Democratic and Popular Republic of Algeria
Ministry of Higher Education and Scientific Research

UNIVERSITY MOHAMED KHIDER BISKRA
FACULTY OF SCIENCES AND TECHNOLOGY
DEPARTMENT ELECTRICAL ENGINEERING

Thesis

Presented for the obtaining Magister degree in Science in Electronics

By:

AMRAOUI Merouane

Theme:

Etude de codage par blocs des algorithmes hiérarchiques EZW et SPIHT
Study of embedded coding with blocks of EZW and SPIHT

Dr. Salim SBAA	President	MCA	U. Biskra
Pr. Zine-Eddine BAARIR	Supervisor	Professor	U. Biskra
Dr. Abdelkrim OUAFI	Examiner	MCA	U. Biskra
Dr. Athmane ZITOUNI	Examiner	MCA	U. Biskra
Dr. Abida TOUMI	Examiner	MCA	U. Biskra

Year: 2016/2017

Dedicated to

my mother,

I love you

so much

SPECIAL THANKS

I would like to thank Professor BAARIR for all his support, tolerance, encouragement, open-mindedness and guidance throughout my thesis. Thank you for making everything easier.

I would like to thank Dr. OUAFI and Dr. ZITOUNI for their amazing feedback. Also, I thank the members of the jury for the time taken in order to correct and validate the work in this thesis.

I thank my family of course, Mama, Bouchra, Imad, Asma and for all their support and love. I thank Walid for never shutting up especially when I am programming, thank you for going to Oran so I can finish writing this thesis. I love you all

Harry and Andrea, my family in Austria, who are nothing but great and awesome with me, words cannot describe how much I adore you two.

Abdullah Touil also known as Fnki, there are friends and then there's you. Nothing can be said about you that I didn't tell you before.

Ilyas Birady thank you for being there for me throughout the years, the moments shared, the lame jokes and all in between.

And Weedy, the person that makes life pretty and full of rainbows, just wanna tell you that I landed on my feet.

A special thanks to all my friends Walid Tawba, Youcef Chorfi, Mohammed Sofiane and his family, Brahim Chataibi, Fouzi Mouaouka.

And last but not least, Katharina Fatima Rusch, the truth and nothing but truth, the most amazing person on the northern hemisphere of the planet, the best nicotine friend, to whom I am a proud Osman Pasha, who enlightened and educated my musical taste, best 9gags link sender, the princess of dunes and my companion when I'm 90 years old, I say thank you. (she made me write this)

Table of content

General introduction……..……..……..1
Chapter I: state of the art……..……3
Abstract……4
1.1. Introduction……5
1.2. Description of image compression methods……...5
1.3. Block diagram of image compression……5
1.3.1 Transformation…….…..6
1.3.2 Quantization……...6
1.3.3 Encoding……6
1.4. Classification of compression methods……..7
1.4.1 Lossless methods……7
1.4.1.1 Coding of Shannon-Fano……...7
1.4.1.2 Huffman coding……7
1.4.1.3 Arithmetic coding……..8
1.4.1.4 Lossless JPEG standard……...10
1.4.2 Transform based Methods……11
1.4.3 Predictive methods……12
1.4.4 Hybrid Methods……12
1.5. Lossy methods compression algorithms……12
1.5.1 Standard JPEG……12
1.6. Embedded coding……14
1.6.1 JPEG 2000 Standard
…….14
1.6.1.1 Principle of JPEG2000……..15
1.6.1.2. EBCOT Coding……15
1.6.2 SPECK Algorithm……16
1.6.2.1 Principle……...16
1.6.2.2 Speck algorithm……...17
1.6.3 Algorithm EZBC……..19
1.7. Conclusion……...19

Chapter II: wavelets and multi-resolution analysis……20
Abstract……...21
2.1. Introduction……..……22
2.2. Directional Transforms……22
2.3. Fourier Transform……22
2.4. Short Time Fourier Transform……..24
2.5. Wavelet Transform……...25
2.6. Discrete Wavelet Transform……27
2.6.1 Inversion - Eligibility……..29
2.7. Approximation spaces……30
2.7.1. Scaling Function……...30
2.7.2. Wavelet Function……..31
2.8. Wavelet Transforms in Two Dimensions……32
2.9. Conclusion……...33

Chapter III: EZW and SPIHT algorithms……..……..34
Abstract……...35
3.1. Introduction…….……36
3.2. Embedded coding……36
3.2. Shapiro's EZW……36
3.2.1. Test of significance……..39
3.2.2. Initialization…….39
3.2.3. Quantification and Refinement……40
3.3. SPIHT……..42
3.3.1. Description of the algorithm……43
3.3.2. SPIHT Algorithm…….44
3.4. Parallel computing...……47
3.5. Proposed algorithms B-EZW and B-SPIHT……47
3.6. Examples of EZW/B-EZW and SPIHT/B-SPIHT……...50
3.6.1. EZW Execution……..…….……50
3.6.2. B-EZW……...……..54
3.6.3. SPIHT……...…...56
3.6.4. B-SPIHT……...…...59
3.7. Conclusion……...61

Chapter IV: results and discussion
4.1. Introduction……..……..……..……..……64
4.2. Validation parameters……..……..……..……..……64
4.2.1. Compression ratio……..……..……..……64
4.2.2. Distortion……..……..……..……64
4.2.3. Computation time……..……..……..……65
4.3. Test Images……..……..……..……..……65
4.4. Choice of the wavelet filter……..……..……..……...66
4.4.1. Results on image Lena……..……..……..……..66
4.4.2. Results on the mage Barbara……..……..……..……66
4.4.3: Test on image Goldhill……..……..……..……..67
4.5. Obtained results from the first proposed method……..……..……67
4.5.1. EZW……..……..……..……..……67
4.5.2. SPIHT……..……..……..……..……68
4.6. Results obtained from the proposed algorithms B-EZW and B-SPIHT……68
4.6.1. Comparing B-EZW and EZW……..……..……...69
4.6.1.1: PSNR……..……..……..……...69
4.6.1.2 Computation times……..……..……..69
4.6.2. Comparing B-SPIHT and SPIHT……..……..……..70
4.6.2.1. PSNR……..……..……..……...70
4.6.2.2. Computation time……..……..……..70
4.6.3. psnr plots……..……..……..……71
4.6.3.1. B-EZW……..……..……..……71
4.6.3.2. B-SPIHT……..……..……..……..73
4.6.4. Computation time plots……..……..……..……74
4.6.4.1. B-EZW ……..……..……..……74
4.6.4.2. B-SPIHT……..……..……..……...76

4.6.5. Comparing the results with SPECK algorithm……..…….77
4.6.5.1. Results obtained from SPECK……..……..77
4.6.5.2. Comparing Speck with B-EZW and B-SPIHT……78
4.6.5.2.1. PSNR……..……..……78
4.6.5.2.2. Computation time……..……...79
4.6.6. visual quality……..……..……..……..81
4.6.6.1. B-EZW……..……..……..……...81
4.6.6.2. B-SPIHT……..……..……..……82
4.6.6.3. SPECK……..……..……..……...83
4.7. Discussion of the results……..……..……..……83
4.8. Conclusion……..……..……..……..84

General Conclusion……86
References……..87
Annex……89

List of tables

Chapter III:

Table 3.1: processing of the first dominant pass, threshold 32……52
Table 3.2: processing of the first subordinate pass……..……...53
Table 3.3: processing of the first dominant pass for the first matrix, threshold 32……..…..……56
Table 3.4: processing of the first dominant pass for the second matrix, threshold 32……..…..…….57
Table 3.5: processing of the first dominant pass for the third matrix, threshold 32……....……57
Table 3.6: processing of the first dominant pass for the fourth matrix, threshold 32……...……57
Table 3.7: processing of SPIHT first pass……58
Table 3.8: First pass of B-SPIHT of the first matrix……60
Table 3.9: First pass of B-SPIHT of the second submatrix……..……61
Table 3.10: First pass of B-SPIHT of the third submatrix……61
Table 3.11: First pass of B-SPIHT of the fourth submatrix……..……62

Chapter IV:

Table4.1: PSNR of the first method with EZW on image Lena……70
Table4.2: PSNR of the first method with EZW on image Barbara……70
Table4.3: PSNR of the first method with EZW on image Goldhill……70
Table4.4: PSNR of the first method with SPIHT on image Lena……70
Table4.5: PSNR of the first method with SPIHT on image Barbara……71
Table4.6: PSNR of the first method with SPIHT on image Goldhill……71
Table4.7: PSNR of B-EZW(64, 16 and 4) and EZW on image Lena……..71
Table4.8: PSNR of B-EZW(64, 16 and 4) and EZW on image Barbara……72
Table4.9: PSNR of B-EZW(64, 16 and 4) and EZW on image Goldhill……73
Table4.10: Computation times of B-EZW(64, 16 and 4) and EZW on image Lena……73
Table4.11: Computation times of B-EZW(64, 16 and 4) and EZW on image Barbara……74
Table4.12: Computation times of B-EZW (64, 16 and 4) and EZW on image Goldhill……75
Table4.13: PSNR of B-SPIHT (64, 16 and 4) and SPIHT on image Lena ……76
Table4.14: PSNR of B-SPIHT (64, 16 and 4) and SPIHT on image Barbara……76
Table4.15: PSNR of B-SPIHT (64, 16 and 4) and SPIHT on image Goldhill……77
Table4.16: Computation times of B-SPIHT (64, 16 and 4) and SPIHT on image Lena……78
Table4.17: Computation times of B-SPIHT (64, 16 and 4) and SPIHT on image Barbara ……...78
Table4.19: PSNR and Computation time for different rates for SPECK on image Lena……80
Table4.21: PSNR and Computation time for different rates for SPECK on image Goldhill……82

Table4.22: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Lena……83
Table 4.23: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Barbara….84
Table 4.24: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Goldhill…85
Table 4.25: time comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Lena……86
Table 4.26: time comparison of B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Barbara…….87
Table 4.27: PSNR comparison of B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Barbara……87
Table 4.28: time comparison of B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Goldhill …….88
Table 4.29: PSNR comparison of B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Goldhill……..88
Table 4.30: the influence of parallel computing for B-EZW(4, 16 and 64) on image Lena…….91
Table 4.31: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Lena……..94
Table 4.32: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Barbara……94
Table 4.33: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Goldhill……94
Table 4.34: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Lena……..96

List of figures

Chapter I:

Figure 1.1: block diagram of image compression coding/decoding……3

Figure1.2: example of huffman coding……5

Figure 1.4: Principle of lossless JPEG algorithm. ……...8

Figure 1.5: block diagram of the JPEG algorithm. ……10

Figure 1.6: Example of a three level wavelet transform of image Lena……11

Figure 1.7: Results image Lena rebuilt to 0.2 bpp, a) JPEG standard with PSNR =23.4 dB; b) JPEG2000 standard, with PSNR=28.7 db……...……..12

Figure 1.8: Principle of the EBCOT coding……...13

Figure 1.9: Partitioning of image X into sets S and Γ……14

Figure.1.10. Partitioning of set S……14

Chapter II:

Figure 2.1: a) Example of a stationary signal and b) its Fourier transform……23

Figure 2.2: a) Example of a non-stationary signal and b) its Fourier transform……24

Figure 2.3 – (a) Daubechies-5 and (b) Meyer wavelets. ……27

Figure 2.4: 1-D wavelet signal decomposition……...29

Figure 2.5: The spatial relation of scaling and wavelet function spaces……31

Figure 2.6. A two-level decomposition of the two-dimensional FWT……...33

Figure2.7. The synthesis filter bank of the two-dimensional FWT……..……...33

Chapter III:

Figure 3.1: Example of decomposition of the image logo University Biskra using three resolutions wavelet

……3

7

Figure 3.2: Parent/offspring Relationship……...37

Figure 3.3: scanning order of the subbands for encoding significance map……39

Figure 3.4: Principe de test de significance des coefficients pour l'EZW……40

Figure 3.5: Principle of quantification and refinement……41

Figure 3.6: EZW algorithm flow chart……42

Figure 3.7: Binary representation of the magnitude-ordered coefficients……43

Figure 3.8. visualization of the refinement pass steps……46

Figure 3.9. visualization of the sorting pass steps……46

Figure 3.10: the principle of workers in parallel computing……47

Figure 3.11: (a) EZW/SPIHT structure. (b) the structure of the first method……48
Figure 3.12: B-EZW /B-SPIHT structure……49
Figure 3.13: example matrix……50
Figure 3.14. Matrix reproduced by the decoder after the first passes……52
Figure 3.15: matrix to be processed by the second pass, threshold 16……53
Figure 3.16: matrix reproduced by the decoder after the second passes……54
figure 3.17: dividing and the wavelet transform of the example matrix……55

Chapter IV:

Figure 4.1: Test images, from left Lena, to right Barbara……67
Figure 4.2: Test images, from left Goldhill, to right Man……67
Figure 4.3: the influence of the filters on image Lena (rate vs PSNR) ……68
Figure 4.4: the influence of the filters on image Barbara (rate vs PSNR) ……68
Figure 4.5: the influence of the filters on image Goldhill (rate vs PSNR) ……69
Figure 4.6: PSNR vs rate for B-EZW and EZW for image Lena……72
Figure 4.7: PSNR vs rate for B-EZW and EZW for image Barbara……72
Figure 4.8: PSNR vs rate for B-EZW and EZW for image Goldhill……73
Figure 4.9: Computation times of B-EZW(64, 16 and 4) and EZW on image Lena ……74
Figure 4.10: Computation times of B-EZW(64, 16 and 4) and EZW on image Barbara……74
Figure 4.11: Computation times of B-EZW(64, 16 and 4) and EZW on image Goldhill……75
Figure 4.12: PSNR vs rate for B-SPIHT and SPIHT for image Lena……76
Figure 4.13: PSNR vs rate for B-SPIHT and SPIHT for image Barbara……77
Figure 4.14: PSNR vs rate for B-SPIHT and SPIHT for image Goldhill……77
Figure 4.15: Computation times of B-SPIHT(64, 16 and 4) and EZW on image Lena……78
Figure 4.16: Computation times of B-SPIHT(64, 16 and 4) and EZW on image Barbara……..79
Figure 4.17: Computation times of B-SPIHT(64, 16 and 4) and EZW on image Goldhill……79
Figure 4.18: PSNR for different rates for SPECK on image Lena……80
Figure 4.19: computation time for different rates for SPECK on image Lena……81
Figure 4.20: PSNR for different rates for SPECK on image Barbara……81
Figure 4.21: computation time for different rates for SPECK on image Barbara……82
Figure 4.22: PSNR for different rates for SPECK on image Goldhill……82
Figure 4.23: computation time for different rates for SPECK on image Goldhill……..83
Figure 4.24: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Lena …..84
Figure 4.25: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Barbara…85
Figure 4.26: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Lena……86
Figure 4.27: PSNR comparison of B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Lena……..87

figure 4.30: visual quality for B-EZW4 a) 0.1820 bpp, threshold 64, 28.91 dB. b) 0.4030 bpp, threshold 32, 30.99dB. c) 1.5292 bpp, threshold 8, 32.11 dB……89
figure 4.31: visual quality for B-EZW16 a) 0.2127 bpp, threshold 64, 28.41 dB. b) 0.4524 bpp, threshold 32, 30.26 dB. c) 1.6207 bpp, threshold 8, 31.01 dB……89
figure 4.32: visual quality for B-EZW64 a) 0.2833 bpp, threshold 64, 27.81 dB. b) 0.5590 bpp, threshold 32, 29.47 dB. c) 1.8211 bpp, threshold 8, 30.15 dB……89
figure 4.33: visual quality for B-SPIHT4 a) 0.1 bpp, 27.67 dB. b) 0.5 bpp, 36.03 dB. c) 1 bpp, 39.77 dB……90
figure 4.34: visual quality for B-SPIHT16 a) 0.1 bpp, 21.56 dB. b) 0.5 bpp, 26.76 dB. c) 1 bpp, 31.2 dB……90
Figure 4.35: visual quality for B-SPIHT64 a) 0.1 bpp, 17.78dB. b) 0.5 bpp, 23.11 dB. c) 1 bpp, 26.77 dB……90
figure 4.36: visual quality for SPECK a) 0.1 bpp, 29.57 dB, b) 0.42 bpp, 36.09 dB, c) 1.72bpp, 43.68 dB……91
Figure 4.37: the influence of parallel computing for B-EZW(4, 16 and 64) on image Lena……..92
Figure 4.38: the influence of parallel computing for B-EZW(4, 16 and 64) on image Barbara……93
Figure 4.39: the influence of parallel computing for B-EZW(4, 16 and 64) on image Goldhill…….93
Figure 4.40: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Lena……95
Figure 4.41: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Barbara……95
Figure 4.42: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Goldhill ……96
Figure 4.43: PSNR of SPIHT and B-SPIHT for different sizes of image man at 1bpp……97
Figure 4.43: PSNR of SPIHT and B-SPIHT for different sizes of image man at 1bpp……97

General Introduction

General introduction

Nowadays, information is transmitted on large scales and the sizes of files increase by day. Image files now can get up to 25MB using a medium quality common use camera. Therefor compression is needed in order to reduce the quantity of information being sent or the disk space needed for the storage.

Many compression methods were developed in order to tackle this problem. They can be divided into two categories; the first one aims for a perfect reconstruction of the original data, designated for medical and sensitive applications; but their inconvenient is that they have a low compression rate. The second category is called lossy, where information is lost in small scales but the compression rate is higher and the visual quality is good.

Many compression algorithms have been proposed and standardized as JPEG standard for still images and MPEG standard for video images. All of these standards are based on the Discrete Cosine Transform (DCT). Other compression algorithms using wavelets have been proposed also. One of the most popular compression application in images is JPEG 2000 which generally gives better results than those obtained by the JPEG standard.

Other compression algorithms called embedded wavelet coding have been proposed. These encoders are based on the concept of zero tree (zerotree) proposed by Shapiro, using such concept allows the coders to become progressive coders where different qualities of the compressed image is achieved at different thresholds and the compression rate are much better comparing the a non-progressive coders. Successive coders like SPIHT (Set-Partitioning in Hierarchical Trees) and SPECK (Set-Partitioning Embedded Block) have improved the principle of EZW by providing a more efficient encoding of significant information that encodes the position of the energy coefficients. As this information is a big part of the image flow. The same principle is also used in the JPEG 2000 encoder of EBCOT, providing superior performance.

Our work in this thesis revolves around the study of these embedded encodings (EZW, SPIHT and SPECK) and a proposed optimization algorithms EZW and SPIHT named B-EZW (Block- EZW and Block-SPIHT). Our method uses the original algorithms with a property that allows the exploitation of parallel computation to speed up the algorithms. The results obtained by this approach, in terms of PSNR and compression ratio, are acceptable especially for medium

and high rates and very good for the computing time. Finally, our results are comparable to those obtained by the algorithms EZW, SPIHT and SPECK.

Our thesis is organized into the following chapters:

The first chapter is dedicated to a state of the art of image compression, including lossless compression. A reminder about the nested encodings is also proposed.

The second chapter begins with a detailed presentation on wavelets and time-frequency analysis.

The third chapter is dedicated to algorithms Shapiro's EZW and Pearlman's SPIHT.

The fourth chapter will present the results obtained on different test images. An analysis of the results and a comparative study of these results is given.

We conclude this thesis with a general conclusion and outlook assumptions about our work.

Chapter I : State of the art of compression methods

Abstract

We limit ourselves in this chapter to the presentation of the main lossless and lossy compression methods that exist in the literature. We begin by describing the main features of an image encoder. Then a brief recall on the lossless compression methods is underlined. Much of this chapter will be devoted to lossy methods of compression, then we move to those based on wavelet transform, specifically embedded encodings.

1. Introduction

For a proper use of digital images, one must compress the files in where they are registered. The image consumes a tremendous amount of bytes when it is scanned. Today, we speak of "megapixel quality" for digital cameras; this means that each image has about several million pixels, each pixel requires three bytes for RGB (red, green, blue) components. Without compression, this would represent a little more than 9 to 10 MB for a single photograph from a 10 megapixels camera.

To overcome these constraints, there is only one solution: compress the images. Researchers have devised many methods of compression, which can be classified into two categories: those that simply compress the data without altering them, and those that compact and modify the data. The first, called lossless, possible to reconstruct the file in the state it was before the compression. The second are lossy methods, they irreparably alter the original data. And they are almost always used to compress images (still or video).

One way to reduce the overall volume of images while maintaining the original image consists of compressing images with minimal degradation and the maximum possible efficiency.

2. Image compression methods

They are the methods for compressing and encoding the reduced number of bits per pixel to be stored or transmitted, by exploiting the redundancy in the informational image [1]. The main evaluation of any compression method criteria are:

- The quality of reconstructed image.
- The compression ratio.
- The rapidity of the encoder and decoder.

3. Block diagram of image compression

The block diagram of the compression is shown in Figure 1.1 below, from this diagram, we will review each step to clarify their role:

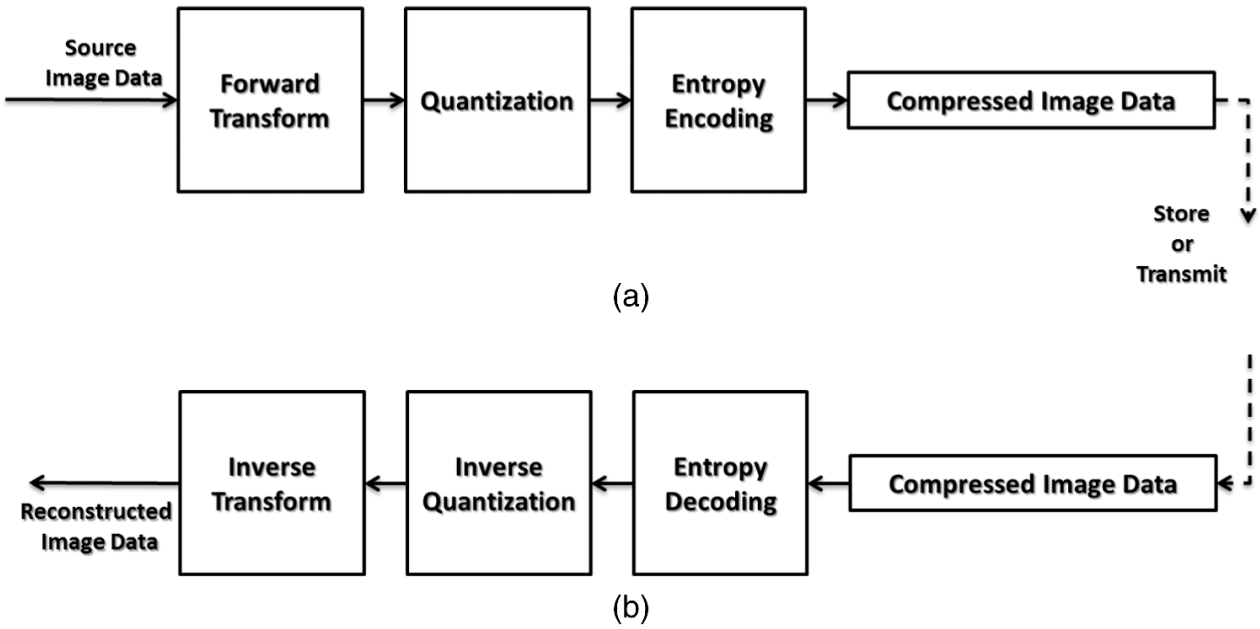


Figure 1.1: block diagram of image compression coding/decoding

3.1 Transformation

The existing dependency between each pixel and its neighbors (brightness varies little from one pixel to an adjacent pixel) reflects a very strong correlation in the image. So we try to take some of this correlation to reduce the amount of information by performing a de-correlation of the pixels.

The de-correlation is to transform the original pixels into a set of a lower correlated coefficients.

3.2 Quantization

The aim of quantizing the coefficients is to reduce the number of bits needed for their representation. It is a key step of compression. It approximates each value of a pixel by an integer multiple of a quantity q, called elementary quantum or quantization step. It may be scalar or vector. The best performance is obtained by using vector quantization.

3.3 Encoding

Once the coefficients are quantized, they are coded. An encoder must satisfy a priori the following two conditions:

- Uniqueness: two different messages should not be coded the same way.

- Decipherability: two successive words codes must be distinct and with no unambiguously.

4. Classification of compression methods

Most compression methods are intended to remove the redundancy in the image, as to reduce the number of bits needed for the representation [1]. Several types of redundancy in terms of correlation may be considered:

- The spatial redundancy between neighboring pixels or blocks in the image.
- Temporal redundancy between successive frames in a video sequence.

The compression methods can be grouped into two classes:

- The methods without loss of information (without distortion or reversible).
- The methods with loss of information (with distortion or irreversible).

Our experiments show that generally the methods that achieve very high compression ratios are the methods with distortion. The con of the methods without distortion is that they generate very low compression ratio and are only used in sensitive applications such as medical images, and satellite images.

4.1 Lossless methods

They allow recovering exactly the original pixels of the digital image. These types of methods belong to the methods without loss of information; they divide into two categories: Entropy coding methods and transformed based methods.

4.1.1 Entropy coding methods

4.1.1.1 Coding of Shannon-Fano

Shannon and Fano have developed a method of encoding based on mere knowledge of the probability of occurrence of each symbol in the message [2].

The process of Shannon-Fano builds a descending tree from the root, by successive divisions. The classification of frequencies is in descending order, which requires a first reading of the file and saving the header.

4.1.1.2 Huffman coding

Huffman coding [3] creates variable length codes in an integer number of bits. The algorithm considers each message to be encoded as a leaf of a tree that remains to be built. The

idea is to assign the two lowest probabilities of an alphabet larger words. These two code words differ only in their last bit. Unlike the Shannon-Fano coding leading from the root and leaves of the tree by successive mergers, back to the root. The principle is as follows:

- Reform the frequencies of letters.
- Sort symbols in a decreasing order sequences of occurrence or decreasing probabilities.
- Group symbol in pairs with lower probabilities, and reclassifying them if necessary.
- Continue the addition of the probabilities until the result equals 1.
- Encode with backwards manner from the last group, adding a 0 or a 1 for different symbols grouped previously.

Huffman coding example:

We have a source with an alphabet of 10 (“a” to “j”) symbols with the probabilities below:

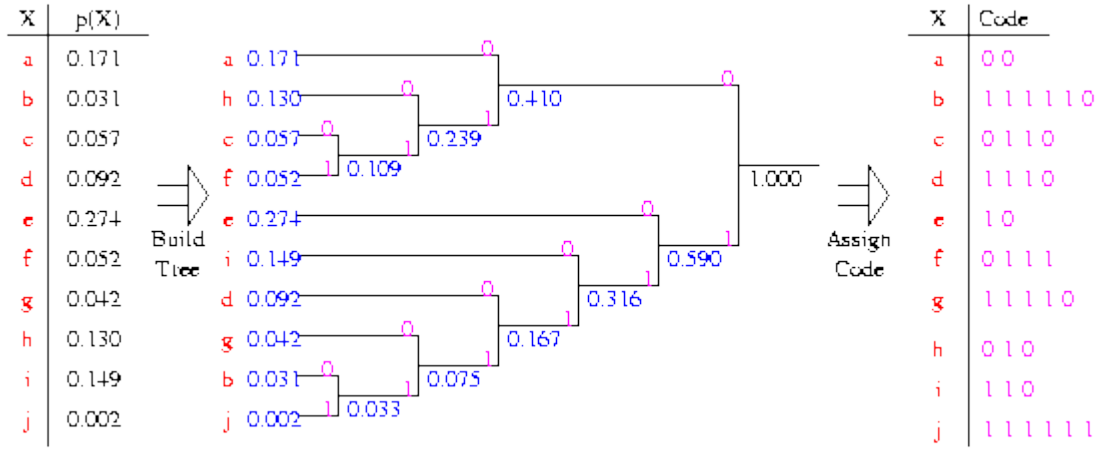


Figure1.2: example of huffman coding

4.1.1.3 Arithmetic coding

Arithmetic coding [4] is a data compression technique that encodes data by creating a code which represents a fraction in the unit interval [0, 1]. The algorithm is recursive. On each recursion, the algorithm successively partitions subintervals of the unit interval [0, 1]. This differs from other forms of entropy encoding such as Huffman coding, which separates the input into component symbols and replacing each with a code.

Arithmetic coding is a method of encoding without this inefficiency. In arithmetic coding, instead of using a sequence of bits to represent a symbol, we represent it by a subinterval

of the unit interval$[0, 1]$. In other words, we encode the data into a number in the unit interval $[0, 1]$, and this technique can be implemented by separating the unit interval into several segments according to the number of distinct symbols. The length of each segment is proportional to the probability of each symbol, and then the output data is located in the corresponding segment according to the input symbol.

This provides a way of assigning codewords to particular sequences without having to generate codewords for all sequences and alleviates the inefficiency and the complexity. Moreover, the code for a sequence of symbols is an interval whose length decreases as we add more symbols to the sequence. This property allows us to have a coding scheme that is incremental, that is, the code for an extension to a sequence can be calculated simply from the code for the original sequence.

Example

Consider a ternary alphabet $\mathcal{A} = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, and $P(a_3) = 0.2$. we have $F_x(1) = 0.7$, $F_x(2) = 0.8$, and $F_x(3) = 1.0$. Now let us consider an input sequence (a_1, a_2, a_3). Then this partitions the unit interval as shown in Figure 1.3.

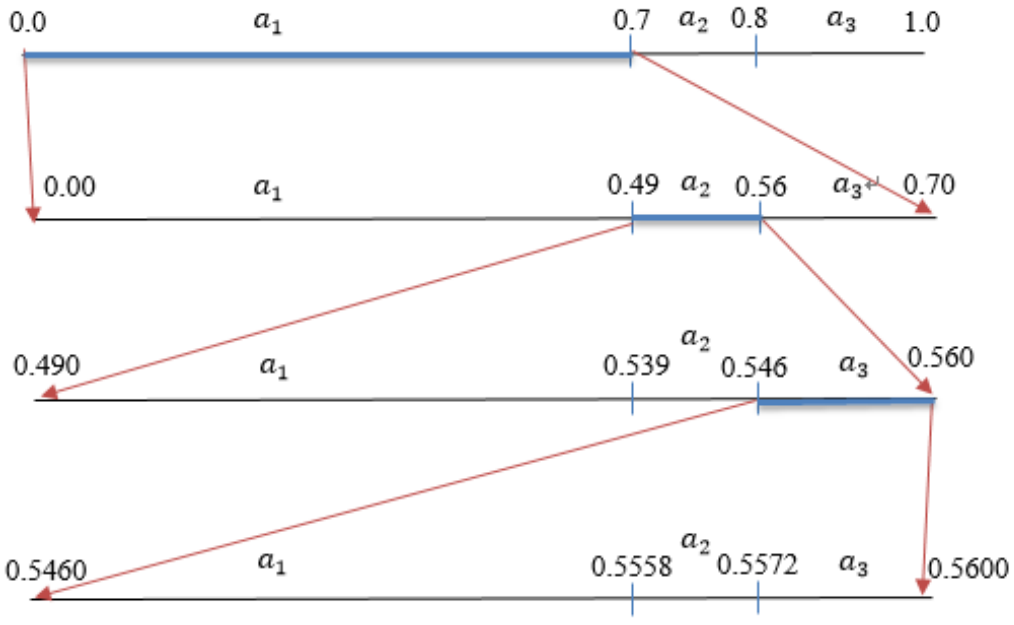


Figure 1.3: Restricting the interval containing the tag for the input sequence (a_1, a_2, a_3)

The partition in which the tag resides depends on the first symbol of the sequence. If the first symbol is a_1, then the tag lies in the interval [0.0, 0.7); if the first symbol is a_2, then the tag lies in the interval [0.7,0.8); if the first symbol is a_3, then the tag lies in the interval [0.8, 1.0). Once the interval containing the tag has been determined, the rest of the unit interval is discarded, and this restricted interval is again divided in the same proportions as the original interval.

Now the input sequence is (a_1, a_2, a_3). The first symbol is a_1, and the tag would be contained in the subinterval [0.0, 0.7). This subinterval is then subdivided in exactly the same proportions as the original interval, yielding the subintervals [0.00, 0.49), [0.49, 0.56), and [0.56, 0.70). The first partition [0.00, 0.49) corresponds to the symbol a_1, the second partition [0.49, 0.56) corresponds to the symbol a_2, and the third partition [0.56,0.70) corresponds to the symbol a_3. The second symbol in the sequence is a_2. The tag value is then restricted to lie in the interval [0.49, 0.56). We now partition this interval in the same proportion as the original interval in order to obtain the subinterval [0.49, 0.539) corresponding to the symbol a_1 , the subinterval [0.539,0.546) corresponding to the symbol a_2 and the subinterval [0.546,0.560) corresponding to the symbol a_3. If the third symbol is a_3, then the tag will be restricted to the interval [0.546,0.560), which can be subdivided further by following the procedure described above.

Note that the appearance of each new symbol restricts the tag to a subinterval that is disjoint from any other subinterval that may have been generated using this process. For the sequence (a_1, a_2, a_3), since the third symbol is a_3, the tag is restricted to the subinterval [0.546, 0.560). If the third symbol is a_1 instead of a_3, the tag would have resided in the subinterval [0.49, 0.539), which is disjoint from the subinterval [0.546,0.560).

As described in the above example, the interval in which the tag for a particular sequence resides is disjoint from all intervals in which the tag for any other sequence may reside. Hence, any number in this interval can be used as a tag since the intervals for distinct sequences are disjoint, and one possible choice is the midpoint of the interval. In the following discussion, we will use the midpoint of the interval as the tag.

4.1.2 Lossless JPEG standard

JPEG [5] (Joint Photographic Experts Group) standard is designed by ISO (International Standards Organization) and the IEC group (International Electronic Commission). It is intended for compressing still images in color and grayscale for storage on digital media. It was conducted with a view to cover the most diverse applications, taking into account realistic constraints compared to the more visible applications, transmission, image banks. Techniques that define the JPEG standard are divided into two classes:

- Lossy compression, which is based on the DCT, followed by quantization and entropy encoder (see section 5.1).

- The second class, the process relates to lossless coding, this class of coders is not based on the DCT but on DPCM coding followed by entropy coding (see figure 1.4). For the lossy

methods, four coders were specified: a basic encoding where the compressed and decompressed image is no longer identical to the original image, this process uses the DCT and Huffman coding. The other three types of encoding are a basic coding extension. They differ mainly in basic coding by entropy coding using arithmetic coding or progressive image reproduction.

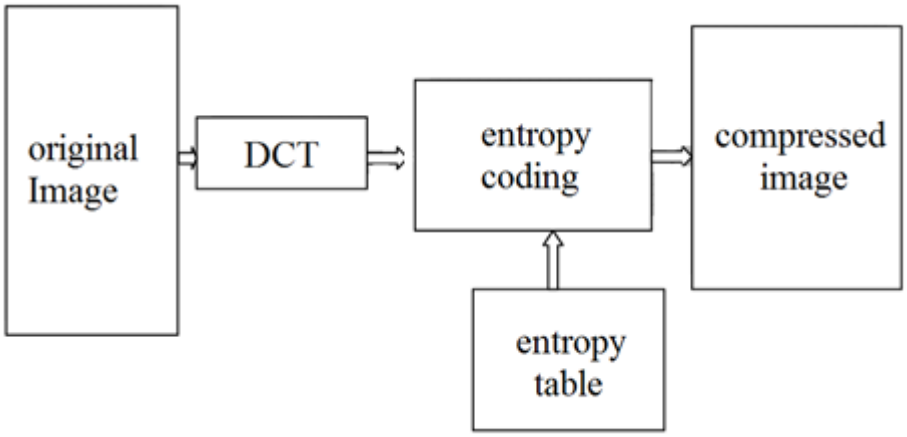


Figure 1.4: Principle of lossless JPEG algorithm.

4.2 Lossy methods

These methods are characterized by the loss of the original image, but the compression rate is higher than that of the lossless methods. They are in three types, transformed based methods, predictive methods and hybrid methods.

4.2.1 Transform based Methods

In these methods [1], the image is split into NxN sub-scale images or blocks of small size (the amount of computation required to perform the processing on the entire image is very high). Each block undergoes orthogonal linear invertible mathematical transformation from the spatial domain to the frequency domain, independently from the other blocks (a set of transform coefficients more or less independent). The resulting coefficients are then quantized and encoded for transmission or storage. To find the intensity of the original pixels, an inverse transformation is applied to these coefficients. Among the existing linear transformations we have:

- Karhunen-Loeve (KLT).
- Discrete Fourier transform (DFT).
- Hadamard transform (HT).
- Discrete Cosine Transform (DCT)
- Wavelet transform (WT).

4.2.2 Predictive methods

The predictive method is one of the oldest, it is a de-correlation method where the principle is as follows. In predictive coding of each pixel value is predicted from previously encoded pixels. Only the difference between the predicted value and the actual value is quantified and then encoded and transmitted. The difference is usually small, thus its representation requires fewer bits than the pixel itself. Predictive methods allow easy implementation and lead to good compression ratio. They are effective for images with small temporal and spatial variations.

4.2.3 Hybrid Methods

The term hybrid refers to the techniques combining predictive coding and transform coding. In the case of still images, a transformation is performed in a dimension along the line and then a prediction along the columns. For moving images, a combination is carried out between a two-dimensional transformation in the spatial domain and a prediction along the temporal component to exploit the temporal redundancy of the image signal. The hybrid coder combines the advantages of both techniques within it.

5. Transform based methods

In this section we will focus on the transformed based methods for image compression as well as video compression, such as JEPG and MPEG, then we will see the embedded coding methods that are based on the wavelet transform.

5.1 Standard JPEG

After a long period of research and development, with the participation of a very large scientific community, the JPEG is now an international standard since 1993. The functional block diagram of the JPEG algorithm is shown in figure 1.5.

The image is divided into blocks of size 8x8. Each block is processed independently in three steps (4, 7):

- A DCT transformation [6] of each block: it is a spectral analysis by local cosine functions, one obtains 8x8 digital coefficients, where the information is concentrated on few coefficients.

- A step of quantification: quantification is adapted to the content of the DCT blocks in order to keep all coefficients significant irrespective of their position.

- Encoding Step: each spectrally quantified coefficients are a form of a few Nonzero and a majority of zero coefficients that have been eliminated by the step of quantification. A zigzag scan is then applied on each 8x8 matrix followed by an entropy coding specific to the JPEG standard.

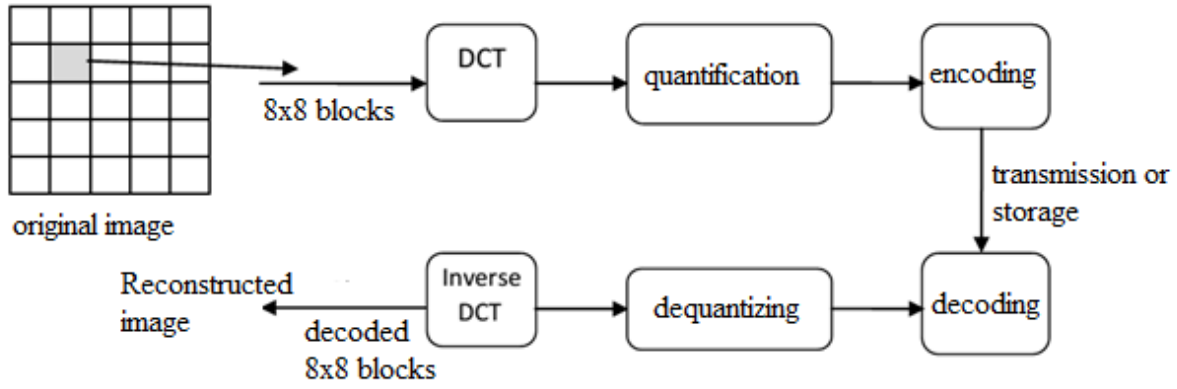


Figure 1.5: block diagram of the JPEG algorithm[5].

The JPEG standard is currently very used to encode the digital images that are found in our daily life (internet, digital cameras ... etc.). It is well suited for the natural images and for compression rate not exceeding 0.8 to 1 bit per pixel [5]. Beyond, artifacts (effect of blocks) will appear and will be very disturbing visually (figure 3.a).

5.2 The standard H-261 compression

It is a standard developed by CCITT (International Consultative Committee on Telegraphy and Telephony). This standard is intended for the coding of moving for video calling (Visual Telephony) images.

The H-261 is a hybrid combining the DCT encoding and predictive coding. DCT is used to reduce spatial redundancy (intra-frame coding). The predictive coding for reducing the temporal redundancy between frames of the sequence (interframe coding).

5.3 The standard MPEG compression

The efforts made by the teams CCITT (Consultative Committee for International Telephony and Telecommunication) for H.261 were used as a starting point for the development

of a standard for moving images by ISO, this standard is called MPEG for Moving Pictures Experts Group.

Unlike the first phase in H.261, MPEG is for the coding of moving images for the storage on digital media which are more flexible than those of H.261 constraints (which is why that it is called the standard multimedia applications).

5.4 Embedded coding

Wavelet transform generates sub-bands corresponding to the orthogonal projections of the vector space. There are structural similarities in the sub-bands details in the same direction at successive resolutions (see Figure 1.6). Algorithms that exploit this property and are called embedded zerotree coding such as EZW, SPIHT (see chapter 3) and SPECK.

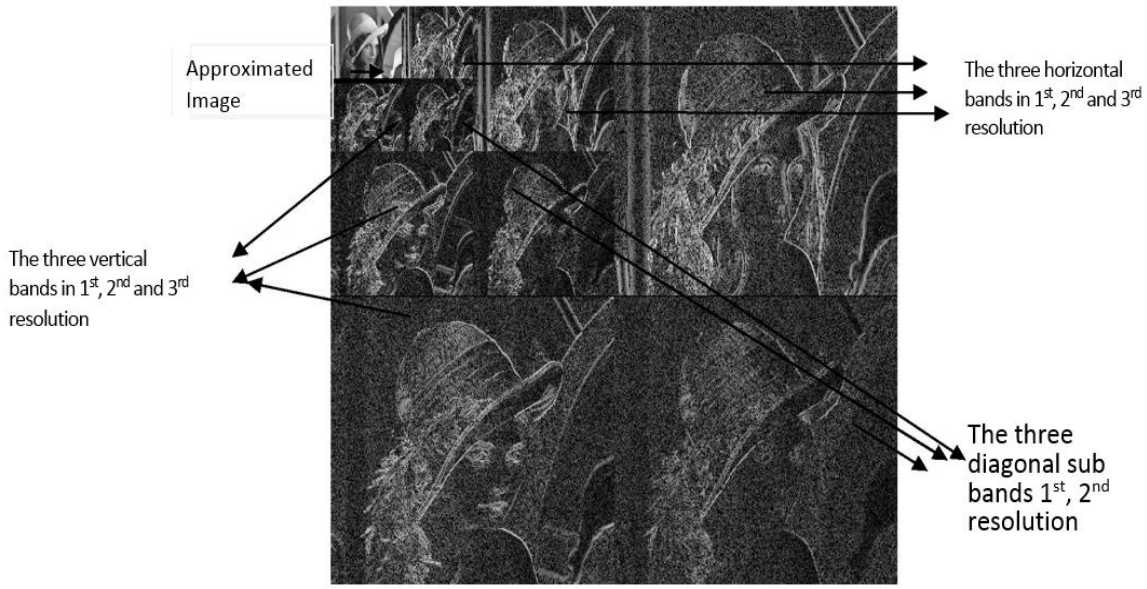


Figure 1.6: Example of a three level wavelet transform of image Lena

5.4.1 JPEG 2000 Standard

After the JPEG standard, JPEG2000 now occupy a prominent place in the world of digital imaging: internet, medicine, digital cameras, image transmission on mobile networks ... etc. The JPEG2000 standard allowed to solve the disturbing problem of artifacts by providing a good visual quality at high levels of compression. Figure 1.7 shows an example of a reconstructed image by the standards JPEG and JPEG2000 for the same compression rate; the difference in the visual quality is clear.

Figure 1.7: Results image Lena rebuilt to 0.2 bpp, a) JPEG standard with PSNR =23.4 dB; b) JPEG2000 standard, with PSNR=28.7 db

5.4.1.1 Principle of JPEG2000

The standard JPEG2000 is based on the discrete wavelet transform DWT (cf. chapter 2) instead of the DCT used by the JPEG standard. The DWT allows a more localized analysis therefore more fine, of the information which is not possible with the DCT.

DWT methods are distinguished by the selection procedure of the coefficients of wavelet before the step of quantification. Two main approaches are used to eliminate the non-significant coefficients:

- The selection by sub-band which is to zero all coefficients non-significant lower than a threshold in each of the sub-bands separately. The inter-band correlation is therefore not taken into account. The selection of coefficients is performed by the EBCOT algorithm (selection by context). This approach is largely integrated in the new JPEG 2000 standard.
- The multi-resolution selection which consists in taking account thresholding carried out through the sub-bands of a same direction. These techniques exploit the structural similarities in the sub-bands associated to a direction and successive resolutions. These techniques are called zerotree based techniques.

5.4.2 EBCOT Coding

The EBCOT encoder [5] does not belong to the family of zerotree compression algorithms. Its principle is summarized in figure 1.8.

The image can be processed as a whole or divided into rectangular 'tiles'. These tiles (of sizes 16x16, 32x32 or 64x64) will be then coded independently. The tiles (or the image) are then broken down into sub-bands on several levels. A scalar quantization is applied on the different sub-bands in order to reduce the accuracy of the coefficients supposed non-significant. Each sub-band is then cut into small independent blocks called "code-blocks'. These latter are then coded in a gradual way, by plan of bits, and by the means of an contextual arithmetic encoder.

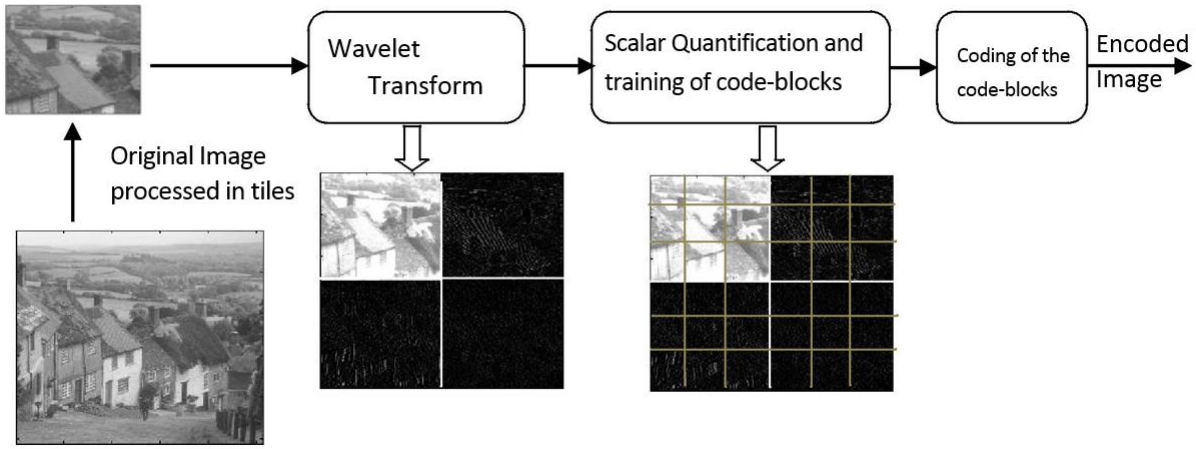


Figure 1.8: Principle of the EBCOT coding

5.4.3 SPECK Algorithm

SPECK algorithm stands for Set-Partitioning Embedded Block Coder [8] It uses a recursive set-partitioning procedure to sort subsets of wavelet coefficients by maximum magnitude with respect to thresholds that are integer powers of two. It exploits two fundamental characteristics of an image transform the well-defined hierarchical structure, and energy clustering in frequency and in space. The two partition strategies allow for versatile and efficient coding of several image transform structures, including dyadic, blocks inside subbands, wavelet packets, and DCT (discrete cosine transform). The SPECK image coding scheme has its roots primarily in the ideas developed in the SPIHT [9], AGP [10], and SWEET [11] image coding algorithms.

5.4.3.1 Principle

The coefficients are initially separated into two sets, one set noted S containing low frequency coefficients and the other, denoted Ґ containing the remaining coefficients (see Figure 1.9). Similarly as in SPIHT (see chapter 3 for more explanation of LIS and LSP), two linked lists are maintained: LIS – List of Insignificant Sets, and LSP – List of Significant Pixels.

The former contains sets of type S of varying sizes which have not yet been found significant against a threshold n while the latter obviously contains those pixels which have tested significant against n.

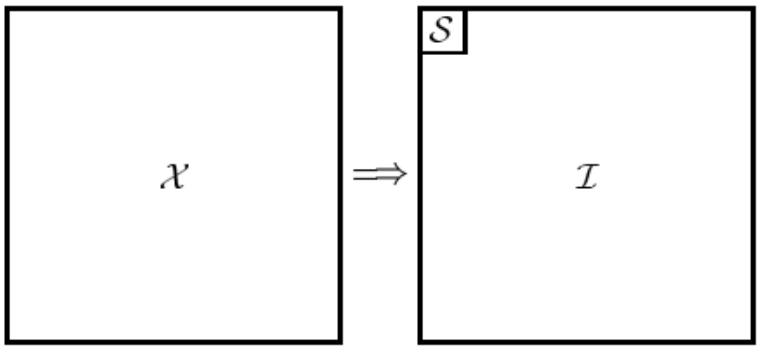


Figure 1.9: Partitioning of image x into sets s and Γ

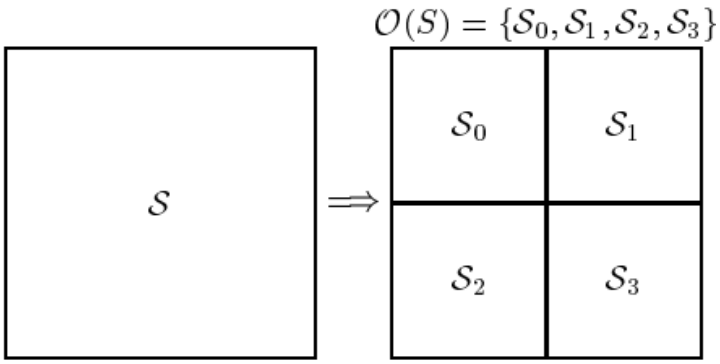


Figure 1.10: Partitioning of set s

5.4.3.2 Speck algorithm

We briefly go into the important steps of the SPECK algorithm in the following:

1) Initialization

- Partition image transform *X* into two sets: S ≡ root, and Γ≡ *X* – S.
- Output $n_{max} = \log_2(\max_{(i,j)\in X} |c_{i,j}|$
- Add S to LIS and set LSP = ϕ

2) Sorting Pass

- In increasing order of size |S| of sets (smaller sets first)

– for each set S Є LIS do ProcessS(S)

- if Γ ≠ ϕ, ProcessI()

3) Refinement Pass

- for each *(i, j)* Є LSP, except those included in the last sorting pass, output the n-th MSB of $|c_{i,j}|$

4) Quantization Step

- decrement n by 1, and go to step 2

The procedures used in the algorithm are shown below:

Procedure ProcessS(S)

1) output $\Gamma_n(S)$

2) if $\Gamma_n(S) = 1$

- if S is a pixel, then output sign of S and add S to LSP.
- else CodeS(S)
- if S $\in$ LIS, then remove S from LIS

3) else

- if S doesn't belong to LIS, then add S to LIS

4) return

Procedure CodeS(S)

1) Partition S into four equal subsets $O(S)$ (see Fig. 1.10)

2) for each set $S_i \in O(S)$ (i = 0, 1, 2, 3)

- output $\Gamma_n(S_i)$
- if $\Gamma_n(S_i) = 1$

– if S_i is a pixel, output its sign and add S_i to LSP

– else CodeS(S_i)

- else

– add S_i to LIS

3) return

Procedure ProcessI()

1) output $\Gamma_n(I)$

2) if $\Gamma_n(I) = 1$

- CodeI()

3) Return

Procedure CodeI()

1) Partition *I* into four sets—three S_i and one *I* (see Fig. 1.9)

2) for each of the three sets S_i (i = 0, 1, 2) .

- ProcessS(SZ)

3) ProcessI()

4) return

5.4.4 Algorithm EZBC

The principle of the coding algorithm EZBC (Embedded ZeroBlocks coding based on Context modeling) [1] similarly to the SPECK algorithm. The innovation of this coder mainly from exploiting the dependence between the nodes of quad-tree of significance. The quad-tree partitioning is increasingly done independently in each sub-band to allow greater separation of statistical significance and of a more effective learning with more extensive [1] contexts.

This coder provides comparable performance to EBCOT, and has also been adapted to video encoding as the MC-EZBC [1].

6. Conclusion

Image compression is designated to take an even more important role because of the evolution of networks and multimedia. Its importance is mainly due to the mismatch between the hardware capabilities of the devices we use. Depending on the desired application, one can choose between lossless and lossy compression of the information. Several algorithms for the two cases are proposed in the literature. However, the algorithms based on wavelet transform have gained considerable attention in recent years. The latter offers a finer level of signal analysis and adapts better to local properties of the image. In the next chapter we will go in details on time-frequency analysis and the use of wavelets in image processing.

Chapter II: Wavelets and multi-resolution analysis

Abstract

We limit ourselves in this second chapter to the presentation of the main tool to process the images: the wavelet transform. We begin by describing the directional Fourier transform and emphasize the limitations in their use in the embedded coding. Then we talk about the main features of the Fourier transform. Then we continue with a brief review of Short Time Fourier Transform. The last part of this chapter will be devoted to the continuous wavelet transform and discrete wavelet specifically multiresolution analysis.

1. Introduction

Since their introduction two decades ago, wavelets have gained considerable interest in signal processing. The idea of representing a signal at different resolutions allows to extract its main trends in a limited number of coefficients, while precisely locating the discontinuities. In the context of image processing, wavelets have been used for various applications, such as sound effects and even compression, leading to standards such as JPEG2000. It is well known that the wavelets are optimum for the representation of one-dimensional signals (1D) having a finite number of discontinuities.

In the first, we include directional transforms and Fourier transform highlighting its shortcomings, and the reasons why wavelets are considered to be better at approximating a signal or an image; we discuss relevant concepts through definitions of the short time Fourier transform and then we define the continuous wavelet transform. Then we present the theory of multi-resolution analysis (MRA) and the pyramidal algorithm for obtaining the coefficients of the discrete wavelet transform.

2. Directional Transforms

The most important transforms that had the most impact is the Fourier transform where we will discuss it next section. Among the directional transforms, we mention the following[1]:

- Radon transform
- Ridgelet transform
- Curvelet transform
- Contourlet transform
- Brushlet transform
- Beamlet transform
- Wedgelet transform
- Etc..

3. Fourier Transform

The Fourier transform of a signal x (t) is given by the relationship 2.1

$$X(f) = \int_{-\infty}^{+\infty} x(t) e^{-j2\pi ft} dt = \text{<x(t), ej2πft>} \qquad (2.1)$$

The Fourier transform [12] is the best tool to visualize the frequencies that constitutes a signal, it's invertible which allows a good handle and understanding of going from the time domain to the frequency domain. The limitation of the FT is that it gives all the signal's frequencies but not when they occur.

The need for Wavelet Transform (WT) arises since no frequency information is available in the time-domain signal, and no time information is available in the Fourier transformed signal. It is sometimes necessary to have both the time and the frequency information at the same time depending on the particular application, and the nature of the signal in hand. Recall that the FT gives the frequency information of the signal, which means that it tells us how much of each frequency exists in the signal, but it does not tell us when in time these frequency components exist. This information is not required when the signal is so-called stationary. Stationary signals are those whose frequency content does not change in time.

In this case, one does not need to know at what times frequency components exist, since all frequency components exist at all times. For example the following signal:
$x(t)=\cos(2\pi*10t)+\cos(2\pi*25t)+\cos(2\pi*50t)+\cos(2\pi*100t)$ It is a stationary signal, because it has frequencies of 10, 25, 50, and 100 Hz at any given time instant. This signal and its FT are shown in figure 2.1 below:

a) b)

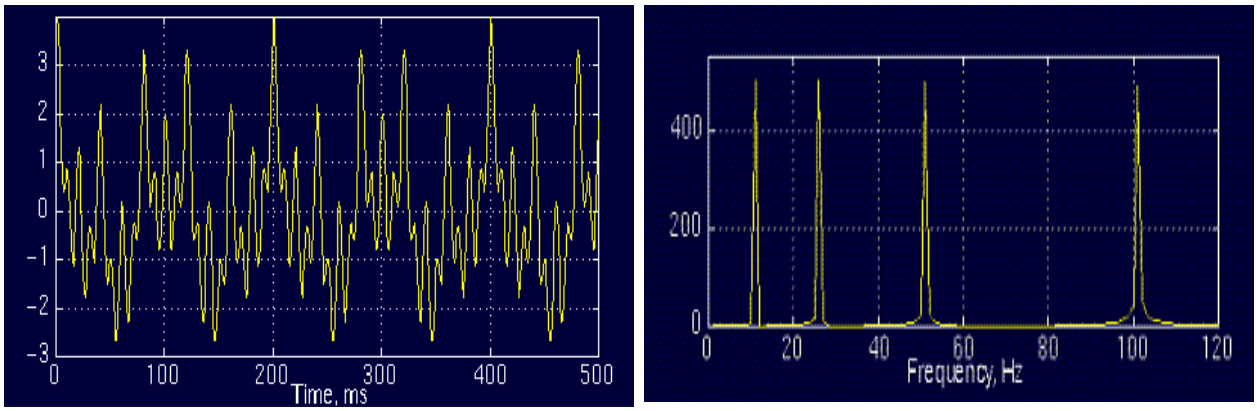


Figure 2.1: a) Example of a stationary signal and b) its Fourier transform [12]

In the above figure, the four spectral components corresponding to the frequencies 10, 25, 50 and 100 Hz are shown.

Contrary to the signal in Figure 2.2, a signal with four different frequency components at four different time intervals, hence a non-stationary signal. The interval 0 to 300 ms has a 100 Hz

sinusoid, the interval 300 to 600 ms has a 50 Hz sinusoid, the interval 600 to 800 ms has a 25 Hz sinusoid, and finally the interval 800 to 1000 ms has a 10 Hz sinusoid.

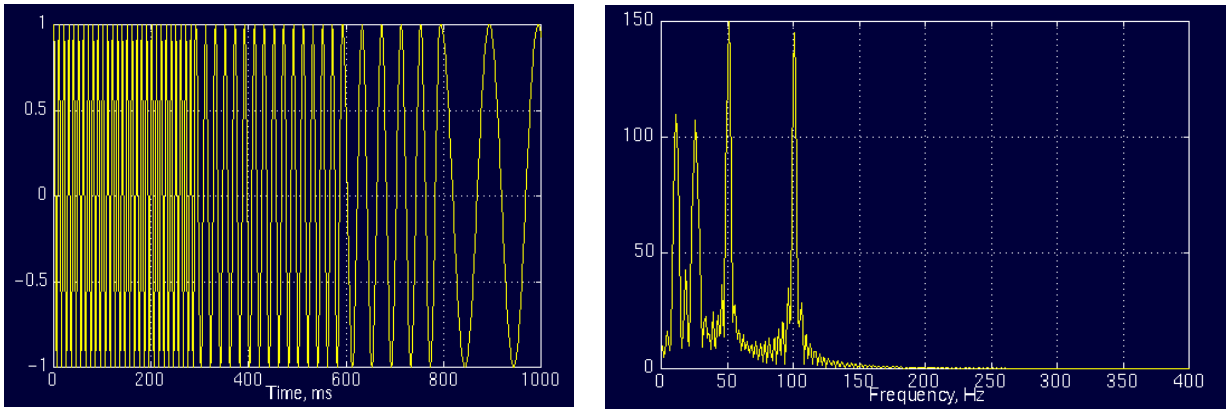


Figure 2.2: a) Example of a non-stationary signal and b) its Fourier transform

Now, comparing the Figures 2.1.a and 2.2.b the two spectrums are similar: Both of them show four spectral components at exactly the same frequencies, i.e., at 10, 25, 50, and 100 Hz. Other than the ripples, and the difference in amplitude (which can always be normalized), the two spectrums are almost identical, although the corresponding time-domain signals are not even close to each other.

The first solution that comes naturally to mind is to limit the scope of integration time using a "window" function that can be dragged to explore the signal; thereby obtaining the sliding window Fourier transform or the short-time Fourier transform.

4. Short Time Fourier Transform

The short-time Fourier transform (STFT), [12] or alternatively short-term Fourier transform, is a Fourier-related transform used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time.

The basic idea is that we can assume that, some portion of a non-stationary signal is stationary. If this region where the signal can be assumed to be stationary is too small, then we look at that signal from narrow windows, narrow enough that the portion of the signal seen from these windows are indeed stationary. This approach of researchers ended up with a revised version of the Fourier transform, so-called: The Short Time Fourier Transform (STFT).

There is only a minor difference between STFT and FT. In STFT, the signal is divided into small enough segments, where these segments (portions) of the signal can be assumed to

be stationary. The short-term Fourier transform or the Short Time Fourier Transform is given by the equation 2.2:

$$STFTx\,(f,\,t) = \int_{-\infty}^{+\infty} g^{*}(t-\tau)x(t)e^{-j2\pi ft}\,dt \qquad (2.2)$$

If we set:

$$G_{f,\tau}(t) \;= g(t-\tau) \qquad (2.3)$$

This can be interpreted as the projection transform of "f" on the base of sliding window functions:

$$STFTx\,(f,\,t) = < G_{f,\tau}(t)\,,\; x(t) > \qquad (2.4)$$

The notation <g, x> is the scalar product defined as follows:

$$< g,\, x > = \int_{-\infty}^{+\infty} g(t) * x(t)\,dt \qquad (2.5)$$

A number of window functions are used, the best known are the windows of Hanning and Gaussian:

$$g(t) = \pi^{-1/4} e^{-x^{2}/2} \qquad (2.6)$$

In FT, the kernel function, allows us to obtain perfect frequency resolution, because the kernel itself is **a window of infinite length**. In STFT is window is of finite length, and we no longer have perfect frequency resolution. We are faced with the following dilemma otherwise called the Heisenberg uncertainty principle:

If we use a window of infinite length, we get the FT, which gives perfect frequency resolution, but no time information. Furthermore, in order to obtain the stationarity, we have to have a short enough window, in which the signal is stationary. The narrower we make the window, the better the time resolution, and better the assumption of stationarity, but poorer the frequency resolution:

Narrow window: good time resolution, poor frequency resolution.

Wide window: good frequency resolution, poor time resolution.

The Wavelet transform (WT) solves the dilemma of resolution to a certain extent, as we will see in the next part.

5. Wavelet Transform

Although the time and frequency resolution problems are results of a physical phenomenon (the Heisenberg uncertainty principle) and exist regardless of the transform used,

it is possible to analyze any signal by using an alternative approach called the multiresolution analysis (MRA). MRA, as implied by its name, analyzes the signal at different frequencies with different resolutions.

MRA is designed to give good time resolution and poor frequency resolution at high frequencies and good frequency resolution and poor time resolution at low frequencies. This approach makes sense especially when the signal at hand has high frequency components for short durations and low frequency components for long durations. Fortunately, the signals that are encountered in practical applications are often of this type.

The continuous wavelet transform [13] was developed as an alternative approach to the short time Fourier transform to overcome the resolution problem. The wavelet analysis is done in a similar way to the STFT analysis, in the sense that the signal is multiplied with a function, the wavelet, similar to the window function in the STFT, and the transform is computed separately for different segments of the time-domain signal.

The continuous wavelet transform is defined as follows:

$$CWT_x^{\psi}(\tau, s) = \Psi_x^{\psi}(\tau, s) = \frac{1}{\sqrt{s}} \int x(t) \psi^*\left(\frac{t-\tau}{s}\right) dt \qquad (2.7)$$

As seen in the above equation, the transformed signal is a function of two variables, and, the translation and scale parameters, respectively, is the transforming function, and it is called the mother wavelet. The term mother wavelet gets its name due to two important properties of the wavelet analysis as explained below:

The term wavelet means a small wave. The smallness refers to the condition that this (window) function is of finite length (compactly supported). The wave refers to the condition that this function is oscillatory. The term mother implies that the functions with different region of support that are used in the transformation process are derived from one main function, or the mother wavelet. In other words, the mother wavelet is a prototype for generating the other window functions. Figure 2.3 illustrates an example of a wavelet.

Figure 2.3 – (a) Daubechies-5 and (b) Meyer wavelets.

6. Discrete Wavelet Transform

The DWT [14] is considerably easier to implement when compared to the CWT. The basic concepts of the DWT will be introduced in this section along with its properties and the algorithms used to compute it.

The main idea is the same as it is in the CWT. A time-scale representation of a digital signal is obtained using digital filtering techniques. In the discrete case, filters of different cutoff frequencies are used to analyze the signal at different scales. The signal is passed through a series of high pass filters to analyze the high frequencies, and it is passed through a series of low pass filters to analyze the low frequencies.

The resolution of the signal, which is a measure of the amount of detail information in the signal, is changed by the filtering operations, and the scale is changed by upsampling and downsampling (subsampling) operations.

Although it is not the only possible choice, DWT coefficients are usually sampled from the CWT on a dyadic grid. Since the signal is a discrete time function, the terms function and sequence will be basically the same. This sequence will be denoted by x[n], where n is an integer.

The procedure starts with passing this signal (sequence) through a half band digital lowpass filter with impulse response h[n]. Filtering a signal corresponds to the mathematical operation of convolution of the signal with the impulse response of the filter. The convolution operation in discrete time is defined as follows:

$$x[n] * h[n] = \sum_{k=-\infty}^{+\infty} x[k].h[n-k] \qquad (2.8)$$

The unit of frequency is of particular importance at this time. In discrete signals, frequency is expressed in terms of radians. Accordingly, the sampling frequency of the signal is equal to 2π radians in terms of radial frequency. Therefore, the highest frequency component that exists in a signal will be π radians, if the signal is sampled at Nyquist's rate (which is twice the maximum frequency that exists in the signal); that is, the Nyquist's rate corresponds to rad/s in the discrete frequency domain. Therefore using Hz is not appropriate for discrete signals. However, Hz is used whenever it is needed to clarify a discussion, since it is very common to think of frequency in terms of Hz. Thus the unit of frequency for discrete time signals is radians.

After passing the signal through a half band lowpass filter, half of the samples can be eliminated according to the Nyquist's rule, since the signal now has a highest frequency of $\pi/2$

radians instead of π radians. Simply discarding every other sample will subsample the signal by two, and the signal will then have half the number of points. The scale of the signal is now doubled. Note that the lowpass filtering removes the high frequency information, but leaves the scale unchanged. Only the subsampling process changes the scale. Resolution, on the other hand, is related to the amount of information in the signal, and therefore, it is affected by the filtering operations. Half band lowpass filtering removes half of the frequencies, which can be interpreted as losing half of the information. Therefore, the resolution is halved after the filtering operation. The subsampling operation after filtering does not affect the resolution, since removing half of the spectral components from the signal makes half the number of samples redundant anyway. Half the samples can be discarded without any loss of information. In summary, the lowpass filtering halves the resolution, but leaves the scale unchanged.
The signal is then subsampled by 2 since half of the number of samples are redundant. This doubles the scale.

This procedure can mathematically be expressed as

$$y[n] = \sum_{k=-\infty}^{+\infty} h[k].x[2n-k] \qquad (2.9)$$

The DWT analyzes the signal at different frequency bands with different resolutions by decomposing the signal into a coarse approximation and detail information. DWT employs two sets of functions, called scaling functions and wavelet functions, which are associated with low pass and highpass filters, respectively. The decomposition of the signal into different frequency bands is simply obtained by successive highpass and lowpass filtering of the time domain signal. The original signal x[n] is first passed through a halfband highpass filter g[n] and a lowpass filter h[n]. After the filtering, half of the samples can be eliminated according to the Nyquist's rule, since the signal now has a highest frequency of π/2 radians instead of π. The signal can therefore be subsampled by 2, simply by discarding every other sample. This constitutes one level of decomposition and can mathematically be expressed as follows:

$$y_{high}[k] = \sum_{n} x[n].g[2k-n] \qquad (2.10)$$

$$y_{low}[k] = \sum_{n} x[n].h[2k-n] \qquad (2.11)$$

Where $y_{high}[k]$ and $y_{low}[k]$ are the outputs of the highpass and lowpass filters, respectively, after subsampling by 2.

This decomposition [15] halves the time resolution since only half the number of samples now characterizes the entire signal. However, this operation doubles the frequency

resolution, since the frequency band of the signal now spans only half the previous frequency band, effectively reducing the uncertainty in the frequency by half. The above procedure, which is also known as the subband coding, can be repeated for further decomposition. At every level, the filtering and subsampling will result in half the number of samples (and hence half the time resolution) and half the frequency band spanned (and hence double the frequency resolution). Figure 2.4 illustrates this procedure, where x[n] is the original signal to be decomposed, and h[n] and g[n] are lowpass and highpass filters, respectively. The bandwidth of the signal at every level is marked on the figure as "f".

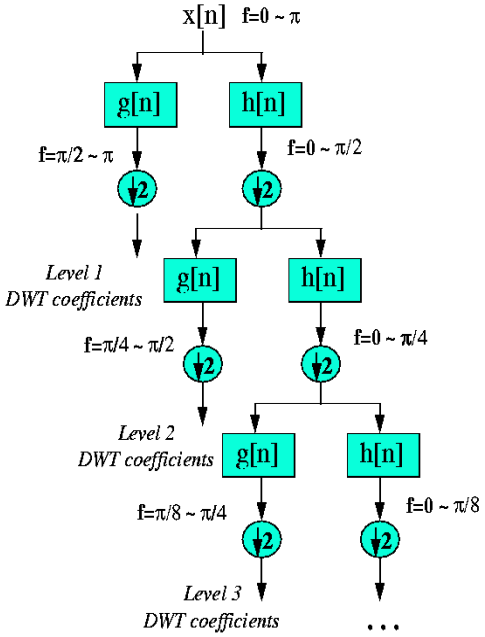


Figure 2.4: 1-D wavelet signal decomposition

6.1 Inversion - Eligibility

It can be shown that if the analyzing function (wavelet) is suitably chosen, the wavelet transform is invertible and the function can be reconstructed after analysis according to the equation (2.12):

$$f = C_{\psi}^{-1} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \frac{1}{a^2} \langle f, \psi_{a,b} \rangle \psi_{a,b} \, da db \qquad (2.12)$$

This possibility remains theoretical because the calculation is only possible numerically and its convergence can be very slow. The coefficient if it exists, is given by equation (2.13):

$$C_{\psi} = 2\pi \int_{-\infty}^{+\infty} \left|\psi\hat{}(w)\right|^2 \frac{dw}{w} \qquad (2.13)$$

The condition of existence of this coefficient is the eligibility condition of analyzing wavelet function. This condition is illustrated by the equation 2.14:

$$\int_{0}^{+\infty} \left|\psi\hat{}(w)\right|^2 \frac{dw}{|w|} = \int_{-\infty}^{0} \left|\psi\hat{}(w)\right|^2 \frac{dw}{|w|} < \infty \qquad (2.14)$$

7. Approximation spaces

7.1. Scaling Function

A scaling function [14] is used to approximate a signal or an image function at different level of approximations. Each approximation is differed by a factor of two from the approximation at the nearest neighboring level. Scaling functions are actually expansion functions which are composed of integer translations and binary scaling and contained in the set $\{\phi_{j,k}(x)\}$. The general scaling functions are:

$$\phi_{j,k}(x)=2^{j/2}\phi(2^j x-k) \qquad (2.15)$$

Both j and k$\in \boldsymbol{Z}$, and $\phi(x) \in L^2(\boldsymbol{R})$. The parameter k and j determine the position of $\phi_{j,k}(x)$ along the x-axis and the width of $\phi_{j,k}(x)$ along the x-axis respectively. For a specific j, the subspace of an expansion set is commonly expressed as:

$$V_j = \overline{\underset{k}{Span}\left\{\varphi_{j,k}(x)\right\}} \qquad (2.16)$$

The parameter j is proportional to the size of V_j [14]. There are four fundamental requirements [14] of multi-resolution analysis that scaling functions must follow:

1. The scaling function is orthogonal to its integer translates.

2. The subspaces spanned by the scaling function at low resolutions are contained within those spanned at higher resolutions: $V_{-\infty} \subset \ldots \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \ldots \subset V_{\infty}$.

3. The only function that is common to all V_j is f(x)=0. That is:

$$V_{-\infty}=\{0\} \qquad (2.17)$$

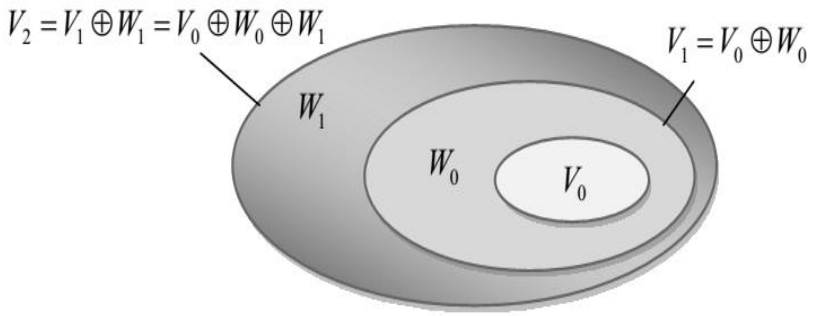


Figure 2.5: The spatial relation of scaling and wavelet function spaces.

This is also called downward completeness property [15].

4. Any function can be represented with arbitrary precision. As the level of the expansion function approaches infinity, the expansion function space V contains all the subspaces.

$$V_{\infty}=\{L^2(\boldsymbol{R})\}$$

This is also called upward completeness property [15]. With the above condition being satisfied, the weighted sum of the expansion functions of subspace V_{j+1} can be used to express the expansion functions of subspace Vj [14]:

$$\varphi_{j,k}(x) = \sum_n \alpha_n \varphi_{j+1,n}(x) \qquad (2.18)$$

7.2. Wavelet Function

Wavelet function [14] is a type of scaling function that satisfied the four MRA requirements for scaling functions described in the previous section. The wavelet function is analogous to the scaling function expression in (2.15). Both integer translation and binary scaling are incorporated. The function ψ(x) is defined as:

$$\psi_{j,k}(x) = 2^{j/2}\psi(2^j x - k) \qquad (2.19)$$

for all k ∈Z that spans the space W_j where

$$W_j = \underset{k}{span}\left\{\psi_{j,k}(x)\right\} \qquad (2.20)$$

The wavelet function spans the difference between any two adjacent scaling subspaces, V_j and V_{j+1} as depicted in Figure 2.5. Therefore, a general equation describing the relationship between the scaling and wavelet function spaces is derived:

$$V_{j+1} = V_j \oplus W_j \qquad (2.21)$$

This expression can be further extended to express the space of all measurable, square-integrable functions:

$$L^2(R) = V_0 \oplus W_0 \oplus W_1 \oplus \ldots\ldots \quad (2.22)$$

Equation (21) can be expressed without the scaling function space:

$$L^2(R) = \ldots \oplus W_{-2} \oplus W_{-1} \oplus W_0 \oplus W_1 \oplus W_2 \oplus \ldots. \quad (2.23)$$

Fig. 1 also illustrates crucial information about the purpose of wavelet and scaling functions. The scaling function at the lowest resolution level V_0 provides an approximation to the actual function f(x) and wavelets from W_0 encodes the difference between this approximation and the actual function. In fact, any wavelet function can be expressed as a weighted sum of shifted, double-resolution scaling functions:

$$\psi(x) = \sum_n h_\psi(n)\sqrt{2}\phi(2x-n) \quad (2.24)$$

where the $h_\psi(n)$ are called the wavelet function coefficients. The $h_\psi(n)$ can be related to $h_\phi(n)$ by:

$$h_\psi(n) = (-1)^n h_\phi(1-n) \quad (2.25)$$

8. Wavelet Transforms in Two Dimensions

A two-dimensional scaling function [13], $\varphi(x, y)$, and three two-dimensional wavelet $\psi^h(x, y)$, $\psi^V(x, y)$, $\psi^D(x, y)$ are critical elements for wavelet transforms in two dimensions [14]. These scaling function and directional wavelets are composed of the product of a one-dimensional scaling function φ and corresponding wavelet ψ which are demonstrated as the following:

$$\varphi(x, y) = \varphi(x)\varphi(y) \quad (2.26)$$

$$\psi^H(x, y) = \psi(x)\varphi(y) \quad (2.27)$$

$$\psi^V(x, y) = \varphi(y)\psi(x) \quad (2.28)$$

$$\psi^D(x, y) = \psi(x)\psi(y) \quad (2.29)$$

Where ψ^h measures the horizontal variations (horizontal edges), ψ^V corresponds to the vertical variations (vertical edges), and ψ^D detects the variations along the diagonal directions.

The two-dimensional DWT can be implemented using digital filters and down samplers. The block diagram in Fig. 7 shows the process of taking the one-dimensional FWT of the rows of $f(x, y)$ and the subsequent one-dimensional FWT of the resulting columns. Three sets of detail coefficients including the horizontal, vertical, and diagonal details are produced. By iterating the single-scale filter bank process, multi-scale filter bank can be generated. This is achieved by tying the approximation output to the input of another filter bank to produce an

arbitrary scale transform. For the one-dimensional case, an image $f(x, y)$ is used as the first scale input. The resulting outputs are four quarter-size subimages: W_ψ^H, W_ψ^V and W_ψ^D which are shown in the center quad-image in Figure 2.6. Two iterations of the filtering process produce the two-scale decomposition at the right of Figure 2.6. Figure 2.7 shows the synthesis filter bank that is exactly the reverse of the forward decomposition process.

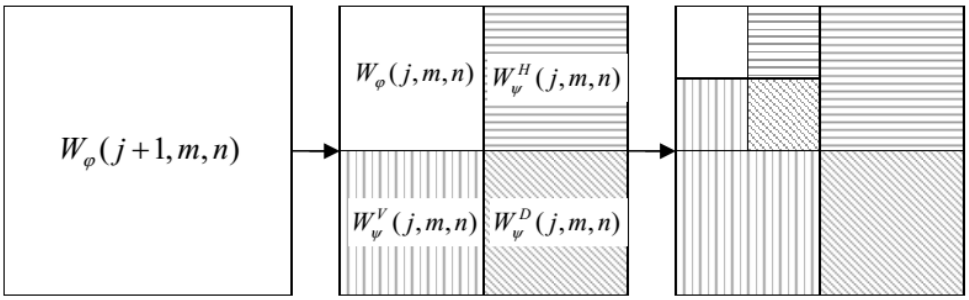


Figure 2.6. A two-level decomposition of the two-dimensional FWT

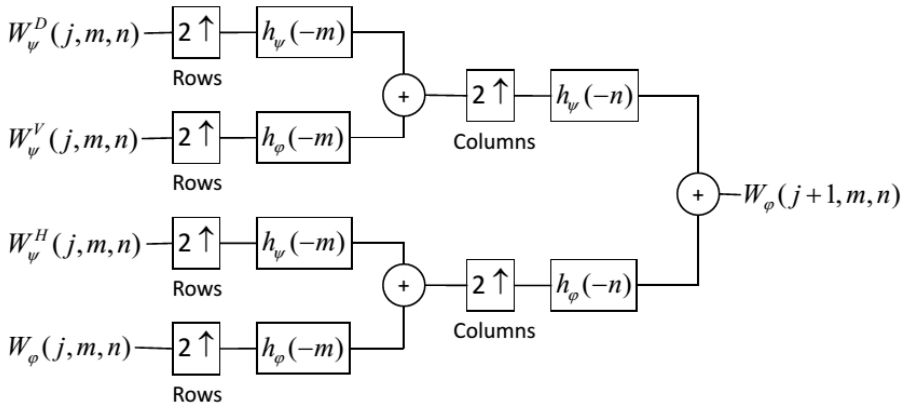


Fig. 2.7. The synthesis filter bank of the two-dimensional FWT.

9. Conclusion

We have discussed the implementation and theoretical foundation of the time-frequency analysis and the multiresolution analysis. Techniques based on or related to the multiresolution theory such as subband coding and pyramid algorithms are introduced. In the next chapter we will see how the wavelet transform is useful by introducing embedded coding and the principals of the zero tree coding.

Chapter III: EZW and SPIHT algorithms

Abstract

Our work is in the study of algorithms using wavelet compression embedded coding EZW SPIHT and the proposal of the optimization of the EZW and SPIHT algorithms to be faster. This chapter will be devoted to the study of Shapiro's EZW and Pearlman's SPIHT. After it will be a detailed explanation of the proposed algorithms with examples of execution.

3.1. Introduction

The use of the wavelet transform in the step of transformation provides a multiresolution image analysis. This property is very interesting for progressive compression of digital images. The purpose of this chapter is to present the EZW and SPIHT algorithms and our modifications, which are named B-EZW and B-SPIHT. First, we will give a brief review of the original EZW method developed by Shapiro and the SPIHT algorithm proposed by Amir Said et al., after we will explain B-EZW and B-SPIHT while giving examples and comparing them to the original algorithms.

3.2. Embedded coding

To better explain the concept of embedded coding, consider the following example. Assume that three users expect a compressed image to be sent. At the same time, they require different quality of the image. The first requires a 10 KB image quality, and the second and third user need a quality of 20 KB and 50 KB, respectively. Most methods of image compression with partial loss of data will have to compress the original image three times with different qualities, in order to generate three different files for the required qualities. A method based on embedded coding produces a single file, and users will receive three fragments of the file where lengths are equal to 10 KB, 20 KB and 50 KB.

3.2. Shapiro's EZW

Wavelet decomposition image can gives set of spatial factors, which consists of trees. The wavelet Tree coefficients are defined as the set of coefficients from different regions that describes the same spatial area in the image. Image 3.1 shows a three-level wavelet decomposition of Biskra logo image.

The arrows in 3.2 show the relationship of the parent - child from the top in the three level tree form; H denotes a high frequency, L – denotes low frequencies. Most low-frequency component of wavelet decomposition is presented root node of the tree, located in the upper left corner. Most high-frequency component of wavelet decomposition is represented by end node of the tree, located in the lower right corner. Except for the root node,

which has only three subsidiary peaks, each parent node has four subsidiary peaks, namely plot size 2x2 with the same spatial location in a high-frequency component.

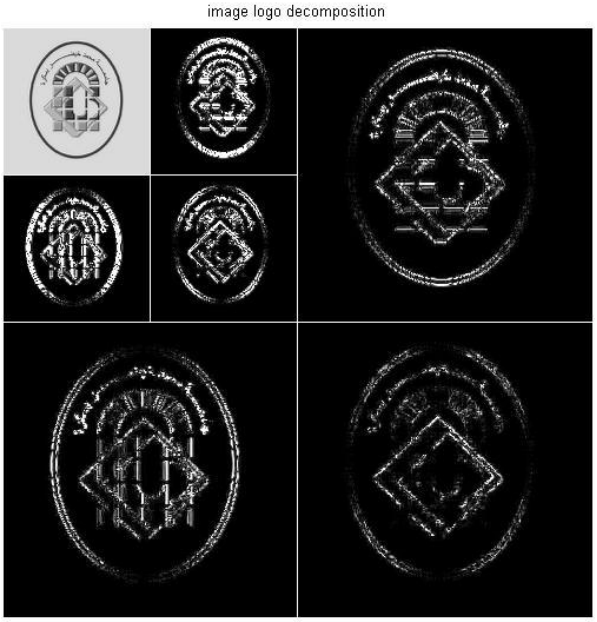


Figure 3.1: Example of decomposition of the image logo University Biskra using three resolutions wavelet

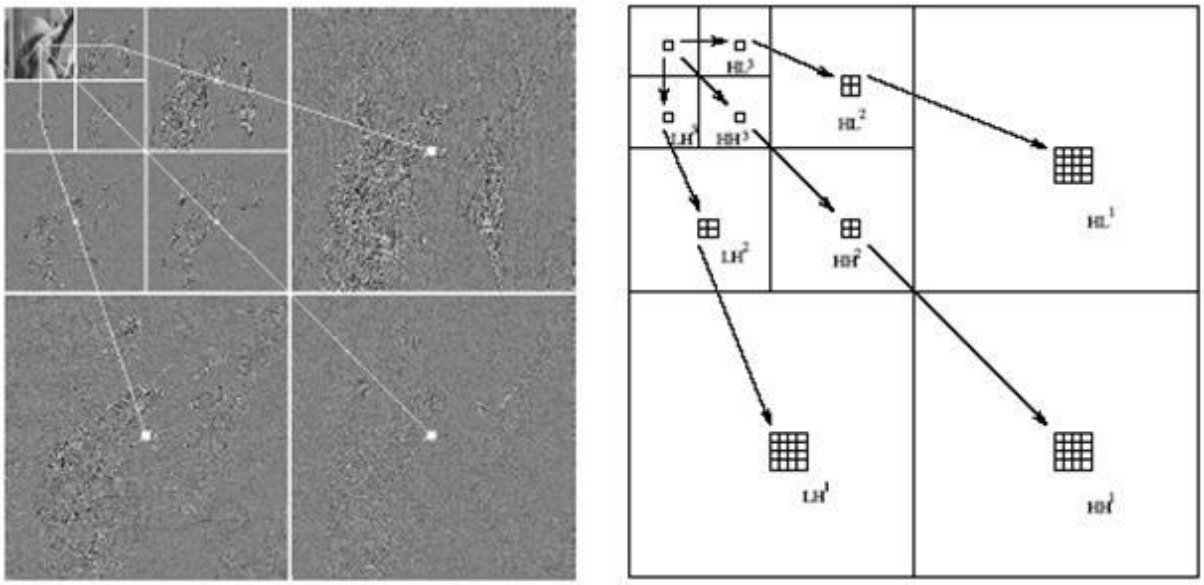


Figure 3.2: Parent/offspring Relationship

The notion zero tree assumes that if a wavelet coefficient at a coarse scale is insignificant compared to a threshold T, then it is highly likely that all the wavelet coefficients at the more fine scales which have the same orientation (horizontal, vertical or diagonal) and the same spatial location are also insignificant compared to the same threshold T. In practice, the likelihood of this phenomenon occurs is very likely [16].

More specifically, in a hierarchical sub-bands system except the lower frequency sub-band (A3 in Figure 3.2), each coefficient in a given scale can be connected to a set of coefficients in the next finer scale in the same spatial direction (see figure 3.2). The coefficient in the coarse scale is called the "parent" and all the coefficients of the same spatial orientation in the next finer scale are called "offspring" or "child".

EZW method using tree zeros (even SPIHT) is based on the idea of using many passes to encode zero trees to transfer most significant coefficients. The set of coefficients called significant, if their value are greater than a certain threshold (eg : the threshold can be Degrees of two). Otherwise, they are not significant.

In each pass, sets of greater significance in the trees are analyzed first. If the set is insignificant, it is encoded by the corresponding symbol, zero tree if all the coefficients in the set are insignificant. Otherwise, the set is divided into subsets (child set) for further analysis of significance. After the pass is done and all the factors are analyzed, the threshold in the next passage is reduced in half.

The basic assumption when coding images using trees zeros is that if the parent node of the wavelet tree coefficient is insignificant, it is very likely that its descendants will be insignificant. This zero tree property is very efficient for compression.

The goal of the Shapiro's encoder [16] is to exploit the dependencies between coefficients of the various sub-bands in order to achieve a notion of zero trees. Shapiro proposed this method in a paper published in 1993. SPIHT method is an extension method EZW.

EZW (embedded zero tree wavelet) the word zerotree (tree zeros) means that the method uses the concept of tree zeros. The word embedded (attached) means that the encoder compresses the image forming a bit stream with increasing accuracy. That is when the stream is add more bits; the reproduced image is more accurate.

3.2.1. Test of significance

The coding of wavelet coefficients involves determining coefficients of two lists:

- A list containing the coordinates of coefficients that are not significant compared to the current threshold Tj. It is called the dominant pass D.

- A list containing the values of the amplitudes of coefficients already found significant. This is called the subordinate pass S.

3.2.2. Initialization

After computing the discrete wavelet transform of the image, we first define a threshold on the first pass [16]:

$$T_0 = 2^{\lfloor \log_2(|C_{max}|) \rfloor}$$

With *C max = max (Wi, j)* where *Wi, j* is the wavelet coefficient at coordinates *(i, j).*

the wavelet transform coefficients are processed in an order called Morton's order. This procedure is shown below for an 8x8 matrix. At each stage, performing dominant and subordinate passes.

1	2	5	6	17	18	21	22
3	4	7	8	19	20	23	24
9	10	13	14	25	26	29	30
11	12	15	16	27	28	31	32
33	34	37	38	49	50	53	54
35	36	39	40	51	52	55	56
41	42	45	46	57	58	61	62
43	44	47	48	59	60	63	64

Figure 3.3: scanning order of the subbands for encoding significance map

The dominant pass

- If the coefficient is positive and greater than the threshold, it is coded symbol P.
- If the coefficient is negative and the absolute value is greater than the threshold, it is coded symbol N.
- The symbol ZTR is coded for zerotree root.

- The symbol IZ or Z code insulated zero coefficients that are smaller than the absolute value of the threshold, but are not zerotree root.

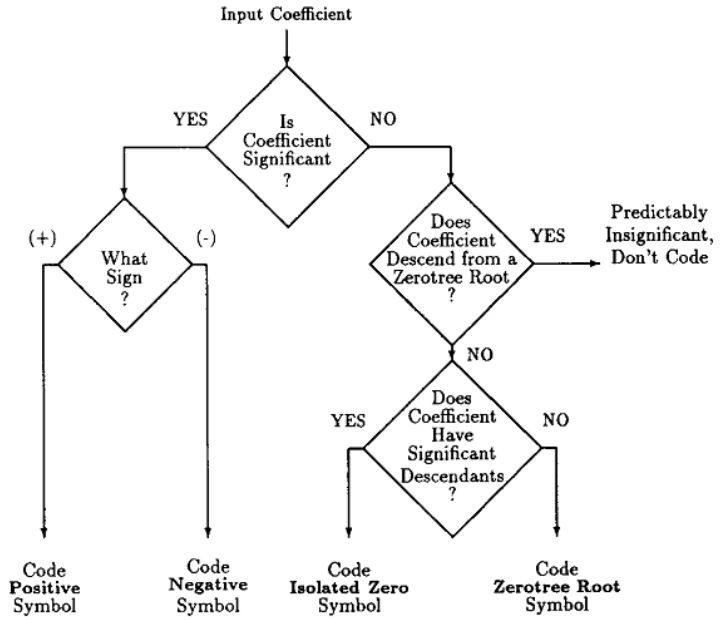


Figure 3.4: Principal of coefficients' significance test for EZW

3.2.3. Quantification and Refinement

The third step focuses on coding the elements of the list S. It is in this list the coefficients with large amplitudes relative to the threshold are, they must be quantified and then encoded. The quantization step takes place in two stages: [16]

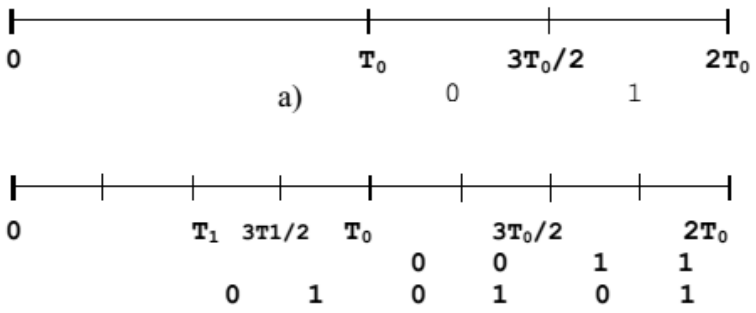
- Refinement of the coefficients that were significant in previous iterations.
- Quantification of newcomers in the list S.

For this, we use a uniform scalar quantizer defined relative to threshold Tj. The quantization ranges is between 0 and the maximum value of the coefficients $2T_0$. For the first iteration significant values are in the interval [T0, 2T0]. Quantification step assigns:

- The bit "0" for the coefficients belonging to the first half of the interval. [T_0, $3T_0/2$[(see Figure 2.10.a).
- And the bit "1" for the coefficients belonging to the second half [$3T_0/2$, $2T_0$] (see Figure 2.10.a).

For the second iteration, we perform a refinement uncertainty intervals by dividing by two. Thus, the bit "0" is assigned to the coefficients belonging to the first half of the interval, while the "1" bit is assigned to the second half. For newcomers, the coefficient of which is the interval [T 1, T 0 [, only performs quantization bit "0" or "1" according to the two intervals [T 1, 3T half [or [3T half, 0 T [successively (see Figure 2.10.b). This is repeated for each stage, performing a refinement uncertainty intervals by dividing by two.

In the test matrix of our example (see Figure 2.8), the coefficients found significant for the first iteration are: {63, -34, 49, 47}. The threshold T 0 = 32, these coefficients will be quantified as follows (see Figure 2.12):



Step 1: quantification

Step2: refinement and quantification

Figure 3.5: Principle of quantification and refinement

To summarize the algorithm, it starts by a forward wavelet transform. Then, the threshold is calculated where it follows by testing the significance using the principle of zerotree that is called the dominant pass. A subordinate pass is done where a quantification and refinement phase take place; an entropy coding is done simultaneously. If the compression rate is not achieved and all the coefficients are tested , the threshold is reduced by half and the whole process starts from the significance test using the new threshold. When the compression rate is achieved, a bitstream is outputted for transmission or storage to be decoded by the decoder.

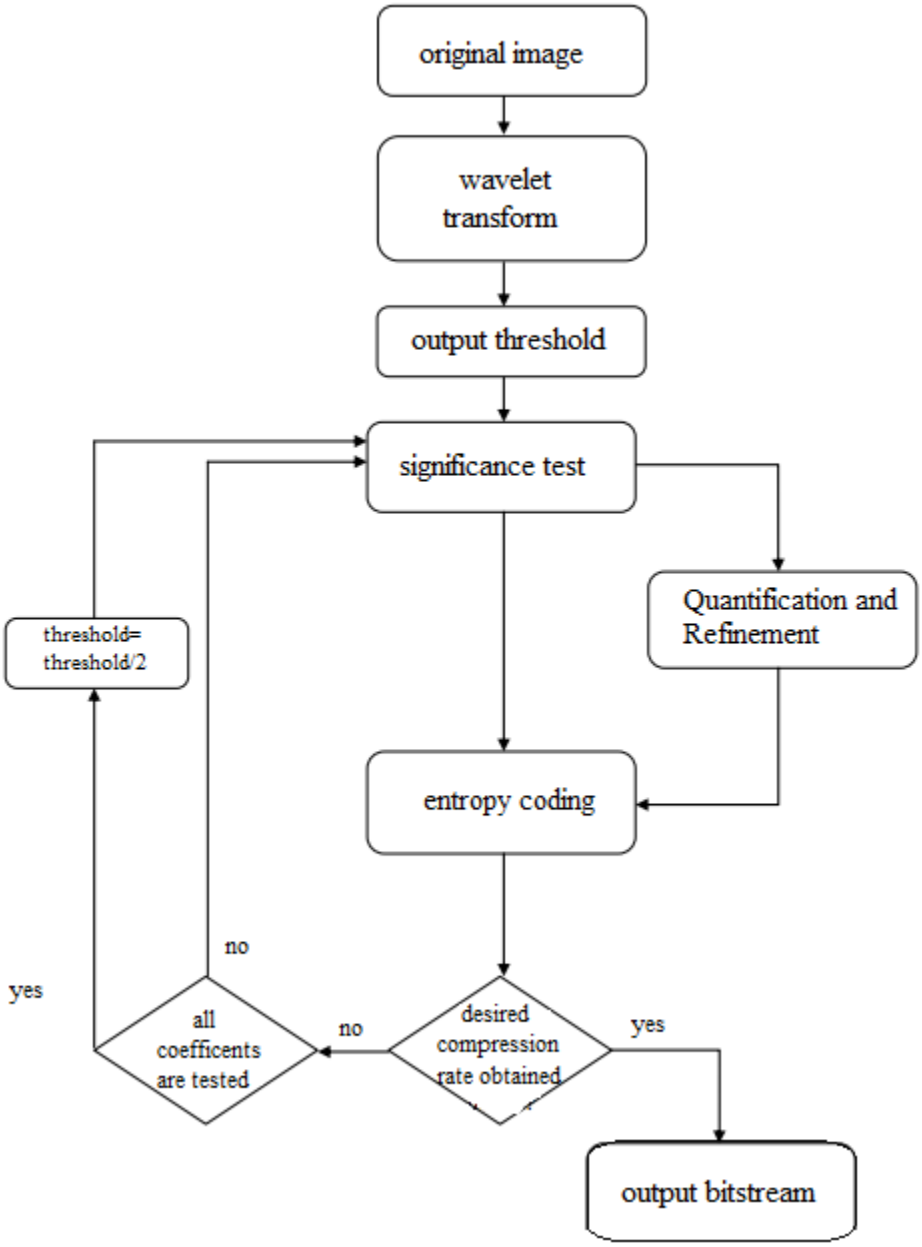


Figure 3.6: EZW algorithm flow chart

3.3. SPIHT

SPIHT algorithm is a method for image compression that uses wavelet transform. SPIHT method was developed to optimize a progressive image transmission, and to compress them. The most important feature of this algorithm is that, at any stage of the decoding, the quality of the displayed image gets better and better according to the amount of the information received. Another distinctive feature of the algorithm is that it also uses embedded coding.

3.3.1. Description of the algorithm:

Wavelet transform coefficients $C_{i,j}$ (or transform coefficients). The principle is based on the observation that the most significant bits of the binary representations of integers tend to be ones where these numbers are close to the maximum values. For example, a 10-bit binary number (1 | 0 0000 0000), the tenth bit has a weight of 32,768, and the first bits have 1, 2, 4 and 8 respectively. This suggests that the high-order bits contain the most significant piece of information, and should be sent to (or stored) first.

Progressive transmission method in SPIHT uses two principles: sorting and sending the most important bits. It sorts (organizes) the coefficients and first sends the most significant bits of these coefficients. To simplify the description of the basis of this method, we assume that the sorted information is directly transmitted to the decoder; in the next section discusses an efficient algorithm to encode the information.

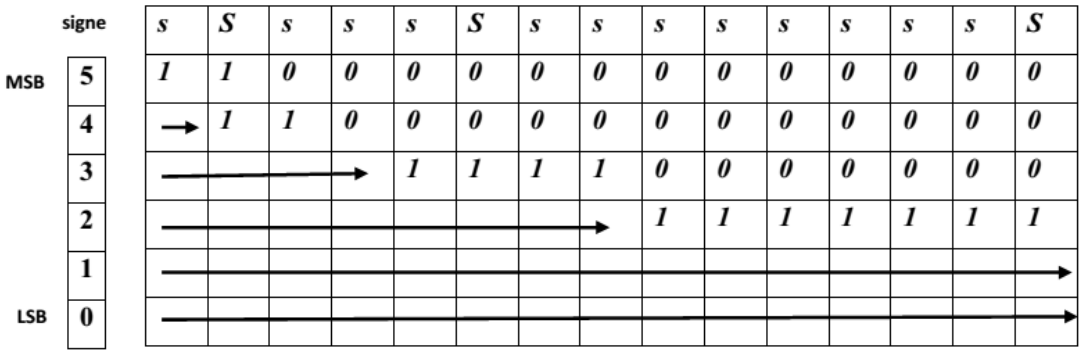


Figure 3.7: Binary representation of the magnitude-ordered coefficients

The procedure described above is very simple, since it assumes that the coefficients were sorted (ordered) before its start. In principle, the image may consist of million coefficients, and sorting can be very slow procedure. Instead of sorting the coefficients SPIHT algorithm sorts by comparing each time a coefficient value to the two values (thresholds), and each result of comparison, is simply put as yes/no. Therefore, if the encoder and decoder use the same algorithm for sorting, the encoder can just send the comparison sequence results to the decoder, and the latter will duplicate the work of the encoder. This is true not only for sorting, but also for any algorithm based on comparisons or any principle of branching.

The main task of sorting phase at each iteration is to identify the factors that satisfy the inequality $2^n \le |c_{i,j}| < 2^{n+1}$. This task is divided into two parts. For a given value *n* If the coefficient $c_{i,j}$ satisfies the inequality $|c_{i,j}| \ge 2^n$, It is said to be significant. Otherwise it is called insignificant. The first step will be relatively few significant coefficients, but the number will increase from iteration to iteration, since the number of *n* will be decreasing. Sort to determine which significant coefficients satisfy the second $|c_{i,j}| < 2^{n+1}$ inequality coordinates and pass them to the decoder. This is an important part of the algorithm used in the SPIHT. The encoder divides all the coefficients for a number of sets of T_k and performs a materiality test $\max_{(i,j)\in T_k} |c_{i,j}| \ge 2^n$?

To make clear the relationship between magnitude comparisons and message bits, the function (4.16) is used, to indicate the significance of a set of coordinates *T*. To simplify the notation of single pixel sets, the expression $S_n(\{(i, j)\})$ is written as $S_n(i, j)$.

$$S_n(T) = \begin{cases} 1, & \max_{(i,j)\in T_k} |c_{i,j}| \ge 2^n, \\ 0, & \text{otherwise} \end{cases} \qquad (4.16)$$

3.3.2. SPIHT Algorithm

The SPIHT coding algorithm is given next [17]:

O(i,j): set of coordinates of all offspring of node (i,j); *children only*

D (i,j): set of coordinates of all descendants of node (i,j); *children, grandchildren, great-grand, etc.*

H (i,j): set of all tree roots (nodes in the highest pyramid level); *parents*

L (i,j): D (i,j) – O(i,j) (all descendants except the offspring); grandchildren, great-grand, etc

1. Initialization: output $n = \log_2 (\max_{(i,j)}\{|c_{i,j}|\})$

set the LSP as an empty list, and add the coordinates (i, *j) that belongs to H* to the LIP, and only those with descendants also to the LIS, as type A entries.

2. Sorting Pass:

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

2.1.1) output S_n (i, j);

2.1.2) if S_n(i, *j)* = 1 then move (i, *j)* to the LSP and output the sign of $c_{i,j}$;

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

2.2.1) if the entry is of type A then

- output $S_n(D(i, j))$;
- if $S_n(D(i, j)) = 1$ then

* for each $(k, 1)$ belongs to $O(i, j)$ do:

output $S_n(k, 1)$;

if S.(k, 1) =1 "then add $(k, 1)$ to the LSP and output the sign of $c_{k,l}$

if $S_n(k, 1) = 0$ then add $(k, 1)$ to the end of the LIP;

* if $L(i, j) \neq 0$ then move (i, j) to the end of the LIS, as an entry of type B, and go to Step 2.2.2); otherwise, remove entry (i, j) from the LIS;

2.2.2) if the entry is of type B then

- output $S_n(L(i, j))$;
- if $S_n(L(i, j)) = 1$ then

* add each $(k. 1)$ belongs to O(i, j) to the end of the LIS as an entry of type A;

* remove (i, j) from the LIS.

3. Refinement Pass: for each entry (i,j) in the LSP, except those included in the last sorting pass (i.e., with same n), output the nth most significant bit of $|c_{i,j}|$;

4. Quanflzation-Step Update: decrement n by 1 and go to Step 2.

The algorithm stops at the desired rate or distortion. Normally, good quality images can be recovered after a relatively small fraction of the pixel coordinates are transmitted.

The fact that this coding algorithm uses uniform scalar quantization may give the impression that it must be much inferior to other methods that use non uniform and/or vector quantization. However, this is not the case: the ordering information makes this simple quantization method very efficient. On the other hand, a large fraction of the "bit-budget" is spent in the sorting pass, and it is there that the sophisticated coding methods are needed.

To better understand the algorithm the next figures visualize both the sorting pass and the refinement pass:

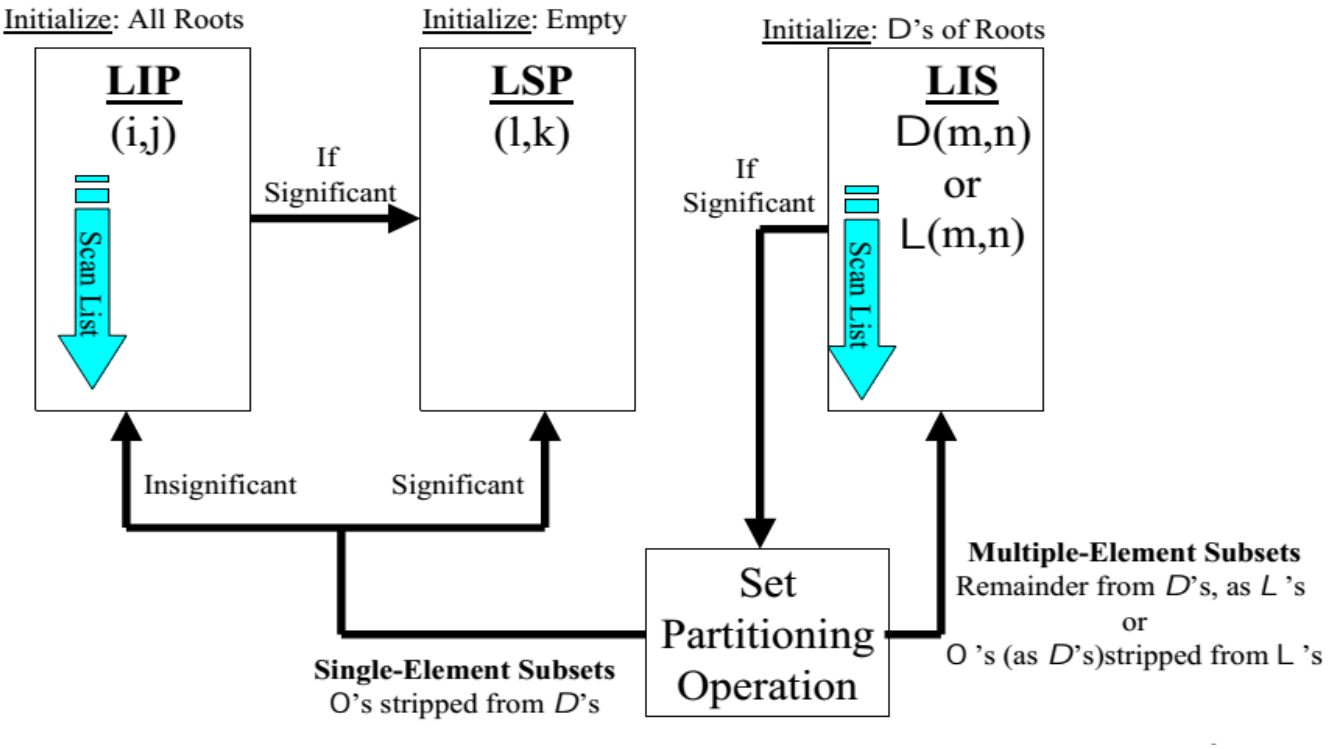


Figure 3.8. visualization of the refinement pass steps

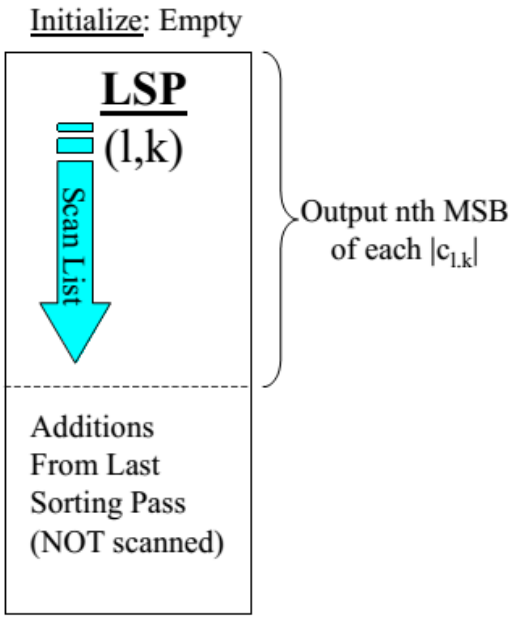


Figure 3.9. visualization of the sorting pass steps

3.4. Paralle computing

Parallel computing takes advantage of multicore desktop machines and clusters. It can take advantage of up to 8 cores there. Parallel programs can be run interactively or in batch. The basic concept of a parfor-loop is the same as the standard for-loop the algorithm executes a series of statements (the loop body) over a Range of values. Part of the "parfor" body is executed on a client (where the parfor is issued) and parts are executed in parallel on workers (A CPU core). The number of workers differ from CPU to another but nowadays with the new technologies such as Core2Duo and i3 and i7, CPUs have at least 2 cores.

The Necessary data parfor Operates on All which is felt from the customer to workers, Where MOST of the computation happens, and the results are felt back to the customer and pieced together. Because several workers can be on the same computing concurrently loop, a parfor-loop can provide significantly better performance than the analogous for-loop.

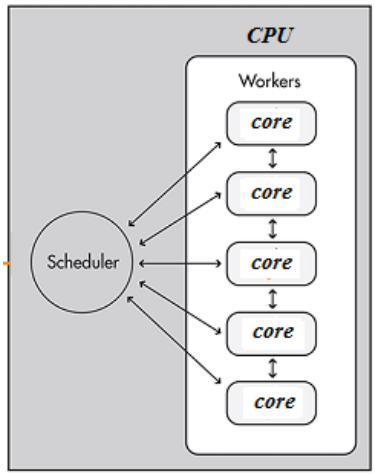


Figure 3.10: the principle of workers in parallel computing

3.5. Proposed algorithms B-EZW and B-SPIHT

Due to the slow nature of both EZW and SPIHT, we sat to use blocks and parallel computing in order to speed the compression process. Intuitively, in the conception phase, the blocks that are used supposed to be derived from the image itself, i.e. dividing the original image into blocks and apply EZW or SPIHT on the sub-blocks. As we will show in chapter 4 and the annex, this method does not yield good results, as block artifacts are clearly visible, even in the cases where high thresholds are applied. The diagram of this method is depicted in figure 3.11 with an example of four divisions, along with the structure of the classical methods.

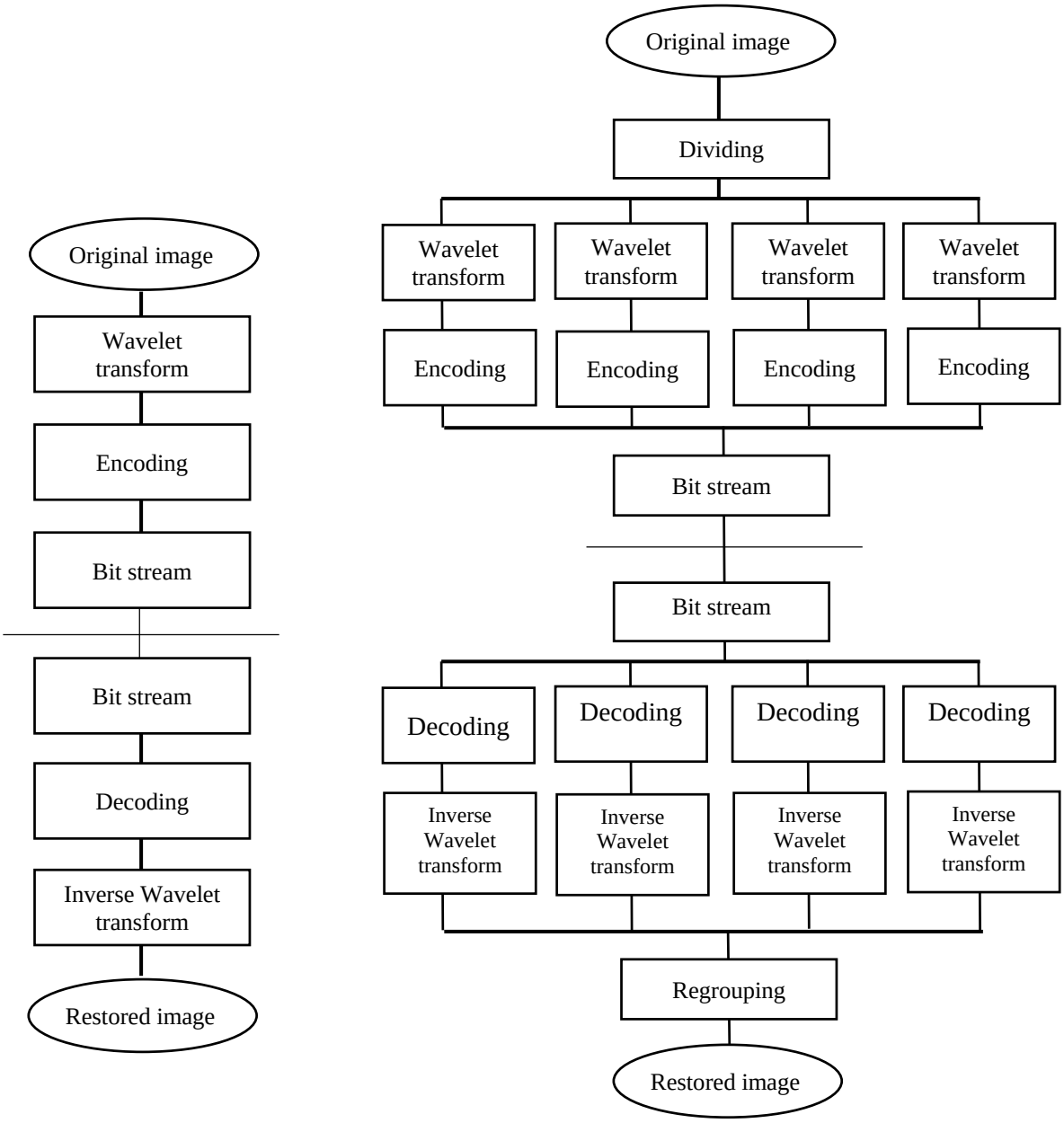


Figure 3.11: (a) EZW/SPIHT structure. (b) the structure of the first method

From the diagram (b) we see that the division happens at the level of the original image. The difference between this method and the original methods resides in multiple wavelet transforms, multiple coding and decoding. As we mentioned before this method resulted in very low PSNR when compared to the original methods, in chapter 4 we will examine it more thoroughly. In addition to low PSNR, the compressed images yielded by this method contain ferocious block artifacts; we give examples of them in the annex.

In order to overcome the flaws of that method, we proposed another one that we named B-EZW and B-SPIHT, the “B” in the name of the proposed algorithms stands for “Block”. The method is based on dividing the wavelet coefficients into blocks; each block is considered as a separate image from the perspective of the original methods. We apply the compression algorithms EZW and SPIHT, while each of the blocks is processed simultaneously in the CPU cores. The following diagram explains the workings of B-EZW and B-SPIHT.

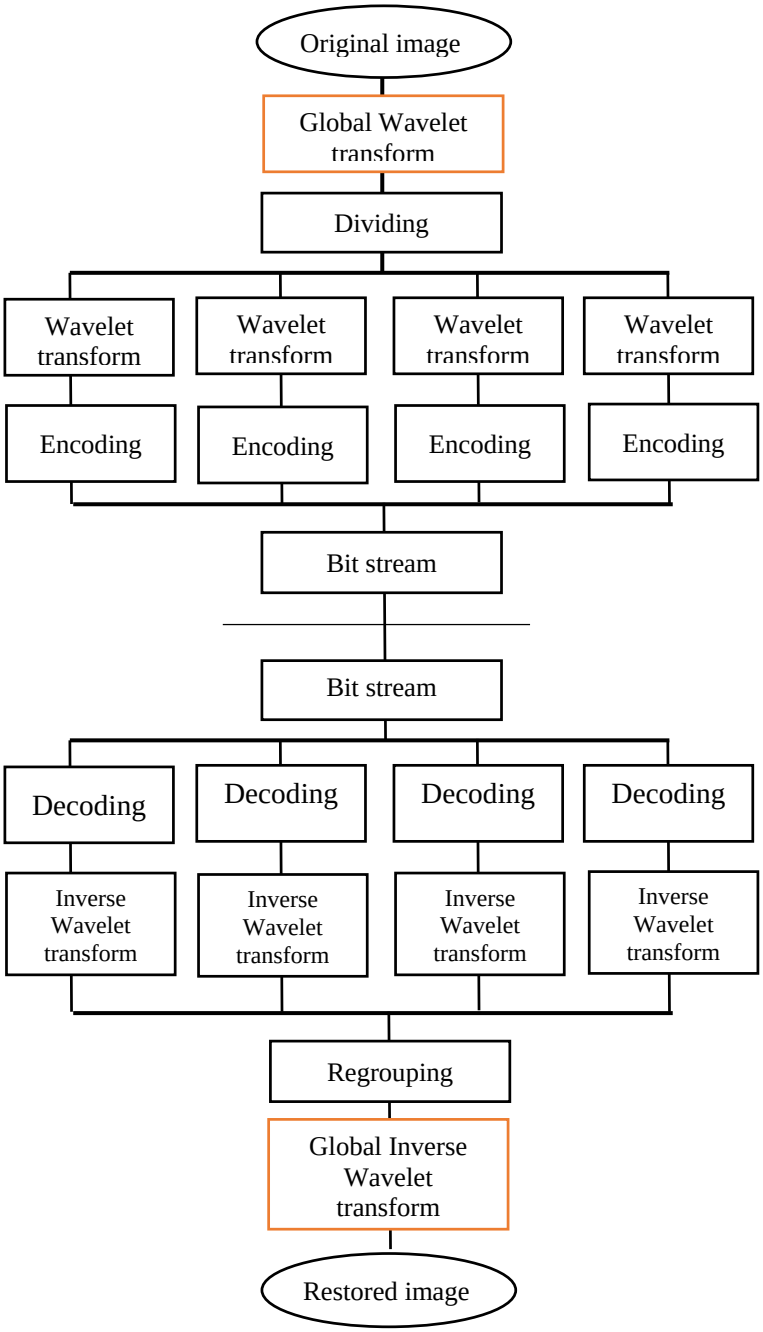


Figure 3.12:B-EZW /B-SPIHT structure

Both algorithms B-EZW and B-SPIHT have the same principle, except for the encoding and decoding, where we apply a local EZW and a local SPIHT respectively. As mentioned

above the parallel computing is the essential property of the algorithms to reduce the computation time greatly as we will see in the results later.

It is important to note that the number of cores differ from a CPU to another, there for a function that detects the number of cores in the CPU used before the execution. Figure 4.6 shows the global difference between the EZW/SPIHT and B-EZW/B-SPIHT.

3.6. Examples of EZW/B-EZW and SPIHT/B-SPIHT

Consider a simple three-level wavelet transform image 8x8 items. We run the EZW/SPIHT algorithms on this test matrix, this example already exists in the literature, it figured in Shapiro's paper on EZW. Matrix values of the wavelet transform is presented below. In order to explain the differences between EZW/B-EZW and SPIHT/B-SPIHT we will examine the execution steps on this example matrix.

The results shown in this section are only to show the execution of the algorithms, not as a definitive test, this example matrix is relatively small to have a final opinion about the algorithms. As we will see later, the choice of the wavelet filter and the more the image is bigger the better the algorithms perform.

63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 3.13: example matrix

3.6.1. EZW Execution

Since most absolute values of the coefficients in the matrix equal 63, we can choose an initial threshold of period (31.5; 63]. Let the initial threshold be 32. Table 1 describes the

treatment of coefficients in the first dominant pass. The following are comments are from table 3.1:

Comment	Subband	Coefficient value	Symbol	Reconstruction Value
(1)	LL3	63	POS	48
	HL3	-34	NEG	-48
(2)	LH3	-31	IZ	0
(3)	11H3	23	ZTR	0
	HL2	49	POS	48
(4)	HL2	10	ZTR	0
	HL2	14	ZTR	0
	HL2	-13	ZTR	0
	LH2	15	ZTR	0
(5)	LH2	14	IZ	0
	LH2	-9	ZTR	0
	LH2	-7	ZTR	0
(6)	HL1	7	z	0
	HL1	13	Z	0
	HL1	3	Z	0
	HL1	4	Z	0
	LH1	-1	z	0
(7)	LH1	47	POS	48
	LH1	-3	Z	0
	LH1	-2	z	0

Table3.1: processing of the first dominant pass, threshold 32

1) the coefficient 63 is greater than the threshold of 32 and positive. Therefore, p symbol is given. When decoding this symbol decoder replaces it with the value of the middle of the interval [32.64), which is 48.

2) Although the factor 31 is close relatively to the threshold of 32, one of its offspring value two levels below, the band LH1, is equal to 47. Therefore, this cofficient is replaced by the symbol isolated zero IZ.

3) The value of 23 is less than 32 and also all child values (3, -12, -14.8) in the band HH2 and all coefficients in the band HH1 are insignificant. Therefor, it is considered as a zerotree and all its decendents are coded.

4) the same case as in comment 3 apply for the coefficient 10, even one its decendents is equal to 12 which is greater than 10 but as long as it is less than 32 it will not b concidered.

5) for the coefficient 10 in HL2, it will be treated the same as the coeffiecent in comment 2, for one of its decendents is equal 47 where a symbol isolated zero.

6) Since the coefficients [7 13 3 4] do not belong to a zerotree and they are all less than the threshold then all of them are coded as zeros.

7) coefficient 47 is greater than the threshold therefor a symbol P is given.

The first dominant passage uses a threshold of 32, four significant coefficient are found. Before the first subordinate pass all the of these valus are in the interval [32.64). The subordinate pass will clarify these values and carry them to the interval [32.48) encode them with symbol 0, or interval [48.64) encode them with symbol 1. That is, the limit for a decision is the 48. Procedure at the first contractors passage illustrated in tabl.8.2.

Coefficient Magnitude	Symbol	Reconstruction Magnitude
63	1	5 6
34	0	4 0
49	1	5 6
47	0	4 0

Table 3.2: processing of the first subordinate pass

The first factor is equal to 63 and placed in the upper interval center, which is 56. The second factor is equal to 34 and placed in the lower interval center. The third factor is 49 in the upper interval, and the fourth factor of 47 in the lower interval.

A matrix that can be reproduced by the decoder, based on the data of the first dominant and subordinate passes below:

56	-40	56 0	0 0 0 0
0	0	0 0	0 0 0 0
0 0		0 0	0 0 0 0
0 0		0 0	0 0 0 0
0 0 0 40			0 0 0 0
0 0 0 0			0 0 0 0
0 0 0 0			0 0 0 0
0 0 0 0			0 0 0 0

Figure 3.14. Matrix reproduced by the decoder after the first passes

Processing continues on the second dominant passes with the new reduced threshold 16. During this passage only factors that have been identified as insignificant in the first pass. In addition, the factors that have previously been identified as significant; are replaced by zeros. The modified wavelet transformation matrix values of the second dominant passage is shown below:

0	0	0	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	0	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Figure 3.15: matrix to be processed by the second pass, threshold 16

In the second dominant pass the coefficient -31 of LH3 encoded as a significant negative, the coefficient in 23 in subband HH3 is significant positive. All the three coefficients in subband HL2, which had not previously been identified as significant, are all coded as the roots of zero trees. Similarly coded as zero tree roots of all four coefficients in subband LH2 and all four coefficients in subband HH2. This second dominant passage ends.

List of the second subordinate contains coefficients (63, 34, 49, 47, 31, 23), which in this passage are represented in three intervals [48.64) [32.48) [16.32) each with a width equal to 16. A matrix that can be reproduced by the decoder, based on data of the first and second dominant and subordinate passes, is shown below.

60	-36	52	0	0	0	0	0
-28	20	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	44	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure 3.16: matrix reproduced by the decoder after the second passes

3.6.2. B-EZW

In this section we will explain the execution of B-EZW for the previous matrix. For this example, we will take a size block of 4x4. B-EZW as well B-SPIHT can work with any block size as an input from the user.

The algorithm starts by deviding the matrix into 4 sub-matrices, each matrix is considered as an image and a wavelet transform is applied on them again, then a local EZW algorithm applied to each one of the wavelet coefficients submatrice. For each of the new matrices (i.e. after the wavelet transforms) the threshold is chosen individually. The decoder will receive 4 matrices and the block size, these information are enough for the decoder to reconstitute the coded image.

In what follows, an explanation is given to coding the whole matrix and the submatrices, figure 3.13 shows the deviding of the original matrix and the wavelet transform coefficients of each one. To avoid repetition, in all the following tables, the comment section will refer to the similar cases in the previous example.

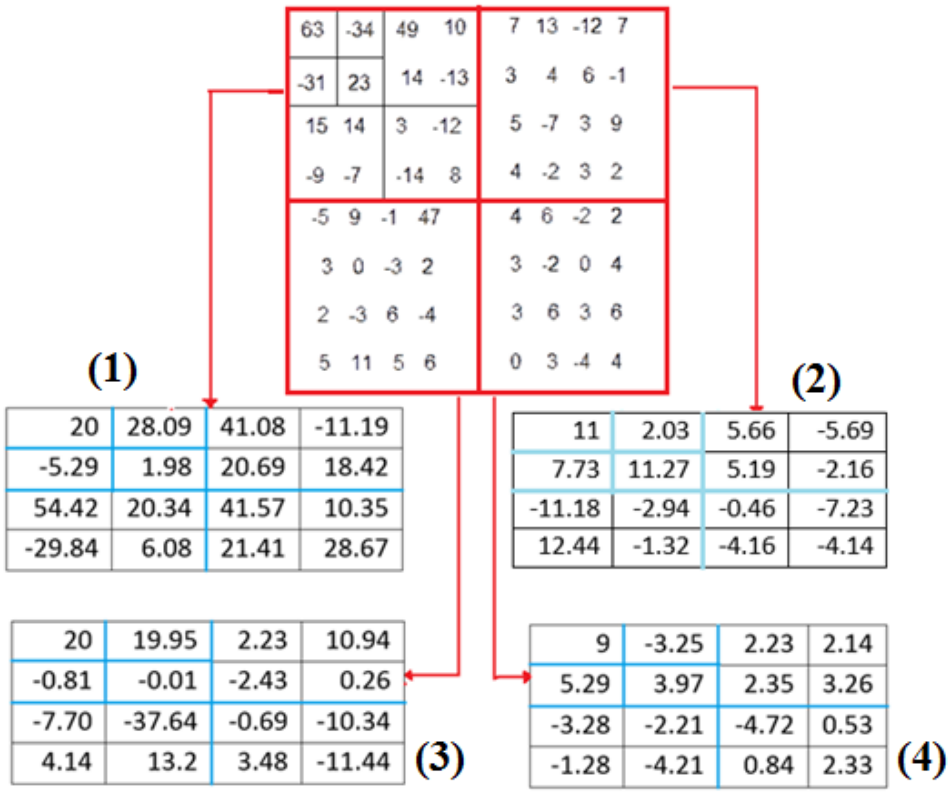


figure 3.17: deviding and the wavelet transform of the example matrix

The EZW coding for first matrix is shown in the following table.

Comment	Subband	Coefficient Value	Symbol	Reconstruction Value
(2)	LL3	20	IZ	0
	HL3	28.08	IZ	0
	LH3	-5.29	IZ	0
	LH3	1.98	IZ	0
(1)	HL2	41.08	POS	48
	HL2	-11.18	Z	0
	HL2	20.69	Z	0
	HL2	18.41	Z	0
	LH2	54.41	POS	48
	LH2	20.34	Z	0
	LH2	-29.83	Z	0
	LH2	6.08	Z	0
	HL1	41.56	POS	48
	HL1	10.35	Z	0
	HL1	21.40	Z	0
	HL1	28.67	Z	0

Table 3.3: processing of the first dominant pass for the first matrix, threshold 32

The following table shows the EZW coding of the second matrix:

Comment	Subband	Coefficient Value	Symbol	Reconstruction Value
(2)	LL3	11	IZ	0
(3)	HL3	2.035	ZTR	0
	LH3	7.734	ZTR	0
	LH3	11.27	ZTR	0

Table 3.4: processing of the first dominant pass for the second matrix, threshold 32

The following table shows the EZW coding of the third matrix:

Comment	Subband	Coefficient Value	Symbol	Reconstruction Value
(2)	LL3	20	IZ	0
(3)	HL3	19.95	ZTR	0
(2)	LH3	-0.8141	IZ	0
(3)	LH3	-0.01	ZTR	0
	LH2	-7.701	Z	0
(5)	LH2	-37.64	N	-48
	LH2	4.142	Z	0
	LH2	13.2	Z	0

Table 3.5: processing of the first dominant pass for the third matrix, threshold 32

The EZW coding is shown in the following table the last matrix.

Comment	Subband	Coefficient Value	Symbol	Reconstruction Value
(2)	LL3	9	IZ	0
(3)	HL3	-3.256	ZTR	0
(3)	LH3	5.292	ZTR	0
(3)	LH3	3.977	ZTR	0

Table 3.6: processing of the first dominant pass for the fourth matrix, threshold 32

3.6.3. SPIHT

(1) The algorithm starts by an empty LSP, LIS with the sets {(0,1)A,(1,0)A(1,1)A} and LIP with the following coefficients {(0,0),(0,1),(1,0),(1,1)}. The initial threshold is set to 32. The coefficient (0,0) is not considered a root.

(2) SPIHT begins coding the significance of the individual pixels in the LIP. When a coefficient is significant it is moved to the LSP, and its sign is also coded. The notation 1+ and 1– is used to indicate the coding of a sign bit.

Comm.	Pixel or Set Tested	Output Bit	Action	Control Lists
1				LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}* LSP = ∅
2	(0,0)	1+	(0,0) to LSP	LIP = *{*(0,1),(1,0),(1,1)*}* LSP = *{*(0,0)*}*
	(0,1)	1–	(0,1) to LSP	LIP = *{*(1,0),(1,1)*}* LSP = *{*(0,0),(0,1)*}*
	(1,0)	0	none	
	(1, 1)	0	none	
3	*D*(0,1)	1	test offspring	LIS = *{***(0,1)A**,(1,0)A,(1,1)A*}*
	(0,2)	1+	(0,2) to LSP	LSP = *{*(0,0),(0,1),(0,2)*}*
	(0,3)	0	(0,3) to LIP	LIP = *{*(1,0),(1,1),(0,3)*}*
	(1,2)	0	(1,2) to LIP	LIP = *{*(1,0),(1,1),(0,3),(1,2)*}*
4	(1,3)	0	(1,3) to LIP	LIP = *{*(1,0),(1,1),(0,3),(1,2),(1,3)*}*
5			type changes	LIS = *{*(1,0)A,(1,1)A,(0,1)B*}*
	D(1,0)	1	test offspring	LIS = *{***(1,0)A**,(1,1)A,(0,1)B*}*
	(2,0)	0	(2,0) to LIP	LIP = *{*(1,0),(1,1),(0,3),(1,2),(1,3),(2,0)*}*
	(2,1)	0	(2,1) to LIP	LIP = *{*(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1)*}*
	(3,0)	0	(3,0) to LIP	LIP = *{*(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0)*}*
	(3,1)	0	(3,1) to LIP	LIP = *{*(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1)*}*
			type changes	LIS = *{*(1,1)A,(0,1)B,(1,0)B*}*
6	*D*(1,1)	0	none	LIS = *{***(1,1)A**,(0,1)B,(1,0)B*}*
7	*L*(0,1)	0	none	LIS = *{*(1,1)A,**(0,1)B**,(1,0)B*}*
8	*L*(1,0)	1	add new sets	LIS = *{*(1,1)A,(0,1)B,(2,0)A,(2,1)A,(3,0)A,(3,1)A*}*
9	*D*(2,0)	0	none	LIS = *{*(1,1)A,(0,1)B,**(2,0)A**,(2,1)A,(3,0)A,(3,1)A*}*
1 0	*D*(2,1)	1	test offspring	LIS = *{*(1,1)A,(0,1)B,(2,0)A,**(2,1)A**,(3,0)A,(3,1)A*}*
	(4,2)	0	(4,2) to LIP	LIP = *{*(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2)*}*
	(4,3)	1+	(4,3) to LSP	LSP = *{*(0,0),(0,1),(0,2),(4,3)*}*
	(5,2)	0	(5,2) to LIP	LIP = *{*(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2),(5,2)*}*
	(5,3)	0	(5,3) to LIP	LIP = *{*(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2),(5,2),(5,3)*}*
1 1			(2,1) removed	LIS = *{*(1,1)A,(0,1)B,(2,0)A,(3,0)A,(3,1)A*}*
1 2	*D*(3,0)	0	none	LIS = *{*(1,1)A,(0,1)B,(2,0)A,**(3,0)A**,(3,1)A*}*
	D(3,1)	0	none	LIS = *{*(1,1)A,(0,1)B,(2,0)A,(3,0)A,**(3,1)A***}*
1 3				LIS = *{*(1,1)A,(0,1)B,(2,0)A,(3,0)A,(3,1)A*}* LIP = *{*(1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1),(4,2),(5,2) ,(5,3)*}* LSP = *{*(0,0),(0,1),(0,2),(4,3)*}*

Table 3.7: processing of SPIHT first pass

(3) Next the algorithm tests sets, following the entries in the LIS. In this example D(0,1) is the set of 20 coefficients of offsprings in the tree. Because D(0,1) is significant SPIHT next tests the significance of the four offspring {(0,2), (0,3), (1,2), (1,3)}.

(4) After all offspring are tested, the set (0,1) is moved to the end of the LIS, and its type changes from ‘A’ to ‘B’, meaning that the new LIS entry changed from D(0,1) to L(0,1) (i.e., from set of all descendants to set of all descendants minus offspring).

(5) Same procedure as in comments (3) and (4) applies to set D(1, 0). Note that even though no offspring of (1,0) is significant, D(1, 0) is significant because L(1, 0) is significant.

(6) Since D(1,1) is insignificant, no action need to be taken. The algorithm moves to the next element in the LIS.

(7) The next LIS element, (0,1), is of type ‘B’, and thus L(0,1) is tested. the co-ordinate (0,1) was moved from the beginning of the LIS in this pass. It is now tested again, but with another interpretation by the algorithm.

(8) Same as above, but L(1, 0) is sigificant, so the set is partitioned in D(2, 0), D(2,1), D(3, 0), and D(3,1), and the corresponding entries are added to the LIS. At the same time, the entry (1,0)B is removed from the LIS.

(9) The algorithm keeps evaluating the set entries as they are appended to the LIS.

(10) Each new entry is treated as in the previous cases. In this case the offspring of (2,1) are tested.

(11) In this case, because G(2,1) = ∅ (no descendant other than offspring), the entry (2,1)A is removed from the LIS (instead of having its type changed to ‘B’).

(12) Finally, the last two entries of the LIS correspond to insignificant sets, and no action is taken. The sorting pass ends after the last entry of the LIS is tested.

(13) The final list entries in this sorting pass form the initial lists in the next sorting pass, when the threshold value is 16.

Without using any other form of entropy coding, the SPIHT algorithm used 29 bits in this first pass.

3.6.4. B-SPIHT

The B-SPIHT has the same principle as the B-EZW, although they differ in the coding of the submatrices. In what follows, we will see how the B-SPIHT codes the example matrix. First, it starts with calculating the threshold, then proceeds with coding each one in order. The decoder recieves the final bits and can restitute the orginal matrix by same principle as B-EZW.

As before, the comments in the following tables are the same as the table from the execution of the original algorithm (table 3.4).

	Pixel or Set Tested	Output Bit	Action	Control Lists
				LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}* LSP = *Ø*
	(0,0)	0	none	LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}*
	(0,1)	0	none	LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}*
	(1,0)	0	none	
	(1, 1)	0	none	
	D(0,1)	1	test offspring	LIS = *{***(0,1)A**,(1,0)A,(1,1)A*}*
	(0,2)	1+	(0,2) to LSP	LSP = *{*(0,2)*}*
	(0,3)	0	(0,3) to LIP	LIP = *{*(0,0),(0,1),(1,0),(1,1),(0,3)*}*
	(1,2)	0	(1,2) to LIP	LIP = *{*(0,0),(0,1),(1,0),(1,1),(0,3),(1,2)*}*
	(1,3)	0	(1,3) to LIP	LIP = *{*(0,0),(0,1),(1,0),(1,1),(0,3),(1,2),(1,3)*}*
			type changes	LIS = *{*(1,0)A,(1,1)A,(0,1)B*}*
	D(1,0)	1	test offspring	LIS = *{***(1,0)A**,(1,1)A,(0,1)B*}*
	(2,0)	1+	(2,0) to LSP	LSP = *{*(0,2), (2,0) *}*
	(2,1)	0	(2,1) to LIP	LIP = *{*(0,0),(0,1), (1,0),(1,1),(0,3),(1,2),(1,3),(2,1)*}*
	(3,0)	0	(3,0) to LIP	LIP = *{*(0,0),(0,1), (1,0),(1,1),(0,3),(1,2),(1,3),(2,1),(3,0)*}*
	(3,1)	0	(3,1) to LIP	LIP = *{*(0,0),(0,1), 1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1)*}*
			type changes	LIS = *{*(1,1)A,(0,1)B,(1,0)B*}*
	D(1,1)	1	test offspring	LIS = *{*(1,1)A,(0,1)B,(1,0)B*}*
	(4,2)	0	(2,2) to LSP	LSP = *{*(0,2), (2,0), (2,2)*}*
	(2,3)	1+	(2,3) to LIP	LSP = *{*(0,0),(0,1),(0,2),(4,3)*}*
	(3,2)	0	(3,2) to LIP	LIP = *{*(0,0),(0,1), 1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1), (3,2) *}*
	(3,3)	0	(3,3) to LIP	LIP = *{*(0,0),(0,1), 1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1), (3,2), (3,3)*}*
				LIS = *{*(1,1)A,(0,1)B,(1,0)B*}* LIP = *{*(0,0),(0,1), 1,0),(1,1),(0,3),(1,2),(1,3),(2,0),(2,1),(3,0),(3,1), (3,2), (3,3)*}* LSP = *{*(0,2), (2,0), (2,2)*}*

Table 3.8: First pass of B-SPIHT of the first matrix

After the algorithm finishes the first submatrix it prooceeds with the second one, table 3.9 shows the sorting pass of the latter.

	Pixel or Set Tested	Output Bit	Action	Control Lists
				LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}* LSP = *Ø*
	(0,0)	0	none	LIP = *{*(0,1),(1,0),(1,1)*}* LSP = *{*(0,0)*}*
	(0,1)	0	none	LIP = *{*(1,0) ,(0,1),(1,1)*}* LSP = *{*(0,0) *}*
	(1,0)	0	none	
	(1, 1)	0	none	LIP = *{*(1,0) ,(0,1) *}* LSP = *{*(0,0) ,(1,1) *}*
	D(0,1)	0	none	LIS = *{***(0,1)A**,(1,0)A,(1,1)A*}*
	D(1,0)	0	none	LIS = *{***(1,0)A**,(1,1)A,(0,1)A*}*
	D(1,1)	0	none	LIS = *{***(1,1)A**,(0,1)A,(1,0)A*}*
				LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}* LSP = *Ø*

Table 3.9: First pass of B-SPIHT of the second submatrix

Next comes the third sub-matrix sorting pass shown in table 3.10:

	Pixel or Set Tested	Output Bit	Action	Control Lists
				LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}* LSP = *Ø*
	(0,0)	0	none	LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}*
	(0,1)	0	none	LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}*
	(1,0)	0	none	
	(1, 1)	0	none	
	D(0,1)	0	none	LIS = *{***(0,1)A**,(1,0)A,(1,1)A*}*
	D(1,0)	1	test offspring	LIS = *{***(1,0)A**,(1,1)A,(0,1)A*}*
	(2,0)	0	(2,0) to LIP	LIP = *{*(0,0),(0,1),(1,0),(1,1),(2,0)*}*
	(2,1)	1-	(2,1) to LSP	LSP= {(2,0)}
	(3,0)	0	(3,0) to LIP	LIP = *{*(0,0),(0,1), (1,0),(1,1),(2,0),(3,0)*}*
	(3,1)	0	(3,1) to LIP	LIP = *{*(0,0),(0,1), (1,0),(1,1),(2,0),(3,0),(3,1)*}*
			type changes	LIS = *{*(1,1)A,(0,1)B,(1,0)B*}*
	D(1,1)	0	test offspring	LIS = *{*(1,1)A,(0,1)B,(2,0)A,**(2,1)A**,(3,0)A,(3,1)A*}*
				LIS = *{*(1,1)A,(0,1)B,(1,0)B*}* LIP = *{*(0,0),(0,1), (1,0),(1,1),(2,0),(3,0),(3,1)*}* LSP= {(2,0)}

Table 3.10: First pass of B-SPIHT of the third submatrix

Finally, the fourth submatrix sorting pass is shown in table 3.11:

	Pixel or Set Tested	Output Bit	Action	Control Lists
				LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}* LSP = ∅
	(0,0)	0	(0,0) to LSP	LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}*
	(0,1)	0	none	LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}*
	(1,0)	0	none	
	(1, 1)	0	none	
	D(0,1)	0	none	LIS = *{***(0,1)A**,(1,0)A,(1,1)A*}*
	D(1,0)	0	none	LIS = *{***(1,0)A**,(1,1)A,(0,1)A*}*
	D(1,1)	0	test offspring	LIS = *{*(1,1)A,(1,0)A, (0,1)A*}*
				LIS = *{*(0,1)A,(1,0)A(1,1)A*}* LIP = *{*(0,0),(0,1),(1,0),(1,1)*}* LSP = ∅

Table 3.11: First pass of B-SPIHT of the fourth submatrix

3.7. Conclusion

In this chapter we introduced the algorithms EZW and SPIHT these algorithms utilize the dependency property in the wavelet transform coefficients, the zerotree coding give good compression rate with the advantage of successive transmission. In addition, we introduced the parallel computing feature and our modifications while giving examples of execution. In the next chapter, we will see the experimental results on various test images.

Chapter IV: Results and discussion

Abstract

In this chapter, we will define the testing protocol that we used to validate our methods. We based our tests MAINLY on three images; three gray levels of size 512 * 512 coded on 8 bits. The metric used is: PSNR, compression ratio, the execution time. We as well will examine the advantage of using parallel computing.

We also try to show the interest of the proper choice of the wavelet transformation where we compared: the db45, bior3.3, bior4.4 bior6.8, rbio3.3 and coif5. We finally finish our study by comparing our methods with the coders, EZW, SPIHT and SPECK.

.

4.1. Introduction

After the learning about of the EZW and SPIHT algorithms and then later the proposed optimization, named B-EZW and B-SPIHT (see Chapter 3); in this chapter a detailed study of the results obtained, by the classic EZW, SPIHT and SPECK and by the proposed algorithms. The tests are applied on test images that are frequently used in the literature. Afterwards, a comparative study of all the algorithms is performed.

4.2. Validation parameters

Depending on the desired application, the compression algorithm must be able to check a number of quality criteria such as: the compression ratio, the degradation ratio and the speed of compression and decompression that is the computation time. These parameters characterize the coding efficiency. The next sections are written exactly like [1] and [5].

4.2.1. Compression ratio

The compression ratio is defined as the ratio between the total number of bits needed to represent the original information and the total number of bits to store as a binary file that results from the compression method:

$$RC\ (\%) = \frac{\textit{number of coded bits}}{\textit{number of bits of the original image}} \times 100$$

In practice, Rather, the flow is used to measure the power of a compaction method. The rate is expressed in bits per pixel:

$$RC\ (bpp) = \frac{\textit{number of coded bits}}{\textit{number of bits of the original image}}$$

4.2.2. Distorsion

The information lost between the original signal and the decoded signal at the end of the chain, is called distortion. The distortion measure that is most commonly used is the Mean Square Error (MSE). This criterion is computed as the average of squared differences between pixels of the reconstructed image and the corresponding pixels of the original image:

$$EQM = \frac{1}{n \times m} \sum_{i=1}^{n} \sum_{j=1}^{m} (f(i, j) - \hat{f}(i, j))^2$$

n, m : size of the image.

^*f:* The values of the intensities of the original image and the reconstructed image respectively.

The other endpoint, deducted directly from the MSE is the *signal / noise* ratio (SNR) given by:

$$SNR = 10 \log_{10} \left(\frac{1}{EQM}\right)$$

Is the image variance. SNR is measured by decibel (dB).

Another widely used criterion measured in dB is the peak signal / noise or PSNR (Peak Signal to Noise Ratio) defined relative:

$$PSNR\ (db) = 10 \log_{10} \left(\frac{\max(f(i,j))}{EQM}\right)$$

Where *max (f i)* is the maximum amplitude of the original image (ie 255 for coded 8-bit image).

4.2.3. Computation time

The constraint of time is a critical factor in evaluating the performance of any compression method, where here in our case is the essential parameter that distinct our algorithm, it is calculation of the time taken by the compression and decompression of images. This constraint is more or less imposed by the target application by compression (transmission or storage). Indeed, it would be a shame, in a communications application, the time saved by reducing the size of data to be transmitted is less than the time spent compression decompression. This quality will however be less critical in applications for data archiving.

4.3. Test Images

For testing our results, we used some standard test images that are widely used in the image compression literature. They vary in sizes that are 256x256, 512x512 and 1024x1024. It is important to note that all images of are 8-bit gray levels. The images which we used widely are the following images:

- Lena, Barbara and Goldhill; size 512x512 pixels (fig4.1).

The rest of the used images are:

- Man: 1024x1024 pixels (4.2)

Figure 4.1: Test images, from left Lena, to right Barbara

Figure 4.2: Test images, from left Goldhill, to right Man

4.4. Choice of the wavelet filter

The wavelet transform can be done using large available filters. In what follows we will see the influence of different types of filters for wavelet applied on the test images, using the SPIHT algorithm to determine which filter gives better performance in PSNR. Determining the best type of a filter for a given task requires a study by itself, since we are focusing on SPIHT algorithm, thus using only SPIHT for the choice of the filter. The filters used are db45, bior3.3, bior4.4 bior6.8, rbio3.3 and coif5. Because the performances are similar almost throughout the plots, we will show the performances from 0.8bbp to 1bbp for better visualization of the results

4.4.1. Results on image Lena

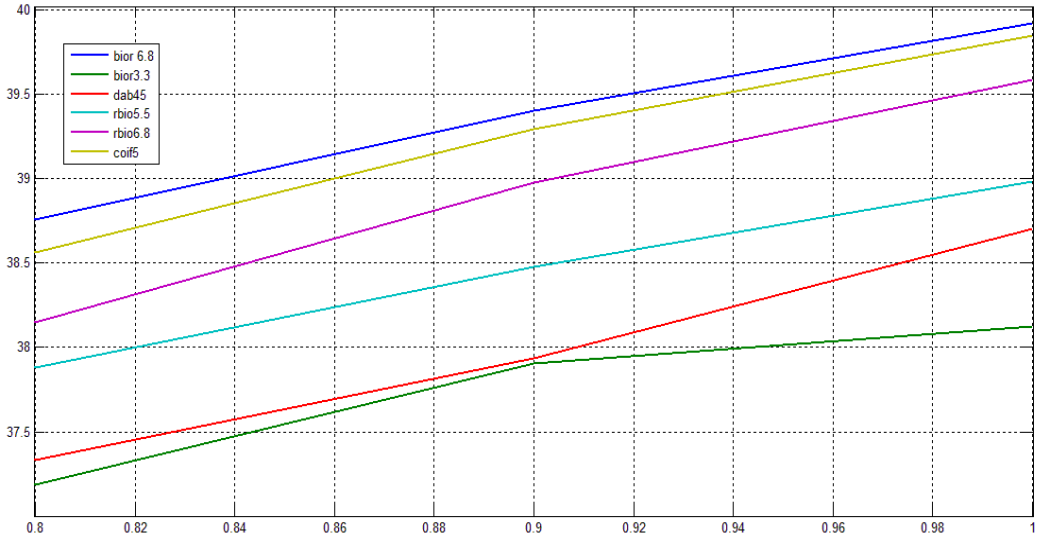


Figure 4.3: the influence of the filters on image Lena (rate vs PSNR)

4.4.2. Results on the mage Barbara

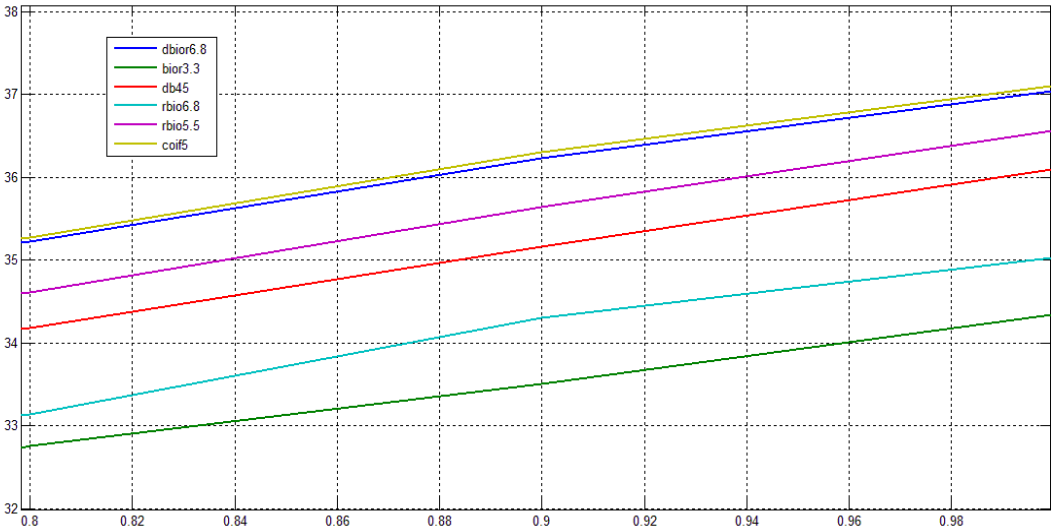


Figure 4.4: the influence of the filters on image Barbara (rate vs PSNR)

4.4.3: Test on image Goldhill

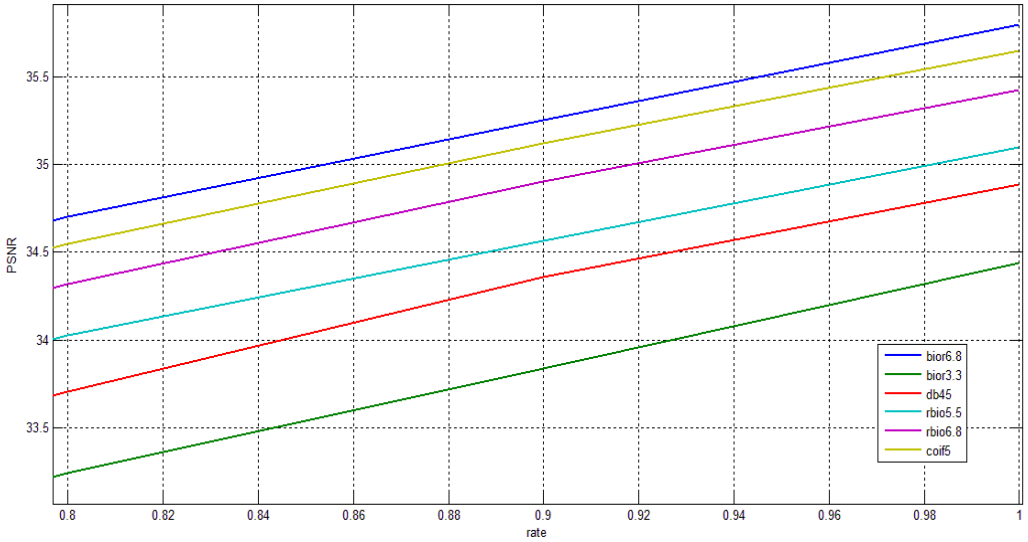


Figure 4.5: the influence of the filters on image Goldhill (rate vs PSNR)

After the executions, the filters bior6.8 and coif5 performed better than the other filters. Bior6.8 gave better performances on both images Lena and Goldhill, for this reason, we decided to use filter bior6.8 throughout all the computations.

4.5. THE OBAINTED RESULTS

Finally, we get to the results section; we will show the results in a detailed manner and demonstrate the influence of each aspect of our algorithm. The whole process is performed on a PC Intel i7, RAM = 8GB, using Matlab 2013 software. The results will be presented as follows:

1. Results of the first proposed method (see figure 3.11b in chapter 3)
2. The comparison between EZW and B-EZW (using 4, 16 and 64 divisions)
3. The comparison between SPIHT and B-SPIHT (using 4, 16 and 64 divisions)
4. The results obtained from SPECK algorithm
5. The overall comparison between EZW, B-EZW4, SPIHT, B-SPIHT4 and SPECK
6. Visual quality of the compressed images.
7. The influence of parallel computing on the computation time of the algorithms
8. The influence of the original images' size on B-SPIHT and B-EZW

4.5.1. Obtained results from the first proposed method

In this section, we explore the results obtained from the first method mentioned in chapter 3 section 5. We will observe the effect of the number of divisions applied on the original images. All the test images in this section are 512x512 pixels

4.5.1.1. EZW

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
64 divisions	**24.01 dB**	**27.58 dB**	**29.28 dB**	**30.20 dB**	**30.88 dB**	**31.33 dB**	**31.65 dB**	**31.94 dB**	**32.19 dB**	**32.40 dB**
16 divisions	**26.05 dB**	**29.40 dB**	**30.80 dB**	**31.79 dB**	**32.45 dB**	**32.90 dB**	**33.29 dB**	**33.57 dB**	**33.91 dB**	**34.15 dB**
4 divisions	**27.11 dB**	**30.68 dB**	**32.08 dB**	**33.02 dB**	**33.57 dB**	**34.08 dB**	**34.64 dB**	**35.13 dB**	**35.46 dB**	**35.85 dB**
EZW	**27.72 dB**	**34.13 dB**	**35.63 dB**	**36.68 dB**	**37.46 dB**	**38.07 dB**	**38.55 dB**	**38.86 dB**	**39.11 dB**	**39.35 dB**

Table4.1: PSNR of the first method with EZW on image Lena

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
64 divisions	**19.96 dB**	**24.58 dB**	**25.16 dB**	**25.42 dB**	**25.8 dB**	**26.64 dB**	**26.93 dB**	**27.18 dB**	**27.38 dB**	**27.61 dB**
16 divisions	**21.96 dB**	**26.01 dB**	**26.69 dB**	**26.98 dB**	**27.27 dB**	**27.79 dB**	**28.28 dB**	**28.57 dB**	**28.95 dB**	**29.45 dB**
4 divisions	**23.27 dB**	**27.33 dB**	**27.85 dB**	**28.28 dB**	**28.64 dB**	**29.18 dB**	**29.68 dB**	**30.05 dB**	**30.58 dB**	**30.98 dB**
EZW	**23.2 dB**	**28.86 dB**	**29.57 dB**	**30.42 dB**	**31.29 dB**	**32 dB**	**32.78 dB**	**33.54 dB**	**34.25 dB**	**35.15 dB**

Table4.2: PSNR of the first method with EZW on image Barbara

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
64 divisions	**22.45 dB**	**25.26 dB**	**26.4 dB**	**27.14 dB**	**27.71 dB**	**28.11 dB**	**28.38 dB**	**28.67 dB**	**28.93 dB**	**29.15 dB**
16 divisions	**24.3 dB**	**26.94 dB**	**27.94 dB**	**28.71 dB**	**29.3 dB**	**29.7 dB**	**30.06 dB**	**30.41 dB**	**30.73 dB**	**31.04 dB**
4 divisions	**25.77 dB**	**28.21 dB**	**29.24 dB**	**29.99 dB**	**30.88 dB**	**31.29 dB**	**31.63 dB**	**32.03 dB**	**32.45 dB**	**32.75 dB**
EZW	**26.16 dB**	**31.11 dB**	**32.11 dB**	**33.22 dB**	**33.82 dB**	**34.33 dB**	**34.8 dB**	**35.27 dB**	**35.8 dB**	**36.12 dB**

Table4.3: PSNR of the first method with EZW on image Goldhill

4.5.1.2. SPIHT

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
64 divisions	**24.42 dB**	**27.34 dB**	**29.09 dB**	**30.54 dB**	**31.56 dB**	**32.45 dB**	**33.25 dB**	**34.97 dB**	**35.54 dB**	**35.08 dB**
16 divisions	**25.62 dB**	**28.47 dB**	**30.41 dB**	**31.82 dB**	**32.92 dB**	**33.94 dB**	**34.78 dB**	**35.54 dB**	**36.23 dB**	**36.86 dB**
4 divisions	**27.4 dB**	**30.24 dB**	**32.07 dB**	**33.45 dB**	**34.58 dB**	**35.46 dB**	**36.25 dB**	**36.97 dB**	**37.62 dB**	**38.21 dB**
SPIHT	**29.61 dB**	**32.59 dB**	**34.49 dB**	**35.88 dB**	**37.04 dB**	**37.8 dB**	**38.53 dB**	**39.15 dB**	**39.79 dB**	**40.32 dB**

Table4.4: PSNR of the first method with SPIHT on image Lena

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
64 divisions	**21.95 dB**	**23.13 dB**	**24.02 dB**	**25.7 dB**	**26.18 dB**	**27.41 dB**	**28.51 dB**	**29.51 dB**	**30.51 dB**	**31.39 dB**
16 divisions	**22.78 dB**	**24.14 dB**	**25.42 dB**	**26.63 dB**	**27.76 dB**	**28.89 dB**	**29.86 dB**	**30.77 dB**	**31.77 dB**	**32.68 dB**

4 divisions	**23.52 dB**	**25.14 dB**	**26.84 dB**	**28.36 dB**	**29.61 dB**	**30.91 dB**	**32.02 dB**	**33 dB**	**33.94 dB**	**34.78 dB**
SPIHT	**23.71 dB**	**26.19 dB**	**28.07 dB**	**29.89 dB**	**31.28 dB**	**32.45 dB**	**33.69 dB**	**34.78 dB**	**35.8 dB**	**36.61 dB**

Table4.5: PSNR of the first method with SPIHT on image Barbara

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
64 divisions	**22.91 dB**	**24.85 dB**	**26.19 dB**	**26.24 dB**	**27.1 dB**	**28.89 dB**	**29.66 dB**	**30.34 dB**	**31.97 dB**	**31.54 dB**
16 divisions	**24.7 dB**	**26.65 dB**	**28 dB**	**28.98 dB**	**29.91 dB**	**30.71 dB**	**31.37 dB**	**32 dB**	**32.65 dB**	**33.27 dB**
4 divisions	**26.39 dB**	**28.4 dB**	**29.72 dB**	**30.75 dB**	**31.63 dB**	**32.36 dB**	**33.05 dB**	**33.71 dB**	**34.35 dB**	**34.95 dB**
SPIHT	**27.45 dB**	**29.37 dB**	**30.85 dB**	**31.85 dB**	**32.73 dB**	**33.57 dB**	**34.38 dB**	**35.05 dB**	**35.61 dB**	**36.16 dB**

Table4.6: PSNR of the first method with SPIHT on image Goldhill

From the previous tables, we notice that the first method using EZW coder performed poorly, on a rate of 1bpp, the PSNR performance was, on average, 4dB less than the PSNR obtained from using EZW on all the test images. This constitutes a large loss in the compressed image quality. In the case of SPIHT, the PSNR performance was, on average, 2dB less; again another a considerable loss. There was no need to present nor to plot the computation times since this method performed poorly. For this reason, we developed the B-EZW and B-SPIHT.

4.5.2 Results obtained from the proposed algorithms B-EZW and B-SPIHT

In this section, we first compare our B-EZW and B-SPIHT methods with the coders EZW and SPIHT respectively using different thresholds. Then we look at the influence of different divisions (4, 16 and 64). In all what comes next, B-EZW4, B-EZW16 and B-EZW64 signify that the use of 4 divisions, 16 divisions and 64 divisions respectively. The same case apply to B-SPIHT. We chose this notation in order to reduce the proposed algorithm's name.

4.5.2.1. Comparing B-EZW and EZW

4.5.2.1.1: PSNR tables and plots

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
B-EZW64	**21.49 dB**	**24.37 dB**	**26.1 dB**	**27.46 dB**	**28.67 dB**	**29.62 dB**	**30.43 dB**	**30.75 dB**	**31.63 dB**	**31.96 dB**
B-EZW16	**23.53 dB**	**25.98 dB**	**27.9 dB**	**29.23 dB**	**29.99 dB**	**31.33 dB**	**32.35 dB**	**33.18 dB**	**33.6 dB**	**34.03 dB**
B-EZW4	**24.59 dB**	**27.41 dB**	**29.56 dB**	**30.99 dB**	**31.94 dB**	**32.69 dB**	**34.06 dB**	**34.66 dB**	**35.46 dB**	**36.14 dB**
EZW	**27.6 dB**	**30.39 dB**	**32.52 dB**	**33.66 dB**	**34.57 dB**	**35.75 dB**	**37 dB**	**38.13 dB**	**38.72 dB**	**39.33 dB**

Table4.7: PSNR of B-EZW(64, 16 and 4) and EZW on image Lena

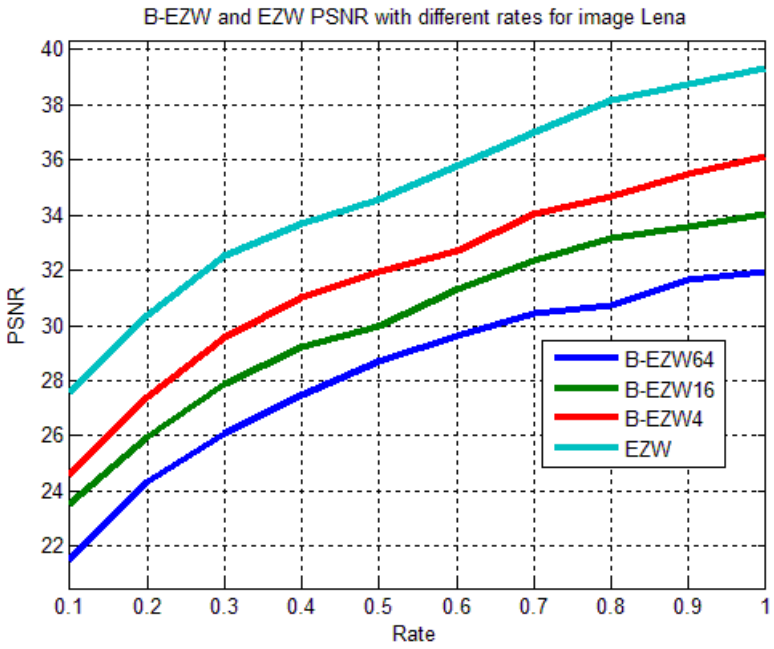


Figure 4.6: PSNR vs rate for B-EZW and EZW for image Lena

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
B-EZW64	**17.96 dB**	**20.31 dB**	**21.22 dB**	**21.8 dB**	**22.62 dB**	**23.31 dB**	**23.99 dB**	**25.14 dB**	**26.92 dB**	**28.44 dB**
B-EZW16	**18.96 dB**	**21.87 dB**	**23.34 dB**	**25.56 dB**	**27.2 dB**	**28.39 dB**	**29.57 dB**	**31.27 dB**	**32.47 dB**	**33.58 dB**
B-EZW4	**19.27 dB**	**21.17 dB**	**22.61 dB**	**24.07 dB**	**25.42 dB**	**26.62 dB**	**27.85 dB**	**29.54 dB**	**31 dB**	**32.09 dB**
EZW	**19.2 dB**	**24.38 dB**	**25.99 dB**	**27.47 dB**	**28.87 dB**	**29.91 dB**	**31.87 dB**	**33.09 dB**	**34.08 dB**	**35.13 dB**

Table4.8: PSNR of B-EZW(64, 16 and 4) and EZW on image Barbara

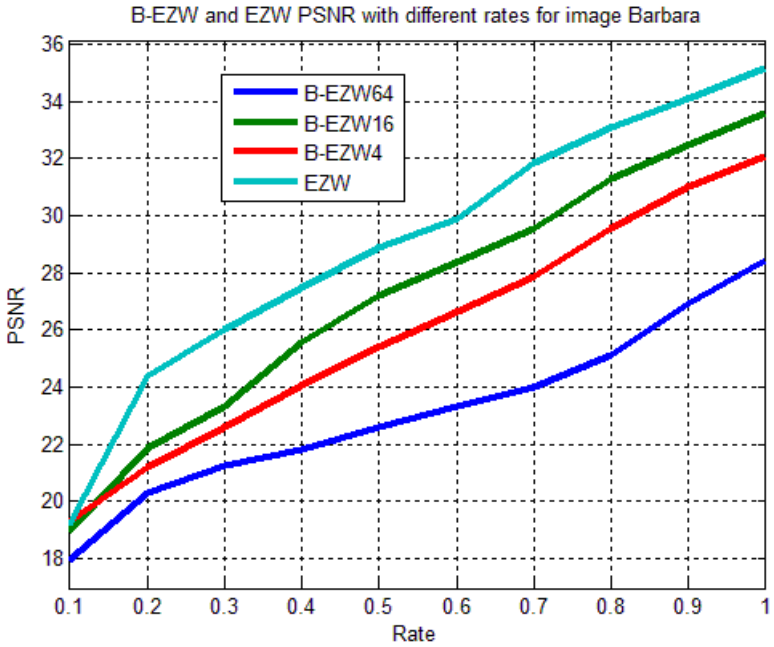


Figure 4.7: PSNR vs rate for B-EZW and EZW for image Barbara

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-EZW64	**24.45 dB**	**27.21 dB**	**28.39 dB**	**29.25 dB**	**30.03 dB**	**30.6 dB**	**30.92 dB**	**31.15 dB**	**31.45 dB**	**31.69 dB**
B-EZW16	**25.3 dB**	**28.17 dB**	**29.22 dB**	**30.02 dB**	**30.85 dB**	**31.35 dB**	**31.8 dB**	**32.34 dB**	**32.68 dB**	**33.03 dB**
B-EZW4	**25.77 dB**	**28.48 dB**	**29.59 dB**	**30.68 dB**	**31.3 dB**	**31.84 dB**	**32.31 dB**	**32.74 dB**	**33.16 dB**	**33.65 dB**
EZW	**26.16 dB**	**28.61 dB**	**30.41 dB**	**31.52 dB**	**32.52 dB**	**33.33 dB**	**34 dB**	**34.67 dB**	**35.4 dB**	**36.12 dB**

Table4.9: PSNR of B-EZW(64, 16 and 4) and EZW on image Goldhill

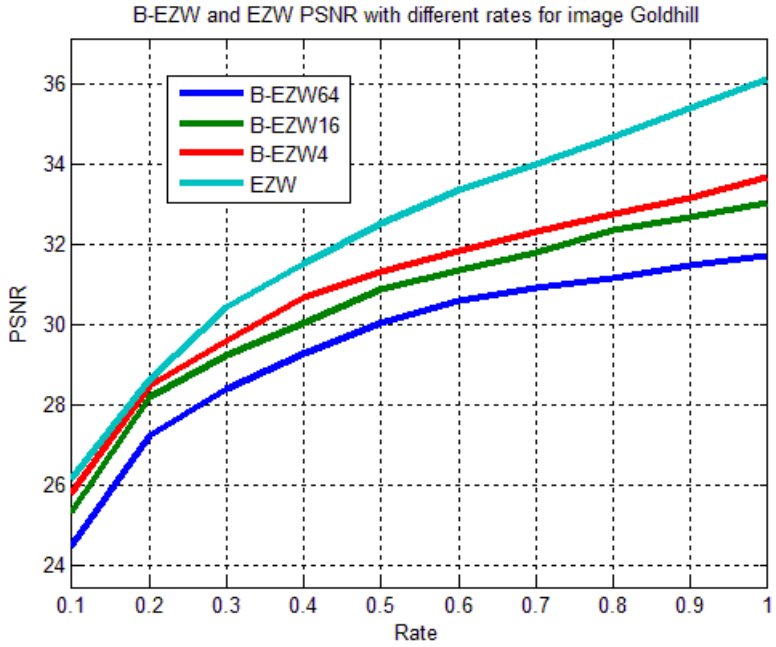


Figure 4.8: PSNR vs rate for B-EZW and EZW for image Goldhill

The previous results show that the EZW gave better PSNR to B-EZW used on a divided image, the PSNR was 1.5dB higher on average when applied on all the test images; but B-EZW is still 2dB lower than EZW. B-EZW with 4 divisions performed better than that with 16 and 64. The results are still lower to those obtained by EZW; this is due to the loss of the zerotrees chains. When applying the wavelet transform to the divisions, the zerotrees chains become smaller in size comparing to those obtained from the wavelet of the whole image which is the core feature of the EZW algorithm.

4.5.2.1.2: Computation times

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-EZW64	**11.35 s**	**12.97 s**	**14.35 s**	**15.16 s**	**15.76 s**	**16.39 s**	**17.12 s**	**17.89 s**	**18.66 s**	**19.41 s**
B-EZW16	**13.24 s**	**15.09 s**	**16.59 s**	**17.95 s**	**19.25 s**	**20.51 s**	**21.73 s**	**22.91 s**	**24.05 s**	**25.16 s**
B-EZW4	**17.27 s**	**22.25 s**	**26.9 s**	**31.23 s**	**35.08 s**	**38.59 s**	**41.9 s**	**45.12 s**	**48.34 s**	**51.53 s**
EZW	**64.96 s**	**110.65 s**	**154.02 s**	**194.07 s**	**229.65 s**	**262.6 s**	**294.37 s**	**326.37 s**	**358.65 s**	**390.74 s**

Table4.10: Computation times of B-EZW(64, 16 and 4) and EZW on image Lena

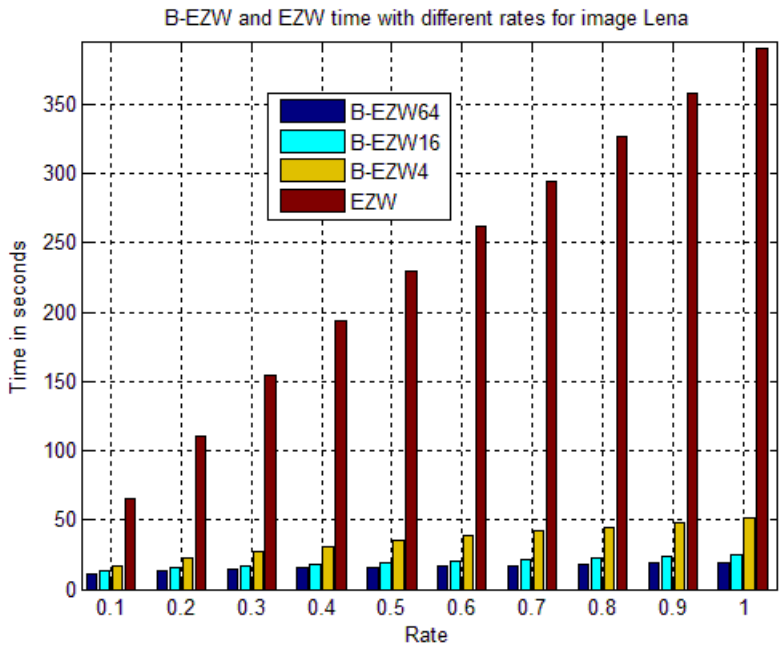


Figure 4.9: Computation times of B-EZW(64, 16 and 4) and EZW on image Lena

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
B-EZW64	**11.75 s**	**12.98 s**	**13.92 s**	**14.74 s**	**15.55 s**	**16.31 s**	**17.03 s**	**17.73 s**	**18.42 s**	**19.09 s**
B-EZW16	**13.26 s**	**14.91 s**	**16.21 s**	**17.41 s**	**18.52 s**	**19.55 s**	**20.54 s**	**21.53 s**	**22.53 s**	**23.53 s**
B-EZW4	**17.21 s**	**20.95 s**	**24.06 s**	**27.28 s**	**30.39 s**	**33.43 s**	**36.4 s**	**39.33 s**	**42.23 s**	**45.08 s**
EZW	**71.69 s**	**108.28 s**	**140.42 s**	**174.12 s**	**207.65 s**	**239.81 s**	**269.44 s**	**295.85 s**	**320.11 s**	**342.83 s**

Table4.11: Computation times of B-EZW(64, 16 and 4) and EZW on image Barbara

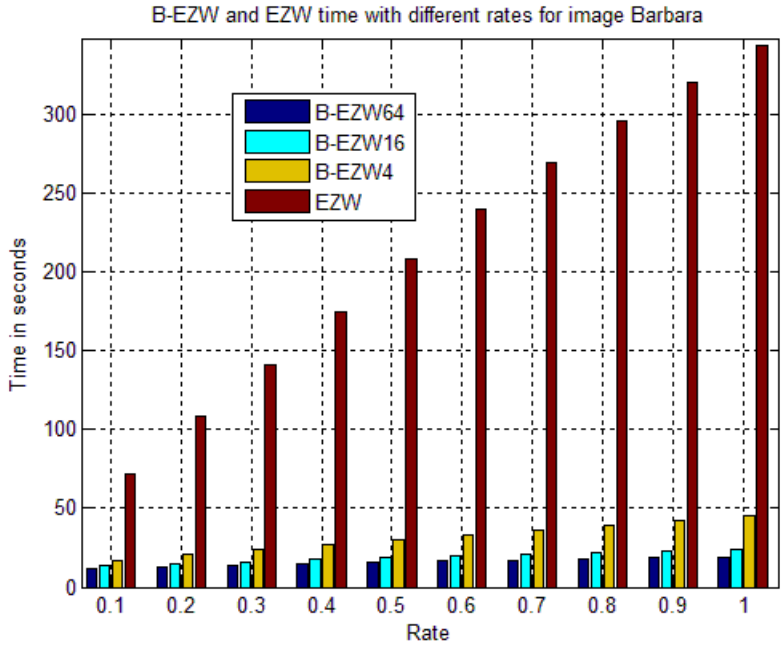


Figure 4.10: Computation times of B-EZW(64, 16 and 4) and EZW on image Barbara

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-EZW64	**11.87 s**	**13.36 s**	**14.5 s**	**15.48 s**	**16.36 s**	**17.17 s**	**17.93 s**	**18.65 s**	**19.33 s**	**19.98 s**
B-EZW16	**13.07 s**	**15.25 s**	**16.95 s**	**18.38 s**	**19.68 s**	**20.97 s**	**22.23 s**	**23.46 s**	**24.65 s**	**25.81 s**
B-EZW4	**16.8 s**	**21.55 s**	**25.97 s**	**30.18 s**	**34.14 s**	**37.88 s**	**41.49 s**	**44.98 s**	**48.35 s**	**51.62 s**
EZW	**63.71 s**	**105.31 s**	**155.65 s**	**209.35 s**	**257.31 s**	**299.72 s**	**339.76 s**	**377.81 s**	**414.23 s**	**449.38 s**

Table4.12: Computation times of B-EZW (64, 16 and 4) and EZW on image Goldhill

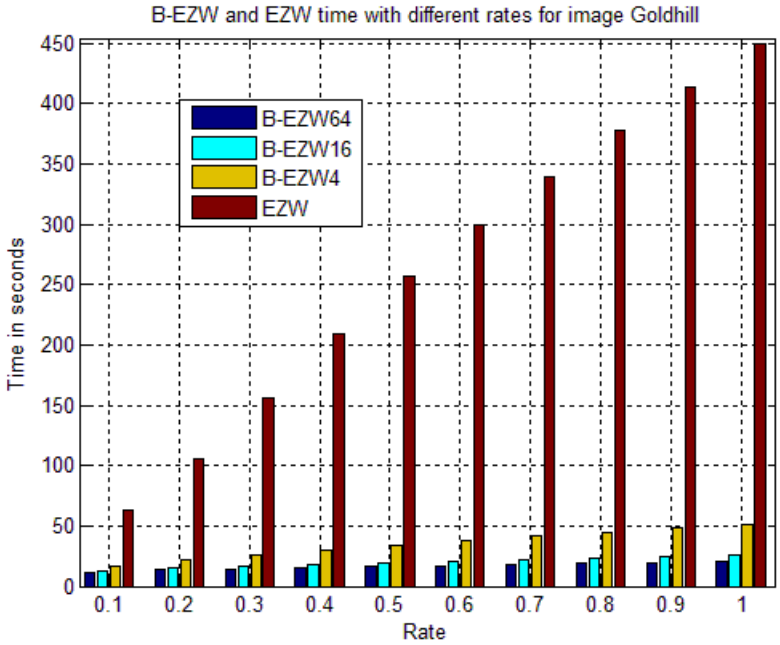


Figure 4.11: Computation times of B-EZW(64, 16 and 4) and EZW on image Goldhill

Although B-EZW did not give desirable enhancement PSNR wise, but we can see a huge reduction in computation time using the parallel computing features. Even on a bit rate of 0.1bpp the comparison is obvious, when comparing the B-EZW and EZW on a bit rate of 1bpp the feature of using parallel computing reduces the computation times by up to 500%.

On all the test images B-EZW64 was the fastest, it is an intuitive result since the more the wavelet coefficients are divided to more smaller blocks the faster the algorithm performs. B-EZW16 and B-EZW4 also are faster the difference becomes apparent on a bit rate of 1bpp, but still much faster than the original EZW.

4.5.2.2. Comparing B-SPIHT and SPIHT

4.5.2.2.1. PSNR

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-spiht64	**18.05 dB**	**21.77 dB**	**23.85 dB**	**25.3 dB**	**26.58 dB**	**27.73 dB**	**28.9 dB**	**30.04 dB**	**31.13 dB**	**32.19 dB**
B-SPIHT16	**20.94 dB**	**25.37 dB**	**27.51 dB**	**29.59 dB**	**31.64 dB**	**32.67 dB**	**33.7 dB**	**34.71 dB**	**35.73 dB**	**36.74 dB**
B-SPIHT4	**27.37 dB**	**30.89 dB**	**33.04 dB**	**34.53 dB**	**35.73 dB**	**36.71 dB**	**37.56 dB**	**38.25 dB**	**38.85 dB**	**39.47 dB**
SPIHT	**29.61 dB**	**32.59 dB**	**34.49 dB**	**35.88 dB**	**37.04 dB**	**37.8 dB**	**38.53 dB**	**39.15 dB**	**39.79 dB**	**40.32 dB**

Table4.13: PSNR of B-SPIHT (64, 16 and 4) and SPIHT on image Lena

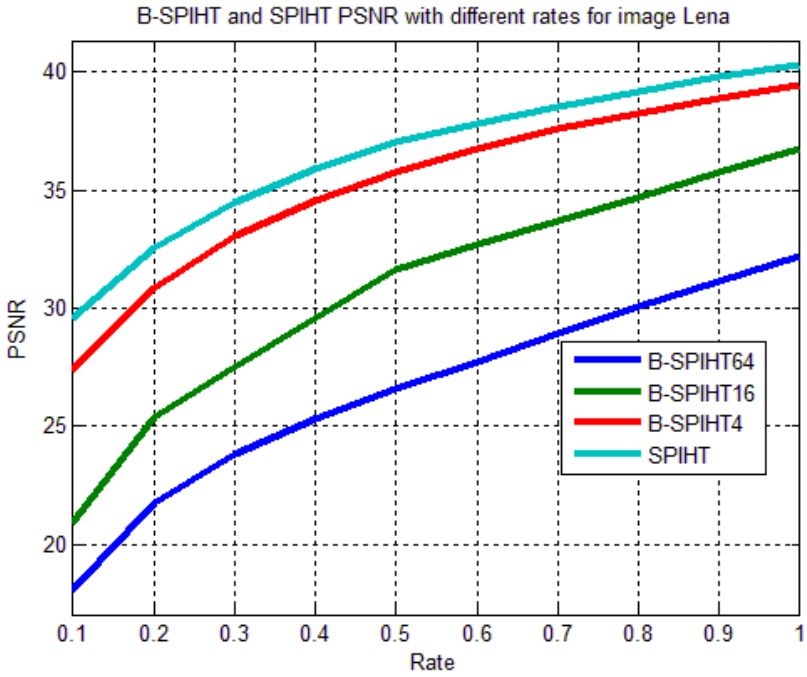


Figure 4.12: PSNR vs rate for B-SPIHT and SPIHT for image Lena

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-spiht64	**6.76 dB**	**12.03 dB**	**15.76 dB**	**18.77 dB**	**21.41 dB**	**23.92 dB**	**26.32 dB**	**28.61 dB**	**30.87 dB**	**32.6 dB**
B-SPIHT16	**14.96 dB**	**18.55 dB**	**21.07 dB**	**23.28 dB**	**25.38 dB**	**27.25 dB**	**29.04 dB**	**30.74 dB**	**32.5 dB**	**34.18 dB**
B-SPIHT4	**23.12 dB**	**25.83 dB**	**27.73 dB**	**29.25 dB**	**30.75 dB**	**32 dB**	**33.13 dB**	**34.28 dB**	**35.25 dB**	**36.08 dB**
SPIHT	**23.71 dB**	**26.19 dB**	**28.07 dB**	**29.89 dB**	**31.28 dB**	**32.45 dB**	**33.69 dB**	**34.78 dB**	**35.8 dB**	**36.61 dB**

Table4.14: PSNR of B-SPIHT (64, 16 and 4) and SPIHT on image Barbara

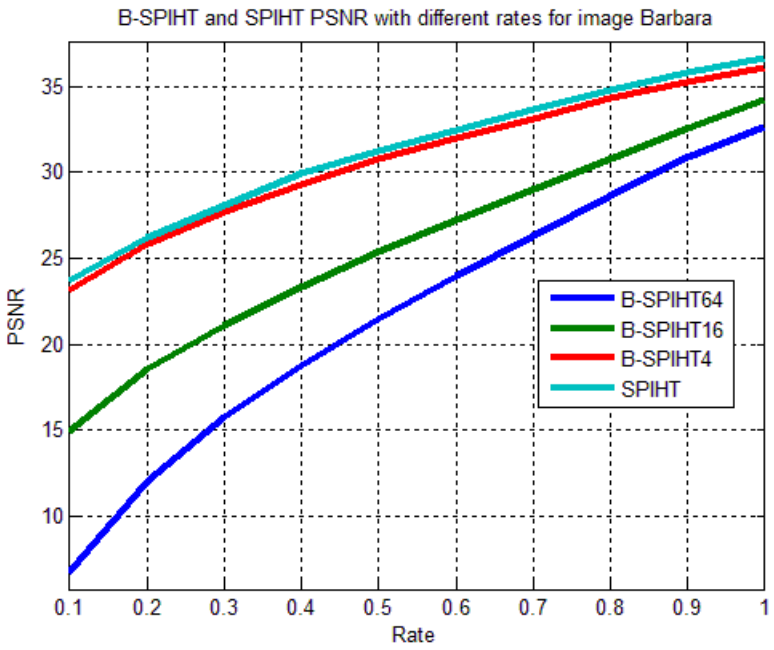


Figure 4.13: PSNR vs rate for B-SPIHT and SPIHT for image Barbara

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
B-spiht64	**16.66 dB**	**20.18 dB**	**22.25 dB**	**23.71 dB**	**24.99 dB**	**26.17 dB**	**27.32 dB**	**28.42 dB**	**29.51 dB**	**30.58 dB**
B-SPIHT16	**20.62 dB**	**23.21 dB**	**25.98 dB**	**27.04 dB**	**28.07 dB**	**29.09 dB**	**30.11 dB**	**31.13 dB**	**32.14 dB**	**33.15 dB**
B-SPIHT4	**26.27 dB**	**28.65 dB**	**30.15 dB**	**31.31 dB**	**32.28 dB**	**33.12 dB**	**33.88 dB**	**34.53 dB**	**35.16 dB**	**35.77 dB**
SPIHT	**27.45 dB**	**29.37 dB**	**30.85 dB**	**31.85 dB**	**32.73 dB**	**33.57 dB**	**34.38 dB**	**35.05 dB**	**35.61 dB**	**36.16 dB**

Table4.15: PSNR of B-SPIHT (64, 16 and 4) and SPIHT on image Goldhill

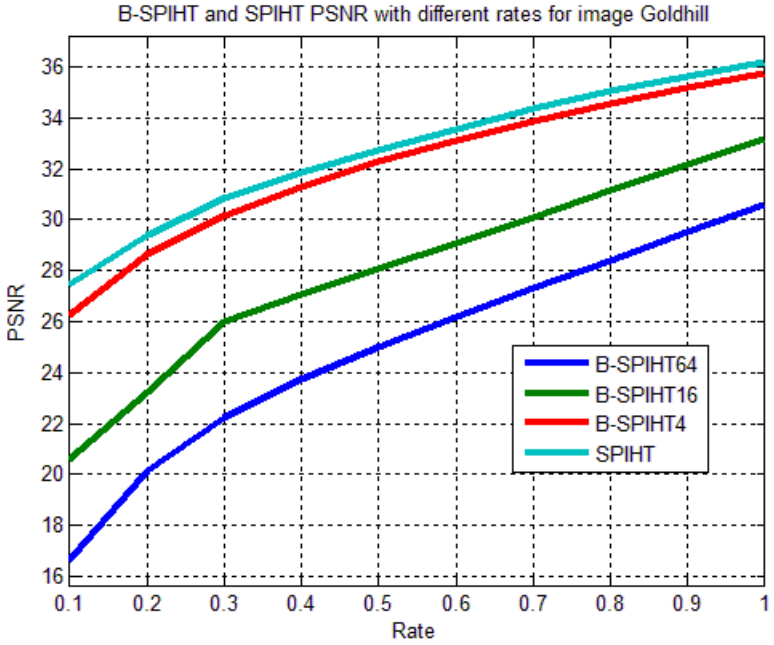


Figure 4.14: PSNR vs rate for B-SPIHT and SPIHT for image Goldhill

B-SPIHT is better than B-EZW in PSNR performance, at best it 1 dB lower than the original SPIHT, the loss in PSNR is attributed to the loss from the wavelet transform which is in nature an irreversible transform. B-SPIHT4 gave the best results when compared to B-SPIHT16 and B-SPIHT64.

4.5.2.2.2. Computation time

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-spiht64	**1.12 s**	**1.78 s**	**2.46 s**	**2.75 s**	**3.22 s**	**3.69 s**	**4.23 s**	**4.68 s**	**5.05 s**	**5.37 s**
B-SPIHT16	**1.13 s**	**1.9 s**	**2.4 s**	**2.93 s**	**3.52 s**	**4.01 s**	**4.66 s**	**5.35 s**	**5.75 s**	**6.12 s**
B-SPIHT4	**1.66 s**	**2.06 s**	**2.7 s**	**3.22 s**	**3.99 s**	**4.5 s**	**5.42 s**	**6.35 s**	**7.09 s**	**7.68 s**
SPIHT	**2.09 s**	**3.76 s**	**5.57 s**	**8.06 s**	**9.7 s**	**13.47 s**	**17.7 s**	**20.7 s**	**25.3 s**	**29.7 s**

Table4.16: Computation times of B-SPIHT (64, 16 and 4) and SPIHT on image Lena

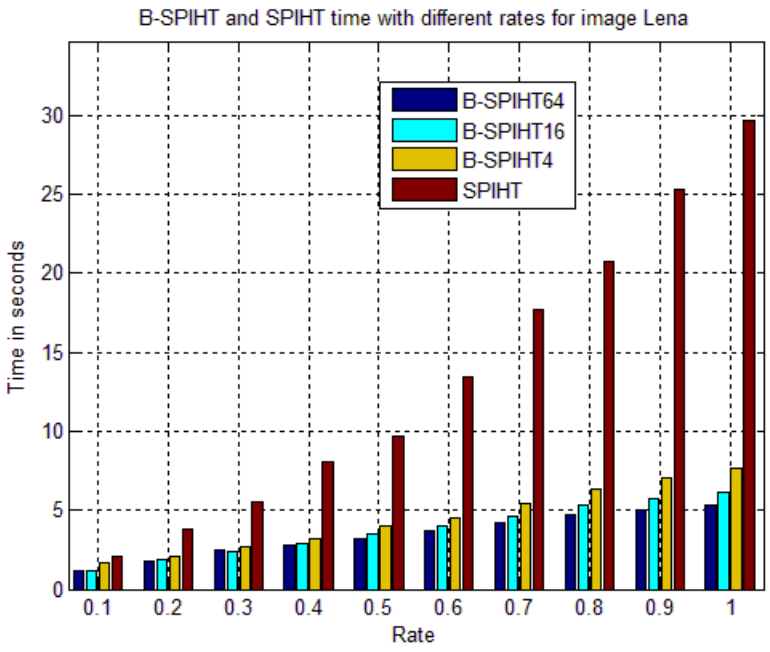


Figure 4.15: Computation times of B-SPIHT(64, 16 and 4) and EZW on image Lena

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-spiht64	**1.12 s**	**1.77 s**	**2.24 s**	**2.71 s**	**3.14 s**	**3.53 s**	**3.98 s**	**4.39 s**	**4.84 s**	**5.27 s**
B-SPIHT16	**1.14 s**	**1.74 s**	**2.24 s**	**2.78 s**	**3.34 s**	**3.83 s**	**5.27 s**	**5.07 s**	**5.6 s**	**5.89 s**
B-SPIHT4	**1.59 s**	**1.87 s**	**2.46 s**	**3.22 s**	**3.75 s**	**4.45 s**	**5.34 s**	**5.91 s**	**6.6 s**	**7.34 s**
SPIHT	**2.22 s**	**3.84 s**	**5.99 s**	**8.37 s**	**10.66 s**	**14.2 s**	**17.94 s**	**19.97 s**	**24.38 s**	**29.42 s**

Table4.17: Computation times of B-SPIHT (64, 16 and 4) and SPIHT on image Barbara

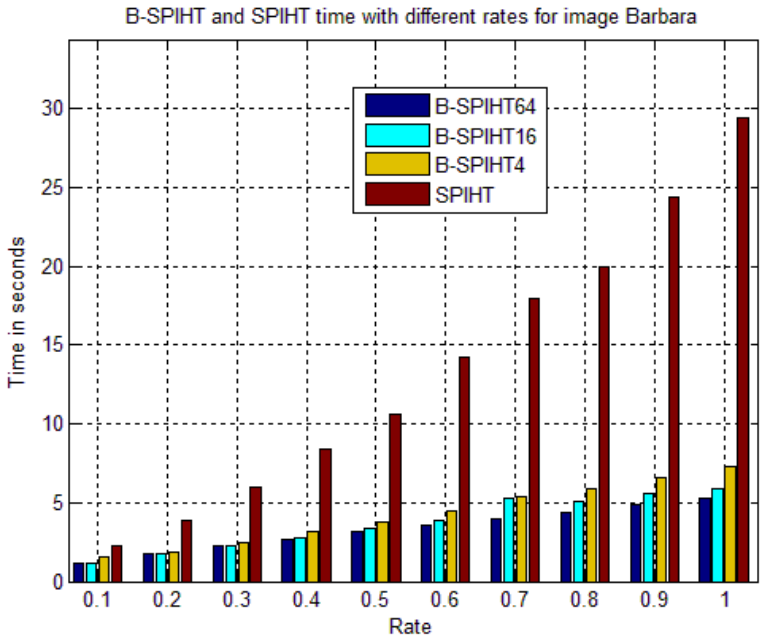


Figure 4.16: Computation times of B-SPIHT(64, 16 and 4) and EZW on image Barbara

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
B-spiht64	**1.14 s**	**1.81 s**	**2.33 s**	**2.84 s**	**3.35 s**	**3.79 s**	**4.17 s**	**4.51 s**	**5.01 s**	**5.44 s**
B-SPIHT16	**1.16 s**	**1.91 s**	**2.61 s**	**3.06 s**	**3.45 s**	**4.41 s**	**4.66 s**	**5.28 s**	**5.82 s**	**6.36 s**
B-SPIHT4	**1.59 s**	**1.98 s**	**2.68 s**	**3.31 s**	**4.23 s**	**5.02 s**	**5.67 s**	**6.49 s**	**7.4 s**	**8.56 s**
SPIHT	**2.26 s**	**4.08 s**	**6.26 s**	**8.81 s**	**13 s**	**17.03 s**	**19.84 s**	**23.58 s**	**29.44 s**	**36.3 s**

Table4.18: Computation times of B-SPIHT (64, 16 and 4) and SPIHT on image Goldhill

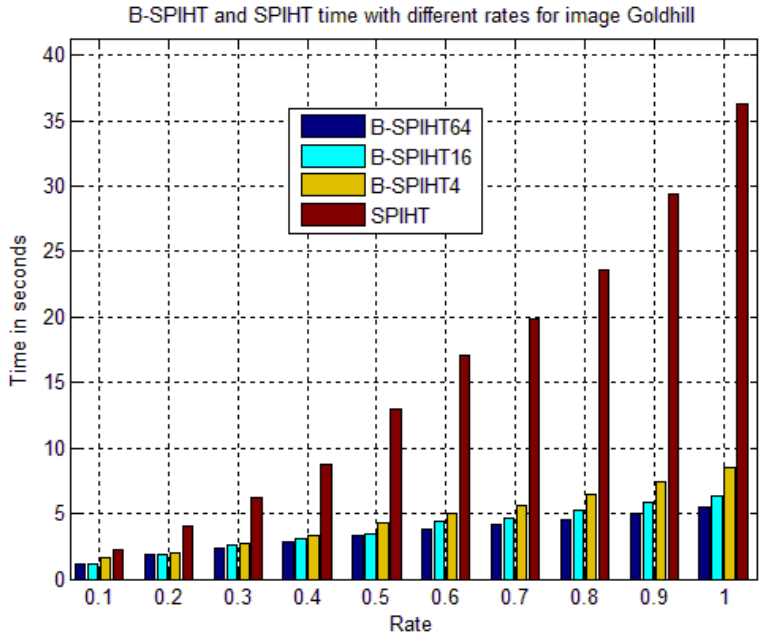


Figure 4.17: Computation times of B-SPIHT(64, 16 and 4) and EZW on image Goldhill

Similarly to B-EZW, B-SPIHT was faster than the SPIHT algorithm in an amount up of 400% since the B-SPIHT do not affect the PSNR excessively, we can conclude that the loss in PSNR is balanced by the fast computation time especially in rates higher than 0.5bpp. It is here where we praise the modification of the SPIHT algorithm. The division of the wavelet coefficients and parallel computing are to be credited for the fast computations.

4.5.3. Results obtained from SPECK

In this section, we will present the performance given by SPECK algorithm, the algorithm is left without any modifications. The same test images are used with the same size of 512x512.

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
PSNR	**29.39 dB**	**32.36 dB**	**34.24 dB**	**35.64 dB**	**36.67 dB**	**37.52 dB**	**38.24 dB**	**38.89 dB**	**39.5 dB**	**40.09 dB**
Time	**2.24 s**	**5.04 s**	**8.29 s**	**11.83 s**	**15.72 s**	**20.13 s**	**25.31 s**	**31.53 s**	**39.04 s**	**47.99 s**

Table4.19: PSNR and Computation time for different rates for SPECK on image Lena

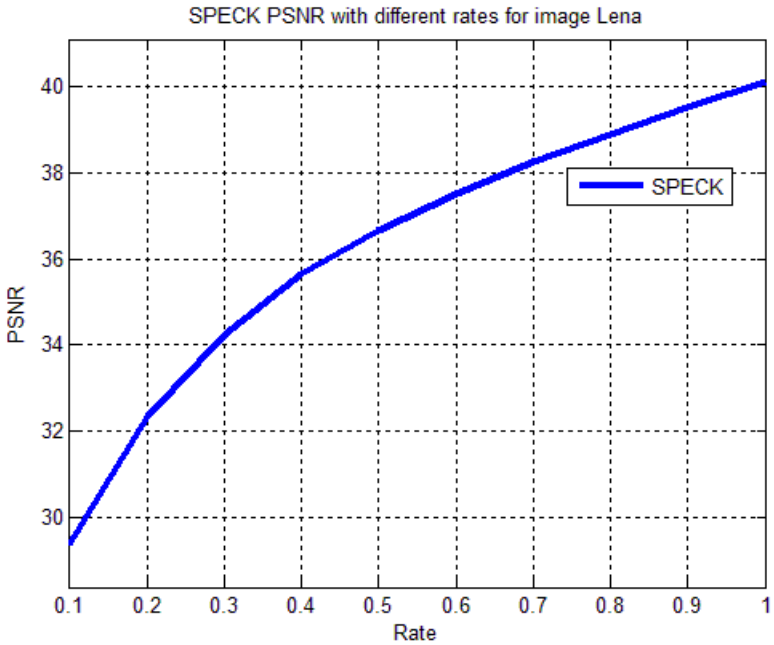


Figure 4.18: PSNR for different rates for SPECK on image Lena

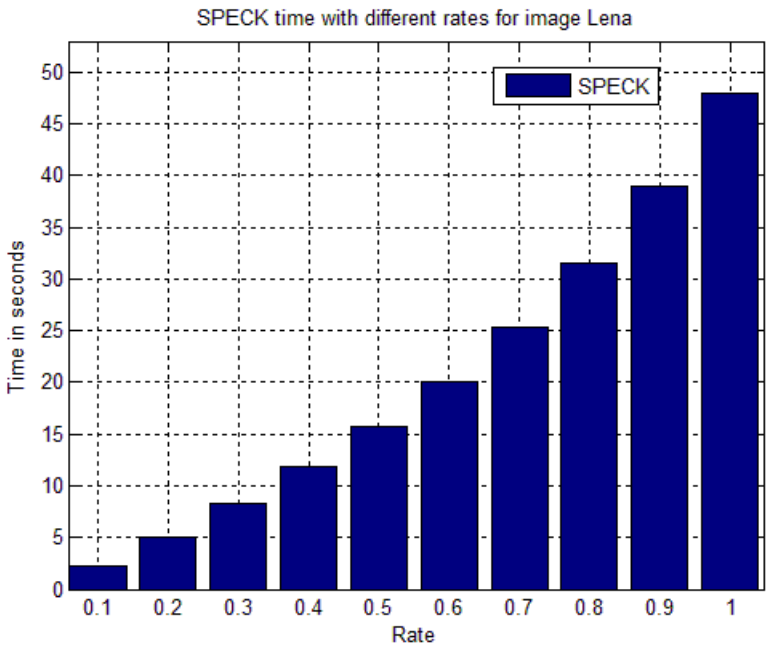


Figure 4.19: computation time for different rates for SPECK on image Lena

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
PSNR	**23.78 dB**	**26.16 dB**	**28.18 dB**	**29.85 dB**	**31.25 dB**	**32.49 dB**	**33.59 dB**	**34.6 dB**	**35.54 dB**	**36.41 dB**
Time	**2.13 s**	**4.42 s**	**7.72 s**	**11.5 s**	**15.56 s**	**20.02 s**	**24.88 s**	**30.18 s**	**35.96 s**	**42.18 s**

Table4.20: PSNR and Computation time for different rates for SPECK on image Barbara

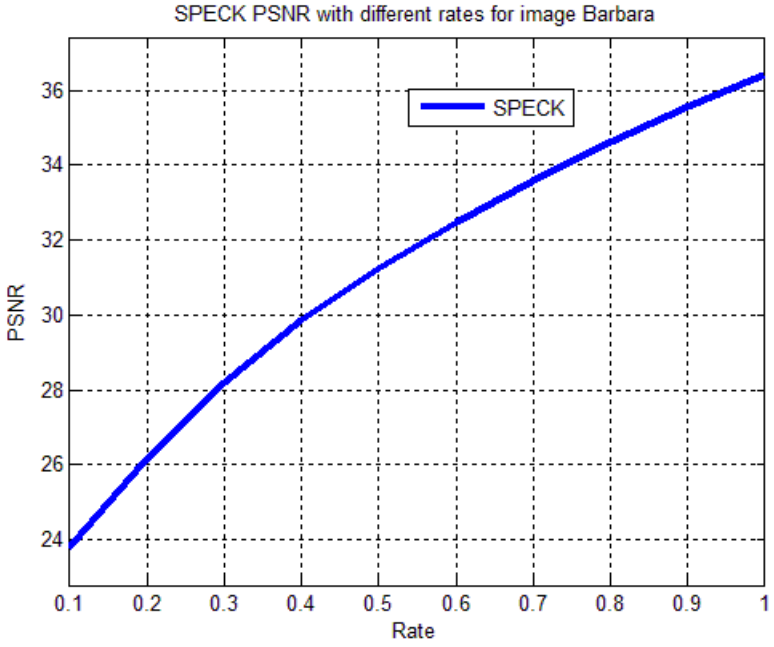


Figure 4.20: PSNR for different rates for SPECK on image Barbara

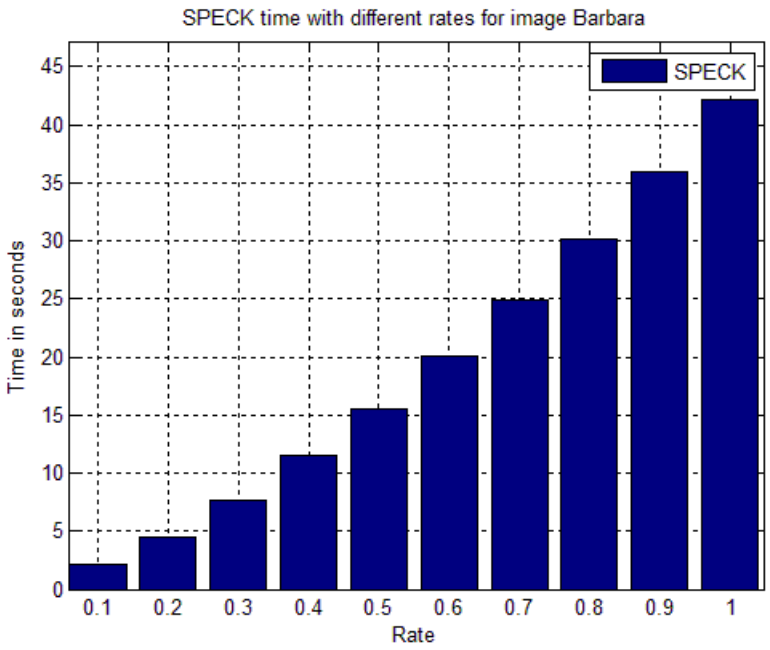


Figure 4.21: computation time for different rates for SPECK on image Barbara

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
PSNR	**27.5 dB**	**29.48 dB**	**30.82 dB**	**31.89 dB**	**32.81 dB**	**33.6 dB**	**34.33 dB**	**34.99 dB**	**35.61 dB**	**36.18 dB**
Time	**2.15 s**	**4.72 s**	**8.1 s**	**12.34 s**	**17.41 s**	**23.24 s**	**29.83 s**	**37.58 s**	**46.36 s**	**56 s**

Table4.21: PSNR and Computation time for different rates for SPECK on image Goldhill

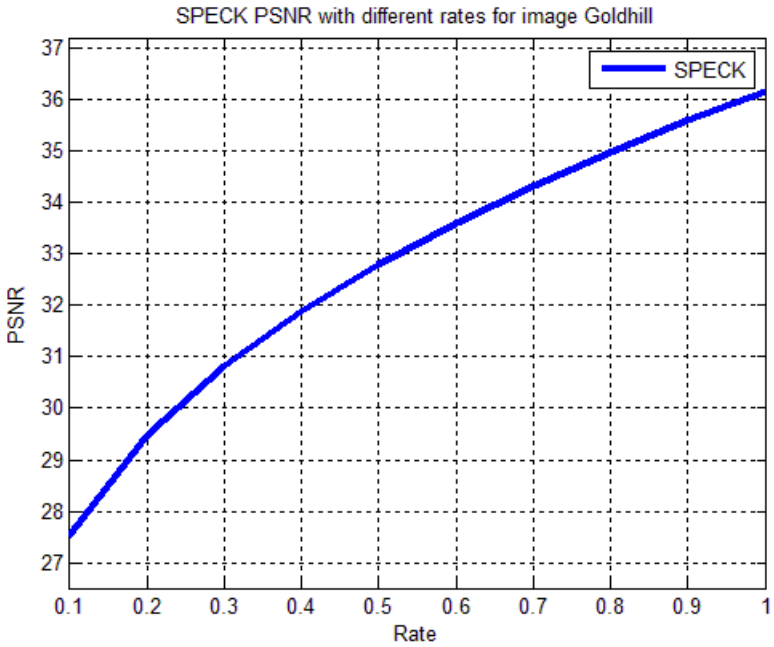


Figure 4.22: PSNR for different rates for SPECK on image Goldhill

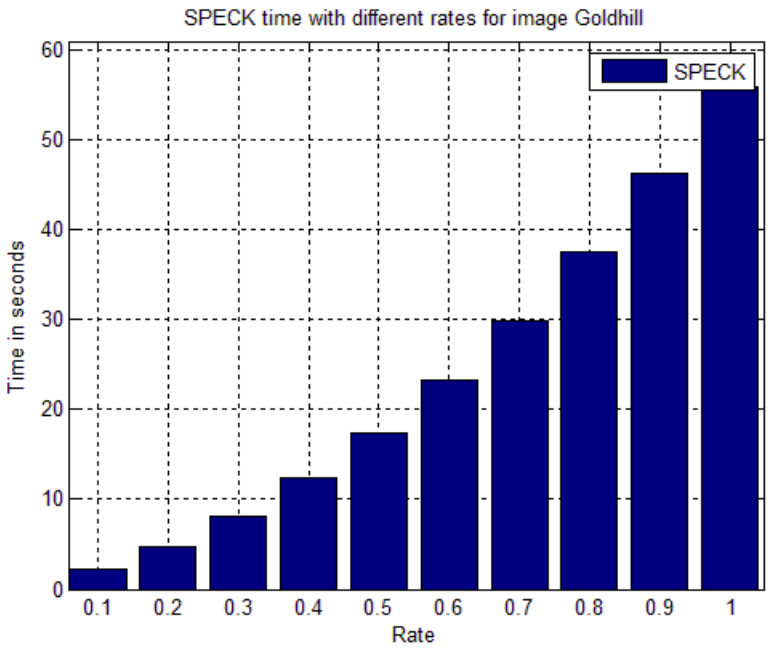


Figure 4.23: computation time for different rates for SPECK on image Goldhill

4.5.4. The overall comparison between EZW, B-EZW4, SPIHT, B-SPIHT4 and SPECK

In this section, we will compare our proposed algorithms (B-EZW4 and B-SPIHT4) with EZW, SPIHT and SPECK. Since B-EZW16, B-EZW64, B-SPIHT16 and SPIHT64 gave lower PSNRs throughout the tests and on all test images, they are not taken into consideration. Firstly, we will compare the PSNR then we compare the computation times.

4.5.4.1. PSNR

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-EZW4	**24.59 dB**	**27.41 dB**	**29.56 dB**	**30.99 dB**	**31.94 dB**	**32.69 dB**	**34.06 dB**	**34.66 dB**	**35.46 dB**	**36.14 dB**
EZW	**27.6 dB**	**30.39 dB**	**32.52 dB**	**33.66 dB**	**34.57 dB**	**35.75 dB**	**37 dB**	**38.13 dB**	**38.72 dB**	**39.33 dB**
B-SPIHT4	**27.37 dB**	**30.89 dB**	**33.04 dB**	**34.53 dB**	**35.73 dB**	**36.71 dB**	**37.56 dB**	**38.25 dB**	**38.85 dB**	**39.47 dB**
SPIHT	**29.61 dB**	**32.59 dB**	**34.49 dB**	**35.88 dB**	**37.04 dB**	**37.8 dB**	**38.53 dB**	**39.15 dB**	**39.79 dB**	**40.32 dB**
SPECK	**29.39 dB**	**32.36 dB**	**34.24 dB**	**35.64 dB**	**36.67 dB**	**37.52 dB**	**38.24 dB**	**38.89 dB**	**39.5 dB**	**40.09 dB**

Table4.22: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Lena

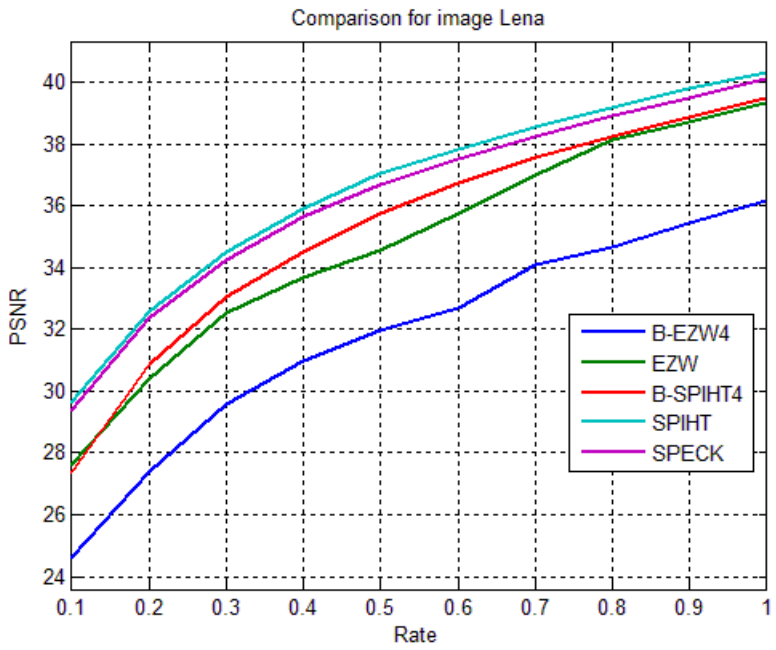


Figure 4.24: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Lena

For image Lena, the algorithms ranked PSNR performance wise as follows: SPIHT, SPECK, B-SPIHT4, EZW and last B-EZW4. B-SPIHT4 was outperformed by SPECK and SPIHT algorithms by 0.62 dB and 0.85 dB on a bit rate of 1bpp. EZW was close in performance to B-SPIHT4 with a difference of 0.13 dB. B-EZW4 was lower throughout the bit rates presented.

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
B-EZW4	**19.27 dB**	**21.17 dB**	**22.61 dB**	**24.07 dB**	**25.42 dB**	**26.62 dB**	**27.85 dB**	**29.54 dB**	**31 dB**	**32.09 dB**
EZW	**19.2 dB**	**24.38 dB**	**25.99 dB**	**27.47 dB**	**28.87 dB**	**29.91 dB**	**31.87 dB**	**33.09 dB**	**34.08 dB**	**35.13 dB**
B-SPIHT4	**23.12 dB**	**25.83 dB**	**27.73 dB**	**29.25 dB**	**30.75 dB**	**32 dB**	**33.13 dB**	**34.28 dB**	**35.25 dB**	**36.08 dB**
SPIHT	**23.71 dB**	**26.19 dB**	**28.07 dB**	**29.89 dB**	**31.28 dB**	**32.45 dB**	**33.69 dB**	**34.78 dB**	**35.8 dB**	**36.61 dB**
SPECK	**23.78 dB**	**26.16 dB**	**28.18 dB**	**29.85 dB**	**31.25 dB**	**32.49 dB**	**33.59 dB**	**34.6 dB**	**35.54 dB**	**36.41 dB**

Table 4.23: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Barbara

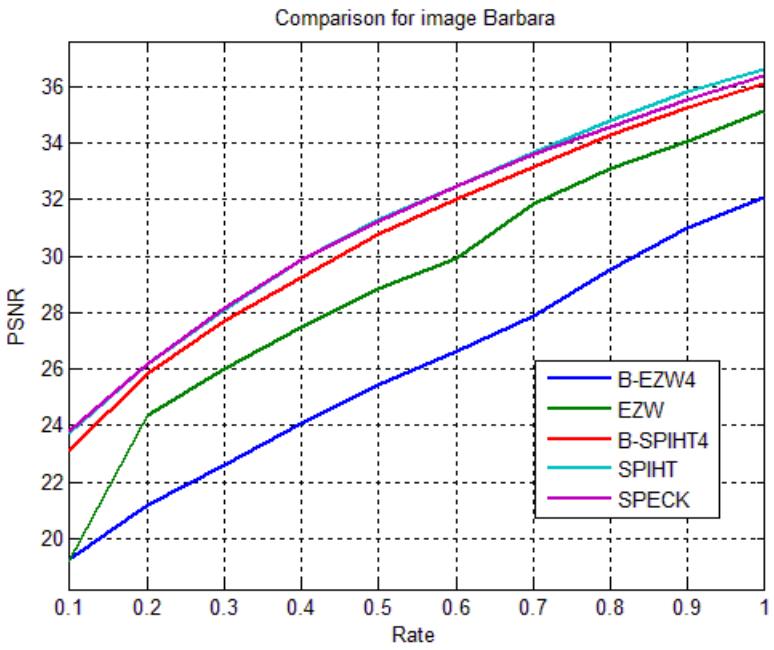


Figure 4.25: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Barbara

In image Barbara, the order of the performance was similar to that obtained on Lena image, except in this case, SPECK and SPIHT were similar for the interval of [0.1 bpp, 0.7 bpp] after 0.7 bpp SPIHT outperforms SPECK. B-SPIHT4 however is close to both SPIHT and SPECK, where they all cluster around a PSNR of 36dB at the bit rate of 1bpp. EZW is slightly lower and B-EZW4 is the lowest at a PSNR of 32dB.

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
B-EZW4	**25.77 dB**	**28.48 dB**	**29.59 dB**	**30.68 dB**	**31.3 dB**	**31.84 dB**	**32.31 dB**	**32.74 dB**	**33.16 dB**	**33.65 dB**
EZW	**26.16 dB**	**28.61 dB**	**30.41 dB**	**31.52 dB**	**32.52 dB**	**33.33 dB**	**34 dB**	**34.67 dB**	**35.4 dB**	**36.12 dB**
B-SPIHT4	**26.27 dB**	**28.65 dB**	**30.15 dB**	**31.31 dB**	**32.28 dB**	**33.12 dB**	**33.88 dB**	**34.53 dB**	**35.16 dB**	**35.77 dB**
SPIHT	**27.45 dB**	**29.37 dB**	**30.85 dB**	**31.85 dB**	**32.73 dB**	**33.57 dB**	**34.38 dB**	**35.05 dB**	**35.61 dB**	**36.16 dB**
SPECK	**27.5 dB**	**29.48 dB**	**30.82 dB**	**31.89 dB**	**32.81 dB**	**33.6 dB**	**34.33 dB**	**34.99 dB**	**35.61 dB**	**36.18 dB**

Table 4.24: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Goldhill

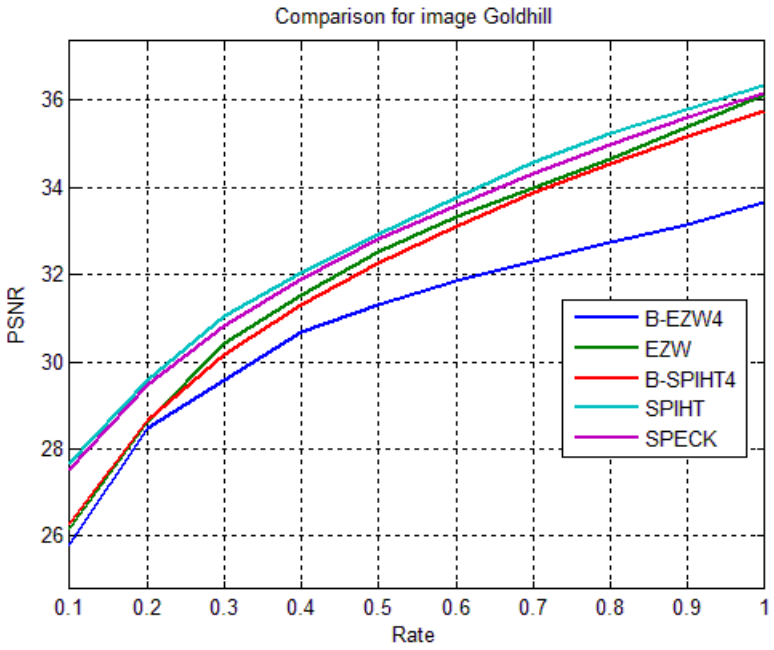


Figure 4.26: PSNR comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Lena

For image Goldhill, B-SPIHT4 was less than EZW, SPECK and SPIHT, it was 0.45dB lower than EZW at the rate of 1 bpp. In this case, EZW was not the 4th in PSNR performance like in the cases of Lena and Barbara. B-EZW4 continued to perform poorly. Since we chose the filters of 'bior6.8' SPIHT performed better than SPECK in all the test images, unlike in the case of Pearlman's paper [9] that introduced SPECK algorithm.

4.5.4.2. Computation times

In this section, we explore the computation times of the same compared algorithms on the test images.

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
B-EZW4	**17.27 s**	**22.25 s**	**26.9 s**	**31.23 s**	**35.08 s**	**38.59 s**	**41.9 s**	**45.12 s**	**48.34 s**	**51.53 s**
EZW	**64.96 s**	**110.65 s**	**154.02 s**	**194.07 s**	**229.65 s**	**262.6 s**	**294.37 s**	**326.37 s**	**358.65 s**	**390.74 s**
B-SPIHT4	**1.66 s**	**2.06 s**	**2.7 s**	**3.22 s**	**3.99 s**	**4.5 s**	**5.42 s**	**6.35 s**	**7.09 s**	**7.68 s**
SPIHT	**2.09 s**	**3.76 s**	**5.57 s**	**8.06 s**	**9.7 s**	**13.47 s**	**17.7 s**	**20.7 s**	**25.3 s**	**29.7 s**
SPECK	**2.24 s**	**5.04 s**	**8.29 s**	**11.83 s**	**15.72 s**	**20.13 s**	**25.31 s**	**31.53 s**	**39.04 s**	**47.99 s**

Table 4.25: time comparison between B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Lena

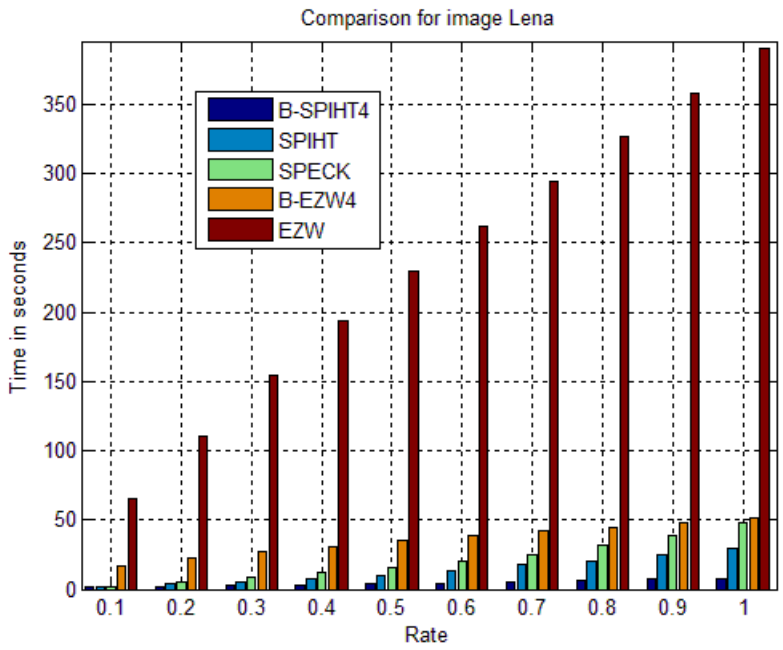


Figure 4.27: PSNR comparison of B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Lena

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
B-EZW4	**17.21 s**	**20.95 s**	**24.06 s**	**27.28 s**	**30.39 s**	**33.43 s**	**36.4 s**	**39.33 s**	**42.23 s**	**45.08 s**
EZW	**71.69 s**	**108.28 s**	**140.42 s**	**174.12 s**	**207.65 s**	**239.81 s**	**269.44 s**	**295.85 s**	**320.11 s**	**342.83 s**
B-SPIHT4	**1.59 s**	**1.87 s**	**2.46 s**	**3.22 s**	**3.75 s**	**4.45 s**	**5.34 s**	**5.91 s**	**6.6 s**	**7.34 s**
SPIHT	**2.22 s**	**3.84 s**	**5.99 s**	**8.37 s**	**10.66 s**	**14.2 s**	**17.94 s**	**19.97 s**	**24.38 s**	**29.42 s**
SPECK	**2.13 s**	**4.42 s**	**7.72 s**	**11.5 s**	**15.56 s**	**20.02 s**	**24.88 s**	**30.18 s**	**35.96 s**	**42.18 s**

Table 4.26: time comparison of B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Barbara

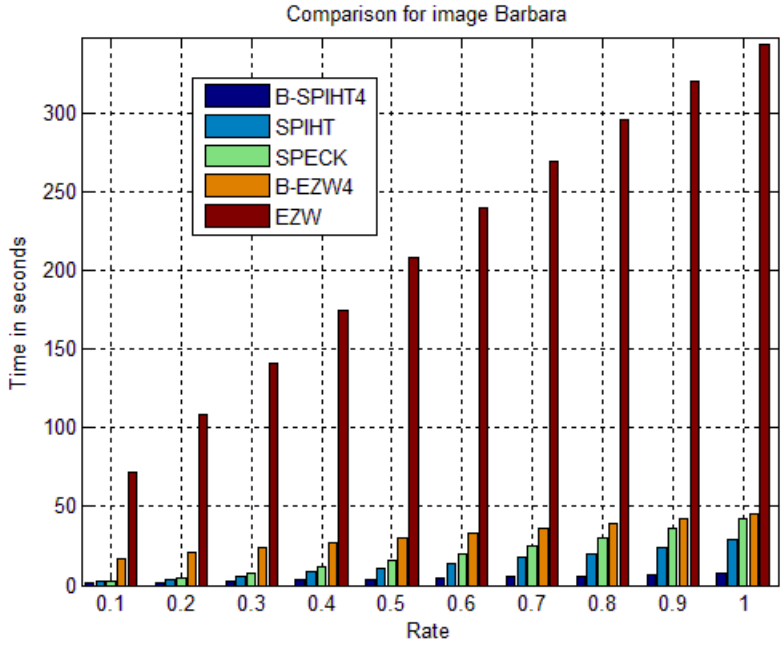


Figure 4.28: PSNR comparison of B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Barbara

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-EZW4	**16.8 s**	**21.55 s**	**25.97 s**	**30.18 s**	**34.14 s**	**37.88 s**	**41.49 s**	**44.98 s**	**48.35 s**	**51.62 s**
EZW	**63.71 s**	**105.31 s**	**155.65 s**	**209.35 s**	**257.31 s**	**299.72 s**	**339.76 s**	**377.81 s**	**414.23 s**	**449.38 s**
B-SPIHT4	**1.59 s**	**1.98 s**	**2.68 s**	**3.31 s**	**4.23 s**	**5.02 s**	**5.67 s**	**6.49 s**	**7.4 s**	**8.56 s**
SPIHT	**2.26 s**	**4.08 s**	**6.26 s**	**8.81 s**	**13 s**	**17.03 s**	**19.84 s**	**23.58 s**	**29.44 s**	**36.3 s**
SPECK	**2.15 s**	**4.72 s**	**8.1 s**	**12.34 s**	**17.41 s**	**23.24 s**	**29.83 s**	**37.58 s**	**46.36 s**	**56 s**

Table 4.27: time comparison of B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Goldhill

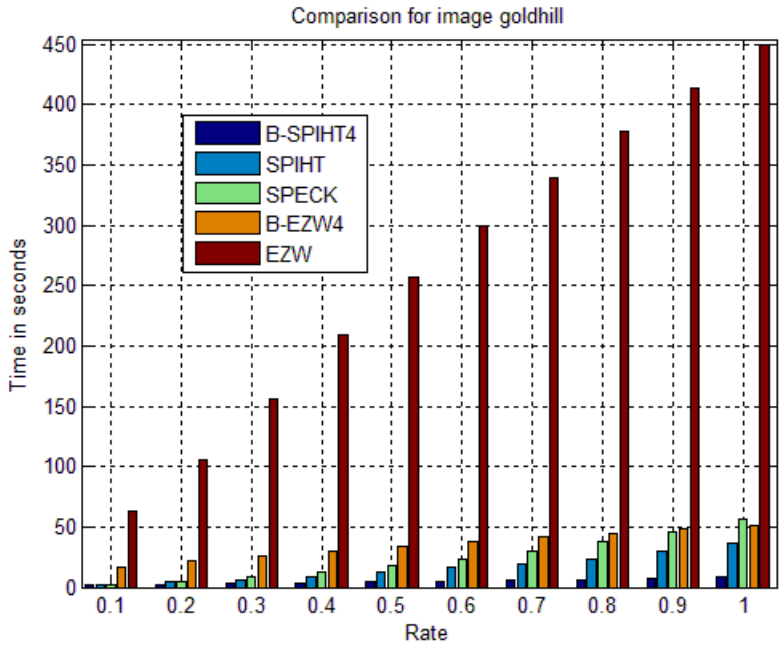


Figure 4.29: PSNR comparison of B-EZW, B-SPIHT, EZW, SPIHT and SPECK for Goldhill

From the tables and figures, we notice high correlation between the tests images. EZW is the slowest algorithm even in bitrates lower than 0.2bpp Even B-EZW4 was slow when compared to SPIHT and SPECK. B-SPIHT however is the fastest method, it shows clearly on bitrates such 1bpp. It is here when the strong feature of B-SPIHT manifests which is the speed of computation.

4.5.5. visual quality

4.5.5.1. B-EZW

a) b) c)

figure 4.30: visual quality for B-EZW4 ***a)*** *0.1820 bpp, threshold 64, 28.91 dB.* ***b)*** *0.4030 bpp, threshold 32, 30.99dB.* ***c)*** *1.5292 bpp, threshold 8, 32.11 dB*

a) b) c)

figure 4.31: visual quality for B-EZW16 ***a)*** *0.2127 bpp, threshold 64, 28.41 dB.* ***b)*** *0.4524 bpp, threshold 32, 30.26 dB.* ***c)*** *1.6207 bpp, threshold 8, 31.01 dB*

a) b) c)

figure 4.32: visual quality for B-EZW64 ***a)*** *0.2833 bpp, threshold 64, 27.81 dB.* ***b)*** *0.5590 bpp, threshold 32, 29.47 dB.* ***c)*** *1.8211 bpp, threshold 8, 30.15 dB*

4.5.5.2. B-SPIHT

a) b) c)

figure 4.33: visual quality for B-SPIHT4 ***a)*** *0.1 bpp, 27.67 dB.* ***b)*** *0.5 bpp, 36.03 dB.* ***c)*** *1 bpp, 39.77 dB*

a) b) c)

figure 4.34: visual quality for B-SPIHT16 ***a)*** *0.1 bpp, 21.56 dB.* ***b)*** *0.5 bpp, 26.76 dB.* ***c)*** *1 bpp, 31.2 dB*

a) b) c)

Figure 4.35: visual quality for B-SPIHT64 ***a)*** *0.1 bpp, 17.78dB.* ***b)*** *0.5 bpp, 23.11 dB.* ***c)*** *1 bpp, 26.77 dB*

4.5.5.3. SPECK

a) b) c)

*figure 4.36: visual quality for SPECK **a)** 0.1 bpp, 29.57 dB, b) 0.42 bpp, 36.09 dB, c) 1.72bpp, 43.68 dB*

4.5.6. influence of parallel computing

After seeing the comparison of the algorithms and noticing the speed of the B-SPIHT and the B-EZW compared to SPIHT and EZW respectively due to parallel computing, in this section we explore the utility of parallel computing. We will execute the B-EZW and the B-SPIHT with and without it. In the following tables, W/ P.C signifies "with parallel computing" and WO/ P.C signifies "without parallel computing". We will start with B-EZW then B-SPIHT. The test images remain the same Lena, Barbara and Goldhill (512x512).

4.5.6.1. B-EZW

rate	***0.1***	***0.2***	***0.3***	***0.4***	***0.5***	***0.6***	***0.7***	***0.8***	***0.9***	***1***
B-EZW64 W/ P.C	**11.87 s**	**13.36 s**	**14.5 s**	**15.48 s**	**16.36 s**	**17.17 s**	**17.93 s**	**18.65 s**	**19.33 s**	**19.98 s**
B-EZW64 W/O P.C	**24.08 s**	**27.05 s**	**29.23 s**	**31.03 s**	**32.61 s**	**34.11 s**	**35.6 s**	**37.03 s**	**38.42 s**	**39.77 s**
B-EZW16 W/ P.C	**13.07 s**	**15.25 s**	**16.95 s**	**18.38 s**	**19.68 s**	**20.97 s**	**22.23 s**	**23.46 s**	**24.65 s**	**25.81 s**
B-EZW16 W/O P.C	**24.15 s**	**27.55 s**	**30.48 s**	**33.17 s**	**35.61 s**	**37.81 s**	**39.86 s**	**41.83 s**	**43.79 s**	**45.77 s**
B-EZW4 W/ P.C	**16.8 s**	**21.55 s**	**25.97 s**	**30.18 s**	**34.14 s**	**37.88 s**	**41.49 s**	**44.98 s**	**48.35 s**	**51.62 s**
B-EZW4 W/O P.C	**29.31 s**	**36.12 s**	**42.34 s**	**48.21 s**	**54.1 s**	**60.02 s**	**65.83 s**	**71.41 s**	**76.7 s**	**81.85 s**

Table 4.28: the influence of parallel computing for B-EZW(4, 16 and 64) on image Lena

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-EZW64 W/ P.C	**11.75 s**	**12.98 s**	**13.92 s**	**14.74 s**	**15.55 s**	**16.31 s**	**17.03 s**	**17.73 s**	**18.42 s**	**19.09 s**
B-EZW64 W/O P.C	**24.35 s**	**26.78 s**	**28.35 s**	**29.68 s**	**31.12 s**	**32.53 s**	**33.89 s**	**35.2 s**	**36.47 s**	**37.7 s**
B-EZW16 W/ P.C	**13.26 s**	**14.91 s**	**16.21 s**	**17.41 s**	**18.52 s**	**19.55 s**	**20.54 s**	**21.53 s**	**22.53 s**	**23.53 s**
B-EZW16 W/O P.C	**24.24 s**	**26.98 s**	**28.99 s**	**31.08 s**	**33.23 s**	**35.34 s**	**37.38 s**	**39.35 s**	**41.23 s**	**43.06 s**
B-EZW4 W/ P.C	**17.21 s**	**20.95 s**	**24.06 s**	**27.28 s**	**30.39 s**	**33.43 s**	**36.4 s**	**39.33 s**	**42.23 s**	**45.08 s**
B-EZW4 W/O P.C	**28.83 s**	**34.5 s**	**39.14 s**	**43.96 s**	**48.66 s**	**53.21 s**	**57.65 s**	**62.01 s**	**66.28 s**	**70.47 s**

Table 4.29: the influence of parallel computing for B-EZW(4, 16 and 64) on image Barbara

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-EZW64 W/ P.C	**11.87 s**	**13.36 s**	**14.5 s**	**15.48 s**	**16.36 s**	**17.17 s**	**17.93 s**	**18.65 s**	**19.33 s**	**19.98 s**
B-EZW64 W/O P.C	**24.32 s**	**27.08 s**	**29.02 s**	**30.78 s**	**32.37 s**	**33.85 s**	**35.26 s**	**36.61 s**	**37.9 s**	**39.15 s**
B-EZW16 W/ P.C	**13.07 s**	**15.25 s**	**16.95 s**	**18.38 s**	**19.68 s**	**20.97 s**	**22.23 s**	**23.46 s**	**24.65 s**	**25.81 s**
B-EZW16 W/O P.C	**24.24 s**	**27.4 s**	**30.21 s**	**32.91 s**	**35.42 s**	**37.74 s**	**39.94 s**	**42.05 s**	**44.07 s**	**46.02 s**
B-EZW4 W/ P.C	**16.8 s**	**21.55 s**	**25.97 s**	**30.18 s**	**34.14 s**	**37.88 s**	**41.49 s**	**44.98 s**	**48.35 s**	**51.62 s**
B-EZW4 W/O P.C	**28.99 s**	**35.59 s**	**41.93 s**	**48.02 s**	**53.74 s**	**59.11 s**	**64.29 s**	**69.28 s**	**74.11 s**	**78.77 s**

Table 4.30: the influence of parallel computing for B-EZW(4, 16 and 64) on image Goldhill

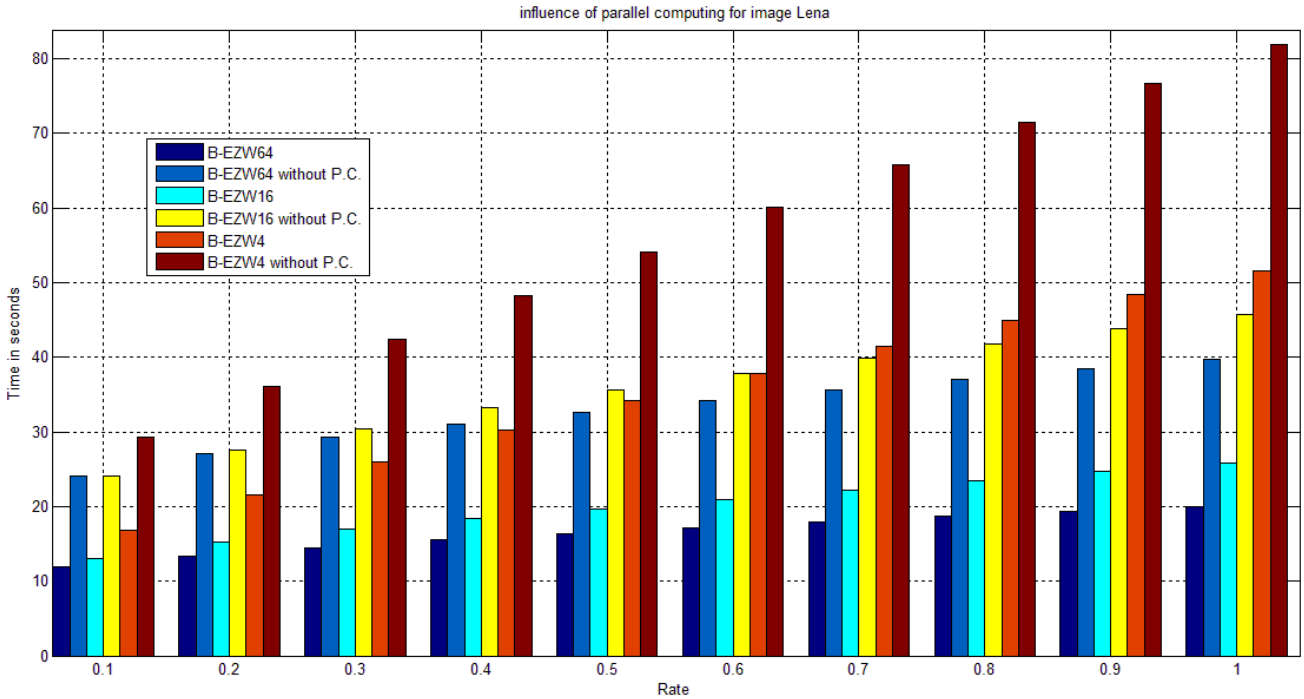


Figure 4.37: the influence of parallel computing for B-EZW(4, 16 and 64) on image Lena

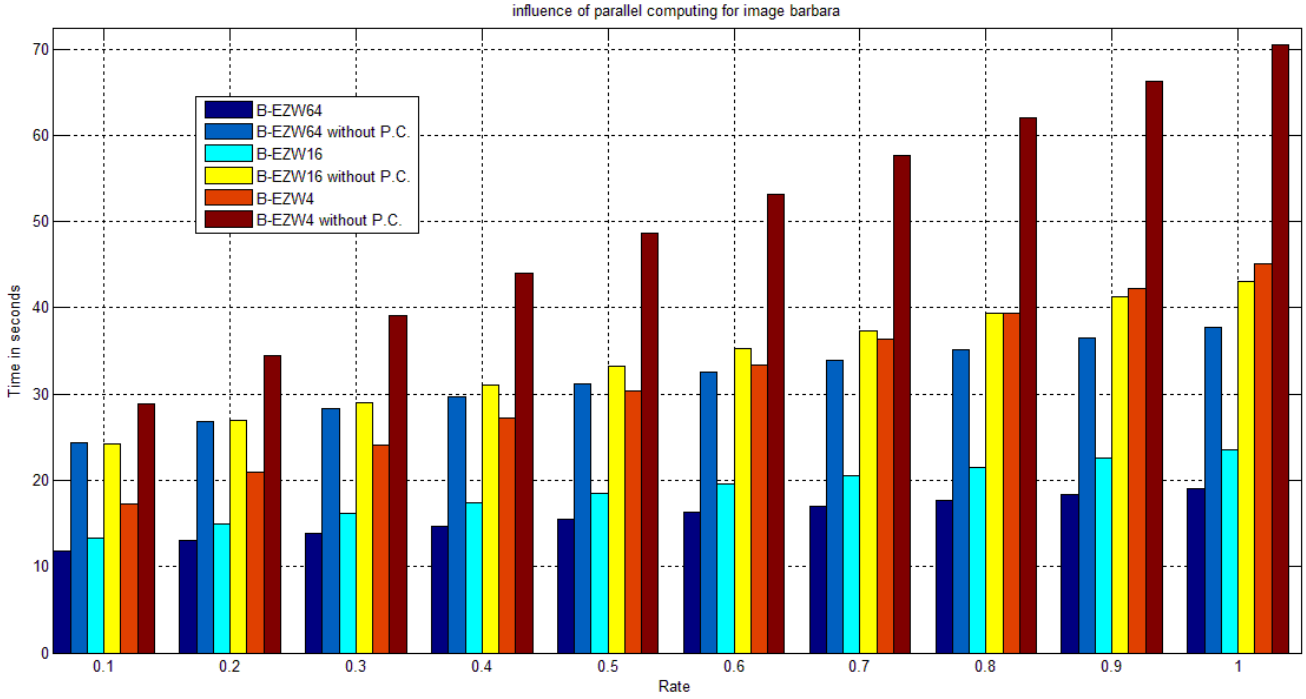


Figure 4.38: the influence of parallel computing for B-EZW(4, 16 and 64) on image Barbara

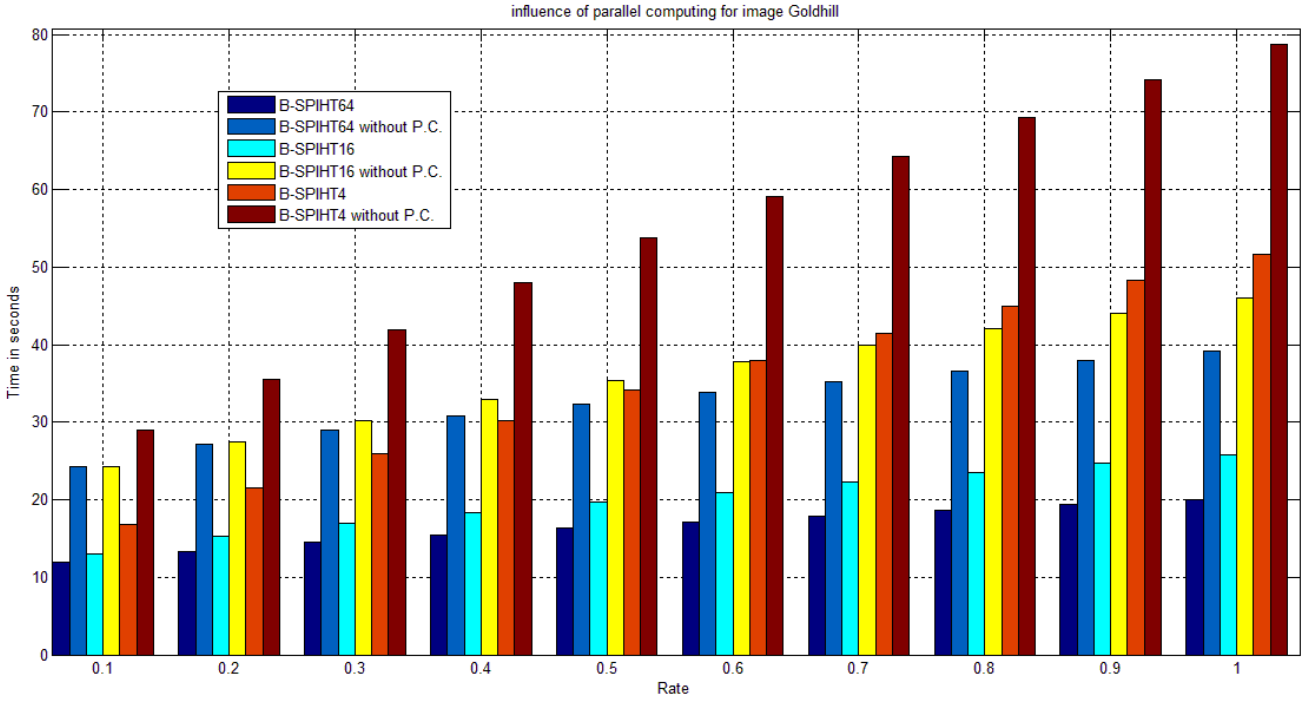


Figure 4.39: the influence of parallel computing for B-EZW(4, 16 and 64) on image Goldhill

After executing the B-EZW with and without parallel computing, we notice clearly its effect on computation time. In all tests, B-EZW was faster with parallel computing reducing a considerable amount of calculation time

4.5.6.2. B-SPIHT

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-SPIHT64 W/ P.C	**1.14 s**	**1.81 s**	**2.33 s**	**2.84 s**	**3.35 s**	**3.79 s**	**4.17 s**	**4.51 s**	**5.01 s**	**5.44 s**
B-SPIHT64 W/O P.C	**1.43 s**	**2.54 s**	**3.45 s**	**4.36 s**	**5.2 s**	**5.95 s**	**6.82 s**	**7.65 s**	**8.29 s**	**8.97 s**
B-SPIHT16 W/ P.C	**1.16 s**	**1.91 s**	**2.61 s**	**3.06 s**	**3.45 s**	**4.41 s**	**4.66 s**	**5.28 s**	**5.82 s**	**6.36 s**
B-SPIHT16 W/O P.C	**1.48 s**	**2.59 s**	**3.62 s**	**4.5 s**	**5.45 s**	**6.43 s**	**7.82 s**	**8.75 s**	**9.85 s**	**10.56 s**
B-SPIHT4 W/ P.C	**1.59 s**	**1.98 s**	**2.68 s**	**3.31 s**	**4.23 s**	**5.02 s**	**5.67 s**	**6.49 s**	**7.4 s**	**8.56 s**
B-SPIHT4 W/O P.C	**1.87 s**	**2.79 s**	**3.85 s**	**5.25 s**	**6.42 s**	**7.78 s**	**9.32 s**	**10.93 s**	**12.26 s**	**13.86 s**

Table 4.31: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Lena

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-SPIHT64 W/ P.C	**1.12 s**	**1.77 s**	**2.24 s**	**2.71 s**	**3.14 s**	**3.53 s**	**3.98 s**	**4.39 s**	**4.84 s**	**5.27 s**
B-SPIHT64 W/O P.C	**1.43 s**	**2.38 s**	**3.27 s**	**4.15 s**	**4.94 s**	**5.67 s**	**6.4 s**	**7.29 s**	**8.06 s**	**8.73 s**
B-SPIHT16 W/ P.C	**1.14 s**	**1.74 s**	**2.24 s**	**2.78 s**	**3.34 s**	**3.83 s**	**5.27 s**	**5.07 s**	**5.6 s**	**5.89 s**
B-SPIHT16 W/O P.C	**1.49 s**	**2.51 s**	**3.35 s**	**4.33 s**	**5.25 s**	**6.29 s**	**7.42 s**	**8.25 s**	**8.98 s**	**9.83 s**
B-SPIHT4 W/ P.C	**1.59 s**	**1.87 s**	**2.46 s**	**3.22 s**	**3.75 s**	**4.45 s**	**5.34 s**	**5.91 s**	**6.6 s**	**7.34 s**
B-SPIHT4 W/O P.C	**1.68 s**	**2.69 s**	**3.82 s**	**5.26 s**	**6.35 s**	**7.6 s**	**9.21 s**	**10.39 s**	**11.83 s**	**13.4 s**

Table 4.32: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Barbara

rate	*0.1*	*0.2*	*0.3*	*0.4*	*0.5*	*0.6*	*0.7*	*0.8*	*0.9*	*1*
B-SPIHT64 W/ P.C	**1.14 s**	**1.81 s**	**2.33 s**	**2.84 s**	**3.35 s**	**3.79 s**	**4.17 s**	**4.51 s**	**5.01 s**	**5.44 s**
B-SPIHT64 W/O P.C	**1.46 s**	**2.49 s**	**3.51 s**	**4.43 s**	**5.31 s**	**6.06 s**	**6.69 s**	**7.43 s**	**8.25 s**	**8.89 s**
B-SPIHT16 W/ P.C	**1.16 s**	**1.91 s**	**2.61 s**	**3.06 s**	**3.45 s**	**4.41 s**	**4.66 s**	**5.28 s**	**5.82 s**	**6.36 s**
B-SPIHT16 W/O P.C	**1.49 s**	**2.67 s**	**3.81 s**	**4.79 s**	**5.49 s**	**6.47 s**	**7.6 s**	**8.76 s**	**9.71 s**	**10.81 s**
B-SPIHT4 W/ P.C	**1.59 s**	**1.98 s**	**2.68 s**	**3.31 s**	**4.23 s**	**5.02 s**	**5.67 s**	**6.49 s**	**7.4 s**	**8.56 s**
B-SPIHT4 W/O P.C	**1.89 s**	**2.83 s**	**4 s**	**5.33 s**	**6.88 s**	**8.33 s**	**9.78 s**	**11.49 s**	**13.19 s**	**15.2 s**

Table 4.33: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Goldhill

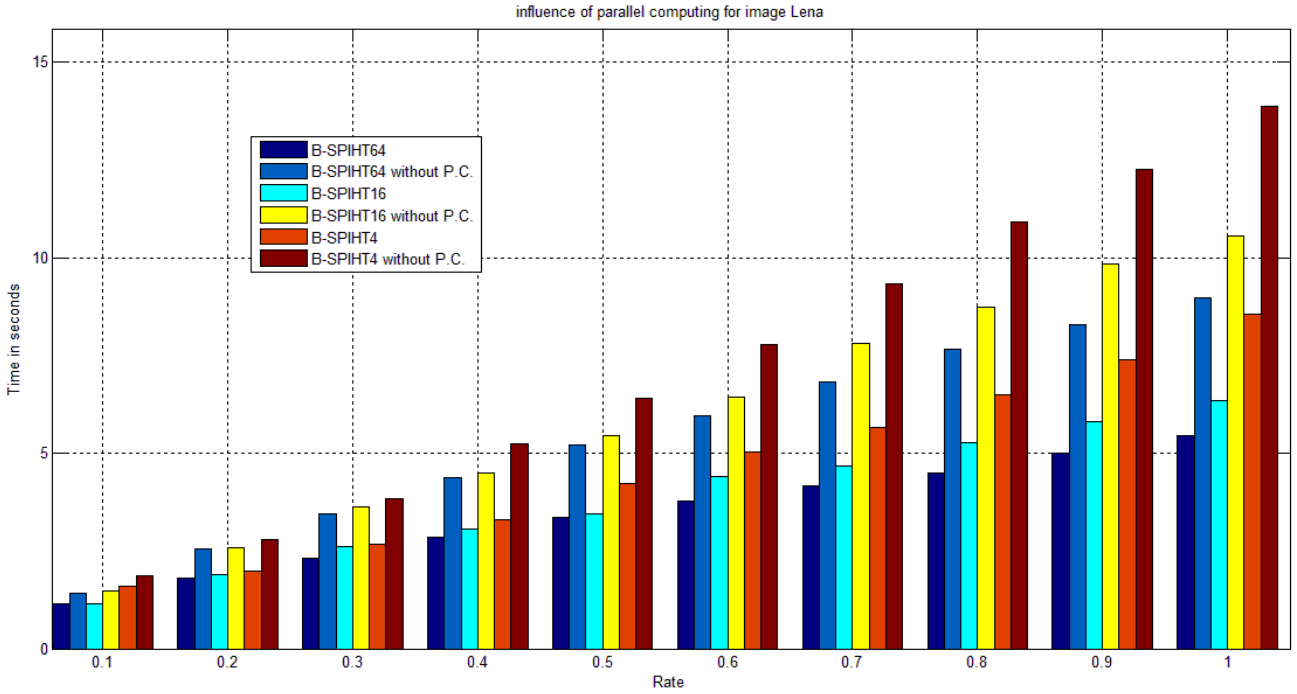


Figure 4.40: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Lena

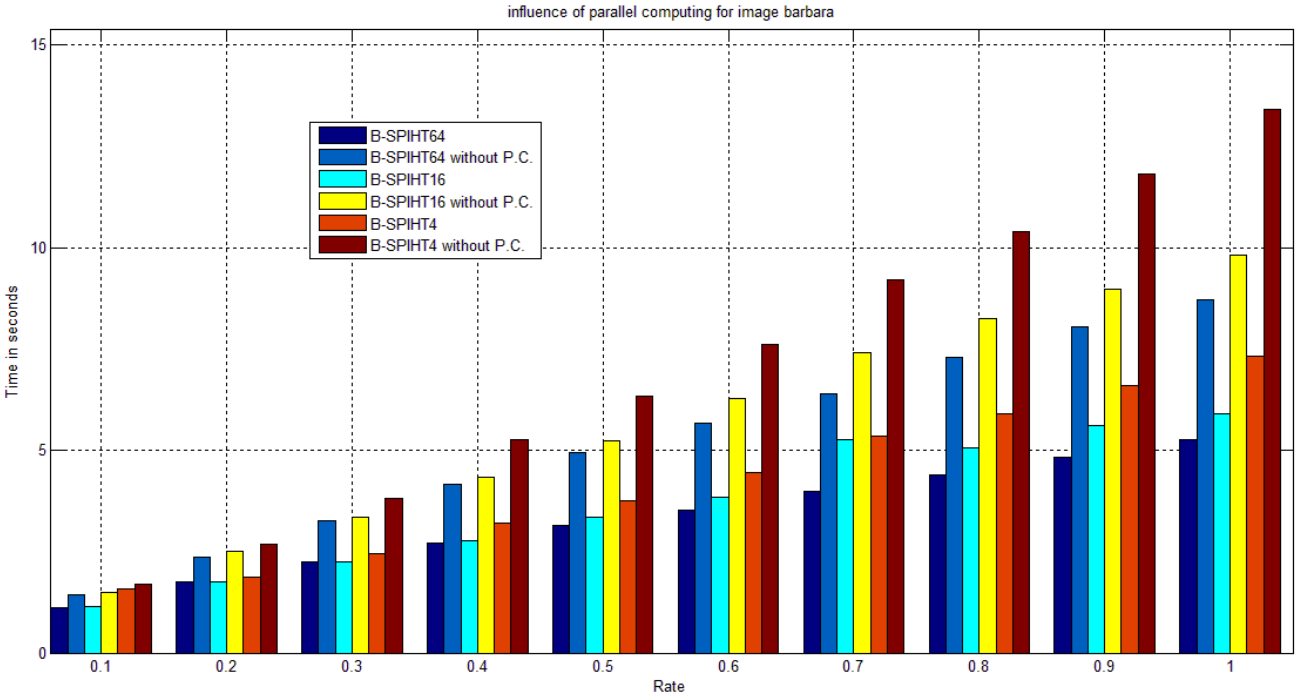


Figure 4.41: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Barbara

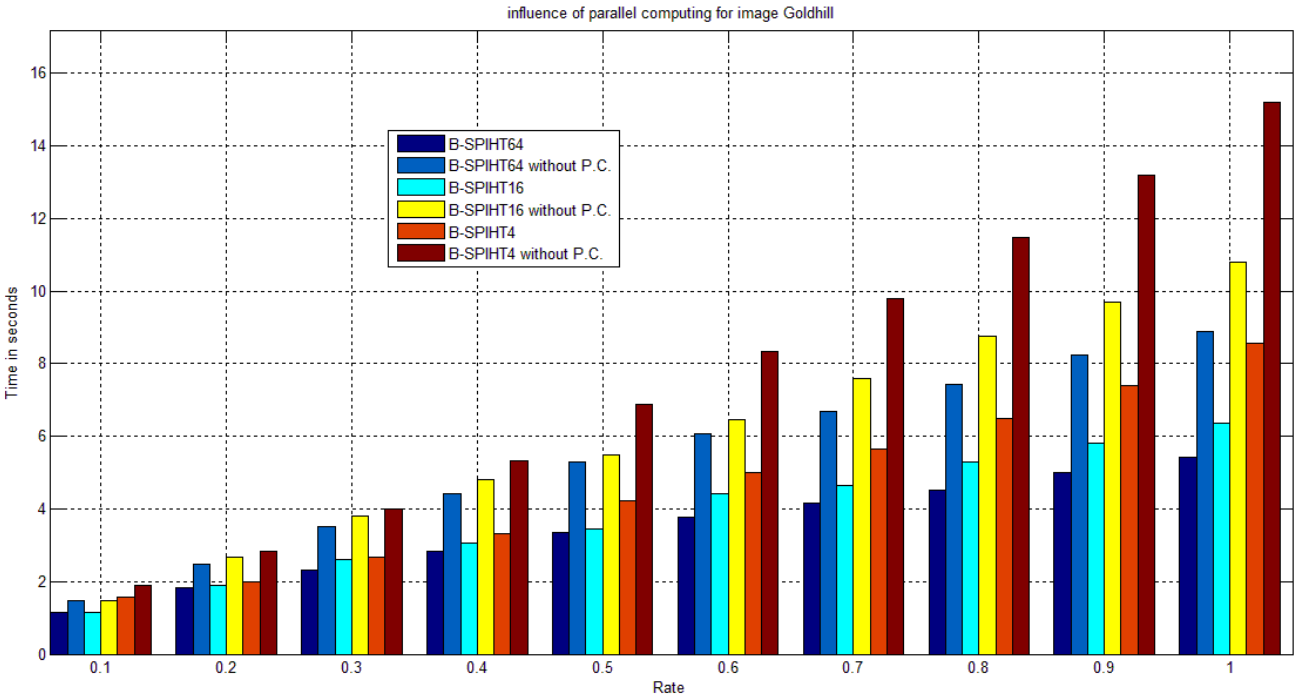


Figure 4.42: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Goldhill

In case B-SPIHT the same observations are made, B-SPIHT with parallel computing reduces the computation time significantly more than without parallel computing.

4.5.7. INFLUENCE OF IMAGE SIZE ON B-SPIHT

Since we executed and tested the algorithms using only images of size 512x512, in this section we will examine how B-SPIHT does perform on different sizes of an image. We chose image man (1024x1024) and resized it to 128x128, 256x256, 512x512 and ran B-SPIHT4 and SPIHT. Both of the algorithms were at 1bpp. The results are given below:

Image size	***128x128***	***256x256***	***512x512***	***1024x1024***
B-SPIHT4 (PSNR)	**27.28 dB**	**30.73 dB**	**34.56 dB**	**36.58 dB**
B-SPIHT4 (time)	**0.52 s**	**1.72 s**	**8.58 s**	**83.34 s**
SPIHT (PSNR)	**29.34 dB**	**32.14 dB**	**35.44 dB**	**36.82 dB**
SPIHT (time)	**0.65 s**	**3.8 s**	**29.28 s**	**742.25 s**

Table 4.34: the influence of parallel computing for B-SPIHT(4, 16 and 64) on image Lena

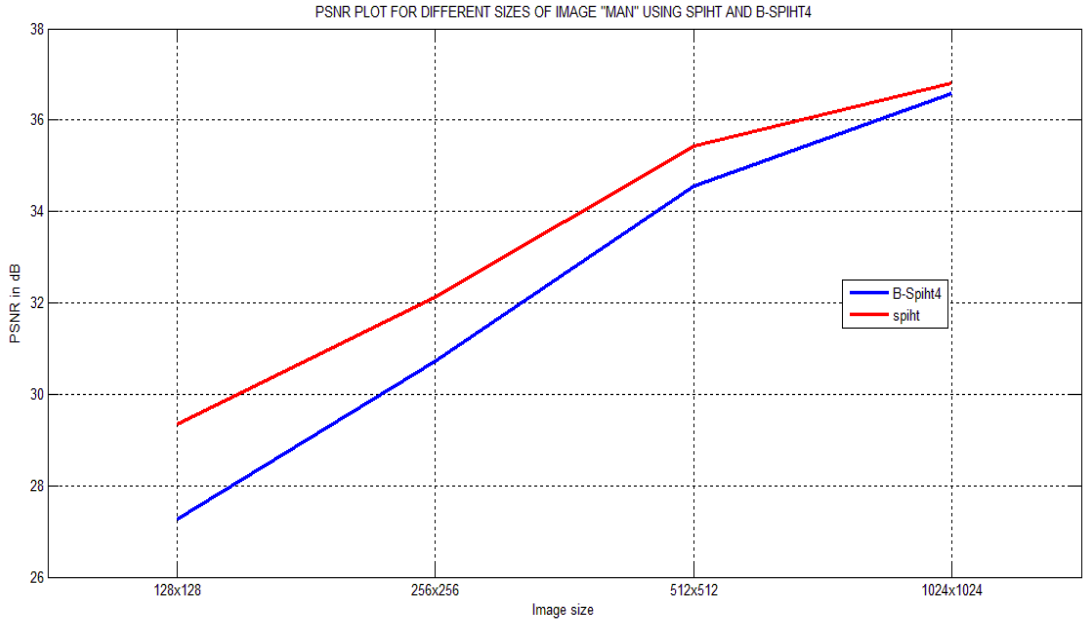


Figure 4.43: PSNR of SPIHT and B-SPIHT for different sizes of image man at 1bpp

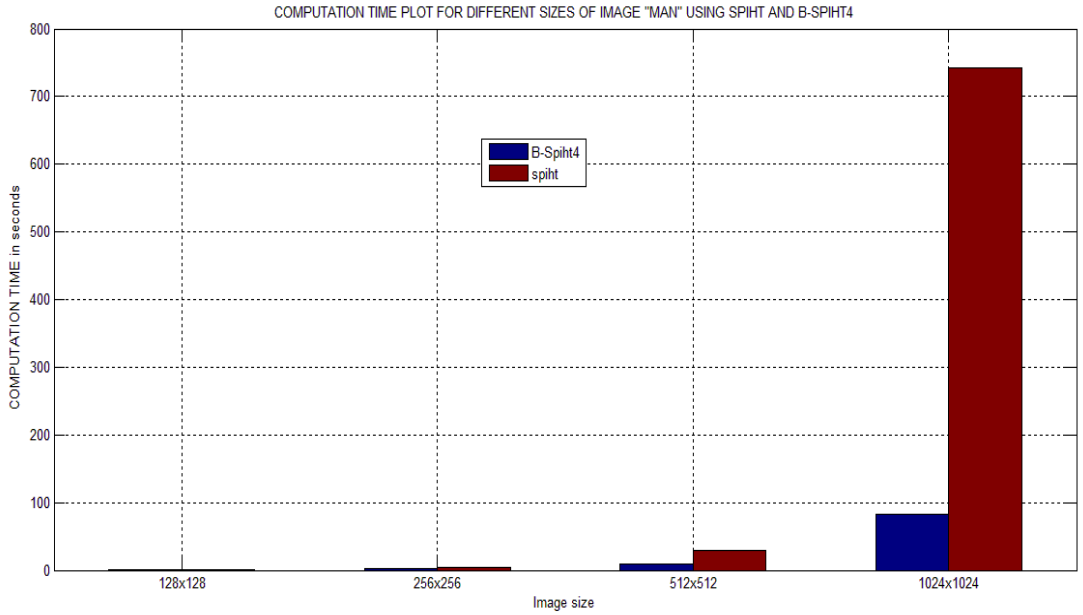


Figure 4.43: PSNR of SPIHT and B-SPIHT for different sizes of image man at 1bpp

From the results we conclude that the bigger is the size of the image the better it is to use B-SPIHT4, PSNR wise it reaches the performance of SPIHT the larger is the image and it reduces the computation time considerably.

4.6. Discussion of the results

At first, our study resided on the use of blocks in EZW and SPIHT on the level of the original image and compare the results to SPECK algorithm; the implementation was referred to in chapters as “the first method”. Unfortunately, the results were not satisfying, where block artifacts were clearly visible for any block size we used. See the images in the annex section. We later considered the wavelet transform of the image as an image itself, divide it and apply the local EZW/SPIHT to the obtained blocks. This method is the basis of the our research.

In section 4.4, we clearly noticed that the filters bior6.8 and coif5 give better performances then the other filters. Bior6.8 gave better performances on both images Lena and Goldhill, thus we decided to use filter bior6.8 throughout all the computations.

In chapter 3, we used the famous Shapiro matrix to explain both B-EZW and B-SPIHT. For EZW with threshold 32, the bit stream of EZW was 20; in comparison, the B-EZW algorithm generated a bit stream of 40 symbols, thus it was expected to perform badly on test images. The bad performance on the example matrix showed that dividing any matrix could lead to the loss of the root-offspring long chains that are the core of the EZW algorithm.

Also in chapter 3, we noticed that B-SPIHT outputted 47 bits instead of 29, but it was far being the double of symbols like in the case of EZW. In addition, we argue that the bigger the image is, the better the algorithm performs due to the 4^{th} matrix usually has all coefficients less than the threshold.

From section 4.5 we run our algorithms with different block sizes. Since the test images are all 512*512 we chose 64, 128, 256 as the test block sizes which consequently are 64, 16 and 4 divisions respectively. Choosing a block size of 64, gave 64 submatrices, a block size of 128 pixels gave 16 submatrices and a 256 block size gave 4 submatrices. We notice that the B-EZW4 and B-SPIHT4 always had better performances than the other algorithms (EZW64, EZW128, SPIHT64 and SPIHT128) therefore we decided to use the B-EZW4 and B-SPIHT4 algorithms for the comparison with the original algorithms, EZW, SPIHT and SPECK.

The B-EZW4 algorithm performed badly due to the splitting in blocks, where there are huge losses in PSNR for the of not being able to make use of the property of a long chain of descending coefficients that could be coded as ZeroTree, which leads to output a bits stream that is longer than necessary.

As for the B-SPIHT4 algorithm, the loss in the PSNR is attributed to the losses that result from the wavelet transforms due to it being a lossy transform and irreversible. The more blocks are used to do the transform, the more we lose in terms of MSE. B-SPIHT4 uses four wavelet transforms, so instead of using one forward and one reversed transform that is lossy, we used five transforms.

In section 4.5.3 we compared our methods with EZW SPECK and SPIHT, the latter gave the best performances in all the test images, followed by SPECK and then B-SPIHT4. B-SPIHT4 performed better than the original EZW in images Lena and Barbara. As it was expected B-EZW4 came in last.

Computation time wise, the proposed algorithms performed well than the originals. Even the B-SPIHT was faster than all the other algorithms; a huge reduction in computation time up to 250%, when the speck algorithm took more than 50 seconds our algorithm B-SPIHT took less than 10 seconds due to the parallel computing feature.

After executing the B-EZW with and without parallel computing, we noticed clearly its effect on computation time. In all tests, B-EZW was faster where parallel computing reducing a considerable amount of calculation time. In case B-SPIHT the same observations are made, B-SPIHT with parallel computing reduced the computation time significantly more than without the use of parallel computing.

In terms of image quality, our methods gave acceptable results with a with 4 divisions to the wavelet transform when going to higher compression rates. In low compression rates, from 0.4bpp and less, they performed badly with sometimes too much noise the image is not acceptable.

Finally, we tested B-SPIHT4 on different sizes of the same image, we concluded that the bigger is the size of the image the better it is to use B-SPIHT4, PSNR wise, it reaches the performance of SPIHT in larger sizes of an image. In addition to that it reduces the computation time considerably.

4.7. Conclusion

In this chapter, we validated our methods B-EZW and B-SPIHT. We analyzed the following: the choice of the wavelet filter, we compared B-EZW with EZW, B-SPIHT and

SPIHT with PSNR (Peak Signal to noise ratio) and computation time. We also compared the algorithms with SPECK; we investigated the effect of parallel computing and the effect of the size of the images on the compression quality. We concluded that our method performed badly when using many blocks but computationally faster. When using B-SPIHT with four blocks, our algorithm was efficient as the other methods and much faster due to the parallel computing.

General conclusion

General Conclusion

The purpose of this thesis has been the research and the introduction of modifications to improve the compression ratio and execution time of images compression/decompression in a digital encoder. The idea was to use blocks. Our choices were oriented to solutions that combine the theory of parallel computation.

We studied wavelet compression algorithms and embedded encodings (EZW, SPIHT and SPECK). The encoder offers an embedded property of progressive transmission of the encoded image while providing excellent rate/distortion performance.

In this thesis, we proposed a new algorithm to optimize the classical coding algorithm EZW and SPIHT. Our so-called Block EZW (B-EZW) and Block SPIHT (B-SPIHT) methods have two characteristics:

- They consist of splitting the image to 2^n parts to use the parallel computation, to enormously reduce the execution time.
- They have a compression that generates images with an acceptable visual quality

In Chapter 4, we then demonstrated the robustness in relative of the image compression rate/quality of the B-SPIHT4 algorithm. B-EZW is much lower in most cases when compared to the original EZW algorithm, the B-SPIHT is slightly less but much faster than the original algorithm SPIHT and comparable with that of SPECK algorithm.

Looking ahead, we propose on the one hand the application of the principles of B-SPIHT to the coders SPECK and JPEG2000. This principle could reduce tremendously the execution time. In addition, we can study the use of a different threshold for different submatrices to better encode all the coefficients.

References

References

[1] ZITOUNI Athmane “Ondelettes et techniques de compression d’images numériques” Thèse de Doctorat en Sciences en Electronique, biskra University 2013.

[2] C. Shannon, “A mathematical theory of communication”. Bell System Technical Journal, pp. 379–423, Juillet 1948.

[3] D.A. Huffman, "A method for the construction of minimum-redundancy codes Proceedings of the I.R.E., sept 1952, pp 1098-1102

[4] Tutorial Arithmetic Coding, Yu-Yun Chang, Graduate Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan

[5] OUAFI Abdelkrim, « Compression d'images avec pertes par codages imbriqués, Proposition d’une optimisation de l’algorithme EZW » Thèse de Doctorat en Sciences en Electronique, biskra University 2009.

[6] W. Chen, C.H. Smith, and S.C. Fralick. A fast computational algorithm for the discrete cosine transform. IEEE Trans. on Communications, COM-25:1004_1009, 1977.

[7] A. Islam and W. A. Pearlman, “An embedded and efficient low-complexity hierarchical image coder,” Visual Communications and Image Processing ’99, Proceedings of SPIE, vol. 3653, pp. 294–305, Jan. 1999

[9] A. Said and W.A. Pearlman, “Low-Complexity Waveform Coding via Alphabet and Sample-Set Partitioning,”Visual Communications and Image Processing ’97, Proceedings of SPIE, vol. 3024, pp. 25-37, Feb. 1997.

[10] J. Andrew, “A simple and efficient Hierarchical Image coder,”IEEE Internationall Conf. on Image Proc. (ICIP-97), vol. 3, pp. 658–661, Oct. 1997.

[11] J. Spring, J. Andrew, and F. Chebil, “Nested Quadratic Splitting,”ISO/IEC/JTC1/SC29, WG1 N1191, July 1999.

[12] Daubechies, I., Grossmann, A,, and Meyer. Y., Painless non-orthogonal expansions, J. Math. Phys. 27, 1986, pp. 1271-1283.

[13] I. Daubechies, "Orthonormal bases of compactly supported wave-lets," Commun. Pure Appl. Math., vol. 41, pp. 909-996, 1988,

[14] A.N. Akansu and M.J.T. Smith, Subband and Wavelet Transforms: Design and Applications, Kluwer Academic Publishers, 1995.

[15] S. G. Mallat and S. Zhong, “Characterization of signals from multiscale edges,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 14, no. 7, pp. 710– 732, Jul. 1992.

Annex

Annex

In what follows, we show the images obtained by the original idea of splitting the original image in to blocks and running the EZW or SPIHT on each block separately. As mentioned before, because the results were not satisfying as block artifacts are clearly and disturbingly visible we discarded the idea.

Figure a) result obtained from dividing the image into 4 blocks with SPIHT, PSNR=24.65dB

Figure b) result obtained from dividing the image into 16 blocks with SPIHT, PSNR=22.89dB

Figure c) result obtained from dividing the image into 4 blocks with EZW, PSNR=23.15dB

Figure d) result obtained from dividing the image into 16 blocks with EZW, PSNR=20.74dB