People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University of Adrar

Faculty of Science and Technology

Department of Mathematics and Computer Science

A Memoir Submitted for Master Degree in Computer Science (LMD System)

Networks and Computer Network Security Option

Theme:

Image Compression Based On Polynomial Interpolation

Prepared And Presented By: Nasreddine KAROUR

Examination Committee:

– Mohamed Amine **Cheragui**	President
– Mohammed **Demri**	Examinator1
– El Mamoum **Mamouni**	Examinator2
– Mohammed **Omari**	Supervisor

June 2015

Abstract

In the way of having well shaped images with fewer and fewer sizes, and since images are used in many fields nowadays many image compression techniques have been proposed. In this paper, we present a new image compression technique based on polynomial interpolation. It is based on the interpolation of image pixels so they would be represented by polynomials. We also proposed some enhancements, which are presented as techniques, in order to make our approach reach better compression measures. The final results of our method show competitive results compared to the common used compression techniques.
Key Terms: Image Compression, PSNR, Compression Ratio, Polynomial Interpolation.

Dedicates

I am dedicating this project to the dearest persons in our life:
To my parents, who have helped us to get to where am I now, and without them I would never reach here, may ALLAH save you for me.
To my brothers, Izzou, Memdouh and Nadir.
To my beloved sister Amel.
To anyone who have helped me in anyway.

Nasro

Acknowledgments

I are grateful to the *ALLAH* for the good health and well being that were necessary to complete this work

I offer our sincerest gratitude to my supervisor, Dr.**Mohammed OMARI**, who has supported us throughout my work with his patience and knowledge whilst allowing us the room to work in may own way.

I would like to thank Mr.**Mohamed CHERAGUI** for his will of for kindly do me the honor of being the president of the jury of this work.

I express my appreciation to Mr.**El Mamoun MAMOUNI** and Mr.**Mohamed DEMRI** for accepting being part of the jury and examine my work.

A special thanks to Mr.**Saleh YAICHI** for being an invited member of the jury.

I wish to express our sincere thanks to all the teachers of the faculty of Science and Technology and to all my classmates in the computer science department.

I also place on record, my sense of gratitude to one and all, who directly or indirectly, have let their hand in this venture.

Table of Contents

List of Figures

List of Tables

General Introduction

Multimedia (graphics, audio, video) data has become part of our daily lives, we take it using many devices such as digital cameras, mobile phone cameras, tablets and audio recorder. We also share them with friends, family and other persons. However it requires considerable storage capacity and transmission bandwidth in order to store it or send it.

That is why good compression techniques had to be introduced so this multimedia would be stored in less space and requires less bandwidth to be sent. These techniques can be divided into two types, lossless where the compressed and decompressed data are the same with a small difference between their size, and lossy in witch the difference between compressed and decompressed data has a tolerable ratio but also achieve higher size difference.

Image compression is not an exception, during years ago many techniques have been introduced. The reduction in image size allows more images to be stored in a given amount of disk or memory space. It also reduces the time required for images to be sent over the Internet or downloaded from Web pages. Compression is achieved by the removal of one or more of the three basic data redundancies

1. Coding Redundancy
2. Inter pixel Redundancy
3. Psycho visual Redundancy

With this memoir we are introducing a new lossy image compression technique based on polynomial interpolation, we tried to take the advantage of saving only the polynomial interpolation coefficients instead of many pixel values. For a given curve has n values we can extract a polynomial of size i with $n \gg i$, considering the interpolated polynomial this curve's function, we can evaluate it, so we regenerate the original curve.

Furthermore, new enhancements to this technique have been proposed trying to reach tolerable compression ratios with image of better qualities, such as using the sorting technique in order to make the image pixels much more ordered. Also the use of interpolation error so the reconstructed image would be as close as possible to the original one.

This memoir has four chapters, the first one contains digital image definition and some related terms to it including its types and its proprieties. The second one provides general information

about the commonly used image compression techniques, image compression types and some compression metrics.

The third chapter presents the implemented techniques, starting from the use of the polynomial interpolation until reaching the final enhancement. While the fourth one, contains the achieved results using each proposed technique. Last but not least, we have a conclusion and future work.

Chapter 1

Images And Digital Images

1.1 Introduction

In this chapter we are introducing the basics of digital images, which are their definition, their characteristics such as the pixel and the resolution. And we mention some common used digital image formats.

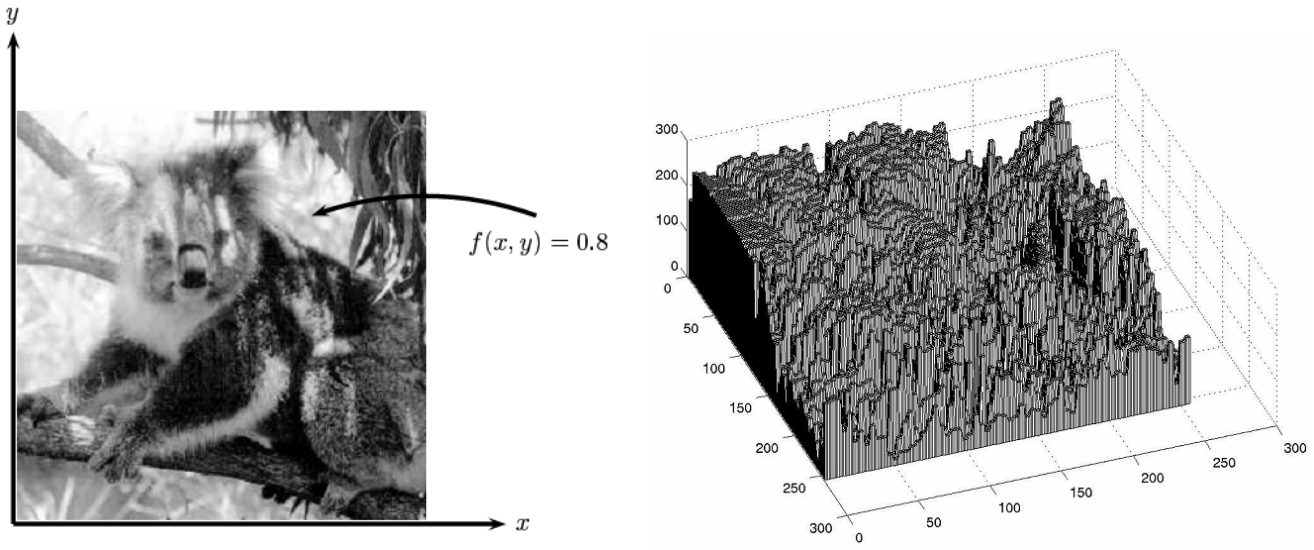
1.2 Digital Images

Are electronic snapshots taken of a scene or scanned from documents, such as photographs, manuscripts, printed texts, and artwork. The digital image is sampled and mapped as a grid of dots or picture elements (pixels). Each pixel is assigned a tonal value (black, white, shades of gray or color), which is represented in binary code (zeros and ones). The binary digits ("bits") for each pixel are stored in a sequence by a computer and often reduced to a mathematical representation (compressed).

The bits are then interpreted and read by the computer to produce an analog version for display or printing. Digital image could be considered as being a two dimensional function, where the function values give the brightness of the image at any given point, as shown in figure 1.1a. We may assume that in such an image brightness values are normalized, so they have real numbers in the range 0.0 (black) to 1.0 (white).

The ranges of x and y will clearly depend on the image, but they can take all real values between their minima and maxima. Such a function can of course be plotted, as shown in figure 1.1b. However, such a plot is of limited use to us in terms of image analysis. The concept of an image as a function, however, will be vital for the development and implementation of image processing techniques.

A digital image differs from a photo in that the x, y, and $f(x, y)$ values are all discrete. Usually they take on only integer values, so the image shown in figure 1.1a will have x and y ranging from 1



(a) *(b)*

Figure 1.1 – An image as a function (a), and plotted as a function of two variables (b)

to 256 each, and the brightness values also ranging from 0 (black) to 255 (white). A digital image, as we have seen above, can be considered as a large array of sampled points from the continuous image, each of which has a particular quantized brightness; these points are the pixels which constitute the digital image. The pixels surrounding a given pixel constitute its neighborhood.

1.3 Digital Images characteristics

1.3.1 Pixel

In digital image, a pixel is a single point in a raster image. It is the smallest unit of picture that can be controlled, and is the smallest addressable screen element as shown in figure 1.2. Each pixel has its own address. The address of a pixel corresponds to its coordinates. They are usually arranged in a 2-D grid, and are often represented with dots or squares. Each pixel is a sample of an original image. More samples typically provide more accurate representations of the original. A

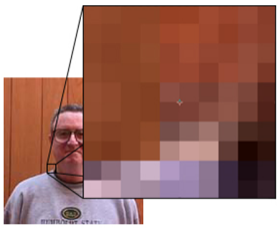


Figure 1.2 – Pixel is smallest element of an image

neighborhood can be characterized by its shape in the same way as a matrix: we can speak, for

example, 3×3 neighborhood, or of a 5×7 neighborhood. Except in very special circumstances, neighborhoods have odd numbers of rows and columns; this ensures that the current pixel is in the center of the neighborhood. An example . If a neighborhood has an even number of rows or columns (or both), it may be necessary to specify which pixel in the neighborhood is the "current pixel".

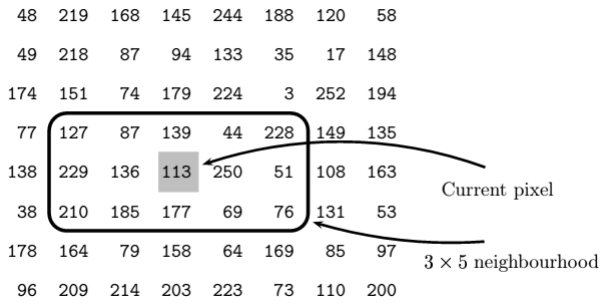


Figure 1.3 – Pixels, with a neighborhood

1.3.2 Resolution

Resolution is determined by the size of the units of information representing an image. A pixel is a unit of information displayed on a monitor. Each pixel holds a defined amount of information stored on your disk. An image of a given area will become more detailed as more pixels are used to describe it.

Resolution can be measured in many ways:

1. Samples per inch (spi, scanners)
2. Pixels per inch (ppi monitors)
3. Dots per inch (dpi, printers)

More pixels in a given area will give you a smoother, more detailed image but it will also give you a much larger file.

Fortunately, as the pixels become smaller and smaller, the difference between high resolution and extremely high resolution are not discernible by the human eye.

1.3.3 Image File Sizes

Image files tend to be large. We shall investigate the amount of information used in different image type of varying sizes. For example, suppose we consider a 512×512 binary image. The number of

bits used in this image (assuming no compression, and neglecting, for the sake of discussion, any header information) is

$$\begin{aligned} 512 \times 512 \times 1 &= 262,144 \\ &= 32,768 \; Bytes \\ &= 32 \; KB \end{aligned}$$

A grayscale image of the same size requires:

$$\begin{aligned} 512 \times 512 \times 8 &= 2,097,152 \\ &= 262,144 \; Bytes \\ &= 256 \; KB \end{aligned}$$

If we now turn our attention to color images, each pixel is associated with three bytes of color information. A 512×512 image thus requires

$$\begin{aligned} 512 \times 512 \times 3 \times 8 &= 6,291,456 \\ &= 786432 \; Bytes \\ &= 768 \; KB \end{aligned}$$

1.4 Types of Digital Images

1.4.1 Binary

Each pixel is just black or white. Since there are only two possible values for each pixel, we only need one bit per pixel. Such images can therefore be very efficient in terms of storage. Images for which a binary representation may be suitable include text (printed or handwriting), fingerprints, or architectural plans.

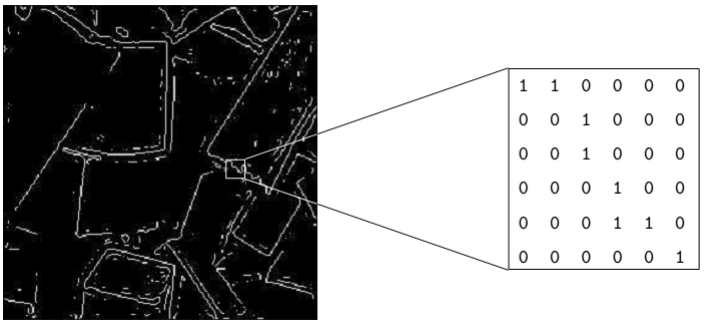


Figure 1.4 – Example of binary image

1.4.2 Grayscale

Each pixel is a shade of gray, normally from 0 (black) to 255 (white). This range means that each pixel can be represented by eight bits, or exactly one byte. This is a very natural range for image file handling. Other grayscale ranges are used, but generally they are a power of 2. Such images arise in medicine (X-rays), images of printed works, and indeed 256 different gray levels is sufficient for the recognition of most natural objects.

Figure 1.5 – Example of grayscale image

1.4.3 True color, or RGB

Here each pixel has a particular color; that color being described by the amount of red, green and blue in it. If each of these components has a range 0-255, this gives a total of $256^3 = 17{,}777{,}216$ different possible colors in the image. This is enough colors for any image. Since the total number of bits required for each pixel is 24, such images are also called 24-bit color images.

Such an image may be considered as consisting of a“stack”of three matrices; representing the red, green and blue values for each pixel. This means that for every pixel there correspond three values.

1.4.4 Indexed

Most color images only have a small subset of the more than sixteen million possible colors. For convenience of storage and file handling, the image has an associated color map, or color palette, which is simply a list of all the colors used in that image. Each pixel has a value which does not give its color (as for an RGB image), but an index to the color in the map.

It is convenient if an image has 256 colors or less, for then the index values will only require one byte each to store. Some image file formats (for example, Compuserve GIF), allow only 256

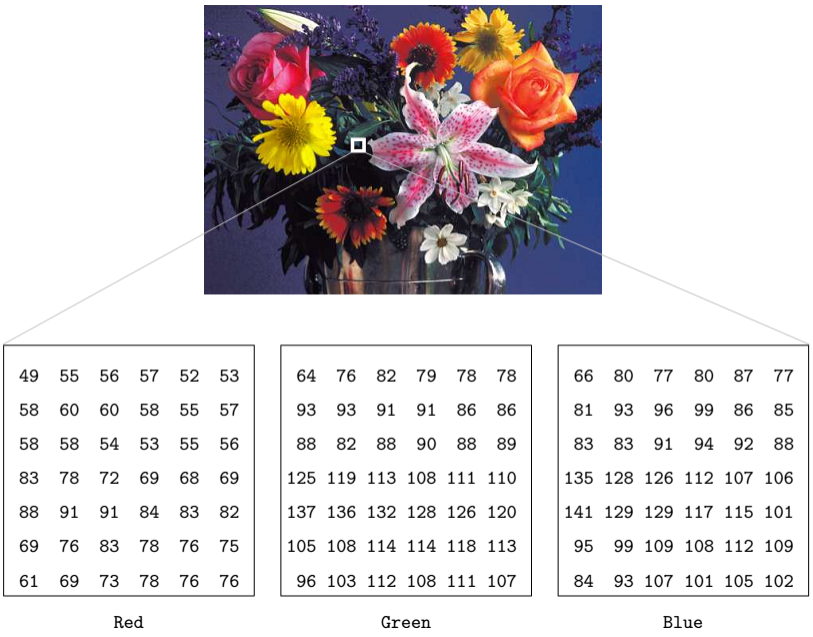


Figure 1.6 – Example of color image

colors or fewer in each image, for precisely this reason.

Figure 1.7 – Example of indexed image

Chapter 2

Image Compression

2.1 Introduction

Data compression involves encoding information using fewer bits than the original representation [1]. Compression is useful because it helps reduce resource usage, such as data storage space or transmission capacity. Image compression is a major field of data compression, in this chapter we are introducing the need of compression, Image compression definition, its types as long as with their techniques and widely common image formats.

2.2 The Need Of Compression

Table 2.1 show the qualitative transition from simple text to full-motion video data and the disk space, transmission bandwidth, and transmission time needed to store and transmit such uncompressed data [3].

Table 2.1 – Multimedia data types and uncompressed storage space, transmission bandwidth, and transmission time required. The prefix kilo-denotes a factor of 1000 rather than 1024.

Multimedia Data	**Size/Duration**	**Bits/Pixel or Bits/Sample**	**Uncompressed size (B for bytes)**	**Transmission Bandwidth (b for bits)**	**Transmission Time (using a 2Mbps Modem)**
A page of text	11" x 8.5"	Varying resolution	4-8 KB	32-64 Kb/page	0.02 - 0.03 sec
Telephone quality speech	10 sec	8 bps	80 KB	64 Kb/sec	0.31 sec
Grayscale Image	512×512	8 bpp	262 KB	2.1 Mb/image	1.02 sec
Color Image	512×512	24 bpp	786 KB	6.29 Mb/image	3 sec
Medical Image	2048×1680	12 bpp	5.16 MB	41.3 Mb/image	20.16 sec
SHD Image	2048×2048	24 bpp	12.58 MB	100 Mb/image	49.14 sec
Full-motion Video	640×480, min 1 min (30 frames/sec)	24 bpp	1.66 GB	221 Mb/sec	1 hrs 48 min

2.3 Image Compression Definition

Image compression addresses the problem of reducing the amount of information required to represent a digital image. It is a process intended to yield a compact representation of an image, thereby reducing the image storage transmission requirements. Every image will have redundant data. Redundancy means the duplication of data in the image. Reduction of redundancy provides helps to achieve a saving of storage space of an image. Image compression is achieved when one or more of these redundancies are reduced or eliminated.

2.3.1 Image Redundancies

In image compression, three basic data redundancies can be identified and exploited. Compression is achieved by the removal of one or more of the three basic data redundancies. which are

1. Coding Redundancy
2. Inter-pixel Redundancy
3. Perceptual Redundancy

Coding redundancy occurs when the codes assigned to a set of events such as the pixel values of an image have not been selected to take full advantage of the probabilities of the events.

Inter-pixel redundancy usually results from correlations between the pixels. Due to the high correlation between the pixels, any given pixel can be predicted from its neighboring pixels.

Perceptual redundancy is due to data that is ignored by the human visual system. In other words, all the neighboring pixels in the smooth region of a natural image have a high degree of similarity and this insignificant variation in the values of the neighboring pixels is not noticeable to the human eye [6].

The above examples clearly illustrate the need for sufficient storage space, large transmission bandwidth, and long transmission time for image, audio, and video data. At the present state of technology, the only solution is to compress multimedia data before its storage and transmission, and decompress it at the receiver for play back [9].

2.3.2 Image Compressing System

Image compression techniques reduce the number of bits required to represent an image by taking advantage of those redundancies. An inverse process called decoding is applied to the compressed data to get the reconstructed image. The objective of compression is to reduce the number of bits as much as possible, while keeping the resolution and the quality of the reconstructed image as close to the original image as possible.

Image compression systems are composed of two distinct structural blocks: an encoder and a decoder, as shown in figure 2.1.

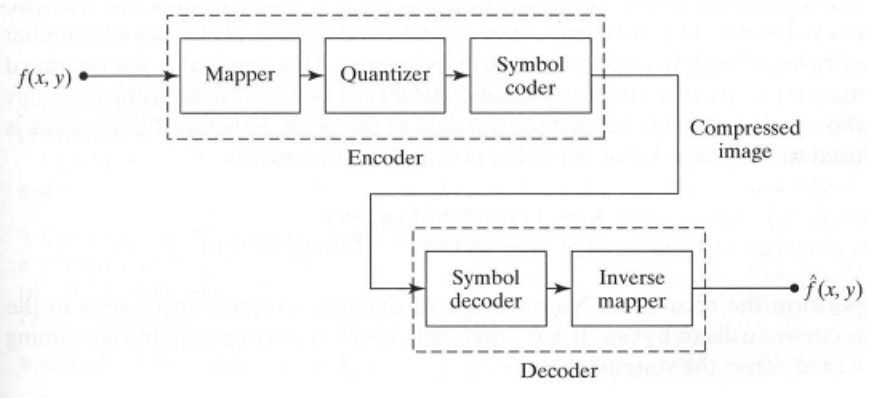


Figure 2.1 – Image compression system

2.4 Image Compression Types

A compression algorithm is “lossless” (or reversible) if the decompressed image is identical with the original. Respectively, a compression method is “lossy” (or irreversible) if the reconstructed image

is only an approximation of the original one [22].

2.4.1 Lossless Image Compression

Lossless compression compresses the image by encoding all the information from the original file, so when the image is decompressed, it will be exactly identical to the original image. As follows some lossless image compression techniques.

2.4.1.1 Run Length Encoding

Run Length Encoding: Run-length encoding (RLE) is a very simple form of image compression in which runs of data are stored as a single data value and count, rather than as the original run. It is used for sequential [19] data and it is helpful for repetitive data. In this technique replaces sequences of identical symbol (pixel), called runs. The Run length code for a grayscale image is represented by a sequence V_i, R_i where V_i is the intensity of pixel and R_i refers to the number of consecutive pixels with the intensity Vi as shown in the figure. This is most useful on data that contains many such runs for example, simple graphic images such as icons, line drawings, and animations. It is not useful with files that don't have many runs as it could greatly increase the file size [7].

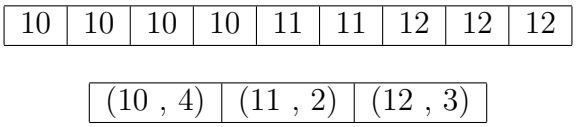


Figure 2.2 – Run length encoding

2.4.1.2 Entropy Encoding

In information theory an entropy encoding is a lossless data compression scheme that is independent of the specific characteristics of the medium. One of the main types of entropy coding creates and assigns a unique prefix-free code for each unique symbol that occurs in the input. These entropy encoders then compress the image by replacing each fixed-length input symbol with the corresponding variable-length prefix free output codeword.

2.4.1.3 Huffman Encoding

In computer science and information theory, Huffman coding is an entropy encoding algorithm used for lossless data compression. It was developed by Huffman. Huffman coding [12] today is often used as a **back-end** to some other compression methods. The term refers to the use of a variable-length code table for encoding a source symbol where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible

value of the source symbol. The pixels in the image are treated as symbols. The symbols which occur more frequently are assigned a smaller number of bits, while the symbols that occur less frequently are assigned a relatively larger number of bits. Huffman code is a prefix code. This means that the (binary) code of any symbol is not the prefix of the code of any other symbol [24]. The following example bases on a data source using a set of five different symbols. The symbol's frequencies are:

Symbol	Frequency
A	24
B	12
C	10
D	8
E	8

Total 186 bits (with 3 bits per code word)

The two rarest symbols **E** and **D** are connected first, followed by **C** and **D**. The new parent nodes have the frequency 16 and 22 respectively and are brought together in the next step. The resulting node and the remaining symbol 'A' are subordinated to the root node that is created in a final step.

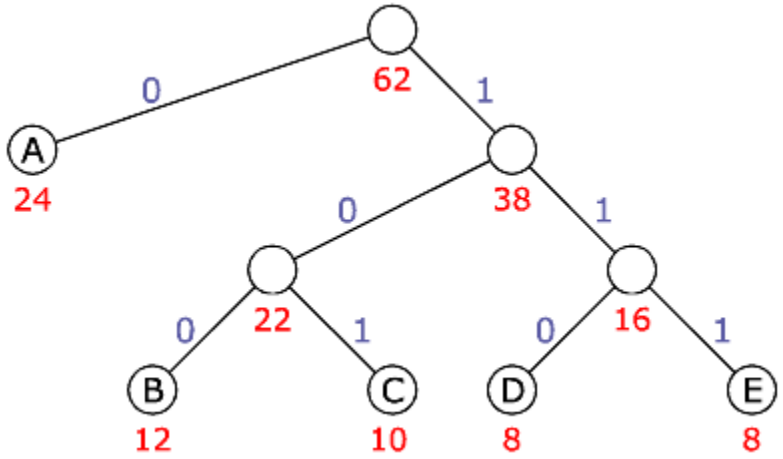


Figure 2.3 – Code Tree according to Huffman

Symbol	Frequency	Code	Code Length	total Length
A	24	0	1	24
B	12	100	3	36
C	10	101	3	30
D	8	110	3	24
E	8	111	3	24

Total 138 bits

2.4.1.4 Arithmetic Coding

Arithmetic coding is a form of entropy encoding used in lossless data compression. Normally, a string of characters such as the words "hello there" is represented using a fixed number of bits per character, as in the ASCII code. When a string is converted to arithmetic encoding, frequently used characters will be stored with little bits and not-so-frequently occurring characters will be stored with more bits, resulting in fewer bits used in total. Arithmetic coding differs from other forms of entropy encoding such as Huffman coding in that rather than separating the input into component symbols and replacing each with a code, arithmetic coding encodes the entire message into a single number [24] [15].

2.4.1.5 Lempel-Ziv-Welch Coding: Lempel-Ziv-Welch

(LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978 [16]. LZW is a dictionary based coding. Dictionary based coding can be static or dynamic. In static dictionary coding, dictionary is fixed when the encoding and decoding processes. In dynamic dictionary coding, dictionary is updated on fly. The algorithm is simple to implement, and has the potential for very high throughput in hardware implementations. It was the algorithm of the widely used UNIX file compression utility compress, and is used in the GIF image format. LZW compression became the first widely used universal image compression method on computers. A large English text file can typically be compressed via LZW to about half its original size [24].

2.4.2 Lossy Image Compression

Lossy compression as the name implies leads to loss of some information. The compressed image is similar to the original uncompressed image but not just like the previous as in the process of compression [4] some information concerning the image has been lost. They are typically suited to images. The most common example of lossy compression is JPEG. An algorithm that restores the presentation to be the same as the original image are known as lossy techniques. Reconstruction of the image is an approximation of the original image,therefore the need of measuring of the quality of the image for lossy compression technique. Lossy compression technique provides a higher compression ratio than lossless compression.Some loss of information can be acceptable for the following three reasons [23]:

1. Significant loss can often be tolerated by the human visual system without interfering with perception of the scene content.

2. In most cases, digital input to the compression algorithm itself is an imperfect representation of the real world scene. This is certainly true when the image sample values are quantized version of the real-valued quantities.

3. Lossless compression is usually incapable of achieving the high compression requirements of many storage and distribution applications.

Major performance considerations of a lossy compression scheme include[24]:

- Compression ratio
- Signal to noise ratio
- Speed of encoding and decoding

Here are 2 used algorithms in lossy image compression.

2.4.2.1 Scalar Quantization

The scalar quantizer is very useful method for lossy compression schemes. The input of quantizer is the original data, and the output is always one among a finite number of levels. Quantizer can be described as a function that maps each element in a subset of the real line to a particular value in that subset. Such function has discrete set as output values, and this set is usually finite. Obviously, this is a process of approximation, and a good quantizer is one which represents the original signal with minimum loss or distortion. A quantizer can be specified by its input partitions and output levels (also called reproduction points). If the input range is divided into parts with equal spacing, then the quantizer is termed as a uniform quantizer, otherwise it is termed as a non-uniform quantizer. A uniform quantizer can be easily specified by its lower bound and the step size, or its number of output levels when range is divided into parts with equal size. Also, implementing a uniform quantizer is easier than a non-uniform quantizer. Let us consider scalar quantizer with output levels M in general case. Partitioning the real line into M disjoint intervals is denoted in a following way [20]:

$$I_q = [t_q, t_{q+1}), q = 0, 1, ..., M-1 \tag{2.1}$$

with

$$-\infty = t_0 < t_1 < ... < t_M = +\infty \tag{2.2}$$

2.4.2.2 Vector Quantization

By grouping input sequences together and encoding them as a single block, an efficient lossy as well as lossless compression algorithms could be obtained. There are also some quantization

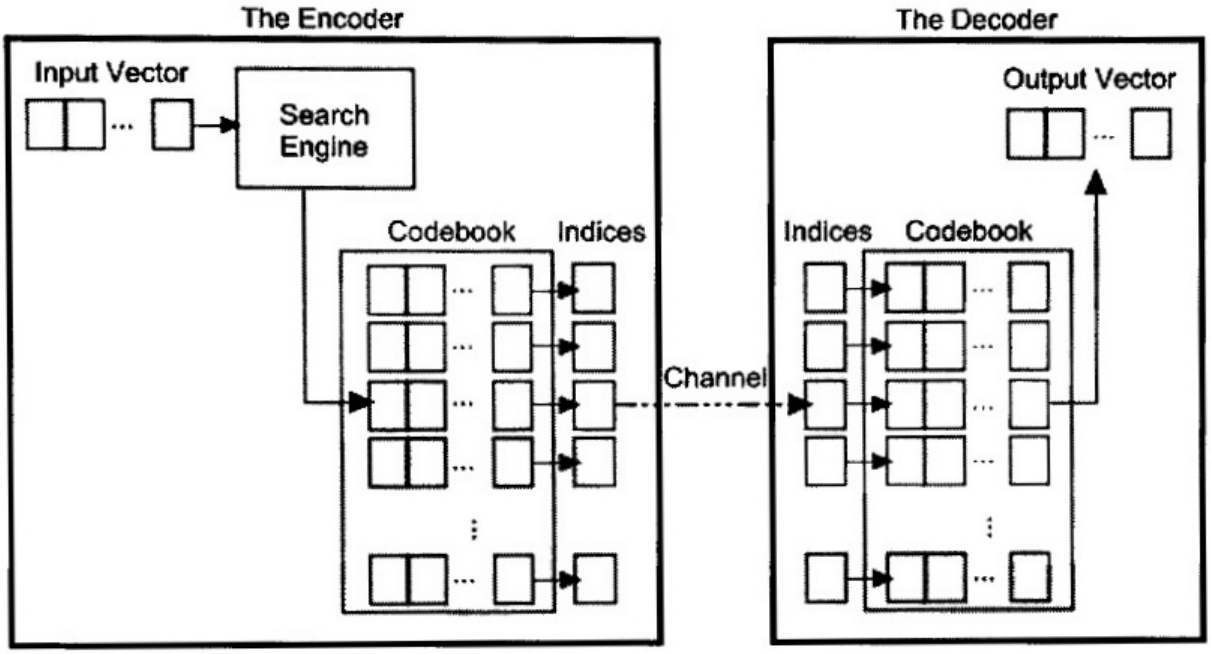


Figure 2.4 – Vector quantization procedure

techniques that operate on blocks of data. We can look at these blocks as vectors. This kind of quantization technique is called vector quantization. In vector quantization we need to build a representative set for the input sequences. If there is an input sequence, we can express it as one of the elements in the representative set. Figure 3.1 shows the block diagram of vector quantization. In vector quantization, we first group the input into blocks or vectors. All the operations in vector quantization will be applied to whole vectors. At both the encoder and decoder sides, there is a set of vectors called the codebook. The vector in the codebook is called the codevector or codeword. Normally, the size of the codevector is the same as the input vector. There is a search engine in the encoder to find the codevector that can best match the input vector. The input vector is compared with each codevector in the codebook. The best match codevector is the quantized value of that input vector. After finding the best match codevector, we just need to transmit the binary index of the codevector to the decoder. Since there is a same codebook at the decoder side, after receiving the index the decoder can recover the codevector. This recovered codevector is sent out as the reconstructed input vector [17].

2.5 Image Compression Formats

2.5.1 BMP (bitmap)

It is a bitmapped graphics format used internally by the Microsoft Windows graphics subsystem (GDI), and used commonly as a simple graphics file format on that platform. It is an uncompressed format.

2.5.2 PNG (Portable Network Graphics) (1996)

A bitmap image format that employs lossless data compression. PNG was created to both improve upon and replace the GIF format with an image file format that does not require a patent license to use. PNG supports palette based (with a palette defined in terms of the 24 bit RGB colors), grayscale and RGB images. PNG was designed for distribution of images on the internet not for professional graphics and as such other color spaces

2.5.3 TIFF (Tagged Image File Format) (last review 1992)

This is a file format for mainly storing images, including photographs and line art. It is one of the most popular and flexible of the current public domain raster file formats. Originally created by the company Aldus, jointly with Microsoft, for use with PostScript printing, TIFF is a popular format for high color depth images, along with JPEG and PNG. TIFF format is widely supported by image-manipulation applications, and by scanning, faxing, word processing, optical character recognition, and other applications.

2.5.4 JPEG (Joint Photographic Experts Group) (1992)

JPEG is an algorithm designed to compress images with 24 bits depth or grayscale images. It is a lossy compression algorithm. One of the characteristics that make the algorithm very flexible is that the compression rate can be adjusted. If we compress a lot, more information will be lost, but the result image size will be smaller. With a smaller compression rate we obtain a better quality, but the size of the resulting image will be bigger.

2.5.5 JPEG 2000 (Joint Photographic Experts Group 2000)

It is a wavelet-based image compression standard. It was created by the Joint Photographic Experts Group committee with the intention of superseding their original discrete cosine transform based JPEG standard. JPEG 2000 has higher compression ratios than JPEG. It does not suffer from the uniform blocks, so characteristics of JPEG images with very high compression rates. But it usually makes the image more blurred that JPEG.

2.5.6 Summary of the formats

The following table summarizes all the discussed image formats

Table 2.2 – Image formats characteristics

Format	Name	Characteristics
BMP	Windows bitmap	Uncompressed format
TIFF	Tagged Image File Format	Lossless: Document scanning and imaging format. Flexible: LZW, CCITT, RLE
PNG	Portable Network Graphics	Lossless: improve and replace GIF. Based on the DEFLATE algorithm.
JPEG	Joint Photographic Experts Group	Lossy: big compression ratio, good for photographic images
JPEG 2000	Joint Photographic Experts Group 2000	Lossy: eventual replacement for JPEG

2.6 Measures Of Compression

The compression methods are evaluated by two main criteria: compression efficiency and distortion.

2.6.1 Compression Ratio (CR)

Compression Ratio is the most common metric of performance measure of an image compression scheme is the compression ratio, which is defined by

$$compression\ ratio = \frac{original\ image\ size\ in\ bits}{compressed\ image\ size\ in\ bits}, \tag{2.3}$$

i.e., the ratio of the number of bits to represent the original image data to the number of bits to represent the compressed image data.

The original image size used in this thesis does not include any image format overhead or byte alignment overhead. For an image of size $w \times h$ with a bit depth b, the original image size is simply calculated by the following formula

$$original\ image\ size\ in\ bits\ = w \times h \times b, \tag{2.4}$$

On the contrary, the compressed image size counts all header or tail overhead needed to reconstruct the original image. For example, if an image of size 512×512 with 8 bits per pixel is compressed to 8192 bytes, the compression ratio will be $32 : 1$ or 32 [21].

2.6.2 Peak Signal-to-Noise Ratio (PSNR)

Often abbreviated (PSNR), which is an expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation. Because many signals have a very wide dynamic range, (ratio between the largest and smallest possible values of a changeable quantity) the PSNR is usually expressed in terms of the logarithmic decibel scale.

PSNR is an approximation to human perception of reconstruction quality. Although a higher PSNR generally indicates that the reconstruction is of higher quality, in some cases it may not. PSNR is most easily defined via the mean squared error (MSE) [26].

The mathematical representation of the PSNR is as follows:

$$\begin{aligned} PSNR &= 20 \log_{10} \left(\frac{MAX_f}{\sqrt{MSE}} \right) \\ &= 10 \log_{10} \left(\frac{MAX_f^2}{\sqrt{MSE}} \right) \end{aligned} \tag{2.5}$$

where the MSE (Mean Squared Error) is:

$$MSE = \frac{1}{mn} \sum_{0}^{m-1} \sum_{0}^{n-1} \|f(i,j) - g(i,j)\|^2 \tag{2.6}$$

With

- f is the matrix data of our original image,
- g is the matrix data of our degraded image in question,
- m is the numbers of rows of pixels of the images and i represents the index of that row,
- n is the number of columns of pixels of the image and j represents the index of that column,
- MAX_f is the maximum signal value that exists in our original "known to be good" image.

Chapter 3

Technique Implementation

3.1 Introduction

In this chapter, we are explaining our proposed compression technique starting from using only polynomial interpolation passing by enhance it using the introduced improvements such as, the use of row image sorting and the quantized error vectors. Giving for each technique an example using a simple grayscale image considered as one layer image.

3.2 The Original Idea

The main idea of our experiment is the use of interpolation to predict the image pixels values. On a digital image each layer was treated alone. In this technique, we considered each segment of each layer of the input image we extract an interpolated polynomial, which after its evaluation would give us the predictable pixel values of the that image.

3.2.1 The Compression Phase

At the first time, the used segment was a whole row of the input image, the contained pixels in this segment were considered as the y values, while the series $1, 2..., w$, with w is the image width, as x values of this interpolation. The advantage of using the interpolation these values is the use of these polynomials instead of the whole values, this process change the number of values to be saved from w pixel values to $n + 1$ interpolation coefficients, with a n is the polynomial degree saving them in a descendent order (the first value represents the coefficient of the polynomial degree).

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + a_1 x + a_0 \tag{3.1}$$

After that, we have changed the polynomial degree and tested the interpolation accuracy. Each polynomial in this technique was considered as a row in a new matrix (since each row has its own

polynomial) containing all the interpolated polynomials. Then, the was a phase of saving this new matrix in file. To make the saving process dynamical there was a necessity of using an auxiliary file containing a matrix with these information of the input image:

- The image height
- The image width
- The image bit depth
- polynomial degree

Next, we saved the polynomials matrix as a binary file, this allow us to have file with less volume comparing with text file, this is due to the each one of the technique, the first technique saves each element as the precision size (uint, int, double, ...etc). However, the second one saves each digit as an ASCII character, that means that the number 16, for example, is saved using the first technique with just 1 byte because it is of type **uint8** (unsigned integer with 8 bits), nevertheless, with the second it needs 2 bytes because it has two digits.

After that saving process, we performed a lossless compression on those two files (the one having interpolation polynomials and the auxiliary) using a rar compressor, so we had finally only file.

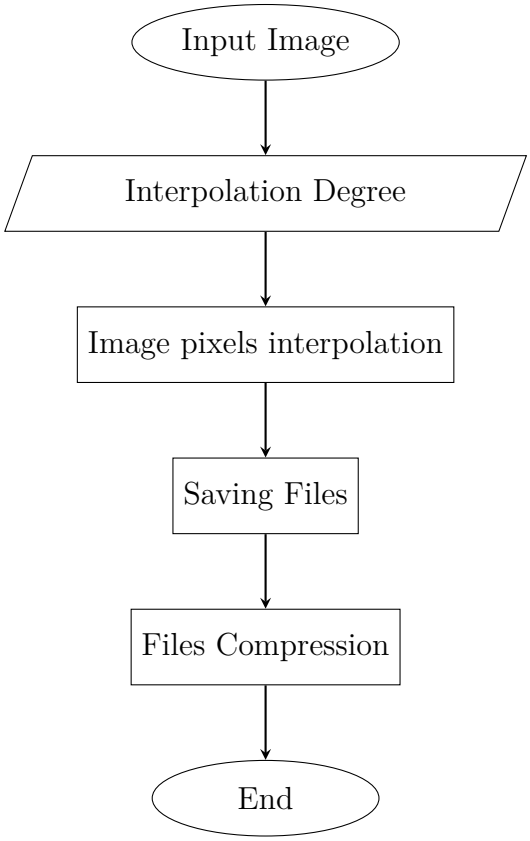
3.2.2 The Decompression Phase

That was the compression process, now describing the decompression process, first, we have done a decompression of the compressed file containing the polynomials and the auxiliary files, then, that auxiliary file have been read so the input information were retrieved ($h \times w \times b$, height, width and images layers respectively) using them reconstruct the decompressed image with the same dimensions.

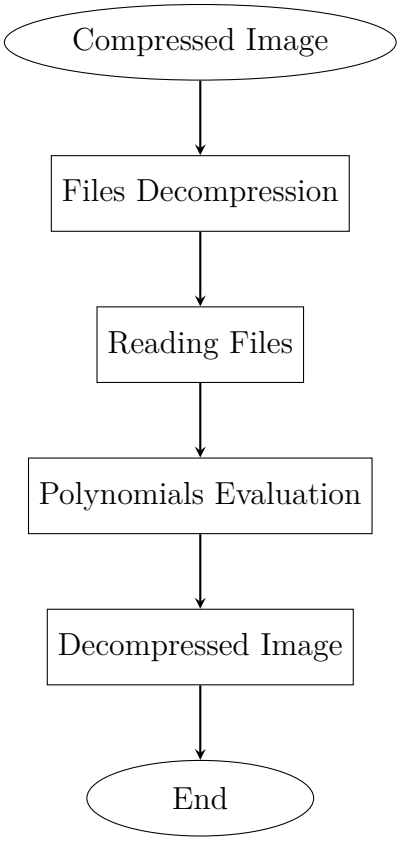
Finally, we have read the polynomials file and evaluated them so we could regenerate (in a somehow precision) the original image pixels.

The figure below contains both compression and decompression phases flow charts.

As an example we have used an 8-bit 10×8 grayscale image, this image represents the letter **e**, as follows, the image, its pixel values, its interpolation coefficients and its regenerated image using polynomial interpolation degree of 1.

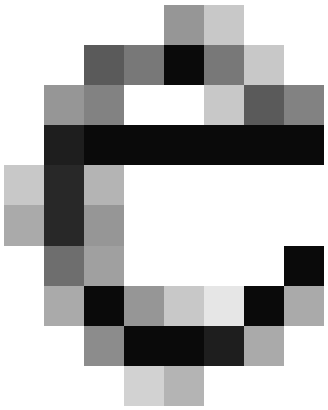


(a) Original idea compression phase



(b) Original idea decompression phase

Figure 3.1 – Original idea flow charts



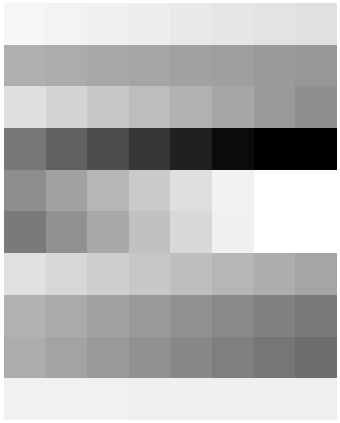
(a) The example image

255	255	255	255	150	200	255	255
255	255	90	120	10	120	200	255
255	150	130	255	255	200	90	130
255	30	10	10	10	10	10	10
200	40	180	255	255	255	255	255
170	40	150	255	255	255	255	255
255	110	160	255	255	255	255	10
255	170	10	150	200	230	10	170
255	255	140	10	10	30	170	255
255	255	255	210	180	255	255	255

(b) Pixel values

-3.21	249.46
-3.51	178.93
-11.49	234.82
-21.61	140.36
20.06	121.61
23.63	98.04
-8.39	232.14
-8.15	186.07
-8.99	181.07
-0.36	241.61

(c) Interpolation coefficients



(d) Result image

Figure 3.2 – The example image forms using polynomial interpolation only

While the auxiliary matrix contains these values,

- The image height: 10,
- The image width: 8,
- The image bit depth: 1,
- polynomial degree: 1.

3.3 The Proposed Technique Enhancements

Since images in general have variant (or sometimes very variant) pixel values, so the use of polynomial interpolation was not that efficient to restore in most of the times the input images, this obliged us to introduce some enhancements.

Performing these enhancements we used the same saving and lossless compression and decompression techniques as proposed in the original technique. These new enhancements are presented as follows.

3.3.1 The Use Of Row Image Segmentation

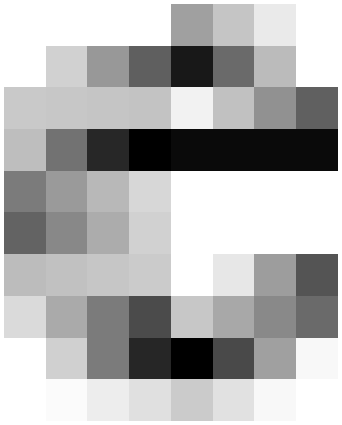
This was the first performed enhancement of the proposed techniques, this technique resulted a better quality decompressed image comparing to the one restored using the main technique, since we have segmented the image horizontally so the variance of the new segments values was less, and the more segments we had used, the more enhanced image quality we got, however, the more segments we had the more amount of data to be stored had increased.

The interpolation process this time was using each row of a segment as y values, and numbers $1, 2, ..., i$, with i the segment size, as x values. However, with the use of this technique we had to add a new value to the auxiliary file, which is the segment size.

Applying this technique on the example matrix segmenting it into 2 segments which means we have a couple polynomials each one has 2 coefficients. The next figures show the example image after applying this technique.

0	255	37	122.5
-57	322.5	81.5	-57.5
-2	202.5	-48.5	290
-75.5	265	0	10
30.5	92.5	0	255
36.5	62.5	0	255
5	182.5	-73.5	377.5
-47.5	265	-31	230
-85	377.5	87.5	-102.5
-13.5	277.5	22.5	180

(a) Interpolation coefficients



(b) Result image

Figure 3.3 – The example image forms using polynomial interpolation and row image segmentation

This time the auxiliary matrix has these values,

- The image height: 10,
- The image width: 8,
- The image bit depth: 1,
- polynomial degree: 1,
- number of segments: 2.

3.3.2 The Use Of Row Pixel Sorting

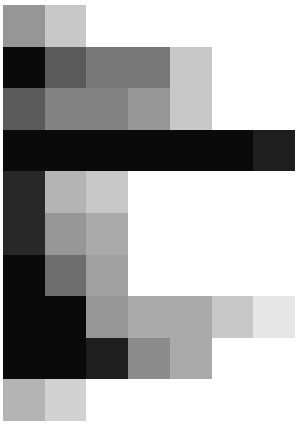
The next enhancement was the sorting of the image pixels horizontally in an ascending order, this allows us making them them as an increasing curve, so this could help using just a small

interpolation polynomial degree to have better precision (higher quality images) comparing to main technique.

But, using this technique another matrix containing the original pixels indexes had to be stored, so we had three files to be saved and compressed using the lossless technique, after their decompression we have evaluated the sorted polynomials so the regenerated values were the sorted ones. Then, we have used the stored indexes matrix in order to restore original pixels positions. This technique was performed as follows

5	6	1	2	3	4	7	8
5	3	4	6	7	1	2	8
7	3	8	2	6	1	4	5
3	4	5	6	7	8	2	1
2	3	1	4	5	6	7	8
2	3	1	4	5	6	7	8
8	2	3	1	4	5	6	7
3	7	4	2	8	5	6	1
4	5	6	3	7	1	2	8
5	4	1	2	3	6	7	8

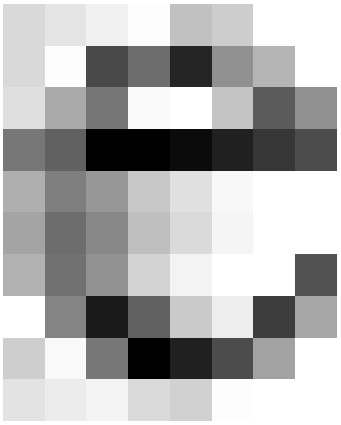
(a) Row sorted pixels positions



(b) Row Sorted image

12.02	180.89
36.01	1.07
26.25	65
21.61	-54.11
24.35	102.32
27.20	81.96
32.44	48.39
35.3	-9.46
43.39	-54.64
8.93	199.82

(c) Interpolation coefficients



(d) Result image

Figure 3.4 – The example image forms using polynomial interpolation and row image sorting

This technique has been performed also by hybridizing it with the row pixels segmentation technique (after sorting them in this case), in order to reduce the pixel values variations to help us increase the interpolation precision.

3.3.2.1 A Proposed Enhancement

In the term of decreasing the amount of data to be stored using this technique, we have proposed another technique to save the indexes matrix file supposing it would reduce its size.

This technique depends on benefiting of what the used saving precision has given us, for example the precision **uint16** can have as maximum value $2^{16} - 1 = 65535$.

Considering the used example image indexes parameters would give us 8 values for each row and a maximum value of 8, so, in order to reduce these 8 values we have decreased their values by one, which have given us values ranged between 0 and 7.

Then, we have converted them into the binary system, creating a matrix of 24 rows (since $log_2(8) \times 8 = 3 \times 8 = 24$) and h columns putting each bit in a cell of that matrix.

Finally we have reconverted to those values the decimal system considering this time the file precision as the code length of this conversion, saying it is a uint8 precision this has resulted us only $\frac{24}{8} = 3$ values per each row.

Applying this technique on the previous sorted pixels gives us as result this matrix

44	162	249
212	106	228
214	83	140
26	235	7
17	198	250
17	198	250
143	48	214
242	114	22
99	101	228
28	162	250

Figure 3.5 – Row sorted pixels positions using the binary technique

3.3.3 The Use Of Interpolation Error Vectors

Each interpolation has an amount of error since it is not that accurate, the more accurate precision the more reduced error, this error was calculated in our case using this formula:

$$Err = \sum_{k=1}^{l} \sum_{i=1}^{m} \sum_{j=1}^{n} (f(i,j,k) - g(i,j,k)) \tag{3.2}$$

With

- f is the input image,
- g is the restored image,
- l is the image bit depth,
- m is the image height,
- n is the image width.

8.75	11.96	15.18	18.39	-83.39	-30.18	28.04	31.25
79.58	83.1	-78.39	-44.88	-151.37	-37.86	45.65	104.17
31.67	-61.85	-70.36	66.13	77.63	34.11	-64.40	-12.92
136.25	-67.14	-65.54	-43.93	-22.32	-0.71	20.89	42.5
58.33	-121.74	-1.79	53.15	33.1	13.04	-7.02	-27.08
48.33	-105.3	-18.93	62.44	38.81	15.18	-8.45	-32.08
31.25	-105.36	-46.96	56.43	64.82	73.21	81.61	-155
77.08	0.24	-151.61	-3.45	54.70	92.86	-118.99	49.17
82.92	91.9	-14.11	-135.12	-126.13	-97.14	51.85	145.83
13.75	14.11	14.46	-30.18	-59.82	15.54	15.89	16.25

Figure 3.6 – Error values using an interpolation degree of 1

3.3.3.1 The Compression Phase

The compression phase of this technique has been done the same way as in the original idea, which means the performing of a polynomial interpolation only.

Then, we have evaluate the obtained interpolation coefficients in order to calculate the interpolation error using the subtraction operation between the evaluated values and the original values, which results a matrix. Next, we save the obtained error matrix in a float precision file.

Finally, we have performed a compression on the obtained files.

3.3.3.2 The Decompression Phase

This phase has been done by decompressing the compressed file, then, the obtained files have been read.

After that, we have evaluated the interpolation coefficients so we have been able to have the same values as obtained using the interpolation only technique.

Finally, an addition operation between the evaluated coefficients and the error matrix, element by element, in order to reconstruct a very identical image to the original one.

3.3.4 The Use of Quantized Interpolation Error Vectors

The previous original technique had a major problem we wanted to avoid which is saving additional data, this data had a huge amount, so to handle this problem the next method was introduced, which is saving that new data (the error values) using Uniform Scalar Quantization, so we could have a significant redundancy.

This redundancy has existed due to the presence of a lot of quantized values. These quantized values have been few in order to reduce the size of the interpolation error file, making them in a number of clusters saved in a codewords file, while the quantized values have been replaced by their clusters.

-136
-98
-60
-22
16
54
92
130

(a) Codewords vector

4	4	4	4	1	3	4	4
6	6	2	2	0	3	5	6
4	2	2	5	6	4	2	3
7	2	2	2	3	4	4	5
5	0	4	5	4	4	3	3
5	1	3	5	5	4	3	3
4	1	2	5	5	6	6	0
6	4	0	4	5	6	0	5
6	6	3	0	0	1	5	7
4	4	4	3	2	4	4	4

(b) 8 levels quantized error matrix

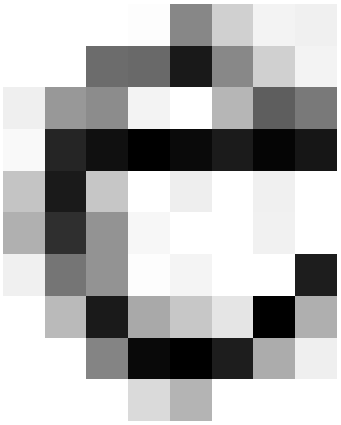
Figure 3.7 – 8 levels quantized error results

We have added to the codewords vector its minimum value if this one was negative so we can save only positive numbers, making the file containing these values of an unsigned type.

Another changing was done on the auxiliary file which was adding the minimum of codewords matrix. So, the saving process has been done on 4 files, which are the polynomials interpolation file, auxiliary file, codewords file and the new quantized values file. As a result, we have the new codewords vector and the result image

0
38
76
114
152
190
228
266

(a) New codewords vector



(b) Result image

Figure 3.8 – Result image using scalar quantization with 8 levels

Chapter 4

Experiments, Results And Discussion

4.1 Introduction

In this chapter we are introducing our application made in order to test the proposed techniques, then, we mention some code source parts of our application.

Next, we propose the images that have used to perform the experiments as long as with the tests parameters, the obtained results in both forms, numbers and graphs, and the decompressed images of each technique.

Finally, we discuss the obtained results of the proposed techniques.

4.2 The Application

This application has been developed using Matlab, which is one of the very important tools used for data analysis and Programming, because it is a high performance programming language for technical computing, developed by MathWorks in 1984, it integrates computation, visualization, and programming environment.

Furthermore, MatLab is a modern programming language environment: it has sophisticated data structures, contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MatLab an excellent tool for teaching and research. The used version of Matlab used to develop the application of this thesis is **Matlab R2013a(8.1.0.604)**.

This application has been named "Image Compressor" is developed to introduce the tested and implemented techniques, its GUI looks like introduced in the next figure This application has its own GUI icon[1] used instead of the default MatLab icon, with a primary figure[2] (no images figure) appears when the application is called no action is performed.

[1] *http://www.pngfactory.net/*, visited on 05/01/2015

[2] *http://www.designofsignage.com/application/symbol/building/largesymbols/no-photo.html*, visited on 05/01/2015

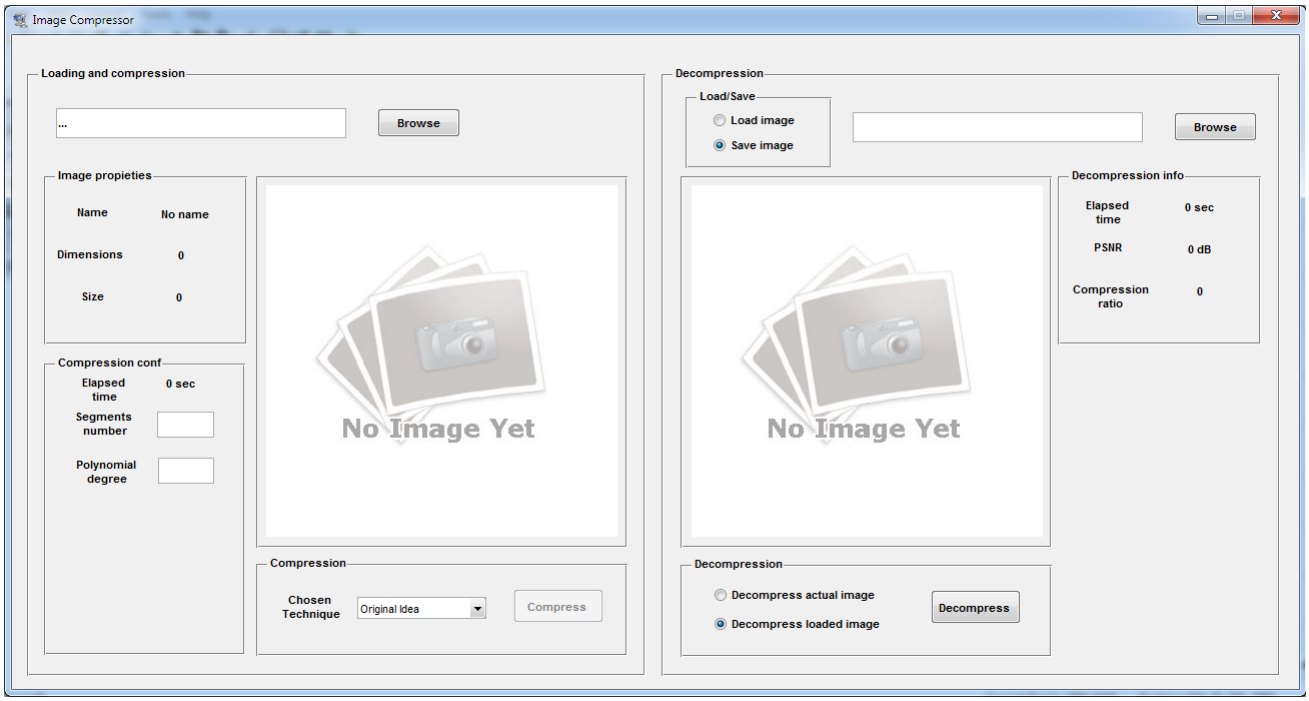


Figure 4.1 – The application interface

The left side of the GUI is used to perform the compression, after choosing an image via the address box above

While the most left panels give the image information and compression configurations respectively. Then we have the button of compression which can be available after choosing the original image to be compressed, then the compression time will appear.

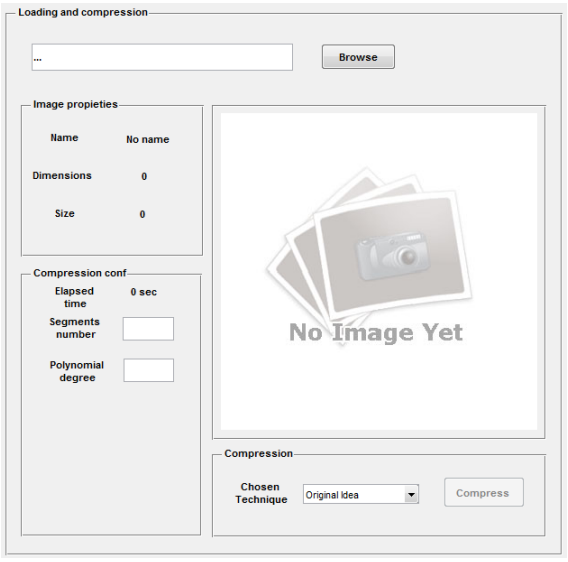


Figure 4.2 – Application left side

The right side panels are responsible of image decompression process, just after clicking the button "Decompress" the resulted image will appear in a short while as long as its resulted metrics (decompression time, PSNR value and the compression ratio).

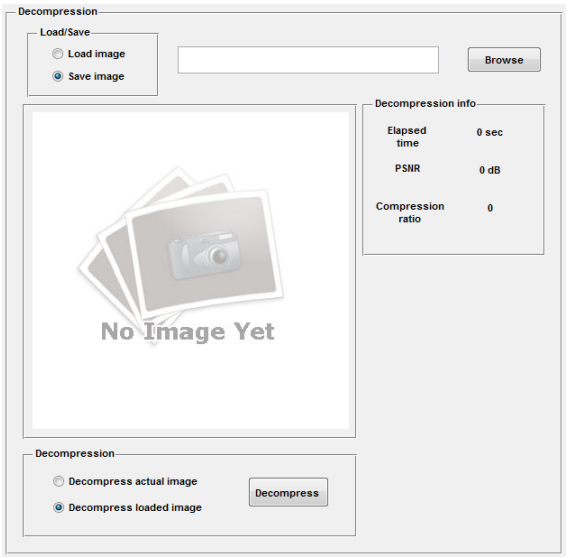


Figure 4.3 – Application right side

4.3 Set of Experiments Sample Images

To perform our experiments, we used 4 uncompressed reference images [3] (*.bmp), which are:

1. **Lena**, 512×512, $768\,kB$, color, (24bits/pixel),

[3] *http://www.eecs.qmul.ac.uk/~phao/CIP/Images/*, visited on 10/02/2014.

2. **Peppers**, 512×512, $768\,kB$, color, (24bits/pixel),
3. **Airplane**, 512×512, $768\,kB$, color, (24bits/pixel),
4. **Baboon** , 512×512, $768\,kB$, color, (24bits/pixel).

The test images are shown in the next figures:

(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.4 – Reference images used to perform tests

4.4 MatLab Source Codes

In order to perform the polynomial interpolation and its evaluation we are using as example the first row of reference image **Lena**, these two processes have been done using polynomial fit curve function (**polyfit**[4]), and polynomial evaluation function (**polyval**[5])

```
%% Example.m
% Using lena image reference
X = imread('Lenna.bmp','bmp');
```

[4] *http://www.mathworks.com/help/matlab/ref/polyval.html*, visited on 10/12/2014.
[5] *http://www.mathworks.com/help/matlab/ref/polyfit.html*, visited on 10/12/2014.

```
% choosing the first row of lena
Y = double(X(1,:,1));
% performing the interpolation with polynomial of 2
pol1 = polyfit(1:size(X,2),Y,1);
% evaluating the interpolation
val1 = polyval(pol1,1:size(X,1));
```

As a result of the previous code we have,

```
>> Example
Y =
  Columns 1 through 19
  226  226  223  223  226  226  228  227  227  225  228  225  223  226  223  221
   221  221  222 ...
 % only part of Y is shown due to its size(1 x 512)
pol1 =
  -0.35  205.8945
val1 =
 Columns 1 through 9
  205.8910  205.8875  205.8841  205.8806  205.8771  205.8737  205.8702  205.8667
   205.8632 ...
 % the same thing for this vector
```

Since the interpolation coefficients are real numbers, so, to save them we have used a precision of float (this precision needs less space to be saved comparing with double precision). However, this wasn't the same for the auxiliary file, this file was given a dynamic precision, starting from the precision uint8, it depended on the image information, if one of them was higher than some precision we used its successor, as an example if image height was 300 we can no longer use uint8 precision, we ought to use uint16 precision. At the end we named this file using its precision, whether 8, 16, ..., which results a file name of <file_name8(16)>.

Saving a binary file via MatLab is performed as shown in the code below

```
%% Example2.m
% Creating a matrix to be saved
X = randi(1:10);
% choosing the file name to be saved
fich = 'imd.osi';
% opening the file
FicH = fopen(fich,'w');
% writing the matrix in the file
% the chosen precision was used
% because max(X) < 256
```

```
fwrite(FicH,X,'uint8');
% closing the file
fclose(FicH);
```

While reading a binary file is done as follows

```
%% Example2.m
% opening the saved file
ImG = fopen('imd.osi');
% deciding the matrix containing the
% file information dimensions
count = [1 10];
% reading the file to the matrix
Img3 = fread(ImG , count,'uint8');
% close the file
fclose(ImG);
```

Matrix sorting is performed as in the next code

```
>> Mat = [4 5 0 2 78 1 0; 8 0 4 105 23 2 14];
>> [sor, idx] = sort(Mat,2) % 2 means row sorting
sor =
     0     0     1     2     4     5    78
     0     2     4     8    14    23   105
idx =
     3     7     6     4     1     2     5
     2     6     3     1     7     5     4
```

The performed lossless compression and decompression using WinRAR via MatLab have been done as the next code example shows

```
%% Example3.m
% choosing the file to be compressed
[status,result] = system(['"C:\Program Files\WinRAR\Rar.exe" a ' '"736_test.rar" '
    'imd.osi ' '']);
% there is no semicolumn
% Now the decompression
[status,result] = system(['"C:\Program Files\WinRAR\Rar.exe" e '  '  ' '"736_test.
    rar" ' '']);
%%
```

4.5 Performed Experiments

4.5.1 The Original Idea

Using this technique we have done three tests, changing in each one the polynomial degree as follows,

1. The first test have been done using an interpolation degree of 1, resulting us a coefficients matrix of size $h \times (i+1) \times b = 512 \times 2 \times 3$,
2. the second one have been done using an interpolation degree of 5, resulting us a coefficients matrix of size $h \times (i+1) \times b = 512 \times 6 \times 3$,
3. and in the third one we have used an interpolation degree of 10, resulting us a coefficients matrix of size $h \times (i+1) \times b = 512 \times 11 \times 3$.

Going back to test performed, the compression ratio have reached good level using the original idea, with a polynomial degree of 1, however, the shown images and the table below show us that the interpolation of variant or very variant y values can have a bad effect on the interpolation evaluation precision (distorted images details, unrecognizable images), and this was so obvious in the case of **Baboon** since it has the most variant values of the used images, as follows the 10 first pixel values of each image curves.

4.5.2 The Use Of Row Image Segmentation

Using this technique we have performed three tests, which are:

1. The first test have been performed using 4 ($\frac{Width}{Segments} = \frac{512}{4} = 128$ y values per each curve), and an interpolation degree of 1,
2. the second one have been performed using 4 image segments with an interpolation degree of 5,
3. while the third test have been performed using 4 image segments and an interpolation degree of 10,

4.5.3 The Use Of Row Image Sorting

Row image sorting technique has 2 sets of tests, in the first set everything was as described, while the second one have been done using the binary technique, both of them have the same test parameters.

1. The first one have been realized as follows: after sorting the all pixels then we have applied an interpolation degree of 1,

2. the second test have been performed like this: after sorting all the pixels then we have segmented the images rows using into 4 the previous technique with an interpolation degree of 1,

3. while the second test have been performed like this: after sorting all the pixels then we have segmented the images rows into 4 using the previous technique with an interpolation degree of 5.

4.5.4 The Use Of Interpolation Error Vectors

After performing all the previous tests, we have realized that the use of interpolation interpolation alone would not result good images, for better compression ratios we had to decrease the polynomial degree or not to use segments, and if we wanted better PSNR we have to increase the interpolation degrees we large numbers and to use a lot of segments.

All that led us to use another values had a close relation to polynomial interpolation, which is the interpolation error. Using this technique, we have performed only one test, using an interpolation degree of 1.

4.5.5 The Use Of Quantized Interpolation Error Vectors

With this technique 2 sets of tests have been done, the first was using this technique only, while the second one, was using the binary technique in order to reduce the amount of data to be stored, these two sets have had 3 tests with the same parameters, which are:

1. The first one has been realized using an interpolation degree of 1 and 8 levels of quantized error vectors,

2. the second test has been realized using an interpolation degree of 1 and 16 levels of quantized error vectors,

3. while the third one has had an interpolation degree of 1 and 32 levels of quantized error vectors.

4.6 Obtained Results

4.6.1 Original Idea

Table 4.1 – Results after performing the original idea with an interpolation degree of 1

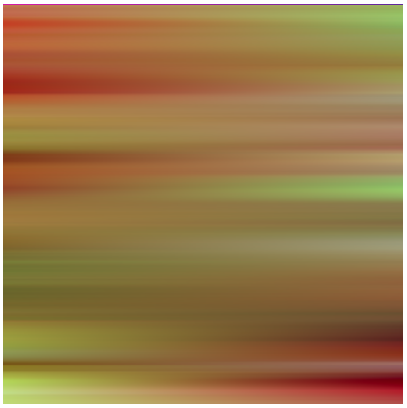
Image	**Interpolation degree of 1**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	87.85	15.46	3.85	1.61
Baboon	79.7	13.97	3.68	1.61
Peppers	83.23	13.16	3.78	2.13
Airplane	85.62	16.14	3.85	2.04

Table 4.2 – Results after performing the original idea with an interpolation degree of 5

Image	**Interpolation degree of 5**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	29.8	16.96	4.41	2.02
Baboon	26.11	15.37	4.56	1.81
Peppers	28.3	15.4	4.62	2.13
Airplane	28.06	18.04	4.5	2.04

(a) Lena

(b) Baboon



(c) Peppers

(d) Airplane

Figure 4.5 – Result images after compression using the original idea with an interpolation degree of 1

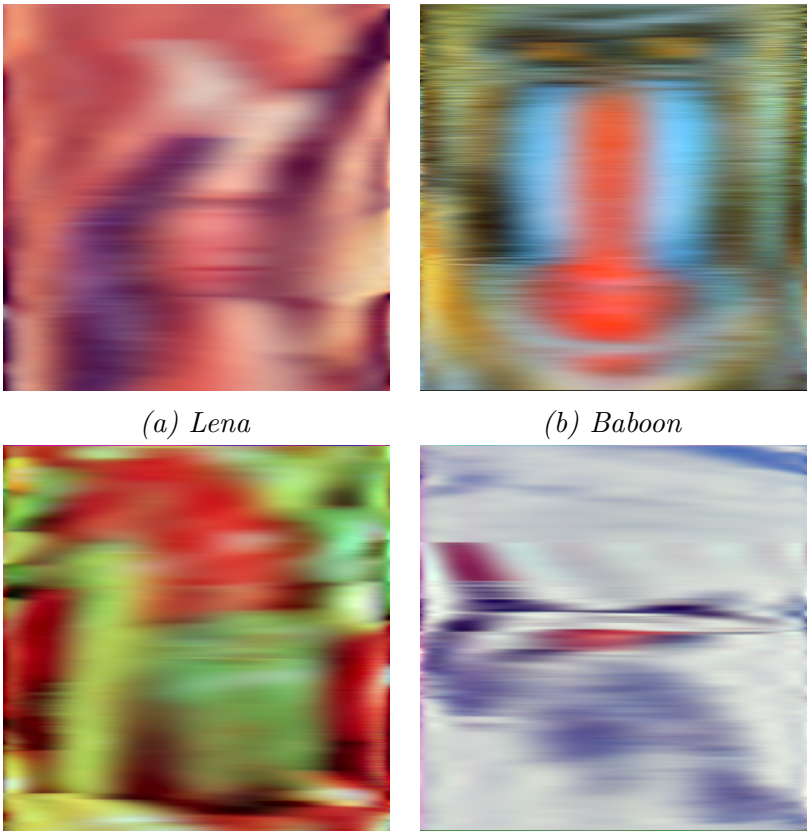
(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.6 – Result images after compression using the original idea with an interpolation degree of 5

Table 4.3 – Results after performing the original idea with an interpolation degree of 10

Image	**Interpolation degree of 10**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	15.17	18.68	4.77	1.75
Baboon	14.37	17.36	5.5	2.16
Peppers	15.3	17.31	4.87	2.08
Airplane	14.71	19.44	5.24	1.7



(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.7 – Result images after compression using the original idea with an interpolation degree of 10

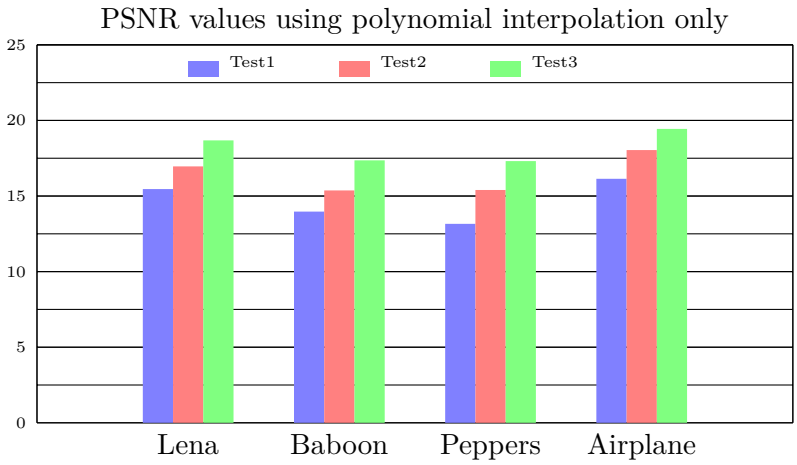


Figure 4.8 – PSNR values obtained using the original technique (polynomial interpolation only)

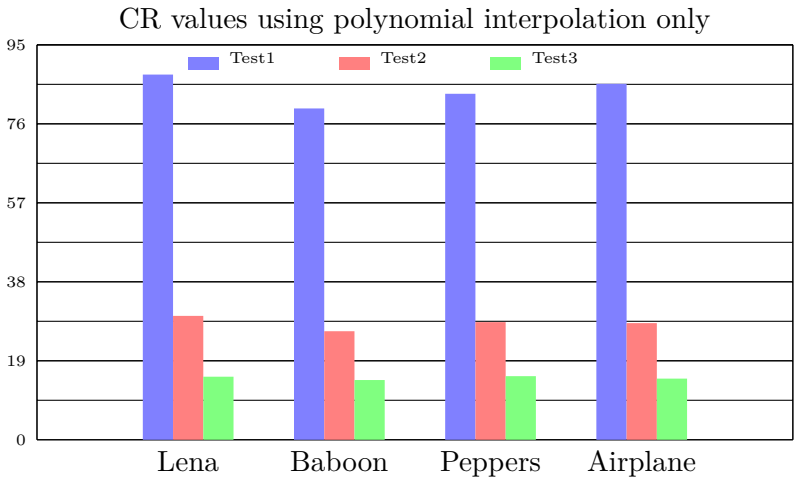
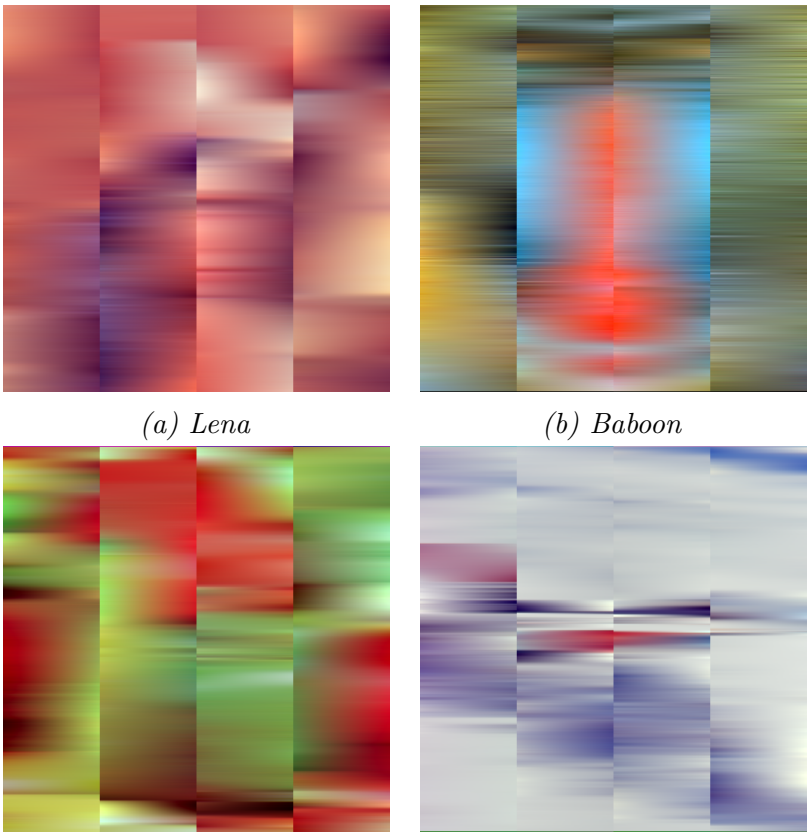


Figure 4.9 – CR values obtained using the original technique (polynomial interpolation only)

4.6.2 Row Image Segmentation

Table 4.4 – Results after performing the segmentation with a interpolation degree of 1 and 4 segments

Image	**Row image segmentation 4 segments and interpolation degree 1**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	22.91	17.62	6.36	2.33
Baboon	20.9	16.51	6.3	2.32
Peppers	22.07	15.91	6.7	2.12
Airplane	22.06	18.44	6.38	2.15



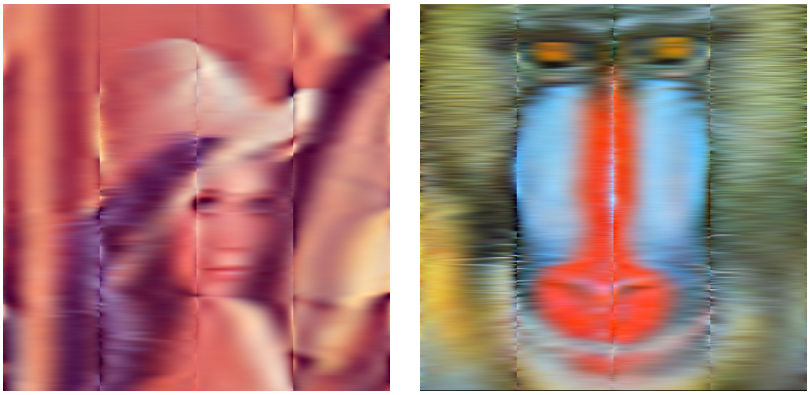
(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.10 – Result images after compression using with 4 segments and an interpolation degree of 1

Table 4.5 – Results after performing the segmentation with a interpolation degree of 5 and 4 segments

Image	**Row image segmentation with 4 segments and interpolation degree of 5**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	7	21.26	13.35	2.7
Baboon	5.5	19.02	13.56	2.61
Peppers	6.89	20.09	11.96	2.55
Airplane	6.1	21.26	14.46	2.6



(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.11 – Result images after compression using the row segmentation with 4 segments and an interpolation degree of 5

Table 4.6 – Results after performing the segmentation with an interpolation degree of 10 and 4 segments

Image	**Row image segmentation with 4 segments and interpolation degree of 10**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	3.48	24.07	17.85	3.25
Baboon	2.97	20.13	18.21	3.4
Peppers	3.6	22.96	17.96	3.48
Airplane	2.97	23.27	15.89	2.53

(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.12 – Result images after compression using with 4 segments and an interpolation degree of 10

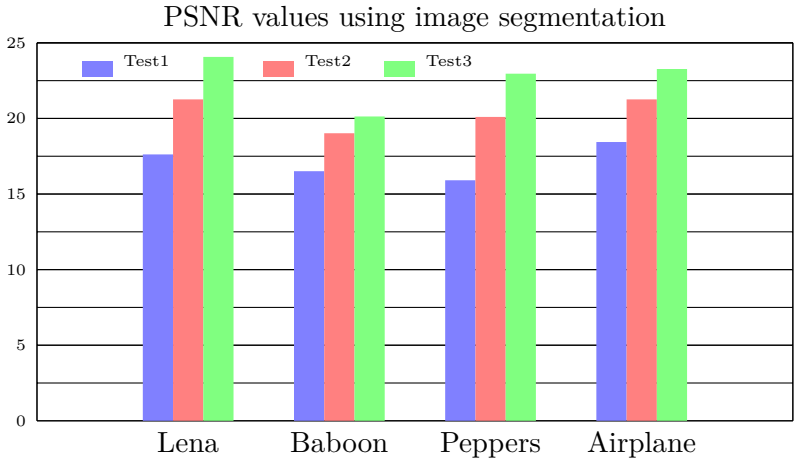


Figure 4.13 – PSNR values obtained using row image segmentation technique

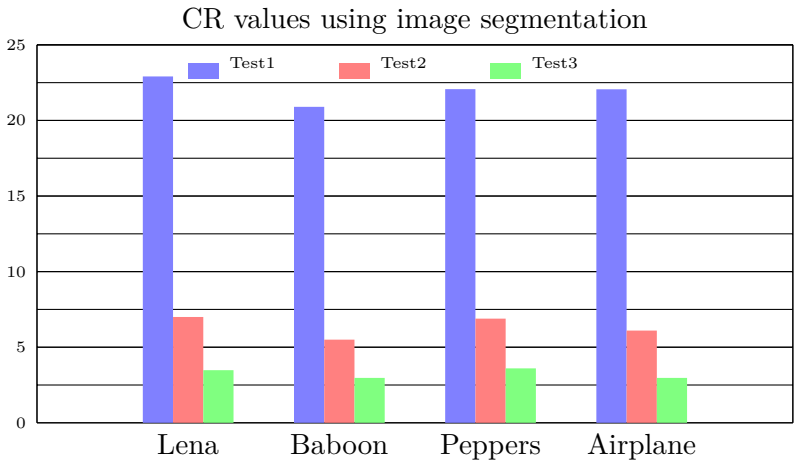


Figure 4.14 – CR values obtained using row image segmentation technique

4.6.3 Row Image Sorting

Table 4.7 – Results of the compression with sorting the values before interpolation (interpolation degree 1)

Image	**Interpolation with sorting using interpolation degree of 1**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	0.88	27.59	3.26	2.18
Baboon	0.76	27.94	3.54	2.43
Peppers	0.82	23.6	3.4	2.17
Airplane	0.81	22.07	3.94	2.38

(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.15 – Result images after compression using the image sorting

Table 4.8 – Results of the compression with sorting using 4 image segments

Image	**Interpolation with sorting using 4 segments and interpolation degree of 1**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	0.85	36.03	4.45	1.31
Baboon	0.75	37.66	4.43	1.53
Peppers	0.8	32.04	4.43	1.59
Airplane	0.79	31.94	5.33	2.49

(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.16 – Result images using the sorting with a interpolation degree of 4

Table 4.9 – Results of the compression with sorting using 4 segments and interpolation degree 5

Image	**Interpolation with sorting using 4 segments and interpolation degree of 5**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	0.76	44.7	9.92	2.43
Baboon	0.68	47.93	10.09	2.8
Peppers	0.72	43.15	10.31	2.62
Airplane	0.72	37.32	10.44	2.66

(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.17 – Result images using the sorting with a interpolation degree of 5 with 4 segments of each image

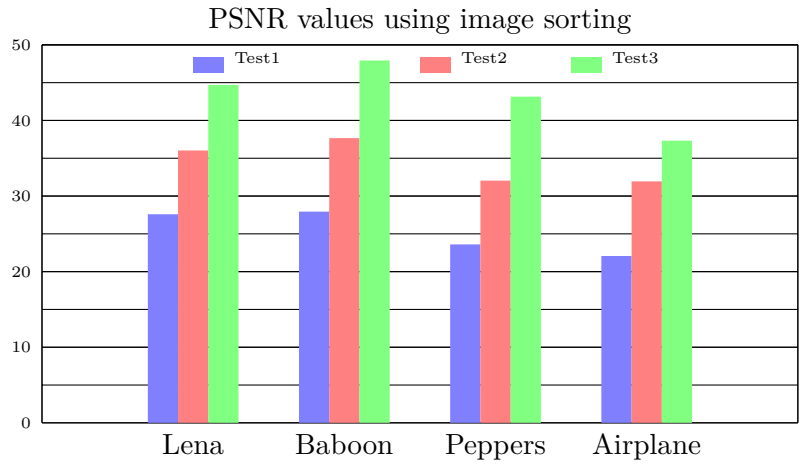


Figure 4.18 – PSNR values obtained using row image sorting technique

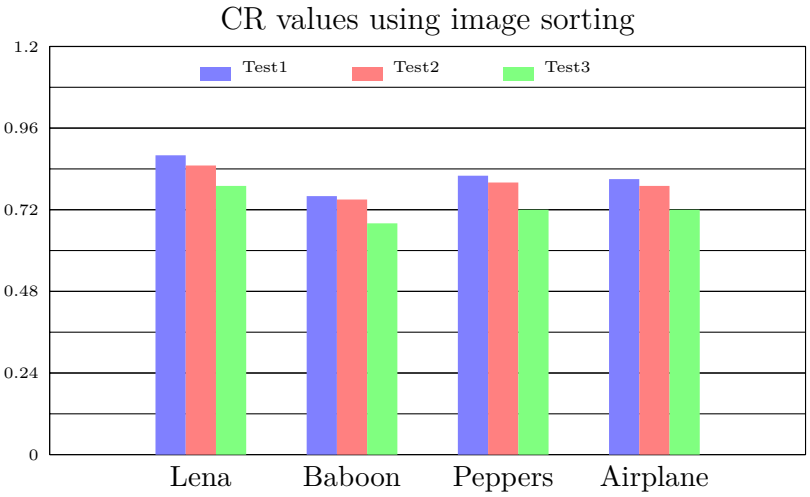


Figure 4.19 – CR values obtained using row image sorting technique

Table 4.10 – Results of the compression with the sorting and the new technique interpolation degree of 1 using the binary technique

Image	**Interpolation with sorting using interpolation degree of 1 and the binary technique**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	0.89	27.59	6.67	7.52
Baboon	0.88	27.94	6.52	7.54
Peppers	0.89	23.6	6.61	7.87
Airplane	0.89	22.07	7.89	7.65

Table 4.11 – Results of the compression with the sorting and the new technique interpolation degree of 4 using the binary technique

Image	**Interpolation with sorting using 4 segments, interpolation degree 1 and the binary technique**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	0.87	36.03	10.78	7.86
Baboon	0.86	37.66	10.37	8.43
Peppers	0.86	32.04	10	6.77
Airplane	0.86	31.94	9.61	7.47

Table 4.12 – Results of the compression with the sorting and the new technique interpolation degree of 5 and 4 segments using the binary technique

Image	**Interpolation with sorting using 4 segments, interpolation degree 5 and the binary technique**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	0.78	44.7	11.46	8.61
Baboon	0.77	47.93	11.3	7.78
Peppers	0.77	43.15	11.51	9.44
Airplane	0.77	37.32	11.52	8.06

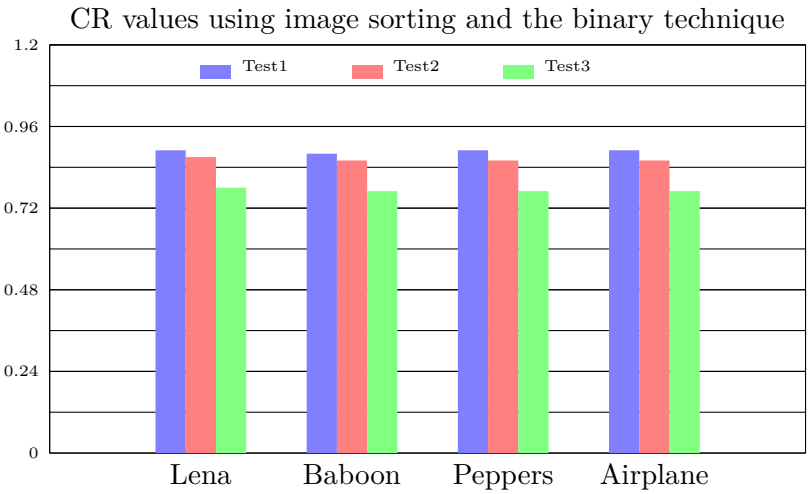


Figure 4.20 – CR values obtained using row image sorting technique with the binary technique

4.6.4 The Use Of Error Vectors

Table 4.13 – Results of compression using error vectors

Image	**Interpolation with error vectors**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	0.31	157.86	3.87	1.47
Baboon	0.28	157	3.67	2.02
Peppers	0.31	157	3.88	1.92
Airplane	0.31	154.68	3.82	1.91

(a) Lena

(b) Baboon

(c) Peppers

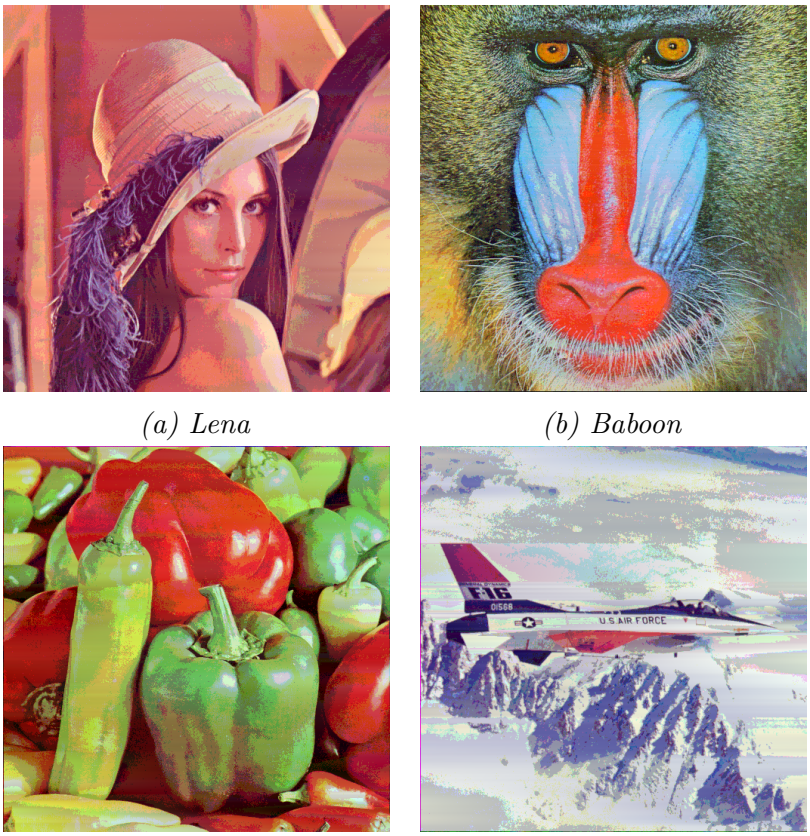
(d) Airplane

Figure 4.21 – Result images of compression with error vectors

4.6.5 The Use Of Quantized Error Vectors

Table 4.14 – Results of compression with quantized error vector of 8 levels

Image	**Interpolation with quantized error vectors of 8 levels**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	8.35	27.36	4.71	1.81
Baboon	4.22	26.53	4.78	1.98
Peppers	8.76	23.9	4.51	1.93
Airplane	8.59	23.77	4.95	1.8



(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.22 – Result images of compression with quantized error vectors of 8 levels

Table 4.15 – Results of compression with quantized error vector of 16 levels

Image	**Interpolation with quantized error vectors of 16 levels**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	5.19	33.17	5.61	2.32
Baboon	2.98	32.32	5.68	1.83
Peppers	5.15	29.93	5	2.38
Airplane	5.75	30.6	5.3	2.36

(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.23 – Result images of compression with quantized error vector of 16 levels

Table 4.16 – Results of compression with quantized error vector of 32 levels

Image	**Interpolation with quantized error vectors of 32 levels**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	3.77	38.8	7.31	2.23
Baboon	2.3	37.68	7.37	2.25
Peppers	3.57	35.64	6.79	2.01
Airplane	4.04	36.5	7.11	2.29

(a) Lena

(b) Baboon

(c) Peppers

(d) Airplane

Figure 4.24 – Result images of compression with quantized error vector of 32 levels

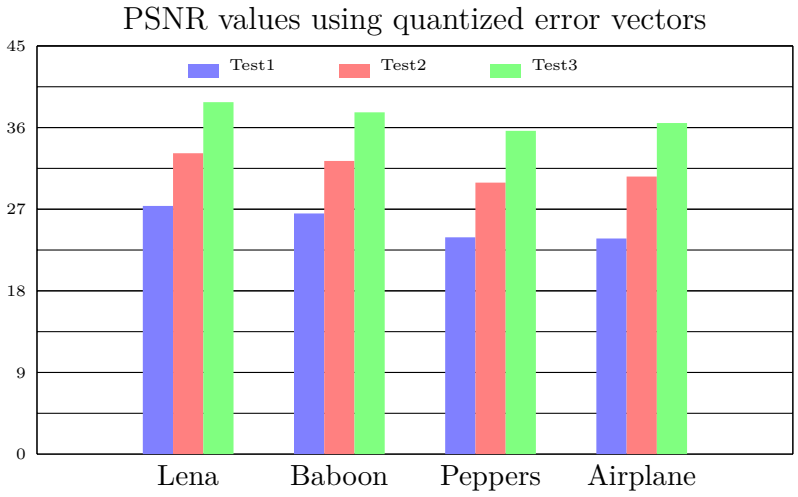


Figure 4.25 – PSNR values obtained using quantized error vectors technique

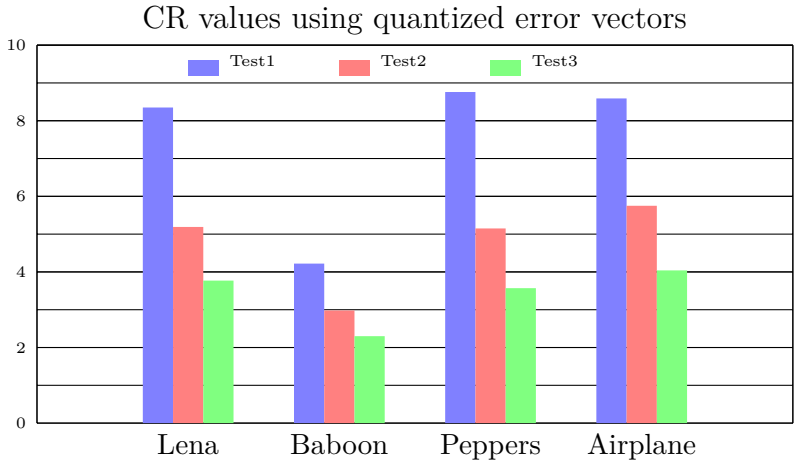


Figure 4.26 – CR values obtained using quantized error vectors technique

Table 4.17 – Results of compression with quantized error vector of 8 levels using the binary technique

Image	**Interpolation with quantized error vectors 8 levels with binary technique**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	8.14	27.36	5.97	3.81
Baboon	4.26	26.53	5.85	4
Peppers	8.92	23.9	5.12	4.09
Airplane	9.1	23.77	5.88	3.63

Table 4.18 – Results of compression with quantized error vector of 16 levels using the binary technique

Image	**Interpolation with quantized error vectors 16 levels with binary technique**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	5.53	33.17	6.7	4.65
Baboon	3.15	32.32	6.65	5.47
Peppers	6.13	29.93	6.15	4.03
Airplane	6.78	30.6	6.51	5.17

Table 4.19 – Results of compression with quantized error vector of 32 levels using the binary technique

Image	**Interpolation with quantized error vectors 32 levels with binary technique**			
	CR	PSNR(dB)	CTime(sec)	Dtime(sec)
Lena	2.9	38.8	7.3	6.7
Baboon	1.91	37.68	7.72	7.32
Peppers	3.01	35.64	7.94	6.5
Airplane	3.51	36.5	6.02	6

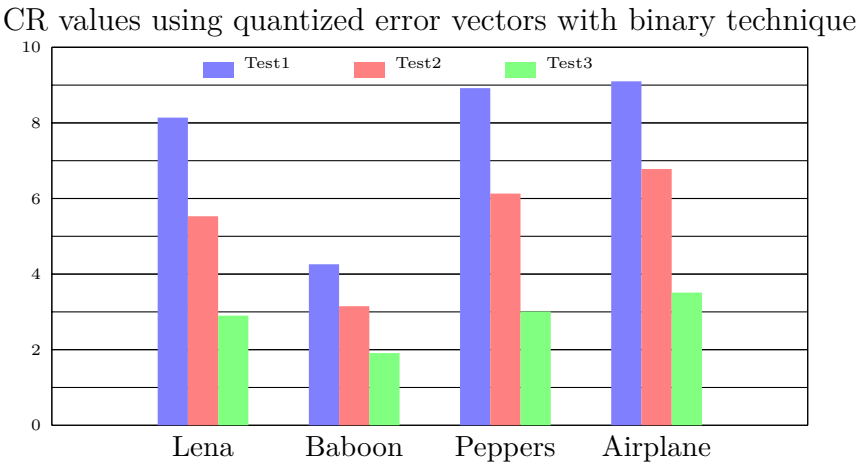
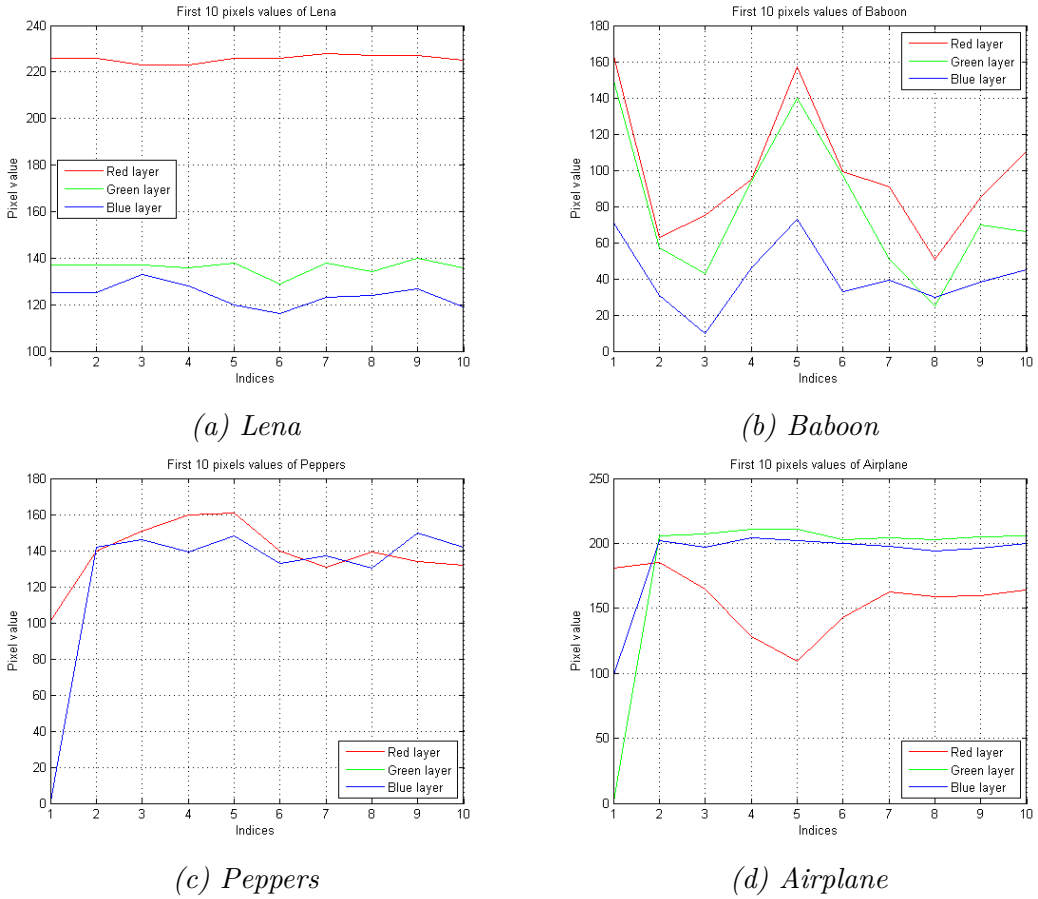


Figure 4.27 – CR values obtained using quantized error vectors technique with the binary technique

4.7 Discussion

The introduced technique using polynomial interpolation only has resulted very distorted and unrecognizable images ($13 < \text{PSNR} < 20$), with good compression ratios ($14 < \text{CR} < 88$), however, the use of this technique is expensive since we need to increase the polynomial degree making it reach higher values costing us huge amount of data, even more than that needed to store uncompressed data, since the daily used images had very variant pixel values as can be seen in the next figures which represent the first 10 pixel of each test image.



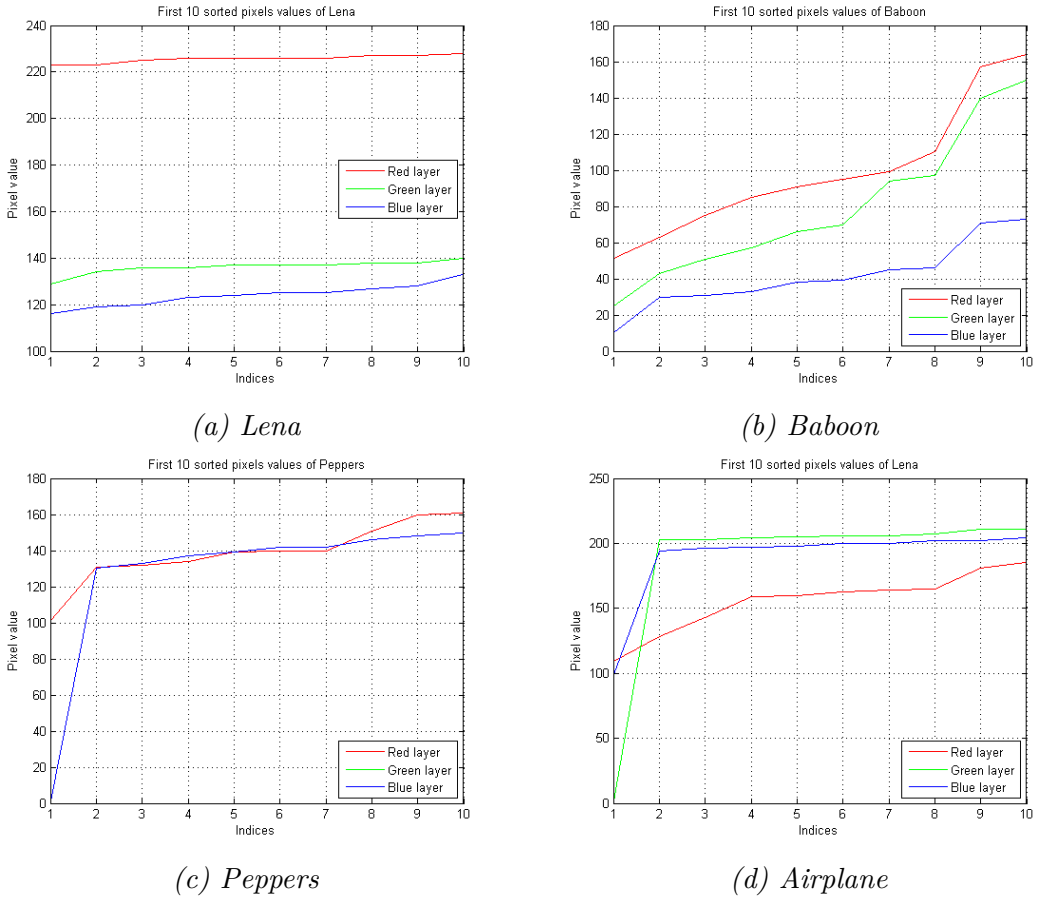
(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.28 – plotting the first 10 pixel of each input image

In order to reduce the variation, the image segmentation was used, even though this technique provided better result images ($15 < \text{PSNR} < 25$) with lower compression ratios ($2 < \text{CR} < 23$), it introduced a new term which is the broken mirror effect that appeared between edges, and if we want to avoid this effect many segments have to be used along with higher interpolation degrees. Also this technique failed to provide better results since it needed somehow high polynomial degree levels with many segments to be used.

Reaching this point, using another factor to reach better compression measures which is the row sorting technique provided better quality images comparing to the ones resulted using the previous techniques ($22 < \text{PSNR} < 48$), but also coasted us higher amount of data to be stored ($0.6 < \text{CR} < 0.9$). Although, the binary technique was supposed to provide us more compressed results, it seemed not that powerful technique with ($0.7 < \text{CR} < 0.9$). the next figures show the 10 first sorted pixel values of each test image. Even though, the proposed improvement appeared to result better compression ratios, we had only a little decreasing in its values.



(a) Lena *(b) Baboon*

(c) Peppers *(d) Airplane*

Figure 4.29 – plotting the first 10 pixel of each input image

Finally, using the polynomial interpolation error technique resulted best quality image (154 <PSNR < 158), with huge amount of data (0.2 < CR < 0.4), making the use of the uniform scalar quantization with different levels which are 8, 16 and 32 have a better effect on the compression ratios, making it reach tolerable values (23 < PSNR < 39) (2 < CR < 9). While the binary technique provided us a compression ratios of (1 < CR < 10).

Conclusion And Future work

As in the first part of this memoir we discussed some definitions and key terms used in the image compression field, like lossless techniques, PSNR, and CR ..., we also mentioned some really wide used image compression formats, whether lossy or lossless, along with compression techniques used in each one of them.

With the next chapter, we proposed our interpolation based image compression technique in detail, starting from the original idea, and mentioning its enhancements, so it could reach some good levels whether on the PSNR value or the compression ratio.

The practical part of this memoir allowed us to test a new image compression technique, using the interpolation.

As can be observed from the performed test, image compression using interpolation as a raw idea has given so bad visually images, very distort resulted ones, since all the result images had very low PSNRs comparing to the originals, but also using it good compression ratios could been reached.

By introducing some new enhancement this technique can reach good compression ratios with visually accepted output images using some techniques, all these output images have a predictable pixel values, so if we consider each row of the input image as a curve so this curve has to have less variations or periodically changeable variations, the first one proposed was dividing each curve on many numbers creating new segments, this helped us have new compression measure values.

The row image sorting technique had very huge compression ratios with better PSNRs than the previous techniques. While the use of interpolation error was a good technique with best quality images, but had very high compression ratio, so, in order to solve this problem we used the uniform scalar quantization to reduce those huge compression ratio values.

As future work, and since we didn't have that good results, we are willing

- making more hybridizing between the proposed techniques,
- performing more tests using other techniques related to polynomial interpolation,
- and to choose better quantization technique than the used one, such as Linde-Buzo-Gray algorithm instead of scalar quantization [31]

Bibliography

[1] MAHDI, Omar Adil, MOHAMMED, Mazin Abed, and MOHAMED, Ahmed Jasim. *Implementing a novel approach an convert audio compression to text coding via hybrid technique*. International Journal of Computer Science Issues, 2012, vol. 9, no 6, p. 53-59.

[2] CLARKE, Roger J. *Digital compression of still images and video*. Academic Press, Inc., 1995.

[3] SALOMON, David. *Data compression: the complete reference*. Springer Science & Business Media, 2004.

[4] V.K Padmaja and Dr. B. Chandrasekhar, *Literature Review of Image Compression Algorithm*, IJSER, Volume 3, pp. 1-6, 2012.

[5] BHASKARAN, Vasudev and KONSTANTINIDES, Konstantinos. *Image and video compression standards: algorithms and architectures*. Springer Science & Business Media, 1997.

[6] DING, Jian-Jiun. Jiun-De Huang,*Image Compression by Segmentation and Boundary Description*. 2007. Diss. Master's Thesis, National Taiwan University, Taipei.

[7] CHEN, Tzong-Jer and CHUANG, Keh-Shih. *A pseudo lossless image compression method. In : The 3rd International Congress on Image and Signal Processing*. 2010. p. 610-615.

[8] GONZALEZ, Rafael C., WOODS, Richard Eugene, and EDDINS, Steven L. *Digital image processing using MATLAB*. Pearson Education India, 2004.

[9] SALOMON, David. *Data compression: the complete reference*. Springer Science & Business Media, 2004.

[10] Ming Yang and Nikolaos Bourbakis,*An Overview of Lossless Digital Image Compression Techniques*,IEEE, pp. 1099-1102,2005.

[11] Sonal, Dinesh Kumar,*A Study of Various Image Compression Techniques*,pp. 1-5.

[12] MATHUR, Mridul Kumar, LOONKER, Seema, et SAXENA, Dheeraj. LOSSLESS HUFFMAN CODING TECHNIQUE FOR IMAGE COMPRESSION AND RECONSTRUCTION USING BINARY TREES. International Journal of Computer Technology & Applications, 2012, vol. 3, no 1.

[13] PUJAR, Jagadish H. and KADLASKAR, Lohit M. *A NEW LOSSLESS METHOD OF IMAGE COMPRESSION AND DECOMPRESSION USING HUFFMAN CODING TECHNIQUES*. Journal of Theoretical & Applied Information Technology, 2010, vol. 15.

[14] WITTEN, Ian H., NEAL, Radford M., and CLEARY, John G. *Arithmetic coding for data compression*. Communications of the ACM, 1987, vol. 30, no 6, p. 520-540.

[15] TAKAMURA, Seishi and TAKAGI, Mikio. *Lossless image compression with lossy image using adaptive prediction and arithmetic coding*. In : Data Compression Conference, 1994. DCC'94. Proceedings. IEEE, 1994. p. 166-174.

[16] ZIV, Jacob and LEMPEL, Abraham. *A universal algorithm for sequential data compression*. IEEE Transactions on information theory, 1977, vol. 23, no 3, p. 337-343.

[17] YE, Linning. *Codebook ordering for vector quantization*. 2003. Diss. Texas Tech University.

[18] SAYOOD, Khalid. *Introduction to data compression*. Newnes, 2012.

[19] Sonal, Dinesh Kumar,*A Study of Various Image Compression Techniques*,pp. 1-5.

[20] NECHAEVA, Nadezda. *Lossless image compression using n-ary context tree*. 2006.

[21] ZHA, Hui. *Progressive Lossless Image Compression Using Image Decomposition and Context Quantization*. 2008.

[22] FRÄNTI, Pasi. *Image Compression*. Lecture Notes, Department of Computer Science, University of Joensuu, 2000.

[23] NECHAEVA, Nadezda. *Lossless image compression using n-ary context tree*. 2006.

[24] VIJAYVARGIYA, Gaurav, SILAKARI, Sanjay, et PANDEY, Rajeev. *A Survey: Various Techniques of Image Compression*. arXiv preprint arXiv:1311.6877, 2013.

[25] G. K. Wallace, *The JPEG Still Picture Compression Standard*, IEEE Trans. On Consumer Electronics, vol.38, No.1, pp. xviii - xxxiv, Feb 1992.

[26] *http://www.ni.com/white-paper/13306/en/*, visited on 01/08/2015.

[27] SELVI, G. Uma Vetri and NADARAJAN, R. DICOM image compression using Bilinear interpolation. In : *Information Technology and Applications in Biomedicine (ITAB)*, 2010 10th IEEE International Conference on. IEEE, 2010. p. 1-4.

[28] *https://www.cs.cmu.edu/~chuck/lennapg/editor.html*, visited on 02/02/2015.

[29] *Image Processing Toolbox for use with Matlab*. The Mathworks Inc., Natick MA, second edition, 1997.

[30] William K. Pratt. *Digital Image Processing*. John Wiley and Sons, second edition, 1991.

[31] ORLITSKY, Alon. *Scalar vs. vector quantization: worst-case analysis*. 2002.