega) \rightarrow \frac{1}{2}\left(F_j(\omega)L_a(\omega) + F_j(\omega+\pi)L_a(\omega+\pi)\right) \rightarrow \\ & \frac{L_s^*(\omega)}{2}\left[F_j(\omega)L_a(\omega) + F_j(\omega+\pi)L_a(\omega+\pi)\right], \end{aligned} \tag{1.4}$$

due to the fact that the Fourier transform of $\bar{l}_s[n]$ (the time reversed sequence of $l_s[n]$) is $L_s^*(\omega)$ (the complex conjugate of $L_s(\omega)$). In the other channel, the calculus is analogous. After the addition, we get the expression:

$$\tilde{F}_j(\omega) = \frac{L_s^*(\omega)}{2}\left[F_j(\omega)L_a(\omega) + F_j(\omega+\pi)L_a(\omega+\pi)\right] + \frac{H_s^*(\omega)}{2}\left[F_j(\omega)H_a(\omega) + F_j(\omega+\pi)H_a(\omega+\pi)\right]. \tag{1.5}$$

Now we group the terms in two groups: the non-aliasing terms (those that depend only on ω) and the alias terms (those that depend on ω and $\omega+\pi$). We get:

$$\tilde{F}_j(\omega) = \frac{F_j(\omega)}{2}\left[L_s^*(\omega)L_a(\omega) + H_s^*(\omega)H_a(\omega)\right] + \frac{F_j(\omega+\pi)}{2}\left[L_s^*(\omega)L_a(\omega+\pi) + H_s^*(\omega)H_a(\omega+\pi)\right]. \tag{1.6}$$

The next theorem concludes our discussion on perfect reconstruction filter banks.

Theorem 1.2 (Vetterli)**.** *The filter bank l_a, h_a, l_s, h_s performs perfect reconstruction for any input signal if and only if*

$$L_a(\omega+\pi)L_s^*(\omega) + H_a(\omega+\pi)H_s^*(\omega) = 0 \tag{1.7}$$

and

$$L_a(\omega)L_s^*(\omega) + H_a(\omega)H_s^*(\omega) = 2. \tag{1.8}$$

It is possible to rewrite last two equations in a linear system:

$$\begin{pmatrix} L_a(\omega+\pi) & H_a(\omega+\pi) \\ L_a(\omega) & H_a(\omega) \end{pmatrix} \cdot \begin{pmatrix} L_s^*(\omega) \\ H_s^*(\omega) \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}. \tag{1.9}$$

Solving this linear system, synthesis and analysis filters are related thought the following expression:

$$\begin{pmatrix} L_s(\omega) \\ H_s(\omega) \end{pmatrix} = \frac{2}{\Delta^*(\omega)} \begin{pmatrix} -H_a^*(\omega+\pi) \\ L_a^*(\omega+\pi) \end{pmatrix}, \tag{1.10}$$

with $\Delta(\omega) = L_a(\omega+\pi)H_a(\omega) - L_a(\omega)H_a(\omega+\pi)$. Note that, in case $\Delta(\omega) = 0$ synthesis filters are not stable.

§ Special case: FIR filters In the special case of finite impulse response (FIR) filters, the following corollary holds.

Corollary 1.3. *For FIR filters, there exist $k \in \mathbb{R}$ and $m \in \mathbb{Z}$ such that:*

$$L_s(\omega) = -ke^{-i(2m+1)\omega}H_a^*(\omega+\pi) \quad \textit{and} \quad H_s(\omega) = k^{-1}e^{-i(2m+1)\omega}L_a^*(\omega+\pi). \tag{1.11}$$

The factor k is a gain which is inverse for the synthesis and the analysis filters and m is a reverse shift. One, generally, set $k = 1$ and $m = 0$. Under this conditions, in the time domain, Equation 1.11 could then be rewritten

$$l_s[n] = -(-1)^{1-n}h_a[1-n] \quad \textit{and} \quad h_s[n] = (-1)^{1-n}l_a[1-n]. \tag{1.12}$$

§ Wavelet transform The procedure described in Figure 1.1 is called **One Level Wavelet Transform** (the forward wavelet transform is that whose outputs are f_{j+1} and g_{j+1}, the inverse wavelet transform is that whose output is f_j). There are some interesting key points:

- An algorithm to compute the One Level Wavelet Transform is due to *Stéphane Mallat* (and it is directly extracted from Figure 1.1. At the analysis (decomposition):

$$f_{j+1}[n] = (f_j * l_a)[2n] \quad \textit{and} \quad g_{j+1}[n] = (f_j * h_a)[2n]. \tag{1.13}$$

  At the synthesis (reconstruction):

$$\tilde{f}_j[n] = (\check{f}_{j+1} * l_s)[n] + (\check{g}_{j+1} * h_s)[n]. \tag{1.14}$$

- It is possible to iterate that scheme either on the coarser approximation, either in the detail (wavelet) coefficients. Usually, the iteration is computed on the approximation coefficients, what is commonly named **Wavelet Transform**. The scheme used is an iterated filter bank. For example, let f_I the input signal to a system that iterates four times the One Level Wavelet Transform (that is a four level Wavelet Transform), the output signals are 5: $g_{I+1}, g_{I+2}, g_{I+3}, g_{I+4}$ and f_{I+4}. Then a wavelet representation f_I is composed of detail coefficients at scales $2^I < 2^j \leq 2^J$ (that is $g_{I+1}, \ldots, g_J$) plus the remaining coarser approximation f_J.
- The number of coefficients remains constant independently of the number of the transform steps and has linear complexity with the number of input signal samples.
- This transform decorrelates the signal's coefficients.
- The transform performance depends on the choice of the filters (l_a, h_a, l_s and h_s). In case synthesis and analysis filters are the same[7], the procedure is called Orthogonal Wavelet Transform. Otherwise, it is called Biorthogonal Wavelet Transform.
- Note that one could change the decomposition and reconstruction filters' role. Choose the filter with highest number of vanishing moments[8] for analysis and with highest regularity for synthesis results in a better-performance transform in most cases.
- If the reader is interested in the structure of MultiResolution Analysis, a good reference is [12].

In order to compute the wavelet transform on an image (2D finite energy signal), one computes the one-dimensional transform along rows first and along columns later. The block diagram in Figure 1.2 shows this algorithm:

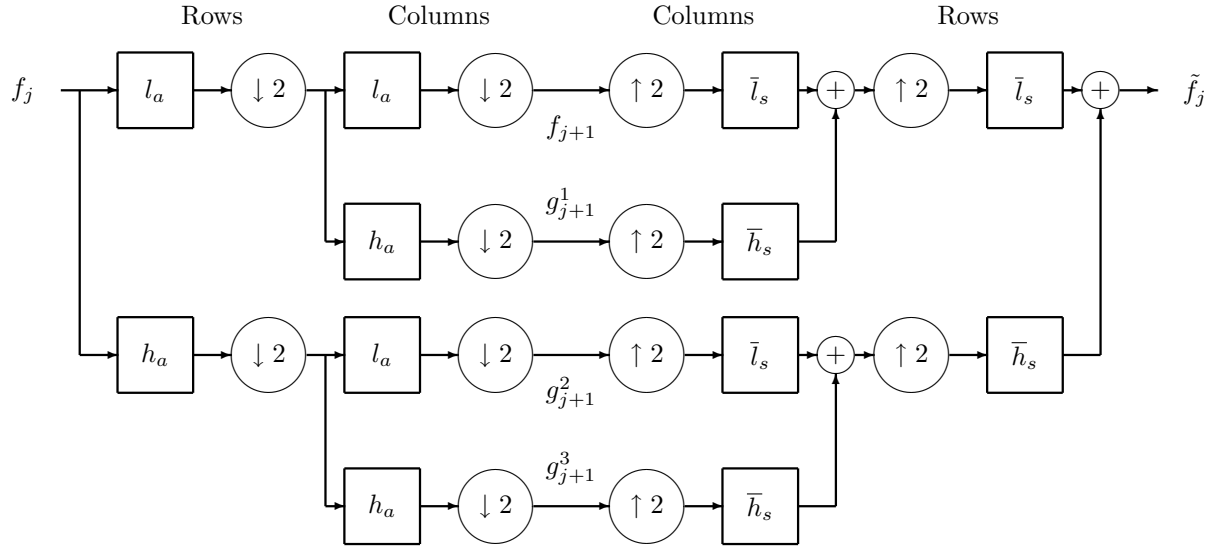

Figure 1.2: Two-dimensional wavelet transform (forward and inverse) block diagram.

In that case, there are 4 output signals: approximation (f_{j+1}), vertical details (g^1_{j+1}), horizontal details (g^2_{j+1}) and diagonal details (g^3_{j+1}). This notation is standard and comes from the kind of details we observe in the detail images.
We can represent those signals as an image either in a pyramidal structure (also called hierarchical structure) or in a co-located structure. The first case is shown in the Figure 1.4. In the second case, each coefficient stays in its original position in the image. Figure

[7] They cannot be exactly the same. At least the synthesis filter has to be a time reversed version of the analysis filter, as shown in Figure 1.2.

[8] The number of vanishing moments is the degree of the highest degree polynomial that the wavelet function associated to that filter could vanish. See [12] for more details.

Figure 1.3: Example of a one level wavelet transform output.

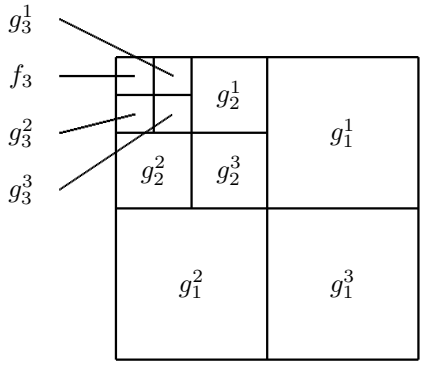

Figure 1.4: Pyramidal or hierarchical structure.

1.3 Lifting scheme

§ Second generation wavelets Lifting scheme was introduced by *Wim Sweldens* in [27, 28]. It is a way to develop "new" wavelets, to do in-place calculation and to efficiently perform biorthogonal wavelet transforms. It involves three kinds of blocks:

LWT This step is called **Lazy Wavelet Transform** and splits the input samples $f_j[n]$ into two output signals. The first one, $f_j^e[n]$ consists of the even samples of the input signal, and the second one, $f_j^o[n]$, of the odd samples.

P This step is called a "prediction" step. It performs a prediction for the odd samples, using the even ones.

U One could think about it as "preparing the signal" for the next prediction step. For example, it could be used in the implementation of a wavelet transform via lifting: the update step performs the low-pass filters. In fact, if we remove the update step, the approximation signal will be a subsampled version of the input signal.

A Lifting Transform could have different P and U steps. Usually, after the LWT is performed, alternative P and U steps are computed. Note that the P block could use all samples of $f_j^e[n]$, since they are all causal to P. However, this block must not use non-causal samples of $f_j^o[n]$. An analogous rule holds for the U block.

§ Invertibility Guaranteed The next figure shows the block diagram for a Lifting Transform with two steps: a P and a U. Note that the invertibility is guaranteed because of the transform's structure. In fact, it is possible to reconstruct the signal because it is possible to compute exactly the same update step and therefore, it is possible to compute exactly the same prediction step. Finally, an Inverse Lazy Wavelet Transform has to be done. Notice that there are two little obvious changes in the structure: the output of the U block is

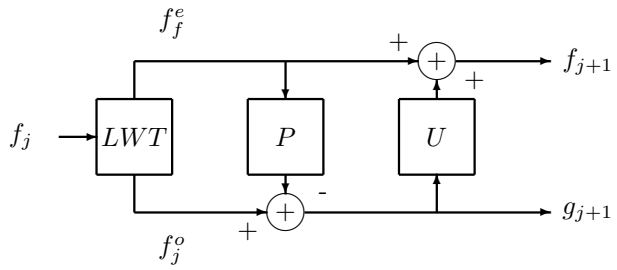

Figure 1.5: Example of a Lifting Transform Block Diagram.

subtracted (to f_{j+1}) instead of added (to f_j^e) and similarly for the output of the P block (is added to g_{j+1} instead of subtracted to f_j^o).

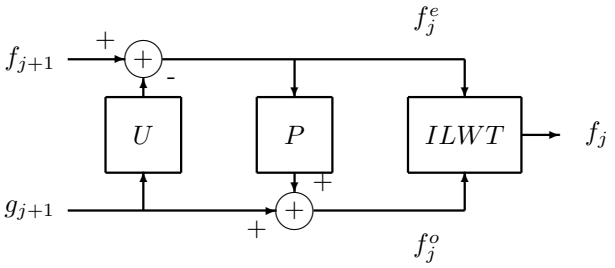

Figure 1.6: Example of an inverse Lifting Transform Block Diagram.

§ WHY LIFTING SCHEME? Lifting scheme became a revolution in signal processing because of the following properties:

- One could implement in-place transforms. These transforms allow reduced memory implementations, which are very useful limited-resources scenarios.
- One could improve and modify existing transforms at a very cheap cost (in terms of software/hardware modifications) due to proposition 7.5 of [12]. Given a finite sequence $r[n]$, filters could be modified as follows:

$$L_a^l(\omega) = L_a(\omega) + H_a(\omega)R^*(2\omega) \qquad H_s^l(\omega) = H_s(\omega) - L_s(\omega)R(2\omega) \tag{1.15}$$

  In that case, those filters are said to be lifted, because we could improve their performance or their characteristics by using this technique.
- One could implement all discrete wavelet transform via Lifting because of theorem 7.14 of [12]. This theorem proves that any wavelet transform with finite support kernels could be implemented by a lazy wavelet transform and a finite succession of prediction and update steps. On the other hand, theorem 7 of [4] provides an algorithm based on Euclidean division of Laurent polynomials to compute how lifting steps must be to implement a perfect reconstruction linear filter bank.
- It is possible to construct wavelets in arbitrary domains. See, for example, [23] for a paper on constructing wavelet on the sphere.

In the next chapter we explain how wavelet transform helped to image source coding. We present some wavelet-based entropy coders, that is, algorithms that take profit of the characteristics of the output signal of a wavelet transform to efficiently code the input image.

In the third chapter we present a recent technique (GL) that should be considered as a generalization of the Lifting Scheme. As we said, in the second part of that report we evaluate GL's performance when it is used as the transform of an image source coding scheme.

Chapter 2

Entropy Coding

2.1 Introduction to Entropy Coding

§ What's entropy coding? In 1948 *Claude Shannon* wrote [24]:

> *The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.*
>
> [[**Claude Shannon**]]

Source coding and channel coding solved, jointly, that challenge in fixed and mobile communications over really hard-behaviored and noisy channels. But, which part of this procedure owns to what we call Source Coding? Given a message (source), one would like to transmit it as faithful as possible and as cheap as possible (in terms of transmitted message length). There are two big steps: remove the message's redundancy and add the precise quantity of redundancy to overcome channel's errors. **Source Coding** techniques try to remove as much redundancy as possible from the message and to represent it with the shortest possible bitstream. The entropy coding block of a source coding procedure has the aim to assign the smallest codeword to the most probable symbol and the largest codeword to the least probable symbol in such a way that the source coding procedure gets its second aim.

§ Why source coding? Source coding has the aim to reduce the amount of information needed to represent a message. Its aim is independent of the channel quality. Even in a perfect channel, we use source coding to remove all the redundancy of a message: we want to represent this message with the minimum possible number of bits. On the other hand, in case the channel is not perfect, we should add some redundancy to overcome channel problems: noise, distortion, ... This is the aim of channel coding.

§ Shannon's (source code) theorem But how much information? How one could precise that idea?

Definition 2.1 (Entropy)**.** *Let be x a discrete source. Let be $X = \{x_1, \ldots, x_n\}$ its alphabet and $\{p_1, \ldots, p_n\}$ the symbol probabilities. The entropy of that source is defined as:*

$$H(x) = \sum_{i=1}^{n} -p_i \log(p_i). \tag{2.1}$$

The entropy provides us with the amount of information generated by that source. In addition, this value determines the maximum throughput for a channel:

Theorem 2.2 (Shannon)**.** *Let be x a discrete source with entropy $H = H(x)$. Let us transmit some symbols of that source through a channel with capacity C. Then it is possible to encode the symbols of the source at the average rate $\frac{C}{H} - \varepsilon$ symbols per second over the channel where ε is arbitrarily small. It is not possible to transmit at an average rate greater than $\frac{C}{H}$.*

The reader can find a proof of this theorem in the original article [24].

This theorem leads to a simple conclusion. The greater is the entropy, the greater should be the channel capacity or the encoder ability. Entropy is a theoretical minimum that is not commonly reached. In order to achieve this minimum, we have to design source coding strategies that allow to efficiently compress the signal.

§ Image entropy coders The next two sections deal with the arithmetic coder and with the wavelet-based entropy coder family respectively. We first present arithmetic coding whose it was not initially designed for

image compression. Despite the good results obtained with arithmetic coding, some wavelet-based image entropy coders appeared. These coders aim to have two desirable features: take profit of the intersubband/intrasubband correlation and provide an embedded bit stream. The second feature is very interesting because the bit stream output by the encoder is sorted by importance. Algorithms get this feature by bit plane quantization. They apply the same procedure (we will see it in detail) from the most significant bit plane to the least significant bit plane, sorting the information by its significance. This feature is really desirable and allows us to change the order of the quantization step and the entropy coding step in the initial block diagram 1: the encoder does not worry about how much information the decoder wants, because the bit stream is sorted and the decoder can stop the decoding procedure when it has enough information.
This entropy coders are not independent. Commonly in the literature, researchers use an arithmetic coder after a wavelet-based image coder (see [9], for example). However this is not the scheme of the most competitive image compression Standard. Nowadays the reference Standard (the JPEG2000) uses a mixture of the two types of codes, but it does not use a concatenation. Although the entropy coder is a high competitive arithmetic coder (MQ-coder), it uses features from wavelet-based algorithms such as the bit plane quantization or intrasubband correlation.

2.2 Arithmetic Coding

2.2.1 Basics

§ HISTORY Arithmetic coding was developed in the 1970's and 1980's. Several authors published results about this technique. Arithmetic coding was not treated seriously until the publication of source code for a multi symbol arithmetic coder by Witten et al. in [31]. From then on, there was a data compression systems revolution due to the new coder. Arithmetic coding is used, nowadays, in text compression [13], image compression [16], ...

§ BLOCKS Arithmetic coding is a very complete concept that takes into account a Model, a Statistics update strategy and the Coder (the Core). Those blocks are described from the high level up to low level intelligence.

Model The model is the intelligence block. Its aim is to understand the input signal and, therefore, gather the signal's characteristics (geometry, statistics, stationarity, ...) It is the high-level part of the arithmetic coder and the procedure whose design is high application dependent. As Alistair Moffat wrote in [13]:

> *One can think of the model as the "intelligence" of a compression scheme, which is responsible for deducing or interpolating the structure of the input.*
>
> [[**Alistair Moffat**]]

Statistics This block has to manage the input statistics. Its main objective is to accurately update the statistics of input symbols possibly conditioned by a context given by the model block. Obviously, that update has to be done after the symbol is coded. Otherwise the scheme would not be invertible.

Core The core is responsible for actually encoding the input symbols using the symbol statistics. That block is the easiest to describe and it is also the less-application dependant.

In the next, the Core block is defined and explained. The Model and the Statistics bloc are not developed in that report due to its high-application dependency.

§ DEFINITION: THE CORE The main idea of the arithmetic code is to represent a message by a point in the interval of real numbers between 0 and 1. In the decoding procedure, one should know how many symbols have to be decoded[9]. Note that the longer the message is, the smaller becomes the interval and, consequently, the longer have to be the output message length.
Let be x a source with alphabet $X = \{x_1, \ldots, x_n\}$ and with probabilities $p_1, \ldots, p_n$ [10]. Before coding any symbol, the interval is initialized to $I^0 = [0, 1]$ (super index j denotes the state after coding j symbols). Let us

[9] Alternatively, one could include in the alphabet an end-of-message symbol to stop the decoder.
[10] In this section, this probabilities are fixed values.

define C^j and A^j as $[C^j, A^j] = I^j$: the lower and upper bound of the interval respectively. When the symbol x_j is received, the encoder reduces the interval size by p_j and sets its bounds as follows:

$$C^j = C^{j-1} + \left(A^{j-1} - C^{j-1}\right) \sum_{i=0}^{j-1} p_i \quad \textit{and} \quad A^j = C^j + p_j \left(A^{j-1} - C^{j-1}\right). \tag{2.2}$$

Figure 2.1 shows an example on how the arithmetic code works. In the example, the source has an alphabet of four symbols (x_i). These symbols have probabilities $\{0.4, 0.1, 0.2, 0.3\}$ as shown in the figure. The message to code is: x_4, x_2, x_3. The vertical lines are the intervals we denoted by I^k, their lower and upper bounds are printed in the figure. The arrow between each pair of intervals denotes the symbol to code, and the numbers beside the vertical lines are the probability to code x_k conditioned to the symbols yet coded. For example, the value 0.0012 in the third interval is the probability that the third symbol to code is x_1 conditioned to the fact that the first symbol coded is x_4 and the second symbol coded is x_2. Notice that the symbols are sorted in the interval as natural order: x_1, x_2, x_3 and x_4. Before coding any symbol, the interval is $I^0 = [0, 1]$. After coding the first symbol, x_4, the interval is $I^1 = [C^0 + (A^0 - C^0) \sum_{i=0}^{3} p_i, C^1 + p_4(A^0 - C^0)] = [0.7, 1]$.

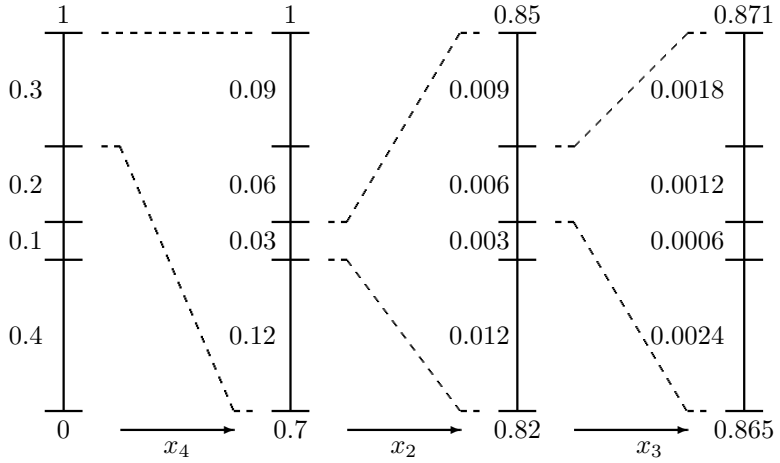

Figure 2.1: Arithmetic code example. Each vertical line represents the interval after a coded symbol ($\emptyset, \{x_4\}, \{x_4, x_2\}, \{x_4, x_2, x_3\}$). Horizontal lines symbolize the accumulate probability for the symbols.

Note that, in case the message contains very probable symbols, the final interval will become very large, and a short representation will be possible. On the other hand, if the message contains symbols with very low probability, the final interval will become shorter and shorter, and we will need a large amount of bits to represent it. This fact corresponds to the idea of "the higher the symbol probability is, the shorter should be the code".

2.2.2 Improvements

§ Low precision arithmetic coding The previous procedure has many practical problems. High precision arithmetics is one of the most serious issue of arithmetic coding. Obviously, no machine has arbitrary high precision arithmetic. So it is mandatory to develop an algorithm that performs arithmetic coding with low precision arithmetic. The root of the problem is the interval I. That interval becomes smaller each time the arithmetic encoding procedure is executed. Suppose we have a register of length b bits: how could we manage that? *Alistair Moffat* in [13] exposed an algorithm whose basic idea is to take the initial interval as $I^0 = [0, 2^b - 1]$. Each time the interval becomes too small, the coder (and the decoder) should renormalize it in order to guarantee that the next symbol could be represented with this low precision arithmetic.

§ Multiplication-free arithmetic coders What about products and divisions? Are they computational comparable to additions or subtracts? The answer is no. In hardware implementations, products and divisions are really expensive if we compare them against additions, subtractions, shifts, and other logical operations. In [13], there is an implementation of the encoding algorithm avoiding the use of any multiplication or division.

This change does not affect to the effectiveness of the coder (the compression rate remains the same), but increases by a factor close to 2 the efficiency (the time consumption).

§ ADAPTIVE ARITHMETIC CODERS In fact, the core only needs a probability distribution for each symbol to be coded. The model block, through the statistics update block, is the responsible for providing a probability distribution to the core procedure. In this way it is possible to think about an adaptive arithmetic coder.
However there is one condition that must be satisfied: the invertibility. The information that has not been coded must not be used to decide the model nor the probability distribution to use.

2.2.3 MQ-coder

§ HISTORY MQ-coder is the **binary alphabet** adaptive low-precision multiplication-free arithmetic coder used in JPEG2000. It comes from the QM-coder used in JPEG and JBIG. In [15] there is an explanation on how the Q-coder (published in [16]) evolved to the QM-coder and which differences exist between these two implementations of an adaptive arithmetic coding.

§ STATE DIAGRAM: THE ADAPTIVE STATISTICS BLOCK In the case of the JPEG2000 & MQ-coder, all three blocks of the arithmetic coder structure are precisely defined and differentiated. There is one block dedicated to extract a "context" from the information yet coded. This procedure should be considered part of the model block, because it tries to understand the input characteristics and, therefore, performs the highest-level action in the arithmetic entropy coder. There is another block that updates the statistic following some simple rules (the statistics block). And finally there is a block that codes the symbols (the core block). The information of the MQ-coder used in JPEG2000 can be found with many detail in [30].
The context is some kind of information helps us to know what we are coding. For example, in a binary sequence, we could condition the probability of the bit to encode to a context in order to decrease the entropy. Actually this is the main reason for using contexts. That context should help us to determine the shape and the parameters of the symbol's distribution. That probability distribution choice is used by the statistics and core blocks.

2.3 Wavelet based image entropy coders history

These algorithms aim to take profit of the wavelet pyramidal structure in order to efficiently encode the image. In fact, all algorithms described here suppose that the transform used in the source coding scheme presented in Figure 1 is a hierarchical decomposition (e.g.: a wavelet transform). All the algorithms described here take the coefficient energy as a "measure of importance", usually named significance. These coders treat the most significant coefficients before the least significance ones. We will see precisely, what does "before" mean.
Wavelet-based image entropy coders started with the Embedded Zerotree Wavelet (EZW) algorithm by Jerome M. Shapiro (see [25]). It has four key concepts: a discrete wavelet transform or hierarchical subband decomposition[11], prediction of the absence of significant information across scales exploiting the self-similarity inherent in images, quantization based on successive approximation and universal lossless data compression which is achieved via adaptive arithmetic coding.
Published three years later, the SPIHT algorithm by W. A. Pearlman and A. Said (see [22]) was an improvement over the Shapiro's EZW. The key point of this improvement is what they called set partitioning in hierarchical trees, a rule to split sets (that are represented as nodes in a tree) into subsets (the descendant nodes of the split set) depending on their significance. To check the significance, the algorithm compares the coefficients in a set with respect to a sequence of decreasing thresholds: in case that no coefficient has its magnitude above the threshold, the set is not split, otherwise, the set is split into four subsets whose significance with respect to the same threshold is tested. When the algorithm tests a pixel and finds that it is significant, it sends the coefficient's sign to the decoder. At the end, the algorithm rescans those pixels found significant with respect to previous thresholds, and refines their magnitude.
The disparity between EZW and SPIHT is in the quantizer; precisely in the symbol stream sent to the encode step. In case of EZW, the quantizer chooses the symbol sent among an alphabet (zerotree-root, isolated zero, positive significant, negative significant). In case of SPIHT, symbols are bits (0 or 1) representing either if the set must be split (if that node has descendents in the hierarchical tree), or the sign of the significant coefficient either the magnitude refinement bits.

[11] This is not a part of the EZW, but this algorithm expect that the input signal is a wavelet decomposition (or another hierarchical decomposition).

In the year 2000 a new algorithm called EBCOT was presented in [29] by *David Taubman*. This algorithm was not much better than those we presented above in terms of the rate-distortion curve. However, it had (has) several new desirable features:

Resolution Scalability The whole image is split into several blocks $\mathcal{B}_i$. Each block contributes to a subband and a resolution level. Then, it is easy to think about a resolution scalable bitstream.

SNR Scalability All bit streams of all block are split in quality layers $\mathcal{Q}_q$ following the PCRD[12] optimization algorithm. Each quality layer represents a reached SNR value. The PCRD algorithm determines which bits of each block bit streams form each quality layer.

Random access The algorithm structures the bitstream in such a way that allows us to select those code-blocks (i.e., that part of the bitstream) that should be used to reconstruct a region in which we have a particular interest.

EBCOT corresponds to the Model and Statistics blocks and the arithmetic core block is performed by an arithmetic adaptive finite state machine coder called MQ-coder. Although MQ-coder is not used in the EBCOT's original paper [29], it is used in the JPEG2000 standard. The use of an arithmetic coding after a bit plane coding motivated a research line we followed in order to try to improve state-of-the-art coding schemes. Although the EBCOT algorithm will not be described deeply, we are going to analyze the MQ-codec in chapter 6.
Four years later, in 2004, the SPECK algorithm was published in [14]. This algorithm is a wavelet-based zero-block image coding. It assumes a hierarchical wavelet transform and the energy clustering in the space-time domain. Its recursive set partitioning procedure efficiently and quickly localizes energy clusters in the well defined hierarchical structure. The whole algorithm is considered a low complexity algorithm due to its simplicity.

> *The key of the algorithm is to properly and efficiently sort sets of transform coefficients according to their maximum magnitude with respect to a sequence of declining thresholds.*
>
> [[**Pearlman et al.**]]

We used this algorithm as a state-of-the-art reference which we want to improve. We tried to modify this algorithm in order to improve its performance when the input image is transformed with a DWT followed by a Generalized Lifting transform[13]. The SPECK algorithm will be deeply described as well as the modifications we tried.

2.4 Entropy codec evaluation

How should we choose among the set of possible entropy encoders, the suitable one for our application? Is there any criteria, from the signal processing perspective, that points us in one direction? As we described, the main objective of an entropy coder is to transmit the largest amount of information in the lowest number of bits. We need a criterion that evaluates how much information has been transmitted depending on the number of bits. This tool is called the rate-distortion curve. This is a plane curve whose points (s,t) represent a distance between the original image and the reconstructed one (t) depending on the compression ratio: that is how many bits per pixel we used in the representation evaluated (s).
The metric will we use is the distance of square sumable sequences,i.e.:

$$\mathcal{D}(x,y) = \sum_{m,n=0}^{M,N} (x[m,n] - y[m,n])^2 \qquad x, y \in l^2\left(\mathbb{Z}^2\right), \tag{2.3}$$

where

$$l^2\left(\mathbb{Z}^2\right) = \left\{ x[m,n] | \sum_{m,n=0}^{M,N} |x[m,n]|^2 < \infty \right\}.$$

[12] Post Compression Rate Distortion, see the paper.
[13] See next chapter.

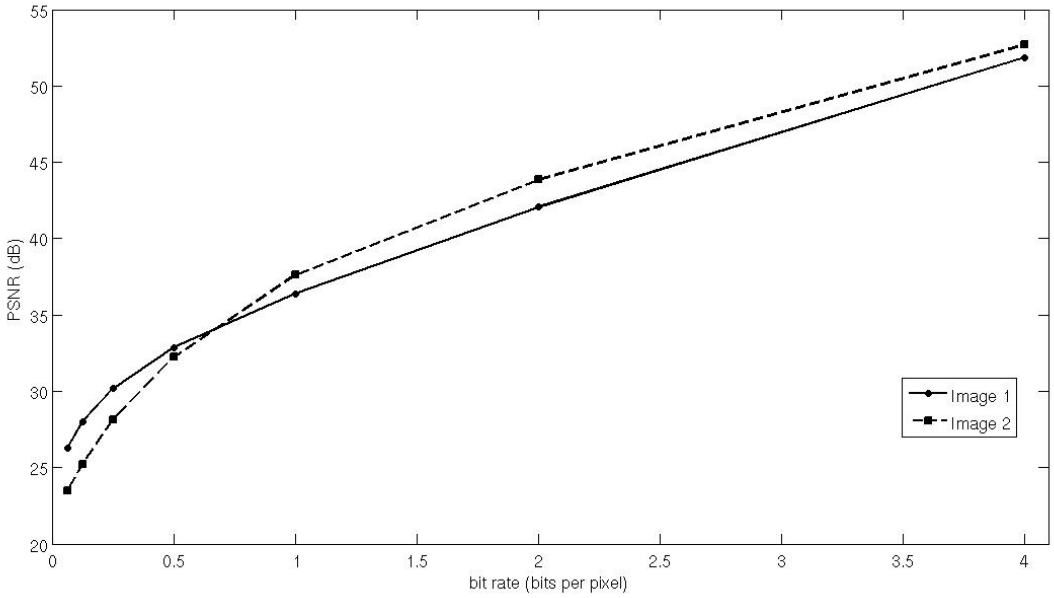

Figure 2.2: Rate distortion curves example.

Precisely, we are going to talk about Peak Signal to Nose Ratio (PSNR):

$$PSNR(x,y) = 10\log\left(\frac{M\,N\,255^2}{\mathcal{D}(x,y)}.\right) \tag{2.4}$$

But, which curve represents the best codec system? Figure 2.2 shows two examples of rate-distortion curves. Notice that between two points at the same height (the same PSNR, information transmitted) but at different bit-rates: we choose the coder represented in the left point because it is able to use least bits for the same amount of transmitted information. Alternatively, between two points at the same bit-rate, but at different PSNR values we choose the coder represented in the top point, because it is able to better select the information to transmit, fixed the number of bits to use. So, in the region from 0 to 0.75 bpp the best curve is the one of the Image 1. However in the region from 0.75 to 4 bpp, the best curve is the other one.

Chapter 3

Generalized Lifting

3.1 History

The Generalized Lifting Scheme, from now on GL, is a recent technique published in the PhD Thesis of *Joel Solé* [26]. The main idea is to generalize the classical Lifting Scheme presented in section 1.3 embedding the addition operation into the prediction step (or the update step). This structural change forces us to analyze the scheme invertibility; this property is not inherent to the scheme structure as in classical lifting. The following sections deal with the formal definition of this technique, with the GL design criterion and the latest results obtained applying it to image coding.

3.2 Definition

The main difference of the GL with respect to the Classical Lifting Scheme is the structure. In Classical Lifting, each step (prediction or update) has its addition or subtraction operation. Therefore, the scheme is restricted by linearity (although P and U block could be non-linear). Alternatively, the GL scheme avoids this restriction by including this addition in the P (and in the U) block and changing it by a non-linear operation if necessary. For this reason we will write about "mappings", from now on the mapping is the generalized operation that substitutes the addition and subtraction of the prediction and update steps. This change is what is really important in the Generalized Lifting Scheme. In Classical Lifting, even samples are used to compute a prediction over the odd samples. In GL, a mapping that may observe the even samples, takes the odd samples as input to directly generate the detail signal. Notice that the Classical Lifting Scheme is a particular case of the Generalized Lifting Scheme.

§ From Classical Lifting to Generalized Lifting Recall, for a moment, the Classical Lifting scheme:

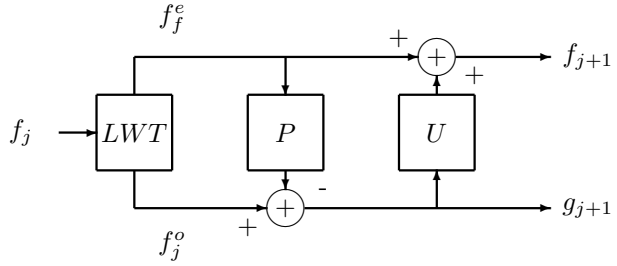

Figure 3.1: Classical Lifting Scheme basic structure.

In order to fix ideas, we will concentrate on a P step in the following. However, it is analogous to what happens in the U step. In Classical Lifting, the relationship between $f_{j+1}[n]$ and $f_j^o[n]$ (see Figure 3.1) is:

$$f_{j+1}[n] = f_j^o[n] - P(f_j^e[n+k_1], \ldots, f_j^e[n-k_2]) \tag{3.1}$$

in which the subtraction is a key point of the equation. The generalized Lifting Scheme uses $f_j^o[n]$ as input of the mapping and removes the subtraction operation. This is mathematically described by the following equation:

$$f_{j+1}[n] = P(f_j^o[n]; f_j^e[n+k_1], \ldots, f_j^e[n-k_2]) \tag{3.2}$$

where P is called a "prediction mapping" and the samples of f_j^e are called the context[14]. The next figure shows the block diagram of the GL:

[14] We deal with the notion of context in the next section, although it is a notion used in many systems like in some arithmetic coders as presented in 2.2.3.

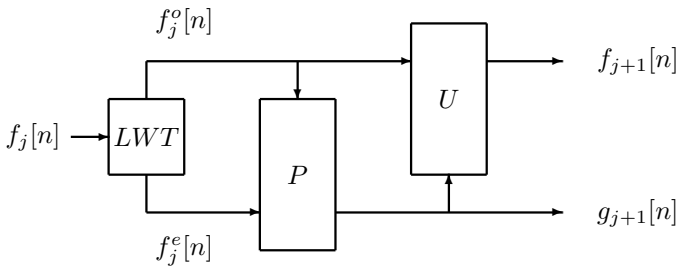

Figure 3.2: Generalized Lifting Scheme block diagram.

In case of adaptive lifting some samples of $f_j^e[n]$ and some causal samples $f_j^o[n]$ could be used to choose among different prediction steps. This is also taken into account in the GL including that samples in last equation, leading:

$$f_{j+1}[n] = P(f_j^o[n]; f_j^o[n-1], \ldots, f_j^o[n-m], f_j^e[n+k_1], \ldots, f_j^e[n-k_2]) \tag{3.3}$$

This generalization includes: linear lifting, non-linear lifting, adaptive lifting, ... This is the reason why it is called Generalized Lifting. It provides an ideal framework for the use of other lifting design criteria and other type of implementations. Usually the set $f_j^o[n-1], \ldots, f_j^o[n-m], f_j^e[n+k_1], \ldots, f_j^e[n-k_2]$ is called **the context**. In the following, this set will be very important, specially in the energy-minimizing design and latest results.

§ INVERTIBILITY The scheme Invertibility is not guaranteed now. We must ensure that the steps we design are invertible. Recall from set theory that:

Definition 3.1. *Let be A, B two (finite or infinite) sets. Let be $f : A \to B$ a mapping from A to B. Then f is called invertible if and only if it is injective (one-to-one), that is:*

$$f(x) = f(y) \Leftrightarrow x = y, \qquad \forall\, x, y \in A. \tag{3.4}$$

Obviously, the mapping f could not be one-to-one if the cardinality of the set A is greater than that of the set B. For example, set A could refer to all possible values for f^e, B could refer to all possible values for g_{j+1} and f could refer to P. In case where both cardinalities are equal and the mapping is one-to-one, it becomes automatically a bijective mapping. Summarizing, we must ensure that all mappings we design are injective, in order to guarantee their invertibility.

3.3 Energy Minimizing Design Criteria

Usually, the prediction step design is based on an energy minimization criteria. As in all lifting versions of a wavelet decomposition, prediction steps are focused on reducing the energy of the detail signal. This is the reason why wavelet-based image entropy coders suppose a hierarchical structure in which the subband energy decreases with the subband level. In this section we present an energy-minimizing criteria based on probability distributions. We will also suppose, from now on to the end of the report, that image coefficients are integers between -127 and 128 (that is, an 8-bit signed integer representation).

§ DEFINITION Suppose we get the input sample $x_o[n]$ and we know the probability density function for that sample. Then, the mapping assigns the most-probable inputs to the lower energy outputs. The following example illustrates this design:

Example 3.2. *Let X be the discrete random variable representing the value of the coefficients. Table 3.1 shows a probability density function (those values that are not represented have zero probability). In the table on the middle, the first column represents x_o, the second column represents* $\text{prob}(X_o = x_o)$ *and the third one represents $x'_o = P(x_o)$. We would like to remark that the choice between k and $-k$, has no effect over the output energy, since both k and $-k$ have the same energy, $k \in \mathbb{Z}$.*

Appendix 5.A of [26] proves that this construction leads to an energy-minimizing mapping. In fact it minimizes the expected energy (so it is optimal in that sense). There are three important questions to solve:

x	prob$(X = x)$
-3	0.2500
-2	0.1333
-1	0.0167
0	0.1667
1	0.1167
2	0.2333
3	0.0833

$\Longrightarrow$

x	prob$(X = x)$	x'_o
-3	0.2500	0
-2	0.1333	-2
-1	0.0167	3
0	0.1667	1
1	0.1167	2
2	0.2333	-1
3	0.0833	-3

$\Longrightarrow$

x	x'_o
-3	**0**
-2	**-2**
-1	**3**
0	**1**
1	**2**
2	**-1**
3	**-3**

Table 3.1: Example: mapping construction.

- What about "context"? Is it useful? How should we use it? Does it depend on the application? How?
- What about the probability distribution. Assuming the knowledge of the exact distribution seems to be unrealistic. We must solve that issue: should we estimate it? What kind of distribution do we have to use? Should it have a concrete shape? What about the distribution parameters?

In the section 3.4 we show 4 examples of GL utilization recently published. In these examples, these issues are discussed, and some of them, evaluated. After that section, we will describe some possibilities to take into account in the GL design.

§ The context The context is a set of pixels whose values are used to choose, among a set of distribution probabilities, the adequate[15] one for the sample we are predicting: that is a conditional probability distribution, because the probability distribution is a choice that depends on the context. But the key point is that good choices reduce the entropy and this is good to achieve high compression rates.
So depending on the values and the distribution of these values among the context, we choose one distribution or another. This context modelling is commonly used in literature for arithmetic coding models, adaptive lifting models, etc. Remark that this is possible due to the correlation between the predicted sample and its neighborhood[16].

§ Probability Distribution Estimation Apart from choosing which type of distribution we want to use, we must estimate it. There are two main strategies: global and local.

Global When GL uses this strategy, the distribution estimated remains invariant among the coding procedure. Indeed, it estimates a pdf for the whole image (the same for all pixels with the same context).

Local In this case the strategy uses information extracted from a causal neighborhood of the sample. The estimated distribution could (would) be modified along the transform procedure.

3.4 Latest Results

This section deals with the Generalized Lifting state-of-the-art. Four published articles are described and discussed. This will help us to reinforce those ideas we presented in the last section and to get familiarized with different Generalized Lifting implementations.

§ GL Potential Use for Image Coding This first approach investigates the potential use of GL to increase the sparseness of the wavelet coefficients for coding. It firstly applies a separable wavelet transform and then, the GL step tries to decorrelate the coefficients remaining from the wavelet transform. Referenced as [19], this article shows that, assuming complete knowledge of the probability distribution function (pdf), the GL reduces the coefficient energy and entropy. This assumption is unrealistic as the pdf would have to be transmitted to the decoder and an enormous amount of information would be necessary to send this pdf. The context used

[15] Here the "adequate" is the one that better predicts. This choice may be among distribution shape and/or parameters.
[16] Under the supposition of a perfect transform in terms of the output decorrelation, this context-dependent model has no sense.

Table 3.2: CDF 9/7 tap filter coefficients

	l_a	h_a	l_r	h_r
z^4	0.0378285	0.0645389		
z^3	0.0238495	−0.0406894	−0.0645389	0.0378285
z^2	−0.110624	−0.418092	−0.0406894	0.0238495
z	0.377403	0.788486	0.418092	−0.110624
1	0.852699	−0.418092	0.788486	−0.377403
z^{-1}	0.377403	−0.0406894	0.418092	0.852699
z^{-2}	−0.110624	0.0645389	−0.0406894	−0.377403
z^{-3}	−0.0238495		−0.0645389	−0.110624
z^{-4}	0.0378285			0.0238495
z^{-5}				0.0378285

are 1-D (sampled as the LWT) and 2-D (in a quincunx sampling grid). Moreover, the presented scheme (DWT + GL + Arithmetic Coding) results in a system with excellent performances in terms of rate-distortion curve.

§ PARTIAL KNOWLEDGE OF THE SIGNAL PDF The paper [20] deals with the Generalized Lifting used over a class of images for coding. It uses this class to estimate a pdf scanning several images. Before coding, all mappings have to be constructed. When this scheme is used, it has to be taken into account the fact the, as the pdf is "trained" over a set of images, there exists the possibility that some of the samples observed in the present image, have not been observed yet. A Decision Algorithm manages this kind of issues. The entropy coding is performed by an arithmetic coder. All experiments used the Sea Surface Temperature (SST) image class, but the main conclusion is not related to the class choice.

§ MODELING OF CONTOURS IN WAVELET DOMAIN In the published article referenced as [18], context-based models of contour in the wavelet domain are developed. This leads to the construction of mappings for a Generalized Lifting Scheme decomposition of a wavelet subband. A strategy to get a reduced number of (geometrically) structured models is described as well as the complete coding scheme. This is an example of global pdf estimation. We used a quincunx sampling grid for the GL iteration. The context we used is a cross. This scheme provides higher energy gains, but very low bit rate gains due to the spatial distribution of the mapped/non-mapped coefficients. This leads a very important research line: find an adequate entropy coder for the GL output. As exposed in the present project's aims, it is a important, but not solved, question.

§ ADAPTIVE LOCAL PDF ESTIMATION In this case, the title is very descriptive. The paper's approach (see [17]) is to estimate the pdf locally. This strategy is independent of the wavelet filter and context used (notice that in the previous cases, we must reestimate the distribution at each wavelet filter or context change) As well as in the previous case, a high energy gain is observed, but also a low coding gain (computed using an arithmetic coder).

3.5 Our Choice

Among these four GL implementations, we chose the last one because of its simplicity and its independence with respect to the context choice and the wavelet filter. This section combines some conceptual discussion and some implementation issues, and it is structured from high level to low level procedures: it describes the transforms we used in our source coding scheme.

§ DISCRETE WAVELET TRANSFORM We used a separable DWT to generate an horizontal detail subband. The filter used is the CDF 9/7 filter whose coefficients are reported in table 3.2. And the software used to compute this DWT is extracted from [6]. After the wavelet transform there is a quantization step in which we ignored their decimal part. This quantization step is the block in the source coding scheme in which we might lose some information. The other blocks of the source coding diagram are information lossless.

§ GENERALIZED LIFTING SCHEME Once we get a signed integer representation of the horizontal wavelet subband, we compute another decomposition to further decorrelate the signal. We choose among three possible decompositions represented in Figure 3.3:

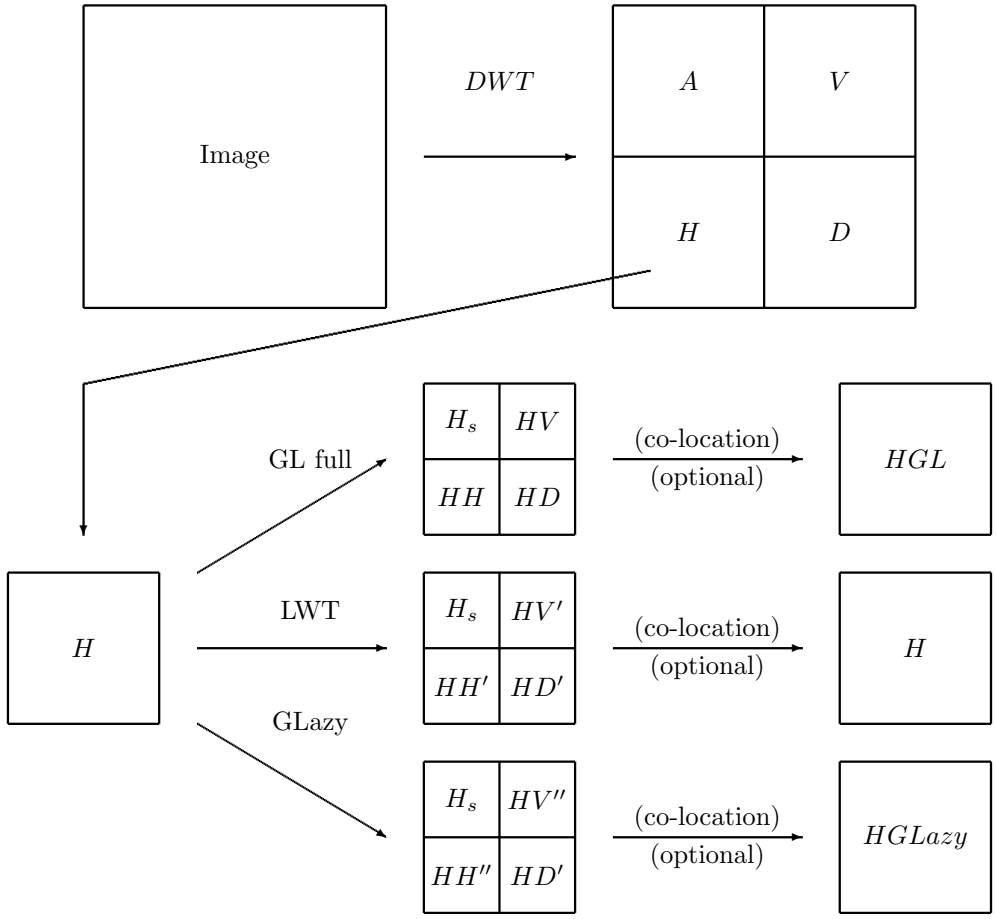

Figure 3.3: Spatial "GLazy" decomposition over the original image.

GL full This procedure computes a row-wise GL generating 2 subsubbands and then it computes a column-wise mapping over this 2 subsubbands to generate the definitive 4 subsubbands labelled: H_s, HV, HH and HD. The "GL full" block diagram is represented in Figure 3.4. Notice that, as we said, the row-wise mapping and the two column-wise mappings are implemented by the same block.

LWT In that case a 2D Lazy Wavelet Transform is applied to the subband. It first splits the subband in odd columns and even columns (row-wise). Secondly, it splits this subsubbands in odd rows and even rows (column-wise). In this way we reorder the coefficients of the original subband generating four subsubbands: H_s, HV', HH' and HD'.

GLazy The procedure "GLazy" computes a row-wise GL, generating 2 subsubbands. It also computes a column-wise GL to generate 4 subsubbands as in the "GL full" case. However, this procedure replaces HD'' for HD', the diagonal subsubband of the LWT transform. Summarizing it generates four subsubbands: H_s, HV'', HH'' and HD'. The block diagram is represented in Figure 3.5.

In the following lines it is mandatory to differentiate between subband (group of coefficients obtained with the first transform step, DWT) and subsubband (group of coefficients obtained with the second transform step). Notice that in all cases the approximation subsubband is the same (H_s): a subsampled version of the original H subband.
A part from the technique used to change the value of the coefficients, it is possible to reorder the subsubband coefficients to generate a **co-located** version of the transformed subband. We can reorder the subsubband image in such a way that each coefficient is in its original place in the subband image. As can be seen in Fig. 3.3, if we use the co-location strategy after the "GL full" (respectively "LWT" and "GLazy") decomposition, the resulting set of coefficients is called HGL (respectively H and $HGLazy$). . Notice that this reordering is

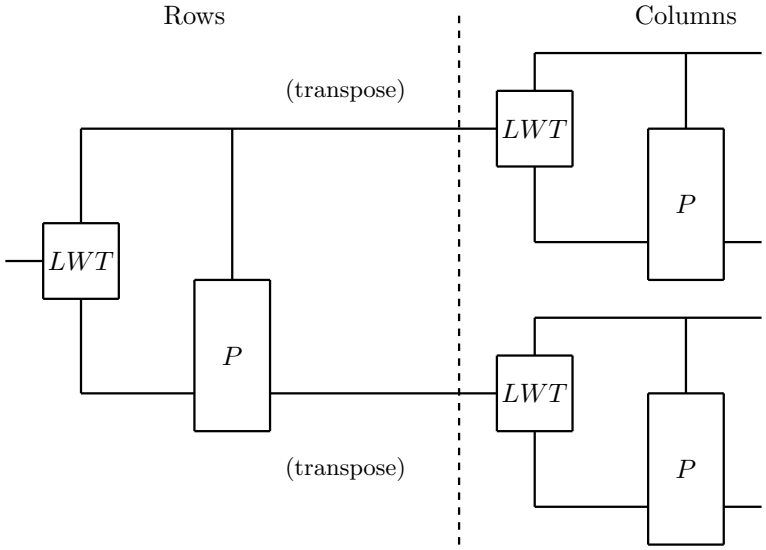

Figure 3.4: "GL full" decomposition block diagram.

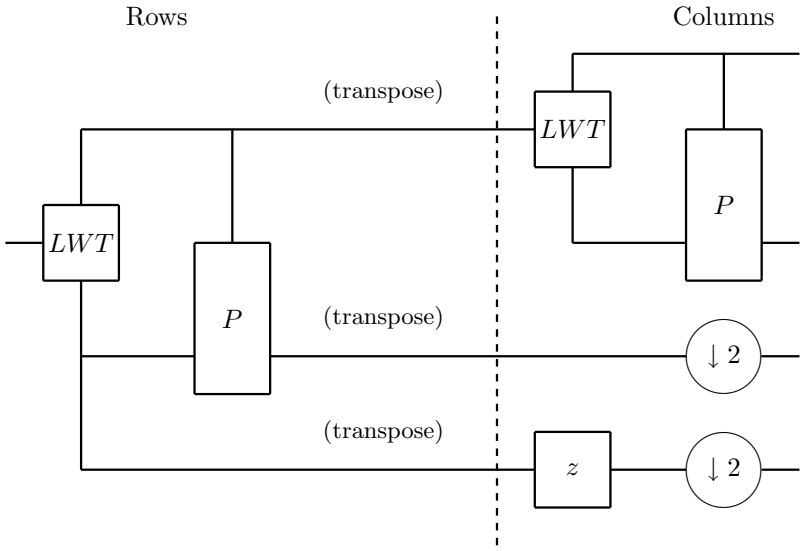

Figure 3.5: "GLazy" decomposition block diagram.

Figure 3.6: Context spatial distribution. The pixel to predict is ★ and we refer to that context as $C_{\bigstar}$.

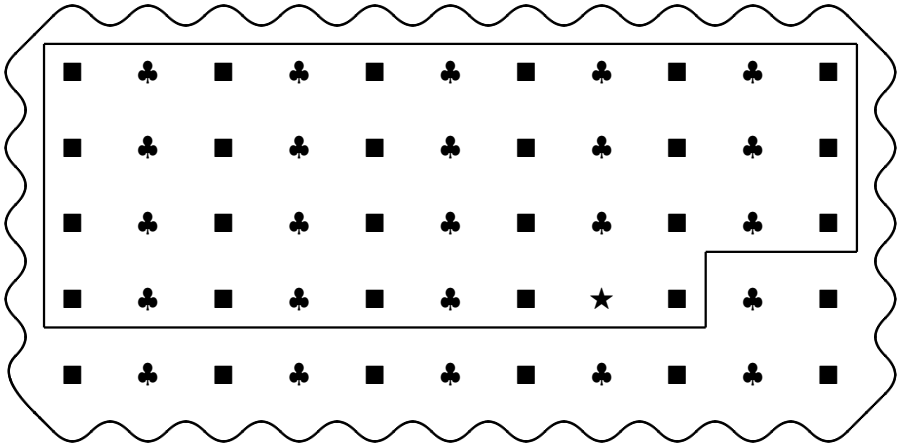

Figure 3.7: Schematic search window.

equivalent to apply an Inverse Lazy Wavelet Transform to the subsubband image.
Figure 3.8 shows an example of the GL computation in horizontal DWT subband of Barbara. The top left image is the original subband, the bottom left image is the subband in a hierarchical structure (after the 2D LWT), the bottom right subband if the GL (in that case we used "GLazy") subband with hierarchical structure and in the top right of the image there is the co-located GL subband.
Although it have not been yet published, the "GLazy" scheme is well-behaviored in terms of coding gain with respect to the one-scale classical GL hierarchical decomposition (see [21]). From now on, to simplify the wording, we forget about DWT (although we use it) and we use the term subbands referring to GL subbands.

§ P BLOCK: PREDICTION MAPPING The Generalized Lifting with Adaptive Local PDF Estimation [17] we use is based only on a prediction mapping P (no update). To estimate the pdf (and therefore compute the prediction), we search among all possible contexts inside a search window, the one that minimizes the distance with respect to the context of the sample to predict. In case there is only a context that reaches that minimum and this distance value is below a threshold, we use the corresponding coefficient of this (suitable) context as the prediction value. We estimate a mono-modal probability density function, whose mode is the prediction value. Therefore, our mapping becomes a subtraction: in order to calculate the mapping's output, we subtract to the predicted sample the corresponding one in this suitable context (i.e., the prediction value). To define P, we must specify a context for each pixel to encode, a search window, a distance between contexts and a threshold:

The context We chose a context that involves 5 samples. The spatial distribution, when the prediction is computed row-wise, is shown in Figure 3.6. ♦ and ■ are odd samples and ♣ and ★ are even samples[17]. The sample to be predicted is ★. Note that values ■ and ♦ are all causal with respect to ★ because they are not in the same phase than ★.

Search window We define a window (height and width) to search for context similar to the predicted sample one. This window must be causal with respect to ★, i.e., the center values of the contexts inside the search window must be causal with respect to the value under prediction. Figure 3.7 represents this window.

[17]Due to the existence of the LWT we have to manage with two type of samples: the odd phase (odd samples) and the even phase (even samples). Although in the original paper of classical lifting scheme, the even phase was used to predict the odd one, our implementation inverts these roles. This has no effect over the performance because both phase are supposed to be statistically very similar.

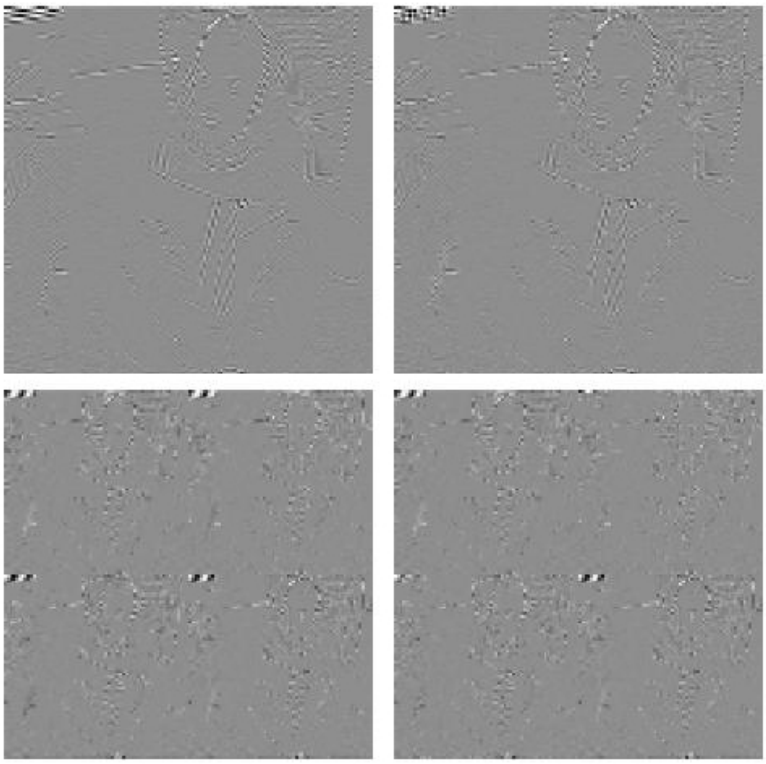

Figure 3.8: Different transform outputs for the horizontal DWT subband of Barbara.The top left image is the original subband, the bottom left image is the subband in a hierarchical structure (after the 2D LWT), the bottom right subband is the GL (in that case we used "GLazy") subband with hierarchical structure and in the top right of the image there is the co-located GL subband.

Distance To determine the similarity between two contexts, we need a distance. We used the weighted distance shown in Equation 3.5.

$$d\left(C_1, C_2\right) = \frac{\left(\clubsuit_1 - \clubsuit_2\right)^2 + \left(\blacksquare_1^L - \blacksquare_2^L\right)^2 + \left(\blacksquare_1^R - \blacksquare_2^R\right)^2 + \alpha\left(\left(\blacklozenge_1^L - \blacklozenge_2^L\right)^2 + \left(\blacklozenge_1^R - \blacklozenge_2^R\right)^2\right)}{\clubsuit_1^2 + \left(\blacksquare_1^R\right)^2 + \left(\blacksquare_1^L\right)^2 + \alpha\left(\left(\blacklozenge_1^L\right)^2 + \left(\blacklozenge_1^R\right)^2\right) + \epsilon}, \tag{3.5}$$

where $\epsilon \approx 10^{-9}$ is fixed in order to avoid exception in case all values of C_1 are zero and α is a weight factor used to take into account that $\blacklozenge$ samples are geometrically far from $\clubsuit$ and $\blacksquare$.

Threshold Once a distance is defined, we also define a threshold. All contexts whose distance with respect to $C_\bigstar$ is greater than T will not be taken into account.

Once the prediction mapping is defined and, therefore, the way we compute the GL transform is clear, we are able to present our research work. The first part called Background ends here and in the following pages the reader can find a report on what have we done, which experiments we have conducted, which are the results for these experiments and the conclusions we extracted from the results.

Part II

Image Entropy Coders: Implementation, Modification and Evaluation

Chapter 4

Methodology

This chapter deals with the work methodology we used to develop that research. The project's Director, Dr. Philippe Salembier, Mr. Julio Rolon and me used to meet once a week to follow this research project and Mr. Rolon's PhD Thesis. In that case, our methodology consisted basically in 3 steps:

Implement and Check algorithms tasks and batches providing

Results that we deeply analyzed to extract some

Conclusions which were our guide and played a key role to properly iterate this process.

It was my task to implement and check algorithms and other programs. By recommendation of Dr. Salembier, I implemented all my programs in MATLAB®. I would like to strongly thank both my project's director and his PhD student, who widely helped me to achieve my objectives as far as I went.
After that implementation and checking step, a deep analysis of the results had been done. Moreover we extracted some interesting conclusions that guided us to choose new implementation tasks. I took a lot of profit of that weekly meetings.
In the checking step, there is a special issue that is mandatory for all coding/decoding implementations: the invertibility must be guaranteed. In the lossless case, the decoder must be able to recover exactly the same signal that was at the input of the encoder. Once that is checked, one could start analyzing algorithm's performance. We focused on the rate-distortion curve: it is our quality measure. Summarizing, we used a check-gather-conclude methodology and we only worried about algorithm characteristics that are strictly related with signal processing. In the next chapters we present two implemented algorithms, some experiments, the results we obtained and our conclusions on the work.

Chapter 5

Set Partitioning Embedded bloCK coder (SPECK)

5.1 Introduction

SPECK algorithm was published in [14]. It's the evolution of the SPIHT, but without taking benefit of the inter-subband correlation. Its aim is to efficiently code large blocks of zeros, quickly identifying those pixels with a large amount of information. The significance of the coefficients is measured by their energy. So, the greater is the pixel's magnitude, the soonest we should start transmitting that pixel.

5.2 SPECK main features

In order to efficiently sort and transmit the information (e.g.: an image x), the hierarchical structure containing the coefficients is partitioned in two sets: S and I. The S set corresponds to the approximation level of the hierarchical subband decomposition. The I set corresponds to the rest of the coefficients. Due to the DWT structure expected at the input of the SPECK, the S set has higher energy coefficients than the I set. This is the reason why the search for important coefficients starts in the S set.
But how could we decide if a set is important or it is not? Starting with the most significant bit plane, the algorithm compares the coefficients in the set under the evaluation, A, against the bit plane following the next rule:

$$\sigma_n(A) = \begin{cases} 1 & \exists\,(j,k) \in A \mid |x[j,k]| \geq 2^n \\ 0 & \text{otherwise} \end{cases} . \tag{5.1}$$

In case $\sigma_n(A) = 1$, we say that the set A is significant with respect to the n-th bit plane, otherwise, the set A is not significant (or insignificant) with respect to the n-th bit plane. After marking A as significant, the algorithm splits the set into four subsets and iterates the process up to the coefficient level if necessary. This procedure quickly finds out where significant coefficients are. When the S set is fully scanned, SPECK tests the I set significance. In case it is significant, the I set is split as in figure 5.1 and each S_i is processed as an S set. Finally, when a coefficient is marked as significant, the encoder sends its sign. When the whole image is scanned, SPECK rescans the image (maintaining partitions of the n-th bit plane) testing insignificant sets and coefficients with respect to the $(n-1)$-th bit plane and refining those coefficients found significants with respect to previous bit planes (that is, sending 0 or 1 depending on its binary representation).
SPECK algorithm maintains 4 dynamic lists[18]:

LIS: List of Insignificant Sets These are all sets marked as not significant.

LIP: List of Insignificant Pixels These are all coefficients marked as not significant.

LNP: List or Newly significant Pixels These are all coefficients found significant in that bit plane.

LSP: List of Significant Pixels These are all coefficients found significant in previous bit planes.

Those lists are maintained by both, the encoder and the decoder. The bit stream order is related to the lists order. For example, in the refinement step, bits are sorted in the same order as LSP is and in the sorting step, the significance bits are sorted in the same order than n the LIP and the LIS. There are four routines in the algorithm:

ProcessS This routine determines and outputs the significance of a set/coefficient. In case a set becomes significant, `ProcessS` calls to `CodeS`. In case a coefficient become significant, its sign is output.

CodeS It splits the block in four sub-blocks and applies to each block the `ProcessS` routine.

[18] In fact, in the original article only 2 lists are maintained. This 2 additional lists (LIP and LNP) have no effect on the algorithm's performance, but the implementation becomes easier.

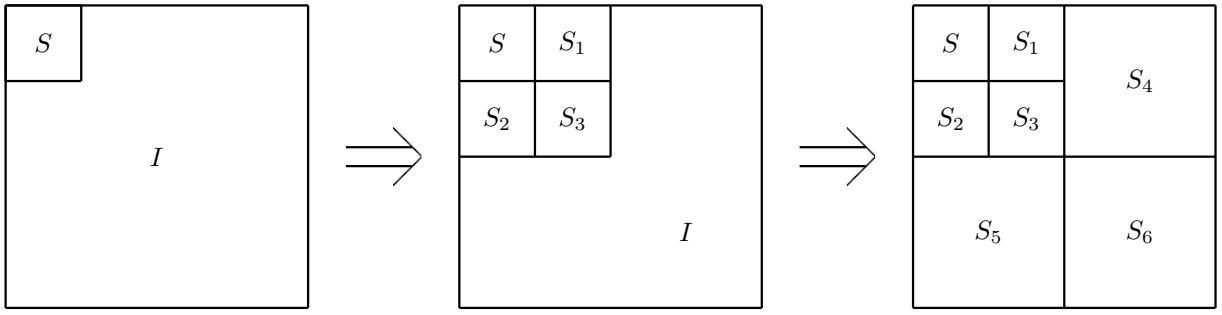

Figure 5.1: SPECK I set splitting scheme.

5	-3	2	-2
2	1	1	2
2	3	0	1
-1	0	-1	0

Figure 5.2: Wavelet image transform example.

ProcessI This routine determines and outputs the significance of the I set, if any. In case of significance, it calls `CodeI`.

CodeI It splits the I set into four subsets as show in image 5.1 and applies to each of S_i the routine `ProcessS`. After that, it calls `ProcessI`.

5.3 Pseudocode

Once presented the main features and variables of the algorithm, we present its pseudocode, extracted from the original article [14]. There are some modifications because, the algorithm maintains 4 lists instead of 2. The main algorithm is in Algorithm 5.1 and the four SPECK routines described in the last section are in Algorithms 5.2, 5.3, 5.4 and 5.5.

Notice that the main algorithm is has four steps. The first one, the initialization defines the S set and the I set and it outputs the most significant bit plane. In the initialization step, the algorithm also defines the 4 lists. The other 3 steps are inside a loop, that repeats this steps from the most significant bit plane ($n = n_{\max}$) to the least significant one ($n = 0$). Inside the loop, the first step is the sorting step, in which all insignificant sets (with respect to previous bit planes) are checked to know if they are significant with respect to the bit plane scanned (n). These sets are those that are in the LIP, those that are in the LIS and the I set, if it is not and empty set. After the sorting step, there is the refinement step: all pixels that are in the LSP (i.e., that had been found significant with respect to previous bit plane) are refined: the algorithm outputs the n-th bit of their binary representation. Finally, there is the quantization step, in which the scanned bit plane decreases by one ($n = n - 1$) and the pixels in the LNP are moved to the LSP. Summarizing, SPECK's bit stream has 3 kind of bits: the significance bits, the sign bits (both output in the sorting pass) and the refinement bits (output in the refinement pass).

Table 5.1 shows the SPECK algorithm's performance step by step when it is applied to the coefficients image represented in 5.2. Notice that the first bit plane is coded with only 3 bits instead of 16 bits (one per coefficient) that would have in a trivial compression scheme. Notice also that the 2×2 set whose upper-left corner is in $(3, 3)$ is coded with a single bit until the least significant bit plane. This one is the key point of the SPECK algorithms: this is its main aim. Although this is not a real situation, it illustrates very well that SPECK

Table 5.1: An example of the application of the SPECK algorithm.

Comment	Set	Action	Bit	LIP	LIS	LSP	LNP
Initialize		S and I		(1,1)			
Sorting (n=2)							
Test LIP	(1,1)	to LNP	1				(1,1)
		code sign	1				(1,1)
Test I	I		0				(1,1)
Quantization		LNP to LSP				(1,1)	
Sorting (n=1)							
Test I	I	split I	1			(1,1)	
	(1,2)	to LNP	1			(1,1)	(1,2)
		code sign	0			(1,1)	(1,2)
	(2,1)	to LNP	1			(1,1)	(1,2),(2,1)
		code sign	1			(1,1)	(1,2),(2,1)
	(2,2)	to LIP	0	(2,2)		(1,1)	(1,2),(2,1)
Test I	I	split I	1	(2,2)		(1,1)	(1,2),(2,1)
Test S	(1,3,2,2)	split S	1	(2,2)		(1,1)	(1,2),(2,1)
	(1,3)	to LNP	1	(2,2)		(1,1)	(1,2),(2,1),(1,3)
		code sign	1	(2,2)		(1,1)	(1,2),(2,1),(1,3)
	(1,4)	to LNP	1	(2,2)		(1,1)	(1,2),(2,1),(1,3),(1,4)
		code sign	0	(2,2)		(1,1)	(1,2),(2,1),(1,3),(1,4)
	(2,3)	to LIP	0	(2,2),(2,3)		(1,1)	(1,2),(2,1),(1,3),(1,4)
	(2,4)	to LNP	1	(2,2),(2,3)		(1,1)	(1,2),(2,1),(1,3),(1,4),(2,4)
		code sign	1	(2,2),(2,3)		(1,1)	(1,2),(2,1),(1,3),(1,4),(2,4)
Test S	(3,1,2,2)	split S	1	(2,2),(2,3)		(1,1)	(1,2),(2,1),(1,3),(1,4),(2,4),(3,1)
	(3,1)	to LNP	1	(2,2),(2,3)		(1,1)	(1,2),(2,1),(1,3),(1,4),(2,4),(3,1)
		code sign	1	(2,2),(2,3)		(1,1)	(1,2),(2,1),(1,3),(1,4),(2,4),(3,1)
	(3,2)	to LNP	1	(2,2),(2,3)		(1,1)	(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)
		code sign	1	(2,2),(2,3)		(1,1)	(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)
	(4,1)	to LIP	0	(2,2),(2,3),(4,1)		(1,1)	(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)
	(4,2)	to LIP	0	(2,2),(2,3),(4,1),(4,2)		(1,1)	(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)
Test S	(3,3,2,2)	to LIS	0	(2,2),(2,3),(4,1),(4,2)	(3,3,2,2)	(1,1)	(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)
Refinement							
	(1,1)		0	(2,2),(2,3),(4,1),(4,2)	(3,3,2,2)	(1,1)	(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)
Quantization		LNP to LSP				(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	
Sorting (n=0)							
	(2,2)	to LNP	1	(2,3),(4,1),(4,2)	(3,3,2,2)	(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2)
		code sign	1	(2,3),(4,1),(4,2)	(3,3,2,2)	(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2)
	(2,3)	to LNP	1	(4,1),(4,2)	(3,3,2,2)	(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3)
		code sign	1	(4,1),(4,2)	(3,3,2,2)	(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3)
	(4,1)	to LNP	1	(4,2)	(3,3,2,2)	(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1)
		code sign	0	(4,2)	(3,3,2,2)	(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1)
	(4,2)	to LIP	0	(4,2)	(3,3,2,2)	(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1)
	(3,3,2,2)	split S	1	(4,2)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1)
	(3,3)	to LIP	0	(4,2),(3,3)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1)
	(3,4)	to LNP	1	(4,2),(3,3)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4)
		code sign	1	(4,2),(3,3)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4)
	(4,3)	to LNP	1	(4,2),(3,3)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4),(4,3)
		code sign	0	(4,2),(3,3)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4),(4,3)
	(4,4)	to LIP	0	(4,2),(3,3),(4,4)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4),(4,3)
Refinement							
	(1,1)		1	(4,2),(3,3),(4,4)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4),(4,3)
	(1,2)		1	(4,2),(3,3),(4,4)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4),(4,3)
	(2,1)		0	(4,2),(3,3),(4,4)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4),(4,3)
	(1,3)		0	(4,2),(3,3),(4,4)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4),(4,3)
	(1,4)		0	(4,2),(3,3),(4,4)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4),(4,3)
	(2,4)		0	(4,2),(3,3),(4,4)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4),(4,3)
	(3,1)		0	(4,2),(3,3),(4,4)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4),(4,3)
	(3,2)		1	(4,2),(3,3),(4,4)		(1,1),(1,2),(2,1),(1,3),(1,4),(2,4),(3,1),(3,2)	(2,2),(2,3),(4,1),(3,4),(4,3)

efficiently code large block of low energy.
Our MATLAB implementation uses 4 lists instead of 2. This has some consequences on the algorithm. The changes we made to maintain these 4 lists are highlighted with blue in the pseudocode. In the main algorithm, we initialize all 4 lists, we scan the LIP before the LIS and, in the quantization step, we add those pixels in the LNP to the LSP. In case we follow the original algorithm, in the `ProcessS` routine, we will add the significant pixel to the LSP instead to the LNP and we will remove it from the LIS, not from the LIP. Similarly, in the `CodeS` routine we will add it to the LSP instead to the LNP and we will add it to the LIS, not to the LIP.

Algorithm 5.1 SPECK algorithm pseudocode

Require: An input matrix X and an subband sizes array.
  // Initialization
  Set S and I.
  `output` $n_{\max} = \lfloor \log_2(\max_{(i,j)\in\mathcal{X}} |\mathcal{X}(i,j)|) \rfloor$.
  $n = n_{\max}$
  Add S to the LIS (or LIP).
  Set LSP and LNP as empty lists.
  repeat
    // Sorting Pass
    for $(i,j) \in$ LIP **do**
      `ProcessS`$((i,j))$
    end for
    for S $\in$ LIS, from the smallest to the largest **do**
      `ProcessS`(S)
    end for
    if I $\neq \emptyset$ **then**
      `ProcessI`()
    end if
    // Refinement Pass
    for $(i,j) \in$ LSP **do**
      `output` the n-th bit of $|\mathcal{X}(i,j)|$
    end for
    // Quantization Step
    Add the pixels in the LNP to the LSP
    $n = n - 1$
  until $n < 0$

5.4 Implementation issues

In this section we discuss some issues we had to manage along the implementation process of the presented pseudo-code. We had to overcome several difficulties, but there are two interesting problems. The first one deals with a size restriction on the input signal. And the other one deals with a time consumption issue.

§ INPUT IMAGE SIZE Due to the dyadic[19] flavor of the wavelet transform, many wavelet-based image entropy coder implementations expect an input signal whose dimensions are a power of two. Despite the fact that the algorithm (in a more conceptual level) does not restrict the input image size, its implementation does. Obviously, that implementation task becomes easy if the programmer knows some extra information about the input signal: the image sizes are the same and a power of two. In that case, it is very easy to know all subband limits. Moreover, all sets (those that are in the LIS, or those that have to be checked by the algorithm) are square and dyadic. This restriction is very strong: there are not many natural images satisfying it.
We must solve two main issues: the subband limits and the set partition rule. The first one is demanded to the user. The second feature is solved internally. The problem is reduced to split sets whose dimensions are not even. Suppose that the split set, S, is $2m+1 \times 2n+1$ and that we split S in four subsets S_0, S_1, S_2 and S_3 as in Figure 5.3. However, in case the set is very eccentric, the SPECK algorithm splits this set is such a way that

[19] The sizes are powers of two.

Algorithm 5.2 Procedure **ProcessS**.

```
PROCEDURE ProcessS(S)
output σ_n(S).
if σ_n(S) = 1 then
  if S is a pixel then
    output its sign and add to the LNP.
    if S ∈ LIP then
      Remove S from the LIP.
    end if
  else
    CodeS(S).
    if S ∈ LIS then
      Remove S from the LIS.
    end if
  end if
else
  if S is a pixel then
    if S ∉ LIP then
      Add S to the LIP.
    end if
  else
    if S ∉ LIS then
      Add S to the LIS.
    end if
  end if
end if
```

Algorithm 5.3 Procedure **CodeS**.

```
PROCEDURE CodeS(S)
Split S in four similar subsets O(S).
for each S_i ∈ O(S) do
  output σ_n(S_i)
  if σ_n(S_i = 1) then
    if S_i is a pixel then
      output its sign and add it to the LNP.
    else
      CodeS(S_i).
    end if
  else
    if S_i is a pixel then
      Add it to the LIP.
    else
      Add it to the LIS.
    end if
  end if
end for
```

Algorithm 5.4 Procedure **ProcessI**

```
PROCEDURE ProcessI()
output σ_n(I).
if σ_n(I) = 1 then
  CodeI()
end if
```

Algorithm 5.5 Procedure **CodeI**

```
PROCEDURE CodeI()
Split I into four sets (three S_i and one I).
for each S_i do
  ProcessS(S_i)
end for
ProcessI
```

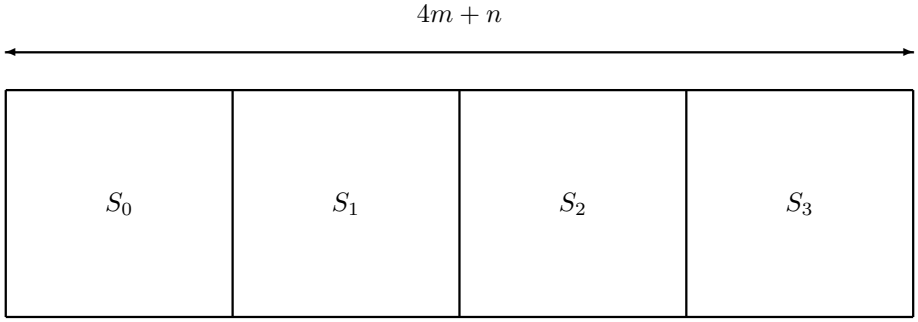

Figure 5.4: SPECK partition rule for eccentric sets.

the resulting subsets are not as eccentric as the original set was. Figure 5.4 shows the SPECK partition rule for an horizontal eccentric set. Although we explain the rule for the horizontal sets, it is analogous for the vertical ones. We consider that an horizontal set is eccentric when $l_h > 4l_v$, where l_h is the horizontal dimension and l_v is the vertical dimension. We apply the Euclidean division to l_h: $l_h = 4m + n$. The SPECK partition rule forces the n first subsets' horizontal dimension to $m + 1$ and the rest with m (the subset order is from the left to the right).

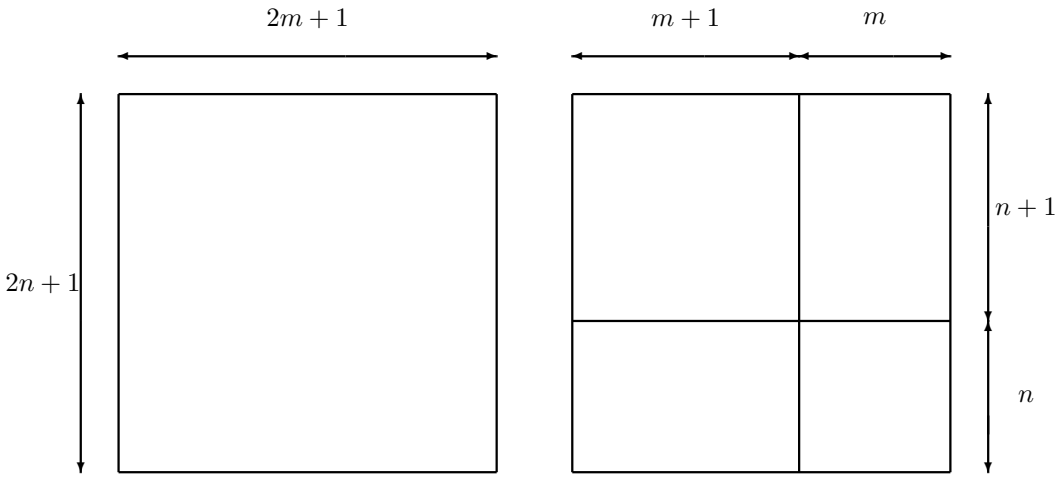

Figure 5.3: Example no how SPECK splits an odd sized set.

§ LIS: SORTING OF A VERY LARGE LIST As we said, the SPECK algorithm maintains four lists: LIS, LIP, LSP and LNP. The LIS has to be sorted in increasing order of size as specified in the original article (in fact, algorithm scans LIP before LIS). To realize about that, suppose we now start the sorting pass and that there are two sets, A_s and A_l, that have not become significant, i.e., they are in the LIS. Suppose that A_s is smaller than A_l. Then, the probability that A_s becomes significant in this bit plane is greater that the probability that A_l becomes significant in this bit plane. This is due to the coefficient correlation and that the smaller set A_s has nearer than A_l, a coefficient that became significant in a previous bit plane.

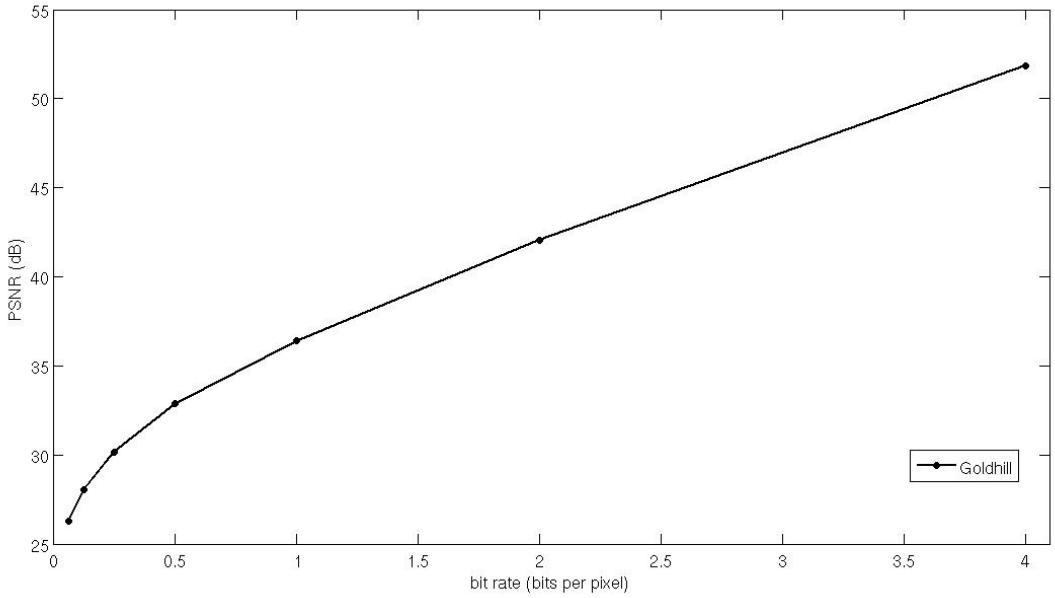

Figure 5.5: SPECK rate-distortion curves for Goldhill.

The reader may be confused because there is no difficulty in sorting a list. However, the LIS is a very big list sometimes: there is a large amount of sets that could not have been yet found significant at some point in the algorithm. If we sort the list before its scanning, we waste an incredible amount of time. Although we are not worried about time consumption, we want to do experiments, and we would have to encode several images. Therefore, we implemented a routine to add elements in a sorted list. As a consequence, we reduced the SPECK time consumption in such a way that we could compute our experiments in a reasonable time interval.

5.5 Performance

We include this section here in order to check the algorithm's performance. We would like to know if our implementation's performance is similar to the one in the original paper. We applied a 5 level wavelet transform with the `bior4.4` filter of MATLAB Wavelet toolbox to the 512×512 image Goldhill. Then we ignore the decimal part of the wavelet coefficients and we execute SPECK over the image. We compute the inverse of SPECK and the inverse wavelet transform to compare original and restored images for all following bit rates:

$$0.25 \qquad 0.5 \qquad 1$$

Results are presented in Table 5.2.

Table 5.2: SPECK performance benchmark for Goldhill.

Bit rate	Original	MATLAB
0.25	30.31	30.16
0.5	32.87	32.85
1	36.20	36.38

We conclude, from this little experiment, that our SPECK's implementation is enough similar to the original paper's results. Figure 5.5 shows the rate distortion curve.

Chapter 6

MQ Arithmetic Coder

6.1 Introduction

This Chapter deals with the arithmetic coder used in the JPEG2000 standard, precisely it deals with the MATLAB implementation we have developed. As we have said in the second chapter, the MQ-coder is a **binary alphabet** adaptive low-precision multiplication-free arithmetic coder. We used the MQ-coder to arithmetically encode the bit stream output by the SPECK encoder. It is not a new concatenation, because the EZBC (see [9]) uses an arithmetic coder after the SPECK algorithm. However they not use the GL technique.
The Chapter structure is very similar to the previous one: we present its main features, its pseudocode and its performance. However, we are not going to develop implementation issues we had, because they are not conceptually interesting.

6.2 MQ-coder main features

MQ-coder has 4 main features:

- Its input symbols are binary and each symbol must be associated to a label. This label is used to split up the bit stream into different kind of bits in order to take profit of different probability distributions: each label represents one of these distributions. So we are conditioning in order to decrease the entropy, as in the GL case. In the encoding procedure, the label is used to know something a priori about the bit's probability distribution.
- It is adaptive: the algorithm learns to estimate the pdf to improve its performance. This adaptation is implemented through the use of two tables (see section below).
- It uses low-precision arithmetics. There are 5 registers in the MQ coder: A and C are the lower bound and the upper bound of the interval (I^k in section 2.2), T is a byte register that accumulates bits from the coding procedure, t shows how many information bits are in the T register (it is used for the bit stuffing policy, see section 6.4) and L register counts the byte stream length.
- It is a multiplication-free arithmetic coder. Therefore it is faster than other coders. Despite this is a desirable characteristic in a hardware scenario, it is not really important in a high-level programming language like MATLAB. However, we made that choice to be able to compare our results to JPEG2000, which uses that arithmetic coder as its entropy coder.

6.3 Performing adaptation: Dynamic and Static tables

But, how can MQ-coder adapt itself to the input signal? The statistics block is implemented as finite state machine (FSM). There are two tables to take into account:

Dynamic Table: Contains the information of each context. Each row represents a context and there are two columns. For each context, the first column represents the state of the Static Table in which the the context stays at that moment (the state in the FSM, called Σ_k) and the second column shows the "most probable symbol" 0 or 1 (called s_k). The state of the Static Table represents the probability of that "least probable symbol". Table 6.1 shows an example of a Dynamic Table. The third row represents a context and it is in the State 12 of the Static Table. Moreover the MPS of this context in that moment is `1`.

Static Table: This table describes the update statistics decision algorithm and has four columns:

Σ_{mps} In case we receive the most probable symbol and we must renormalize the interval, the algorithm may change the state in the Dynamic Table: $\Sigma_k = \Sigma_{mps}(\Sigma_k)$. Table 6.3 shows an example of the

Table 6.1: Example of a Dynamic Table for 5 contexts.

Context	State	MPS
1	3	0
2	5	1
3	12	1
4	1	0
5	34	1

MQ-coder performance. The lines whose symbol is printed in green symbolize a state change when the symbol received is the expected one.

Σ_{lps} In case we receive the least probable symbol, the algorithm change the state in the Dynamic Table: $\Sigma_k = \Sigma_{lps}(\Sigma_k)$. In Table 6.3, the lines whose symbol is printed in red symbolize a the state change when the received symbol is not the expected one.

λ In case we receive the least probable symbol, this number indicates if we have to change the role of MPS and LPS.

p This value represents the probability of the LPS.

In the Static Table, there are three set of states. The first set (from 1 to 6) is the initial set: all pointers of Dynamic Table must signal one of this states when we start coding. This first set is used to initially train the adaptive FSM. If the received symbols are not the expected one, the algorithm goes to the second set (states from 7 to 14). Finally the algorithm arrives to the third set (states from 15 to 46). The second set is an intermediate stage in order to train a little bit more the FSM. The third state corresponds to the stable zone of the table: when the algorithm arrives at this set, it is supposed to be trained. Once we arrive at the third set, we did not go back. The reader should observe that there is another set composed by only one state. This is used in JPEG2000, but we are not going to use it. One could gather more information in [30].
The state in the Static Table also affects to the output length. The higher is the state in the table, the lower is the estimated probability (represented by the state) of the "least probable symbol", and the shorter is the output length. This Static Table is shown as Table 6.2 and the state diagram is printed in Figure 6.1.
To clarify a little bit the MQ-coder algorithm, we present two examples (Tables 6.3 and 6.4). In the first example (example A) the probability of the MPS is 0.99 and in the second example (example B) the probability of the MPS is 0.6. In both cases the MPS is 1. In both examples, the first two columns are the input symbol and the input label respectively. The third column is the state of the FSM in which the label stays before code the symbol. The fourth column is the expected symbol. The fifth is the probability of the **Least** Probable Symbol (expressed as a 32-bit integer) and the sixth is the same probability normalized to the unit interval (that is $P_{mps} = 1 - P_{lps}$). Notice that in both cases the MQ-coder estimates adaptively the probability distribution: in the example A the probability of the LPS is nearer 0.007, therefore, the probability of the MPS is nearer 0.993. In the example B, the mean estimated probability of the LPS is nearer 0.42 which corresponds to a estimated MPS probability near 0.58.

6.4 Pseudocode

This section deals with the pseudocode presented in [30] for the MQ-coder. There are mainly two types of routines: those used in the encoding procedure and those used in the decoding procedure. Before their pseudocode, we briefly present on the aim of each routine:

Encoding Procedures:

Initialization (Alg. 6.1) This routine initializes all registers of the MQ-encoder.

Encode (Alg. 6.2) This one encodes a symbol into the T register. Moreover it also performs the arithmetic coder interval renormalization (that is necessary for all low-precision implementations).

Table 6.2: Static Table of the MQ-coder.

k	Σ^k_{mps}	Σ^k_{lps}	λ	P	k	Σ^k_{mps}	Σ^k_{lps}	λ	P
1	2	2	1	0.4745	24	25	22	0	0.1876
2	3	7	0	0.2869	25	26	23	0	0.1545
3	4	10	0	0.1324	26	27	24	0	0.1324
4	5	13	0	0.0593	27	28	25	0	0.1214
5	6	30	0	0.0283	28	29	26	0	0.1104
6	39	34	0	0.0117	29	30	27	0	0.0993
7	8	7	1	0.4745	30	31	28	0	0.0938
8	9	15	0	0.4635	31	32	29	0	0.0593
9	10	15	0	0.3973	32	33	30	0	0.0538
10	11	15	0	0.3090	33	34	31	0	0.0476
11	12	18	0	0.2649	34	35	32	0	0.0283
12	13	19	0	0.1986	35	36	33	0	0.0235
13	14	21	0	0.1545	36	37	34	0	0.0145
14	30	22	0	0.1214	37	38	35	0	0.0117
15	16	15	1	0.4745	38	39	36	0	0.0069
16	17	15	0	0.4635	39	40	37	0	0.0059
17	18	16	0	0.4469	40	41	38	0	0.0029
18	19	17	0	0.3973	41	42	39	0	0.0016
19	20	18	0	0.3090	42	43	40	0	0.0008
20	21	19	0	0.2869	43	44	41	0	0.0005
21	22	20	0	0.2649	44	45	42	0	0.0002
22	23	20	0	0.2207	45	46	43	0	0.0001
23	24	21	0	0.1986	46	46	44	0	0.4745

Table 6.3: MQ-coder performance example A.

Symbol	Label	State	MPS	Prob	N.Prob.
-1	1	6	1	545	0.012
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
1	1	39	1	273	0.006
0	1	37	1	545	0.012
1	1	37	1	545	0.012
1	1	37	1	545	0.012
1	1	37	1	545	0.012
1	1	38	1	321	0.007
1	1	38	1	321	0.007
1	1	38	1	321	0.007
1	1	38	1	321	0.007

Table 6.4: MQ-coder performance example B.

Symbol	Label	State	MPS	Prob	N.Prob.
-1	1	1	1	22017	0.475
1	1	2	1	13313	0.287
0	1	7	1	22017	0.475
0	1	7	0	22017	0.475
0	1	8	0	21505	0.463
1	1	15	0	22017	0.475
1	1	15	1	22017	0.475
1	1	16	1	21505	0.463
1	1	17	1	20737	0.447
0	1	16	1	21505	0.463
0	1	15	1	22017	0.475
0	1	15	0	22017	0.475
0	1	16	0	21505	0.463
1	1	15	0	22017	0.475
1	1	15	1	22017	0.475
1	1	16	1	21505	0.463
0	1	15	1	22017	0.475
1	1	16	1	21505	0.463
1	1	17	1	20737	0.447
0	1	16	1	21505	0.463
1	1	17	1	20737	0.447
1	1	18	1	18433	0.397
1	1	19	1	14337	0.309
1	1	19	1	14337	0.309
1	1	20	1	13313	0.287
0	1	19	1	14337	0.309
1	1	19	1	14337	0.309
0	1	18	1	18433	0.397
1	1	18	1	18433	0.397
1	1	19	1	14337	0.309
1	1	20	1	13313	0.287
1	1	20	1	13313	0.287
1	1	21	1	12289	0.265

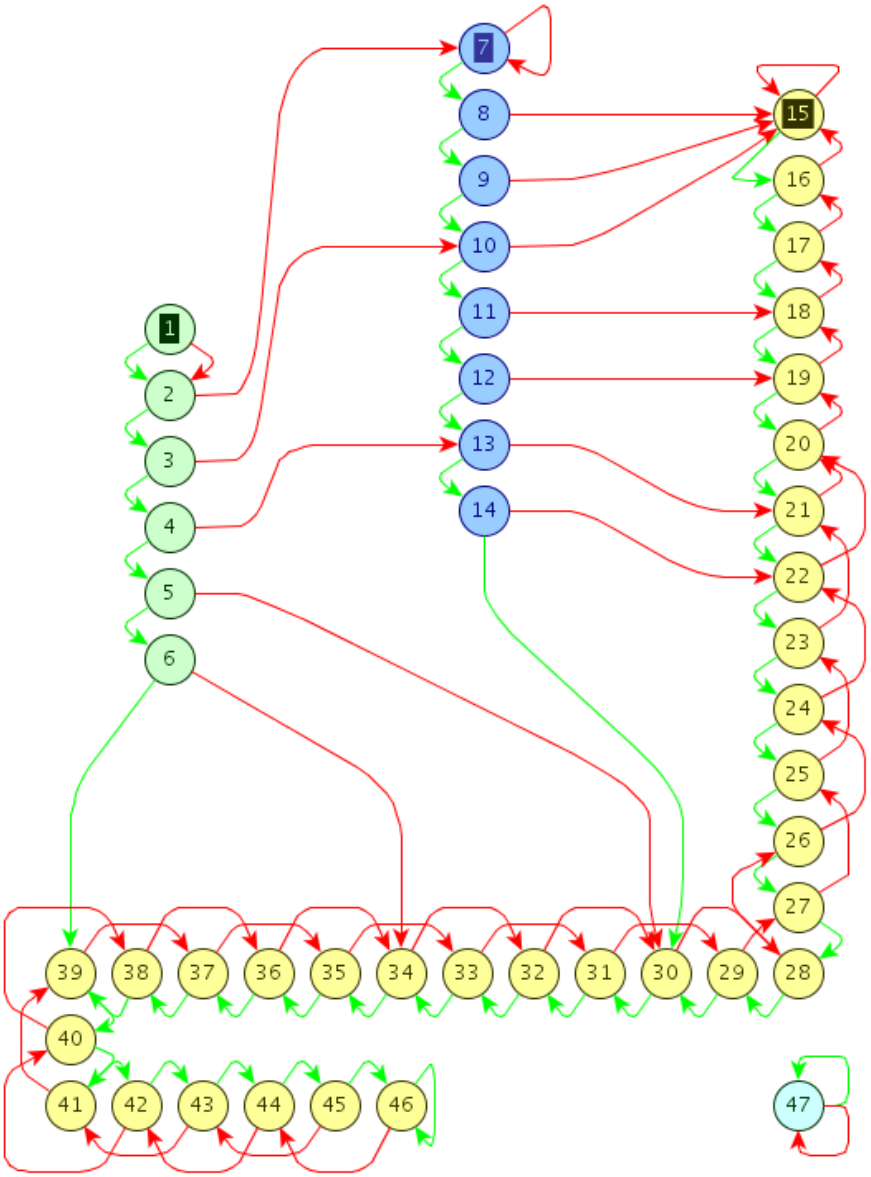

Figure 6.1: MQ coder state diagram.

Transfer-Byte (Alg. 6.3) There are some reserved codes that must be avoided (range from $FF90_h$ to $FFFF_h$). These codes are 8-bit length (1 byte) and it is mandatory in case we are implementing the JPEG2000 Standard, but it has nothing to do with arithmetic coding. Anyway, we implemented the MQ-coder as it was defined in [30] so in case a transfered byte is FF_h, the next should not be in the range from 90_h to FF_h. This routine manages the bit stuffing policy, a few rules to solve that issue.

Put-Byte (Alg. 6.4) This routine simply puts the byte stored in the T register into the byte stream (B). Here we use byte stream because the MQ-coder's output is byte oriented. Conceptually there is not any difference between the bit stream output by the SPECK algorithm and the byte stream output by the MQ-coder. However, the MQ-coder algorithm is designed to use those registers and it forces us to keep in mind that the MQ-coder expects a byte stream, not a bit stream.

Codeword Termination (Alg. 6.5) At the end of the input bit stream, there might be some remaining bits that were not enough to output a byte. This routine shift they out to the byte stream.

Decoding Procedures:

Initialization (Alg. 6.6) This routine initializes the decoder registers using the byte stream.

Fill-LSBs (Alg. 6.7) This routine fills register T with the byte stream. In this way, the decoding routine could get bits from the byte stream.

Decoder (Alg. 6.8) This routine decodes a symbol from registers.

Renormalize-Once (Alg. 6.9) This routine is used to renormalize the arithmetic coding interval. Moreover, in case we decoded all bits in T, there is a call to Fill-LSBs in order to load in T a new byte.

Algorithm 6.1 Procedure **MQ Encoder Initialization**

```
PROCEDURE MQ Encoder Initialization
```

$A = 8000_h$, $C = 0$, $t = 12$, $T = 0$, $L = -1$.

6.5 Performance

To check the performance of this arithmetic entropy coder, we compared our results with those obtained with the software provided in [30] from Kakadu. Kakadu implemented the entire JPEG2000 Standard and it is not easy to extract an executable MQ-coder from the source code provided. So, we implemented the MQ-coder in MATLAB and we must check our software. We need to check two things:

Invertibility Our MQ-decoder implementation must be able to decode the MQ-encoder output's providing the bit stream at the entrance of MQ-encoder.

Coding ability Given a bitstream, the encoder's output length should be as similar as possible than the one we get with the reference software from Kakadu.

To find out if our MQ-CODEC implementation satisfies these two conditions, we modified the reference software to get the symbols and the labels that are sent to the MQ-coder of the software. We used 5 images (bike, lena, goldhill, barbara and shapes) and two wavelet transforms (9/7 tap filter and LeGall 5/3 filter). So we had 10 images to encode. Each image was split in 72 blocks, that is 72 bit streams sent to the MQ-coder per image. Figure 6.2 shows an histogram. The variable plotted in the histogram is the block output byte length percentage relative to the reference software, i.e. it is the histogram of $x = 100 * \frac{(l_t - l_k)}{l_k}$, where l_t is the output block length of the tested software, and l_k is the output block length of the reference software (from Kakadu).

We observe in the figure that our MQ-coder implementation has higher compression efficiency that the reference software. But, this is someway confusing because we implemented the algorithm presented in the book [30]. Actually, our implementation is based on the algorithm of the Chapter 12 called "Sample Data Coding" in Part 2, "The JPEG2000 Standard", but there is Chapter in Part 3 "Working with JPEG2000" called "Implementation Considerations" (chapter number 17) that provides some tricks on implementation. Some of these increase the implementation efficiency, by decreasing a little bit the implementation effectiveness: that is, the algorithm resources usage decreases, but the output length increases. This is the reason why our implementation works better (only) in terms of compression rate.

Algorithm 6.2 Procedure **MQ Encode**

`PROCEDURE MQ Encode`(x, k)
Require: Registers C,A and t, an input symbol x and an input label k
  Set $s = s_k$ and $p = p(\Sigma_k)$.
  $A = A - p$.
  if $A < p$, **then**
    $s = 1 - s$// *conditional exchange of MPS and LPS*
  end if
  if $x = s$ **then**
    $C = C + p$// *assign MPS the upper sub-interval*
  else
    $A = p$// *assign LPS the lower sub-interval*
  end if
  if $A < 2^{15}$ **then**
    if $x = s_k$ **then**
      // *the symbol was a real MPS*
      $\Sigma_k = \Sigma_{mps}(\Sigma_k)$
    else
      // *the symbol was a real LPS*
      $s = s_k \oplus \lambda(\Sigma_k)$// *i.e., switch MPS/LPS if* $\lambda(\Sigma_k) = 1$
      $\Sigma_k = \Sigma_{lps}(\Sigma_k)$
    end if
  end if
  while $A < 2^{15}$ **do**
    //*perform renormalization shift*
    $A = 2A$, $C = 2C$, $t = t - 1$
    if $t = 0$ **then**
      `Transfer-Byte`(T, C, L, t)
    end if
  end while

Algorithm 6.3 Procedure **Transfer-Byte** (encoder)

`PROCEDURE Transfer-Byte`(T, C, L, t)
Require: Registers T, C, L and t
  if $T = \mathrm{FF_h}$ **then**
    //*can't propagate any carry past T; need bit stuff*
    `Put-Byte`(T, L)
    $T = C^{msbs}$, $C^{msbs} = 0$, $t = 7$//*transfer 7 bits plus carry*
  else
    $T = T + C^{carry}$//*propagate any carry bit from C into T*
    $C^{carry} = 0$//*reset the carry bit*
    `Put-Byte`(T, L)
    if $T = \mathrm{FF_h}$ **then**
      //*decoder will see this as a bit stuff; need to act accordingly*
      $T = C^{msbs}$, $C^{msbs} = 0$, $t = 7$//*transfer 7 bits plus carry*
    else
      $T = C^{partial}$, $C^{partial} = 0$, $t = 8$//*transfer full byte*
    end if
  end if

Algorithm 6.4 Procedure **Put-Byte** (encoder)

`PROCEDURE Put-Byte`(T, L)
Require: Registers T and L
  if $L \geq 0$ **then**
    $B_L = T$
  end if
  $L = L + 1$

Algorithm 6.5 Procedure **MQ Codeword Termination** (encoder)

  $n^{bits} = 27 - 15 - t$//*the number of bits we need to flush out of* C
  $C = 2^t\, C$//*move the next 8 available bits into the partial byte*
  while $n^{bits} > 0$ **do**
    `Transfer-Byte`(T, C, L, t)
    $n^{bits} = n^{bits} - 7$//*new value of* t *is the number of bits just transferred*
    $C = 2^t\, C$//*move bits into available position for next transfer*
  end while
  `Transfer-Byte`(T, C, L, t)//*flush the byte buffer* T

Algorithm 6.6 Procedure **MQ Decoder Initialization**

`PROCEDURE MQ Decoder Initialization`
Require: All registers
  $T = 0$, $L = 0$, $C = 0$
  `Fill-LSBs`(C, T, t, L)
  $C = C\, 2^t$//*i.e., left shift* C *by* t *position; we can be sure that* $t = 8$
  `Fill-LSBs`(C, T, t, L)
  $C = C\, 2^7$
  $t = t - 7$
  $A = 8000_{\mathrm{h}}$

Algorithm 6.7 Procedure **Fill-LSBs** (decoder)

`PROCEDURE Fill-LSBs`(C, T, t, L)
Require: Registers C,T,t and L and the byte stream
  $t = 8$
  if $(L = L_{max})$ or $(T = \mathrm{FF_h}$ and $B_L > 8\mathrm{F_h})$ **then**
    $C = C + \mathrm{FF_h}$//*codeword exhausted; fill* C *with 1's from now on*
  else
    if $T = \mathrm{FF_h}$ **then**
      $t = 7$
    end if
    $T = B_L$, $L = L + 1$
    $C = C + T\, 2^{8-t}$
  end if

Algorithm 6.8 Procedure **MQ Decode** (returns x)

`PROCEDURE MQ Decode`(k)
Require: All registers and an input label k
  Set $s = s_k$ and $p = p(\Sigma_k)$
  $A = A - p$
  if $A < p$ **then**
    $s = 1 - s$//*conditional exchange of MPS and LPS*
  end if
  if $C^{active} < p$ **then**
    //*compare active region of* C
    `Output` $x = 1 - s$
    $A = p$
  else
    `Output` $x = s$
    $C^{active} = C^{active} - p$
  end if
  if $A < 2^{15}$ **then**
    if $x = s_k$ **then**
      //*the symbol was a real MPS*
      $\Sigma_k = \Sigma_{mps}(\Sigma_k)$
    else
      //*the symbol was a real LPS*
      $s_k = s_k \oplus \lambda(\Sigma_k)$//*i.e., switch MPS/LPS if* $\lambda(\Sigma_k) = 1$
      $\Sigma_k = \Sigma_{lps}(\Sigma_k)$
    end if
  end if
  while $A < 2^{15}$ **do**
    `Renormalize-Once`(A, C, T, t, L)
  end while

Algorithm 6.9 Procedure **Renormalize-Once** (decoder)

`PROCEDURE Renormalize-Once`(A, C, T, t, L)
if $t = 0$ **then**
  `Fill-LSBs`(C, T, t, L)
end if
$A = 2A$, $C = 2C$, $t = t - 1$

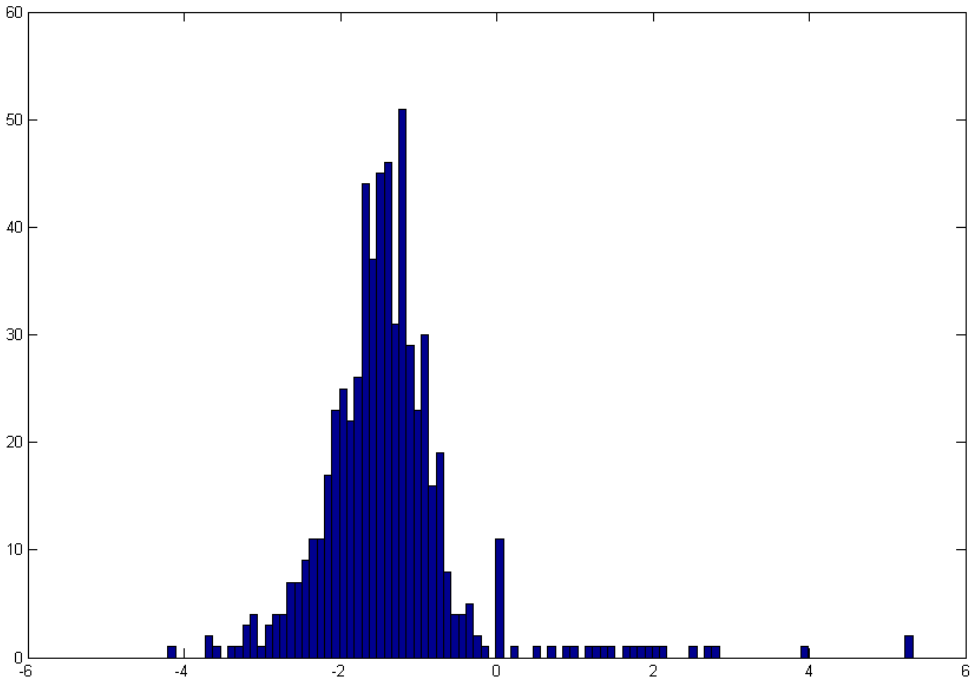

Figure 6.2: Histogram of block output byte length relative to the reference software in %.

Chapter 7

Experiments and results

7.1 Introduction

This Chapter deals with the explanation of the experiments we have done in order to increase the coding efficiency of the combination of GL and SPECK. The body of the Chapter has several sections, one per described experiment. All these sections are structured in the same way: first we describe the experiment, second we present the results and finally we expose our conclusions.

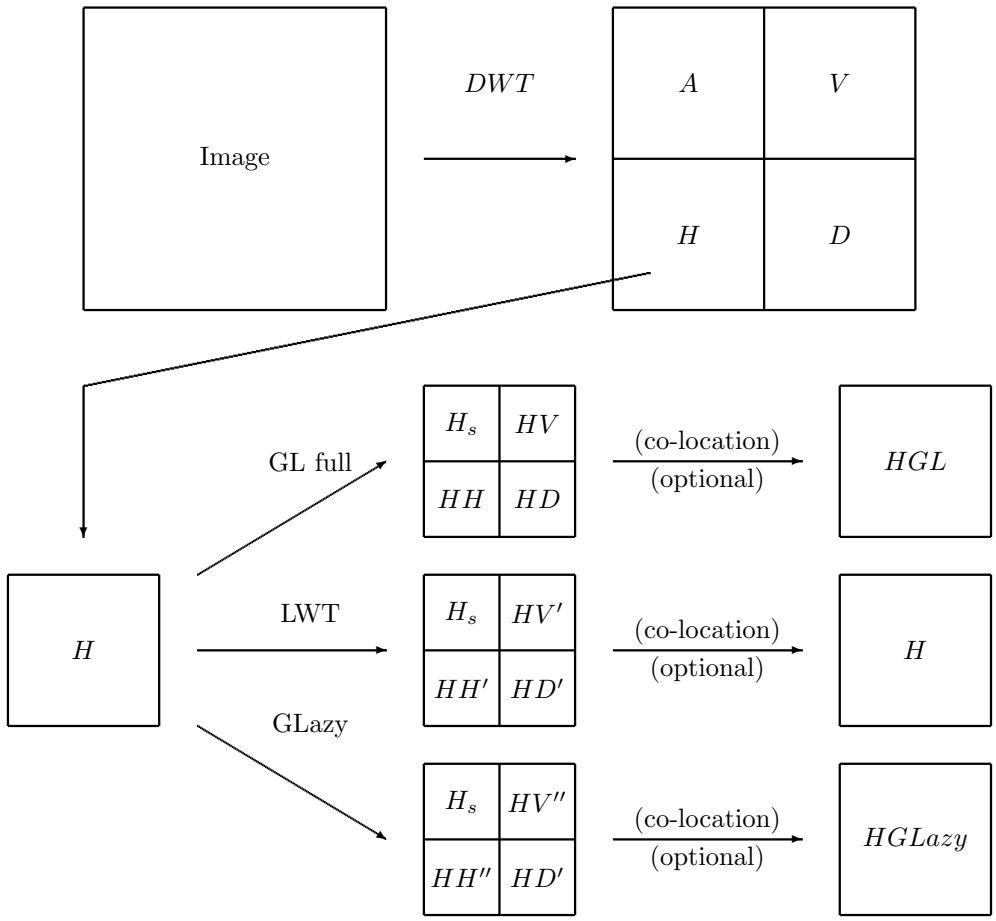

Figure 7.1: Spatial "GLazy" decomposition over the original image.

Figure 7.1 shows the spatial distribution of the transforms we used in our experiments. Recall that all these ("GL full", "LWT" and "GLazy") transforms are detailed in Section 3.5. The "GL full" decomposition is used in Section 7.2. The "LWT" decomposition is used in all experiments. And the "GLazy" decomposition is used in all experiments except in the first one (Section 7.2).
On the other hand we can apply an Inverse LWT to reorder the coefficients. In this way, the transform change the value of the coefficients but it does not change their position. When we apply this ILWT, we say that the coefficients are co-located, otherwise we say that the image has a pyramidal or hierarchical structure.

In the Tables presented in this chapter, we refer always to four techniques. Each technique is defined by choosing the type of the Generalized Lifting Scheme (either "GL full" or "GLazy") and by choosing between co-location or pyramidal structure. The techniques are:

(no suffix) It is the original horizontal DWT subband image, we did not apply the GL transform and there is no hierarchical structure.

GL-co We applied the corresponding GL transform ("GL full" in the first experiment, "GLazy" in the other ones) and we co-located the coefficients.

Lazy We applied a LWT to the original DWT subband.

GL We applied the corresponding GL transform to the DWT subband (then the subband has a hierarchical structure, because we did not co-located the coefficients as in GL-co).

7.2 Taking benefit of zero blocks independently of the SPECK partition

The SPECK algorithm aims to quickly find significant pixels and to encode zero blocks in a cheap way. However, the algorithm splits the sets without taking into account the geometric structure of the image. What about those zero blocks that do not correspond with the SPECK partition? Are we going to take profit of them? Is there any difference between GL and wavelet? To answer these questions, we proposed this experiment.

§ THE EXPERIMENT We manually search three large rectangular blocks of low energy in the image Barbara. To identify this blocks we use the top left corner coordinates, the width, the height and the low bit plane. These coordinates mean that there is no significant pixel with respect to the bit plane specified inside the spatial region defined by the coordinates.
Then, we suppose that we send the coordinates of the three blocks to the decoder, but we do not worry about how we send it. Finally, we run the SPECK algorithm without taking into account these zero blocks we specified to the decoder, and we compute the output length. In other words, we do not send any information about the significance of those sets that are inside the zero block, while scanning a bit plane higher or equal than the low bit plane manually specified.
Before the results, notice that, when the algorithm scans one bit plane below the zero block's limit we have two alternatives:

- inherit the image partition (Inherited Partition, IP), that is the algorithm uses the same partition as the one we would use if we did not specified any zero block, or
- start a new partition inside the region, that is equivalent to run the SPECK algorithm inside the region without taking into account the rest of the image (SPECK in SPECK, SiS).

§ RESULTS These two alternatives are evaluated for 3 regions in image Barbara (see Figure 7.2 for an example). We show the results in Table 7.1. The first column shows if GL is used or it is not, the second column shows if the image is co-located or it is not and the SPECK's output length in bits without specifying any zero block. The third column specifies the top-left corner of the region, and the fourth one, the bottom right corner. The fifth column is the lower bit plane of the region. The sixth and seventh column show the bit difference and the ‰ corresponding to that difference with respect the original length, with the IP technique. The eighth and ninth columns show the same structure, but with the SiS technique.

§ MAIN CONCLUSION Generally speaking we observe that the IP technique has a better performance than the SiS technique, because we saving some bits in all cases using the first technique. However there are no spectacular savings. In fact, the maximum saving is 34 bits in the first row of the table. We also observe that the amount saved bits does not depend very much neither on the technique nor on the region size (from 0.01 to 0.06 bits per zero block pixel).
We must also manage a very important issue: we must send to the decoder the coordinates of these zero blocks. How much do we have to pay for that?, i.e., how many bits do we need to transmit these coordinates? It depends on the way to code this value. However, we are not going to further study these issues, because the

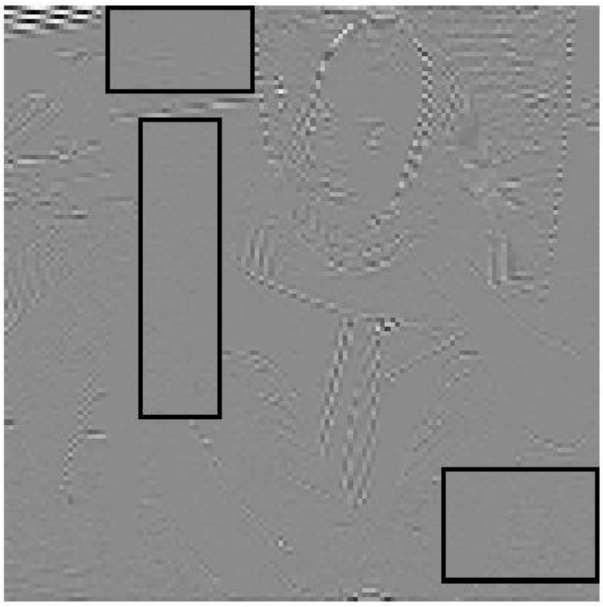

Figure 7.2: Image Barbara with the three regions highlighted.

Table 7.1: Bit budget savings for different regions. Black region technique evaluation.

GL	Co-location (bits)	Region (up:left)	Region (down:right)	(bit-plane)	IP bits	IP ‰	SiS bits	SiS ‰
No	Yes	1:23	19:54	3	34	0.56	-24	-0.4
		25:30	89:47	3	15	0.25	-177	-2.9
	60372	100:95	124:128	3	11	0.18	-248	-4.1
	No	13:16	46:24	3	4	0.06	-113	-1.8
		48:76	74:85	3	11	0.17	-5	-0.07
	64332	77:95	101:128	2	13	0.2	-51	-0.8
Yes	Yes	1:124	19:54	3	22	0.36	-53	-0.86
		25:31	75:50	3	16	0.26	-131	-2.2
	61597	100:95	123:128	3	13	0.2	-195	-3.2
	No	51:48	64:70	3	9	0.14	-102	-1.6
		86:74	105:87	3	10	0.15	-91	-1.4
	64860	14:80	34:89	2	10	0.15	-58	-0.9

ideal gain is not encouraging. The ideal number of saved bits represents a 0.5‰ of the bit stream length. This is a too moderate gain.
The main conclusion is that although the SPECK partition does not depend on the image, it seems to encode very efficiently blocks of zeros even if they do not coincide with the speck partitioning process. Therefore, we must try something different to remove the redundant bits.
For the next experiment, we try to increase the compression ratio by using an arithmetic encoder to further compress the bitstream generated by the SPECK. But some questions arises: which arithmetic coder do we have to use? Should we condition the bits with respect to a context? Is there any statistical difference among the bits? We chose to use two types of arithmetic coder and some different context sets.

7.3 Arithmetic coding after the wavelet-based SPECK algorithm

§ THE EXPERIMENT This experiment consists in using an arithmetic encoder to compress the bitstream generated by the SPECK algorithm. This experiment tries to determine if there is any possibility to rise the system performance by adding a binary-alphabet adaptive arithmetic coder after the SPECK algorithm. Actually, we used two different arithmetic coders:

- the MQ-coder presented in Chapter 6 and used in the JPEG2000 Standard (column labelled MQ in the tables). This coder estimates adaptively the probability density function through a Finite State Machine mechanism.
- an executable adaptive arithmetic coder called `B_ARITH_CODING` from [2]. This arithmetic coder uses a n-th order Markov Model to internally condition the symbol pdf. That is, the algorithm conditions the symbol to the n past samples. n is an integer parameter between 0 and 3, both included. This arithmetic coder estimates accumulatively the pdf: the algorithm accumulates all the knowledge about the signal counting how many times it has observed each symbol with a given set of values for the n past samples.. We called it MM1 and MM2, because we used this Markov-Model based adaptive arithmetic coder with orders 1 and 2 respectively. We do not use the 0-th order and the 3-rd order because preliminary experiments (not reported here) have shown that these orders lead to bad results.

Note that the bitstream generated by the SPECK algorithm involves bits with different meaning. Some of the bits describe the significance of the coefficients, other their signs and other their refinement. It is natural to assume that these different pieces of information may lead to very different statistics. Therefore encoding all the bits generated by the SPECK with a unique pdf seems to be a potential source of inefficiency. To solve this, an initial idea consists in encoding the three types of bits independently. For example, if we use the MQ-encoder, this strategy can be simply implemented by assigning a specific label (with three possible values) to each bit to encode. This is the first experiment we should do if we add an arithmetic coder after the SPECK algorithm.
We used 4 test images called Barbara (I1), Cameraman (I2), Bike (I3) and Lena (I4). Four techniques are also applied, before the entropy coding step, and they are specified in the table as suffixes: Original DWT subband (no suffix), GL and co-location (GL-co), the LWT on the DWT subband (Lazy) and the GL without co-location, in hierarchical structure (GL). From now on, as we said at the beginning of the Chapter, when we refer to GL (or GL-co) we refer to the "GLazy" decomposition presented in section 3.5. See Figure 7.3 for the original images and Figure 7.4.

§ RESULTS The results are presented in Table 7.3 for the co-located techniques and in Table 7.4 for the hierarchical decompositions. The first column corresponds to the name of the image. The second column corresponds to the SPECK's algorithm output. The third column corresponds to the result we would get if we used an ideal entropy coder after the SPECK algorithm. To get this result we computed the entropy $H(X)$ of the bit stream, using the conditioned entropy formula:

$$H(X) = \sum_{l} H(X \mid L = l)\, \mathrm{prob}\{L = l\}. \tag{7.1}$$

Then we computed the ideal entropy coder output bit stream length: $L_i = H(X) \cdot L_{SPECK}$, where L_{SPECK} is the SPECK's output bit stream length. And finally we compute the variation in %:

$$\text{Th.}(\%) = 100\,\frac{L_{SPECK} - L_i}{L_{SPECK}} = 100\,\frac{L_{SPECK} - H(X)\,\cdot\, L_{SPECK}}{L_{SPECK}} = 100\,\left(1 - H(X)\right).$$

Table 7.2: SPECK's output bit stream length for horizontal DWT subbands of Barbara, Cameraman, Bike and Lena.

Image	Bit stream length
Barbara (I1)	60372
Cameraman (I2)	61244
Bike (I3)	82656
Lena (I4)	70085

Table 7.3: Results for the "SPECK + Arithmetic Coder" experiment for co-located techniques.

Image	SPECK	SPECK+AC experiment			
	(%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)
I1		0,33	-2,59	1,29	1,21
I2		0,88	-2,25	1,44	1,38
I3		1,97	-0,16	2,07	1,99
I4		1,38	-2,1	1,5	1,5
I1 GL-co	-1,26	0,35	-3,77	0,19	0,17
I2 GL-co	-0,59	0,92	-2,81	0,99	0,93
I3 GL-co	0,03	2,64	-0,09	2,64	2,52
I4 GL-co	-0,11	1,38	-2,44	1,43	1,41

The next three columns correspond to the scheme SPECK + arithmetic coder: the MQ-coder (MQ) the MM1 and the MM2. All values represent a % referenced to the length of the bitstream originally generated by the SPECK used after the DWT. These reference length values are shown in Table 7.2.

§ Main Conclusion We extract the following conclusions from this first experiment:

- The use of the GL-co plus an arithmetic coder is not recommended in I1 and I2 because it implies a coding loss nearer the 1% with respect to the original image. However there is a gain in I3 (0.7%) and there is no appreciable effect over I4.
- The use of hierarchical decompositions (GL and Lazy) are not recommendable in front of the co-located ones, specially in I1 and I4, because there is a coding loss near 4% with respect to the co-located techniques. In other words, the arithmetic coders tested are not appropriated to compress those signals.
- In all the experiment, the use of the MM2 performed better than the use of the MM1, and much better that our MQ-coder MATLAB implementation.
- Notice that the theoretical results are very encouraging. Theoretically the GL works much better than the Lazy and the GL-co works much better than the original image.
- Finally, notice also that the use of the MM2 provided very good results (between 1% and 2%) in GL-co and in the original image.

Immediately a question arises: is there any clever way to condition any of these pixels in order to increase the compression efficiency? Due to the fact that significance bits represent between a 55% and a 62% of the total bit stream, we decide to condition more precisely the significances bits. To this goal we define several labels, as in the JPEG2000 Standard terminology.

Table 7.4: Results for the "SPECK + Arithmetic Coder" experiment for pyramidal techniques.

Image	SPECK	SPECK+AC experiment			
		Th.	MQ	MM1	MM2
	(%)	(%)	(%)	(%)	(%)
I1 Lazy	-6,54	1,6	-7,63	-3,47	-3,49
I2 Lazy	-5,68	2,03	-6,39	-2,49	-2,57
I3 Lazy	-7,6	4	-5,49	-2,71	-2,86
I4 Lazy	-3,05	1,9	-4,58	-0,58	-0,61
I1 GL	-6,92	1,43	-8,02	-3,86	-3,86
I2 GL	-5,61	1,91	-6,42	-2,55	-2,61
I3 GL	-5,17	4,08	-3,51	-0,55	-0,72
I4 GL	-3,1	1,89	-4,63	-0,62	-0,67

7.4 Splitting the significance label: the JPEG2000, the Bit Plane and the Size label sets

§ THE EXPERIMENT We decided to split the significance label into a group of labels (significance labels). But how can we do it? Here are some intuitive ideas that we exploit in this experiment.

- The first idea is to define labels in the same way as the JPEG2000 Standard does. This Standard does not precisely work with the set significance, but it does with pixel significance. Actually it defines three parameters representing some directional significances (significance functions):

  σ_h Describes the horizontal significance: 0 if any of the horizontal neighbors are not significant, 1 if any of those pixels are significant and 2 if both pixels are significant. To avoid causality problems while testing the set significance, we used the LSP for non-causal blocks and the LSP and LNP for causal-blocks. Notice that the information in the LSP is always causal because the LSP has information about the previous bit planes.

  σ_v Describes the vertical significance: 0 if any of the vertical neighbors are not significant, 1 if any of those pixels are significant and 2 if both pixels are significant.

  σ_d In the same way, σ_d describes the diagonal significance taking values from 0 to 4 depending on the significance of the diagonal neighbors.

  We extended these variables to sets of pixels. But, what does that mean the significance of the left neighbor of the $\mathcal{S}$ set? We defined it using two SPECK lists: the LSP and the LNP, and the notion of virtual neighbor of $\mathcal{S}$. A virtual neighbor of $\mathcal{S}$ is a set of the same size as $\mathcal{S}$. For example, the left virtual neighbor of $\mathcal{S}$ is the set at the left of $\mathcal{S}$ that has the same dimensions. This notion is used to compute if there is any pixel in the intersection between one of this blocks (that are "equivalent" to the pixel neighbors in JPEG2000) and LSP (for non-causal blocks) or LSP and LNP (for causal blocks). The JPEG2000 Standard defines the set of labels shown in table 7.5, from the significance functions.

- The second idea to define a label for each Bit Plane. So, all significance bits in the same bit plane have the same label. In the high bit planes there are a small number of significant coefficients: therefore the SPECK algorithms splits the sets in order to search those significant positions. However, in the low bit planes there are a lot of significant coefficients and the SPECK partition is more refined. The probability distribution of significance bits may change along the bit planes, and we would like to capture this statistical differences.

- The third idea is to define a label for each block size. In this case it is really easy to define a label for each block size, because all blocks are $2^n \times 2^n$ with $n \in \{0, \ldots, M\}$. In other type of images, one should search the adequate way, in terms of compression efficiency, to group blocks by their sizes. We propose this experiment because the number of blocks depend on their size. There are a few big block and a lot of

Table 7.5: JPEG2000 significance labels defined from significance functions σ_h, σ_v and σ_d.

Label	σ_h	σ_v	σ_d
8	2	x	x
7	1	≥ 1	x
6	1	0	≥ 1
5	1	0	0
4	0	2	x
3	0	1	x
2	0	0	≥ 2
1	0	0	1
0	0	0	0

Table 7.6: Bit stream length variation (in % with respect to the bit stream length of the original image) for the JPEG2000, the Bit Plane and the Size conditioning label sets when using a co-located technique.

	SPECK	JPEG2000				Bit Plane				Size			
	(%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)
I1		1,83	-0,66	1,54	1,2	0,47	-2,59	0,58	0,3	0,51	-1,84	1,19	0,95
I2		2,16	-0,33	1,57	1,23	1,02	-2,27	0,72	0,44	1,08	-1,51	1,38	1,27
I3		2,9	1,09	2,35	2,09	2,06	-0,18	1,47	1,24	2,12	0,11	1,91	1,87
I4		2,82	-0,51	1,73	1,47	1,61	-2,09	0,87	0,73	1,6	-1,25	1,58	1,48
I1 GL-co	-1,26	1,95	-1,82	0,5	0,22	0,58	-3,78	-0,51	-0,71	0,55	-3,03	0,14	-0,02
I2 GL-co	-0,59	2,25	-0,92	1,08	0,72	1,11	-2,79	0,27	-0,02	1,13	-1,94	0,95	0,87
I3 GL-co	0,03	3,57	0,93	2,66	2,4	2,82	-0,1	2,06	1,77	2,81	0,15	2,52	2,42
I4 GL-co	-0,11	2,81	-0,58	1,64	1,32	1,62	-2,41	0,79	0,62	1,6	-1,57	1,49	1,38

small blocks. We think that this change in the population number could affect to the statistic distribution of the significance bits.

§ Results The results of this experiment are presented in Table 7.6 for co-located techniques and in Table 7.7 for pyramidal decompositions. As specified, the first column corresponds to the SPECK output, from the second to the fifth columns correspond to JPEG2000-like labels, from the sixth to the ninth columns correspond to Bit Plane conditioning and from the tenth to the thirteenth columns correspond to Size labels. The image order is the same than in the previous experiment.

§ Main Conclusion We present some conclusions about the experiment. They are structured in four blocks: conclusions about JPEG2000 labels, conclusions about Bit Plane labels, conclusions about Size labels and general conclusions about the experiment.

JPEG2000 labels The first conclusion is that the conditioning idea is good because the MM1 and MM2 arithmetic coders have appreciable gains. Despite the fact that theoretically speaking the co-located Generalized Lifting transform should work better in terms of compressions efficiency, the reality is that the compression efficiency decreases when the GL-co is used in front of the original image results (for all images except for I3, Bike). This tendency is specially strong when we use the MQ-coder. This is somehow surprising because JPEG2000 significance labels were designed for the MQ-coder. However, we are not working with pixel significance but with set significance and the signal characteristics may be different from the expected ones.

Table 7.7: Bit stream length variation (in % with respect to the bit stream length of the original image) for the JPEG2000, the Bit Plane and the Size conditioning label sets when using a pyramidal technique.

	SPECK	JPEG2000				Bit Plane				Size			
	(%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)
I1 Lazy	-6,54	2,9	-5,7	-3,53	-3,81	1,87	-7,64	-4,17	-4,34	1,83	-6,65	-3,52	-3,66
I2 Lazy	-5,68	3,21	-4,47	-2,62	-2,92	2,28	-6,39	-3,18	-3,43	2,25	-5,48	-2,59	-2,62
I3 Lazy	-7,6	4,74	-4,21	-2,65	-2,96	4,17	-5,5	-3,24	-3,48	4,3	-5,17	-2,83	-2,93
I4 Lazy	-3,05	3,29	-2,64	-0,52	-0,78	2,36	-4,59	-1,18	-1,35	2,08	-3,77	-0,59	-0,68
I1 GL	-6,92	2,82	-6,17	-3,93	-4,26	1,78	-8,02	-4,57	-4,7	1,65	-7,19	-3,94	-4,06
I2 GL	-5,61	3,17	-4,6	-2,55	-2,91	2,21	-6,43	-3,23	-3,47	2,14	-5,61	-2,66	-2,67
I3 GL	-5,17	4,98	-2,17	-0,52	-0,85	4,3	-3,53	-1,1	-1,37	4,34	-3,09	-0,67	-0,77
I4 GL	-3,1	3,26	-2,71	-0,56	-0,84	2,35	-4,63	-1,21	-1,41	2,06	-3,81	-0,62	-0,75

Moreover the practical use of both Lazy and GL decompositions is between a 5% and a 9% below their theoretical prevision. And, although the theoretical performance is better than co-located techniques, they work worse in practice than the non-pyramidal decompositions.
We also compared the use of the three arithmetic coders. In that case, we recommend the use of the MM1 before the MM2, and the MM2 before the MQ. Notice that the Markov-model coder results have inverted their use recommendation with respect to the last experiment. This is due to the "context dilution" effect. Generally speaking the "context dilution" takes place when the number of contexts increase. All adaptive arithmetic coders estimated somehow the probability density function, but they all require some bits to train this pdf before the pdf really represent the signal statistics. In case the number of contexts increases, the number of bits per context decreases, and it is more difficult to get a reliable pdf estimation.

Bit Plane labels This label set has the same pattern that the one above. The theory recommends the use of the GL-co transform, but the practical results are not encouraging. However, I3 remarks its difference in terms of compression efficiency, because this image prefers the GL-co rather than the original DWT subband.
The pyramidal techniques are also worse-behaviored than the other ones in practice, although the theory says that those that are pyramidal are better than the co-located ones. In that case the order of the arithmetic coders is the same than in the JPEG200 label set.
The GL scheme coding gain is higher than the original image coding gain, in theory and in practice, for those images with a high geometrical content. This encourages us to continue working with the GL technique and searching how to improve its compression efficiency.

Size labels This conditioning way has the same pattern as the previous two (JPEG2000 and Bit Plane). The use of GL-co is better than the original DWT in case the image has structural content (e.g.: contours). Respect to I4, the variation is small. And GL-co technique is really worse when it is used over I1 and I2.
Despite the fact that theoretical predictions point to pyramidal decompositions, the use of these techniques is not recommended due to its bad performance in practical evaluations.
All techniques prefer the MM1 rather than the MM2 rather than the MQ coder.

General conclusions In case we must choose among those three methods, we will choose JPEG2000, because it seems that the significance labels defined in JPEG2000 for pixels are well-behaviored when we extend them to the set significance from the SPECK algorithm. Moreover we will choose Size before Bit Plane as a set of labels.

Table 7.8: Bit stream length variation (in %) for JPEG200 & Bit Plane, JPEG2000 & Size and Bit Plane & Size label sets for co-located techniques.

	SPECK	JPEG2000 & Bit Plane				JPEG2000 & Size				Bit Plane & Size			
	(%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)
I1		1,96	-0,68	-2,87	-3,9	1,92	-0,56	-0,72	-1,32	0,94	-1,89	-0,83	-1,48
I2		2,33	-0,5	-3,04	-4,25	2,29	-0,03	-0,49	-1,13	1,67	-1,51	-0,53	-1,16
I3		3,19	1,11	-1,67	-2,69	3,04	1,24	0,75	0,32	2,25	0,1	0,16	-0,23
I4		3,05	-0,54	-1,99	-2,95	2,98	-0,18	-0,02	-0,5	2,3	-1,19	0,1	-0,32
I1 GL-co	-1,26	2,08	-2,01	-3,85	-4,88	2,05	-1,69	-1,73	-2,3	1,08	-3,09	-1,9	-2,39
I2 GL-co	-0,59	2,43	-1,01	-3,57	-4,7	2,39	-0,78	-1	-1,6	1,77	-1,99	-0,97	-1,56
I3 GL-co	0,03	3,84	0,97	-1,36	-2,47	3,72	1,05	1,06	0,62	3,07	0,15	0,79	0,33
I4 GL-co	-0,11	3,04	-0,62	-2,12	-3,03	2,97	-0,31	-0,11	-0,61	2,3	-1,52	0,01	-0,46

Table 7.9: Bit stream length variation (in %) for JPEG200 & Bit Plane, JPEG2000 & Size and Bit Plane & Size label sets for hierarchical decompositions.

	SPECK	JPEG2000 & Bit Plane				JPEG2000 & Size				Bit Plane & Size			
	(%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)	Th. (%)	MQ (%)	MM1 (%)	MM2 (%)
I1 GL	-6,92	3,06	-6,08	-8,34	-9,48	2,98	-6,05	-6,26	-6,9	2,44	-7,16	-5,73	-6,22
I2 GL	-5,61	3,43	-4,7	-7,07	-8,25	3,36	-4,32	-4,68	-5,21	2,93	-5,64	-4,34	-4,83
I3 GL	-5,17	5,36	-2,17	-4,49	-5,57	5,24	-2,12	-2,09	-2,56	4,65	-3,1	-2,26	-2,71
I4 GL	-3,1	3,63	-2,81	-4,51	-5,51	3,43	-2,7	-2,41	-2,9	3,03	-3,74	-1,9	-2,3
I1 Lazy	-6,54	3,09	-5,8	-8	-9,14	3,07	-5,51	-5,85	-6,42	2,53	-6,69	-5,29	-5,81
I2 Lazy	-5,68	3,45	-4,68	-7,05	-8,25	3,39	-4,17	-4,67	-5,28	3,01	-5,47	-4,29	-4,77
I3 Lazy	-7,6	5,17	-4,23	-6,55	-7,56	5,02	-4,15	-4,2	-4,67	4,54	-5,19	-4,41	-4,84
I4 Lazy	-3,05	3,65	-2,74	-4,43	-5,39	3,46	-2,52	-2,32	-2,78	3,04	-3,76	-1,84	-2,24

Although theoretical gains are greater in the pyramidal techniques than in the GL-co technique and the theoretical gain in the GL-co technique is greater than in the one in the original image, this order is inverted in practical evaluation. The entropy coders tested work fine with the GL-co technique, but they work even better with the original DWT image.
We must remark now that the use of Markov model based arithmetic coder is preferred than the MQ, but the conditioning sample number has been inverted with respect to the first experiment.

7.5 Increasing the number of labels: combination of two label sets

§ The experiment This experiment is very similar to the previous one. In fact, we conditioned the bits with respect to two of the three label sets among JPEG2000, Bit Plane and Size. In other words, we conditioned with respect to three new sets of labels: JPEG2000 & Bit Plane, JPEG2000 & Size and Bit Plane & Size. Notice the the number of labels increased a lot.

§ Results Results are presented in the same structure that in the previous two experiments in Table 7.8 for co-located structures and in Table 7.9 for pyramidal structures.

§ MAIN CONCLUSION First we expose our conclusions on the evaluation of JPEG2000 & Bit Plane labels, second we expose those on the evaluation of JPEG2000 & Sizes, third those on Size & Bit Plane and fourth some conclusions about the benchmark of this three label sets.

JPEG2000 & Bit Plane We do not recommend the practical use of hierarchical decompositions, but we recommend the co-located structures. On the other hand, the GL-co decomposition has only positive results on theoretical calculus, not in the practical use.
The MQ-coder takes much more profit of this label set than the Markov model based ones. This is probably due to the context dilution phenomena, that has a fatal effect over the MM coder results. Recall that, in addition to the label set we impose, the MM arithmetic coder conditions to the n past samples. This increases the number of context.

JPEG2000 & Size The use of the GL-co technique is also recommended in theory, but not in practice. The only image that seems to agree with the GL-co technique is Bike (I3), whose geometrical content is higher than in the other images. The hierarchical decompositions follows exactly the same pattern with respect to the co-located ones than in previous experiments. The arithmetic coder order is the same as in the last label set.

Size & Bit Plane The patterns with respect to the use of GL-co and the hierarchical decompositions take place another time. However, the arithmetic coder order has changed with respect to the last two experiments. We have not deeply analyzed this results, but there might be at least two reasons:

- The context dilution phenomena: the distribution of the bits along the contexts has changed and this may lead to a few overpopulated contexts and several underpopulated ones.
- The signal's stationarity: MQ and MM estimate in a different way the pdf. MQ coder does not take into account all coded symbols and it is able to better estimate some non-stationarity distributions.

Moreover we notice that the coding gain in the image 3 is higher for the GL-co technique than for the original image. This phenomena does not take place in any of the other 3 images. We think that the high geometrical content is what precisely better decorrelates the GL technique.

General conclusions Despite the fact that theoretically the first label set (JPEG2000 & Bit Plane) is the best one, in practice the second label set (JPEG2000 & Size) is the best one (always in terms of coding gain). The hierarchical decompositions are not well behaviored in practice. This means that we do not use the adequate entropy coder for those signals. However the theoretical results are very encouraging, because the coding gain is very high.
The GL-co technique performs much better in theory that in practice. Moreover, this technique is well-behaviored when the image to code has a high geometric content as in I3. Notice that the coding gains in I3 GL-co are higher than the coding gains in I3 (original image). This encourages us to maintain our efforts to fins an entropy coder that takes benefit from this theoretical results.

7.6 Taking into account the block occupancy instead of the block significance

§ THE EXPERIMENT In that case we forget about conditioning with two labels and we recall the experiment with the best results up to now: the JPEG2000 label set. We try to condition the significance pixels to the block occupancy instead of the block significance. In this experiment, each one of the eight virtual neighbor blocks has a block occupancy label: 0, 1 or 2. Given a threshold $0 < T_o < 0.5$ and the significance pixel occupancy in this virtual block $\mathcal{O}(\mathcal{S})$, defined as:

$$\mathcal{O}(\mathcal{S}) = \frac{\#\text{ of significant pixels inside the block}}{\#\text{ of pixels in the block}},$$

the label for the block is

Table 7.10: Bit stream variation (in %) for block occupancy conditioning (BO and SBO) for co-located techniques.

	SPECK	Block Occupancy (BO)				Split Block Occupancy (SBO)			
		Th.	MQ	MM1	MM2	Th.	MQ	MM1	MM2
	(bits)	(bits)	(bits)	(bits)	(bits)	(bits)	(bits)	(bits)	(bits)
	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)
I1		2,46	-0,1	-4,39	-5,61	1,89	-0,3	-0,96	-1,9
I2		2,86	0,61	-3,87	-5,14	2,3	0,19	-0,82	-1,81
I3		4,21	2,75	-1,41	-2,51	4,16	2,91	1,73	1
I4		3,52	0,2	-2,89	-3,98	2,53	-0,44	-0,78	-1,53
I1 GL-co	-1,26	2,45	-1,49	-5,32	-6,59	1,9	-1,53	-2,22	-3,19
I2 GL-co	-0,59	2,94	-0,01	-4,38	-5,64	2,28	-0,49	-1,51	-2,46
I3 GL-co	0,03	4,29	1,97	-1,66	-2,72	4,02	1,73	1,06	0,37
I4 GL-co	-0,1	3,52	0,13	-2,85	-4,01	2,53	-0,64	-0,78	-1,64

0 if $\mathcal{O}(\mathcal{S}) < T_o$,

1 if $T_o \leq \mathcal{O}(\mathcal{S}) \leq 1 - T_o$ and

2 if $1 - T_o < \mathcal{O}(\mathcal{S})$.

This strategy will be called Block Occupancy (BO) in the sequel. We also studied a variant called Split Block Occupancy (SBO) in which we also condition to the fact that the block has more or less than 64 pixels. This is done in order to split sets in small sets and big sets.

§ Results We present the results for the BO and the SBO conditioning techniques in the same way that we have done before. The first column is for the SPECK's output, from the second to the fifth are the BO results and from the sixth to the ninth are the SBO results. The rows are structured as we have done before. Results are shown in Table 7.10 for co-located techniques and in Table 7.11.

§ Main conclusions In this section we discuss the results for the two experiments based on the block occupancy: the Block Occupancy (BO) experiment and the Split Block Occupancy (SBO) experiment.

Block Occupancy (BO) In the results of this experiment we do not observe any important variation in the theoretical prediction for the GL-co with respect to the original image. However, the use of Generalized Lifting is not recommended generally speaking for those images. Although the theoretical calculus says that hierarchical decompositions should have more compression rate that the co-located ones, the practical results really disappoint with that idea. On the other hand, the MQ-coder seems to work better than the MM arithmetic coder.

Split Block Occupancy (SBO) The results show the same pattern that in the previous experiment. In fact, our conclusions are exactly the same:

- Theory does not prioritize any technique between GL-co and the original image.
- Practice do not recommend the use of GL-co.
- Hierarchical decompositions seems to work worse than the other ones.
- The MQ-coder works better than the MM arithmetic coder.

Table 7.11: Bit stream variation (in %) for block occupancy conditioning (BO and SBO) for pyramidal techniques.

	SPECK	Block Occupancy (BO)				Split Block Occupancy (SBO)			
		Th.	MQ	MM1	MM2	Th.	MQ	MM1	MM2
	(bits)	(bits)	(bits)	(bits)	(bits)	(bits)	(bits)	(bits)	(bits)
	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)
I1 Lazy	-6,54	3,23	-5,53	-9,81	-11,1	2,76	-5,74	-6,26	-7,26
I2 Lazy	-5,68	3,9	-3,66	-8,21	-9,5	3,3	-3,96	-5,15	-6,19
I3 Lazy	-7,6	6,15	-2,7	-6,3	-7,37	5,99	-2,73	-3,5	-4,17
I4 Lazy	-3,05	3,94	-2,12	-5	-6,23	3,11	-2,72	-2,92	-3,77
I1 GL	-6,92	3,14	-5,97	-9,96	-11,43	2,7	-6,13	-6,72	-7,71
I2 GL	-5,61	3,85	-3,82	-8,09	-9,4	3,22	-4,29	-5,18	-6,17
I3 GL	-5,17	5,99	-1,27	-4,51	-5,56	5,84	-1,36	-1,68	-2,45
I4 GL	-3,1	3,93	-2,24	-5,08	-6,32	3,08	-2,89	-3,02	-3,87

SBO vs. BO Generally speaking the SBO conditioning technique works better than the BO conditioning technique. Notice that this is very interesting because we say that the impact of the "context dilution" phenomena has not increased when we split the BO labels into two sets (those blocks smaller or equal than 8×8 and the rest). However, it is somehow surprising because we appreciate the reverse effect on the theoretical calculus: the entropy using the BO label set is lower than the entropy using the SBO set.

7.7 JPEG2000 fine conditioning

§ THE EXPERIMENT We refined the JPEG2000 labels, that is, we defined a label for each combination of significant/not significant virtual neighbor block. Moreover, inspired in [9], we decided to condition to block size (1×1, 2×2 and greater) and to the statement "my parent block has become significant in the same bit plane as me". So we have $2^8 \times 3 \times 2$ labels in this label set.

§ RESULTS Results are shown in Table 7.12 for co-located techniques and in Table 7.13 for hierarchical ones, in the same structure that in the previous experiments.

§ MAIN CONCLUSION Although theory does not prioritize any of the co-located techniques, practice does. The Generalized Lifting co-located technique is not recommended in front of the original subband image.
Due to the "context dilution" phenomena, the use of the MM arithmetic coder is not really recommended, in fact, it is recommended not to use this label set with this arithmetic coder. We observe the worst efficiency in all experiments we have previous done.
The next chapter deals with a brief summary on what we have done, some general conclusions and a few future possible research lines.

Table 7.12: Bit stream variation (in %) for JPEG2000 fine labels for co-located techniques.

	SPECK	JPEG2000 (fine)			
		Th.	MQ	MM1	MM2
	(bits)	(bits)	(bits)	(bits)	(bits)
	(%)	(%)	(%)	(%)	(%)
I1		4,94	-0,13	-99,33	-106,59
I2		5,35	0,24	-97,93	-104,85
I3		6,79	2,82	-70,15	-75,86
I4		5,87	-0,14	-88,94	-96,22
I1 GL-co	-1,26	4,96	-1,5	-101,45	-108,94
I2 GL-co	-0,59	5,49	-0,39	-98,5	-105,67
I3 GL-co	0,03	6,93	1,94	-72,95	-79,16
I4 GL-co	-0,11	5,85	-0,31	-89,28	-96,39

Table 7.13: Bit stream variation (in %) for JPEG2000 fine labels for hierarchical techniques.

	SPECK	JPEG2000 (fine)			
		Th.	MQ	MM1	MM2
	(bits)	(bits)	(bits)	(bits)	(bits)
	(%)	(%)	(%)	(%)	(%)
I1 Lazy	-6,54	6,05	-5,33	-109,73	-117,82
I2 Lazy	-5,68	6,59	-3,98	-104,51	-112,2
I3 Lazy	-7,6	8,98	-2,77	-77,6	-84,05
I4 Lazy	-3,05	6,39	-2,49	-91,78	-99,22
I1 GL	-6,92	5,92	-6,1	-111,32	-119,41
I2 GL	-5,61	6,45	-4,24	-104,93	-112,48
I3 GL	-5,17	8,7	-1,55	-78,2	-84,77
I4 GL	-3,1	6,38	-2,54	-91,45	-98,87

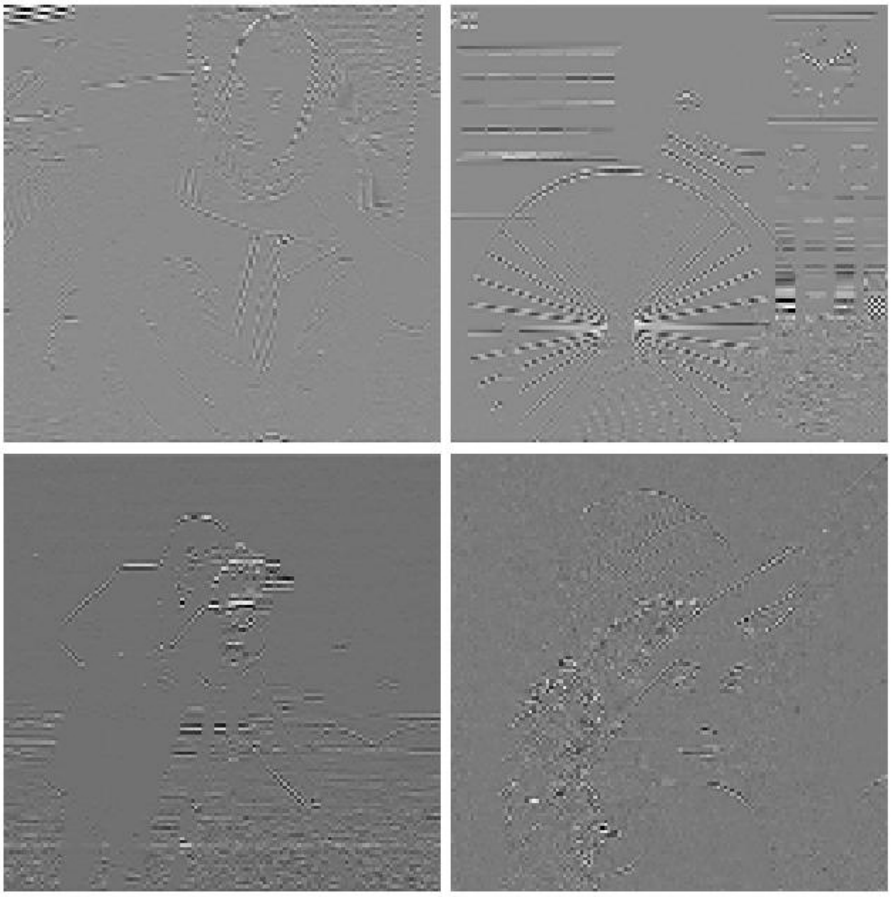

Figure 7.3: Horizontal DWT subbands for Barbara (top-left), Bike (top-right), Cameraman (bottom-left) and Lena (bottom-right).

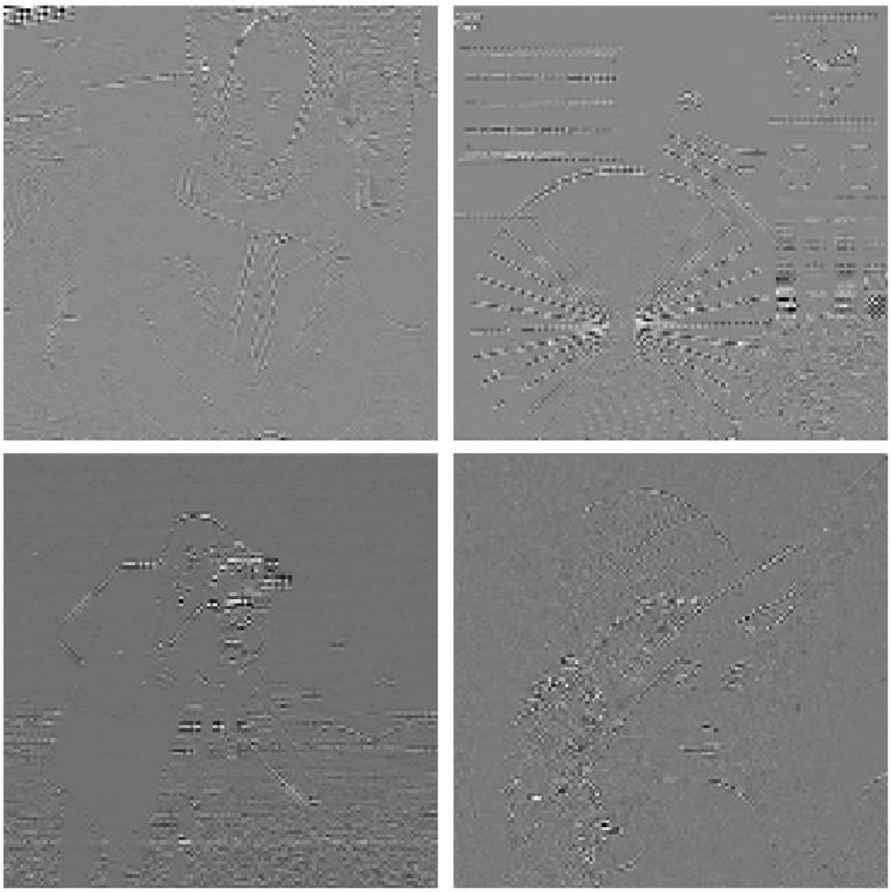

Figure 7.4: Horizontal GL-co subbands for Barbara (top-left), Bike (top-right), Cameraman (bottom-left) and Lena (bottom-right).

Chapter 8

Main conclusions and future research lines

This chapter deals with the summary of the presented report. Recall that the aim of the research work is to find a good entropy coder for the Generalized Lifting transform. We have done several experiments to get some clues to improve the state-of-the-art entropy coders performance when they are applied after a DWT and possibly a GL Mapping.
We wanted to use the SPECK and the MQ-coder algorithms. Due to the lack (at the author's knowledge) of a MATLAB implementation for these algorithms, we implemented both of them. We checked our software with the reference data. In the case of the SPECK, the data was extracted from the original paper. And in the case of the MQ-coder, the data was extracted from the reference software by Kakadu. Once the algorithms' performance is checked, we used them in our experiments.
We made some experiments in order to improve the coding efficiency of the state-of-the-art systems. Some of these experiments provided us very interesting results, reported and deeply analized in chapter 7. We would like to highlight here some interesting and general conclusions that we extracted from the experiments we made.

- The practical use of hierarchical techniques is not recommended, despite the fact that the theoretical results are really better than those of the co-located ones. The theoretical results show that there are still room to try to find alternative entropy coding strategies.
- The practical use of the Generalized Lifting co-located technique is not recommended expect for I3, Bike. Althought we did not find an adequate entropy coder for the GL-co signal for arbitrary images, we think that the behavior that the GL scheme on I3 suggests that the GL is a good technique but mainly when the content involves a strong geometrical content such as contours of high contrast.
- Comparing the practical experiments (MQ, MM1 and MM2), the best results in terms of compression efficiency are provided by the original image + JPEG2000 label conditioning + the MM1 arithmetic coder. Depending on the image, this combination leads to coding gains between 0.5% and 2.66%.
- For all images, the best results using the GL-co technique are obtained with JPEG2000 conditioning labels + the MM1 arithmetic coder.
- In case we search the best MQ-coder results, we find that all co-located techniques for all images prefer the BO label set (except the original bike image, that prefers SBO).

To finalyze this report, we expose here some possible future research lines. They are very interesting and could lead to new clues to find an entropy coder that is able to take benefit of the theoretical results we observed in both, the GL-co and the GL decompositions.

- Find a suitable arithmetic coder for the GL-co signal and for the hierarchical decompositions. Theoretically, they work better than the original subband, but our entropy coders are not able to take benefit of this theoretical advantage. We could also try to define entropy coders based on different strategy, for example the EBCOT strategy.
- Further analysis would be necessary to better understand the differences of adaptation strategy between the MQ and the MM arithmetic encoders. We have seen that depending on the type of conditioning information and on the context dilution problem the encoders provide different performances but we are not able to precisely interpret those differences.
- As the use of the GL mapping has been shown to be of interest for images with a strong structural content, we could try to use the GL mapping only in areas of the image where the structure is strong. This would lead to a scheme where after the DWT, we locally analyze whether or not the mapping has to be applied.

We think this will help to entropy coding, because in the zones where the geometric content is very high, we will use the GL and in the rest of the image we will not do anything. Moreover, we would probably use the best entropy coding strategy found here: SPECK + JPEG2000 + MM.

- Modify the GL transform step in order to get an adequate signal for the high competitive state-of-the-art entropy coders.

Bibliography

[1] Coffman, K. G., and Odlyzko, A. M. The size and growth rate of the internet. *AT&T Labs* (1998).

[2] Comas, D. A finetuning on the adaptive arithmetic entropy coder with accumulative estimation and markov-model conditioning developed in the frame of the momusys european project.

[3] Daubechies, I. Orthonormal bases of compactly supported wavelets. *Comm. Pure and Appl. Math.* (1988).

[4] Daubechies, I., and Sweldens, W. Factoring wavelet transforms into lifting steps. *Journal of Fourier Analysis and Applications 4*, 3 (May 1998), 247–269.

[5] Fourier, J. B. J. *Théorie Analytique de la Chaleur*. Firmin Didot, 1822.

[6] Getreuer, P. Self-contained function for daubechies, symlets, splines, s+p wavelets, jpeg2000 wavelets, and more (`http://www.math.ucla.edu/~getreuer/wavelet.m`).

[7] Gábor, D. Theory of communication. *IEE* (1946).

[8] Haar, A. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen* (1911).

[9] Hsiang, S.-T. Embedded image coding using zeroblokcs of subband/wavelet coefficients and context modeling. In *Data Compression Conference, DCC 2001* (March 27-29, 2001).

[10] Huffman, D. A. A method for the construction of minimum redundancy codes. *Proc. Inst. Radio Eng.* (1952).

[11] Mallat, S. An efficient image representation for multiscale analysis. In *Optical Society of America, Topical Meeting on Machine Vision* (1980), pp. 172–175.

[12] Mallat, S. *A wavelet tour of signal processing*. Academic Press, 1998.

[13] Moffat, A., Neal, R. M., and Witten, I. H. Arithmetic coding revisited. *ACM Trans. on Information Systems 16*, 3 (1998), 256–294.

[14] Pearlman, W., Islam, A., Nagaraj, N., and Said, A. Efficient, low-complexity image coding with a set-partitioning embedded block coder. *IEEE Transactions on Circuits and Systems for Video Technology 14*, 11 (November 2004), 1219–1225.

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

[16] Pennebaker, W. B., Mitchell, J. L., Langdon, G. G., and Arps, R. B. An overview of the basic principles of the q-coder adaptive binary arithmetic coder. *IBM Journal of Research and Development 32* (1988), 717–726.

[17] Rolon, J., Mendonça, E., and Salembier, P. Generalized lifting with adaptive local pdf estimation for image coding.

[18] Rolon, J., Ortega, A., and Salembier, P. Modeling of contours in wavelet domain for generalized lifting image compression. submitted to ICASSP 2009.

[19] Rolon, J., and Salembier, P. Generalized lifting for sparse image representation and coding. In *Picture Coding Symposiu, PCS 2007* (Lisbon, Portugal, Nov 7-9, 2007).

[20] Rolon, J., Salembier, P., and Alameda, X. Image compression with generalized lifting and partial knowledge of the signal pdf. In *International Conference on Image Processing, ICIP'08* (San Diego, USA, Oct 12-15, 2008), IEEE, pp. 129–132.

[21] Rolón, J. C. *Generalized Lifting for Sparse Image Representation and Coding.* PhD thesis, TSC-UPC, Work in progress, to be published in 2010.

[22] Said, A., and Pearlman, W. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology 6*, 3 (June 1996), 243–250.

[23] Schröder, P., and Sweldens, W. Spherical wavelets: Efficiently representing functions on the sphere. *Computer Graphics Proceedings (SIGGRAPH 95)* (1995), 161–172.

[24] Shannon, C. E. A mathematical theory of communication. *The Bell System Technical Journal 27* (1948).

[25] Shapiro, J. Embedded image coding using zerotrees of wavelet coefficients. *Signal Processing, IEEE Transactions on 41*, 12 (Dec 1993), 3445–3462.

[26] Solé, J. *Optimization and Generalization of Lifting Schemes: Application to Lossless Image Compresion.* PhD thesis, TSC-UPC, 2006.

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

[28] Sweldens, W. The lifting scheme: A construction of second generation wavelets. *SIAM J. Math. Anal. 29*, 2 (1997), 511–546.

[29] Taubman, D. High performance scalable image compression with ebcot. *Image Processing, IEEE Transactions on 9*, 7 (Jul 2000), 1158–1170.

[30] Taubman, D. S., and Marcellin, M. W. *JPEG2000 Image Compression Fundamentals, Standards and Practice.* Kluwer Academic Publishers, 2002.

[31] Witten, I. H., Neal, R. M., and Cleary, J. G. Arithmetic coding for data compression. *Comm. ACM 30*, 6 (1987), 520–540.

[32] Ziv, J., and Lempel, A. An universal algorithm for sequential data compression. *IEEE Transactions on Information Theory* (1977).