

*Republic of Iraq
Ministry of Higher Education and Scientific Research
Al-Nahrain University
College of Science*



Block Symmetry Predictor to Improve Fractal Image Compression

*A Thesis Submitted to the College of Science, AL-Nahrain
University In Partial Fulfillment of the Requirements for
The Degree of Master of Science in Computer Science*

Submitted by:

Ruaa Abdullah Jaber

(B.Sc. 2006)

Supervised by:

Dr. Loay E. George

November 2008

Thu Alqeda 1429

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

وَأَسْأَلُكَ يَا نَبِيَّ اللَّهِ
عَسَى أَنْ يَكُونَ مِنْ عَمَلِكُمْ
قَدِيمًا لَأَلْقَى اللَّهُ فِي
الْكِتَابِ الْكَلِمَاتِ لَعَلَّ
يُتَّقَى

وَأَسْأَلُكَ يَا نَبِيَّ اللَّهِ
عَسَى أَنْ يَكُونَ مِنْ عَمَلِكُمْ
قَدِيمًا لَأَلْقَى اللَّهُ فِي
الْكِتَابِ الْكَلِمَاتِ لَعَلَّ
يُتَّقَى

عَسَى أَنْ يَكُونَ مِنْ عَمَلِكُمْ
قَدِيمًا لَأَلْقَى اللَّهُ فِي
الْكِتَابِ الْكَلِمَاتِ لَعَلَّ
يُتَّقَى

سُوْرَةُ الرَّحْمٰنِ (الْبَقَرَةُ 85)

Supervisor Certification

I certify that this thesis was prepared under our supervision at the Department of Computer Science/College of Science/Al-Nahrain University, by **Ruaa Abdullah Jaber** as partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Supervisor

Signature:

Name : **Dr. Loay E. George**

Title : **Senior Research**

Date : / / **2008**

The Head of the Department Certification

In view of the available recommendations, I forward this thesis for debate by the examination committee.

Signature:

Name : **Dr. Taha S. Bashaga**

Title : **Head of the department of Computer Science,
Al-Nahrain University.**

Date : / / **2008**

Certification of the Examination Committee

We certify that we have read this thesis and as an examining committee, examined the student in its content and what is related to it, and that in our opinion it meets the standard of a thesis for the degree of Master of Science in Computer Science.

Signature:

Name: **Dr. Abdul Monem S. Rahma**

Title : **Assistance Professor**

Date : / / **2008**

(Chairman)

Signature:

Name: **Dr. Bushra Q. Al-Abudi**

Title : **Assistant Professor**

Date : / / 2008

(Member)

Signature:

Name: **Dr. Haithem A. Al-Ani**

Title : **Lecturer**

Date : / / 2008

(Member)

Signature:

Name: **Dr. Loay E. George**

Title : **Assistant Professor**

Date : / / **2008**

(Supervisor)

Approved by the Dean of the College of Science, Al-Nahrain University.

Signature:

Name: **Dr. LAITH ABDUL AZIZ AL-ANI**

Title: **Assist. Prof.**

Date: / / **2008**

(Dean of College of Science)

DEDICATED TO MY

PARENTS...

SISTERS...

AND BROTHERS...

To everyone

Taught me a letter

Ruaa



Acknowledgment

I would like to express my sincere appreciation to my supervisor, Dr. Loay E. George, for giving me the major steps to go on to explore the subject, sharing with me the ideas in my research "Block Symmetry Predictor to Improve Fractal Image Compression" and discuss the points that I felt they are important.

Grateful thanks for the Head of Department of Computer Science, Dr. Taha S. Bashaga.

Also, I wish to thank the staff of Computer Science Department at Al-Nahrain University for their help.

I would like to say "thank you" to my faithful friends for supporting and giving me advises.



Abstract

Various compression methods have been proposed to achieve high compression ratios and high image qualities in low computation time. One of these methods is *Fractal Image Compression*. The basic idea of fractal image compression is the partitioning of input image into non-overlapping range blocks. For every range block a similar but larger domain block is found. The set of coefficients of mapping the domain blocks to the range block, using affine transform, is recorded as compression data. The compressed image data set is called the Iterated Function System (IFS) mapping set. Decoding process applies the determined IFS transformations on any initial image, and the process is repeated many times till reaching the attractor.

In this research work, *four IFS coding schemes* have been established and tested. *The first scheme* is the traditional Fractal Image Compression (FIC) method, it is implemented on color images after transforming the (RGB) color components to (YCbCr) components. The compression results led to encoding time=144.02 sec, compression ratio=8.89 and PSNR=33.39.

The second scheme uses the FIC method with a predictor based on centralized moment features, this predictor is introduced to predict the type of symmetry operation required to set the domain block in a proper state to best matches the range block. The use of this predictor helps in reducing the number of trials of symmetry mappings from 8 trials to only one symmetry case. The use of predictor had reduced the encoding to approximately 14% in comparison with that of traditional method.

The third and fourth scheme implies the use of FIC method enhanced by the use of moment descriptor (order-1) and (order-3),

respectively. Either of these descriptors is used to classify the domain and range blocks into classes, each class is assigned by a class index whose value is equal to moments descriptor value. For encoding each range blocks only the domain blocks have similar descriptor values to that for range block will be IFS-matched with it. In these schemes the symmetry predictor, used in the second scheme, had been used to reduce the search about the best available similar domain block. The attained encoding time in both 3rd and 4th scheme is approximately 0.9% of that spend by a traditional scheme.

List of Abbreviations

| Abbreviation | Meaning |
|---------------------|---------------------------------------|
| BMP | Bitmap |
| BR | Bit Rate |
| Cr | Compression Ratio |
| dB | decibels |
| DPCM | Differential Pulse Coding Modulation |
| FIC | Fractal Image Compression |
| HV | Horizontal Vertical |
| HVS | Human Vision System |
| IFS | Iterated Function System |
| ISM | Improved Searching Mechanism |
| LCM | Loosely Coupled Multiprocessing |
| MAD | Mean Absolute Difference |
| MAE | Mean Absolute Error |
| MSE | Mean Square Error |
| NTSC | National Television Systems Committee |
| PAL | Phase Alternating Line |
| PIFS | Partitioned Iterated Function System |
| PSNR | Peak Signal to Noise Ratio |
| RGB | Red, Green, Blue |
| RMSE | Root Mean Square Error |
| SSD | Sum of the Squared Difference |
| SSE | Sum of the Squared Error |

Table of Contents

Chapter One: General Introduction

| | |
|--|----|
| 1.1 Preface ----- | 1 |
| 1.2 Image Compression ----- | 2 |
| 1.3 Fractal Image Compression ----- | 3 |
| 1.4 Partitioned Iterated Function System ----- | 3 |
| 1.5 Related Work ----- | 6 |
| 1.6 Aim of Thesis ----- | 10 |
| 1.7 Thesis Layout ----- | 11 |

Chapter Two: Fractal Image Compression

| | |
|---|----|
| 2.1 Introduction ----- | 12 |
| 2.2 Fractals ----- | 12 |
| 2.3 Self Similarity ----- | 13 |
| 2.4 Famous Fractal Shapes ----- | 14 |
| 2.5 Color Models ----- | 16 |
| 2.6 Image Fractal Coding ----- | 20 |
| 2.7 Iterated Function System for Zero-Mean Blocks ----- | 22 |
| 2.8 Moment Descriptor ----- | 24 |
| 2.9 Moments Ratio Factor ----- | 25 |
| 2.10 Down Sampling Methods ----- | 26 |
| 2.11 Affine Transformation ----- | 28 |
| 2.12 Partition Schemes ----- | 30 |
| 2.13 Quantization ----- | 34 |
| 2.13.1 Scalar Quantization ----- | 34 |
| 2.14 DPCM (Differential Pulse Coding Modulation) ----- | 35 |
| 2.15 The Test Measures ----- | 39 |
| A. Fidelity Criteria ----- | 39 |

| | |
|----------------------------------|----|
| B. Compression Compactness ----- | 42 |
| 2.16 Entropy ----- | 42 |
| 2.17 Huffman Coding ----- | 43 |
| 2.18 Arithmetic Coding ----- | 44 |
| 2.19 Shift Coding ----- | 45 |

Chapter Three: The Enhanced FIC-Scheme

| | |
|--|----|
| 3.1 Introduction ----- | 47 |
| 3.2 The System Model ----- | 48 |
| 3.3 Encoder Module ----- | 49 |
| 3.3.1 Load BMP Image ----- | 49 |
| 3.3.2 Conversion from RGB to YCbCr Color Space ----- | 51 |
| 3.3.3 Down Sampling ----- | 51 |
| 3.3.4 Resizing the Bands (Y, Cb, Cr) ----- | 53 |
| 3.3.5 FIC Encoder ----- | 56 |
| A. Range Pool Generation ----- | 56 |
| B. Domain Pool Generation ----- | 58 |
| C. Determination of Some Involved Coding Parameters ---- | 58 |
| D. Blocks Isometry State Assignment ----- | 59 |
| E. Blocks Classification Using Moments-Based Descriptor | 64 |
| F. Sorting of Domain Blocks ----- | 65 |
| G. Range Blocks Coding ----- | 67 |
| 3.3.6 Encoding the IFS Code ----- | 73 |
| 3.4 Decoder Module ----- | 76 |
| 3.4.1 Load and Decode the IFS Code ----- | 76 |
| 3.4.2 FIC Decoder ----- | 77 |
| A. Dequantization ----- | 79 |
| B. Reconstruction of Range Pool ----- | 79 |

| | |
|--|----|
| 3.4.3 Range Pool Resizing ----- | 80 |
| 3.4.4 Up Sampling ----- | 83 |
| 3.4.5 Conversion from YCbCr to RGB ----- | 84 |

Chapter Four: Performance Test Results

| | |
|---|-----|
| 4.1 Introduction ----- | 86 |
| 4.2 Image Test Material ----- | 87 |
| 4.3 Testing Strategy ----- | 88 |
| 4.4 Block Length Test ----- | 88 |
| 4.5 Jump Step Test ----- | 91 |
| 4.6 Maximum Scale Test ----- | 94 |
| 4.7 Scale Bits Test ----- | 97 |
| 4.8 Offset Bits Test ----- | 101 |
| 4.9 Minimum Error Test ----- | 104 |
| 4.10 Minimum Block Error Test ----- | 108 |
| 4.11 Number of Bins Test ----- | 112 |
| 4.12 Window Size Test ----- | 115 |
| 4.13 The Effect of Both No. of Bins and Window Size ----- | 118 |
| 4.14 Implementing Dis1FIC on Different Images ----- | 120 |
| 4.15 Discussion ----- | 122 |

Chapter Five: Conclusions and Future Work

| | |
|------------------------|-----|
| 5.1 Conclusions ----- | 124 |
| 5.2 Future Works ----- | 125 |

References



Chapter One

General Introduction

Chapter One

General Introduction

1.1 Preface

The term fractal was first used by Benoit Mandelbrot to designate objects that are self-similar at different scales. Such objects have details at every scale [NeGa95].

Mandelbrot's fractal geometry provides both a description and a mathematical model for many of the seemingly complex forms found in nature. Shapes such as coastlines, mountains and clouds are not easily described by traditional Euclidean geometry. Nevertheless, they often possess a remarkable simplifying invariance under changes of magnification. This statistical self-similarity is the essential quality of fractals in nature [BDM88].

Fractals are ubiquitous in complex natural phenomena. They are observed in the architecture of the mammalian lung, they determine the inter-beat interval in human heartbeats and the variation in human strides, they influence the information content of DNA sequences, and they describe the branching of trees and the root systems in plants, as well as the growth of bacterial colonies and many other biological systems. In physical phenomena they are also seen everywhere, in viscous fingering, dielectric breakdown, snowflake growth, and so on [WBG03].

Michael Barnsley and his coworkers at the Georgia institute of technology were the first to recognize the potential interest of fractal methods for image compression. Barnsley developed the theory of Iterated Function Systems (IFS), which was introduced by J. Hutchinson in 1981. After the publication of Barnsley's book "Fractals Everywhere" in 1988, and his paper in the January-1988 issue of BYTE magazine,

fractal image compression (FIC) became a very fashionable subject. The interest in this technique was aroused by the fantastic compression ratios claimed by Barnsley (i.e. up to 10,000 to 1). Together with Alan Sloan, Barnsley found Iterated Systems, Inc. and obtained US patent (4,941,193) on image compression using IFS.

A breakthrough was made in 1988 by Arnaud Jacquin, one of Barnsley's Ph.D. students. Instead of trying to find an IFS for a complete image, Jacquin brought the idea of partitioning the image into non-overlapping ranges, and finding a local IFS for each range. Jacquin developed the theory of Partitioned Iterated Function Systems (PIFS) and implemented a version of his algorithm. The main difficulty of FIC process is to find within the image reduced copies of the whole image. Real-world images often contain some self-similarity, but only between selected portions of the image. The breakthrough made by Jacquin was to partition the input image, and to find a local IFS for each partition. With this new method, it finally became possible to completely automate the compression process and furthermore to do it in a reasonable amount of time.

Yuval Fisher, Roger Boss, and Bill Jacobs were also among the first pioneers to make public contributions to the theory of PIFS [NeGa95].

1.2 Image Compression

The fast development of multimedia computing has led to the demand of using digital images. The manipulation, storage and transmission of these images in their raw form is very expensive, it significantly slows the transmission of the applications contain them and makes their storage costly. However, digital image processing is exploited in many diverse applications, but the size of these images places excessive demands for storage and transmission technology. Image

data compression is required to permit further use of digital image processing, it is the process of reducing the number of bits required to represent these images with lower bit rate, better quality and fast implementation [KD98].

1.3 Fractal Image Compression

In fractal image compression the image to be coded is partitioned into blocks called ranges. Each range is approximated by another part of the image called domain [RHS97].

Fractal image compression is a block based image compression, it detects and encodes the existing similarities between different regions in the image. It allows interesting compression ratios; however it suffers from long compression (encoding) time, whereas the decompression is fast. The time consuming part of the encoding step is due to the search for an appropriate domain block for each range block. Most of time required in the fractal compression is spent in the matching of a large number of blocks in the image. To speed up the fractal coding time, several methods have been devised to accelerate the search and reduce the encoding complexity, such as the Fisher classification method, or other methods which some of them are based on using artificial intelligence techniques (like, genetic algorithms and artificial neural networks) [Moha03].

1.4 Partitioned Iterated Function System

The theory of iterated function systems defines mathematically some concepts of chaos and irregularity. The research done mainly by Barnsley led to significant new methods for image understanding. Other researchers have followed those ideas and focused on the special characteristics of IFS fractals (such as, the measures over IFS attractors).

IFS description provides a potential new method for researching the image shape and texture. It forms, through a set of simple geometric transformations, a basic set of tools for interactive image construction.

Iterated function systems are based on the mathematical foundations laid by Hutchinson. IFS fractals have an elegant recursive definition: A fractal is constructed from a collage of transformed copies of itself; it is inherently self-similar and infinitely scalable.

The transformation is performed by a set of affine maps. An affine mapping of the plane is a combination of a reflection, rotation, scaling, sheer and translation (See Figure 1.1).

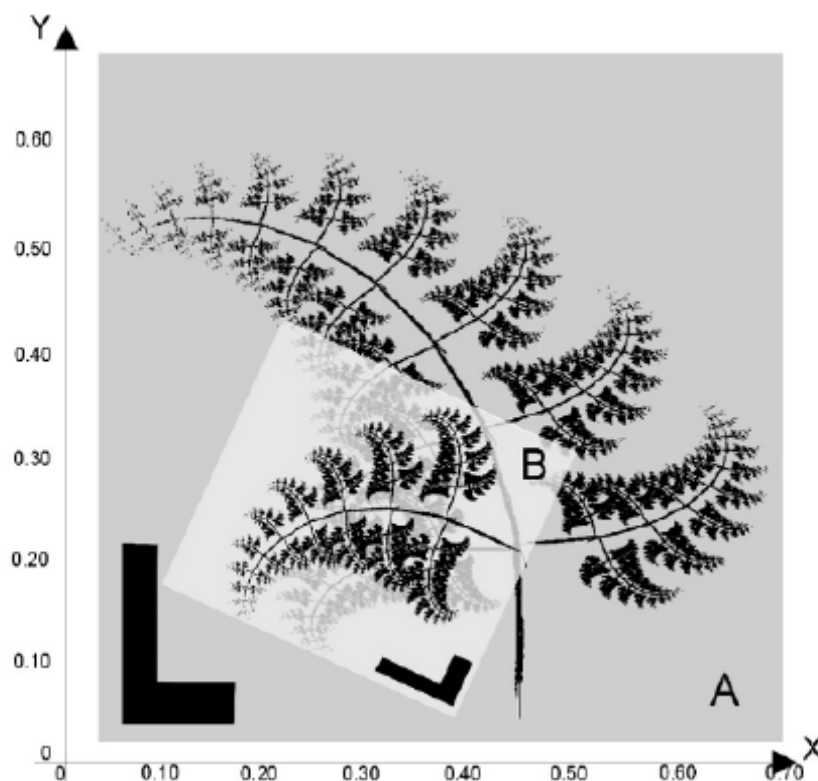


Figure (1.1) Geometric transformations implemented in the IFS model [Niki07]

Partitioned iterated function systems (PIFSs) are utilized in fractal image compression schemes. To solve the image encoding problem, it is

important to find a PIFS such that its attractor is as close to the encoded image as possible [Niki07].

The idea of fractal compression had reached the practical reality by Jacquin when he introduced the partitioned IFS (PIFS); which differs from an IFS in that each of the individual mappings operates on a subset of the image, rather than the entire image. Since the image support is tiled by "range blocks," each of which is mapped from one of the "domain blocks", as depicted in Figure (1.2), the combined mappings constitute a transform on the image as a whole. The transform that minimize the collage error within this framework is constructed by individually minimizing the collage error for each range block. This transform requires locating the domain block which may be made closest to each approximated range (using affine mapping). This transform is represented by specifying, for each range block, the identity of the matching domain block together with the block mapping parameters which minimizes the collage error for that block. Distances are usually measured by the mean-squared error (MSE) metric since optimization of the standard block mappings is simple under this measure [WoJa99].

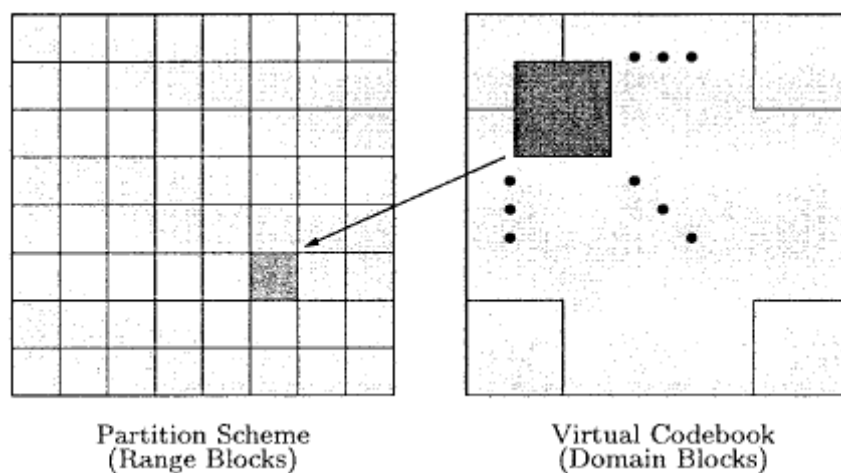


Figure (1.2) One of the block mappings in a PIFS representation [WoJa99]

1.5 Related Work

Many researchers have considered FIC as headlines in their work; some of their published works are the following:

1. Wohlberg and Jager (1994) [WoJa94], they indicated that lossy image coding by partitioned iterated function systems, popularly known as fractal image compression, had become an active area of research. In this scheme, an image is coded as a set of contractive transformations in a complete metric space. As a result of a well known theorem in metric space theory, the set of contractive transformations (subject to a few constraints) is guaranteed to produce an approximation to the original image, when iteratively applied to any initial image.
2. Rejeb and Anheier (1997) [ReAn97], proposed a time improved fractal image coder with a reduced domain pool and optimal luminance transform parameters calculation. This scheme is applicable to the Fischer's (1994) classification method. The encoding process is accelerated by reducing the domain pool, and then by minimizing the number of operations for the similarity search; this reduction is based on discarding the domains with nearly the same variance from each class of the domain pool. This approach provided a greater speed loss with a slight loss in the compression ratio and a slight improvement in image quality. The results of the conducted experiments showed that an acceleration of 6.7 for the image "Lena" is reached with a good decoded image quality. For a speed up factor of 2, the compression ratio is about 0.8% reduced and the image quality is about 0.23% improved. In order to increase the compression ratio again Jaquin's (1992) method was used, and some of the range blocks with shade property have been removed from the search.

3. Salih and Smith (1999) [SaSm99], presented a method of mapping similar regions within an image by an approximation of the collage error, this resulted in writing range blocks as a linear combination of domain blocks. Also, they addressed the complexity of the encoder, by proposing a new classification scheme based on the domain and range blocks moments which reduced the encoding time by a factor of hundreds with insubstantial loss in the image quality.
4. Chang and Kuo (2000) [ChKu00], they referred that the domain pool design is one of the dominant issues which affect the coding performance of fractal image compression. This paper employed the LBG algorithm and proposed a block averaging method to design an efficient domain pool based on a proposed iteration-free fractal image codec. The redundancies between the generated domain blocks have been reduced by the proposed methods. Therefore, the obtained domain pool is more efficient than that generated in the conventional fractal coding scheme, and thus the coding performance is improved. On the other hand, the iteration process in the conventional fractal coding scheme not only required a large size of memory and a high computation complexity but also prolongs the decoding process. The proposed iteration-free fractal codec can overcome the problems mentioned above. By the conducted computer simulation, it was noticed that both the LBG-based and block-averaging methods for the domain pool design in the proposed iteration free scheme have achieved excellent performances. For example, based on the proposed block-averaging method, the decoded Lena image has at least a 0.5 dB higher PSNR (under the same bit rate) and an eight-time faster decoding speed than the conventional fractal coding schemes that require iterations.

5. Al-A'mri (2001) [Alam01], presented a hierarchical quad-tree scheme for partitioning image, in FIC, in two different ways; fixed block size and variable block size. In these methods the image is partitioned into sub-squares called ranges. The domain blocks are obtained by shifting a block of twice the range size over the original image. Although, various kinds of criteria could be used for image partitioning; in this study a uniformity criterion had been utilized to perform image partitioning.
6. Tong and Wong (2002) [ToWo02], they referred that fractal image encoding is a computationally intensive method of compression due to its need to find the best match between image sub-blocks, this done by repeatedly searching a large virtual codebook constructed from the image under compression. One of the most innovative and promising approaches to speed up the encoding is to convert the range-domain block matching problem to a nearest neighbor search problem. This paper presented an improved formulation of approximate nearest neighbor search based on orthogonal projection and pre-quantization of the fractal transform parameters. Furthermore, an optimal adaptive scheme is derived for the approximate search parameter to further enhance the performance of the new algorithm. Experimental results showed that this new technique was able to improve both the fidelity and compression ratio, while significantly reduce memory requirement and encoding time.
7. Al-Dulaimy (2003) [Aldu03], the main purpose of his work is to reduce the encoding time of fractal image compression method. He proposed two approaches: the first is based on a new mathematical approach, called Improved Searching Mechanism (ISM), which determines IFS codes with less number of computation steps. While in the second approach, called Loosely Coupled Multiprocessing (LCM),

the encoding operations are executed using loosely coupled multiprocessing system.

8. Yung-Gi (2005) [Yung05], in his work he proposed an algorithm to improve the time-consuming encoding drawback by an adaptive searching window, partial distortion elimination (PDE), and characteristic exclusion algorithms. The proposed methods efficiently had decreased the encoding time. In addition, the compression ratio is also raised due to the reduced searching window. While conventional full search fractal encoding to compress a 512×512 image needs to search 247,009 domain blocks for every range block, this experimental results showed that the proposed method only needs to search 122 domain blocks, which is only 0.04939% compared to a conventional fractal encoder, at a bit rate of 0.2706 bits per pixel (bpp) while maintaining almost the same decoded quality in visual evaluation.
9. Distasi, Nappi, and Riccio (2006) [DNR06], the proposed a method to reduce the complexity of the image coding phase by classifying the blocks according to an approximation error measure. It was formally shown that postponing range/domain comparisons with respect to a preset block, is possible and can reduce drastically the amount of operations needed to encode each range. The proposed method had been compared with three other fractal coding methods, showing under which circumstances it performs better in terms of both bit rate and/or computing time.
10. Al-Hilo (2007) [Alhi07], designed and implemented a color image scheme using PIFS method. Since the main weak point in FIC is its need for long encoding time, in this research project a new block indexing method was suggested in order to reduce the long encoding time. The idea of reducing the mapping search operation is based on

making IFS matching between the range and domain blocks that have similar block indexing values; this leads to significant reduction in the encoding time. The proposed block indexing process is based on using moments (m_{01} , m_{10}) to produce an invariant descriptor to classify domain and range blocks. The utilization of this feature had significantly reduced the number of matching trials to find the closest domain block for each range block. The invariance of the proposed descriptor against affine transforms was the main reason behind reducing the number of range-domain comparisons which in turn led to speeding-up the domain search task.

1.6 Aim of Thesis

The aim of this work is to design and implement a fractal image compression system based on IFS-transform for zero-mean range-domain blocks. Some improvements were performed on the IFS-matching stage, these improvements implies the use of two moment based indexes as criterion to reduce the number of range-domain matching trials. This first moment based index is IFS-invariant, it is used to classify the range and domain blocks, and then only the blocks have similar indexes are passed through the domain-range matching test. While, the second index is utilized to predict the type of isometric process needed to be applied on the domain block to ensure the best IFS-matching state with the tested range block. Some additional steps are proposed to improve the performance of the improved FIC scheme, due to these additional steps the range pool partition could be done for any chosen block size without necessity for choosing block size as a divisible factor of both image width and height.

1.7 Thesis Layout

In addition to chapter one, the remaining parts of this thesis consists of the following chapters:

Chapter Two: (Fractal Image Compression)

In this chapter some image compression methods beside to fractal image compression technique are presented. Also, the relevant concepts and theorems with partitioned iterated function system, block symmetry predictor and moments descriptors are explained.

Chapter Three: (The Enhanced FIC-Scheme)

In this chapter, the proposed system design and implementation steps are given. The encoding and decoding modules are described in details.

Chapter Four: (Performance Test Results)

This chapter is dedicated to present the results of the conducted tests on the established coding system using different bitmap test images.

Chapter Five: (Conclusions and suggestions)

Some conclusion remarks that derived from the analysis of test results are given in this chapter. Also, some suggestions for future work are listed.

Chapter Two

Fractal Image Compression

Chapter Two

Fractal Image Compression

2.1 Introduction

This chapter introduces the definition of fractal, and the classification of its models. Also some of the famous fractal shapes, and some relevant concepts used in fractal image compression are presented. The aspects and equations deal with fractal image compression are described.

The concept of moments descriptor (which is used in this research work), affine transformations, down sampling methods, fidelity criteria, image partitioning schemes, quantization, Differential Pulse Coding Modulation (DPCM), and some color models are also described in this chapter.

2.2 Fractals

A good definition of the term fractal is elusive. Any particular definition seems to either exclude sets that are thought of as fractals or to include sets that are not thought of as fractals.

The definition of a "fractal" should be regarded in the same way as the biologist regards the definition of "life." There is no hard and fast definition, but just a list of properties characteristic of a living thing. In the same way, it seems best to regard a fractal as a set that has properties such as those listed below, rather than to look for a precise definition which will almost certainly exclude some interesting cases [Fish95].

When referring to fractal objects, most researchers typically define them as things that [Niki07]:

- Have a 'fine' structure, continual zoom in any region of a fractal can lead to fascinating complex details.
- Show some form of self-similarity, mostly approximate or statistical. Fractals provide a repeated graphical content that is easy to recognize and is visually appealing.
- Are too irregular to be described by the classic Euclidean geometry, both globally and locally. The lack of tangents presents a serious drawback for differential geometry-based analysis and modeling.
- Usually have a non-integer 'dimension' (defined in some way). Such a dimension is greater than the fractal topological dimension. Unlike the widely known Euclidean dimension (0 for a point, 1 for lines and curves, 2 for filled circles and 3 for cubes and other volumetric objects), the fractal dimension is not necessarily an integer.
- Are usually defined in a very simple way. Most fractals have relatively simple models that exploit recursive or iterative rendering schemes.

2.3 Self Similarity

A typical image does not contain the type of self-similarity found in fractals. But, it contains a different sort of self-similarity. Figure (2.1) shows regions of Lenna that are self-similar at different scales. A portion of her shoulder overlaps a smaller region that is almost identical, and a portion of the reflection of the hat in the mirror is similar to a smaller part of her hat.

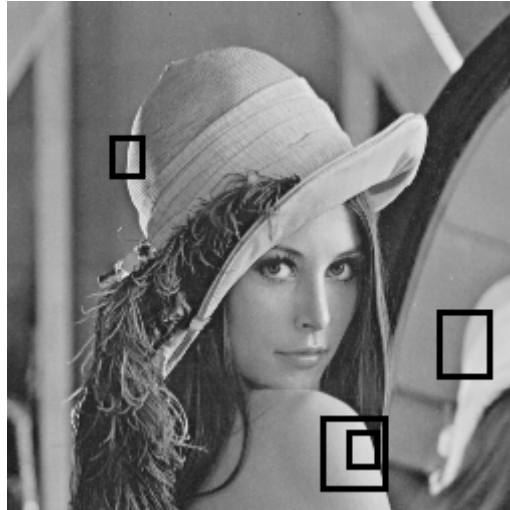


Figure (2.1) Lena image with self-similarity

The difference here is that the entire image is not self-similar, but parts of the image is self-similar with properly transformed parts of itself. Various studies indicated that most of the natural images contain this type of self-similarity. It is the restricted redundancy type that fractal image compression schemes attempt to eliminate [Sank98].

2.4 Famous Fractal Shapes

In general, fractals can be classified into two categories: Deterministic and Random fractals. The first category represents a type of fractals that are composed of several scaled down and rotated copies of themselves (such as Sierpinski triangle, Von Koch curve, Hilbert curve, Mandelbrot and Julia set). The second category represents natural phenomena that are everywhere in nature (such as clouds, mountains, coastlines, turbulence, roots, branches of tree, blood vessels, etc...) [TaLo98].

Two popular shapes of deterministic fractals are described in the following:

1. The Sierpinski Triangle

The Sierpinski triangle is named after the Polish mathematician Waclaw Sierpinski, who described some of its interesting properties in 1916 [Mand04]. It is one of the simplest fractal shapes. It can be generated by infinitely repeating a procedure of connecting the midpoints of the three sides of triangle to form four separate triangles, and cutting out the triangle in the center. Figure (2.2) illustrates the stages of Sierpinski triangle construction [Lani04].

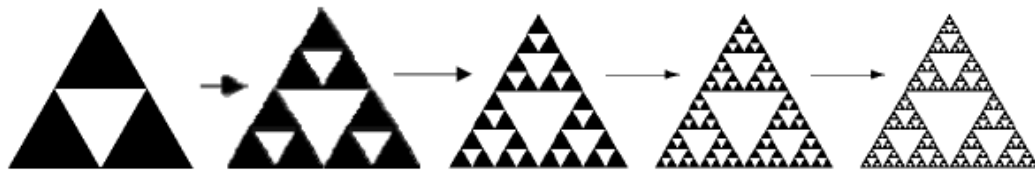


Figure (2.2) Sierpinski triangle construction stages [Lani04]

2. Von Koch Curve

The curve of Von Koch is generated by a simple geometric procedure, which can iterate an infinite number of times by dividing a straight line segment into three equal parts and substituting the intermediate part with two segments of the same length. Von Koch curve is a very elementary example of fractal; it follows a simple rule of construction. Figure (2.3) presents the stages of Von Koch construction [Bour91].



Figure (2.3) Von Koch curve construction stages [Bour91]

2.5 Color Models

The purpose of a color model (also called *color space* or *color system*) is to facilitate the specification of colors in some standard in accepted way. In essence, a color model is a specification of a coordinate system and a subspace within that system, where each color is represented by a single point [Gonz02].

In the following some of the popular color models used in various compression schemes are given:

1. RGB model

The red-green-blue (RGB) primary color system is the best known of several color systems. This is due to the main feature of the human perception of color. The color sensitive area in the Human Vision System (HVS) consists of three different sets of cones and each set is sensitive to the light of one of the three primary colors: red, green, and blue. Consequently, any color sensed by the HVS can be considered as a particular linear combination of the three primary colors [ShSu00].

Figure (2.4) shows the RGB color space, using a cube created by three axes representing pure red, green, and blue color. A main property of this color space is that the sum of all three basic colors, using maximum intensity, is white. Gray-scale values follow the line from black (the origin of the coordinate system) to white [Klin03].

The RGB model is used mainly in color image acquisition and display systems. In color signal processing, including image and video compression, the luminance-chrominance color system is more efficient and, hence, widely used. This has something to do with the color perception of the HVS. It is known that the HVS is more sensitive to green than to red, and is least sensitive to blue. An equal representation of red, green, and blue leads to inefficient data representation when the HVS

is the ultimate viewer. Allocating data only to the information that the HVS can perceive can make video coding more efficient.

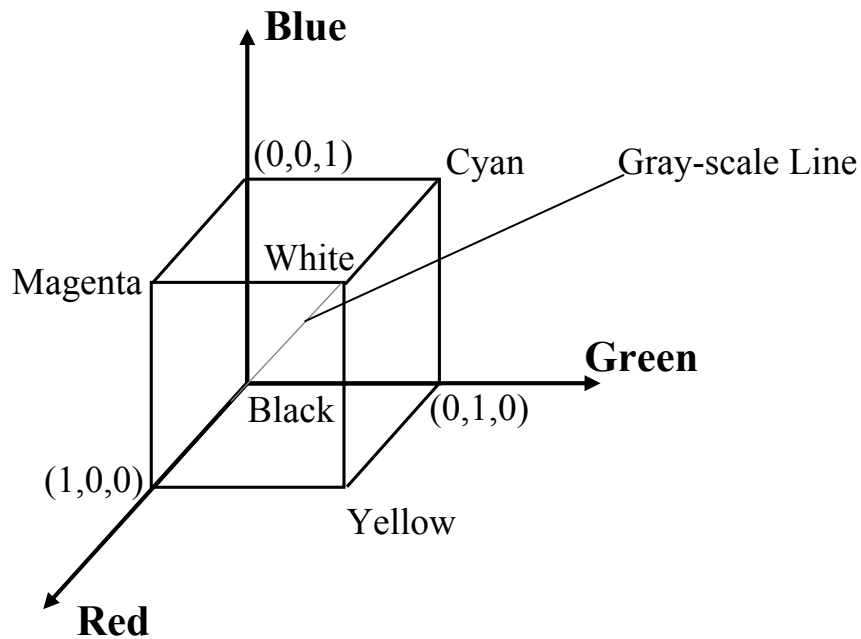


Figure (2.4) RGB Color Cube [Klin03]

Luminance is concerned with the perceived brightness, while chrominance is related to the perception of hue and saturation of color. Roughly speaking, the luminance-chrominance representation agrees more with the color perception of the HVS. This feature makes the luminance-chrominance color models more suitable for color image processing than RGB representation [ShSu00].

2. YUV model

The color space in Phase Alternating Line (PAL) TV-Standard System is represented by YUV, where Y represents the luminance and U and V represent the two color components [Ghan03]. The luminance Y can be determined from the RGB model via the following relation:

$$Y = 0.299R + 0.587G + 0.114B, \dots\dots\dots(2.1)$$

It is noted that the three weights associated with the three primary colors, R, G, and B, are not the same. Their different magnitudes reflect the different responses of the HVS to different primary colors.

Instead of being directly related to hue and saturation, the other two chrominance components, U and V, are defined as color differences, as follows:

$$U=0.492(B- Y) ,.....(2.2)$$

$$V=0.877(R- Y) ,.....(2.3)$$

In this way, the YUV model lowers computational complexity. It has been used in Phase Alternating Line (PAL) video standard. Note that PAL is an analog composite color TV standard and is used in most European countries, some Asian countries, and Australia. In composite systems, both the luminance and chrominance components of the TV signals are multiplexed within the same channel. For completeness, the transform equations from expression of RGB to YUV are listed below:

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} ,.....(2.4)$$

3. YIQ model

This color space has been utilized in National Television Systems Committee (NTSC) TV systems. NTSC is an analog composite color TV standard and is used in North America and Japan [ShSu00].

The luminance information is still in Y, which represents the gray scale information, while hue (I) and saturation (Q) carry the color information [TiAj05].

The two equations below shows that the two chrominance components (I, Q) are the linear transformation (i.e., rotation by 33°) of the U and V components defined in the YUV model. Specifically,

$$I = -\cos(33)U + \sin(33)V, \dots\dots\dots(2.5)$$

$$Q = \sin(33)U + \cos(33)V, \dots\dots\dots(2.6)$$

Substituting the U and V expressed in Equations (2.2) and (2.3) into the above two equations, the YIQ could directly expressed in terms of RGB. That is,

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}, \dots\dots\dots(2.7)$$

4. YDbDr

The YDbDr model is used in the Sequential Couleur a Memoire (SECAM) TV system. SECAM is used in France, Russia, and some eastern European countries. The relationship between YDbDr and RGB is shown by the following expression:

$$\begin{pmatrix} Y \\ Db \\ Dr \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.450 & -0.883 & 1.333 \\ -1.333 & 1.116 & -0.217 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}, \dots\dots\dots(2.8)$$

That is,

$$Db = 3.059U, \dots\dots\dots(2.9)$$

$$Dr = -2.169V, \dots\dots\dots(2.10)$$

5. YCbCr model

From the above mentioned models, it can be seen that the U and V chrominance components are the differences between the gamma-corrected color B and the luminance Y, and the gamma-corrected R and the luminance Y, respectively. The chrominance component pairs I and Q, and Db and Dr are both linear transforms of U and V. Hence they are very closely related to each other. It is noted that U and V may be negative as well. So, in order to make chrominance components nonnegative, the Y, U, and V are scaled and shifted to produce the YCbCr model, which is used in the international coding standards JPEG and MPEG [ShSu00], where Y is the luminous component while Cb and Cr provide the color information [TiAj05, ShSu00]:

$$\begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 0.257 & 0.504 & 0.098 \\ -0.148 & -0.291 & 0.439 \\ 0.439 & -0.368 & -0.071 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 16 \\ 128 \\ 128 \end{pmatrix}, \dots\dots\dots(2.11)$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0.001 & 1.582 \\ 1 & 1.863 & -0.0002 \\ 1 & -0.188 & -0.469 \end{pmatrix} \begin{pmatrix} Y \\ Cb \\ Cr \end{pmatrix}, \dots\dots\dots(2.12)$$

2.6 Image Fractal Coding

PIFS image encoder consists of a set of transforms on regions of the image. The set of regions (i.e., the domain blocks) from which the transform domains are chosen are overlapped, while the regions (i.e., the range blocks) forming the ranges of the transformation are tiled.

The set of transformations consist of a spatial contraction (e.g., averaging each 4 neighboring pixels) to construct a $k \times k$ blocks from a $2k \times 2k$ blocks, followed by one of the 8 square symmetry operations (4 rotations and 4 reflections), and followed by a contractive affine transformation on the grey scale values (for a block with pixel values).

For a range block with pixel values $(r_0, r_1, \dots, r_{m-1})$, and the domain block $(d_0, d_1, \dots, d_{m-1})$, the contractive affine approximation is,

$$r'_i = sd_i + o_i \text{ ,.....(2.12)}$$

Where s (scale) and o (offset) are the affine transform coefficients, r'_i s are the approximate (constructed) range values. The scale (s) and offset (o) parameters are determined by applying the method of least sum of square errors (χ^2) between r' and r values [ToPi01]:

$$\chi^2 = \sum_{i=0}^{m-1} (r'_i - r_i)^2 \text{ ,.....(2.13)}$$

$$\frac{\partial \chi^2}{\partial s} = 0; \quad \frac{\partial \chi^2}{\partial o} = 0$$

After straight forward manipulation to above equations, the following expressions for scale (s) and offset (o) coefficients are obtained:

$$s = \frac{m \sum_{i=0}^{m-1} d_i r_i - \sum_{i=0}^{m-1} d_i \sum_{i=0}^{m-1} r_i}{m \sum_{i=0}^{m-1} d_i^2 - \left(\sum_{i=0}^{m-1} d_i \right)^2} \text{ ,.....(2.14)}$$

$$o = \frac{\sum_{i=0}^{m-1} d_i^2 \sum_{i=0}^{m-1} r_i - \sum_{i=0}^{m-1} d_i \sum_{i=0}^{m-1} r_i d_i}{m \sum_{i=0}^{m-1} d_i^2 - \left(\sum_{i=0}^{m-1} d_i \right)^2} \text{ ,.....(2.15)}$$

In each range-domain matching instance before determining the value of χ^2 , the scale (s) and offset (o) values should firstly imposed to the clipping conditions $(o_{min} \leq o \leq o_{max})$ and $(|s| \leq s_{max})$, where (o_{min}, o_{max}) are the lower and upper boundaries of the permissible values of offset, s_{max} is the maximum permissible scale value. Secondly, they should be quantized by using the following equations:

$$i_s = \text{round}\left(\frac{s}{s_{\max}}(2^{a-1} - 2)\right), \dots \dots \dots (2.16)$$

$$i_o = \text{round}\left(\frac{2^b - 1}{o_{\max} - o_{\min}}(o - o_{\min})\right), \dots \dots \dots (2.17)$$

$$s'_q = \frac{s_{\max}}{2^{a-1} - 2} i_s, \dots \dots \dots (2.18)$$

$$o'_q = \frac{o_{\max} - o_{\min}}{2^b - 1} i_o + o_{\min}, \dots \dots \dots (2.19)$$

Where, i_s and i_o are the quantization indices of scale and offset coefficients. s_q and o_q are the quantized values of scale and offset coefficients respectively. The quantized values of scale and offset parameters should be used to construct the approximates r' and the sum of errors (χ^2).

To assess the involved computational complexity; consider an $n \times n$ image partitioned into non-overlapping range blocks, each block has a size $(k \times k)$. The number of tiled range blocks is n^2/k^2 , while the number of domain blocks is $(n-2k-1)^2$. The computation of best match between a range block and a domain block is $O(k^2)$. Considering k to be constant, the computational complexity of an exhaustive search is $O(n^4)$.

The most direct and easy way to reduce the search complexity is by monitoring the matching error; at any matching instance the IFS matching error is checked. If it is below a pre-defined permissible level ϵ (threshold) then the registered domain block is considered as the best matched block and, then, the search across the domain blocks is stopped [Geor06].

2.7 Iterated Function System for Zero-Mean Blocks

The traditional offset factor has dynamic range $[-255, 255]$, this may cause large errors in some image regions (or points), especially those

points belong to high contrast area. Also, the traditional offset factors require an additional bit (sign-bit). The results of some conducted tests indicated that the offset values of adjacent range blocks doesn't show significant correlation similar to that registered between the average brightness values of the adjacent blocks. So, to handle this disadvantage a change in the traditional IFS scheme was introduced, where the contractive affain transform is changed to become [ToPi01]:

$$r'_i - \bar{r} = s(d_i - \bar{d}) , \dots \dots \dots (2.20)$$

Where,

$$\bar{r} = \frac{1}{m} \sum_{i=0}^{m-1} r_i , \dots \dots \dots (2.21)$$

$$\bar{d} = \frac{1}{m} \sum_{i=0}^{m-1} d_i , \dots \dots \dots (2.22)$$

To determine the scale (s) value, the method of least sum of square errors (equations 2.13) is applied to get [Geor06],

$$s = \begin{cases} \frac{\frac{1}{m} \sum_{i=0}^{m-1} d_i r_i - \bar{d} \bar{r}}{\sigma_d^2} & \text{if } \sigma_d^2 > 0 \\ 0 & \text{if } \sigma_d^2 = 0 \end{cases} , \dots \dots \dots (2.23)$$

$$\chi^2 = \sigma_r^2 + s \left[s \sigma_d^2 + 2 \bar{d} \bar{r} - \frac{2}{m} \sum_{i=0}^{m-1} d_i r_i \right] , \dots \dots \dots (2.24)$$

Where,

$$\sigma_d^2 = \frac{1}{m} \sum_{i=0}^{m-1} d_i^2 - \bar{d}^2 , \dots \dots \dots (2.25)$$

$$\sigma_r^2 = \frac{1}{m} \sum_{i=0}^{m-1} r_i^2 - \bar{r}^2 , \dots \dots \dots (2.26) \text{ [Geor06]}$$

2.8 Moment Descriptor

In general, moments are set of parameters which describes the distribution of material (in image processing it is equivalent to brightness) relative to a reference point or an axis. The idea of using moments to construct the image feature vectors is one of the most common methods used today. Each moment order reflects different information for the same image.

For a 2-D continuous function $f(x,y)$, the moment of order $(p+q)$ is defined as :

$$m_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x,y) dx dy , \dots \dots \dots (2.27)$$

For $p, q = 0, 1, 2, \dots$

A uniqueness theorem states that: if $f(x,y)$ is piecewise continuous and has nonzero values only in a finite part of the xy -plane, moments of all orders exist, and the moment sequence (m_{pq}) is uniquely determined by $f(x, y)$. Conversely, the set of moments $\{m_{pq}\}$ uniquely determines $f(x, y)$.

The central moments are defined as:

$$\mu_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - x_c)^p (y - y_c)^q f(x,y) dx dy , \dots \dots \dots (2.28)$$

Where

$$x_c = \frac{m_{10}}{m_{00}} \quad \text{and} \quad y_c = \frac{m_{01}}{m_{00}}$$

For a 2-D discrete function $f(x,y)$, the moment of order $(p+q)$ about the center point (x_c, y_c) is defined as [Gonz02]:

$$\mu_{pq} = \sum_y \sum_x (x - x_c)^p (y - y_c)^q f(x, y) , \dots\dots\dots(2.29)$$

When this definition is applied to determine the first order moments of the domain and range blocks the following expressions are obtained:

$$M_d(1,0) = \sum_{i=0}^{m-1} (x_i - k_c)(d_i - \bar{d})$$

$$M_d(0,1) = \sum_{i=0}^{m-1} (y_i - k_c)(d_i - \bar{d})$$

, \dots\dots\dots(2.30)

$$M_r(1,0) = \sum_{i=0}^{m-1} (x_i - k_c)(r_i - \bar{r})$$

$$M_r(0,1) = \sum_{i=0}^{m-1} (y_i - k_c)(r_i - \bar{r})$$

Where,

$$\bar{d} = \frac{1}{m} \sum_{i=0}^{m-1} d_i , \dots\dots\dots(2.31)$$

$$\bar{r} = \frac{1}{m} \sum_{i=0}^{m-1} r_i , \dots\dots\dots(2.32)$$

$$k_c = \frac{k-1}{2} , \dots\dots\dots(2.33)$$

k is the block width (or height).

2.9 Moments Ratio Factor

Consider the following Moments-Ratio factor (R):

$$R = \begin{cases} \frac{M(0,1)}{M(1,0)} & \text{if } |M(1,0)| \geq |M(0,1)| \\ \frac{M(1,0)}{M(0,1)} & \text{if } |M(0,1)| > |M(1,0)| \end{cases} , \dots\dots\dots(2.34)$$

It is easily to prove that the magnitude of R factor is rotation and reflection invariant. Also, by combining equations (2.34), (2.30) and (2.20), we can easily prove that:

$$R_d = R_r , \dots\dots\dots(2.35)$$

This result implies that "if the range and domain blocks satisfy the contractive affine transform, then their moments ratio factors (R_d and R_r) should have similar magnitudes. This doesn't mean that any two blocks have similar R magnitudes are necessarily similar to each other".

This fact is utilized to improve (speed up) the range-domain search task. Instead of comparing all domain blocks with each affine transformed range block, only the domain blocks whose moments-ratio factors (R) are similar to that of the tested range block should be passed through the IFS-matching test [Geor06].

2.10 Down Sampling Methods

Down sampling is a process used for minification only. It may be used to create thumbnail representations of an image. The basic idea behind down sampling process is to represent a block of adjacent pixels with one pixel. The type of down-sampling method depends on the speed and quality requirements. The most popular down-sampling methods are [Cran97]:

1. Median Representation

Median representation replaces a block of pixels with its median value, see Figure (2.5) where an $n \times n$ window is passed over the image. For each down sampled block, its pixels values are read and put into an array, and then sorted in ascending order according to their values. The middle value is then used to represent that block. This method requires much computation time due to the number of comparisons needed to sort the block of pixels.

2. Average Representation

Average representation also uses the $n \times n$ window (see Figure 2.6). Each block of pixels is represented by the average of all pixels values. This is not as slow as median representation.

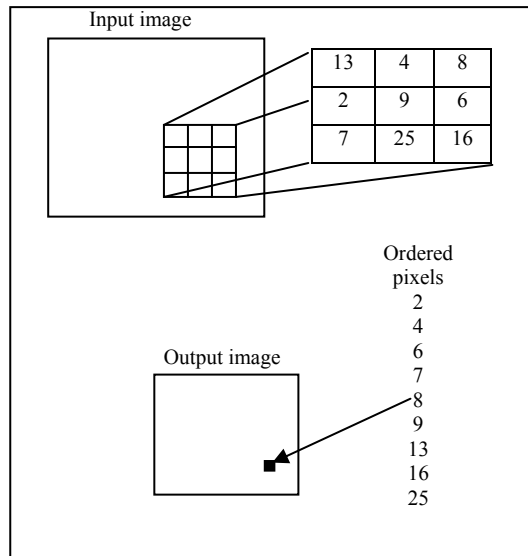


Figure (2.5) Minification by median representation

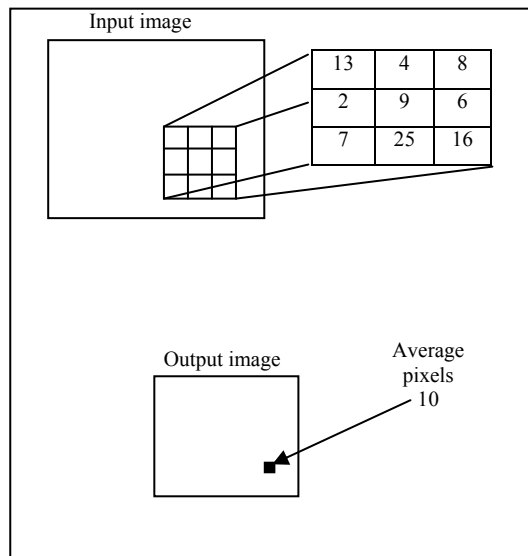


Figure (2.6) Minification by average representation

2.11 Affine Transformation

An affine transformation is the composition of a linear transformation with translation. It can be written as:

$$w(x, y) = (ax + by + e, cx + dy + f) = (x', y') \quad , \dots \dots \dots (2.36)$$

Where, w is the affine transformation, $(a, b, c, d, e,$ and $f)$ are real numbers, (x, y) are the old coordinates of the transformed point, and (x', y') are the new coordinates of the point.

This transformation is a two-dimensional affine transformation [Colv96], it maps a plane to itself. In matrix form, the general equation of an affine transformation is:

$$w \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} \quad , \dots \dots \dots (2.37) \text{ [San97]}$$

$$w(X) = AX + T \quad , \dots \dots \dots (2.38)$$

Where, A is a (2×2) real matrix

e is the translation along x-direction

f is the translation along y-direction

(a, b, c, d) are the coefficients of combined isometric operations
(i.e., scaling, rotation, skew, reflection).

$T = \begin{pmatrix} e \\ f \end{pmatrix}$ is called the translation vector [Colv96]

Affine transformations can skew, rotate, scale and translate a matrix [Sank98]. As a special case, a matrix A can be written in the form:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} r_1 \cos \theta & -r_2 \sin \theta \\ r_1 \sin \theta & r_2 \cos \theta \end{pmatrix} \quad , \dots \dots \dots (2.39)$$








Where, $r_1 = \sqrt{a^2 + c^2}$ is the scaling factor along x-direction.

$r_2 = \sqrt{b^2 + d^2}$ is the scaling factor along y-direction.

$\tan \theta = \frac{c}{a}$ is the angle of rotation around x-direction.

There are seven (simple) special cases of affine transformations; Table (2.1) illustrates these cases [Ning97, BaHu93]:

Table (2.1) The special cases of affine transformations

| Special Cases | Affine Transformation | Figures |
|----------------------|---|---|
| 1.Identity | $w(X) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ |  |
| 2.Translation | $w(X) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$ |  |
| 3.Dilation | $w(X) = \begin{pmatrix} r_1 & 0 \\ 0 & r_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ If $r > 0$ stretch, if $r < 0$ contrast |  |
| 4.Reflection | about x axis: $w(X) = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ about y axis: $w(X) = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ |  |
| 5.Rotation | $w(X) = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ $0 \leq \theta \leq 2\pi$ |  |
| 6.Skewing | $w(X) = \begin{pmatrix} 1 & 0 \\ c & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ or $w(X) = \begin{pmatrix} 1 & b \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$ |  |
| 7.Similitude | $w(X) = \begin{pmatrix} r \cos \theta & r \sin \theta \\ r \sin \theta & -r \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$ $w(X) = \begin{pmatrix} r \cos \theta & -r \sin \theta \\ r \sin \theta & r \cos \theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$ $0 \leq \theta \leq 2\pi$ |  |

Eight transformation matrices could be obtained from the processes of rotation (0, 90, 180, 270) and reflection; these transformations matrices are called the standard indexed spatial matrices. Table (2.2) illustrates the effects of these 8 transformation matrices [Ning97]:

Table (2.2) The standard indexed spatial matrices [Ning97]

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|--|---|---|
|  |  |  |  |  |  |  |  |
| Identity | rotation 90 | rotation180 | rotation270 | x-flip | Flip+ rotation 90 | Flip+ rotation180 | Flip+ rotation270 |
| $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ | $\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$ | $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ | $\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$ | $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ | $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ | $\begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$ |

2.12 Partition Schemes

The first decision to be made when designing a fractal coding scheme is the choice of the type of image partition used for the range blocks. Since domain blocks must be transformed to cover range blocks, this decision, together with the choice of block transformation, restricts the possible sizes and shapes of the domain blocks. A wide variety of partitions have been investigated, the majority being composed of rectangular blocks.

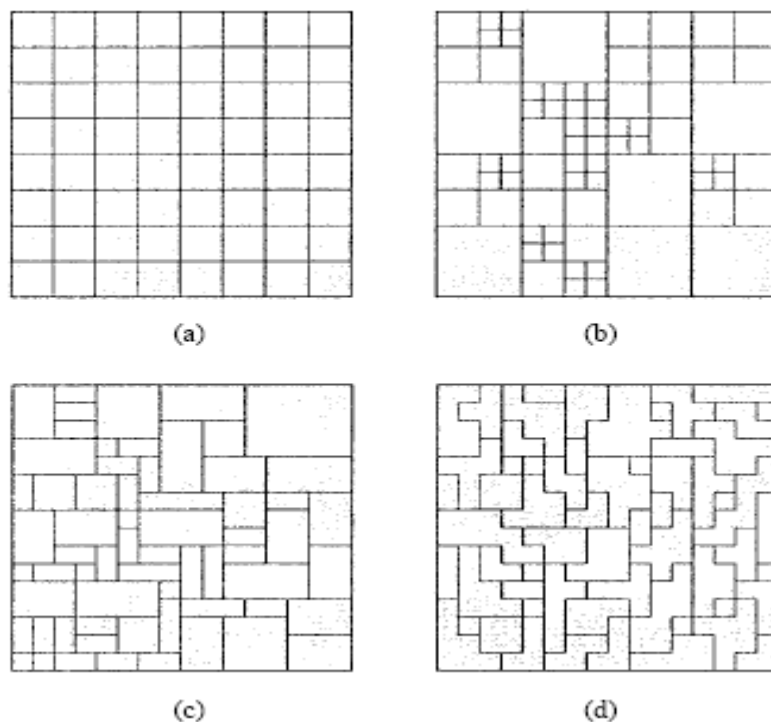


Figure (2.7) Right-angled range partition schemes. (a) Fixed block size, (b) Quadtree, (c) Horizontal-vertical, (d) Irregular partition [WoJa99]

1. Fixed Size Square Blocks

The simplest possible range partition consists of fixed size square blocks, as depicted in Figure (2.7a). This type of block partition is successful in transform coding of individual image blocks since an adaptive quantization mechanism is able to compensate for the varying “activity” levels of different blocks, allocating few bits to blocks with little detail and many to detailed blocks [WoJa99].

An image is partitioned into a set of nonoverlapped, equally spaced, fixed size, small rectangular blocks. In such case the translation, rotation and zooming can be made easily [ShSu00].

Fractal coding based on the standard block transform, is not capable of such adaptation, representing a significant disadvantage of this type of block partition for fractal coding. This deficiency may be addressed by introducing adaptivity to the available block transforms, but the usual

solution is to introduce an adaptive partition with large blocks in low detail regions and small blocks where there is significant detail. Of course, there is a tradeoff between the lower distortion expected by adapting the partition to the image content, and the additional bits required to specify the partition details.

2. Quadtree

The quadtree partition, see Figure (2.7b), employs the well-known image processing technique that based on a recursive splitting of selected image quadrants [WoJa99]. An image is represented as a tree in which each node, corresponding to a square portion of the image, contains four subnodes, corresponding to the four quadrants of the square. The root of the tree is the initial image [Fish95]. The usual top-down construction starts by selecting an initial level in the tree, corresponding to some maximum range block size, and recursively partitioning any block for which a match better than some pre-selected threshold is not found. The partitioning decision could depend of volume of details existing in the block, the details could be measured using various homogeneity measures. The alternative bottom-up construction begins with a uniform partition using the smallest block size, and then proceeds to merge those neighboring blocks for which a more efficient representation is provided by the resulting larger block, which is one level up the quadtree. Compact coding of partition details is possible by taking advantage of the tree structure of the partition [WoJa99].

Jacquin's original PIFS scheme used a variant of the quadtree partition in which the block splitting was restricted to two levels. Instead of automatically discarding the larger block prior to splitting it into four sub-blocks if an error threshold was exceeded, it was retained if

additional transforms on up to two sub-blocks were sufficient to reduce the error below the threshold [Fish95].

3. Horizontal-Vertical

The horizontal-vertical (HV) partition, see Figure (2.7c), like the quadtree, produces a tree-structured partition of the image. Instead of recursively splitting quadrants, however, each image block is split into two sub-blocks by a horizontal or vertical line. Splitting positions may be constructed so that boundaries tend to fall along prominent edges, or it is based on the accuracy of approximation by constant pixel values in each of the new blocks created by a particular split. Compact coding of the partition details, similar to that utilized for the quadtree partition, is possible [WoJa99].

4. Irregular Regions

A tiling of the image by right-angled irregular-shaped ranges may be constructed by a variety of merging strategies on an initial fixed square block, see Figure (2.7d), or quadtree partition. Chain codes allow the range shapes to be coded efficiently [Fish95].

5. Overlapped Blocks

Overlapping range blocks have been used to reduce blocking artifacts, within a quadtree partition, and with multiple domain transforms in a fixed block size partition. This overlapping step may not lead to a corresponding improvement in mean square error. A more complex form of block overlapping, but with a fixed block size range partition, provided improved MSE and subjective quality. These techniques, while are promising, have been overtaken to a large extent by developments in wavelet domain fractal coding [WoJa99].

2.13 Quantization

The dictionary definition of the term “quantization” is “to restrict a variable quantity to discrete values rather than to a continuous set of values” [Salo07]. Any analog quantity that is to be processed by a digital computer or digital system must be converted to an integer number proportional to its amplitude. The conversion process between analog samples and discrete-valued samples is called quantization [Prat01].

In the field of data compression, quantization is used in two ways:

1. If the data to be compressed is in the form of large numbers, quantization is used to convert it to small numbers. Small numbers take less space than large ones, so quantization generates compression. On the other hand, small numbers generally contain less information than large ones, so quantization results in lossy compression.
2. If the data to be compressed is analog (e.g., a voltage that changes with time) quantization is used to digitize it into small numbers. The smaller the numbers the better the compression, but also the greater the loss of information. This aspect of quantization is used by several speech compression methods.

2.13.1 Scalar Quantization

Scalar quantization is an example of a lossy compression method, where it is easy to control the trade-off between compression ratio and the amount of loss. However, because it is so simple, its use is limited to cases where much loss can be tolerated [Salo07].

The amplitude of an analog signal sample is compared to a set of decision levels. If the sample amplitude falls between two decision levels, it is quantized to a fixed reconstruction level lying in the quantization

band. In digital systems, each quantized sample is assigned a binary code [Prat01].

2.14 DPCM (Differential Pulse Coding Modulation)

The DPCM system was developed at Bell Laboratories a few years after World War II. It is most popular as a speech-encoding system, and it is widely used in telephone communications [Sayo06].

The DPCM compression method is a member of the family of differential encoding (compression) methods, which itself is a generalization of the simple concept of relative encoding. It is based on the well-known fact that neighboring pixels in an image (and also adjacent samples in digitized sound) are correlated. Correlated values are generally similar, so their differences are small which resulting in compression. Table (2.3) lists 25 consecutive values of the function $\sin \theta$, calculated for θ values from 0 to 360° in steps of 15° . The values therefore range from -1 to $+1$, but the 24 differences $\sin \theta_{i+1} - \sin \theta_i$ (also listed in the table) are all in the range $[-0.259, 0.259]$. The average of the 25 values is zero, as is the average of the 24 differences. However, the variance of the differences is small, since they are all closer to their average.

Table (2.3) 25 Sine values and 24 Differences

| | | | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|-------|
| Sin(t) | 0 | 0.259 | 0.500 | 0.707 | 0.866 | 0.966 | 1.000 | 0.966 | |
| Diff | – | 0.259 | 0.241 | 0.207 | 0.159 | 0.100 | 0.034 | -0.034 | |
| Sin(t) | 0.866 | 0.707 | 0.500 | 0.259 | 0 | -0.259 | -0.500 | -0.707 | |
| Diff | -0.100 | -0.159 | -0.207 | -0.241 | -0.259 | -0.259 | -0.241 | -0.207 | |
| Sin(t) | -0.866 | -0.966 | -1.000 | -0.966 | -0.866 | -0.707 | -0.500 | -0.259 | 0 |
| Diff | -0.159 | -0.100 | -0.034 | 0.034 | 0.100 | 0.159 | 0.207 | 0.241 | 0.259 |

Figure (2.8a) shows the histogram of an image that consists of 8-bit pixels. For each possible pixel value (between 0 and 255) there is a different number of pixels. Figure (2.8b) shows the histogram of the differences between consecutive pixels. It is easy to see that most of the differences (which, in principle, can be in the range $[-255, 255]$) are small; only a few are outside the range $[-50, +50]$.

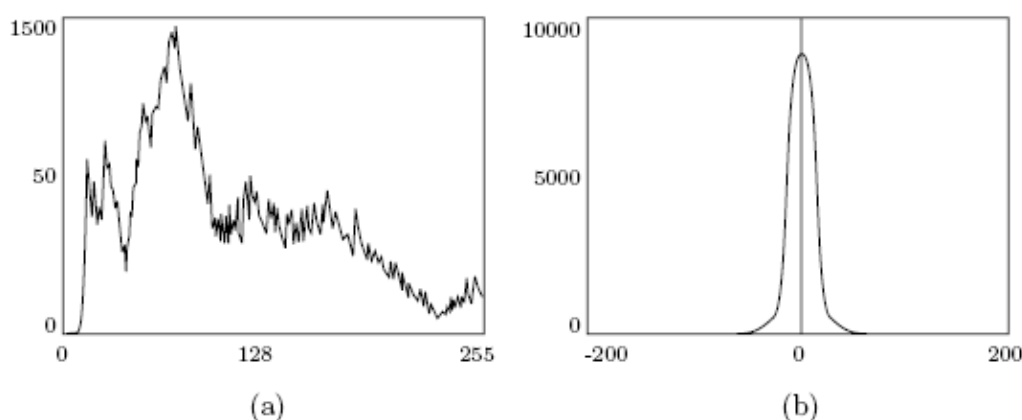


Figure (2.8) A Histogram of an image pixel values and of their differences

Differential encoding methods calculate the differences $d_i = a_i - a_{i-1}$ between consecutive data items a_i , and encode the d_i 's [Sayo06, Salo07]. The first data item, a_0 , is either encoded separately or is written on the compressed stream in raw format. In either case the decoder can decode and generate a_0 in exact form. In principle, any suitable method, lossy or lossless, can be used to encode the differences. In practice, quantization is often used, resulting in lossy compression. The quantity encoded is not the difference d_i but a similar, quantized number that is denoted by \hat{d}_i . The difference between d_i and \hat{d}_i is the *quantization error* q_i [Salo07].

It turns out that the lossy compression of differences introduces a new problem, namely, the accumulation of errors. This is easy to see

when the operation of the decoder is considered. The decoder get (as input) the encoded values of \hat{d}_i , decodes them, and uses them to generate "reconstructed" values \hat{a}_i (where $\hat{a}_i = \hat{a}_{i-1} + \hat{d}_i$) instead of the original data values a_i . The decoder starts by reading and decoding a_0 . It then inputs $\hat{d}_1 = d_1 + q_1$ and calculates $\hat{a}_1 = a_0 + \hat{d}_1 = a_0 + d_1 + q_1 = a_1 + q_1$. The next step is to input $\hat{d}_2 = d_2 + q_2$ and to calculate $\hat{a}_2 = \hat{a}_1 + \hat{d}_2 = a_1 + q_1 + d_2 + q_2 = a_2 + q_1 + q_2$. The decoded value \hat{a}_2 contains the sum of two quantization errors. In general, the decoded value \hat{a}_n equals [Salo07]

$$\hat{a}_n = a_n + \sum_{i=1}^n q_i, \dots\dots\dots(2.40) \text{ [ShSu00]}$$

and it includes the sum of n quantization errors. Sometimes, the errors q_i are signed and tend to cancel each other out in the long run. In general, however, this is a problem.

The solution is easy to understand once it is realized that the encoder and the decoder operates on different pieces of data. The encoder generates the exact differences d_i from the original data items a_i , while the decoder generates the reconstructed \hat{a}_i using only the quantized differences \hat{d}_i . The solution is, therefore, to modify the encoder to calculate differences of the form $d_i = a_i - \hat{a}_{i-1}$. The difference d_i is calculated by subtracting the most recent reconstructed value \hat{a}_{i-1} (which both encoder and decoder have) from the current original item a_i .

The decoder starts by reading and decoding a_0 . It then inputs $\hat{d}_1 = d_1 + q_1$ and calculates $\hat{a}_1 = a_0 + \hat{d}_1 = a_0 + d_1 + q_1 = a_1 + q_1$. The next step is to input $\hat{d}_2 = d_2 + q_2$ and calculates $\hat{a}_2 = \hat{a}_1 + \hat{d}_2 = \hat{a}_1 + d_2 + q_2 = a_2 + q_2$. The decoded value \hat{a}_2 contains just the single quantization error q_2 . And, in general, the decoded value \hat{a}_i equals $a_i + q_i$, so it contains just

quantization error q_i . The *quantization noise* in decoding \hat{a}_i equals the noise generated when a_i was quantized.

Figure (2.9a) summarizes the operations of both encoder and decoder. It shows how the current data item a_i is saved in a storage unit (a delay), to be used for encoding the next item a_{i+1} .

The next step in developing a general differential encoding method is to take advantage of the fact that the data items being compressed are correlated. This means that in general, an item a_i depends on *several* of its near neighbors, not just on the preceding item a_{i-1} . Better prediction (and, as a result, smaller differences) can therefore be obtained by using N of the previously-seen neighbors to encode the current item a_i (where N is a parameter). Therefore, it would like to have a function $p_i = f(\hat{a}_{i-1}, \hat{a}_{i-2}, \dots, \hat{a}_{i-N})$ to predict a_i , see Figure (2.9b) [Salo07].

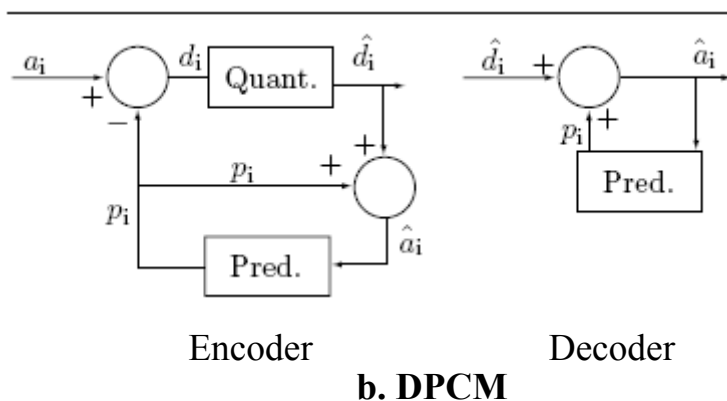
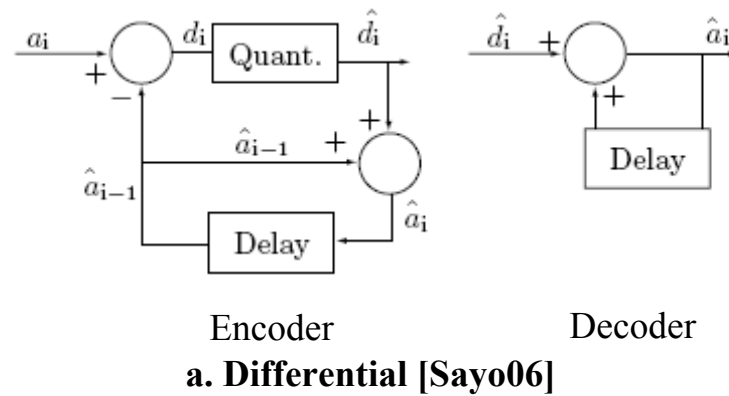


Figure (2.9) A Differential and DPCM codecs

Notice that f has to be a function of the \hat{a}_{i-j} , not the a_{i-j} , since the decoder has to calculate the same f . Any method using such a predictor is called *differential pulse code modulation*, or DPCM. In practice, DPCM methods are used mostly for audio compression, but are illustrated here in connection with image compression [Salo07].

2.15 The Test Measures

A lot of key parameters were utilized in the literature to describe the performance of various compression methods. In this research the fidelity criteria (MAE, MSE, and PSNR) in addition to the compression ratio and bit rate were used to describe the performance of the established four FIC schemes at different coding conditions:

A. Fidelity Criteria

There are several types of matching criteria, among which the mean square error (MSE) and mean absolute difference (MAD) are used most often. It is noted that the sum of the squared difference (SSD) or the sum of the squared error (SSE) is essentially same as MSE. The mean absolute difference is sometimes referred to as the mean absolute error (MAE).

In the MSE matching criterion, the dissimilarity metric $M(u,v)$ is defined as [ShSu00]

$$M(u,v) = (u-v)^2, \dots \dots \dots (2.41)$$

While, in the MAD,

$$M(u,v) = |u-v|, \dots \dots \dots (2.42)$$

Developers of lossy image compression methods need a standard metric to measure the quality of reconstructed images compared with the

original ones. Well reconstructed image resembles the original one, and the metric value should indicate this resemblance in proper way. Such a metric is a dimensionless number, and that number should not be very sensitive to small variations in the reconstructed image. The most common measure used for this purpose is the *peak signal to noise ratio* (PSNR). It is familiar to workers in the field, it is also simple to calculate, but it has only a limited, approximate relationship with the perceived errors noticed by the human visual system. This is why higher PSNR values imply closer resemblance between the reconstructed and the original images, but they do not provide a guarantee that viewers will like the reconstructed image.

Denoting the pixels of the original image by P_i and the pixels of the reconstructed image by Q_i (where $1 \leq i \leq n$), then the *mean square error* (MSE) between the two images is defined as:

$$MSE = \frac{1}{n} \sum_{i=1}^n (P_i - Q_i)^2 \quad \dots\dots\dots(2.43)$$

It is the average of the square of the errors (pixels' differences) of the two images. The *root mean square error* (RMSE) is defined as the square root of the MSE [Salo07]. The Peak signal-to-noise ratio (PSNR) is defined as [Fish95]:

$$PSNR = 20 \log_{10} \left(\frac{\max_i |P_i|}{RMSE} \right) \quad \dots\dots\dots(2.44)$$

The absolute value is normally not needed, since pixel values are rarely negative. For a bi-level image, the numerator is 1. For a grayscale image with eight bits per pixel, the numerator is 255.

Greater resemblance between the images implies smaller RMSE and, as a result, larger PSNR. The PSNR is dimensionless, since the units of

both numerator and denominator are pixel values. However, because of the use of the logarithm, it can be said that the PSNR value is expressed in *decibels* (dB). The use of the logarithm also implies less sensitivity to changes in the RMSE. The PSNR has no absolute meaning, it is meaningless to say that a PSNR of, say, 25 is good. PSNR values are used only to compare the performance of different lossy compression methods, or to describe the effects of different parametric values on the performance of an algorithm. The MPEG committee, for example, uses an informal threshold of PSNR= 0.5 dB to decide whether to incorporate a coding optimization, since they believe that an improvement of that magnitude would be visible to the eye.

Typical PSNR values range between 20 and 40. Assuming pixel values in the range [0, 255], a RMSE value of 25.5 results in a PSNR of 20, and a RMSE value of 2.55 results in a PSNR of 40. A RMSE of zero (i.e., identical images) results in an infinite (more precisely, undefined) PSNR. A RMSE of 255 results in a PSNR of zero, and RMSE values greater than 255 yield negative PSNRs.

Some authors define the PSNR as

$$PSNR = 10 \log_{10} \left(\frac{\max_i |P_i|^2}{MSE} \right), \dots\dots\dots(2.45)$$

In order for the two formulations to produce the same result, the logarithm is multiplied in this case by 10 instead of 20, since $\log_{10}(A^2) = 2\log_{10}(A)$. Either definition is useful, because only relative PSNR values are used in practice [Salo07].

B. Compression Compactness

Various measures are used to describe the achieved reduction in data size due to compression, in this research work the compression ration (Cr) and bit rate were adopted as compression measures.

Compression ratio is used to refer to the degree of reduction of image file (or data) size due to compression process. This measure is defined as the ratio between the size of the original uncompressed image file to the size of the overall compressed file [Umba98]:

$$Cr = \frac{\text{uncompression file size}}{\text{compression file size}}, \dots\dots\dots(2.46)$$

Bit rate (BR) refers to the average number of bits required to represent the value of each image pixel, usually it is determined as the ratio between the size of compressed file and the size of the original image file:

$$BR = \frac{\text{compression file size (in bits)}}{\text{image file size (in pixels)}}, \dots\dots\dots(2.47)$$

The above defined Cr and BR parameters have been used as indicators for the compactness ability of the proposed compression schemes in this research project.

2.16 Entropy

The entropy of a single symbol a_i is defined as $-P_i \log_2 P_i$, where P_i is the probability of occurrence of a_i in the data. The entropy of a_i is the smallest number of bits needed, on average, to represent symbol a_i . Claude Shannon, the creator of information theory, coined the term entropy in 1948, since this term is used in thermodynamics to indicate the amount of disorder in a physical system.

Assume the H is the amount of information, in bits, sent by the transmitter in one time unit. The amount of information contained in one base- n symbol is thus H/s (because it takes time $1/s$ to transmit one symbol), or $-\sum_1^n P_i \log_2 P_i$. This quantity is called the entropy of the data being transmitted. In analogy we can define the entropy of a single symbol a_i can be defined as $-P_i \log_2 P_i$. This is the smallest number of bits needed, on average, to represent the symbol.

The entropy of the data depends on the individual probabilities P_i , and its largest value occurred when all n probabilities are equal [Salo04].

2.17 Huffman Coding

This technique was developed by David Huffman as part of a class assignment; the class was the first ever in the area of information theory, and was taught by Robert Fano at MIT. The codes generated using this technique or procedure are called *Huffman codes*. These codes are prefix codes and are optimum for a given model (set of probabilities).

The Huffman procedure is based on two observations regarding optimum prefix codes [Sayo06].

1. The more frequently occurring symbols can be allocated with shorter codewords than the less frequently occurring symbols .
2. The two least frequently occurring symbols will have codewords of the same length, and they differ only in the least significant bit [TiAj05].

It is easy to see that the first observation is logical and correct. If symbols that occur more often had longer codewords than the codewords for symbols that occurred less often, the average number of bits per symbol would be larger than that obtained when the conditions were

reversed. Therefore, a code that assigns longer codewords to symbols that occur more frequently cannot be optimum.

A simple application of Huffman coding to image compression would be to generate a Huffman code for the set of values that any pixel may take. For monochrome images, this set usually consists of integers from 0 to 255 [Sayo06].

Huffman coding is a popular method for data compression. It serves as the basis for several popular programs run on various platforms. Some programs use just the Huffman method, while others use it as one step in a multistep compression process. It generally, it produces better codes. It produces the best code when the probabilities of the symbols are negative powers of 2. Huffman constructs a code tree from the bottom up (builds the codes from right to left). Since its development, in 1952, this method has been the subject of intensive research into data compression.

The algorithm starts by building a list of all the alphabet symbols in descending order of their probabilities. It then constructs a tree, with a symbol at every leaf, from the bottom up. This is done in steps, where at each step the two symbols with smallest probabilities are selected, added to the top of the partial tree, deleted from the list, and replaced with an auxiliary symbol representing the two original symbols. When the list is reduced to just one auxiliary symbol (representing the entire alphabet), the tree is complete. The tree is then traversed to determine the codes of the symbols [Salo07].

2.18 Arithmetic Coding

The method of generating variable-length codes called *arithmetic coding*. Arithmetic coding is especially useful when dealing with sources with small alphabets, such as binary sources, and alphabets with highly skewed probabilities. It is also a very useful approach when, for various

reasons, the modeling and coding aspects of lossless compression are to be kept separate [Sayo06].

The Huffman method is simple, efficient, and produces the best codes for the individual data symbols. However, the only case where it produces ideal variable-size codes (codes whose average size equals the entropy) is when the symbols have probabilities of occurrence that are negative powers of 2 (i.e., numbers such as $1/2$, $1/4$, or $1/8$). This is because the Huffman method assigns a code with an integral number of bits to each symbol in the alphabet. Information theory shows that a symbol with probability 0.4 should ideally be assigned a 1.32-bit code, since $-\log_2 0.4 \approx 1.32$. The Huffman method, however, normally assigns such a symbol a code of 1 or 2 bits.

Arithmetic coding overcomes the problem of assigning integer codes to the individual symbols by assigning one (normally long) code to the entire input data. The method starts with a certain interval, it reads the input file symbol by symbol, and it uses the probability of each symbol to narrow the interval. Specifying a narrower interval requires more bits, so the number constructed by the algorithm grows continuously. To achieve compression, the algorithm is designed such that [Salo07] more probable symbols reduce the interval less than the less probable symbols and hence add fewer bits in the encoded message [TiAj05], with the result that high-probability symbols contribute fewer bits to the output [Salo07].

2.19 Shift Coding

The idea of this method is to encode the sequence of numbers by codewords whose bit length is less than the bit length required to represent the maximum value of the sequence of numbers to be coded. The numbers whose values are large may be splitted into a sequence of codewords, by using the following formula:

$$X = nW_m + W_r \text{ ,(2.48)}$$

Where:

X is the number to be coded.

n is the number of codewords that used to encode the number X .

W_m is the lowest value which cannot be coded by using a single codeword.

W_r is the value of the last codeword used to encode X .

The values of W_m , W_r , and n are determined by using the following equations

$$W_m = 2^b - 1 \text{ ,(2.49)}$$

$$W_r = X \bmod W_m \text{ ,(2.50)}$$

$$n = X \operatorname{div} W_m \text{ ,(2.51)}$$

Where b is the number of bits used to represent each single shift codeword.

The performance of Huffman coding and shift coding are better when the sequence of numbers has a histogram whose shape is highly peaked. The performance of shift coding is better than Huffman and arithmetic coding when the histograms have long tails [Mahm07, Gonz02].

Chapter Three

The Enhanced FIC Scheme

Chapter Three

The Enhanced FIC-Scheme

3.1 Introduction

This chapter is dedicated to present the design considerations and implementation requirements, which were taken into consideration throughout the design stage of the proposed enhanced fractal image compression scheme, that has some additional stages to speed-up the compression task in comparison with the traditional scheme.

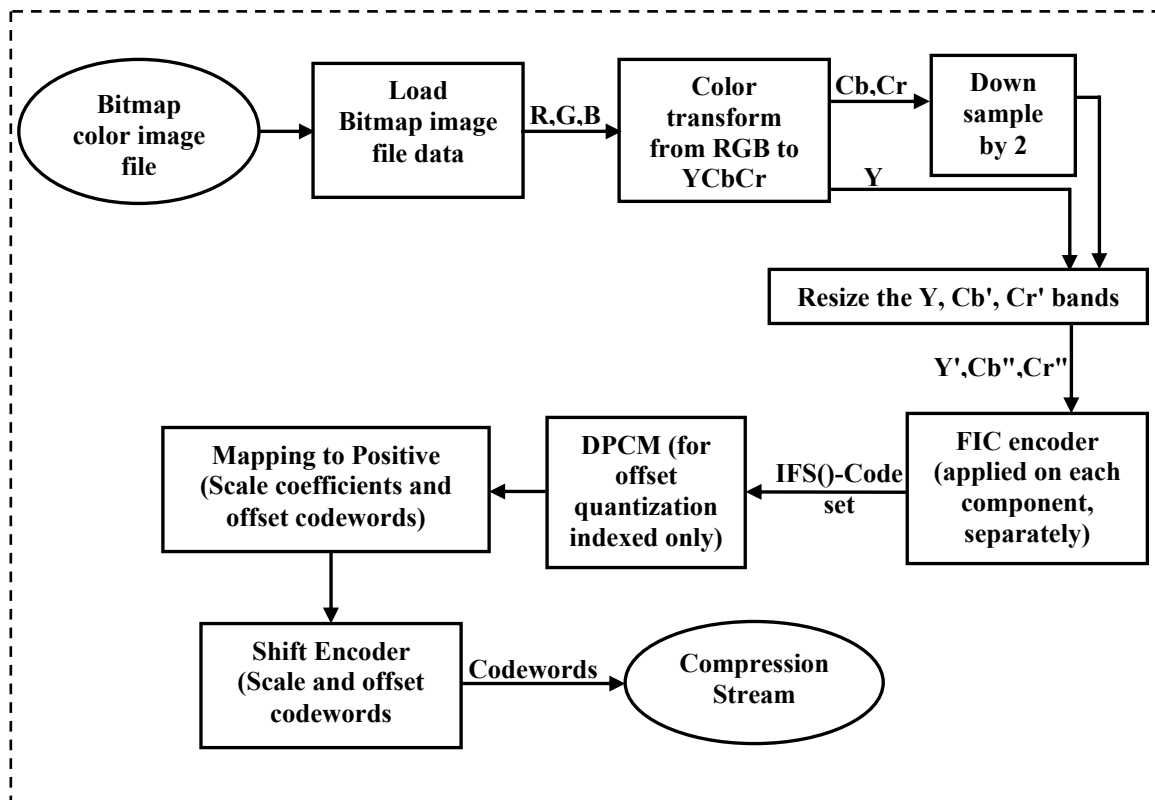
In this research project, two major enhancement steps have been introduced to significantly reduce the time of the elapsed encoding process, without making significant reduction in image quality. The first enhancement step is using the developed moment based predictor to reduce the number of isometric mapping trials, applied on each domain block, from 8 to 1 trial; in other words, the introduced predictor can assign the proper isometric mapping that required to set the domain block in closest form to the matched domain block. The second enhancement step implies using the moment based descriptor to classify the domain and range blocks into classes, and instead of testing all domain blocks belong to domain pool to find out the block the best matches the coded range block (using IFS-mapping), only the domain blocks that have similar class index to that of range block will be passed through the IFS-matching test.

The two main modules of the established FIC-schemes are: encoding and decoding modules. The structure of the established system and the functionality of its modules will be discussed in details in the next sections.

3.2 The System Model

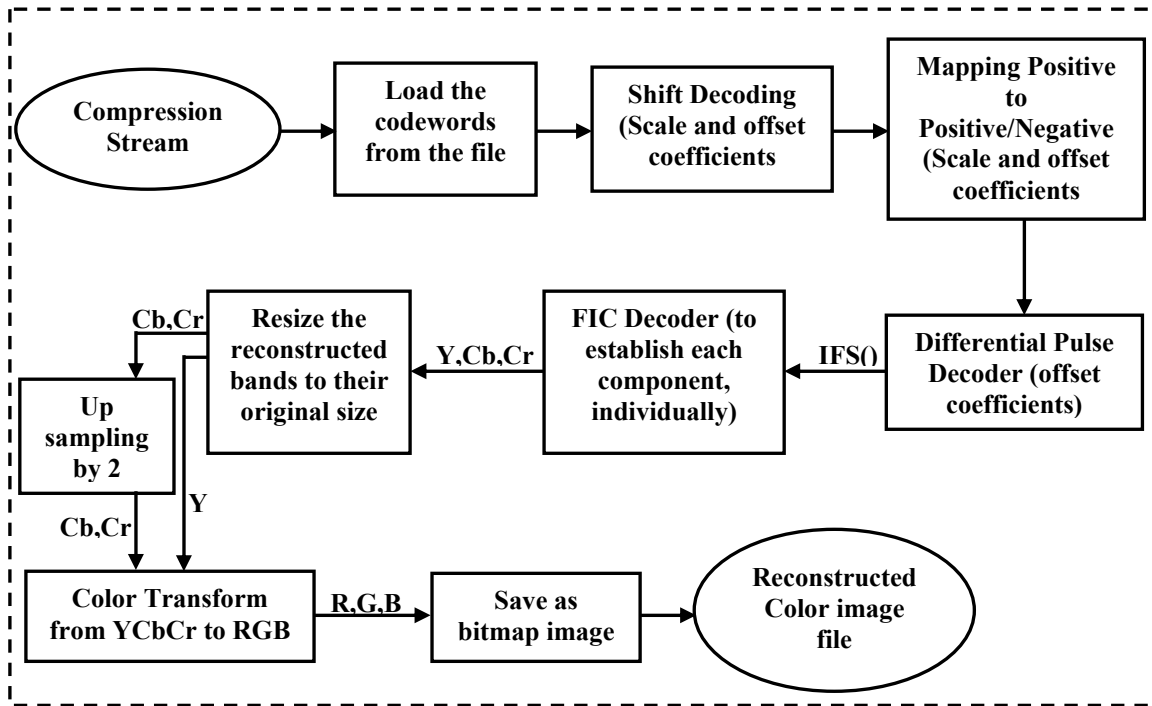
The general structure of the proposed system is illustrated in figure (3.1). It consists of two basic modules: encoding and decoding modules. The input to the encoder module is a BMP (Bitmap) image file. The data of this image is passed through the encoding stages, and subjected to various operations to produce the compressed file. This compressed file could be passed through decoding stages, and subjected to a sequence of operations, to reconstruct the bitmap image.

Each module (i.e. encoder and decoder) implies several operations, working systematically to lead to the final result.



a. Encoding Module

Figure (3.1) The System Model



b. Decoding Module
Figure (3.1) Continue

3.3 Encoder Module

As shown in figure (3.1a) the main stages of the encoder are started from loading image data, and passed through color transform, downsampling, FIC encoding, DPCM, and, as a final stage, shift encoding. The output of the last stage (i.e., codewords) are saved in compressed file.

3.3.1 Load BMP Image

The input to this system is a BMP image file; in the established system, the BMP image file was used as an input to the system. The considered color resolution of the images is either 24 or 8 bit/pixel. The image data is loaded and used to fill-up the Red, Green, and Blue arrays, each array is assigned for one primary color, as illustrated in Algorithm (3.1).

Algorithm (3.1) Read BMP Image**Goal:** Read 24 or 8 bit/pixel BMP image file**Input:***ImgFileName*// image file name**Output:***Wid, Hgt*// image width and height*Red*(0 to *Wid-1*, 0 to *Hgt-1*)// Red component of image*Grn*(0 to *Wid-1*, 0 to *Hgt-1*)// Green component of image*Blu*(0 to *Wid-1*, 0 to *Hgt-1*)// Blue component of image**Step1:** Get from *ImgFileName* the *BMPH* // *BMPH* is the BMP Header**Get** image's width and height values from its header**Set** *Wid* ← *BMPH.Wid***Set** *Hgt* ← *BMPH.Hgt***Step2:** Check image pixel resolution**If** *BMPH.BitPlane* = 24 **Then****Set** *DataSize* ← *BMPH.FileSize* - *BMPH.Size***Get** *ImgFileName*, *Img*(*DataSize-1*) // *Img* contains the image's data**Set** *I* ← 0**For all** *X, Y* **Do** {where $0 \leq X \leq \text{Wid}-1$, $0 \leq Y \leq \text{Hgt}-1$ }**Set** *Red*(*X, Y*) ← *Img*(*I*)**Set** *Grn*(*X, Y*) ← *Img*(*I+1*)**Set** *Blu*(*X, Y*) ← *Img*(*I+2*)**Increment** *I* by 3**End For****Else if** *BMPH.BitPlane* = 8 **Then****Set** *NoColor* ← (*BMPH.OffsetPosition* - 54) div 4**Get** *ImgFileName*, *RGBrecord*(*NoColor-1*) // *RGBrecord* contains 4 cells Red, Green, Blue, and *A* is a reserved byte for BMP images of 8 bit/pixel resolution**Set** *DataSize* ← *BMPH.FileSize* - *BMPH.Size* - *NoColor***Get** *ImgFileName*, *Img*(*DataSize-1*)**Set** *I* ← 0**For all** *X, Y* **Do** {where $0 \leq X \leq \text{Wid}-1$, $0 \leq Y \leq \text{Hgt}-1$ }**Set** *Red*(*X, Y*) ← *RGBrecord*(*Img*(*X*)).*Red***Set** *Grn*(*X, Y*) ← *RGBrecord*(*Img*(*X*)).*Green***Set** *Blu*(*X, Y*) ← *RGBrecord*(*Img*(*X*)).*Blue***End For****Return** (*Wid, Hgt, Red, Grn, Blu*)**Else****Display Message** "The Selected Image's BitPlane is neither 24 nor 8"**End If****Step3:** End.

3.3.2 Conversion from RGB to YCbCr Color Space

In this stage the three obtained basic colors (Red, Green, and Blue) are converted into the YCbCr color representation. This stage is important to make the image data representation more suitable for compression. This conversion is made using equation (2.11). Algorithm (3.2) shows the implemented steps to make this color conversion.

| |
|--|
| <p>Algorithm (3.2) YCbCr color model transformation</p> <p>Goal: Convert the image from RGB to YCbCr color model</p> <p>Input: <i>Wid, Higt</i> <i>Red(0 to Wid-1, 0 to Hgt-1)</i> <i>Grn(0 to Wid-1, 0 to Hgt-1)</i> <i>Blu(0 to Wid-1, 0 to Hgt-1)</i></p> <p>Output: <i>Yc(0 to Wid-1, 0 to Hgt-1)</i>// Y component of the image <i>Cb(0 to Wid-1, 0 to Hgt-1)</i>// Cb component of the image <i>Cr(0 to Wid-1, 0 to Hgt-1)</i>// Cr component of the image</p> <p>Step1: Convert each RGB pixel value into its corresponding YCbCr value For all <i>X, Y</i> Do {where $0 \leq X \leq \text{Wid}-1, 0 \leq Y \leq \text{Hgt}-1$} Set $Yc(X,Y) \leftarrow 0.257 * Red(X,Y) + 0.504 * Grn(X,Y) + 0.098 * Blu(X,Y) + 16$ Set $Cb(X,Y) \leftarrow -0.148 * Red(X,Y) - 0.291 * Grn(X,Y) + 0.439 * Blu(X,Y) + 128$ Set $Cr(X,Y) \leftarrow 0.439 * Red(X,Y) - 0.368 * Grn(X,Y) - 0.071 * Blu(X,Y) + 128$ End For</p> <p>Step2: Return (<i>Yc, Cb, Cr</i>).</p> |
|--|

3.3.3 Down Sampling

The three color components of YCbCr model are: Y-component which represents the luminance, and (Cb, Cr) components which represent the chrominance components of the color image. Most of the data energy is concentrated in Y component, while the components Cb, Cr convey little part of the image information energy. Beside to that the Human Vision System (HVS) doesn't show high spatial discrimination for the chrominance components (Cb, Cr), while it has high discrimination power against the contents of Y-component. So, the chrominance components are

down-sampled by 2 using the averaging method, see section (2.10), which is depicted in algorithm (3.3).

| Algorithm (3.3) Down-Sampling by Averaging |
|--|
| <p>Goal: Down-sampling C_b and C_r by 2</p> <p>Input: Wid, Hgt $C_b(0 \text{ to } Wid-1, 0 \text{ to } Hgt-1)$ $C_r(0 \text{ to } Wid-1, 0 \text{ to } Hgt-1)$</p> <p>Output: $nWid, nHgt$// new width and height after down-sampling $C_b'(0 \text{ to } Wid/2-1, 0 \text{ to } Hgt/2-1)$// C_b component of the image after down-sampling $C_r'(0 \text{ to } Wid/2-1, 0 \text{ to } Hgt/2-1)$// C_r component of the image after down-sampling</p> |
| <p>Step1: Convert the width and height into its corresponding down-sampled ones Set $Wh \leftarrow Wid \text{ div } 2-1$ Set $Hh \leftarrow Hgt \text{ div } 2-1$ If Wid is even number then Set $nWid \leftarrow Wh$ else Set $nWid \leftarrow Wh+1$ If Hgt is even number then Set $nHgt \leftarrow Hh$ else Set $nHgt \leftarrow Hh+1$</p> <p>Step2: Convert C_b and C_r into C_b' and C_r' by down-sampling For all X, Y Do {where $0 \leq X \leq Wh, 0 \leq Y \leq nHgt-1$} Set $C_b'(X, Y) \leftarrow (C_b(X*2, Y*2) + C_b(X*2+1, Y*2) + C_b(X*2, Y*2+1) + C_b(X*2+1, Y*2+1))/4$ Set $C_r'(X, Y) \leftarrow (C_r(X*2, Y*2) + C_r(X*2+1, Y*2) + C_r(X*2, Y*2+1) + C_r(X*2+1, Y*2+1))/4$ End For</p> <p>Step3: Check if there are un-computed pixels in rows or columns If Wid is an odd number of columns then Set $Wm \leftarrow Wid-1$ For all Y Do {where $0 \leq Y \leq Hh$} Set $C_b'(nWid, Y) \leftarrow (C_b(nWid, Y*2) + C_b(nWid, Y*2+1))/2$ Set $C_r'(nWid, Y) \leftarrow (C_r(nWid, Y*2) + C_r(nWid, Y*2+1))/2$ End For End If If Hgt is an odd number of rows then Set $Hm \leftarrow Hgt-1$ For all X Do {where $0 \leq X \leq Wh$} Set $C_b'(X, nHgt) \leftarrow (C_b(X*2, nHgt) + C_b(X*2+1, nHgt))/2$ Set $C_r'(X, nHgt) \leftarrow (C_r(X*2, nHgt) + C_r(X*2+1, nHgt))/2$ End For End If If both Wid and Hgt are odd numbers Then Set $C_b'(nWid, nHgt) \leftarrow C_b(Wm, Hm)$ Set $C_r'(nWid, nHgt) \leftarrow C_r(Wm, Hm)$ End If</p> <p>Step4: Return $(C_b', C_r', nWid, nHgt)$.</p> |

Beside to the manipulation of down-sampling, by applying the averaging method, some additional steps taken to handle the problem of the odd numbers of columns and rows, because the averaging method requires that the numbers of columns and rows should be even.

3.3.4 Resizing the Bands (Y, Cb, Cr)

In this step the bands sizes (i.e., width and height) are adjusted before partitioning each band into range blocks.

The image will loose some of its pixels if the block length is not suitable relative to the width and height of each band. This problem is solved by resizing the three bands (Y, Cb, and Cr) to set their dimensions as multiples of the block length. The bilinear interpolation method was used to create the additional columns and rows. The maximum number of columns or rows that may required to be generated depends on the block size. The number of additional columns or rows is determined using the following equation:

$$n = b \left\lceil \frac{S}{b} \right\rceil - S \quad \dots\dots\dots (3.1)$$

Where:

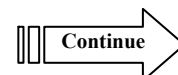
S is the width or height of the image.

b is the block size.

n is the number of additional columns or rows.

$\lceil x \rceil$ is the lowest integer number higher than or equal to x.

Algorithm (3.4) shows the steps taken to perform the bands resizing task.

Algorithm (3.4) Resize the Three Bands**Goal:** Resize the bands Y, Cb, and Cr**Input:***Bnd()*// one of these bands Y, Cb, or Cr*W*// width of the band*H*// height of the band*BlkLength*// the block length**Output:***nBnd*// the new band after resizing*nW*// the new width*nH*// the new height**Step1:** Check if the width and the height are accept the division by the block length without any restCheck **If** ($W \bmod \text{BlkLength}$) not zero **Or** ($H \bmod \text{BlkLength}$) not zero **Then** **Set** $N_x \leftarrow W \text{ div } \text{BlkLength}$ **Set** $W_p \leftarrow N_x * \text{BlkLength}$ Check **If** $W_p < W$ **Then** **Increment** N_x by 1 **Set** $W_p \leftarrow W_p + \text{BlkLength}$ **End If** **Decrement** N_x by 1 **Set** $W_{pm} \leftarrow W_m - 1$ **Set** $W_m \leftarrow W - 1$ **Set** $L_x \leftarrow W_p - W$ **Set** $T_x \leftarrow W_m / (L_x + 1)$ //These statements were set to solve the width problem **Set** $N_y \leftarrow H \text{ Div } \text{BlkLength}$ **Set** $H_p \leftarrow N_y * \text{BlkLength}$ Check **If** $H_p < H$ **Then** **Increment** N_y by 1 **Set** $H_p \leftarrow H_p + \text{BlkLength}$ **End If** **Decrement** N_y by 1 **Set** $H_{pm} \leftarrow H_p - 1$ **Set** $H_m \leftarrow H - 1$ **Set** $L_y \leftarrow H_p - H$ **Set** $T_y \leftarrow H_m / (L_y + 1)$ **Set** $\text{Img} \leftarrow \text{Bnd}$ // where the size of Img is the new size $W_{pm} * H_{pm}$ Check **If** L_x is a larger than zero **Then** **For all** Y **Do** {where $0 \leq Y \leq H_m$ } **Set** $S_t \leftarrow 0$ **Set** $X_x \leftarrow -1$ **For all** I_x **Do** {where $1 \leq I_x \leq L_x$ } **Set** $E_d \leftarrow T_x * L_x$ **For all** X **Do** {where $S_t \leq X \leq E_d$ } **Increment** X_x by 1 **Set** $A(X_x) \leftarrow \text{Img}(X, Y)$ **End For**

```

        Increment Xx by 1
        Set A(Xx) ← (Img(Ed, Y) + Img(Ed+1, Y))/2
        Set St ← Ed+1
    End For

    For all X Do {where St ≤ X ≤ Wm}
        Increment Xx by 1
        Set A(Xx) ← Img(X, Y)
    End For
    For all X Do {where 0 ≤ X ≤ Wpm}
        Set Img(X, Y) ← A(X)
    End For
End For
End If

Check If Ly is a larger than zero Then
    For all X Do {where 0 ≤ X ≤ Wpm}
        Set St ← 0
        Set Yy ← -1
        For all Iy Do {where 1 ≤ Iy ≤ Ly}
            Set Ed ← Ty * Ly
            For all Y Do {where St ≤ Y ≤ Ed}
                Increment Yy by 1
                Set A(Yy) ← Img(X, Y)
            End For
            Increment Yy by 1
            Set A(Yy) ← (Img(X, Ed) + Img(X, Ed+1))/2
            Set St ← Ed+1
        End For
        For all Y Do {where St ≤ Y ≤ Hm}
            Increment Yy by 1
            Set A(Yy) ← Img(X, Y)
        End For
        For all Y Do {where 0 ≤ Y ≤ Hpm}
            Set Img(X, Y) ← A(Y)
        End For
    End For
End If
Set nW ← Wp
Set nH ← Hm

Set nBnd ← Img
End If
Step2: Return(nBnd, nW, nH).

```

The above algorithm describes the resizing steps applied on one band, so this algorithm is implemented on the three bands (i.e., Y, Cb, Cr) individually, one after the other.

3.3.5 FIC Encoder

The inputs to this module are: (1) the block length (BlkLength), which is the width and height of the square block, (2) jump step (JmpStp), it is the distance between any two adjacent domain blocks, (3) minimum allowed block error (MinBErr) between any two IFS matched blocks (i.e., domain and range block) which have same moment descriptor value, (4) the allowed error (MinErr) between the matched range and domain blocks, (5) maximum allowed scale value (MaxScl), (6) the no. of bits used to encode the offset (i.e., oNoBits) and scale (i.e., sNoBits) coefficients, (7) the number of bins (NoBins), it is the number of uniform quantization bins of the blocks moment descriptor, (8) window size (WinSiz), it is used to set the search space according to the blocks moment descriptor, (9) width (nWid) and height (nHgt) of the coded resized band.

The three bands (Yc, Cb', Cr') are passed sequentially in FIC encoder in order to be encoded individually (i.e. this module will be applied three times, once for every band). Figure (3.2) illustrates the main stages of the FIC encoder. In the following subsections the involved stages of this module are described.

A. Range Pool Generation

The first step in FIC stage is the generation of range pool. In this generator the resized Y-component, and the resized-downsampled bands (Cb, Cr) are partitioned into non-overlapped blocks, and each block is considered as a range block belong to the range pool array.

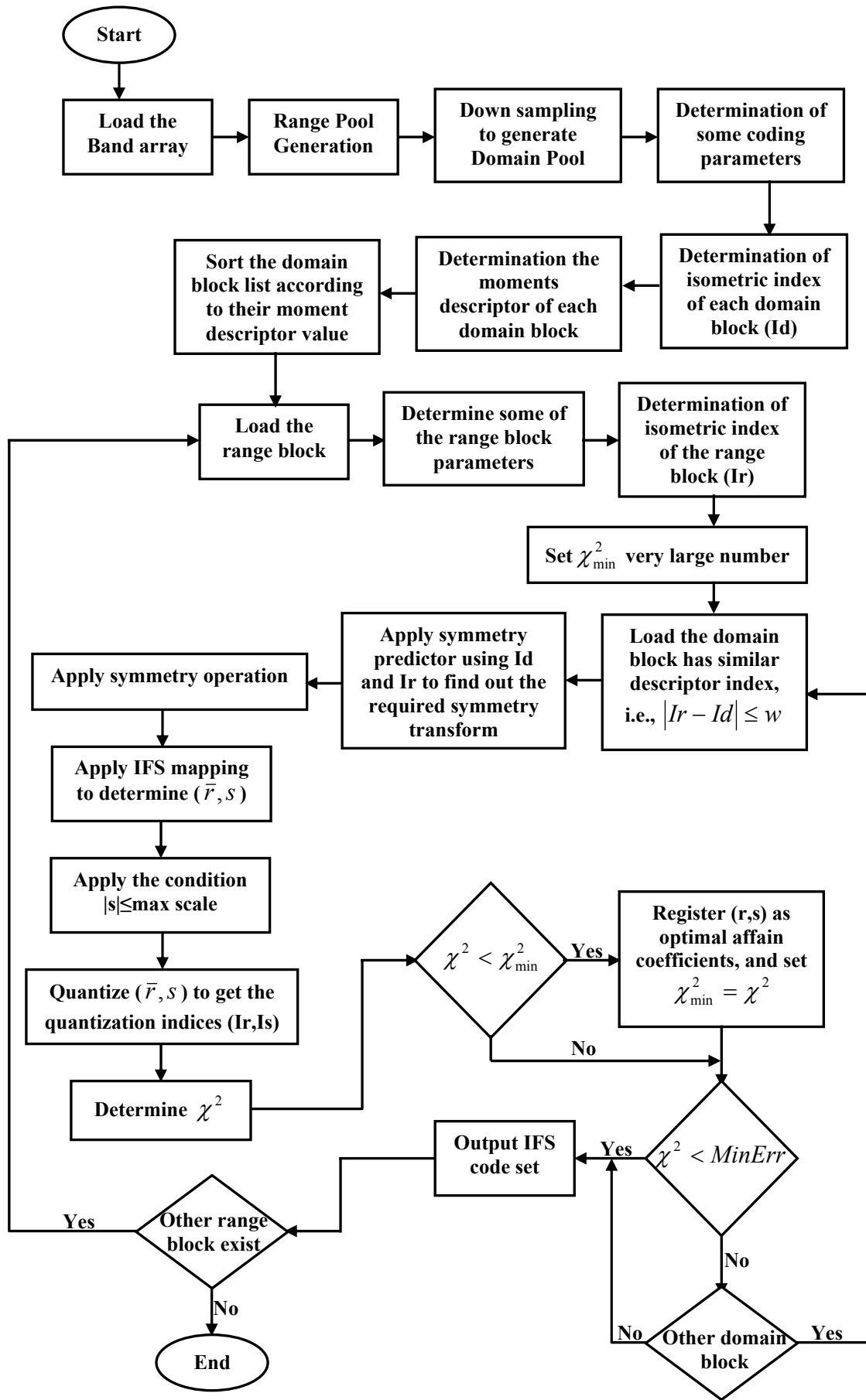


Figure (3.2) The FIC Encoder

B. Domain Pool Generation

The second step is down sampling (by 2) the components Y and the down sampled Cb and Cr. Then, the domain pool is generated by partitioning the downsampled bands into overlapped blocks using moving window method. The overlapping space depends on the jump step of the moving window.

C. Determination of Some Involved Coding Parameters

Some parameters are used in the encoding phase, such as:

1. The quantization step of scale coefficients, which is computed using equation (2.16).
2. The quantization step of offset coefficient which is computed using equation (2.17).
3. The number of range blocks in horizontal and vertical directions (N_xR and N_yR) in the range pool.
4. The number of domain blocks in the horizontal and vertical directions (N_xD , N_yD) in the domain pool.
5. The number of blocks in domain pool (NoD).

$$OffsetQuantizationStep(rQstp) = \frac{2^{OffsetNoBits} - 1}{255}, \dots\dots\dots(3.2)$$

$$ScaleQuantizationStep(sQstp) = \frac{2^{ScaleNoBits-1} - 1}{MaximumScale}, \dots\dots\dots(3.3)$$

$$N_xR = nWid \div BlockLength - 1, \dots\dots\dots(3.4)$$

$$N_yR = nHgt \div BlockLength - 1, \dots\dots\dots(3.5)$$

$$N_xD = (Wid'' - BlockLength) \div JumpStep, \dots\dots\dots(3.6)$$

$$N_yD = (Hgt'' - BlockLength) \div JumpStep, \dots\dots\dots(3.7)$$

$$NoD = N_xd * N_yD + N_xD + N_yD, \dots\dots\dots(3.8)$$

D. Blocks Isometry State Assignment

Table (3.1) shows the mapping equations of the following considered eight transforms (i.e., isometric or symmetry mappings):

1. No operations,
2. Rotation-90,
3. Rotation-180,
4. Rotation-270,
5. Reflection around Y-axis,
6. Reflection with rotation-90,
7. Reflection with rotation-180,
8. Reflection with rotation-270.

In table (3.1) the symbol c denotes the coordinates of the center point of the mapped square block (whose size is $m \times m$).

Table (3.1) The considered isometric mappings

| ID | Transform | Mapping Equations |
|----------------------|------------------------------|---|
| 0 | No operation | $x' = x$ $y' = y$ |
| 1 | Rotation_90 | $x' = (x-c)\cos(90) + (y-c)\sin(90) + c = y$ $y' = -(x-c)\sin(90) + (y-c)\cos(90) + c = 2c - x$ |
| 2 | Rotation_180 | $x' = (x-c)\cos(180) + (y-c)\sin(180) + c = 2c - x$ $y' = -(x-c)\sin(180) + (y-c)\cos(180) + c = 2c - y$ |
| 3 | Rotation_270 | $x' = (x-c)\cos(270) + (y-c)\sin(270) + c = 2c - y$ $y' = -(x-c)\sin(270) + (y-c)\cos(270) + c = x$ |
| 4 | Reflection | $x' = 2c - x$ $y' = y$ |
| 5 | Reflection with rotation 90 | $x' = (-x-c)\cos(90) + (y-c)\sin(90) + c = y$ $y' = -(-x-c)\sin(90) + (y-c)\cos(90) + c = x$ |
| 6 | Reflection with rotation 180 | $x' = (-x-c)\cos(180) + (y-c)\sin(180) + c = x$ $y' = -(-x-c)\sin(180) + (y-c)\cos(180) + c = 2c - y$ |
| 7 | Reflection with rotation 270 | $x' = (-x-c)\cos(270) + (y-c)\sin(270) + c = 2c - y$ $y' = -(-x-c)\sin(270) + (y-c)\cos(270) + c = -x$ |
| Where, $c = (m-1)/2$ | | |

For an image block $I(x,y) \{x,y | 0,1,\dots,m-1\}$, its first order centralized moments are defined as:

$$M_{10} = \sum_{y=0}^{m-1} \sum_{x=0}^{m-1} I(x,y)(x-c) \quad \dots\dots\dots(3.9)$$

$$M_{01} = \sum_{y=0}^{m-1} \sum_{x=0}^{m-1} I(x,y)(y-c) \quad \dots\dots\dots(3.10)$$

By combining both equations (3.9) and (3.10) with the equations listed in table (3.1), the relationship between the new moments values (M'_{10}, M'_{01}) of the mapped block (using isometric mappings) with its old moments values (M_{10}, M_{01}) could determined, table (3.2) lists these relationships.

Table (3.2) The relationship between moments before and after the transform

| Transform ID | Transform | Relationship |
|--------------|---------------------------|---|
| 0 | No operation | $M'_{10}=M_{10} \quad M'_{01}=M_{01}$ |
| 1 | Rotation_90 | $M'_{10}=M_{01} \quad M'_{01}=M_{10}$ |
| 2 | Rotation_180 | $M'_{10}=-M_{10} \quad M'_{01}=-M_{01}$ |
| 3 | Rotation_270 | $M'_{10}=-M_{01} \quad M'_{01}=M_{10}$ |
| 4 | Reflection | $M'_{10}=-M_{10} \quad M'_{01}=-M_{01}$ |
| 5 | Reflection + rotation_90 | $M'_{10}=M_{01} \quad M'_{01}=M_{10}$ |
| 6 | Reflection + rotation_180 | $M'_{10}=M_{10} \quad M'_{01}=-M_{01}$ |
| 7 | Reflection + rotation_270 | $M'_{10}=-M_{01} \quad M'_{01}=-M_{10}$ |

In this section, a new method for block classification according to its isometric state is described; the classification is based on applying three Boolean criteria, they depends on the status of its first order moments (i.e., M_{10}, M_{01}). These used criteria are:

1. Is $|M_{10}| \geq |M_{01}|$ or not ?
2. Is $|M_{10}| \geq 0$ or not ?
3. Is $|M_{01}| \geq 0$ or not ?

The use of these three Boolean criteria on any block leads to eight block states, as shown in table (3.3).

Table (3.3) The truth table for the eight blocks states

| Block's Class Index | Boolean Criteria | | |
|---------------------------|--------------------------|-------------------|-------------------|
| | $ M_{10} \geq M_{01} $ | $ M_{10} \geq 0$ | $ M_{01} \geq 0$ |
| 0 | T | T | T |
| 1 | T | T | F |
| 2 | T | F | T |
| 3 | T | F | F |
| 4 | F | T | T |
| 5 | F | T | F |
| 6 | F | F | T |
| 7 | F | F | F |

Now, if the relationship between the new and old moment values is taken into consideration when the block is mapped by one of the considered isometric mapping (see table 3.2), then the relationship between the indexes of block could be established (see table 3.4).

Table (3.4) The Block's state indexes before and after isometric mappings

| Old Class Index (Nop) | New Class Index | | | | | | |
|-----------------------|-----------------|--------|--------|--------|--------|--------|--------|
| | R90 | R180 | R270 | M | M+R90 | M+R180 | M+R270 |
| 0(TTT) | 6(FTF) | 3(TFF) | 5(FFT) | 2(TFT) | 4(FTT) | 1(TTF) | 7(FFF) |
| 1(TTF) | 4(FTT) | 2(TFT) | 7(FFF) | 3(TFF) | 6(FTF) | 0(TTT) | 5(FFT) |
| 2(TFT) | 7(FFF) | 1(TTF) | 4(FTT) | 0(TTT) | 5(FFT) | 3(TFF) | 6(FTF) |
| 3(TFF) | 5(FFT) | 0(TTT) | 6(FTF) | 1(TTF) | 7(FFF) | 2(TFT) | 4(FTT) |
| 4(FTT) | 1(TTF) | 7(FFF) | 2(TFT) | 5(FFT) | 0(TTT) | 6(FTF) | 3(TFF) |
| 5(FFT) | 3(TFF) | 6(FTF) | 0(TTT) | 4(FTT) | 2(TFT) | 7(FFF) | 1(TTF) |
| 6(FTF) | 0(TTT) | 5(FFT) | 3(TFF) | 7(FFF) | 1(TTF) | 4(FTT) | 2(TFT) |
| 7(FFF) | 2(TFT) | 4(FTT) | 1(TTF) | 6(FTF) | 3(TFF) | 5(FFT) | 0(TTT) |

R90≡ Rotation_90; R180≡ Rotation_180; R270≡ Rotation_270;
M≡ Mirror or Reflection;

The arrangement of the contents of table (3.4) could be inverted such that the type of transform needed to map the block from certain isometric state to other state is assigned, see table (3.5).

Table (3.5) The required isometric operation to convert the block state

| | | New Block State Index | | | | | | | |
|-----------------------|---|-----------------------|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Old Block State Index | 0 | 0 | 6 | 4 | 2 | 5 | 3 | 1 | 7 |
| | 1 | 6 | 0 | 2 | 4 | 1 | 7 | 5 | 3 |
| | 2 | 4 | 2 | 0 | 6 | 3 | 5 | 7 | 1 |
| | 3 | 2 | 4 | 6 | 0 | 7 | 1 | 3 | 5 |
| | 4 | 5 | 1 | 3 | 7 | 0 | 4 | 6 | 2 |
| | 5 | 3 | 7 | 5 | 1 | 4 | 0 | 2 | 6 |
| | 6 | 1 | 5 | 7 | 3 | 6 | 2 | 0 | 4 |
| | 7 | 7 | 3 | 1 | 5 | 2 | 6 | 4 | 0 |

0≡ No Operation; 1≡ Rotation_90;
2≡ Rotation_180; 3≡ Rotation_270;
4≡ Reflection; 5≡ Reflection+Rotation_90;
6≡ Reflection+Rotation_180;
7≡ Reflection+Rotation_270;

Algorithm (3.5) lists the steps taken to find the index of block symmetry state, it uses the determined two moments (M_x , M_y) of a block to find its moment index.

Algorithm (3.5) Get Block Index

Goal: Using block moments to get the block index

Input:

M_x // Moment which its $q=0$

M_y // Moment which its $p=0$

Output:

Index // Block Index

Step1: Check the moment value

If $|M_x| > |M_y|$ **Then**

 Check **if** $M_x > 0$ **Then**

 Check **if** $M_y \geq 0$ **Then**

Set Index $\leftarrow 0$

Else

Set Index $\leftarrow 6$

Else

 Check **If** $M_y \geq 0$ **Then**

Set Index $\leftarrow 4$

Else

Set Index $\leftarrow 2$

End If

Else

 Check **If** $M_x \geq 0$ **Then**

 Check **If** $M_y \geq 0$ **Then**

Set Index $\leftarrow 7$

Else

Set Index $\leftarrow 1$

Else

 Check **If** $M_y \geq 0$ **Then**

Set Index $\leftarrow 3$

Else

Set Index $\leftarrow 5$

End If

End If

Step2: **Return**(Index)

E. Blocks Classification Using Moments-Based Descriptor

Beside to using moments to index the isometric (symmetry) state of each range and domain block, and these isometric state indices will be used to assess the type of symmetry operation required to make the matched domain block in its best isometric state before trying to determine its best IFS- mapping coefficients. This step should reduce the elapsed encoding time around 7-8 times.

In order to gain more decrease in encoding time an additional descriptor, based also on the centralized moments, had been used. This descriptor is used to classify the domain and range block into classes, and each class is given an index. So when trying to find out the domain block that shows best IFS-match with the tested range block, then only those domain blocks belong to classes have similar index number to the class index of the range block will subjected to IFS-matching test.

In this research work the first-order centralized moments (M10, M01) or the third-order centralized moments (M30, M03) have been used to determine the moment descriptor, using equations (2.29), (2.30) and (2.34). The moment descriptor (MomIdx) values have been determined for all domain and range blocks.

For the purpose of reducing the computational redundancy the moment descriptors of the domain blocks are precomputed and registered in memory beside to other parameters and terms, which have been predetermined and saved in an array of records. This array includes the moment descriptors (i.e., MomIdx) in addition to the following parameters: (1) the position of the domain block (i.e., Xd, Yd), (2) the average (AvgD) of the domain blocks (using equation 2.31), (3) the variance (mVarD) of the domain blocks (using equation 2.25), and (4) the symmetry (or

isometric) state indexes of the domain blocks, which are determined using algorithm (3.5).

F. Sorting of Domain Blocks

The above mentioned array of records (i.e., DomIdx) has to be sorted in ascending order according to the value of moment descriptor (i.e., MomIdx), the steps of the applied sorting algorithm are described in algorithm (3.6).

| |
|--|
| <p>Algorithm (3.6) Sorting Algorithm</p> |
| <p>Goal: Sort the array DomIdx in ascending order according to MomIdx</p> |
| <p>Input:</p> <p>DomIdx()// unsorted array of record of 6 cells NoD// the number of domain blocks NoBin// the number of bins of the MomIdx values</p> |
| <p>Output:</p> <p>DomIdx()// sorted array of record of 6 cells</p> |
| <p>Step1: Sort the array</p> <p>Set $M \leftarrow 0$ Set $K \leftarrow 0$ For all I Do {where $0 \leq I \leq (\text{NoBin}-1)$} For all J Do {where $K \leq J \leq \text{NoD}$} Check If DomIdx(J).MomIdx=I Then Check If not(J=M) Then Swap(DomIdx(J),DomIdx(M)) Decrement J by 1 End If Increment M by 1 End If End For Set $K \leftarrow M$ End For</p> |
| <p>Step2: Return(DomIdx)</p> |

To more simplify the search task in the sorted array of records an array of pointers is used to point out to the boundaries (i.e., start index and

end index) of each set of sequential records have same moment descriptor values. The involved steps of this stage are depicted in algorithm (3.7).

| Algorithm (3.7) Finding Limits |
|--|
| <p>Goal: Find the limits of each moment class</p> <p>Input: <i>DomIdx()</i>// sorted array of record of 6 cells <i>NoD</i>// the number of domain blocks <i>NoBin</i>// the number of bins of the <i>MomIdx</i> values</p> <p>Output: <i>St</i>(<i>NoBin</i>)// array of start limits of each moment index <i>Ed</i>(<i>NoBin</i>)// array of end limits of each moment index</p> |
| <p>Step1: Find the limits</p> <pre> Set K←0 For all I Do {where 0≤I≤NoBin} Check If DomIdx(K).MomIdx>I Then Set St(I)←-1 Set Ed(I)←-2 Else Set St(I)←K Set Ed(I)←-1 For all L Do {where (K+1)≤L≤NoD} Check If DomIdx(L).MomIdx>I Then Set Ed(I)←L-1 Set K←L Set L←NoD// End the loop of L End If End For Check If Ed(I)=-1 Then Set Ed(I)←NoD For all J Do {where (I+!)≤J≤NoBin} Set St(J)←-1 Set Ed(J)←-2 End For Set I←NoBin// Ending the loop of I End If End If End For </pre> <p>Step2: Return(<i>St</i>, <i>Ed</i>).</p> |

Now the array of records of the domain blocks is ready to be searched to find the most available similar domain block for each tested range block.

G. Range Blocks Coding

As a first step in this stage, some of the range blocks parameters must be precomputed. These parameters are:

1. The average (AvgR) of the range block which is computed by equation (2.21).
2. The mean variance (mVarR) of the range block (equation 2.26).
3. The first moments (M10, M01), or equivalently, the third order moments (M30, M03), of the range block (equation 2.29), and then using these moments to determine moment descriptor, (i.e., moments ratio factor) (MomIdxR) for each range block (using equations 2.34).
4. The isometric state index of the range block, the value of this parameter depends on two moments values (as illustrated in algorithm 3.4).
5. The offset quantization index, which is determined using the quantization step value (rQstp).

As a next step the blocks of domain pool are searched to find out the domain block that can best matches the range block using IFS-mapping. This searching process should be repeated as long as there is still range block, in range pool need to be coded. Since, there is large number of range blocks and domain blocks, repeating the exhaustive search within domain pool causes a huge number of blocks matchings, and in such case the computational complexity of the encoding process become too high. To handle this problem, the moments based descriptor is used as classifier index, such that only the domain blocks that have similar descriptor values to that of range block are imposed to IFS-mapping tests. The implemented steps to handle the range block coding stage are illustrated in algorithm (3.8).

Algorithm (3.8) Search within Domain Blocks**Goal:** Find the nearest similar domain block for each range block**Input:**

ImpStp// number of pixels to jump between two domain blocks
NxR// the number of blocks in the width of the range pool
NyR// the number of blocks in the height of the range pool
WinSiz// the interval of the domain blocks classes
NoBin// the number of bins of the *MomIdx* values
St()// array of start limits of each moment index
Ed()// array of end limits of each moment index
DomIdx()// array of record of 6 cells
Tbl()// array of symmetry operation to convert the block's moment status using table (3.5)
Y" or *Cb"* or *Cr"* as *Dom()*// is the domain pool
NoD// the number of domain blocks
MinChi// The minimum value of the Chi squared {where *MinChi*=0}

Output:

IFSr()// array of record of 4 cells (*Ip*, *Sym*, *Irng*, and *Isc*).

Step1: For all *Ix*, *Iy* Do (For all range block compute (*AvgR*, *mVarR*, *Mx*, *My*, *MomIdxR*, *SymR*, *Ir*) as mentioned above){where $0 \leq Ix \leq NxR$ and $0 \leq Iy \leq NyR$ }

Step2: Search the domain blocks

Set *Flag* ← 0

For all *Iwin* Do {where $0 \leq I \leq WinSiz$ }

Check If *Iwin* > 0 Then

Set *M* ← 1

Else

Set *M* ← 0

End If

For all *J* Do {where $0 \leq J \leq M$ }

Case *J*

0: Set *K* ← *MomIdxR* + *I*

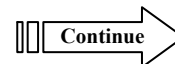
1: Set *K* ← *MomIdx* - *I*

End Case

Check If *K* value is between 0 and *NoBin* Then

Check If *St*(*K*) ≥ 0 and *Ed*(*K*) ≤ *NoBin* Then

For all *L* Do {where $St(K) \leq L \leq Ed(K)$ }



```

Set Xd←DomIdx(L).Xd
Set Yd←DomIdx(L).Yd
Check If DomIdx(L).mVarD is not 0 Then
Step3://Predict and Perform the symmetry operation
Set I←Tbl(SymR, DomIdx(L).SymIdx)
Case I
0:// Symmetry=0 (Identity)
  For all X,Y Do {where 0≤(X,Y)≤BlkLength-1}
    Set D2(X,Y)←Dom(X',Y')
    //where X', Y' is computed in table (3.1)
    when ID=0
  End For

1://Symmetry=1 (Rotation 90°)
  For all X,Y Do {where 0≤(X,Y)≤BlkLength-1}
    Set D2(X,Y)←Dom(X',Y')
    //where X', Y' is computed in table (3.1)
    when ID=1
  End For

2://Symmetry=2 (Rotation 180°)
  For all X,Y Do {where 0≤(X,Y)≤BlkLength-1}
    Set D2(X,Y)←Dom(X',Y')
    //where X', Y' is computed in table (3.1)
    when ID=2
  End For

3://Symmetry=3 (Rotation 270°)
  For all X,Y Do {where 0≤(X,Y)≤BlkLength-1}
    Set D2(X,Y)←Dom(X',Y')
    //where X', Y' is computed in table (3.1)
    when ID=3
  End For

4://Symmetry=4 (Reflection)
  For all X,Y Do {where 0≤(X,Y)≤BlkLength-1}
    Set D2(X,Y)←Dom(X',Y')
    //where X', Y' is computed in table (3.1)
    when ID=4
  End For

5://Symmetry=5 (Reflection with Rotation 90°)
  For all X,Y Do {where 0≤(X,Y)≤BlkLength-1}
    Set D2(X,Y)←Dom(X',Y')
    //where X', Y' is computed in table (3.1)
    when ID=5
  End For

6://Symmetry=6 (Reflection with Rotation 180°)
  For all X,Y Do {where 0≤(X,Y)≤BlkLength-1}
    Set D2(X,Y)←Dom(X',Y')
    //where X', Y' is computed in table (3.1)
    when ID=6
  End For

```



```

7: //Symmetry=7 (Reflection with Rotation 270°)
   For all X,Y Do {where  $0 \leq (X,Y) \leq \text{BlkLength}-1$ }
       Set  $D2(X,Y) \leftarrow \text{Dom}(X',Y')$ 
       //where  $X', Y'$  is computed in table (3.1)
       when  $ID=7$ 
   End For
End Case

Step4:// Determine Scale Coefficients and Bound the value, finally Quantize
//Determine the Scale coefficient (Scl) using
equation(2.23)

Check If  $Scl > \text{MaxScl}$  Then
    Set  $Scl \leftarrow \text{MaxScl}$ 
Else If  $Scl < -\text{MaxScl}$  Then
    Set  $Scl \leftarrow -\text{MaxScl}$ 
End If

//Then quantize this Scl coefficient using  $sQstp$  to be
Isc coefficient (i.e. Isc is the quantized scale)

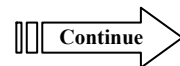
Step5://Determine the Chi Square coefficient using equation(2.24)

Step6:// Compare Chi and Register the Optimal Chi-Case
Check If  $\text{Flag}=0$  Or  $\text{Flag}=1$  And  $\text{Chi} < \text{MinChi}$  Then
    Set  $\text{MinChi} \leftarrow \text{Chi}$ 
    Set  $\text{OptSym} \leftarrow I$ 
    Set  $\text{OptIsc} \leftarrow \text{Isc}$ 
    Set  $\text{OptXd} \leftarrow Xd$ 
    Set  $\text{OptYd} \leftarrow Yd$ 
    Check If ( $\text{MinChi} < \text{MinBErr}$  And  $\text{Iwin}=0$ ) Or
        ( $\text{MinChi} < \text{MinErr}$  And  $\text{Iwin} > 0$ ) Then
        //End Loops L, J, Iwin
    End If
End If
Set  $\text{Flag} \leftarrow 1$ 

Else
Step7://Handle the Case  $\text{Dom}(L).m\text{VarD}=0$  for all domain values
    Set  $\text{Isc} \leftarrow 0$ 
    Set  $\text{Scl} \leftarrow 0$ 
    Set  $\text{Chi} \leftarrow m\text{VarR}$ 
    //Compare Chi and Register the Optimal Chi-Case as
    in (Step6)
End If

End For
End If

```



```

        End If
    End For
End For

Step8://Compare the Optimal quantized scale coefficient
    Check If OptIsc>0 Then
        Set OptIsc←OptIsc+OptIsc
    ElseIf OptIsc<0 Then
        Set OptIsc←-OptIsc-OptIsc-1
    End If

Step9://Register the optimal IFS coefficients and save them in IFSr()
    Set OptX←OptX / JmpStp
    Set OptY←OptY / JmpStp
    Set IFSr(Ix,Iy).Ip←(Nx*D+1)*OptY+OptX //The optimal position of the
        most similar domain block
    Set IFSr(Ix,Iy).Irng←Ir //The optimal quantized offset coefficient of this
        domain block
    Set IFSr(Ix,Iy).Isc←OptIsc //The optimal quantized scale coefficient of this
        domain block
    Set IFSr(Ix,Iy).sym←OptSym //The optimal symmetry index of this domain
        block
End For

Step10: Return(IFSr)

```

The steps of the above listed algorithm requires less computation time in comparison with traditional algorithm because there is a reduction in both the number of tested domain blocks for each range block, and in the number of isometric mappings trials (i.e., instead of the 8 trials only one isometric mapping case is tested). Also, the use of sorted array of records of the domain block with pointers refer to the boundaries of each domain class will be useful to reduce the search space with the domain pool.

In this project the applied similarity condition the coded range block and any domain block listed in the domain pool is the following:

$$|F_d - F_r| < w \text{ then the range and domain block are similar}$$

Otherwise they are dissimilar

In the above condition, the symbols F_d and F_r denote the index of the moment descriptor of domain and range blocks, respectively.

The symbol (w) denotes the permissible similarity margin (called window size) between the moment descriptor indexes of the two matched block.

The main steps of any IFS matching instance between any pair of domain and range blocks are the following steps:

1. Computing the scale coefficient value using equation (2.23).
2. Applying the bounding condition on the determined scale coefficient, i.e.,

$$\begin{aligned} \text{if } s < -\max \text{Scale} \text{ then } s &= -\max \text{Scale} \\ \text{else if } s > \max \text{Scale} \text{ then } s &= \max \text{Scale} \end{aligned}$$

3. Quantize the determined values of scale and offset coefficients using the following equations:

$$S_I = \text{round}\left(\frac{s}{sQstp}\right), \dots \dots \dots (3.11)$$

$$S_q = S_I * sQstp, \dots \dots \dots (3.12)$$

$$O_I = \text{round}\left(\frac{\bar{r}}{rQstp}\right), \dots \dots \dots (3.13)$$

$$O_q = O_I * rQstp, \dots \dots \dots (3.14)$$

Where: $sQstp$, $rQstp$ are the quantization steps of scale and offset coefficients, respectively (see equations 3.2 and 3.3).

S_I , S_q are the quantization index and the quantized value, respectively, of the scale coefficient.

O_I , O_q are the quantization index and the quantized value, respectively, of the offset coefficient.

4. Determining the mean square error (χ^2) between the actual values of the range block elements are the corresponding approximate values due to IFS-mapping of the domain block.

5. Comparing the value of error (χ^2) with lowest value registered error (χ_{\min}^2) attained through the previous matching trials between the range block and other pre-tested domain blocks.

3.3.6 Encoding the IFS Code

To increase the attained compression ratio, the determined IFS coefficients in the system are coded using both DPCM and shift coding, and the output (i.e., codewords) of the shift encoder are saved in the compressed file, to be restored later for decoding purpose.

At first the file must be prepared to save the shift encoder codewords in it. Some overhead information (i.e., width and height of the image, the block length, the jump step, the number of offset bits, the number of scale bits, the maximum scale value) must be saved at the beginning of the file, these parameters are considered as a part of the header of the compressed file, and their registration in the file is necessary for decoding operations.

After the header section, the binary codewords, produced by applying shift encoder on IFSr() coefficients, are saved in the file. Algorithm (3.9) shows the implemented steps to shift encoding the IFS coefficients, taking into consideration the following three remarks:

- A. The offset coefficients had been first coded using DPCM and the output of this encoder was shift coded.
- B. Before applying shift coding the scale and offset coefficients have been mapped using the following function:

$$c' = \begin{cases} 2c & \text{if } c \geq 0 \\ 2|c|-1 & \text{otherwise} \end{cases} \dots\dots\dots(3.15)$$

Where c' is the original value of scale index or of the DPCM output of the offset coefficient. The value of c' is always positive, which is a necessary condition to conduct shift encoding.

- C. Before applying shift encoding the proper size of its codewords should be determined, so in algorithm (3.9) a simple optimization technique is implemented, it is based on testing all possible codeword sizes to find out the best size that lead to lowest consumption in bits (i.e., lowest output size).

Algorithm (3.9) Encode IFS Code

Goal: Encode The IFSr() components

Input:

NxR // number of blocks in the width of the range pool
 NyR // number of blocks in the height of the range pool
 $IFSr()$ // array of record of 4 cells (I_p , Sym , I_{rng} , I_{sc})

Output:

Compressed File

Step1: Get the component from IFSr() array

Set $N \leftarrow NxR * NyR + NxR + NyR$

Set $I \leftarrow -1$

For all I_y Do {where $0 \leq I_y \leq NyR$ }

For all I_x Do {where $0 \leq I_x \leq NxR$ }

Increment I by 1

Set $Z(I) \leftarrow IFSr(I_x, I_y).component$

End For

Check If $I_y < NyR$ Then

Increment I_y by 1

For all I_x Do {where $NxR \geq I_x \geq 0$ }

Increment I by 1

Set $Z(I) \leftarrow IFSr(I_x, I_y).component$

End For

End If

End For

Step2: DPCM Encoding the IFS component

For all I Do {where $N \geq I \geq 1$ }

Set $Z_z(I) \leftarrow Z(I) - Z(I-1)$

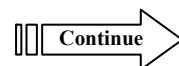
End For

Step3: Mapping to Positive values the component

Set $MaxI \leftarrow 0$

For all I Do {where $1 \leq I \leq N$ }

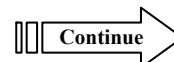
Check If $Z_z(I) > 0$ Then



```

        Set  $Z_z(I) \leftarrow Z_z(I) + Z_z(I)$ 
    Else Check If  $Z_z(I) < 0$  Then
        Set  $Z_z(I) \leftarrow -Z_z(I) - Z_z(I) - 1$ 
    End If
    Check If  $Max1 < Z_z(I)$  Then
        Set  $Max1 \leftarrow Z_z(I)$ 
    End If
End For
Step4: Shift-Code Optimizer for the component
For all I Do 1 To N
    Set  $J \leftarrow Z_z(I)$ 
    Set  $His(J) \leftarrow His(J) + 1$  // histogram of each component
End For
Set  $Tot1 \leftarrow N + 1$ 
Set  $oSm \leftarrow rNoBit * Tot1$ 
Set  $I \leftarrow 1$ 
Set  $Bt2 \leftarrow 1$ 
Step5: Check If  $I < Max1$  Then
    Set  $I \leftarrow I * 2 + 1$ 
    Set  $Bt2 \leftarrow Bt2 + 1$ 
    Goto Step 5
End If
Set  $Bt1 \leftarrow Bt2 - 4$ 
Check If  $Bt1 < 1$  Then
    Set  $Bt1 \leftarrow 1$ 
End If
For all Bt Do {where  $Bt1 \leq Bt \leq Bt2$ }
    Set  $Rg \leftarrow 2^{Bt} - 1$ 
    Set  $Max2 \leftarrow Max1 - Rg$ 
    Set  $I \leftarrow 1$ 
    Set  $Btt \leftarrow 1$ 
Step6: Check If  $I < Max2$  Then
    Set  $I \leftarrow I * 2 + 1$ 
    Set  $Btt \leftarrow Btt + 1$ 
    Goto Step6
End If
Set  $Sm \leftarrow 0$ 
For all I Do {where  $Rg \leq I \leq Max1$ }
    Set  $Sm \leftarrow Sm + His(I)$ 
End For
Set  $Tot \leftarrow Tot1 * Bt + Sm * Btt$ 
Check If  $Bt = Bt1$  Or ( $Bt > Bt1$  And  $Tot < OptTot$ ) Then
    Set  $OptTot \leftarrow Tot$ 
    Set  $OptBt1 \leftarrow Bt$ 
    Set  $OptBt2 \leftarrow Btt$ 
End If
End For

```



Step7: Encoding the component using the shift encoding

Check If (OptTot + 8) < oSm *Then*

PutBit 1 //on the compressed file

PutWord OptBt1, 4 //on the compressed file

PutWord OptBt2, 4

Set Rg = $2^{\text{OptBt1}} - 1$

PutWord Z(0), NoBit

For all I Do {where $1 \leq I \leq N$ }

Check If Zz(I) < Rg *Then*

PutWord Zz(I), OptBt1

Else

PutWord Rg, OptBt1

PutWord Zz(I) - Rg, OptBt2

End If

End For

Else

Step8: Fix Length Encoding the component Indices

PutBit 0

For all I Do {where $0 \leq I \leq N$ }

PutWord Z(I), NoBit

End For

End If

Step9:End.

3.4 Decoder Module

Figure (3.1b) illustrates the main stages of the decoding module, it is obvious that the sequence of its stages takes the inverse order of the encoding module sequence. Also, the functionality of decoding module would mainly be the reverse of the functionality of the corresponding stage in encoder module.

In the following subsections the main stage of the decoder module are described.

3.4.1 Load and Decode the IFS Code

These two stages are the first two stages in decoding module. The decoding process begins with loading the data of the compressed file (ComFileName). At first, the contents of header section must be extracted

because they are necessary to setup some parameters of the decoding modules, and to make the decoder capable to load the registered codewords in the compression file. As a second stage the shift decoding and DPCM are implemented to decode the scale and offset coefficients. Also, in this stage the inverse mapping process (from positive to negative-positive) is applied, using the following equation:

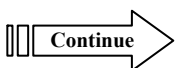
$$c = \begin{cases} c' \text{ div } 2 & \text{if } c' \text{ is even} \\ -((c' + 1) \text{ div } 2) & \text{if } c' \text{ is odd} \end{cases} \dots\dots\dots(3.16)$$

This process will retrieve the decoded offset and scale coefficients to their original values. The output from this stage is an array, nIFSr(), of IFS-coefficients.

3.4.2 FIC Decoder

In this stage, the steps of decoding the IFS code are implemented to establish the compressed image. This stage consist of two main processes:

1. Dequantization of IFS coefficients.
2. IFS mapping to reconstruct YCbCr bands.

| |
|--|
| Algorithm (3.10) Loading and Shift Decoding |
| <p>Goal: Decode The IFS code</p> <p>Input: Compressed File</p> <p>Output: NxR// number of blocks in the width of the range pool NyR// number of blocks in the height of the range pool IFSr()// array of record of 4 cells (Ip, Sym, Irng, Isc)</p> |
| <p>Step1:// Get all parameters from the compressed file</p> <p>Step2:// Decode the component of IFS code Set $N \leftarrow NxR * NyR + NxR + NyR$ Check If GetBit=1 Then</p> <p>Step3:// Shift decode the component Set $Bt1 \leftarrow GetWord(4)$ Set $Bt2 \leftarrow GetWord(4)$ Set $Rg \leftarrow 2^{Bt1} - 1$</p> |
|  |

```

Set Z(0)←GetWord(NoBit)
For all I Do {where 1≤I≤N}
  Set Z(I)←GetWord(Bt1)
  Check If Z(I) = Rg Then
    Set Z(I)←Rg + GetWord(Bt2)
  End If
End For
Step4:// Mapping to Negative/Positive the values of the component
For all I Do {where 1≤I≤N}
  Check If not(Z(I) = 0) Then
    Check If (Z(I) And 1) = 0 Then
      Set Z(I)←Z(I) \ 2
    Else
      Set Z(I)← -(Z(I) + 1) \ 2
    End If
  End If
End For
Step5:// DPCM Decoding of the component
For all I Do {where 1≤I≤N}
  Set Z(I)←Z(I) + Z(I - 1)
End For

Else
Step6:// Fix Length Decoding of the IFS component
For all I Do {where 0≤I≤N}
  Set Z(I)←GetWord(NoBit)
End For
End If

Step7:// Put the Component Indices in IFSr Array
Set I←-1
For all Iy Do {where 0≤Iy≤NyR}
  For all Ix Do {where 0≤Ix≤NxR}
    Increment I by 1
    Set IFSr(Ix, Iy).component←Z(I)
  End For
  Check If Iy < NyR Then
    Increment Iy by 1
    For all Ix Do {where NxR≥Ix≥0}
      Increment I by 1
      Set IFSr(Ix, Iy).component←Z(I)
    End For
  End If
End For
Step8:// Return (IFSr).

```

A. Dequantization

After shift decoding and inverse mapping (from positive to positive/negative) to retrieve the IFS() coefficients, the values of scale and offset coefficients need to be dequantized because the stored values of both coefficients are their quantization indices (i.e., s_1 , o_1) and not their quantized values (i.e., s_q , o_q). Algorithm (3.11) shows the steps taken to dequantize the IFS() coefficients.

Algorithm (3.11) Dequantization

Goal: Dequantize the IFSr() components

Input:

JmpStp// the jump step between each two domain blocks

Nxd// the width of the domain pool

nIFSr()//array of record of 4 cells (*Ip*, *Sym*, *Irng*, and *Isc*).

Output:

oIFSr()// dequantized array of record of 5 cells (*Xd*, *Yd*, *Sym*, *Irng*, and *Isc*).

Step1: Dequantize all the components

For all *Ix, Iy* **Do** {where $0 \leq Ix \leq NxR$ and $0 \leq Iy \leq NyR$ }

Set *oIFSr(Ix, Iy).Xd* ← (*nIFSr(Ix, Iy).Ip* mod (*Nxd*+1)) * *JmpStp*

Set *oIFSr(Ix, Iy).Yd* ← (*nIFSr(Ix, Iy).Ip* div (*Nxd*+1)) * *JmpStp*

Set *oIFSr(Ix, Iy).Sym* ← *nIFSr(Ix, Iy).Sym*

Set *oIFSr(Ix, Iy).Irng* ← *nIFSr(Ix, Iy).Irng* * *rQstp*

Set *oIFSr(Ix, Iy).Isc* ← *nIFSr(Ix, Iy).Isc* * *sQstp*

End For

Step2: Return(*oIFSr*)

B. Reconstruction of Range Pool

This stage is initialized by generating a domain pool whose elements values are assigned in arbitrary way. In the established system the elements of the domain pool have given same value (i.e., zero value). The dequantized values of IFS() coefficients are used to map the domain blocks to produce the range blocks approximates. Then, each generated range block is set in its position in the empty range pool.

After the complete generation of all range blocks of the range pool, then the contents of this pool are down sampled (by 2), using averaging method, to regenerate the domain pool. This newly generated domain pool is used again with the IFS() code set to regenerate the range pool. This sequence (i.e., domain pool generation, IFS mapping, and range pool generation) is repeated till the reconstructed range pool reaches the attractor state. The range pool reconstruction is performed three time to reconstruct the three color bands (i.e., Y-component, downsampled Cb and Cr components).

3.4.3 Range Pool Resizing

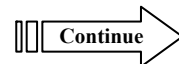
In this stage the sizes of the three reconstructed range pools are adjusted to be equal to their original size. As mentioned in paragraph (3.2.5.2), the resizing step is done to make the range pool dimensions (i.e., width and height) multiples of range block length. The rows and columns decimation method was used to adjust the size of range pools to its original size. Algorithm (3.12) lists the implemented steps to adjust the size of one range pool.

Algorithm (3.12) Resize the Range Pool to Its Original Size**Goal:** Resize the range pool after reconstruction**Input:**

W// the width of the range pool
H// the height of the range pool
BlkLength// the block length
Rng()// the range pool

Output:

nRng()// the new range pool

Step1: Check if the height and width value are accepted to division by the block length without any restCheck **If** ($W \bmod \text{BlkLength}$) is not zero **Or** ($H \bmod \text{BlkLength}$) is not zero**Then**Set $N_x \leftarrow W \text{ div } \text{BlkLength}$ Set $W_p \leftarrow N_x * \text{BlkLength}$ Check **If** $W_p < W$ **Then**Increment N_x by 1Set $W_p \leftarrow W_p + \text{BlkLength}$ **End If**Decrement N_x by 1Set $W_{pm} \leftarrow W_p - 1$ Set $W_m \leftarrow W - 1$ Set $L_x \leftarrow W_p - W$ Set $T_x \leftarrow W_m \text{ div } (L_x + 1)$ **End If**Set $N_y \leftarrow H \text{ div } \text{BlkLength}$ Set $H_p \leftarrow N_y * \text{BlkLength}$ Check **If** $H_p < H$ **Then**Increment N_y by 1Set $H_p \leftarrow H_p + \text{BlkLength}$ **End If**Decrement N_y by 1Set $H_{pm} \leftarrow H_p - 1$ Set $H_m \leftarrow H - 1$ Set $L_y \leftarrow H_p - H$ Set $T_y \leftarrow H_m \text{ div } (L_y + 1)$ Set $\text{Img} \leftarrow \text{Rng}$ // Img is a temporary array of dimension $W_{pm} * H_{pm}$ Check **If** L_x is larger than zero **Then****For all** Y **Do** {where $0 \leq Y \leq H_{pm}$ }Set $S_t \leftarrow 0$ Set $X_1 \leftarrow -1$ Set $X_2 \leftarrow -1$ 


```

For all  $I_x$  Do {where  $1 \leq I_x \leq L_x$ }
  Set  $Ed \leftarrow Tx * I_x$ 
  For all  $X$  Do {where  $St \leq X \leq Ed$ }
    Increment  $X1$  by 1
    Increment  $X2$  by 1
    Set  $A(X2) \leftarrow \text{Img}(X1, Y)$ 
  End For
  Increment  $X1$  by 1
  Set  $St \leftarrow Ed + 1$ 
End For
For all  $X$  Do {where  $St \leq X \leq Wm$ }
  Increment  $X1$  by 1
  Increment  $X2$  by 1
  Set  $A(X2) \leftarrow \text{Img}(X1, Y)$ 
End For
For all  $X$  Do {where  $0 \leq X \leq Wm$ }
  Set  $\text{Img}(X, Y) \leftarrow A(X)$ 
End For
End For
End If
Check If  $Ly$  is larger than zero Then
  For all  $X$  Do {where  $0 \leq X \leq Wm$ }
    Set  $St \leftarrow 0$ 
    Set  $Y1 \leftarrow -1$ 
    Set  $Y2 \leftarrow -1$ 
    For all  $I_y$  Do {where  $1 \leq I_y \leq Ly$ }
      Set  $Ed \leftarrow Ty * I_y$ 
      For all  $Y$  Do {where  $St \leq X \leq Ed$ }
        Increment  $Y1$  by 1
        Increment  $Y2$  by 1
        Set  $A(Y2) \leftarrow \text{Img}(X, Y1)$ 
      End For
      Increment  $Y1$  by 1
      Set  $St \leftarrow Ed + 1$ 
    End For
    For all  $Y$  Do {where  $St \leq X \leq Hm$ }
      Increment  $Y1$  by 1
      Increment  $Y2$  by 1
      Set  $A(Y2) \leftarrow \text{Img}(X, Y1)$ 
    End For
    For all  $Y$  Do {where  $0 \leq Y \leq Hm$ }
      Set  $\text{Img}(X, Y) \leftarrow A(Y)$ 
    End For
  End For
End For
End If
Set  $nRng \leftarrow \text{Img}$ 
Step2: Return ( $nRng$ )

```

3.4.4 Up Sampling

In this stage the reconstructed range pools of the chromatic bands (i.e., Cb and Cr) are up sampled (by 2) using nearest neighbor interpolation method. Algorithm (3.13) shows the steps taken to apply this stage.

Algorithm (3.13) Up-Sampling Method

Goal: Up sample the two bands Cb, and Cr

Input:

Wid// the width of the bands
Hgt// the height of the bands
sCb()// the down sampled Cb
sCr()// the down sampled Cr

Output:

rCb(), rCr()// the reconstructed Cb and Cr

Step1: Initialize some parameters

Set $W_m \leftarrow Wid - 1$
Set $W_{hm} \leftarrow (Wid + 1) \text{ div } 2 - 1$
Set $H_m \leftarrow Hgt - 1$
Set $H_{hm} \leftarrow (Hgt + 1) \text{ div } 2 - 1$

Step2: For all X,Y Do {where $0 \leq X \leq W_{hm}$ and $0 \leq Y \leq H_{hm}$ }

Set $rCb(X*2, Y*2) \leftarrow sCb(X, Y)$
Set $rCr(X*2, Y*2) \leftarrow sCr(X, Y)$

Check If $(X*2+1) < Wid$ Then

Set $rCb(X*2+1, Y*2) \leftarrow sCb(X, Y)$
Set $rCr(X*2+1, Y*2) \leftarrow sCr(X, Y)$

End If

Check If $(Y*2+1) < Wid$ Then

Set $rCb(X*2, Y*2+1) \leftarrow sCb(X, Y)$
Set $rCr(X*2, Y*2+1) \leftarrow sCr(X, Y)$

Check If $(X*2+1) < Wid$ Then

Set $rCb(X*2+1, Y*2+1) \leftarrow sCb(X, Y)$
Set $rCr(X*2+1, Y*2+1) \leftarrow sCr(X, Y)$

End If

End If

End For

Step3: Return(rCb, rCr)

3.4.5 Conversion from YCbCr to RGB

As a last decoding stage the reconstructed bands (Y, Cb, Cr) are converted to RGB color representation, using equations (2.12), Algorithm (3.14) shows the implemented steps to make the color conversion.

Algorithm (3.14) Convert from YCbCr to RGB color space

Goal: Extract RGB from the YCbCr

Input:

Yc()// the Y component

Cb()// the Cb component

Cr()// the Cr component

Output:

Red()// the red component

Grn()// the green component

Blu()// the blue component

Step1: Determine the reconstructed values and check if they are acceptable

Set $A = Y + 1.58163 * Cr + 0.00131 * Cb$

Check If $A \leq 0$ **Then**

Set $Red \leftarrow 0$

Else If $A \geq 255$ **Then**

Set $Red \leftarrow 255$

Else

Set $Red \leftarrow A$

End If

Set $A \leftarrow Y + 1.86324 * Cb - 0.00018 * Cr$

Check If $A \leq 0$ **Then**

Set $Blu \leftarrow 0$

Else If $A \geq 255$ **Then**

Set $Blu \leftarrow 255$

Else

Set $Blu \leftarrow A$

End If

Set $A \leftarrow Y - 0.18817 * Cb - 0.46978 * Cr$

Check If $A \leq 0$ **Then**

Set $Grn \leftarrow 0$

Else If $A \geq 255$ **Then**

Set $Grn \leftarrow 255$

Else

Set $Grn \leftarrow A$

End If

Step2: Return(Red, Grn, Blu).

The reconstructed red, green, and blue bands will be saved in a bitmap formatted file, as a decompressed image file, and it will have the same size as the original file. For testing purpose, the contents of the two images (i.e., original and decompressed images) are compared using some fidelity criteria (such as MAE, MSE, PSNR) in order to find how much error (i.e., difference) is introduced due to compression.



Chapter Four

Performance Test Results

Chapter Four

Performance Test Results

4.1 Introduction

In this research project four FIC schemes have been applied. The first scheme is the traditional FIC scheme, denoted as "TradFIC" and established for comparison purpose with other three enhanced FIC schemes. The second scheme, denoted as "PredFIC", it is the enhanced version of FIC scheme, where the isometric (or symmetric) predictor is added to traditional FIC-scheme to reduce the number of domain block mapping trials from 8 to only 1. The index of the selected isometric mapping is assigned by the introduced moment-based predictor. The third established FIC-scheme is an improved version of PredIFS, where beside to using moment-based symmetry predictor a block descriptor, where based on first order moments, is used to more speed-up to FIC coding stage, this third scheme is denoted as "Dis1FIC". The fourth FIC scheme uses another blocks descriptor which is based on third order moments, instead of first order moments, and this scheme is denoted as "Dis3FIC".

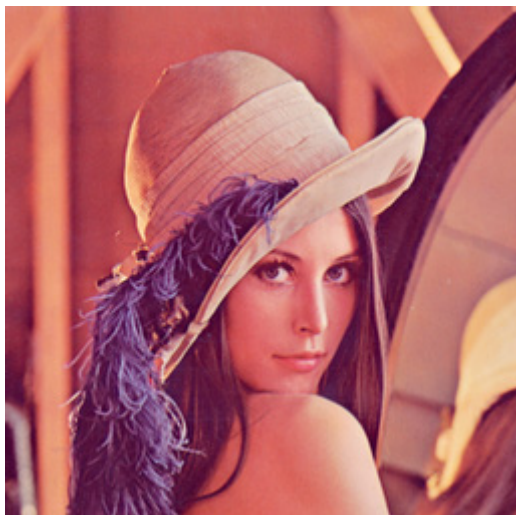
This chapter is devoted to present the results of the conducted tests to study the compression performance of the suggested fractal image compression schemes. Some of the famous fidelity measures (i.e. MSE, MAE, PSNR, CR) have been used to assess the quality of the reconstructed image.

The effects of some involved coding parameters on the performance of the four applied IFS scheme (i.e., TradFIC, PredFIC, Dis1FIC, and Dis3FIC) have been investigated.

The developed systems have been established using Visual Basic (version 6.0) programming language, and they work under Microsoft windows XP Professional operating system. The tests have been conducted using laptop computer (Processor: mobile AMD Athlon™ XP-M (LV) 2400+, MMX, 3DNow, ~1.8GHz; Memory: 480MB).

4.2 Image Test Material

Two bitmap images have been taken as test samples, each image consists of the same number of pixels (i.e., 256x256), and they have color resolution (24bpp), and size (192KB). Figure (4.1) shows these two images.



a. Lena



b. Girl

Figure (4.1) The bitmap images used as test samples

4.3 Testing Strategy

The testing operations have been applied on the above mentioned two image samples. The tests were conducted to explore the effectiveness of each involved parameter in the compression scheme on the compression performance parameters; including the three fidelity criteria (MAE, MSE, PSNR), compression ratio, bit rate, and the elapsed time of the compression process. The tested compression scheme parameters are: block length, jump step, maximum scale, scale bits, offset bits, minimum error, minimum block error, number of bins, window size.

The test procedure followed to investigate the effectiveness of each parameter is "changing the value of this parameter, while the values of other parameters are set fixed at their default values". The adopted default values are described in table (4.1).

Table (4.1) The default values of the relevant coding parameters

| Parameter | Default Value |
|-----------------------|---------------|
| Block Length | 4 |
| Jump Step | 1 |
| Maximum Scale Value | 3 |
| Number of Scale Bits | 6 |
| Number of Offset Bits | 8 |
| Minimum Error | 1.5 |
| Minimum Block Error | 1 |
| Number of Bins | 100 |
| Window Size | 1 |

4.4 Block Length Test

In this set of tests the effects of block length on compression performance parameter are investigated. The tests have been conducted on Lena and Girl images. The tests results indicated that the performance parameters are significantly affected by the value of block length. Here, in this set of tests the value of block length was varied while the values of

other involved parameters are kept fixed at their default values. The noticed effects of the block length parameter are clarified in the following remarks:

1. Table (4.2) shows the effects of the block length parameter on the compression performance parameters (MAE, MSE, PSNR, CR, BR, encoding time) when the Dis1FIC scheme is applied on Lena image.

Table (4.2) The effect of block length parameter of Dis1FIC scheme

| Block Length | Over All | | | Compression Ratio | Bit Rate | Time in Seconds |
|--------------|----------|--------|-------|-------------------|----------|-----------------|
| | MAE | MSE | PSNR | | | |
| 4 | 3.92 | 42.01 | 31.90 | 8.875 | 2.704 | 1.26 |
| 5 | 4.73 | 63.61 | 30.10 | 13.453 | 1.784 | 1.32 |
| 6 | 5.53 | 85.12 | 28.83 | 19.326 | 1.242 | 1.25 |
| 7 | 6.19 | 106.20 | 27.87 | 26.117 | 0.919 | 1.26 |
| 8 | 6.91 | 130.61 | 26.97 | 35.393 | 0.678 | 1.34 |

2. Table (4.3) illustrates the effects of the block length on the compression performance parameters, when Dis3FIC scheme is applied on Lena image.

Table (4.3) The effect of block length parameter of Dis3FIC scheme

| Block Length | Over All | | | Compression Ratio | Bit Rate | Time in Seconds |
|--------------|----------|--------|-------|-------------------|----------|-----------------|
| | MAE | MSE | PSNR | | | |
| 4 | 3.91 | 42.05 | 31.89 | 8.863 | 2.708 | 1.21 |
| 5 | 4.75 | 63.65 | 30.09 | 13.440 | 1.786 | 1.33 |
| 6 | 5.61 | 86.96 | 28.74 | 19.334 | 1.241 | 1.27 |
| 7 | 6.23 | 107.16 | 27.83 | 26.089 | 0.920 | 1.35 |
| 8 | 6.91 | 130.24 | 26.98 | 35.368 | 0.679 | 1.38 |

3. Figure (4.2) illustrates the effects of block length variation compression performance parameters of the four FIC schemes. The test image was Lena image.

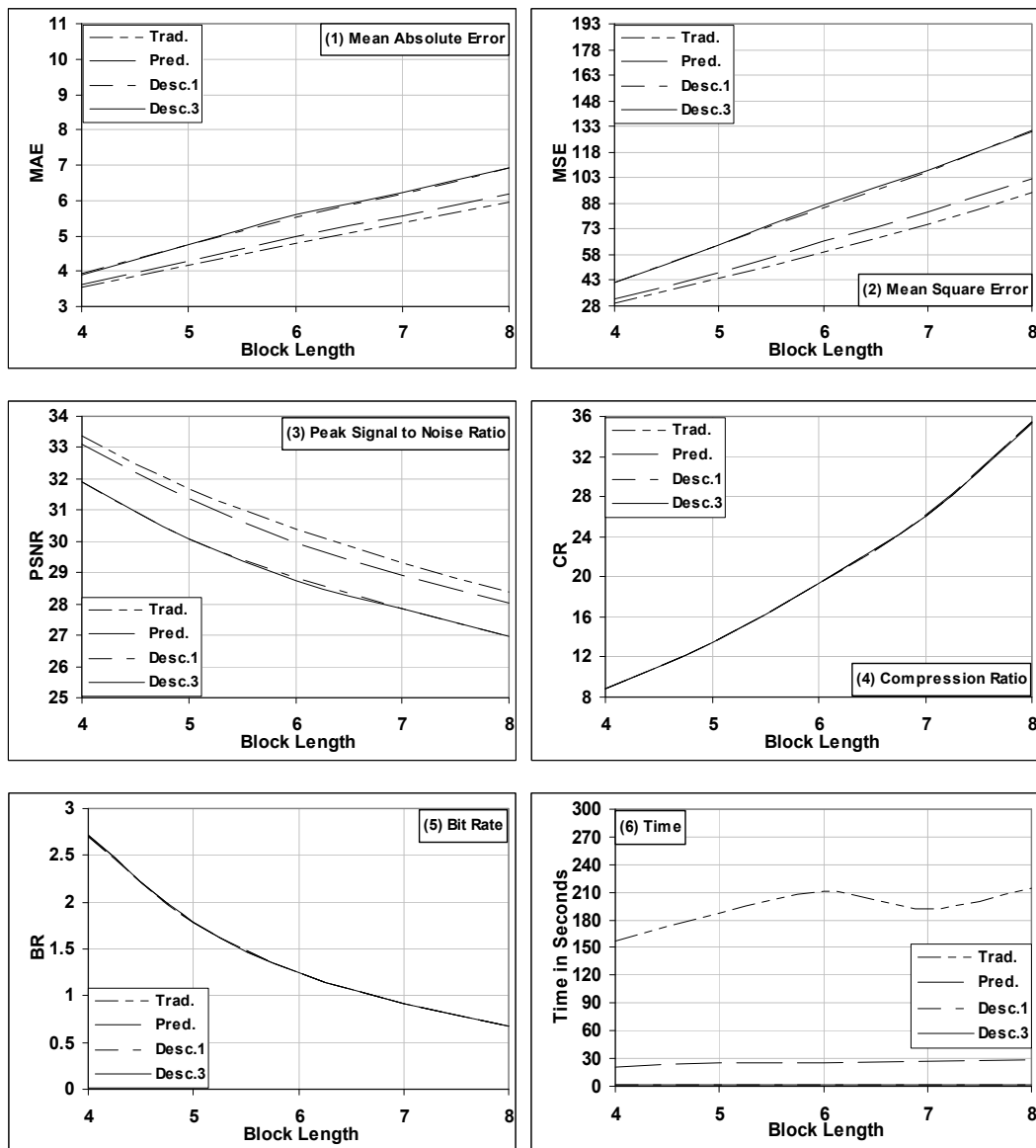


Figure (4.2) The effects of block length on the compression performance parameters, the test image was Lena

4. Figure (4.3) shows the effects of block length variation on compression performance of the four FIC schemes. The test image was Girl image.

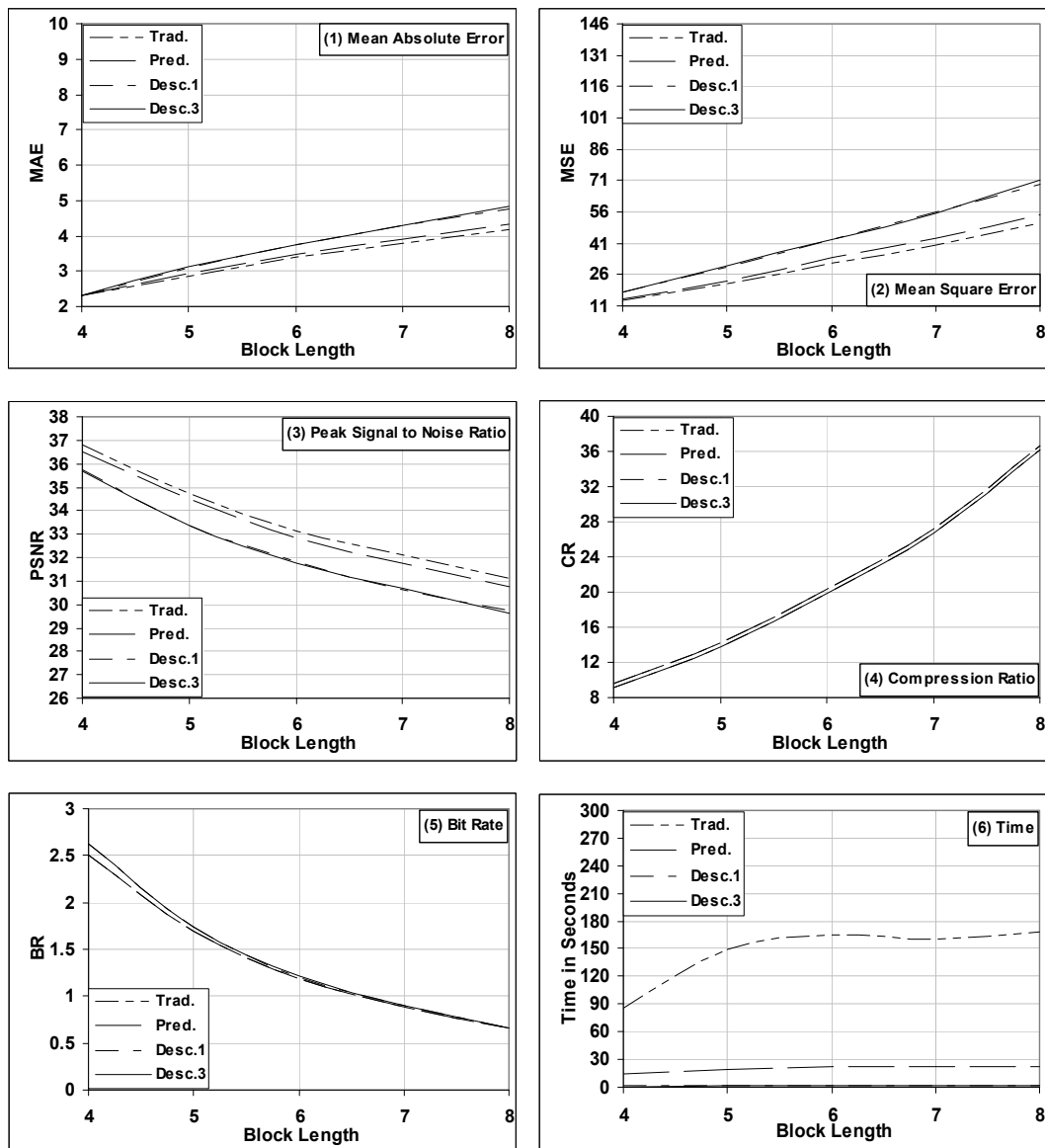


Figure (4.3) The effects of block length on the compression performance parameters, when Girl image is used as test material

4.5 Jump Step Test

This set of conducted tests is dedicated to investigate the effects of jump step parameter on the compression performance parameters for the four FIC schemes. This set of tests was conducted on both images (Lena, Girl). The results of this set of tests indicated that the performance parameters are significantly affected by jump step values, which had been varied between 1 and 4. The values of other parameters were set fixed at their default values. The obtained results are described in the following:

1. Figure (4.4) shows some of the reconstructed Lena images, where the applied FIC scheme is Dis1FIC.

| | | | |
|---|---|---|--|
|  |  |  |  |
| Jump Step=1 MAE=3.92 MSE=42.01 PSNR=31.90 CR=8.875 BR=2.704 Time=1.18 | Jump Step=2 MAE=4.30 MSE=52.38 PSNR=30.94 CR=9.532 BR=2.518 Time=0.41 | Jump Step=3 MAE=4.72 MSE=64.31 PSNR=30.05 CR=9.881 BR=2.429 Time=0.27 | Jump Step=4 MAE=4.91 MSE=70.58 PSNR=29.64 CR=10.268 BR=2.337 Time=0.23 |

Figure (4.4) Samples of the reconstructed Lena images when Dis1FIC scheme is applied

2. Figure (4.5) shows the samples of jump step test set for Dis3FIC scheme applied on Lena image.

| | | | |
|---|---|---|--|
|  |  |  |  |
| Jump Step=1 MAE=3.91 MSE=42.05 PSNR=31.89 CR=8.863 BR=2.708 Time=1.24 | Jump Step=2 MAE=4.31 MSE=52.08 PSNR=30.96 CR=9.525 BR=2.520 Time=0.43 | Jump Step=3 MAE=4.72 MSE=63.97 PSNR=30.07 CR=9.886 BR=2.428 Time=0.37 | Jump Step=4 MAE=4.93 MSE=71.46 PSNR=29.59 CR=10.258 BR=2.340 Time=0.31 |

Figure (4.5) The jump step test results for Dis3FIC scheme

3. Figure (4.6) shows the effects of jump step parameter value on the compression performance parameters of the four FIC schemes when they applied on Lena image.

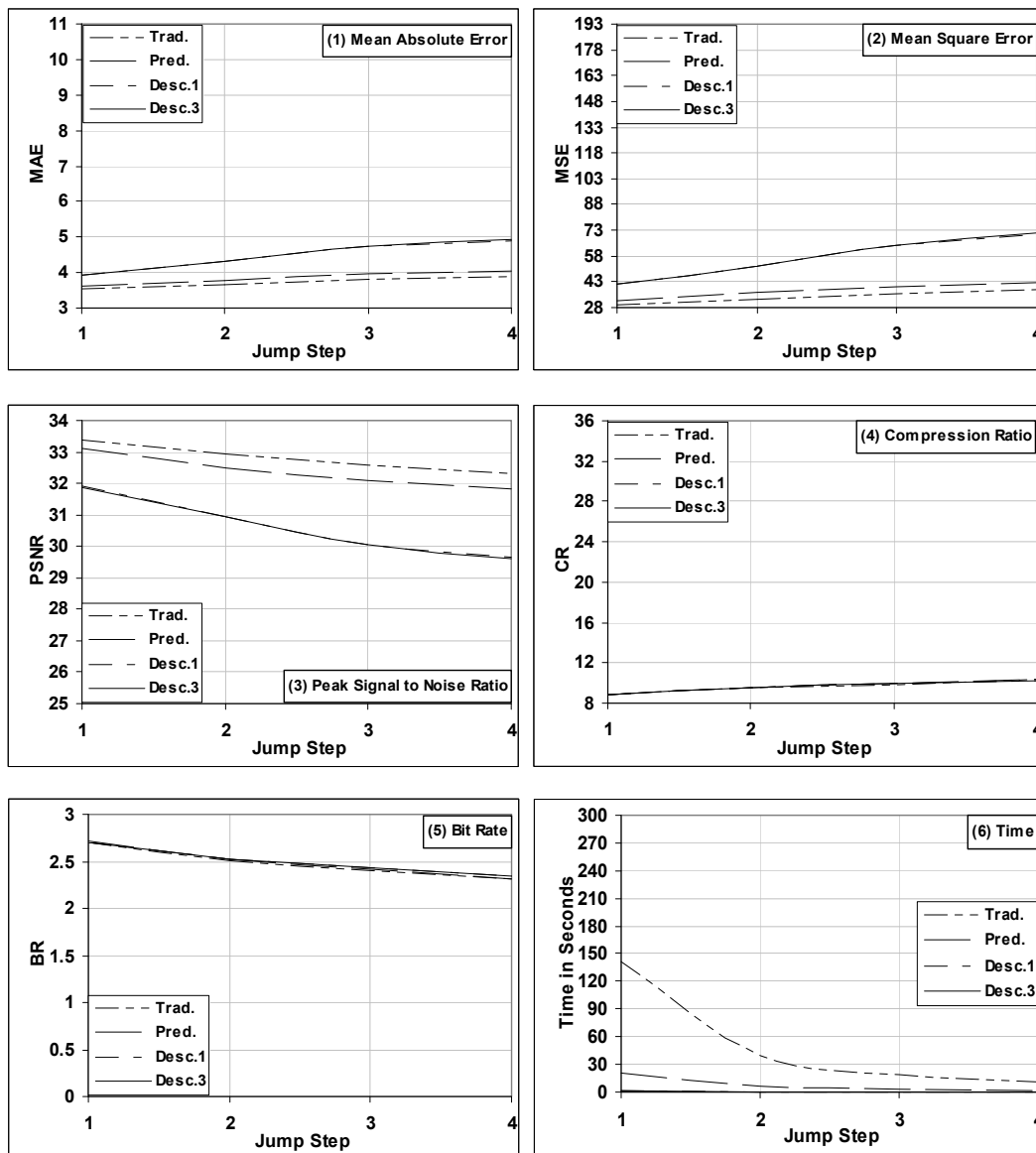


Figure (4.6) The effect of jump step parameter on the performance of the four FIC schemes, for the case of Lena image

4. Figure (4.7) shows the difference between the four FIC schemes when they implemented on Girl image using same coding parameters:

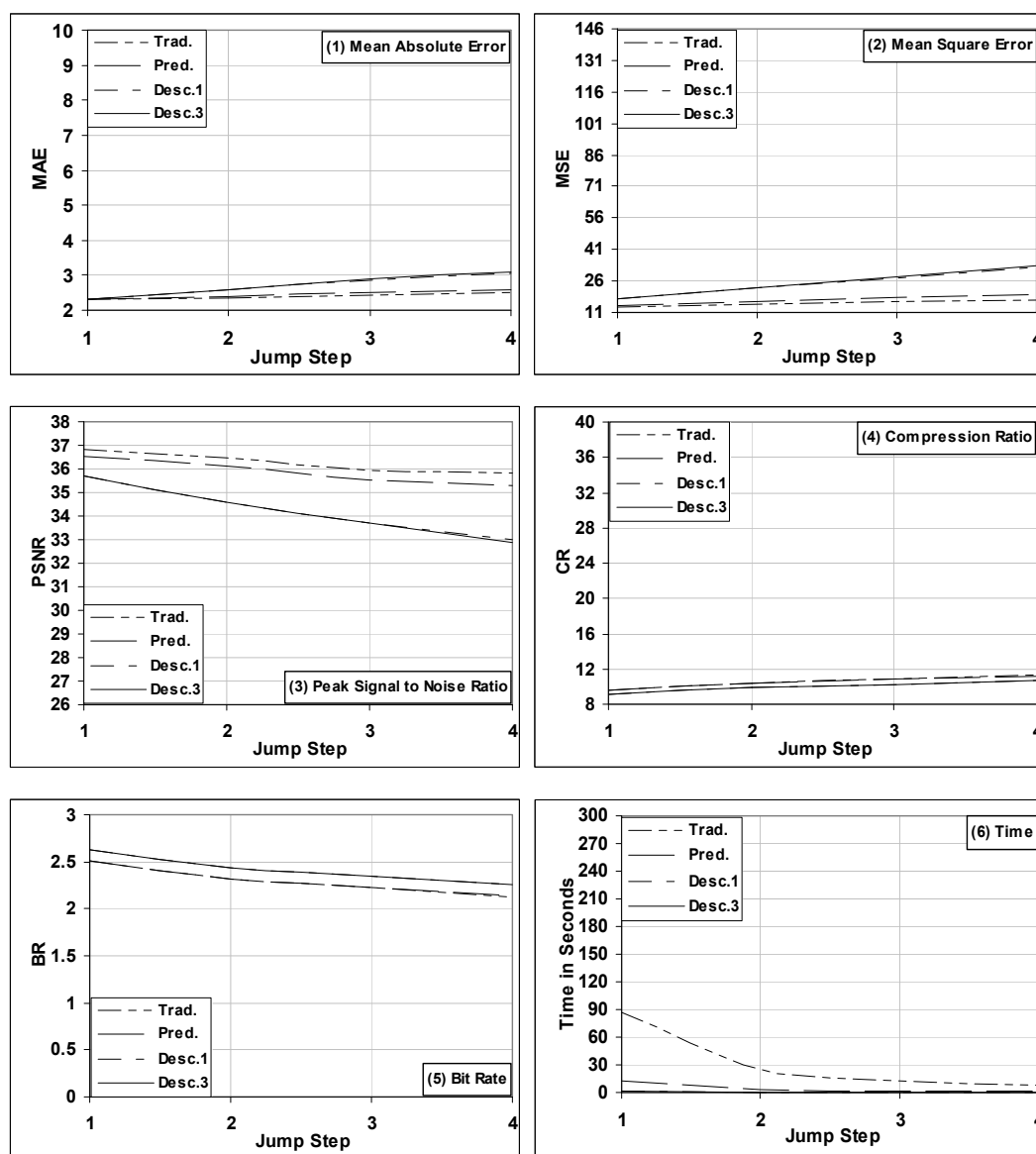


Figure (4.7) The effect of jump step of the four FIC schemes when they applied on Girl image

4.6 Maximum Scale Test

In this set of tests the effect of maximum scale parameter is studied for the four established IFS- schemes. In this set of tests the value of maximum scale parameter was varied within the range [1, 5]. The values of other parameters were fixed at their default values. The results of this set are summarized in the following tables and figures:

1. Table (4.4) shows the effects of maximum scale parameter on the compression performance of Dis1FIC scheme, applied on Lena image.

Table (4.4) The results of maximum scale test of Dis1FIC scheme

| Max Scale | Over All | | | Compression Ratio | Bit Rate | Time in Seconds |
|-----------|----------|-------|-------|-------------------|----------|-----------------|
| | MAE | MSE | PSNR | | | |
| 1 | 4.04 | 46.23 | 31.48 | 8.786 | 2.732 | 1.14 |
| 2 | 3.92 | 42.35 | 31.86 | 8.786 | 2.732 | 1.24 |
| 3 | 3.92 | 42.01 | 31.90 | 8.875 | 2.704 | 1.26 |
| 4 | 3.92 | 41.85 | 31.91 | 8.929 | 2.688 | 1.16 |
| 5 | 3.94 | 42.13 | 31.92 | 8.993 | 2.669 | 1.19 |

2. Table (4.5) shows the effects of maximum scale parameter on the compression performance of Dis3FIC scheme, applied on Lena image.

Table (4.5) The results of maximum scale test of Dis3FIC scheme

| Max Scale | Over All | | | Compression Ratio | Bit Rate | Time in Seconds |
|-----------|----------|-------|-------|-------------------|----------|-----------------|
| | MAE | MSE | PSNR | | | |
| 1 | 4.04 | 46.82 | 31.43 | 8.786 | 2.732 | 1.24 |
| 2 | 3.92 | 42.37 | 31.86 | 8.786 | 2.732 | 1.22 |
| 3 | 3.91 | 42.05 | 31.89 | 8.863 | 2.708 | 1.26 |
| 4 | 3.93 | 42.07 | 31.89 | 8.934 | 2.686 | 1.20 |
| 5 | 3.93 | 42.02 | 31.90 | 8.991 | 2.669 | 1.21 |

3. Figure (4.8) illustrates the effects of maximum scale parameter on the performance parameters of the four FIC schemes, the Lena image was used as test material.

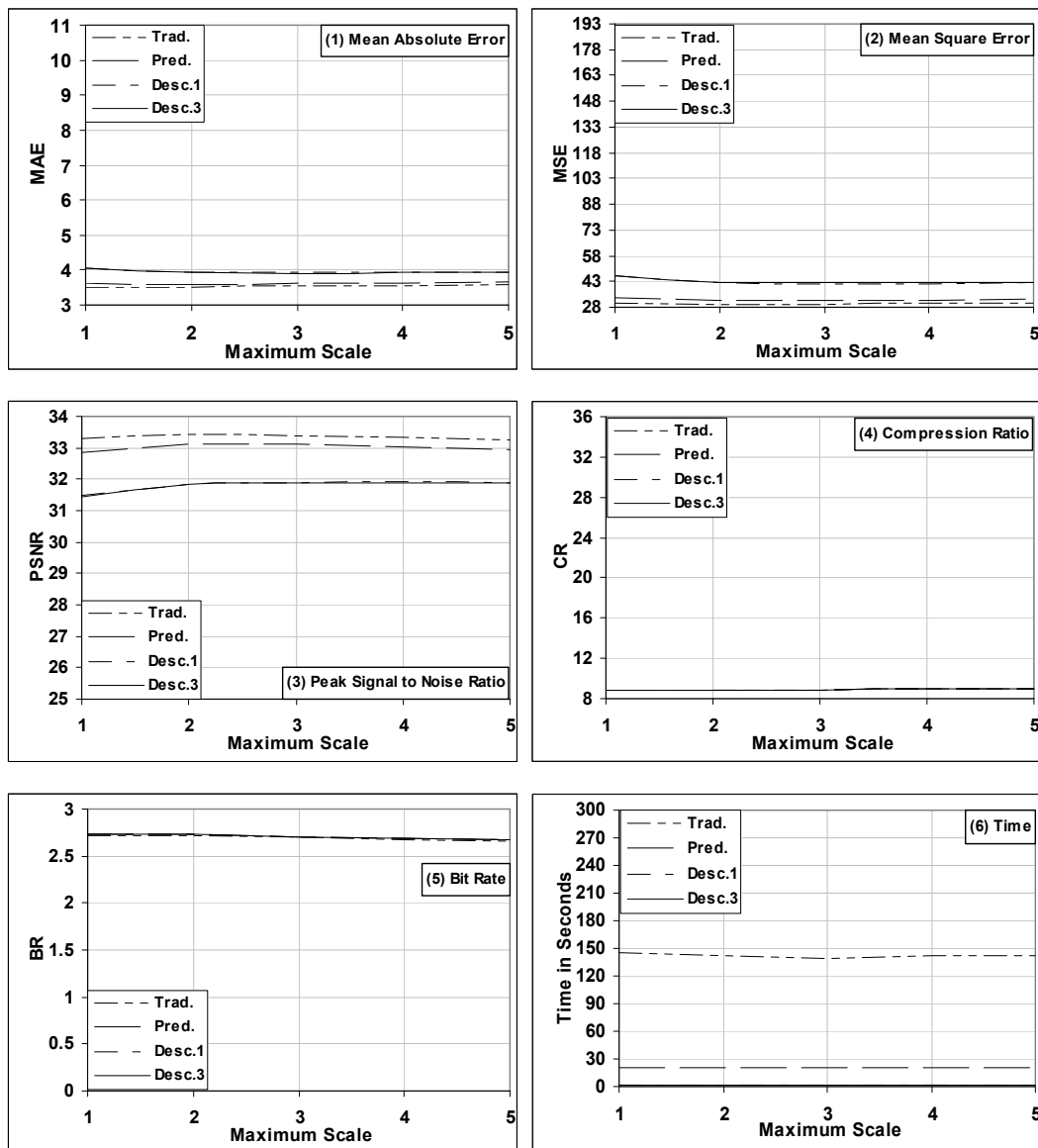


Figure (4.8) The effect of maximum scale parameter on the performance parameters of the four FIC-schemes, when they applied on Lena image

4. Figure (4.9) shows the effects of maximum scale parameter of the four FIC schemes when they applied on Girl image.

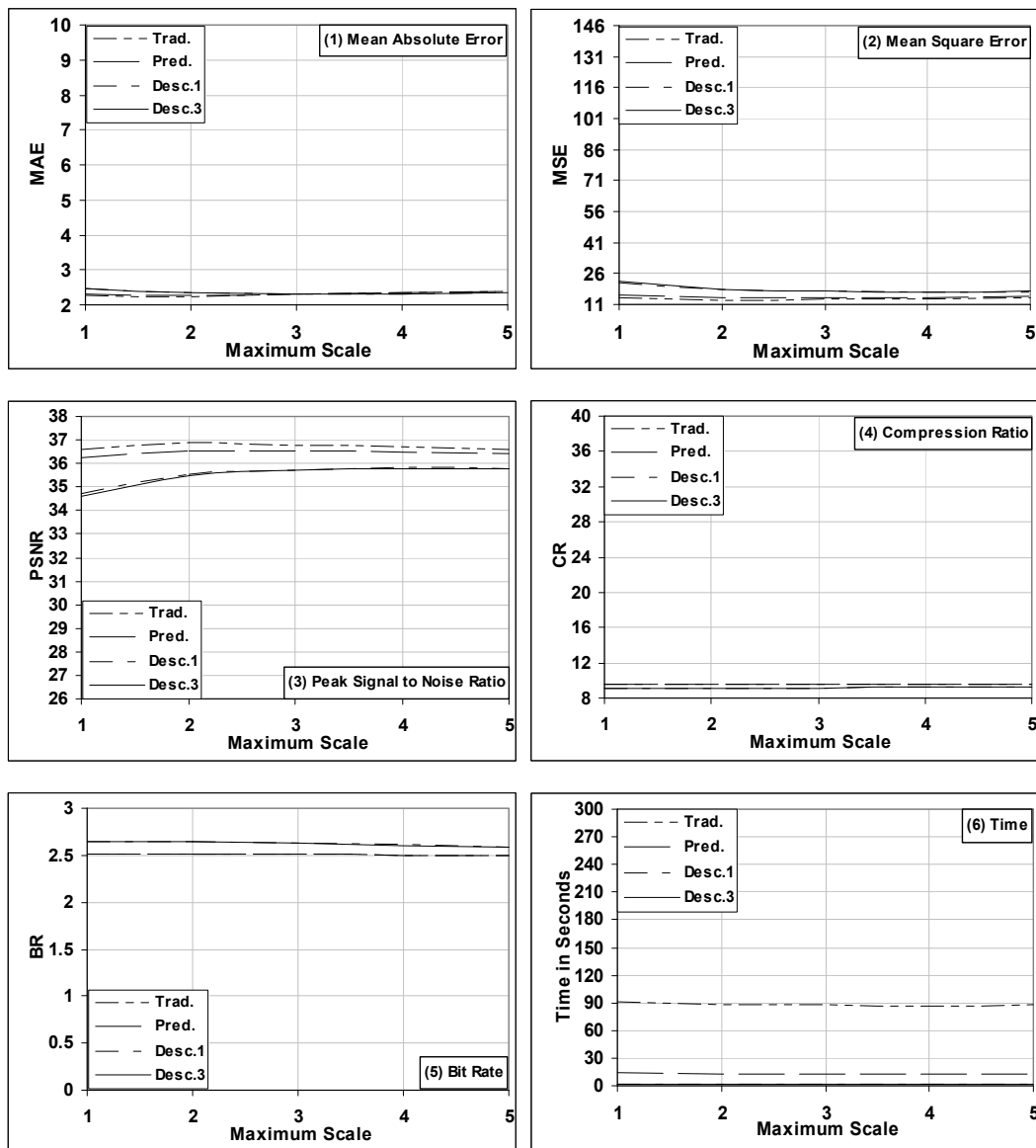


Figure (4.9) The effect of maximum scale on the performance of the four FIC schemes, when they applied on Girl image

4.7 Scale Bits Test

In this section the result of some tests made on both images (Lena and Girl) are shown to investigate the effects of changing the number of bits used to represent the value of scale coefficients. In this set of tests the value of scale bits parameter was varied from 2 to 8. The values of other parameters were fixed at their default values. The following figures illustrate the obtained results:

1. Figure (4.10) shows the effect of the scale bits parameter on the performance of Dis1FIC scheme, the test image was "Girl".



Figure (4.10) Some samples of reconstructed Girl images when it is compressed by Dis1FIC using different values of scale bit parameter

2. Figure (4.11) describes the results of applying Dis3FIC scheme on Girl image, the number of scale bits was varied.



Figure (4.11) Some samples of reconstructed Girl images when it is compressed by Dis3FIC scheme using different values of scale bit parameter

3. Figure (4.12) shows the effect of scale bits parameter on the performance of the four FIC schemes, when they applied on Lena image.

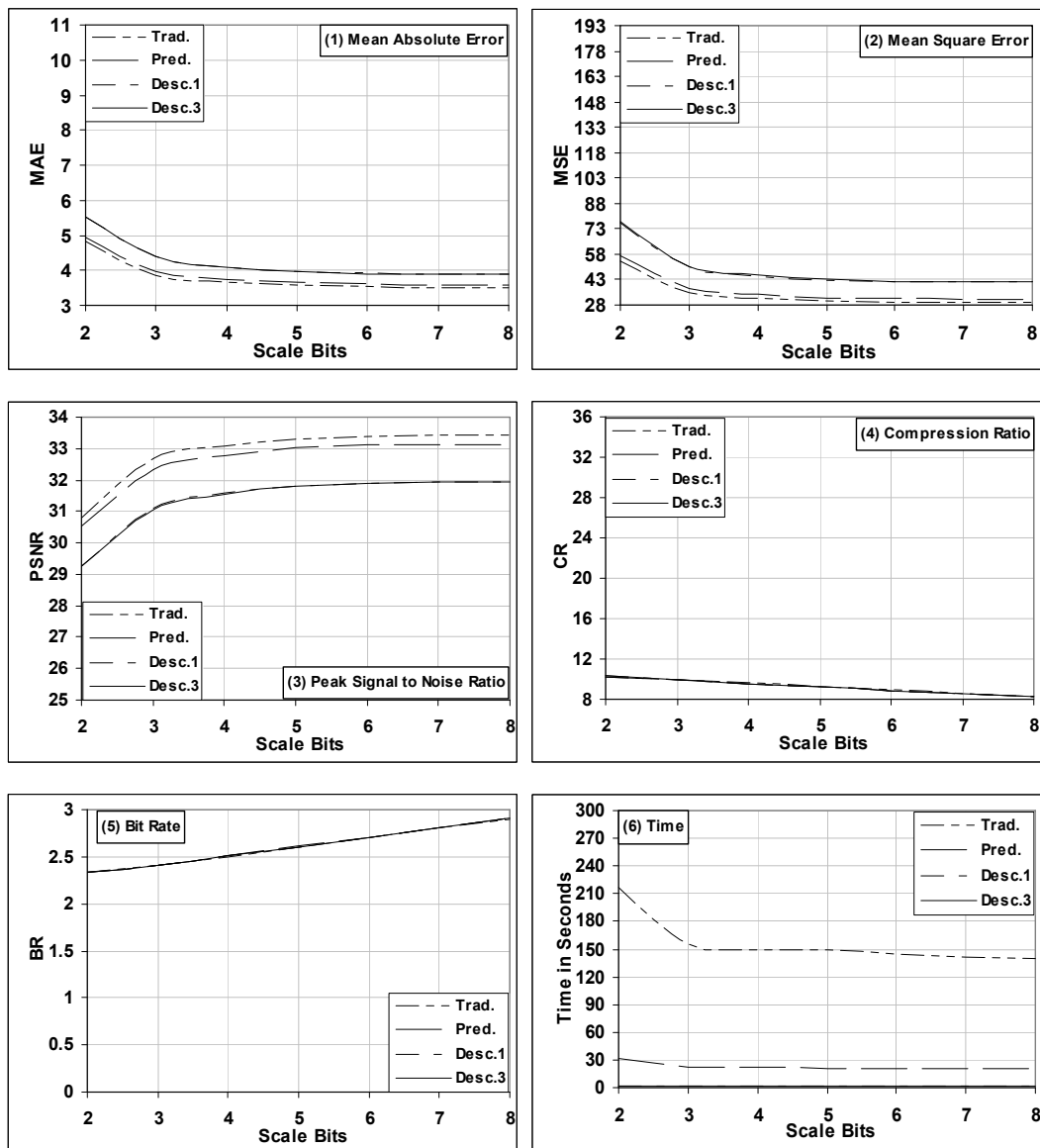


Figure (4.12) The effect of scale bits parameter on the performance of the four FIC schemes, when they applied on Lena image

4. Figure (4.13) shows the effect of scale bits parameter on the compression performance parameters of the four FIC-schemes, applied on Girl image.

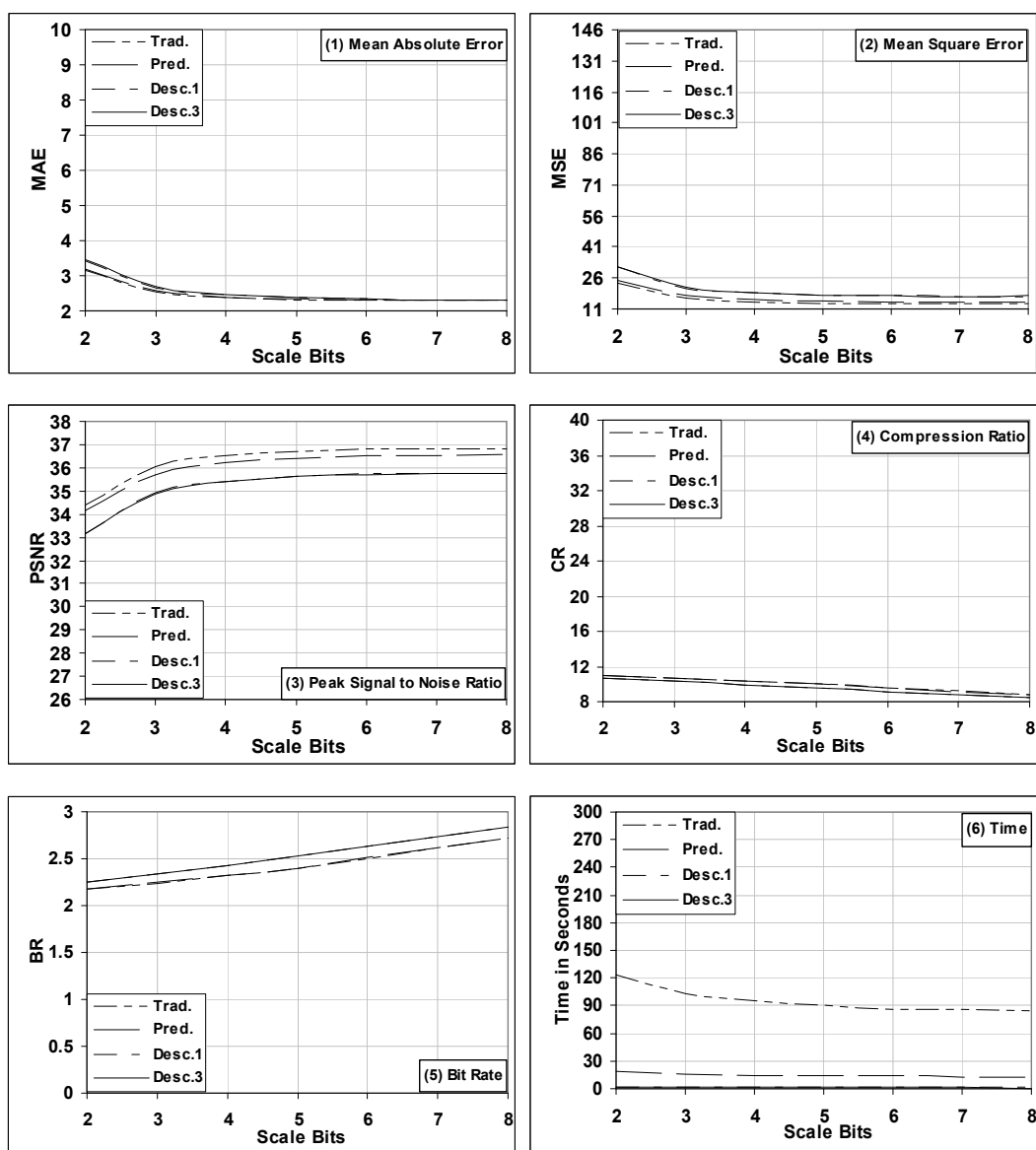


Figure (4.13) The effect of scale bits parameter on the performance parameters of the four FIC schemes, when they applied on Girl image

4.8 Offset Bits Test

This set of conducted tests is to investigate the effects of the parameter "number of offset bits" on the performance parameters of the established FIC schemes. The value of the offset bits parameter was varied between 4 to 8. While, the values of other parameters were set fixed to have their default values. The results of this set of tests is described in the following tables and figures:

1. Table (4.6) presents the effect of offset bits parameter on the performance of Dis1FIC, when it is applied on Girl image.

Table (4.6) The results of offset bits tests of Dis1FIC scheme, applied on Girl image

| Offset Bits | Over All | | | Compression Ratio | Bit Rate | Time in Seconds |
|-------------|----------|--------|-------|-------------------|----------|-----------------|
| | MAE | MSE | PSNR | | | |
| 4 | 8.74 | 134.46 | 26.84 | 10.504 | 2.285 | 1.51 |
| 5 | 4.90 | 46.99 | 31.41 | 10.163 | 2.361 | 1.42 |
| 6 | 3.19 | 24.43 | 34.25 | 9.846 | 2.438 | 1.31 |
| 7 | 2.56 | 18.63 | 35.43 | 9.491 | 2.529 | 0.99 |
| 8 | 2.33 | 17.39 | 35.73 | 9.126 | 2.630 | 0.93 |

2. Table (4.7) lists the tests results of Dis3FIC scheme, when it is applied on Girl image with different values of offset bits parameter.

Table (4.7) The results of offset bits test of Dis3FIC scheme, applied on Girl Image

| Offset Bits | Over All | | | Compression Ratio | Bit Rate | Time in Seconds |
|-------------|----------|--------|-------|-------------------|----------|-----------------|
| | MAE | MSE | PSNR | | | |
| 4 | 8.73 | 133.63 | 26.87 | 10.498 | 2.286 | 1.42 |
| 5 | 4.90 | 47.45 | 31.37 | 10.164 | 2.361 | 1.34 |
| 6 | 3.17 | 24.17 | 34.30 | 9.844 | 2.438 | 1.30 |
| 7 | 2.56 | 18.69 | 35.41 | 9.483 | 2.531 | 1.02 |
| 8 | 2.33 | 17.55 | 35.69 | 9.123 | 2.631 | 0.81 |

3. Figure (4.14) shows the effect of offset bits parameter when Lena image is used as test material.

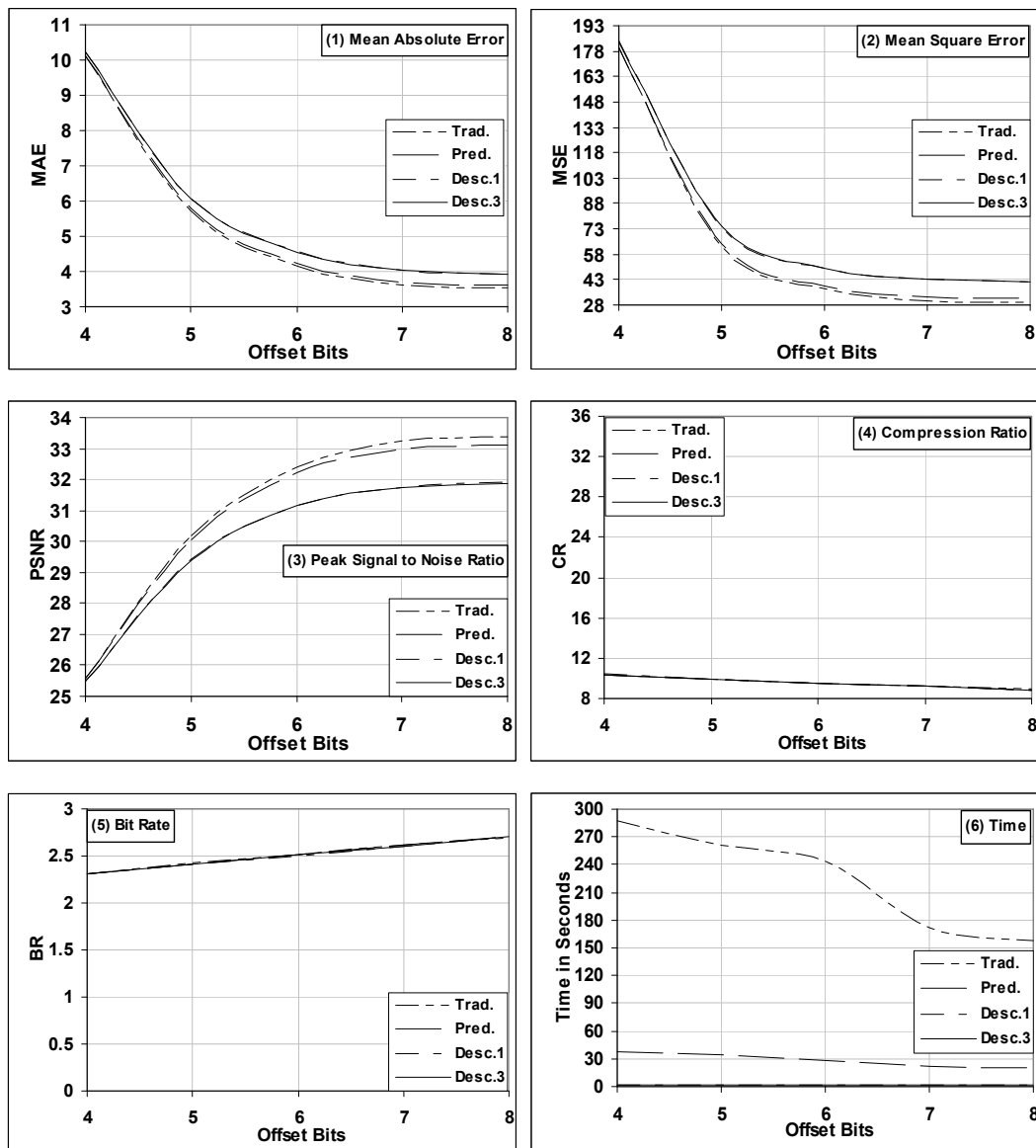


Figure (4.14) The effect of the parameter "offset bits" on the compression performance, when Lena image is used as test object

4. Figure (4.15) illustrates the performance behavior of the four FIC-schemes when the value of scale bits parameter is varied. This test was conducted using Girl image.

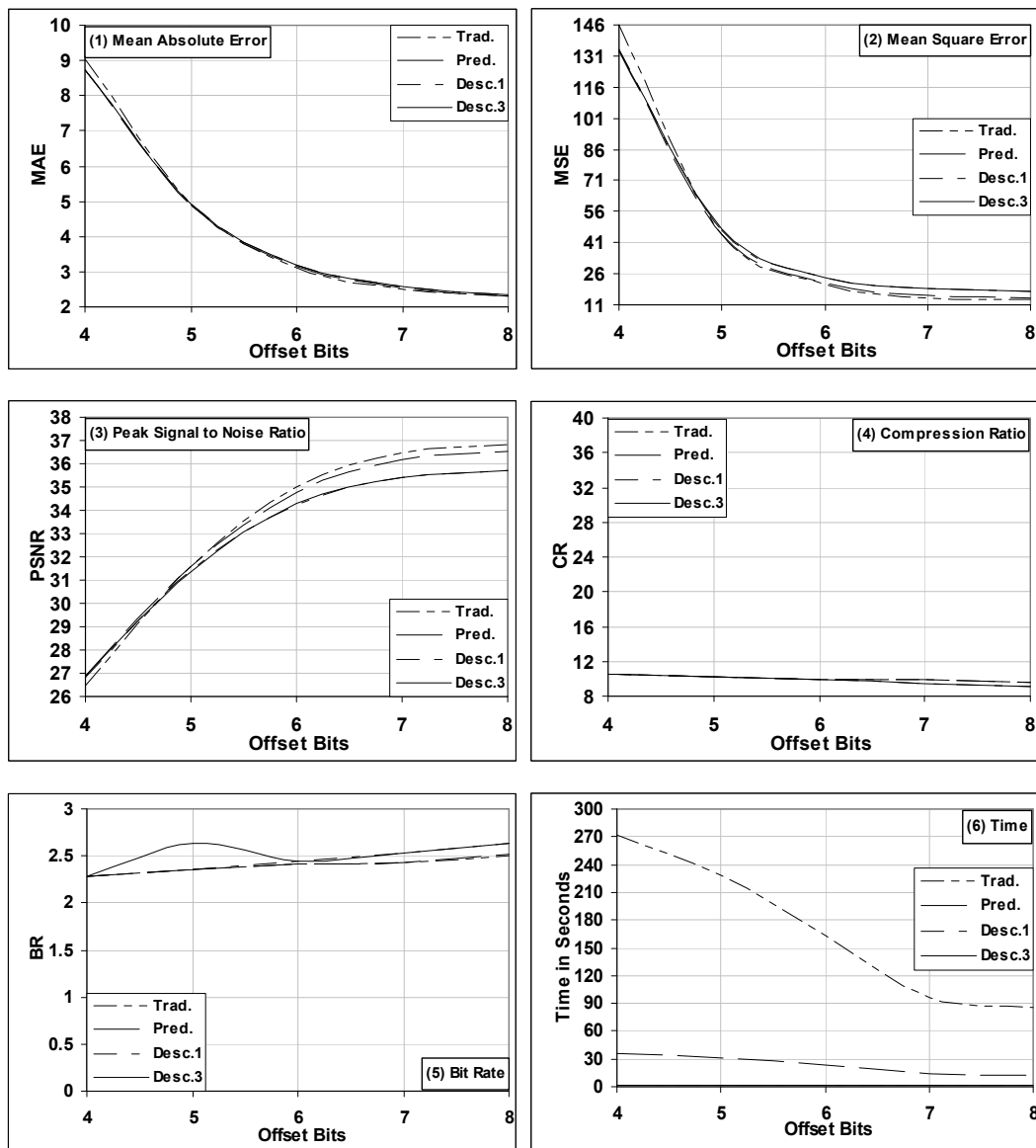


Figure (4.15) The effect of the parameter "offset bits" on the performance of the four FIC schemes, when Girl image is used as test object

4.9 Minimum Error Test

In this set of tests the effects of the parameter "Minimum Error" are investigated for the four established FIC schemes. The value of minimum error parameter was varied from 1 to 15. The values of other coding parameters were set fixed at their default values. The results of this set of tests are summarized in the following:

1. Figure (4.16) shows some samples of the reconstructed Lena images. When it compressed by Dis1FIC using different values of minimum error parameter.



Figure (4.16) Samples of the reconstructed Lena image when it is compressed by Dis1FIC scheme using different values of minimum error

2. Figure (4.17) presents some of the reconstructed Lena images. When it is compressed using Dis3FIC scheme with different values of minimum error parameter.

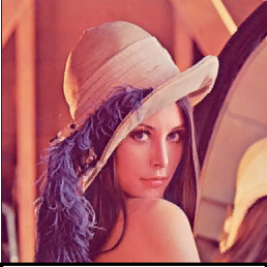
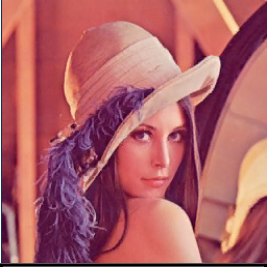
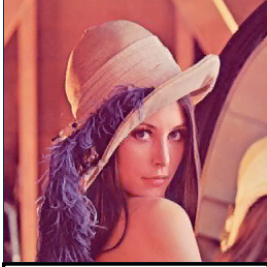

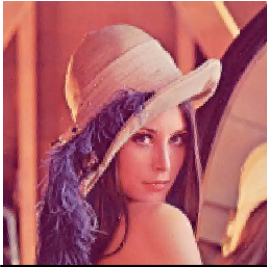
| | | | |
|---|--|--|---|
|  |  |  |  |
| Min Error=1 MAE=3.91 MSE=42.03 PSNR=31.90 CR=8.859 BR=2.709 Time=1.31 | Min Error=3 MAE=3.98 MSE=43.18 PSNR=31.78 CR=8.859 BR=2.709 Time=1.02 | Min Error=5 MAE=4.35 MSE=50.95 PSNR=31.06 CR=8.864 BR=2.708 Time=0.97 | Min Error=7 MAE=4.37 MSE=51.43 PSNR=31.02 CR=8.864 BR=2.708 Time=0.96 |
|  |  |  |  |
| Min Error=9 MAE=4.53 MSE=56.29 PSNR=30.63 CR=8.865 BR=2.707 Time=0.95 | Min Error=11 MAE=4.53 MSE=56.33 PSNR=30.62 CR=8.866 BR=2.707 Time=0.94 | Min Error=13 MAE=4.54 MSE=56.66 PSNR=30.60 CR=8.865 BR=2.707 Time=0.94 | Min Error=15 MAE=6.16 MSE=115.34 PSNR=27.51 CR=8.863 BR=2.708 Time=0.93 |

Figure (4.17) Some samples of the reconstructed Lena image compressed by Dis3FIC scheme using different values of parameter "minimum error"

3. Figure (4.18) illustrates the effectiveness of minimum error parameter on the performance behavior of the four established FIC schemes. These compression schemes were applied on Lena image.

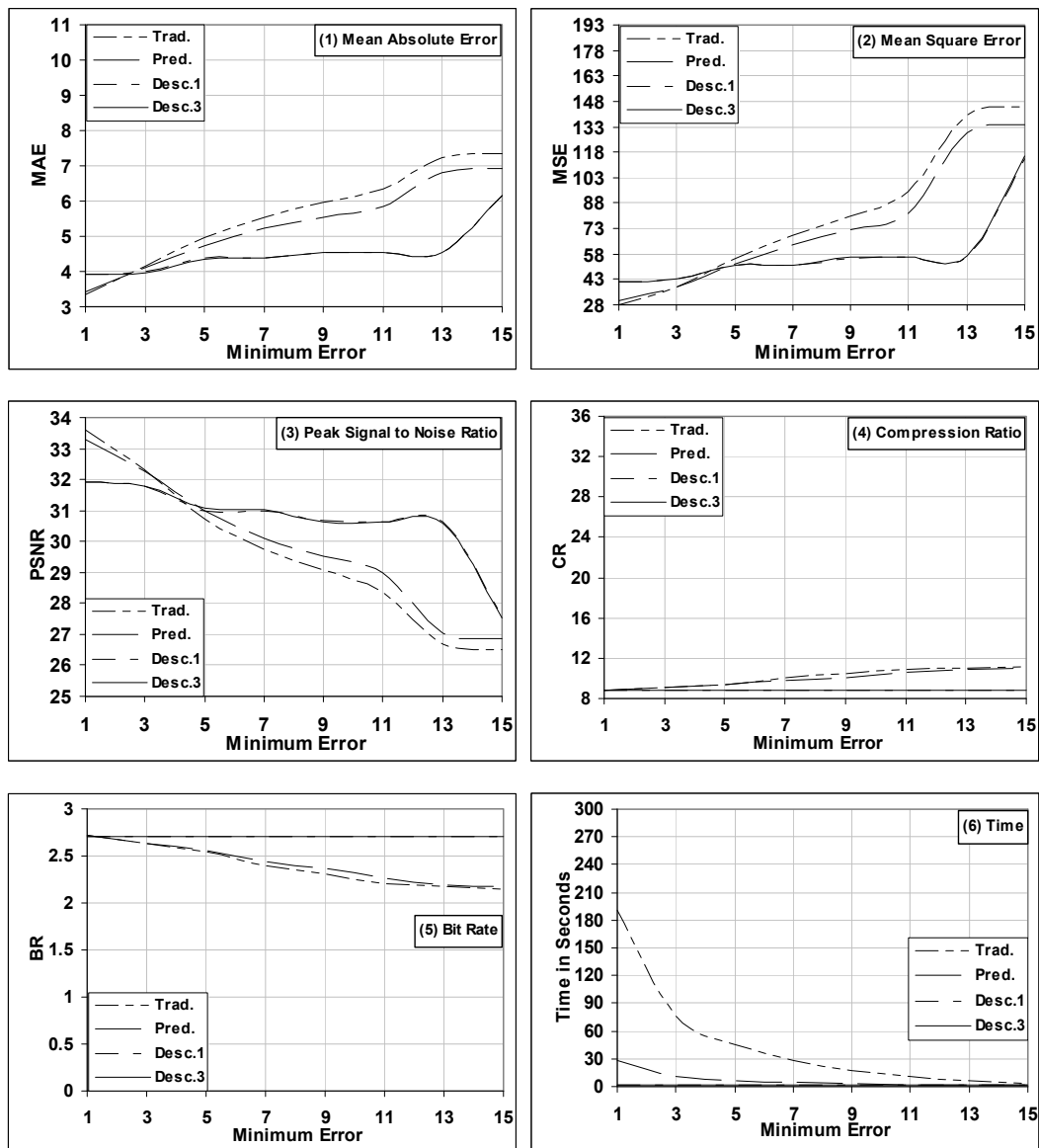


Figure (4.18) The effect of minimum error parameter on the performance of the four FIC-schemes applied on Lena image

4. Figure (4.19) shows the differences in performance behavior of the four FIC-schemes, when they applied on Girl image using different values of minimum error parameter.

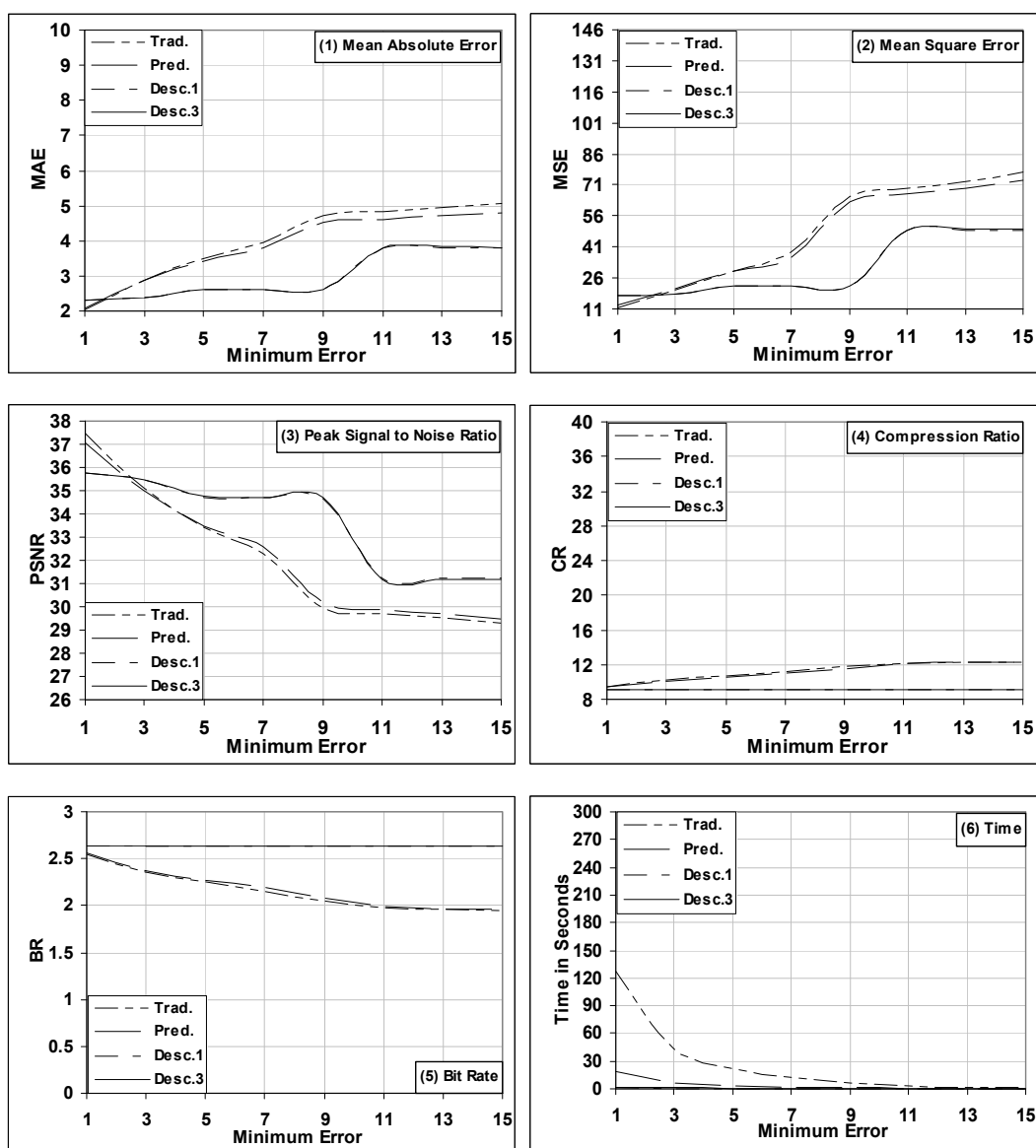


Figure (4.19) The effect of minimum error parameter on the performance parameters of the four FIC-schemes when they applied on Girl image

4.10 Minimum Block Error Test

The minimum block error coding parameter belongs, exclusively, to both Dis1FIC and Dis3FIC. In this set of tests the effects of this parameter on the performance of the two enhanced FIC schemes are investigated.

The listed figures in this section illustrates the effectiveness of minimum block error on the parameters MAE, MSE, PSNR, compression

ratio (CR), bit rate (BR), and encoding time. The value of minimum block error parameter was varied to have values from 1 to 19, and it is important to mention that the value of this parameter should be higher than or equal to the value of minimum error parameter. In this set of tests the values of other coding parameter were set fixed to have their default values. The results of this set of tests are summarized as follows:

1. Figure (4.20) shows samples of the reconstructed Girl image when it was compressed by Dis1FIC scheme using different values of minimum block error parameter.



Figure (4.20) Some samples of the reconstructed Girl image when it is compressed by Dis1FIC using different values of the parameter "minimum block error"

2. Figure (4.21) illustrates the effect of minimum block error parameter on the compression performance of Dis3FIC scheme, when it is applied on Girl image.

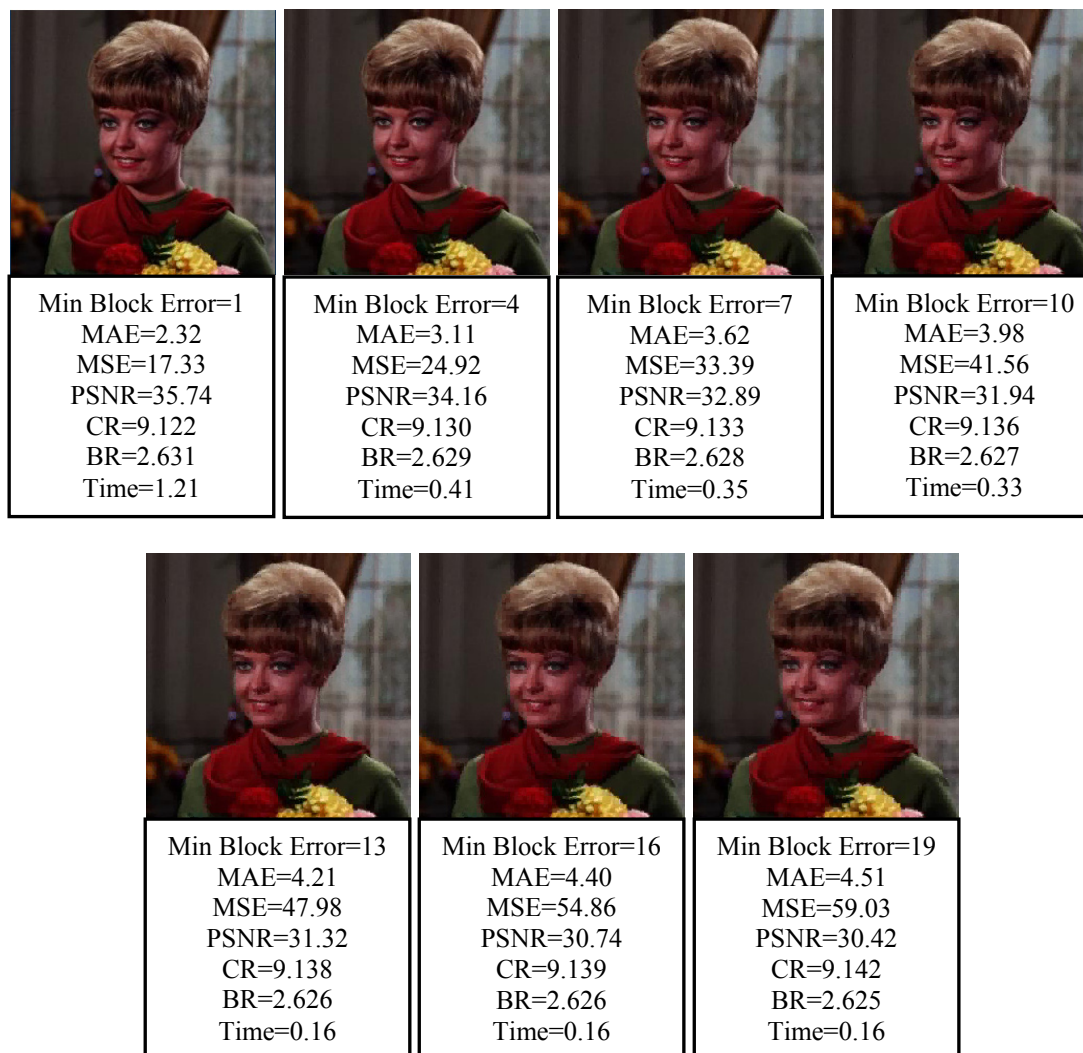


Figure (4.21) Some samples of reconstructed Girl image compressed by Dis3FIC scheme using various values of the parameter "minimum block error"

3. Figure (4.22) shows the effect of minimum block error parameter of Dis1FIC and Dis3FIC schemes when they applied on Lena image.

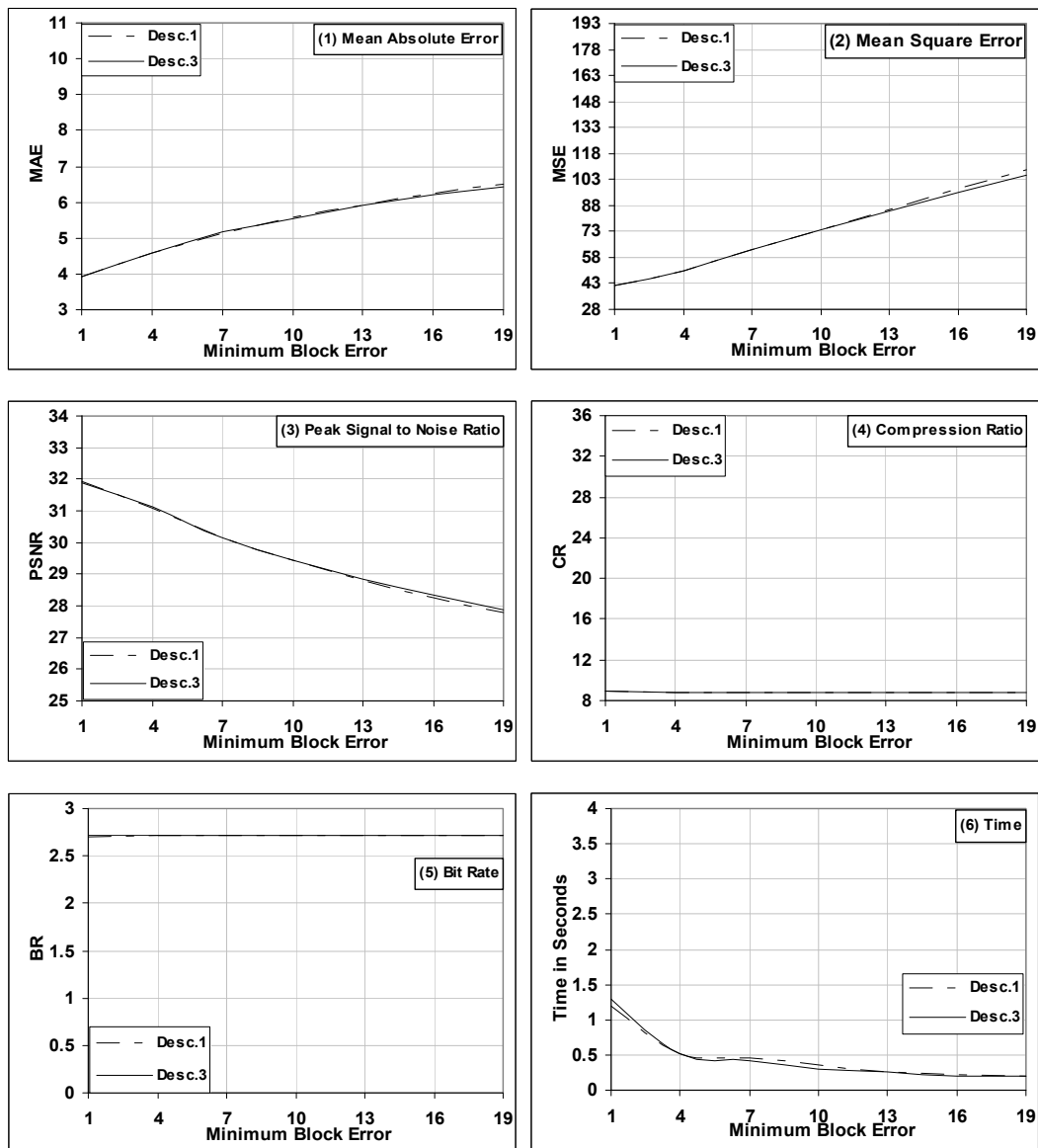


Figure (4.22) The effect of minimum block error parameter on the performance of Dis1FIC and Dis3FIC, when they applied on Lena image

4. Figure (4.23) shows the difference in behaviors of the performance parameters of the two enhanced FIC-schemes (i.e., Dis1FIC and Dis3FIC) when the value of parameter "minimum block error" was varied.

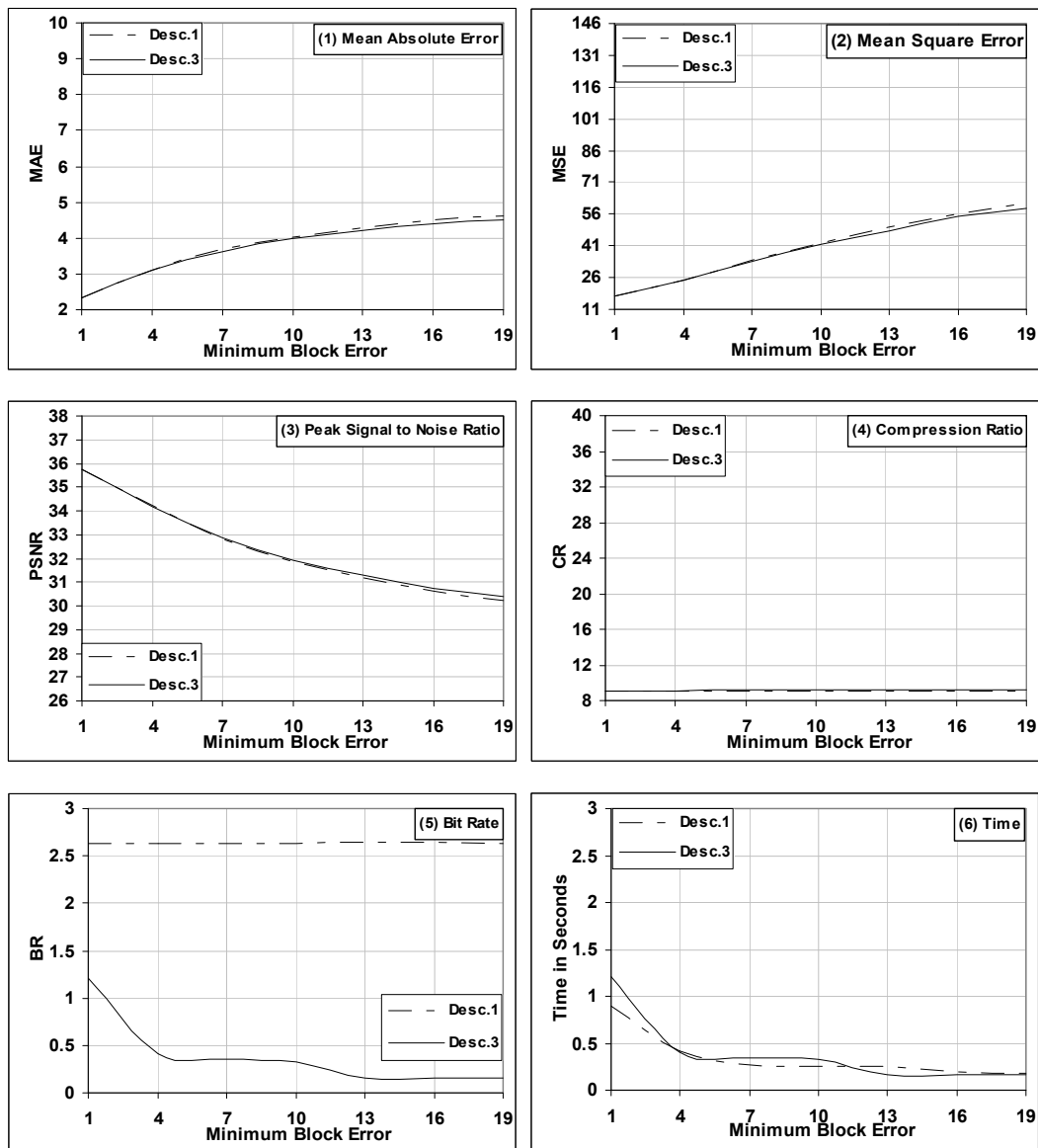


Figure (4.23) The effect of minimum block error parameter on the performance of Dis1FIC and Dis3FIC schemes when they applied on Girl image

4.11 Number of Bins Test

This set of conducted tests was applied on the two enhanced FIC-schemes (i.e., Dis1FIC and Dis3FIC), because this coding parameter belongs to these two schemes only. In this set of tests the value of this parameter was varied to investigate its effectiveness on the performance parameters of the two enhanced FIC-schemes. The values of other coding

parameters were fixed to have their default values. The tests results are summarized as follows:

1. Table (4.8) lists the values of compression performance parameters of Dis1FIC scheme, when it is applied on Lena image using different values of the parameter "no. of bins".

Table (4.8) The effect of number of bins parameter on the performance of Dis1FIC scheme, when it is applied on Lena image

| No. of Bins | Over All | | | Compression Ratio | Bit Rate | Time in Seconds |
|-------------|----------|-------|-------|-------------------|----------|-----------------|
| | MAE | MSE | PSNR | | | |
| 100 | 3.91 | 41.93 | 31.91 | 8.872 | 2.705 | 1.29 |
| 250 | 4.15 | 48.14 | 31.31 | 8.867 | 2.707 | 0.62 |
| 400 | 4.33 | 53.29 | 30.86 | 8.868 | 2.706 | 0.45 |
| 550 | 4.46 | 56.91 | 30.58 | 8.864 | 2.707 | 0.41 |
| 700 | 4.58 | 60.58 | 30.31 | 8.866 | 2.707 | 0.41 |
| 850 | 4.67 | 63.09 | 30.13 | 8.860 | 2.709 | 0.41 |
| 1000 | 4.76 | 66.14 | 29.93 | 8.857 | 2.710 | 0.36 |

2. Table (4.9) shows the values of compression performance parameters of Dis3FIC scheme, when it is applied on Lena image using various values of the parameter "no. of bins".

Table (4.9) The effect of number of bins parameter on the performance of Dis3FIC scheme, applied on Lena image

| No. of Bins | Over All | | | Compression Ratio | Bit Rate | Time in Seconds |
|-------------|----------|-------|-------|-------------------|----------|-----------------|
| | MAE | MSE | PSNR | | | |
| 100 | 3.91 | 42.03 | 31.90 | 8.859 | 2.709 | 1.26 |
| 250 | 4.17 | 48.54 | 31.27 | 8.870 | 2.706 | 0.67 |
| 400 | 4.31 | 52.12 | 30.96 | 8.869 | 2.706 | 0.55 |
| 550 | 4.44 | 55.79 | 30.66 | 8.866 | 2.707 | 0.43 |
| 700 | 4.54 | 58.58 | 30.45 | 8.865 | 2.707 | 0.39 |
| 850 | 4.62 | 61.09 | 30.27 | 8.862 | 2.708 | 0.38 |
| 1000 | 4.70 | 63.61 | 30.10 | 8.865 | 2.708 | 0.38 |

3. Figure (4.24) shows the difference in behavior of the performance parameters of the two enhanced FIC-schemes, when they applied on Lena image using different values of the parameter "no. of bins".

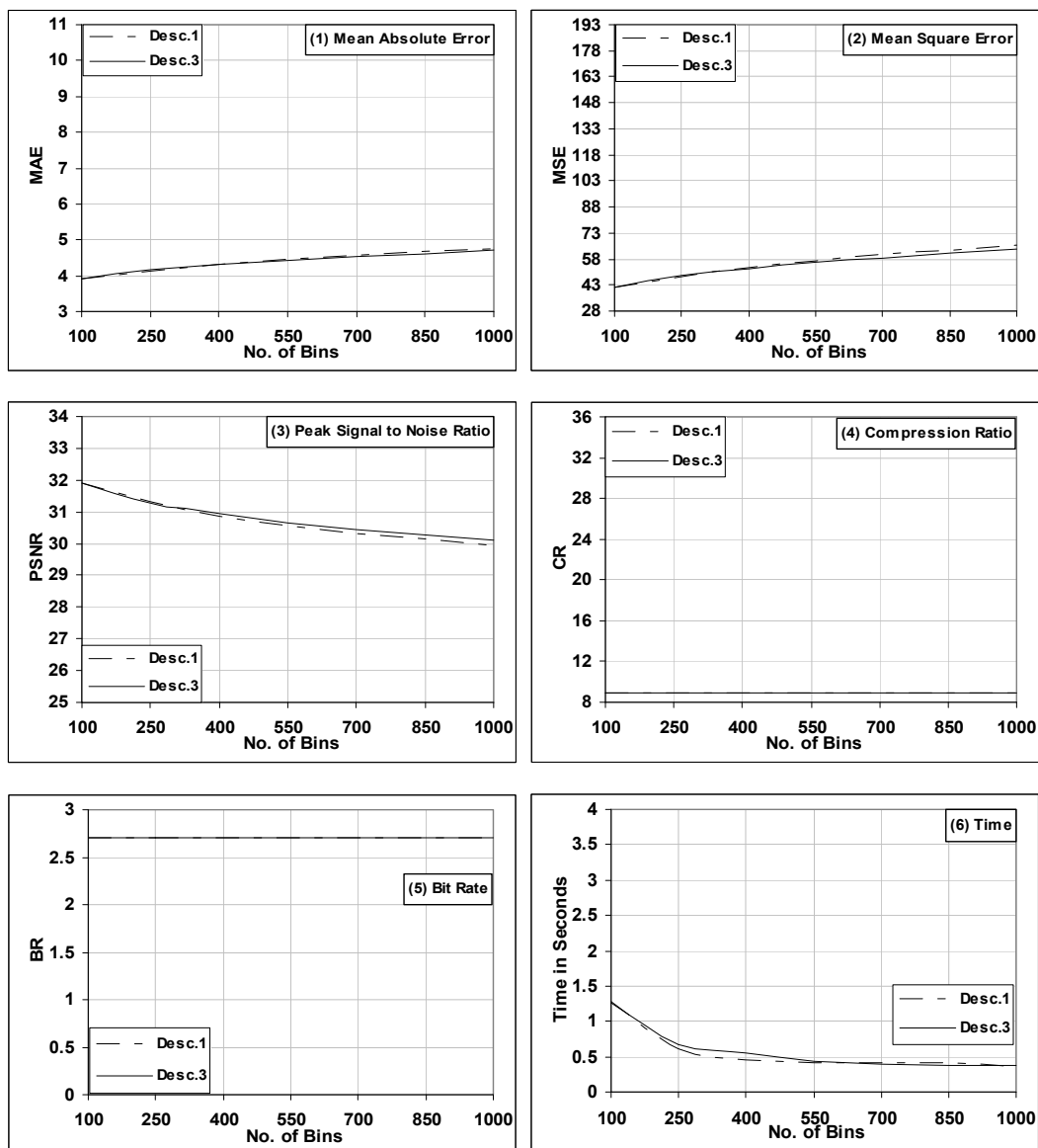


Figure (4.24) The effect of number of bins parameter on the performance of the two enhanced FIC-schemes, applied on Lena image

4. Figure (4.25) presents the difference in performance behavior of the two enhanced FIC-schemes (i.e., Dis1FIC and Dis3FIC), when they applied on Girl image using different values of the parameter "no. of bins".

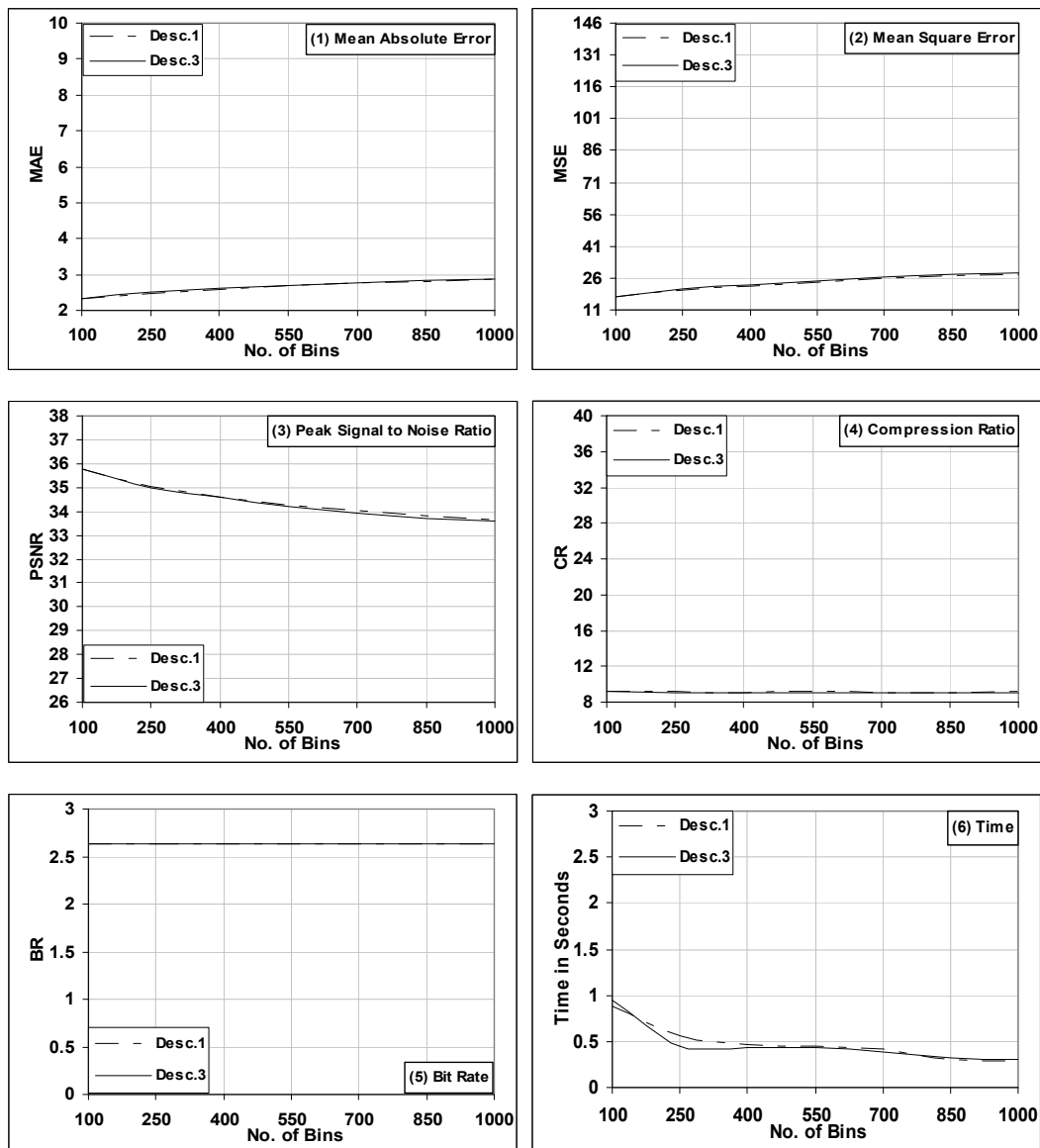


Figure (4.25) The effect of number of bins parameter on the performance of the two enhanced FIC-schemes, when they applied on Girl image

4.12 Window Size Test

In this set of tests, the effects of window size parameter on the performance of the two enhanced FIC-schemes (i.e., Dis1FIC and Dis3FIC) were explored, taking into consideration the window size parameter belongs to these two FIC-schemes only. In this set of tests, the value of window size was varied to have different integer values which

lay within the range [1, 5]. The values of other parameters were fix at their default values. The tests results are summarized as follows:

1. Table (4.10) lists the values of performance parameters of Dis1FIC scheme, when it is applied on Lena image using different values of window size.

Table (4.10) The effect of window size parameter on the performance of Dis1FIC scheme

| Window Size | Over All | | | Compression Ratio | Bit Rate | Time in Seconds |
|-------------|----------|-------|-------|-------------------|----------|-----------------|
| | MAE | MSE | PSNR | | | |
| 1 | 3.91 | 41.93 | 31.91 | 8.872 | 2.705 | 1.21 |
| 2 | 3.78 | 38.46 | 32.28 | 8.874 | 2.704 | 2.01 |
| 3 | 3.70 | 36.54 | 32.50 | 8.870 | 2.706 | 2.51 |
| 4 | 3.65 | 35.55 | 32.62 | 8.867 | 2.706 | 3.21 |
| 5 | 3.62 | 34.93 | 32.70 | 8.869 | 2.706 | 3.72 |

2. Table (4.11) lists the performance parameters of Dis3FIC scheme, when it is applied on Lena image using different values of the parameter "window size".

Table (4.11) The effect of window size parameter on performance parameters of Dis3FIC scheme

| Window Size | Over All | | | Compression Ratio | Bit Rate | Time in Seconds |
|-------------|----------|-------|-------|-------------------|----------|-----------------|
| | MAE | MSE | PSNR | | | |
| 1 | 3.91 | 42.03 | 31.90 | 8.859 | 2.709 | 1.21 |
| 2 | 3.78 | 38.83 | 32.24 | 8.866 | 2.707 | 1.93 |
| 3 | 3.71 | 36.90 | 32.46 | 8.866 | 2.707 | 2.53 |
| 4 | 3.67 | 35.74 | 32.60 | 8.870 | 2.706 | 3.37 |
| 5 | 3.64 | 35.00 | 32.69 | 8.868 | 2.706 | 3.85 |

3. Figure (4.26) shows the difference between the behaviors of the two FIC-schemes when they applied to compress Lena image, using different values of the parameter "window size".

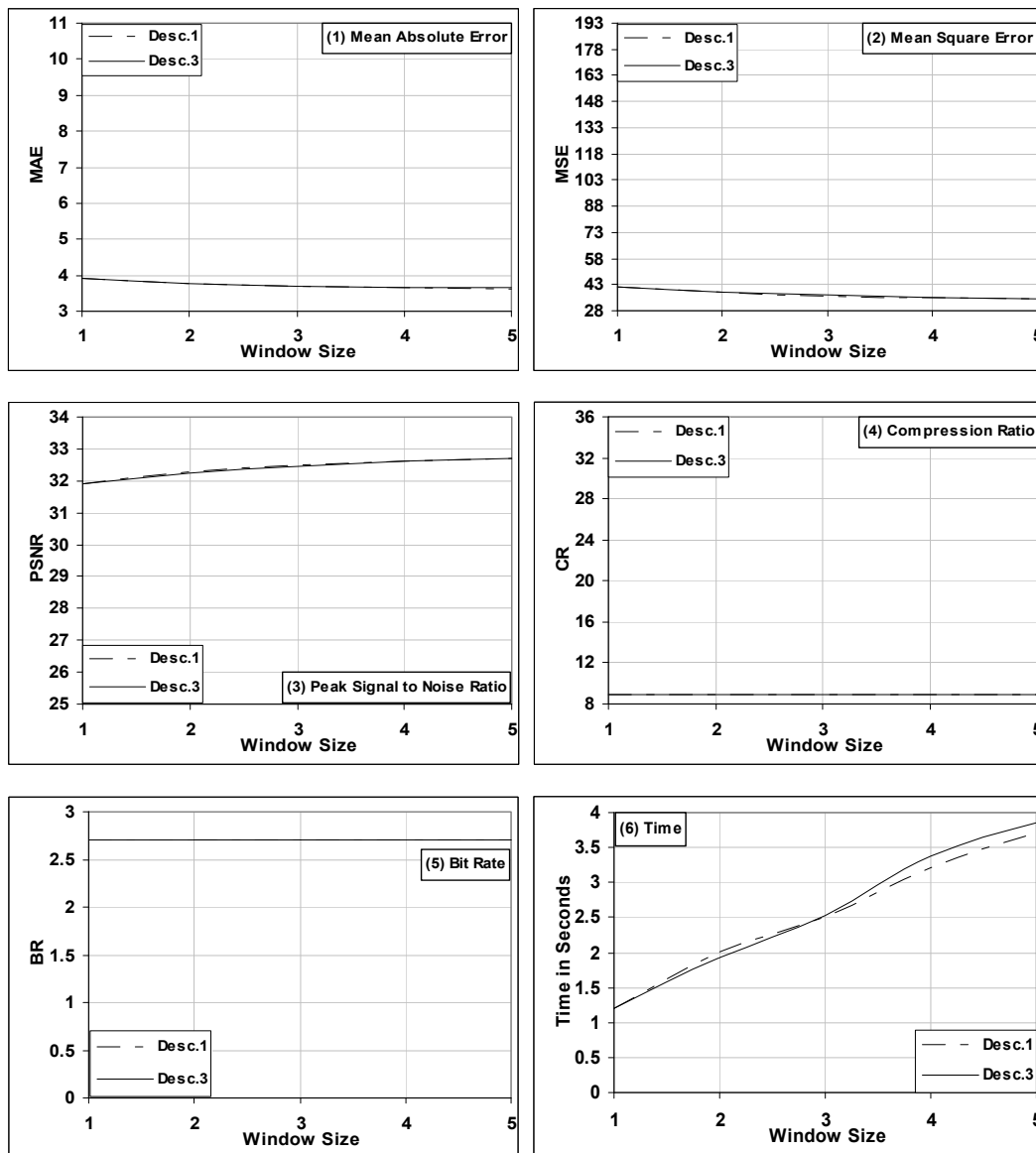


Figure (4.26) The effect of window size parameter on performance of the two enhanced FIC-methods, when they applied on Lena image

4. Figure (4.27) shows the difference between the performance of two enhanced FIC-schemes when they applied on Girl image, using different window size values.

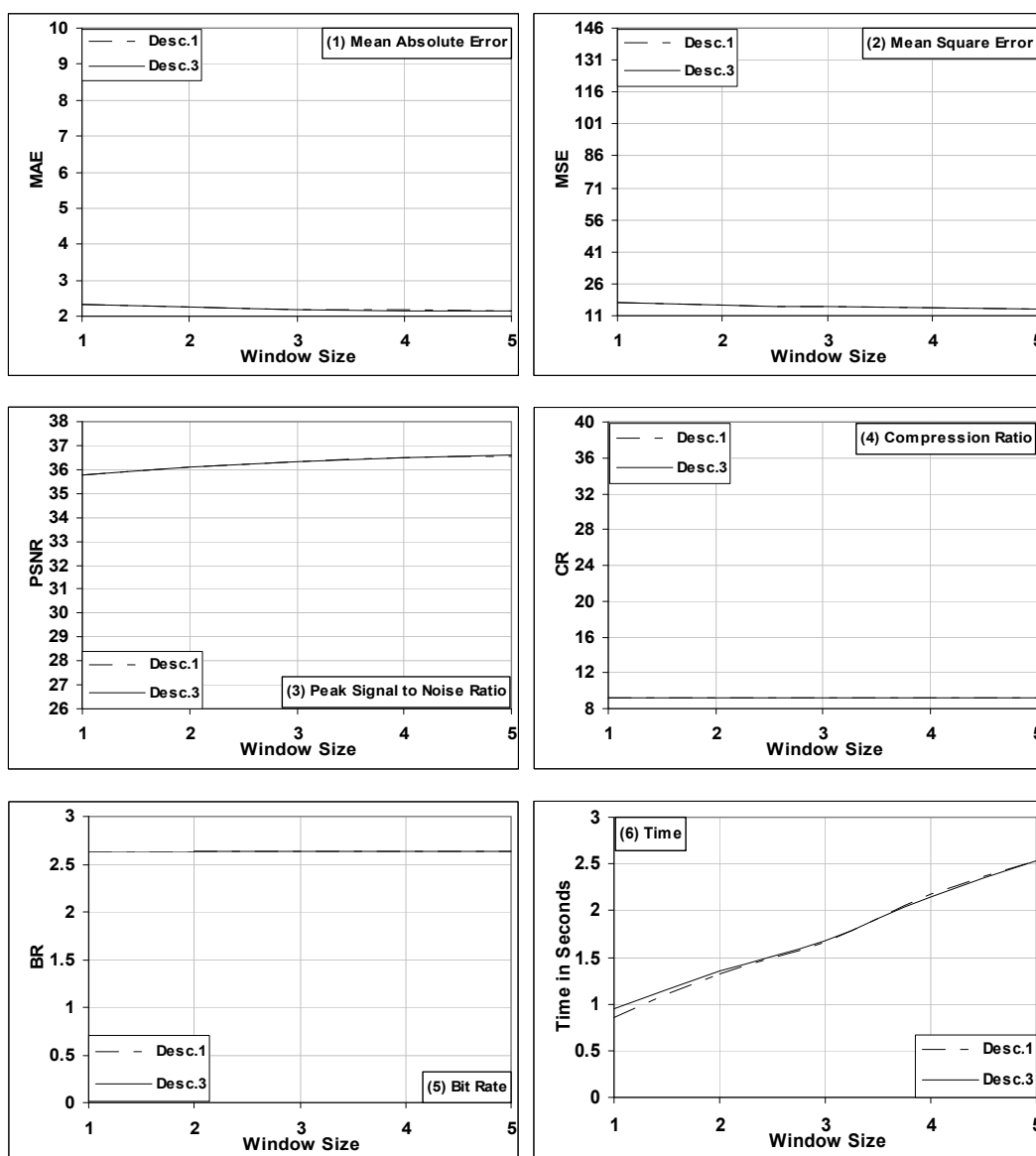


Figure (4.27) The effect of window size on the performance of the two enhanced FIC-schemes, when they applied on Girl image

4.13 The Effect of Both No. of Bins and Window Size

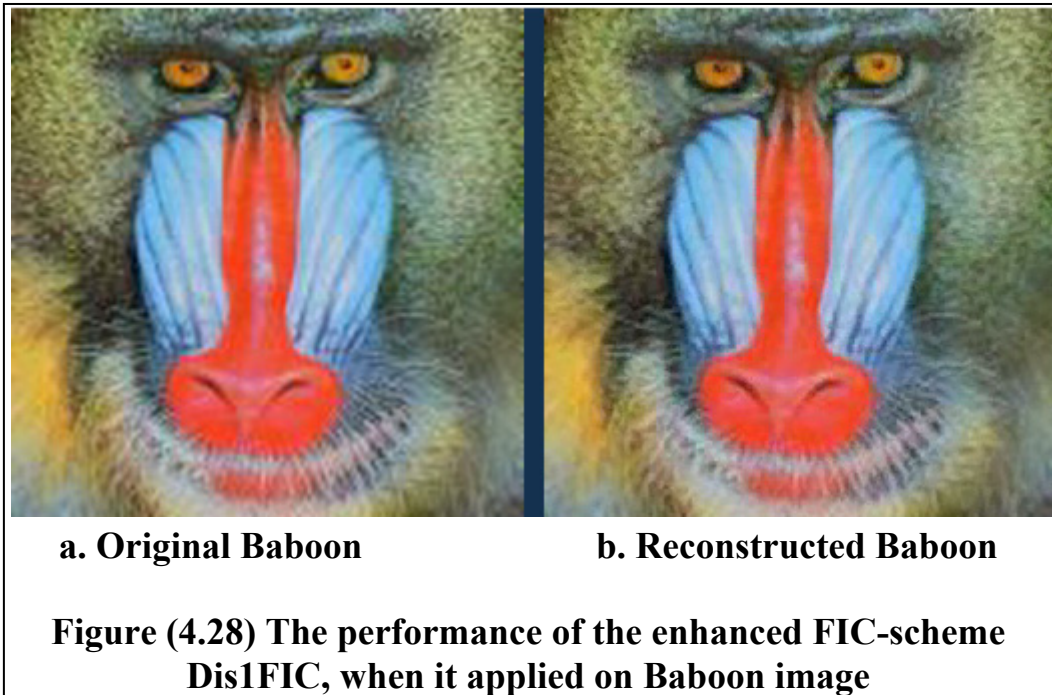
Table (4.12) lists the values of performance parameters of Dis1FIC scheme, when it is applied on Lena image using different values of both no. of bins and window size parameters.

Table (4.12) The effect of both no. of bins and window size parameters on performance parameters of Dis1FIC scheme

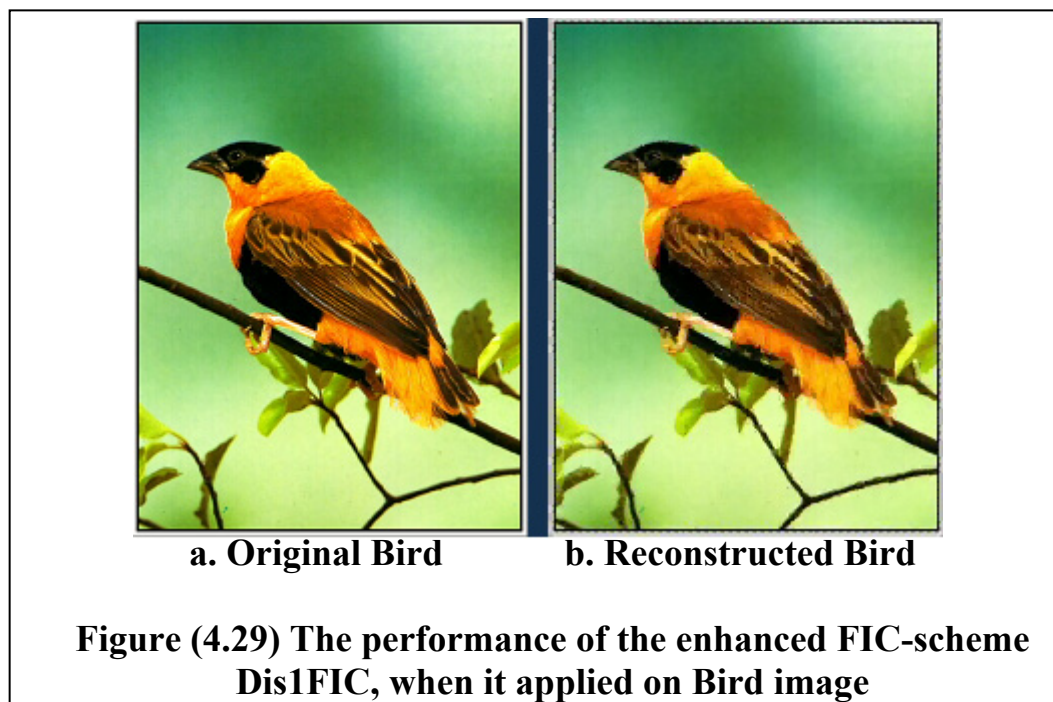
| No. of Bins | Window Size | Over All | | | Compression Ratio | Bit Rate | Time in Seconds |
|-------------|-------------|----------|-------|-------|-------------------|----------|-----------------|
| | | MAE | MSE | PSNR | | | |
| 100 | 1 | 3.92 | 42.01 | 31.90 | 8.875 | 2.704 | 1.32 |
| | 2 | 3.79 | 38.54 | 32.27 | 8.877 | 2.703 | 1.70 |
| | 3 | 3.71 | 36.68 | 32.49 | 8.874 | 2.705 | 2.16 |
| | 4 | 3.67 | 35.69 | 32.61 | 8.871 | 2.705 | 2.82 |
| | 5 | 3.64 | 35.05 | 32.68 | 8.873 | 2.705 | 3.10 |
| 250 | 1 | 4.17 | 48.25 | 31.30 | 8.865 | 2.707 | 0.57 |
| | 2 | 4.03 | 44.71 | 31.63 | 8.869 | 2.706 | 0.95 |
| | 3 | 3.94 | 42.16 | 31.88 | 8.869 | 2.706 | 1.07 |
| | 4 | 3.87 | 40.57 | 32.05 | 8.872 | 2.705 | 1.45 |
| | 5 | 3.82 | 39.40 | 32.18 | 8.873 | 2.705 | 1.45 |
| 400 | 1 | 4.34 | 53.40 | 30.86 | 8.867 | 2.707 | 0.53 |
| | 2 | 4.16 | 48.09 | 31.31 | 8.870 | 2.706 | 0.64 |
| | 3 | 4.06 | 45.45 | 31.56 | 8.870 | 2.706 | 0.73 |
| | 4 | 4.06 | 45.45 | 31.56 | 8.870 | 2.706 | 0.94 |
| | 5 | 3.95 | 42.40 | 31.86 | 8.870 | 2.706 | 1.01 |
| 550 | 1 | 4.47 | 57.02 | 30.57 | 8.864 | 2.708 | 0.42 |
| | 2 | 4.27 | 51.21 | 31.04 | 8.866 | 2.707 | 0.48 |
| | 3 | 4.16 | 48.03 | 31.32 | 8.871 | 2.705 | 0.62 |
| | 4 | 4.09 | 46.18 | 31.49 | 8.872 | 2.705 | 0.68 |
| | 5 | 4.04 | 44.70 | 31.63 | 8.870 | 2.706 | 0.80 |
| 700 | 1 | 4.59 | 60.67 | 30.30 | 8.866 | 2.707 | 0.41 |
| | 2 | 4.36 | 53.59 | 30.84 | 8.867 | 2.707 | 0.47 |
| | 3 | 4.23 | 50.16 | 31.13 | 8.867 | 2.707 | 0.62 |
| | 4 | 4.16 | 48.03 | 31.32 | 8.869 | 2.706 | 0.70 |
| | 5 | 4.10 | 46.49 | 31.46 | 8.872 | 2.705 | 0.66 |
| 850 | 1 | 4.68 | 63.18 | 30.13 | 8.859 | 2.709 | 0.42 |
| | 2 | 4.45 | 56.57 | 30.60 | 8.871 | 2.706 | 0.54 |
| | 3 | 4.31 | 52.07 | 30.96 | 8.864 | 2.708 | 0.46 |
| | 4 | 4.22 | 49.74 | 31.16 | 8.867 | 2.707 | 0.64 |
| | 5 | 4.15 | 47.62 | 31.35 | 8.870 | 2.706 | 0.75 |
| 1000 | 1 | 4.77 | 66.21 | 29.92 | 8.858 | 2.709 | 0.38 |
| | 2 | 4.51 | 58.39 | 30.47 | 8.864 | 2.708 | 0.41 |
| | 3 | 4.38 | 54.58 | 30.76 | 8.864 | 2.708 | 0.42 |
| | 4 | 4.27 | 51.15 | 31.04 | 8.867 | 2.707 | 0.65 |
| | 5 | 4.21 | 49.38 | 31.20 | 8.870 | 2.706 | 0.55 |

4.14 Implementing Dis1FIC on Different Images

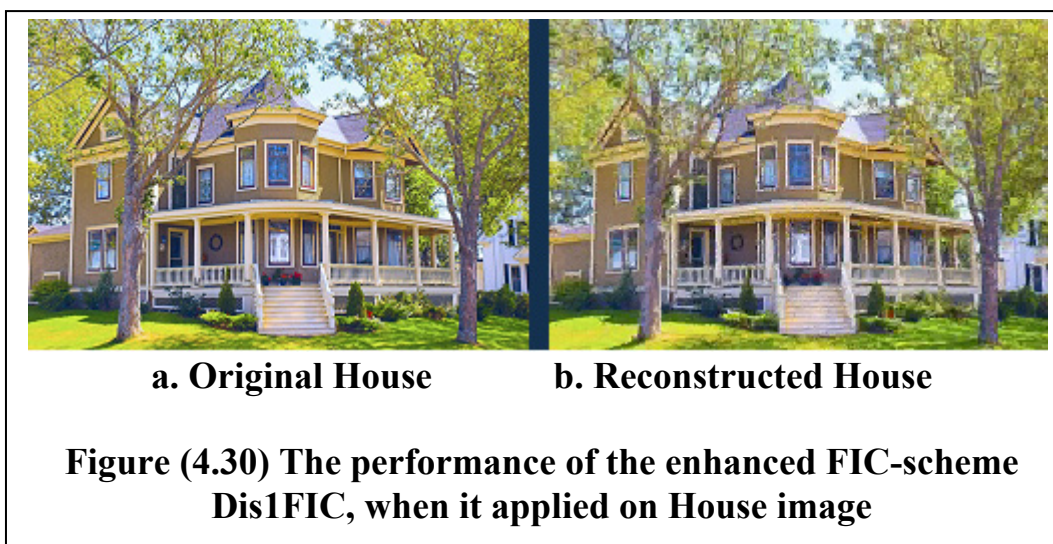
1. Figure (4.28) shows the effect of Dis1FIC scheme when it implemented on Baboon image. The results were: (MAE=3.92, MSE=31.98, PSNR=33.08, CR=8.998, BR=2.667, Time in seconds=1.30).



2. Figure (4.29) shows the original and reconstructed Bird image when applying Dis1FIC scheme. The results were: (MAE=7.15, MSE=202.46, PSNR=25.07, CR=8.786, BR=2.731, Time in seconds=0.81).

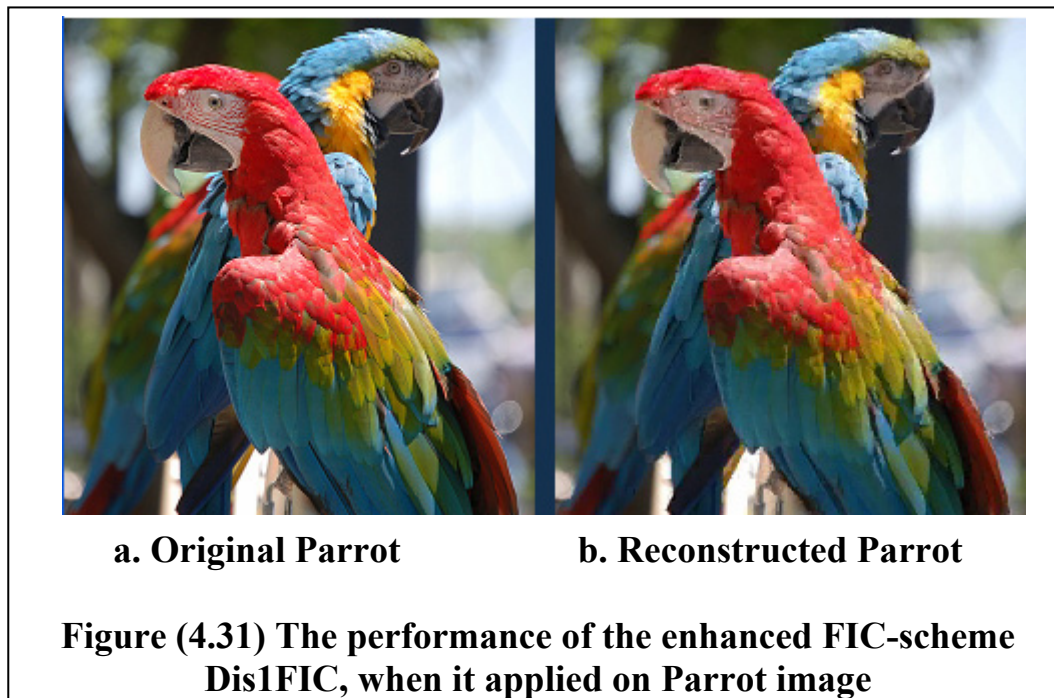


3. Figure (4.30) shows the performance of Dis1FIC scheme when it applied on House image. The results were: (MAE= 18.91, MSE=701.34, PSNR=19.67, CR=8.625, BR=2.782, Time in seconds=0.78).



4. Figure (4.31) shows original and reconstructed images of Parrot when implementing Dis1FIC scheme on it. The results were: (MAE=6.49,

MSE=135.81, PSNR= 26.80, CR=8.526, BR=2.815, Time in seconds=0.93).



4.15 Discussion

1. The increase in the block length parameter causes an increase in Cr and a decrease in PSNR.
2. The increase in jump step causes a decrease in PSNR and elapsed encoding time, and an increase in Cr value.
3. The increase in maximum scale causes little increase in compression ratio value, and a decrease in PSNR.
4. The increase in the number of bits used to encode the scale and offset coefficients causes a decrease in the value of Cr and an increase in PSNR value of the decompressed image.
5. The increase in the value of minimum error and minimum block error parameters causes a little increase in Cr value, and a decrease in PSNR value and encoding time.

6. The increase in the number of bins values causes a decrease in encoding time, and PSNR, and a little decrease in Cr value.
7. The increase in window size parameter causes an increase in time and PSNR, and a small decrease in compression ratio.
8. In general, the two enhanced FIC-schemes (i.e., Dis1FIC and Dis3FIC) have encoding time much less than that spend by the traditional FIC-scheme (i.e., TradFIC) and by the enhanced scheme (i.e., PredFIC) which is based on using the symmetry predictor only.



Chapter Five

Conclusions and Suggestions

Chapter Five

Conclusions and Future Works

5.1 Conclusions

From the test results presented in previous chapter, some remarks related to the behavior and performance of the investigated FIC schemes are stimulated. Among these remarks are the followings:

1. The use of symmetry predictor causes a speed-up in encoding process, and the use of moment description causes more significant speeding-up in encoding process. But, the image quality had little degraded in comparison with its level when the traditional method is applied.
2. The block length parameter mainly affects on the (MAE, MSE, PSNR, Cr, BR, and time). It was found that the suitable values are (4, 5) which led to good PSNR (less distortion) and Cr.
3. When the jump step is taken small then the schemes have better performance in terms of image quality and compression ratio. But if the jump step is set large, the encoding process become faster. The best values of jump step are (1, 2, 3).
4. The use of maximum scale parameter value within the range [2.5, 3] will nearly preserve the image quality but with a little bit degradation. The encoding time is little affected with the variation of maximum scale values.
5. The value of the parameters (scale bits and offset bits) affects the performance of the FIC scheme, where the increase in the number of encoding bits leads to less error and good quality but a little bit decrease in compression ratio. But the compression time

will little affected. The suitable values of these parameters are (7, 8).

6. The minimum error and minimum block error approximately have little effect on compression ratio, while they have significant effect on PSNR and encoding time. When they are increased, the PSNR value is decreased, it has a suitable value if these parameters' values are set (from 1 to 5) in case of using descriptor order-1, but these values could be expanded to 7 in case of descriptor order-3.
7. The use of large no. of bins with larger window size will lead to suitable compression ratio and PSNR, and it speeds up the compression time significantly.

5.2 Future Works

1. Other advanced partitioning schemes (such as HV, quadtree) can be implemented to enhance the compression performance parameter.
2. Using another window size scheme, like making it local adaptive according to the population of each bin and its surrounding neighbors.
3. Speeding up the descriptors using double blocks descriptors instead of single descriptor.



References

References

[Acha05]

Acharya, T. and Ray, A. K.; "Image Processing Principles and Applications"; Wiley-Interscience; United States of America; 2005.

[Alam01]

Al-A'mri, J. H.; "Fractal Image Compression"; Ph.D. Thesis; Baghdad University; College of Science; 2001.

[Aldu03]

Al-Dulaimy, A. A.; "Fractal Image Compression with Fastening Approaches"; M.Sc. Thesis; Al-Nahrain (Saddam) University; College of Science; 2003.

[Alhi07]

Al-Hilo, E. A.; "Speeding-up Fractal Colored Image Compression Using Moments Features"; Ph.D. Thesis; Al-Mustansiriyah University; College of Science; 2007.

[BaHu93]

Barnsley, M., and Hurd, L.; "Fractal Image Compression"; Wellesley; Massachusetts: AK Peter. Ltd.; 1993.

[BDM88]

Barnsley, M. F., Devaney, R. L., Mandelbrot, B. B., Peitgen, H. O., Saupe, D., Voss, R. F., Fisher, Y., and McGuire, M.; "The Science of Fractal Images"; Springer-Verlag; New York; 1988.

[Bour91]

Bourke, P., "An Introduction to Fractals", Internet paper, 1991, <http://astronomy.swin.edu.au/~pbourke/fractals/fracintro/>.

[ChKu00]

Chang, H.T., and Kuo, C.J.; "Iteration-free fractal image coding based on efficient domain pool design"; Dept. of Inf. Manage., Chao Yang Univ. of Technol., Taichung; IEEE Transactions on Image Processing; vol. 9; pp. 329-339; 2000.

[Colv96]

Colvin, J.; "Iterated Function Systems and Fractal Image Compression"; 1996; kd4syw@usit.net.

[Cran97]

Crane, R.; "A Simplified Approach to Image Processing"; United States of America; New Jersey; Hewlett-Packard Company; 1997.

[DNR06]

Distasi, R., Nappi, M., and Riccio, D.; "A Range/Domain Approximation Error-Based Approach for Fractal Image Compression"; IEEE Transactions on Image Processing; vol. 15; pp. 89- 97; 2006.

[Fish95]

Fisher, Y.; "Fractal Image Compression"; Springer-Verlag; United States of America; New York; 1995.

[Geor06]

George, L. E.; "IFS Coding for Zero-Mean Image Blocks"; University of Baghdad; College of Science; Iraqi Journal of Science; vol.47; no.1; 2006.

[Ghan03]

Ghanbari, M.; "Standard Codecs: Image Compression to Advanced Video Coding"; Magazine of the Institution of Electrical Engineers; London; United Kingdom; 2003.

[Gonz02]

Gonzalez, R., and Woods, R.; " Digital Image Processing"; Pearson Education International; Prentice Hall; Inc.; 2nd Edition; New Jersey; 2002.

[Klin03]

Klinger, T.; "Image Processing with LabVIEW and IMAQ Vision"; Prentice Hall PTR; United States of America; 2003.

[Lani04]

Lanins, C., "Fractals", A Fractal Unit for Elementary and Middle School Unit, 2004, <http://math.rice.edu/~Lanins/fractals/self.html>.

[Mahm07]

Mhamood, R. F.; "Improved Once-Time-Search Method Based on Inter-Block Correlation"; M.Sc. Thesis; Al-Nahrain University; College of Science; 2007.

[Mand04]

Mandelbrot, B., "Fractal", Internet paper, Microsoft ® Encarta ® online Encyclopedia, 2004, <http://Encarta.msn.com>.

[Moha03]

Mohamed, M.; "Optimization of Fractal Image Compression Based on Kohonen Neural Networks"; EEDIS Laboratory, Engineering Faculty, University of SBA; mohamedmokht@yahoo.fr; Internet paper; <http://www.eurasip.org/Proceedings/Ext/ISCCSP2006/defevent/papers/cr1051.pdf>; 2003.

[NeGa95]

Nelson, M. and Gailly, J. L.; "The Data Compression Book"; Prentice-Hall; Second Edition; England; 1995.

[Niki07]

Nikiel, S.; "Iterated Function Systems for Real-Time Image Synthesis"; Springer; Poland; 2007.

[Ning97]

Ning, L.; "Fractal Imaging"; Academic Press; 1997.

[Prat01]

Pratt, W. K.; "Digital Image Processing"; 3rd Edition; John Wiley and Sons; New York; 2001.

[ReAn97]

Rejeb, B., and Anheier, W.; "A New Approach for the Speed Up of Fractal Image Coding"; 13th International Conference on Digital Signal Processing Proceedings; vol.2; pp. 853-856; 1997.

[RHS97]

Ruhl, M., Hartenstein, H. and Saupe, D.; "Adaptive Partitionings for Fractal Image Compression"; IEEE International Conference on Image Processing (ICIP'97); Santa Barbara; ruhl, hartenst, saupe@informatik.uni-freiburg.de; 1997.

[Salo04]

Salomon, D.; "Data Compression"; Springer-Verlag; United States of America; New York; 3rd Edition; 2004.

[Salo07]

Salomon, D.; "Data Compression"; Springer; United Kingdom; London; 4th Edition; 2007.

[Sank98]

Sankaranarayanan, V.; "Fractal Image Compression Literature Survey"; Internet Paper; 1998.

[SaSm99]

Salih, I. and Smith, S. H.; "Encoding Time Reduction in Fractal Image Compression"; IEEE Computer Society; Washington, DC, USA; 1999.

[Sayo06]

Sayood, K.; "Introduction to Data Compression"; 3rd Edition; United States of America; Elsevier; 2006.

[ShSu00]

Shi, Y. Q., and Sun, H.; "Image and Video Compression for Multimedia Engineering"; CRC Press LLC; United States of America; 2000.

[TaLo98]

Taylor, M., and Louret, J.; "Scientific Fractals FAQ", 1998, <http://www.mta.ca/~mctaylor/sci.fractals-faq/>.

[ToPi01]

Tong, C. S., and Pi, M.; "Fast Fractal Image Encoding Based on Adaptive Search". IEEE Transactions on Image Processing; vol. 10; no. 9; pp. 1269-1277; 2001.

[ToWo02]

Tong, C. S., and Wong, M.; "Adaptive Approximate Nearest Neighbor Search for Fractal Image Compression"; Dept. of Math., Hong Kong Baptist Univ., Kowloon; IEEE Transactions on Image Processing; vol. 11; pp. 605-615; 2002.

[Umba98]

Umbaugh, S. E.; "Computer Vision and Image Processing: A Practical Approach using CVIP Tools"; Prentice Hall, Inc.; 1998.

[WBG03]

West, B. J., Bologna, M., and Grigolini, P.; "Physics of Fractal Operators"; Springer-Verlag; New York; 2003.

[WoJa94]

Wohlberg, B., and Jager, G.; "On the Reduction of Fractal Image Compression Encoding Time"; IEEE South African Symposium on Communications and Signal Processing (COMSIG '94); pp. 158-161; 1994.

[WoJa99]

Wohlberg, B. and Jager, G.; "A Review of the Fractal Image Coding Literature"; IEEE Transactions on Image Processing; vol. 8, no. 12; pp. 1716-1729; 1999.

[Yung05]

Yung-Gi Wu; "Fast and Low Bit Rate Fractal Image Encoding"; Journal Paper; SPIE Digital Library © 2008; Optical Engineering vol. 44; no. 11; 2005.

الخلاصة

لقد استحدثت طرائق عديدة لضغط الصور باستخدام تقانات مختلفة، وكان الغرض منها تحقيق نسبة ضغط عالية مع المحافظة على جودة الصورة المضغوطة وانجاز الضغط بأقل وقت ممكن. إن طريقة الضغط الكسوري هي إحدى هذه الطرق وهي تقنية حديثة لضغط الصور تعتمد على مبدأ التشابه الذاتي في الصور.

إن البحث الحالي يهتم بتطوير نظام الضغط الكسوري. وهو يتكون من مرحلتين أساسيتين، الأولى هي مرحلة التشفير والثانية مرحلة فك التشفير. في مرحلة التشفير تجزأ الصورة الأصلية إلى نوعين من المقاطع، الأولى تدعى مقاطع المدى وهي مقاطع غير متداخلة، والثانية تدعى مقاطع المنطلق ومن الممكن أن تكون متداخلة.

تجزأ الصورة باستخدام طريقة البلوكات المتساوية الحجم وبعد ذلك يتم إيجاد أفضل بلوك منطلق لكل بلوك من بلوكات المدى وذلك بتطبيق تحويل أفين. تنتهي مرحلة التشفير بخزن معاملات التحويل لكل بلوك من بلوكات المدى. إن عملية إيجاد البلوكات المتشابهة تتطلب عملية حسابية معقدة تستغرق وقتاً طويلاً وهذه إحدى سلبيات الضغط الكسوري. أما مرحلة فك التشفير فتطبق على أي صورة ابتدائية حتى نحصل على الصورة المسترجعة، وهذا يستغرق وقتاً قصيراً.

في هذا البحث تم تصميم وتطبيق أربعة نماذج لتحسين مراحل التشفير للصور الملونة واختزال الوقت المستغرق. في الهيكل الأول تم تطبيق طريقة الضغط الكسوري التقليدية على الصور الملونة وذلك بتحويل المركبات (RGB) إلى (YCbCr). وقد استغرقت عملية الضغط 144.02 ثانية، وكانت نسبة الضغط المتحققة 8.89، وبلغت جودة الصورة حوالي 33.39 ديسيل.

في الهيكل الثاني هو طريقة الضغط الكسوري للصور مع استخدام متنبية التحويل التناظري، ويعتمد المتنبية أساساً على قيم العزوم للمقاطع التي سيتم مطابقتها. إن استخدام المتنبية سيققل من عدد عمليات التناظر (الثمانية) التي تستخدم في الطريقة التقليدية ليجعلها عملية تناظر واحدة وهذه الطريقة أدت إلى تقليل وقت الضغط ليصبح حوالي 14% من وقت الضغط للطريقة التقليدية.

في الهيكل الثالث والرابع استحدثت طريقة الضغط الكسوري للصور محسنة من خلال استخدام واصف للمقاطع يعتمد على العزوم من الدرجة الأولى والثالثة، فقد استخدمت لوصف المقاطع لعمل فهرسة لبلوكات المنطلق، وبالنتيجة ليختزل معادلات البحث عن أفضل بلوك منطلق شبيه لكل بلوك من بلوكات المدى. إن نتائج فحص هذين الهيكلين أشارت إلى حصول اختزال كبير في وقت البحث وبالتالي أدى إلى نقصان وقت الضغط ليصل إلى 0.9% من وقت الضغط بالطريقة التقليدية.



جمهورية العراق
وزارة التعليم العالي والبحث العلمي
جامعة النهرين
كلية العلوم

متبىء تناظر البلوك لتحسين ضغط الصور الكسوري

رساله

مقدمه إلى كلية العلوم في جامعة النهرين كجزء من
متطلبات نيل درجة الماجستير في علوم الحاسبات

من قبل

رؤى عبدالله جابر

(بكالوريوس جامعة النهرين 2006)

إشراف

د. لؤى أدور جورج