Image compression quality measurement: A comparison of the performance of JPEG and fractal compression on satellite images.

Ernst Nolte

Thesis presented in partial fulfillment of the requirements for the degree of Master of Engineering at the University of Stellenbosch.

Dr Hendrik Boshoff
Prof Johan Lourens
March 2000

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature

Date:

Abstract

The purpose of this thesis is to investigate the nature of digital image compression and the calculation of the quality of the compressed images. The work is focused on greyscale images in the domain of satellite images and aerial photographs. Two compression techniques are studied in detail namely the JPEG and fractal compression methods. Implementations of both these techniques are then applied to a set of test images. The rest of this thesis is dedicated to investigating the measurement of the loss of quality that was introduced by the compression. A general method for quality measurement (signal To Noise Ratio) is discussed as well as a technique that was presented in literature quite recently (Grey Block Distance). Hereafter, a new measure is presented. After this, a means of comparing the performance of these measures is presented. It was found that the new measure for image quality estimation performed marginally better than the SNR algorithm. Lastly, some possible improvements on this technique are mentioned and the validity of the method used for comparing the quality measures is discussed.

Opsomming

Die doel van hierdie tesis is om ondersoek in te stel na die aard van digitale beeldsamepersing en die berekening van beeldkwaliteit na samepersing. Daar word gekonsentreer op grysvlak beelde in die spesifieke domein van satellietbeelde en lugfotos. Twee spesifieke samepersingstegnieke word in diepte ondersoek naamlik die JPEG en fraktale samepersingsmetodes. Implementasies van beide hierdie tegnieke word op 'n stel toetsbeelde aangewend. Die res van hierdie tesis word dan gewy aan die ondersoek van die meting van die kwaliteitsverlies van hierdie saamgeperste beelde. Daar word gekyk na 'n metode wat in algemene gebruik in die praktyk is asook na 'n nuwer metode wat onlangs in die literatuur veskyn het. Hierna word 'n nuwe tegniek bekendgestel. Verder word daar 'n vergelyking van hierdie mates en 'n ondersoek na die interpretasie van die 'kwaliteit' van hierdie kwaliteitsmate gedoen. Daar is gevind dat die nuwe maatstaf vir kwaliteit net so goed en selfs beter werk as die algemene maat vir beeldkwaliteit naamlik die Sein tot Ruis Verhouding. Laastens word daar moontlike verbeterings op die maatstaf genoem en daar volg 'n bespreking oor die geldigheid van die metode wat gevolg is om die kwaliteit van die kwaliteitsmate te bepaal.

Acknowledgements

Firstly, I would like to thank the Good Lord who gave me the strength and capacity to complete this task.

I would also like to thank my two study leaders for their continued support during this project. Dr Boshoff and Prof Lourens, without your advice I would have gotten permanently lost on all the sidetracks that a work such as this, presents in its evolution to completion.

I would like to thank my wife Odell for the support during all those times that I did not feel like working. You have given me the resolve to finish a great many things that I would never even have started without your confidence in me.

I would like to specially thank three of my colleagues whose technical (and not so technical) advice on the progamming elements of this work saved me from countless hours of useless frustration. Paul van Spronsen, Hans Loedolff and Charlotte Ackerman – thanks.

Table of Contents

1. Introduction

Moore's law [Schal] states that computer memory, speed and disk space doubles approximately every eighteen months. Although high, the increase in demands made on computer capabilities is even more rapid.

Computer graphics is one of the fields where it seems that user requirements grow faster than technology. With the growing popularity of the internet, many people have access to large amounts of digital image data. These images may vary from pretty pictures used as background screens for personal computers, to archives of satellite photographs used for weather prediction.

Digital images generally take up large amounts of disk space. Consider for example the previously mentioned example of an archive of satellite images that is used for long term weather prediction. Such an archive would typically contain sets of images covering the whole planet or parts thereof. These images would be taken at daily or even more frequent intervals. The images would require a high resolution and there could possibly be more than one image per region (for instance infra red photographs to give temperature measurements and water vapour images to study cloud formations). Given a long enough observation period, the amount of space necessary to represent such an archive could reach considerable proportions. It is therefore understandable that image compression can play a vital role in many computer application fields.

The main purpose of this work is to investigate the measurement of digital image quality after degradation due to compression. In order to achieve this goal, more must be learned of compression techniques themselves and this work will consider the JPEG and fractal compression methods in some detail. Armed with this knowledge of image compression, the rest of this work will be focused on the measurement of the quality of these techniques applied to satellite images and the evaluation of the value or quality of these measures.

There are currently an abundance of different image compression technologies available. The two main categories that separate these methodologies are lossless and lossy compression. Lossless compression will not be discussed in this work for two reasons. Firstly, the gains achieved from this technique are minimal and have an upper bound ([Shan] and [BarnC, chapter 5]). Secondly, the theme of this work is to

find some way to objectively measure the quality of a compressed image and lossless compression suffers no quality degradation, thus making the exercise trivial.

In chapter 2, two specific lossy compression techniques will be discussed. These are respectively the JPEG and fractal image compression techniques. The theory of each method will be discussed and an example of the implementation of each will be given. Due to the relative youth of fractal image compression in comparison with the methods underlying JPEG, a brief description will be given on the history of the development of fractal image compression and some of the early advances made in this field will be discussed shortly. Some test results of both methods will also be given in these chapters.

By the end of chapter 2, armed with a basic understanding of these compression methods, the focus will be shifted to the measurement of the performance of these compression techniques. The general performance of an image compression technique is measured by three parameters: the compression ratio, the compression/decompression time and the compression quality or degradation. The first of these three parameters is easily quantifiable since it is simply the ratio of the file size of the original image to that of the compressed image (this value is generally expressed as bits per pixel). The calculation of the quality of a degraded image in comparison with its original image is not as simple to obtain as this is a more subjective measure. There are several methods available for calculating this value but they do not always correlate very well with each other. This gives rise to many variations that are used by different people, making it difficult to perform a comparative study from different sources.

Chapter 3 describes the problem involved in this particularly difficult task. Some existing quality measures are examined and implemented, and their results are listed in Table 11.

In section 3.1.5, a new measure of image quality is proposed and implemented. A variation of this implementation is also investigated and implemented. The results of both of these methods are listed in Table 11.

It is however easy to invent arbitrary measures to calculate the difference between images without having any tools for evaluating the 'quality' of these quality measures in comparison with existing measures, The real test lies in how well these results correlate with human perception of image quality. For this reason, section 0 of chapter 3 describes a method for collecting some subjective judgement information

on image compression quality. This data is then interpreted to produce a baseline for comparison of different image grading techniques.

Chapter 4 examines the results of the different quality measurements that were implemented in the previous chapter and tries to find the reasons for the good or bad comparison that was achieved using the technique also described in that chapter. This chapter also examines some possible improvements to the two quality measurement techniques that were presented in this work.

In chapter 5, the fractal and JPEG compression techniques are compared and a critical look is taken at the subjective image qualification test that was performed for this work as well as the validity of the cross correlation calculation used to grade the different image quality measures.

2. Image Compression

There are currently many different methods for compressing image data, each with its own good qualities and drawbacks. Currently popular compression techniques include the JPEG, fractal and wavelet compression techniques. A comparison of these three methods can be found in [Fisher]. Fractal and wavelet compression methods are based on transforms by that same name and JPEG is based on the discrete cosine transform. A very good introductory article for JPEG compression can be found in [Wall].

The choice of which method to use is largely dependent on the application requirements. This can become a nontrivial exercise in linear programming to optimise the compression results with the requirements.

Image compression can be subdivided into two main categories namely lossless and lossy compression. In both these categories, there are many techniques available. These include transform coding, predictive coding, vector quantisation, pixel coding and hybrid coding. Chapter 11 of [Jain] and chapter 5 of [Kak] supply a good introduction to image compression in general.

The statistical nature of digital image data imposes strict upper bounds on the performance of lossless compression techniques depending on the entropy of the source (see p. 42 of [Jain]). Current implementations can achieve a maximum compression ratio of about 3:1.

Due to the nature of real world images (image energy is concentrated at the lower spatial frequencies) and the effect of the human visual system on the perception of these images, much information can be removed from an image without noticeable loss of quality. This happens because the human visual system acts as a filter to what is perceived so that some aspects of an image are attenuated and need not be represented with full accuracy. The human brain also performs image processing on what it sees and can fill in detail that might not be present in an image merely because it expects the information to be there. The understanding and successful exploitation of these functions of the human visual system and brain leads to the category of lossy image compression.

Lossy image compression is performed when a specific level of degradation is allowed, and this in turn will influence the compression ratio. The general level of

degradation and compression ratio is dictated by the specific implementation with an attempt at optimising the *cost and effectivity equation*. The cost and effectivity equation for image compression consists of a trade-off between compression quality, compression ratio and the compression/decompression time. The compression ratio and compressed image quality can only be increased at cost to each other after a certain point. This can be deduced logically as follows: Consider an image containing a fixed amount of information in a fixed size. The size can be larger than that necessary to contain all the information as is generally the case in most data collections. This information can now be stored more optimally up to a point where there is no more data redundancy and the data size is the same as the amount of information. This is the maximum level of lossless compression that may be obtained for a certain amount of information and can be calculated from Shannon's coding theories [Shan]. Any further compression will lead to the loss of information that cannot be recovered during decompression, thus leading to a decrease in image quality. The compression and decompression time is largely a function of the type of compression used. Fractal image compression for instance has a very slow encoding time but is very fast to decode. JPEG on the other hand is very fast to both compress and decompress. This feature makes it viable for the type of application where images are compressed and decompressed continually for transmission. This can also be seen from the fact that the JPEG image format is one of the Internet standards used today.

Most implementations of image compression algorithms available today consist of a combination of lossy and lossless techniques. The lossy of part the compression algorithm is generally based on some form of transform coding, as has already been mentioned. The principle of transform coding is to find a transformation on a data set with very good energy packing properties. After transformation of an image, most of the image energy is concentrated in relatively few of the transform coefficients and compression is achieved by discarding coefficients that have a negligible influence on the final result as interpreted by the human visual system. To apply this in practice an image is divided into smaller rectangular blocks that are transform coded separately. More information on transform coding can be found in chapter 5.3-4 of [Jain]. Fractal coding is somewhat different in nature to general transform coding. The fractal transform does not truly pack image energy into a more compact form but exploits image redundancy to achieve compression.

The above mentioned techniques can achieve compression ratios of 10:1 to 50:1 depending on the image, with acceptable levels of image degradation. All these

techniques consist of a main lossy transform compression that is optimised by a combination with some lossless technique like entropy or runlength coding.

It is difficult to make a quantitative comparison between all these compression techniques since each has its own weaknesses and strengths and some are optimised for specific types of images, or to perform optimally on certain features of an image. The specific application therefore plays a very important role in judging the performance of a compression algorithm.

The problem is also made more difficult by the fact that the performance of a compression technique is measured by the three factors already mentioned. It is therefore important to decide which factors will be used to judge the techniques and to keep the other factors constant and similar to provide reliable test data.

This thesis is concerned with the estimation of the quality of the image and as far as possible, the compression ratio of the test images will be kept similar. The compression time for these two methods differ by a few orders of magnitude and will be ignored in this work since it does not affect the compression ratio or quality if the test algorithms are not constrained by the compression time in any way.

For the purpose of this thesis, a set of thirty images were selected as test data. The images are all either satellite images or aerial photographs of the earth's surface. The image content varies between weather phenomena, continental views and cities. Some of the images were taken in the visual range whilst some are infra red images or water vapour photographs. The images also differ in resolution, scale and quality to provide a general combination of images from this category. The images are contained in the CD-ROM accompanying this thesis.

The first section in this chapter examines the fractal image compression method. Section 2.1.1 serves as an introduction to the mathematical theory required for the understanding and proof of this technique. Section 2.1.2 describes the development of fractal image compression from the groundbreaking work done by Barnsley and Jacquin ([Barn], [JacqA], [JacqB], [JacqC], [JacqD]) in the early 1990's and also takes a closer look at some of the later contributions that improved on this initial work. This is by no means a definitive history of the work done on fractal image compression but only serves as a brief introduction to this topic. An extensive list of published articles and some very good introductory works can be found at ftp://ftp.informatik.uni-freiburg.de/papers/fractal/. The bibliography and some of these articles are also included on the CD-ROM. This section is followed by a detailed description of an actual compression algorithm as implemented by Fisher in

[FisherB]. Section 2.1.4 concludes the discussion of fractal image compression by looking at some of the results obtained from application of the algorithm described in section 2.1.3.

The last section of this chapter deals with the JPEG compression method. Section 2.1.5 describes the discrete cosine transform upon which JPEG compression is based and also discusses all other necessary techniques required to understand the application of JPEG compression. This is followed by a description of the actual JPEG compression algorithm and the last section of this chapter looks at some of the results obtained from JPEG compression.

Two other methods of image compression that are currently undergoing active research but is not discussed in this work are the zero-tree wavelet and set partitioning in hierarchical trees. These methods may be incorporated into the JPEG 2000 standard and are well worth some further investigation.

Fractal Image Compression

2.1.1 Mathematical background

This section examines some of the preliminary concepts necessary for the understanding of fractal image compression. Firstly, a brief description of fractals is given and then some necessary definitions and theorems are considered. This is followed by the discussion of two methods for generating fractals. The latter of these two methods is then used in an example to illustrate the basic idea of fractal image compression.

The field of fractals is much too diverse to discuss in detail in this work. Interested readers are referred to two classical books by Mandelbrot ([MandA] and [MandB]) for a good background on fractals. Two other sources are the book *Fractals for the Classroom* [Peitg] which is aimed at the reader without a strong mathematical background and the more advanced text by Michael Barnsley, *Fractals Everywhere* [BarnB].

The following short introduction of the term fractal and the definition thereof was adapted from the introduction of Benoit B. Mandelbrot's book *Fractals: Form, Chance, and Dimension* [MandA].

The term fractal was coined by Benoit B. Mandelbrot. It is derived from the Latin adjective *fractus* which means irregular or fragmented and is related to the Latin word *frangere* which means to break. The concept of fractal geometry was introduced to resolve some of the shortcomings of classical Euclidian geometry to describe the behaviour of certain natural as well as artificial phenomena. Classical geometry deals with objects like lines and circles and properties such as their derivatives. Problems arose for example when trying to measure the length of a coastline or its derivative at some point. Intuitively, measurements should be more accurate with a smaller resolution or scale of measurement. In the example mentioned however, the measured length of the coastline increases with every increase in scale and the derivative changes with every change in scale and does not seem to converge to a fixed value. An artificial example of this phenomenon is the now famous Mandelbrot set which features on several pages of Barnsley [BarnB].

In order to obtain a definition for fractals, it is necessary to first introduce the concept of dimension of a set. There are two definitions of dimension to be considered. The first is the topological dimension denoted by D_T which is used in Euclidian geometry. The other is the Hausdorf-Besicovitch or fractal dimension which is denoted by D. See Peitgen [Peitg] for a detailed desription of these two definitions and how to calculate them for specified sets. The two dimensions equal at least 0 and at most the dimension E of the Euclidian space in which the set is embedded. Furthermore: $D \geq D_T$. The cases where $D = D_T$ include all Euclidian sets. This leads to the definition of fractals proposed by Mandelbrot in [MandA]:

A fractal will be defined as a set for which the Hausdorf-Besicovitch dimension strictly exceeds the topological dimension.

An important feature of fractals which is worth mentioning here and on which the key to fractal image compression rests, is the fact that many fractals are invariant to certain transformations of scale and many fractals are self-similar. This can be seen in natural fractal shapes like that of a cauliflower or the vein system in the human liver, and also in artificial fractals like the Sierpinski triangle (see Figure 2-3) and the Koch snowflake.

To get to the purpose of this chapter, which is the application of fractal theory to obtain compression of images, some definitions and theorems are given. The following definitions and theorems are directly related to the concept of fractal image compression and although it might seem like a slight digression from the

topic, the purpose and relevance of this will become clear as the chapter progresses.

Definition 1 ([BarnB, p50]): A transformation $\omega: \Re^2 \rightarrow \Re^2$ of the form

$$\omega(\boldsymbol{x}) = \boldsymbol{A}(\boldsymbol{x}) + \boldsymbol{B} \tag{2.1}$$

or

$$\omega\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{2.2}$$

where *a, b, c, d, e* and *f* are real numbers is called a two-dimensional *affine* transformation.

This transformation can rotate, scale, shear, reflect or translate a region of the plane to another region. The ***B*** matrix is responsible for the translation of points. To better understand the effect of the ***A*** matrix in this transformation, it can be rewritten in the form

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} r_1 \cos\theta_1 & -r_2 \sin\theta_2 \\ r_1 \sin\theta_1 & r_2 \cos\theta_2 \end{bmatrix} \tag{2.3}$$

The effects of the new parameters on the transformation is illustrated in Figure 2-1.

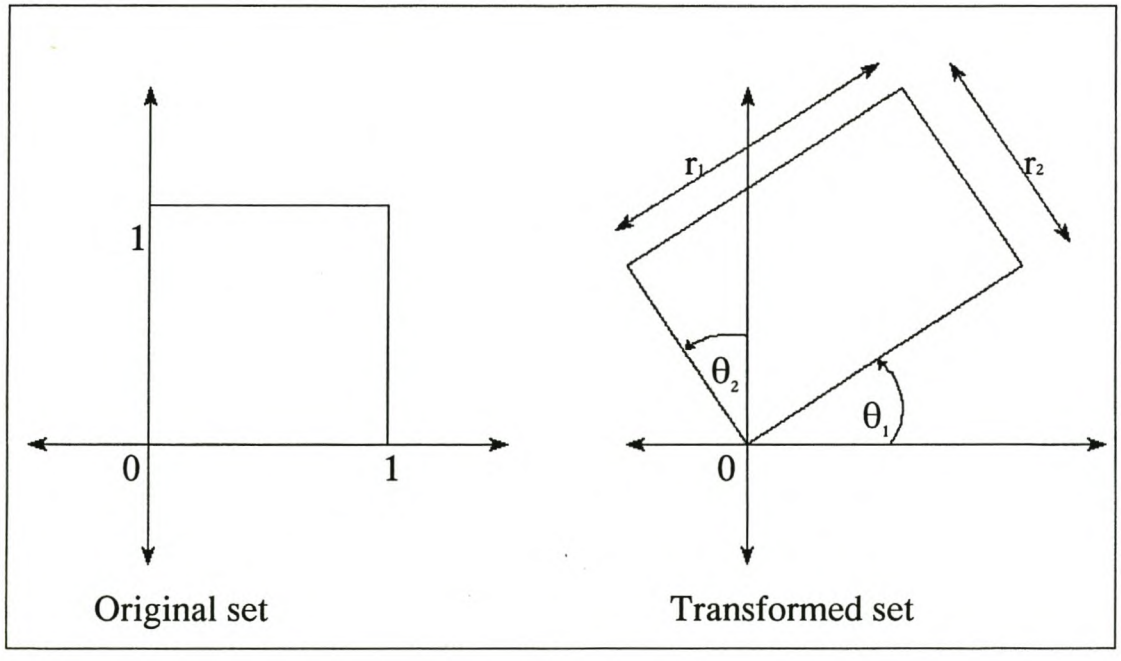


Figure 2-1 Effects of an affine transformation on the unit square

Definition 2 ([BarnB, p73]): Let f:**X**→**X** be a transformation on a metric space. A point

$x_f \in \mathbf{X}$ such that $f(x_f) = x_f$ is called a *fixed point* of the transformation.

Definition 3 ([BarnB, p75]): A transformation f: **X** → **X** on a metric space (**X**, d) is called *contractive* or a *contraction mapping* if there is a constant $0 \leq s < 1$ such that $d(f(x), f(y)) \leq s \cdot d(x, y)\ \forall x, y \in \mathbf{X}$. Any such number s is called a *contractivity factor* for f.

Theorem 1 [The Contraction Mapping Theorem] ([BarnB, p76]): Let f: **X** → **X** be a contraction mapping on a complete metric space (**X**, d). Then f posesses exactly one fixed point $x_f \in \mathbf{X}$ and moreover for any point $x \in \mathbf{X}$, the sequence $\{f^{on}(x) : n = 0,1,2...\}$ converges to x_f. That is, $\lim_{n\to\infty} f^{on}(x) = x_f$, for each $x \in \mathbf{X}$.

The proof of this theorem can be found on p.76 of Barnsley [BarnB].

This theorem implies that no matter what the initial set to which a contraction mapping is repetitively applied, it will eventually converge to a specified fixed point - in other words, if the fixed point of some contraction mapping is a circle image, the result of iteratively applying this contraction mapping to the image of a tree, will still be an image of a circle.

Definitions 4, 5, 6 and 7 and theorem 2 are specifically related to the iterated function system method for generating fractals.

Definition 4 ([BarnB, p82]): An *iterated function system* (IFS) consists of a complete metric space (X, d) together with a finite set of contraction mappings ω_n: **X** → **X**, with respective contractivity factors s_n, for n = 1, 2,..., N. The notation for the IFS is

{X; ω_n, n = 1, 2,..., N} and its contractivity factor is s = Max{s_n: n = 1, 2,..., N}

Definition 5 ([BarnB, p30]): Let (**X**, d) be a complete metric space. Then H(**X**) denotes the space whose points are compact subsets of **X**, other than the empty set.

Definition 6 ([BarnB, p34]): Let (**X**, d) be a complete metric space. Then the *Hausdorf distance* between points A and B in H(**X**) is defined by

$$h(A,B) = d(A,B) \vee d(B,A) \tag{2.4}$$

with

$$d(A,B) = \text{Max}\{d(x,B) : x \in A\}, \tag{2.5}$$

d(A,B) is the distance from the set $A \in$ H(**X**) to the set $B \in$ H(**X**). The symbol $\vee$ is used to denote the maximum of the two real numbers given by *d(A, B)* and *d(B, A)*.

Theorem 2 ([BarnB, p82]): Let {**X**; ω_n, n = 1, 2,..., N} be an IFS with contractivity factor *s*. Then the transformation *W*: H(**X**) $\rightarrow$ H(**X**) defined by

$$W(B) = \bigcup_{n=1}^{N} \omega_n(B) \tag{2.6}$$

for all B $\in$ H(X), is a contraction mapping on the complete metric space (H(**X**), h(d)) with contractivity factor *s*. That is

$$h(W(B), W(C)) \leq s \cdot h(B,C) \tag{2.7}$$

for all $B, C \in$ H(**X**). Its unique fixed point, A $\in$ H(**X**), obeys

$$A = W(A) = \bigcup_{n=1}^{N} w_n(A), \tag{2.8}$$

and is given by

$$A = \lim_{n \to \infty} W^{\circ n}(B) \tag{2.9}$$

for any B $\in$ H(**X**).

Theorem 2 is basically the same as theorem 1 but the contraction mapping considered in this case is an IFS.

Definition 7 ([BarnB, p82]): The fixed point A $\in$ H(**X**) described in theorem 2 is called the *attractor* of the IFS.

Theorem 3 [The Collage Theorem - Barnsley 1985] ([BarnB, p97]): Let (**X**, d) be a complete metric space.Let $L \in$ H(**X**) be given, and let $\varepsilon \geq 0$ be given. Choose an IFS {**X**; ω_1, ω_2,..., ω_N} with contractivity factor $0 \leq s \leq 1$, so that

$$h\left(L, \bigcup_{n=1}^{N} \omega_n(L)\right) \leq \varepsilon, \tag{2.10}$$

where h(d) is the Hausdorf metric. Then

$$h(L,A) \leq \varepsilon / (1-s) \tag{2.11}$$

where *A* is the attractor of the IFS.

This theorem can be interpreted as follows: An IFS can be found whose attractor is arbitrarily close to *any given set* (image) by finding a collection of transformations on the given set such that the union of the transformations on the given set is arbitrarily close to the given set itself. Or more simply, an IFS can be found that is an arbitrarily close approximation of a specified image, by finding different transformations that approximate different regions of the specified image. An approximation of the specified image is then found by uniting these transformations.

A simple but illustrative example of a fractal that can be generated with an iterated function system is the Sierpinski gasket. This fractal is created with the three simple transformations shown in Figure 2-2.

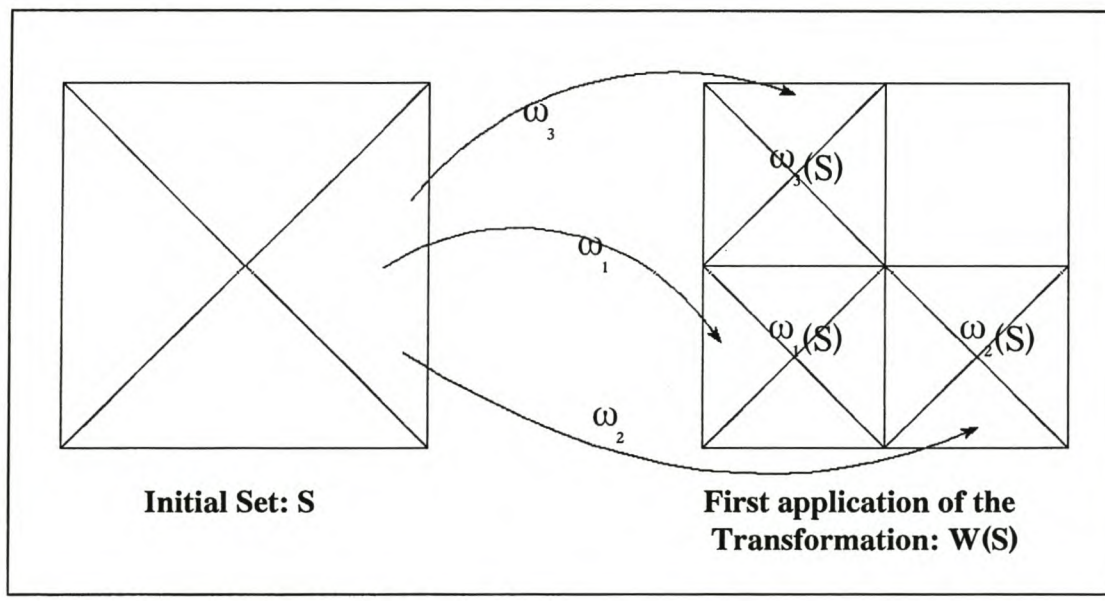


Figure 2-2 Affine mappings for the Sierpinski gasket

The three transformations for the Sierpinski gasket can be written as follows:

$$\omega_1 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.12}$$

$$\omega_2 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} \tag{2.13}$$

$$\omega_3 = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0 \end{bmatrix} \tag{2.14}$$

The attractor for **W** = $\omega_1 \cup \omega_2 \cup \omega_3$ is shown in figure 3.

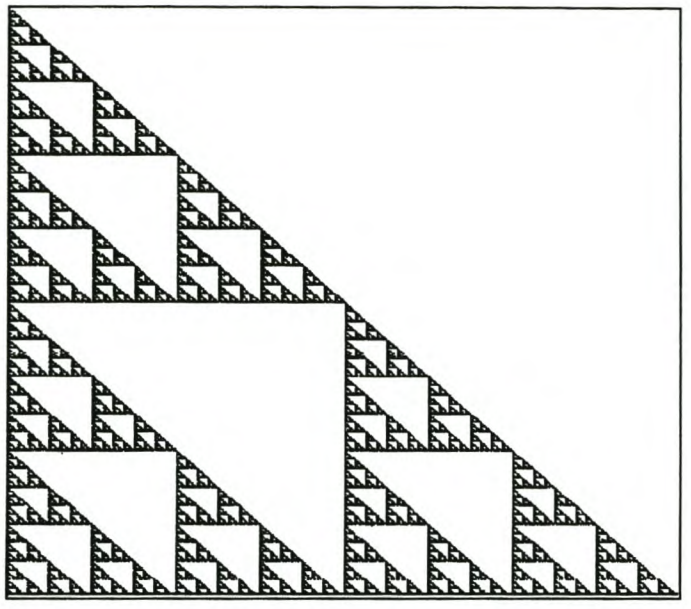


Figure 2-3 Sierpinski gasket

An extension of the concept of iterated function systems, is *partitioned iterated function systems* (PIFS). With PIFS, each transformation is only performed on a segment of the set and is then translated to some region of the plane. This is illustrated in Figure 2-4.

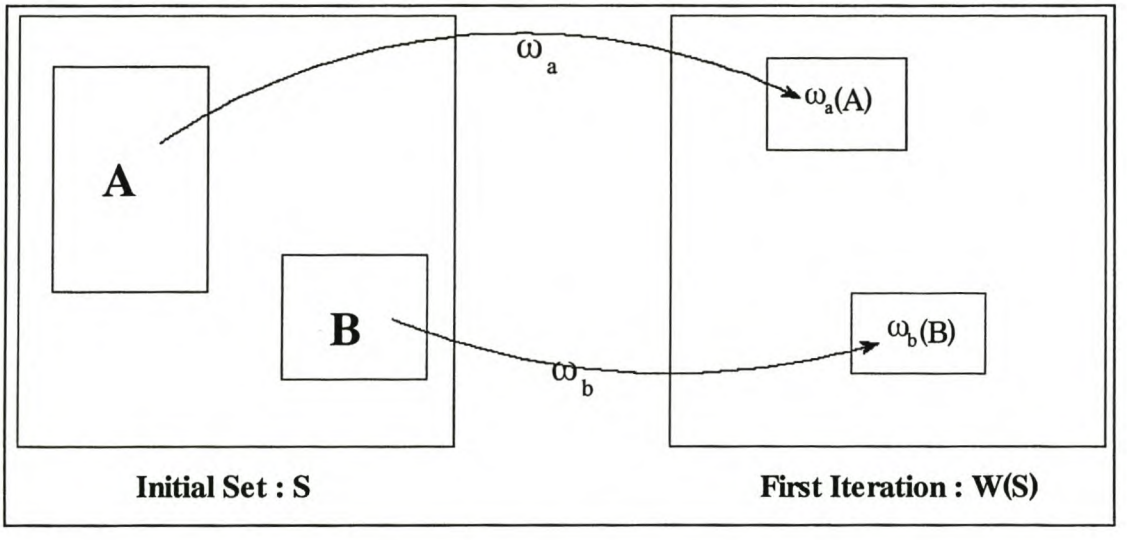


Figure 2-4 Illustration of a Partitioned Iterated Function System

The application of PIFS in image compression is perhaps best explained with the help of a simple example. Consider the image shown in Figure 2-5. Assume that this is a 1bit/pixel, 128x128 pixel image. To digitally represent this image would require

16kBytes of memory. The idea of fractal image compression is to represent this image by a PIFS of which the attractor is close to the image.

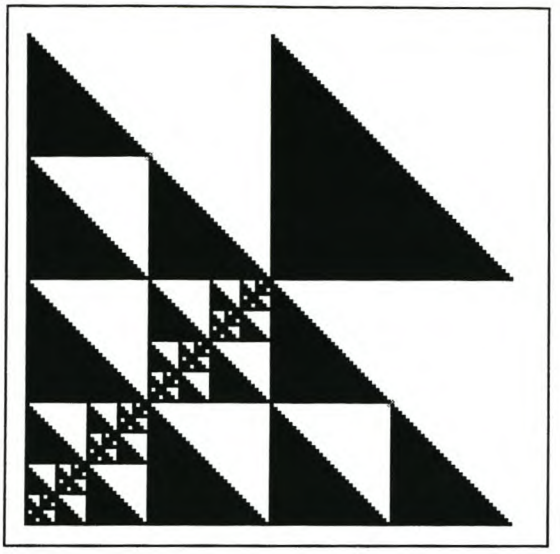


Figure 2-5 Image to be represented by a PIFS

Firstly the image is segmented into nonoverlapping blocks of size 8x8 and 16x16 pixels (see Figure 2-6). The smaller blocks will be called *range* blocks and the larger ones *domain* blocks. Next an affine mapping must be found for each range block so that some domain block can be mapped to an exact replica or as close as possible match to it. The notation adopted to represent the PIFS is

$$A{:}D_i \rightarrow R_j \tag{2.15}$$

where R_i denotes the range block in question, D_i denotes the matching/approximating domain block and A is the transformation matrix.

No translational terms are necessary since the range and domain block numbers already specify any necessary translations completely (see Figure 2-6). The fixed domain and range block sizes also implies a fixed contractivity factor of 0.5 for all the transformations and therefore the transformation matrix A for this case is given by:

$$A = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \tag{2.16}$$

The complete set of PIFS also includes two non affine transformations. These will be called the *zero* and *full* transformation. The zero transformation simply replaces a range block by an empty block and is denoted by $D_z \rightarrow R_i$. The full

transformation replaces a range block with a fully coloured/all black block and is denoted by $D_f \rightarrow R_i$.

These transformations might seem trivial but are necessary for the existence of the desired attractor. In the absence of the full transformation, a black and white image that is contractively transformed will eventually become all white since the black areas keep shrinking and no 'substance' is added.

All the transformations necessary to represent the image in Figure 2-5 are shown below.

$A{:}D_2 \rightarrow R_{1,3,5,6,8,9,11,14,15,16}$

$A{:}D_3 \rightarrow R_{10,13}$

$D_z \rightarrow R_{2,4,12}$

$D_f \rightarrow R_7$

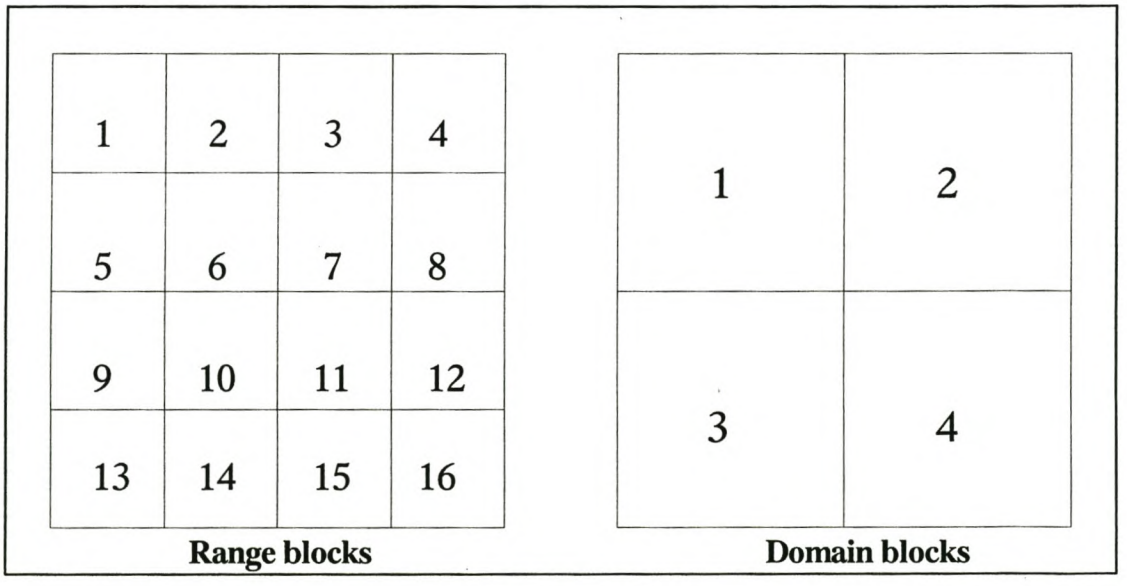


Figure 2-6 Range and domain block division

To reconstruct the original image from this mapping, it can simply be applied iteratively to *any* initial image. This is illustrated with a solid black starting image and the successive iterations are shown in Figure 2-7. As can be seen, this specific PIFS converges to an exact copy of the original with no degradation. To store the transformations required to create this image, much less memory than the pixel representation of the image is required.

Since all **A** matrices in this example are the same, further compression can be achieved by only saving the domains for each range. This means that each range block can be stored using three bits since there are only four domains and two special transformations. The image is then completely represented by 48 bits - a compression ratio of 341:1. There are only six possibilities for each range so three bits are actually a waste of storage space. This can be reduced further by some lossless coding technique like Huffman coding. In practice however, compression ratios such as this one will not be encountered due to the nature and complexity of real world images. Although very simple, this is the basis for most fractal image compression algorithms in use today.

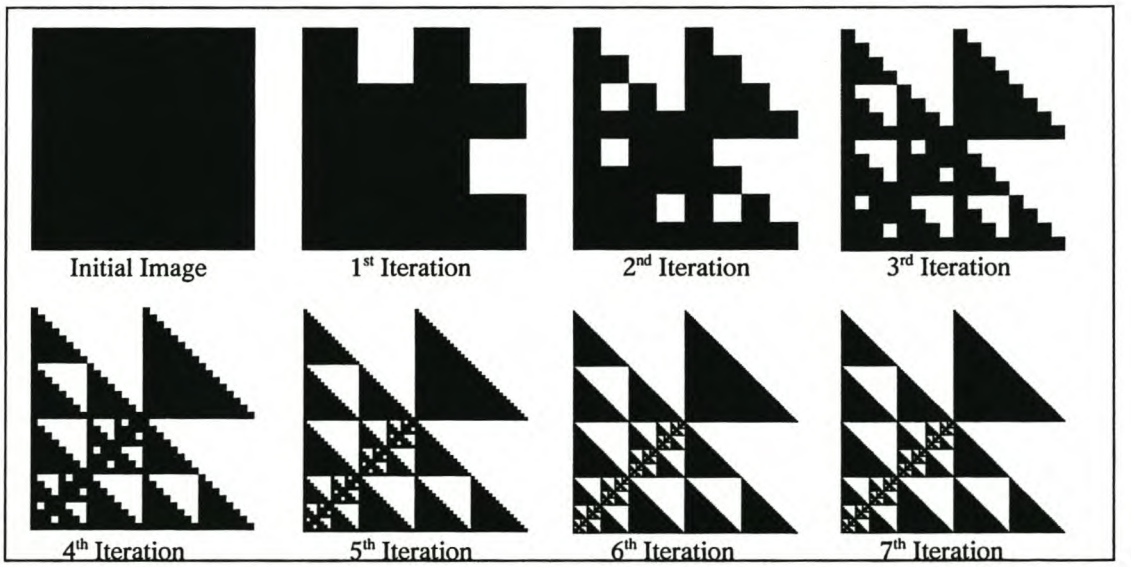


Figure 2-7 First 7 iterations of the PIFS

The examples only considered the case of black and white images. The progression to grayscale images is somewhat more complicated. A grayscale image can be viewed as a three dimensional landscape where height represents grayscale intensity values. An affine transformation for grayscale images will look as follows :

$$\omega_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix} \tag{2.17}$$

where s_i represents the scaling or contrast and o_i represents the offset or brightness of the grayscale colour values. Grayscale fractals can be generated in the same way as before, but overlapping transformed blocks have their pixel values added together to find the resulting intensity value in overlapping regions.

The extension to full colour images can be done in several ways. Colour images can be broken up into their red green and blue components (RGB) and each of these can be treated as a separate grayscale image on its own. This is not a very effective way to accomplish the extension to colour images. A better way is to convert the colour image into its luminance and chrominance representations (YIQ- or YUV- models). The conversion formulas for these representations can be found in chapter 14 of Watt [Watt] and chapter 3 of Jain [Jain]. The luminance component is equivalent to a grayscale representation of the image and carries most information. The luminance component is then treated as a standard grayscale image whilst the chrominance components can be represented at a lower resolution without loss of quality. This method will have a higher compression ratio than the RGB-separation method.

This section should leave the reader with an understanding of the basic theory of image compression through fractal methods. The next section takes a look at some of the research that has been done in this field to improve the basic compression method.

2.1.2 Development

The concept of fractal image compression is based on the mathematical results of iterated function systems. This theory can be traced back to work done by Williams and Hutchinson ([Will], [Hutch]). It was Barnsley and Sloan however, who first connected this theory with the compression of digital images in their 1988 publication [Barn]. The compression method tries to exploit image redundancy by modelling it as self-similarity, ie. the redundancy present in fractal objects. This can be accomplished in two steps. Firstly, an image is broken into segments and in the second step these segments are replaced by similar fractals from an existing library of fractals. The results of applying this method showed very slow encoding and decoding times although encoding was more than an order of

magnitude slower than decoding. The compression ratios achieved however were extremely high. The method unfortunately did not lend itself very well to complete automation. In 1990 and 1991 Barnsley and Sloan were granted two patents[1, 2] which led to commercial compression software and hardware.

The first fully automated fractal image compression routine was developed by Arnaud Jacquin, a student of Barnsley, and was presented in his PhD thesis and followed up by several now classic articles on this topic ([JacqB], [JacqC], [JacqD], [JacqE]). The compression scheme discussed hereafter is based on greyscale images. The progression to colour images is a logical extension where each of the colour components can be treated as a greyscale image in itself (end of Section 2.1.1).

Many publications after the groundbreaking work of Jacquin are based on his idea and attempt to improve some aspects of it. The rest of this section will be dedicated to discussing Jacquin's method and some of the improvements that have been made on it.

Most publications referred to in this section compare their results with some standard form of fractal compression. This standard form is generally based on Jacquin's article without any improvements but varies from article to article. This is a very slow and inefficient method on which remarkable improvements have been made. When compared with published results that seem very much improved it must just be kept in mind that they are compared to a very slow and sub optimal algorithm. Different measures for compression quality are also used in different publications and these measures are difficult to compare to each other in absolute terms.

Since the execution of this type of algorithm is also image dependent, it is difficult to obtain quantitative results for the effect on performance of some parameter changes in the algorithm. At best, compression ratio, encoding and decoding time, as well as signal to noise ratio can be calculated for a specific set of images, and these results can be used for comparison. The question of distortion is also a difficult one to resolve. Because of the properties of the human visual system (Chapter 3 of [Jain]), some images with a specific SNR are perceived to be of a

---

[1] Barnsley, M. F., Sloan, A. D., Methods and apparatus for image compression by iterated function system, United States Patent # 4 941 193.

[2] Barnsley, M. F., Sloan, A. D., Method and apparatus for processing digital data, United States Patent # 5 065 447.

better quality whilst others with a higher SNR are perceived to be of a lower quality. There are also a few definitions for the calculation of SNR which all yield different results for different types of images and different types of distortion. To adhere to some standard, the distortion measure used in this section is the same as that used by Jacquin in [JacqD]. This is the peak-to-peak signal-to-noise-ratio given by:

$$SNR = 10\log\left(\frac{dr(\mu)^2}{d(\mu,\tilde{\mu})/r^2}\right) \tag{2.18}$$

where dr(μ) denotes the dynamic range of the image μ,

r^2 is the number of pixels in μ,

$d(\mu,\tilde{\mu})$ is the RMS metric given by:

$$d(\mu,\tilde{\mu}) = \sum_{i=1}^{MN}\sqrt{(\mu_i - \tilde{\mu}_i)^2} \tag{2.19}$$

with μ the original image and $\tilde{\mu}$ the distorted image.

Section 0 will consider some alternative measures for compression quality calculation and also investigate how they compare to each other. The rest of this chapter will use the definition for signal to noise ratio given above to quantify image compression quality.

Once the effects of some changes to the basic algorithm have been calculated for a general set of images, it must be decided whether the trade-offs in the features of this scheme are viable for its specific application. In an image archiving application for example, it would make sense to increase compression time for higher image quality but not at the cost of too much compression ratio.

The aim of Jacquin's fractal encoding is to construct a contractive image transformation *W* defined from the metric space (**X**, d) of digital images to itself such that a specified original image μ_{orig} is the fixed point of the transformation. d is a given metric, the distortion measure already discussed.

Jacquin defines a set of allowed transformations for a PIFS. He separates each transformation into a geometric and massic part. The geometric transformation is simply a spatial contraction. The massic part of the transformation affects specific pixel values. It consists of the eight possible rotations through 90° and the associated reflections about the x and y axes. These rotations and reflections are

called isometries. Apart from the isometries, the massic part of the transformation can also consist of the following four operations :

i) Absorption at a specified grey level.

ii) Luminance shift.

iii) Contrast scaling.

iv) Colour reversal.

He partitions the image into non-overlapping range blocks and searches from a pool of transformed domain blocks for a best match for each range block. The domain pool consists of a collection of subimages of the original image upon which the possible transformations have been carried out. To keep the domain pool size to a manageable level, he restricts the choice of domain blocks to all blocks of a certain size starting at specific intervals of pixels. The choice of the domain block size depends on the contractivity factor of the allowed transformations and is generally twice the size of the range blocks. He chooses the domain pool source blocks to be all the blocks with their bottom left corner at the origin or a multiple of B or B/2 pixels in both directions (with B the height and width of the range blocks).

Finally, the image can be represented by the transformation W consisting of the union of the respective transformations for each range block. The fact that the attractor will be similar to the original image is guaranteed by the Collage Theorem (Theorem 3 section 2.1.1). Therefore the distance between the fixed point A and the given image μ_{orig} can be minimised by minimising the distance between μ_{orig} and the collage of μ_{orig} which is $W(\mu_{orig})$. This transformation constitutes a lossy compressed representation of the original image. To decompress this image, the transformations need to be carried out iteratively on any initial image. After about eight iterations, the image generally converges to within 0.1dB of its fixed point. Convergence depends on the contraction factor $s \leq 1$. For more detailed information on the specifics of Jacquin's algorithm, see [JacqD] and [JacqE].

One of the biggest drawbacks of fractal image compression is the lengthy encoding time due to the extensive domain pool searching that must be carried out for each range block. This prohibits the use of fractal compression for any real time applications. There is much work being done in this field and there are several publications regarding the use of fractal compression for video coding.

Looking at Jacquin's work, there are some obvious improvements that immediately come to mind. Firstly, the allowed transformations seem somewhat limited. In any image, a better match for some domain block to a certain range block might be found if for instance the domain block is rotated by some angle not equal to one of the existing isometries. More transformations would therefore certainly result in better encoding quality, but with more transformations to represent, the compression ratio would decrease and the larger domain pool that needs to be searched would drastically increase encoding time. In this same line of thought, the choice of possible domain blocks also seem to impair the quality of compression but the price of a wider choice of domain blocks is again a loss of compression ratio and an increased encoding time.

Fisher replaced the four transformations that Jacquin used to change the grayscale pixel values with a simple transformation that consists of a contrast and brightness adjustment.

The search time required to compare a range block with all possible domain blocks and all the different combinations of brightness and contrast scaling for each block would become ridiculously large. Fisher however wrote down a closed form solution to find values for brightness and contrast scaling that minimise the difference between two image blocks as follows:

Consider two images *a* and *b* that contain *n* pixels each. It is desirable to transform *a* to minimise the difference R given by:

$$R = \sum_{i-1}^{n} (s \cdot a_i + o - b_i)^2 \tag{2.20}$$

with s-brightness(offset), o-contrast(scaling)

By partial differentiation with respect to s and o, the solution for the minima can be found and is given on p 21 of Fisher [FisherB].

This method will of course remove a considerable amount of unnecessary comparisons and thus reduce the compression time.

In an article of Øien et al. [Øien], the abovementioned problem is tackled somewhat differently. The idea is again to find a match for the subimage *b* by the scaling and offset of *a*. This gives a new subimage :

$$b' = s \cdot a + t \tag{2.21}$$

with $t = o \cdot m$

where *m* is the unity matrix of size *b*.

b' is now an element of a two-dimensional subspace spanned by the basis $\{s_1, s_2\} = \{m, a\}$ or in simpler terms, *b'* is a linear combination of *m* and *a*. It must be kept in mind that if *b* and *b'* are of size K x L pixels, the images can be seen as K·M-dimensional vectors. By the projection theorem ([Kreysig, p147]), it then follows that the minimum distance $d_{\min}(b,b')$ is found by making *b-b'* orthogonal to the subspace s_i for all i. Thus: $\langle b - b', s_i \rangle = 0$ i = 1, 2. These are called the orthogonality equations.

The next logical step from here, is to try and reduce $d_{\min}(b,b')$ by making *b'* an element of a higher dimensional subspace and thus finding a better match for *b* through the solution of $\langle b - b', s_i \rangle = 0$ for $i = 1..n$. This is exactly what B. Bani-Eqbal did in [Bani]. He kept the greyscale scaling of pixels constant across the whole image block but made the offset value a linear term given by: $o_{ij} = a_1 + a_2 i + a_3 j$ with *i* and *j* the x and y indices of the sub image. This offset can represent any plane of greyscale values in an image block. The idea of the offset is to match the planar component of the range block and the residue must then be fitted by the scale factor multiplied by the domain block. Solutions for the coefficients a_i is then given by $\partial R / \partial a_i = 0$ where R is the difference given above (equation 2.20) except for the offset which is now variable over the summation of all pixels.

Munro and Dudbridge used the same approach in their article [Munro], but set the offset to a third degree polynomial in *i* and *j*. This will increase computation time for the calculation of optimal coefficient values but this added cost was negated by restricting the set of domain blocks considerably.

Jacquin tried another method to decrease encoding time by classifying image blocks into different categories. His classification is based on block classification work done by Ramamurthi and Gersho [Ramam]. He defined three types of blocks - shade blocks, edge blocks, and midrange blocks and classified all blocks into these categories. The domain to range search is then limited to domain blocks of the same class as the range block under scrutiny.

Yuval Fisher went further with this idea by defining three main classes and a further 24 sub classes for each of the main classes. This classification is based on the average and variance distribution of pixel values in an image block [FisherB,

chapter 3]. This classification scheme produced 72 available classes that shortened encoding time considerably but Fischer also showed that this decreases the signal to noise ratio of the decompressed image noticeably.

Another approach to the segmentation problem was attempted by J. Jang and S. A. Rajala [Jang], They segmented the image according to properties of the human visual system (HVS). The modulation transfer function (MTF) of the HVS was used to calculate two thresholds of fractal dimension. These thresholds are used to classify image regions into three textural classes. Since the human eye has a response to spatial frequencies similar to that of a bandpass filter, the eye is less sensitive to some regions and image blocks in different classes can be encoded with different accuracies. This will decrease compression time by decreasing the number of range to domain comparisons and also result in higher compression ratios since some transformations can be represented with less accuracy. The calculation of fractal dimension will add to encoding time but this is still considerably less than the time saved on shortening the comparison time.

To further improve encoding time Jacquin stated in [JacqE, p1458] that optimal domain blocks are normally found in the region of the specified range block and therefore search times can be reduced by starting a search in the vicinity of the range. Fisher contradicts this statement in [FisherB, p76] by saying that there is no local self similarity that can be exploited. Some further evidence is therefore necessary to resolve this issue.

Another method that seemed to deliver very good results in speeding up the encoding of images was discussed by B. Bani-Eqbal ([Bani]). Apart from increasing image fidelity by better block matching as already discussed, he created a tree structure where each node is a pixel value. Each node can therefore branch off into the number of colours available and tree depth is given by the number of pixels in the image. This will create a tree containing all possible images. He reduces the tree by retaining only the branches of a certain range block and all the domain blocks to be compared with it. As the tree is being constructed, a cumulative error calculation between the range and domains are being done. Whenever a domain falls below some tolerance level, the rest of its branches are discarded. This means that it is not necessary to do a complete domain to range comparison for every domain block as before. In practice however this method still seemed inefficient and not only was it slow but the tree structure used up huge amounts of memory. To effectively implement this idea, the tree length had to be decreased. He accomplished this by using averaged

pixel subsets. With this improvement, the results were drastically better. The 'Lenna' image was encoded 58 times faster than the full search method with no visible degradation in image quality.

Up to now, most of the work discussed, regarded the improvement of compression time. This has been one of the main areas of interest in fractal image compression since the long compression times are one of the biggest drawbacks of this method.

In work done by L. Thomas and F. Deravi [Thom], they have managed to double the compression ratio and more than halve the encoding time by a further progression of the segmentation problem. This method seems to be a step closer to a more intelligent and natural compression algorithm. All algorithms discussed so far use a square or rectangular segmentation of images. Because of the 'more random' nature of real world images, it seems logical that better compression ratios as well as higher image fidelity can be obtained by using regions of uniformity as segmented image blocks. This is exactly what they did in this work. Image segments were built up with 8x8 blocks to try and fill a whole self-similar region. As these segments grew, all potential domains would also be enlarged similarly and discarded once they crossed a specified error margin. This decreased the comparison time since domains could be discarded without a full error calculation and the larger ranges also meant less ranges to search for. Fewer ranges to represent would also mean higher compression ratios if the shape information of each range can be represented effectively. This method is very image dependent and the results can therefore only be calculated for specific images. For the 'Lenna' image this method achieved a compression ratio of 41:1 with a PSNR = 26.56 dB as opposed to the 'standard' method which achieved 19:1 with PSNR = 27.86 dB. This method was 2.188 times faster than the 'standard' search method. Details of their standard method can be found on pp 832-833 of Thomas [Thom].

2.1.3 Implementation

This section examines an implementation of the fractal image compression method described by Yuval Fisher in his book on Fractal Image Compression ([FisherB]). It was decided to use this particular implementation since it closely resembles the original work presented by Jacquin ([JacqD]) and can serve as a

basic algorithm to work from. The source code for this algorithm was also available on the Internet and did not have to be implemented from scratch.

A detailed description of the algorithm can be found in chapter 3 of Fisher's book and sample C code of this implementation is given in Appendix A of the same source. The source code can also be downloaded from Fisher's website at http://inls.ucsd.edu/y/ or it can be found on the accompanying CD-ROM. Since the basic theory of this technique has already been discussed in section 2.1.1 and more complete information is available from the source already mentioned, only a brief description of the technique and algorithm will be given here to avoid unnecessary duplication of work.

The quadtree implementation of fractal image compression described here, is based on the algorithm presented by Jacquin in [JacqD]. It can be considered as a very basic method that can serve as a point of reference with which to compare and measure the results of other algorithms and improvements. A flow diagram of the algorithm is shown in Figure 2-8.

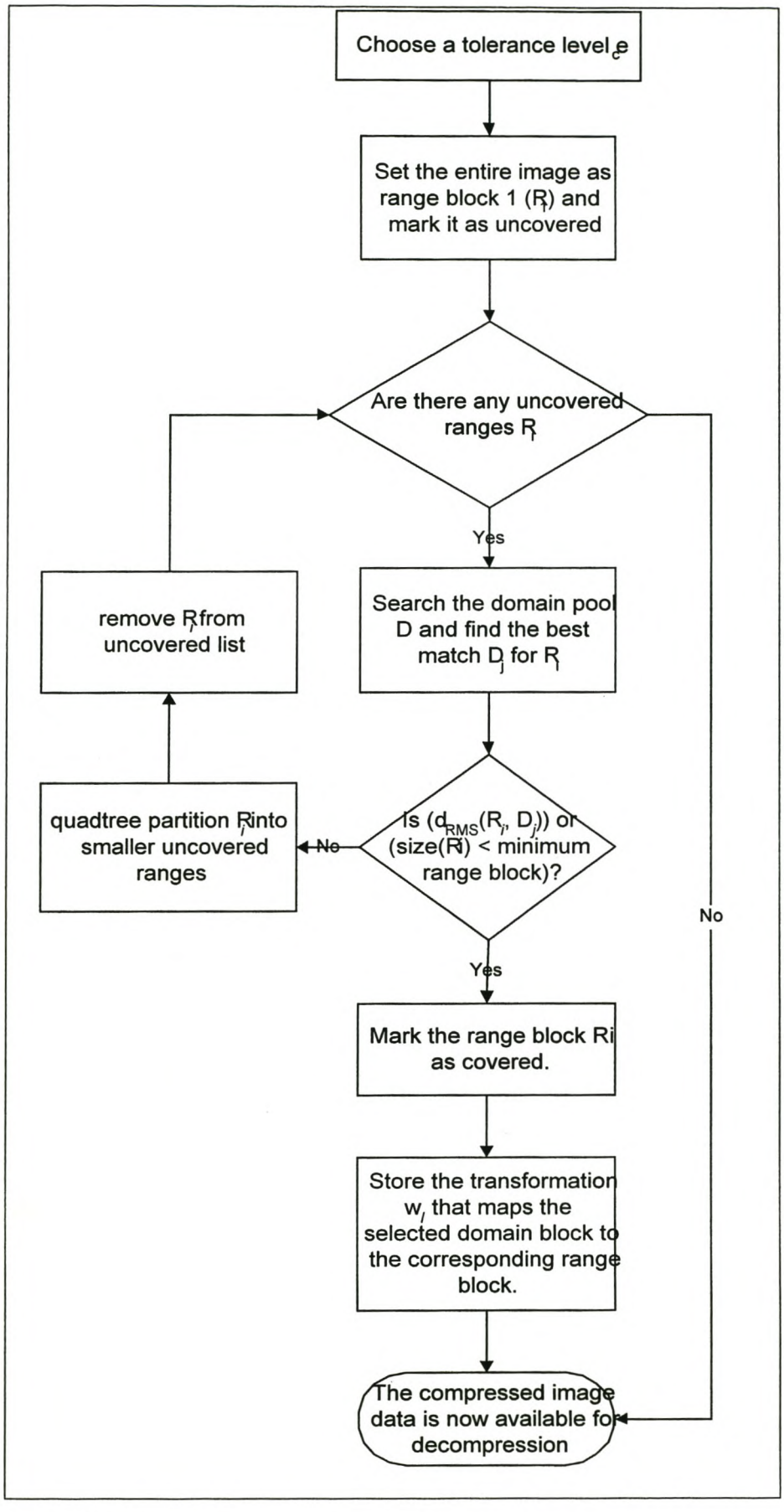


Figure 2-8 Fisher's quadtree compression algorithm

The idea of the quadtree algorithm is to take each square range block for which no suitable domain transformation could be found and partition it into four subsquares. In this way, the range blocks form a tree like structure with the largest subsquare of the image as root. For each range block, the domain pool consists of image blocks twice the size of the range block. The pixel values of the domain blocks are averaged in groups of four to shrink the domain to the range block size. The optimal affine transformation is then found to match the domain with the range through a scaling and offset adjustment of the pixel values. The optimal transformation is found through equation 1.5 on p.21 of Fisher ([FisherB]). Unfortunately, the optimal scaling values can be found to be greater than some arbitrary $|s_i|$ and this can cause the resulting map not to be eventually contractive. There is however, no closed form solution to calculate the upper limit of scaling values at which contractivity will be lost. This parameter must therefore be experimented with to obtain some value for s_i. It must also be kept in mind that the scaling and offset values must be quantised. Storing the exact calculated values, would reduce compression ratio considerably.

As was mentioned earlier, the domain pool is made up of image blocks twice the size of the range block that must be covered. Looking at all possible blocks twice the size of a certain range block would increase comparison time drastically. Fisher uses three types of domain libraries $D_{1,2,3}$ whose components are subsquares of the image with their upper left corners positioned on a lattice with spacing *l*. The domains are defined as follows:

D_1, a lattice with fixed spacing *l*.

D_2, a lattice with spacing equal to the domain block size divided by *l*. Thus, the smaller the domain, the more domain blocks.

D_3, a lattice with the opposite spacing size relationship than the previous domain group.

All of these possible domain blocks must still be rotated/flipped in the 8 allowed orientations, and from these the optimal scaling and offset coefficients must then be calculated.

Choosing one of the defined domain libraries still leaves a lot of domains that need to be compared with each range block. To further decrease the comparison time, Fisher classified domain blocks according to their average and variance

distributions. By calculating the average pixel values of each quadrant, all blocks can be put into one of three categories (see fig. 3.2 p. 58 of Fisher [FisherB]). By further calculating the variance of each quadrant, every category can be subdivided into an additional 24 sub classes. Negative scaling values also influence the ordering of the three main classes to give an additional three orientations. Thus there are two orientations with three main classes each, and an additional 24 subclasses to each of these.

Encoding time can therefore be influenced by deciding whether all domains are compared to a specific range or just a main class or even just a subclass. This decision can be made during operation and is dependent on the encoding time and quality tradeoffs required by the application.

The data format of the encoded image with its domain to range mappings and the quantised transformation values are by no means optimally stored and can still be compressed losslessly through some statistical method like Huffman coding or run length encoding.

Decoding is accomplished by simply performing the transformations W iteratively upon any initial image. This process is repeated with the resultant image of each iteration until the successive images do not increase significantly in quality from one iteration to the next.

As was mentioned earlier, it must just be kept in mind that that convergence is not necessarily guaranteed and the maximum scaling must be set so that divergence is avoided in decompression. The speed of convergence is also influenced by the choice of the initial image, but this effect is small and not very important.

A feature of fractal image compression is that an image can be decoded at different sizes, independent of the size of he original image. This implies that an image can be enlarged and due to the nature of this decompression method, there is no occurrence of pixelization that accompanies the pixel doubling methods of enlargement. Edges and lines are therefore maintained with enlargement at the price of some artifacts being introduced into the decompressed image. These artifacts are however more natural looking and therefore less disturbing to the human eye than other methods. An example of this effect can be found on p. 60 of Fisher [FisherB].

In the final decoded image, there are also edge effects visible at the borders of all range blocks. One way to reduce this effect is to perform some form of post processing on the last iteration of the decoded image. There are several ways to

accomplish this and Fisher uses a simple averaging formula to adapt the values of pixels a and b at the 2 sides of a range block boundary. The new values of the pixels are given by a' and b' in the equations given below.

$$a' = \omega_1 a + \omega_2 b \tag{2.22}$$

and

$$b' = \omega_2 a + \omega_1 b \tag{2.23}$$

with

$$\omega_1 + \omega_2 = 1 \tag{2.24}$$

Ranges at the maximum quadtree depth are averaged with $\omega_1 = 5/6$ and $\omega_2 = 1/6$ whilst other ranges use $\omega_1 = 2/3$ and $\omega_2 = 1/3$.

The source code supplied for this algorithm was incorporated into a Windows graphical user interface application for ease of use and is included in the accompanying CD-ROM. The software can load any Windows bitmap image and compress it according to the specified parameters. The compressed image can be decompressed from any selected initial image and each step can be viewed to follow the convergence of the final image. The application also calculates the SNR of the decompressed image against a given original image.

2.1.4 Performance

2.1.4.1 Test Data

In order to have a consistent set of test data that can be used throughout this work, 30 digital images were acquired through various means. The set of images all share the common theme of satellite images and aerial photographs. The images vary from very high altitude high resolution images to lower resolution photographs and include everything from natural landscapes and man made structures to weather phenomena. The images also include images taken in the visual range, water vapour and infra red ranges. The images also differ in quality from very high quality images to very blurry images.

The test images have all been cropped to the same size of 512x512 pixels and are displayed as 256 level grey scale images. The images are supplied on the accompanying CD-ROM.

The test images have all been cropped to the same size of 512x512 pixels and are displayed as 256 level grey scale images. The images are supplied on the accompanying CD-ROM.

2.1.4.2 Test Results

Using the Fisher Quadtree implementation described in section 2.1.3, the 30 test images were compressed with the parameters given in Table 1.

Parameter	Value
Tolerance level	2.0
Min recursion depth	4
Max recursion depth	6
Domain pool step size	1
Domain pool type	0
Domain pool step type	0
Scaling bits	5
Offset bits	7
Max scale factor	1.0
Positive scale values only	Yes
Search 24 domain classes	Yes
Search 3 domain classes	Yes

Table 1 Quadtree compression parameters

The implications of the parameters on the compression is described in section 2.1.3 and explained in more detail in Appendix A of [FisherB]. Table 2 lists the results obtained from application of this algorithm to the 30 test images. The images required between 7 and 15 minutes to compress on a 90Mhz Pentium processor running the Windows NT operating system. This time is not very accurate however since the tests were performed in conjunction with other processes executing simultaneously on a multitasking system. To correctly

measure the compression time, it would be best to calculate the number of operations required for compression. This value can then be used for comparison to other techniques regardless of relative CPU speed.

	Image	Compression Ratio		Image	Compression Ratio
1.	Bangladesh	18.7440 : 1	16.	isle2	25.2540 : 1
2.	Clouds	18.8717 : 1	17.	night1	18.7480 : 1
3	Clouds2	19.1574 : 1	18.	night2	18.7440 : 1
4.	Clouds3	19.1811 : 1	19.	night3	18.7440 : 1
5.	frozen	18.8043 : 1	20.	ray1	24.1134 : 1
6.	frozen2	18.7453 : 1	21.	ray2	22.5825 : 1
7.	frozen3	18.7440 : 1	22.	ray3	22.7033 : 1
8.	goes1	19.2555 : 1	23.	san francisco	19.1004 : 1
9.	goes2	19.2808 : 1	24.	seattle	20.6497 : 1
10.	goes3	19.1671 : 1	25.	spain	18.7440 : 1
11.	goes4	19.1017 : 1	26.	typhoon1	21.6163 : 1
12.	hurricane1	21.3498 : 1	27.	Typhoon2	20.0413 : 1
13.	hurricane2	18.7949 : 1	28.	Valley1	19.1253 : 1
14.	hurricane3	19.2583 : 1	29.	Valley2	18.7694 : 1
15.	isle1	24.7273 : 1	30.	Venus crater	18.7440 : 1

Table 2 Fractal compression results

The decompressed images are available for viewing on the accompanying CD-ROM. As can be seen from the table above, the compression ratios achieved are relatively high and although the perceived quality of the compressed images is somewhat degraded, they are still acceptable for certain applications.

The effects of fractal compression techniques are clearly visible in the example shown below.

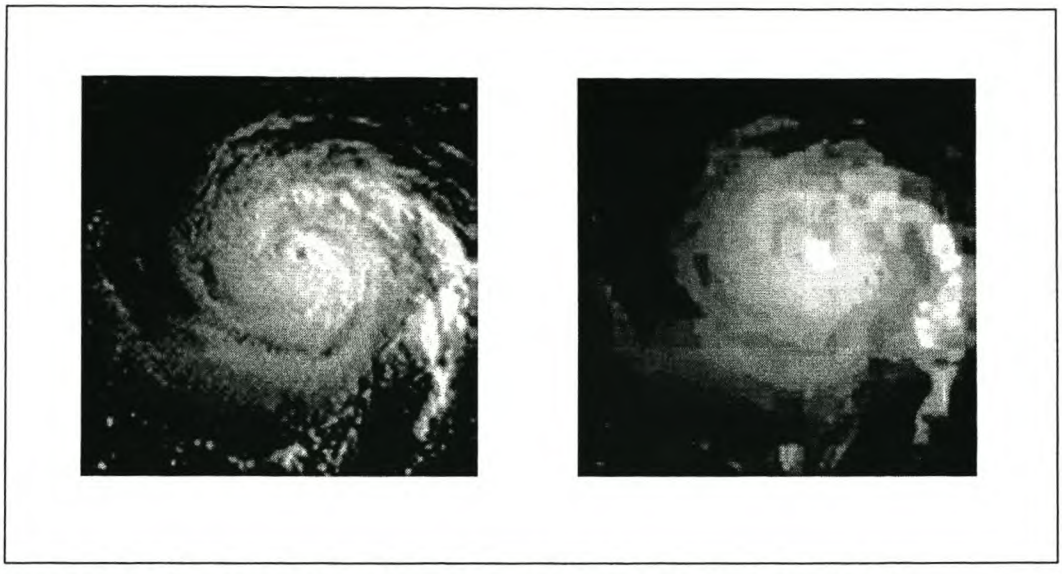


Figure 2-9: Effects of fractal compression on an image (enlarged 2 x)

As can be seen from the decompressed image segment on the right in Figure 2-9, the blocky effects resulting from the image segmentation is clearly visible. Even with the segmentation effects, it is also possible to see that this method does retain curves (observe part of the top right arm of the cloud spiral).

JPEG Image Compression

The Joint Photographic Experts Group proposed this method of compression to set a standard that can be used across different systems and in many different applications. JPEG image compression exploits redundancies in image data as well as non linearities of human vision.

JPEG compression can be divided into four different modes or techniques as listed below:

- *Sequential DCT-based encoding*: This method encodes images in left to right top to bottom 8x8 segments.
- *Progressive DCT-based encoding*: This method encodes an image at multiple scans to provide a rough first image and iteratively increase the image quality as more data is received. This method is particularly suited to cases where transmission times are very slow and a preview of the image can be very helpful.
- *Lossless encoding*: This method produces an exact copy of an original image with the related bounds on compression ratio.

- *Hierarchical encoding*: This technique is used to compress the image at several resolutions. Small images are then decompressed at lower resolutions and gradually decompressed to maximum resolution.

Sequential DCT-based encoding is the most popular technique currently used and also the one that will be discussed in this chapter. This compression method is based on the two dimensional discrete cosine transform which is described in the next section. That section will also consider the Huffman coding technique for lossless compression and any other necessary theory for JPEG compression.

Section 2.1.6 puts all of these tools together in a description of the JPEG compression algorithm and illustrates this with a step by step example.

In section 2.1.7, the algorithm is applied to the test data used in the fractal image compression test and these results are discussed.

2.1.5 Mathematical background

2.1.5.1 The two dimensional discrete cosine tranform

The N x N discrete cosine transformation *v* for an image *u*, is defined by equation 2.25 and its inverse is given by equation 2.26. The crux of this transform is given by the cosine transform matrix *c(k.n)* given in equation 2.27. The inverse transform can be found by using the complex conjugate of the transform matrix, but since this matrix is real, its complex conjugate is identical to itself.

$$v(k,l) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} c(k,m)u(m,n)c(l,n) \tag{2.25}$$

$$u(m,n) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} c^*(k,m)v(k,l)c^*(l,n) \tag{2.26}$$

$$c(k,n) = \begin{cases} 1/\sqrt{N} & \mathrm{k} = 0, 0 \le \mathrm{n} \le \mathrm{N}-1 \\ \sqrt{2/N}\cos\dfrac{\pi(2\mathrm{n}+1)\mathrm{k}}{2N} & 1 \le \mathrm{k} \le \mathrm{N}-1, 0 \le \mathrm{n} \le \mathrm{N}-1 \end{cases} \tag{2.27}$$

This transform has many properties including some that make it particularly suitable to image compression.

Firstly, the DCT has very good energy packing qualities. Looking at the results of a 2 dimensional DCT upon a standard image, it can easily be seen that only the lower spatial frequencies (those closer to the (x=0, y=0) axis) have any significant values and the higher spatial frequencies have much smaller values or are generally zero. Consider the example given in section 2.1.6.3.

The DCT is also a very fast transform and can be calculated in $O(N \log_2 N)$ operations via an N point FFT. Since the cosine terms of equation 2.27 are also independent of the pixel values, the DCT calculation can be reduced to a matrix multiplication with a precomputed 2 dimensional cosine matrix to save calculation time.

More information on the properties and uses of the DCT can be found in chapter 5.6 of [Jain].

2.1.5.2 Run length Coding

Run length coding is a simple technique that exploits repetitions of a single symbol or sub sequence within a sequence. This form of coding is lossless but also very limited in compression ratio due to the constraints on lossless coding and the occurrence of the data property that this technique tries to exploit.

The technique can best be illustrated using a simple and very contrived example.

Consider a 256 level gray scale image where the pixels are stored sequentially from the top left to the bottom right pixel. Assume that although pixel values are stored in 8 bit values, the image only contains pixel values in the range [0..254] with 255 being reserved to indicate a run length of a specified symbol. Whenever a long sequence of the same value appears in an image (in a very uniform region for example), the actual row of pixel values can now be replaced by the predefined run length symbol followed by the pixel value of the sequence and the number of consecutive pixels with this value. If there are more pixels with the same colour than can be represented by one byte, the sequence of run length symbol followed by colour value and sequence length can be repeated as many times as necessary. Obviously, this technique will only be useful for sequences longer than 3 consecutive symbols.

This method can be expanded to not only include repetitions of single symbols but also repetitions of symbol sequences. This expansion forms the basis of the

Lempel-Ziv coding technique upon which the GIF image compression format is based.

2.1.5.3 Huffman Coding

The idea of Huffman coding is to assign each possible value a certain binary code with more likely values given shorter codes whilst unlikely values are coded with more bits. Depending on the statistical nature of the data to be coded, this can cause a considerable saving in storage space (Approximately 35% in English text [BarnC, chapter 5]). An example would best illustrate the idea:

Consider 5 symbols with the probabilities: p_A=0.264, p_B=0.053, p_C=0.108, p_D=0.137, p_E=0.438.

To construct a Huffman code for these symbols, rearrange them from smallest to highest probability. Assign the two smallest values a binary 1 and 0. Next merge the two smallest values to form a new symbol BC with a probability the sum of the two separate values and start again from the top. Repeat this process until only one symbol remains that contains all the others. To find the Huffman code for a certain symbol, start from the main symbol and simply follow the tree down to where it terminates in the wanted symbol, reading off the binary values after each branch. It is easy to see that more likely symbols will have less branch points and thus shorter codes. To decode a message, each symbol must simply be followed down the code tree until a terminating symbol is reached. One drawback to this method is that when there are very many symbols, this can lead to extremely long code words for unlikely symbols. If the codebook has to be included in the compressed file, this can easily negate the effect of the compression. The codebook for a certain set of symbols with probabilities is also not unique and therefore two machines using different algorithms to calculate the codebooks might not necessarily understand each other.

2.1.6 Implementation

This two dimensional discrete cosine transform on which JPEG compression is based, has very good energy packing qualities [Jain, chapter 5] as has already been mentioned in section 2.1.5.1 and will be illustrated later in this chapter. This transform is combined with several other compression techniques to achieve

maximal compression with minimal visible loss in image quality. This section examines each step of the compression technique in detail and includes an example to show the actual implementation of each step as the compression progresses. The example given in the following sections is taken from [Fuhrt]. Figure 2-10 shows the different steps of the JPEG compression algorithm.

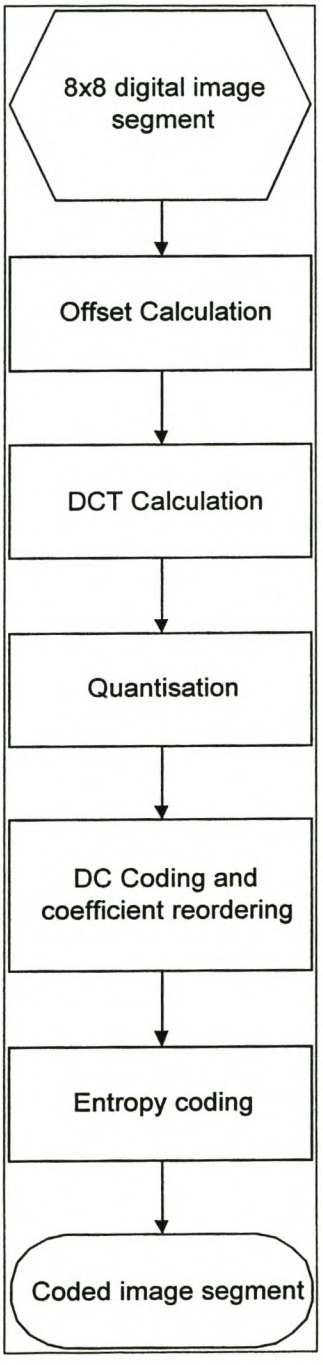


Figure 2-10 JPEG Compression Algorithm

2.1.6.1 Image Segmentation

As with most forms of image compression, the actual application of the compression algorithm is not performed on the image as a whole, but on segments of the image and all the compressed segments are put together in the end to rebuild the decompressed image. JPEG has decided to use a default block size of 8 x 8 pixels and each of these blocks are separately coded using the rest of this algoritm.

The choice of 8x8 image segments is based on the fact that the 8x8 2 dimensional DCT for real world images has very small or zero coefficients in the higher spatial frequencies [Fuhrt, p55]. This value is based on extensive research and experimentation on the eventual difference in compression ratio given different segment sizes.

The block size also imposes the condition that images must be of a size that is a multiple of the segment size. This limitation is overcome by various means that are not discussed in this work.

As an example, consider the following 8 bit (256 level) grey scale image segment u(m,n) shown in Table 3. This image segment will be used in the examples of the rest of the steps required for the compression algorithm.

140	144	147	140	140	155	179	175
144	152	140	147	140	148	167	179
152	155	136	167	163	162	152	172
168	145	156	160	152	155	136	160
162	148	156	148	140	136	147	162
147	167	140	155	155	140	136	162
136	156	123	167	162	144	140	147
148	155	136	155	152	147	147	136

Table 3 8x8 Gray scale image segment

2.1.6.2 Offset calculation

During this stage, the pixel values are offset to the signed range $[-2^{(p-1)}, 2^{(p-1)} - 1]$ from the unsigned range $[0, 2^p - 1]$, where p is the bits per pixel of the image.

For an 8 bit 256 level grey scale image (as used in the test data) this equates to a translation of pixel values in the range from [0, 255] to [-128, 127].

The reason for this step in the process of compression, is to further reduce the size of the data. If it is assumed that the average grey scale level of an image is close to the middle of the range of possible pixel values, a translation as described above, would put the average very close to zero. Since the DCT

coefficient of a two dimensional cosine transform is the average and also the element with the most image energy, it would seem that this translation is ideal for minimising the size of the representation of this component of the image data.

The results of the offset calculation on the sample data shown in Table 3 can be seen in the table below.

12	16	19	12	11	27	51	47
16	24	12	19	12	20	39	51
24	27	8	39	35	34	24	44
40	17	28	32	24	27	8	32
34	20	28	20	12	8	19	34
19	39	12	27	27	12	8	34
8	28	-5	39	34	16	12	19
20	27	8	27	24	19	19	8

Table 4 Image segment after offset calculation

During the decompression phase, the pixel values are offset to their original range by performing the reverse of this operation.

2.1.6.3 Discrete Cosine Transformation

The discrete cosine transform and its inverse given in equations 2.25 and 2.26 can be specialised for 8x8 image segments to give the following equations:

$$v(u,v) = \frac{C(u)}{2}\frac{C(v)}{2}\sum_{x=0}^{7}\sum_{y=0}^{7} u(x,y)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16} \tag{2.28}$$

$$u(x,y) = \frac{1}{4}\left[\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)v(u,v)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}\right] \tag{2.29}$$

$$C(u) = \frac{1}{\sqrt{2}} \quad \text{for u} = 0$$
$$C(u) = 1 \quad \text{for u} > 0 \tag{2.30}$$

Application of the forward cosine transform on the image segment given in Table 4 results in the data shown below. The values shown in the table have been rounded to the equivalent integer values since the quantisation step that will follow this phase will cause a much bigger data loss, effectively negating the rounding error.

185	-17	14	-8	23	-9	-13	-18
20	-34	26	-9	-10	10	13	6
-10	-23	-1	6	-18	3	-20	0
-8	-5	14	-14	-8	-2	-3	8
-3	9	7	1	-11	17	18	15
3	-2	-18	8	8	-3	0	-6
8	0	-2	3	-1	-7	-1	-1
0	-7	-2	1	1	4	-6	0

Table 5 Cosine transform of image sement of Table 4

2.1.6.4 Quantisation

This is the step that performs the actual compression, and is also where the data loss occurs. The compression ratio and image fidelity is set by changing the quantization levels. Since the different frequency components contain different amounts of the image energy, each component can be quantized by a different amount. Several quantization tables have been suggested that truncate the coefficients so that none is represented by greater precision than is necessary for a desired image quality. Borko Fuhrt [Fuhrt] gives a routine for the generation of a quantization table for any specified quality factor. The recommended quality factors lie between 1 and 25 where 1 gives the highest quality but lowest compression ratio. Other quantization tables can also be used, but these must be included as part of the compressed image(thus reducing compression quality somewhat).

The formula for generating a quantization table for an n x n image segment given in [Fuhrt] is shown below:

```
For i = 0 to n {
   For j = 0 to n {
      QuantizationValue[i,j] = 1+ (1+i+j)*quality
   }
}
```

Using this algorithm, the quantization table shown in Table 6 is generated.

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

Table 6 8x8 Quantization table for a quality factor of 2

The 8x8 image segment is then quantized using equation 2.31 where *Q(u, v)* are the quantization coefficients produced in Table 6 and this results in the image segment shown in Table 7.

$$F_q(u,v) = Round\left[\frac{F(u,v)}{Q(u,v)}\right] \tag{2.31}$$

61	-3	2	0	2	0	0	-1
4	-4	2	0	0	0	0	0
-1	-2	0	0	-1	0	-1	0
-1	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	-1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Table 7 Image segment after quantization

2.1.6.5 DC Coding and coefficient reordering

The DC coefficient of each block is a measure of the average pixel value of the block. For real world images, there is a strong correlation between adjacent

blocks and therefore the DC coefficient is coded as the difference from the DC term of the previous block. This is worthwhile since a significant amount of image energy is contained in the DC coefficients and the difference can be coded with higher accuracy. The range of the DC coefficients is also double that of the AC coefficients.

All these coefficients are then reordered into a zig-zag sequence as shown in [Wall, p35]. This specific sequence is in order of ascending spatial frequency and this correlates well with the actual descending pixel values at this stage of the compression process. This greatly facilitates the entropy coding step.

The reordering phase reduces the 8x8 matrix give in Table 7 to the sequence listed below:

61, -3, 4, -1, -4, 2, 0, 2, -2, 0, 0, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, -1, 0, 0, -1, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

2.1.6.6 Entropy Coding

The entropy coding stage achieves additional lossless compression by making use of the statistical properties of the already compressed data.

Firstly, the data sequence is converted into an intermediate symbol sequence. AC coefficients are coded as a pair of symbols as shown below

symbol 1: (runlength, size)

symbol 2: (amplitude)

The runlength symbol gives the number of consecutive zero valued AC coefficients preceding a nonzero AC coefficient. The value of runlength is in the range 0 to 15 which requires 4 bits to represent. The size symbol is the number of bits used to encode amplitude which is the value of the non zero coefficient. If the runlength is greater than 15, the symbol 1: (15, 0) is interpreted as an extension symbol with 16 zeros. There can be up to 3 consecutive runlength extensions before a terminating symbol 1. The symbol 1 that actually ends the sequence is designated by the special symbol (0, 0) to signal the end of the block.

The DC coefficients are coded in the same way but without the runlength symbol and with double the range of the AC coefficients's amplitudes.

The symbol sequence for the example data sequence shown in the previous section is shown below:

(6)(61),

(0, 2)(-3),

(0, 3)(4),

(0, 1)(-1),

(0, 3)(-4),

(0, 2)(2),

(1, 2)(2),

(0, 2)(-2),

(5, 2)(2),

(3, 1)(1),

(6, 1)(-1),

(2, 1)(-1),

(4, 1)(-1),

(7, 1)(-1),

(0, 0)

The next part of the entropy coding stage is the Huffman coding stage. [Wall] mentions arithmetic coding as an alternative but also states that this method is too complex for certain implementations.

To effectively implement this in the JPEG compression, the following can be done. At this stage the data is in the intermediate symbol form. The amplitude symbols have too many possibilities to code effectively and they will simply be coded using the standard binary representation. The DC component *size* symbol can have 11 possible values and the AC component's (*runlength, size*) symbols can have a further 162 possible values (see [Wall]). By calculating the intermediate symbol sequence for a specific image, a codebook can be constructed for the image that has maximum efficiency. This calculation for every image will however be very time consuming and the codebook for every image must be included in the compressed file. For very large images, the codebook does not affect the compression ratio too much but it can influence smaller files considerably. Another more effective way to overcome this problem is to define some specific codebooks based on certain types of images and include these codebooks as standard in both the coder and decoder software. Only a pointer to the specific codebook needs to be included then. To keep code words from becoming unmanageably long, the JPEG group have also specified that code words may only be Huffman coded to a length of 16 bits. After this, each word must be designated by an additional 4 bits using standard binary representation. This is an acceptable trade-off since the additional bits allow for all possibilities arising from the possible distribution of the available symbols.

After the Huffman coding phase, the data is represented by the following bit sequence:

1110111101001001000001000110110110111001011111111011111011101011
1110110111000111011011111101001010

The original image segment has been reduced from 64 pixels at 8 bits per pixel (512 bits) to a total of 98 bits. This is a compression ratio of 5.22:1.

2.1.6.7 Decompression

Decompression of the image can be achieved by applying each of the above steps in reverse by using the inverse equations given in the relevant steps. All the decompressed image segments must then be added to reproduce the original image with the losses introduced by the quantisation step.

2.1.7 Performance

The JPEG algorithm was tested using a freely available compression program ([ftp]) which is included in the accompanying CD-ROM. The program takes as

parameters only a single value between 0 and 100. This value determines the level of compression where 100 means the lowest level of compression for the highest quality.

There are other parameters available for greater control over the compression algorithm but for this case only the quality level was used.

As was already mentioned in a previous chapter, the three parameters that image compression are measured by is compression/decompression time, compression ratio and compression quality. In order to make any useful comparison between different methods of image compression, some of these factors have to be fixed. Since the next chapter explores some methods for calculating image quality, the compression ratio will be fixed to the same level as that obtained for each image using the fractal compression algorithm. The speed of compression and decompression will be ignored for now since they are difficult to compare. It can be noted however that the fractal compression algorithm is several orders of magnitude slower than the JPEG algorithm.

Since the JPEG compression algorithm is controlled by a quality factor which results in a level of compression corresponding to that quality value, the quality values for specified compression ratios were calculated iteratively. A small program was written to compress each image at the average quality setting. If the compression ratio was too high, the image was compressed again using a higher quality value and a lower quality value if the compression ratio was too low. The closest possible match to the required compression ratio was normally found within 7 or 8 iterations. This technique might seem like a very expensive operation but since it was only required to be performed once, it was the quickest and most accurate way. The source code and executable for this program can be found on the accompanying CD Rom. The following paragraph shows an excerpt from the program output describing the iteration steps for obtaining the compression quality values for required compression ratios. Table 8 shows the required compression ratios with the actual obtained values and the corresponding quality values for the 30 test images.

Image 18: night3

Attempting compression ratio of 18.744001

Trying quality factor 50 => getting ratio of 11.357525

Trying quality factor 25 => getting ratio of 19.871810

Trying quality factor 38 => getting ratio of 14.034764

Trying quality factor 32 => getting ratio of 16.274391

Trying quality factor 29 => getting ratio of 17.553985

Trying quality factor 27 => getting ratio of 18.595691

Trying quality factor 26 => getting ratio of 19.188074

Trying quality factor 27 => already calculated getting ratio of 18.595691

So I think the best qf is: 27 with a compression ratio of 18.595691

	Image	Desired Compression Ratio	JPEG Compression Ratio	Quality factor
1.	Bangladesh	18.744001	18.123244	7
2.	Clouds	18.871666	18.585187	7
3	clouds2	19.157351	20.012317	7
4.	clouds3	19.181083	19.547156	13
5.	Frozen	18.804258	18.760031	33
6.	frozen2	18.745335	18.932748	31
7.	frozen3	18.744001	18.788151	31
8.	goes1	19.255450	19.605393	11
9.	goes2	19.280838	19.643433	7
10.	goes3	19.167116	18.862200	11
11.	goes4	19.101742	19.554416	14
12.	Hurricane1	21.349826	21.424548	41
13.	Hurricane2	18.794859	18.685455	57
14.	Hurricane3	19.258267	19.009316	50
15.	isle1	24.727290	24.836950	15
16.	isle2	25.253958	25.434535	21
17.	night1	18.748006	18.962755	21
18.	night2	18.744001	18.765381	22
19.	night3	18.744001	18.595691	27
20.	ray1	24.113412	24.485767	47

21.	ray2	22.582533	22.740562	48
22.	ray3	22.703295	22.561241	47
23.	san francisco	19.100356	19.708146	10
24.	seattle	20.649722	19.826906	10
25.	spain	18.744001	19.801550	8
26.	typhoon1	21.616326	21.351557	10
27.	typhoon2	20.041267	20.726142	8
28.	valley1	19.125336	19.135068	8
29.	valley2	18.769395	18.366034	8
30.	venus crater	18.744001	18.919140	43

Table 8: JPEG Compression results

The decompressed images listed in the table above, are available for viewing on the accompanying CD-ROM.

The effects of the JPEG compression scheme can clearly be seen in the example shown below.

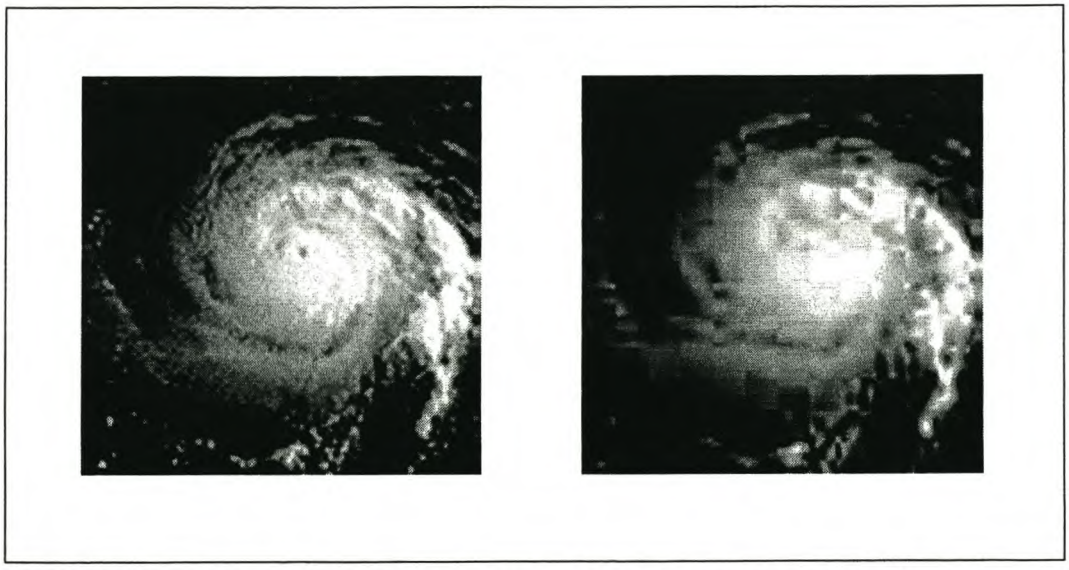


Figure 2-11: Effects of JPEG compression on an image (enlarged 2 x)

The decompressed image segment on the right in Figure 2-11, clearly shows the regular 8x8 block segmentation effects caused by JPEG compression. It

can also be seen that the high resolution detail (regions with high spatial frequency) are degraded most as can be expected from the nature of the DCT.

3. Image Quality Measurement

In the previous chapters, the Signal to Noise Ratio was mentioned as a measure of the amount of degradation introduced by the compression algorithm. It has also been hinted that this measure is not ideal for the calculation of the perceptual quality of compression. Yuval Fisher devotes an appendix of his book on fractal image compression to the comparison of different compression techniques [FisherB] using the SNR measure, but he also gives a long list of reasons why this method is not ideal. Although it seems that there are some problems with SNR as a measure for image quality, it is still the most widely used in publications for lack of something better.

The calculation/measurement of the quality of image compression differs from other types of quality measurements in that the original image is available for comparison and therefore the error is exactly quantifiable to give what is expected to be a good objective judgement of the image quality. The calculated quality is therefore purely a function of the original and compressed images.

With image restoration and enhancement, the resulting image is influenced by the means of acquisition. This means for example that a bad lens or movement of the camera can lead to degradation in quality. Unfortunately, the original image is not available and it is difficult to quantify the quality of an image. It can however be derived from knowledge of the cause of degradation.

This chapter will describe some of the problems involved in measuring image compression quality and will then take a look at some specific measures. The last part of this chapter discusses the means of comparing the results of these different measures to find an acceptable reference for performance evaluation.

Performance Criteria

3.1.1 The problem

As described in the introduction to this section, image quality measurement should be a straightforward operation for compressed images where the source images are available. Unfortunately the results obtained from standard quality calculations

have very little correlation with human perception of image quality. This is because the nature of different types of image distortion have varying degrees of effect on the human visual system. Image degradation due to compression can be caused by block artefacts, brightness scaling, segmentation borders and many other factors induced by the specific type of compression used.

Another factor which plays an important role is the perceived quality of the image being compressed. If the original image used for compression did not have very high perceptual quality, it is very easy for a subjective judge to give the compressed image a bad quality rating although the original and degraded image might be a very close match. It is important to keep in mind that in judging the quality of image compression, only the comparison between images should be used and not the absolute quality of the test image.

There are two main types of image fidelity criteria namely the quantitative and subjective methods. The quantitative method is easy to implement and automate and gives consistent results for the same tests. The subjective method requires human intervention and is normally very slow and not at all consistent. It seems obvious which is the better choice but unfortunately these two methods do not correlate very well in their results. Since humans are the ultimate users of most digital images, their subjective perception of image quality cannot simply be discarded in favour of the more easily obtainable measure. This is especially true for the case where these measures disagree strongly. It would be useless to grade image compression quality through a quantitative method if the humans that used the compressed images were not happy with the level of quality.

Obviously the object is to find some objective measure that agrees better with human perception. Consider as an example, some of the images used in the previous two chapters. The quality of the compressed images shown below were calculated using the SNR as defined in equation 3.34 in section 3.1.3.

Consider the following examples:

Figure 3-1 shows a section of a GOES satellite image together with the same image compressed and decompressed using fractal compression (refer to Section 0) and JPEG compression (refer to Section 0). The JPEG image has an SNR of more than 3dB better than the fractal image which implies that it should be twice as good as the fractal image when compared to the original.

Figure 3-2 shows a portion of an aerial photograph of a hurricane. In this case, the JPEG image is approximately 0.2dB better than the fractal image. The cloud

pattern in this image very subtle and the images are very similar, but this example was specifically included to illustrate the point that SNR calculations can show a difference that is visually insignificant.

Figure 3-3, which shows an aerial shot of a valley, the JPEG image's SNR is 2dB better than the fractal image although the images both resemble the original closely. The segment borders introduced by compression are more visible in the JPEG image that should result in a lower SNR than that of the fractal image.

It would be wrong to subjectively decide that the SNR measure is inaccurate but it would also be fair to surmise from the examples that there is at least room for improvement in the measurement of image compression quality through SNR calculation. The examples used small portions of larger images because a larger image can easily conceal mistakes and the human eye tends to be very forgiving of something in the proper context. The examples were also enlarged to accentuate the errors introduced during compression (which is incidentally also a very good illustration of the errors introduced by the two different methods of compression).

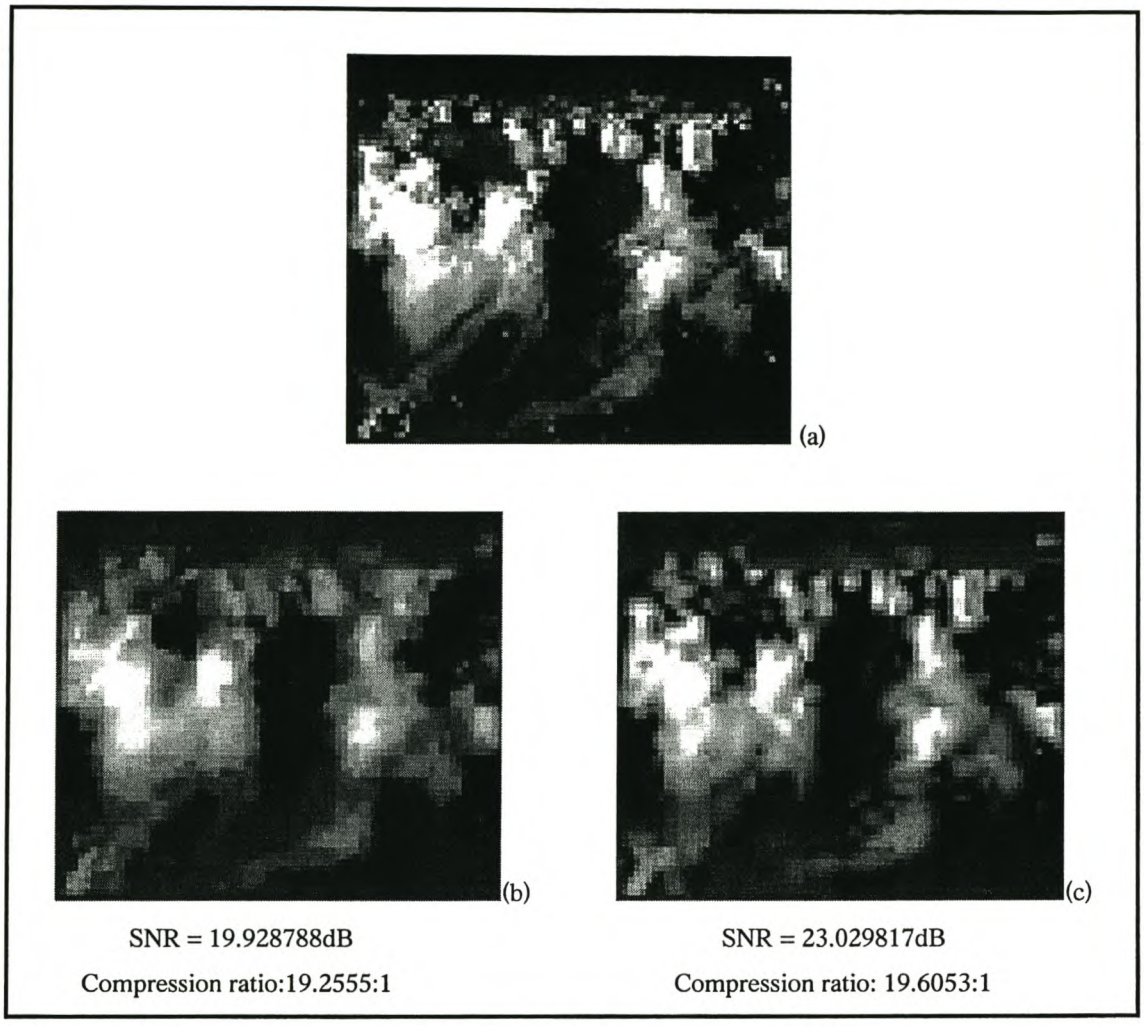


Figure 3-1 A 3x enlarged portion of the 'goes1' image with the original (a), fractal (b) and JPEG (c)

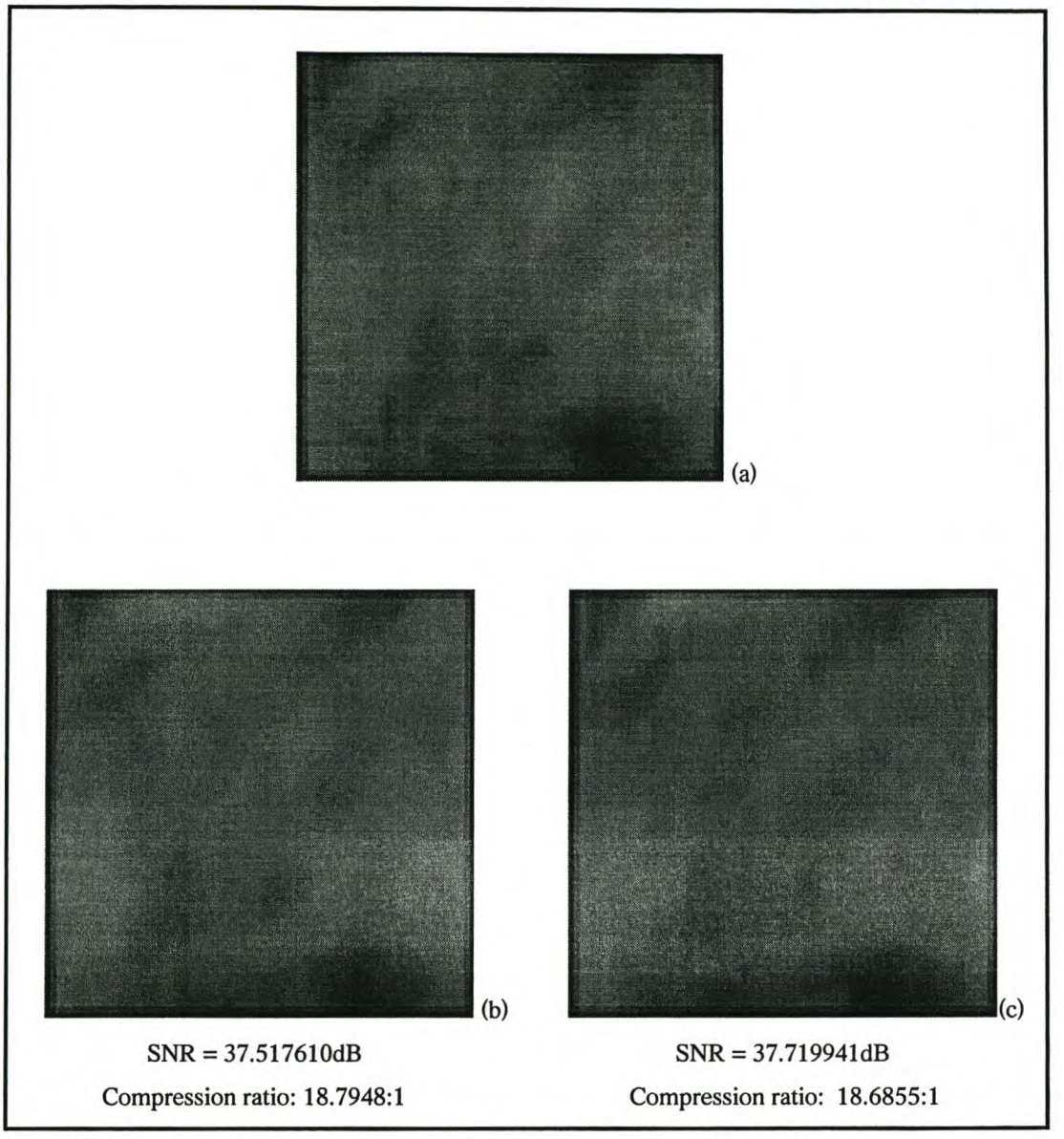


Figure 3-2 A 3x enlarged portion of the 'hurric2' image with the original (a), fractal (b) and JPEG (c)

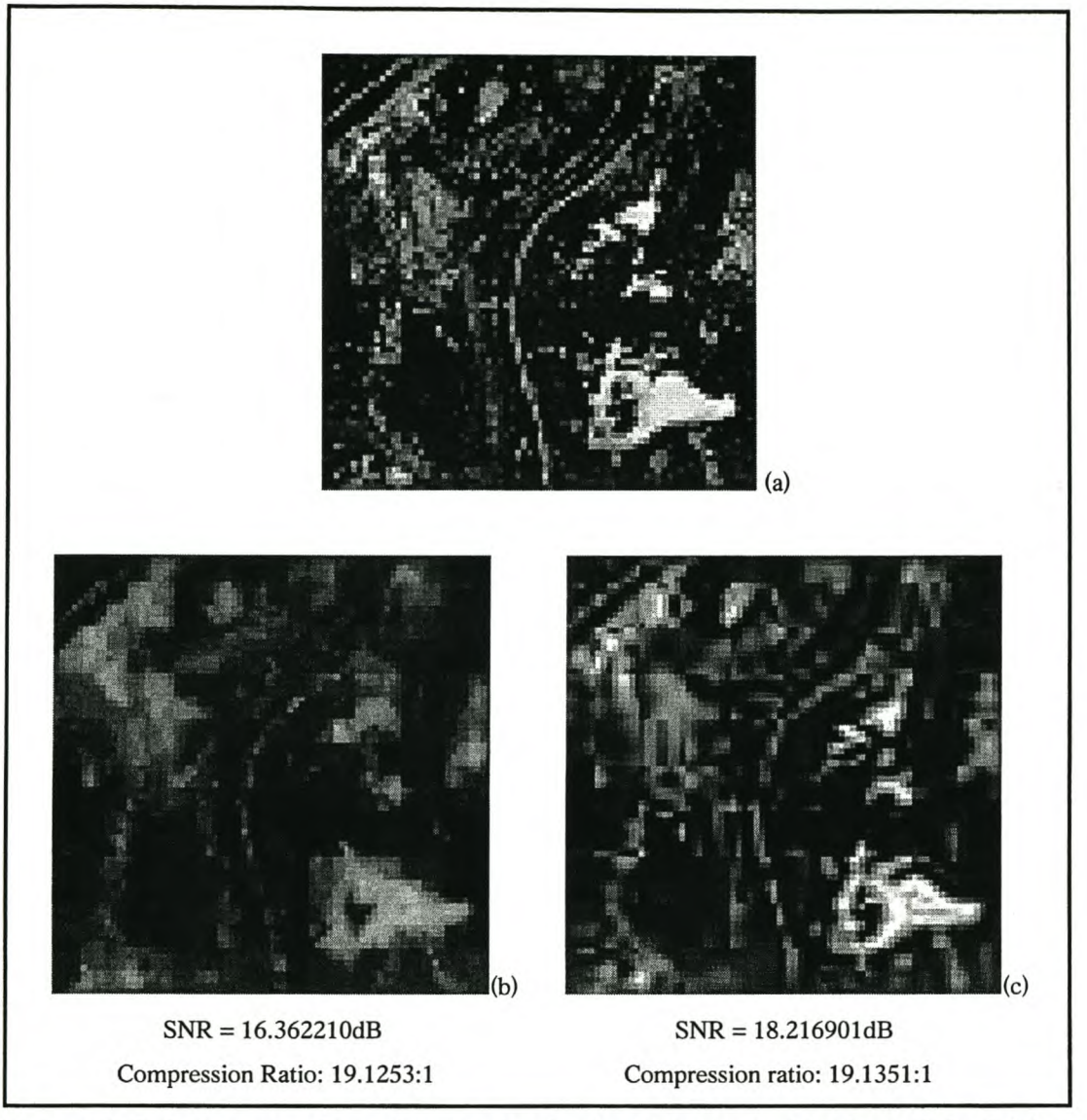


Figure 3-3 A 3x enlarged portion of the 'valley1' image with the original (a), fractal (b) and JPEG (c)

3.1.2 Quality Evaluation

In the following sections on different image quality measures, the 60 test images that were generated during the implementation of the JPEG (Section 0) and fractal image compression (Section 0) sections, will be used as sample data for the calculation of quality for each measure. Since the JPEG images were compressed as closely as possible to the compression ratios achieved by the fractal compressed images, the resulting quality values should reveal interesting information regarding the quality of compression of these two methods compared against each other.

Section 0 describes a subjective test that was designed and used to obtain quality information on these test images. This data can be used as a measure of the accuracy with which the measures listed in this section can be described.

Section 0 shows the results of each of the techniques described in the next section applied to the test images and compares these results with the results derived from human evaluation.

3.1.3 Signal to Noise Ratio

The term signal to noise ratio (SNR) is self explanatory in that it gives the ratio of a signal to the noise that is superimposed on that signal. This figure is normally expressed as the ratio of the signal powers and is expressed in decibels (dB).

Generally in the field of signal processing, this calculation is performed for non-deterministic signals and noise and can be a very complex problem. Fortunately for this application, both the signal and noise are available for calculation. There are many variations on the expression defining the signal to noise ratio ([Jain], [JacqD] and [FisherB]) for digital images but most of these usually only differ by some constant factor.

The average least squares image error between two images *u(m, n)* and *u´(m, n)* with size MxN pixels, is given as:

$$\sigma_{ls}^2 = \frac{1}{MN}\sum_{n=1}^{N}\sum_{m=1}^{M}\left|u(m,n)-u'(m,n)\right|^2 \tag{3.32}$$

The signal to noise ratio is then expressed as (Section 3.6 of [Jain]):

$$SNR = 10\log\left(\frac{\sigma^2}{\sigma_{ls}^2}\right) \tag{3.33}$$

with σ^2 the variance of the reference image. The variance is sometimes replaced with the dynamic range (peak to peak value) of the pixel values of the reference image which yields higher results. Another variation is to use the maximum possible pixel variation of the reference image. For an 8 bit grey scale image the value 256 will therefore be used.

In this work, the dynamic range of the reference image will be used for all SNR calculations since for certain cases, using the variance of the reference image can result in negative values for SNR. The dynamic range of the image is taken as the difference between the values of the lightest and darkest pixels. Source code for

the implementation of the calculation of the SNR between two images is supplied on the accompanying CD-ROM.

$$SNR = 10 \log\left(\frac{dr(u)^2}{\sigma_{ls}^2}\right) \tag{3.34}$$

3.1.4 Grey Block Distance

The grey block distance (GBD) as a distance measure was proposed by Juffs, Beggs and Deravi in [Juffs]. The idea of this measure is to approximate visual quality as perceived by the human visual system. Since the aim of this work is to find such a measure that performs better than current methodologies, it was considered worthwhile to examine the GBD in some detail. The theory will be briefly discussed, and the implementation will then be applied to the test data set generated in the previous chapters. Section 0 will compare the results of this experiment with the other measures examined in this chapter to find some idea of its relative performance.

3.1.4.1 Theoretical Description

The grey block distance between 2 digital images I and I' is defined as:

$$G(I,I') = \sum_{r=1}^{N} d_r \tag{3.35}$$

where N is determined by the resolution of the image and d_r is given by:

$$d_r = \frac{1}{2^r}\frac{1}{2^{2r-2}}\sum_{j=1}^{2^{r-1}}\sum_{i=1}^{2^{r-1}}\left|g_{ij} - g'_{ij}\right| \tag{3.36}$$

The value g_{ij} is calculated at each resolution from 1 to N as follows:

- Consider an image I at resolution r = 1. g is defined as the average pixel value (gray level value) over the entire image.
- At resolution r = 2, the image is quadtree partioned into four segments as shown in Figure 3-4 and the average grey level (g_{ij}) is calculated for each segment.

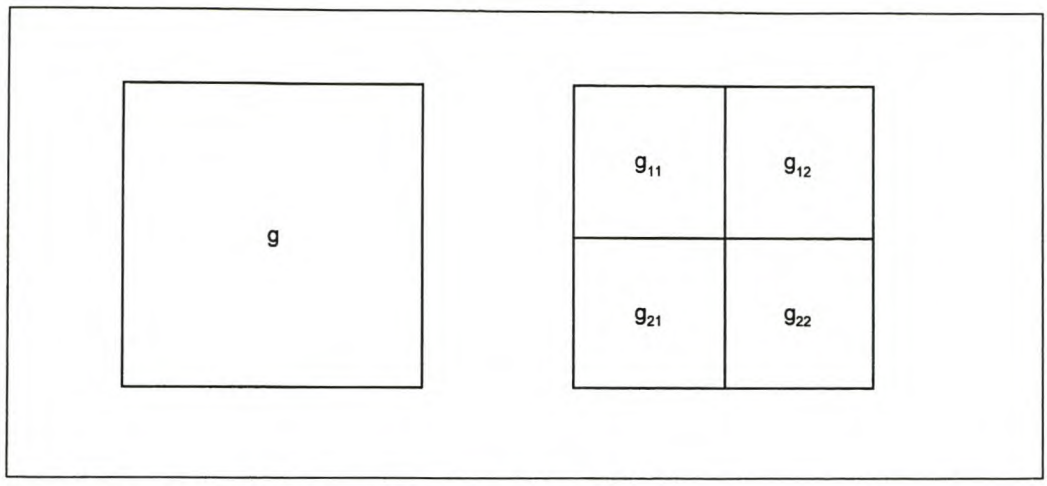


Figure 3-4 Image segments at different resolutions

Section II of [Juffs] continues to give a more formal definition of this measure and then goes on to prove that the GBD is a metric in the space of digital images.

The rest of the article is dedicated to comparing the GBD with the Hutchinson metric [BarnB, p355] and finally shows that the GBD performs as least as good at separating images as the Hutchinson metric. The GBD is however much less computationally intensive.

3.1.4.2 Application

A program that calculates the GBD between two digital images was implemented In C++ and applied to the 60 test images (The source code for this implementation is available on the accompanying CD-ROM).

The Grey Block Distance is subtly different from the other image quality measures in that it calculates the difference between images as opposed to the closeness between images. This means that the other quality measures will have high values for images that are closer to each other with an infinite value for identical images. The GBD on the other hand, will have a zero result for identical images and this value should increase with increasing dissimilarities between images. Some sort of transformation needs to be performed in order to be able to compare the results of these techniques with each other, but this will be discussed in the next section.

The algorithm was implemented to calculate the normalised quality values as described in the article. The source code and executable file for this program can be found on the accompanying CD ROM.

3.1.5 Gradient Method

Consider the images shown in Figure 3-5. The image on the left is a section taken from a satellite photograph and the right hand image is the same section but degraded by fractal compression. In comparison, the obvious differences between these two images seem to be in regions where the image changes quickly, for example at borders of uniform areas. These are the regions with a high spatial frequency or high gradient.

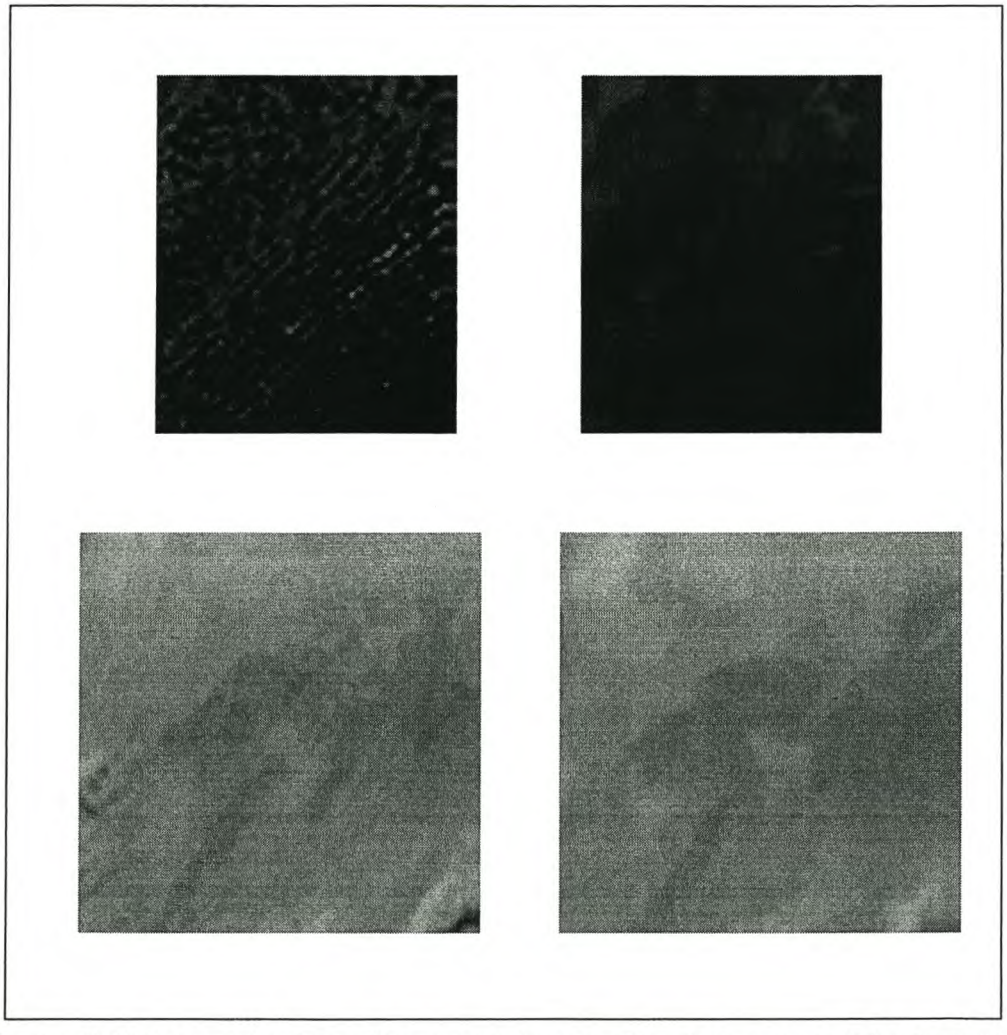


Figure 3-5 Segments from the same image showing areas of high and low gradients.

Since differences between images seem more visible in these regions, it seems logical to try and exploit this in calculating the quality value of a degraded image. There are many ways in which this can be applied and this section examines some of the possibilities. The two methods discussed below were named purely for convenience and to be descriptive.

3.1.5.1 Gradient Weighted SNR

As a first approach to try and exploit image gradient to measure quality, one could simply calculate the difference between the original and degraded images as a weighted sum where the regions of high gradient carry more weight. This weighted error image can then be used in the standard SNR formula given by equation 3.34. The first step in this process is to decide how the regions of high spatial frequency or gradient will be identified and calculated and then how the weighting scheme must be applied.

In section 9.4 of [Jain], a set of gradient operators called the compass operators, are described. The compass operators consist of 8 spatial masks representing the 8 compass directions (refer to Table 9). The gradient of the image at any point is then the maximum of the results of the 8 masks convoluted with the image at that point. The direction of the gradient is the direction of the corresponding maximum value mask. This definition of the gradient operator for discrete data is a simplification of the definition for the continuous case as given in section 8.8 of [Kreysig].

There is however a problem in calculating the gradient at the borders of the image since the outside of the image will result in a discontinuity. The easiest solution at this stage will be to calculate the gradient of the image only for the regions where the compass masks are completely surrounded by the image itself. This means that the outer layer of pixels will be ignored which will not be a problem as long as this is done consistently.

Another approach would be to use the rate of change of the gradient as the image border is neared to predict a value for the border gradient and use this value for the calculation of the weight of the error.

The final step would be to actually calculate the relative weights for pixel errors corresponding to the gradient of the original image. From the definition used for the gradient, it can be seen that the actual gradient value can range from 0 to three times the maximum pixel value. The simplest form of weighting to apply here is to use the normalized gradient value as the actual weight for the error value.

The image quality value calculated from this method will be slightly less than the values obtained from equation 3.34 since the outer pixel frame is ignored and the SNR equation effectively uses weighted values of 1 for every pixel

whereas this method uses weights that are mostly less than 1. This should not pose any problem if the method is applied consistently. For comparison with other methods, correlations should be used instead of the actual calculated quality values.

To summarize:

1. The average least squares gradient weighted image error (σ^2_{gwls}) is given by:

$$\sigma^2_{gwls} = \frac{1}{(M-2)(N-2)} \sum_{n=2}^{N-1} \sum_{m=2}^{M-1} \alpha(m,n) |u(m,n) - u'(m,n)|^2 \tag{3.37}$$

where $\alpha(m,n)$ is the gradient calculated weight given by

$$\alpha(m,n) = \frac{1}{G} g(m,n) \tag{3.38}$$

with G the maximum possible gradient value for an image (3 times the maximum possible pixel value)

and $g(m,n)$ the gradient of the image at location *(m, n)* given by

$$g(m,n) = \max_k \{|g_k(m,n)|\} \tag{3.39}$$

$g_k(m,n)$ is the result of one of the 8 compass masks applied to the pixel at location *(m,n).*

The definition of the 8 compass masks are shown in Table 9.

The gradient weighted signal to noise ratio will then be given by:

$$SNR_{gwls} = 10 \log \left(\frac{dr(u)^2}{\sigma^2_{gwls}} \right) \tag{3.40}$$

North			North West			West			South West		
1	1	1	1	1	0	1	0	-1	0	-1	-1
0	0	0	1	0	-1	1	0	-1	1	0	-1
-1	-1	-1	0	-1	-1	1	0	-1	1	1	0

South			South East			East			North East		
-1	-1	-1	-1	-1	0	-1	0	1	0	1	1
0	0	0	-1	0	1	-1	0	1	-1	0	1
1	1	1	0	1	1	-1	0	1	-1	-1	0

Table 9 Compass gradient masks

3.1.5.2 Gradient Error

As an alternative to the previously discussed method, the gradient of the error image can be calculated and this can be used to obtain the signal to noise ratio using equation 3.34.

The gradient error signal to noise ratio is given as:

$$SNR_{ge} = 10 \log\left(\frac{dr(g)^2}{\sigma_{gels}^2}\right) \tag{3.41}$$

where the average least squares image gradient error is given by:

$$\sigma_{gels}^2 = \frac{1}{(M-2)(N-2)} \sum_{n=2}^{N-1} \sum_{m=2}^{M-1} |g(m,n)|^2 \tag{3.42}$$

with *g(m,n)* the gradient of the image produced by the absolute difference between the test and original images The gradient is the same as defined in equation 3.39 and *dr(g)* is the dynamic range of this error image.

There are many other variations on the two methods explained above, but these are the ones that will be used for comparison against other methods. Section 4.1.1 will consider some approaches that could possibly lead to improvements on these algorithms. The results of this comparison will be discussed in section 0.

Subjective image quality analysis

3.1.6 Image comparison test

The previous section describes some methods for calculating image compression quality. In this section, a method for obtaining some sort of measure of image quality based on human interpretation is proposed and implemented. Section 0 compares this method with the results of the methods discussed in the previous section.

To try to get a good average of human image quality estimation, a computerised test was put together. Research (section 3.6 of [Jain]) shows that the human eye can discern between 5 and 7 levels of image quality with reasonable certainty. The evaluation program presents a quarter section of each original image against the equivalent section of either the JPEG or fractal compressed image. The test subject is then prompted to select a quality value for the degraded image. The

choice of quality values varies from 1 to 5 where 1 is the lowest possible quality and 5 is the least degradation.

The test subject uses the first few images to get an idea of the general spread of image quality. This is done to try to force a maximum variance of values in the five point quality range since this range is already very limited. Furthermore, the test is also set up to prevent participants from reviewing previous choices before making a new choice to let every participant adjust to the image values in a natural way without being influenced by previous choices.

The test images were presented in quartered sections of the original images. The images were cropped into four quarter sections for two reasons. Firstly, it is desirable to view both the original and test images alongside each other for best comparison and since these images are 512 pixels wide, this would only be possible on computers with video cards that are able to handle screen resolutions above at least 1024 pixels. This would narrow down the field of possible test stations considerably. The second reason for separating images is that it provides a convenient mechanism for judging the consistency of each participant since each test image is presented four times during the duration of the test.

The actual test was presented as a web based program. This presented a problem since most Internet browsers automatically dither images according to what they believe the machine can display properly. Unfortunately, the browser performs this task without any warning and different browsers can also use different dithering techniques. Any of these techniques applied to an image will invalidate the results since dithering introduces different artefacts to the image. To try and avoid this problem, the image comparison test starts off with a range of 16 and 256 shade grey scale images created by different dithering methods. If any of these images cannot be discerned, it means that the test cannot be completed at the current display settings.

Each participant of the test was allocated a unique session id and the results of each completed test was stored in text format on the web server machine with this identifying number.

All relevant source code and image data used for this test is available on the accompanying CD-ROM.

3.1.7 Interpretation of test results

Once enough tests have been completed to start calculating averages, the data must be sorted in some meaningful way to provide an average that describes image quality on this five point scale.

The subjects were volunteers that visited the test website on the request of the author and completed the test. Due to the long time required to complete the test and the monotonous nature of the test, many subjects did not complete the test. The incomplete results were discarded. Server problems also caused many browsers to freeze during the test and these results were also discarded. Due to these reasons, only 27 sets of completed test data was collected. The test data was then analysed using a C++ program (the source code can be found on the accompanying CD ROM) that was written for this purpose.

The analysis program parses each test separately and tests the candidate for consistency. The program counts the number of times that a candidate has a certain deviation in his/her grading of the four quarters of the same image. The deviations are calculated from a zero error to an error of four quality points for the same image.

The test program also calculates the average quality value for each of the 60 test images per test subject, and also the total average of all of the tests. The average per image of each subject is then cross-correlated with the average of the total to measure the similarity of each candidate to the average. The correlation is also calculated without the first 40 test image segments to try and determine whether there is some sort of learning curve whereby the test candidate acclimatises to the general spread of image quality.

Lastly, the analysis program also calculates the average test score and standard deviation of each candidate for all the images together.

An example of the output of this program for two test results is shown below:

```
Test case 10: 1575520122.tst
Error Margin : Deviation
 0              41
 1              13
```

2 0

3 0

4 0

Correlation coefficient with average: 0.755634

Correlation coefficient without first 40 samples: 0.756720

Average: 4.345833

Standard Deviation: 0.617609

Test case 17: *333774550.tst*

Error Margin : Deviation

0 56

1 18

2 3

3 1

4 0

Correlation coefficient with average: 0.740336

Correlation coefficient without first 40 samples: 0.755597

Average: 2.654167

Standard Deviation: 0.878080

The two samples clearly show two rather diverse interpretations on the quality of the test images. The first case shows an average of 4.35 with a small deviation of 0.62 around this value. The second candidate on the other hand graded the images with a much better average and a larger spread: The absolute average and deviation values are not really important. The purpose of the test is to find some sort of value set that can be used to check correlation with other measures. It is however, up to the candidate to find some meaningful average and distribution of the image quality values from the already very limited range of five discrete values.

After these values have been calculated for all the test cases, the results were studied. It was found that ignoring the first 40 test images did not improve the correlation values with the average test scores. The results of this and the 2 subsequent analyses are included on the accompanying CD-ROM together with the source code for the analysis program. To try to get a better test sequence, the five test subjects with the lowest correlation coefficient were removed from the set of test data. A new average sequence was calculated from the remaining 22 test samples and the analysis was repeated. After this, the average correlation coefficient increased by 0.0228 from 0.8063 to 0.8291. Another five test samples were removed and the test was repeated again, resulting in a further increase in average correlation coefficient of 0.0172 from 0.8291 to 0.8463. It was decided to stop trying to refine the final set of subjective image quality values for fear of 'over training'. The smaller the number of tests used to calculate an average, the higher the correlation of each individual test with the average. The set of 17 tests left after the two refinements will be considered sufficient for a good average quality value. Table 10 shows the final average image quality values for the test images that will be used for the grading of the quality measures discussed earlier.

Appendix B gives a brief discussion on the use of the cross correlation calculation as a means for this comparison and its validity in this capacity.

	JPEG	Fractal		JPEG	Fractal
1	3.21	3.19	16	4.79	4.71
2	2.76	2.87	17	4.66	4.69
3	3.09	3.07	18	4.81	4.68
4	3.06	3.15	19	4.82	4.84
5	4.41	4.54	20	4.88	4.82
6	4.46	4.46	21	4.87	4.90
7	4.76	4.57	22	4.81	4.85
8	3.01	2.91	23	2.63	2.50
9	3.04	2.90	24	2.13	2.26
10	3.28	3.26	25	1.99	1.99
11	3.65	3.69	26	2.79	2.90
12	4.59	4.66	27	2.97	2.96
13	4.65	4.51	28	2.60	2.51
14	4.68	4.78	29	2.35	2.32
15	4.63	4.71	30	3.87	3.88

Table 10: Final average image quality values

The next section will examine the quality values as calculated by the methods discussed earlier. These values will then be correlated with those determined in Table 10 to see how these measures perform against each other. The definition of the cross correlation coefficient shows that the result is linearly independent for two series. This means that the different techniques can be compared directly without having to do any normalisation.

Test results

This section begins by listing the image quality values as calculated by the different techniques discussed in section 0 as shown in Table 11 and Table 12.

As was mentioned in section 3.1.4, the calculation of the Grey Block Distance is fundamentally different from that of the other measures.

The subjective measure ranges from 1 to 5 with 5 being the highest possible image quality value.

The Signal to Noise Ratio, Gradient Weighted SNR and Gradient Error SNR measures cannot be put in such easy categories. The SNR for very similar images

tend towards infinity for identical images, and the SNR for very dissimilar images can reach negative values. It is very difficult to find the lower bound on these measures. The worst possible SNR would be two images with pixel values as far as possible from each other. An easy example of this would be to calculate the SNR between a completely white images (pixel values = 255 in the test images) and completely black images (pixel values = 0). This method unfortunately leads to a problem since the equation for SNR also makes use of the variance or dynamic range of the original image and for a completely uniform image this value is zero. Substituting the relevant equations leads to the calculation of the log of zero, which of course does not provide any solution.

An obvious solution would be to simulate the worst possible SNR by using images with alternating black and white pixels throughout the image such that the two images always have oppositely coloured pixels in the same positions. Substituting this into equation 3.34 results in a signal to noise ratio of –6.055 dB. This value can be decreased even further by decreasing the dynamic range of the reference image. This in turn will decrease the calculated error between the two image, but with much smaller influence to the final solution. It does not however fall in the scope of this work to try and optimise 2 images for the worst possible signal to noise ratio, but only to get an idea of the range of possible values. It is also important to note the final range of results of any SNR, GWSNR or GESNR calculations is very much dependent on the size and depth of the images involved.

This leaves the Grey Block Distance whose results fall in the range of 0 to 1 (for discrete digital images where pixel values x lie in the range $x \in [0..255]$) with 0 being the quality values for identical images. This is completely opposite from the other measures where higher values imply better quality. To be able to compare these values to the other measures, the GBD must be inverted. The easiest way to accomplish this, is to simply invert the calculated values (multiply by –1) and then translate by 1 to keep the values in the 0 to 1 range. This new value will be referred to as the Grey Block Distance in the rest of this section since it is essentially the same.

	Image	Signal To Noise Ratio	Grey Block Distance	Gradient Weighted SNR	Gradient Error	Subjective Image grading test
1.	Bangladesh	18.547	0.985	19.753	15.848	3.191
2.	Clouds	18.865	0.998	20.670	16.170	2.868
3	clouds2	20.072	0.997	21.277	16.308	3.074
4.	clouds3	25.277	0.999	28.195	19.627	3.147
5.	Frozen	35.712	0.999	43.605	18.951	4.544
6.	frozen2	34.494	0.998	42.866	17.623	4.456
7.	frozen3	35.248	0.998	42.835	21.097	4.574
8.	goes1	22.719	0.998	24.752	17.944	2.912
9.	goes2	19.817	0.997	20.584	15.829	2.897
10.	goes3	22.212	0.997	24.763	17.603	3.265
11.	goes4	24.364	0.998	25.768	19.233	3.691
12.	hurricane1	38.395	0.999	45.368	23.225	4.662
13.	hurricane2	38.698	0.999	51.010	13.975	4.515
14.	hurricane3	36.463	0.999	47.683	17.320	4.779
15.	isle1	29.302	0.999	33.749	17.345	4.706
16.	isle2	33.977	1.000	40.056	19.145	4.706
17.	night1	27.239	0.999	34.692	13.879	4.691
18.	night2	29.601	0.999	37.122	14.407	4.676
19.	night3	28.808	0.998	36.472	14.823	4.838
20.	ray1	41.324	0.999	53.083	18.648	4.824
21.	ray2	41.455	0.999	53.038	16.529	4.897
22.	ray3	41.028	0.999	51.937	16.348	4.853
23.	san francisco	22.980	0.997	25.522	17.324	2.500

24.	seattle	21.602	0.998	23.521	16.515	2.265
25.	spain	19.679	0.990	22.902	14.677	1.985
26.	typhoon1	22.647	0.999	24.327	17.885	2.897
27.	typhoon2	20.852	0.997	22.378	15.620	2.956
28.	valley1	19.517	0.997	21.277	16.035	2.515
29.	valley2	19.605	0.996	21.490	15.417	2.324
30.	venus crater	34.235	0.998	41.799	21.735	3.882

Table 11 Fractal Image results

	Image	Signal To Noise Ratio	Grey Block Distance	Gradient Weighted SNR	Gradient Error	Subjective Image grading test
1.	Bangladesh	19.249	0.999	20.983	14.211	3.206
2.	Clouds	19.902	0.999	21.984	13.452	2.765
3	Clouds2	21.499	0.999	23.163	15.192	3.088
4.	Clouds3	26.873	1.000	30.552	16.150	3.059
5.	Frozen	37.029	1.000	46.455	14.640	4.412
6.	frozen2	35.803	1.000	45.421	12.932	4.456
7.	frozen3	36.705	1.000	46.016	13.557	4.765
8.	goes1	24.525	0.999	27.226	16.058	3.015
9.	goes2	21.220	0.998	22.396	16.214	3.044
10.	goes3	23.907	0.999	26.992	15.383	3.279
11.	goes4	26.790	0.999	29.170	17.843	3.647
12.	hurricane1	40.022	1.000	50.117	13.583	4.588
13.	hurricane2	39.133	1.000	51.548	11.521	4.647
14.	hurricane3	37.068	1.000	48.662	11.911	4.676
15.	isle1	31.954	0.999	36.950	17.234	4.632
16.	isle2	35.835	1.000	42.842	14.607	4.794
17.	night1	27.579	1.000	35.219	13.392	4.662
18.	night2	30.061	1.000	37.931	14.844	4.809
19.	night3	29.681	1.000	37.993	12.953	4.824
20.	ray1	41.667	1.000	54.461	12.251	4.882
21.	ray2	41.921	1.000	54.449	12.032	4.868
22.	ray3	41.752	1.000	53.798	12.693	4.809
23.	san francisco	24.898	0.999	28.017	15.352	2.632
24.	seattle	23.189	1.000	25.785	14.851	2.132

25.	spain	20.291	0.999	23.688	13.438	1.985
26.	typhoon1	24.540	1.000	27.057	18.209	2.794
27.	typhoon2	22.415	0.997	24.494	14.818	2.971
28.	valley1	20.663	0.999	22.769	14.086	2.603
29.	valley2	20.759	0.998	23.000	14.051	2.353
30.	venus crater	35.999	1.000	45.116	16.139	3.868

Table 12 JPEG Image results

Table 13 shows the cross correlation coefficients of the quantitative image quality measures with the subjective quality values represented by the three series of subjective test results A, B and C.

The three series used as control series for these correlation calculations are calculated as follows:

A: The average image quality values of all 27 test sets.

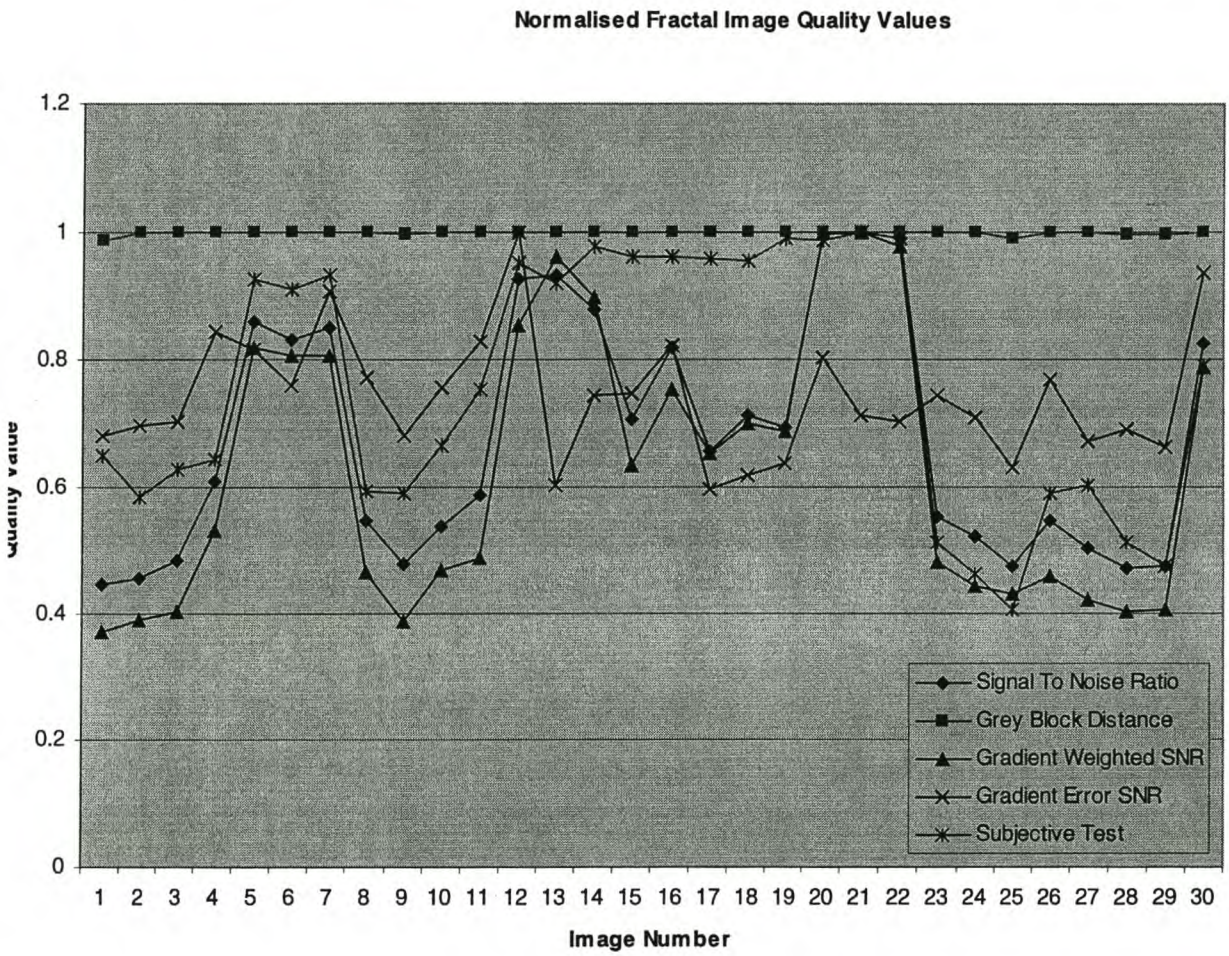
B: The average image quality values with 5 samples that correlate worst with the average series removed.

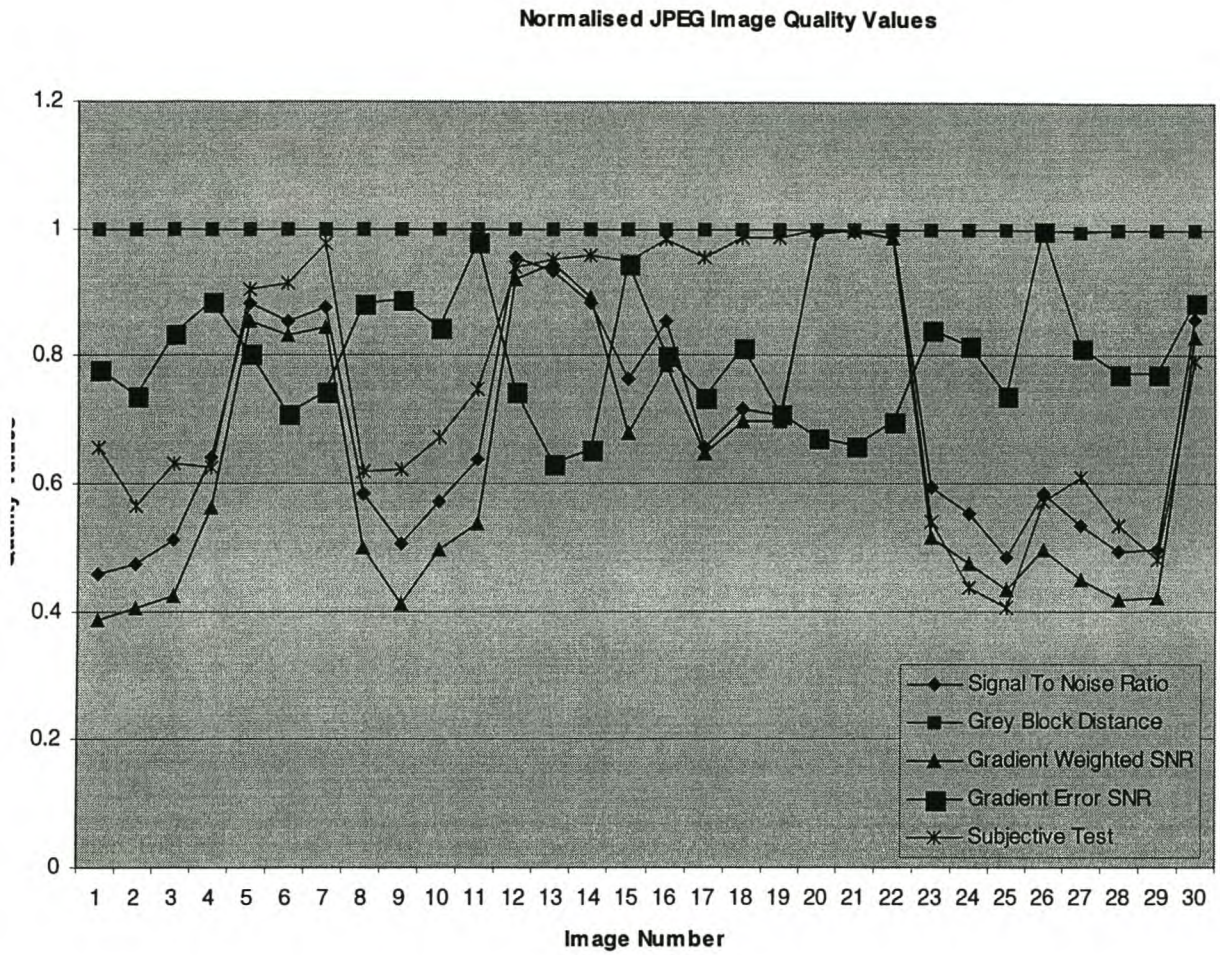
C: The average image quality values of the remaining 17 test sets with 5 tests removed that correlated wost with the average of the remainder series B.

The two graphs below shows the image quality values (for the JPEG and Fractal image sets) on a normalised scale to illustrate the correlations listed in Table 13. The graphs are drawn as line graphs although the data is discrete. This was done purely so that the different data sets could be seperated more easily. The correlations in Table 13 are the correlations for all sixty test images represented as a single discrete time series.

	A	B	C
Signal To Noise Ratio	0.866403	0.866374	0.858422
Grey Block Distance	0.378475	0.380775	0.385403
Gradient Weighted SNR	0.869851	0.870472	0.863873
Gradient Error	-0.041678	-0.046209	-0.053598

Table 13 Quality measure cross correlation coefficients for all the test images





4. Discussion and interpretation of results

Image Quality Measurement

The results in section 0 show a widely varying range of image quality measures and their correlation with subjective results. The two measures that clearly stand above the rest are the Signal to Noise Ratio and Gradient Weighted SNR (GWSNR).

This section will attempt to explain the reasons for the good performance of the GWSNR and SNR and explore some possible improvements on the GWSNR. The rest of the chapter will investigate reasons for the poor performance of the Grey Block Distance and Gradient Error SNR and will finish off with a closer scrutiny of the reliability of the subjective image quality test as a means of comparison.

4.1.1 Gradient Weighted Signal To Noise Ratio

The results achieved with the application of the Gradient Weighted SNR confirm that the method is based on sound assumptions. As a first implementation, the GWSNR already performed slightly better than the traditional SNR. This leaves the way open for further enhancements to this algorithm.

The most obvious attempt at improvement to this algorithm is the weighting factors assigned to the gradient values. The weight assigned in the current algorithm is a normalised linear factor of the size of the gradient between adjacent pixels. If some value could be found to quantify the sensitivity of the human eye to different gradient levels, this information could be used for a superior grading technique. This information could be found by doing some research using fabricated samples in much the same way as the original quality analysis test was presented.

An alternative to using a weighted function based on a priori knowledge of the human visual system would be to try some non linear function that assigns a larger weight to higher gradients. There are many functions that could be used

towards this end and only the experimental results would show which are most suitable.

Another approach would be to utilise the rate of change of the derivative. The second derivative of the gradient could be used to estimate the gradient at the borders of the image so that these pixels can also be used in the quality calculations.

The second derivative can also be used to get a better judgement of the nature of the image. As it was originally assumed that the human eye is more sensitive to regions of high gradient, it also seems logical that a region where the rate of change of the gradient is higher than other regions would also be more visible to the eye. Such regions should therefore weigh more heavily in the calculation of image degradation.

The above argument could be taken further. If the second derivative of an image would help in finding regions where change is more rapid, and thus more visible to the human eye , the higher order derivatives would by the same reasoning provide even more information regarding the nature of the image. Unfortunately, the effects of these higher order derivatives on the human visual system have not been researched or investigated in the course of this work and any speculation in this regard would be groundless. It would be relatively simple to put together a subjective test to measure the sensitivity of the human visual system towards these higher order image components.

Since no work has been done towards exploring any of the possibilities mentioned above, it is difficult to estimate the effect that any of these methods could have on the GWSNR measure.

It is impossible to judge the level of influence of a non linear weighting function on the GWSNR. If the human eye has a sensitivity to gradient error that is very different from the linear scale used in this work, the difference that such a variant could introduce to the final quality calculation could be substantial. This line of thought definitely warrants further investigation.

Using the second derivative (or any other technique) to estimate the values of the image gradient at the border pixels, will have a minimal effect on the final GWSNR. This is true because the additional gradient values in the summation of equation 3.37 will be cancelled by the division of the increased size of the gradient image. Even if these estimations are inaccurate, the additional number of points used for this calculation will only be a small percentage of the total number of

points and will therefore have a negligible effect (for larger images) on the averaged total given by equation 3.37.

The best solution to improving this measure would be to investigate all of the possible improvements discussed above and implementing them in some sort of a hybrid scheme.

4.1.2 Signal To Noise Ratio

The GWSNR and SNR are mathematically similar with the GWSNR having a additional weighing factor that is a function of the gradient of the image. This explains the high cross correlation coefficients between the two measures (Table 13). This high correlation also shows that the Signal to Noise ratio should definitely not be disregarded as a measure of image quality. Apart from the good correlation that this measure has with subjective test results, it is also computationally less intensive that the GWSNR. This can be seen from the similarity between the calculation two measures with the addition of the gradient based weighing factor in the GWSNR (compare equations 3.32 and 3.37).

4.1.3 Grey Block Distance

The results for the GBD method are not very good compared to the subjective test results. The variance of the calculated image quality values is very small compared to that of the other methods as can be seen from the graphs comparing the image quality values.

To try and find the reason for this poor performance, it would be prudent to first confirm the validity of the implementation of this algorithm.

Towards this end, the algorithm was reapplied with the 30 original images, each compared to all the others. The table in appendix A shows the results of this test. The inversion and scaling used in the previous section to adjust these values for comparative reasons have been removed for simplification. As should be expected, all the values with the same row and column number have a zero distance value since this is the GBD of an image with itself. The actual values are all small considering that the values were calculated between completely different images.

The next step was to apply the algorithm to 2 opposite images to try to find the extent of GBD values for dissimilar images. The algorithm was applied to a black

and a white image and also to the first test image and its inverse. These results are shown in Table 14.

Test	Grey Block Distance
Black image vs white image	0.993958
Test Image vs inverse image	0.076367

Table 14 Grey Block Distance for dissimilar images

Many conflicting conclusions can be drawn from the results of Table 14. The black and white images show a large distance value that seems correct since these images are complete opposites. The inverse of the Bangladesh river image shows a much lower distance value with its original. It can be argued that these images although opposites are visually much closer since they can still be recognised as the same image. Unfortunately, by that same argument, the only difference between the black and white images is a scaling or inversion and they are also essentially the same. They are however much more visually dissimilar than the positive/negative image pair.

Since these GBD values range between 0 and 1, it seems that they would have a better chance at good correlation with the subjective test results since these are also constrained to a fixed range (as opposed to the other measures which have a much wider range – refer to equation 3.32 where it can be seen that the SNR for identical images tend to infinity), but this is not the case.

Despite all these conflicting observations, the grey block distance does seem to have merit as a measure of visual image similarity, but this technique might not be best suited for the application used in this work. Before this technique can be applied to its best advantage, some rules must first be established as to what is a good and bad match between images. As can be seen from the figures in Table 14, two images although exact opposites are still interpreted as very similar using this measure. This subject also warrants further investigation.

4.1.4 Gradient Error Signal To Noise Ratio

The GESNR was the most uncorrelated measure of all those investigated when compared to the subjective tests.

The idea upon which this method was built still seems to make sense, but unfortunately the calculated results do not concur. At this time, no reason for this behaviour can be found and this result definitely warrants further investigation.

A first approach in this regard would be to confirm that the correctness of the implementation of this technique.

5. Conclusion and recommendations

This chapter will begin by looking at the relative performance of the two compression techniques discussed in chapter 2. In the previous chapter, different measures are all compared to a subjective measure that was obtained through experimental methods. The 'accuracy' of these measures were calculated through a cross correlation with the results obtained from the subjective tests. In this chapter, the validity of the subjective test as a measure for the quality of other measures will be investigated.

Fractal vs JPEG compression

Since the idea of finding a new image quality measure resulted from a need to measure the comparative performances of the 2 compression techniques against each other, it would be worthwhile to use this new found measure to compare the two techniques.

Table 15 shows the average scores for image quality measurements using the 4 discussed methods and also includes the average compression ratio for the 2 sets of images.

	Fractal Images	JPEG Images
GBD	0.998	0.999
GESNR	17.236	14.453
SNR	28.158	29.431
GWSNR	33.416	35.475
Subjective test	3.736	3.742
Compression Ratio	20.029	20.125

Table 15 Average quality values for the JPEG and Fractal image sets

As can be seen from the above table, the JPEG method performed consistently better on average quality measurement with all but the GESNR measure. This exception is also the measure with the lowest cross correlation coefficient to the

subjective test results and can therefore be disregarded as a measure for image quality. The table also shows that the JPEG images were compressed to a slightly higher compression ratio although this difference is so small as to be negligible.

The above results do not however imply that JPEG compression is superior to fractal techniques. The JPEG algorithm employed for this work was obtained from source code that is available in the public domain. This code has been improved by several users over a period of a few years. The JPEG algorithm also includes additional image enhancements such as low pass filtering around image segment borders after decompression to reduce the block like effect that is found in JPEG compression. This will obviously further improve the subjective image quality and can be applied to the fractal algorithm with the same gains since fractal compression also introduces a edge effect on image segment borders. The fractal compression algorithm that was used on the other hand is based on the original work presented by Jacquin in [JacqD]. Given enough time, this algorithm could be improved upon considerably as is obvious from the host of other articles that appeared on this topic after the publication of Jacquin's first work. The accompanying CD-ROM contains a list of sources for further reading on this topic. The articles themselves can be obtained from several internet sites (the addresses are listed on the CD-ROM), but copyright restrictions prevent them from being reproduced on the CD-ROM.

Subjective Image Quality Evaluation

The subjective image quality estimation test was constructed to find a way to evaluate the performance of different image quality measurement techniques. Since the processed results from this test was used as the absolute gauge of the quality of all the other discussed measures, these results must be obtained with a high degree of certainty.

The results were filtered to remove all those subjects that correlated badly with the average and a new average was calculated with the remainder. This process was performed twice but cannot be performed too many times since this will lead to specialisation where very few subjects influence the eventual average.

Another technique employed to try and filter out inconsistent test subjects, was to partition all image into four separate quarters and present each of these as a separate image at different stages of the test. Unless the content of a specific image is divided so that a single quarter might have a lot of high spatial frequency components whilst another quarter might contain regions of low spatial frequency, it

can be assumed that the sub images will have the same quality factor on average. Using this assumption, candidates with too high an average quality value spread per image can be disregarded in calculating the final values. It was found however that some candidates that occasionally have a very high spread per image can also still have a high correlation with the average, and candidates should not be disregarded on their consistency alone but on a combination of consistency and correlation with the average.

As was already mentioned in section 3.1.7, regarding a certain amount of the first images as training data and ignoring these in the calculations did not seem to produce any significant improvement on either candidate consistency or correlation with the average series. The fact that the test was also presented in such a way as to prevent a candidate from viewing their previous selections, and thus 'learning' how to grade the images, reinforces the finding that there seems to be no general trend in the statistical nature of the series of test values as they were entered.

The amount of raw data used for these results was also not enough to try and use the final set of quality values as a standard for measurement. It would seem prudent to have the final results be the average of as many as possible separate test subjects to avoid any specialisation in the final result. It would also be good practice to have more than one set of test data using different types of images and a wider range of distortion types to put together some sort of baseline for image grading measures to be compared against. Some completely independent tests would also serve as a confirmation or not, of the validity of this type of test. Chapter 3 of [Jain] describes some image quality measures and discusses subjective tests as well.

Appendix A

Image	Bangladesh	Clouds	Clouds2	Clouds3	Frozen	frozen2	frozen3
Bangladesh	0	0.191921	0.107282	0.153261	0.11699	0.173829	0.157286
Clouds		0	0.20947	0.094654	0.089155	0.038734	0.062009
Clouds2			0	0.121299	0.153472	0.189513	0.157928
Clouds3				0	0.046605	0.069788	0.038583
Frozen					0	0.05819	0.041502
Frozen2						0	0.043261
Frozen3							0

Table 16 Grey Block Distances between some of the test images to illustrate correctness of calculation

Appendix B

Cross Correlation as a means of calculating degree of comparison

In this work, the cross correlation coefficient (normalised second order central moment) is used as a means to describe how well one set of test data compares with another. This method is usually applied to noisy time sequences of signals in order to try and detect or extract the signal data from the noise.

The subjective test data in this work can be considered as a time series of 30 samples and all the measures can then be considered as noisy variants on the original.

The cross correlation coefficient is given by:

$$\rho = \left[\frac{(X - \overline{X})}{\sigma_X} \frac{(Y - \overline{Y})}{\sigma_Y} \right] \tag{5.43}$$

The properties of this function make it ideal for the type of comparison that it was used for.

- $-1 \leq \rho \leq 1$ The range of values for the correlation measure can be interpreted as 1 being a perfect match, 0 implies no correlation, and –1 implies inverse series.
- $\rho_{(X,X)} = 1$ The function correlates perfectly with itself. This is what we would expect from a function used for calculating the similarity between 2 series.
- $\rho_{(X,(aX+b))} = 1$ The cross correlation coefficient of two time series is linearly independent. This means that we can compare the control series to any other measure without any scaling or translation.

The mathematical proof of the above 3 properties is relatively simple and falls beyond the scope of this work.

The CD-ROM accompanying this work contains a section with source code for the generic implementation of the correlation coefficient and other statistical operators that were used during this project.

Finally, since the cross correlation coefficient between two series is such an effective measure for determining similarity, it stands to reason that this could be expanded to the two dimensional case and used as a measure for image quality in itself. This topic is definitely worth further investigation.

References

[Bani] Bani-Eqbal, B., Speeding up Fractal Image Compression, Proceedings of SPIE: Still Image Compression, vol. 2418, pp.67-74, 1995.

[BarnA] Barnsley, M. F., Sloan, A. D., A better way to compress images, BYTE Magazine, Jan. 1988

[BarnB] Barnsley, M. F., Fractals Everywhere, New York: Academic, 1988.

[BarnC] Barnsley, M. F., and L. P. Hurd. *Fractal Image Compression*. AK Peters, Ltd., Wellesley, Ma., December 1992.

[FisherA] Fisher, Y., Shen, T., Rogovin, D., A Comparison of Fractal Methods with DCT and Wavelets, Proc. SPIE: Neural and Stochastic Methods in Image and Signal Processing III, vol. 2304-16, 1994.

[FisherB] Fisher, Y. (ed.), Fractal Image Compression: Theory and application, Springer Verlag New York Inc., 1995.

[ftp] ftp.simtel.net:/pub/simtelnet/msdos/graphics/jpegsr6b.zip. The source code can be downloaded from the website listed here or at any Simtel mirror site:

[Fuhrt] Fuhrt, Borko, A Survey of Multimedia Compression Techniques and Standards. Part 1: JPEG Standard, Real Time Imaging, Volume 1, Number 1, April 1995.

[HuanA] Huang, T.S. (ed.), Two-Dimensional Digital Signal Processing I : Transforms and Median Filters, Springer-Verlag Berlin Heidelberg, 1981.

[HuanB] Huang, T.S. (ed.), Two-Dimensional Digital Signal Processing II : Linear Filters, Springer-Verlag Berlin Heidelberg, 1981.

[Hutch] Hutchinson, J., Fractals and self-similarity, Indiana Journal of Mathematics 30, 1981, pp. 713-747.

[JacqA] Jacquin, A. E., A Fractal Theory of Iterated Markov Operators with Applications to Digital Image Coding, PhD Thesis, Georgia Institute of Technology, August 1989.

[JacqB] Jacquin, A. E., Fractal image coding based on a theory of iterated contractive image transformations, SPIE Vol. 1360, Visual Communications and Image Processing (1990) 227-239.

[JacqC] Jacquin, A. E., A novel fractal block-coding technique for digital images, Proc. ICASSP 4 (1990) 2225-2228.

[JacqD] Jacquin, A. E., Image coding based on a fractal theory of iterated contractive image transformations, IEEE Trans. on Image Processing 1 (1992) 18-30.

[JacqE] Jacquin, A. E., Fractal image coding: A review, Proceedings of the IEEE 81, 10 (1993) 1451-1465.

[Jain] Jain, A. K., Fundamentals of Digital Image Processing, Prentice-Hall International Editions., 1989.

[Jang] Jang, J., Rajala, S., Segmentation Based Image Coding Using Fractals and the Human Visual System, Proceedings of ICASSP, vol. 4, pp. 1957-1960, 1990.

[Juffs] Juffs, P. , Beggs, E. , Deravi, F. , A Multiresolution Distance Measure for Images, IEEE Signal Processing Letters, Vol. 5, No. 6, June 1998.

[Kak] Kak, A. , Rosenfeld A, Digital Picture Processing, New York, N.Y. , Academic Press, 1982

[Kats] Katsaggelos, A. K., (Ed.), Digital Image Restoration, Springer-Verlag Berlin Heidelberg, 1991.

[Kreysig] Kreysig, E., Introductory Functional Analysis with Applications, John Wiley & Sons, New York, 1978.

[MandA] Mandelbrot, B. B., Fractals: Form, Chance, and Dimension, W.H. Freeman and Company, San Francisco, 1977.

[MandB] Mandelbrot, B. B., The Fractal Geometry of Nature, W.H. Freeman and Company, San Francisco, 1983.

[Munro] Munro, D. M., Dudbridge, F., Fractal Block Coding of Images, Electronics Letters, vol. 28, no. 11, May 1992.

[Øien] Øien, G. E., Lepsoy, S., Ramstad, T. A., An Inner Product Space Approach to Image Coding by Contractive Transformations, IEEE

[Peitg] Peitgen, H. O., Jurgens, H., Saupe, D., Fractals for the Classroom, Springer-Verlag New York, Inc., 1992.

[Ramam] Ramamurthi, B., Gersho, A., Classified Vector Quantisation of Images, IEEE Trans. on Commun., v. 34, Nov 1986.

[Schal] Schaller, R. R., Moore;s Law: past, present and future, IEEE Spectrum, June 1997, 52-59.

[Sedg] Sedgewick, R., *Algorithms*. Addison-Wesley, August 1984.

[Shan] C. E. Shannon, ``A mathematical theory of communication,'' Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July and October, 1948.

[Thom] Thomas, L., Deravi, F., Region-Based Fractal Image Compression Using Heuristic Search, IEEE Trans. Image Proc. Vol. 4, No. 6, June 1995.

[Wall] Wallace, G. K., The JPEG Still Picture Compression Standard, *Comm. of the ACM*, Volume 34, Number 4, 1991.

[Watt] Watt, A., Fundamentals of Three-Dimensional Computer Graphics, Addison-Wesley, Britain, 1989.

[Will] Williams, R. F., Compositions of contractions, Bol. Soc. Brazil. Mat. 2, 1971, pp 55-59.