Image Compression Using Fractional Fourier Transform

A Thesis

Submitted in the partial fulfillment of requirement for the award of the degree of

Master of Engineering
In
Electronics and Communication

by

Parvinder Kaur

Regn. No. 8024116

Under the guidance of

Mr. R.K Khanna
Assistant Professor
Department of Electronics & Communication Engineering

Mr. Kulbir Singh
Lecturer
Department of Electronics & Communication Engineering

Department of Electronics & Communication Engineering
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
(Deemed University)
PATIALA-147004

ABSTRACT

The rapid growth of digital imaging applications, including desktop publishing, multimedia, teleconferencing, and high-definition television (HDTV) has increased the need for effective and standardized image compression techniques. The purpose of image compression is to achieve a very low bit rate representation, while preserving a high visual quality of decompressed images. It has been recently noticed that the fractional Fourier transform (FRFT) can be used in the field of image processing. The significant feature of fractional Fourier domain image compression benefits from its extra degree of freedom that is provided by its fractional orders 'a'.

The fractional Fourier transform is a time-frequency distribution and an extension of the classical Fourier transform. The FRFT depends on a parameter 'a' can be interpreted as a rotation by an angle $\alpha=a\pi/2$ in the time–frequency plane. An FRFT with $\alpha=\pi/2$ corresponds to the classical Fourier Transform, and an FRFT with $\alpha=0$ corresponds to identity operator.

In the present study, the FRFT, which is generalization of Fourier transform, is used to compress the image with variation of its parameter 'a'. It is found that by using FRFT, high visual quality decompressed image can be achieved for same amount of compression as that for Fourier transform. By adjusting 'a' to different values, FRFT can achieve low mean square error (MSE), better peak signal to noise ratio (PSNR), a high compression ratio (CR), while preserving good fidelity of decompressed image. By varying 'a', it can achieve high CR even for same cutoff. As cutoff increases, CR increases but image quality degrades since there is tradeoff between image quality and CR.

CHAPTER –1
INTRODUCTION

Now a days, the usage of digital image in various applications is growing rapidly. Video and television transmission is becoming digital and more and more digital image sequences are used in multimedia applications.

A digital image is composed of pixels, which can be thought of as small dots on the screen and it becomes more complex when the pixels are colored. An enormous amount of data is produced when a two dimensional light intensity function is sampled and quantized to create a digital image. In fact, the amount of data generated may be so great that it results in impractical storage, processing and communications requirements [1].

1.1 Fundamentals of Digital Image

An image is a visual representation of an object or group of objects. When using digital equipment to capture, store, modify and view photographic images, they must first be converted to a set of numbers in a process called digitization or scanning. Computers are very good at storing and manipulating numbers, so once the image has been digitized it can be used to archive, examine, alter, display, transmit, or print photographs in an incredible variety of ways. Each pixel of the digital image represents the color (or gray level for black & white images) at a single point in the image, so a pixel is like a tiny dot of a particular color. By measuring the color of an image at a large number of points, we can create a digital approximation of the image from which a copy of the original image can be reconstructed. Pixels are a little grain like particles in a conventional photographic image, but arranged in a regular pattern of rows and columns [1,2]. A digital image is a rectangular array of pixels sometimes called a bitmap. It is represented by an array of N rows and M columns and usually N=M. Typically values of N and M are 128, 256, 512 and 1024 etc.

1.2 Types of Digital Image

For photographic purposes, there are two important types of digital images: color and black & white. Color images are made up of colored pixels while black & white images are made of pixels in different shades of gray.

1.2.1 Black & White Images

A black & white image is made up of pixels, each of which holds a single number corresponding to the gray level of the image at a particular location. These gray levels span the full range from black to white in a series of very fine steps, normally 256 different grays [1]. Assuming 256 gray levels, each black and white pixel can be stored in a single byte (8 bits) of memory.

1.2.2 Color Images

A color image is made up of pixels, each of which holds three numbers corresponding to the red, green and blue levels of the image at a particular location. Assuming 256 levels, each color pixel can be stored in three bytes (24 bits) of memory. Note that for images of the same size, a black & white version will use three times less memory than a color version.

1.2.3 Binary Images

Binary images use only a single bit to represent each pixel. Since a bit can only exist in two states- ON or OFF, every pixel in a binary image must be one of two colors, usually black or white. This inability to represent intermediate shades of gray is what limits their usefulness in dealing with photographic images.

1.3 Image Compression

1.3.1 Need for compression

The following example illustrates the need for compression of digital images.

- To store a color image of a moderate size, e.g. 512×512 pixels, one needs 0.75 MB of disk space.
- A 35mm digital slide with a resolution of 12 μm requires 18 MB.
- One second of digital PAL (Phase Alternation Line) video requires 27 MB.

To store these images, and make them available over network (e.g. the internet), compression techniques are needed. Image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is the removal of redundant data. According to mathematical point of view, this amounts to transforming a two-dimensional pixel array into a statistically uncorrelated data set. The transformation is applied prior to storage or transmission of the image. At receiver, the compressed image is decompressed to reconstruct the original image or an approximation to it. The initial focus of research efforts in this field was on the development of analog methods for reducing video transmission bandwidth, a process called bandwidth compression. The advent of digital computer and subsequent development of advanced integrated circuits, however, caused interest to shift from analog to digital compression approaches. With the recent adoption of several key international image compression standards, the field is now poised for significant growth through the practical application of the theoretical work that began in the 1940s, when

C.E. Shannon and others first formulated the probabilistic view of information and its representation, transmission, and compression. The example below clearly shows the importance of compression [1].

An image, 1024 pixel×1024 pixel×24 bit, without compression, would require 3 MB of storage and 7 minutes for transmission, utilizing a high speed, 64 kbits/s, ISDN line. If the image is compressed at a 10:1 compression ratio, the storage requirement is reduced to 300 KB and the transmission time drop to less than 6 seconds.

1.3.2 Principle behind compression

A common characteristic of most images is that the neighboring pixels are correlated and therefore contain redundant information. The foremost task then is to find less correlated representation of the image. Two fundamental components of compression are redundancy and irrelevancy reduction.

Redundancies reduction aims at removing duplication from the signal source (image/video).

Irrelevancy reduction omits parts of the signal that will not be noticed by the signal receiver, namely the Human Visual System.

In an image, which consists of a sequence of images, there are three types of redundancies in order to compress file size. They are:

- **Coding redundancy:** Fewer bits to represent frequent symbols.
- **Interpixel redundancy:** Neighboring pixels have similar values.
- **Psychovisual redundancy:** Human visual system cannot simultaneously distinguish all colors.

1.3.3 Types of compression

Compression can be divided into two categories, as Lossless and Lossy compression. In lossless compression, the reconstructed image after compression is numerically identical

to the original image. In lossy compression scheme, the reconstructed image contains degradation relative to the original.

In the case of video, compression causes some information to be lost; some information at a detail level is considered not essential for a reasonable reproduction of the scene. This type of compression is called **lossy compression**. Audio compression on the other hand, is not lossy, it is called **lossless compression**. An important design consideration in an algorithm that causes permanent loss of information is the impact of this loss in the future use of the stored data.

Lossy technique causes image quality degradation in each compression/decompression step. Careful consideration of the human visual perception ensures that the degradation is often unrecognizable, though this depends on the selected compression ratio. In general, lossy techniques provide far greater compression ratios than lossless techniques.

The following are the some of the lossless and lossy data compression techniques [1]:

- **Lossless coding techniques**
  - Run length encoding
  - Huffman encoding
  - Arithmetic encoding
  - Entropy coding
  - Area coding
- **Lossy coding techniques**
  - Predictive coding
  - Transform coding (FT/DCT/Wavelets)

1.3.4 Applications

Over the years, the need for image compression has grown steadily. Currently it is recognized as an "enabling technology." It plays a crucial role in many important and diverse applications [1,2] such as:

- Business documents, where lossy compression is prohibited for legal reasons.

- Satellite images, where the data loss is undesirable because of image collecting cost.
- Medical images, where difference in original image and uncompressed one can compromise diagnostic accuracy.
- Televideoconferencing.
- Remote sensing.
- Space and hazardous waste control applications.
- Control of remotely piloted vehicles in military.
- Facsimile transmission (FAX).

Image compression has been and continues to be crucial to the growth of multimedia computing. In addition, it is the natural technology for handling the increased spatial resolutions of today's imaging sensors and evolving broadcast television standards.

1.4 Fractional Fourier transform

1.4.1 Fractional operations

Going from the whole of an entity to fractions of it represents a relatively major conceptual leap. The fourth power of 3 may be defined as 3^4= 3x3x3x3, but it is not obvious from this definition how one might define $3^{3.5}$. It must have taken sometime before the common definition $3^{3.5} = 3^{7/2} = \sqrt{3^7}$ emerged. The first and second derivatives of the function $f(x)$ are commonly denoted by:

$\frac{df(x)}{dx}$ and $\frac{d^2 f(x)}{dx} = \frac{d}{dx}\left[\frac{df(x)}{dx}\right] = \frac{d[df(x)/dx]}{dx} = \left(\frac{d}{dx}\right)^2 f(x)$ respectively.

Higher order derivatives are defined similarly. Now let us generalize this property by replacing n with the real order '**a**' and take it as the ath derivative of $f(x)$. Thus to find $\frac{d^a f(x)}{dx^a}$, the ath derivative of $f(x)$, find the inverse Fourier transform of $(i2\pi\mu)^a$ F (μ).

In both of these examples we are dealing with the fractions of an operation performed on an

entity, rather than fractions of the entity itself. $2^{0.5}$ is the square root of the integer 2. The function $[f(x)]^{0.5}$ is the square root of the function $f(x)$. But $\frac{d^{0.5} f(x)}{dx^{0.5}}$ is the 0.5th derivative of $f(x)$ with $\left(\frac{df(x)}{dx}\right)^{0.5}$ being the square root of the derivative operator $\frac{d}{dx}$. The process of going from the whole of an entity to fractions of it underlies several of the more important conceptual developments. e.g. fuzzy logic, where the binary 1 & 0 are replaced by continuous values representing our certainty or uncertainty of a proposition.

1.4.2 Historical Development of FRFT

The FRFT, which is a generalization of the ordinary Fourier transform (FT), was introduced 75 years ago, but only in the last two decade it has been actively applied in signal processing, optics and quantum mechanics. The Fourier Transform (FT) is undoubtedly one of the most valuable and frequently used tools in signal processing and analysis. Little need be said of the importance and ubiquity of the ordinary Fourier transform in many areas of science and engineering. A generalization of Fourier Transform- the Fractional Fourier Transform (commonly referred as FRFT in available literature) was introduced in **1980** by **Victor Namias** [3] and it was established in the same year that the other transforms could also be fractionalized [4]. **McBride and Keer** explored the refinement and mathematical definition in **1987** [5]. In a very short span of time, FRFT has established itself as a powerful tool for the analysis of time varying signals [6,7]. Furthermore, a general definition of FRFT for all classes of signals (one-dimensional & multidimensional, continuous & discrete and periodic & non-periodic) was given by **Cariolario** *et al.* in. But when FRFT is analyzed in discrete domain there are many definitions of Discrete Fractional Fourier Transform (DFRFT)[8-10]. It is also established that none of these definitions satisfies all the properties of continuous FRFT. **Santhanam and McClellan** first reported the work on DFRFT in 1995. Thereafter within a short span of time a lot many definitions of DFRFT came into existence and these

definitions are classified according to the methodology used for calculations in **2000** by Pie *et al*.

1.4.3 Applications

The FRFT has been found to have several applications in the areas of optics and signal processing and it also lead to generalization of notion of space (or time) and frequency domains which are central concepts of signal processing. FRFT has been related to a certain class of wavelet transforms, to neural networks, and has also inspired the study of the fractional versions of many other transforms employed in signal analysis and processing.

FRFT has many applications in solution of differential equations [3,4,8], optical beam propagation and spherical mirror resonators, optical diffraction theory, quantum mechanics, statistical optics, optical system design and optical signal processing, signal detectors, correlation and pattern recognition, space or time variant filtering, multiplexing, signal recovery, restoration and enhancement, study of space or time–frequency distributions [14] etc.

It is believed that these are only a fraction of the possible applications. Despite the fact that most of the publications in this area have so far appeared in mathematics, optics, and signal processing journals, it is believed that the Fractional Fourier transform will have a significant impact also in other areas of science and engineering where Fourier concepts are used.

1.5 Objective of thesis

The main aims of thesis are:

- To achieve image compression using a novel technique i.e. Fractional Fourier Transform.

- To compare the performance of this new transform with existing transform i.e Fourier transform.
- To analyses the amount of compression that can be achieved by varying cutoff at constant 'a'.
- To study the effect of variation of parameter 'a' on MSE, PSNR for same CR.
- To study the effect of variation of both 'a' and CR on performance metrics.
- To study the effect of variation of 'a' on CR for different cutoff.
- Analyses the variation of PSNR, MSE with CR.
- To reduce the blocking artifacts by variation of parameter 'a'.
- Comparison of performance parameters i.e. MSE, PSNR, CR for different images.

1.6 Organisation of Thesis

In chapter 2 the various types of data redundancies i.e. Coding redundancy, Interpixel redundancy, Psychovisual redundancy present in image are discussed.

In chapter 3 the basic image compression techniques i.e. Lossless compression and Lossy compression techniques are presented. A general compression model is also discussed.

Chapter 4 describes the Fractional Fourier Transform. The mathematical definition and properties of Fractional Fourier Transforms have been discussed. Algorithm for fast computation of fractional Fourier transform is also discussed. The introduction to discrete Fractional Fourier Transform has been provided.

Chapter 5 describes the image compression using fractional Fourier transform.

Chapter 6 gives the introduction to MATLAB tool.

Chapter 7 provides the simulation results of four images i.e. Lenna, Cameraman, Barbara, Rice with the variation of parameter 'a' in the Fractional Fourier domain.

Chapter 8 enlists the important conclusions and prospects of the future work

CHAPTER –2

TYPES OF DATA REDUNDANCY

2.1 Fundamentals

The term data compression refers to reducing amount of data required to represent a given amount of information. There should be a clear distinction between data and information. In fact, data are the form of information representation. Thus the same information may be represented by completely different data. If certain information has two representations differing in size, one of them is said to have data redundancy [1]. The data redundancy is a quantifiable entity. If n_1 and n_2 are the number of information units in two data sets representing the same information, the relative data redundancy R_D of the first data set (the one characterized by n_1) is defined as

$$R_D = 1 - \frac{1}{C_R} \tag{2.1}$$

where C_R commonly called the *compression ratio*, is

$$C_R = \frac{n_1}{n_2} \tag{2.2}$$

For the case $n_2 = n_1, C_R = 1$ and $R_D = 0$ indicating that (relative to the second data set) the first representation of the information contains no redundant data.

When $n_2 << n_1, C_R$ and R_D, implying significant compression and highly redundant data. In the final case, $n_2 >> n_1, C_R$ and R_D, indicating that the second data set contains much more data than the original representation. This is of course, is the normally undesirable case of data expansion. In general C_R and R_D lie in the open intervals (0,∞) and (-∞, 1) respectively. A compression ratio 8:1 means that the first data set has 8 information carrying units per every 1 unit in the second or compressed data set. The corresponding redundancy of 0.875 implies that 87.5 percent of the data in the first data set is redundant.

2.2 Various Types of Data Redundancy

In digital image compression, three basic data redundancies can be identified and exploited:

- Coding redundancy
- Interpixel redundancy
- Psychovisual redundancy

Data compression is achieved when one or more of these redundancies are reduced or eliminated.

2.2.1 Coding Redundancy

A gray level image having n pixels is considered. The number of gray levels in the image is L (i.e. the gray levels range from 0 to *L*-1) and the number of pixels with gray level r_k is n_k. Then the probability of occurring gray level r_k is $p_r(r_k)$. If the number of bits used to represent the gray level r_k is $l(r_k)$, then the average number of bits required to represent each pixel is.

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k) \tag{2.3}$$

where

$$p_r(r_k) = \frac{n_k}{n} \quad , \quad \text{k= 0, 1, 2,……..., L-1} \tag{2.4}$$

Hence the number of bits required to represent the whole image is n x L_{avg}. Maximal compression ratio is archived when L_{avg} is minimized (i.e. when $l(r_k)$, the length of gray level representation function, leading to minimal L_{avg} , is found). Coding the gray levels in such a way that the L_{avg} is not minimized results in an image containing coding redundancy. Generally coding redundancy is presented when the codes (whose lengths are represented here by $l(r_k)$ function) assigned to a gray levels don't take full advantage of gray level's probability ($p_r(r_k)$ function). Therefore it almost always presents when an image's gray levels are represented with a straight or natural binary code [1]. A natural binary coding of

their gray levels assigns the same number of bits to both the most and least probable values, thus failing to minimize equation 2.3 and resulting in **coding redundancy.**

Example of Coding Redundancy

An 8-level image has the gray level distribution shown in table 2.1. If a natural 3-bit binary code (see code 1 and $l_1(r_k)$ in table 2.1) is used to represent 8 possible gray levels, L_{avg} is 3-bits, because $l_1(r_k)$ = 3 bits for all r_k. If code 2 in table 2.1 is used, however the average number of bits required to code the image is reduced to:

L_{avg} = 2(0.19) + 2(0.25) + 2(0.21) + 3(0.16) + 4(0.08) + 5(0.06) + 6(0.03) + 6(0.02) = 2.7 bits.

From equation (2.2), the resulting compression ratio C_R is 3/2.7 or 1.11.Thus approximately 10% of the data resulting from the use of code 1 is redundant. The exact level of redundancy can be determined from equation (2.1).

Table 2.1: Example of Variable Length Coding

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
r_0 = 0	0.19	000	3	11	2
r_1 = 1/7	0.25	001	3	01	2
r_2 = 2/7	0.21	010	3	10	2
r_3 = 3/7	0.16	011	3	001	3
r_4 = 4/7	0.08	100	3	0001	4
r_5 = 5/7	0.06	101	3	00001	5
r_6 = 6/7	0.03	110	3	000001	6
r_7 = 1	0.02	111	3	000000	6

$$R_D = 1 - \frac{1}{1.11} = 0.099 = 9.9\%$$

It is clear that 9.9% data in first data set is redundant which is to be removed to achieve compression.

Reduction of coding redundancy (variable-length coding)

Figure 2.1 illustrates the underlying basis for the compression achieved by code 2. It shows both the histogram of the image [a plot of $p_r(r_k)$ versus r_k] and $l_2(r_k)$ because these two functions are inversely proportional, that is $l_2(r_k)$ increases as $p_r(r_k)$ decreases; the shortest code words in code 2 are assigned to the gray levels that occur most frequently in an image.

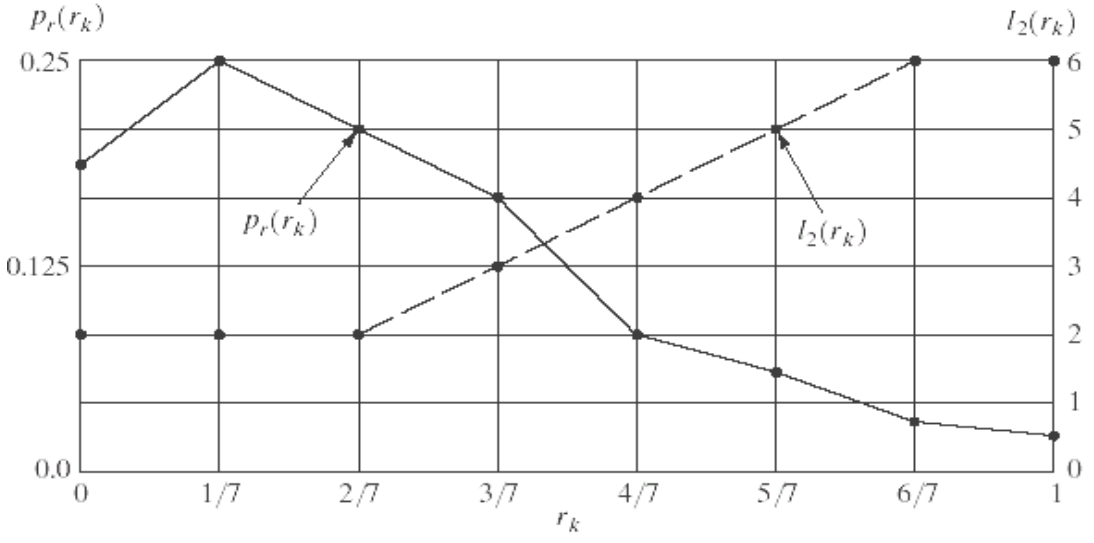


Figure 2.1. : Graphical representation of the fundamental basis of data compression through variable-length coding.

In the preceding example, assigning fewer bits to the more probable gray levels than to the less probable ones achieves data compression. This process commonly is referred to as variable length coding. There are several optimal and near optimal techniques for constructs such a code i.e. Huffman coding, Arithmetic coding etc.

2.2.2 Interpixel Redundancy

Another important form of data redundancy is interpixel redundancy, which is directly related to the interpixel correlations within an image. Because the value of any given pixel can be reasonable predicted from the value of its neighbours, the information carried by individual pixels is relatively small. Much of the visual contribution of a single pixel to an image is redundant; it could have been guessed on the basis of its neighbour's values. A variety of names, including *spatial redundancy, geometric redundancy*, and *interframe redundancy* have been coined to refer to these interpixel dependencies. In order to reduce the interpixel redundancies in an image, the 2-D pixel array normally used for human viewing and interpretation must be transformed into a more efficient but usually non-visual format. For example, the differences between adjacent pixels can be used to represent an image. Transformations of this type are referred to as *mappings*. They are called *reversible* if the original image elements can be reconstructed from the transformed data set [1,2].

Reduction of Interpixel Redundancy

To reduce the interpixel redundancy we use various techniques such as:

- Run length coding.
- Delta compression.
- Constant area coding.
- Predictive coding.

2.2.3 Psychovisual Redundancy

Human perception of the information in an image normally does not involve quantitative analysis of every pixel or luminance value in the image. In general, an observer searches for distinguishing features such as edges or textural regions and mentally combines them into recognizable groupings. The brain then correlates these groupings with prior knowledge in order to complete the image interpretation process. Thus eye does not respond with equal sensitivity to all visual information. Certain information simply has less relative importance than other information in normal visual processing. This information is said to be psychovisually redundant. It can be eliminated without significantly impairing the quality of image perception. Psychovisual redundancy is fundamentally different from the coding redundancy and interpixel redundancy [2].Unlike coding redundancy and interpixel redundancy, psychovisual redundancy is associated with real or quantifiable visual information. Its elimination is possible only because the information itself is not essential for normal visual processing. Since the elimination of psychovisual redundant data results in a loss of quantitative information. Thus it is an *irreversible process.*

Reduction of Psychovisual Redundancy

To reduce psychovisual redundancy we use Quantizer. Since the elimination of psychovisually redundant data results in a loss of quantitative information. It is commonly referred to as quantization. As it is an irreversible operation (visual information is lost) quantization results in lossy data compression.

CHAPTER –3

IMAGE COMPRESSION TECHNIQUES

3.1 Introduction

Two general techniques for reducing the amount of data required to represent an image are Lossless compression and Lossy compression. In both of these techniques one or more redundancies as discussed in last chapter is removed. However, these techniques are combined to form practical image compression system.

Generally a compression system consists of two distinct structural blocks: an encoder and a decoder. An input image f (x,y) is fed into the encoder, which creates a set of symbols from the input data. After transmission over the channel, the encoded representation is fed to the decoder, where a restructured output image g (x,y) is generated. In general g (x,y) may or may not be an exact replica of f (x,y).

3.2 A General Compression Model

A general compression model is shown in figure 3.1. It shows that encoder and decoder consist of two relatively independent functions or sub blocks [1]. The encoder is made up of source encoder, which removes input redundancies, and a channel encoder, which increases the noise immunity of the source encoder's output. Similarly, the decoder includes a channel decoder followed by a source decoder. If the channel between the encoder and decoder is noise free, the channel encoder and decoder are omitted, and the general encoder and decoder is noise free, the channel encoder and decoder are omitted, and the general encoder and decoder become the source encoder and decoder, respectively.

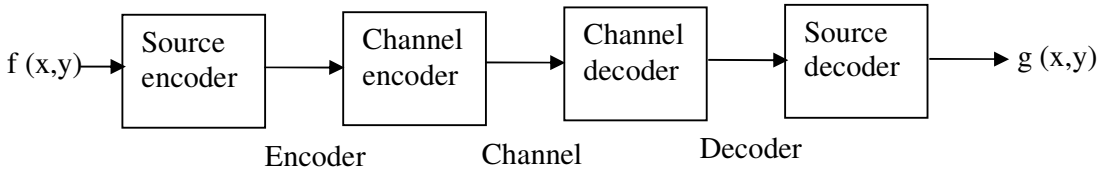
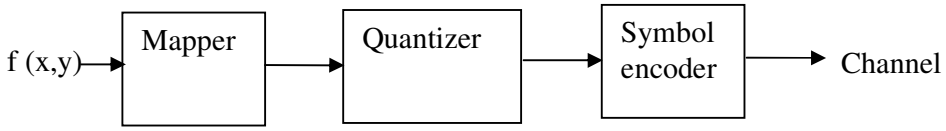


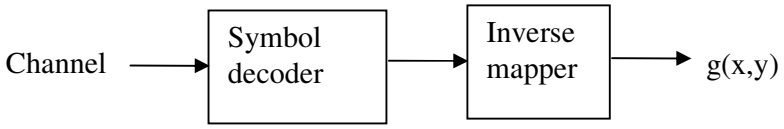
Figure 3.1: General Compression Model

3.2.1 The Source Encoder

The source encoder is responsible for reducing or eliminating any coding, interpixel, or psychovisual redundancies in the input image. The specific application dictates the best encoding approach. Normally, the approach can be modeled by a series of three independent operations. Operation is designed to reduce one of the three redundancies discussed earlier.



Figures 3.2 (a) Source encoder



Figures 3.2 (b) Source decoder

3.2.2 Mapper

In the first stage of the source encoding process, the mapper transforms the input data into a (usually non-visual) format designed to reduce interpixel redundancies in the input image. This operation generally is reversible and may or may not reduce directly the amount of data required to represent the image.

3.2.3 Quantizer

The second stage or quantizer block reduces the accuracy of the mapper's output in accordance with some pre-established fidelity criterion. This stage reduces the Psychovisual redundancies of the input image. This operation is irreversible. Thus, it must be omitted when error-free compression is desired.

3.2.4 Symbol Encoder

In the third and final stage of source encoding processes, the symbol coder creates a fixed or variable-length code to represent the quantizer output and maps the output in accordance with the code. The term symbol coder distinguishes this coding operation from the overall source encoding processes. In most cases, a variable length code is used to represent the mapped and quantized data set. It assigns the shortest code words to the most, frequently occurring output values and thus reduces coding redundancy. The operation is completely reversible. Upon completion of symbol coding step, the input image has been processed to remove each of the three redundancies discussed earlier. It is shown that the source encoding processes consist three successive operations, but all three operations are not necessarily included in every compression. For example, the quantizer must be omitted when error free compression is desired. In addition, some compression techniques normally are modeled by merging blocks that are physically separate in figure 3.2 (a).

3.2.5 Source Decoder

The source decoder shown in figure contains only two components: a symbol decoder and an inverse mapper. These blocks perform, in reverse order, the inverse operations of the source encoder's symbol encoder and mapper blocks. Because quantization results in irreversible information loss, an inverse quantizer block is not included in the general source decoder model shown in the figure 3.2 (b).

3.2.6 Channel Encoder and Decoder

The channel encoder and decoder play an important role in the overall encoding-decoding process when the channel of above figure 3.1 is noisy or prone to error. They are designed to reduce the impact of channel noise by inserting a controlled form of redundancy into the source-encoded data. As the output of the source encoder contains little redundancy, it would be highly sensitive to transmission noise without the addition of this "controlled redundancy".

3.3 Lossless Compression Techniques

In lossless compression scheme, the reconstructed image after compression, is numerically identical to the original image, i.e. original image can be reconstructed without any errors. However lossless compression can only achieve modest amount of compression. This is important for applications like compression of text. It is very important that the reconstruction is identical to the original text, as very small differences can result in statements with very different meanings. Consider the sentences, **"do now send money"** and **"do not send money"**. A similar argument holds for computer files and for certain types of data such as bank records. Various techniques for lossless compression are below:

3.3.1 Huffman Coding

The basic idea in Huffman coding is to assign short codeword to those input blocks with high probabilities and long code words to those with low probability. A Huffman code is designed by merging together the two least probable characters, and repeating this process until there is only one character remaining. A code tree is thus generated and the Huffman code is obtained from the labeling of the code tree [11]. An example of how this is done is shown in table 3.1.

Table 3.1: Huffman Source Reductions

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

At the far left, a hypothetical set of the source symbols and their probabilities are ordered from top to bottom in terms of decreasing probability values. To form the first source reductions, the bottom two probabilities, 0.06 and 0.04 are combined to form a "compound symbol" with probability 0.1. This compound symbol and its associated probability are placed in the first source reduction column so that the probabilities of the reduced source are also ordered from the most to the least probable. This process is than repeated until a reduced source with two symbols (at the far right) is reached. The second step if Huffman's procedure is to code each reduced source, starting with the smallest source and working back to its original source. The minimal length binary code for a two-symbol source, of course, is the symbols 0 and 1. As shown in table 3.2, these symbols are assigned to the two symbols on the right (the assignment is arbitrary; reversing the order of the 0 and would work just and well). As the reduced source symbol with probabilities 0.6 was generated by

combining two symbols in the reduced source to its left, the 0 used to code it is now assigned to both of these symbols, and a 0 and 1 are arbitrary appended to each to distinguish them from each other. This operation is then repeated for each reduced source until the original course is reached. The final code appears at the far-left in table 3.2. The average length of the code is given by the average of the product of probability of the symbol and number of bits used to encode it. This is calculated below:

L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5) = 2.2 bits/ symbol and the entropy of the source is 2.14 bits/symbol, the resulting Huffman code efficiency is 2.14/2.2 = 0.973.

Table 3.2: Huffman Code Assignment Procedure

Original source			Source reduction							
Sym.	Prob.	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

Huffman's procedure creates the optimal code for a set of symbols and probabilities subject to the constraint that the symbols be coded one at a time. After the code has been created, coding and/or decoding is accomplished in a simple look-up table manner. The code itself is an instantaneous uniquely decodable block code. It is called a block code, because each source symbol is mapped into a fixed sequence of code symbols. It is instantaneous, because each code word in a string of code symbols can be decoded without referencing succeeding symbols. It is uniquely decodable, because any string of code symbols can be decoded in only one way. Thus, any string of Huffman encoded symbols can be decoded by

examining the individual symbols of the string in a left to right manner. For the binary code of table 3.2, a left-to-right scan of the encoded string 010100111100 reveals that the first valid code word is 01010, which is the code for symbol a_3. The next valid code is 011, which corresponds to symbol a_1. Continuing in this manner reveals the completely decoded message to be $a_3a_1a_2a_2a_6$.

3.3.2 Arithmetic Coding

Arithmetic coding generates non-block codes. In arithmetic coding, a one-to-one correspondence between source symbols and code words does not exist. Instead an entire sequence of source symbols (or message) is assigned a single arithmetic code word. The code word itself defines an interval or real numbers between 0 and 1. As the number of symbols in the message increases, the interval used to represent it becomes smaller and the number of information units (say, bits) required to represent the interval becomes larger. Each symbol of the message reduces the size of the interval in accordance with its probability of occurrence.

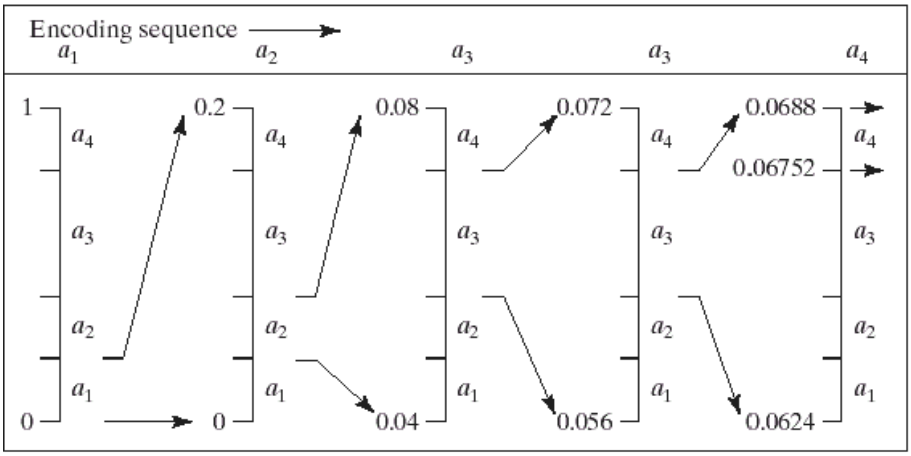


Figure 3.3: Arithmetic Coding Procedure

3.3.3 Run Length Coding

The technique of run length coding exploits the high interpixel redundancy that exists in relatively simple images [2]. In run length coding we look for gray levels that repeat along

each row of the image. A 'run' of consecutive pixels whose gray level is identical is replaced with two values the length of the run and the gray level of all the pixels in the run. Hence, the sequence (50, 50, 50, 50) becomes (4, 50). Run length coding can be applied on a row-by-row basis, or we can consider the image to be a one-dimensional data stream in which the last pixel in a row is adjacent to the first pixel in the next row. This can lead to slightly higher compression ratio if the left and right–hand sides of the image are similar.

For the special case of binary images, we don't need to record the value of a run, unless it is the first run of the row. This is because there are only two possible values for a pixel in a binary image. If the first run has one of the values, the second run implicitly has the other value; the third run implicitly has the same value as the first, and so on. Note that, if the run is of length 1, run length coding replaces one value with a pair of values. It is therefore possible for run length coding to increase the size of the dataset in images where run of length 1 are numerous. This might be the case in noisy or highly textured images. For this reason, it is most useful for the compression of binary images or very simple grayscale images.

3.3.4 Delta Compression

Delta compression (also known as differential coding) is a very simple, lossless technique in which we recode an image in terms of the differences in gray level between each pixel and the previous pixel in the row. The first pixel, of course, must be represented as an absolute value, but subsequent values can be represented as differences, or 'deltas'. Most of those differences will be very small, because gradual changes in gray level are more frequent than sudden changes in the majority of image. These small differences can be coded using fewer bits. Thus, delta compression exploits interpixel redundancy to create coding redundancy, which we than remove to achieve compression.

3.4 Lossy Compression Techniques

Lossy compression schemes involve some loss of information, and data that have been compressed using lossy techniques generally cannot be recovered or reconstructed exactly. Often this is because the compression completely discards redundant information. However, lossy schemes are capable of achieving much higher compression. This is important for applications like TV signals, teleconferencing. Here is tradeoff between compression and accuracy. Various techniques for lossy compression are discussed below:

3.4.1 Lossy Predictive Coding

A quantizer, that also executes rounding, is added between the calculation of the prediction error e_n and the symbol encoder. It maps e_n to a limited range of values q_n and determines both the amount of extra compression and the deviation of the error-free compression [1,2]. This happens in a closed circuit with the predictor to restrict an increase in errors. The predictor does not use e_n but rather q_n, because both the encoder and decoder know it.

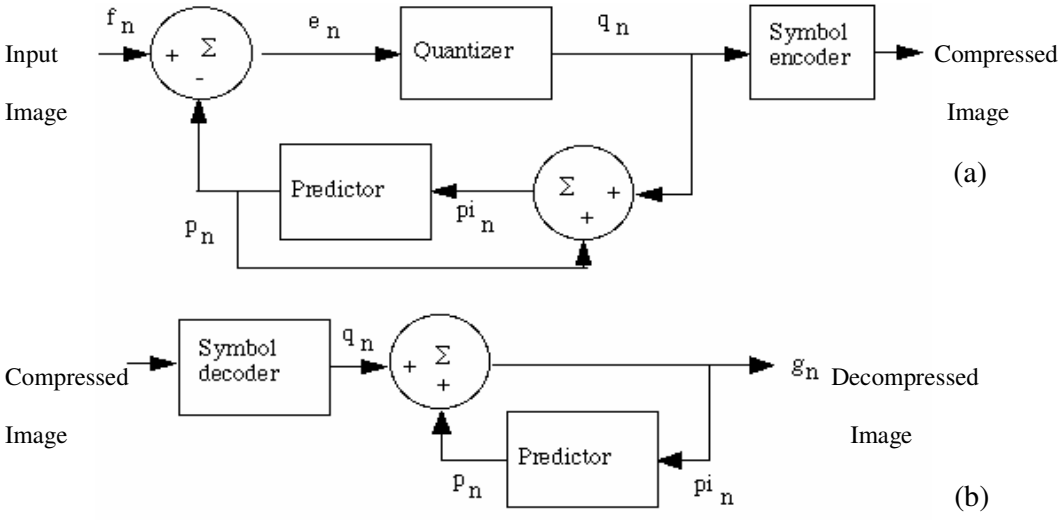


Figure 3.4: A lossy predictive coding model: (a) encoder; (b) decoder

3.4.2 Transform Coding

Transform coding first transforms the image from its spatial domain representation to a different type of representation using some well-known transform and then codes the transformed values (coefficients). The goal of the transformation process is to decorrelate the pixels of each subimage, or to pack as much information as possible into the smallest number of transforms coefficients [12]. This method provides greater data compression compared to predictive methods, although at the expense of greater computational requirements. The choice of particular transform in a given application depends on the amount of reconstruction error that can be tolerated and the computational resources available.

3.4.2.1 General Model

As shown in figure 3.5 (a), encoder performs three relatively straightforward operations i.e. Sub image decomposition, Transformation and Quantization. The decoder implements the inverse sequence of steps with the exception of the quantization function of the encoder shown in figure 3.5 (b).

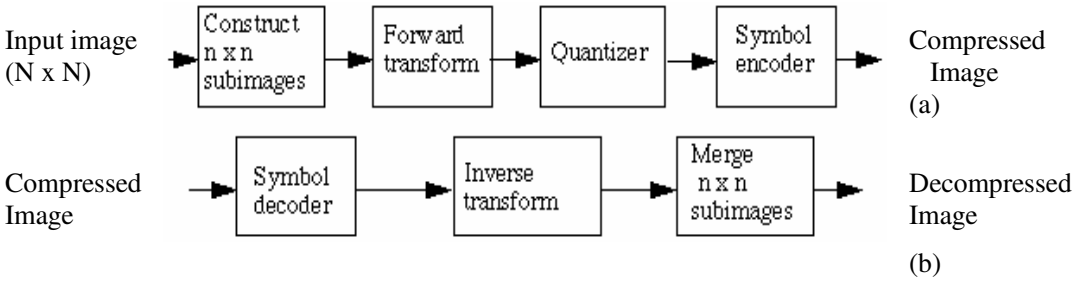


Figure 3.5: A transform coding system: (a) encoder; (b) decoder

An N x N input image is first subdivided into sub images of size n x n, which are then transformed to generated $(N/n)^2$, n x n sub image transform arrays. The quantization stage selectively eliminates or more coarsely quantize the co-efficient that carry the least information. These co-efficient have the smallest impact on reconstructed sub image quality. Any or all of the transform coding steps can be adapted in local image content, called adaptive transform coding, or fixed for all sub images, called non-adaptive transform coding. In the present implementation non-adaptive transform coding has been chosen.

3.4.2.2 Selection of the Transformation

In transform coding system any kind of transformation can be chosen, such as Karhunen Loeve (KLT), Discrete Cosine (DCT), Walsh-Hadamard (WHT), Fourier (FT) etc. The choice of a particular transformation in a given application depends on the amount of reconstruction error that can be tolerated and the computational resources available [13]. To receive good result the transform should have the following properties:

- Fast to compute.
- Decorrelate transform coefficient to remove redundancies.
- Pack energy into only a few transform coefficients.
- Preserve energy.

3.4.2.3 Subimage Size Selection

A significant factor affecting transform coding, error and computational complexity is function of subimage size. In most application images are subdivided so that the correlation (redundancy) between adjacent sub images is reduced to some acceptable level so that n can be expressed as an integral power of 2, where n is the sub image dimension. The latter condition simplifies the computation of the sub image transformation. In general, both the levels of compression and computational complexity increase as the sub image size increases. The most popular sub image sizes are 8 x 8 and 16 x 16. Figure 3.6 illustrates graphically the impact of sub image size on transform coding reconstruction error.

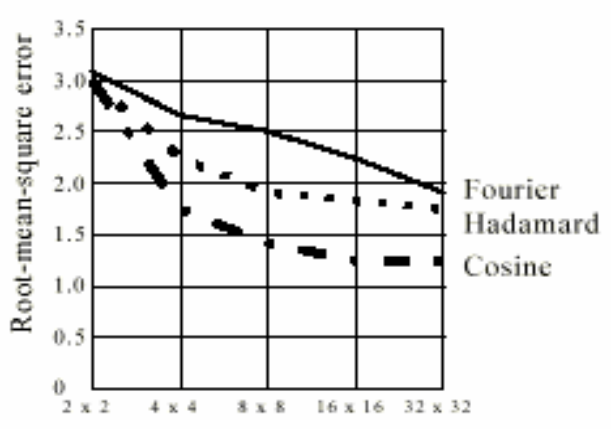


Figure 3.6: Reconstruction Error versus sub image size

It is clear from figure 3.6 that the Hadamard and Cosine curves flatten as the size of the sub image becomes greater than 8 x 8, whereas the Fourier reconstruction error decreases even more rapidly in this region.

3.4.2.4 Bit Allocation

The reconstruction error associated with quantization is a function of number and relative importance of the transformed co-efficient of the discarded, as well as precision of the retained co-efficient. In transformed coding system, the retained co-efficient can be selected on the basis of maximum variance, called zonal coding, or on the basis of maximum magnitude, called threshold coding. The overall process of truncating, quantizing and coding the co-efficient of a transformed sub image is commonly called bit allocation.

Zonal coding is based on the information theory. The zonal sampling process can be viewed, as multiplying each transformed coefficient by the corresponding element in a zonal mask, which is constructed by placing 1 in the locations of maximum variance and 0 in all other locations. The co-efficient retained during the zonal sampling process must be quantized and coded. Zonal coding is usually implemented by using a single fixed mask for all sub images [12].

Threshold coding, however, is inherently adaptive in the sense that the location of the transform co-efficient retained for each sub image varies from one sub image to another. There are three basic ways to threshold a transformed sub image:

(1) A single global threshold can be applied to all sub images.

(2) A different threshold can be used for each sub image.

(3) The threshold can be varied as a function of the location of each co-efficient within the sub image.

In the first approach, the level of compression differs from image to image, depending on the number of co-efficient that exceed the global threshold. In the second, called N-largest coding, the same number of co-efficient is discarded for each sub image. As a result, the code rate is constant and is known in advance. The third technique, like the first, results in a variable code rate, but offers the advantage the thresholding and quantization can be combined.

3.4.2.5 Coding

The final step in the compression process is coding the quantized image. First the co-efficient of the image are arranged in the zig-zag sequence. Then they have been encoded using run-length coding and variable-length coding techniques as discussed earlier.

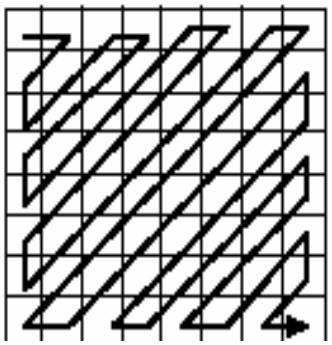


Figure 3.7: The Zig-Zag Sequence

CHAPTER–4
FRACTIONAL FOURIER TRANSFORM

4.1 Introduction

The FRFT belongs to the class of time–frequency representations that have been extensively used by the signal processing community [14]. In all the time–frequency representations, one normally uses a plane with two orthogonal axes corresponding to time and frequency. If we consider a signal x (t) to be represented along the time axis and its

ordinary Fourier transform X (f) to be represented along the frequency axis, then the Fourier transform operator (denoted by F) can be visualized as a change in representation of the signal corresponding to a counterclockwise rotation of the axis by an angle $\pi/2$. This is consistent with some of the observed properties of the Fourier transform. For example, two successive rotations of the signal through $\pi/2$ will result in an inversion of the time axis. Moreover, four successive rotations will leave the signal unaltered since a rotation through 2π of the signal should leave the signal unaltered. The FRFT is a linear operator that corresponds to the rotation of the signal through an angle which is not a multiple of $\pi/2$, i.e. it is the representation of the signal along the axis u making an angle α with the time axis [15]. With the advent of FRFT and related concept, it is seen that the properties and applications of the ordinary Fourier transform are special cases of those of the FRFT.

4.2 Mathematical Definition

The FRFT is defined with the help of the transformation kernel K_α, as [14]

$$k_\alpha(t,u)=\begin{cases}\delta(t-u) & \text{if } \alpha \text{ is a multiple of } 2\pi \\ \delta(t+u) & \text{if } \alpha+\pi \text{ is a multiple of } 2\pi \\ \sqrt{\dfrac{1-j\cot\alpha}{2\pi}}e^{j\left(\left(u^2+t^2\right)/2\right)\cot\alpha-jut\operatorname{cosec}(\alpha)} & \text{if } \alpha \text{ is not a multiple of } \pi\end{cases} \tag{4.1}$$

Another useful form of writing the square root factor preceding the transformation kernel K_α can be obtained using the relation

$$\sqrt{\frac{1-j\cot\alpha}{2\pi}}=\sqrt{\frac{-je^{j\alpha}}{2\pi\sin\alpha}} \tag{4.2}$$

The FRFT is defined using this kernel as (FRFT of order α of x (t) denoted by $X_\alpha(u)$)

$$X_\alpha(u)=\int_{-\infty}^{\infty}x(t)k_\alpha(t,u)dt, \tag{4.3}$$

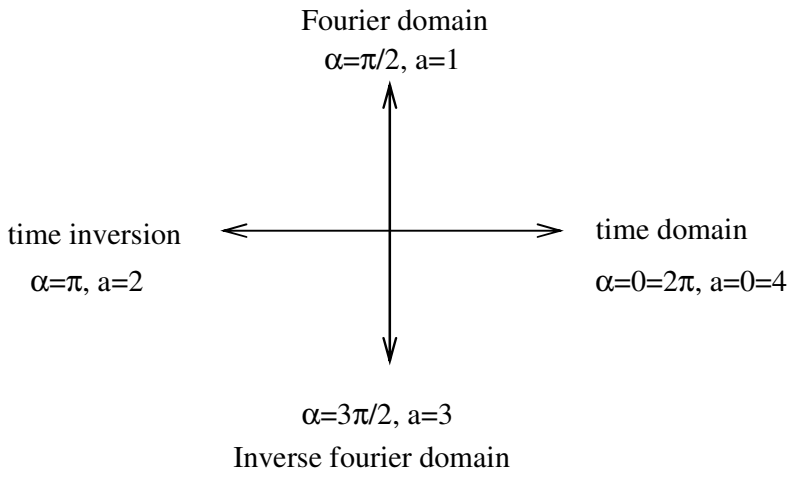
where

$$X_\alpha(u)=\begin{cases}\sqrt{\dfrac{1-j\cot\alpha}{2\pi}}e^{j\left(u^2\cot\alpha/2\right)}\displaystyle\int_{-\infty}^{\infty}x(t)e^{j\left(t^2\cot\alpha/2\right)-jut\operatorname{cosec}(\alpha)} & \text{if } \alpha \text{ is not a multiple of } \pi \\ x(t) & \text{if } \alpha \text{ is a multiple of } 2\pi \\ x(-t) & \text{if } \alpha+\pi \text{ is a multiple of } 2\pi\end{cases} \tag{4.4}$$

where $\alpha = a\pi/2$.

Let F_α denote the operator corresponding to the FRFT of angle α. Under this notation, some of the important properties of the FRFT operator are listed below [16]:

- For α =a=0 we do get the identify operator: $F^0 = F^4 = I$
- For $\alpha = \pi/2$; i.e. a=1,we get the Fourier operator: $F^1 = F$
- For $\alpha = \pi$; i.e. a=2,we get the reflection operator: $F^2 = FF = I$
- For $\alpha = 3\pi/2$; i.e. a=3,we get the inverse Fourier operator: $F^3 = FF^2 = F$



FRFT domain in t-f plane

So for an angle from 0 to 2π, we have the values of 'a' from 0 to 4 and it can be shown that the transform kernel is periodic with a period 4.

One important conclusion that can be drawn from these properties is that the signal x (t) and its FRFT (of order α) $X_\alpha(u)$ form a transform pair and are related to each other by the following equation:

$$X_\alpha(u) = \int_{-\infty}^{\infty} x(t) k_\alpha(t,u) dt, \tag{4.5}$$

$$x(t) = \int_{-\infty}^{\infty} X_\alpha(u) k_{-\alpha}(u,t) du \tag{4.6}$$

4.3 Properties of fractional Fourier transform

The previous relations imply several properties for the FRFT. The FRFT satisfies the following properties [14]:

4.3.1 Conservation of symmetry

The symmetry properties of the Fourier transform for even and odd real sequences do not extend to the FRFT, since the FRFT of a real function is not necessarily Hermitian. The symmetry property for the FRFT is slightly different and is shown below.

Let $x(t)$ be real.

$$X_\alpha^*(u) = \left(\sqrt{\frac{1-j\cot\alpha}{2\pi}}\right)^* \int_{-\infty}^{\infty} x^*(t) e^{-j\left(\left(u^2+t^2\right)/2\right)\cot\alpha + jut\,\mathrm{cosec}(\alpha)} dt$$

$$= \left(\sqrt{\frac{1-j\cot(-\alpha)}{2\pi}}\right)^* \int_{-\infty}^{\infty} x(t) e^{j\left(\left(u^2+t^2\right)/2\right)\cot(-\alpha) - jut\,\mathrm{cosec}(-\alpha)} dt$$

$$= X_{-\alpha}(u) \tag{4.7}$$

4.3.2 Parseval's theorem

The well-known Parseval's theorem for Fourier transform can be easily extended to the FRFT as well.

$$\int_{-\infty}^{\infty} x(t) y^*(t) dt = \int_{-\infty}^{\infty} X_\alpha(u) Y_\alpha^*(u) du \tag{4.8}$$

An interesting observation that develops from the Parseval's theorem is the energy preservation property as given below. This property can be obtained by the application of Parseval's theorem

$$\int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |X_\alpha(u)|^2 du \tag{4.9}$$

The energy preservation property of the FRFT is only to be expected because the FRFT is based on the decomposition of the signal on the orthonormal basis set of the chirp functions. Due to the energy preserving property of the Fourier transform, the squared magnitude of the Fourier transform of a signal is often called the energy spectrum of the signal and is interpreted as the distribution of the signal energy among the different frequencies. Although less intuitive, this allows us to call the squared magnitude of the FRFT of a signal as the fractional energy spectrum of the signal and interpret it as the distribution of the signal energy between the different chirp basis functions.

The other well established properties of FRFT are listed in Table 4.1.

Table 4.1: Properties of Fractional Fourier Transform

Operation	Signal, $x(t)$	Fractional Fourier Transform, $X_\alpha(u)$
Time shift	$x(t-\tau)$	$e^{j(\tau^2/2)\sin\alpha\cos\alpha - ju\tau\sin\alpha} X_\alpha(u-\tau\cos\alpha)$
Modulation	$x(t)e^{jvt}$	$e^{-jv^2(\sin\alpha\cos\alpha)/2 + juv\cos\alpha} X_\alpha(u-v\sin\alpha)$
Inversion of time axis	$x(-t)$	$X_\alpha(-u)$
Scaling of time axis	$x(ct)$	$\sqrt{\frac{1-j\cot\alpha}{c^2-j\cot\alpha}}\, e^{j(u^2/2)\cot\alpha\left(1-\left(\cos^2\beta/\cos^2\alpha\right)\right)}$
Differentiation	$x'(t)$	$X'_\alpha(u)\cos\alpha + ju\sin\alpha X_\alpha(u)$
Integration	$\int_a^t x(t')dt'$	$\sec\alpha e^{-j(u^2/2)\tan\alpha}\int_a^u X_\alpha(z)e^{j(z^2/2)\tan\alpha}dz$ if $\alpha-\pi/2$ is not a multiple of π
Multiplication with ramp	$tx(t)$	$u\cos\alpha X_\alpha(u) + j\sin\alpha X'_\alpha(u)$
Division by ramp	$x(t)/t$	$-j\sec\alpha e^{j(u^2/2)\cot\alpha}\int_{-\infty}^u x(z)e^{-j(z^2/2)\cot\alpha}dz$ if α is not a multiple of π
Convolution	$x(t)*g(t)$	$F^{-a}\left[\left(F^a x\right)\left(F^a g\right)\right]$

4.4 Discrete Fractional Fourier Transform (DFRFT)

With the advent of computers and enhanced computational capabilities the Discrete Fourier Transform (DFT) came into existence in evaluation of FT for real time processing. Further these capabilities are enhanced by the introduction of DSP processors and Fast Fourier Transform (FFT) algorithms. On similar lines, so there arises a need for

discretization of FRFT. Furthermore, DFT is having only one basic definition and nearly 200 algorithms are available for fast computation of DFT. But when FRFT is analyzed in discrete domain there are many definitions of Discrete Fractional Fourier Transform (DFRFT) [8-10]. It is also established that none of these definitions satisfies all the properties of continuous FRFT. Santhanam and McClellan first reported the work on DFRFT in 1995. Thereafter within a short span of time a lot many definitions of DFRFT came into existence and these definitions are classified according to the methodology used for calculations in 2000 by Pie *et al*.

4.4.1 Algorithm for Discrete Fractional Fourier Transform computation

The fractional Fourier transform is a member of a more general class of transformations that are sometimes called linear canonical transformations or quadratic-phase transforms. Members of this class of transformations can be broken down into a succession of simpler operations, such as chirp multiplication, chirp convolution, scaling, and ordinary Fourier transformation [8].

The FRFT of a signal x (t) as given by Eq. (4.4) can be computed by the following steps. In the first step, multiply the function x (t) by a chirp function u (t) as below:

g(x) = u(t) x (t)

$$= \exp\left[-i\pi x^2 \tan(\alpha/2)\right] \text{ x(t)} \qquad (4.10)$$

The bandwidth and time-bandwidth product of g(x) can be as large as twice that of x(t). Thus, we require the samples of g(x) at intervals of $1/2\Delta$ x. if the samples of x(t) spaced at $1/\Delta$ x are given to begin with, we can interpolate these and then multiply by the samples of the chirp function to obtain the desired samples of g(x). There are efficient ways of performing the required interpolation.

The next step is to convolve g(x) with a chirp function, as below:

$$g'(x) = A_\alpha \int_{-\infty}^{\infty} \exp\left[i\pi\beta(x-x')^2\right] g(x')dx' \tag{4.11}$$

where

$$A_\alpha \equiv \frac{\exp(-i\pi\,\mathrm{sgn}(\sin\alpha)/4 + i\alpha/2)}{|\sin\alpha|^{1/2}}, \alpha \equiv \frac{a\pi}{2} \tag{4.12}$$

To perform this convolution, since g(x) is bandlimited, the chirp function can also be replaced with its bandlimited version without any effect, that is

$$\begin{aligned} g'(x) &= A_\alpha \int_{-\infty}^{\infty} \exp\left[i\pi\beta(x-x')^2\right] g(x')dx' \\ &= A_\alpha \int_{-\infty}^{\infty} h(x-x')g(x')dx' \end{aligned} \tag{4.13}$$

where

$$h(x) = \int_{-\Delta x}^{\Delta x} H(v)\exp(i2\pi vx)dv \tag{4.14}$$

and where

$$H(v) = \frac{1}{\sqrt{\beta}} e^{i\pi/4} \exp(-i\pi v^2/\beta) \tag{4.15}$$

is the Fourier transform of $\exp(i\pi\beta x^2)$.

Now, (4.13) can be sampled, giving

$$g'\left(\frac{m}{2\Delta x}\right) = \sum_{n=-N}^{N} h\left(\frac{m-n}{2\Delta x}\right) g\left(\frac{n}{2\Delta x}\right) \tag{4.16}$$

This convolution can be evaluated using a fast Fourier transform. Then the final step is again multiply with chirp function as below:

$$x_a(t) = \exp\left[-i\pi x^2 \tan(\alpha/2)\right] g'(x) \tag{4.17}$$

Then, after performing the last step, we obtain the samples of $x_a(t)$ spaced at $1/2\Delta$x. since we have assumed that all transforms of $x(t)$ are band limited to the interval $[-\Delta x/2, \Delta x/2]$, we finally decimate these samples by a factor of 2 to obtain samples of $x_a(t)$ spaced at $1/\Delta$x.

4.5 Two-Dimensional Fractional Fourier Transform

The one-dimensional FRFT is useful in processing single-dimensional signals such as speech waveforms. For analysis of two-dimensional (2D) signals such as images, we need a 2D version of the FRFT. For an M×N matrix, the 2D FRFT is computed in a simple way: The 1D FRFT is applied to each row of matrix and then to each column of the result. Thus, the generalization of the FRFT to two-dimension is given by [16,18]:

$$X_{\alpha,\beta}(u,s) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} k_{\alpha,\beta}(u,s;t,r)x(t,r)dtdr \tag{4.18}$$

where

$$k_{\alpha,\beta}(u,s;t,r) = k_{\alpha}(u,t)k_{\beta}(s,r). \tag{4.19}$$

In the case of the two-dimensional FRFT we have to consider two angles of rotation $\alpha=a\pi/2$ and $\beta=b\pi/2$. If one of these angles is zero, the 2D transformation kernel reduces to the 1D transformation kernel. The FRFT can be extended for higher dimensions as:

$$X_{(\alpha_1 \text{-----} \alpha_n)}(u_{1,\text{-----}} u_n) = \int_{-\alpha}^{\alpha} \text{--} \int_{-\alpha}^{\alpha} k_{\alpha_{1,\text{----}}\alpha_n}(u_{1,\text{----}}, u_n; t_1, \text{-----}\ t_n)x(t_{1,\text{-----}} t_n)dt_{1\text{----}}dt_{n,} \tag{4.20}$$

CHAPTER –5

IMAGE COMPRESION USING FRACTIONAL FOURIER TRANSFORM

In image processing, an important part is the compression. This means the reducing the dimensions of the images, to a level that can be easily used or processed. Image compression using transform coding yields extremely good compression, with controllable degradation of image quality. In the present implementation FRFT, generalization of FT has been chosen. One of the reasons for this is that, the FRFT provides additional degree of freedom to the problem as parameter 'a' gives multidirectional application. With the extra degree of freedom provided by the FRFT, its fractional order 'a', high visual quality decompressed image can be achieved for same amount of compression as that for Fourier transform [19,20].

5.1 FRFT Compression Model

In image compression using FRFT, a compression model encoder performs three relatively straightforward operations i.e. Subimage decomposition, Transformation and Quantization. The decoder implements the inverse sequence of steps of encoder. Because quantization results in irreversible information loss, so inverse quantizer block is not included in the decoder as shown in figure 5.1(b). Hence it is lossy compression technique.

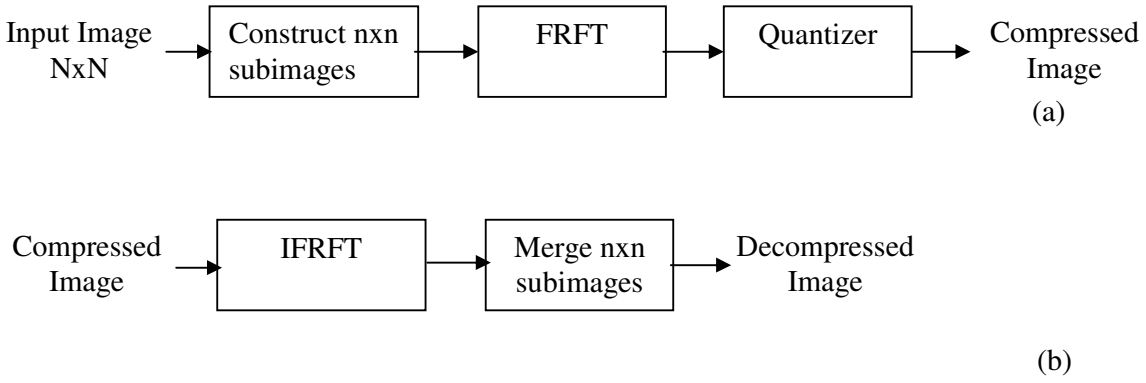


Figure 5.1: FRFT compression model: (a) encoder; (b) decoder

5.1.1 Subimage decomposition

An image is first partitioned into non-overlapped n×n subimages as shown in figure 5.1(a). The most popular subimage sizes are 8×8, 16×16. In present implementation subimage size chosen is 8×8. As subimage size increases, error decreases but computational complexity increases. Compression techniques that operate on block of pixels i.e. subimages are often described as block coding techniques. The transform of a block of pixels may suffer from discontinuity effects resulting in the presence of blocking artifacts in the image after it has been decompressed.

5.1.2 Transformation

In this step, a 2D-FRFT is applied to each block to convert the gray levels of pixels in the spatial domain into coefficients in the frequency domain. By using FRFT a large amount of information is pack into smallest no. of transform coefficients, hence small amount of compression is achieved at this step. At decoder inverse FRFT is applied. By changing the value of parameter ‘a’ to ‘-a’ we get inverse FRFT.

5.1.3 Quantization

The final step in compression process is quantized the transformed coefficients according to cutoff selected and variation of 'a'. By adjusting the cutoff of the transform coefficients, a compromise can be made between image quality and compression factor. With the FRFT by varying its free parameter 'a', high compression ratio can achieve even for same cutoff. The quantized coefficients are then rearranged in a zig-zag scan order to form compressed image which can be stored or transmitted.

At decoder simply inverse process of encoder is performed by using inverse 2D-FRFT. The non-overlapped subimages merged to get decompressed image. As we know it is lossy compression technique so there is some error between decompressed image and original image which can be evaluate.

CHAPTER–6
INTRODUCTION TO MATLAB

6.1 History

The founders of the Math Works recognized the need among engineers and scientists for more powerful and productive computation environments beyond those provided by language such as Fortran and C. In response to that need, the founders combined their expertise in mathematics, engineering, and computer Science to develop MATLAB, a high performance technical computing environment. MATLAB combines comprehensive math and graphics functions with a powerful high-level language.

Cleve Moter is Chairman and Chief Scientist at The Math Works who, in addition to being the author of the first version of MATLAB is one of the authors of the LINPACK and EISPACK scientific subroutine libraries created in the late 70's LINPACK, which was written in FORTRAN, was a package of programs to be used for the solution of linear systems and related problems. The goal of the Math Works was to provide to students the ability to utilize those packages without having to write FORTRAN code.

6.2 Introduction

MATLAB is a software package for high performance numerical computation and visualization. It provides an interactive environment with hundreds of built in functions for technical computation, graphics, and animation. Best of all, it also provides easy extensibility with its own high level programming Language. The name MATLAB stands for Matrix Laboratory.

MATLAB's built in functions provide excellent tools for linear algebra computations, data analysis, signal processing, and optimization, numerical, solution of ordinary differential equations, (ODEs), quadrature, and many other type of scientific computations. These are numerous functions for 2-D and 3-D graphics as well as for animation. Also for these who cannot do without their Fortran or C codes, MATLAB even provides an external interface to run those programs from within MATLAB. MATLAB's language is very easy to learn and to use.

There are also several optional ‘Toolboxes' available from the developers of MATLAB. These toolboxes are collections of functions written for special applications such as Symbolic computation, Image processing, Statistics, Control System Design, Neural networks, etc.

The basic building block of MATALAB is the matrix. The fundamental data-type is the array. Vectors, scalars real matrices and complex matrices are all automatically handled as special cases of the base data type. There is no need to declare the dimensions of a matrix. MATLAB simply loves matrices and matrix operations.

6.3 Areas of Application

Because of MATLAB's numerous matrix and vector computation and manipulation algorithms, the software is primarily used for:

- Production solution to complex system of equations.
- Modeling, simulation, and prototyping.
- Data analysis, exploration, and visualization.

6.4 Image Analyses and Enhancement

MATLAB and the image processing toolbox support a broad range of advanced image processing functions. You can extract and analyze image features, compute feature measurements, and apply filter algorithms. You can use filter for different type of image noise of build customized filters with the filter design tools that are included, Interactive tools allow you to select arbitrary regions of interest, measure distances in images, and obtain pixel values and statistics.

6.4.1 Commands used for Image Analysis

Some command which are used to read, display, resize and show the image are given below:

i) Read an Image:

To read an image, use the imread command. To read in a TIFF image named pout.tif, and store it in an array named I, the command is

- I = imread ('pout.tif');

ii) Display an image:

To display an image, use the imshow command. Now call imshow to display I, the command is

- imshow (I)

iii) Check the Image in Memory:

Enter the whos command to see how I is stored in memory.

- whos;

MATLAB response with

- Name Size Bytes Class
- I 291x240 69840 unit8 array

- Grand total is 69840 elements using 69840 bytes

iv) To resize an image:

To change the size of an image, use the imresize command

- Imresize (I, 2)

Thus MATLAB and its image processing toolbox is a very handy tool for various image-processing techniques.

CHAPTER –7
SIMULATION RESULTS

In the case of lossy compression, the reconstructed image is only an approximation to the original. Although many performance parameters exist for quantifying image quality, it is most commonly expressed in terms of mean squared error (MSE) and peak signal to noise ratio (PSNR), which is defined as follows.

$$\mathrm{MSE} = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left[\hat{f}(i,j) - f(i,j)\right]^2$$

$$\mathrm{PSNR} = 10\log_{10}\left[\frac{255\times 255}{\mathrm{MSE}}\right]\mathrm{dB}$$

where M×N is the size of the images, $\hat{f}$ (i,j) and f(i,j) are the matrix elements of the decompressed and original images at (i,j) pixel. The larger PSNR values correspond to good image quality.

In order to evaluate the performance of image compression systems, compression ratio metric is often employed. In our results, compression ratio (CR) is computed as the ratio of non-zero entries in the original image to the non-zero entries in the transformed image.

Image compression experiments using fractional Fourier transform are conducted on four natural images: the first of course, is the standard Lenna, whose face has graced the pages of many signal processing books, papers, and projects. The second, third and fourth are the Cameraman, Barbara and Rice pictures used frequently in the image compression literature.

1) Lenna Image

a) Effect of varying cutoff

Fig. 7.1 (a)-(f) shows the decompressed Lenna image after a cutoff of 20, 40, 60, 80 and 100 respectively at 'a'=1. It is clear from these figures as cutoff increases, compression ratio increases (CR), mean square error (MSE) increases so image quality degrades. Generally there is tradeoff between compression ratio and image quality. At cutoff=20 image quality is best but as cutoff increases to 100 image quality is very poor.

b) Effect of varying 'a'

Fig. 7.2 (a)-(h) shows the results of Lenna image by changing the value of parameter 'a' by keeping CR=30. Here we are assuming ac=ar=a. It is clear that at 'a'=0.1, PSNR is very low so image quality is very poor. As 'a' increases, MSE decreases, PSNR increases therefore image quality improves. At 'a'=0.91 we get the optimum domain for which MSE is very low, PSNR is very high and better-decompressed image is retained. With further increase in 'a', MSE increases so image quality degrades.

c) Effect of varying CR

The curves of MSE and PSNR versus the changes of fractional order 'a' for different CR have been calculated and depicted in Fig. 7.3, 7.4 respectively. It is clear from above that for same CR as we increase 'a' MSE decreases, PSNR increases till 'a'=0.91 after that as 'a' increases, MSE increases, PSNR decreases. As we increase CR, by increasing the value of cutoff, MSE increases. It is clear from graphs that for every CR at 'a'=0.91 is an optimum domain.

d) Effect of Varying cutoff and 'a'

The curves of CR versus the changes of fractional order 'a' for different CO have been calculated and depicted in Fig. 7.5. It is clear that for same cutoff as we increase 'a', CR increases. So there is flexibility in FRFT to increase CR without increases cutoff value. To get more compression both cutoff and 'a' varies. For cutoff=100 and 'a'=1 we can achieve CR of 130:1 but image quality degrades. So there is some limit up to which error is tolerable.

e) PSNR versus no. of bytes to store

The curves of PSNR versus no. of bytes to store the images have been calculated and depicted in Fig. 7.6. It is clear from figure that as no. of bytes to store increases, CR decreases, MSE decreases and PSNR improve.

Fig. 7.1(a): The Original Lenna Image

Fig. 7.1(b): Lenna Image after decompression (cut off=20, MSE=57.91)

Fig. 7.1(c): Lenna Image after decompression (cut off=40, MSE=113.99)

Fig.7.1(d): Lenna Image after decompression (cut off=60, MSE=180.55)

Fig. 7.1: Simulation results of the Lenna Image with 256x256 pixels at 'a' =1 for varying cutoff

Fig. 7.1(e): Lenna Image after decompression
(cut off=80, MSE=232.73)

Fig. 7.1(f): Lenna Image after decompression
(cut off=100, MSE=288.27)

Fig. 7.1: Simulation results of the Lenna Image with 256x256 pixels at 'a' =1 for varying cutoff

Fig. 7.2(a): The Original Lenna Image

Fig. 7.2(b): Lenna Image after decompression
(a= 0.1, PSNR=5.69)

Fig. 7.2(c): Lenna Image after decompression
(a= 0.3, PSNR=7.23)

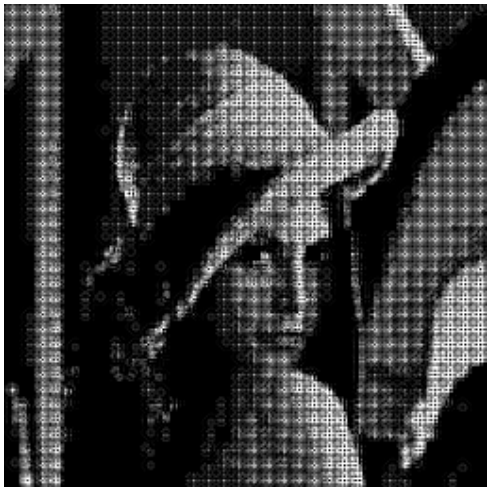


Fig. 7.2(d): Lenna Image after decompression
(a= 0.5, PSNR=9.29)

Fig. 7.2: Simulation results of the Lenna Image with 256x256 pixels with CR = 30 for varying ‘a’

Fig. 7.2(g): Lenna Image after decompression
(a= 0.7, PSNR= 16.21)

Fig. 7.2(f): Lenna Image after decompression
(a= 0.91, PSNR=30.51)

Fig. 7.2(e): Lenna Image after decompression
(a= 0.95, PSNR=27.27)

Fig. 7.2(h): Lenna Image after decompression
(a= 1, PSNR=25.21)

Fig. 7.2: Simulation results of the Lenna Image with 256x256 pixels with CR = 30 for varying ‘a’

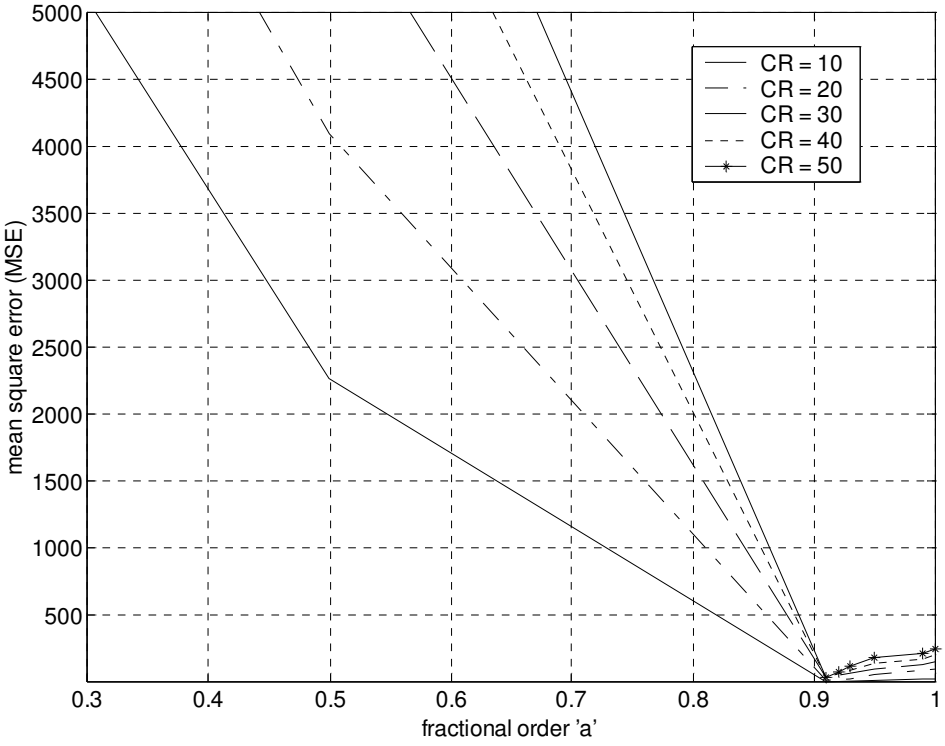


Fig. 7.3: Simulation results of the Lenna Image with 256×256 pixels for different CR. MSE vs. fractional orders for Lenna Image.

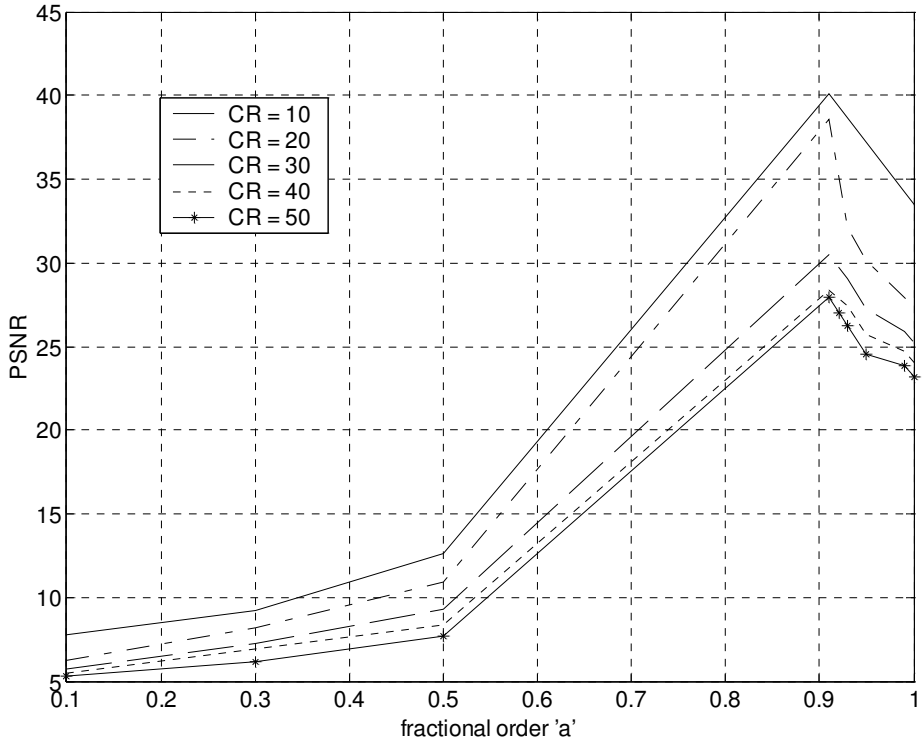


Fig. 7.4: Simulation results of the Lenna Image with 256×256 pixels for different CR. PSNR vs. fractional orders for Lenna Image.

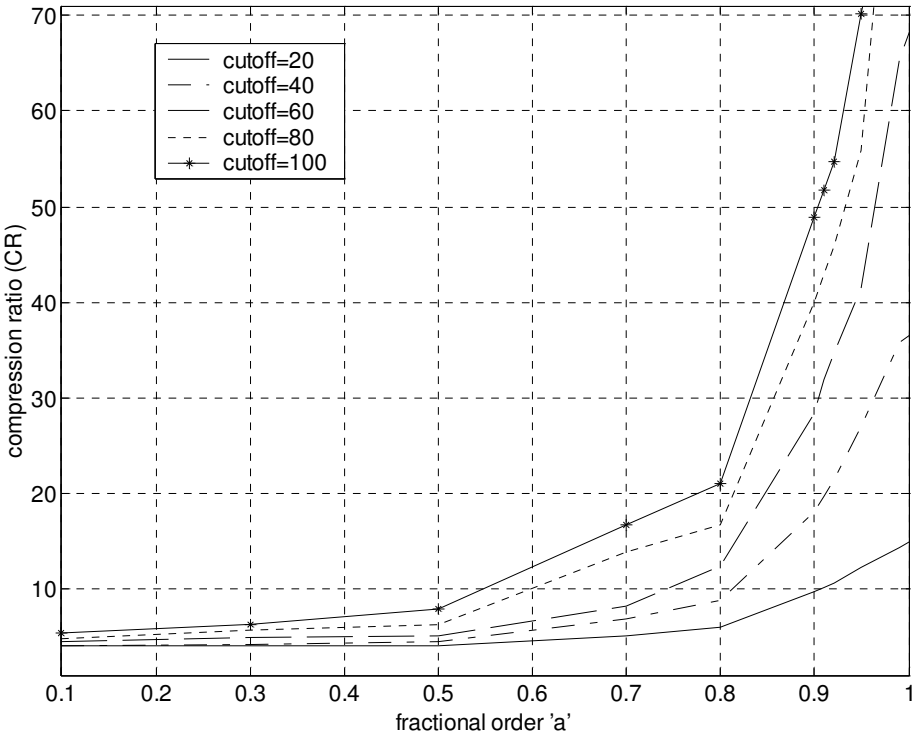


Fig. 7.5: Simulation results of the Lenna Image with 256×256 pixels for different cutoff. CR vs. fractional orders for Lenna Image.

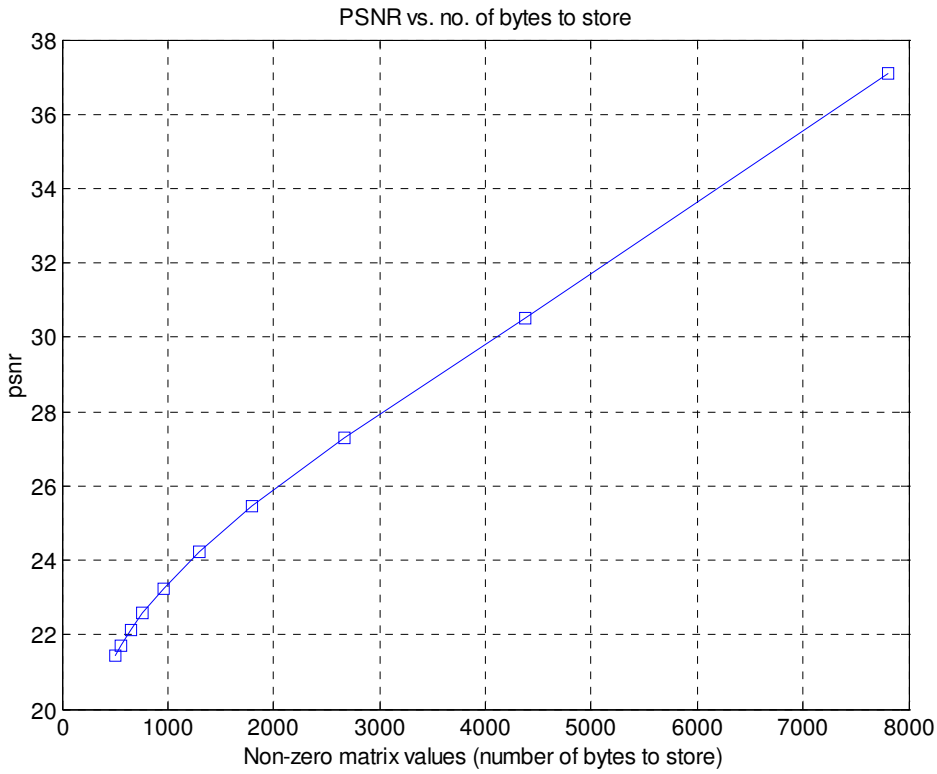


Fig. 7.6: Simulation results of the Lenna Image with 256×256 pixels.
PSNR vs. No.of bytes to store for Lenna Image.

2) Cameraman Image

a) Effect of varying cutoff

Fig. 7.7 (a)-(f) shows the decompressed Cameraman image after a cutoff of 20, 40, 60, 80 and 100 respectively at 'a'=1. It is clear from these figures as cutoff increases, compression ratio increases (CR), mean square error (MSE) increases so image quality degrades. At cutoff=20 image quality is best but as cutoff increases to 100 image quality is very poor.

b) Effect of varying 'a'

Fig. 7.8 (a)-(h) shows the results of Cameraman image by changing the value of parameter 'a' by keeping CR=30. It is clear that at 'a'=0.1, PSNR is very low so image quality is very poor. As 'a' increases, MSE decreases, PSNR increases therefore image quality improves. At 'a'=0.9 we get the optimum domain for which MSE is very low, PSNR is very high and better-decompressed image is retained. With further increase in 'a', MSE increases so image quality degrades.

c) Effect of varying CR

The curves of MSE and PSNR versus the changes of fractional order 'a' for different CR have been calculated and depicted in Fig. 7.9, 7.10 respectively. It is clear from above that for same CR as we increase 'a' MSE decreases, PSNR increases till 'a'=0.9 after that as 'a' increases, MSE increases, PSNR decreases. As we increase CR, by increasing the value of cutoff, MSE increases. It is clear from graphs that for every CR at 'a'=0.9 is an optimum domain.

d) Effect of varying cutoff and 'a'

The curves of CR versus the changes of fractional order 'a' for different CO have been calculated and depicted in Fig. 7.11. It is clear that for same cutoff as we increase 'a', CR increases. To get more compression both cutoff and 'a' varies. For cutoff=100 and 'a'=1 we can achieve CR of 109:1 but image quality degrades.

e) PSNR versus no. of bytes to store

The curves of PSNR versus no. of bytes to store the images have been calculated and depicted in Fig. 7.12. It is clear from figure that as no. of bytes to store increases, CR decreases, MSE decreases and PSNR improve.

Fig. 7.7 (a): The Original Cameraman Image

Fig. 7.7(b): Cameraman Image after decompression (cut off=20, MSE=36.82)

Fig. 7.7(c): Cameraman Image after decompression (cut off=40, MSE=102.43)

Fig. 7.7(d): Cameraman Image after decompression (cut off=60, MSE=164.16)

Fig. 7.7: Simulation results of the Cameraman Image with 256x256 pixels at 'a' =1 for varying cutoff

Fig. 7.7(e): Cameraman Image after decompression (cut off=80, MSE=224.50)

Fig. 7.7(f): Cameraman Image after decompression (cut off=100, MSE=284.14)

Fig. 7.7: Simulation results of the Cameraman Image with 256x256 pixels at 'a' =1 for varying cutoff

Fig. 7.8 (a): The Original Cameraman Image

Fig. 7.8(b): Cameraman Image after decompression (a= 0.1, PSNR=5.35)

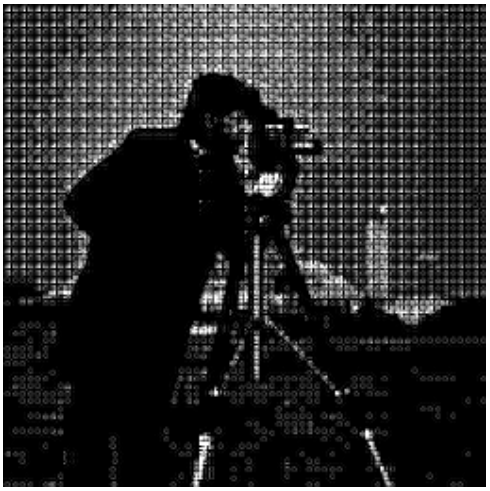


Fig. 7.8(c): Cameraman Image after decompression (a= 0.3, PSNR=7.43)

Fig. 7.8(d): Cameraman Image after decompression (a= 0.5, PSNR=9.81)

Fig. 7.8: Simulation results of the Cameraman Image with 256x256 pixels at 'a' =1 for varying ‘a’

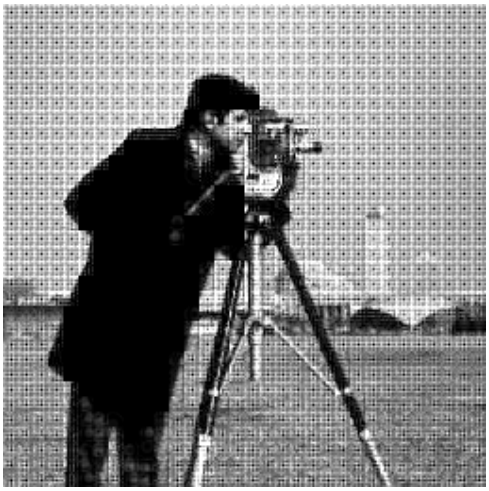


Fig. 7.8(e): Cameraman Image after decompression (a= 0.7, PSNR=18.23)

Fig. 7.8(f): Cameraman Image after decompression (a= 0.9, PSNR=28.44)

Fig. 7.8(g): Cameraman Image after decompression (a= 0.95, PSNR=27.17)

Fig. 7.8(h): Cameraman Image after decompression (a= 1, PSNR=25.67)

Fig. 7.8: Simulation results of the Cameraman Image with 256x256 pixels with CR = 30 for varying ‘a’

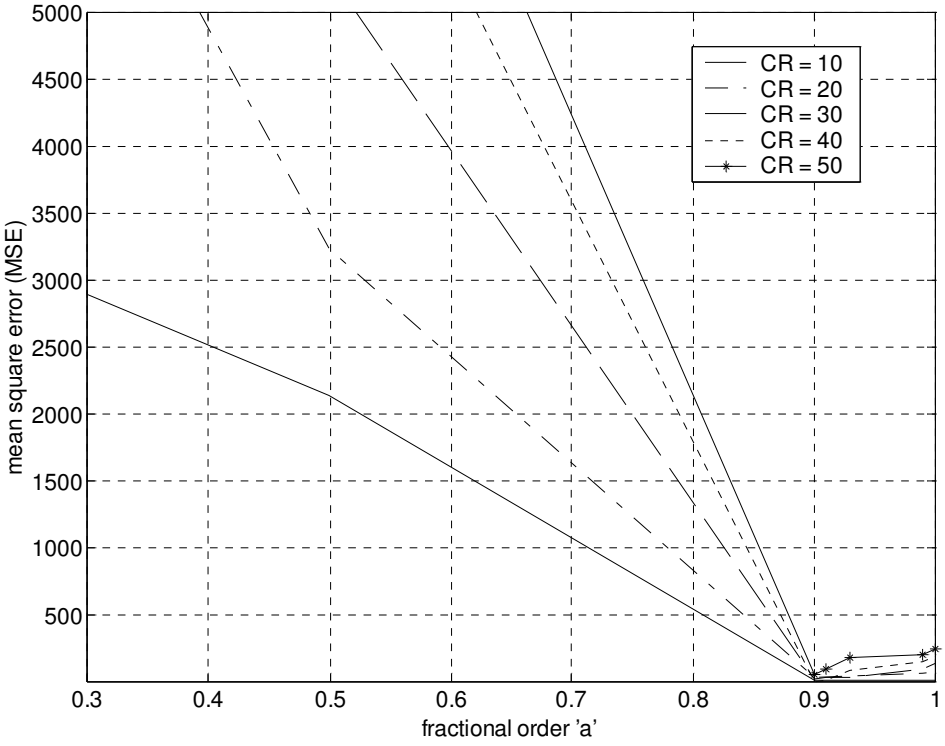


Fig. 7.9: Simulation results of the Cameraman Image with 256×256 pixels for different CR. MSE vs. fractional orders for Cameraman Image.

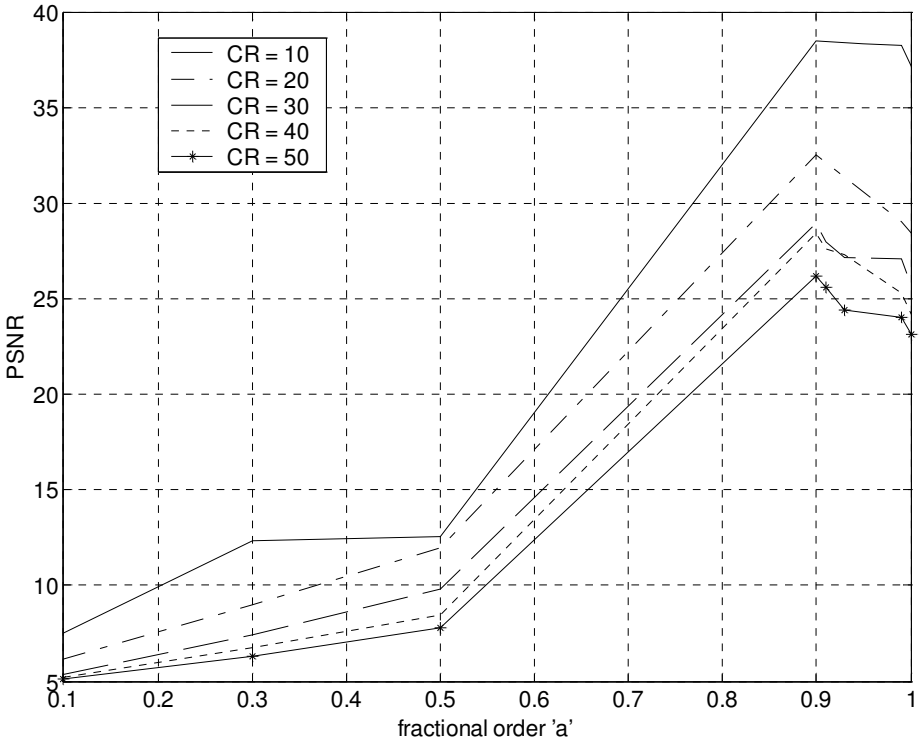


Fig. 7.10: Simulation results of the Cameraman Image with 256×256 pixels for different CR. PSNR vs. fractional orders for Cameraman Image.

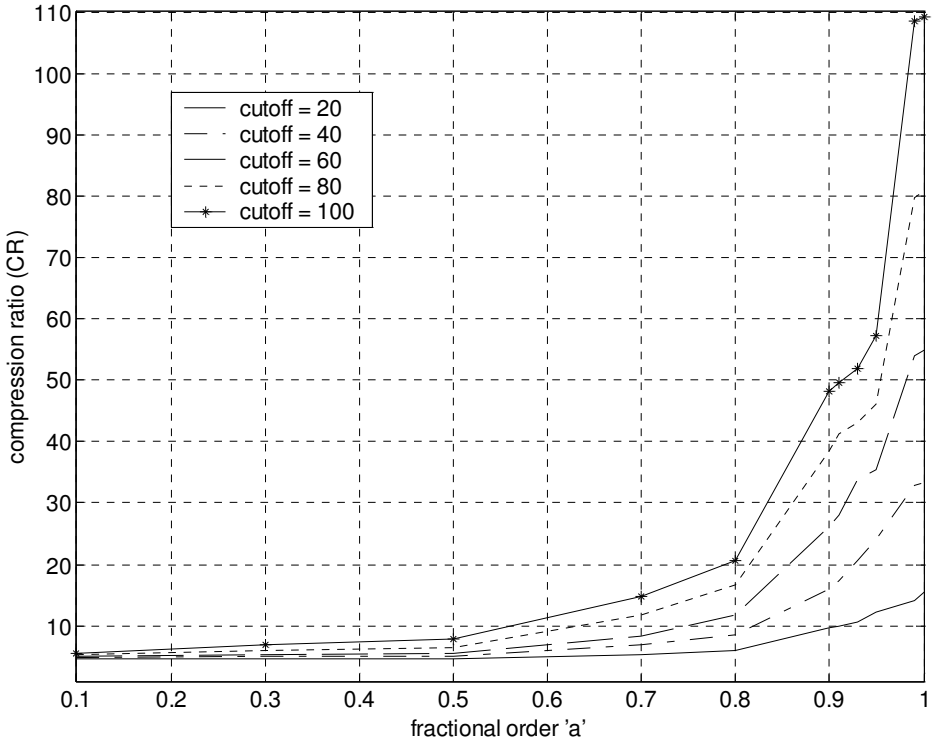


Fig. 7.11: Simulation results of the Cameraman Image with 256×256 pixels for different cut off. CR vs. fractional orders for Cameraman Image.

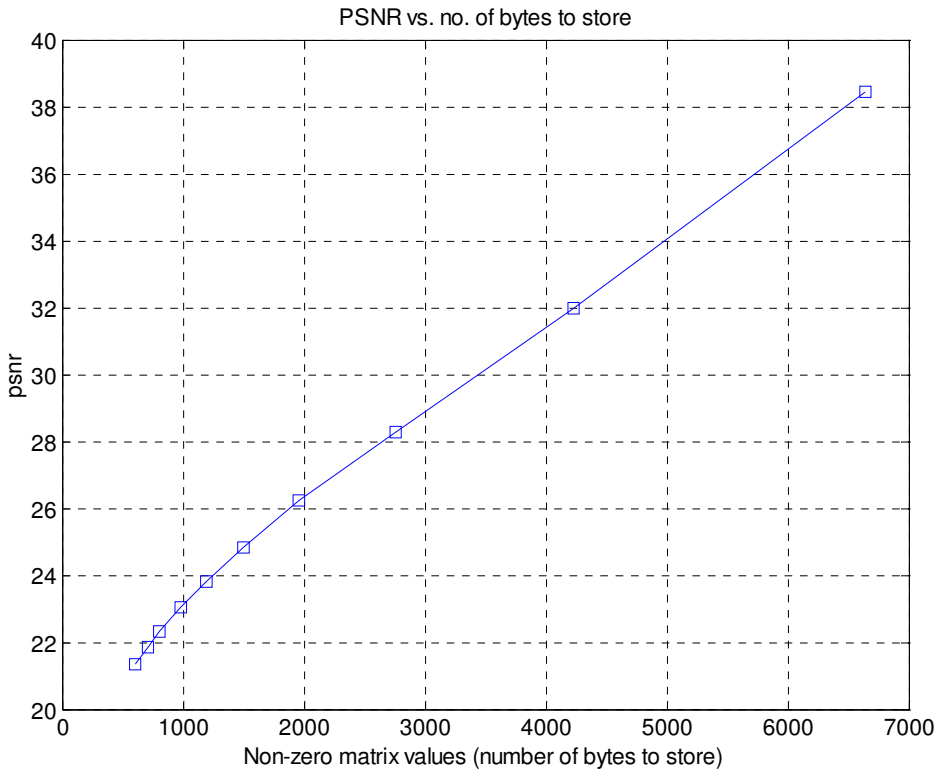


Fig. 7.12: Simulation results of the Cameraman Image with 256×256 pixels. PSNR vs. No. of bytes to store for Cameraman Image.

3) Barbara Image

a) Effect of varying cutoff

Fig. 7.13 (a)-(f) shows the decompressed Barbara image after a cutoff of 20, 40, 60, 80 and 100 respectively at 'a'=1. It is clear from these figures as cutoff increases, compression ratio increases (CR), mean square error (MSE) increases so image quality degrades. As compression increases, quality of image degrades. At small cutoff image quality is best but as cutoff increases image quality is very poor.

b) Effect of varying 'a'

Fig. 7.14 (a)-(h) shows the results of Barbara image by changing the value of parameter 'a' by keeping CR=30. It is clear that at 'a'=0.1, PSNR is very low so image quality is very poor. As 'a' increases, MSE decreases, PSNR increases therefore image quality improves. At 'a'=0.9 we get the optimum domain.With further increase in 'a', MSE increases so image quality degrades.

c) Effect of varying CR

The curves of MSE and PSNR versus the changes of fractional order 'a' for different CR have been calculated and depicted in Fig. 7.15, 7.16 respectively. It is clear from above that for same CR as we increase 'a' MSE decreases, PSNR increases till 'a'=0.9 after that as 'a' increases, MSE increases, PSNR decreases. As we increase CR, by increasing the value of cutoff, MSE increases. It is clear from graphs that for every CR at 'a'=0.9 is an optimum domain.

d) Effect of varying cutoff and 'a'

The curves of CR versus the changes of fractional order 'a' for different CO have been calculated and depicted in Fig. 7.17. It is clear that for same cutoff as we increase 'a', CR increases. So there is flexibility in FRFT to increase CR without increases cutoff value. To get more compression both cutoff and 'a' varies. For cutoff=100 and 'a'=1 we can achieve CR of 92:1 but image quality degrades.

e) PSNR versus no. of bytes to store

The curves of PSNR versus no. of bytes to store the images have been calculated and depicted in Fig. 7.18. It is clear from figure that as no. of bytes to store increases, CR decreases, MSE decreases and PSNR improve.

Fig. 7.13 (a): The Original Barbara Image

Fig. 7.13(b): Barbara Image after decompression (cut off=20, MSE=57.26)

Fig. 7.13(c): Barbara Image after decompression (cut off=40, MSE=194.23)

Fig. 7.13(d): Barbara Image after decompression (cut off=60, MSE=338.90)

Fig. 7.13: Simulation results of the Barbara Image with 512x512 pixels at 'a'=1 for varying cutoff

Fig. 7.13(e): Barbara Image after decompression
(cut off=80, MSE=470.01)

Fig. 7.13(f): Barbara Image after decompression
(cut off=100, MSE=594.49)

Fig. 7.13: Simulation results of the Barbara Image with 512x512 pixels at ‘a’=1 for varying cutoff

Fig7.14(a): The Original Barbara Image

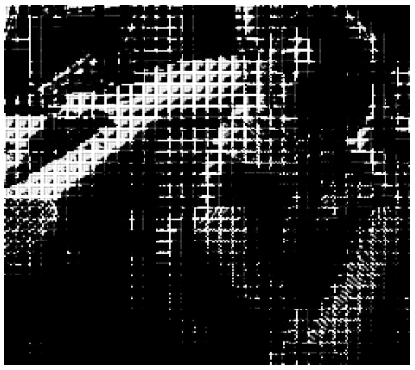


Fig. 7.14(b): Barbara Image after decompression
(a= 0.1, PSNR=5.05)

Fig. 7.14(c): Barbara Image after decompression
(a= 0.3, PSNR=7.28)

Fig. 7.14(d): Barbara Image after decompression
(a= 0.5, PSNR=9.13)

Fig. 7.14: Simulation results of the Barbara Image with 512x512 pixels with CR = 30 for varying 'a'

Fig. 7.14(e): Barbara Image after decompression
(a= 0.7, PSNR= 15.94)

Fig. 7.14(f): Barbara Image after decompression
(a= 0.9, PSNR=27.94)

Fig. 7.14(g): Barbara Image after decompression
(a= 0.95, PSNR=26.75)

Fig. 7.14(h): Barbara Image after decompression
(a= 1, PSNR=23.67)

Fig. 7.14: Simulation results of the Barbara Image with 512x512 pixels with CR = 30 for varying ‘a’

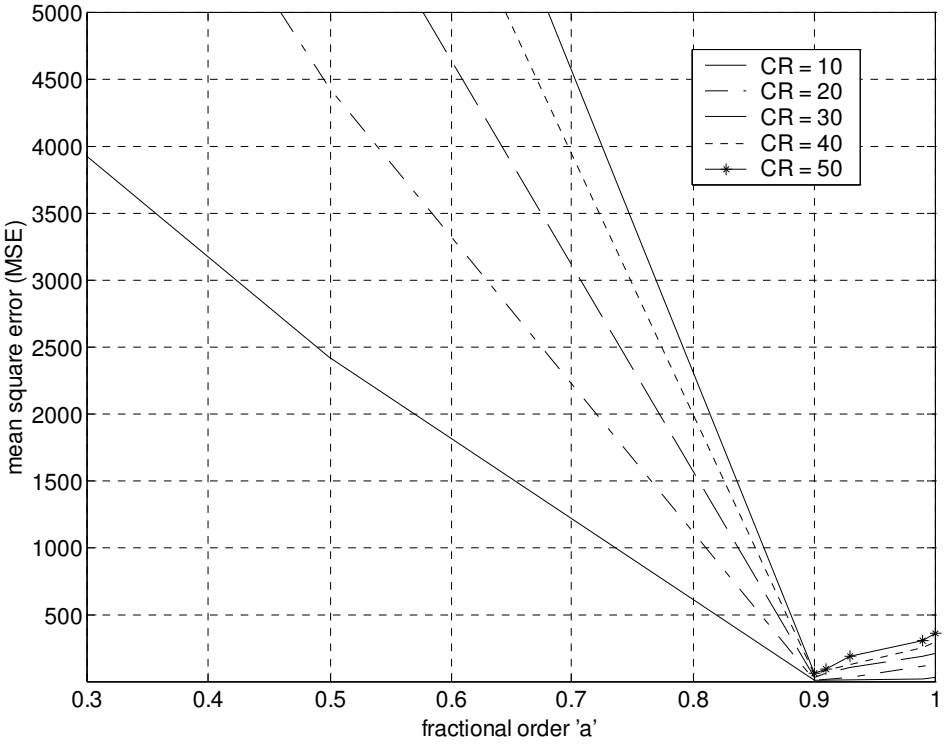


Fig. 7.15: Simulation results of the Barbara Image with 512×512 pixels for different CR. MSE vs. fractional orders for Barbara Image.

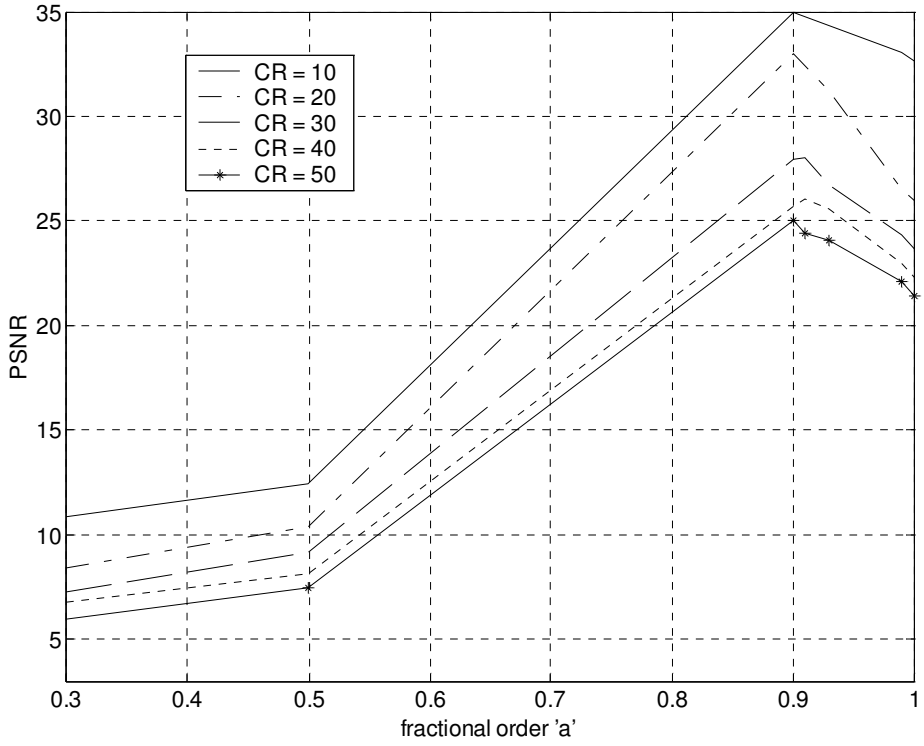


Fig. 7.16: Simulation results of the Barbara Image with 512×512 pixels for different CR. PSNR vs. fractional orders for Barbara Image.

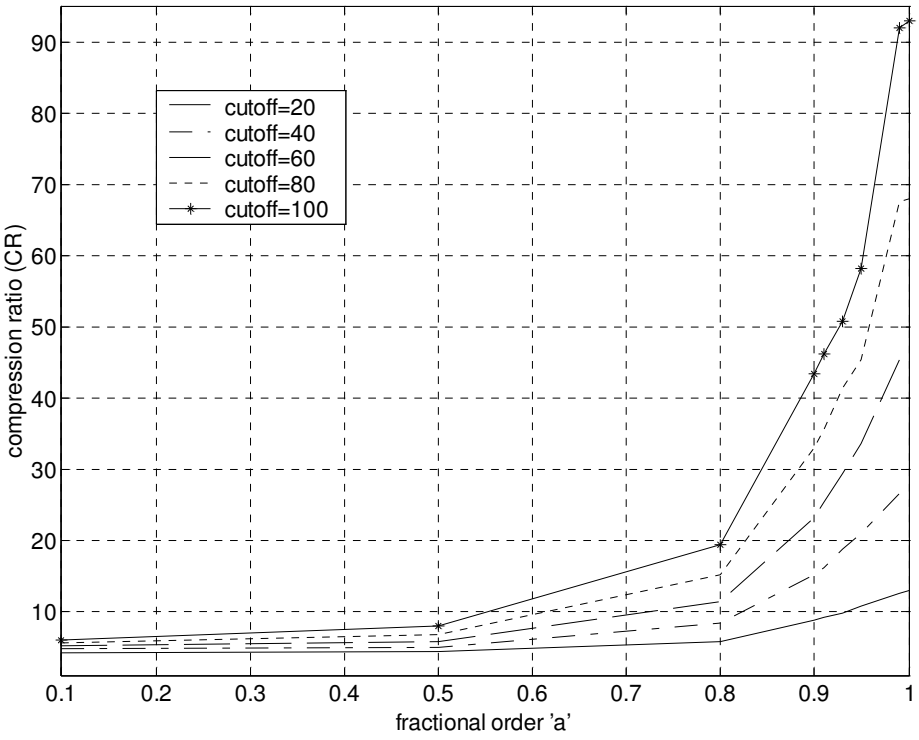


Fig. 7.17: Simulation results of the Barbara Image with 512×512 pixels for different cut off. CR vs. fractional orders for Barbara Image.

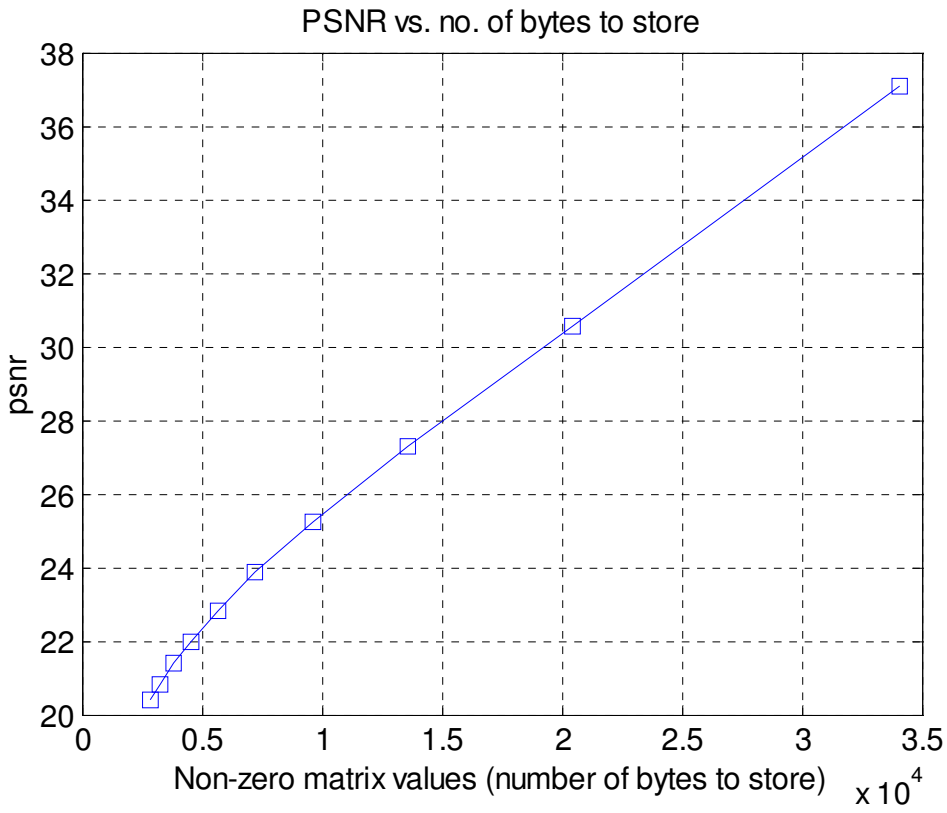


Fig. 7.18: Simulation results of the Barbara Image with 512×512 pixels. PSNR vs. No. of bytes to store for Barbara Image.

4) Rice Image

a) Effect of varying cutoff

Fig. 7.19 (a)-(f) shows the decompressed Rice image after a cutoff of 20, 40, 60, 80 and 100 respectively at 'a'=1. It is clear from these figures as cutoff increases, compression ratio increases (CR), mean square error (MSE) increases so image quality degrades. Generally there is tradeoff between compression ratio and image quality. For small cutoff, compression is less so image quality is best. As cutoff increases, compression increases correspondingly so image quality degrades.

b) Effect of varying 'a'

Fig. 7.20 (a)-(h) shows the results of Rice image by changing the value of parameter 'a' by keeping CR=30. It is clear that at 'a'=0.1, PSNR is very low so image quality is very poor. As 'a' increases, MSE decreases, PSNR increases therefore image quality improves. At 'a'=0.9 we get the optimum domain for which MSE is very low, PSNR is very high and better-decompressed image is retained. With further increase in 'a', MSE increases so image quality degrades.

c) Effect of varying CR

The curves of MSE and PSNR versus the changes of fractional order 'a' for different CR have been calculated and depicted in Fig. 7.21, 7.22 respectively. It is clear from above that for same CR as we increase 'a' MSE decreases, PSNR increases till 'a'=0.9 after that as 'a' increases, MSE increases, PSNR decreases. As we increase CR, by increasing the value of cutoff, MSE increases. It is clear from graphs that for every CR at 'a'=0.9 is an optimum domain.

d) Effect of varying cutoff and 'a'

The curves of CR versus the changes of fractional order 'a' for different CO have been calculated and depicted in Fig. 7.23. It is clear that for same cutoff as we increase 'a', CR increases. So there is flexibility in FRFT to increase CR without increases cutoff value. To get more compression both cutoff and 'a' varies. For cutoff=100 and 'a'=1 we can achieve CR of 102:1 but image quality degrades.

e) PSNR versus no. of bytes to store

The curves of PSNR versus no. of bytes to store the images have been calculated and depicted in Fig. 7.24. It is clear from figure that as no. of bytes to store increases, CR decreases, MSE decreases and PSNR improve.

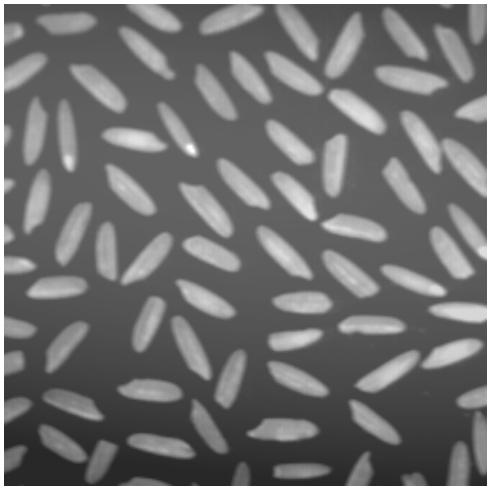


Fig. 7.19(a): The Original Rice Image

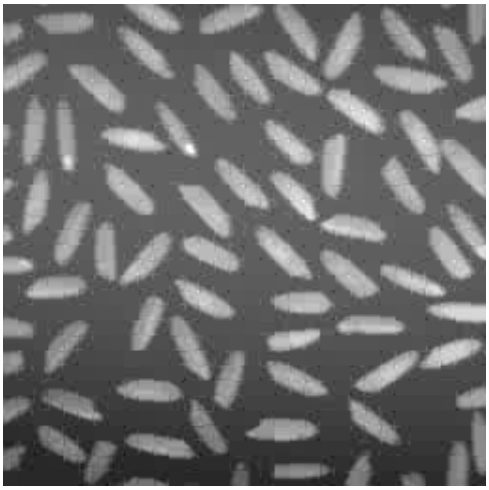


Fig. 7.19(b): Rice Image after decompression (cutoff = 20, MSE =36.06)

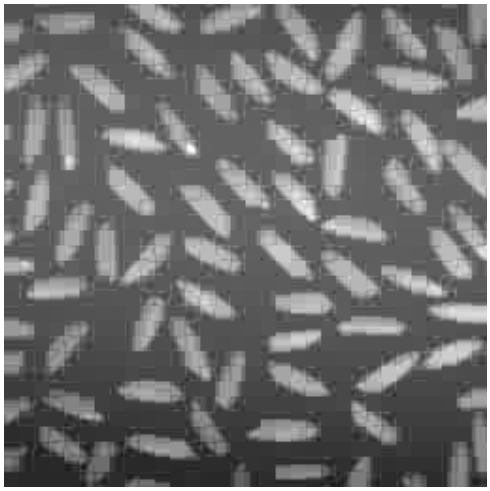


Fig. 7.19(c): Rice Image after decompression (cutoff = 40, MSE =97.44)

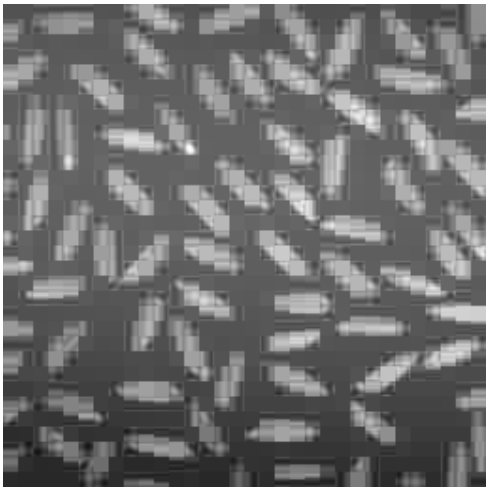


Fig. 7.19(d): Rice Image after decompression (cutoff = 60, MSE =178.35)

Fig. 7.19: Simulation results of the Rice Image with 256x256 pixels at 'a' =1 for varying cutoff

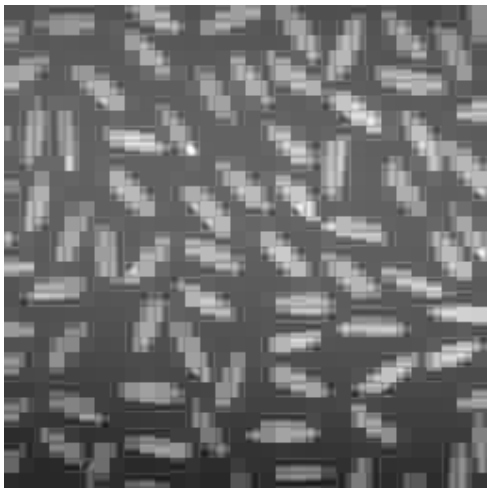


Fig. 7.19(e): Rice Image after decompression
(cutoff = 80, MSE =242.90)

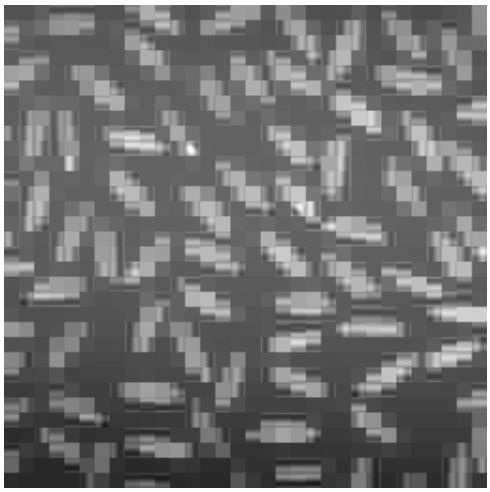


Fig. 7.19(f): Rice Image after decompression
(cutoff = 100, MSE = 289.71)

Fig. 7.19: Simulation results of the Rice Image with 256x256 pixels at 'a' =1 for varying cutoff

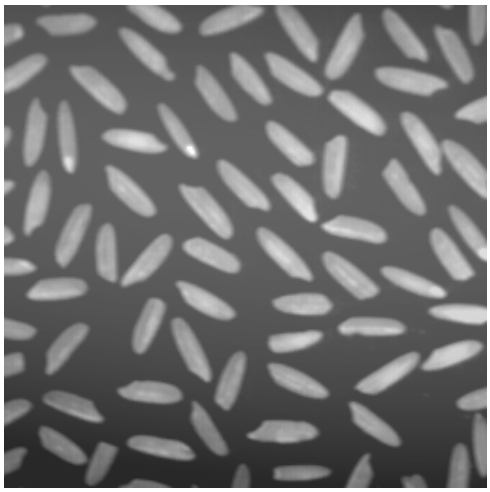


Fig. 7.20(a): The Original Rice Image

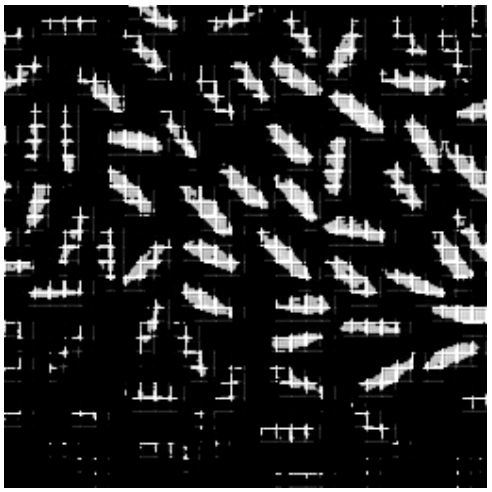


Fig. 7.20(b): Rice Image after decompression
(a= 0.1, PSNR=7.03)

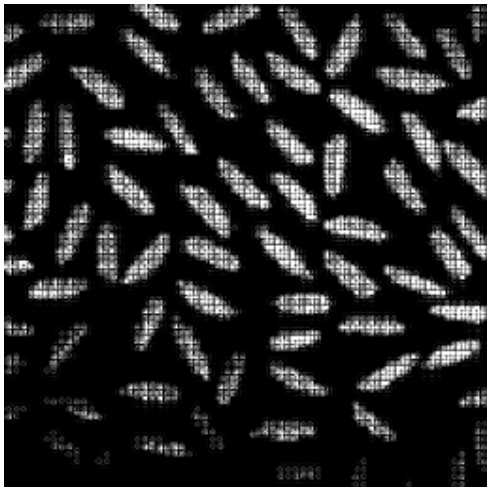


Fig. 7.20(c): Rice Image after decompression
(a= 0.3, PSNR=8.77)

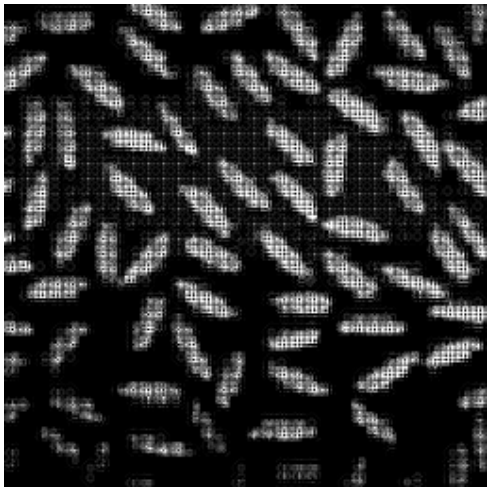


Fig. 7.20(d): Rice Image after decompression
(a= 0.5, PSNR=10.27)

Fig. 7.20: Simulation results of the Rice Image with 256x256 pixels with CR = 30 for varying ‘a’

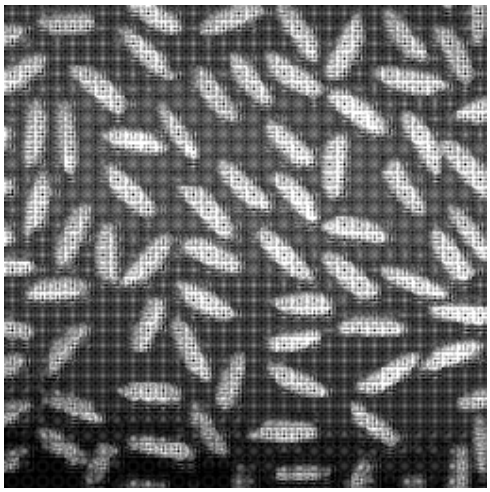


Fig. 7.20(e): Rice Image after decompression
(a= 0.7, PSNR=16.31)

Fig. 7.20(f): Rice Image after decompression
(a= 0.9, PSNR=30.79

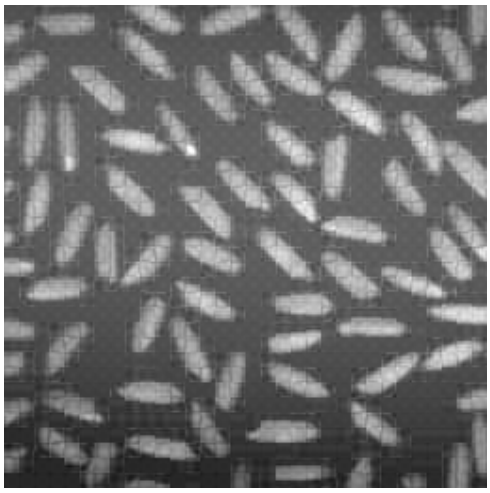


Fig. 7.20(g): Rice Image after decompression
(a= 0.95, PSNR=26.59)

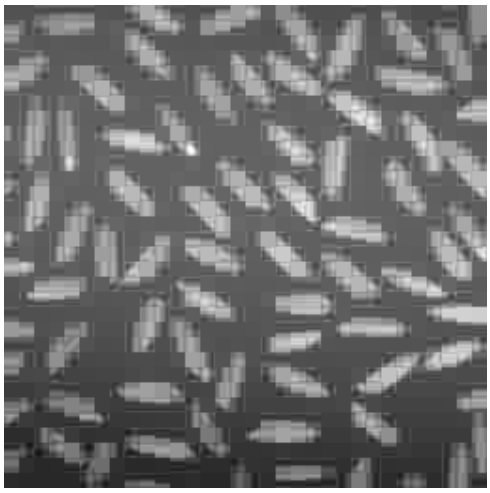


Fig. 7.20(h): Rice Image after decompression
(a= 1, PSNR=24.39)

Fig. 7.20: Simulation results of the Rice Image with 256x256 pixels with CR = 30 for varying ‘a’

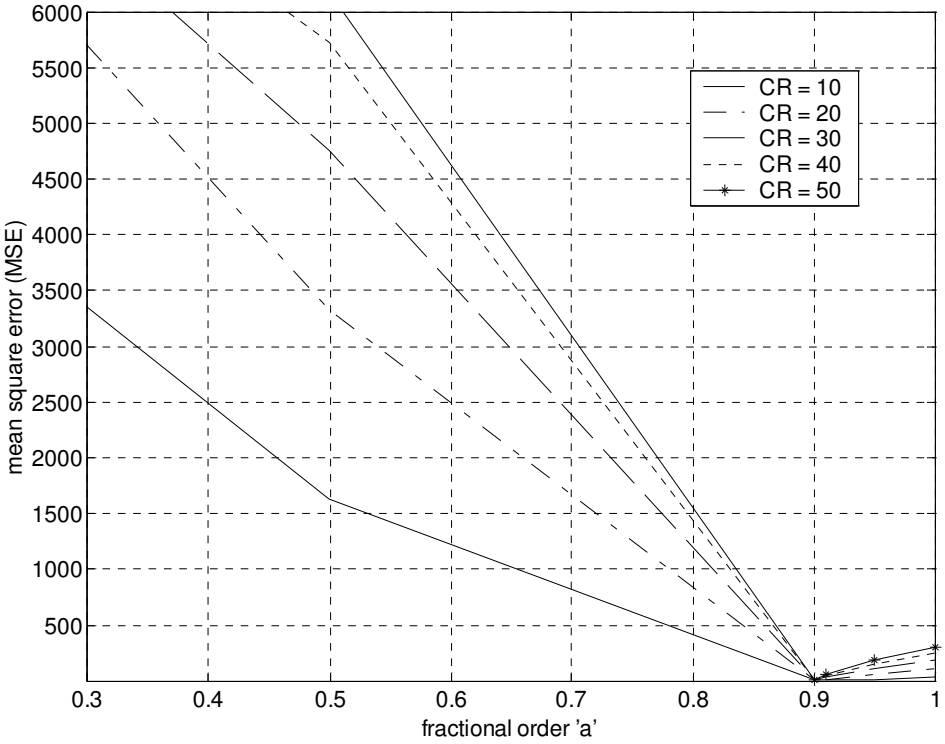


Fig. 7.21: Simulation results of the Rice Image with 256×256 pixels for different CR. MSE vs. fractional orders for Rice Image.

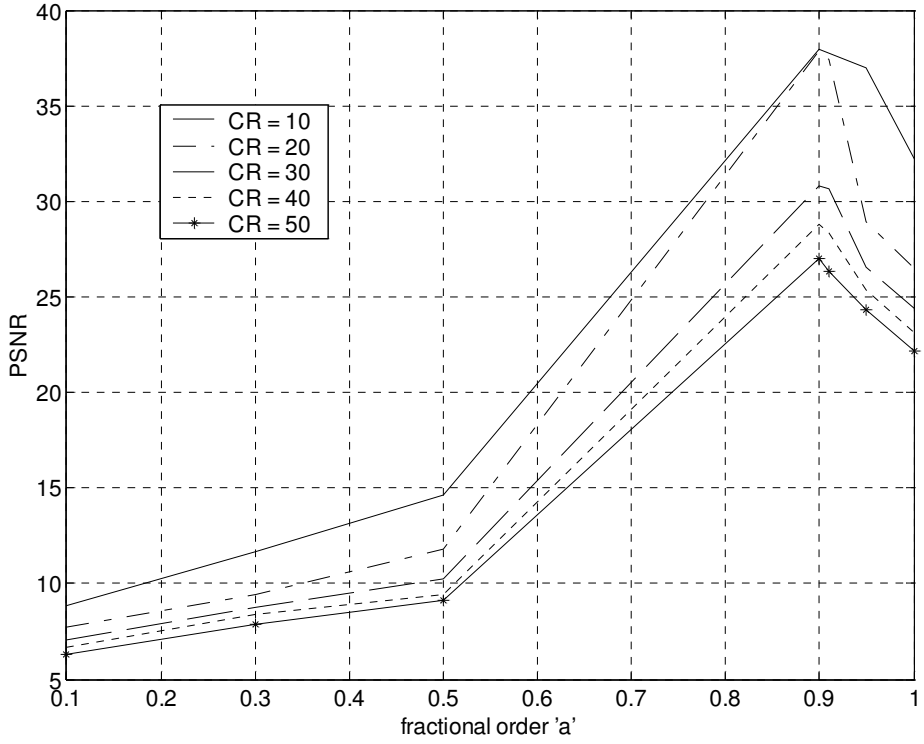


Fig. 7.22: Simulation results of the Rice Image with 256×256 pixels for different CR. PSNR vs. fractional orders for Rice Image.

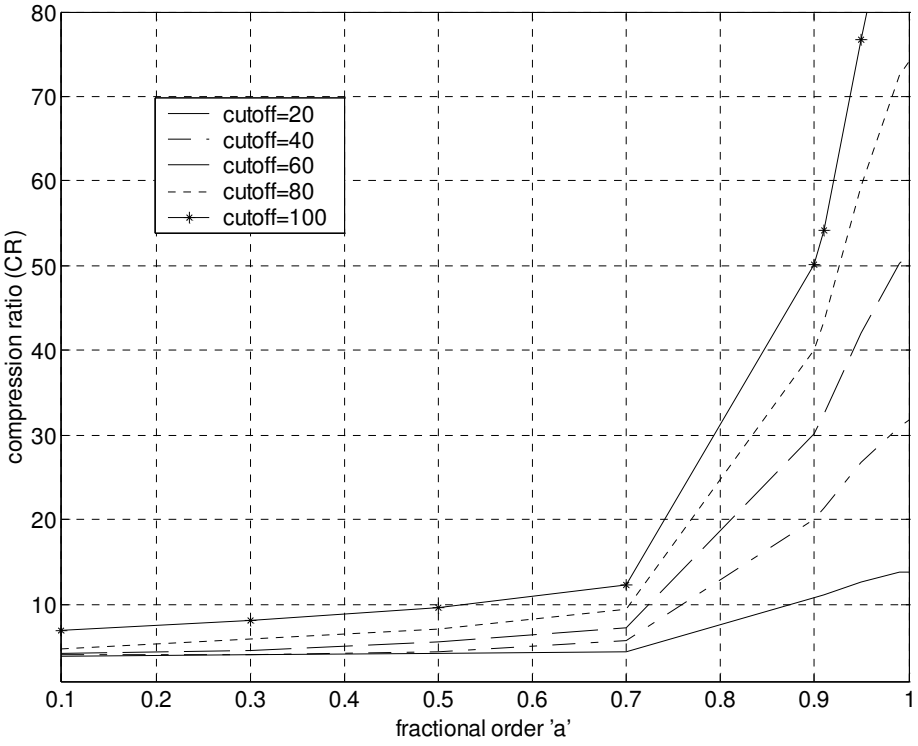


Fig. 7.23: Simulation results of the Rice Image with 256×256 pixels for different cut off. CR vs. fractional orders for Rice Image.

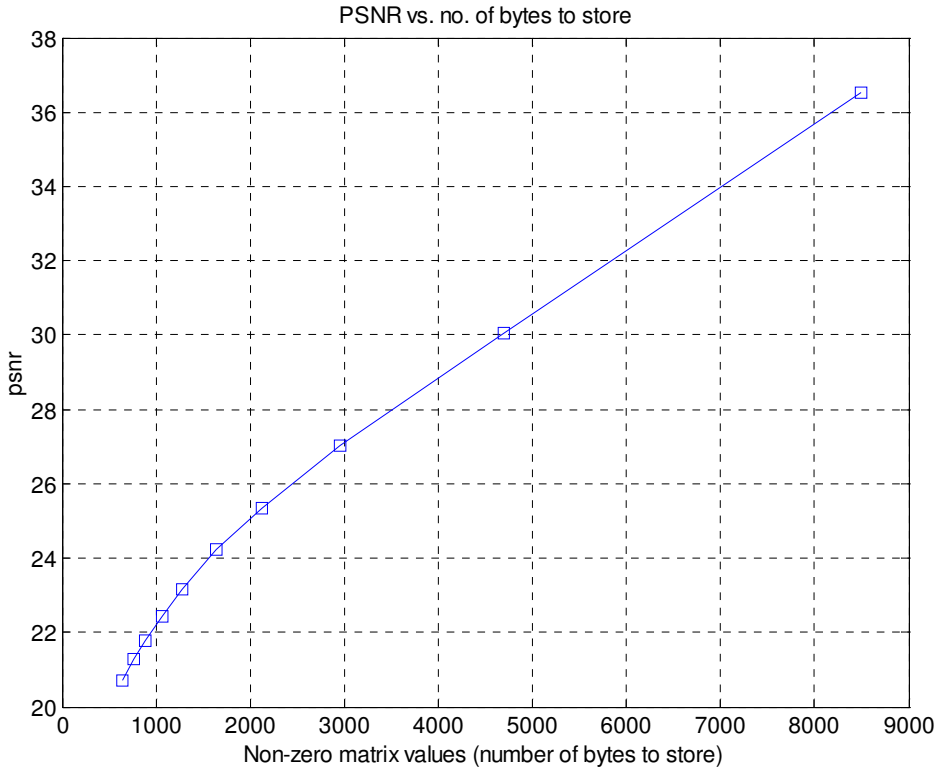


Fig. 7.24: Simulation results of the Rice Image with 256×256 pixels.
PSNR vs. No. of bytes to store for Rice Image.

Table 7.1: Table of values for Lenna Image

Compression Ratio (CR)	**a = 0.91**	**a = 1**

	MSE	PSNR	MSE	PSNR
20	5.62	38.61	92.79	27.37
30	28.32	30.51	152.42	25.21
40	29.80	28.37	201.99	23.99
50	30.97	27.92	243.48	23.18

Table 7.2: Table of values for Cameraman Image

Compression Ratio (CR)	a = 0.9		a = 1	
	MSE	PSNR	MSE	PSNR
20	5.6	34.28	72.28	28.45
30	22.57	28.44	137.09	25.67
40	28.59	27.94	193.32	24.18
50	52.16	26.17	245.48	23.14

Table 7.3: Table of values for Barbara Image

Compression Ratio (CR)	a = 0.9		a = 1	
	MSE	PSNR	MSE	PSNR
20	8.13	32.16	160.18	26.31

30	29.15	27.94	217.54	23.67
40	54.27	25.72	297.82	22.30
50	60.67	24.39	363.52	21.44

Table 7.4: Table of values for Rice Image

Compression Ratio (CR)	**a = 0.9**		**a = 1**	
	MSE	**PSNR**	**MSE**	**PSNR**
20	5.16	37.86	112.93	26.52
30	7.65	30.79	184.04	24.39
40	5.58	28.82	246.29	23.13
50	12.54	26.99	305.69	22.19

Table 7.5: Comparison of Images in FRFT and FT ('a'=1) domain for CR=50.

DOMAIN / **IMAGE**	**'a'=0.91**	**'a'=1**

	MSE	PSNR	MSE	PSNR
LENNA	30.97	27.92	243.48	23.18
CAMERAMAN	95.61	25.59	245.48	23.14
BARBARA	97.17	25.05	363.52	21.44
RICE	54.26	26.31	305.69	22.19

It is clear from tables 7.1-7.5 that for all the images with increase in CR, there is an increase in MSE and correspondingly there is decrease in PSNR. For same CR, by using FRFT, MSE reduces to nearly 8 times as that of FT for all the images. For CR=50, Lenna image gives better result at 'a'=0.91 as compared to other three images. However at 'a'=0.9, Rice image gives better results as compared to other three images.

CHAPTER –8

CONCLUSION AND FUTURE SCOPE OF WORK

In image an important part is the compression. Image compression reduces the amount of data required to represent the image. In image compression using FRFT it is observed that

FRFT makes the full use of the additional degree of freedom provided by its fractional order ‘a’ to achieve an optimum domain for which compression is more, MSE is less and better-decompressed image is retained. By varying parameter ‘a’ we can achieve same amount of compression as that of FT, but FRFT gives MSE which is nearly 8 times less as compared to FT. It is clear that for all images $0.9<a<1$ is an optimum domain for which better-decompressed image is retained.

In this thesis the transform implemented is the FRFT. The suitability of various other fractional integral transforms like Fractional Cosine Transform (FRCT) may be examined and the results may be plotted as a function of the overall compression ratio and the quality of reconstructed image. For further compression the quantized coefficient can be entropy encoded. There exist many entropy-coding schemes in literature such as Arithmetic coding, Huffman coding and Run-length coding etc. which can be implemented to achieve further better results.

REFERENCES

[1] R.C. Gonzalez & R. E. Woods, “Digital Image Processing”, Addison-Wesley Co., 1991.

[2] A.K.Jain, “Fundamentals of Digital Image Processing”, Prentice-Hall Inc,Englewood Cliffs,1989.

[3] Victor Namias, “The Fractional Order Fourier Transform and its Applications to Quantum Mechanics”, J. Inst. Math Applications, vol 25, pp 241-265, 1980.

[4] Victor Namias, “Fractionalization of Hankel Transforms”, J. Inst. Math Applications, vol 26, pp 187-197, 1980.

[5] A.C.McBride and F.H.Keer, “On Namia's Fractional Fourier Transform”, IMA J. Appl. Math, vol 239, pp 159-175, 1987.

[6] David Mendlovic and H.M.Ozaktas, “Fractional Fourier Transforms and their optical Implementation-I”, J. Opt. Soc. of Am.-A, vol 10, no. 9, pp 1875-1881, 1993.

[7] H.M.Ozaktas and David Mendlovic, “Fractional Fourier Transforms and their optical Implementation-II”, J. Opt. Soc. of Am.-A, vol 10, no. 12, pp 2522-2531, 1993.

[8] H.M.Ozaktas, O.Arikan, M.A.Kutay and G.Bozdagi, “Digital Computation of the Fractional Fourier Transforms”, IEEE Trans. on Signal Processing, vol 44, no. 9, pp 2141-2150, 1996.

[9] S.C.Pei, M.H.Yeh and C.C. Tseng, “Discrete fractional Fourier transform Based on Orthognal Projections”, IEEE Trans. on Signal Processing, vol 47, no. 2, pp 1335-1348, 1999.

[10] C.Candan, M.A.Kutay and H.M. Ozaktas, “Discrete fractional Fourier transform”, IEEE Trans. on Signal Processing, vol. 48, no. 5, pp. 1329-1337,2000.

[11] D.A.Huffman, A Method for the construction of Minimum-redundancy Codes, Proc. IRE, vol.40, no.10, pp.1098-1101,1952.

[12] A.B.Watson, “Image Compression using the DCT”, Mathematica Journal, 1995, pp.81-88.

[13] V.A. Narayanan, K.M.M.Prabhu, “The fractional Fourier transform: theory, implementation and error analysis”, Microprocessors and Microsystems 27(2003),511-521.

[14] A.Bultheel and H.Martinez, “A shattered survey of the Fractional Fourier Transform”, Preprint April 28, 2002.

[15] H. M. Ozaktas, Z. Zalevsky and M. A. Kutay, “The Fractional Fourier Transform with Applications in Optics and Signal Processing”, John Wiley & Sons, 2001

[16] I. S. Yetik, M.A. Kutay, H.M.Ozaktas, “Image representation and compression with the fractional Fourier transform”, Opt. Commun. 197(2001) 275-278.

[17] Adolf W. Lohmann, “Image Rotation, Wigner Rotation and The Fractional Fourier Transform”, J. Opt. Soc. of Am.-A, vol 10, no. 10, pp 2181-2186, 1993.

[18] L.B.Almedia, “The Fractional Fourier Transform and time-frequency representations”, IEEE Trans. on Signal Processing, vol 42, pp. 3084-3091, 1994.

[19] C.C.Chen, “On the Selection of Image Compression Algorithms”, Proc. International Conference on Pattern Recognition, Brisbane, Australia Aug1998.

[20] A.I. Zayed , “On the Relationship Between the Fourier and Fractional Fourier Transforms”,IEEE Signal Processing Letters,vol.3, no. 12,1996.