**UGC PROJECT REPORT**

On

FAST FRACTAL COMPRESSION USING VARIANCE ORDERED BLOCK SEARCH ALGORITHM

*Submitted By-*

MOHAMMED

*In partial fulfilment of requirement*

*for the fellowship of UGC*

in

ELECTRONICS & COMMUNICATION ENGINEERING

****

SCHOOL OF ENGINEERING UNIVERSITY GRANTS COMMISSION

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

KOCHI – 682022

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

DIVISION OF ELECTRONICS ENGINEERING

SCHOOL OF ENGINEERING UNIVERSITY GRANTS COMMISSION

****

CERTIFICATE

*Certified that this report titled*

FAST FRACTAL COMPRESSION USING VARIANCE ORDERED BLOCK SEARCH ALGORITHM

*is a bonafide record of UGC project work of*

MOHAMMED

*towards the partial fulfilment of the requirement for the award of* *fellowship of UGC* ***in Electronics and Communication Engineering*** *of Cochin University of Science & Technology.*

**Project-Guide Head**

**Mrs. Deepa Sankar Dr. R .Gopikakumari**

ACKNOWLEDGEMENT

I express my heartfelt thanks and gratitude to my parents, without whose encouragement, unwavering support and blessings, successful completion of the project would have been a distant dream. I express my genuine gratitude to my project guide **Mrs. Deepa Sankar** and Head of Department **Dr.R.Gopikakumari** for providing me the required support and encouragement for the successful completion of the project in the given time frame. Last but not the least I would like to thank all the teachers and our classmates who helped in one way or other helped my project come alive.

I above all thank **Almighty God** for having given me such a wonderful project to work on, under an equally wonderful guide.

MOHAMMED

**ABSTRACT**

A new fast fractal encoding algorithm based on the variances of image blocks is proposed. With the domain blocks sorted according to their variances, and the best matched domain block to a given range block is searched in the order that the variance-distance is closer. A great number of domain blocks could be safely rejected by the prior comparison of the current minimum distortion and variance difference between the candidate domain block and the range block during the search process. It was proved that our algorithm produces completely identical fractal codes with that of the conventional full search in reduced time. The simulation results confirmed the effectiveness of the proposed algorithm.

Fractal Image Compression

**CHAPTER 1: INTRODUCTION**

1. Introduction

With the advance of the information age the need for mass information storage and fast communication links grows. Storing images in less memory leads to a direct reduction in storage cost and faster data transmissions. These facts justify the efforts, of private companies and universities, on new image compression algorithms.

Storing an image on a computer requires a very large memory. This problem can be averted by the use of various image compression techniques. Most images contain some amount of redundancy that can be removed when the image is stored and then replaced when it is reconstructed.

Fractal image compression is a recent technique based on the representation of an image. The self-transformability property of an image is assumed and exploited in fractal coding. It provides high compression ratios and fast decoding. Apart from this it is also simple and is an easily executable technique.

Images are stored on computers as collections of bits (a bit is a binary unit of information which can answer “yes” or “no” questions) representing pixels or points forming the picture elements. Since the human eye can process large amounts of information (some 8 million bits), many pixels are required to store moderate quality images. These bits provide the “yes” and “no” answers to the 8 million questions that determine the image.

Most data contains some amount of redundancy, which can sometimes be removed for storage and replaced for recovery, but this redundancy does not lead to high compression ratios. An image can be changed in many ways that are either not detectable by the human eye or do not contribute to the degradation of the image.

The standard methods of image compression come in several varieties. The current most popular method relies on eliminating high frequency components of the signal by storing only the low frequency components (Discrete Cosine Transform Algorithm). This method is used on JPEG (still images), MPEG (motion video images), H.261 (Video Telephony on ISDN lines), and H.263 (Video Telephony on PSTN lines) compression algorithms.

Fractal Compression was first promoted by M.Barnsley, who founded a company based on fractal image compression technology but who has not released details of his scheme. The first public scheme was due to E.Jacobs and R.Boss of the Naval Ocean Systems Centre in San Diego who used regular partitioning and classification of curve segments in order to compress random fractal curves (such as political boundaries) in two dimensions. A doctoral student of Barnsley’s, A. Jacquin, was the first to publish a similar fractal image compression scheme.

1.1 A Brief History of Fractal Image Compression

The birth of fractal geometry (or rebirth, rather) is usually traced to IBM mathematician Benoit B. Mandelbrot and the 1977 publication of his seminal book “The Fractal Geometry of Nature”. The book put forth a powerful thesis: traditional geometry with its straight lines and smooth surfaces does not resemble the geometry of trees and clouds and mountains. Fractal geometry, with its convoluted coastlines and detail ad infinitum, does.

This insight opened vast possibilities. Computer scientists, for one, found a mathematics capable of generating artificial and yet realistic looking landscapes, and the trees that sprout from the soil. And mathematicians had at their disposal a new world of geometric entities.

It was not long before mathematicians asked if there was a unity among this diversity. There is, as John Hutchinson demonstrated in 1981, it is the branch of mathematics now known as Iterated Function Theory. Later in the decade Michael Barnsley, a leading researcher from Georgia Tech, wrote the popular book “Fractals Everywhere”. The book presents the mathematics of Iterated Functions Systems (IFS), and proves a result known as the Collage Theorem. The Collage Theorem states what an Iterated Function System must be like in order to represent an image.

This presented an intriguing possibility. If, in the forward direction, fractal mathematics is good for generating natural looking images, then, in the reverse direction, could it not serve to compress images? Going from a given image to an Iterated Function System that can generate the original (or at least closely resemble it), is known as the inverse problem. This problem remains unsolved.

Barnsley, however, armed with his Collage Theorem, thought he had it solved. He applied for and was granted a software patent and left academia and found Iterated Systems Incorporated (US patent 4,941,193. Alan Sloan is the co-grantee of the patent and co-founder of Iterated Systems.) Barnsley announced his success to the world in the January 1988 issue of BYTE magazine. This article did not address the inverse problem but it did exhibit several images purportedly compressed in excess of 10,000:1. Alas, it was a slight of hand. The images were given suggestive names such as "Black Forest" and "Monterey Coast" and "Bolivian Girl" but they were all manually constructed. Barnsley's patent has come to be derisively referred to as the "graduate student algorithm."

Attempts to automate this process have continued to this day, but the situation remains bleak. As Barnsley admitted in 1988: "Complex color images require about 100 hours each to encode and 30 minutes to decode on the Masscomp [dual processor workstation]." That's 100 hours with a person guiding the process.

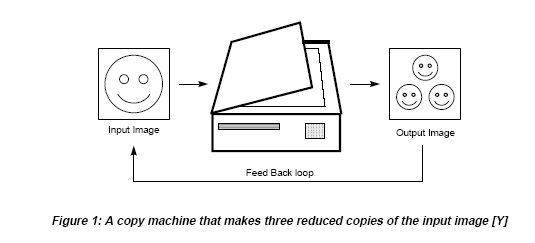
Ironically, it was one of Barnsley's PhD students that made the graduate student algorithm obsolete. In March 1988, according to Barnsley, he arrived at a modified scheme for representing images called Partitioned Iterated Function Systems (PIFS). Barnsley applied for and was granted a second patent on an algorithm that can automatically convert an image into a Partitioned Iterated Function System, compressing the image in the process. (US patent 5,065,447. Granted on Nov. 12 1991.) For his PhD thesis, Arnaud Jacquin implemented the algorithm in software, a description of which appears in his landmark paper "Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations." The algorithm was not sophisticated, and not speedy, but it was fully automatic. This came at price: gone was the promise of 10,000:1 compression. A 24-bit color image could typically be compressed from 8:1 to 50:1 while still looking "pretty good." Nonetheless, all contemporary fractal image compression programs are based upon Jacquin's paper.

That is not to say there are many fractal compression programs available. There are not. Iterated Systems sell the only commercial compressor/decompressor, an MS-Windows program called "Images Incorporated." There are also an increasing number of academic programs being made freely available. Unfortunately, these programs are of merely academic quality.

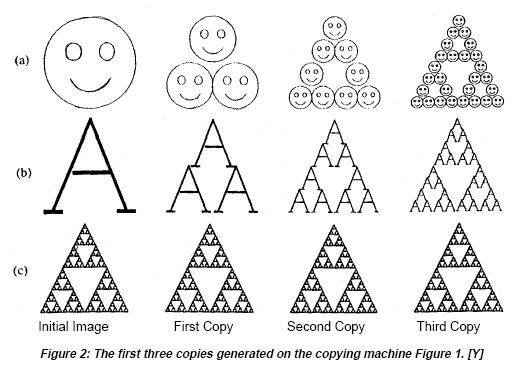
This scarcity has much to do with Iterated Systems tight lipped policy about their compression technology. They do, however, sell a Windows DLL for programming. In conjunction with independent development by researchers elsewhere, therefore, fractal compression will gradually become more pervasive. Whether it becomes all-pervasive remains to be seen.

# **CHAPTER 2: WHAT IS FRACTAL IMAGE COMPRESSION?**

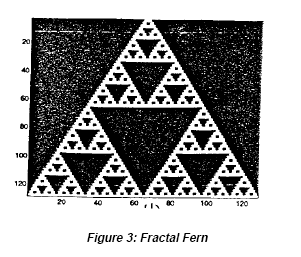
Imagine a special type of photocopying machine that reduces the image to be copied by half and reproduces it three times on the copy (see Figure 1). What happens when we feed the output of this machine back as input? Figure 2 shows several iterations of this process on several input images. We can observe that all the copies seem to converge to the same final image, the one in 2(c). Since the copying machine reduces the input image, any initial image placed on the copying machine will be reduced to a point as we repeatedly run the machine; in fact, it is only the position and the orientation of the copies that determines what the final image looks like.



The way the input image is transformed determines the final result when running the copy machine in a feedback loop. However we must constrain these transformations, with the limitation that the transformations must be contractive (see contractive box), that is, a given transformation applied to any two points in the input image must bring them closer in the copy. This technical condition is quite logical, since if points in the copy were spread out the final image would have to be of infinite size. Except for this condition the transformation can have any form.In practice, choosing transformations of the form is sufficient to generate interesting transformations called affine transformations of the plane. Each can skew, stretch, rotate, scale and translate an input image.

A common feature of these transformations that run in a loop back mode is that for a given initial image each image is formed from a transformed (and reduced) copies of itself, and hence it must have detail at every scale. That is, the images are fractals. This method of generating fractals is due to John Hutchinson.

Barnsley suggested that perhaps storing images as collections of transformations could lead to image compression. His argument went as follows: the image in Figure 3 looks complicated yet it is generated from only 4 affine transformations.

Each transformation wi *i*s defined by 6 numbers, *ai, bi, ci, di, ei,* and *fi* , see eq(1), which do not require much memory to store on a computer (4 transformations x 6 numbers /transformations x 32 bits /number = 768 bits). Storing the image as a collection of pixels, however required much more memory (at least 65,536 bits for the resolution shown in Figure 2). So if we wish to store a picture of a fern, then we can do it by storing the numbers that define the affine transformations and simply generate the fern whenever we want to see it. Now suppose that we were given any arbitrary image, say a face. If a small number of affine transformations could generate that face, then it too could be stored compactly. The trick is finding those numbers.

2.1 Why The Name “Fractal”

The image compression scheme described later can be said to be fractal in several senses. The scheme will encode an image as a collection of transforms that are very similar to the copy machine metaphor. Just as the fern has detail at every scale, so does the image reconstructed from the transforms. The decoded image has no natural size; it can be decoded at any size. The extra detail needed for decoding at larger sizes is generated automatically by the encoding transforms. One may wonder if this detail is “real”; we could decode an image of a person increasing the size with each iteration, and eventually see skin cells or perhaps atoms. The answer is, of course, no. The detail is not at all related to the actual detail present when the image was digitized; it is just the product of the encoding transforms which originally only encoded the large-scale features. However, in some cases the detail is realistic at low magnifications, and this can be useful in Security and Medical Imaging applications. Figure 4 shows a detail from a fractal encoding of “Lena” along with a magnification of the original.

2.2 Properties of Fractals

A set F is said to be a fractal if it possesses the following properties.

* F is found to contain detail at every scale.
* F is self-similar.
* The fractal dimension of F is greater than it’s topological dimension.
* F has got a simple algorithmic description.

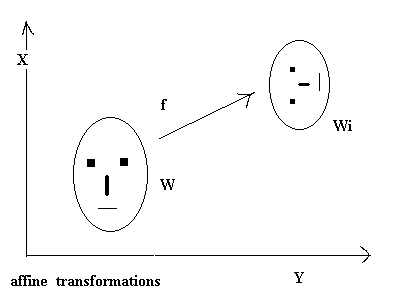
2.3 Affine Transformations

An affine transformation is any [transformation](http://mathworld.wolfram.com/Transformation.html) that preserves [collinearity](http://mathworld.wolfram.com/Collinear.html) (i.e., all points lying on a [line](http://mathworld.wolfram.com/Line.html) initially still lie on a [line](http://mathworld.wolfram.com/Line.html) after [transformation](http://mathworld.wolfram.com/Transformation.html)) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation). In this sense, affine indicates a special class of projective transformations that do not move any objects from the affine space to the plane at infinity or conversely. An affine transformation is also called an affinity.

[Geometric contraction](http://mathworld.wolfram.com/GeometricContraction.html), [expansion](http://mathworld.wolfram.com/Expansion.html), [dilation](http://mathworld.wolfram.com/Dilation.html), [reflection](http://mathworld.wolfram.com/Reflection.html), [rotation](http://mathworld.wolfram.com/Rotation.html), [shear](http://mathworld.wolfram.com/Shear.html), [similarity transformations](http://mathworld.wolfram.com/SimilarityTransformation.html), [spiral similarities](http://mathworld.wolfram.com/SpiralSimilarity.html), and [translation](http://mathworld.wolfram.com/Translation.html) are all affine transformations, as are their combinations. In general, an affine transformation is a composition of [rotations](http://mathworld.wolfram.com/Rotation.html), [translations](http://mathworld.wolfram.com/Translation.html), [dilations](http://mathworld.wolfram.com/Dilation.html), and [shears](http://mathworld.wolfram.com/Shear.html).

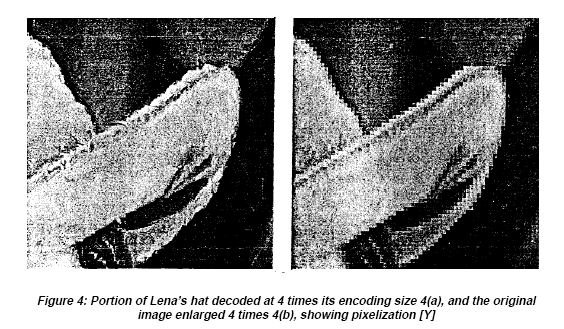
While an affine transformation preserves *proportions* on lines, it does not necessarily preserve angles or lengths. Any triangle can be transformed into any other by an affine transformation, so all triangles are affine and, in this sense, affine is a generalization of congruent and similar.

These are combinations of rotation, scaling and translation of the co-ordinate axis in an N-dimensional space. The figure shows an example of an affine transformation W which moves towards W(f)-that is it is a contractive transformation.



2.4 How much Compression can Fractal achieve?

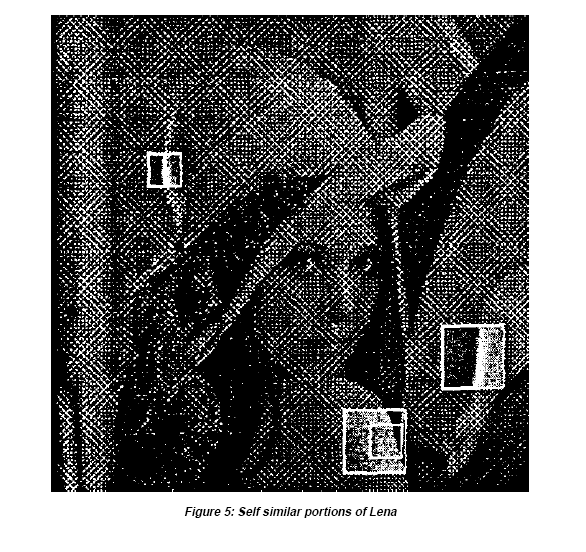
The compression ratio for the fractal scheme is hard to measure since the image can be decoded at any scale. For example, the decoded image in Figure 3 is a portion of a 5.7 to 1 compression of the whole Lena image. It is decoded at 4 times its original size, so the full decoded image contains 16 times as many pixels and hence this compression ratio is 91.2 to 1. This many seem like cheating, but since the 4-times-later image has detail at every scale, it really is not.

******

**CHAPTER 3: ENCODING IMAGES**

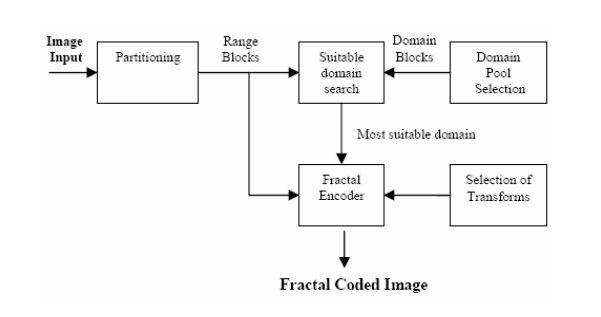
The theorem tells us that transformation *W* will have a unique fixed point in the space of all images. That is, whatever image (or set) we start with, we can repeatedly apply *W* to it and we will converge to a fixed image.

Suppose we are given an image *f* that we wish to encode. This means we want to find a collection of transformations *w1, w2, ...,wN* and want *f* to be the fixed point of the map *W*. In other words, we want to partition *f* into pieces to which we apply the transformations *wi* , and get back the original image *f* .

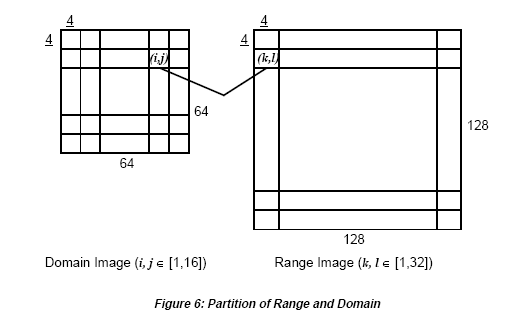
A typical image of a face does not contain the type of self-similarity like the fern. The image does contain other type of self-similarity. Figure 5 shows regions of Lena identical, and a portion of the reflection of the hat in the mirror is similar to the original. These distinctions form the kind of self-similarity; rather than having the image be formed by whole copies of the original (under appropriate affine transformations), here the image will be formed by copies of properly transformed parts of the original. These transformed parts do not fit together, in general, to form an exact copy of the original image, and so we must allow some error in our representation of an image as a set of transformations.

3.1 Proposed Algorithm

**Encoding:**

****

The following example suggests how the Fractal Encoding can be done. Suppose that we are dealing with a 128 x 128 image in which each pixel can be one of 256 levels of gray. We called this picture Range Image. We then reduce by averaging (down sampling and lowpass-filtering) the original image to 64 x 64. We called this new image Domain Image. We then partitioned both images into blocks 4 x 4 pixels (see Figure 6).



We performed the following affine transformation to each block:

In this case we are trying to find linear transformations of our Domain Block to arrive to the best approximation of a given Range Block. Each Domain Block is transformed and then compared to each Range Block *Rk,l* . The exact transformation on each domain block, i.e. the determination of a and *to* is found minimizing

where *m, n, Ns* = 2 or 4 (size of blocks).

Each transformed domain block G*(Di,j*) is compared to each range block *Rk,l* in order to find the closest domain block to each range block. This comparison is performed using the following distortion measure.

Each distortion is stored and the minimum is chosen. The transformed domain block which is found to be the best approximation for the current range block is assigned to that range block, i.e. the coordinates of the domain block along with its a and *to* are saved into the file describing the transformation. This is what is called the Fractal Code Book.

**Decoding**

The reconstruction process of the original image consists on the applications of the transformations describe in the fractal code book iteratively to some initial image Winit*,* until the encoded image is retrieved back. The transformation over the whole initial image can be described as follows:

can be expressed as two distinct transformations:

represents the down sampling and low-pass filtering of an image to create a domain image e.g. reducing a 128x128 image to a 64x64 image as we describe previously. represents the ensemble of the transformations defined by our mappings from the domain blocks in the domain image to the range blocks in the range image as recorded in the fractal. *n* will converge to a good approximation of *orig* in less than 7 iterations.

**CHAPTER 4: ITERATED FUNCTION SYSTEM**

4.1 Introduction

Let us consider the case of the photocopying machine which we saw earlier. Suppose we were to feed the output of this machine back as input and continue the process iteratively. We can find that all the output images seem to be converging back to the same output image and also that the final image is not changed by the process. We call this image the attractor for the copying machine.

Because the copying machine reduces the input image the copies of the initial image will be reduced to a point as we repeatedly feed the output back as input; there will be more and more copies but the copies at each stage get smaller and smaller. So the initial image doesn’t affect the final attractor ; in fact, it is only the position and orientation of the copies that determines what the final image will look like.

The final result is determined by the way the input image is transformed , we only describe these transformations. Different transformations lead to different attractors with the technical limitation that the images must be contractive; i.e, a given transformation applied to any point in the input image must bring them closer in the copy. This technical condition is very natural since if the points in the copy were to be spread out the attractor might have to be of infinite size.

A common feature of these attractors thus formed is that in the position of each of these images of the original square there is a transformed copy of the whole image. Thus the images thus formed are all fractals. This method of creating fractals was put forward by John Hutchinson.

M.Barnsley suggested that perhaps storing images as collections of transformations could lead to image compression. The complex figure of a fern is generated from just four affine transforms. Each affine transform is defined by six numbers ‘a’, ‘b’, ‘c’, ‘d’, ‘e’ and ‘f’. Storing these on a computer do not require much memory. Suppose we are required to store the image of a face. If a small number of transformations could generate that face then it could be stored compactly.

If we scale the transformations defining any image the resulting attractor will also be scaled. This is the implication of fractal image compression. The extra detail needed for decoding at larger sizes is generated automatically by the encoding transforms. However in some cases the detail is realistic at low magnification and this is a useful feature of the method.

Magnification of the original shows pixelation; the dots that make up the image are clearly discernable. This is because of the magnification produced.

Standard image compression methods can be evaluated using their compression ratios : the ratio of the memory required to store an image as a collection of pixels and the memory require to store a representation of the image in compressed form.

The compression ratio of fractal is easy to misunderstand since the image can be decoded at any scale. In practice it is important to either give the initial and decompressed image sizes or use the same sizes for a proper evaluation. Because the decoded image is not exactly the same as the original such schemes are said to be lossy.

|  |  |
| --- | --- |
|  |  |

4.2 Mathematical model

A mathematical model for the copying machine described earlier is called an Iterative Function System (IFS). An IFS system consists of a collection of contractive transformations wi , this collection defines a map, W(s) =, where I = 1, 2, ….,n. Where s is the input and W(s) is the output of the copier. Whatever the initial image S, the image after infinite interactions will tend to the attractor Xw. This Xw is unique for that particular W(s).

Two important facts are listed as below:

When the wi are contractive in the plane then W is contractive in a ste of the plane. This was proved by Hutchinson.

If we are given a contractive map W on a space of images then there is a special image called the attractor denoted by Xw with the following properties.

1. The attractor is called the fixed point of W
2. Xw is unique. If we find any set S and an image transformation w satisfying W(s) =S, then S is the attractor of W.

Fractal image representation can be described mathematically as an iterated function system (IFS).

**For Binary Images**

We begin with the representation of a binary image, where the image may be thought of as a subset of \mathbb{R}^2. An IFS is a set of contraction mappings *ƒ*1,...,*ƒN*,

f_i:\mathbb{R}^2\to \mathbb{R}^2.

According to these mapping functions, the IFS describes a two-dimensional set *S* as the fixed point of the Hutchinson operator

H(A)=\bigcup_{i=1}^N f_i(A), \quad A \subset \mathbb{R}^2.

That is, *H* is an operator mapping sets to sets, and *S* is the unique set satisfying *H*(*S*) = *S*. The idea is to construct the IFS such that this set *S* is the input binary image. The set *S* can be recovered from the IFS by fixed point iteration: for any nonempty compact initial set *A*0, the iteration *Ak*+1 = *H*(*Ak*) converges to *S*.

The set *S* is self-similar because *H*(*S*) = *S* implies that *S* is a union of mapped copies of itself:

S=f_1(S)\cup f_2(S) \cup\cdots\cup f_N(S)

So we see the IFS is a fractal representation of *S*.

**Extension to Grayscale**

f_i:\mathbb{R}^3\to \mathbb{R}^3.IFS representation can be extended to a gray-scale image by considering the image's graph as a subset of \mathbb{R}^3. For a gray-scale image *u*(*x*,*y*), consider the set *S* = {(*x*,*y*,*u*(*x*,*y*))}. Then similar to the binary case, *S* is described by IFS using a set of contraction mappings *ƒ*1,...,*ƒN*, but in \mathbb{R}^3,

**Encoding**

A challenging problem of ongoing research in fractal image representation how to choose the *ƒ*1,...,*ƒN* such that its fixed point approximates the input image, and how to do this efficiently. A simple approachfor doing so is the following:

1. Partition the image domain into blocks *Ri* of size *s*×*s*.
2. For each *Ri*, search the image to find a block *Di* of size 2*s*×2*s* that is very similar to *Ri*.
3. Select the mapping functions such that *H*(*Di*) = *Ri* for each *i*.

In the second step, it is important to find a similar block so that the IFS accurately represents the input image, so a sufficient number of candidate blocks for *Di* need to be considered. On the other hand, a large search considering many blocks is computationally costly. This bottleneck of searching for similar blocks is why fractal encoding is much slower than for example DCT and wavelet based image representations.

**Features**

With fractal compression, encoding is extremely computationally expensive because of the search used to find the self-similarities. Decoding however is quite fast. While this asymmetry has so far made it impractical for real time applications, when video is archived for distribution from disk storage or file downloads fractal compression becomes more competitive.

At common compression ratios, up to about 50:1, Fractal compression provides similar results to DCT-based algorithms such as JPEG. At high compression ratios fractal compression may offer superior quality. For satellite imagery, ratios of over 170:1have been achieved with acceptable results. Fractal video compression ratios of 25:1-244:1 have been achieved in reasonable compression times (2.4 to 66 sec/frame).

Compression efficiency increases with higher image complexity and color depth, compared to simple gray-scale images.

4.3 Resolution independence and fractal scaling

An inherent feature of fractal compression is that images become resolution independent after being converted to fractal code. This is because the iterated function systems in the compressed file scale indefinitely. This indefinite scaling property of a fractal is known as "fractal scaling".

4.4 Fractal interpolation

The resolution independence of a fractal-encoded image can be used to increase the display resolution of an image. This process is also known as "fractal interpolation". In fractal interpolation, an image is encoded into fractal codes via fractal compression, and subsequently decompressed at a higher resolution. The result is an up-sampled image in which iterated function systems have been used as the inter-polant. Fractal interpolation maintains geometric detail very well compared to traditional interpolation methods like bilinear interpolation and bi-cubic interpolation.

**\*\*\*\*\***

**CHAPTER 5: SELF SIMILARITY IN IMAGES**

Natural images are not self-similar. A typical image of a face does not contain the type of self-similarity found in the fractals. The image does not appear to contain affine transformations of itself. But this image contains a different type of self-similarity. Here the image is formed of properly transformed parts of itself. These transformed parts do not fit together in general to form an exact copy of the original image and so we must allow some amount of error in our representation of an image as a set of self-transformations. This means that an image that we encode as a set of transformations will not be an identical copy but an approximation.

To get a mathematical model of an image we express it as a function z = f(x,y) where z is the grayscale.

Now, we want to find a map *W* which takes an input image and yields an output image. If we want to know when *W* is contractive, we will have to define a *distance* between two images. The distance is defined as

where *f* and *g* are value of the level of grey of pixel (for gray-scale image), *P* is the space of the image, and *x* and *y* are the coordinates of any pixel. This distance defines position (*x*,*y*) where images *f* and *g* differ the most.

Natural images are not exactly self-similar. Lena image, a typical image of a face, does not contain the type of self-similarity that can be found in the Sierpinski triangle. But next image shows that we can find self-similar portions of the image.

|  |  |
| --- | --- |
| lenna | lenna2 |
| Original image | Self-similar portions of the image |

A part of her hat is similar to a portion of the reflection of the hat in the mirror. The main distinction between the kind of self-similarity found in the Sierpinski triangle and Lena image is that the triangle is formed of copies of its **whole** self under appropriate affine transformation while the Lena image will be formed of copies of properly transformed **parts** of itself. These parts are not exact the same; this means that the image we encode as a set of transformations will not be an identical copy of the original image. Experimental results suggest that most images such as images of trees, faces, houses, clouds etc. have similar portions within itself.

5.1 Metric on Images

If we want to know whether W transformation is contractive we will have to define a distance between two images. A metric is a function that measures the distance between two images. There are metrics that measure the distance between two images , the distance between two points , distance between two sets etc.

Two of the most commonly used metrics are:

Dsup(f,g) = Sup(f(x,y) – g(x,y) )

And the rms metric

Drms(f.g) =

The supremium metric finds the position where two images f and g differ the most and sets this value as the distance between the two points. The rms metric is more convenient in applications.

5.2 Partitioned IFS

The individual transformations described earlier are performed on parts of the image and not on the whole. This forms a partitioned IFS also called a PIFS. There are two spatial dimensions and the grey level adds a third dimension. A portion of the original image called domain (Di) is mapped to a part of the produced image called range (Ri) by the transformation Wi. Since W(f) is an image the Ri covers the whole square page and are adjacent but not overlapping.

In the PIFS case, a fixed point or attractor is an image f that satisfies W(f) =f. Thus if we apply the transformations to the image we get back the original image. Thus if we are required to code a natural image this is first partitioned to several domain blocks and iterative function systems are formed of the different parts of the image to yield the output image.

5.3 Partitioning of Images

The basic idea behind partitioning of images is to first partition the image by some collection of ranges Ri, then for each Ri  seek from some collection of image pieces a Di that has a low rms error when mapped to Ri. If we know Ri and Di then we can determine the remaining co-efficient. The various partitioning schemes used are dealt in the next chapter.

**\*\*\*\*\***

**CHAPTER 6: ALGORITHM & CODE**

6.1 Algorithm for Fractal Image Compression

1. *Input a binary image, call it M.*
2. *Cover M with square range blocks. The total set of range blocks must cover M, without overlapping.*
3. *Introduce the domain blocks D; they must intersect with M. The sides of the domain blocks are twice the sides of the range blocks.*
4. *Define a collection of local contractive affine transformations mapping domain block D to the range block Ri.*
5. *For each range block, choose a corresponding domain block and symmetry so that the domain block looks most like the part of the image in the range block.*
6. *Write out the compressed data in the form of a local IFS code.*
7. *Apply a lossless data compression algorithm to obtain a compressed IFS code.*

In practice these steps can be carried out on a digital image. The compression is attained by storing the coefficients of the transformations, rather than storing the image pixel by pixel. The following is an explanation of 'fenc', a simple fractal image compression program and 'fdec' the corresponding decompression program, both included after the explanation.

These two programs can be run on MATLAB and only compress grayscale square images that are in ‘tif’ format, although further changes can be implemented later to account for non-square images and other formats. The transformation coefficients are saved as mat files are saved representing the compressed images. 'fenc' searches for the transformation with least error from domain blocks to range blocks. So, the program searches for a transformation until it finds a transformation with dist < dmin, where the expression of dmin is given below.

First, the user must enter the name of the tif image file in the first line of the program: M = imread ('lena\_gray\_256.tif'). Then, the user specifies the desired range block size by setting rsize equal to the length of the side of the desired range block. Presently, rsize is set equal to 4, which allows range blocks of size . We next create the domain blocks, which are twice the size of the range blocks, in this case . In determining which mapping will need to be made from the domain blocks to the range blocks, we will need to compare the domain blocks to the range blocks. To accurately compare these blocks, they must be the same size. So, we do some averaging over the domain blocks which allows us to shrink the domain blocks to half of its size in order to match the size of the range blocks.

Originally, each domain block is 8x8. The averaging only takes place over each distinct block of 2x2 pixels within the domain block. Then the average gray-scale value in each 2x2 block of pixels is represented in one pixel in the scaled domain blocks, called M1. M1 is a 4x4 block at this point. We subtract the average of the domain block from each entry in the domain block to account for possible darkening of the decompressed image. The resulting scaled domain block is D.

Now, we save 8 different transformations of each domain block in an eight dimensional monster matrix called bigM. The transformations include the original domain block, 900, 1800 and 2700 rotation, a horizontal flip, and a vertical flip, as well as the transform of the domain block and a 1800 rotation of the transformed domain block. We introduce a vector s, which contains different specific scalings to transform the grayscale of the domain block to make a better match to a range block.

At this point, 'fenc' goes through all of the range blocks and offsets each of them by subtracting the average of the range block from each entry in the range block. Now we can equally compare the domain to the range blocks. We save the offset of the range blocks in o, which we will add back to the image later.

Next the program cycles through each domain block and tests each symmetry that is stored in bigM, along with the four possible gray scales for the best transformation that will map to a given range block . When the best map is found, the location of that domain block i0 and j0, the best symmetry m0 of the domain block, the best scaling s0, and the difference of mean of Range and corresponding Domain g0 is saved in the five dimensional matrix T(k,l,:)=[i0 j0 m0 s0 g0]. It is the entries of this matrix that determine the number of bytes needed to store the compressed image file. The program saves the number of rows of the original image, the size of the range blocks, and the time the program took to achieve the compression. The time is recorded to compare the results with the amount of time the process took. Once this information is saved in a file, it is possible to compress that file even more by applying a lossless coding algorithm. It is from the matrix, T, that the program 'fdec' can regenerate the image.

It is important to note that each transformation from the original 8x8 domain is a contraction mapping because the domain must be scaled by ½ in order to map the domain to the range. Also, the information stored in each (k,l,:) entry of T represent the coefficients of the mappings , i = 1,2,3,…N that make up the N local IFS mappings. The image regenerated after all the mappings in T are applied to some seed image, is the attractor of the local IFS.

In order to regenerate the attractor of the contractive transformations found, we must use the program 'fdec' along with the saved information from 'fenc'. First we load the correct data using the name that we saved it under in the ‘.mat’ file. Then we initialize a matrix to perform the mappings on. This matrix must be the same size as the original image. As we discussed with IFS, it makes no difference what seed image is used. Although, in the program we initialize the seed image to all ones, which is a uniformly gray image, choosing another image as the seed to the local IFS will arrive at the same result.

Depending on the block size chosen for the range blocks, one may need to vary the number of iterations applied to the seed image in order to arrive at the attractor image. As more iteration of the IFS is applied to the image, the clearer the attractor will become. After the nth iteration, the image produced corresponds to the compact set as discussed in the local IFS theory. First, the domain blocks of the seed image must be created and rescaled to the size of the range blocks. Then using the T matrix, the domain blocks are transformed and mapped to the range blocks. This process is repeated for each iteration. The attractor, M, is then output to be displayed on the screen. The quality of the attractors vary depending on the size of the range blocks used and the error allowed in finding an appropriate transformation form domain block to range block.

Our implementation of this simple method of fractal compression produced great compression ratios. Considering that each pixel requires 8 bits to store the values of 0 to 255, to store an 256x256 image pixel by pixel would require 65536 bytes (around 65KB). Using 'fenc' and 'fdec', to store an image of this size with a range block size of  pixels only requires 11776 bytes. The compression ratio is better than 5:1. Of course, increasing the range block size to  pixels improves the compression to only 2688 bytes, with a compression ratio of approximately 24:1. The larger range block sizes allow higher compression ratios. The use of the image will determine the required amount of compression and image quality.

6.2 The MATLAB® code for fractal compression [fenc.m]:

clc;

clf;

% Set timers

begrun=clock

cpu=cputime

M=imread('lena\_gray\_256.tif');

[sv sh]=size(M);

if sv~=sh

display('Matrix is not square');

return

end

% Begin batch runs

min0=100;

rsize=4;

nd=sv/rsize/2;

nr=sv/rsize;

% Scale the Domain Blocks

for i=1:rsize\*nd

for j=1:rsize\*nd

M1(i,j)=mean(mean(M((i-1)\*2+1:i\*2,(j-1)\*2+1:j\*2)));

end

end

% Matrix of 4 possible scalings to transform grayscale

s=[0.45 0.60 0.80 1];

% Create monster matrix containing all possible 2D transformations

% of the domain blocks. Store in multidimensional matrix bigM.

for i=1:nd

i1=(i-1)\*rsize+1;

i2=i\*rsize;

for j=1:nd

j1=(j-1)\*rsize+1;

j2=j\*rsize;

D=M1(i1:i2,j1:j2);

bigM(i1:i2,j1:j2,1)=D;

tmp=rotmat(D);

bigM(i1:i2,j1:j2,2)=tmp;

tmp=rotmat(tmp);

bigM(i1:i2,j1:j2,3)=tmp;

tmp=rotmat(tmp);

bigM(i1:i2,j1:j2,4)=tmp;

bigM(i1:i2,j1:j2,5)=fliph(D);

bigM(i1:i2,j1:j2,6)=flipv(D);

bigM(i1:i2,j1:j2,7)=D';

bigM(i1:i2,j1:j2,8)=rotmat(rotmat(D'));

end

end

% Compare the range blocks and scaled domain blocks.

% k,l - used to cycle through blocks Rkl.

clear T;

for k=1:nr

k1=(k-1)\*rsize+1;

k2=k\*rsize;

for l=1:nr

l1=(l-1)\*rsize+1;

l2=l\*rsize;

R=M(k1:k2,l1:l2);

% Offset o is the average in the block Rkl

o=mean(mean(R));

R=double(R);

% Initialize error to large value

dmin=10^9;

i0=0;

j0=0;

m0=0;

% Now cycle through each Domain Dij

for i=1:nd

i1=(i-1)\*rsize+1;

i2=i\*rsize;

for j=1:nd

j1=(j-1)\*rsize+1;

j2=j\*rsize;

% Test each transformation

for n=1:4

for m=1:8

D=s(n)\*bigM(i1:i2,j1:j2,m);

del\_g=o-mean(mean(D));

D=D+del\_g;

sum\_dist=sum(sum((R-D).^2));

dist=sqrt(sum\_dist);

if dist<dmin

dmin=dist;

i0=i;

j0=j;

m0=m;

s0=s(n);

g0=del\_g;

end

end

end

end

end

T(k,l,:)=[i0 j0 m0 s0 g0];

end

end

% Stop the clock, store computation time in tim

% and elapsed cpu time in cpu0.

cpu0=cputime-cpu

stoprun=clock

tim=etime(begrun,stoprun)

% Save data in mat file - need to change the name after each use.

save 'gs\_norm' sv rsize T tim cpu0;

6.3 The MATLAB® code for fractal image decoding [fdec.m]

load 'gs\_norm'

% Initialize matrix

M=100\*ones(sv);

% Start Iteration

for iter=1:10

% Enter range block size used in fcomp

rsize=4;

nd=sv/rsize/2;

nr=sv/rsize;

% Rescale Domain Blocks

for i=1:rsize\*nd

for j=1:rsize\*nd

M1(i,j)=mean(mean(M((i-1)\*2+1:i\*2,(j-1)\*2+1:j\*2)));

end

end

% Transform Domain Block Using T matrix

for k=1:nr

k1=(k-1)\*rsize+1;

k2=k\*rsize;

for l=1:nr

l1=(l-1)\*rsize+1;

l2=l\*rsize;

i0 = T(k,l,1);

j0 = T(k,l,2);

m0 = T(k,l,3);

s0 = T(k,l,4);

g0 = T(k,l,5);

i1 = (i0-1)\*rsize+1;

i2 = i0\*rsize;

j1 = (j0-1)\*rsize+1;

j2 = j0\*rsize;

D = M1(i1:i2,j1:j2);

if m0==2

D=rotmat(D);

elseif m0==3

D=rotmat(rotmat(D));

elseif m0==4

D=rotmat(rotmat(rotmat(D)));

elseif m0==5

D=fliph(D);

elseif m0==6

D=flipv(D);

elseif m0==7

D=D';

elseif m0==8

D=rotmat(rotmat(D'));

end

R=s0\*D+g0\*ones(size(D));

MM(k1:k2,l1:l2)=R;

end

end

M=MM;

end

% Output Image which is in M

imagesc(M)

colormap(gray);

A Fast Variance-Ordered Domain Block Search Algorithm for Fractal Encoding

**CHAPTER 7: INTRODUCTION**

Since Jacquin introduced a practical fractal image coding scheme based on the block-wise iterated function system (IFS), many efforts to reduce the computational encoding complexity have been attempted. The major part of the encoding process which need a long encoding time is known to be the searching step to find the best matched domain block to each range block in order to construct the fractal code.

The range block approximation from a domain block by affine transformation in Jacquin's algorithm can be written as

where , and are the isometry transformation, gray level scale factor and luminance shift factor, respectively, and *Dk* is a spatially contracted domain block. The luminance shift is computed so that the average gray levels of the range block and the scaled domain block arc the same. That is,

The best estimate, *Rk* , can be obtained by minimizing the squared Euclidean distance (SED) defined as

where the block size of the range block is assumed to be BxBpixels.

Finding the best estimate usually requires the exhaustive sequential search of all domain blocks, which would take too long time for practical applications in spite of a high reconstructed image quality.

This paper presents a new fast search algorithm using the squared variance distance (SVD) measure which safely rejects a great number of domain blocks during the search process while maintaining the quality of thc decoded image exactly the same to that of the full search.

7.1 Proposed Algorithm

In searching for the closest domain block to a given range block in the SED sense, we used its correlation with the squared variance distance (SVD), defined as

Animportant observation in real image is that the greater the variance differences between two image-blocks are, the greater the SED becomes. This motivated the use of variance as a feature of the image-block when comparing image-blocks in the SED sense. That is to say, the similarity in the complexity of image blocks, SVD, is a nearly sufficient statistic for the small Euclidean distance, SED, of interblocks. The explicit relationships between SVD and SED for any image blocks can be given by the inequality

as shown in Appendix. Therefore, if the search for the minimum SED domain block starts with the minimum SVD domain block and proceeds to the next nearer ones in the SVD since, there is a high probability of arriving at the minimum SED domain block in only a few searching steps.

Let be a known current minimum Euclidean distance of represented by a certain domain block. For any domain block *,* so for , if the inequality

is satisfied, then will not be the closest domain block to and it is unnecessary to calculate *.* This fact facilitated the early termination of our search process without worrying over the possible existence of closer domain blocks in the remaining domain pool. For our purposes, all of the domain blocks in the domain pool are sorted in descending (or ascending) order of their variances.

*N* contrast scale factors ( are applied to find the closest domain block in fractal encoding as shown in Eq. (1).

For the contrast scaled domain block, the variance can be easily calculated by the following equation:

Therefore, our partial domain block search algorithm can be adopted to find the closest contrast scaled domain block in the same manner without a great deal of additional computational complexity. The composition of the domain pool and the search procedure for our algorithm are illustrated in Fig. 1.

All of the domain blocks are sorted according to their variances in descending order, assuming that the original image is partitioned into overlapping M domain blocks and 4 contrast scalings are applied. Including the contrast scaled domain blocks whose variances can be obtained from Eq. (9), 4M domain blocks are produced totally and sorted in the domain pool.

For a range block whose variance is , the search process starts from the domain block having the closest variance to *,* i.e. the domain block *Du,* in Fig. 1. With the minimum SVD domain block *Du,* regarded as the reference block, the upper neighboring domain block is searched, then the lower neighboring domain block is searched afterwards. This search process continues in a back-and-forth manner until the termination condition, Eq. (8), is satisfied in both directions, at and domain block in this case. The rest of the lower index domain blocks than the domain block and the domain block are rejected as a whole because their SVD are so large that there is no possibility of being closer to the range block than the domain block having current minimum SED. The rest of the higher-index domain blocks than the domain block and the domain block are rejected as a whole for the same reason.

7.2 The MATLAB® code of variance-ordered partial search (VPS) algorithm:

clc;

clf;

% Set timers

begrun=clock

cpu=cputime

M=imread('lena\_gray\_256.tif');

[sv sh]=size(M);

if sv~=sh

display('Matrix is not square');

return

end

rsize=4;

nd=sv/rsize/2;

nr=sv/rsize;

% Scale the Domain Blocks

for i=1:rsize\*nd

for j=1:rsize\*nd

M1(i,j)=mean(mean(M((i-1)\*2+1:i\*2,(j-1)\*2+1:j\*2)));

end

end

% Matrix of 4 possible scalings to transform grayscale

s=[0.45 0.60 0.80 1];

% Find variances of domain block for different scalings

Vd=zeros(nd,nd,4);

for i=1:nd

i1=(i-1)\*rsize+1;

i2=i\*rsize;

for j=1:nd

j1=(j-1)\*rsize+1;

j2=j\*rsize;

D=s(n)\*M1(i1:i2,j1:j2);

D=double(D);

Z=mean(mean(D));

Z=D-Z;

for n=1:4

Vd(i,j,n)=(s(n)^2)\*sum(sum(Z.^2));

end

end

end

% Sorting variances of domain block

Vd\_ord=zeros(4,nd\*nd\*4);

Vd\_ord=Vd\_ord';

w=Vd(:);

Vd\_ord(:,1)=w;

e=1:nd;

e=e';

e=e\*ones(1,nd);

e=e(:);

e=e\*ones(1,4);

e=e(:);

f=1:nd;

f=f'\*ones(1,nd);

f=f';

f=f(:);

f=f\*ones(1,4);

f=f(:);

g=[1 2 3 4];

g=g';

g=g\*ones(1,nd\*nd);

g=g';

g=g(:);

Vd\_ord(:,2)=e;

Vd\_ord(:,3)=f;

Vd\_ord(:,4)=g;

for k=1:(nd\*nd)

for i=1:(nd\*nd-1)

temp=zeros(1,4);

if Vd\_ord(i,1)<Vd\_ord(i+1,1)

for j=1:4

temp(j)=Vd\_ord(i,j);

Vd\_ord(i,j)=Vd\_ord(i+1,j);

Vd\_ord(i+1,j)=temp(j);

end

end

end

end

% Begin batch runs

clear T;

% Compare the range blocks and scaled domain blocks.

% k,l - used to cycle through blocks Rkl.

for k=1:nr

k1=(k-1)\*rsize+1;

k2=k\*rsize;

for l=1:nr

l1=(l-1)\*rsize+1;

l2=l\*rsize;

R=M(k1:k2,l1:l2);

R=double(R);

% Offset o is the average in the block Rkl

O=mean(mean(R));

Rm=R-O;

Vr=sum(sum(Rm.^2));

min=10000;

u=1;

for i4=1:(nd\*nd)

er=(sqrt(Vr)-sqrt(Vd\_ord(i4,1)))^2;

if er<min

min=sqrt(er);

u=i4;

end

end

% Initialize error to large value

i0=0;

j0=0;

m0=0;

g0=0;

s0=0;

dmin=10^9;

stf=0;

q=zeros(1,2);

q(1)=u-1;

q(2)=u;

while stf==0

dir=1;

while (dir>=0)&(stf==0)

if ((sqrt(Vd\_ord(q(dir+1),1))-sqrt(Vr))^2)>=dmin^2

stf=2;

else

i=Vd\_ord(q(dir+1),2);

j=Vd\_ord(q(dir+1),3);

n=Vd\_ord(q(dir+1),4);

% Now cycle through each Domain Dij

i1=(i-1)\*rsize+1;

i2=i\*rsize;

j1=(j-1)\*rsize+1;

j2=j\*rsize;

bigM=zeros(rsize,rsize,8);

D=M1(i1:i2,j1:j2);

del\_g=O-s(n)\*mean(mean(D));

D=D+del\_g;

bigM(:,:,1)=D;

tmp=rotmat(D);

bigM(:,:,2)=tmp;

tmp=rotmat(tmp);

bigM(:,:,3)=tmp;

tmp=rotmat(tmp);

bigM(:,:,4)=tmp;

bigM(:,:,5)=fliph(D);

bigM(:,:,6)=flipv(D);

bigM(:,:,7)=D';

bigM(:,:,8)=rotmat(rotmat(D'));

% Test each transformation

for m=1:8

D=bigM(:,:,m);

sum\_dist=sum(sum((R-D).^2));

dist=sqrt(sum\_dist);

if dist<dmin

dmin=dist;

i0=i;

j0=j;

m0=m;

s0=s(n);

g0=del\_g;

end

end

end

q(dir+1)=q(dir+1)+((-1)^dir);

if ((q(dir+1)<1)|(q(dir+1)>(nd\*nd\*4)))==1

stf=2;

end

end

end

T(k,l,:)=[i0 j0 m0 s0 g0];

end

end

% Stop the clock, store computation time in tim

% and elapsed cpu time in cpu0.

cpu0=cputime-cpu

stoprun=clock

tim=etime(begrun,stoprun)

% Save data in mat file - need to change the name after each use.

save 'gs\_ffic' sv rsize T tim cpu0;

The transformation co-efficeints were saved onto a file named “gs\_ffic” MATLAB data file (‘.mat’).

The fractal compression decoder file is same as that of the normal fractal compression, without any changes.

**\*\*\*\*\***

**CHAPTER 7: FRACTAL IMAGE COMPRESSION VERSUS JPEG COMPRESSION**

At high compression ratios, fractal compression has similar compression and quality performance as the JPEG standard. Let see some comparisons:

  
Original image (184,320 bytes)

|  |  |
| --- | --- |
| Lena_jpg | Lena_fif |
| JPEG-max. quality (32,072) comp. ratio: **5.75:1** | FIF-max. quality (30,368) comp. ratio: **6.07:1** |

|  |  |
| --- | --- |
| Lenajpg2 | Lenafif2 |
| JPEG-med. quality (11,278) comp. ratio: **16.34:1** | FIF-med. quality (7,339) comp. ratio: **25.11:1** |

|  |  |
| --- | --- |
| Lenajpg3 | Lenafif3 |
| JPEG-min. quality (8,247) comp. ratio: **22.35:1** | FIF-min. quality (3,924) comp. ratio: **46.97:1** |

These are expected results for Fractal Compression. Some images can be compressed over 80:1 or higher. We can see worse image quality for higher compression ratios.

Fractal Compression is an asymmetric process, compression taking a long time compared with decompression. JPEG/DCT is symmetric process, encoding and decoding taking the same amount of time. Fractal compression times can be decreased by using a dedicated hardware. For example, for a 640 x 400 pixel 24 bit colour image, JPEG software compression and decompression took 6 seconds each, for fractal compression the time was 68 seconds and decompression was only one second. Reading the original uncompressed image required two seconds.  
Another one very important feature of the Fractal Image Compression is *Resolution Independence*.

Fractal encoding image has no natural size. It is set of transformations defined on domains and ranges of the image. When we want to decode image, only thing we have to do is apply these transformations on any initial image (it is usually grey rectangle for grey-scale images) iteratively. Every iteration is step closer to the decoded image. After each iteration, details on the decoded image are sharper and sharper. That means, the decoded image can be decoded at any size. The extra detail needed for decoding at larger sizes is generated automatically by the encoding transforms. The next question is: if we decode an image at larger and larger sizes, will we be able to see atoms of the Lena's face? The answer is, of course, no because we are encoding digitized image which is only approximation of the nature image but on the larger sizes we do not have "pixelization" effect than we get sharper details.

|  |  |
| --- | --- |
| Lenaeye1 | Lenaeye2 |
| Lena's eye original image enlarged to 4 times | Lena's eye  decoded at 4 times its encoding size |

This feature of Fractal Image Compression is unique. The greatest irony of the coding community is that great pains are taken to precisely measure and quantify the error present in a compressed image, and great effort is expended toward minimizing an error measure that most often is - let us be gentle - of dubious value. These measures include signal-to-noise ratio, root mean square error, and (less often) mean absolute error. A simple counter example is systematic error: add a value of 10 to every pixel. Standard error measures indicate a large distortion, but the image has merely been brightened.

With respect to those dubious error measures, the results of tests reveal the following: for low compression ratios JPEG is better, for high compression ratios fractal encoding is better. The crossover point varies but is often around 40:1 to 50:1. This figure bodes well for JPEG since beyond the crossover point images so severely distorted that they are seldom worth using.

Proponents of fractal compression counter that signal-to-noise is not a good error measure and that the distortions present are much more 'natural looking' than the “block”y effect of JPEG, at both low and high bit rates. This is a valid point but is by no means universally accepted.

**\*\*\*\*\***

**CHAPTER 9: APPLICATIONS OF FRACTALS**

Creating a fractal requires just a few lines of software. The resulting picture could be quite rich in details and would require large memory if stored as such. This forms the basis of fractal compression of pictures. Given any arbitrary picture one has to find out which of the portions of images could be thought of as self-similar versions of other portions. Self-affine transformations can then be found out. The image can then be displayed quickly and at any magnification with infinite levels of fractal detail. Genetic algorithms in MATLAB are generally used for the efficient encoding of fractals.

The most amazing thing about fractals is the variety of their applications. Besides theory, they were used to compress data in the Encarta Encyclopaedia and to create realistic landscapes in several movies like Star Trek.

[](http://en.wikipedia.org/wiki/File:Julian_fractal.jpg)[](http://en.wikipedia.org/wiki/File:Square1.jpg)The places where you can find fractals include almost every part of the universe, from bacteria cultures to galaxies to your body. In this section, we have the most important applications, trying to include them from as many areas of science and everyday life as possible. Most applications use a specific topic related to fractals that is discussed either in the tutorial or types of fractals. Examples are Galaxies, Rings of Saturn, Bacteria Cultures, Chemical Reactions, Human Anatomy, Molecules, Plants , Population Growth, Clouds, Coastlines and Borderlines, Data Compression, Diffusion, Economy, Fractal Art, Fractal Music, Landscapes, Newton's Method, Special Effects (Star Trek), Weather.

A fractal created using the program High voltage breakdown within a 4″ block

Apophasis and a *julian* transform of acrylic creates a fractal Lichtenberg figure

[](http://en.wikipedia.org/wiki/File:Glue1_800x600.jpg)[](http://en.wikipedia.org/wiki/File:Microwaved-DVD.jpg)

Fractal branching occurs in a fractured A fractal is formed when pulling apart

surface such as a microwave-irradiated [DVD](http://en.wikipedia.org/wiki/DVD). two glue-covered [acrylic](http://en.wikipedia.org/wiki/Acryl) sheets

**So why isn't everyone using Fractal Image Compression?**

Fractal Image Compression is still under development. Many different researchers and companies are trying to develop new algorithms to reach shorter encoding time and smaller files. But there are still some problems with fractal compression. Fractal Compression Standard does not exist ! FIF, Fractal Image Format is not standardized, is not embedded into any browsers .In spite of all, total number of publications on fractal image compression is growing; over 280 papers was published last year.

**\*\*\*\*\***

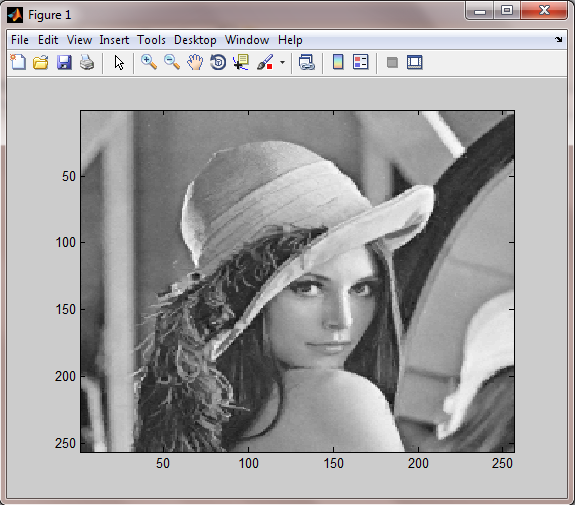
**CHAPTER 10: RESULTS**

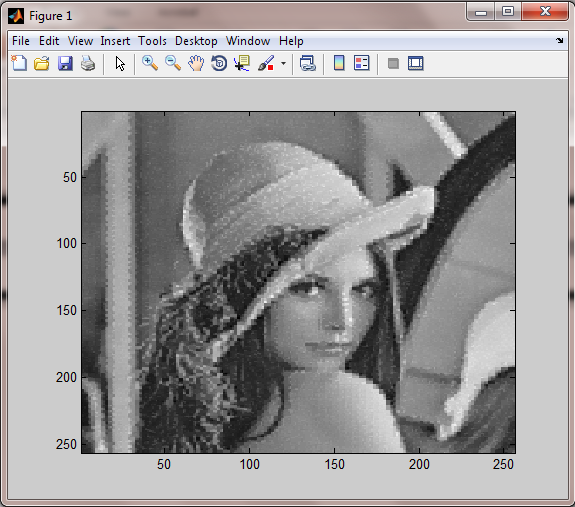
We encoded Lena (256x256) grey-scale image using the MATLAB® program code described above. This is performed using the 4x4 block size using the normal fractal compression method and the fast fractal compression method. Here is a summary of the results:

|  |  |  |
| --- | --- | --- |
|  | Normal Fractal Compression | Fast Fractal Compression |
| Block Size | 4x4 | 4x4 |
| Time to encode | secs | 253 secs |
| Time to decode | 32 secs | 28 secs |
| Size of fractal code | 30.3 KB | 23.7 KB |

\* PC used: Intel Core-i7, Clock speed 3.1 Ghz, 8GB RAM, NVIDIA GEFORCE GT 2 GB Graphics, 1 TB Hard Disk

Original image ‘Lena’ size 256x256

****

****Lena image after fractal decoding with block size 4x4

Lena image after decoding from fast fractal compression with block size 4x4

**CHAPTER 10: CONCLUSIONS**

In this paper, we proposed a new fast fractal encoding algorithm using the variances of image blocks. The experimental results showed that the numbers of searched domain blocks are reduced to less than a quarter of that of the full search while maintaining the quality of the decoded image exactly the same.

If the image block classifier is employed in a fractal encoder, the VPS algorithm can also be independently applied to each class such as midrange blocks and edge blocks to reduce the encoding time. Furthermore, since the domain pool is already sorted by the variance of each domain block in the VPS algorithm, a variance block classifier can be effectively adopted to accelerate the fractal image encoding.

**REFERENCES:**

1. *Arnaud E. Jacquin, Image Coding Based on a Fractal theory of Iterated Contractive Image Transformations," IEEE Trans. on Image Processing, vol. 1, no. 1, pp. 18-30, Jan. 1992.*
2. *Yuval Fisher, Fractal Image Compression - Theory and Application, Springer-Verlag, ISBN 0-387-94211-4, 1994.*
3. *Yuval Fisher," Fractal Image Compression" ,Course Notes, vol. 12, ACM SIGGRAPH, 1992.*
4. *Dietmar Saupe, \Breaking the Time Complexity of Fractal Image Compression," Technical Report, vol. 53, Institute for Informatics, University at Freiburg, 1994.*
5. *Michael F. Barnsley and Lyman P. Hurd, Fractal Image Compression, AK Peters Ltd.,ISBN 1-56881-000-8, 1993.*
6. *Benoit B. Mandelbrot, The Fractal Geometry of Nature, W.H. Freeman and Company, ISBN 0-7167-1186-9, 1983.*
7. *Maaruf Ali and Trevor G. Clarkson, Fractal Image Compression," Proc. 1st Seminar on Information Technology and it Applications (ITA '91), Sept. 29, 1991, Leicester.*
8. *www.wikipedia.com*