DUBLIN CITY UNIVERSITY

SCHOOL OF ELECTRONIC ENGINEERING

Fractal methods in Image Analysis and Coding

David Neary B.Sc.

This thesis is submitted as fulfillment of the requirements for the award of the degree of M.Eng. by research to **Dublin City University.**

Supervisor: Dr. Sean Marlow
School of Electronic Engineering
Dublin City University, Dublin 9
Ireland

January 22, 2001

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award on M.Eng. is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: [illegible] Candidate.

ID No.: 96970481

Date: 23/01/2001

Acknowledgements

I would like to express my sincere gratitude to Dr. Sean Marlow and Dr. Noel Murphy, who were both extremely supportive, even when things were going badly. Their guidance, comments and assistance were invaluable.

I would also like to thank Prof. Charles McCorkell and the School of Electronic Engineering for financial assistance during the course of my research.

Among the many people who helped, encouraged and distracted me during the course of this project, all of which I am grateful for, were Emmet Caulfield, Miguel Monclus and Anne Fraysse.

Abstract

In this thesis we present an overview of image processing techniques which use fractal methods in some way. We show how these fields relate to each other, and examine various aspects of fractal methods in each area.

The three principal fields of image processing and analysis that we examine are texture classification, image segmentation and image coding.

In the area of texture classification, we examine fractal dimension estimators, comparing these methods to other methods in use, and to each other. We attempt to explain why differences arise between various estimators of the same quantity. We also examine texture generation methods which use fractal dimension to generate textures of varying complexity.

We examine how fractal dimension can contribute to image segmentation methods. We also present an in-depth analysis of a novel segmentation scheme based on fractal coding.

Finally, we present an overview of fractal and wavelet image coding, and the links between the two. We examine a possible scheme involving both fractal and wavelet methods.

Contents

List of Figures

Chapter 1

Introduction

1.1 Goals and aims

In the relatively short period that they have been studied, methods based on the analysis of nowhere differentiable functions which exhibit exact or statistical self-similarity have been used in a wide range of image processing and analysis applications. Fractals, from the Latin fractus (meaning fragmented or irregular), have been used to synthesize images, to classify textures, to segment and compress images and to generate amazingly complex and beautiful images.

Its sister field of chaos theory has been used to examine the nature of turbulence in fluid flow, to analyse weather systems and stock markets, and to offer theoretical insights into problems in particle physics.

One cannot hope to give a complete account of all the applications of fractal geometry which exist. We present here an overview of the principal uses of fractal methods in image analysis and compression, as well as proposing several extensions to existing methods.

We hope to present a unified overview of the methods used, from the study of fractal dimension to image compression using partitioned iterated function systems.

1.2 Overview

In this thesis we present a summary of some of the fractal methods used in image analysis and coding. We examine fields as disparate as texture classification, image segmentation and image compression. Within these fields, we examine various methods which are of a fractal nature, and present qualitative analysis of these methods compared to other methods which are popular in the fields. We hope to emphasise the common threads which run through much of fractal geometry as it applies to digital image processing.

In chapter 2 we introduce methods of texture classification, including fractal dimension estimation, and show how various estimators of fractal dimension perform differently. We try to explain why these types of behaviors occur.

We also introduce some of the methods which have been used for texture generation using fractal methods, and fractional Brownian methods.

In chapter 3 we present several common segmentation tools, including region growing and histogram analysis. We present the basic theory of segmentation, and we use various estimators of fractal dimension to segment images into distinct regions.

We also perform a detailed analysis of a novel segmentation method which is based on fractal image compression methods.

In chapter 4, we present an overview of the mathematical basis for fractal image coding, along with other coding methods. We also present a detailed introduction to wavelet analysis. We then examine the possible links between the fractal transform and the wavelet transform.

Chapter 5 is the conclusion and discussion. We discuss the common threads which run throughout the thesis, and propose possible extensions of this work for the future.

Chapter 2

Fractal methods in texture analysis

2.1 Introduction

We analyse textures, in general, in the hope of discovering characteristics about them which will help us to completely describe that texture or family of textures. This is useful for several types of applications, particularly texture generation, in terms of artificial images, and machine recognition of objects. Much of the work done in texture classification is relevant in image segmentation as well, since texture is one of the characteristics one might use to segment.

The great difficulty in dealing with the concept of texture is arriving at an objective, stochastic definition of what we are dealing with. Ask someone what texture is, and the chances are they'll reply with an equally vague answer ("It's roughness and smoothness"), or by touching or rubbing something. This lack of rigour makes it difficult to define exactly why one texture differs from another.

Researchers have managed, however, by focusing on three or four key characteristics of a surface [1]

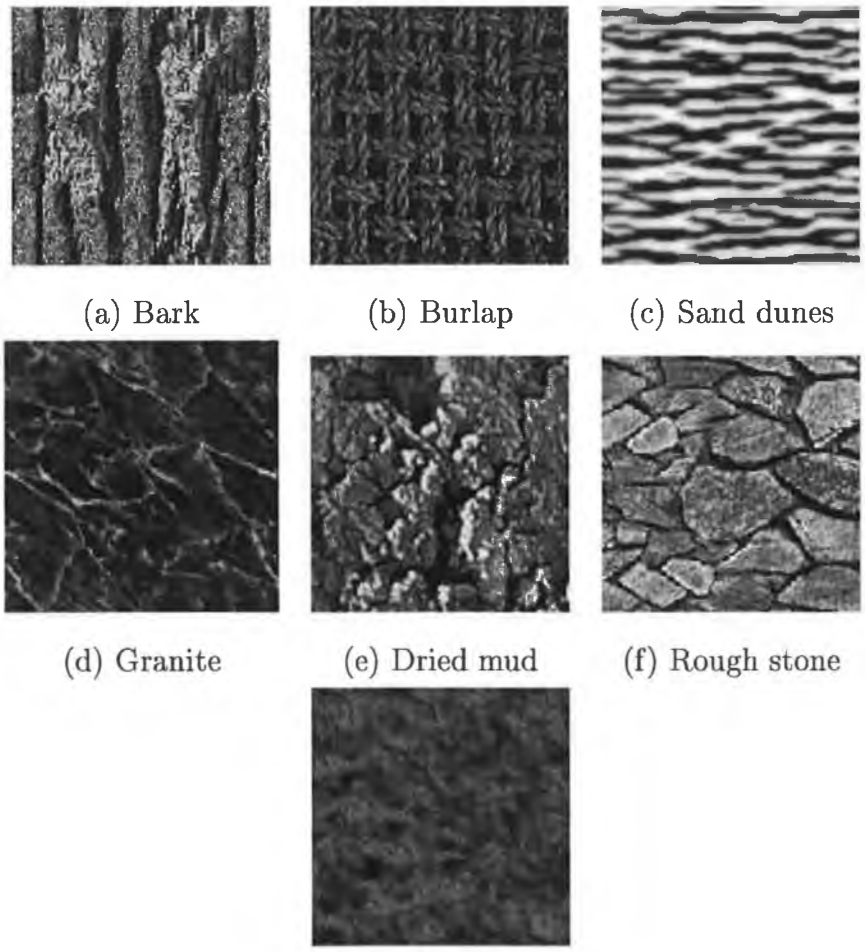
1. **Contrast** This is, broadly, the range of grey-scales in the image.
2. **Coarseness** Whether the image is fine or grainy.
3. **Regularity** This is concerned with the uniformity within the texture.
4. **Directionality** This defines the degree of difference along different axes of the image. The grain of wood, for example, would be more directional than a pebble-dash wall.

A number of textures are presented in Figure 2.1. These show all of the traits mentioned here, including high contrast, coarseness and directionality.

Here, we give an overview of the various texture analysis methods which exist, and we give an in-depth analysis of fractal dimension as one such method.

It has been established [2] that, at least on a heuristic and intuitive level, surfaces of a given fractal dimension correspond well to our ideas of roughness and smoothness. Given this, we attempt to use fractal dimension as a measure of texture.

When we refer to the fractal dimension, we mean specifically the Hausdorff-Besicovitch dimension [3]. We wish to examine the accuracy of the various estimators of the Hausdorff dimension.



(a) Bark (b) Burlap (c) Sand dunes

(d) Granite (e) Dried mud (f) Rough stone

(g) Walnut shell

Figure 2.1: Assorted textures

To this end, we generate surfaces of a given fractal dimension by two different methods, the inverse Fourier power spectrum method [2] and the midpoint displacement method [4]. For each of these generated textures, we calculate the fractal dimension by the methods of Fourier power spectrum estimation and the box-counting dimension.

By this means we evaluate the accuracy of each of these estimators with respect to the fractal dimension.

2.2 Texture classification methods

The majority of texture classification methods are based on some kind of statistical method [5]. The goal in statistical texture classification is to define a feature vector based on some characteristics of the texture, which represents a point in multi-dimensional space. We then hope that this feature vector can be used to assign the texture to a specific class of textures.

Within this class, we find many classification methods which can broadly be split into three groups: those based on spatial frequencies, co-occurrence matrices and fractal methods.

2.2.1 Frequency based classification

Several estimators exist which utilise frequency transforms. The most popular of these in recent times has been the *Gabor filter*[6, 7], which uses the wavelet transform to generate the Gabor convolution energy measure of the texture.

This method is not rotation invariant, which is both a benefit and flaw in the method. It means that the directionality of the texture is figured into the measure, if we calculate the feature over many directions, but as a non-global measure, small phase changes in the texture could cause it to classify the same texture in different classes. Methods have been proposed [7] to get around this.

There also exist a number of classification methods based on the Fourier transform, which can characterise both directionality and coarseness by analysis of the frequency domain. If, for example, the power spectrum of the texture shows a peak in a ring around the origin, we can deduce the coarseness of the image from both the distance from the origin of the ring (higher frequencies, indicating finer texture) and the breadth of the ring (broader range of frequencies, indicating greater randomness). We can also deduce the directionality of the image by analysis of wedges around the origin.

In general, frequency based methods have a number of problems. They tend not to be invariant under simple greyscale mappings, and it has been shown that frequency based methods are less efficient than others.

2.2.2 Co-occurrence matrices

This represents a large range of methods, which are based on analysing repeating structures in a texture. Normally, a number of co-occurrence matrices are calculated for the texture, and from these a number of features are calculated.

The co-occurrence matrix is defined by a distance and an angle, and its mathematical definition is

$$\begin{aligned} C_{\theta,d}(x,y) &= \big|\{(\mathbf{m},\mathbf{n}) \in (M \times N) \times (M \times N) : \mathrm{d}(\mathbf{m},\mathbf{n}) = d, \\ &\qquad \tan^{-1}(\mathbf{m}-\mathbf{n}) = \theta \text{ or } \pi - \theta \text{ and } f(\mathbf{m}) = x, f(\mathbf{n}) = y\}\big| \end{aligned}$$

where $\mathrm{d}(\mathbf{a},\mathbf{b})$ is an appropriate metric, usually $\mathrm{d}((a,b),(c,d)) = \max(|a-c|,|b-d|)$, $f\,((M \times N) \mapsto \mathbb{N}(0,255))$ is the image to be analysed. This definition of the co-occurrence matrix ensures a diagonal matrix $C_{\theta,d}$.

Given this matrix, there are a large number of features which we can deduce. Some of these are [5, 8, 9] energy, or angular second moment, entropy, correlation, inverse difference moment, inertia, maximum probability, contrast and variance.

Given the large number of features, and the large number of possible co-occurrence measures, this family of methods tend to provide accurate classification of textures. It is also invariant under simple greyscale transforms. Also, given the large number of features, it is rarely necessary to use more than six or seven features [8] to get an accurate characterisation. Grey-levels can be scaled to the range $(0, 32)$ or $(0, 64)$ with little loss of accuracy, but vastly improved calculation times [5].

However, because the dimensions of the co-occurrence matrix are the number of grey-levels squared, these methods tend to be computationally expensive. On textures with large texture elements (texels), however, it is less effective than spatial methods, since it does not consider the spatial relationship of texture primitives.

In fact, it has been shown [10] that co-occurrence features perform better than others, including fractal methods, in texture classification. It should be said, however, that on different types of textures that some methods are better suited to characterisation than others.

2.2.3 Fractal dimension

It has been shown [11] that there is a relationship between texture coarseness and the fractal dimension of a surface. It has therefore been suggested [11, 12, 13] that fractal dimension might be used as a feature for texture classification.

The fractal dimension, or *Hausdorff-Besicovich dimension* is defined in terms of several quite abstract mathematical ideas, and it is worth giving quick definitions of the terms we will use in explaining its definition.

- **diameter** The diameter of a subset $\mathcal{A}$ of $\mathbb{R}^n$ is defined, with a relevant metric $|.-.| : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$, as

$$|\mathcal{A}| = \sup \{|x-y| : x, y \in \mathcal{A}\}$$

- **δ-cover** A countable collection of sets $\{\mathcal{U}_i\}$ where $|\mathcal{U}_i| \leq \delta$ for all i, is a *δ-cover* of the set $\mathcal{A}$ if $\mathcal{A} \subset \bigcup_i^\infty \mathcal{U}_i$.

Given these definitions, we define a function $\mathcal{H}_\delta^s(\mathcal{A})$ as

$$\mathcal{H}_\delta^s(\mathcal{A}) = \inf \left\{ \sum_{i=1}^{\infty} |\mathcal{U}_i|^s : \{\mathcal{U}_i\} \text{ is a } \delta\text{-cover of } \mathcal{A} \right\} \tag{2.1}$$

Thus, as δ decreases, the number of possible δ-covers increases, and the infimum eventually approaches a limit. This is

$$\mathcal{H}^s(\mathcal{A}) = \lim_{\delta \to 0} \mathcal{H}_\delta^s(\mathcal{A}) \tag{2.2}$$

It is obvious that $\mathcal{H}_\delta^s$ is non-decreasing, and by equation (2.2) $\mathcal{H}^s$ is also non-decreasing.

Also, if $t > s$ and $\{\mathcal{U}_i\}$ is a δ-cover of $\mathcal{A}$, we have

$$\begin{aligned} \sum_{i=1}^{\infty} |\mathcal{U}_i|^t &= \sum_{i=1}^{\infty} \left(|\mathcal{U}_i|^s |\mathcal{U}_i|^{(t-s)} \right) \\ &\leq \sum_{i=1}^{\infty} \delta^{t-s} |\mathcal{U}_i|^s \end{aligned}$$

and taking infima, , we have $\mathcal{H}_\delta^t(\mathcal{A}) \leq \delta^{t-s} \mathcal{H}_\delta^s(\mathcal{A})$. Letting $\delta \to 0$, it follows that if $\mathcal{H}^s(\mathcal{A}) < \infty$ for any value of s, $\mathcal{H}^t(\mathcal{A}) = 0$ for all $t > s$.

The fractal dimension, $\dim_H(\mathcal{A})$, is defined as

$$\begin{aligned}\dim_H(\mathcal{A}) &= \inf\{s : \mathcal{H}^s(\mathcal{A}) = 0\} \\ &= \sup\{s : \mathcal{H}^s(\mathcal{A}) = \infty\}\end{aligned}$$

Among the properties of the fractal dimension we use, some are that it is always greater than or equal to the topological dimension of any point-set, and that the fractal dimension of a self-similar set is directly related to the scaling ratios of the self-similar subsets. $\dim_H$ is also invariant under translation, rotation and simple grey-scale transformations. We also rely on the premise that the fractal dimension of a point-set increases with its complexity. More rigorously, $\dim_H(E \cup F) = \max(\dim_H(E), \dim_H(F))$ and if $E \subset F$, $\dim_H(E) \leq \dim_H(F)$ [3].

In fact, the definition of a fractal is any point-set whose fractal dimension is strictly greater than its topological dimension. This definition implies a number of things, one of which is that a fractal will have detail on every scale.

The fractal dimension of geometric shapes does not differ from their topological dimension, since, for example, a line is the same on all scales, and no magnification will turn up new detail. Also, a smooth curve approximates to a line segment under adequate magnification, giving them the same complexity one would intuitively expect.

Since most real-life textures exhibit detail on many scales, and since a degree of statistical self-similarity exists in most textures, we suggest that for classes of textures which exhibit these characteristics, the fractal dimension can serve as a good texture classifier.

In reality, calculating the fractal dimension from first principles is difficult computationally. We will instead use alternatives which are related to the fractal dimension, the box-counting dimension [12, 3] and the Fourier Estimator [4].

2.3 Fractal dimension estimators

2.3.1 Box-counting dimension

The *box-counting dimension* is defined as [3]

$$\begin{aligned}
\underline{\dim}_B(F) &= \underline{\lim}_{\delta\to 0}\frac{\log N_\delta(F)}{-\log\delta} && (2.3)\\
\overline{\dim}_B(F) &= \overline{\lim}_{\delta\to 0}\frac{\log N_\delta(F)}{-\log\delta} && (2.4)\\
\dim_B(F) &= \lim_{\delta\to 0}\frac{\log N_\delta(F)}{-\log\delta} && (2.5)
\end{aligned}$$

if this limit exists, where N_δ is the number of boxes of diameter δ needed to cover the set F. In fact, if it exists $\dim_B(\mathcal{A}) = \underline{\dim}_B(\mathcal{A}) = \overline{\dim}_B(\mathcal{A})$. This simply calculates the complexity of an image as you decrease in scale, and in most cases we can assume convergence of $\dim_B$.

For example, the box-counting dimension of the well-known Cantor set is $\log 2/\log 3$ since at every scale, 2^n boxes of diameter $\left(\frac{1}{3}\right)^n$ are needed to cover the set. Note that a box B of diameter δ is a set where $d(x,y) \leq \delta \;\forall x,y \in B$, in whatever metric we happen to be using.

In terms of the fractal surfaces we will be measuring, we take $\delta = 2^{-n}$ for $n \in \mathbb{N}$, normalise F such that $\max F \leq 1$ and the domain of F is $(0,1)\times(0,1)$.

We define $N_\delta(F)$ as

$$N_\delta(F) = \sum_{i,j=1}^{2^n} n_\delta(F_{i,j})$$

$$n_\delta(F_i) = \left\lceil \frac{\max F_i(x)}{\delta} \right\rceil - \left\lfloor \frac{\min F_i(x)}{\delta} \right\rfloor$$

and

$$F_{i,j}(x,y) = \begin{cases} F(x,y) & \text{if } (i-1)2^{-n} \leq x < i2^{-n} \text{ and } (j-1)2^{-n} \leq y < j2^{-n} \\ 0 & \text{otherwise} \end{cases}$$

By the standard method, we evaluate $N_\delta(F)$ for a number of values of δ, and take $\dim_B F$ as the negative of the slope of the graph derived from plotting $\log N_\delta(F)$ versus $\log \delta$.

It can be shown quite easily (by use of equation (2.1)) that the box-counting dimension is related to the fractal dimension by the inequality $\dim_H(F) \leq \dim_B(F)$. Since the box-counting dimension uses coverings by sets of equal size, as opposed to the fractal dimension, which merely limits the maximum size of the cover, it is a much easier measure to calculate. This introduces a number of flaws in the method also, and in general the box-counting dimension may be considered a measure of the efficiency of equal sized small sets in covering a set [3].

In general, there are a number of problems with the box-counting dimension as a texture classifying measure. Firstly, discontinuities in the image artificially raise the dimension, making it unsuitable as an estimator of dimension on noisy images. This property also introduces large artifacts at texture boundaries. Secondly, on certain types of images, the box-counting dimension fails because of its requirement that box-coverings are of fixed size.

That said, the box-counting dimension is easily calculable, and as a measure in its own right, is a useful guide to the complexity of a set, making it a tool worth considering for texture classification.

2.3.2 Fourier estimator

We call a function $V_H(t)$ a *Fractal Brownian* line function, or fractional Brownian function, if it has zero-mean Gaussian increments with variance

$$E\left([V_H(t+\delta) - V_H(t)]^2\right) \propto |\delta|^{2H}$$

where $H \in (0,1)$. H is the Hurst coefficient of a Brownian motion, and as has been shown by Pentland [2, 11] it is directly related to the fractal dimension D_F of $V_H(t)$ by $D_F = E + 1 - H$, where E is the topological dimension. It can also be shown [2] that $V_H(t)$ has a Fourier spectrum with power $F_H(f)$ such that

$$F_H(f) \propto f^{-\beta}$$

and H is related to β by $\beta = 2H + 1$

Since any cross section of a two-dimensional Fractal Brownian surface $V_H(x,y)$ is a Fractal Brownian line function of identical H, we want

$$E\left([V_H(x + \delta\cos\gamma, y + \delta\sin\gamma) - V_H(x,y)]^2\right) \propto |\delta|^{2H}$$

independent of γ [2]. We call a function which fulfills this condition two-dimensional fractional Brownian motion. It has been shown by Voss [4] that this requires the two-dimensional power spectrum of the surface to be

$$\mathbf{F}_H(f,\theta) \propto f^{-\beta}$$

where $\beta = 2H + 2$.

Therefore we can relate β, the spectral fall-off of the image to estimate D, the fractal dimension, using the formula

$$D = 4 - \frac{\beta}{2}$$

where $2 \leq \beta \leq 4$, as shown by Pentland [2]. This method of estimation has been shown [14] to be more accurate than the box-counting dimension.

2.4 Texture synthesis methods

2.4.1 Midpoint displacement method

To evaluate the fractal dimension estimators, it was necessary for us to synthesise textures of known fractal dimension. Two methods were used to do this, each with their merits. The first, described here, is by midpoint displacement. The second, which will be described in the next section, is the inverse Fourier power spectrum method.

The midpoint displacement method started out life as a means of approximating fractional Brownian motion with $H = \frac{1}{2}$. Since the link between fractal dimension and fractional Brownian motion have become well established, this method has been expanded to synthesize landscapes and generate textures of any given fractal dimension.

The method of synthesis is actually quite simple. We start with a grid of size $(2^n + 1) \times (2^n + 1)$. For each of the four corners we use a Gaussian random variable $N(\mu, \sigma^2)$ to generate heights.

Each successive iteration adds the midpoints of all the squares in the previous level, producing a grid rotated by 45 degrees, of resolution $\sqrt{2}$ times the previous level. By repeating this iteration, we double the number of points every two iterations (see Figure 2.2).

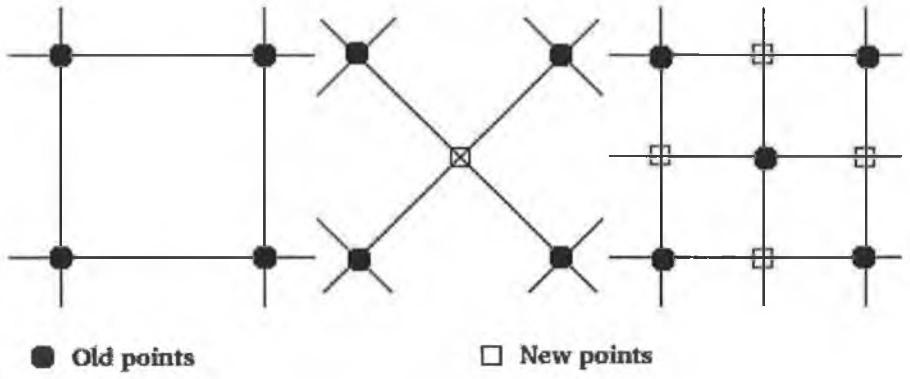
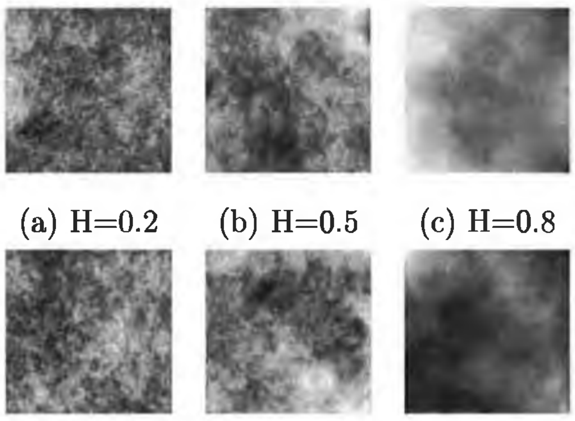


Figure 2.2: A doubling in resolution with midpoint displacement



(a) H=0.2 (b) H=0.5 (c) H=0.8

(d) H=0.2 (e) H=0.5 (f) H=0.8

Figure 2.3: Various clouds generated with random additions (a)-(c) and midpoint displacement (d)-(f)

Since a surface X is Brownian if

$$E\left(d\left(X\left(p_1\right), X\left(p_2\right)\right)^2\right) \propto d(p_1, p_2)^{2H} \tag{2.6}$$

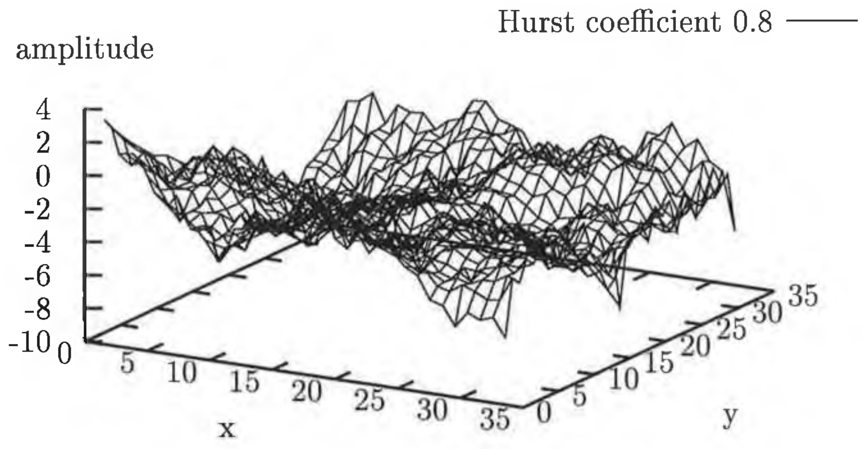


Figure 2.4: Surface of dimension 2.2 generated by midpoint displacement

for $p_1, p_2 \in X$ and $0 < H < 1$,we can deduce that for $p_1, p_2 \in X$ and $p_3 = \frac{p_1+p_2}{2}$,

$$d(p_1,p_3) = d(p_2,p_3) = d(p_1,p_2)/2 \text{ for the standard } \mathbb{R}^2 \text{ metric}$$

$$E\left(d(X(p_1),X(p_3)\right) \propto 2^{-2H} d(p_1,p_2)^{2H}$$

In practice, what we do is set $X(p_3) = \frac{X(p_1)+X(p_2)}{2} + D$, where D is taken from a random variable with zero mean and variance $\sigma^2/2$, where σ^2 is the constant of proportionality in equation 2.6, and is the variance of our random variable for initial selection of corner heights. This gives

$$\begin{aligned} E\left\langle d\left(X(p_1),X(p_3)\right)\right\rangle &= E\left\langle d\left(X(p_1),X(p_2)\right)/2 + D\right\rangle \\ &= E\left\langle \frac{d\left(X(p_1),X(p_2)\right)}{2}\right\rangle + E\left\langle D\right\rangle \\ &= d(p_1,p_2)^{2H}\left(\frac{\sigma^2}{2^{2H}}\right) + 0 \end{aligned}$$

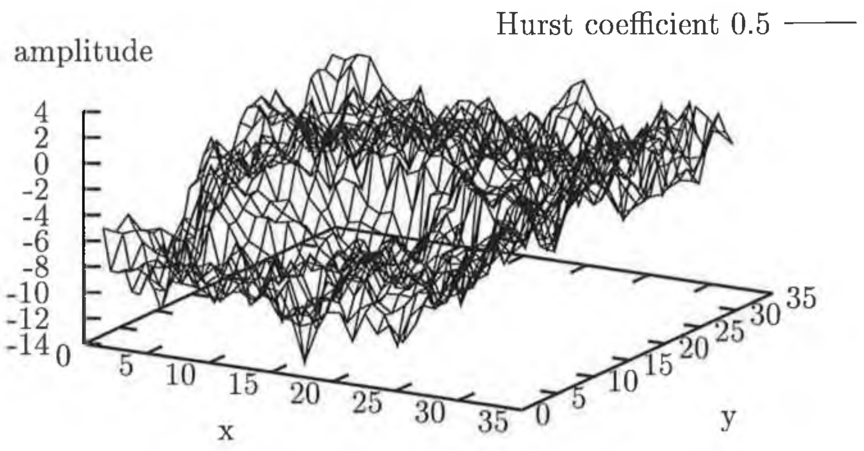


Figure 2.5: Surface of dimension 2.5 generated by midpoint displacement

By iteration, we can resolve our surface to any resolution.

There is a problem with this method, however. It has been shown that for values other than $H = 0.5$, it does not produce true Brownian surfaces, and therefore, does not produce pure statistically self-similar surfaces either. The method can be modified to the *random additions method* which compensates for this phenomenon by adding to points which were previously calculated at each successive level.

For the most part, we expect results from the random additions method to be almost exactly the same as the midpoint displacement method.

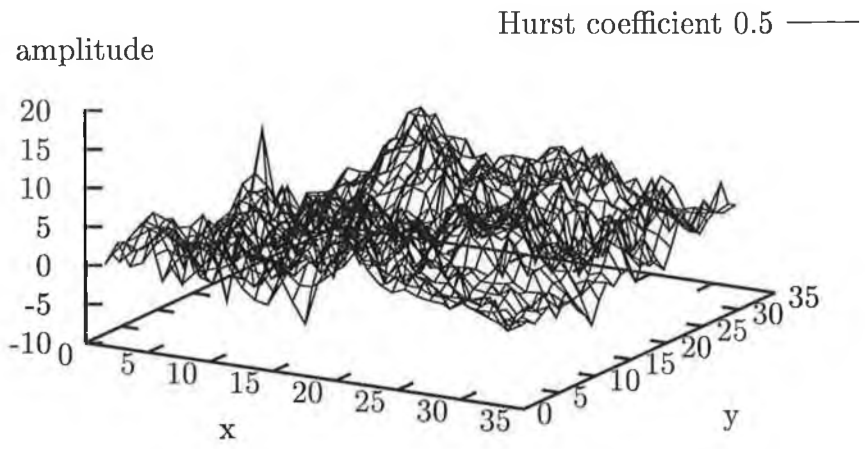


Figure 2.6: Surface of dimension 2.2 generated by random additions

2.4.2 Inverse Fourier power spectrum method

The second major method we use to synthesise textures is the inverse Fourier power spectrum method. As has been shown previously, the Fourier power spectrum of a fractal surface can be expected to have a simple power law. That is to say, in one dimension the power spectrum of a Brownian curve is expected to have the form

$$V(f) = |F(f)|^2 \propto \frac{1}{f^\beta}$$

with $\beta = 2H + 1$ relating the coefficient to the Brownian parameter [15]. In two dimensions, this extends easily so that we expect

$$V(u,v) \propto \frac{1}{(u^2+v^2)^{H+1}}$$

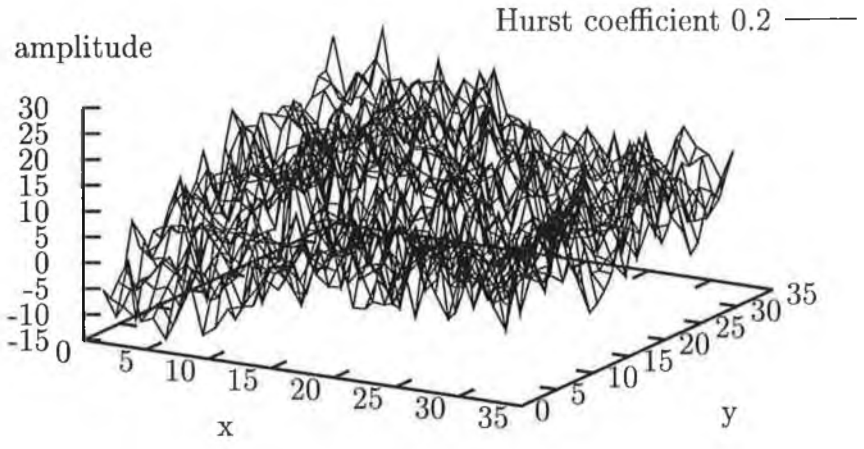


Figure 2.7: Surface of dimension 2.8 generated by random additions

and we can thus produce a function X such that

$$X(x,y) = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1} \alpha_{kl} e^{2\pi i(kx+ly)}$$

for $x, y = 0, \frac{1}{N}, \frac{2}{N}, \cdots, \frac{N-1}{N}$, and letting our Fourier coefficients fulfill

$$E(|\alpha_{kl}|^2) \propto \frac{1}{(k^2+l^2)^{H+1}}$$

taking into account the symmetries which are implied with X being a real function.

The fractal dimension of this surface X will be $D_f = 3 - H$. The algorithm for generation consists simply of generating our coefficients in the frequency domain and then performing a 2-D inverse Fourier transform.

Figures 2.8 and 2.9 contain samples of surfaces generated using this method.

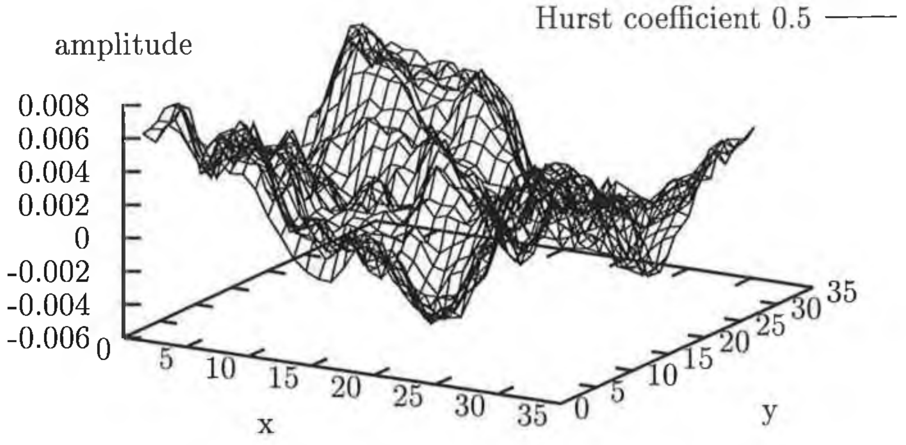


Figure 2.8: Surface of dimension 2.5 generated by Fourier filtering method

2.5 Results and discussion

For each of the three texture synthesis methods used, Fourier synthesis, mid-point displacement and successive random additions, one hundred test surfaces were generated, with $H = 0, 0.01, 0.02, \cdots, 1$. For each surface, both the Fourier power spectrum estimator and the box-counting dimension were calculated. The resulting graphs of H_{est} versus H are contained in Figures 2.10 – 2.15. For the purposes of normalisation, we do not use D_B for the box-counting dimension, instead we use $H_{est} = 3 - D_B$. The dimensions of the generated surfaces were 32×32.

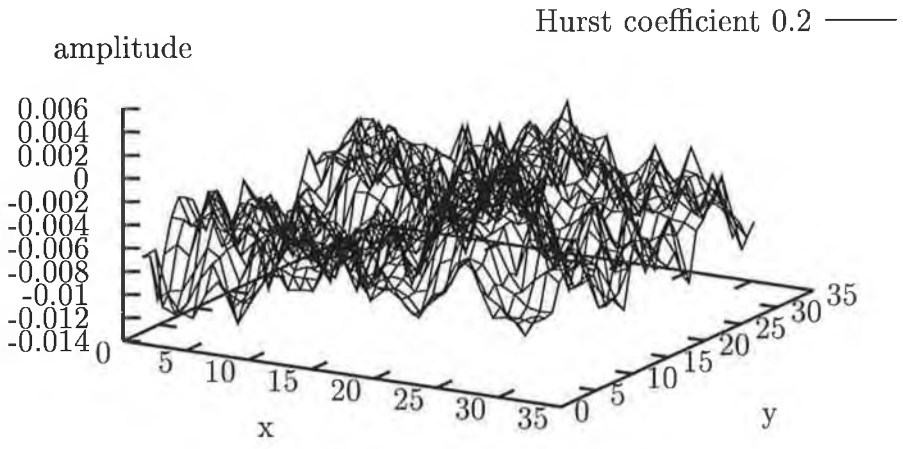


Figure 2.9: Surface of dimension 2.8 generated by Fourier filtering method

2.5.1 Box-counting dimension results

From Figures 2.10, 2.12, 2.14, it is obvious that in the majority of cases, the box-counting dimension underestimates the true dimension of the surface, in some cases by up to 40%. In addition, the correlation coefficients of the graphs are quite low, particularly on surfaces generated by Fourier synthesis. The correlation coefficients of all the graphs are laid out in Table 2.1.

	Box-counting	Fourier estimator
Fourier synthesis	0.83	0.92
Midpoint displacement	0.91	0.90
Random additions	0.89	0.92

Table 2.1: Correlation coefficients from Figures 2.10 – 2.15

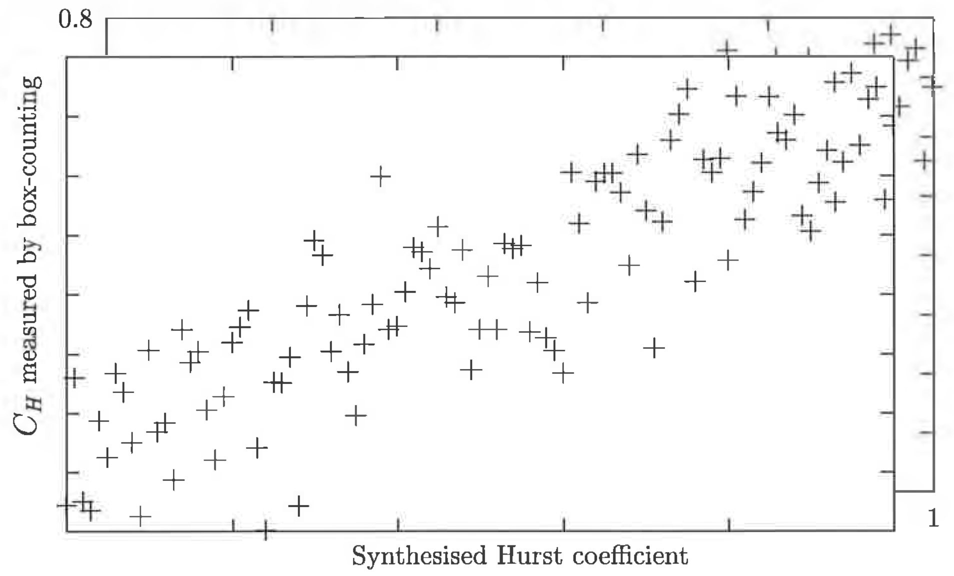


Figure 2.10: Dotplot of dimension measured by box-counting method versus dimension generated with Fourier synthesis

However, there are several things which can recommend the box-counting dimension. It is computationally much faster than most other fractal estimation methods, and out-performed the Fourier power synthesis consistently. It is also easy to calculate, even for large images, where the Fourier transform in the power spectrum estimate is noticeably slower on larger images.

In terms of the synthesis methods, there is very little discernible difference in the resulting surfaces.

The box-counting dimensions of the seven textures shown in Figure 2.1 were also measured, as were the power spectrum estimates. The results are shown in Figure 2.16. As expected, for textures which do not have universal statistical self-similarity, such as "burlap" and "dunes", the methods perform badly, and produce unlikely estimates, but produce reasonable estimates,

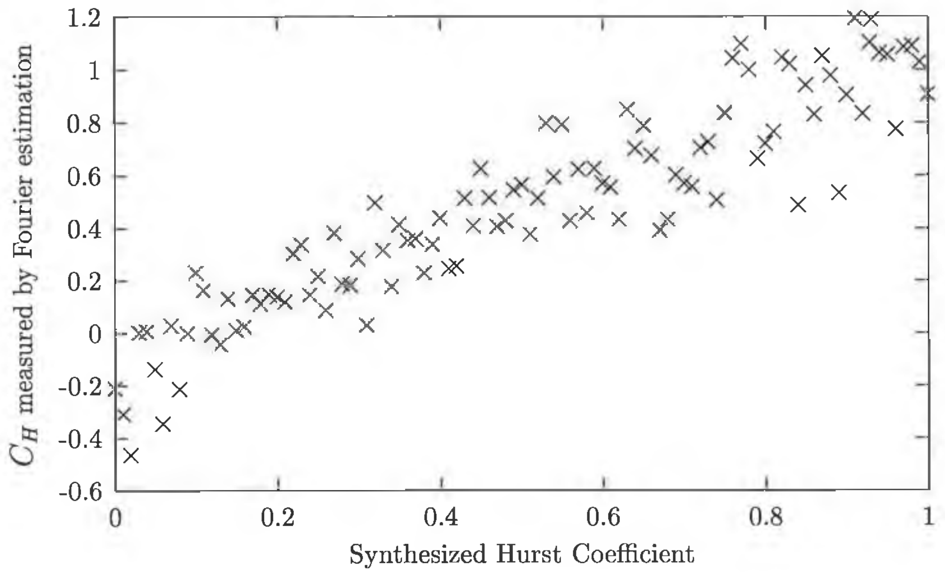


Figure 2.11: Dotplot of dimension measured by Fourier power spectrum method versus dimension generated with Fourier synthesis

which correspond with our idea of roughness, for the less directional textures, for example "walnut".

However, on quite different textures, the estimator produced almost identical results, in particular with "stone" and "granite". The estimated dimensions would classify these quite different textures into the same texture class. It is worth noting that the effective range of the box-counting dimension appears to be 2.1 to 2.6 for all but a small class of images.

In particular, it appears that the dimension was over-estimated for both "bark" and "burlap", two images which displayed high contrast and sharp edges within the texture. I would question whether this small dynamic range is adequate for texture classification in general. However, I feel that for certain classes of images, namely non-directional textures with small texture

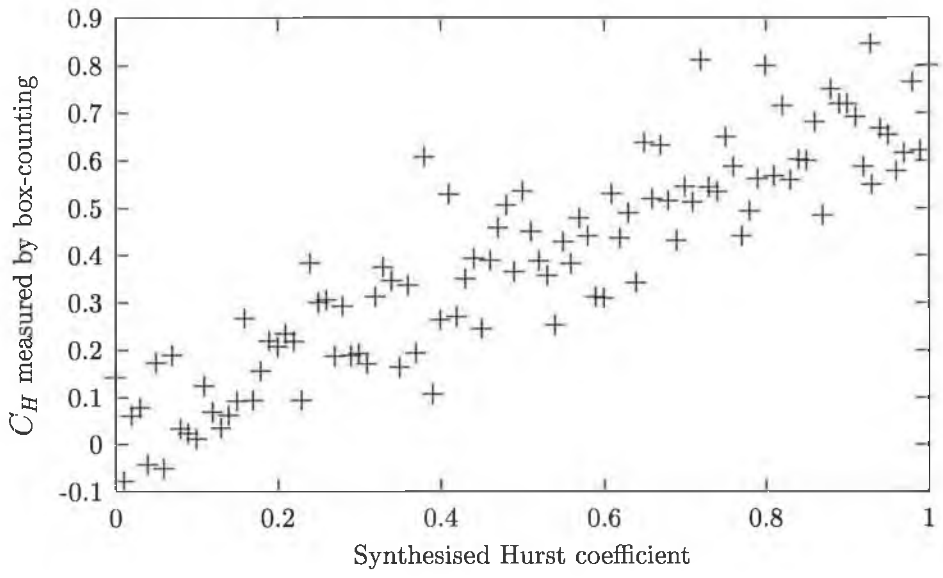


Figure 2.12: Dotplot of dimension measured by box-counting method versus dimension generated with midpoint displacement

elements, the box-counting dimension is worth considering.

2.5.2 Fourier power spectrum estimator results

The most notable thing about the Fourier estimator results is that when the dimension is closer to 2, it underestimates, but as the dimension approaches 3, it overestimates. However, the graphs generated have a considerably higher straight-line correlation than the box-counting dimension, making this a more usable measure for texture classification.

We see, particularly with the "dunes" texture, that the estimate resulting from the Fourier estimator seems to be lower if the texture has a directional nature. This may be because of a periodicity introduced, which shows up in

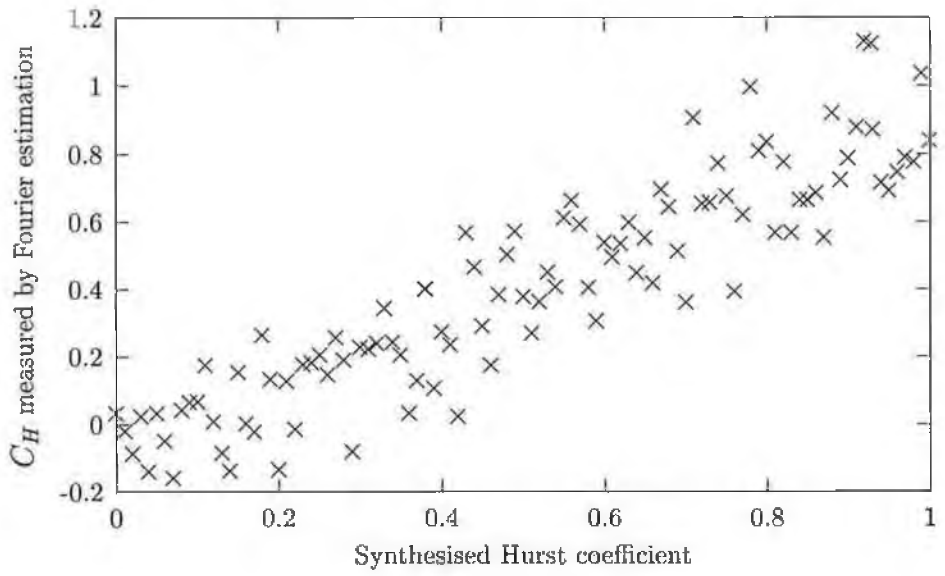


Figure 2.13: Dotplot of dimension measured by Fourier power spectrum method versus dimension generated with midpoint displacement

the frequency domain as a spike, lowering the slope of the log-log relationship which results.

2.5.3 Summary

We have shown that on artificially created surfaces, the Fourier estimator produces results which imply it to be a better tool for texture classification than the box-counting dimension. However, when applied to real-life textures, the dimension estimates were somewhat erratic, particularly with the Fourier estimator.

In fact, quite different textures can have similar fractal dimensions. The

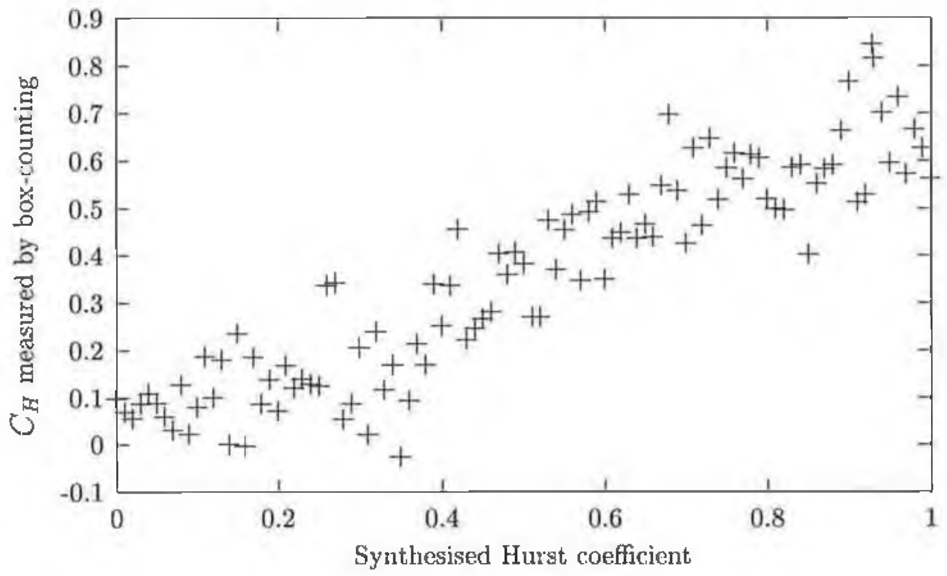


Figure 2.14: Dotplot of dimension measured by box-counting method versus dimension generated with successive random additions

fractal dimension is just one measure we can use to classify textures, however, and we believe that it can play a large part in successfully classifying a texture, as part of a larger classification vector.

In fact, the box-counting and Fourier estimators produce quite different results as well, on some textures. This is the case especially on textures which are not strictly fractal, such as the dry clay texture, and is probably due to the fact that spikes in the Fourier domain have the ability to substantially affect the dimension estimate.

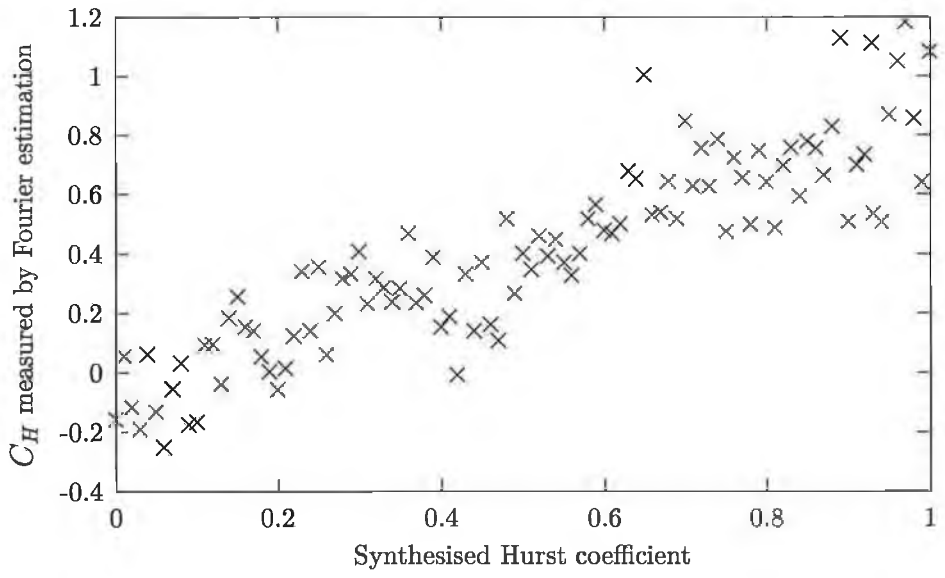
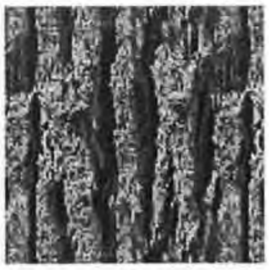
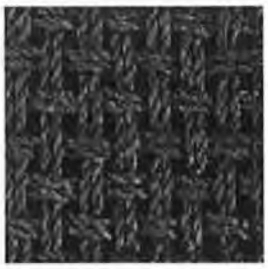


Figure 2.15: Dotplot of dimension measured by Fourier power spectrum method versus dimension generated with successive random additions



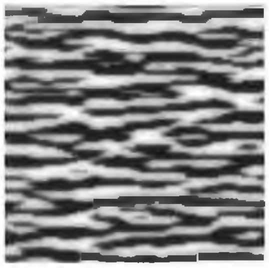
$D_F = 2.43$

$D_B = 2.43$



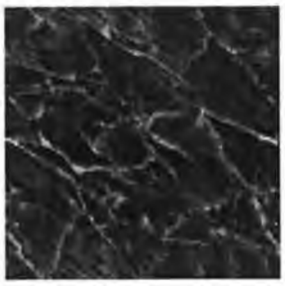
$D_F = 2.54$

$D_B = 2.46$



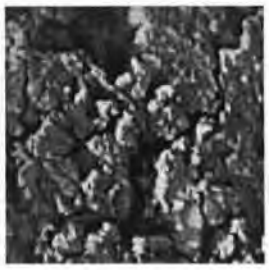
$D_F = 2.03$

$D_B = 2.29$



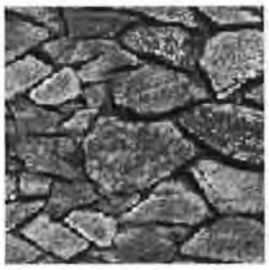
$D_F = 2.28$

$D_B = 2.25$



$D_F = 2.33$

$D_B = 2.30$



$D_F = 2.24$

$D_B = 2.33$

$D_F = 2.38$

$D_B = 2.20$

Figure 2.16: D_F and D_B of the textures in Figure 2.1

Chapter 3

Segmentation

3.1 Overview

In terms of image segmentation, the goal is easy to state, and difficult to achieve. We wish to have a method of segmenting an image into useful regions without the use of human intervention. Essentially, we want a computer to be able to differentiate, by some means, the different areas of an image.

This does not necessarily, it should be said, imply understanding. Where we see a tree and can identify it as such, a computer will see a region which it recognises as different, without knowing what it is, or whether it's important in the image or not. A human face and a cloud in the sky hold much the same significance.

For the sake of clarity, we will refer to *objects* within an image to be specific regions of interest, for example a head, a car, a tree, and so on. We refer to *regions* as being areas of an image which share some characteristic, and are to some degree homogeneous with respect to some characteristic. Image segmentation may perform a complete segmentation (where regions correspond to individual objects in the image), or partial segmentation, in which case objects and regions in the image may not correspond exactly.

Segmentation algorithms have been proposed which use many different tactics to segment digital images. Generally these involve taking certain features, and trying to find a good classifying algorithm in the resulting feature-space, for the purpose of determining within certain probabilistic bounds whether a point is in a region. Most methods of this type require some kind of human intervention, if only to tell the computer how many regions to isolate.

3.2 Image segmentation methods

There are many characteristics we might use to segment an image into regions – colour, brightness, texture and edge detection are just a few.

However, there are three principal underlying methods involved in almost all image segmentation. Those methods are edge detection, region growing and histogram analysis [5].

3.2.1 Histogram analysis

This method is usually the first step in attempting to segment an image, since it is one of the easier methods to implement and it can be quite effective for certain classes of images.

The histogram is made up of some global characteristic of the image, usually colour or brightness (luminance). However, given an image $f(i,j)$, we can also construct and analyse a function $g(i,j)$ from any context-specific feature, such as texture, which may be assigned to each pixel in the original image [2]. Mapping $f(i,j)$ to $\{g_n(i,j)\}$, based on local characteristics, is also one of the most popular methods of producing feature vectors for region growing techniques.

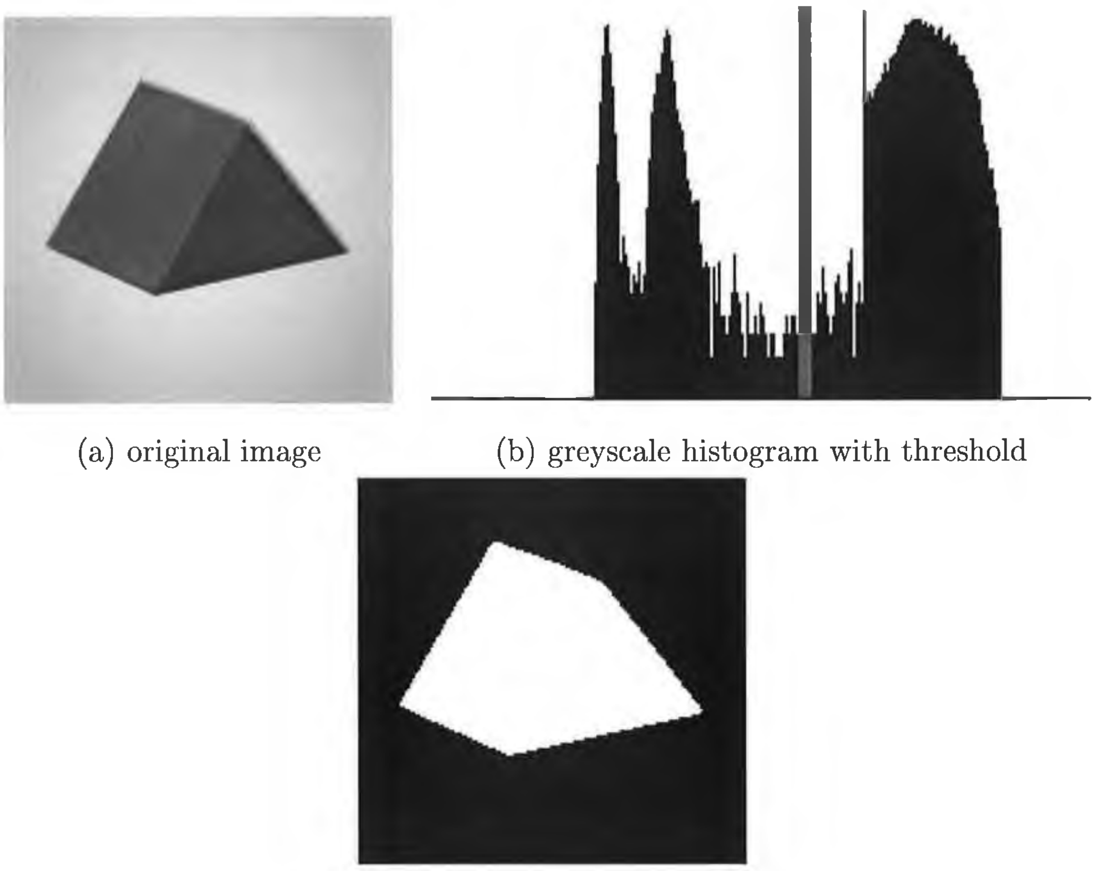
The basic theory of histogram thresholding is quite simple. If we are looking for n regions, we are seeking to find $n-1$ values $\{v_1, \cdots, v_{n-1}\}$ in the range of the histogram which best mark those regions. Then, we set

$$\begin{aligned} s(i,j) \quad &= l(0) \quad \text{for } f(i,j) < v_1 \\ &= l(1) \quad \text{for } v_1 \leq f(i,j) < v_2 \\ &\vdots \\ &= l(n-1) \quad \text{for } v_{n-1} \leq f(i,j) \end{aligned}$$

where $l(i)$ are the greylevels we assign to each group.

Thresholding techniques address the task of finding the $\{v_i\}$ which are the best boundaries for the given image. In images where we seek to isolate objects from a background of a distinct background, thresholding is ideal.

Sometimes the task of selecting a threshold is quite easy, when there is a clear difference between the greylevels of the objects we wish to segment (see Figure (3.1)). In this case, we find the local maxima, or the highest point of a given peak, and take our threshold value as the local minima (lowest valley) between them. In our example, there are three clear peaks, corresponding (on the left) to the two darker sides of the block, and (on the right) a wider peak corresponding to the light coloured background. The thresholding level


(a) original image

(b) greyscale histogram with threshold

(c) segmented image

Figure 3.1: A simple image to be segmented

we chose is marked in grey.

Things are not normally so simple. There are several quite complicated adaptive thresholding schemes [16], most of which revolve around finding local maxima and minima in the histogram, or parameterising the histogram into a useful form.

However, because it does not consider positional data, and is a purely global parameter, it is usually not suitable for the segmentation of all but the simplest images.

3.2.2 Edge detection

Edge-based segmentation methods involve finding the edges of objects in the image, possibly by more than one method, and using this edge information to guess at complete boundaries for the principle objects in the image.

Edge detection has many problems, principally through noise in the image providing pales edge information, or fragmenting the true edges.

Principal edge detection techniques include convolution matrix based operators and Hough transforms, among others, and some of the post-processing techniques used to refine results from these are edge image thresholding, edge relaxation and border tracing.

3.2.2.1 Convolution matrices

This family of filters primarily concerns itself with detecting gradients in a number of directions, and combining these filtered images to produce an edge image. Often, convolution operators are defined as 2×2, 3×3 or 5×5 matrices, which are then convolved with our image to produce a number of filtered images, which are added together to provide our final edge image.

The simplest, and earliest, of these filters is the Roberts operator [5], which is defined as

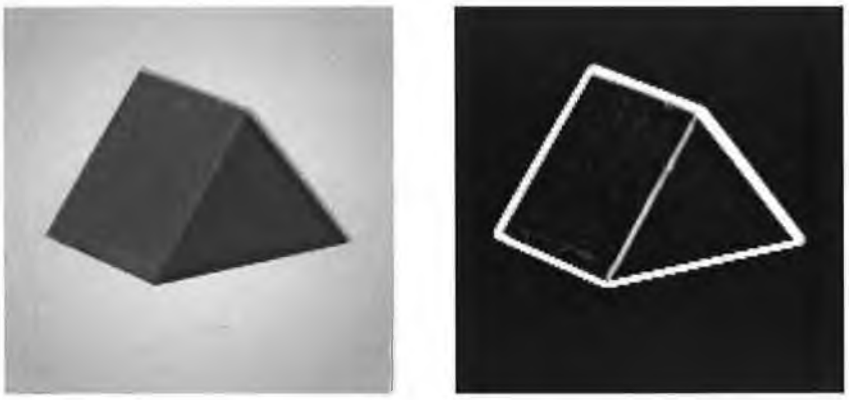
$$h_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad h_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

If our image is $A(i,j)$, we calculate

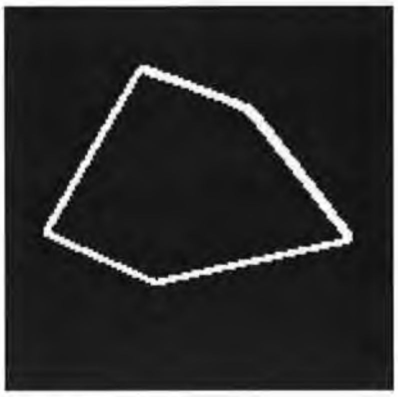
$$g_k(i,j) = \left| h_k \times \begin{bmatrix} \mathrm{A(i,j)} & \mathrm{A(i+1,j)} \\ \mathrm{A(i,j+1)} & \mathrm{A(i+1,j+1)} \end{bmatrix} \right|$$

$$g(i,j) = \sum_{k=1}^{2} g_k(i,j)$$

Because this is a small filter, it gains in execution speed and loses in fidelity. Because it only takes account of a 2×2 neighbourhood, it is very sensitive to noise.



(a) original image (b) Prewitt operator edge image



(c) Thresholded edge image

Figure 3.2: Edge detection by Prewitt operator

Among the many other operators in this class, the most popular are the Prewitt operator, the Sobel operator, and the Laplacian gradient operator. Of these, the Sobel operator and the Prewitt operator approximate the first derivative of the image in a given direction, the Laplace operator approximates the second derivative of the image.

The Prewitt operator is defined by eight convolution masks for 3×3 masks, corresponding to the image slope in the eight compass directions. It is possible to extend the mask to larger than 3×3 masks, but this reduces the fidelity of the final edge image, and increases computation time.

The first three Prewitt operators, h_1, h_2, h_3 are defined as below, and the other operators are obtained by rotation.

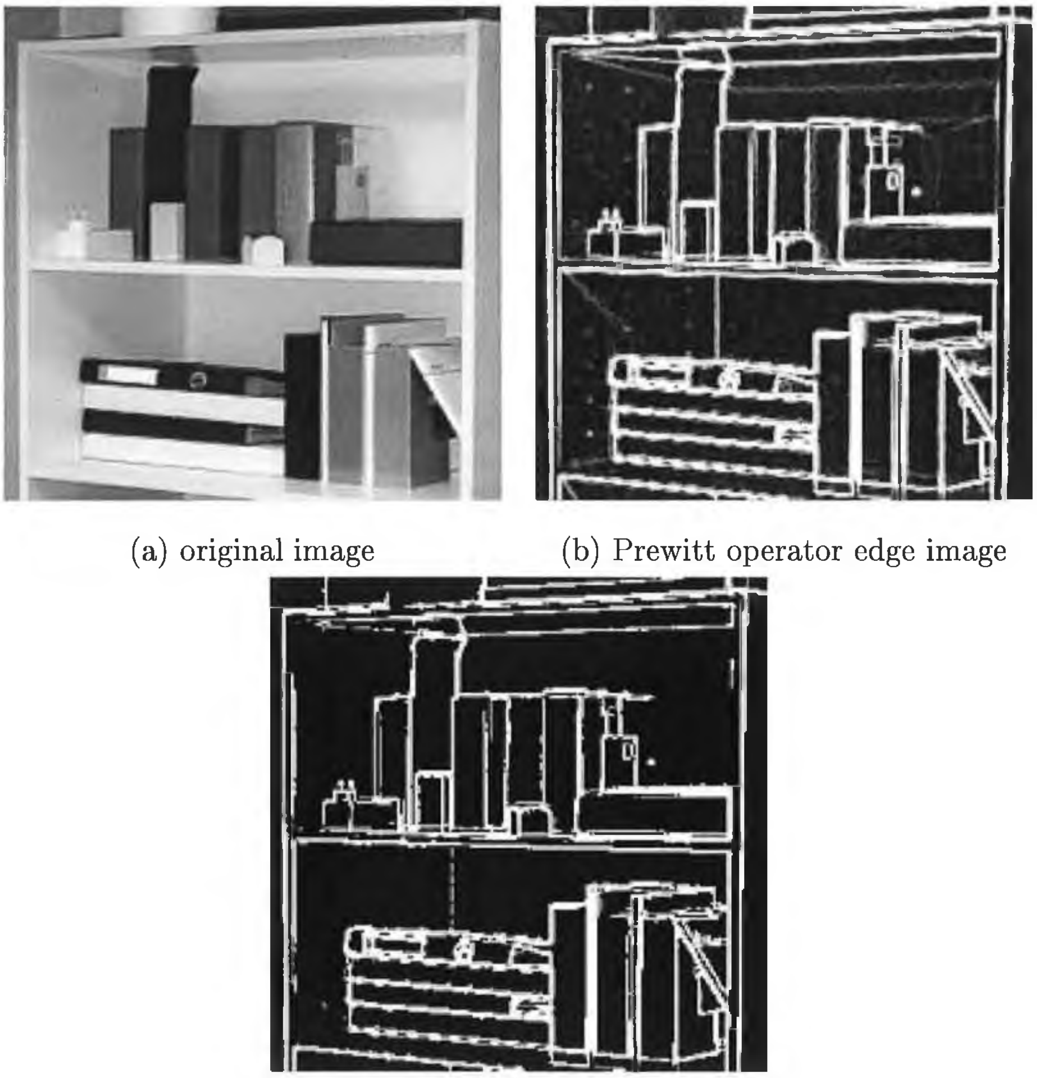
$$h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad h_2 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix} \quad h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Examples of edge detection using the Prewitt operator, first on the image in Figure (3.1, and then on a real image (Figure (3.3)) are shown in Figures (3.2, 3.3).

The Sobel operator consists of three filters, defined as

$$h_1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad h_2 = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix} \quad h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Normally, only h_1 and h_3, which can be interpreted as finding horizontal and vertical gradients respectively, are used, and as we did before, each image is



(a) original image (b) Prewitt operator edge image

(c) Thresholded edge image

Figure 3.3: Edge detection on picture of shelves

combined as

$$g(i,j) = \sum_{k=1}^{2} |g_k(i,j)|$$

$$\text{or} \quad \sqrt{\sum_{k=1}^{2} g_k(i,j)^2}$$

The Laplace operator, which is used to calculate the second derivative of the image is defined in one of two ways, one form for 4-neighbourhood calculation and another for 8-neighbourhood calculations. It is rotation invariant, and as such consists of just one filter.

$$h = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{or} \quad h = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

This family of filters perform well, particularly on images with clear, straight edges. However, on more textured images, these methods do tend to introduce a lot of noise to the edge image, possibly resulting in the false detection of edges.

3.2.2.2 Hough transform

The Hough transform was originally intended to be a method of finding straight lines in images, but the methods have evolved to take into account other more complex shapes in an image. It is often used both to detect edges, and to post-process other edge detection techniques to reduce false information in the resulting edge image.

The general principle of the Hough transform is that regular curves in image space, when parameterised, can be represented as a point in parameter space. So, for example, a straight line in an image can be parameterised as $y_i = mx_i + c$ for some m and c and for all points (y_i, x_i) on the line. And if we take every pair of edge points from our edge image and calculate the m and c which correspond to a line between those points, many more points will map to the same m and c if a line is present.

So if we discretise parameter space as a grid, and increment a counter for each time a given value of m and c result from a pair of edge points, the counter in parameter space corresponding to that line will be much larger that other counters. So the problem of finding a line in the image has been reduced to finding local maxima in parameter space.

Since we need to know beforehand what type of curve we are looking for (in order to calculate parameters from our points), the Hough transform is of limited use in general.

3.2.2.3 Edge relaxation

As we have seen, once we have an edge image, there is often a need to improve on these results to have a satisfactory segmentation. These post-processing techniques have two purposes, to remove false edges, and to complete edges which have lost information due to noise, or a weak edge. Edge relaxation techniques attempt to address both of these problems.

The method involves producing a confidence measure on how likely every edge detected pixel is to be a true edge as opposed to a false one. The probability associated with each edge considers a number of factors, including the proximity and direction of adjacent edge pixels, as well as the strength and length of the edge. For example, a weak edge between two strong edges is very likely to be a true edge, while an isolated edge, even a strong one, is likely to be false [5].

The problem is usually addressed iteratively, with an edge pixel being given a label of yes, no or maybe at each iteration, until there are no maybes left and the segmentation is clear. Gaps in edges can also be filled in by this method, which also evaluates the possibility of unmarked pixels belonging to an edge.

Edge relaxation techniques appear to work well, but are computationally expensive. Also, for several methods [17], non-convergence of confidences to zero or one is possible, resulting in a degradation of results after several iterations.

3.2.2.4 Border tracing

Once we have a complete edge image, our task is then to isolate the regions in the image for segmentation. One of the easiest and most intuitive methods of doing this is by boundary tracing [5], which is very similar to a chain-code representation of shape [18].

The methods used are to detect either the inner or outer boundary perimeters, or to calculate what's called the extended boundary [5], and to store the pixels defining the edge in a string $P_0P_1P_2\cdots P_nP_0$. For this task either 8-neighbourhoods or 4-neighbourhoods can be used. Once we have defined the boundary of a region, it is an easy task to label the region for segmentation purposes.

3.2.3 Region growing

Region growing and merging is the most popular segmentation method for the majority of image classes. It improves on edge detection for noisy images, and out-performs histogram methods in almost all situations. The method relies on the homogeneity of an image segment, in colour, grey level, texture, or any other criteria.

Methods in this class involve defining some initial method of splitting the image into a large number of small regions, and recursively merging smaller regions into larger regions by some criterion. For example, initial segmentation might be by local clustering in feature-space after the calculation of a feature vector by some of the methods described in the previous chapter. These regions could then be used as a starting point for the iterative merging of regions.

Among the methods used in this family are merging by the nearest neighbour or the k-nearest neighbour rules, other cluster analysis algorithms, boundary melting and splitting and merging. We present here an overview of some of these methods.

3.2.3.1 Nearest neighbour classifier

If we know, in advance, K, the number of regions we wish to segment from the image, we can classify the feature vectors into clusters by first setting $K1 > K$ seed points $x_1, x_2, \cdots x_{k1}$ in feature space, defining $K1$ sets $X_1, X_2, \cdots X_{K1}$ and a metric $d(.,.)$ such that $x_i \in X_i$ and, for any point x in feature space, $d(x, X_i) = d_i$ and $x \in X_k$ where $d_k = \min_{i=1}^{K1}\{d_i\}$.

We then recursively process all points until $x \in \bigcup_{i=1}^{K1}\{X_i\}$ for all x. The metric is normally defined as the distance from the point to the centroid of the set being analysed.

Once all the points have been assigned to the set, we recursively reduce the number of sets by taking the smallest set X_n and assigning $X_k = X_n \cup X_k$

where $d_k = \min_{i \neq n}\{d(X_n, X_i)\}$. We have thus reduced the number of sets by one. When we have exactly K sets left, our segmentation is complete, and we map our feature points from feature space back to the points in image space which produced them to produce our final segmented image.

However, it is rare for the number of distinct regions in an image to be known before processing, so we need a way of estimating the optimum number of regions for classification. One method which provides such an estimate is the *k-means* method [5]. Another such method is *Akaike's i information criterion* [19].

3.2.3.2 k-nearest neighbour classifier

This method is similar to the last method, except that a number of training samples are used, rather than feature domain clustering, to designate the feature-domain classification.

For each of the N training points, the mean feature vector is calculated for each class. Then, for an unknown point, its k nearest neighbours in the training samples are computed, and the new point is allocated to a class according to the classification of these neighbour points [20].

3.3 Segmentation by texture using fractal dimension

We will examine and compare existing methods of segmenting textured image, based on fractal dimension estimators.

We will use the fractal dimension estimators described in the previous chapter, the Fourier power spectrum estimator and the box-counting dimension, to build up a feature vector for our textured image, as proposed by Chaudhuri and Sarkar [20, 13, 12] with the box-counting dimension, and by Pentland et. al. [2, 11] for the Fourier estimator.

The features we will use are the differential box-counting dimension, the multi-fractal dimension [12] and the Fourier power spectrum estimator [2] measured over a moving window.

Therefore, for an image A, our features will be

$$\begin{aligned}
F_1(i,j) &= 3 - D_B\{A(i+k, j+l) : -W \leq k,l \leq W\} && (3.1)\\
F_2(i,j) &= 2 - D_M(2) && (3.2)\\
F_3(i,j) &= 3 - D_F\{A(i+k, j+l) : -W \leq k,l \leq W\} && (3.3)\\
F_4(i,j) &= \frac{1}{(2W+1)^2} \sum_{k,l=-W}^{W} A(i+k, j+l) && (3.4)
\end{aligned}$$

where $D_M(2)$ is the multifractal dimension, defined by

$$D_M(2) = \lim_{r \to 0} \frac{\ln \sum_x [n'_r]^2}{\ln r}$$

where n'_r is $\frac{n_r}{N_r}$, with n_r being the number of boxes centered on (i,j) of radius r required to cover the image divided by the number of boxes of that size required to cover the whole image A.

In reality, we will take a log-log relationship of r against $\sum_x [n'_r]^2$ for a number of values of r, and estimate the slope of the graph.

Once we have our feature vector $F_n(i,j) : n = 1,2,3$ for each pixel $A(i,j)$, we will use the edge-preserving noise smoothing quadrant filter (EPNSQ)

[20, 21] to smooth each of the filter images before deriving seed points for clustering from analysis of each feature image histogram.

We will also examine variations over scales, and see how these variations fare in segmentation. That is to say, we will define features by

$$F_i(i,j) = \frac{\log(N_r)}{\log r}$$

where $r = s/(i+1)$, and s the maximum box side in image space. This is a variation on the "fractal signature" method used to characterise shape, as well as texture [1].

After smoothing the feature domain, we will seed feature space in exactly the same way as Chaudhuri and Sarkar [20]. The seeding of feature space involves finding points which we believe are in a specific region, and then using these seed points to find the other points belonging in that region. For each histogram $H(F_i)$, we will identify the set of dominant local maxima $M_i = \{m_{i,1}, m_{i,2}, ..., m_{i,n_i}\}$, and seed feature space with the cross product of the sets of histogram peaks for each feature. For example, assuming we have two features F_1 and F_2 with maxima $M_1 = \{m_{1,1}, m_{1,2}\}$ and $M_2 = \{m_{2,1}, m_{2,2}, m_{2,3}\}$, we will seed feature space with 6 points, namely $M_1 \times M_2 = \{(m_{1,1}, m_{2,1}), (m_{1,1}, m_{2,2}), \cdots, (m_{1,2}, m_{2,3})\}$.

We will then assign each point to a set by the nearest-neighbour rule, and merge the regions together until we have a predetermined number of segmented regions.

3.4 Segmentation using fractal codes

A method, previously proposed [22], will be used to isolate regions with only one feature being used, that being the basins of attraction resulting from the affine maps generated by a fractal coder. These maps will form a dynamical system which, at least intuitively, results in a logical partitioning of an image into regions of similar fractal characteristics. That is to say that if two parts of an image are similar on different scales, the relevant points will end up in the same basin of attraction, and be classified as being in the same region.

An in-depth introduction to fractal coding will be covered in section 4.4. We will not preempt that discussion, but the definition of a number of terms, and the explanation of some ideas are necessary for the understanding of this section. A *partitioned iterated function system* (PIFS) is a set of maps from image space to itself. The general principle of a fractal coder is that under iteration, a PIFS can be found which approximates any image A, according to the collage theorem [23].

The mappings involved in a PIFS are of the form

$$f_i \begin{pmatrix} x \\ y \end{pmatrix} = r_i M_i \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} s_i \\ t_i \end{pmatrix}$$

[23]. M_i can be any affine map of the form $I = \begin{pmatrix} \cos\theta & -\sin\phi \\ \sin\theta & \cos\phi \end{pmatrix}$, but that normally we will only consider the isometries of the square.. These are maps from the image to itself, which include a contraction, a spatial offset, and an affine mapping, normally one of the isometries of the square. There is also an affine mapping of the grey-scale values in the block of the form $g(\mu(x,y)) = \alpha\mu(x,y) + \beta$, where $\mu(x,y)$ is the grey-scale value at point (x,y)

in the image, and $\alpha, \beta \in \mathbb{R}$.

Upon coding an image, we have a set of maps $M = \bigcup_{R_i \in A} (f_i, v_i)$, where A is our image, and $\{R_i\}$ is the set of range blocks in our coding procedure.

If one assumes, and it is logical to do so, that domain blocks map onto range blocks in similar regions (that is, edge blocks map onto edge blocks, interior blocks map onto interior blocks and exterior blocks map onto the exterior of a region), then we can invert this process, map smaller blocks onto larger blocks, and reach a steady state eventually, in which a small number of points characterise a region completely. See Figure 3.4 for a visual explanation.

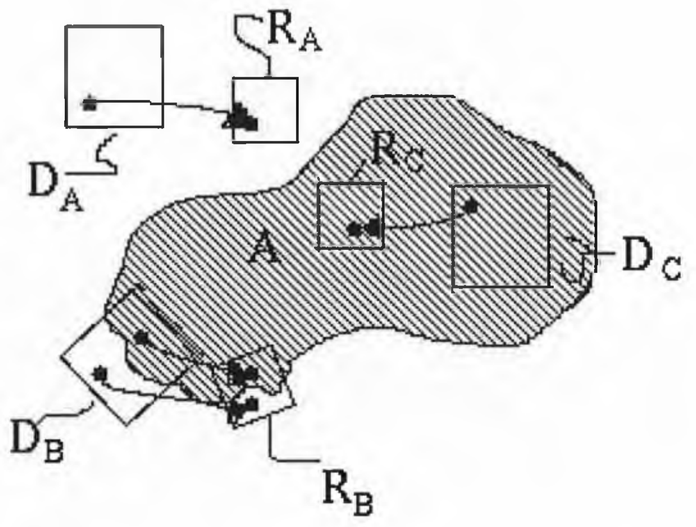


Figure 3.4: Examples of how we assume blocks map onto each other

As can intuitively be seen from the diagram, points in a region not only remain within the region, but tend towards an attractor, a small subset of the basin of attraction. To see the intuitive merit of this, observe that points of distance ϵ from the boundary of the region map onto a point of distance δ from the boundary, where $\delta > \epsilon$ for points near the boundary.

To segment the image, if this is true, we need only find the attractors for each region, and find out which points get mapped onto them. This is the basis for our segmentation algorithm.

Mathematically, we define a map $M(x, y)$ such that

$$M(x, y) = f_i^{-1}(x, y) \text{ when } (x, y) \in D_i$$

This map M maps our image I to itself, since $f_i(D_i) \in I \; \forall i$. M is determined purely by our fractal codes, and the inverses are trivial to calculate.

$$\begin{aligned}
f_i(v) &= Av + b \\
\Rightarrow f_i^{-1}(v) &= A^{-1}(v - b) \\
\text{and } A^{-1} &= \frac{1}{|A|} A^a \\
\text{with } A^a &= \begin{pmatrix} r_i \cos\phi_i & r_i \sin\phi_i \\ -r_i \sin\theta_i & r_i \cos\theta_i \end{pmatrix} \\
\text{and } |A| &= r_i^2 \cos(\theta_i - \phi_i), \\
\text{so } f_i^{-1}\begin{pmatrix} x \\ y \end{pmatrix} &= \frac{1}{r_i(\cos(\theta_i - \phi_i))} \begin{pmatrix} \cos\phi_i & \sin\phi_i \\ -\sin\theta_i & \cos\theta_i \end{pmatrix} \left(\begin{pmatrix} x \\ y \end{pmatrix} - \begin{pmatrix} s_i \\ t_i \end{pmatrix} \right)
\end{aligned}$$

By iterating M, we approach the limit set $L = \lim_{n \leftarrow \infty} M^n(I)$, but since I is a discrete set, and we are working in a discrete space, $M^n(I) = M^m(I)$ for $n > k$ and $m > n$, where k is finite, and less than 300, as found by experiment.

The segmentation algorithm to be implemented finds this limit set L, and classifies each of the stable cycles in this set.

3.5 Results and discussion

3.5.1 Fractal dimension segmentation results

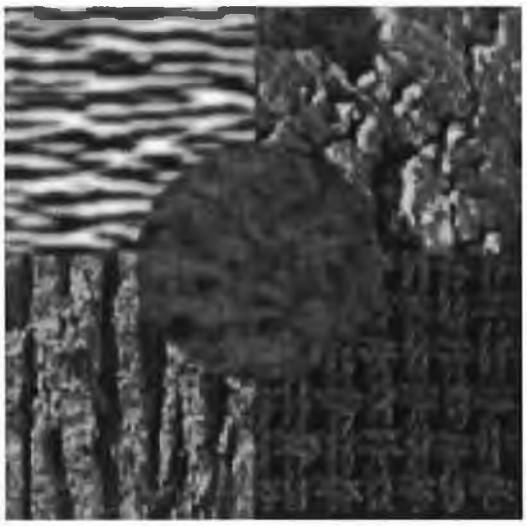


Figure 3.5: Test image containing five textures for segmentation

As test images we used a 256×256 picture of Lenna (Figure 3.6) and a 5-texture mosaic made up of some of the textures in Figure 2.1 (Figure 3.5). For all examples we used 17×17 or 19×19 windows for feature calculation, and 7×7 or 13×13 for feature domain smoothing.

To detect dominant peaks in the histograms of the feature domain images, we first smooth the histogram by replacing $H(i)$ with $(H(i-1)+H(i)+H(i+1)/3$. We then calculate the histogram gradient $\Delta H(i) = H(i) - H(i-1)$, and the second derivative $\Delta^2 H$.

A value s_i is then classified as a dominant peak if

1. $H(s_i) > \frac{\max(H)}{3}$

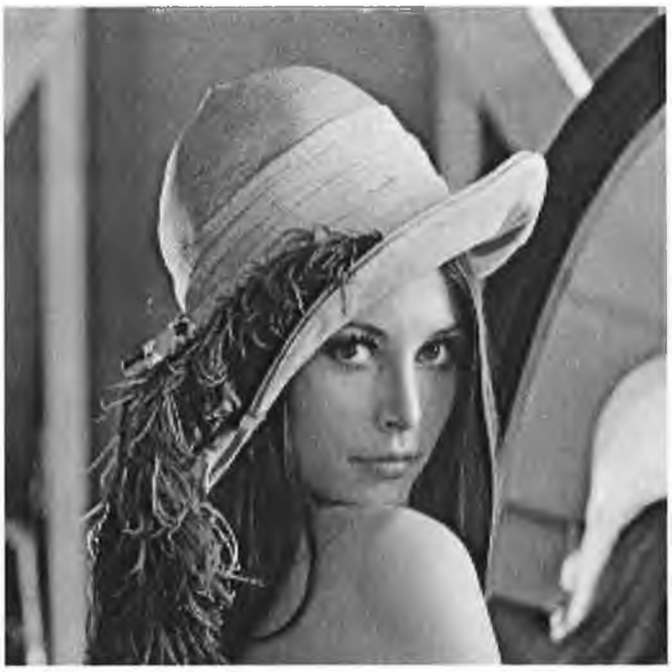


Figure 3.6: "Lenna" test image used for segmentation

2. $\Delta H(s_i) = 0$ or $\Delta H(s_i) > 0$ and $\Delta H(s_i + 1) < 0$
3. $\Delta^2 H(s_i) < 0$
4. $H(s_i) - H(s_{i-1}) > \epsilon$ where ϵ is a value preset to minimise false maxima from the same peak, normally around $1/30^{\text{th}}$ of the histogram span.

Rules 2 and 3 ensure that we have a local maximum, rule 1 guarantees that the peak is dominant, and rule 4 is, as previously stated, an effort to eliminate multiple maxima from the same histogram peak.

As described earlier, we have examined two sets of features for classification. First we use the differential box-counting dimension [20], the Fourier estimator [11], the multi-fractal measure of order 2 [12], and a local averaging filter. Secondly, we use a method derived from the description of the fractal signature [24, 1], using the estimates of the box-counting dimension over a number of scales as our feature vector for classification. We use box-sizes between 4 and 8, resulting in 5 features. We will call this method the fractal signature segmentation method.

For the montage image, we segment the image into 5 regions, and for Lenna we attempt to segment into 32 regions, which should be sufficient to demonstrate the properties of the segmentation algorithm.

Figure 3.7: Segmentation of montage with 13x13 feature smoothing and 17x17 feature estimation window

Initial segmentation results were slightly disappointing, but a closer look at the feature images helps to explain the reasons. In particular, the histograms of the fractal based features show considerable bunching between 0.2 and 0.5 times the dynamic range (see Figures 3.14, 3.15, 3.16 and 3.17), which make the decision on optimum seed points extremely difficult.

Also, it is worth noticing that textures which were more homogeneous, for ex-

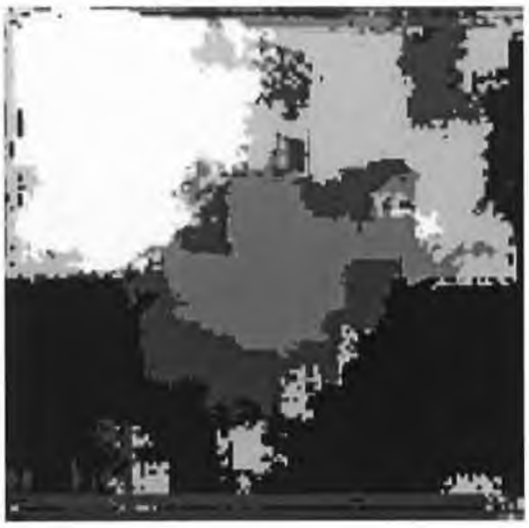


Figure 3.8: Segmentation of montage with 7x7 feature smoothing and 17x17 feature estimation window

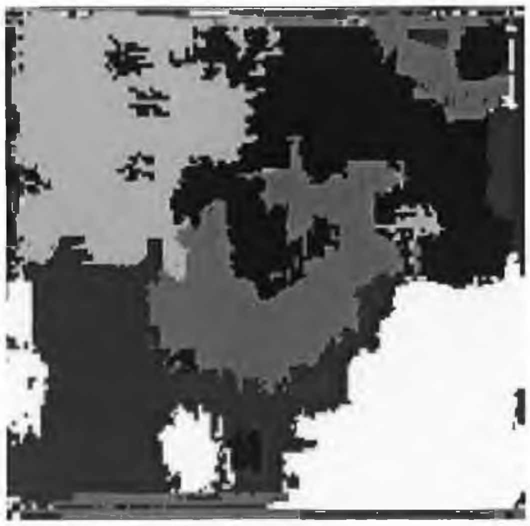


Figure 3.9: Segmentation of montage with 7x7 feature smoothing and 19x19 feature estimation window

ample burlap, tended to provide more consistent regions than non-homogeneous textures, such as mud, and more entropic textures, such as walnut. The reasons for this, I believe, are that in the burlap section, the fractal dimension was higher than it should have been, particularly in the Fourier measure, and distinctly higher than the other features, resulting in a consistent cluster in feature space. However, it is evident that among the other features (Figures 3.10, 3.11, 3.12 and 3.13) both the narrowness of the range of the fractal measures and the range of values within each region result in considerable mixing in feature space, thus resulting in less accurate segmentation.

Figure 3.10: Feature image generated by box-counting dimension

It is also fairly obvious that the feature with the greatest dynamic range, and the greatest contrast between regions, is feature four, the simple averaging filter. This feature, especially in the context of the burlap and dunes sections, dominates feature space.

The methods used to find histogram peaks, and also the clustering methods, are open to question. However, there are clearly flaws in using the box counting dimension to segment texture, namely the narrow dynamic range of values on real textures, and the tendency to overestimate dimension on

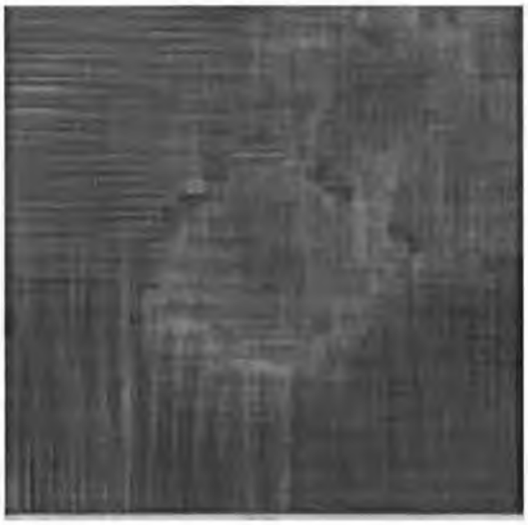


Figure 3.11: Feature image generated by the multifractal dimension

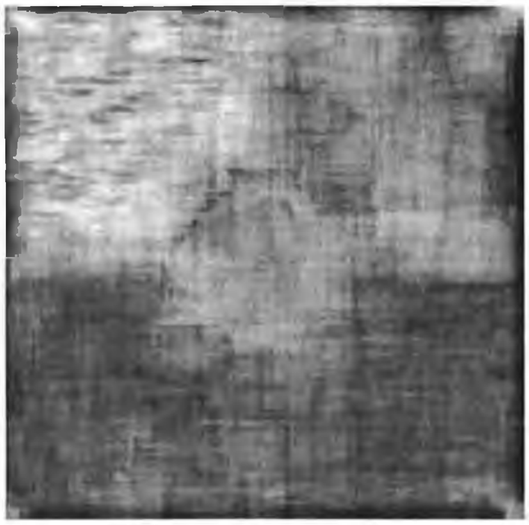


Figure 3.12: Feature image generated by Fourier estimator

sharp edges.

The attempt at segmentation of the Lenna test image also shows up several

Figure 3.13: Feature image generated by smoothing filter

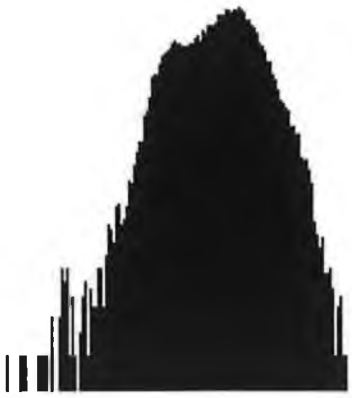


Figure 3.14: Histogram for feature 1 (Figure 3.10)

flaws in the fractal dimension measure. The principle among these is the necessity for a relatively large smaple of pixels to calculate the measure. Since images such as Lenna typically contain many smaller regions of interest, a box of 17×17 will almost certainly overlap two or more regions, making the fractal dimension inconsistent, even within a region.

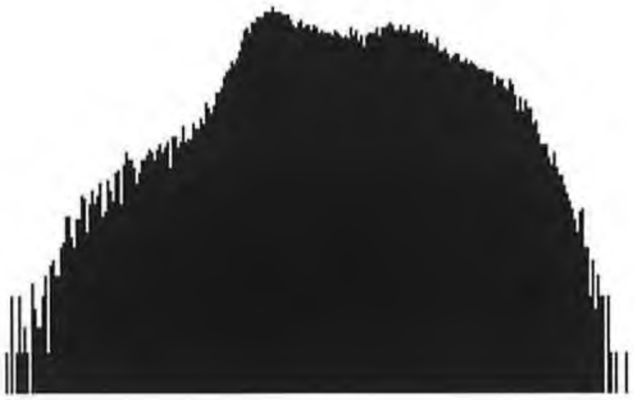


Figure 3.15: Histogram for feature 2 (Figure 3.11)

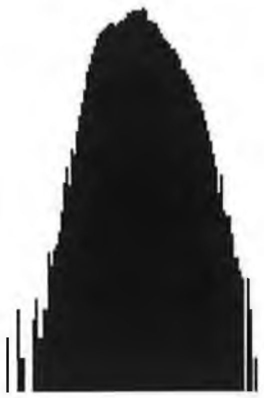


Figure 3.16: Histogram for feature 3 (Figure 3.12)

The resulting attempt at segmentation bears little resemblance to the original, since many of the smaller regions of interest were eliminated and merged with larger ones long before the algorithm finished, and the considerable misclassification near edges makes regions barely identifiable.

Our attempts to segment by the fractal signature method was similarly unconvincing. Here the high correlation between the features, and the narrow range of values were again the main problems.

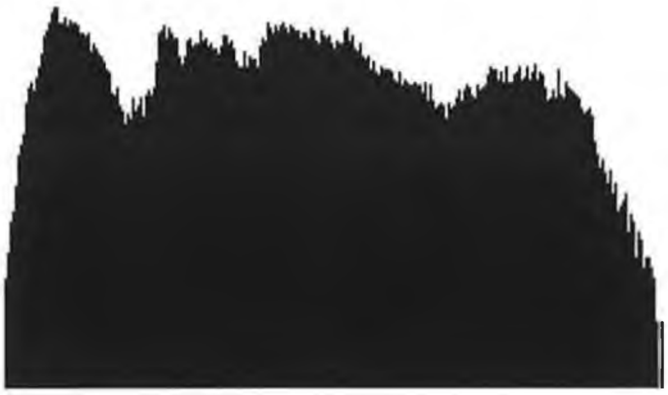


Figure 3.17: Histogram for feature 4 (Figure 3.13)

Figure 3.18: Segmentation of "Lenna" with 13x13 feature smoothing and 17x17 feature estimation window

Having said all that, however, the feature provided by the Fourier estimator, in particular, shows promise, and if used with features which have a lower correlation with itself, it may be able to contribute to a good segmentation algorithm.

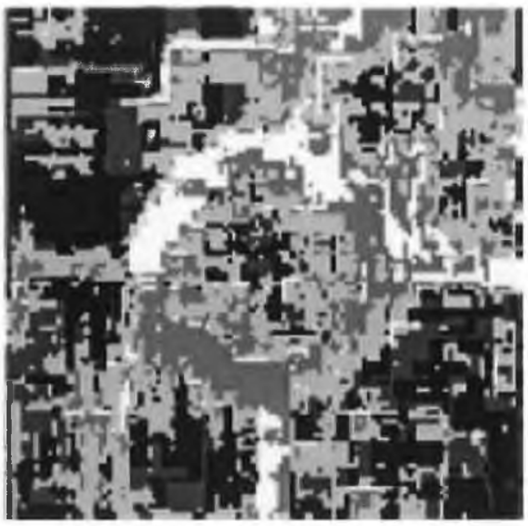


Figure 3.19: Segmentation of texture montage by fractal signature method using 17×17 estimation boxes

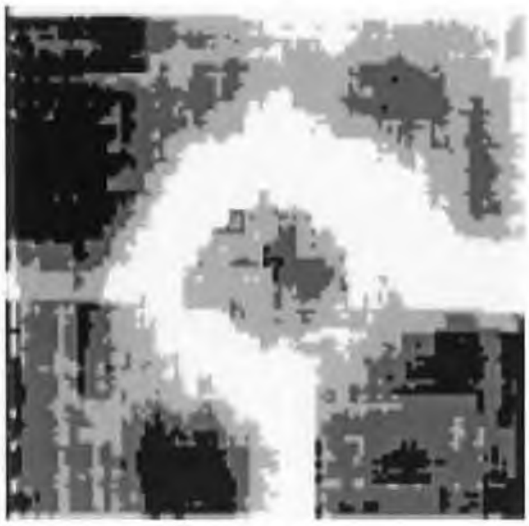


Figure 3.20: Segmentation of texture montage by fractal signature method using 25×25 estimation boxes

3.5.2 Fractal codes segmentation results

Three types of mappings were used in our experiments to verify whether mapping do occur in this fashion.

First, we constrained the spatial offset, s_i, t_i, but we left α_i unconstrained. We constrain the offset to the range $s_i, t_i \in \{-16, \cdots, 16\}$, so that effectively, the range block is a subset of the domain block for all i.

Second, we constrain $\alpha_i \in \{0.4, 0.6\}$, and allow unconstrained offset. This is because in mapping from 16×16 blocks to 8×8 blocks, we can expect the variance of grey-levels in the block to contract by $\frac{1}{2}$ as well. We will show that varying this scaling factor substantially can change the fractal dimension of the block, substantially changing the characteristics we wish to keep to preserve our assumption of blocks mapping within a region.

To illustrate the point, we will demonstrate with the box-counting dimension, as presented in chapter two.

Given a domain block, D, and a range block onto which its mapped, R, by a spatial scaling factor of s, and a greyscale scaling factor of α, and taking $N_\delta(S)$ as the number of cubes of side δ required to cover the surface, we have

$$R = sI(\alpha(D) + o) + t$$

where I is an isometry, and o and t are greyscale and spatial offsets respectively. This gives

$$N_\delta(R) \simeq sd^2 N_\delta(D) \tag{3.5}$$

and since the box-counting dimension is defined as

$$D_B(S) = \lim_{\delta \to 0} \frac{\log N_\delta(S)}{-\log \delta}$$

we have

$$\begin{aligned} D_B(R) &= \lim_{\delta \to 0} \frac{\log N_\delta(D) + 2\log s + \log \alpha}{-\log \delta_R} && (3.6) \\ &= D_B(D) && (3.7) \end{aligned}$$

under normal circumstances. This is the result we expect.

However, in practice we do not take a limit, we merely correlate the interior of the limit $(\frac{\log N_\delta(S)}{-\log \delta})$ over a number of δ, and take a best fit line. For a 16×16 box, this will probably be limited to 4 or 5 values. Thus we have a situation where δ takes a number of values between 0.125 and 0.5, while s is normally constant at 0.5.

At these scales, the line of best fit is skewed substantially in the range block, resulting in an estimate of dimension for the range block which is lower than the domain block for values of α lower than 0.5, and higher for α greater than 0.5. To allow for an acceptable domain pool, however, we set 0.4 and 0.6 as reasonable lower and upper bounds for the greyscale scaling ratio.

Finally, as was done by Ida and Sambonsugi [22], we constrain both spatial offsets, and greyscale scaling.

In the first example, we performed perfect classification, that is, every limit point was classified in a different region. In the second and third example, we treated the attractor as a two dimensional feature space, and used nearest neighbour clustering, as described earlier in the chapter, to segment the image.

By finding the attractor of the maps, we can classify each point in the skeleton as belonging to a distinct orbit of values. These orbits define the initial clustering we use in the classification of regions. We determine the centroids of each distinct orbit, which describes n distinct regions corresponding to the number of distinct orbits in the skeleton.

Once we have classified the skeleton points into regions, we calculate, for each point (i,j) in our original image A, $M^n(A(i,j))$, and terminate when $M^n(A(i,j))$ is an element of the skeleton. We then assign the value of the skeleton element to $S(i,j)$, the $(i,j)^{\text{th}}$ element of the segmented image.

The centroid values of each region are then calculated (using (i,j) as the feature vector), and regions are merged according to the nearest neighbour method defined earlier, until we have arrived at the desired number of regions.

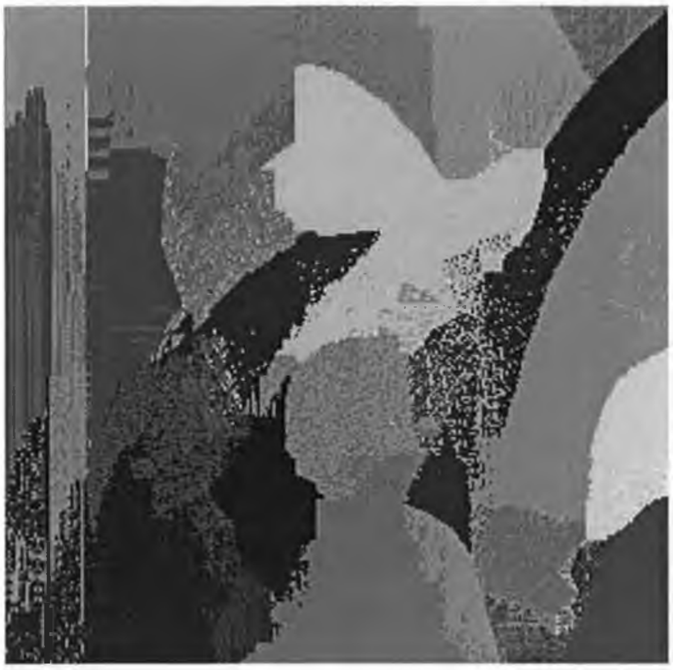


Figure 3.21: Segmentation of "Lenna" by fractal codes with α and offset constrained

As can be seen from the first Lenna example, and the segmentation of the texture mosaic, this method can produce excellent results, once we tightly constrain conditions on the fractal coding associated with the method. The most crucial factor without a doubt is the greylevel contraction factor, which provides good segmentation when it is constrained, but we find that most

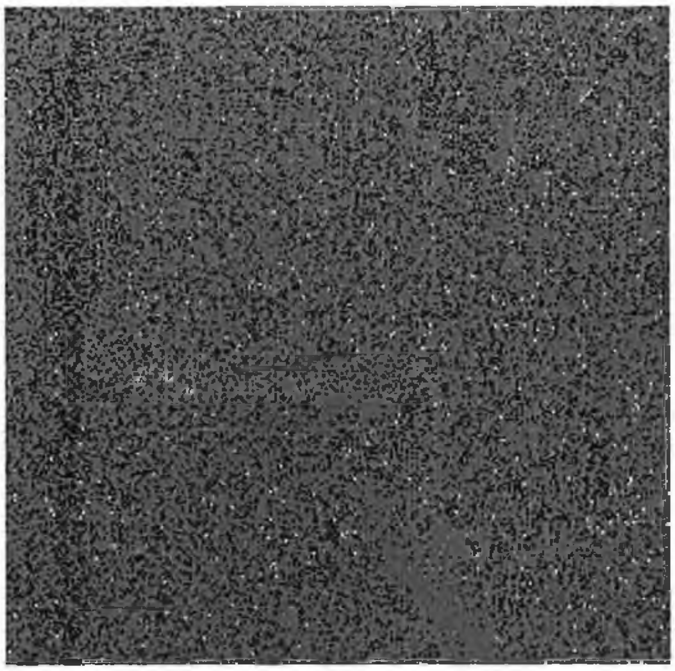


Figure 3.22: Segmentation of "Lenna" with constrained α and unconstrained offset

blocks tend to almost identical domain blocks, resulting in an almost uniform segmentation image, if an unconstrained α is used at the coding stage.

There is also considerable misclassification at the boundary of the circular section of the texture montage. I believe that this is due to the block-based nature of the coder, which may have some difficulty matching straight edges. However, the segmentation along the straight horizontal and vertical edges (which are, coincidentally, also bounding edges of range blocks) is almost perfect.

This approach, while being conceptually attractive and intuitively sensible, fails to achieve acceptable segmentation in general, for a number of reasons.

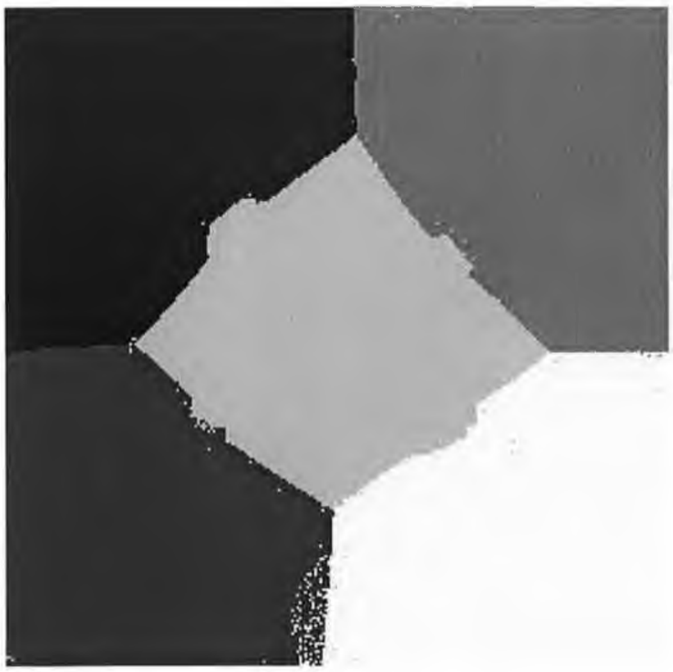


Figure 3.23: Segmentation of texture montage with constrained α and unconstrained offset

When the mappings are not constrained to the immediately local area, we end up with disparate and unrelated regions being mapped onto each other, since this introduces the possibility of regions being matched when, for example, one is light and the other is dark.

A larger domain pool introduces a large number of diverging orbits, as a result of points which are matched to each other from different regions. These orbits propagate throughout the image, leaving only small, and almost indistinguishable, basins of attraction for quite large regions.

Also, when the contraction factor of the greyscale is outside a very small range (about $0.4 \leq \alpha \leq 0.6$ for a spatial contraction of 0.5), we find regions with completely different characteristics being mapped. Since we are dealing with discrete image space, we find that areas we would call rough, with high conrast in a small area, can be mapped onto smooth areas with the mapping having a low value of α.

This is a flaw in the initial logic of the segmentation system. Under an affine map in continuous space, fractal dimension (the characteristic we assume marks out regions under this scheme) is conserved. However, in discrete space, when we are dealing with images of sizes 16×16 and 8×8, fractal dimension is not conserved, when the greyscale scaling factor is unconstrained.

We have shown that this is true for the box-counting dimension estimator. Also, it is worth pointing out that discrete objects, such as images with finite resolution, are not strictly fractal at all, since there is a minimum scale beyond which they are constant. It stands to reason, then, that scaling does have an effect on the apparrent dimension of the image, or image segment, for other estimators.

This method shows considerable promise, however, and attempts to improve the orbit classifications, as well as attempts to provide better fractal codes for the generation of the dynamical system, could produce substantially better results.

Chapter 4

Fractal methods in image coding

4.1 Introduction

By image coding we mean the compression of images for the purposes of improving storage efficiency or increasing transmission speed. To demonstrate the need for compression, it is probably easiest to give a case in point.

Considering that the storage size of an image of dimensions 512×512 in 24-bit colour is $512 \times 512 \times 24$ bits, or 768 kilobytes, one second of film, assuming a frequency of 50 Hz, contains 38400 kilobytes. At that rate, the entire storage size of, say, "2001: A Space Odyssey" would be 124 minutes $\times$ 60 seconds $\times$ 38400 kilobytes, or 285 gigabytes. Note that this is merely image data, and does not take account for the sound-track data which would also have to be transmitted.Obviously this kind of storage space is not available.

In terms of transmission, the top of the range modems available at the moment support data transmission at the rate of 96 Kb per second. Which means that to send our complete, uncompressed copy of the movie 2001 (stored as a series of frames), we would need to maintain a constant, fully saturated connection with our modem for 2976000 seconds, or 49600 minutes, over 826 hours. At current Eircom phone charge rates, that single phone call would result in a charge of IR£357.12.

A DVD disk can only hold up to 8 gigabytes of information, and entire films are stored on them using the MPEG-2 coding standard [25], compression factors of 40:1 or more are required on the image data. This compression ratio is obtained without noticeable loss of image quality through a number of measures, including motion estimation and lossy compression of reference frames.

In this chapter we will concentrate on fractal methods of coding still images, and we will mention how this relates to coding of video segments. We will also explore other methods of image coding, including transform methods and vector quantisation methods.

Finally, we will present a new approach to the computation of the fractal transform, and examine the success of this method in comparison to other popular methods. We also explore the relationship between the fractal transform and the wavelet transform, which has been made famous in recent years through its adoption by the FBI for the coding of its fingerprint database [26].

4.2 Image coding methods

4.2.1 Lossless compression

Several good lossless compression algorithms exist, and although most of them are more used with data streams than images, they are worth mentioning here for reference. In fact, some of these methods are used after lossy compression to further reduce the information needed to store the image.

Possibly the best known of the lossless compression algorithms is Shannon-Fano coding [23]. The general principle used in Shannon-Fano coding is that if there are n symbols used in the string to be coded, and each appears with a probability P_n, then the higher P_i, the smaller the number of bits which will be used to code it. That is, the more common a letter is, less space is required to store it.

For example, ASCII coding is an extremely poor coding scheme in this respect, since every character uses the same number of bits, 7 for character data or 8 for binary data. A Shannon-Fano scheme might, for example, only use 3 bits for the letter 'e', since in written English it is much more probable than the letter 'q' or the punctuation mark ';', for example.

Perhaps a simple example would be the best way to demonstrate. Assume we are dealing with an alphabet of size 4, and we wish to encode a 16-character string. Say our alphabet is $\{A, B, C, D\}$, and the string we wish to encode is $ABACBBAAADABDACA$. We can then assign the following probabilities to each letter.

Letter	Probability
A	8/16
B	4/16
C	2/16
D	2/16

Under standard ASCII storage, a 4-letter alphabet would be assigned 2 bits per letter, resulting in a storage space of 32 bits. This is equivalent to a probability of 1/4 being associated with each letter. Under the Shannon-Fano scheme, however, we combine the probabilities of the individual letters until they are close to, or equal to, 1/2. Them we assign one bit to one branch, and the other bit (at that level) to the other branch. We then repeat for the sub-branches until we have assigned a unique value to each character.

So our coding scheme in this case would end up as shown in the table below. And the final storage space occupied by our 16-character string is $8 \times 1 + 4 \times 2 + 2 \times 3 + 2 \times 3$, which is 28 bits, a saving of 12.5% over normal coding.

$A(8/16)$	0		
$B(4/16)$	1	0	
$C(2/16)$	1	1	0
$D(2/16)$	1	1	1

Obviously, this is a very simple example. In fact, substantial compression ratios are achieved with Shannon-Fano coding for larger sets of symbols. It has, however, been shown that Shannon codes do not produce optimal codes for a given set of symbols in general [23].

Huffman coding, which is a variant on Shannon-Fano coding, provides optimal codes for symbol sets whose probabilities are all an integral power of $\frac{1}{2}$, as is the case above. The only difference between the algorithms is that, whereas in Shannon codes the division of probabilities to make up 1/2, or as close to 1/2 as possible, is arbitrary, with Huffman codes, symbols are combined in order of probability.

The symbol table is listed in order of probability, the two symbols with lowest probability are combined to form a composite symbol, marking one with 0 and the other with 1, and the process is repeated until a complete binary tree is achieved. In the case above, the resulting coding is exactly the same.

A mainstay of many compression algorithms used for computer data files, including `compress`, `gzip` and `pkzip` is the Lempel-Zif coding algorithm. There are several variants on this algorithm, known as LZ77, LZ78, LZH and LZB, but the general principle is the same.

This method stores a buffer of the last n characters encountered, and a buffer to look ahead in the file. It then attempts to match the longest string which matches one of the characters or phrases in the buffer, and outputs the initial phrase, the length of the phrase and the position of the match in the file.

Variants of this method apply other coding algorithms, such as variable-length coding or Huffman coding, to the output to produce further compression.

4.2.2 Transform methods

Transform coding methods involve transforming the image from discrete image space to a transform space, and then applying some kind of meaningful filter to the transformed data. The best known transform methods are the Fourier transform and its variants, the Hadamard transform and the wavelet transform.

The general form of any transform method involves pre-multiplying and post-multiplying the original image data by matrices of the form

$$F = P \times f \times Q \tag{4.1}$$

where f is an image of dimension $m \times n$, P is of size $n \times n$ and Q is a matrix of dimension $m \times m$. The transform is invertible if matrices P^{-1} and Q^{-1} exist, giving

$$f = P^{-1} \times F \times Q^{-1} \tag{4.2}$$

Once the transform has been computed, there are a number of ways we can compress this data without substantial loss of image quality. Among these are the various entropy coders, such as arithmetic coding, Huffman coding and Lempel-Zif coding, as mentioned in the previous sections.

We can also use various feature domain filters to remove data which is likely to be superfluous in the transform domain. Among the methods used for this are low-pass filters and quantization of transform domain coefficients.

A low-pass filter is merely a hard thresholding of all coefficients above a certain frequency in the transform domain, while coefficient quantization normally uses bandpass filters to assign different quantization levels according to the frequency (and thus the relative importance) of the coefficients.

4.2.2.1 Fourier transform

The Fourier transform is normally interpreted as a mapping from spatial to frequency based domains. It is defined, for continuous functions, as

$$\mathcal{F}(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-2\pi i(xu+yv)}dx\ dy$$

However, since digital images do not reside in continuous 2-D space, we must use a discrete transform, which is directly analogous to the continuous transform. This is defined as

$$\mathcal{F}(u,v) = M_{n,n} \times f(x,y) \times M_{m,m}$$

where

$$M_{n,n}(x,y) = \frac{1}{i}e^{-2\pi ixy/n}$$

and f is an $m \times n$ image, as described in equation 4.1. The inverse transform is defined by the relation

$$M_{n,n}^{-1}(u,v) = e^{2\pi ikl/n}$$

and equation 4.2.

So the forward and inverse discrete Fourier transforms, written in longhand, are as follows, with an obvious analogy to the continuous transform.

$$\mathcal{F}(u,v) = \frac{1}{mn}\sum_{a=0}^{n-1}\sum_{b=0}^{m-1} f(a,b)e^{-2\pi i(au/n+bv/m)} \tag{4.3}$$

$$f(u,v) = \sum_{a=0}^{n-1}\sum_{b=0}^{m-1} \mathcal{F}(a,b)e^{2\pi i(au/n+bv/m)} \tag{4.4}$$

where $u = 0, 1, \cdots, n-1$ and $v = 0, 1, \cdots, m-1$.

Many variants of this method exist, but the best known and most popular is the *discrete cosine transform* (DCT), which uses only the cosine part of the exponential factor. This is the variant used in JPEG coding standard [1].

4.2.2.2 Hadamard transform

Where the Fourier transform has as its basis sinusoidal wave-forms, the Hadamard transform has square waves, or more accurately Walsh functions. These functions consist entirely of the values ± 1 [27, 5].

The Hadamard transform consists of using Hadamard matrices of size $m \times m$ and $n \times n$ in equation 4.1, instead of P and Q.

The Hadamard matrix itself is defined (for orders $2^n \times 2^n$) recursively, as

$$\begin{aligned} H_{2,2} &= \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\ H_{2k,2k} &= \begin{bmatrix} H_{k,k} & H_{k,k} \\ H_{k,k} & -H_{k,k} \end{bmatrix} \end{aligned}$$

The calculation of the inverse Hadamard matrices is a simple task, and with a little effort it can be shown that

$$H_{k,k}^{-1} = \frac{1}{k} H_{k,k}$$

Because only additions are calculated during the matrix multiplication in the transform, and because there are no trigonometric or complex terms to consider, the Hadamard transform is both fast and easy to calculate.

4.3 Wavelet transform

We will address the wavelet transform separately from the other transform methods, since it acts somewhat differently in the discrete case than the

4.3.2 Frames and orthonormal bases

Fourier analysis is based on the fact that for the space $L^2(\mathbb{R})$ of square integrable functions, the functions $\{exp(2\pi inx)\}$ form an orthonormal basis under the norm $\langle f, g\rangle = \int_{-\infty}^{\infty} f(x)\overline{g(x)}\,dx$. This means that any function $f \in L^2(\mathbb{R})$ can be expressed as

$$f(x) = \sum_{n=-\infty}^{\infty} \langle f(x), e^{2\pi inx}\rangle e^{2\pi inx}$$

In general, however, it is not easy to satisfy the myriad of conditions necessary to easily find an orthonormal basis for an arbitrary space. We must be able to easily find the basis, easily calculate the coefficients, and so on. However, for wavelet theory we use a slightly less stringent criterion than orthonormality. We will use instead the concept of a *frame* [30]. A set $\{x_n\}$ is a frame if, for all $x \in H$, there exist numbers $0 < A \leq B < \infty \in \mathbb{R}$ such that

$$A\|x\| \leq \sum_n |\langle x, x_n\rangle|^2 \leq B\|x\|$$

We call a frame *tight* if $A = B$, and *exact* if it ceases to be a frame upon the removal of any element of the frame.

The goal is, for a given frame $\{x_n\}$, bounded by A, B, to find a means of calculating $\{c_n\}$ such that $x = \sum_n c_n x_n$ for all $x \in H$. We define an operator S by the equation

$$Sx = \sum_n \langle x, x_n\rangle\, x_n$$

It can be shown that this operator is bounded below by a positive number [30], and is therefore invertible, that S^{-1} is also bounded, that $\{S^{-1}x_n\}$ is also a frame, with bounds B^{-1}, A^{-1}, and finally that for every $x \in H$, that

$$x \;=\; \sum_n \langle x, S^{-1}x_n\rangle\, x_n$$

$$= \sum_n \langle x, x_n \rangle S^{-1} x_n$$

$\{x_n\}$ and $\{S^{-1}x_n\}$ are called dual frames. It can be shown that every exact frame is a basis, that is, that the representation by the frame of any $x \in H$ is unique [30]. We also have that if $\{x_n\}$ is an exact frame, that $\{x_n\}$ and $\{S^{-1}x_n\}$ are biorthonormal, that is,

$$\langle x_n, S^{-1}x_m \rangle = \delta_{n,m}$$

where $\delta_{a,b} = 1$ if $a = b$ and $\delta_{a,b} = 0$ otherwise. We now have a means, given a frame, to break down and re-constitute any function. Next, we will define exactly what we mean by a wavelet, and expand from the wavelet series, which corresponds to the extension above, to the continuous and discrete wavelet transforms.

The following operations are defined for all functions $f \in H$.

1. Translation: $T_a f(x) = f(x - a)$ for $a \in \mathbb{R}$
2. Modulation: $E_a f(x) = e^{2\pi i a x} f(x)$ for $a \in \mathbb{R}$
3. Dilation: $D_a f(x) = |a|^{-1/2} f\left(\frac{x}{a}\right)$ for $a \in \mathbb{R}$

A function $g \in L^2(\mathbb{R})$ is called a *wavelet*, or a *mother wavelet* if

$$\{D_a T_b g\}_{a,b \in \mathbb{Z}}$$

defines a frame in $L^2(\mathbb{R})$. This definition implies a number of things, many of which are very involved mathematically [31]. Some of these are that the function vanishes at infinity, that $\int_{-\infty}^{\infty} g(x)\, dx = 0$, and that

$$C_g = \int_{-\infty}^{\infty} \frac{|\hat{g}(y)|}{|y|}\, dy < \infty$$

where $\hat{g}$ is the Fourier transform of g.

Define $\{\tilde{g}_{a,b}\}$ as the dual frame to g, that is

$$\tilde{g}_{a,b} = S^{-1} g_{a,b}$$

where $g_{a,b} = D_a T_b g = |a|^{-1/2} g\left(\frac{x-b}{a}\right)$ and $S = \sum_{a,b\in\mathbb{Z}} \langle \cdot, g_{a,b}\rangle\, g_{a,b}$. Then the wavelet series of a function $f \in L^2(\mathbb{R})$ is

$$f(x) = \sum_{a,b\in\mathbb{Z}} \langle f, \tilde{g}_{a,b}\rangle\, g_{a,b}$$

In practice, we usually discretise a such that the dilations are powers of an integer, setting $a_0 = 1$, for example, and $a_n = 2a_{n-1}$, where $a > 1$ is a contraction, and $a < 1$ is a dilation.

The continuous wavelet transform $\Phi_g f$ is then deduced directly from the series [29] by

$$\Phi_g f(u,v) = \int_{-\infty}^{\infty} f(x) e^{-u/2} \overline{g\left(e^{-u}x - v\right)}\, dx$$

or $\Phi_g f(u,v) = \langle f, g_{e^u,v}\rangle$, for $u, v \in \mathbb{R}$. The inverse of this transform is

$$f(x) = \int_{-\infty}^{\infty} \Phi_g f(u,v) e^{-u/2} g(e^{-u}x - v)\, du\, dv$$

4.3.3 Multiresolution analysis

There remains the problem of how to find a mother wavelet which generates a frame, and how the continuous transform extends to a discrete transform. This leads us to the idea of a multiresolution analysis.

A *multiresolution analysis* (MRA) for $L^2(\mathbb{R})$ consists of [32]

1. Closed subspaces $V_n \subset L^2(\mathbb{R})$ for $n \in \mathbb{Z}$ satisfying
   (a) $V_n \subset V_{n+1}\ \forall n \in \mathbb{Z}$
   (b) $\cdots \cap V_n \cap V_{n-1} \cap V_{n-2} \cap \cdots = \emptyset$
   (c) $\bigcup_{n\in\mathbb{Z}} V_n$ is dense in $L^2(\mathbb{R})$
   (d) $V_n = D_2 V_{n+1} = \{D_2 f : f \in V_{n+1}\}$
2. A function $\varphi \in V_0$ such that $\{T_m\varphi\}_{m\in\mathbb{Z}}$ is an orthogonal basis for V_0.

It is easy to find V_0 and φ which fulfill these conditions, for example, $\varphi = \chi_{[0,1]}$ and $V_0 = \{f \in L^2(\mathbb{R}) : f \text{ is constant on each } [m, m+1]\}$ produce the well-known Haar wavelet. It has been shown [29] that every multiresolution analysis produces an orthonormal basis for $L^2(\mathbb{R})$.

Since V_n is contained in V_{n+1}, we can define W_n to be the orthogonal compliment of V_n in V_{n+1}. It is then possible to find a function $\phi \in W_0$ such that $\{T_m\phi\}_{m\in\mathbb{Z}}$ is an orthonormal basis for W_0. From the definition of our V_n, it is obvious that $W_{n+1} = D_2 W_n = \{D_2 f : f \in W_n\}$, so $\{D_{2^n} T_m \phi\}$ forms an orthogonal basis for W_n. Finally, since $V_{n+1} = V_n \oplus W_n$ where $A \oplus B = \{a + b : a \in A, b \in B\}$, we have $V_{n+1} = \bigoplus_{i=-\infty}^{n} W_i$, and by extension, $L^2(\mathbb{R}) = \bigoplus_{-\infty}^{\infty} W_i$, so $\{D_{2^n} T_m \phi\}_{m,n\in\mathbb{Z}}$ forms an orthonormal basis in $L^2(\mathbb{R})$.

This gives us an interesting way of expressing a function. Given a function $f(x)$, we can find $\{g_i(x) : g_i(x) \in W_i\}$ and $h_i(x) \in V_i$ such that

$$\begin{aligned} f(x) &= h_0(x) + g_1(x) + g_2(x) + \cdots \\ &= h_1(x) + g_2(x) + g_3(x) + \cdots \end{aligned}$$

We can consider the g_i to be detail functions, and h_i to be an averaging filter.

This is how the discrete wavelet transform works, by repeatedly filtering a signal through both high-pass and low-pass filters, and using the output of the low-pass filter as the input to the next high-pass filter, keeping only the final low-pass output at the required sampling level.

The definition of an MRA gives us an interesting means of expressing the wavelet which generates one level with each of the wavelets in the next level of the analysis. This is

$$\phi\left(x\right) = \sum_{k=0}^{M-1} c_k \phi\left(2x + k\right)$$

[33]. The value M, which is the number of non-zero coefficients required to satisfy the relationship, is the order of the wavelet. The self-similar nature of this equation is what gives some wavelets a distinctly fractal appearance.

A number of constraints must be met for a set of given coefficients to generate a wavelet. The area under the curve should normally be 1, which requires that $\sum c_k = 2$. We also wish to have a curve φ which is not only orthonormal to translations, as ϕ is (since $\int \phi(x)\phi(x-k)\,dx = 0$), but is also orthonormal to dilations.

This curve φ exists, and is given by

$$\varphi(x) = \sum_{k} \left(-1\right)^k c_{1-k} \phi\left(2x - k\right)$$

Normalization of both ϕ and φ require that

$$\sum_{k} c_k c_{k-2m} = 2\delta_{0m}$$

that is, that the sum of the squares of the coefficients is two, and the sum of the pairwise products of odd and even coefficients is zero. It can also be shown [31] that

$$\sum_k (-1)^k c_{1-k} c_{k-2m} = 0$$

for all m.

The coefficients which generate the Haar basis, for example, are $c_0 = 1, c_1 = 1$. The Daubechies 4-wavelet [33] is generated by the coefficients

$$\begin{aligned} c_0 &= \frac{1}{4}\left(1+\sqrt{3}\right) \\ c_1 &= \frac{1}{4}\left(3+\sqrt{3}\right) \\ c_2 &= \frac{1}{4}\left(3-\sqrt{3}\right) \\ c_3 &= \frac{1}{4}\left(1-\sqrt{3}\right) \end{aligned}$$

Once we have the coefficients for the wavelet we're using for the task at hand, we then proceed to apply our high- and low-pass filters to the signal as follows. Given a discrete signal f, we calculate the low-pass filter as

$$a_i = \frac{1}{2}\sum_{j=1}^{N} c_{2i-j+1} f_i$$

and the high-pass filter as

$$b_i = \frac{1}{2}\sum_{j=1}^{N} (-1)^{j+1} c_{j+2-2i} f_i$$

for $i = 1, 2, \cdots, N/2$ [33].

4.3.4 Wavelet transform in image processing

There exist means of extending the wavelet to two or more dimensions [34], but computationally, these are unwieldy and are not often used. The normal extension to two or more dimensions of the discrete wavelet transform is by the means described above for one dimension, changed only slightly.

Given a discrete surface $A(i,j) : 0 < i,j < N-1$, we first apply our high-pass and low-pass filters horizontally, generating $L(A)(i,j) : 0 < i < N-1, 0 < j < (N-1)/2$ and $H(A)(i,j) : 0 < i < N-1, 0 < j < (N-1)/2$. Then for each of G and H, we perform the low- and high-pass filter operations vertically, resulting in four sub-images, $L(L(A))(i,j)$, $L(H(A))(i,j)$, $H(L(A))(i,j)$ and $H(H(A))(i,j)$ for $0 < i,j < (N-1)/2$. We then repeat the operation with $L(L(A))(i,j)$ as the input image for the next iteration, as with the one-dimensional version. A sample decomposition of the Lena sample image using the Daubechies 4-wavelet (Figure 3.6), greyscale equalised to emphasise high-pass filter output, is shown in Figure 4.1.

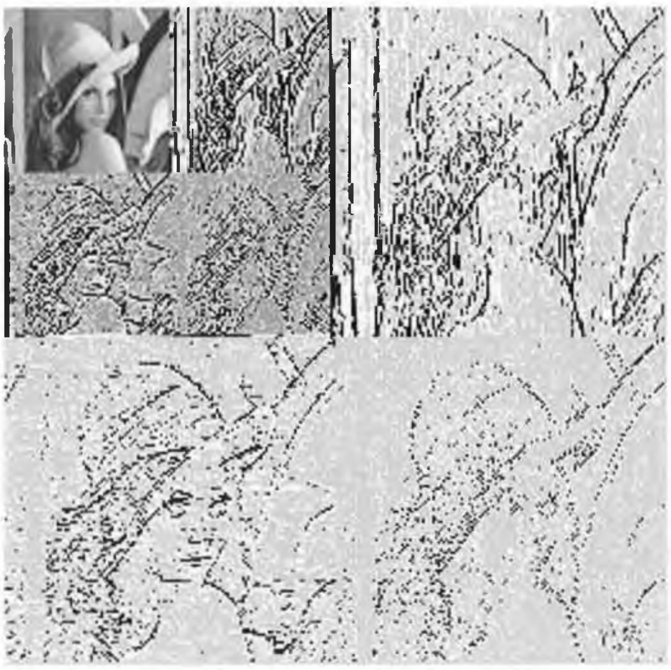


Figure 4.1: Two stages of wavelet decomposition of Lena

This image can then be compressed with any of the methods which are used with other transform methods, namely entropy coding methods, and bandpass quantisation filters. Since the wavelet transform is essentially a series of bandpass filters, this quantisation process is considerably easier than with other transform methods. Using only entropy coding and coefficient quantisation, the wavelet transform has been shown [35] to achieve compression ratios exceeding 100:1, without considerable loss of image fidelity.

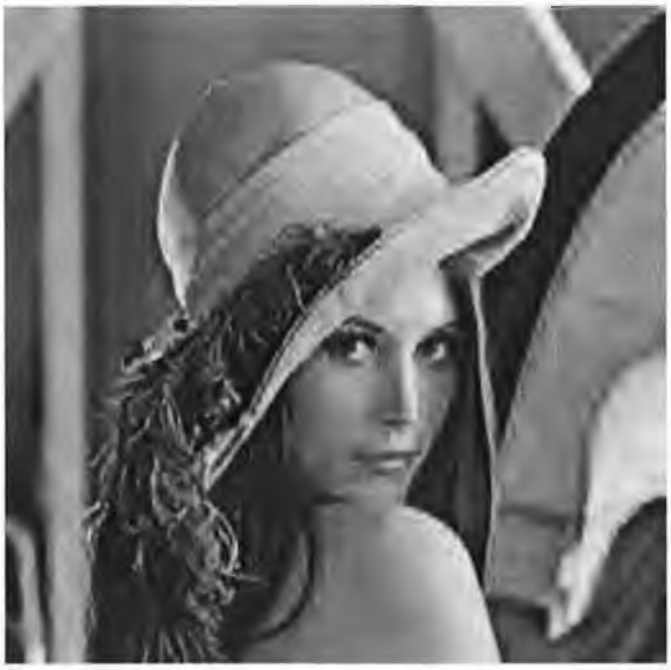


Figure 4.2: Wavelet transform encoded Lena (128:1)

As with the Fourier transform, the principle loss of image quality is near edges (see Figure 4.2), but the edge effects, even at high compression ratios, are less prominent that with the DCT, or the fractal transform.

4.4 Fractal transform

Fractal image compression is based on two principle assumptions. First, that every natural image has self-similarity contained within it, and second, that there is a means of finding this self-similarity and extracting it.

The presence of fractals in nature, from ferns to coastlines, has been discussed for many years [36, 15, 37], and in the late 1980s, Michael Barnsley and Arnaud Jacquin [38, 23] introduced a means of extracting redundant self-similarity.

4.4.1 Contraction mappings

The methods they proposed are based on contraction mappings and iterated function system (IFSs). A contraction mapping, informally, is any mapping where the distance between any two points before the mapping is greater than the distance between them after the mapping. It can be thought of as the cooling of a surface, or any of the other processes where the same content is shrunk into a smaller space. More formally, a function $f : X \to X$ is called a contraction mapping of the set X under the metric $d(\cdot,\cdot) : X \times X \to \mathbb{R}[0,\infty]$ if, for all $x, y \in X$,

$$d\left(f\left(x\right), f\left(y\right)\right) \leq c \times d\left(x, y\right)$$

with $0 < c < 1$.

The most important property of a contraction mapping is that there is exactly one point which is invariant under the mapping, that is that $f(x_f) = x_f$, and that under iteration, every point eventually contracts to this point. This is the "Contraction mapping theorem", and these properties are the foundation of the theory of fractal image compression.

Theorem 4.1 (Contraction mapping fixed point theorem) *Let $\tau : X \rightarrow X$ be a contraction mapping on a complete metric space (X, d). Then τ has exactly one fixed point $x_F \in X$, and*

$$\lim_{n\rightarrow\infty} \{\tau^n(x)\} = x_F \ \forall x \in X$$

Proof

Let $x \in X$, $d(\tau(x), \tau(y)) \leq sd(x, y)\ \forall x, y \in X$ and $0 \leq s < 1$. Then define

$$\begin{aligned} x_0 &= x \\ x_1 &= \tau(x) \\ &\vdots \\ x_n &= \tau(x_{n-1}) \end{aligned}$$

And we have

$$\begin{aligned} d(x_m, x_n) &\leq sd(x_{m-1}, x_{n-1}) \\ &\vdots \\ &\leq s^n d(x_{m-n}, x_0) \\ &\leq s^n \left(d(x_{m-n}, x_{m-n-1}) + \cdots + d(x_2, x_1) + d(x_1, x_0)\right) \\ &\leq s^n \left(s^{m-n-1} d(x_1, x_0) + \cdots + sd(x_1, x_0) + d(x_1, x_0)\right) \\ &\leq s^n \left(\frac{1 - s^{m-n}}{1 - s}\right) d(x_1, x_0) \\ &\leq \frac{s^n}{1 - s} d(x_1, x_0) \end{aligned}$$

Since $\frac{d(x_1,x_0)}{1-s}$ is constant and finite, and s^n can be made arbitrarily small, so for any $\epsilon > 0$, we can choose an n such that

$$d(x_m, x_n) \leq \frac{s^n}{s - 1} d(x_1, x_0) < \epsilon$$

And we have that $\{x_n\}$ is a Cauchy sequence, and thus converges in X. Let $\lim_{n\to\infty} x_n = x_F$ and

$$\begin{aligned} \tau(x_F) &= \tau\left(\lim_{n\to\infty} \tau^n(x_0)\right) \\ &= \lim_{n\to\infty} \tau^{n+1}(x_0) \\ &= x_F \end{aligned}$$

so x_F is a fixed point of τ. The uniqueness of x_F follows from the observation that if τ were to have two different fixed points, they would have to satisfy the equation

$$d(\tau(x_F), \tau(y_F)) \leq sd(x_F, y_F)$$

$$\Rightarrow d(x_F, y_F) \leq sd(x_F, y_F) < d(x_F, y_F)$$

which is obviously a contradiction. This concludes the proof.

□

What this means, in the context of our problem, is that if we consider digital images as points in image space, in other words as elements of the set of all possible digital images, and if we find a contraction mapping for which our image is the fixed point, then we need only that mapping to reproduce our original image. That is, starting with any image we can iterate that map a number of times to get an approximation of our image. Our only problem is how to find an appropriate contraction.

4.4.2 Iterated function systems

A particular group of contraction mappings which are of interest to us are IFSs. An IFS is a group of contraction mappings which each act on the set

X, with the overall mapping being made up of the union of the individual maps. More formally, an IFS is a set $F = \{f_1, f_2, \cdots, f_n\}$ of contraction mappings. The transform $F : X \to X$ is defined as

$$F(x) = \cup_{i=1}^{n} f_i(x)$$

for all $x \in X$. It is easy to show that if each of the f_i is a contraction mapping, then F is also a contraction.

A wide variety of weird and wonderful images can be drawn with just three or four contractions making up an IFS, and many of these have an oddly natural look to them. Some examples are contained in Figures 4.3, 4.4 and 4.5).

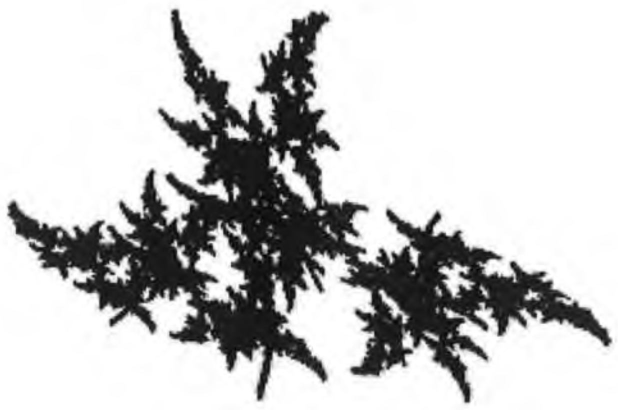


Figure 4.3: IFS attractor generated by 4 contractions

4.4.3 Collage theorem

The collage theorem was an attempt to provide a way of finding an iteration function system which has an arbitrary image as its attractor. The Hausdorff

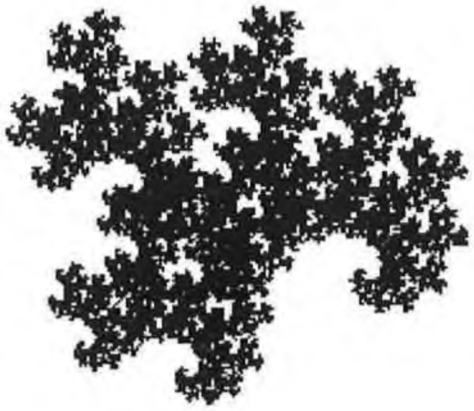


Figure 4.4: Moss-like attractor generated by 3 contractions

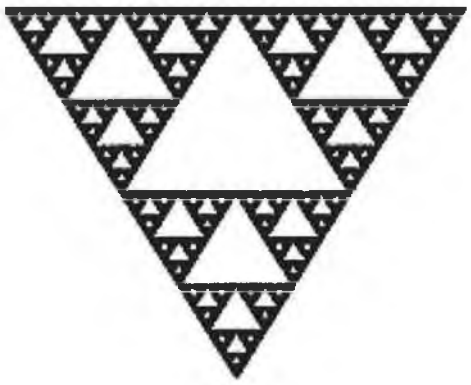


Figure 4.5: Sierpinski triangle, generated by 3 contractions

metric, which provides a way of measuring the distance between point sets, is central to the theorem.

Given a complete metric space (X, d), and point sets $A, B \subset X$, we define

$$d(x, B) = \min\left(d\left(x, y\right) : y \in B\right)$$

$$d(A, B) = \max\left(d\left(x, A\right) : x \in A\right)$$

$$h(A, B) = \max\left(d\left(A, B\right), d\left(B, A\right)\right)$$

It can easily be shown that d_H fulfills the conditions to be a metric, namely that $h(x,x) = 0$, $h(x,y) = h(y,x) \geq 0$ and $h(x,z) \leq h(x,y) + h(y,z)$. As such, if we can find a contraction mapping on h we are guaranteed a fixed point. We also have a metric which measures in some sense the similarity between point sets.

Having laid the foundations, we can now proceed with a proof of the Collage theorem.

Theorem 4.2 (Collage theorem, Barnsley, 1985) *Let (E,h) be a complete metric space, let $W = \bigcup_{n=1}^{N} w_n$ be an IFS with contraction factor s, and fixed point T_F. Let T be a closed subset of X. Let $\epsilon > 0$ be any positive number, and suppose that the $\{w_n\}$ are chosen such that*

$$h\left(T, W(T)\right) < \epsilon$$

Then

$$h(T, T_F) \leq \frac{\epsilon}{1-s} \ \forall x \in E$$

Proof

$$\begin{aligned}
h(T,T_F) &= \lim_{n\to\infty} h(T, W^n(T)) \\
&\leq \lim_{n\to\infty} \sum_{i=1}^{n} h(W^{i-1}(T), W^i(T)) \\
&\leq \lim_{n\to\infty} \sum_{i=1}^{n} s^{i-1} h(T, W(T)) \\
&\leq h(T,W(T)) \lim_{n\to\infty} \left(\frac{1-s^n}{1-s}\right) \\
&\leq \frac{h(x, W(T))}{1-s}
\end{aligned}$$

$$\leq \frac{\epsilon}{1-s}$$

□

The general meaning of this theorem is that if we can find an IFS which approximately covers a point set, we are guaranteed that the final attractor will also approximate the point set, and we have a well-defined bound, based on the contraction factor of the IFS.

There still remains the problem of how to find the individual contraction mappings. We need to minimise N, the number of mappings, and ϵ, the initial error in the collage, and even if we limit our search to affine maps in $\mathbb{R}^2$, the problem of finding maps which minimise the error is difficult.

4.4.4 Partitioned iterated function systems

A method of systematically coding arbitrary images was proposed by Jacquin [38]. He proposed using partitioned IFSs, which are more flexible for general applications. The general principle is that we divide the image into range blocks, small sections which partition the image, and then proceed to search for larger regions (domain blocks) which minimise the difference between the original block and the transformed block under a contraction map.

More formally, let X be a complete metric space, and let $D_i \subset X$ for $i = \{1, 2, \cdots, n\}$. A partitioned iterated function system (PIFS) is a collection of contractive maps $\{w_i : D_i \to X\}_{i=1}^{n}$.

We choose our maps w_i such that $\bigcup_{i=1}^{n} w_i(D_i) = X$ and $\bigcap_{i=1}^{n} w_i(D_i) = \emptyset$, so that the range blocks are non-overlapping and partition the image. We then search for D_i which map to R_i under a contraction [39, 23].

There are a number of decisions which must be made when coding an image by a PIFS. First, we must decide on how we partition the image into range blocks. Second, we must decide what type of contractions we will use to map domain blocks to range blocks. Thirdly, we must decide what search strategy we will use to find good domain block matches. Finally, we need to decide what type of quantisation we will use to further compress the resulting mappings.

4.4.4.1 Partitioning strategies

The first partitioning schemes used involved uniform partitioning of the image into 8×8 range blocks, but this strategy limits the compression ratio which can be achieved, and takes no account of the context of the image, with a plain greyscale image requiring the same number of maps as a portrait, for example.

Quadtree partitioning [39] is a normal extension to this partitioning scheme, which attempts to find an acceptable match for the largest possible block, and if it fails to do so, splits the block into quadrants, each of which is coded individually.

Several other novel partitioning schemes have been proposed, including hexagonal or triangular partitions with quadtrees [40, 39]. Irregular block shapes, obtained by adaptive merging of individual squares, have also been proposed.

However, quadtree partitioning schemes are by far the most used, and produce good results for most images, even at very low bit rates [39].

4.4.4.2 Contraction choice

While any contractive mappings suffice for coding, the amount of time required to search for good matches mean that we limit our mappings to simple affine maps, that is, maps of the form

$$w_i(x,y) = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \end{bmatrix}$$

where (e_i, f_i) is a spatial offset, and $\begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix}$ is an affine map, which can take into account scaling, shear and isometries of the square. Normally to save coding time only square domain blocks are used, which negates the need for asymmetric transformation, and reduces the transformation matrix to αI_n where I_n is one of the isometries of the square and α is the contraction factor. However, other affine maps [23] have been used successfully.

Mappings also take into account a greyscale scaling and offset, which can be figured into the transform if we consider grey scale as a third dimension [1]. Normally, only spatial contractions of order 2^{-n} are used, primarily for ease of calculation.

4.4.4.3 Domain block searching

The number of possible domain blocks is huge, in comparison to the number of range blocks. For example, using 16×16 domain blocks, and a 256×256 image, there are 250×250 possible (overlapping) domain blocks for each range block. This means that a full search, even on this fairly basic partitioning, requires the scaling and comparison of $250 \times 250 \times 32 \times 32$, or approximately 2^{26}, 8×8 matrices. Obviously, the required time to calculate the transform is excessive for this type of search. We wish to reduce the number of domain blocks we check against each range block, without a considerable loss in the quality of the coding.

Among the schemes which have been proposed are classification of domain blocks into various categories (edge, mid-range, smooth, etc.), and only examine domain blocks from the same class for range blocks of that type, and local searches [39].

4.4.4.4 Coefficient quantisation

Once we have calculated a best-matching domain block and transform, we can further increase compression ratios by reducing the number of bits we use for each stored transform.

Transforms are identified, normally, by six values, the scaling ratio α, the isometry of the square I_n, the spatial offset (e_i, f_i), the greyscale scaling ratio s_i and the greyscale offset o_i.

It is reasonable to store the isometry used as three bits, since there are eight possible isometries, although some authors [41, 42] have suggested that larger domain pools, or a subset of the isometries, may behave as well, while reducing some of the computation, and reducing the per-transform data which must be stored.

The scaling ratio, which as I have previously mentioned is normally fixed at 0.5, is not usually stored. However, it seems reasonable to use 5 bits (a resolution of 1/32 for numbers between 0 and 1) to code this, if we decide to use non-dyadic scaling factors.

The spatial offset of the domain block, in a worst case scenario, requires $\log_2(n) + 1$ bits for an image of size $2^n \times 2^n$. However, with local block searches, and non-comprehensive searches, we can use considerably less than that.

Quantisation levels for the greyscale transform (s_i, o_i) of 5 bits for s_i and 7 bits for o_i have been suggested by Fisher [39].

In total, for a 256×256 image, with a fixed range block size of 8×8, yields 1024 transforms, each requiring 3 (square isometry) + 5 (scaling ratio) + 9 (domain offset in x direction) + 9 (domain offset in y direction) + 5 (greyscale scaling) + 7 (greyscale offset) bits, totalling 38 bits per transform, or 4864 bytes, for an original image size of 256×256, or 65536, bytes, yielding a compression ratio of 13.47.

However, this compression ratio is considerably improved by measures such as a fixed scaling ratio, a quadtree partitioning scheme, or some other partitioning scheme which reduces the number of transforms required. We can also substantially limit the size of the pool of candidate domain blocks we match against each range block.

4.5 Wavelet-fractal hybrid transforms

Because of the means by which the wavelet transform is created and implemented, many authors have suggested a link between the wavelet and fractal transforms [43, 44, 45]. Bearing in mind that in its creation by a multiresolution analysis, the wavelet itself is self-similar, and that the discrete wavelet transform examines an image on many related frequency levels, it is easy to see how this suggestion could be made.

4.5.1 Fractal coding in the wavelet domain

A means of exploiting the similarities between the wavelet and fractal transforms has been suggested [44]. This method centres on coding the image by the fractal transform in the wavelet domain. The reasoning behind this method is that if the fractal transform would normally map a domain block onto a range block in the spatial domain, that the frequency information at one level of the wavelet decomposition would correspond well with the frequency information of the range block at the previous level of decomposition, assuming dyadic fractal contractions.

Using the decomposition of Lena introduced earlier, if the fractal transform were, for example, to map the squares in Figure 4.6 to each other, we would expect these sections to be similar at different decomposition levels of the wavelet domain, as they are shown to be in Figure 4.7. Figure 4.7 shows a thresholded version of the wavelet decomposition (to improve visibility) shown in figure 4.1, with this mapping highlighted in the wavelet domain.

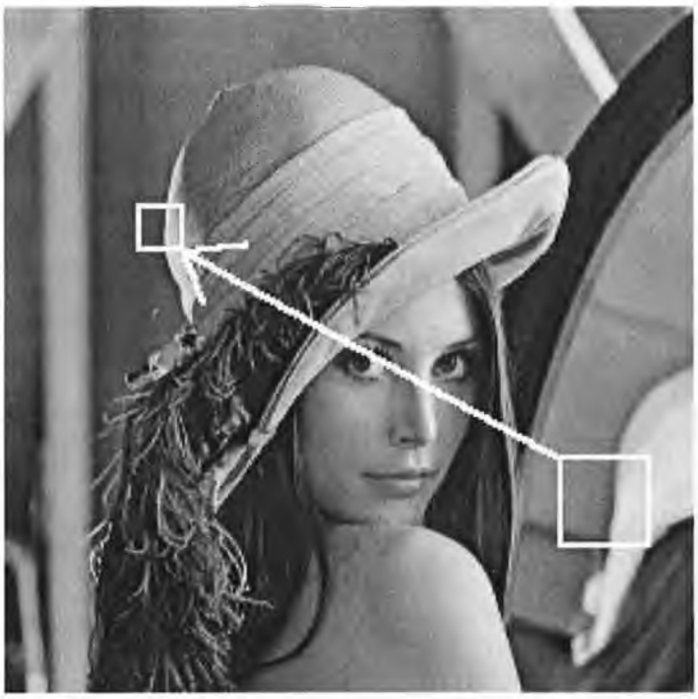


Figure 4.6: Sample mapping in Lena

4.5.2 Wavelet post-processing of fractal coding

Rather than the method used above, we present a different method of combining the fractal and wavelet transforms which codes the image to a high compression ratio using the fractal transform, and codes the errors only with very high entropy using the wavelet transform.

Since the errors in any fractal coding of an image occur mostly at region boundaries in blocking effects, the majority of the wavelet transform information will be contained in higher frequencies, and the result can be entropy coded to give very high compression ratios due to the sparse nature of the low energy sections of the transform. In this way we hope to get the best of both transforms, efficient compression of non-edge data with the fractal

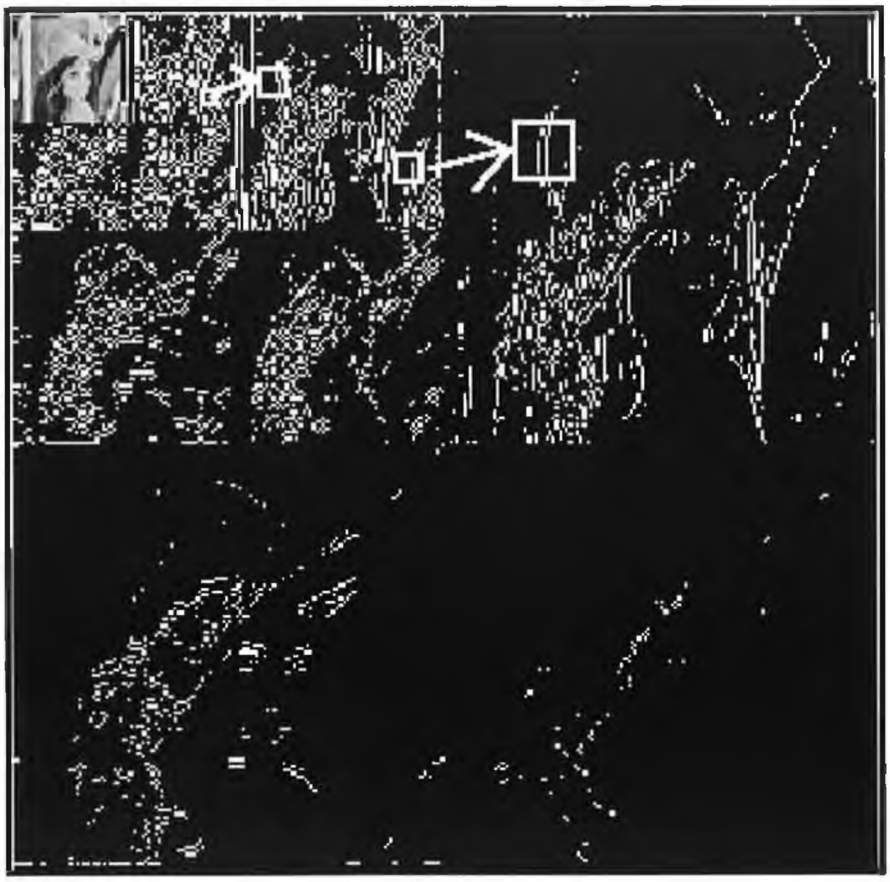


Figure 4.7: Mapping from figure 4.6 in wavelet domain

transform, and the edge preserving qualities of the wavelet transform, as testified by its use as the compression algorithm used for the FBI fingerprint library [26].

A flowchart representing the algorithm is shown in Figures 4.9 and 4.10. This coder should produce good visual quality, and should improve considerably on the fractal transform in signal to noise measurements. Because most of the edge information in the wavelet domain will be well coded, we also expect not to see the edging effects which are common with the wavelet transform.

4.5.3 Results and discussion

The hybrid coder consists of a standard quadtree based algorithm [39] and the Baseline Wavelet Transform Coder Toolkit wavelet coder [35]. The parameters used to vary the compression ratios of the fractal coder were the number of bits required to store greyscale offset and contraction, and spatial offset, the maximum and minimum size of range blocks, and the acceptable error tolerance in comparison of domain blocks and range blocks.

For the hybrid algorithm one main decision had to be made. We had to decide what portion of the final codes would be made up of fractal codes, and what proportion would be wavelet codes. For this decision, we used a number of combinations of parameters with the fractal coder, which yielded compression ratios from 30:1 to 250:1. We then coded the difference image with the required compression ratio to give a final compression ratio of between 30:1 and 100:1. The results for this operation are given in Figure 4.8.

It can be seen that the optimum proportion appears to be around 40:60, that is, if we wish to compress an image by 60:1, then we will compress the image by 150:1 with the fractal coder and the difference image by 120:1 using the wavelet transform to give the final compression.

For the comparison of the algorithms, three test images were used, Lena (Figure 3.6), Barbara (Figure 4.14) and Goldhill (Figure 4.19). These represent a good cross-section of the types of images a coder might be expected to handle, and also cover all of the problems which traditional coders experience. For example, the checked shawl of Barbara and the buildings of Goldhill will show up any problems with edge artifacts. The peak signal to noise ratio (PSNR) was used as the measure of fidelity for comparison.

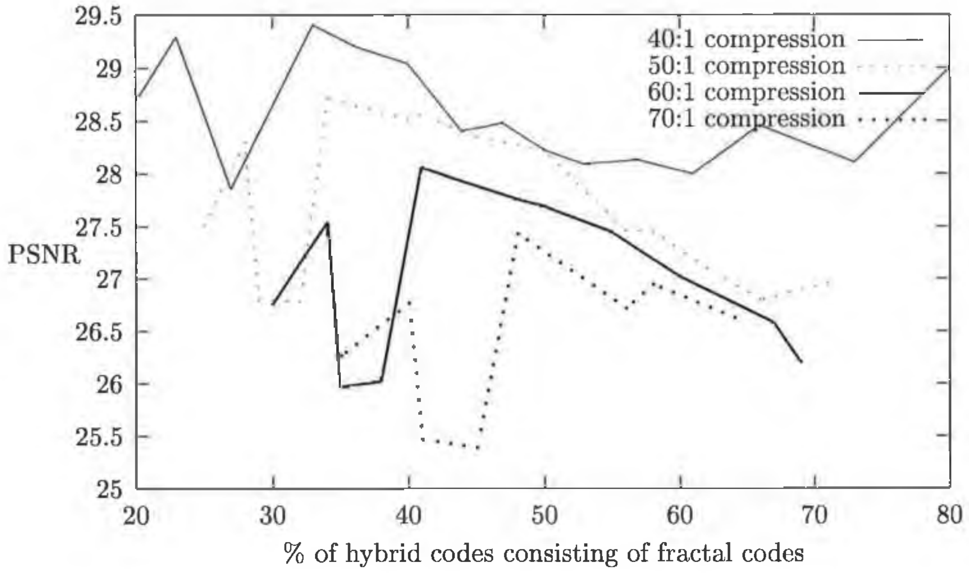


Figure 4.8: Analysis over various compression ratios of hybrid coder

Figure 4.13 contains the results of a comparison of the three coders used for the test image Lena. From this comparison it is obvious that the results achieved with the hybrid are worse than those achieved with the wavelet coder only, but improve substantially on those given by the fractal transform alone, at least with regard to the peak SNR.

Visually, the results achieved by the hybrid do reduce the edging artifacts which result from the fractal transform, but fall somewhat short of the wavelet transform results. There may be several reasons for this. In effect, the wavelet coding of the image codes the high frequency detail (edges, complex areas) at a higher bit rate than the fractal-wavelet hybrid. Ideally, we would like the wavelet part of the hybrid to concentrate exclusively on the edge information, since the fractal coder is already taking care of the larger, smoother areas quite well.

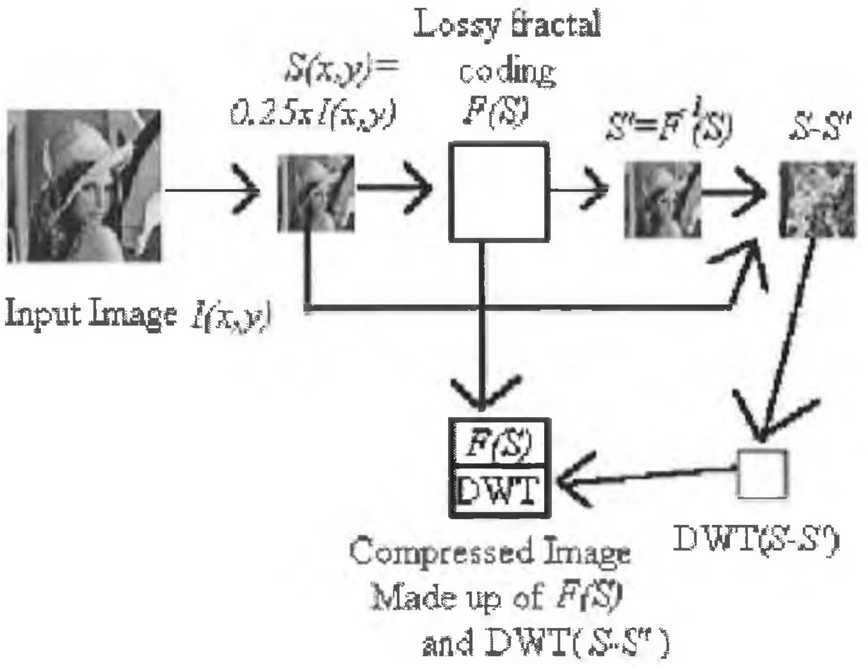


Figure 4.9: Image coding with hybrid fractal-wavelet codec

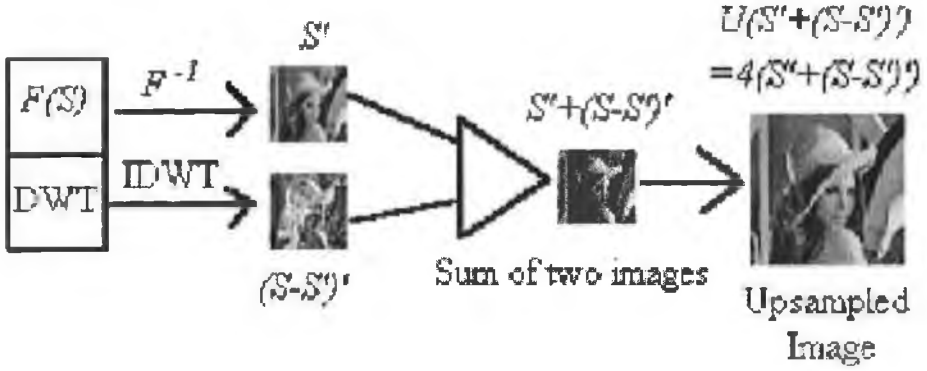


Figure 4.10: Image decoding with hybrid fractal-wavelet codec

Since essentially this approach appears to do nothing more than split the difference between the best results of the wavelet transform and the relevant

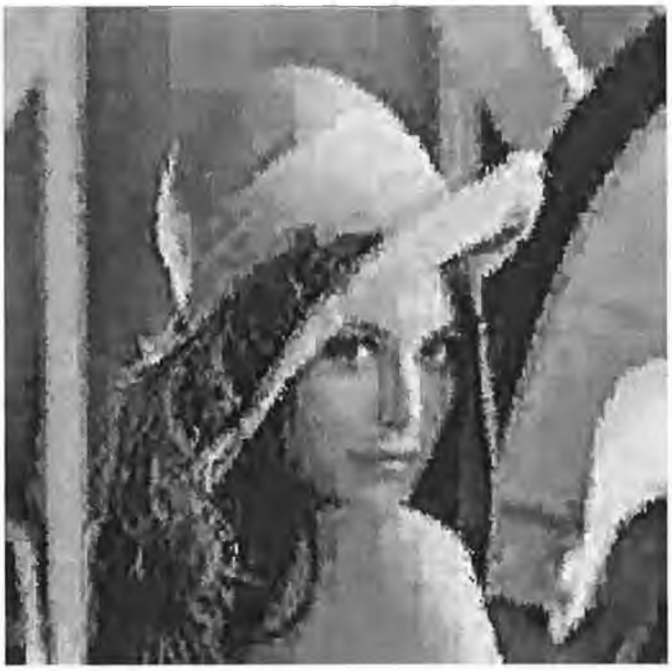


Figure 4.11: Lena encoded at 40:1 with hybrid coder

fractal transform, I believe that we can eliminate this as a possible future line of work. While the idea of a wavelet based coder complimenting a fractal coder by coding only edge effects at a very low bitrate is attractive, it is obvious that this approach does not improve on the results achieved by using only one coder at the same compression ratio. However, we feel that this result is in itself useful in eliminating this possibility for the future.

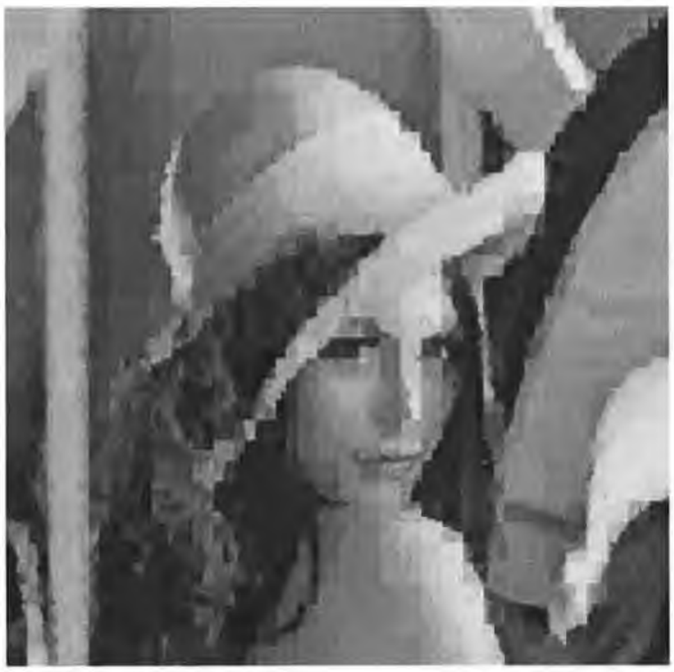


Figure 4.12: Lena encoded at 40:1 with fractal coder

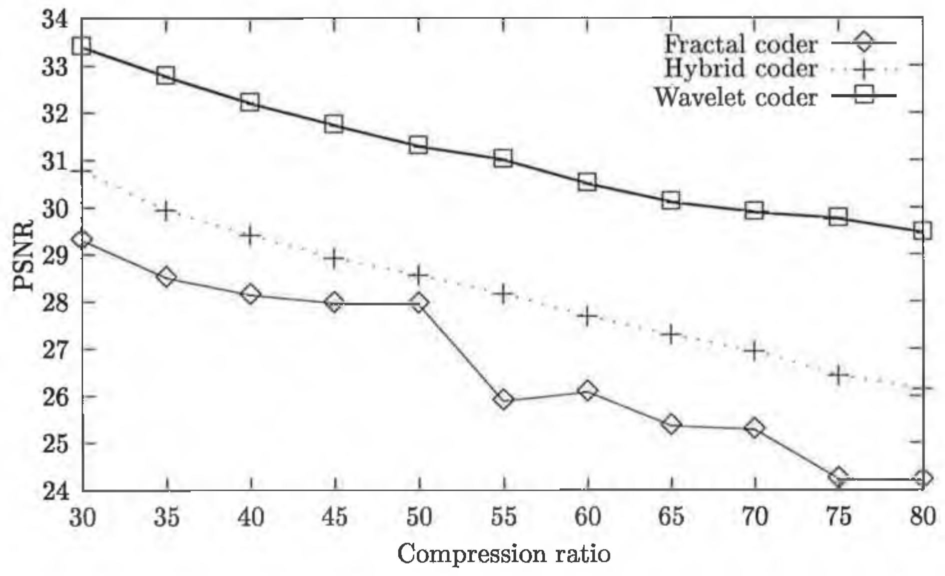


Figure 4.13: Comparison of fractal, hybrid and wavelet codecs using PSNR for Lena

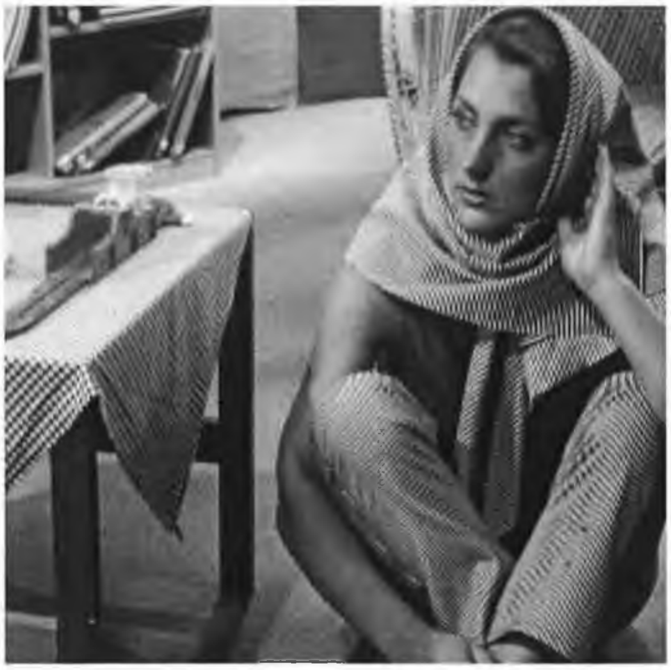


Figure 4.14: Barbara test image

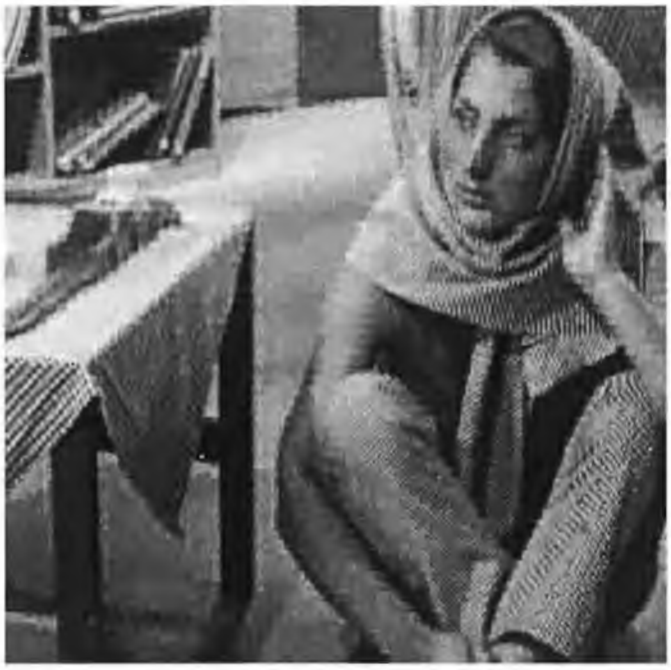


Figure 4.15: Barbara encoded at 30:1 with hybrid coder

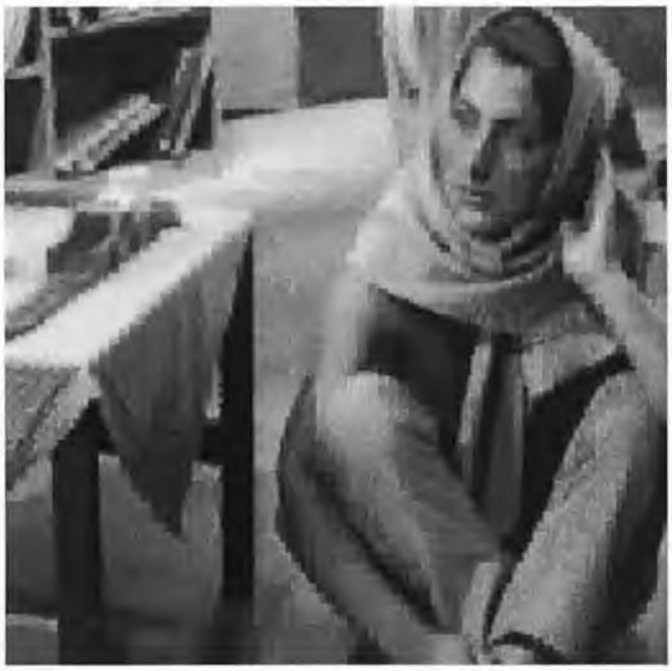


Figure 4.16: Barbara encoded at 30:1 with fractal coder

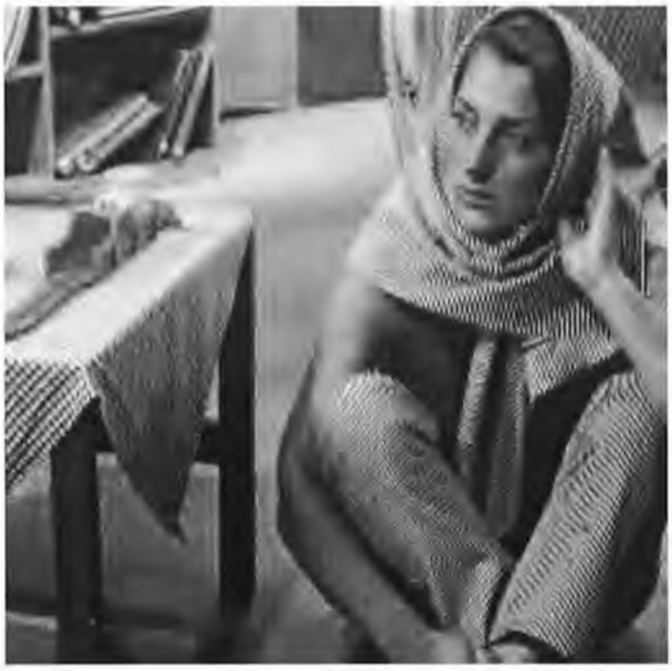


Figure 4.17: Barbara encoded at 30:1 with wavelet coder

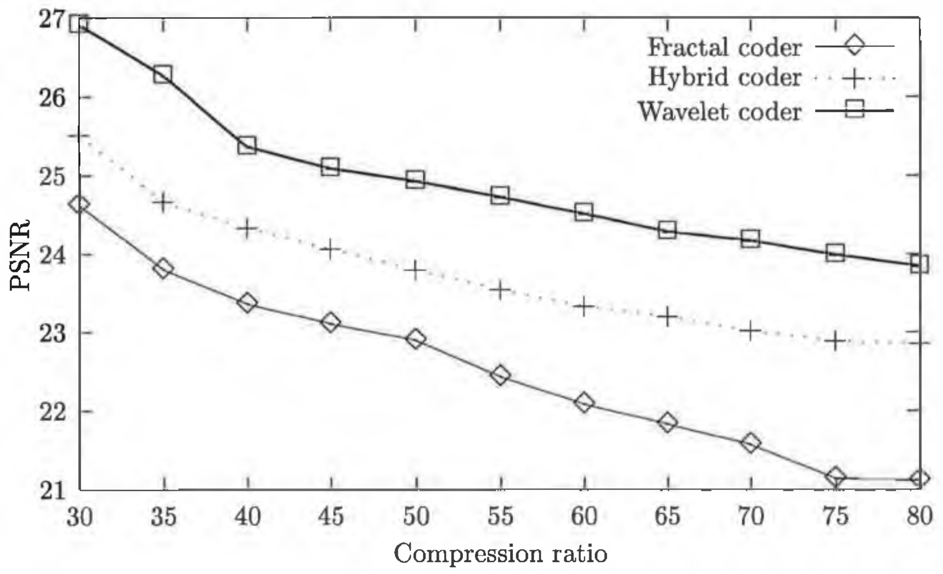


Figure 4.18: Comparison of three codecs for Barbara

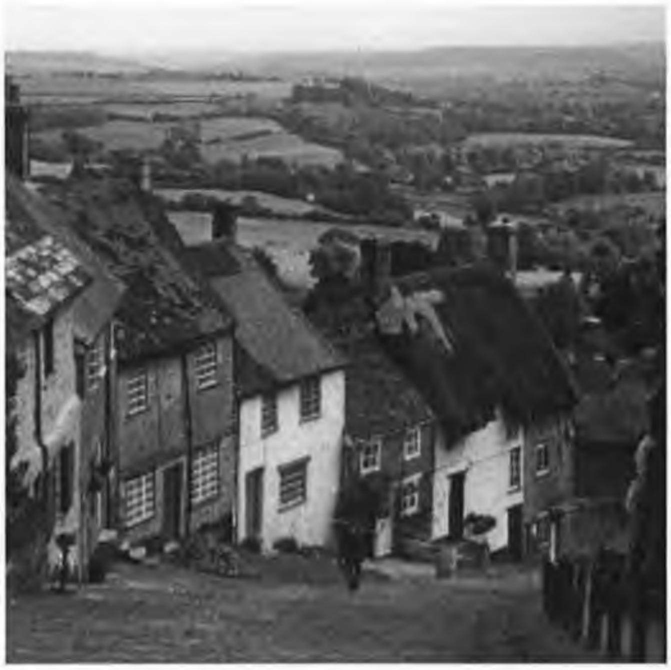


Figure 4.19: Goldhill test image

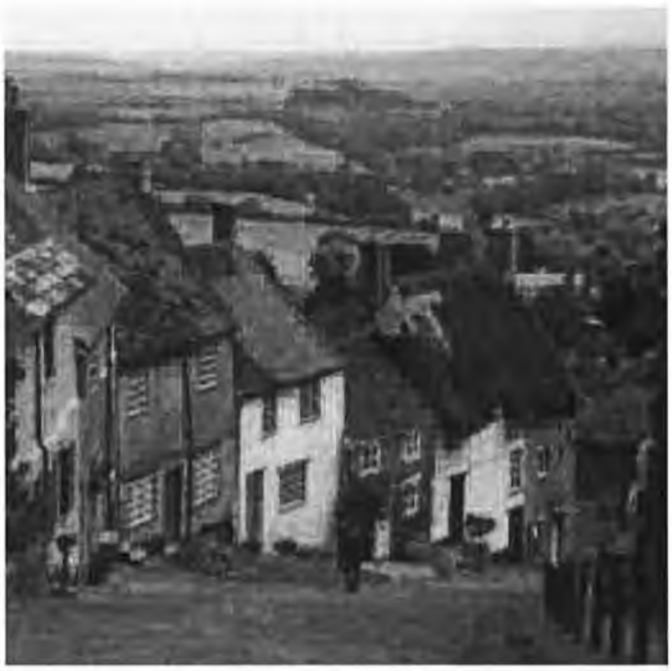


Figure 4.20: Goldhill encoded at 35:1 with hybrid coder

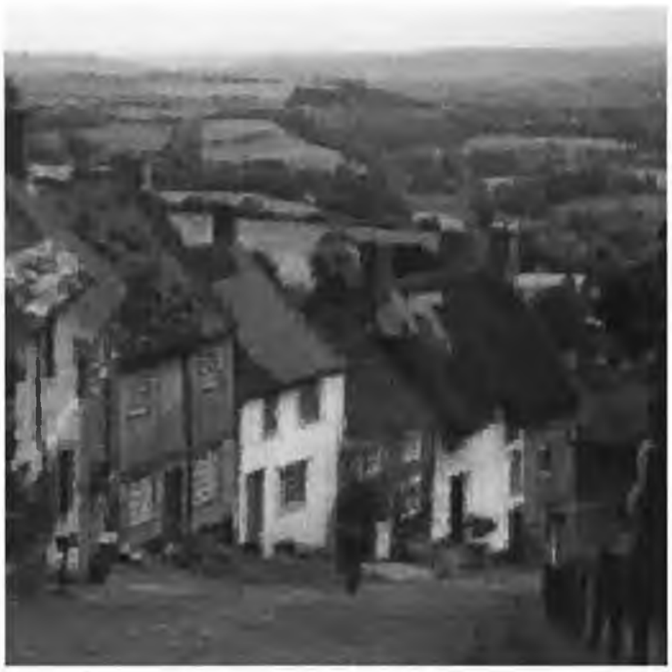


Figure 4.21: Goldhill encoded at 35:1 with fractal coder

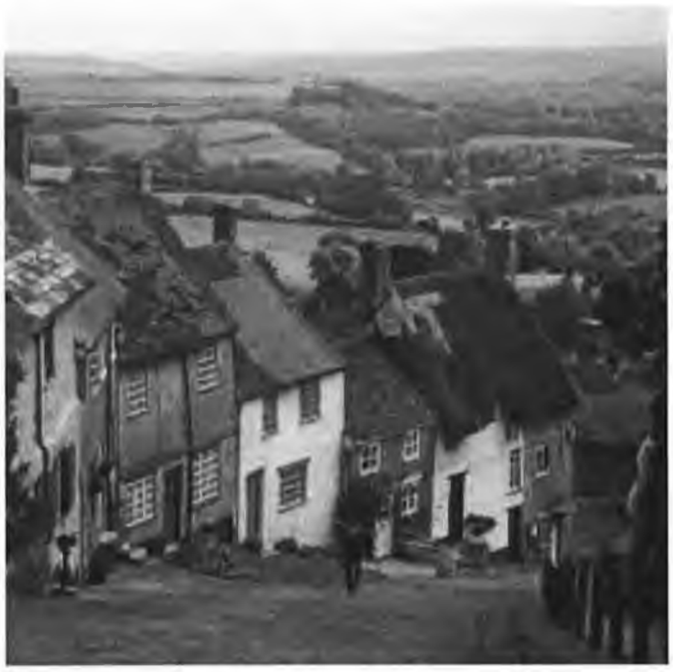


Figure 4.22: Goldhill encoded at 35:1 with wavelet coder

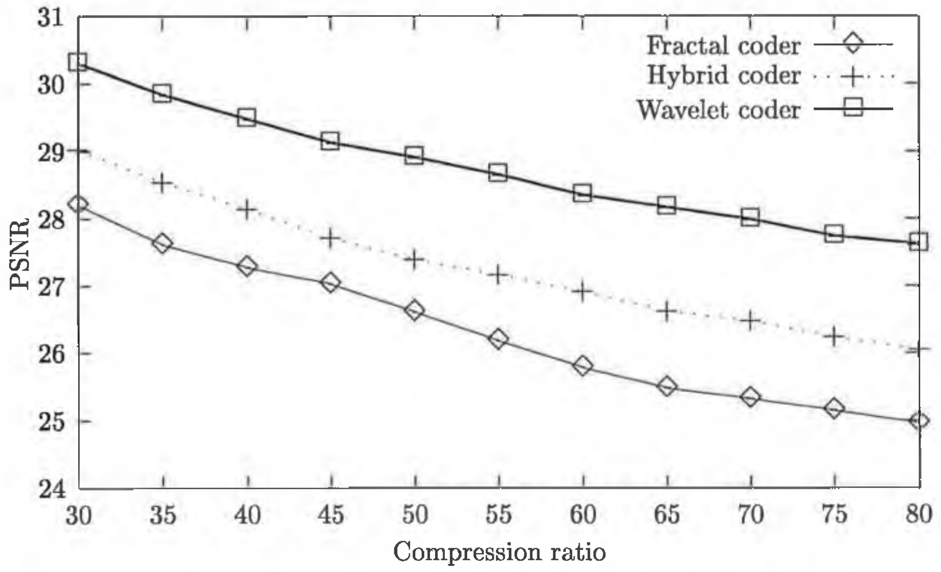


Figure 4.23: Comparison of three codecs for Goldhill

Chapter 5

Conclusion and discussion

5.1 Summary

While the field of fractal geometry is only 20 years old, people have examined nowhere differentiable functions since Poincare at the beginning of this century. Fractals have been shown to be of aesthetic use, as well as practical use [4, 36]. Fractal art has become widespread, and beautiful and complex images have been created using iterated function systems and chaotic functions. However, it is in the description of natural phenomena that fractal analysis has had the greatest impact.

In particular, fractals have been used to examine and model physical phenomena, such as the curviness of coastlines and rivers and the complexity of mountain slopes. This has been extended to the compression and analysis of physical structures which are not self-similar, or even globally pseudo self-similar, such as real-life images and textures.

In this thesis, we have given a comprehensive overview of fractal methods in texture analysis and image segmentation, as well as other methods of interest in these fields. We have also given an introduction to iterated function systems, as well as some of the vast body of work which has been written on their application to image compression, as well as showing some of the relationships between fractal and wavelet methods.

In chapter two, we layed out the basic premises behind texture analysis, and examined some of the methods which have been used to classify textures in the past. We also introduce fractal dimension as a means of measuring the complexity of textures. We have shown that very different textures can have a similar fractal dimension (see figure 2.16), limiting its value as a measure, but our hypothesis is that it can successfully be used as part of a larger feature vector.

In chapter three, we expand on the feature vector, incorporating both Fourier and box-counting dimensions, and a luminance feature, which simply averages the greylevels in a square around a given point. We also use the multi-fractal dimension proposed by Chaudhuri and Sarkar [12, 13, 20].

While the segmentation given by this combination of features was somewhat flawed, and segmentation of more complicated images was very difficult due to the relative size of window needed for a decent estimator, we believe that this approach holds some promise.

We also perform an in-depth analysis of segmentation using fractal codes, a method which requires a coding of the image using a partitioned iterated function system, as proposed by Ida and Sambonsugi [22]. This revealed some interesting facts. In constraining the greyscale scaling factor allowed, we produced some excellent results. However, when the scaling ratio was unconstrained, the coding which we used produced results which were almost homogeneous, and failed to pick up regions at all.

This is due to the fact that, in smaller (8×8 and 16×16) image segments, a greyscale scaling ratio substantially different to 0.5 changes the characteristics of the block (particularly the fractal dimension) such that our premise, that fractal coding maps regions of similar fractal dimension to each other, no longer holds. In addition, we found that spatially unconstrained mappings produced similar results.

Although this is more difficult to explain, we believe that this effect is due to the fact that when mapping over a number of different regions with similar characteristics, that a substantial number of orbits which do not strictly belong to the (spatially separated) regions are introduced, and that as we repeatedly iterate the map, that these orbits dominate the image, leaving only a small number of points as the basins of regions. It is also possible that our means of classifying orbits into regions may have been flawed.

In chapter four, we present an overview of the theory behind fractal image compression, and behind wavelet analysis. We also present a summary of the work which shows a link between these apparently disparate fields.

We propose using a wavelet coder with very low bit-rate as a means of improving the results from fractal coding. We show that this substantially reduces the edging effects in a fractally encoded image.

5.2 Future possibilities

While fractal techniques are widely used, this is still a relatively young area of research. In the area of texture analysis, there is a lot which is not yet known about the fractal dimension, particularly the relationship between the various estimators. In the area of image segmentation, for example, the method of segmenting using fractal codes shows substantial promise. More robust measures of the fractal dimension would also contribute positively to more accurate feature vectors for both texture classification and image segmentation. And improving fractal coders, possibly through more region-oriented partitioning schemes, would be one way of linking these disparate areas for one coherent purpose.

5.2.1 Image segmentation

The analysis presented here of the segmentation by fractal codes shows a number of shortcomings in this method which can be addressed in the future. The segmentation could be substantially improved if we understood a little more about the nature of how the orbits generated by better fractal coding schemes behave and interact, and by examining further the constraints under which this segmentation scheme can be improved.

More generally, fractal methods in image segmentation have some problems. Fractal dimension estimators are not, in general, robust on smaller data sets. There is room for substantial improvement in terms of improving the robustness of fractal dimension estimators. It is also worth investigating whether the fractal dimension really is a valid means of classifying textures, and segmenting images. We may find that the fractal dimension is not appropriate as a measure at all. More likely is that we will find that the fractal dimension does have a part to play in image segmentation, albeit as part of a larger feature vector.

5.2.2 Image coding

The weakest link in fractal coders is the fact that they are block based. This results in blocking effects near edges, and means that fractal coders are not sensitive to the context in an image.

The obvious solution to these problems is a context-based fractal coder. This would combine image segmentation with a fractal coder which is not block-based. In this way, edging effects would be reduced, and the possibility would exist of coding and storing individual objects in an image.

This could be accomplished by using partitioning schemes based on non-block areas, triangular or curved, for example, which could exploit similarities within a region. The major tasks in accomplishing this are the isolation of control points which would be the vertices for the partitioning, the parameterisation of the mappings from areas which are not necessarily congruent, and the efficient storage of those mapping, once calculated. Block based partitioning schemes do not have any of these problems, which is one of the main reasons they are widely used.

The advent of an efficient region-based fractal coder, which would presumably encompass segmentation as a first stage, will be a tremendous boost to fractal coding in general. Once the problems with deciding partitioning schemes, and then coding these efficiently, are overcome, I believe we will see a greater mainstream acceptance of the method in general.

5.2.3 Video telephony

One of the applications which holds the greatest potential for context-based image coders in general is video telephony and video conferencing applications. In general, the object of interest is the face and shoulders of the caller, and the background is not important. An object based coder could more efficiently code the object of interest, rather than wasting resources coding the background with equal fidelity.

In combination with image segmentation, a context based fractal coder would be ideal for this type of application. Currently, encoding speed would be a huge issue for this type of application, but with network bandwidth capability improving, and processor speeds increasing, and prices for both of these decreasing at an amazing rate, it may not be too long in the future until real-time encoding and decoding are possible for video streams.

Already fractal techniques are ideally suited to decoding stored video or images, in that decoding time is substantially less than encoding time, and with dedicated hardware can already be done almost in real-time. It is only a matter of time, in my opinion, until an object-based fractal coder, in combination with other video coding techniques such as motion estimation, will be capable of real-time video telephony.

Bibliography

[1] Martin J. Turner, Jonathan M. Blackedge, and Patrick R. Andrews. *Fractal Geometry in Digital Imaging.* Academic Press, London, 1998.

[2] P. Kube and A. Pentland. On the imaging of fractal surfaces. *IEEE Trans. on PAMI*, 10(5):704–707, September 1988.

[3] Kenneth Falconer. *Fractal Geometry – Mathematical Foundations and Applications.* John Wiley & Sons, Chichester, 1990.

[4] R.F. Voss. Random fractal forgeries. In R.A. Earnshaw, editor, *Fundamental Algorithms in Computer Graphics.* Springer-Verlag, 1985.

[5] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis and Machine Vision.* International Thompson Computer Press, London, 1993.

[6] S. Livens, P. Scheunders, G. Van der Wouwer, and D. Van Dyck. Wavelets for texture analysis. `http://wcc.ruca.ua.ac.be/~livens/WTA/WTA.html`.

[7] Computer Vision Group. Segmentation of textured images. Technical report, University of Bonn, 1997.

[8] G. Smith and I. Burns. Meastex image texture database and test suite v1.1. Available at

http://www.cssip.elec.uq.edu.au/~guy/meastex/meastex.html, May 1997.

[9] J. McCauley. pgmtexture. Software package, available in libgr-progs for Redhat Linux.

[10] P. Ohanian and R. Dubes. Performance evaluation of four classes of textural features. *Pattern Recognition*, 25(8):819–833, 1992.

[11] A. Pentland. Fractal-based description of natural scenes. *IEEE Trans. on Pattern Analysis and Machine Vision*, 6:661–674, 1984.

[12] N. Sarkar and B.B. Chaudhuri. An efficient differential box-counting approach to compute fractal dimension of an image. *IEEE Trans. on Sys., Man. and Cyber. A*, 24(1):115–120, January 1994.

[13] B.B. Chaudhuri and N. Sarkar. Texture segmentation using fractal dimension. *IEEE Trans. on PAMI*, 17:72–77, January 1995.

[14] P.A. Freeborough. A comparison of fractal texture descriptors. Available via the web at http://peipa.essex.ac.uk/bmva/bmvc97/papers/002/fractal.html.

[15] H.-O. Peitgen and D. Saupe. *The Science of Fractal Images.* Springer-Verlag, New York, 1988.

[16] P. Sahoo, S Soltani, A. Wong, and Y. Chen. Survey of thresholding techniques. *Computer Vision, Graphics and Image Processing*, 41(2):233–260, 1988.

[17] M. Levy. New theoretical approach to relaxation, application to edge detection. In *9th International Conference on Pattern Recognition*, pages 208–212, Rome, Italy, 1988.

[18] T. Kaneko and M. Okudaira. Encoding of arbitrary curves based on the chain code representation. *IEEE Trans. on Comms.*, COM-33(7):697–706, July 1985.

[19] Y. Hu and T. Dennis. Textured image segmentation by context enhanced clustering. *IEE Proc. - Vis. Image Signal Process.*, 141(6):413–421, December 1994.

[20] B.B. Chaudhuri, N. Sarkar, and P. Kundu. Improved fractal geometry based texture segmentation technique. *IEE Proc. E*, 140(5):233–241, September 1993.

[21] J. Hsiao and A. Sawchuk. Supervised texture image segmentation using feature smoothing and probailistic relaxation techniques. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11(12):1279–1292, 1989.

[22] T. Ida and Y. Sambonsugi. Image segmentation using fractal coding. *IEEE Trans on Circuits and Systems for Video Technology*, 5(6):567–570, December 1995.

[23] M.F. Barnsley. *Fractal Image Compression.* AK Peters, Wellesley, Mass., 1993.

[24] R.I Taylor and P.H. Lewis. 2D shape signature based on fractal measurements. *IEE Proceedings on Vision and Image Signal Processing*, 141(6):422–430, December 1994.

[25] *MPEG Home Page.* `http://drogo.cselt.stet.it/mpeg/`, August 1997.

[26] T. Hopper, C. Brislawn, and T. Bradley. The FBI wavelet/scalar quantisation standard for gray scale fingerprint image compression. *Visual Info. Processes II, SPIE Proc.*, 1961:293–304, April 1993.

[27] H. Myler and A. Meeks. *The Pocket Handbook of Image Processing Algorithms in C.* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[28] H. Royden. *Real Analysis, Third Edition.* Macmillan Publishing Company, New York, 1988.

[29] Christopher Heil and David Walnut. Continuous and discrete wavelet transforms. *SIAM Review*, 31(4):628–666, December 1994.

[30] Ingrid Daubechies. The wavelet transform, time-frequency localisation and signal analysis. *IEEE Trans. on Information Systems*, 36(5):961–1005, Setpember 1990.

[31] C.K. Chui. *An Introduction to Wavelets.* Academic Press, New York, 1992.

[32] Gerald Kaiser. *The Friendly Guide to Wavelets.* Birkhäuser, Boston, 1994.

[33] T. Edwards. Discrete wavelet transforms: Theory and implementation. Technical report, Stanford University, September 1991. Available at `http://qss.stanford.edu/~godfrey/wavelets/`.

[34] O. Rioul and M. Vetterli. Wavelets and signal processing. *IEEE Signal Processing magazine*, pages 14–38, October 1991.

[35] G. Davis. Baseline wavelet transform coder construction kit version 0.3. Technical report, University of

Dartmouth, January 1997. Available via the web at `http://www.cd.dartmouth.edu/~gdavis/wavelet/wavelet.html`.

[36] Benoit B. Mandelbrot. *The Fractal Geometry of Nature.* W.H. Freeman & Co., 1983.

[37] M.F. Barnsley. *Fractals Everywhere.* Academic Press, Boston, 1988.

[38] A. Jacquin. *A Fractal Theory of Iterated Markov Operators with Applications to Digital Image Coding.* Ph.D. thesis, Georgia Institute of Technology, Atlanta, GA, 1989.

[39] Yuval Fisher. *Fractal Image Compression – Theory and Applications.* Springer-Verlag, New York, 1994.

[40] K. Schröder and R. Mech. Combined description of shape and motion in an object based coding scheme using curved triangles. In *ICIP95*, Washington DC, October 1995.

[41] J. Liu. *Fractal Block Coding Techniques in Image Compression.* PhD thesis, Dublin City University, 1994.

[42] D. Saupe. The futility of square isometries in fractal image compression. In *ICIP 96*, pages 161–164, Lausanne, Switzerland, 1996.

[43] George W. Wornell. *Signal Processing with Fractals: A Wavelet Based Approach.* Prentice-Hall, Englewood Cliffs, New Jersey, 1996.

[44] B. Simon. Image coding using overlapping fractal transform in the wavelet domain. In *ICIP 96*, pages 177–180, Lausanne, Switzerland, September 1996.

[45] Geoffrey M. Davis. A wavelet based analysis of fractal image compression. Available via the web at `http://www.cs.dartmouth.edu/~gdavis/papers/ieee.ps.gz`.