Two- and Three-Dimensional Coding Schemes for Wavelet and Fractal-Wavelet Image Compression

by

Simon K. Alexander

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Applied Mathematics

Waterloo, Ontario, Canada, 2001

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESES

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis presents two novel coding schemes and applications to both two- and three-dimensional image compression. Image compression can be viewed as methods of functional approximation under a constraint on the amount of information allowable in specifying the approximation.

Two methods of approximating functions are discussed: Iterated function systems (IFS) and wavelet-based approximations. IFS methods approximate a function by the fixed point of an iterated operator, using consequences of the Banach contraction mapping principle. Natural images under a wavelet basis have characteristic coefficient magnitude decays which may be used to aid approximation.

The relationship between quantization, modelling, and encoding in a compression scheme is examined. Context based adaptive arithmetic coding is described. This encoding method is used in the coding schemes developed. A coder with explicit separation of the modelling and encoding roles is presented: an embedded wavelet bitplane coder based on hierarchical context in the wavelet coefficient trees. Fractal (spatial IFSM) and fractal-wavelet (coefficient tree), or IFSW, coders are discussed. A second coder is proposed, merging the IFSW approaches with the embedded bitplane coder.

Performance of the coders, and applications to two- and three-dimensional images are discussed. Applications include two-dimensional still images in greyscale and colour, and three-dimensional streams (video).

Acknowledgements

I wish to thank my supervisor Edward Vrscay for support and encouragement during the writing of, and work preceding, this thesis. Many thanks must as well be extended to my readers, George Freeman and Stanley Lipschitz. Their suggestions for final polishing were also appreciated. I am further indebted for tireless proofreading by Patricia Lindsay. Thank you.

Contents

List of Figures

Introduction

Over the past several decades the display, storage, and transportation of digital images has moved from obscurity to the commonplace. Today many people interact with digital imagery, in one form or another, on a daily basis. The amount of raw data to be transported demands the development of compression approaches. Recent expansion in use of three-dimensional images such as video streams and some medical imaging modalities makes the data compression problem more pressing.

A certain amount of compression may be achieved without loss of information, using *lossless* methods. However, many applications call for an order of magnitude more reduction of size than is possible to meet using such methods. Once the realm of *lossy* compression is entered, a trade-off in size vs. distortion must be made. The fundamental problem is to meet the competing goals of accurate approximation and reduction in storage requirements.

The aim of this thesis is to present some methods of addressing these goals. Essentially there are two questions that have driven this work:

1. In theory, the modelling and encoding of an image fill separate roles. In practice, many successful encoding methods mix these functions in an ad-hoc way. Is it beneficial, and practical, to make this separation explicit?

2. In many ways, 'pure' wavelet compression methods have outperformed both 'pure' fractal compression and hybrid fractal-wavelet schemes. On the other hand, fractal methods have some attractive properties. Can one use the best parts of both methods?

Attempting to answer these questions involves the interface between three somewhat distinct areas: entropy coding, wavelet transforms, and fractal transforms. Thus, in order to meet these goals it is necessary to present an overview of all three areas. In doing so, the background theory has mostly been separated from the specifics of imaging and image compression. The first three chapters are primarily devoted to presenting the necessary background. For this reason, introductory discussion of image compression does not occur until the beginning of Chapter 4. A reader having no familiarity with the area might wish to start with the introductory section of that chapter.

Layout of the thesis

Chapter 1 introduces wavelets. The groundwork for 'pure' wavelet methods is laid here. The chapter begins by introducing wavelet bases and the one-dimensional wavelet transform. Following this, discretization and its relation the Multiresolution Analysis (MRA) are presented. The Mallat Algorithm, an $O(n)$ algorithm for performing the discrete wavelet transform, is described along with connections to subband filtering schemes. Perfect reconstruction conditions are discussed. After this basic framework is in place, several extensions to the theory are needed before it may be applied to images. Later sections of the chapter discuss compact support, biorthogonal bases, and bases in an interval. The chapter concludes by discussing the extension of the theory to two and three dimensions.

Following the presentation of wavelets, Chapter 2 introduces the Iterated Function Systems (IFS). IFS methods attempt to take advantage of the self-similarity occurring in natural images. The chapter begins with an overview of the basic IFS theory, which is a consequence of the Banach contraction mapping principle. The inverse problem for IFS is presented, and the collage theorem introduced as a response. The theory shown to this point is not applicable to images; the last two sections of the chapter are devoted to extending the IFS theory to grey-level maps (IFSM) and then to wavelet coefficient trees (IFSW). The IFSM method forms the basis for 'pure' fractal image compression, whilst IFSW is the core of many fractal-wavelet compression methods.

Chapter 3 presents the ideas behind entropy coding. After a brief introduction to relevant results from information theory, the chapter concentrates on a single coding method, arithmetic coding. The algorithm is discussed, along with practical considerations necessary to make this approach usable in a computer. Additional sections discuss adaptive coding, and the use of context based coders.

These first three chapters lay the necessary framework for discussing the image coders devised and presented in this thesis. Chapter 4 begins with a brief overview of image compression. Following sections of the chapter are devoted to 'pure' IFS, 'pure' wavelet, and hybrid fractal-wavelet methods. In each of these, standard methods are outlined, and some discussion of the merits of each method is made.

Two original image coding methods are then presented. The first is a embedded wavelet bitplane coder. This algorithm addresses the first question given above; it implements an approach distinctly separating the modelling and coding of the image. The arithmetic encoder presented in Chapter 3 is used to encode, with adaptive context-based models, the wavelet coefficient tree of an image. Results from this algorithm are compared with those of the well known SPIHT algorithm.

The second method involves a hybrid fractal-wavelet approach that attempts to take advantage of the strengths of both wavelet and fractal methods. Standard fractal-wavelet methods are briefly discussed. A hybrid of these methods and the aforementioned bitplane coder is presented. Comparisons of the two approaches are then made.

The coding schemes are both presented as two-dimensional methods. The final part of the chapter is devoted to applying the bitplane method to three-dimensional data. Issues surrounding colour images are discussed, and some comparisons with the industry standard MPEG-2 video compression scheme are made.

Some concluding remarks are made in Chapter 5, followed by some additional material in appendices. Appendix A provides all of the still images used as data, to avoid unnecessary duplication. Throughout the thesis, these images may be referred to by name. Appendix B briefly describes some software that was developed by the author in order to facilitate implementing the coders described herein. The packages discussed in this appendix are quite general, and should be applicable to other work in compression and image processing.

Finally, for convenience, a list of acronyms and index are provided. In the index entries, definitions or primary references are indicated by boldface page numbers. An electronic version of this thesis will be available from the University of Waterloo's electronic thesis database. Additionally, Postscript and PDF versions of the text, and related software, will be placed on the web at:

`http://links.uwaterloo.ca/~simon .`

Preliminaries and Notation

Throughout this thesis, new notation will be introduced as needed. Some common notations are included here.

Sets

Denote the following classical sets as shown

$$\begin{aligned}
\mathbb{R} &= \text{the set of real numbers,} \\
\mathbb{C} &= \text{the set of complex numbers,} \\
\mathbb{Z} &= \text{the set of integers,} \\
\mathbb{N} &= \text{the positive integers including } 0 = \{0, 1, 2, 3, \ldots\}, \\
\mathbb{N}^+ &= \text{the positive integers} = \{1, 2, 3, \ldots\}.
\end{aligned}$$

Limits of Integration

Where limits on an integral or summation are not shown, they are assumed to be infinite. That is

$$\int f(x)\,dx = \int_{-\infty}^{+\infty} f(x)\,dx$$

$$\sum_n x_n = \sum_{n=-\infty}^{+\infty} x_n$$

and similarly,

$$\sum_{j,k} x_{j,k} = \sum_{j=-\infty}^{+\infty} \sum_{k=-\infty}^{+\infty} x_{j,k}$$

Complex Conjugate

The complex conjugate of a variable or function will be denoted by an overbar, such as

$$\overline{x} \qquad \text{and} \qquad \overline{f(x)}$$

Fourier Transform

Denote the continuous Fourier transform as

$$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(x) e^{-i\omega x}\,dx$$

and the discrete Fourier transform for a signal f of period N is

$$\hat{f}[k] = \sum_{n=0}^{N-1} f[n] e^{-2i\pi n \frac{k}{N}}.$$

$\mathcal{L}^p$ Spaces

The space $\mathcal{L}^p(\mathbf{X}), p \in \mathbb{N}^+$ of Lebesgue integrable functions on $\mathbf{X}$ is defined as the set

$$\mathcal{L}^p(\mathbf{X}) = \{f : \mathbf{X} \to \mathbb{R} \mid \int_{\mathbf{X}} |f(x)|^p\,dx < \infty\}.$$

ℓ^p Spaces

The space $\ell^p(\mathbf{X})$, $p \in \mathbb{N}^+$ of p-summable sequences on $\mathbf{X}$ is defined as the set

$$\ell^p(\mathbf{X}) = \{\{x_n\}_{n\in\mathbb{N}}, x_n \in \mathbf{X} \mid \sum_{n=0}^{\infty} |x_n|^p < \infty\}.$$

Kronecker delta

Denote the Kronecker delta as

$$\delta_j^k = \begin{cases} 1 & j = k, \\ 0 & j \neq k. \end{cases}$$

Proofs, Theorems, and Examples

Proofs will be ended by the symbol ■ at the far side of the text. Similarly, theorems, examples, etc. will be delimited at the end by the symbol □.

Algorithms

Where it is appropriate to do so, algorithms may be presented in pseudo-code. Notation contained in these code blocks is mostly self-explanatory. The notation $\Leftarrow$ will be used for "takes the value of", and braces $\{comment\}$ used for comments, for example.

Algorithm 0.1 Example pseudo-code

```
x ⇐ 1.0 {x is now 1.0}
for i = 0 until 10 do
   do something ...
end for
```

Several algorithms discussed in this thesis are designed to be implemented in a computer; it is hoped that their operation is made clearer by presentation in this format.

Chapter 1

Wavelet Transforms

Wavelets are introduced in the form of the one-dimensional real wavelet transform, and then discussion is extended to the discretization of this process. The multiresolution approach is detailed, arriving at the Mallat Algorithm and its connection to filter banks. Extensions to the basic theory, including biorthogonal bases and higher dimensional transforms are then presented. In this manner the general framework is laid for wavelets applied to finite two- and three-dimensional images.

1.1 Overview

Haar Wavelets

Much of the background for what is now called *wavelets* was laid by Haar in 1910. He demonstrated [29] the simple piecewise constant function could be used to generate an orthonormal basis of $\mathcal{L}^2(\mathbb{R})$. As a motivating example, first examine this basis. Consider the function on $\mathbb{R}$:

$$\psi(t) = \begin{cases} 1 & 0 \leq t < \frac{1}{2}, \\ -1 & \frac{1}{2} \leq t < 1, \\ 0 & \textit{elsewhere}. \end{cases} \tag{1.1}$$

The function ψ is often known as a *mother wavelet*, as it is used to generate a family of wavelets: Take the following translations and dilations of equation 1.1,

$$\psi_{j,k}(x) = 2^{-j/2}\,\psi(2^{-j}\,x - k)\ , \tag{1.2}$$

and note that $\psi_{0,0}(t) = \psi(t)$.

Proposition 1.1 *$\{\psi_{j,k}(t)\}_{(j,k)\in\mathbb{Z}^2}$ forms an orthonormal basis (called the Haar basis) for $\mathcal{L}^2(\mathbb{R})$.* □

PROOF proof for a more general case will be given later. ■

For any function $f \in \mathcal{L}^2(\mathbb{R})$, f may be represented as

$$f = \sum_{j,k} <f, \psi_{j,k}> \psi_{j,k},$$

since $\{\psi_{j,k}(t)\}_{(j,k)\in\mathbb{Z}^2}$ forms a basis of $\mathcal{L}^2(\mathbb{R})$. Briefly, call the mapping of f to the Haar basis a *wavelet transform*, and the $c_{j,k} = <f, \psi_{j,k}>$ wavelet coefficients. All of this will be made more concrete in the sections to follow. It turns out that this approach is not limited to the function given in Equation 1.1, which is but one example of a mother wavelet ψ generating a basis such as above.

Transformation to a wavelet basis retains some spatial information, while providing frequency information as well (as compared to the frequency representation after a Fourier transform). Natural[1] signals (images) tend to have most of their energy in low-frequency components. Thus after the wavelet transform is performed, much of the energy (coefficient magnitude) will be concentrated in a few of the wavelet coefficients. If the smaller magnitude wavelet coefficients are truncated, the result is an approximation of the function. By ignoring some coefficients, the signal is being approximated with less information, in some sense. Without going into the details of encoding yet (see Chapter 4), the signal has been "compressed" into a few coefficients.

As an example, consider as input samples from the function $\sin(x^2)$ on $[0, 2\pi]$. From Equation 1.1 it is clear that under the Haar basis, the above approximations will be piecewise constant. Better approximations can be made by other wavelet bases. Figure 1.1 shows the function $\sin(x^2)$ on $[0, 2\pi]$, transformed to each of the Haar, Daubechies 2, and Antonini 9/7 bases[2]. After transformation, all wavelet coefficients whose magnitude was less than about twenty-five percent (this value is different for each of the bases) of the maximum magnitude were set to zero. Out of a total of 512 coefficients, the 22 largest coefficients were retained in each case. After the inverse transformation, the approximations are as shown.

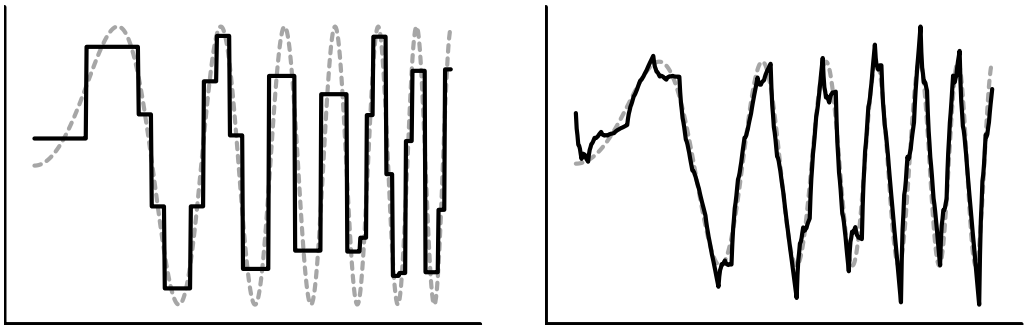
1.2 Wavelets

Analytic Wavelets

There is no theoretical need for restriction to real-valued wavelets. Complex valued, or analytic, wavelets have the important property of separating phase and amplitude information of a signal. As the applications to be discussed in this thesis do not utilize analytic wavelets, no further discussion will be presented, and hereafter the assumption of real valued functions is made. For discussion of analytic wavelets see [37, 18].

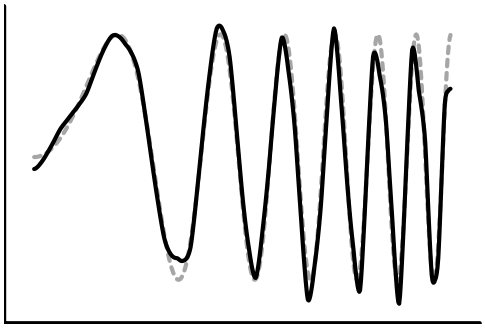
[1] I.e., those occurring from natural processes.

[2] These will be defined later. The Haar and Daubechies 4 are orthogonal bases of $\mathcal{L}^2(\mathbb{R})$, whilst the Antonini 9/7 is a symmetric biorthogonal basis.



(a) Haar

(b) Daubechies 2



(c) Antonini 9/7 (biorthogonal)

Figure 1.1: Approximations to $\sin(x^2)$ on $[0, 2\pi]$ resulting from truncating coefficients below about 25% of maximum magnitude for (a) Haar (b) Daubechies 2 (c) Antonini 9/7 bases. The approximations are the result of retaining 22 coefficients.

Real Wavelets

A wavelet is a function $\psi \in \mathcal{L}^2(\mathbb{R})$ with zero average, that is,

$$\int \psi(x)\, dx = 0. \tag{1.3}$$

From ψ, a family of translations and dilations is generated as follows,

$$\psi_{a,b}(x) = |a|^{-1/2}\psi\left(\frac{x-b}{a}\right), \quad a, b \in \mathbb{R},\, a \neq 0. \tag{1.4}$$

The wavelet function is normalized, which gives $\|\psi\| = \|\psi_{a,b}\| = 1$. It must also satisfy an *admissibility condition*,

$$2\pi \int |\xi|^{-1} |\hat{\psi}(\xi)|^2\, d\xi < \infty. \tag{1.5}$$

The Wavelet Transform

Definition 1.1 (Continous Wavelet Transform) *For any $f \in \mathcal{L}^2(\mathbb{R})$, the wavelet transform of f is defined as*

$$(Wf)(a,b) = <f, \psi_{a,b}> \tag{1.6}$$

$$= \int f(x)|a|^{-1/2}\overline{\psi\left(\frac{x-b}{a}\right)}\,dx. \tag{1.7}$$

□

The Continuous Wavelet Transform (CWT) describes the mapping of f onto a basis defined by the mother wavelet ψ. It is natural to ask what conditions suffice in order to be able to recover f from the coefficients $<f, \psi_{a,b}>$.

Theorem 1.1 (Reproducibility of f) *Let $\psi \in \mathcal{L}^2(\mathbb{R})$ be a real function satisfying*

$$C_\psi = 2\pi \int |a|^{-1/2}|\hat{\psi}(\xi)|^2\,d\xi < \infty.$$

Then for all $f \in \mathcal{L}^2(\mathbb{R})$

$$f(t) = \frac{1}{C_\psi}\int\int (Wf)(a,b)\,|a|^{-1}\psi\left(\frac{t-b}{a}\right)\frac{da\,db}{a^2}. \tag{1.8}$$

□

PROOF See [18] or [37]. ■

This key result for the continuous transform describes the conditions under which f may be recoverably decomposed into projections on the spaces defined by the family $\psi_{a,b}$ (i.e., Equation 1.4).

In applications, often discrete signals (a continuous signal that is sampled at a uniform interval) are important. In particular, image compression will apply to discrete signals, and so we concentrate on a discrete version of the wavelet transform.

1.3 Discrete Wavelets

1.3.1 Discretization of the Continuous Wavelet Transform

Consider the discretization of the continuous wavelet transform. Recall the family of functions in Equation 1.4

$$\psi_{a,b}(x) = |a|^{-1/2}\psi\left(\frac{x-b}{a}\right), \quad a,b \in \mathbb{R},\, a \neq 0\,.$$

The wavelet family may be discretized by fixing particular $a_0, b_0 \geq 1$ in place of a, b, and scaling them by $n, m \in \mathbb{Z}$. The resulting family of functions will be

$$\psi_{a,b}(x) = a_0^{-m/2}\psi\left(\frac{x - nb_0a_0^m}{a_0^m}\right) \tag{1.9}$$

$$= a_0^{-m/2}\psi(a_0^{-m}x - nb_0). \tag{1.10}$$

Having discretized the family of translations and dilations of ψ, it remains to determine if a "discrete admissibility condition" (i.e. a discrete version of Equation 1.5) exists. Additionally, a discrete version of Theorem 1.1 would be useful. It turns out that there is no direct analogue to Equation 1.5. However, under certain constraints on a_0, b_0 and ψ, there exists a function $\tilde{\psi}_{m,n}$ such that for all $f \in \mathcal{L}^2(\mathbb{R})$,

$$f = \sum_{m,n} <f, \psi_{m,n}> \tilde{\psi}_{m,n}\,.$$

For details of this result, see [18]. In application to digital signals, it is especially convenient to choose the values $a_0 = 2$, $b_0 = 1$. This special case will be the only one considered in the remainder of this development.

1.3.2 Multiresolution Analysis

Definition 1.2 (resolution) *Under the previously mentioned restrictions, the translations and dilations of the mother wavelet Equation 1.10 become*

$$\psi_{j,k}(x) = 2^{-j/2}\psi(2^{-j}x - k).$$

In the following, we will refer to a ***resolution*** *of 2^{-j}, which has a scaling parameter of 2^j. Dilation of the wavelet by 2^j carries signal variations at the resolution 2^{-j}. In other words, the scale parameter 2^j is the inverse of the resolution 2^{-j}.* □

This concept of a Multiresolution Analysis (MRA) was introduced in [36]. In this presentation the definition and development follow more closely the one found in in [18]. A slightly more general form is discussed in in [37].

The MRA will not define a wavelet ψ; rather, given an MRA it will be possible to (non-uniquely) construct a wavelet. An important aspect of the MRA will be the so-called *scaling function*, ϕ. Similar to the wavelet ψ, the translations and dilations of the scaling function are interesting. These form the family $\{\phi_{j,k}\}$ where

$$\phi_{j,k}(x) = 2^{-j/2}\phi(2^{-j}x - k). \tag{1.11}$$

Definition 1.3 (MRA) *Any sequence $\{V_j\}_{j\in\mathbb{Z}}$ of closed subspaces of $\mathcal{L}^2(\mathbb{R})$ is called an MRA if it has the following properties:*

$$\forall j \in \mathbb{Z} \qquad V_{j+1} \subset V_j \tag{1.12}$$

$$\lim_{j\to+\infty} V_j = \bigcap_{j=-\infty}^{+\infty} V_j = \{0\} \tag{1.13}$$

$$\lim_{j\to-\infty} V_j = \text{Closure}\left(\bigcup_{j=-\infty}^{+\infty} V_j\right) = \mathcal{L}^2(\mathbb{R}) \tag{1.14}$$

$$f \in V_j \Leftrightarrow f(2^j\cdot) \in V_0 \tag{1.15}$$

$$f \in V_0 \Rightarrow f(\cdot - j) \in V_0 \; \forall j \in \mathbb{Z} \tag{1.16}$$

$$\exists\, \phi \in V_0 \quad \textit{such that} \quad \{\phi_{0,n} : n \in \mathbb{Z}\} \quad \textit{is an orthonormal basis of} \; V_0\,. \tag{1.17}$$

□

Before showing how the MRA is related to wavelet bases, some exploration of the MRA properties is appropriate.

Projections

Definition 1.4 *Denote the orthogonal projection operator onto a space X as P_X.* □

Let P_{V_j} denote the orthogonal projection operator onto V_j. Consider a function f in $\mathcal{L}^2(\mathbb{R})$. Property 1.14 implies that as j tends to infinity,

$$\lim_{j\to+\infty} \|P_{V_j} f\| = 0,$$

while similarly Property 1.13 implies that

$$\lim_{j\to-\infty} \|f - P_{V_j} f\| = 0.$$

Thus as the resolution is decreased, i.e. $2^{-j} \to 0$, detail is lost; increasing the resolution , $2^{-j} \to \infty$, increases detail.

Multiresolution

The Multiresolution aspect is imposed by Property 1.15 which implies that all the V_j spaces are scaled versions of the central space V_0. Property 1.16, taken with Property 1.15 implies

$$f \in V_j \Rightarrow f(\cdot - 2^j n) \; \in V_j \quad \forall n \in \mathbb{Z}\,.$$

From Equation 1.11 we have the family $\{\phi_{j,k}\}$. Together, the Properties 1.17 and 1.15 imply [18] that for each $j \in \mathbb{Z}$, the set $\{\phi_{j,n}\}_{n\in\mathbb{Z}}$ forms an orthonormal basis for V_j.

As $\phi \in V_0 \subset V_{-1}$, and $\{\phi_{-1,n}\}_{n\in\mathbb{Z}}$ is an orthonormal basis for V_{-1}, ϕ may be written as

$$\phi = \sum_n <\phi, \phi_{-1,n}> \phi_{-1,n}\,. \tag{1.18}$$

Let

$$h_n = <\phi, \phi_{-1,n}>, \tag{1.19}$$

then the fundamental scaling result for the scaling function $\phi(x)$ is

$$\phi(x) = \sqrt{2}\sum_n h_n \phi(2x-n). \tag{1.20}$$

Implications

The central point of the MRA is that these properties are sufficient to build a wavelet basis of $\mathcal{L}^2(\mathbb{R})$.

Theorem 1.2 *Let $\{V_n\}_{n\in\mathbb{Z}}$ be an MRA. Then there exists an associated orthonormal wavelet basis $\{\psi_{j,k}\}_{(j,k)\in\mathbb{Z}^2}$ such that*

$$\psi(x) = \sqrt{2}\sum_n (-1)^{n-1} h_{-n-1}\phi(2x-n) \tag{1.21}$$

$$P_{V_{j-1}} = P_{V_j} + \sum_k < \cdot, \psi_{j,k} > \psi_{j,k} \tag{1.22}$$

where h_n is defined as in 1.19, and $\psi_{j,k}$ as defined in Equation 1.2. □

PROOF see [18]. ■

It is important to note that this does not uniquely determine ψ. In particular we may choose (for future simplicity)

$$\psi = \sum_n g_n \phi_{-1,n} \qquad \text{where } g_n = (-1)^n h_{-n+1}, \tag{1.23}$$

yielding

$$\psi(x) = \sqrt{2}\sum_n (-1)^n h_{-n+1}\phi(2x-n). \tag{1.24}$$

1.3.3 Mallat Algorithm

From Property 1.12, V_{j+1} is a proper subset of V_j. Hence, there is an orthogonal complement of V_{j+1} in V_j, call it W_j. Thus $W_j \perp V_j$ and

$$V_j = V_{j+1} \oplus W_{j+1}\,. \tag{1.25}$$

Thus if Equation 1.25 is recursively applied

$$V_j = V_N \oplus \bigoplus_{k=0}^{N-j-1} W_{N-k}\,, \text{for } j < N \tag{1.26}$$

Call the W_j *detail spaces*, and define an operator for orthogonal projection onto the detail space

$$D_{j+1} = P_j - P_{j+1}\,, \tag{1.27}$$

where P_j is the orthogonal projection operator defined in Equation 1.22.

Denote $P_j f = f_j$, $D_j f = d_j$. Thus $d_{j+1} = f_j - f_{j+1}$ and $f_j \in V_j = V_{j+1} \oplus W_{j+1}$ is such that $f_j = f_{j+1} + d_{j+1}$. This shows the decomposition of f_j into an *average* part f_{j+1} (f projected onto the coarser space V_{j+1}) and a *detail* part d_{j+1} which represents the information lost moving from resolution 2^{-j} to 2^{-j-1}, i.e., the information in f_j that is not in f_{j+1}.

From Equation 1.22 it follows that $\{\psi_{j,k}\}$ forms an orthonormal basis for W_j. Let $f \in \mathcal{L}^2(\mathbf{X})$ and consider the projection of f onto the V_j, W_j spaces, then Equation 1.26 may be written as

$$P_j f = P_N f \oplus \bigoplus_{k=0}^{N-j-1} D_{N-k} f\,, \text{for } j < N \tag{1.28}$$

Now if f is given as a sequence[3] of discrete samples, the units may be arranged so that the resolution of the signal is considered to be 2^0, i.e. $j = 0$. This leads to an algorithm for computing the wavelet coefficients. Recall from Equation 1.23

$$\psi = \sum_n g_n \phi_{-1,n} \qquad \text{where } g_n = <\psi, \phi_{-1,n}> = (-1)^n h_{-n+1}. \tag{1.29}$$

[3] In practice finite and often of dyadic length.

By Equation 1.2,

$$\begin{aligned}
\psi_{j,k}(x) &= 2^{-j/2}\psi(2^{-j}x-k) \\
&= 2^{-j/2}\sum_n g_n \phi_{-1,n}(2^{-j}x-k) \\
&= 2^{-j/2}\sum_n g_n 2^{1/2}\phi(2^{-j+1}x-2k-n) && \text{by Equation 1.2} \\
&= \sum_n g_n 2^{(-j+1)/2}\phi(2^{-j+1}x-2k-n) \\
&= \sum_n g_n 2^{-(j-1)/2}\phi(2^{-(j-1)}x-(2k+n)) \\
&= \sum_n g_n \phi_{j-1,2k+n}(x) && \text{by Equation 1.2} \\
&= \sum_n g_{n-2k}\phi_{j-1,n}(x) && \text{by change of variable } n \Rightarrow n-2k
\end{aligned} \tag{1.30}$$

Thus

$$<f,\psi_{j,k}>=<f,\sum_n g_{n-2k}\phi_{j-1,n}>=\sum_n \overline{g_{n-2k}}\,<f,\phi_{j-1,n}>\,. \tag{1.31}$$

Similarly, by Equation 1.2 (by way of Equation 1.20),

$$\begin{aligned}
\phi_{j,k}(x) &= 2^{-j/2}\phi(2^{-j}x-k) \\
&= \sum_n h_{n-2k}\phi_{j-1,n}(x).
\end{aligned} \tag{1.32}$$

Therefore

$$<f,\phi_{j,k}>=\sum_n \overline{h_{n-2k}}\,<f,\phi_{j-1,n}>\,. \tag{1.33}$$

Equations 1.31 and 1.33 give a recursive relationship for calculating $<f,\psi_{j,k}>$, and $<f,\phi_{j,k}>$. The two equations represent the same operations: convolution with a filter ($\bar{g}, \bar{h}$ respectively) followed by decimating by a factor of two[4].

The notation of subscripts on filters at different levels in this recursion becomes cumbersome. In order to simplify certain equations, the following notation for discrete signals is introduced.

[4]I.e., take only the even entries.

Notation 1.1 *Let $\{x_n\}_{n\in\mathbb{Z}}$ be a sequence, often called a **signal**. Considering only real valued sequences, denote the following quantities:*

$$x[n] = x_n \tag{1.34}$$
$$\bar{x}[n] = x[-n] = x_{-n} \qquad \text{as } x_n \in \mathbb{R} \tag{1.35}$$
$$y[n] = x[2n] = x_{2n} \qquad \textit{downsampling} \tag{1.36}$$
$$\check{x}[n] = \begin{cases} x[p] & n = 2p \text{ for some } p \in \mathbb{Z} \\ 0 & n = 2p+1 \end{cases} \qquad \textit{upsampling} \tag{1.37}$$
$$y \star x[n] = \sum_k y[n-k]x[k] \qquad \textit{discrete convolution} \tag{1.38}$$

□

In keeping with the above notation, define $a_j[k] = <f, \phi_{j,k}>$ and $d_j[k] = <f, \psi_{j,k}>$, and call these the *average* and *detail* signals at level j, respectively. Thus the original sampling of f is a_0. This leads to the following characterization.

Decomposition Algorithm

In this new notation, Equation 1.31 and Equation 1.33 may be recast as

$$a_j[k] = \sum_n \bar{h}[n-2k]a_{j-1}[n] \qquad \text{and} \qquad d_j[k] = \sum_n \bar{g}[n-2k]d_{j-1}[n]. \tag{1.39}$$

Now, having decomposed the signal, consider how to go about reconstructing it. Recall from Equation 1.2 and Theorem 1.2 that $\{\phi_{j,k}\}$,$\{\psi_{j,k}\}$ form orthonormal bases of V_j, W_j, respectively. Equation 1.25 states that

$$V_j = V_{j+1} \oplus W_{j+1}.$$

Hence $\phi_{j,k}$ may be decomposed under the union of the two bases

$$\phi_{j-1,k} = \sum_n <\phi_{j-1,k}, \phi_{j,n}> \phi_{j,n} + \sum_n <\phi_{j-1,k}, \psi_{j,n}> \psi_{j,n}\,. \tag{1.40}$$

Note from Equation 1.32 that

$$\begin{aligned} <\phi_{j-1,k}, \phi_{j,n}> &= <\phi_{j-1,k}, \sum_m h_{m-2n}\phi_{j-1,m}> \\ &= \sum_m \overline{h_{m-2n}} <\phi_{j-1,k}, \phi_{j-1,m}> \\ &= \sum_m \overline{h_{m-2n}}\, \delta_m^k \qquad \text{as } \{\phi_{j,k}\} \text{ is an orthonormal basis of } V_j \\ &= \overline{h_{k-2n}}\,. \end{aligned}$$

Similarly, by Equation 1.30

$$<\phi_{j-1,k}, \psi_{j,n}> = \overline{g_{k-2n}}\,.$$

Given the above results, restate Equation 1.40 as

$$\phi_{j-1,k} = \sum_n \overline{h_{k-2n}} \phi_{j,n} + \sum_n \overline{g_{k-2n}} \psi_{j,n} \ , \tag{1.41}$$

which implies that

$$\begin{aligned} <f, \phi_{j-1,k}> &= \sum_n \overline{\overline{h_{k-2n}}} <f, \phi_{j,n}> + \sum_n \overline{\overline{g_{k-2n}}} <f, \psi_{j,n}> \\ &= \sum_n h_{k-2n} <f, \phi_{j,n}> + \sum_n g_{k-2n} <f, \psi_{j,n}> \ . \end{aligned} \tag{1.42}$$

Recombination Algorithm

Writing Equation 1.42 in the discrete signal notation yields

$$a_{j-1}[k] = \sum_n h[k-2n] a_j[n] + \sum_n g[k-2n] d_j[n] \tag{1.43}$$

which is in some sense the reverse procedure to that employed in the decomposition stage. The average and detail signals a_j, d_j are first *upsampled* by a factor of two, i.e. interleaved with zero values, and then convolved with the h and g filters, respectively.

Implementation

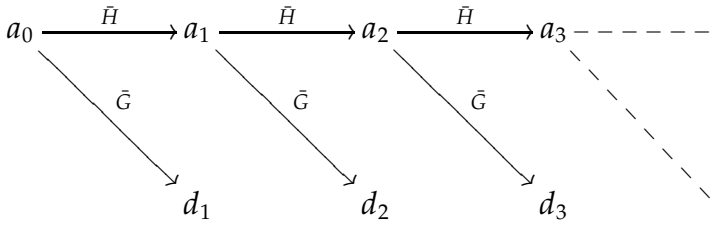
In practice this procedure operates on finite length signals, and so the process is only iterated a finite number of times. It is convenient for applications to start off with a_0 a dyadic length, as this way a_j and d_j are both exactly half the length of a_{j-1}, and hence will fit in the storage of a_{j-1}. Let us adopt an "operator" notation for the action of the filters in the decomposition step. That is,

$$\begin{aligned} a_j[k] = \sum_n \bar{h}[n-2k] a_{j-1}[n] \qquad &\Longrightarrow \qquad a_j = \bar{H} a_{j-1} \\ d_j[k] = \sum_n \bar{g}[n-2k] d_{j-1}[n] \qquad &\Longrightarrow \qquad d_j = \bar{G} d_{j-1} \end{aligned}$$

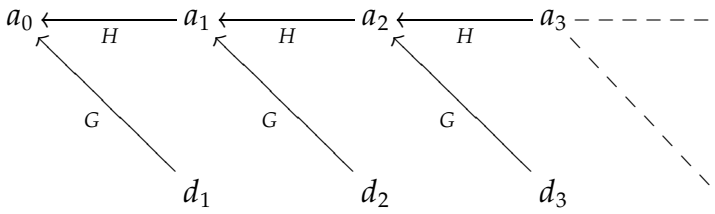
and similarly for the recombination step,

$$a_{j-1}[k] = \sum_n h[k-2n] a_j[n] + \sum_n g[k-2n] d_j[n] \Longrightarrow a_{j-1} = H a_j + G d_j.$$

With this notation, Figure 1.2 illustrates the decomposition and recombination steps. This process is, in fact, a well understood part of signal processing theory, called *sub-band filtering*.



(a) decomposition



(b) recombination

Figure 1.2: Mallat algorithm: (a) the decomposition of Equation 1.39 (b) the recombination of Equation 1.43

Example 1.1 For convenience, introduce the following *interval notation*

$$I_{[a,b]}(x) = \begin{cases} 1 & a \leq x \leq b, \\ 0 & \text{otherwise.} \end{cases}$$

Beginning with the scaling function $\phi(x) = I_{[0,1]}(x)$, the filter h is easily determined, since (Equation 1.19)

$$h[n] = <\phi(x), 2^{1/2}\phi(2x-n)> .$$

This implies that

$$h[n] = \begin{cases} 2^{-1/2} & \text{for} \quad n = 0, 1 \\ 0 & \text{otherwise.} \end{cases}$$

Now applying Equation 1.23 yields

$$\begin{aligned} 2^{-1/2}\psi(x) &= \sum_n (-1)^n h[-n+1]\phi(2x-n) \\ &= (-1)^1 h[0]\phi(2x-1) + (-1)^0 h[1]\phi(2x) \\ &= 2^{-1/2}\phi(2x) - 2^{-1/2}\phi(2x-1). \end{aligned}$$

Therefore

$$\begin{aligned}\psi(x) &= \phi(2x) - \phi(2x + -1) \\ &= I_{[0,1/2]} - I_{[1/2,0]}\,.\end{aligned}$$

Which is the Haar wavelet given in Equation 1.1. □

1.3.4 Subband Filtering

Electrical engineers have developed a theory of filtering systems [33, 44]. In this terminology Equations 1.39 and 1.43 form the *analysis* and *synthesis* filtering steps, respectively, in a *subband filtering scheme*. It is convenient[5] to adopt some of the filter-bank terminology for discussing the implementation. For an introduction to filter bank theory with application to compression see [48]. In practice Finite Impulse Response (FIR) filters are used, that is, filters with a finite number of non-zero entries.

Notation 1.2 (z-transform) *Consider a sequence* $\{x_n\}_{n\in\mathbb{Z}}$. *The sequence may be represented by a formal series:*

$$x(z) = \sum_{n=-\infty}^{+\infty} x_n z^n \tag{1.44}$$

this is often called ***z-notation****, or the* ***z-transform****.* □

Note that if $z = e^{i\xi}$ is restricted to lie on the unit circle, then equation 1.44 is simply a Fourier series. It is convenient at times to consider a general $z \in \mathbb{C}$, hence this notation. The z-transform shares several properties with the discrete Fourier transform, most importantly convolutions are mapped to multiplications under the transform

$$f \star g \text{ under z-transform is } f(z)g(z).$$

It is also simple to express the "downsampling" and "upsampling" operations in the z-transform notation. Recall (equation 1.37) the notation for upsampling, $\check{x}$, then

$$y(z) = \check{x}(z) = \sum_{n=-\infty}^{+\infty} x[n]z^{2n} = x(z^2). \tag{1.45}$$

Similarly for downsampling (equation 1.36)

$$y(z^2) = \sum_{n=-\infty}^{+\infty} y[n]z^{2n} = \sum_{n=-\infty}^{+\infty} x[2n]z^{2n} = \frac{1}{2}\left[x(z) + x(-z)\right]. \tag{1.46}$$

Consider a general case of a filter bank. The filters are non-optimal, and the analysis and synthesis filters may be different. Following the notation introduced in §1.3.3, define a filter bank as follows.

[5] This work may be carried out in terms of Fourier series also [37].

Definition 1.5 (Filter Bank) *A* filter bank *consists of a decomposition stage and a recombination stage. In the decomposition stage a signal x is convolved with a low-pass filter h and high-pass filter g, followed by subsampling the results by a factor of two, yielding*

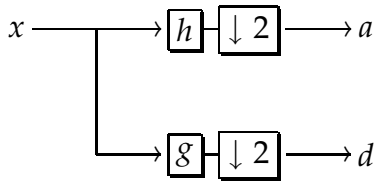
$$a = x \star h[2n] \quad \text{and} \quad d = x \star g[2n]. \tag{1.47}$$

The reconstruction stage convolves the upsampled (zero-interleaved) versions of a and d with the reconstruction filters $\tilde{h}$ and $\tilde{g}$:

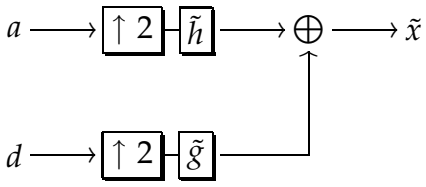
$$\tilde{x} = \check{a} \star \tilde{h}[n] + \check{d} \star \tilde{g}[n]. \tag{1.48}$$

□

The action of the filter bank is easily shown graphically. The decomposition, or *analysis* step can be represented as in Figure 1.3(a). The input signal x is transformed



(a) analysis



(b) synthesis

Figure 1.3: *Subband filtering scheme* (a): The *analysis* step convolves x with filters h, g, followed by subsampling by a factor of 2 (retaining only the even entries). (b): The *synthesis* step takes as input an average, a, and detail, d. Each is first upsampled (interleaving with zeros) by a factor of two, then filtered with $\tilde{h}$, $\tilde{g}$, respectively. These results are summed (piecewise) to yield the output signal $\tilde{x}$.

into two signals of half the length of x. A low pass filter h results in the average signal a, and a high pass filter gives the detail signal d. In both cases, the process involves convolution with the filter followed by decimation of the signal by factor of two.

Similarly the reconstruction, or *synthesis*, step is shown in Figure 1.3(b). In this case, the average and detail signals are first "upsampled" (i.e. $\check{a}$ in Notation 1.1). The resulting sequences are convolved with filters $\tilde{h}$, $\tilde{g}$ respectively. The resulting sequences are summed piecewise, yielding the resulting sequence $\tilde{x}$.

Perfect Reconstruction

We are often interested in filter banks with perfect reconstruction. That is to say, we wish to impose the condition on equation 1.51 that $\hat{x}(z) = x(z)$, which leads us to the following theorem (Vetterli). For more detailed discussion of perfect reconstruction conditions, see [56].

Theorem 1.3 (perfect reconstruction) *A filter bank mapping any input signal x to $\tilde{x}$ performs a perfect reconstruction ($x = \tilde{x}$) iff the following conditions on the filters hold:*

$$\tilde{h}(z)h(-z) + \tilde{g}(z)g(-z) = 0 \qquad \textit{alias cancellation} \tag{1.49}$$

$$\tilde{h}(z)h(z) + \tilde{g}(z)g(z) = 2 \qquad \textit{reconstruction} \tag{1.50}$$

□

PROOF Begin by using equation 1.46 to write the reconstruction equations (1.47) in z-notation:

$$a(z^2) = \frac{1}{2}\left[h(z)x(z) + h(-z)x(-z)\right]$$

$$d(z^2) = \frac{1}{2}\left[g(z)x(z) + g(-z)x(-z)\right]$$

and similarly by use of equation 1.45, equation 1.48 becomes:

$$\tilde{x}(z) = \tilde{h}(z)a(z^2) + \tilde{g}(z)d(z^2)$$

Hence by substitution and grouping like terms of x,

$$\hat{x}(z) = \frac{1}{2}\left[\tilde{h}(z)h(z) + \tilde{g}(z)g(z)\right]x(z) + \frac{1}{2}\left[\tilde{h}(z)h(-z) + \tilde{g}(z)g(-z)\right]x(-z) \tag{1.51}$$

In order to have perfect reconstruction, the first term on the right hand side must be $x(z)$, and the second must be 0, implying that 1.49 and 1.50 must hold. ■

One implication of Theorem 1.3 is that the reconstruction filters $\tilde{h}$, $\tilde{g}$ are specified entirely by the decomposition filters[6] h,g. A scheme for alias cancellation (i.e., satisfying Equation 1.49) that has been presented in several places is as follows [56, 41].

Definition 1.6 (alias cancellation scheme) *Pick (h, g), $(\tilde{h}, \tilde{g})$ filters such that the following hold:*

$$\begin{aligned} g(z) &= z^{-1}h(-z^{-1}), \\ \tilde{h}(z) &= h(z^{-1}), \\ \tilde{g}(z) &= g(z^{-1}) = zh(-z). \end{aligned} \tag{1.52}$$

□

[6] Or vice-versa, as they fill symmetric roles.

Proposition 1.2 *Under the above constraints, alias cancellation (Equation 1.49) is satisfied.* □

PROOF

$$\begin{aligned}\tilde{h}(z)h(-z)+\tilde{g}(z)g(-z) &= h(z^{-1})h(-z)+zh(-z)(-z)^{-1}h(-(-z)^{-1})\\ &= h(z^{-1})h(-z)+\frac{-z}{z}h(-z)h(z^{-1})\\ &= 0\end{aligned}$$

■

Design of Filter Banks

Note that under this scheme, the filter bank is completely determined by the filter h. With the constraints of Equations 1.52, the reconstruction condition (Equation 1.50) becomes

$$\begin{aligned}2 &= \tilde{h}(z)h(z)+\tilde{g}(z)g(z)\\ &= h(z^{-1})h(z)+zh(-z)z^{-1}h(-z^{-1})\\ &= h(z^{-1})h(z)+h(-z)h(-z^{-1}).\end{aligned}$$

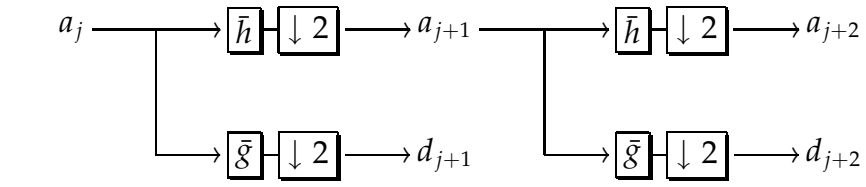
Thus design of perfect reconstruction filter banks of this type (often called *conjugate mirror filters*) is equivalent to finding h to satisfy the above relationship[7].

1.3.5 Subband Coding and the Mallat Algorithm

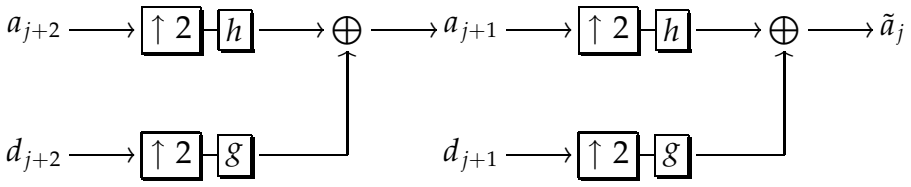
Comparison of equations 1.39, 1.43 with 1.47 ,1.40 shows that the subband filter bank may be used to implement the Mallat algorithm. There are a few minor details, as $h(z)$ in this section must be associated with $h(-z)$ of Equation 1.39. Notwithstanding these minor details, the wavelet filters of an orthogonal wavelet basis are conjugate mirror filters. Note that the converse is not true — there are CMFs that are not wavelet filters.

In order to illustrate this process, an extension of Figure 1.3 is helpful. Figure 1.4 shows the full procedure. At each level of decomposition in the Mallat algorithm, input the previous level's "average" signal. Now a_0 is the original signal (replacing x in Figure 1.3) and recursively define each level in terms of the previous one. Similarly, the *synthesis* signal $\tilde{a}_j$ is formed from the average and detail signals from the previous level, a_{j+1}, d_{j+1}. In this way, the process of Figure 1.2 may be implemented. The relationship between the wavelet and CMF can be made explicit by the following theorem [36].

[7] In practice the equivalent equation in Fourier space is often used (as in Theorem 1.4) [37].



(a) analysis



(b) synthesis

Figure 1.4: Two channel filter bank implementing the Mallat algorithm

Theorem 1.4 *Let $\phi \in \mathcal{L}^2(\mathbb{R})$ be a scaling function and h be the corresponding conjugate mirror filter. Let ϕ be the function with Fourier transform*

$$\hat{\psi}(\omega) = 2^{-1/2}\hat{g}\left(\frac{\omega}{2}\right)\hat{\phi}\left(\frac{\omega}{2}\right) \tag{1.53}$$

where

$$\hat{g}(\omega) = e^{-i\omega}\hat{\bar{h}}(\omega + \pi). \tag{1.54}$$

Then the family of dilations and translations defined by

$$\psi_{j,k}(x) = 2^{-j/2}\psi(2^{-j}x - k) \tag{1.55}$$

is such that $\{\psi_{j,k}\}_{j,k\in\mathbb{Z}}$ forms an orthonormal basis of $\mathcal{L}^2(\mathbb{R})$. At any particular resolution 2^{-j}, $\{\psi_{j,k}\}_{k\in\mathbb{Z}}$ forms a basis for W_j. □

PROOF See [37]. ■

There are orthogonal wavelet bases that are not associated with an MRA and corresponding CMF, however all practical bases seem to be of this form [18]. The connection between the multiresolutions and filter banks forms the core for implementation of orthogonal wavelet transforms, but there remain several important practical issues which must be addressed before this theory can be applied to image compression. The following section will briefly outline several important areas.

1.4 Extensions of the basic theory

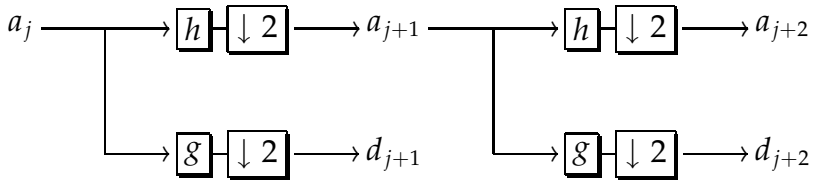
1.4.1 Biorthogonal Wavelet Bases

In §1.3.4 a filter bank was first designed with arbitrary analysis filters h, g and synthesis filters $\tilde{h}, \tilde{g}$ (see Figure 1.5). However, when using the filter bank to implement the Mallat algorithm of §1.3.3, it turned out that the analysis and synthesis steps were performed with the same filters. It is natural to ask what relation the more general setting has to a wavelet MRA. In the other direction, it is also natural to ask if the conditions on a MRA may be relaxed.

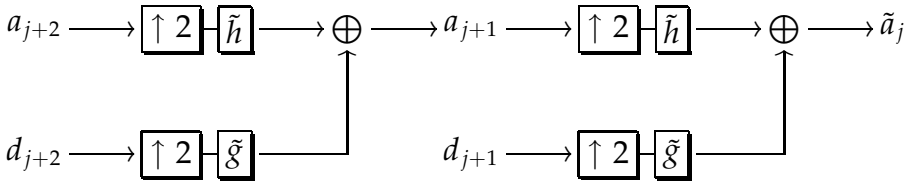
These questions lead to the same generalization: biorthogonal wavelet bases. Biorthogonal bases are a pair of bases (of the same space) $\{e_n\}, \{\tilde{e}_n\}$ that are not orthogonal, but have a biorthogonal (dual) relationship. That is to say, in general

$$<e_n, e_m> \neq \delta_n^m \ , \ <\tilde{e}_n, \tilde{e}_m> \neq \delta_n^m \ , \ <e_n, \tilde{e}_m> = \delta_n^m \ .$$

A very brief outline of the central results is presented here; for details see [18, 37].



(a) analysis



(b) synthesis

Figure 1.5: Two channel filter bank for biorthogonal wavelet bases

Perfect Reconstruction in the more general case

Theorem 1.5 *Symmetric perfect reconstruction FIR filter banks h, $\tilde{h}$, g, $\tilde{g}$ satisfy the relationship*

$$h(z)\tilde{h}(-z) + h(-z)\tilde{h}(-z) = 2 \tag{1.56}$$

Furthermore, the filters are related by

$$\tilde{g}(z) = zh(-z), \qquad g(z) = z^{-1}\tilde{h}(-z). \tag{1.57}$$

□

PROOF Recall the general form of the alias cancellation and perfect reconstruction conditions (Equations 1.49, 1.50)

$$\tilde{h}(z)h(-z) + \tilde{g}(z)g(-z) = 0 \tag{1.58}$$

$$\tilde{h}(z)h(z) + \tilde{g}(z)g(z) = 2. \tag{1.59}$$

Assume that the filters are all FIR, thus the z-transform sums are finite. Equation 1.59 implies that h, g, have no common zeros and thus by Equation 1.58 there exists a $p(z)$ such that

$$\tilde{h}(z) = -g(-z)p(z), \qquad \tilde{g}(z) = h(-z)p(z). \tag{1.60}$$

Substituting this result back into Equation 1.59 yields

$$-g(-z)p(z)h(-z) + h(-z)p(z)g(-z) = 2.$$

$p(z)$ divides the constant 2, thus it must be of the form $p(z) = \alpha z^{\beta}$, $\alpha, \beta \in \mathbb{C}$ so

$$[h(-z)g(-z) - g(-z)h(-z)]\alpha z^{\beta} = 2.$$

Hence from Equation 1.60,

$$\tilde{g}(z) = \alpha z^{\beta} h(-z) \qquad \text{and} \qquad g(-z) = -\alpha^{-1} z^{\beta} \tilde{h}(z).$$

We are free to choose α, β. In order to make g, $\tilde{g}$ symmetric choose $\alpha = \beta = 1$, then

$$\tilde{g}(z) = zh(-z), \qquad g(z) = -\alpha^{-1}(-1)^{-\beta} z^{-\beta} \tilde{h}(z) = z^{-1}\tilde{h}(-z). \tag{1.61}$$

Substitution into Equation 1.59 will yield Equation 1.56. ■

With this result providing appropriate constraints on the filters, the filter banks can be related to bases of $\mathcal{L}^2(\mathbb{R})$. Suppose that h, $\tilde{h}$ are FIR filters as above. It can be shown [18] that

$$\hat{\phi}(\omega) = \prod_{j=1}^{\infty} \frac{\hat{h}(2^{-j}\omega)}{\sqrt{2}} \quad \text{and} \quad \hat{\tilde{\phi}}(\omega) = \prod_{j=1}^{\infty} \frac{\hat{\tilde{h}}(2^{-j}\omega)}{\sqrt{2}} \tag{1.62}$$

are Fourier transforms for distributions of compact support. Further constraints are necessary to ensure finite energy.

Similar to the orthogonal MRA system (§1.3), define scaling relations from these filters,

$$\phi(x) = \sqrt{2}\sum_n h[n]\phi(2x-n), \qquad \tilde{\phi}(x) = \sqrt{2}\sum_n \tilde{h}[n]\tilde{\phi}(2x-n) \tag{1.63}$$

$$\psi(x) = \sqrt{2}\sum_n g[n]\psi(2x-n), \qquad \tilde{\psi}(x) = \sqrt{2}\sum_n \tilde{g}[n]\tilde{\psi}(2x-n) \tag{1.64}$$

The following theorem was introduced by Cohen, Daubechies and Feauveau in [17]. These conditions are sufficient to synthesize perfect reconstruction filters for biorthogonal Riesz bases of $\mathcal{L}^2(\mathbb{R})$.

Definition 1.7 (Riesz Basis) *Let $\{e_n\}_{n\in\mathbb{N}}$ be a linearly independent family of vectors in a Hilbert space* **H**. *$\{e_n\}$ is called a Riesz basis of* **H** *if there exists $\alpha > 0$, $\beta < \infty$ such that for any $f \in$ **H***

$$\alpha\|f\|^2 \leq \sum_n |<f, e_n>|^2 \leq \beta\|f\|^2 . \tag{1.65}$$

□

Theorem 1.6 *Suppose that there exist strictly positive trigonometric polynomials $P(e^{i\omega})$ and $\tilde{P}(e^{i\omega})$ that are unique up to normalization and satisfy the following*

$$|\hat{h}\left(\frac{\omega}{2}\right)|^2 P(e^{i\omega}) + |\hat{h}\left(\frac{\omega}{2}+\pi\right)|^2 P(e^{i(\omega/2+\pi)}) = 2P(e^{i\omega}) , \tag{1.66}$$

$$|\hat{h}\left(\frac{\omega}{2}\right)|^2 \tilde{P}(e^{i\omega}) + |\hat{h}\left(\frac{\omega}{2}+\pi\right)|^2 \tilde{P}(e^{i(\omega/2+\pi)}) = 2\tilde{P}(e^{i\omega}) , \tag{1.67}$$

Additionally, suppose that

$$\inf_{\omega\in[-\pi/2,\pi/2]} |\hat{h}(\omega)| > 0, \qquad \inf_{\omega\in[-\pi/2,\pi/2]} |\hat{\tilde{h}}(\omega)| > 0. \tag{1.68}$$

Then the functions defined by Equation 1.62 are in $\mathcal{L}^2(\mathbb{R})$ and satisfy the biorthogonal relationship

$$<\phi(x), \tilde{\phi}(x-n)>= \delta_n . \tag{1.69}$$

Furthermore, the wavelet families $\{\psi_{j,k}\}$,$\{\tilde{\psi}_{j,k}\}$ form biorthogonal Riesz bases of $\mathcal{L}^2(\mathbb{R})$.

□

PROOF See [18]. ■

These biorthogonal bases are related to a pair of MRAs, rather than a single MRA, as in the orthogonal case. The requirement (Equation 1.17) of an orthonormal basis in the MRA definition may be relaxed [18] to requiring that $\{\phi(x-n) : n \in \mathbb{Z}\}$ forms a Riesz basis for the space V_0 (rather than an orthonormal basis). With this

modification, the situation is completely analogous to that of orthogonal wavelets, except there are two families [37]. The functions of Equation 1.62 may be used to construct two families $\{\phi_{j,n}\}$, $\{\tilde{\phi}_{j,n}\}$, (where $\phi_{j,n}(x) = 2^{-j}\,\phi(2^{-j}\,x - n)$, as before) which are Riesz bases of the spaces $\{V_j\}$, $\{\tilde{V}_j\}$ respectively. Similarly the $\{\phi_{j,n}\}$, $\{\tilde{\phi}_{j,n}\}$ form Riesz bases for the detail spaces W_j, $\tilde{W}_j$. Furthermore, these spaces are related by

$$V_j = V_{j+1} \oplus W_{j+1}, \qquad \tilde{V}_j = \tilde{V}_{j+1} \oplus \tilde{W}_{j+1}\,.$$

Due to the biorthogonality relationship, V_j and $\tilde{W}_j$ are orthogonal, as are $\tilde{V}_j$ and W_j whilst V_j and W_j are not in general orthogonal. The subband filter bank (as shown in Figure 1.5) implements a discrete wavelet transform for the biorthogonal wavelet bases.

Advantages of Biorthogonal Bases

There are three primary advantages in using biorthogonal bases. First, from subband filtering theory, it is known that if the same FIR filters are used for analysis and synthesis (e.g. CMFs and orthogonal wavelet bases), then the filters may not be symmetric [7]. In the biorthogonal case, as there are different filters in the analysis and synthesis banks, we are free to construct symmetric filters.

Secondly, the properties of regularity and vanishing moments of the basis functions may be used to our advantage. It is possible to construct bases such that $\tilde{\phi}$ is much more regular than ϕ, corresponding[18] to more vanishing moments for ψ than for $\tilde{\psi}$, with the result that

$$f = \sum_{j,k} <f, \psi_{j,k}> \tilde{\psi}_{j,k}$$

will have small coefficients where f is regular. This is helpful in compression of images, because much of the important information in images is carried by the edges (non-regular areas) [39]. Furthermore, any error introduced by truncating or quantizing the coefficients will be smoothed by the reconstruction process [3].

Finally, biorthogonal bases are easier to construct than orthogonal basis (see [18, 37]).

1.4.2 Wavelet Design

There are several factors important in the design of wavelet bases for particular applications. The applications discussed in this thesis use a single (biorthogonal) wavelet basis, thus wavelet construction is not influential to the results. For completeness, a brief outline is given.

Compact Support

In applications, it is inconvenient to have basis functions with infinite support. The following theorem gives conditions for orthonormal wavelet bases with compact support

Proposition 1.3 (Compact Support) *The scaling function ϕ of an MRA has compact support iff the filter h has compact support. In this case, the supports of ϕ and h are equal. Furthermore, if ϕ and h have support of $[a, b]$, then the support of the associated wavelet ψ will be $[\frac{a-b+1}{2}, \frac{b-a+1}{2}]$.* □

PROOF See [37]. ■

Vanishing Moments

It turns out that the number of vanishing moments a wavelet has may be related to the size of the support.

Definition 1.8 (Vanishing Moments) *A function f has p vanishing moments if*

$$\int_{-\infty}^{+\infty} x^n f(x)\, dx = 0 \qquad \text{for} \quad 0 \leq n < p. \tag{1.70}$$

□

In the following proposition, Daubechies[18] relates this value to a minimum size of support.

Proposition 1.4 *If an orthonormal wavelet basis of $\mathcal{L}^2(\mathbb{R})$ is generated by ψ , and ψ has p vanishing moments, then the support size of ψ is greater than or equal to $2p-1$. The "Daubechies" family of wavelets have minimum size support $[-p+1, p]$, and support for scaling factor ϕ of $[0, 2p-1]$.* □

PROOF See [18]. ■

This theorem[18] refers to the "Daubechies" family of wavelets, which are designed to have minimum support size for a particular number of vanishing moments.

Regularity

As previously mentioned, the regularity of ψ has a notable cosmetic effect on the error introduced by approximating wavelet coefficients. Discussion of the regularity of compactly supported wavelet bases may be found in [18, ch. 7]

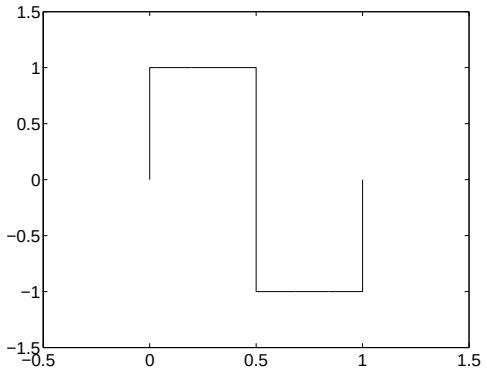
Design

Wavelet bases may be designed with a particular application in mind. Wavelets with a high number of vanishing moments will result in small coefficients in areas where the input function f is regular [18]. On the other hand, a high number of vanishing moments implies a large support (see above). If f has a singularity, all wavelet basis functions with support coincident with this singularity will have large coefficients. The trade-offs are non-obvious, and will depend on the application and expected inputs.

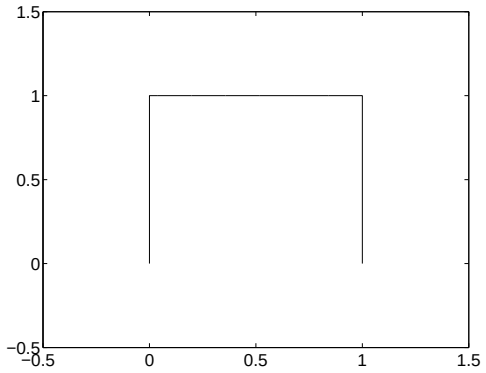
Examples

Some examples of Daubechies orthogonal wavelets ψ of different supports (the Haar basis can be considered to be the first Daubechies wavelet) and associated scaling functions ϕ are shown in Figure 1.6.

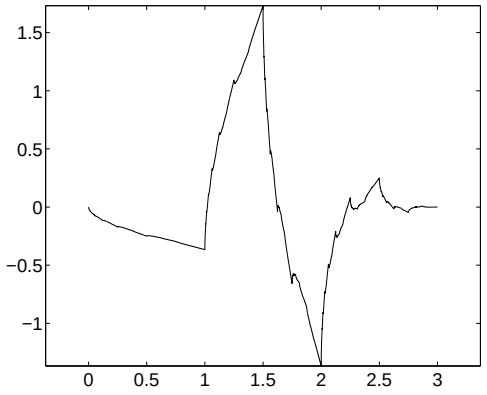
Analogous relationships between support, vanishing moments, and regularity affect the design of biorthogonal wavelets [18, 37]. An example is given in Figure 1.7. This figure shows the two (biorthogonal) wavelets ψ, $\tilde{\psi}$ and associated scaling functions ϕ, $\tilde{\phi}$. This example is taken from the Spline family of biorthogonal wavelets, and demonstrates the symmetry (compare to Figure 1.6) that is not possible for orthogonal bases.



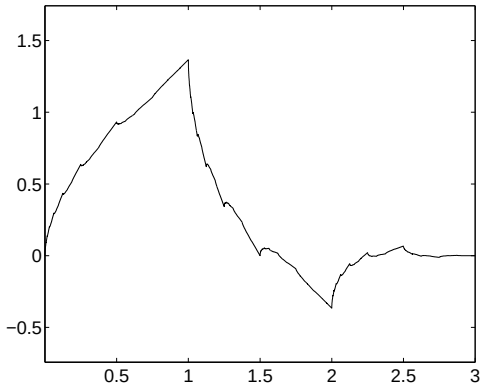
(a) Haar wavelet ψ



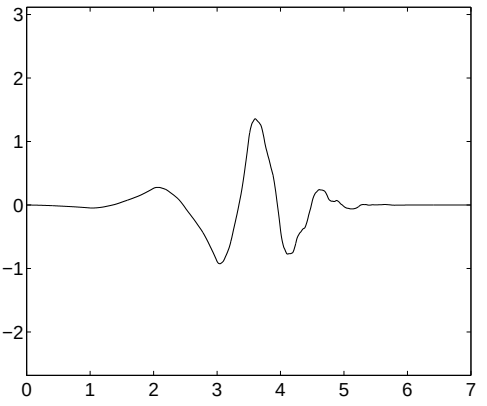
(b) Haar scaling function ϕ



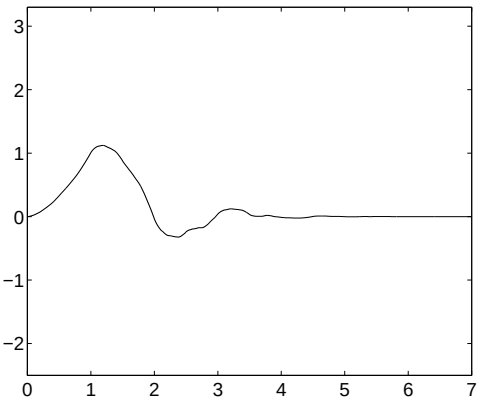
(c) Daubechies 2 wavelet ψ



(d) Daubechies 2 scaling function ϕ

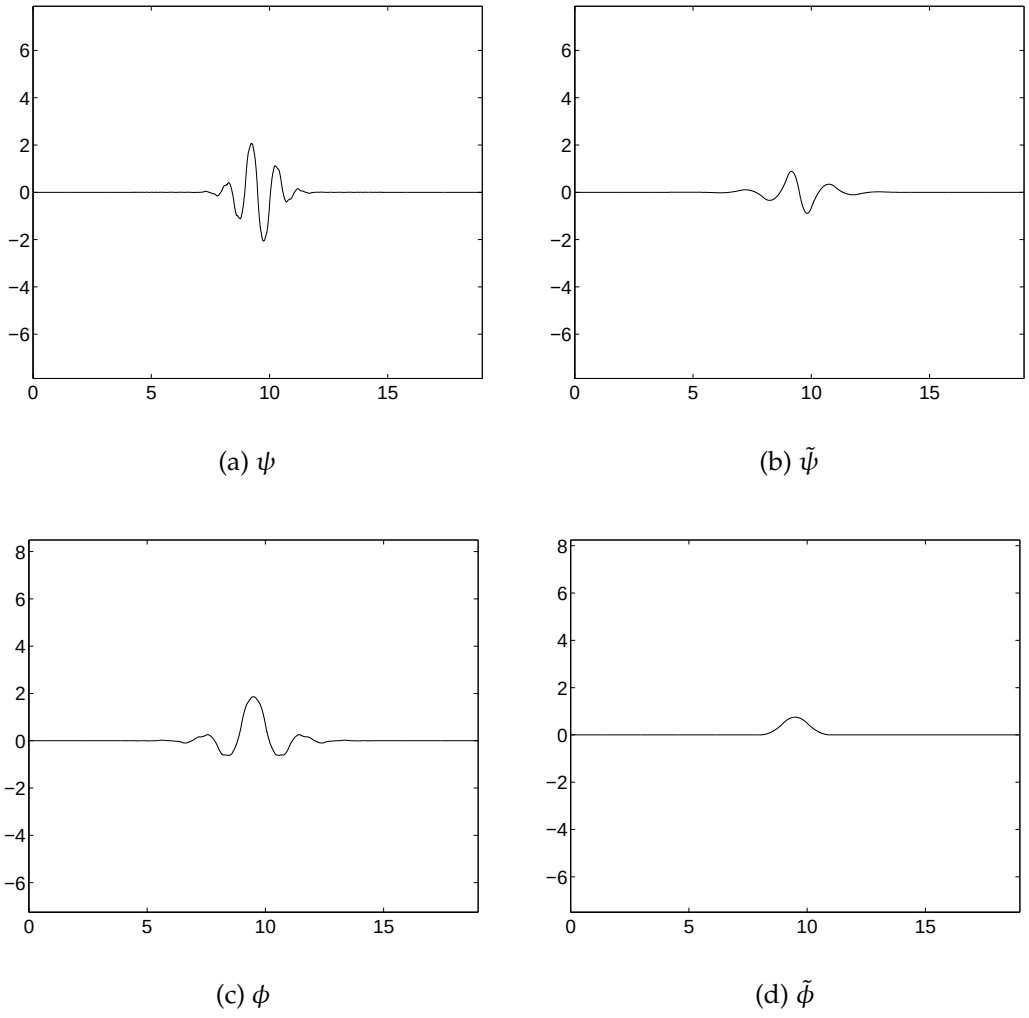


(e) Daubechies 4 wavelet ψ



(f) Daubechies 4 scaling function ϕ

Figure 1.6: Wavelet ψ and scaling function ϕ for Haar, Daubechies 2, and Daubechies 4 orthogonal wavelets.



(a) ψ (b) $\tilde{\psi}$

(c) ϕ (d) $\tilde{\phi}$

Figure 1.7: Wavelet ψ and scaling function ϕ for Spline 3/9 biorthogonal wavelet.

1.4.3 Bases on an Interval

Until now, we have discussed bases of $\mathcal{L}^2(\mathbb{R})$. In applications, we need to consider the effect of finite length signals, specifically the issue of what happens at the edges. Without loss of generality, consider signals defined on the interval $[0, 1]$. The key question is "what will happen when the support of the wavelet or scaling functions overlaps the interval?". It turns out that a wavelet basis of $\mathcal{L}^2(\mathbb{R})[0, 1]$ can be constructed from any wavelet basis of $\mathcal{L}^2(\mathbb{R})$ and the associated scaling functions [37]. These interval bases consist of the set of wavelet and scaling functions

$$\left\{ \{\phi^{\text{int}}_{J,n}\}_{0 \le n < 2^{-J}},\ \{\psi^{\text{int}}_{j,n}\}_{-\infty < j \le J,\ 0 \le n < 2^{-j}} \right\}. \tag{1.71}$$

Where the superscript "int" denotes wavelet and scaling functions modified for the interval. There are three primary methods of constructing such a basis.

Periodic Extension

This method is the simplest to implement; simply periodize the signal f over $\mathcal{L}^2(\mathbb{R})$, resulting in periodic wavelets. The disadvantage of this method is that the boundary wavelets (those whose support overlapped the interval before periodizing) have no vanishing moments [37]. This leads to large wavelet coefficients at the edges [37].

Symmetric Extension, or folded wavelets

This method is a bit more complicated to implement, but has the advantage that boundary wavelets retain a vanishing moment. For many wavelet bases, a single vanishing moment is significantly less than the internal wavelets, so coefficients at the boundary will still be unnecessarily large.

Symmetric extensions can be approached purely from a subband coding point of view, for a larger class of two-channel filter banks than have been considered. Briefly, the input signal is extended by *folding* at the boundaries in a symmetric or anti-symmetric manner. The symmetry of the extension is determined by the length and symmetry characteristics of the filters and input signal, and furthermore by the constraint of perfect reconstruction [52, 10, 11].

The method of Symmetric Extension is used in the applications presented later. Figure 1.8 demonstrates symmetric and anti-symmetric extensions of a signal.

Boundary Wavelets

It is possible [15] to explicitly construct boundary wavelets with as many vanishing moments as the original wavelet ϕ, thereby avoiding large coefficients at the boundaries. Implementation is more difficult than the previous two methods, and this approach is not used in the applications presented later.

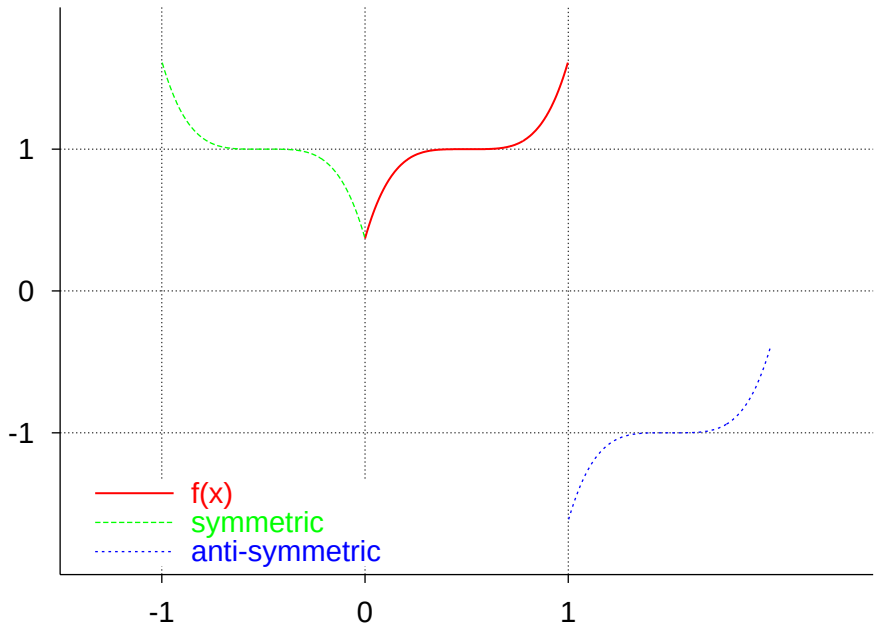


Figure 1.8: symmetric (left) and anti-symmetric (right) extensions of a signal on $[0,1]$

1.5 Separable Wavelet Bases

In the work to follow, finite, discrete images (signals) in two and three dimensions will be considered. The previous discussion, however, has concentrated on bases for $\mathcal{L}^2(\mathbb{R})$. A method to extend these results to two dimensions is now discussed. In a completely analogous manner, extensions to n dimensions may be made.

Given $\{\psi_{j,k}(t)\}_{(j,k)\in\mathbb{Z}^2}$, an orthonormal basis for $\mathcal{L}^2(\mathbb{R})$, the goal is to create an orthonormal basis for $\mathcal{L}^2(\mathbb{R}^2)$. The extension $\{\psi_{j,k}(x)\psi_{l,m}(y)\}_{(j,k,l,m)\in\mathbb{Z}^4}$, while straightforward, has the inconvenient property of mixing x and y information at different resolutions. In order to avoid this, we construct higher dimension multiresolutions.

1.5.1 Separable Multiresolutions

Suppose MRA is an $\{V_n\}_{n\in\mathbb{Z}}$ of $\mathcal{L}^2(\mathbb{R})$. Consider the tensor product spaces,

$$\mathbf{V}_n^2 = V_n \otimes V_n = \text{span}\,\overline{\{F(x,y) = f(x)g(y) : f,g \in V_n\}}.$$

With the appropriate modification to the MRA (Definition 1.3), the $\{\mathbf{V}_n\}_{n\in\mathbb{Z}}$ form a multiresolution in $\mathcal{L}^2(\mathbb{R}^2)$, [37, 18]. Essentially following the discussion of §1.3.3, de-

note the orthogonal complement of $\mathbf{V}_j$ as $\mathbf{W}_j$. Then

$$\mathbf{V}_j = \mathbf{V}_{j+1} \oplus \mathbf{W}_{j+1} \tag{1.72}$$

where (using W_j to denote the orthogonal complement of V_j, as before)

$$\mathbf{W}_j = (V_j \otimes W_j) \oplus (W_j \otimes V_j) \oplus (W_j \otimes W_j). \tag{1.73}$$

For simplicity, we denote this as

$$\mathbf{W}_j = \mathbf{W}_j^h \oplus \mathbf{W}_j^v \oplus \mathbf{W}_j^d, \tag{1.74}$$

where the superscripts h, v, and d are for horizontal, vertical, and diagonal respectively. The separable multiresolution results in a family of wavelet functions [37]

$$\begin{aligned}
\Psi^h_{n,j,k}(x,y) &= \phi_{n,j}(x)\psi_{n,k}(y) \\
\Psi^v_{n,j,k}(x,y) &= \psi_{n,j}(x)\phi_{n,k}(y) \\
\Psi^d_{n,j,k}(x,y) &= \psi_{n,j}(x)\psi_{n,k}(y)
\end{aligned}$$

and the scaling function,

$$\Phi_{j,k}(x,y) = \phi_{j,k}(x)\phi_{j,k}(y).$$

Analogous to §1.3.3, consider the *complete decomposition*, arrived at by repeated application of 1.72

$$\begin{aligned}
\mathbf{V}_j &= \mathbf{V}_N \oplus \bigoplus_{k=0}^{N-j-1} \mathbf{W}_{N-k}, \text{ for } j < N \\
&= \mathbf{V}_N \oplus \bigoplus_{k=0}^{N-j-1} \left(\mathbf{W}^h_{N-k} \oplus \mathbf{W}^v_{N-k} \oplus \mathbf{W}^d_{N-k} \right), \text{ for } j < N
\end{aligned}$$

where, in terms of the above wavelet functions (overbar denoting closure),

$$\begin{aligned}
\mathbf{V}_n &= \text{span } \overline{\{\Phi_{n,j,k}(x,y) : 0 \le j,k \le 2^n - 1\}} \\
\mathbf{W}^h_n &= \text{span } \overline{\{\Psi^h_{n,j,k}(x,y) : 0 \le j,k \le 2^n - 1\}} \\
\mathbf{W}^v_n &= \text{span } \overline{\{\Psi^v_{n,j,k}(x,y) : 0 \le j,k \le 2^n - 1\}} \\
\mathbf{W}^d_n &= \text{span } \overline{\{\Psi^d_{n,j,k}(x,y) : 0 \le j,k \le 2^n - 1\}}
\end{aligned}$$

If we restrict ourselves to functions $f(x,y)$ that admit the wavelet expansion

$$\begin{aligned}
f(x,y) = b_{0,0,0}\Phi_{0,0,0}(x,y) + \\
\sum_{i=-\infty}^{0} \sum_{j=0}^{2^{-i}-1} \sum_{k=0}^{2^{-i}-1} \left[c^h_{i,j,k}\Psi^h_{i,j,k}(x,y) + c^v_{i,j,k}\Psi^v_{i,j,k}(x,y) + c^d_{i,j,k}\Psi^d_{i,j,k}(x,y) \right],
\end{aligned}$$

then the coefficients may be conveniently arranged into a "pyramid" (see [18, 37]) of blocks at each level of decomposition. There are three *subbands*; horizontal, vertical and diagonal. Denote the high resolution (detail) blocks as D_k^h, D_k^v, D_k^d respectively for each decomposition level $k \geq 0$. At each level, the blocks consist of 2^{2k} coefficients $d_{k,i,j}^h, d_{k,i,j}^v, d_{k,i,j}^d$. If the decomposition stops at some point before reaching level $k = 0$, there will be an *average* image of low resolution coefficients that may be denoted by A_k. Figure 1.9 shows these components arranged into the standard 'pyramid' form. The extension to $\mathcal{L}^2(\mathbb{R}^3)$ etc., is achieved in the analogous way.

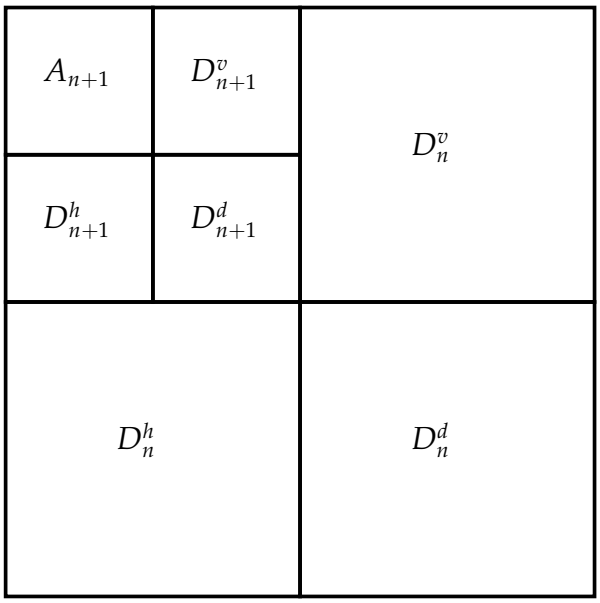


Figure 1.9: Coefficient pyramid showing decomposition at levels n and $n+1$

1.5.2 The Mallat Algorithm on Separable Bases

For a separable basis, the Mallat algorithm is easily applied by means of the filter banks presented in §1.3.4, just as in the one-dimensional case. Because the basis is separable [18], the decomposition is achieved by filtering along one dimension at a time at each level of decomposition. In the case of $\mathbb{R}^2$, first filter row-by-row, and then again column-by-column. Figure 1.10 shows this process.

The "HL" or high-low band of the detail signal is usually referred to as the "vertical" band. Vertical lines show up in this band: it contains high-pass information in the horizontal (row) direction and low-pass information in the vertical (column) direction. Similarly, the "LH" and "HH" bands are called the "horizontal" and "diagonal" bands, respectively. Figure 1.11 gives examples to illustrate the roles of the three bands.

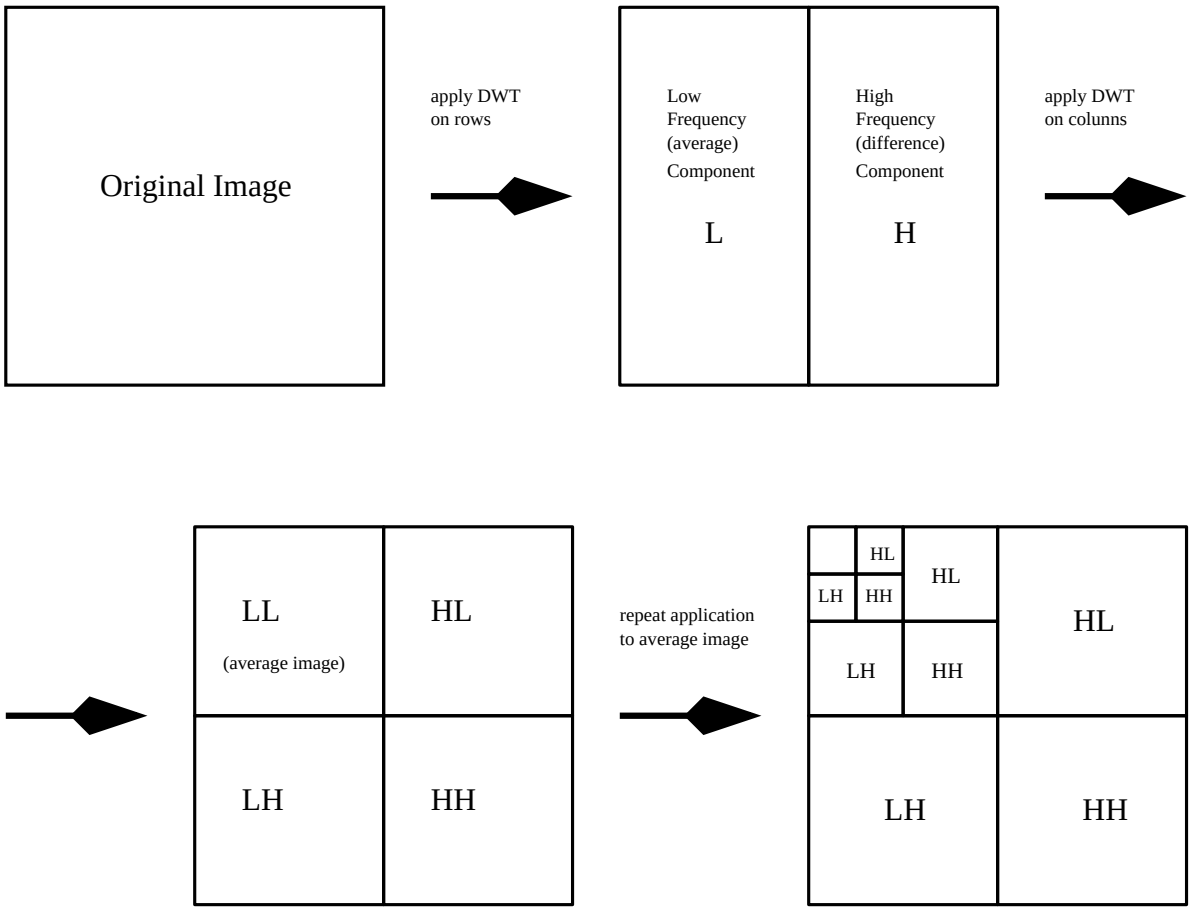
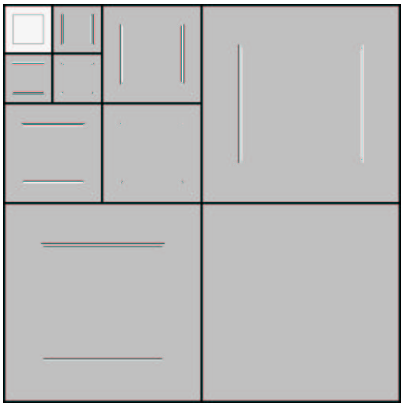
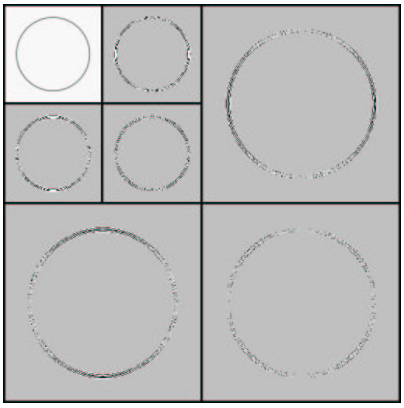


Figure 1.10: Two-dimensional DWT

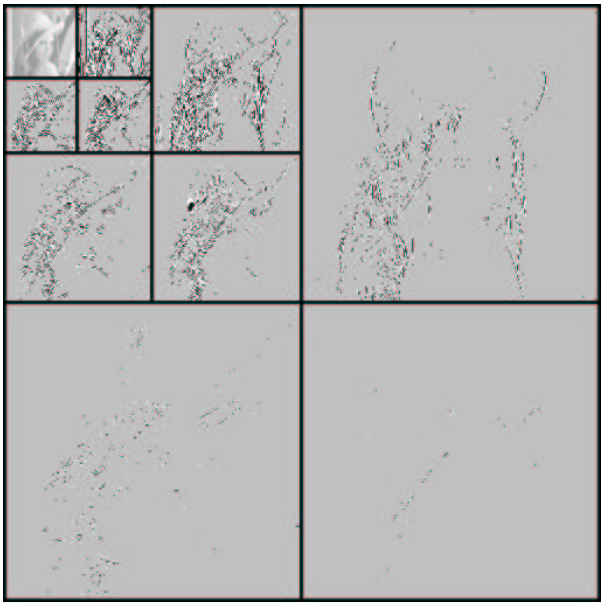
An analogous procedure applies to three-dimensional images. Figure 1.12 depicts (without labels, for simplicity) two levels of decomposition in three dimensions.



(a) dwt of square



(b) dwt of circle



(c) dwt of lena

Figure 1.11: Examples of DWT with two-dimensional separable wavelet bases. Because the size of the coefficients decreases rapidly for higher resolutions, the images have been manipulated. The upper-left corner contains the average image at the final level. All other areas have been mapped so that the pixel is white for coefficients $c > 0.1$, black for $c < -0.1$, and otherwise grey. (a) a square with 2 level DWT (b) a circle with 2 level DWT (c) "lena" image with 3 level DWT

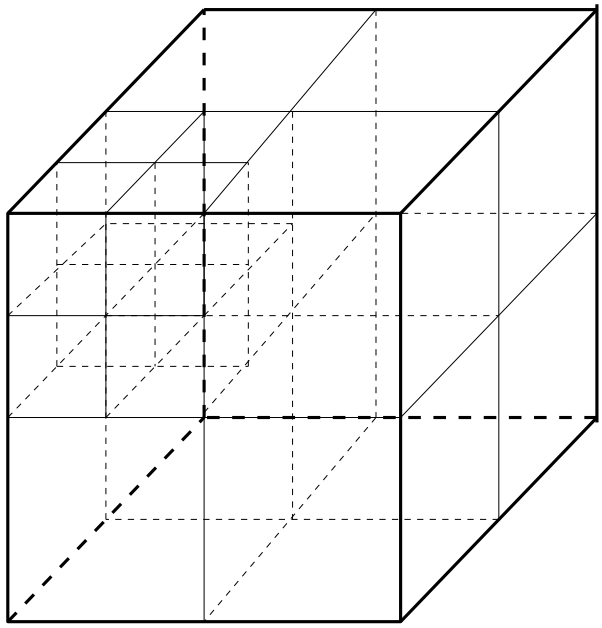


Figure 1.12: Three-dimensional DWT structure

1.6 Wavelet Coefficient Trees

For any function $f \in \mathcal{L}^2(\mathbb{R})$ and an orthonormal basis $\{\psi_{j,k}\}$ of $\mathcal{L}^2(\mathbb{R})$,

$$f = \sum_{j,k} <f, \psi_{j,k}> \psi_{j,k} .$$

If, for reasons that will become apparent in §2.8, we restrict ourselves to wavelet expansions of the following form

$$f(x) = b_{0,0}\phi(x) + \sum_{j=-\infty}^{0} \sum_{k=0}^{2^{-j}-1} c_{j,k}\psi_{j,k}(x) , \tag{1.75}$$

then the coefficients $c_{j,k} = <f, \psi_{j,k}>$ (plus $b_{0,0}$, if non-zero) may be represented as a binary tree:

<table>
<tr><td colspan="8">$b_{0,0}$</td></tr>
<tr><td colspan="8">$c_{0,0}$</td></tr>
<tr><td colspan="4">$c_{-1,0}$</td><td colspan="4">$c_{-1,1}$</td></tr>
<tr><td colspan="2">$c_{-2,0}$</td><td colspan="2">$c_{-2,1}$</td><td colspan="2">$c_{-1,2}$</td><td colspan="2">$c_{-1,3}$</td></tr>
<tr><td>$c_{-3,0}$</td><td>$c_{-3,1}$</td><td>$c_{-3,2}$</td><td>$c_{-3,3}$</td><td>$c_{-3,4}$</td><td>$c_{-3,5}$</td><td>$c_{-3,6}$</td><td>$c_{-3,7}$</td></tr>
<tr><td>⋮</td><td>⋮</td><td>⋮</td><td>⋮</td><td>⋮</td><td>⋮</td><td>⋮</td><td>⋮</td></tr>
</table>

A more compact representation is

<table>
<tr><td colspan="4">$b_{0,0}$</td></tr>
<tr><td colspan="4">$c_{0,0}$</td></tr>
<tr><td colspan="2">$c_{-1,0}$</td><td colspan="2">$c_{-1,1}$</td></tr>
<tr><td>$C_{-2,0}$</td><td>$C_{-2,1}$</td><td>$C_{-1,2}$</td><td>$C_{-1,3}$</td></tr>
</table>

where each $C_{j,k}$ represents a binary subtree of infinite depth.

For higher dimensional signals, similar trees exist. With a two-dimensional image, the coefficient tree is a *quadtree*; in three dimensions, the trees are *octrees*, and so on. The wavelet expansion in $\mathcal{L}^2(\mathbb{R}^2)$ analogous to Equation 1.75 will be (using the notation introduced in §1.5.1)

$$f(x,y) = b_{0,0,0}\Phi_{0,0,0}(x,y) + \sum_{i=-\infty}^{0} \sum_{j=0}^{2^{-i}-1} \sum_{k=0}^{2^{-i}-1} \left[c^h_{i,j,k}\Psi^h_{i,j,k}(x,y) + c^v_{i,j,k}\Psi^v_{i,j,k}(x,y) + c^d_{i,j,k}\Psi^d_{i,j,k}(x,y) \right].$$

The resulting coefficients $c^b_{i,j,k}$ for $b \in \{h, v, d\}$ can be organized into a quadtree as shown in Figure 1.13. These structures will be useful in discussing the manipulation of wavelet coefficients.

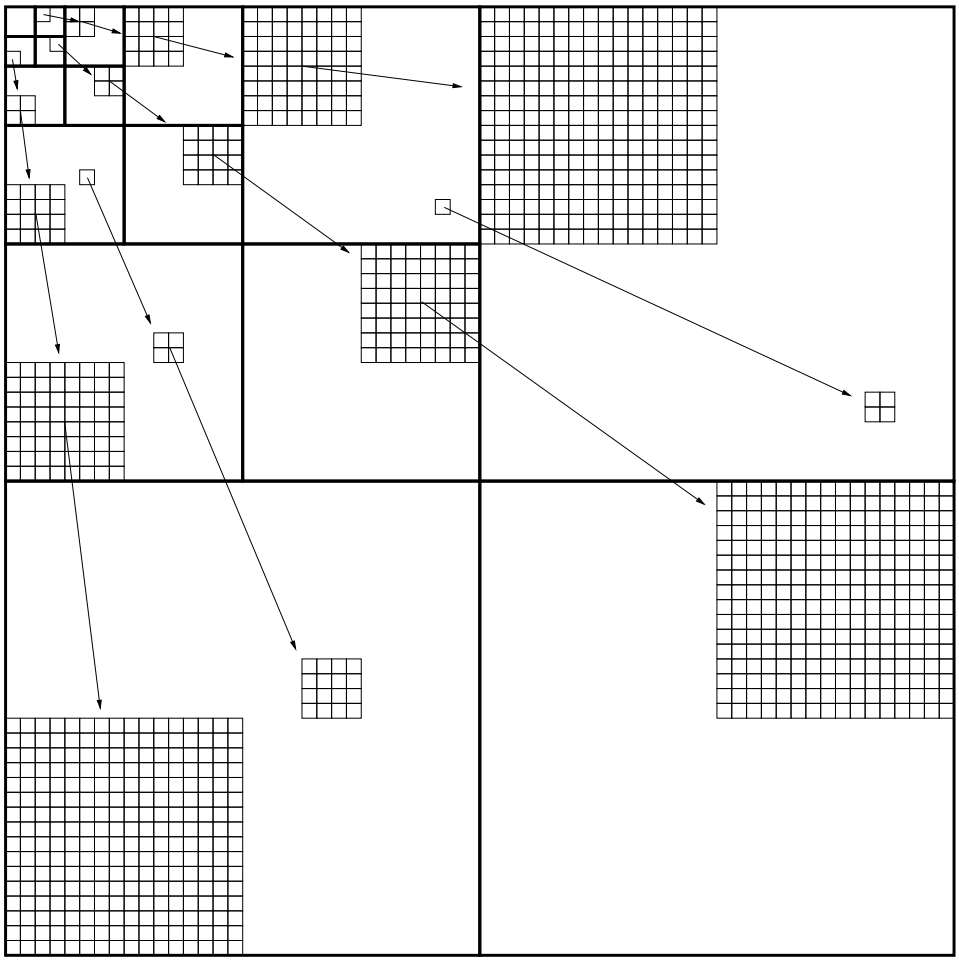


Figure 1.13: Discrete two-dimensional MRA tree structure showing quadtrees rooted at various levels

Remarks

The chapter began by introducing the basic ideas of a wavelet basis. After developing the basic theory, some further developments were presented. Separable multiresolutions allow moving from one-dimensional signals to two- and three-dimensional images. Extending the results on $\mathcal{L}^2(\mathbb{R})$ to work with bases of $\mathcal{L}^2([0,1])$ addresses the need for finite images. Wavelet design, and biorthogonal wavelets will play a part in the wavelet based compression algorithms discussed in §4.3, and wavelet coefficient trees will be central to the methods of manipulating wavelet coefficients.

Chapter 2

Fractal Transforms

The Iterated Function Systems (IFS) theory is quite simple, and as many of the required proofs are straightforward and short, it is possible to present here a fairly complete development of the basic theory. Some proofs are omitted, with citations, to simplify the discussion. A good introduction to IFS can be found in [5], while a more complete discussion of the theory underlying IFS can be found in [22, ch.1,2] and [21, ch.1,2,9].

After developing the background the inverse problem for IFS is discussed; it may be stated as "given a particular set, is is possible to generate an IFS whose invariant set will approximate it?". The final goal is to apply these ideas in the wavelet domain, and a brief introduction to Iterated Function Systems on Wavelet trees (IFSW) by way of Iterated Function Systems on Grey Level Maps (IFSM) will be presented.

2.1 Overview

The Idea of IFS

Example 2.1 Take, as an initial example, the middle third Cantor set, call it C. Now C is a union of two (left and right) copies of itself. Consider the two mappings $S_1(x) = \frac{1}{3}x$ and $S_2(x) = \frac{1}{3}x + \frac{2}{3}$ defined on $(\mathbb{R}, d)$ where $d(x, y) = \mid x - y \mid$. Then S_1 and S_2 are contraction mappings and

$$C = S_1(C) \cup S_2(C).$$

Thus C is invariant under the map $S(X) = S_1(X) \cup S_2(X)$. Invariance under this map completely specifies the set C. □

Roughly speaking, families of contraction maps like S are called IFS (an exact definition will be presented in the following sections). Many such invariant sets are fractals[1]. The terminology of fractals is quite context dependent, but seems to universally include self-similarity of the set (invariance of scale) in some way.

[1] A rather difficult term to define.

2.2 Background

Notation 2.1 *Denote a metric space of the metric d on the set $\mathbf{X}$ by $(\mathbf{X}, d)$.* □

Notation 2.2 *Consider a function $f : \mathbf{X} \to \mathbf{X}$. Denote the n^{th} **iteration** or n-fold composition, of f to be $f^{\circ(n)}$, which is defined as*

$$\begin{aligned} f^{\circ(1)}(x) &:= f(x) \\ f^{\circ(n+1)}(x) &:= f(f^{\circ(n)}) \qquad \text{for } n \text{ in } \mathbb{N}^+, n > 1\ . \end{aligned}$$ □

Definition 2.1 (attractor) *Consider a function $f : \mathbf{X} \to \mathbf{X}$. If there exists a $y \in X$ such that $lim_{n\to\infty} f^{\circ(n)}(x) = y$, $\forall x \in X$ then y is called the **attractor** of f.* □

Definition 2.2 (fixed point) *If a function f has an argument with the property that $f(x) = x$, call x a **fixed point** of f, and denote it $\bar{x}_f$ (or $\bar{x}$ where there is no possibility of ambiguity).* □

In the language of fractals, the term *attractor* is often used interchangeably with that of a *fixed point* due to the following property:

Proposition 2.1 *Let $f : \mathbf{X} \to \mathbf{X}$ be a continuous function. If f has an attractor $x \in X$, then x is a fixed point.* □

PROOF By continuity, $f(x) = f(lim_{n\to\infty} f^{\circ(n)}(x)) = lim_{n\to\infty} f^{\circ(n+1)}(x) = x$. ■

Clearly it is possible to have fixed points which are not attractors, but these will not be relevant to the development.

Contractive Maps

The following class of functions is central to the idea of IFS.

Definition 2.3 (Lipschitz) *Consider a function on a metric space $(\mathbf{X}, d)$, $f : \mathbf{X} \to \mathbf{X}$. If there exists an $s \in [0, \infty)$ such that $\forall x, y \in \mathbf{X}$,*

$$d(f(x), f(y)) \leq sd(x, y). \tag{2.1}$$

*Then f is called **Lipschitz** on $\mathbf{X}$, and s is called a **Lipschitz constant** for f.* □

Definition 2.4 (contractive) *If a function f has a Lipschitz constant $c < 1$, f is called **contractive**, or a **contraction**, with contractivity factor of (at most) c.* □

Proposition 2.2 (contractivity factor) *Let f be a contractive function on $\mathbf{X}$. Let C be the set of all contractivity factors of f. Define $c_f = \inf(C)$. Then c_f is a contractivity factor of f, and c_f is called **the** contractivity factor of f.* □

PROOF Let $x, y \in X$, then $\forall c \in C$

$$d(f(x), f(y)) \leq cd(x, y)$$

which implies,

$$\begin{aligned} d(f(x), f(y)) &\leq \inf(C) d(x, y) \\ &= c_f d(x, y). \end{aligned}$$

Thus c_f is a contractivity factor for f, and

$$d(f(x), f(y)) \leq c_f d(x, y). \tag{2.2}$$

■

Definition 2.5 *Let $(\mathbf{X}, d)$ be a complete metric space. Denote the set of all contraction mappings on $\mathbf{X}$ to be $Con(\mathbf{X}, d)$.* □

The following properties of the set $\mathrm{Con}(\mathbf{X}, d)$ are important to the applications that will be presented later.

Lemma 2.1 *Let f be a contractive map on the metric space $(\mathbf{X}, d)$. Then f is uniformly continuous.* □

PROOF Let $\epsilon > 0$. Consider the contractivity factor of f, c. Suppose $c \neq 0$, and let $\delta = \frac{\epsilon}{c}$. Thus $\forall d(x, y) < \delta$

$$d(f(x), f(y)) \leq cd(x, y) < \epsilon.$$

The inequality holds trivially for the case $c = 0$. ■

The continuity of f is important in that members of $\mathrm{Con}(\mathbf{X}, d)$ will map compact sets to compact sets. A further useful property of $\mathrm{Con}(\mathbf{X}, d)$ is that, under an appropriate metric, fixed points vary continuously under contractive maps. In order to show this, a suitable metric is required.

Proposition 2.3 *Let $(\mathbf{X}, d)$ be a compact metric space. Then $d_{Con(\mathbf{X},d)} : Con(\mathbf{X}, d) \to [0, \infty)$ defined as*

$$d_{Con(\mathbf{X},d)}(f, g) = \min\{1, \sup_{x \in \mathbf{X}} d(f(x), g(x))\} \quad \forall\, f, g \in Con(\mathbf{X}, d)$$

is a metric on the space $Con(\mathbf{X}, d)$. The min with 1 is taken to avoid infinite values. □

PROOF Let $f, g, h \in \mathrm{Con}(\mathbf{X}, d)$. Assume $\sup_{x \in \mathbf{X}} d(f(x), g(x)) < 1$.

$$\begin{aligned} d_{\mathrm{Con}(\mathbf{X},d)}(f,g) &\geq 0 && \text{as } d(f(x),g(x)) \geq 0 \ \forall x \in \mathbf{X} \\ d_{\mathrm{Con}(\mathbf{X},d)}(f,g) &= 0 \iff f = g && \text{by definition of sup} \\ d_{\mathrm{Con}(\mathbf{X},d)}(f,g) &= d_{\mathrm{Con}(\mathbf{X},d)}(g,f) && \text{by symmetry of } d. \end{aligned}$$

For the triangle inequality, apply the triangle inequality of d,

$$\begin{aligned} d_{\mathrm{Con}(\mathbf{X},d)}(f,g) &= \sup_{x \in \mathbf{X}} d(f(x), g(x)) \\ &\leq \sup_{x \in \mathbf{X}} \{d(f(x), h(x)) + d(h(x), g(x))\} \\ &\leq \sup_{x \in \mathbf{X}} d(f(x), h(x)) + \sup_{x \in \mathbf{X}} d(h(x), g(x)) \\ &= d_{\mathrm{Con}(\mathbf{X},d)}(f,h) + d_{\mathrm{Con}(\mathbf{X},d)}(h,g). \end{aligned}$$

The case where $\sup_{x \in \mathbf{X}} d(f(x), g(x)) \geq 1$ is straightforward. ■

Now, applying this metric, the continuity of fixed points can be shown. This result (introduced in [14]) is important in applications of the Collage Theorem (2.5), which will be described later.

Theorem 2.1 (Continuity of Fixed Points) *Let $(\mathbf{X}, d)$ be a compact metric space and $f, g \in Con(\mathbf{X}, d)$ with fixed points $\bar{x}_f$, $\bar{x}_g$ and contraction factors c_f, c_g, respectively. Let $c = min\{c_f, c_g\}$. Define the function $F(f) : Con(\mathbf{X}, d) \to \mathbf{X}$ by*

$$F(f) = \bar{x}_f \quad \forall f \in Con(\mathbf{X}, d).$$

Then F is continuous under $d_{Con(\mathbf{X},d)}$. □

PROOF Let $f, g \in \mathrm{Con}(\mathbf{X}, d)$. Let $\epsilon > 0$. Let $\delta = \epsilon(1 - c)$. Then for all δ such that $d_{\mathrm{Con}(\mathbf{X},d)}(f,g) < \delta$,

$$\begin{aligned} d(\bar{x}_f, \bar{x}_g) &= d(f(\bar{x}_f), g(\bar{x}_g)) \\ &\leq d(f(\bar{x}_f), f(\bar{x}_g)) + d(f(\bar{x}_g), g(\bar{x}_g)) \\ &\leq d(f(\bar{x}_f), f(\bar{x}_g)) + d_{\mathrm{Con}(\mathbf{X},d)}(f,g) \\ &\leq c d(\bar{x}_f, \bar{x}_g) + d_{\mathrm{Con}(\mathbf{X},d)}(f,g) \\ &< c d(\bar{x}_f, \bar{x}_g) + \epsilon(1 - c) && \text{by hypothesis.} \end{aligned}$$

Therefore,

$$d(\bar{x}_f, \bar{x}_g) < \epsilon.$$ ■

Corollary 2.1 *Let $(\mathbf{X}, d)$ be a compact metric space. Let $f, g \in Con(\mathbf{X}, d)$, and define $c = min\{c_f, c_g\}$. Then*

$$d(\bar{x}_f, \bar{x}_g) < \frac{1}{1 - c} d_{Con(\mathbf{X},d)}(f,g).$$ □

Now that the necessary background has been filled in, the central theorem on which IFS depends may be presented.

The Contraction Mapping Principle

The theory of IFS is centrally reliant on the Contraction Mapping Principle (CMP) (due to Banach [4]). The CMP identifies unique fixed points of contractive maps; these fixed points are the attractors of any iterative process on the maps.

Theorem 2.2 (Contraction Mapping Principle) *Let $(\mathbf{X}, d)$ be a complete metric space, and $f : \mathbf{X} \to \mathbf{X}$ be a contractive map with contractivity factor c. Then f has a unique fixed point $\bar{x}_f$, and $\bar{x}_f$ is an attractor of f.* □

PROOF Construct a sequence $\{x_n\}_{n=0}^{\infty}$ in the following manner:

$$\begin{aligned} &\text{pick any } x_0 \in \mathbf{X}, \\ &x_{n+1} = f(x_n). \end{aligned}$$

Now for any $n, m \in \mathbb{N}^+$ with $m > n$,

$$\begin{aligned} d(x_n, x_m) &= d(f^{\circ(n)}(x_0), f^{\circ(m)}(x_0)) \\ &\leq cd(f^{\circ(n-1)}(x_0), f^{\circ(m-1)}(x_0)) \end{aligned}$$

since f is contractive. The final line follows from Equation 2.2. If f is iterated (from equation 2.1) n times, we find that

$$d(x_n, x_m) \leq c^n d(x_0, f^{\circ(m-n)}(x_0)). \tag{2.3}$$

Now let $k \in \mathbb{N}^+$ and by repeated application of the triangle inequality:

$$\begin{aligned} d(x_0, f^{\circ(k)}(x_0)) &\leq d(x_0, f(x_0)) + d(f(x_0), f^{\circ(k)}(x_0)) \\ &\leq d(x_0, f(x_0)) + d(f(x_0), f^{\circ(2)}(x_0)) + \ldots + d(f^{\circ(k-1)}(x_0), f^{\circ(k)}(x_0)) \\ &\leq d(x_0, f(x_0)) + cd(x_0, f(x_0)) + c^2 d(x_0, f(x_0)) + \ldots + c^{k-1} d(x_0, f(x_0)) \\ &= \sum_{i=0}^{k-1} c^i d(x_0, f(x_0)). \end{aligned}$$

Hence by comparison to a geometric series in c,

$$d(x_0, f^{\circ(k)}(x_0)) \leq \frac{1}{1-c} d(x_0, f(x_0)).$$

Combining this result and equation 2.3 yields

$$d(x_n, x_m) \leq \frac{c^n}{1-c} d(x_0, f(x_0)). \tag{2.4}$$

Since $c < 1$, $d(x_n, x_m) \to 0$ as $n \to \infty$. Thus by equation 2.4 $\{x_n\}$ is a Cauchy sequence. Therefore $\{x_n\}$ converges in the complete space $(\mathbf{X}, d)$. That is, $\exists\ \bar{x}$ such that $x_n \to \bar{x}$. Now suppose that there is another such fixed point, $y \in X$ such that $x_n \to y$. So

$$\begin{aligned} d(\bar{x}, y) &= d(f(\bar{x}), f(y)) && \text{as } \bar{x}, y \text{ are fixed points} \\ &\leq cd(\bar{x}, y) && f \text{ is contractive.} \end{aligned}$$

However, this inequality is only satisfied by $d(\bar{x}, y) = 0$, since $0 < c < 1$. And since d is a metric, this implies that $\bar{x} = y$, hence $\bar{x}$ is unique. ■

With this result, the background is nearly in place to define an IFS. Before doing so a convenient complete metric space is introduced, in which to work.

2.3 A Complete Metric Space for IFS

This section will briefly outline the development of a complete metric space suitable for IFS. A more complete discussion may be found in [5], which this section loosely follows.

Definition 2.6 (Hausdorff Space) *Let $(\mathbf{X}, d)$ be a complete metric space. $\mathcal{H}(\mathbf{X})$ denotes the set of all non-empty compact subsets of $\mathbf{X}$.* □

In approaching a distance function to use in $\mathcal{H}(\mathbf{X})$, first address the distance between a point and a set.

Notation 2.3 *Let $(\mathbf{X}, d)$ be a complete metric space, $x \in \mathbf{X}$, and $B \in \mathcal{H}(\mathbf{X})$. Define $d(x, B)$, the distance from the point x to the set B as*

$$d(x, B) = \inf_{b \in B} \{d(x, b)\}.$$ □

Now consider the distance between two sets.

Notation 2.4 *Let $(\mathbf{X}, d)$ be a complete metric space. Let $A, B \in \mathcal{H}(\mathbf{X})$. Define $d(A, B)$, the distance from the set A to the set B as*

$$d(A, B) = \sup_{a \in A} \{d(a, B)\}.$$ □

However, this will not quite form a metric (it clearly lacks symmetry) so the following modification is introduced.

Definition 2.7 (Hausdorff Distance) *Let $A, B \in \mathcal{H}(\mathbf{X})$. Define the **Hausdorff distance** between A and B as*

$$h(A, B) = \max \{d(A, B), d(B, A)\}.$$ □

The following lemma succinctly describes the intuitive sense of the Hausdorff metric: Two sets are ϵ-close under h iff they are fully contained in the ϵ extension of each other.

Lemma 2.2 *Let $(\mathbf{X}, \mathbf{d})$ be a metric space, $A, B \in \mathcal{H}(\mathbf{X})$, $\epsilon > 0$. Then*

$$h(A, B) \leq \epsilon \text{ iff } A \subset B + \epsilon \text{ and } B \subset A + \epsilon. \qquad \square$$

PROOF Omitted, see [5]. ■

Theorem 2.3 *Let $(\mathbf{X}, d)$ be a complete metric space. Then $(\mathcal{H}(\mathbf{X}), \mathbf{h})$ is a complete metric space. Furthermore, if $\{A_n\}_{n=1}^{\infty}$ is a Cauchy sequence in $(\mathcal{H}(\mathbf{X}), h)$, then $A = \lim_{n\to\infty} A_n \in \mathcal{H}(\mathbf{X})$ can be characterized as:*

$$A = \{x \in \mathbf{X} : \exists \text{ Cauchy sequence } \{\mathbf{x_n} \in \mathbf{A_n}\} \text{ converging to } \mathbf{x}\} \qquad \square$$

PROOF See [5, ch.2]. ■

With this result, a complete metric space suitable for IFS has been arrived at.

2.4 Iterated Function Systems

Definition 2.8 (IFS) *An Iterated Function Systems (IFS) consists of a complete metric space $(\mathbf{X}, d)$ and a finite set of contraction mappings $\mathbf{w} = \{w_1, w_2, \ldots, w_N\}$ where $w_n : \mathbf{X} \to \mathbf{X}$, each with contraction factor c_n respectively, for $n = 1, 2, \ldots, N$. The IFS has contraction factor $c = \max\{c_1, c_2, \ldots, c_N\}$. The IFS can be denoted $\{\mathbf{X}; w_n, n = 1, 2, \ldots, N\}$.* $\square$

It remains to be shown that the above system is contractive, and has the desired contraction factor. Before doing so, a few lemmas are introduced, beginning with continuous maps in the metric space inducing maps on the Hausdorff space.

Lemma 2.3 *Let f be a continuous map on the metric space $(\mathbf{X}, d)$. Then f maps $\mathcal{H}(\mathbf{X}) \to \mathcal{H}(\mathbf{X})$* $\square$

PROOF Let S be a non-empty compact subset of $\mathbf{X}$. Then $\tilde{f}(S) = \{f(s) : s \in S\}$ is non-empty. Consider a sequence $\{y_n\}_{n=1}^{\infty} \in \tilde{f}(S)$. There exists an associated sequence, $\{x_n\}_{n=1}^{\infty} \in S$, such that $y_n = f(x_n)$. Now $\{x_n\}$ is an infinite sequence contained in a compact set, so there exists a convergent subsequence $\{x_{n_k}\}$ with $x_{n_k} \to \bar{x}$ as n (and hence, n_k) tends to infinity. By Lemma 2.1, f is continuous. Thus y_{n_k} converges, that is $y_{n_k} \to \bar{y} = f(\bar{x})$ as n tends to infinity. Therefore, $\tilde{f}(S)$ is compact, and $\tilde{f}(S) \in \mathcal{H}(\mathbf{X})$. ■

Having shown that sets under such functions will remain in the Hausdorff space, the goal is to to develop a version of the CMP (Theorem 2.2) for IFS. This result follows directly from the following lemmas, and the CMP. Proofs are straightforward, and omitted. Details may be found in [5, 21].

Definition 2.9 *Let $f \in Con(\mathbf{X}, d)$. Define the map $\tilde{f} : \mathcal{H}(\mathbf{X}) \to \mathcal{H}(\mathbf{X})$ by*

$$\tilde{f}(S) = \{f(s) : s \in S\}. \qquad \square$$

Lemma 2.4 *Let f be a contractive map on the metric space $(\mathbf{X}, d)$, with contraction factor c_f. Then $\tilde{f} : \mathcal{H}(\mathbf{X}) \to \mathcal{H}(\mathbf{X})$, $\tilde{f}(S) = \{f(s) : s \in S\}$ is a contractive on the metric space $(\mathcal{H}(\mathbf{X}), h)$, with contraction factor $c_{\tilde{f}}$ and furthermore $c_f = c_{\tilde{f}}$.* $\square$

Lemma 2.5 $h(A \cup B, C \cup D) \leq \max\{h(A, C), h(B, D)\} \forall A, B, C, D \in \mathcal{H}(\mathbf{X})$. $\square$

Lemma 2.6 *Let $(\mathbf{X}, \mathbf{d})$ be a metric space, and $\{w_1, w_2, \ldots, w_N\}$ be a finite set of contraction mappings on $(\mathcal{H}(\mathbf{X}), \mathbf{h})$, with contraction factor c_n for each w_n, respectively. Define $W : \mathcal{H}(\mathbf{X}) \to \mathcal{H}(\mathbf{X})$*

$$W(A) = \bigcup_{n=1}^{N} w_n(A) \qquad \forall A \in \mathcal{H}(\mathbf{X}).$$

Then W is a contraction mapping on $\mathcal{H}(\mathbf{X})$ with contraction factor

$$c = \max\{c_1, c_2, \ldots, c_N\} \qquad \square$$

Putting these results together, the Contraction Mapping Principle may be restated for IFS.

Theorem 2.4 (CMP for IFS) *Let $\{\mathbf{X}; w_n, n = 1, 2, \ldots, N\}$ be an IFS with contraction factor c. Then the map $W : \mathcal{H}(\mathbf{X}) \to \mathcal{H}(\mathbf{X})$,*

$$W(B) = \bigcup_{n=1}^{N} w_n(B)$$

is a contraction mapping on the complete metric space $(\mathcal{H}(\mathbf{X}), \mathbf{h}(\mathbf{d}))$ with contraction factor c. The unique fixed point of W, $\bar{A}_W \in \mathcal{H}(\mathbf{X})$ is such that the following hold:

$$\bar{A}_W = W(\bar{A}_W) = \bigcup_{n=1}^{N} w_n(\bar{A}_W) \tag{2.5}$$

$$\bar{A}_W = \lim_{n \to \infty} W^{\circ(n)}(B) \; \forall B \in \mathcal{H}(\mathbf{X}). \tag{2.6}$$

$\square$

Thus the IFS is a set of maps on a complete metric space $(\mathbf{X}, d)$, with an attractor (Equation 2.6) in $\mathcal{H}(\mathbf{X})$.

2.5 Algorithms

Definition 2.10 (affine transformation) *A mapping $S : \mathbf{X} \to \mathbf{X}$ is an affine transformation if*

$$S(x) = T(x) + b \,,$$

where $T : \mathbf{X} \to \mathbf{X}$ is a linear transformation. $\square$

If $w_1, w_2, \ldots, w_N$ in Definition 2.8 are all affine transformations, then the unique fixed point of W given by Theorem 2.4 is termed a *self-affine set*. Many interesting fractal sets are of this type.

From the above theory two methods are evident for producing renditions of self-affine sets on a computer. Here is a brief outline of each. Similar descriptions are found in both [21] and [5].

Brute Force Calculation

Start with a set C in $\mathcal{H}(\mathbf{X})$. Apply each map to the set, to generate a first approximation, $W(C)$. Iterate ($W^{\circ(k)}(C)$ is the k^{th} iterate of W on C) this process until it has converged[2], i.e. the computer representation is invariant under W. Algorithm 2.1 illustrates this process.

Algorithm 2.1 Brute force calculation of IFS

```
pick C ∈ H(X)
while W(C) ≠ C do
  C ⇐ W(C)
end while
```

Random Iteration Algorithm

Let $\{\mathbf{X}; w_n, n = 1, 2, \ldots, N\}$ be an IFS. To each w_n, assign a probability $p_n > 0$, where $\sum_{n=1}^{N} p_n = 1$. For example let $p_n = \frac{c_n}{\sum_{k=1}^{N} c_k}$, so that more slowly converging maps are chosen more often. Construct a sequence $\{x_i\}_{i=1}^{I}$ in the following manner: Pick $x_0 \in \mathbf{X}$, then repeatedly choose randomly (under the probabilities p_n) a map w_i, and let $x_i = w_i(x_{i-1})$. Algorithm 2.2 illustrates this process.

Algorithm 2.2 Random iteration algorithm

```
pick x_0 ∈ X
I ⇐ number of iterations
for i = 1 to I do
  pick random n with probability p_n
  x_i ⇐ w_n(x_{i-1})
end for
```

Under the appropriate conditions [5], the sequence $\{x_i\}_{i=1}^{I}$ converges to the attractor, $\bar{x}$, of the IFS. That is, $\lim_{i\to\infty} x_i = \bar{x}$. This process is often called the "Chaos Game". For large I , $\{x_i\}$ will appear to be randomly distributed across $\bar{x}$. This method can

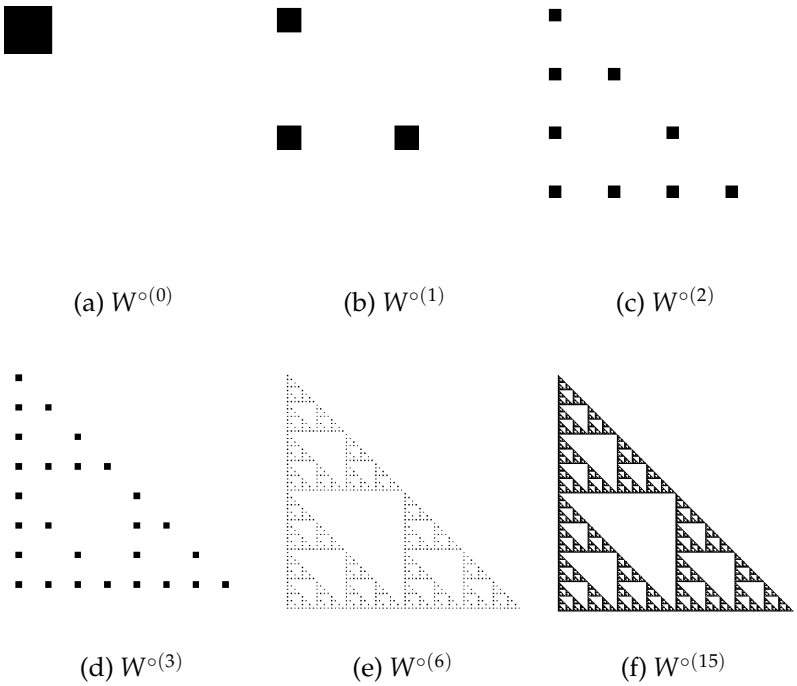
[2]To within some tolerance for the numerical representation used.

give a good rendering of the attractor, much faster than the previous method if individual calculations are expensive.

Example 2.2 (Sierpinski Triangle) The following maps define a IFS on $\mathbb{R}^2$, more precisely $[0, 100]^2$, whose attractor is often called the *Sierpinski Triangle*.

$$w_1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$
$$w_2 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 50 \end{bmatrix}$$
$$w_3 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 25 \\ 50 \end{bmatrix}$$ □

Figure 2.1 shows a progression of maps under $W(B) = \bigcup_{n=1}^{3} w_n(B)$, initially applied to the square $B = [0, 20]^2$, where W^n is the n^{th} iteration of W on B. Figure 2.2 shows a



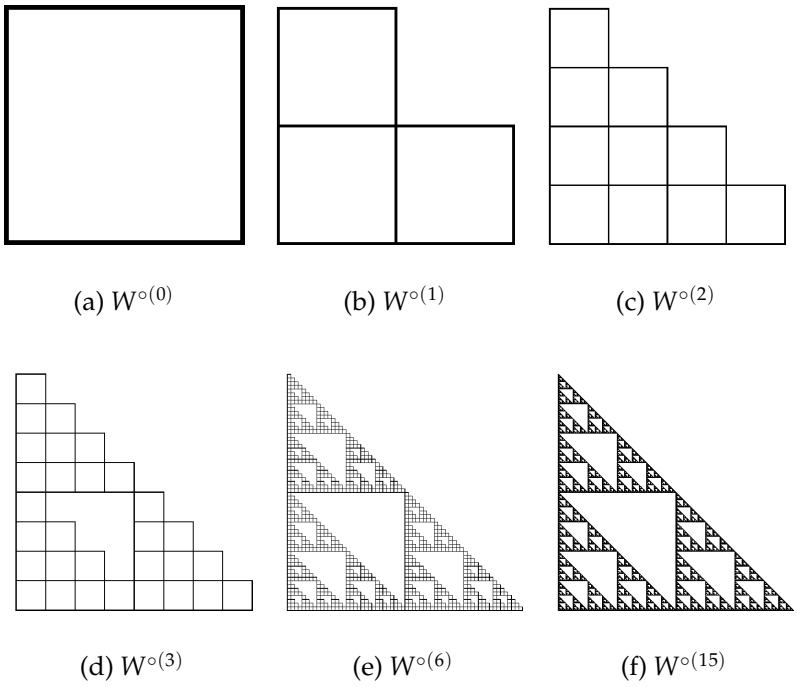
(a) $W^{\circ(0)}$ (b) $W^{\circ(1)}$ (c) $W^{\circ(2)}$

(d) $W^{\circ(3)}$ (e) $W^{\circ(6)}$ (f) $W^{\circ(15)}$

Figure 2.1: Sierpinski triangle from $[0, 20]^2$

similar progression of maps, this time starting with a 'box' of 100×100 with walls of width 10. This illustrates the application of $W(B)$ to different sets in $\mathbf{X}$ arriving at the attractor.

Figure 2.3 is a detail of the 'Sierpinski Triangle', illustrating the self-similarity of this set.



(a) $W^{\circ(0)}$ (b) $W^{\circ(1)}$ (c) $W^{\circ(2)}$

(d) $W^{\circ(3)}$ (e) $W^{\circ(6)}$ (f) $W^{\circ(15)}$

Figure 2.2: Sierpinski triangle from box (walls width 10)

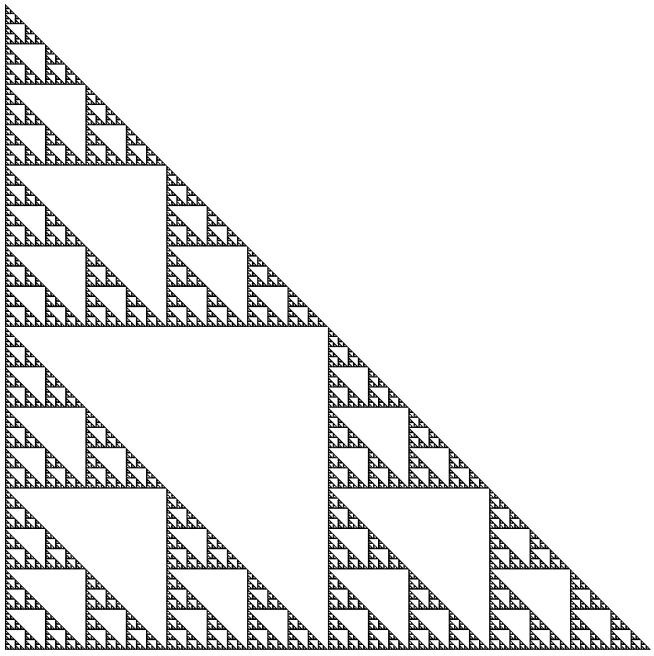


Figure 2.3: Sierpinski triangle

Example 2.3 (Fern) The following maps define a IFS with an attractor that looks very much like a fern.

$$
\begin{aligned}
w_1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 0 & 0 \\ 0 & 0.16 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
w_2 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.6 \end{bmatrix} \\
w_3 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 0.2 & -0.26 \\ 0.23 & 0.22 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.6 \end{bmatrix} \\
w_4 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0.44 \end{bmatrix}
\end{aligned}
$$

□

Figure 2.4 is a detail of this 'fern' IFS.

Figure 2.4: Fern IFS

2.6 An Inverse Problem for IFS

There has been much interest in the subject of fractals as models for natural objects [38], driven by the compelling likenesses such as that shown in Example 2.3. It is therefore natural to consider an inverse problem: given a set A in a complete metric space, (for example, a compact subset of $\mathbb{R}^2$), is it possible to find a function f whose attractor is A? The following states this more carefully.

Inverse Problem

> Let $(\mathbf{X}, d)$ be a complete metric space, and let $x \in \mathbf{X}$. Let $\epsilon > 0$. Does a contractive $f : \mathbf{X} \to \mathbf{X}$ exist such that $d(x, \bar{x}_f) < \epsilon$?

This can be approached as an optimization problem; given a space of parameters $\mathcal{P}$ defining contraction mappings $f \in \mathrm{Con}(\mathbf{X})$, search for the optimal parameters. That is, find $p \in \mathcal{P}$ such that the induced $f_p \in \mathrm{Con}(\mathbf{X})$ has the property $d(x, \bar{x}_{f_p}) \leq d(x, \bar{x}_{f_q})\ \forall q \in \mathcal{P}$. For the sake of generality, assume that this parameter space is rich enough that the optimal fixed point $\bar{x}_{f_p}$ is acceptably close to x. Finding p, however, can be a tedious process.

The following theorem, a consequence of Banach's contraction mapping principle, offers a more attractive approach. Instead of measuring the distance $d(x, \bar{x}_f)$, the collage distance, $d(x, f(x))$ can be utilized to simplify the problem.

Theorem 2.5 (Collage Theorem) *Let $\{\mathbf{X}; w_n, n = 1, 2, \ldots, N\}$ be an IFS with contractivity factor c (recall $0 \leq c < 1$). Let $\epsilon > 0$ and suppose there exists $A \in \mathcal{H}(\mathbf{X})$ such that $h(A, W(A)) \leq \epsilon$. Then the attractor (fixed point) of W, $\bar{A}_W$ is such that*

$$h(A, \bar{A}_W) \leq \frac{\epsilon}{1 - c}$$

or, equivalently,

$$h(A, \bar{A}_W) \leq \frac{1}{1 - c} h(A, W(A)) \qquad \forall A \in \mathcal{H}(\mathbf{X}). \qquad \square$$

PROOF

$$\begin{aligned} h(A, \bar{A}_W) &\leq h(A, W(A)) + h(W(A), \bar{A}_W) \\ &= h(A, W(A)) + h(W(A), W(\bar{A}_W)) \\ &\leq h(A, W(A)) + c h(A, \bar{A}_W). \end{aligned}$$

Therefore,

$$h(A, \bar{A}_W) \leq \frac{1}{1 - c} h(A, W(A)). \tag{2.7}$$

■

Corollary 2.2 (Anti-Collage Theorem) *Under the same conditions,*

$$h(A, \bar{A}_W) \geq \frac{1}{1+c} h(A, W(A)) \qquad \forall A \in \mathcal{H}(\mathbf{X}). \tag{2.8}$$

□

Therein lies the key to a method of attacking the inverse problem. Equation 2.7 shows that by finding maps W such that A is mapped close to itself, the attractor $\bar{A}_W$ is forced to be close (in the above sense) to A. Instead of tackling the inverse problem directly, we may consider the related problem:

Re-formulation of the inverse problem

> Let $(\mathbf{X}, d)$ be a complete metric space, and let $x \in \mathbf{X}$. Let $\epsilon > 0$. Does a contractive $f : \mathbf{X} \to \mathbf{X}$ exist such that $d(x, f(x)) < \epsilon$?

The above problem is much more computationally tractable, and forms the basis of most, if not all, fractal compression methods. It is important to note that while Theorem 2.5 points the way toward fractal-based approximation methods, the methods also require Theorem 2.1 (continuity of fixed points with respect to contraction maps), as such methods work by varying the map W.

Clearly, the Collage Theorem need not be stated in terms of IFS under the Hausdorff metric; it may be stated for any complete metric space and contractive function on that space. Together, the Collage Theorem and Anti-Collage theorem give upper and lower bounds on the approximation error $h(A, \bar{A}_W)$, thus a non-zero collage error ensures a positive approximation error. In general, optimizing the collage distance does not result in optimal maps. A detailed discussion of inverse problems for IFS may be found in [58], [26], where a generalized fractal transform is discussed.

2.7 IFS on Grey Level Maps

The IFS demonstrated on the Hausdorff metric have fixed points that are sets. Hence only binary 'pixels' may be represented this way; a 'pixel' is either in the set or it is not. Such images are called *bitmaps*. However, we wish to address a more general class of images; we are typically dealing with grey levels[3], not bitmaps. In this case in/exclusion in a fixed point set contained in of $\mathcal{H}(\mathbf{X})$ is not applicable.

The theory of IFSM, introduced by Forte and Vrscay [27], was developed to provide a way around this problem. This section briefly describes the application of IFS method over an appropriate function space $\mathcal{F}(\mathbf{X})$. Typically, signals are represented in $\mathcal{L}^p(\mathbb{R})$ (in practice, the space of functions is often $\mathcal{L}^2(\mathbb{R})$). A detailed discussion of IFSM may be found in [26], [58].

[3] Or colour, as will be addressed later.

Definition 2.11 (IFSM) *An (N-map) Iterated Function Systems on Grey Level Maps (IFSM) consists of a complete metric space* $(\mathbf{X}, d)$ *and two components*

1. ***IFS component***: $\mathbf{w} = \{w_1, w_2, \ldots, w_N\}$ *where each* $w_n : X \to X$ *is a contraction with contraction factor* c_n. *It is convenient, although not strictly necessary, to add the constraint that* $\cup_n w_n(x) = \mathbf{X}$, *thus ensuring that for all* $x \in \mathbf{X}$, $w_n^{-1}(x)$ *exists for some* n .

2. ***grey level component***: $\mathbf{\Phi} = \{\phi_1, \phi_2, \ldots, \phi_N\}$ *where each* $\phi_n : \mathbb{R} \to \mathbb{R}$ *is Lipschitz (i.e.* $\exists s_i \geq 0$ *such that* $|\phi_i(x) - \phi_i(y)| \leq s_i|x - y| \ \forall \ x, y \in \mathbb{R}$*).*

The IFSM may be denoted $\{\mathbf{X}; w_n, \phi_n, n = 1, 2, \ldots, N\}$. □

With the IFSM $\{\mathbf{X}; w_n, \phi_n, n = 1, 2, \ldots, N\}$, associate an operator performing the "fractal transform":

Definition 2.12 *Define an operator* $T : \mathcal{F}(\mathbf{X}) \to \mathcal{F}(\mathbf{X})$ *in the following manner. For each* $x \in X$ *define its* N *fractal components as*

$$g_i(x) = \begin{cases} \phi_i(f(w_i^{-1}(x))), & x \in w_i(\mathbf{X}) \\ 0, & x \notin w_i(\mathbf{X}). \end{cases} \tag{2.9}$$

And given an $f \in \mathcal{F}(\mathbf{X})$, *the image of* f *under* T *is*

$$(Tf)(x) = \sum_{i=1}^{N} g_i(x). \tag{2.10}$$

□

Thus the i^{th} "fractal component", $g_i(x)$, scales the grey-level value of f in the preimage $w_i^{-1}(x)$ if said preimage exists.

An interesting special case of this IFSM operator is arrived at by adding, in the role of a "condensation" function, some map $\theta(x) : \mathbf{X} \to \mathbf{X}$, as

$$(Tf)(x) = \theta(x) + \sum_{i=1}^{N} g_i(x). \tag{2.11}$$

In applications, affine IFS maps, and affine grey level maps (see below) are often used. The Hausdorf metric becomes cumbersome to use, and images are often modelled in $\mathcal{L}^p$ spaces, especially $\mathcal{L}^2(\mathbb{R})$. Under certain conditions, the IFSM is contractive.

Theorem 2.6 *Let* $\mathbf{X} \subset \mathbb{R}^D$. *Let* $\{\mathbf{X}; w_n, \phi_n, n = 1, 2, \ldots, N\}$ *be an* N*-map IFSM. Let* $f, g \in \mathcal{L}^p(\mathbf{X})$ *for some* $p \geq 0$. *Then*

$$d_p(Tf, Tg) \leq \sum_{n=1}^{N} c_n^{D/p} s_n d_p(f, g). \tag{□}$$

PROOF

$$
\begin{aligned}
d_p(Tf,Tg) &= \|Tf - Tg\|_p \\
&= \left[\int_{\mathbf{X}} \Big| \sum_{n=1}^{N} \phi_n(f(w_n^{-1}(x)) - \phi_n(g(w_n^{-1}(x)) \Big|^p dx \right]^{\frac{1}{p}} \\
&\leq \sum_{n=1}^{N} \left[\int_{\mathbf{X}_n} \Big| \phi_n(f(w_n^{-1}(x)) - \phi_n(g(w_n^{-1}(x)) \Big|^p dx \right]^{\frac{1}{p}} \quad \text{where } \mathbf{X}_n = w_n(X) \\
&\leq \sum_{n=1}^{N} \left[c_n^D \int_{\mathbf{X}} \Big| \phi_n(f(\xi)) - \phi_n(g(\xi)) \Big|^p d\xi \right]^{\frac{1}{p}} \\
&\leq \sum_{n=1}^{N} c_n^{D/p} s_n \left[\int_{\mathbf{X}} \Big| \phi_n(\xi) - \phi_n(\xi) \Big|^p d\xi \right]^{\frac{1}{p}} \\
&= \sum_{n=1}^{N} c_n^{D/p} s_n d_p(f,g).
\end{aligned}
$$

■

Thus the IFSM operator is contractive if $\sum_{i=1}^{N} c_i^{D/p} s_i < 1$. In principle, implementations of this method should check this contractivity factor; in practice it is rarely done.

The inverse problem for IFSM

The inverse problem can again be posed,

> Let $f \in \mathcal{L}^p(\mathbb{R})$ and $\epsilon > 0$. Does a contractive IFSM with fixed point $\bar{f}$ exist such that $d(f,\bar{f}) < \epsilon$?

As was the case with IFS, the key technique is the collage theorem. Again, we reformulate:

> Let $f \in \mathcal{L}^p(\mathbb{R})$ and $\epsilon > 0$. Does a contractive IFSM map T exist such that $d(f,Tf) < \epsilon$?

Instead of searching for IFSM that converge close to f, search for IFSM with associated operators T such that $d(f,Tf) < \epsilon$.

Affine grey level maps

In application to image compression, it will be convenient to consider for the grey level ϕ a family of affine maps,

$$\phi_n(x) = \alpha_n x + \beta_n. \tag{2.12}$$

Use of this family of maps greatly simplifies application of the collage theorem. Consider a function f on $\mathcal{L}^p(\mathbb{R})$ and wish to find maps ϕ_n that minimize the collage distance Δ,

$$\begin{aligned}\Delta^2 = \|f(x) - Tf(x)\|_p^2 &= <f(x) - Tf(x), f(x) - Tf(x)> \\ &= <f(x) - \sum_{n=1}^{N} \phi_n(f(w_n^{-1}(x))), f(x) - \sum_{n=1}^{N} \phi_n(f(w_n^{-1}(x)))> .\end{aligned}$$

Make the simplifying assumption that the IFS maps 'w_n' are non-overlapping. Denoting $\xi = w_n^{-1}(x)$ as before, each of the N components of Δ may be minimized separately,

$$\begin{aligned}\Delta_n^2 &= <f(x) - \alpha_n f(\xi) - \beta_n, f(x) - \alpha_n f(\xi) - \beta_n> \\ &= <f(x), f(x)> + \alpha_n^2 <f(\xi), f(\xi)> + \beta_n^2 \\ &\quad + 2\alpha_n\beta_n <f(\xi), 1> - 2\alpha_n <f(\xi), f(x)> - 2\beta_n <f(x), 1> .\end{aligned}$$

Applying the conditions $\frac{\partial \Delta_n^2}{\partial \alpha_n} = 0$, and $\frac{\partial \Delta_n^2}{\partial \beta_n} = 0$ results in a linear system that may be solved for α_n, β_n as

$$\begin{aligned}\alpha_n &= \frac{<f(\xi), 1><f(x), 1> - <f(\xi), f(x)>}{<f(\xi), 1>^2 - <f(\xi), f(\xi)>} \\ \beta_n &= \frac{<f(\xi), f(x)><f(x), 1> - <f(x), 1><f(\xi), f(\xi)>}{<f(\xi), 1>^2 - <f(\xi), f(\xi)>}.\end{aligned}$$

In the case of the $\mathcal{L}^2(\mathbb{R})$ norm, this is a linear least squares fit. Note that if $f(x) = k$ is constant, the determinant of the above system is zero. In this case, the system may be solved by setting $\alpha_n = 0$, and $\beta_n = <f(x), 1> = k$.

2.8 IFS in the Wavelet Domain

This section describes an application of the IFSM to the wavelet domain, called IFSW, as introduced in [27, 59, 40]. Consider $f \in \mathcal{L}^2(\mathbb{R})$ that admit the wavelet expansions[4] (see Equation 1.71):

$$f(x) = b_{0,0}\phi(x) + \sum_{j=-\infty}^{0} \sum_{k=0}^{2^{-j}-1} c_{j,k}\psi_{j,k}(x). \tag{2.13}$$

Recall from §1.3.3 the wavelet ψ and scaling function ϕ generate families of translated and dilated versions

$$\begin{aligned}\phi_{j,k}(x) &= 2^{-j/2}\phi(2^{-j}x - k) \qquad \forall j, n \in \mathbb{Z} \\ \psi_{j,k}(x) &= 2^{-j/2}\psi(2^{-j}x - k) \qquad \forall j, n \in \mathbb{Z}.\end{aligned}$$

In §1.6 wavelet expansions of this type were displayed as coefficient trees:

[4] In practice, finite resolution is needed, so j will sum from $L < 0$, not $-\infty$.

$b_{0,0}$							
$c_{0,0}$							
$c_{-1,0}$				$c_{-1,1}$			
$c_{-2,0}$		$c_{-2,1}$		$c_{-1,2}$		$c_{-1,3}$	
$C_{-3,0}$	$C_{-3,1}$	$C_{-3,2}$	$C_{-3,3}$	$C_{-3,4}$	$C_{-3,5}$	$C_{-3,6}$	$C_{-3,7}$

which will aid the discussion to follow. To motivate the IFSW, consider the following example.

Example 2.4 Consider a function f under the Haar basis. Recall (Example 1.1) that the Haar basis has

$$\begin{aligned}\phi(x) &= I_{[0,1]}(x)\\ \psi(x) &= I_{[0,1/2]}(x) - I_{[1/2,1]}(x).\end{aligned}$$

Define the simple two-map IFSM defined by

$$\begin{aligned} w_1(x) &= \frac{1}{2}x, & \phi_1(x) &= \alpha_1 x + \beta_1,\\ w_2(x) &= \frac{1}{2}x + \frac{1}{2}, & \phi_2(x) &= \alpha_2 x + \beta_2.\end{aligned}$$

The fractal operator (from Equation 2.10) will be

$$(Tf)(x) = \sum_{i=1}^{2} g_i(x) \tag{2.14}$$

$$= \alpha_1 f(2x) + \beta_1 I_{[0,1/2]} + \alpha_2 f(2x-1)\beta_2 I_{[1/2,1]} \tag{2.15}$$

$$= \alpha_1 b_{0,0}\phi_{0,0}(2x) + \alpha_1 \sum_{j=-\infty}^{0}\sum_{k=0}^{2^{-j}-1} c_{j,k}\psi_{j,k}(2x) + \beta_1 I_{[0,1/2]} \tag{2.16}$$

$$+ \alpha_2 b_{0,0}\phi_{0,0}(2x-1) + \alpha_2 \sum_{j=-\infty}^{0}\sum_{k=0}^{2^{-j}-1} c_{j,k}\psi_{j,k}(2x-1) + \beta_2 I_{[1/2,1]}\,. \tag{2.17}$$

Note that

$$\begin{aligned}\psi_{j,k}(2x-1) &= 2^{-j/2}\psi\left(2^{-j}(2x-1) - k\right)\\ &= 2^{-1/2}\,2^{\frac{-(j-1)}{2}}\psi\left(2^{-(j-1)}\,x - (k+2^{-j})\right)\\ &= 2^{-1/2}\,\psi_{j-1,k+2^{-j}}(x).\end{aligned}$$

Similarly,

$$\begin{aligned}\psi_{j,k}(2x) &= 2^{-1/2}\psi_{j-1,k}(x),\\ \phi_{j,k}(2x-1) &= 2^{-1/2}\phi_{j-1,k+2^{-j}}(x),\\ \phi_{j,k}(2x) &= 2^{-1/2}\phi_{j-1,k}(x).\end{aligned}$$

Substituting these into Equation 2.17 and regrouping terms yields

$$\begin{aligned}(Tf)(x) &= \frac{\alpha_1}{\sqrt{2}} \sum_{j=-\infty}^{0} \sum_{k=0}^{2^{-j}-1} c_{j,k}\psi_{j-1,k}(x) + \frac{\alpha_2}{\sqrt{2}} \sum_{j=-\infty}^{0} \sum_{k=0}^{2^{-j}-1} c_{j,k}\psi_{j-1,k+2^{-j}}(x) \\ &\quad + \frac{\alpha_1}{\sqrt{2}} b_{0,0}\phi_{-1,0}(x) + \beta_1 I_{[0,1/2]} + \frac{\alpha_2}{\sqrt{2}} b_{0,0}\phi_{-1,1}(x) + \beta_2 I_{[1/2,1]} \\ &= \frac{\alpha_1}{\sqrt{2}} \sum_{j=-\infty}^{0} \sum_{k=0}^{2^{-j}-1} c_{j,k}\psi_{j-1,k}(x) + \frac{\alpha_2}{\sqrt{2}} \sum_{j=-\infty}^{0} \sum_{k=0}^{2^{-j}-1} c_{j,k}\psi_{j-1,k+2^{-j}}(x) \\ &\quad + (\alpha_1 b_{0,0} + \beta_1) I_{[0,1/2]} + (\alpha_2 b_{0,0} + \beta_2) I_{[1/2,1]} \,.\end{aligned}$$

Ignoring the constant contributions, the coefficient tree may be written as

<table>
<tr><td colspan="4">0</td></tr>
<tr><td colspan="4">0</td></tr>
<tr><td colspan="2">$\frac{\alpha_1}{\sqrt{2}}C_{0,0}$</td><td colspan="2">$\frac{\alpha_2}{\sqrt{2}}C_{0,0}$</td></tr>
<tr><td>$\frac{\alpha_1}{\sqrt{2}}C_{-1,0}$</td><td>$\frac{\alpha_1}{\sqrt{2}}C_{-1,1}$</td><td>$\frac{\alpha_2}{\sqrt{2}}C_{-1,0}$</td><td>$\frac{\alpha_2}{\sqrt{2}}C_{-1,1}$</td></tr>
</table>

Or, more compactly,

<table>
<tr><td colspan="2">0</td></tr>
<tr><td colspan="2">0</td></tr>
<tr><td>$\frac{\alpha_1}{\sqrt{2}}C_{0,0}$</td><td>$\frac{\alpha_2}{\sqrt{2}}C_{0,0}$</td></tr>
</table>

Thus we have defined an operator on the coefficient trees that maps the original tree $C_{0,0}$ onto two scaled copies of $C_{0,0}$, at one level lower in the tree. □

The above example gives the flavour of IFSW maps. Under the Haar basis, this process is equivalent to a local IFSM [26]. This approach can be generalized to other compactly supported orthogonal[5] bases [26, 19]. Details may be found in [59, 40]; here a simple example (from [40]) is presented in order to illustrate the process.

Consider mapping from the top level of the wavelet decomposition to the next level down, with four scaling maps:

$$\begin{aligned} W_1 &: C_{-1,0} \to C_{-2,0} \quad C_{-2,0} = \alpha_1 C_{-1,0} \\ W_2 &: C_{-1,1} \to C_{-2,1} \quad C_{-2,1} = \alpha_2 C_{-1,1} \\ W_3 &: C_{-1,0} \to C_{-2,2} \quad C_{-2,2} = \alpha_3 C_{-1,0} \\ W_4 &: C_{-1,1} \to C_{-2,3} \quad C_{-2,3} = \alpha_4 C_{-1,1} \end{aligned}$$

Denote the fractal wavelet (FW) maps as M; it may be represented in a diagram as

[5]The generalization to biorthogonal bases has not been done.

$M : C_{0,0} \Longrightarrow$

$c_{0,0}$			
$c_{-1,0}$		$c_{-1,1}$	
$\alpha_1 C_{-1,0}$	$\alpha_2 C_{-1,1}$	$\alpha_3 C_{-1,0}$	$\alpha_4 C_{-1,1}$

If M is iterated a second time, the result is

$M^{\circ(2)} : C_{0,0} \Longrightarrow$

$c_{0,0}$							
$c_{-1,0}$				$c_{-1,1}$			
$\alpha_1 c_{-1,0}$		$\alpha_2 c_{-1,1}$		$\alpha_3 c_{-1,0}$		$\alpha_4 c_{-1,1}$	
$\alpha_1^2 C_{-1,0}$	$\alpha_1\alpha_2 C_{-1,0}$	$\alpha_2\alpha_1 C_{-1,1}$	$\alpha_2^2 C_{-1,1}$	$\alpha_3^2 C_{-1,0}$	$\alpha_3\alpha_4 C_{-1,0}$	$\alpha_4\alpha_3 C_{-1,1}$	$\alpha_4^2 C_{-1,1}$

Iterating this process will converge to a wavelet expansion of a function in $\mathcal{L}^2(\mathbb{R})$ provided that $|a_i| < \frac{1}{\sqrt{2}}\ \forall i$ [40]. Note that mapping from the first to second levels of the coefficient trees was merely one possible choice of maps W_n; many other choices of map are possible. In order to apply this approach to image compression, it is necessary to work in the space of two-dimensional wavelet coefficient trees.

Extension of the IFS theory to wavelet coefficient trees

A generalized approach to IFSW on images is described by Vrscay[59]. The following is a brief outline of those results. Instead of mapping binary trees as in the above example, quadtrees (or octrees, for three-dimensional images) are mapped to each other. The two-dimensional case is discussed here; the three-dimensional case is analogous For simplicity, make a restriction on the maps considered. There will be a *parent level* and a *child level*, and the roots of all parent trees will appear in one block, whilst the roots of all child trees appear in another block. The parent and child levels are represented by integers i_1^* and i_2^*, respectively, where $i_2^* < i_1^* \leq 0$.

Recalling the notation of §1.5.1, we wish to find functions $f \in \mathcal{L}^2(\mathbb{R})$ that admit wavelet expansions of the form

$$f(x,y) = b_{0,0,0}\Phi_{0,0,0}(x,y) + \sum_{i=-\infty}^{0}\sum_{j=0}^{2^{-i}-1}\sum_{k=0}^{2^{-i}-1}\left[c^h_{i,j,k}\Psi^h_{i,j,k}(x,y) + c^v_{i,j,k}\Psi^v_{i,j,k}(x,y) + c^d_{i,j,k}\Psi^d_{i,j,k}(x,y)\right]. \quad (2.18)$$

Definition 2.13 *Denote the space of all $f \in \mathcal{L}^2(\mathbb{R})$ admitting wavelet expansions of the form given by Equation 2.18 as $\mathcal{L}^2_0(\mathbb{R})$. This space is complete with respect to the usual $\mathcal{L}^2(\mathbb{R})$ metric [59].* □

The wavelet coefficients may be arranged into a 'pyramid' of blocks (e.g., Figure 1.9), consisting of the zeroth level average block A_0 and the detail blocks $D^{\lambda}_{i_1^*,j,k}$. Here $\lambda \in \{h,v,d\}$ and $i_2^* + 1 \leq i_1^* \leq 0$.

Define the following sets of affine transformations on the blocks. The child blocks are two-dimensional, with $2^{2(-i_2^*)}$ coefficients ranging $0 \leq j,k \leq 2^{-i_2^*}-1$. To emphasize the fixed parent-child block relationships, we denote (for each subband) the parent's array positions as functions of the child's; for example $j^h(j,k)$ and $k^h(j,k)$. In this setting, the block transforms will be:

$$\begin{aligned} W^h_{j,k} &: D^h_{i_1^*,j^h(j,k),k^h(j,k)} \rightarrow D^h_{i_2^*,j,k}, & D^h_{i_2^*,j,k} &= \alpha^h_{j,k} D^h_{i_1^*,j^h(j,k),k^h(j,k)} \\ W^v_{j,k} &: D^v_{i_1^*,j^v(j,k),k^v(j,k)} \rightarrow D^v_{i_2^*,j,k}, & D^v_{i_2^*,j,k} &= \alpha^v_{j,k} D^v_{i_1^*,j^v(j,k),k^v(j,k)} \\ W^d_{j,k} &: D^d_{i_1^*,j^d(j,k),k^d(j,k)} \rightarrow D^d_{i_2^*,j,k}, & D^d_{i_2^*,j,k} &= \alpha^d_{j,k} D^d_{i_1^*,j^d(j,k),k^d(j,k)}. \end{aligned} \tag{2.19}$$

The above defines an IFSW operator, call it M. In order for the IFS theory to apply to this operator, it remains to show that M is contractive in some suitable complete metric space of wavelet coefficients.

Definition 2.14 (Coefficient quadtree space $\mathcal{Q}$) *Similar to the one-dimensional example, denote an (infinite) quadtree as C and the coefficients of C as $c_{i,j,k}$ If a subtree is rooted on subband λ at level i, position j,k, denote it as $C^\lambda_{i,j,k}$. Let $\mathcal{Q}$ denote the set of all real quadtrees C that are square-summable,*

$$\mathcal{Q} = \left\{ C : c_{i,j,k} \in \mathbb{R}, i \leq 0, 0 \leq j,k \leq 2^{-i}-1, \sum_{i,j,k} |c_{i,j,k}|^2 < \infty \right\}. \tag{2.20}$$

□

Definition 2.15 (Quadtree metric) *Denote the $\ell^2(\mathbb{N})$ metric on $\mathcal{Q}$ as $d_\mathcal{Q}$. That is,*

$$d_\mathcal{Q}(C,D) = \left[\sum_{i,j,k} |c_{i,j,k} - d_{i,j,k}|^2 \right]^{1/2}, \ \forall\, C,D \in \mathcal{Q} \tag{2.21}$$

□

Definition 2.16 (Quadtree product) *The inner product between two quadtrees in $\mathcal{Q}$ is*

$$<C,D>_\mathcal{Q} = \sum_{i,j,k} c_{i,j,k} d_{i,j,k}, \ C,D \in \mathcal{Q} \tag{2.22}$$

□

In application to image compression, the goal will be to approximate functions $f \in \mathcal{L}^2_0(\mathbb{R}^2)$ with wavelet coefficient expansions as given in Equation 2.18. For this reason, we define a subset of $\mathcal{Q}$ for which the wavelet coefficient blocks A_0 and $D^\lambda_{i_1^*,j,k}$, $\lambda \in \{h,v,d\}$, $i_2^*+1 \leq i_1^* \leq 0$ are fixed.

Definition 2.17 *For any $f \in \mathcal{L}^2_0(\mathbb{R}^2)$ and $i_2^* \in \mathbb{N}$, Let $\mathcal{Q}_{f,i_2^*}$ denote the set of all quadtrees in $\mathcal{Q}$ with blocks at levels $i_2^*+1 \leq i \leq 0$ fixed.* □

The following propositions lay the framework for applying the previously discussed IFSM results to this space [59].

Proposition 2.4 *The distance function*

$$d_{\mathcal{Q}_{f,i_2^*}}(C,D) = \max_{\lambda,j,k} d_{\mathcal{Q}}\left(C^{\lambda}_{i_2^*,j,k}, D^{\lambda}_{i_2^*,j,k}\right),\ C,D \in \mathcal{Q}_{f,i_2^*}$$

defines a metric on $\mathcal{Q}_{i_2^*}$. *Furthermore, the metric space* $(\mathcal{Q}_{f,i_2^*}, d_{\mathcal{Q}_{i_2^*}})$ *is complete.* □

Proposition 2.5 *Given* $f \in \mathcal{L}^2_0(\mathbb{R}^2)$ *and* $-i_2^* \in \mathbb{N}^+$, *the operator* M *(Equation 2.19) maps* $\mathcal{Q}_{f,i_2^*}$ *into itself. Furthermore,*

$$d_{\mathcal{Q}_{f,i_2^*}}(M(C),M(D)) \leq c_{f,i_2^*} d_{\mathcal{Q}_{f,i_2^*}}(C,D),$$

where

$$c_{f,i_2^*} = 2^{-(i_2^*-i_1^*)} \max_{\lambda,j,k} |\alpha^{\lambda}_{j,k}|$$ □

Corollary 2.3 *By the Banach contraction mapping principle, if* $c_{f,i_2^*} \leq 1$ *then there is a unique fixed point of* M. *That is, there exists a unique* $\bar{C} \in \mathcal{Q}_{f,i_2^*}$ *such that* $M(\bar{C}) = \bar{C}$. $\bar{C}$ *is an attractor of* M; *iteration of* M *will converge to* $\bar{C}$. □

The inverse problem for IFSW

Having made the connection between wavelet coefficient trees and IFS type operators, the previously developed theory may now be applied. Thus the collage theorem is again applicable, and the inverse problem can be stated in this context as:

> Let $f \in \mathcal{L}^2_0(\mathbb{R}^2)$ and $-i_2^* \in \mathbb{N}^+$. Let $\epsilon > 0$. Does an IFSW map M exist such that $d_{\mathcal{Q}_{f,i_2^*}}(C, M(C)) \leq \epsilon$?

As was the case with IFSM, the approximation problem has become a question of minimizing the collage distance Δ:

$$\begin{aligned} \Delta &= d_{\mathcal{Q}_{f,i_2^*}}(C, M(C)) \\ &= \max_{\lambda,j,k} \Delta^{\lambda}_{j,k}, \end{aligned} \tag{2.23}$$

where

$$\Delta^{\lambda}_{j,k} = d_{\mathcal{Q}_{f,i_2^*}}(D^{\lambda}_{i_2^*,j,k}, \alpha^{\lambda}_{j,k} D^{\lambda}_{i_1^*,j^{\lambda}(j,k),k^{\lambda}(j,k)})\ \lambda \in \{h,v,d\}\ 0 \leq j,k \leq 2^{-i_2^*} - 1 \tag{2.24}$$

This is a quadratic programming problem. In practice, similar to the IFSM case, this minimization problem is treated as a "least-squares" fit to find the scaling factors $\alpha^{\lambda}_{j,k}$ by

$$\alpha^{\lambda}_{j,k} = \frac{<D^{\lambda}_{i_1^*,j^{\lambda}(j,k),k^{\lambda}(j,k)}, D^{\lambda}_{i_2^*,j,k}>_{\mathcal{Q}}}{<D^{\lambda}_{i_2^*,j,k}, D^{\lambda}_{i_2^*,j,k}>_{\mathcal{Q}}} \tag{2.25}$$

In order to ensure contraction of the operator M, there is a constraint of $|\alpha^{\lambda}_{j,k}| \leq 2^{i_2^*-i_1^*}$. In practice, this is rarely checked. In fact, the nature of wavelet coefficient decay near strong edges in the spatial domain makes this condition difficult to meet for all scaling factors[59].

Remarks

In this chapter, IFS as a method of approximation of a set was introduced. This forms the core behind fractal methods of compression, but is not directly applicable to non-binary images (the majority). The extension of IFS to the IFSM operating on $\mathcal{L}^2(\mathbb{R})$, and associating the IFS with a series of grey level maps was made. The concept of IFSW, which are IFSMs applied to wavelet coefficients was also introduced.

The inverse problem for all three (IFS, IFSM, IFSW) domains was discussed. The collage theorem was introduced as an approach to the problem. This method will lead directly to compression algorithms for grey level images (see §4.2) and wavelet images (see §4.4).

Although the notation becomes quite cumbersome, it should be apparent from the preceding discussion that the approach to deal with wavelet coefficient trees in two dimensions will extend to three dimensions in a straightforward manner.

Chapter 3

Entropy Coding

A basic overview of the relevant parts of Information Theory is presented, followed by a brief discussion of entropy coding in general. A more detailed presentation of the arithmetic coding algorithm (the one that will be used in the work to follow) is then presented, followed by some discussion of performance and implementation issues.

A broader introduction to entropy coding and its applications to image compression may be found in [48].

3.1 Background

Notation 3.1 *The following notation will be used throughout.*

$\lg(x) = \log_2(x)$

$P(x) =$ *probability of event* x

$P(x|y) =$ *conditional probability of event* x *given event* y □

Notation 3.2 *Consider a finite set* $\mathcal{A} = \{a_1, a_2, \ldots, a_m\}$. *Call* $\mathcal{A}$ *an* ***alphabet****, and the elements of* $\mathcal{A}$ ***symbols****. Let* $\mathcal{A}^N$ *denote the set of sequences of length* N *taken from* $\mathcal{A}$. *Such sequences are called* ***strings*** *of length* N. *Denote the set of all finite strings from* $\mathcal{A}$ *as* $\mathcal{A}^*$. □

Strings are commonly used in the information theory literature as referring to a subclass of sequences; the elements of a string typically take on values from a finite alphabet (as opposed to, for example, a sequence of real numbers).

Definition 3.1 *Let* X *be a random variable on an alphabet* $\mathcal{A}$. *Define the* ***cumulative probability density function (cdf)*** *as*

$$F_X(x) = P(X \leq x) = P(\{a \in \mathcal{A} : X(a) \leq x\}).$$ □

Notation 3.3 *Where there is no chance of ambiguity, the subscript references to a particular random variable will be dropped. Thus if a is a symbol from an alphabet, then $P(a)$ denotes the probability of that symbol occurring, and $F(x)$ denotes the cumulative probability, $P(X \leq x)$.* □

Notation 3.4 *For an alphabet $\mathcal{A}$, denote a **probability model***

$$\mathcal{P} = \{P(a_1), P(a_2), \ldots, P(a_m)\}$$

with the constraint that

$$\sum_{i=1}^{m} P(a_i) = 1.$$ □

3.1.1 Information Theory

In his seminal paper of 1948 [49], Claude Shannon introduced a quantity called *self-information*.

Definition 3.2 (self-information) *Let $P(x)$ be the probability of occurrence for a random event x. Define the self-information, I, associated with x as*

$$I(x) = \log_y \frac{1}{P(x)} = -\log_y P(x),$$

where the base of the log, y, is determined by the unit of information under discussion (i.e., the base of the number system in use). So for binary information, use $\log_2$, and so on. Where there is no possibility of ambiguity, the base will be dropped from the notation. □

This definition is quite intuitive: information is carried by less likely events. One of the important aspects of Shannon's work was to introduce a quantity he called *entropy*. Roughly speaking, the entropy is a weighted average of the self-information.

Definition 3.3 (entropy) *Consider a source $\mathcal{S}$ with alphabet $\mathcal{A}$. If $\mathcal{S}$ generates an infinite sequence $\{x_n\}$ (with $x_n \in \mathcal{A}\ \forall n$), then define the **entropy** of the source as*

$$H(\mathcal{S}) = \lim_{n\to\infty} \frac{1}{n} G_n$$

where

$$\begin{aligned} G_n &= -\sum_{i_1=1}^{m}\sum_{i_2=1}^{m}\ldots \\ &\quad \sum_{i_n=1}^{m} P(x_1{=}a_{i_1}|x_2{=}a_{i_2}|\ldots|x_n{=}a_{i_n}) \log P(x_1{=}a_{i_1}|x_2{=}a_{i_2}|\ldots|x_n{=}a_{i_n}). \end{aligned}$$ □

Definition 3.4 (first order entropy) *Now make the assumption that the symbols are* ***independent and identically distributed (i.i.d)*** *then* $P(a_i|a_j) = P(a_i)$*, i.e. the events are independent so the conditional probabilities are simply the probabilities. In this case, the above sums will collapse, leaving the* ***first-order entropy***

$$H(\mathcal{S}) = -\sum_{i=1}^{m} P(x_1{=}a_i) \log P(x_1{=}a_i).$$

Or, since the sum runs over all symbols in $\mathcal{A}$*,*

$$H(\mathcal{S}) = -\sum_{i=1}^{m} P(a_i) \log P(a_i). \qquad \square$$

In practice, this quantity is often referred to as "the entropy", but implicit in this statement is the assumption of an i.i.d. source.

Shannon showed (paraphrased here) that the best theoretical performance possible for any encoding algorithm[1] is to encode the source with an average number of units (e.g. bits, for a binary source) equal to the entropy of the source. This value is important for our purposes, because the entropy turns out to be a measure of the average number of binary symbols needed to encode a string output from $\mathcal{S}$ [49].

3.2 Entropy Coding

When looking at various methods of encoding a signal (in this work, discrete; continuous is also valid), Shannon's work shows that the best rate achievable would be equal to the entropy of the signal. The obvious question then, is to ask if this limit is achievable. In the same paper a method now known as Shannon-Fano coding was detailed[49]. With this encoding, for any signal, the encoding rate approaches the entropy of the signal as the length of the signal approaches infinity. Another key result in this work was to show that for a general source $\mathcal{S}$, the average code length l of the Shannon-Fano algorithm will obey the inequality

$$H(\mathcal{S}) \leq l < H(\mathcal{S}) + 1. \tag{3.1}$$

Although Shannon-Fano coding proved that nearly optimal coders were possible, the algorithm is impractical to implement. Shortly after Shannon's work, D.A. Huffman introduced what is now called Huffman Coding [32], which proved to be quite a flexible and useful coding scheme. Although Huffman coding is not used in the compression algorithms to be presented later, much of the preliminary work on adaptive and context based coders was done with Huffman coders [16, 20]. More recently, a method called Arithmetic Coding has gained popularity due to several practical advantages it has over Huffman coding. In the encoding of a sequence of length n, Huffman coders must generate codewords for all sequences of length n. The primary advantage is that Arithmetic coding does not have this restriction[9].

[1] Hence any compression algorithm, such as those presented here.

3.3 Arithmetic Coding

This section describes a useful algorithm for performing entropy coding. A fairly abstract discussion of the general idea is followed by an exact description of the algorithm as it may be implemented in a computer system.

3.3.1 Idea of Arithmetic Coding

Arithmetic coding is a method of addressing the unit interval. Using this algorithm, any string $\mathbf{s} \in \mathcal{A}^N$ may be mapped to the unit interval in such a way that the resulting "tag" (e.g., a rational number) will lie within a subinterval of $[0, 1)$ and this subinterval will be unique for all strings in $\mathcal{A}^N$. Of course, since finite sequences are being mapping into $[0, 1)$, there will be an uncountable number of possible representations under different mappings — one such mapping must be picked so that all sequences of length N are uniquely determined by subintervals.

The algorithm, conceptually, consists of two parts:

- mapping a string to some number $t \in [0, 1)$
- encoding t

Although in practice these are done simultaneously, for clarity they are described separately.

Mapping a string into $[0, 1)$

Consider a source, $\mathcal{S}$, alphabet $\mathcal{A}$, and string $\mathbf{s} \in \mathcal{A}^N$. In order to code $\mathbf{s}$, a probability model $\mathcal{P}$ is needed. Suppose that $\mathcal{A}$ consists of m symbols and furthermore (this is not generally the case), that $\mathcal{P}$ represents the actual probability distribution of $\mathcal{S}$. Having this model, partition the unit interval into m subintervals using the cdf associated with $\mathcal{P}$ in the following manner:

$$\begin{aligned} I_{a_1} &= [0, F(a_1)) \\ I_{a_i} &= [F(a_{i-1}), F(a_i)), \qquad \text{for } 1 \leq i \leq m\,. \end{aligned}$$

Note that $\{I_{a_i}\}_{i=1}^m$ forms a disjoint cover of $[0, 1)$,

$$I = I_{a_1} \cup I_{a_2} \cup \cdots \cup I_{a_m},$$

and therefore

$$\forall x \in [0, 1), \exists \text{ unique } a_i \in \mathcal{A} \text{ such that } x \in I_{a_i}\,.$$

Thus define $I_x \equiv I_{a_i}$.

Now consider the first symbol s_1 in $\mathbf{s}$. Chose the (unique) subinterval I_{s_1}, according to the above scheme. Now move to the next symbol, s_2, and repeat the process, with the exception that instead of starting with the interval $[0,1)$, begin with I_{s_1}. This interval is subdivided in the analogous way, either mapping I_{s_1} onto $[0,1)$ before subdividing or, equivalently, scaling the $[F(a_{i-1}), F(a_i))$ intervals by the length of I_{s_1} and translating. Upon conclusion, a subinterval $I_{s_2} \subset I_{s_1}$ has been found. This process is repeated for each symbol in $\mathbf{s}$, where at each step the subinterval from the previous iteration is the initial interval.

After all symbols have been processed, a nested set of intervals $\{I_{s_n}\}_{n=1}^{N}$ has been defined where $I_{s_1} \supset I_{s_2} \supset \cdots \supset I_{s_N}$.

Encoding t

With this construction, any number lying in the final subinterval suffices to encode the string, so pick some $t \in I_{s_{n-1}}$, and call t the *tag* representing string $\mathbf{s}$. In order to make this a well defined mapping, $\mathcal{A}^N \to [0,1)$, t must be defined uniquely. If $I_{s_N} = [a,b)$, the choice of $t = \frac{b-a}{2}$ will serve this purpose.

At this point any convenient coding of the tag t may be picked. Of course, in applications one is constrained to pick tags that are readily representable in a computer. Algorithm 3.1 illustrates this process; a more rigorous description will be presented in § 3.3.2.

Algorithm 3.1 Encoding a string $\mathbf{s}$ of length N

$I \Leftarrow [0,1)$
for $n = 1$ to N **do**
  $s \Leftarrow s_n$
  subdivide I according to cdf
  such that $I = I_{a_1} \cup I_{a_2} \cup \cdots \cup I_{a_m}$
  $I \Leftarrow I_s$
end for
pick tag $t \in I$ {encode t}

Example 3.1 Suppose $\mathcal{S}$ is a source on the alphabet $\mathcal{A} = \{a,b,c\}$ with probability model $\mathcal{P} = \{0.7, 0.2, 0.1\}$, and that $\mathcal{S}$ produces the sequence $\mathbf{s} = \{s_n\} = \{a,a,b\}$. Proceed to generate a tag encoding the sequence, following algorithm 3.1:

- $I = [0,1)$
- subdivision: $I = [0,0.7) \cup [0.7,0.9) \cup [0.9,1)$
- input symbol $s_1 = a$, so tag lies in the first subinterval, $[0,0.7)$
- $I \Leftarrow [0,0.7)$

- subdivision: $I = [0, 0.49) \cup [0.49, 0.63) \cup [0.63, 0.7)$
- input symbol $s_2 = a$, so $I \Leftarrow [0, 0.49)$
- subdivision: $I = [0, 0.343) \cup [0.343, 0.441) \cup [0.441, 0.49)$
- input symbol $s_3 = b$, so $I \Leftarrow [0.343, 0.441)$
- pick any value in I, e.g. 0.4 to represent the sequence

In Figure 3.1, this encoding process is represented graphically. □

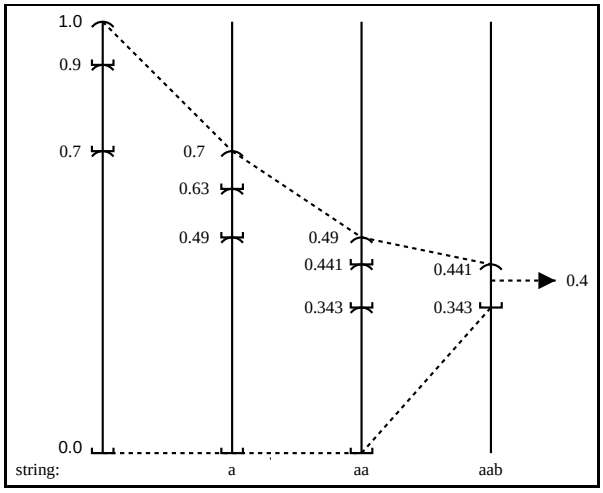


Figure 3.1: Encoding of the sequence "aab"

Mapping a $t \in [0, 1)$ to a string

Decoding a sequence is nearly the reverse procedure to that of encoding. To perform decoding requires a tag $t \in [0, 1)$, a probability model $\mathcal{P}$, and the length of the string **s**. Begin by subdividing $[0, 1)$ according to the cdf, then pick the unique subinterval containing the tag, $t \in I_a$ for some $a \in \mathcal{A}$. Thus $x_0 = a$, and we may move to the next symbol, this time starting with the interval I_a found in the current step. This process (illustrated in Algorithm 3.2) is repeated until all symbols in **s** are found.

Example 3.2 To demonstrate decoding, take $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{P}$ as given in example 3.1. Furthermore, suppose that the sequence length $n = 3$ is given, and the tag $t = 0.4$. Decode the sequence following algorithm 3.2:

- $I = [0, 1)$
- subdivision: $I = [0, 0.7) \cup [0.7, 0.9) \cup [0.9, 1)$

Algorithm 3.2 Decoding a tag to a sequence $\{x_n\}$ of length N

```
input tag t
I ⇐ [0, 1)
for n = 0 to N − 1 do
  subdivide I according to cdf
  such that I = I_{a_1} ∪ I_{a_2} ∪ ··· ∪ I_{a_m}
  find j with t ∈ I_{a_j}
  x_n ⇐ a_j
  I ⇐ I_{a_j}
end for
```

- $t = 0.4 \in [0, 0.7)$ so $x_1 \Leftarrow a$, $I \Leftarrow [0, 0.7)$
- subdivision: $I = [0, 0.49) \cup [0.49, 0.63) \cup [0.63, 0.7)$
- $t = 0.4 \in [0, 0.49)$ so $x_2 \Leftarrow a$, $I \Leftarrow [0, 0.49)$
- subdivision: $I = [0, 0.343) \cup [0.343, 0.441) \cup [0.441, 0.49)$
- $t = 0.4 \in [0.343, 0.441)$ so $x_3 \Leftarrow b$, $I \Leftarrow [0.343, 0.441)$
- $n = 3$ symbols reached, so were are done and $\{x_n\} = \{a, a, b\}$ □

3.3.2 Encoding and Decoding

In this section the preceding discussion is made rigorous. Additionally, binary coding of the tag is introduced, the second part of this algorithm. Before starting, notation for the *tag*, t is required. This tag represents the string $\mathbf{s}$ as a number in $[0, 1)$, i.e. the tag is the arithmetic encoding of t.

Proposition 3.1 *Given an alphabet $\mathcal{A}$, and a probability model $\mathcal{P}$ on $\mathcal{A}$, there exists a mapping $T : \mathcal{A}^* \to [0, 1)$.* □

PROOF This section will present an algorithmic construction of one such map. ■

There is another purely practical consideration that has been ignored so far. From the discussion it should be apparent (although details remain to be shown) that given a string $\mathbf{s}$ of any finite length N, a real number $t = T(s) \in [0, 1)$ may be found that encodes it. However, nothing has been said about the nature of t. For any finite string, Algorithm 3.1 describes a method to pick a subinterval of $[0, 1)$, such that any number in this subinterval will uniquely decode to the original (length N) string. Clearly t may be constrained to be a rational number, however the number of decimal digits needed to represent t will, in general, grow with the length of the string. This is problematic for a practical implementation of this algorithm in a computer system,

and so additional considerations are made to ensure *incremental coding*. In incremental coding, the bits of the binary representation of t are encoded as symbols are read from $\mathbf{s}$; there is no need to work with the entire sequence at once. This ensures that fixed finite-precision representations in the computer may be used.

Encoding a binary representation of $T(s)$

Consider a string $\mathbf{s}$ of length N, $\mathbf{s} = \{s_n\}_0^{N-1}$. Keep track of the current subinterval by use of the following two relationships:

Notation 3.5 *Let l^n be the lower bound of the interval at step n, and r^n be the range, or length, of the subinterval at the n^{th} iteration. Hence*

$$l_n = l_{n-1} + r_{n-1}F(s_{n-1}),$$

and

$$r_n = r_{n-1}\Big(F(s_n) - F(s_{n-1})\Big). \qquad \square$$

In order to handle the incremental coding problem, three mappings are introduced. The idea is that when a binary digit is known for certain, it is coded and then the current interval is rescaled to be $[0, 1)$. This way the encoder and decoder can manage with finite precision representations. Note, however, that this puts a lower bound (dependent on the representation) on the smallest interval that can be represented. Thus the probability model must account for this. In practice this is not a difficult restriction.

Definition 3.5 *Introduce the following re-scaling maps:*

$$\begin{array}{lll} S_1 : [0, 1/2) & \to [0, 1) & S_1(x) = 2x \\ S_2 : [1/2, 3/4) & \to [0, 1) & S_2(x) = 2(x - 1/4) \\ S_3 : [1/2, 1) & \to [0, 1) & S_3(x) = 2(x - 1/2). \end{array} \qquad \square$$

As previously mentioned, incremental coding is needed in order to enable representation by fixed precision numbers. This is achieved by modifying Algorithm 3.1 slightly. When the current subinterval becomes too small, rescale it onto [0,1) and continue. Algorithm 3.3 shows the modified algorithm. The re-scaling condition chosen here is length$(I) < 1/4$. When the current interval I meets this condition, there are three cases:

1. $I \subsetneq [0, 0.5)$
2. $I \subsetneq [0.5, 1)$

3. $0.5 \in I$ and $I \nsubseteq [0, 0.5)$, $I \nsubseteq [0.5, 1)$

Cases 1 & 2 are easily handled, simply encode a '0' or '1', respectively, telling the decoder that the relevant subinterval is contained entirely in the 'top' or 'bottom' of the current interval. Case 3 is problematic, as there are only two symbols to work with — no way to tell the coder 'in the middle'. This difficulty is managed using a method introduced by Witten, Neal, and Cleary [61]. In this case, the encoder does not yet know what the next output bit is, but it *is* known what the bit *following* it will be: it must have the opposite value to that of the next bit. To illustrate, suppose $I_{x_n} \subsetneq [1/4, 3/4)$ and $I_{x_{n+1}} \subsetneq [1/4, 1/2)$. That is, the next subinterval is completely contained in $[0, 1/2)$. Thus we send '0' and map $[0, 1/2)$ onto $[0, 1)$, the mapping of $I_{x_{n+1}}$ must be contained in $[1/2, 1)$, as its pre-image is contained in $[1/4, 1/2)$.

In this manner, the algorithm keeps track of the fact that the current interval is straddling 0.5, and rescales the interval as before. This procedure may be repeated any number of times, thereby maintaining the current interval length $\geq 1/4$, and keep a count of the number of *follow-bits* needed. When the current interval is completely contained[2] in either the 'top' or 'bottom', send the appropriate bit and follow it with the opposite value (or more than one bit of the opposite value, if there have been multiple re-scalings straddling 0.5).

Algorithm 3.3 describes the process of encoding the string $\mathbf{s}$, resulting in a binary representation of a number lying in the interval $[l_N, l_N + r_N)$ (i.e., the final subinterval in the iterative process). The notation $\lfloor x \rfloor$ denotes the largest integer smaller than or equal to x.

Decoding a binary representation of $T(\mathbf{s})$

Similarly, Algorithm 3.4 details decoding the tag t, resulting in the original string $\mathbf{s}$. These iterations essentially mirror the encoding iterations. In setting the algorithm up, it is necessary to read in a certain number (k) of bits of input in order to uniquely determine the first interval. After this, bits are read in as needed during the iterations.

3.3.3 Adaptive Models

So far in this section the probability model $\mathcal{P}$ actually associated with a particular source has been discussed; these probabilities reflect the underlying distribution of the source. In practice, however, this probability model is generally not known.

There are several possible approaches to addressing this issue. One obvious possibility is to use a *two pass* approach. The first pass generates statistics, and the second pass uses those statistics to encode with. Of course, the model must be transmitted to the decoder, at a cost relative to the size of the alphabet. This *static* approach cannot be used for real-time applications, for obvious reasons.

[2] Of course, this allows for pathological inputs; in practice this is not a problem.

Algorithm 3.3 Encoding a sequence $\mathbf{s} = \{s_n\}_1^N$

$s_2 \Leftarrow 0$ {counts the number of S_2 scalings performed}
$l_0 \Leftarrow 0$ {holds lower bound of initial interval}
$r_0 \Leftarrow 1$ {holds length of initial interval}
for $n = 1$ to N **do**
  {map to the new subinterval}
  $l_n \Leftarrow l_{n-1} + r_{n-1}F(s_{n-1})$
  $r_n \Leftarrow r_{n-1}(F(s_n) - F(s_{n-1}))$
  while $r_n \leq 1/2$ **do**
    {apply re-scaling, and code leading bits if possible}
    if $l_n \geq 1/2$ **then**
      $l_n \Leftarrow S_3(l_n)$, $r_n \Leftarrow S_3(r_n)$
      code(1) {we are in $[1/2, 1]$ so leading bit is 1}
      while $s_2 \neq 0$ **do**
        code(0) {code any S_2 maps that were done}
        $s_2 \Leftarrow s_2 - 1$
      end while
    else if $l_n + r_n \leq 1/2$ **then**
      $l_n \Leftarrow S_1(l_n)$, $r_n \Leftarrow S_1(r_n)$
      code(0) {we are in $[0, 1/2]$ so leading bit is 1}
      while $s_2 \neq 0$ **do**
        code(1) {code any S_2 maps that were done}
        $s_2 \Leftarrow s_2 - 1$
      end while
    else
      $s_2 \Leftarrow s_2 + 1$
      $l_n \Leftarrow S_2(l_n)$, $r_n \Leftarrow S_2(r_n)$
    end if
  end while
end for
$t = l_n + \frac{r_n}{2}$ {take the midpoint of interval for tag value}

Algorithm 3.4 Decoding a string $\mathbf{s} = \{s_n\}_{n=1}^{N}$

find k such that $2^{-k} < \text{argmin}_{a_i} P(a_i)$
read first k bits of input into t
$l_0 \Leftarrow 0$ {holds lower bound of initial interval}
$r_0 \Leftarrow 1$ {holds length of initial interval}
for $n = 0$ to $N - 1$ **do**
  find i such that $F(a_{i-1}) \leq \lfloor \frac{(t-l_n+1)F[0]-1}{r_n} \rfloor \leq F(a_i)$
  $x_n = a_i$
  {map to the new subinterval}
  $l \Leftarrow l + rF(a_{i-1})$
  $r \Leftarrow r(F(a_i) - F(a_{i-1}))$
  while $r_n \leq 1/2$ **do**
    if $l_n \geq 1/2$ **then**
      $l \Leftarrow S_3(l_n)$, $r \Leftarrow S_3(r_n)$, $t \Leftarrow S_3(t)$
    else if $l_n + r_n \leq 1/2$ **then**
      $l \Leftarrow S_1(l_n)$, $r \Leftarrow S_1(r_n)$, $t \Leftarrow S_1(t)$
    else
      $l \Leftarrow S_2(l_n)$, $r \Leftarrow S_2(r_n)$, $t \Leftarrow S_2(t)$
    end if
  end while
  $t \Leftarrow t + \text{input}(bit)$
end for

A review of the coding process will reveal that the method does not require a static probability model. In fact, as long as the encoder and decoder agree on the changes, a different model (and/or alphabet) can be used for each iteration of the algorithm.

A variation on the static model approach is the *semi-adaptive*, or *decrementing* code. In this case the signal statistics are calculated ahead of time, just as in the static approach. However, as each symbol is encoded (or decoded) the probabilities are adjusted (i.e., the weight of this symbol is decremented) so that at each symbol, the model holds the best estimates for the probability model at that step.

In some cases, the statistics from a large set of similar samples may be taken as a starting point. For example, if encoding English text, we could generate statistics from a few thousand books, and be reasonably confident in our model.

On the other hand, there may be no (or insufficient) *a priori* knowledge to resort to. In this case, the strategy is to have an initial probability model (which could be generated from *a priori* knowledge, or uniform), and then as each symbol is coded, the model will *adapt*, using the current symbol as input to an update algorithm.

The strength of this approach is that no extra information needs to be transmitted to the decoder — so long as the decoder and encoder use the same update algorithm, their models will proceed in lock-step. It can be shown [9] that performance of the adaptive and semi-adaptive methods are equal in the case that the semi-adaptive method must send the original statistics.

3.3.4 Context Based Coding

At this point it is worth noting that for our purposes — that of compression — one of our primary goals will be to encode information with the least number of bits. Shannon's theorem gives us bounds on the performance expected; this inequality (Equation 3.1) is arrived at from the actual probability distribution for the source.

In general (in most practical applications), the true probability distribution is not known. Clearly a better approximation to the true distribution can improve the encoding rate. One way to improve upon the adaptive model discussed above is to forgo using a single model for the encoding, but rather to use some kind of local context to decide which model to use. In practice each of these several models may (separately) be adaptive or not, whichever is most efficient. Of course, any context that the encoder uses to decide which model to use must also be available to the decoder. This introduces a causal constraint on the data stream, as contexts must only depend on symbols already encoded.

This approach allows the application of *a priori* knowledge about the stream, on a local scale. For example, consider the encoding of English text. One could take for context the previous three characters seen. Then on input of "...tha", the model for previously seen symbols "tha" would be chosen[3].

[3]Which presumably would have a very high probability of symbol 't', and likewise small probability of 'z'.

Note, however, that the proliferation of models has consequences. If the above example is treating American Standard Code for Information Interchange (ASCII) text, then eight[4] bits are required for each character. Since each combination of characters is possible, this needs $2^{3\times 8} = 2^{24} = 16777216$ models. If each model is using 32 bit integers to count frequencies for each symbol, then at least 16777216 models$\times$4 bytes/integer$\times$256 integers/model = 17179869184 bytes (or 16384 Megabytes) of storage is required. This method quickly becomes impractical!

However, with appropriate constraints on the number of models needed, this method can be very effective. In the applications to follow, only the binary alphabet is used, which makes both the models' size and the number of combinations manageable. Up to 16 separate symbols/states in a context are quite reasonable in this case.

3.4 Performance of Arithmetic Coding

As previously mentioned, in practice, only an approximation of the true probability distribution of the source is known. Clearly, it is important to establish performance of the algorithm under these conditions.

Consider a binary alphabet and i.i.d. source, then the entropy (i.e., the optimal coding rate) is given by

$$H = -\sum_{i=1}^{m} P(x_i{=}a_i) \lg P(x_i{=}a_i).$$

For any other probability model,

$$Q = \{Q(a_1), Q(a_2), \ldots, Q(a_m)\} = \{q_1, q_2, \ldots, q_m\}$$

the average code length, L, will be

$$L = -\sum_{i=1}^{m} Q(x_i{=}a_i) \lg Q(x_i{=}a_i).$$

Thus the error introduced by modelling P with Q will simply be $E = H - L$.

Proposition 3.2 *The modelling error using binary arithmetic coding is given by*

$$E \sim \sum_{i=1}^{m} \left[\frac{1}{2 \ln 2} \frac{d_i^2}{p_i} + O(\frac{d_i^3}{p_i^2}) \right],$$

where $p_i = P(x_i = a_i)$ *and* $d_i = P(x_i = a_i) - Q(x_i = a_i)$. □

PROOF This result is a corrected [31] version of one given in [9]. ■

From the above, since there is no linear term, it follows that any reasonable approximation to the true distribution P will achieve performance near entropy.

[4] Assuming high ASCII characters are required, but reduction to seven bits doesn't help much!

Remarks

This chapter introduced the idea of entropy coding, and detailed the arithmetic coder, a particular method of doing so. The arithmetic coding approach will be used to encode coefficient approximations in the image compression algorithms developed in §4.3.3 and §4.4.

Chapter 4

Image Compression

The subject of data compression has many facets. In this chapter, the discussion is limited to areas directly relevant to the compression of digital images, and particular algorithms. A more comprehensive introduction to data compression (including many subjects not discussed here) may be found in [48]. Following a brief background, specific algorithms in the areas of (spatial) IFS, wavelet, and hybrid fractal-wavelet are discussed. In addition to the discussion of several basic and/or well known algorithms, two novel algorithms are proposed (one wavelet, one hybrid fractal-wavelet). The chapter is concluded with a discussion of further investigations into colour images and compression of three-dimensional images.

4.1 Overview

4.1.1 Images

The question "What is an Image?" turns out to be rather difficult[1] to answer. In this work it is sufficient to use a rather broad definition (which will include lots of 'images' that are not of interest, as well as those that are) of a digital image. The following definitions are entirely motivated by the storage of digital images in a computer system.

Definition 4.1 (Pixel) *A pixel is a positive integer fitting in a binary representation of n bits. Thus a pixel may take on values from $0 \ldots 2^{n-1}$.* □

Definition 4.2 (Image Channel) *Define an image channel to be a finite, two- or three-dimensional array of* pixels *of a like number of bits.* □

[1] I.e., an open research problem.

Definition 4.3 (Image) *An Image consists of a collection of one or more Image Channels. In the case of a single channel, the images are usually called* greyscale. Multi-channel *images may take many forms, from RGB (Red, Green, Blue) 3-channel colour images to N-channel satellite images.* □

For most of the remainder of this presentation, the existence of multi-channel images will be ignored; greyscale images are assumed. For algorithmic reasons, images with dyadic dimensions are more convenient; in other words they take the form of arrays of $2^N \times 2^M$. Furthermore, the restriction to square images ($2^N \times 2^N$) will often be made, as it simplifies certain implementations. In this setting, the image can be considered as regular samples from $[0, 1]^2$.

4.1.2 Compression

Broadly stated, data compression is the process of representing information in a (more) compact form. Given a signal x the compression algorithm represents x with a new signal $\bar{x}$, where the length (i.e., storage space) of $\bar{x}$ is less than that of x. There are two general classes of compression algorithms.

Lossless Compression

A compression algorithm is called *lossless* when it preserves all of the information in the original signal. In other words, the original signal may be recovered perfectly from the compressed signal.

Lossy Compression

When the original signal is not recoverable from the compressed signal, information has been lost. Hence, such algorithms are termed *lossy*. In some sense, all lossy algorithms may be represented as lossless algorithms that do not transmit the entire compressed signal. In this way, much of the underlying theory performed for lossless compression may be extended to lossy algorithms. On the other hand, looking at lossy compression this way can be misleading. In a lossy algorithm the choice of what information is transmitted and what is not is fundamental to the success of the algorithm. The compression schemes discussed in this thesis are all lossy.

4.1.3 Modelling and Coding

Conceptually, if not always in practice, image compression can be separated into two processes. In the *modelling* phase, the algorithm organizes and analyses the signal — the purpose is to provide a model that will represent redundancies in the signal. In

this context, redundancies are exactly what we wish to remove by the act of compression. Any information that can be determined from our current state should not be encoded by the algorithm.

After modelling has been performed, the *coding* phase begins. The *encoder* transmits (e.g., writes to a file for later decoding) the model. After this is accomplished, the decoder will be able to use the model to predict the values in our signal. Thus the remainder of the encoder's task is to transmit in some manner the differences between the predictions of the model and actual data.

The *decoder* receives the modelling information and then proceeds to adjust the results, based on the difference information received from the coder. Taken together, the two sides of this procedure are often referred to as a *coder*, or *codec* (from CODer-DECoder).

4.1.4 Quantization

There are many different methods of quantization, of which only two are applied in this work. Introductory discussion of quantization and its place in lossy compression may be found in [48, 52].

Definition 4.4 (Quantization) *Let X be a (possibly continuous) set, and K be a discrete set. Let Q and D be mappings $Q : X \to K$ and $D : K \to X$. Q and D are such that*

$$\|x - D(Q(x))\| \leq \|x - D(d)\| \qquad \forall\, d \in K$$

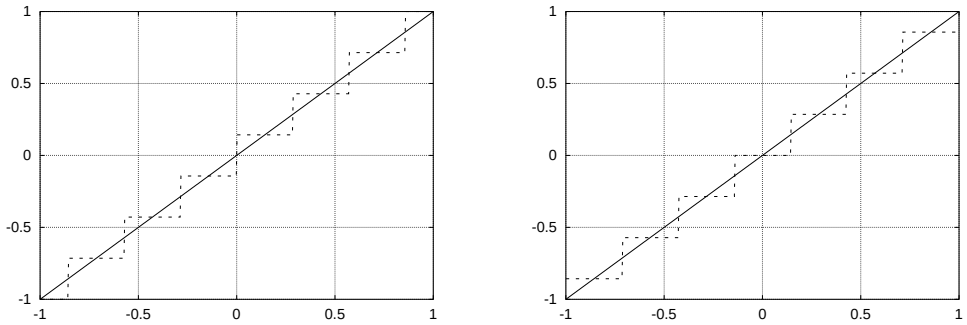
*Applying Q to some $x \in X$ is called **quantization**, and $Q(x)$ is the quantized value of x. Likewise, applying D to some $k \in K$ is called **dequantization** and $D(k)$ is the dequantized value of k.* □

In applications, X is often a continuous space such as $\mathbb{R}$, and K a small set of integers (e.g. $0 \ldots 2^{N-1}$ for N on the order of 10). Clearly Q is not invertable. In some sense, D is an "approximate inverse" of Q. However, the approximation can be defined in many ways.

Figure 4.1 illustrates two ways to uniformly quantize $[-1, 1]$ into values capable of fitting in 3 bits (there are $2^3 = 8$ different levels available). The choice of which method to use is important; if the value "0" must be realizable by the quantized values, only the midstride type will do. It should also be clear that these mappings are non-linear[2]. With specific knowledge about the problem domain, it is possible to do a much better job (i.e., smaller error) than with a uniform quantizer.

One other quantizer will be discussed, along with its use, in §4.3.3. Before going further, a brief example tying several of these ideas together is worthwhile.

[2]Consider $Q(0.4) + Q(0.4) \neq Q(0.8)$ in Figure 4.1-a.



(a) mid-rise (b) mid-stride

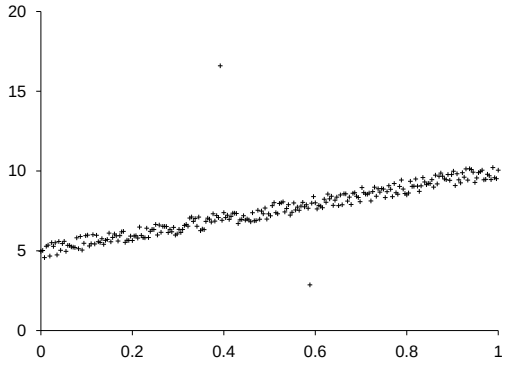
Figure 4.1: Uniform quantizers: (a) zero is in "mid-rise" (b) zero is in "mid-stride"

Example 4.1 Consider a data set that is roughly linear, with a couple of outlying points (Figure 4.2-a). If the data is quantized with a simple nearest integer quantizer[3], the result is Figure 4.2-b. The error introduced by this process is shown in Figure 4.2-c. This error is non-recoverable. The result of dequantizing the data is 4.2-b, not 4.2-a. The benefit of this operation is that there are now only five distinct values to represent. Now an obvious model for this data set is a line. Figure 4.2-d shows such a model, and Figure 4.2-e shows the quantized version of this model. The final figure, Figure 4.2-f shows the *error that the model makes in predicting the quantized data*. This is the error between the predicted Figure 4.2-e and the data Figure 4.2-b, which brings us to the coding step. The job of the encoder is to transmit the model parameters and Figure 4.2-f. All of the zero values (of Figure 4.2-f) are values that do not have to be encoded, as the model predicts them accurately. All of the non-zero values represent (quantized) predictive errors, and the coder must send these deltas. There are a large number of ways to do this, but it should be clear that under many reasonable schemes, the amount of information to be transmitted has been greatly reduced.

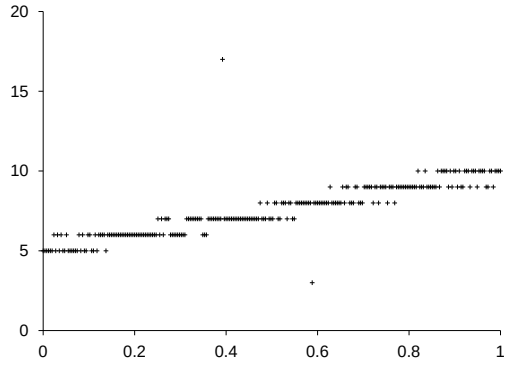
4.1.5 Measuring Performance

Once a compression algorithm is devised, it is necessary to evaluate its performance. There are two primary quantities of interest, the *amount* of compression, and the *quality* of the result. Other quantities, such as execution time or memory use, may also be relevant in comparing particular algorithms.

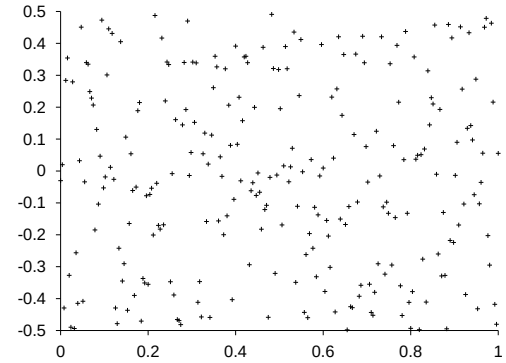
[3]This is clearly suboptimal if it is known that the data is likely to be clustered around a line as shown.



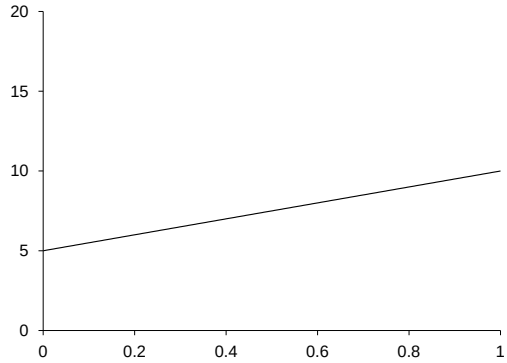
(a) original data set



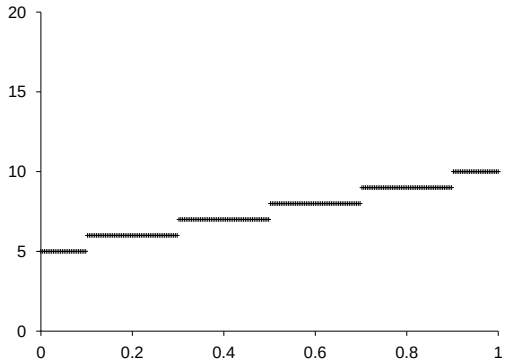
(b) quantized data set



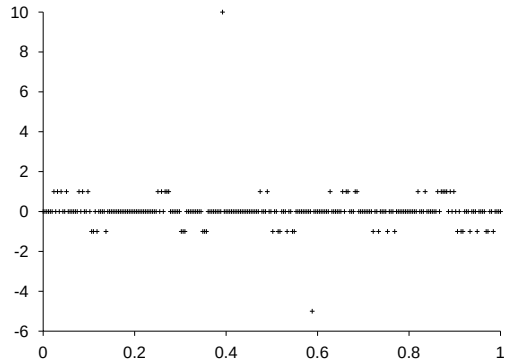
(c) quantization error



(d) model: $5x + 5$



(e) quantized model



(f) modelling error

Figure 4.2: Example of modelling and coding (a) shows data set (b) after quantization (c) irrecoverable error introduced by quantization with a uniform (integer) quantizer (d) the linear model used (e) model after quantizing with the same uniform quantizer (f) the modelling error: the coder will transmit deltas for all non-zero values

It is easy to quantify the amount of compression. Two common measurements are the *compression ratio* and the *rate*, or *bitrate*. The compression ratio measures the ratio of the original data size to the compressed data size and is often stated in the form "original:compressed". For example, if a 256×256 8-bit grey level image (65536 bytes) is reduced to 650 bytes, then it has been compressed at about 100:1. The rate, on the other hand, is a measure of the average number of bits per pixel. The hypothetical compressed file above has $650 \times 8 = 5200$ bits and $256^2 = 65536$ pixels, so the rate is about 0.08 bpp.

The "quality" of compression is much harder to quantify. The result of any lossy compression scheme is an approximation to the original signal. What is required is a measure of the inaccuracy of this approximation. Quantification is complicated by the fact that in practice what is actually important is the *perceived* inaccuracy (to a human observer) of the approximation. Since the human visual system is itself a lossy and non-linear process, this becomes difficult to model. Notwithstanding these difficulties, there are several approaches; the error introduced by the approximation is termed the *distortion*.

One obvious approach is to model the image within a metric space (usually $\mathcal{L}^2([0,1])$) and use the distance function to define the error. The usual approach is arrived at from the $\ell^2(\mathbb{N})$ distance and often termed the *mean-squared error*

Definition 4.5 (MSE) *Define the Mean Squared Error (MSE) between sequences (and similarly for images)* $\{x_n\}_{n=1}^N$ *and* $\{y_n\}_{n=1}^N$ *as*

$$\sigma^2 = \frac{1}{N} \sum_{n=1}^{N} (x_n - y_n)^2 \tag{4.1}$$

□

The resulting values of σ^2 are somewhat inconvenient to compare. In order to address this problem, introduce the following definition.

Definition 4.6 (PSNR) *The Peak Signal to Noise Ratio (PSNR) is defined in terms of the MSE. If the peak value*[4] *obtainable by* x_n *is* $\bar{x}$ *then define the PSNR as*

$$PSNR = 10 \log_{10} \left(\frac{\bar{x}^2}{\sigma^2} \right) \text{ in decibels [dB]} \tag{4.2}$$

□

The PSNR will, in practice, have values in the range of about 10 to 50 dB (with the exception that the PSNR of an image with respect to itself is infinite). A value of less than 10 dB would suggest that the images are unrelated, whilst 50 dB or above is often imperceptible to the eye.

The above method is the *de-facto* standard. It has obvious flaws[5]. In practice, it is fairly effective in that a high PSNR is usually sufficient to ensure small (in the human

[4] For example, 8-bit pixels range from 0 to 255, so the peak value is 255.

[5] Consider an image shifted in any direction by one pixel.

observer sense) distortion. On the other hand, any algorithm that can induce small translations of image features will fare poorly in PSNR, but not necessarily for an observer. The problem of improved distortion measurements is quite difficult. For that reason this work uses PSNR exclusively. Discussion of applying information about the human visual system to distortion metrics may be found in [45, 13]. There has been some work done in application of such metrics to image compression, for example [42].

To compare performance under these measurements of rate and distortion it is convenient to produce *rate/distortion* curves. For example, Figure 4.3 shows the results of several different coders applied to the lena image, which is a 512×512 8-bit

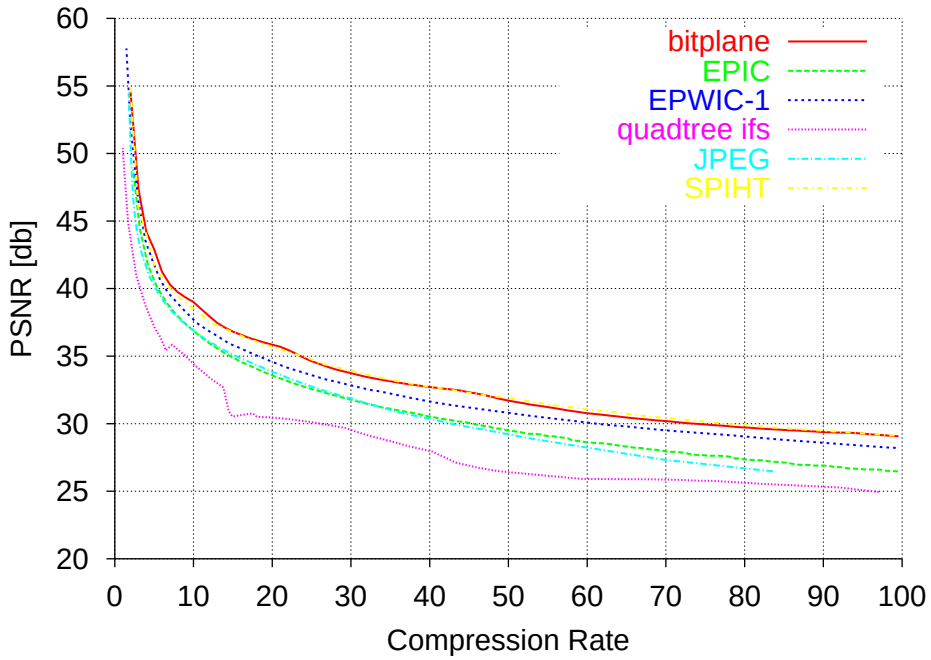


Figure 4.3: Comparison of algorithms applied to the lena image

greyscale image. Several images such as lena are used throughout the following sections; the original images are included as Appendix A in order to avoid duplication. In addition to demonstrating the rate-distortion curve, this plot gives a rough idea of the relative performance of various algorithms. All of the coders included in this plot will be discussed later. They include "bitplane": discussed in §4.3.3, EPIC: an early wavelet coder [1], EPWIC-1: a more sophisticated statistical approach to wavelet coding [12], SPIHT: a popular modern wavelet coder, "quadtree ifs" a (very poorly optimized) pure fractal coder of the type discussed in §4.2, and finally the Joint Photographic Experts Group (JPEG) industry standard coder.

Armed with this terminology, we will now present and contrast some methods of

image compression.

4.2 IFS Codecs

It is possible to use the methods of §2.7 directly in the spatial domain of an image to perform image compression. A brief outline of the method is provided here. For more a detailed introduction, see [23]. Recall the inverse problem described in §2.7, and the collage theorem proposed as a method to address it. By finding an IFSM map that maps the image close to itself, we ensure that the fixed point of that map is close to the original image. To this end, a least-squares fit of all domain blocks is performed for each range block. Once the best maps are found, the list of maps is encoded. So long as this encoding of the maps takes less storage than the original image, compression has been achieved. Since the fixed point of the IFSM maps is an approximation of the original image, this method will always be lossy.

One of the recurring problems in image compression is that complicated schemes require a lot of information to be sent to the decoder in order for the decoder to decide what to do next. This information takes space in the bitstream that cannot be used for image information. The trade-offs are complicated and non-obvious. As a starting point, consider a very simple algorithm that does not require much of this *side band* information.

A simple IFS compression algorithm

Begin with a greyscale image of dimensions $2^N \times 2^N$ (for this example,512×512). Our task is to construct an IFSM that approximates this image. Partition the image into small, say 8×8 pixel, blocks. These will be the *range blocks*, and specifying a range block can be done by specifying the lower left corner, for example. In this case, there are 64×64 range blocks, so 12 bits (6 each for row and column) are needed to specify all range blocks. This results in a set of range blocks $\{R_n\}_{n=1}^{4096}$.

Having determined the range blocks, domains are required to map onto these ranges in an IFSM manner. In order to ensure[6] contractivity, and to remain simple, pick a pool of 16×16 pixel *domain blocks*. This could be, similar to the ranges, a 16×16 block partitioning of the image, or any of numerous other schemes. Assuming the former scheme, the set of domain blocks will be $\{D_m\}_{m=1}^{1024}$.

For the grey level maps, consider affine maps as in §2.7. For each range block R_n, the task is to find the best domain and mapping $\phi_{R_n} : D_m \to R_n \quad \phi_{R_n}(x) = \alpha_n x + \beta_n$. In order to do this, try every combination of domain and range blocks, keeping the best mapping for each R_n. Having found the best mapping for each range block, store the parameters of the map (and the domain block it maps) and send those to the decoder, as shown in Algorithm 4.1.

[6] There must be constraints on the grey level maps as well, of course.

Algorithm 4.1 IFS encoding of greyscale image G

$G \Leftarrow$ greyscale image
$\{R_n\} \Leftarrow$ range blocks
$\{D_m\} \Leftarrow$ domain blocks
for each range block R_n **do**
  for each domain block D_m **do**
    fit $\phi_{D_m}(x) = \alpha_m x + \beta_m$
    measure error $E_n = \|R_n - \phi_{D_m}(R_n)\|$
    if $E_n < E_{\text{best}}$ **then**
      {keep track of the current best map}
      $\phi_{\text{best}} \Leftarrow \phi_{D_{\text{best}}}$, $D_{\text{best}} \Leftarrow D_m$
    end if
  end for
  store $\phi_{R_n} : D_{R_n} \rightarrow R_n$ $\phi_{R_n} \Leftarrow \phi_{\text{best}}$, $D_{R_n} \Leftarrow D_{\text{best}}$
end for
send to decoder list of ϕ_{R_n}

Having performed the above procedure for each range, the result is a list of the best α_n, β_n and D_n associated with each range R_n. This list completely defines the IFSM. Thus encoding is simply a matter of transmitting this list to the decoder; an entropy coding method may first be applied to further increase the rate[23].

Decoding

The decoder reads in the list of values, and creates maps for each range block which is, in fact an IFSM. Starting with any initial image, the decoder iterates the maps until they converge. The resulting image is the IFSM approximation to the original image. In practice, convergence does not take long, on the order of 10 iterations [23]. Algorithm 4.2 illustrates the decoding process.

Of course, the smaller the range blocks and the larger the domain pool, the better the result is likely to be. More range blocks require more information to transmit, so the compression rate is reduced. More domain blocks requires more (potentially very much more) time to process. Another problem with this simple method is blocking artifacts (see Figure 4.4), which can be greatly improved upon[43].

The method described above is quite simple and does not perform very well. Many refinements of all parts of the technique have been tried, see for example [30, 47]. Figure 4.4 shows some examples of IFSM image coding, and includes one such improvement. In Figure 4.4-d, a *quadtree* approach is used — essentially the method is this: if a range block R_n does not have a "good" fit, it is equally subdivided into four subblocks and these new range blocks are used.

The quadtree approach is one of the simplest refinements that may be made; far

Algorithm 4.2 IFS decoding of greyscale image G

$\bar{G} \Leftarrow$ any initial greyscale image
$\{R_n\} \Leftarrow$ range blocks
$\{D_m\} \Leftarrow$ domain blocks
$\{\phi_{R_n} : D_{R_n} \rightarrow R_n\} \Leftarrow \{$read from encoder$\}$
for $i = 1$ to number of iterations **do**
  for each domain block R_n **do**
    apply map $R_n = \phi_n(D_{R_n})$
  end for
end for
$\bar{G}$ now contains IFS approximation of G

more sophisticated methods are still practical[30]. Although for some applications and image classes spatial domain IFSM methods perform quite well, in general they have under performed wavelet based compression algorithms, at least in the PSNR sense. Overall, the performance has lagged behind several popular wavelet algorithms, and encoding times can be quite long[7].

[7]However, decoding is comparatively very fast. IFSM methods are quite asymmetric in this regard, as all the decoder must do is iterate the maps a few times, while the encoder has to find them.

(a) original image

(b) 16×16 range blocks

(c) 8×8 range blocks

(d) quadtree range blocks

Figure 4.4: Examples of IFSM compression on the goldhill image (a) original goldhill image (b) 16×16 range blocks and 32×32 domain blocks with rate of 60:1 and PSNR of 25.13 (c) 8×8 range blocks and 16×16 domain blocks with rate of 60:1 and PSNR of 29.107(d) a quadtree scheme where range blocks vary from 4×4 to 64×64 in size (and domain blocks are always one power of 2 larger) with rate of 5:1 and PSNR of 34.088

4.3 Wavelet Codecs

There are many techniques for image compression using wavelet coefficients (see for examples and discussion [52, 48]). Early coders based on a "pyramid method" [1] and later improvements [12] brought wavelet methods to the fore-front of compression techniques. The most successful family of coders to date are based on the Embedded Zerotree Coder (EZW) method introduced by Shapiro [50]. Perhaps the best known general purpose codec descended from EZW is the Set Partitioning In Hierarchical Trees (SPIHT) introduced by Said and Pearlman [46]. Although there are more recent general purpose codecs with slightly better performance than SPIHT (including the commercially significant JPEG2000), SPIHT has become something of a benchmark. In this thesis, comparisons with SPIHT are made for that reason.

Zerotree Coders

At a high level, many wavelet coders operate on the same basic principles. After the wavelet transform has been performed, most of a signal's energy is present in relatively few coefficients. In compressing the signal, the idea is to encode (or approximate, then encode) these coefficients and ignore the rest. One approach is to apply a magnitude threshold: every coefficient smaller than some value is considered to be zero. The problem now is to identify and encode the small number of non-zero coefficients. Shapiro [50] tackled this problem by identifying *zerotrees*, that is, subtrees rooted at some position in the coefficient tree where the root and all children are zero (hence the name). In a sparsely populated coefficient tree, identifying these zerotrees (so that the decoder can ignore them) allows efficient identification of the non-zero coefficients. Of course this process can be repeated for several thresholds, allowing progressive improvement in the approximation. Implementation details may be found in [50, 46]. A key feature of these approaches is that they are *embedded* coders: the decoder does not need all of the output of the encoder in order to decode, rather, it may decode bits as they are read from the encoder. This aspect makes them more flexible than non-embedded coders, such as the IFS coders previously described.

4.3.1 Choice of Wavelet

For a given compression algorithm, the choice of wavelet used can make a significant difference in performance. In the following, comparison of a few filters is made to demonstrate the advantages of the choice of filter used in later numerical results.

The Haar and Daubechies 8 filters have been mentioned earlier. The Antonini 9/7 filter [3] has become nearly ubiquitous for compression with biorthogonal wavelets. It represents a good trade-off between filter length (and thus run-time of the wavelet transform) and PSNR; it also tends to have visually pleasing smoothing of quantization error (see §4.3.2). After this section, all algorithms use the Antonini 9/7 filter

unless otherwise specified.

A quick comparison of results (in all cases using the algorithm described in §4.3.3) demonstrates the effects of the choice of wavelet basis on compression. Figure 4.5 shows the result of using the (above) three bases with exactly the same compression rate (100:1). In comparison, Figure 4.6 shows the same process, but with the rates of the Haar and Daubechies versions varied until the PSNR of the resulting image was the same as that achieved by the Antonini 9/7 filter at a rate of 100:1. Although harder to see, this case shows that even with the same PSNR, the artifacts in the Antonini 9/7 image seem less noticeable.

In both figures, the blocking artifacts in the Haar (due to the piecewise-linear components) are readily apparent. The Daubechies 8 (picked because it is an orthogonal filter with similar length to that of the Antonini 9/7) looks much better but still under performs in comparison to the biorthogonal filters.

(a) original image

(b) Haar

(c) Daubechies 8

(d) Antonini 9/7

Figure 4.5: Comparison of Haar, Daubechies 8, and Antonini 9/7 bases at a compression rate of 100:1 (a) original image (b) Haar with PSNR of 24.916 (c) Daubechies 8 with PSNR of 25.495 (d) Antonini 9/7 with PSNR of 26.425

(a) original image

(b) Haar

(c) Daubechies 8

(d) Antonini 9/7

Figure 4.6: Comparison of Haar, Daubechies 8, and Antonini 9/7 bases at a PSNR of 26.425 (a) original image (b) Haar with rate of 64.5:1 (c) Daubechies 8 with rate of 77.5:1 (d) Antonini 9/7 with rate of 100:1

4.3.2 Biorthogonal Filter Direction

In §1.4.1 it was mentioned that although the analysis and synthesis filters in a biorthogonal wavelet basis may be exchanged, the resulting coefficients are not the same. Recall that if one filter has more vanishing moments, the other will tend to be smoother. If the filter with more vanishing moments is used as the analysis filter, the result will be smaller coefficients in areas where the signal is regular [18, 37]. Additionally, the error introduced by the compressed approximation will be smoothed (thus looking better). If, on the other hand, the filters are reversed, then both of these visually beneficial effects are lost. In fact, the choice of filter may exaggerate the errors. Figure 4.7 shows this effect on the "lena" image.

While this effect has been discussed in the literature [3, 18], it does not seem to be well known. Of course, any in-depth investigation of 'visually optimal' bases would involve a great deal of subjective measurement. The difficulty of doing this in a meaningful way may have contributed to the current acceptance of certain bases (such as the Antonini 9/7) as 'standard' and leaving it at that.

(a) original image

(b) 7/9

(c) 9/7 matched rate

(d) 9/7 matched PSNR

Figure 4.7: Comparison of Antonini 9/7 filter with Antonini 7/9 (a) original image (b) 7/9 filter at rate 143:1, psnr 26.495 (c) 9/7 at rate 143:1, psnr 27.497 (d) 9/7 at rate 198:1, psnr 26.492

4.3.3 Proposed Embedded "Bitplane" Coder

The EZW and SPIHT coders keep quite a bit of state in the *zerotree* structures and *significance sets* [50, 46]. In effect, this constitutes a model for the data that is updated during the coding process. One interesting side-effect of this approach is that entropy coding (with an adaptive arithmetic coder) the bitstream after the SPIHT algorithm resulted in only a minor gain [46]. It is natural to ask if this is due to the algorithm having assumed some of the role of the entropy coder.

In this section, a coder is presented that was designed to separate the tasks of coding and modelling more explicitly, yet achieve similar performance. The basic design is that of a *bitplane coder*. This is in effect a multiple-threshold approach: a number of thresholds, or planes, are defined and the coder traverses the image plane by plane. In this coder, each plane is a (negative) power of two, and as many planes are used as are needed $\{0.5, 0.25, 0.125, \ldots\}$. This scheme allows for a *successive approximation* quantizer.

Quantization

In order to facilitate embedded coding, the quantizer does not quantize each coefficient only once. Instead, successively improved approximations to a coefficient are made. The general idea is as follows. Consider the approximation of some $t \in [0, 1]$; first ask "is t larger than $1/2$?". If it is, first approximate it as $\bar{t}_1 = 3/4$; on the other hand if it is smaller approximate it as $\bar{t}_1 = 1/4$. Now to improve the approximation consider the error $t - \bar{t}_1$. By construction, this must be less (in magnitude) than $1/2$. So now ask "is it larger than $1/4$?". If so improve the approximation by adding (or subtracting, as appropriate) $5/8$. Otherwise add/subtract $1/8$. In this manner the second approximation, $\bar{t}_2$, is made. Of course, this process can be carried out indefinitely. The error at each step is bounded, by construction, by $|t - \bar{t}_n| \leq 2^{-n}$. Hence the approximation converges quite quickly to t. A slightly modified version of this approach is used in the coder.

The problem with using the above scheme as written is that each adjustment requires one bit to signify above/below threshold, and another bit to specify the sign of the adjustment. It turns out that in a rate/distortion sense it is better to spend fewer bits on an inferior approximation. The modified scheme works as follows.

Suppose, as is the case in this coder, $x \in [-1, 1]$ is the value to be approximated. Instead of a single approximation by application of the uniform quantizer, construct a sequence of successive approximations $\{\bar{x}_n\}$. The encoding of this sequence is also important, as it motivates the design of the quantizer.

Begin with threshold $t_0 = 1/2$, and at each pass update the threshold as $t_{n+1} = \frac{t_n}{2}$. While $|x| < t_n$ consider x to be *insignificant*, set $\bar{x}_n = 0$, and encode this with a single bit set to "0". Now if $x \neq 0$, for some N the equation $|x| > t_N$ holds, and x is now considered *significant*. When x becomes significant, signify this with a single

bit set to "1", and a second bit specifying the sign of x. In this way, the first non-zero approximation to x is $\bar{x}_N = \text{sign}(x)\frac{3t_N}{2}$.

For every pass, the goal is to reduce the error of the approximation $e_n = |x - \bar{x}_n|$. Until the coefficient becomes significant, this error is simply x itself; after the threshold drops below x, the error is updated at each pass. At this point e_n could just be treated as x was previously, by encoding "0" bits until the error itself becomes significant, then sending two bits to signify significance and sign.

In the wavelet coefficient trees, the (in)significance of a coefficient relative to a particular bitplane tends to be more important than the accuracy of approximation. It turns out that from a rate/distortion point of view it is better to spend fewer bits on a less accurate approximation. For all planes $n > N$ merely adjust the error by $t_n/2$ in either the positive or negative direction, depending on the sign of the error. That is, $\bar{x}_{n+1} = \bar{x}_n + \text{sign}(x - \bar{x}_n)\frac{t_n}{2}$. Encoding this update requires a single bit to specify the sign of the correction. Clearly this will sometimes make the approximation worse. However, the error is still bounded by $e_n < 2^{-n}$ while fewer bits are spent to encode the approximation.

This quantization method allows embedded coding. The coder keeps decreasing the threshold level t_n and improving the approximation of the wavelet coefficients until the target coding rate is reached.

Encoding

The encoding process is illustrated in Algorithm 4.3. At each pass, or bitplane, every coefficient in the wavelet quadtree is quantized by the aforementioned method. The coefficients are processed in a descending, breadth-first manner on the coefficient tree (that is, level by level across all subbands in the wavelet coefficient tree, with lower levels first).

The quantization results are encoded by a context based arithmetic coder using binary models. Only the (in)significance of a coefficient is effectively modelled, since the sign and approximation improvements (i.e., the lower order bits) of a coefficient are highly de-correlated. For this reason, when encoding sign bits or improved approximations, a fixed $(0.5, 0.5)$ model is used. The choosing of contexts represents (yet another) trade-off; more contexts ought to provide better modelling, but each additional context doubles the number of models. Every one of these models requires enough input data to converge to a good approximation of the underlying statistics. Clearly too many models will decrease performance of the algorithms. The contexts used in this work were arrived at empirically; it is important to note that no claim of optimality can be made. Analysis of correlation structures in typical wavelet trees, decay characteristics, and convergence properties of the model could point the way to much more effective modelling.

When encoding the significance of coefficients, two sources of context are used. Four bits of *local context*, meaning the last four bits encoded, are used. The purpose of

this context is to help make the coder reactive to the changes in subtree level, etc. In addition to the local context, *hierarchical context* relates the current coefficient's significance to that of its parent, grandparent, and siblings in the other subbands. Figure 4.8 shows these relationships. The idea here is that the significance of the coefficient is

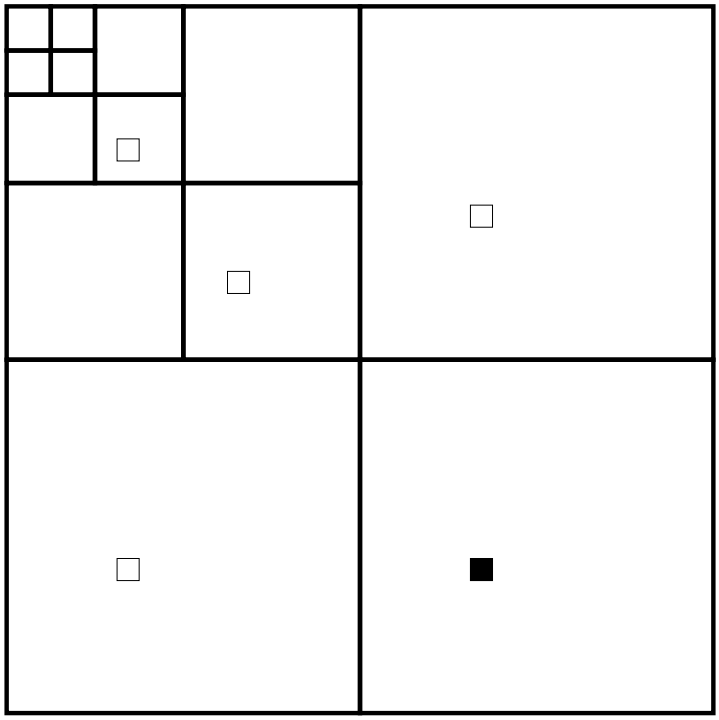


Figure 4.8: Partial encoding context. The significance of the current coefficient (black box) is evaluated in the context of the significance of other coefficients. These related coefficients (white boxes) have both parent and sibling relationships in the MRA quadtree.

highly correlated to the significance of these related coefficients, and therefore modelling with this context has strong predictive power. Taken together, there are eight bits of context, requiring $2^8 = 256$ separate models.

Decoding

The decoder operates as a sort of "mirror-image" of the encoder. By construction, all correlation information used by the encoder is available to the decoder, thus the decoder's models mirror those used in the encoder. Algorithm 4.4 describes the decoding process.

Algorithm 4.3 Bitplane encoding of grey scale image G

$G \Leftarrow$ greyscale image
$W \Leftarrow \mathrm{DWT}(G)$ {Discrete Wavelet Transform of input image}
normalize W coefficients to the range $[-1, 1]$, $s \Leftarrow$ scaling factor
encode(s) {send the scaling factor to decoder}
$N_{\text{target}} \Leftarrow$ target rate in bpp $\times$ pixels in G {number of output bits desired}
$N_{\text{output}} \Leftarrow$ {counts the number of output bits coded}
$t \Leftarrow 1.0$ {holds the current bitplane level(threshold)}
$\mathcal{T} \Leftarrow$ MRA decomposition (quad/octree) of W
repeat
  $t \Leftarrow t/2$
  for l over all levels in $\mathcal{T}$ **do**
    for b over all bands in the $\mathcal{T}$ **do**
      $\mathbf{T} = \mathcal{T}_b^l$ {the current subtree level}
      for i over rows of $\mathbf{T}$ **do**
        for i over columns of $\mathbf{T}$ **do**
          if $\mathbf{T}_{i,j}$ is not significant **then**
            {$\mathbf{T}_{i,j}$ hasn't been significant before, check against threshold t}
            $\mathcal{C} \Leftarrow$ context for $\mathbf{T}_{i,j}$
            encode($\mathcal{C}$,$|\mathbf{T}_{i,j}| \geq t$) {send one bit for significance}
            if $\mathbf{T}_{i,j} \geq t$ **then**
              {$\mathbf{T}_{i,j}$ is newly significant, so send sign and update the error}
              encode($\mathcal{C}_{\text{fixed}}$,$|\mathbf{T}_{i,j}| < t$) { one bit of sign information}
              $\mathbf{T}_{i,j} \Leftarrow \mathbf{T}_{i,j} - \mathrm{sign}(\mathbf{T}_{i,j})\frac{3t}{2}$ {update the error on $\mathbf{T}_{i,j}$ }
            end if
          else if $\mathbf{T}_{i,j}$ is significant **then**
            $\mathbf{T}_{i,j}$ has been coded before, so update approximation
            encode($\mathcal{C}_{\text{fixed}}$,$|\mathbf{T}_{i,j}| < t$) {send one bit with sign information}
            $\mathbf{T}_{i,j} \Leftarrow \mathbf{T}_{i,j} - \mathrm{sign}(\mathbf{T}_{i,j})\frac{t}{2}$ {update the error on $\mathbf{T}_{i,j}$ }
          end if
        end for
      end for
    end for
  end for
until $N_{\text{output}} \leq N_{\text{target}}$
prepend(N_{output}) {tell decoder how many bits were encoded.}

Algorithm 4.4 Bitplane decoding of grey scale approximation image $\bar{G}$

$N_{\text{input}} \Leftarrow \text{decode}(N_{output})$ bpp {number of bits the encoder output}
$W \Leftarrow 0$) {Empty array to store wavelet coefficients}
$N_{\text{input}} = 0$ {counts the number of output bits coded}
$t \Leftarrow 1.0$ {holds the current bitplane level(threshold)}
repeat
  $t \Leftarrow t/2$
  for l over all levels in the wavelet decomposition **do**
    for b over all bands in the wavelet decomposition **do**
      $\mathbf{T} = \mathcal{T}_b^l$ {the current subtree level}
      for i over rows of **T do**
        for i over columns of **T do**
          if $\mathbf{T}_{i,j}$ is not significant **then**
            {$\mathbf{T}_{i,j}$ hasn't been significant before, check against threshold t}
            $\mathcal{C} \Leftarrow$ context for $\mathbf{T}_{i,j}$
            significant $\Leftarrow \text{decode}(\mathcal{C})$ {find out if $\mathbf{T}_{i,j}$ is newly significant}
            if significant **then**
              {$\mathbf{T}_{i,j}$ is newly significant, so read sign bit and update the error}
              sign $\Leftarrow \text{decode}(\mathcal{C}_{\text{fixed}})$ {read in the sign of the approximation adjustment}
              $\mathbf{T}_{i,j} \Leftarrow \mathbf{T}_{i,j} + \text{sign}\,\frac{3t}{2}$ {update the error on $\mathbf{T}_{i,j}$ }
            end if
          else if $\mathbf{T}_{i,j}$ is significant **then**
            {$\mathbf{T}_{i,j}$ has been coded before, so update approximation}
            sign $\Leftarrow \text{decode}(\mathcal{C}_{\text{fixed}})$ {read in the sign of the approximation adjustment}
            $\mathbf{T}_{i,j} \Leftarrow \mathbf{T}_{i,j} + \text{sign}\,\frac{t}{2}$ {update the error on $\mathbf{T}_{i,j}$ }
          end if
        end for
      end for
    end for
  end for
until $N_{\text{input}} \leq N_{\text{target}}$
$s \Leftarrow$ decode(s) {read in the scaling factor used to normalize image}
$W \Leftarrow sW$
$\bar{G} \Leftarrow IDWT(W)$ {approximate image is inverse wavelet transform of W}

Numerical Results

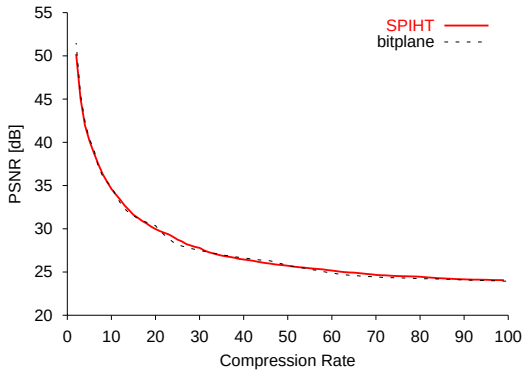
Figure 4.9 shows rate distortion comparisons between this algorithm and the SPIHT algorithm on six images (see Appendix A for the images). These results show the similar performance characteristics of the two algorithms. Note that Figure 4.9-f, the "peppers" image shows the signs of some sort of software error in the SPIHT coder. On those parts of the curve that seem to represent correct behaviour, the results are again quite similar. The two algorithms show comparable performance across a much larger group of images than the results shown here.

Remarks

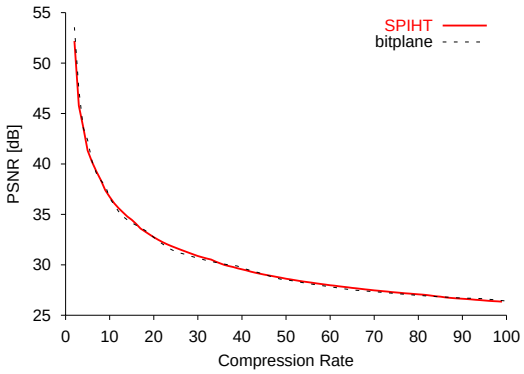
In some ways this method is similar to the zerotree approach. Quantization is similar, and in both methods the coefficient subtrees are central to the models. In fact, Shapiro discusses the relationship between EZW and bitplane coding in [50]. The most significant difference here is that Shapiro considered only the case of adaptive entropy coding of thresholded bits, whereas in this method the encoding context fills the role of EZW's zerotrees.

It turns out that this approach is also similar to one presented by Li et. al. [34]. A few numerical results (single points on a rate/distortion curve) are given, suggesting superior performance of their coder compared to the bitplane coder presented here.

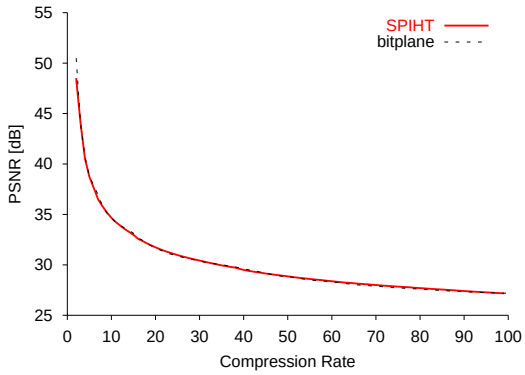
By separating the compression procedure more distinctly into encoding and modelling tasks, this coder remains quite simple yet effective. Encoding is performed by the arithmetic coder, while modelling is managed by a combination of arithmetic coding context and quantization. It is worth noting that implicit in this model is the hypothesis that insignificant coefficients will be predictable from hierarchical context in the wavelet coefficient trees. In other words, the wavelet coefficient decay will be as expected, and an individual coefficient's significance may be accurately predicted this way.



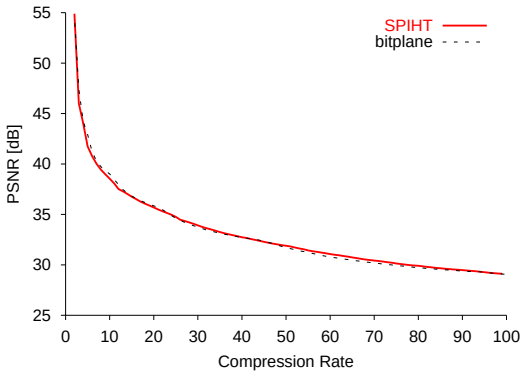
(a) barbara



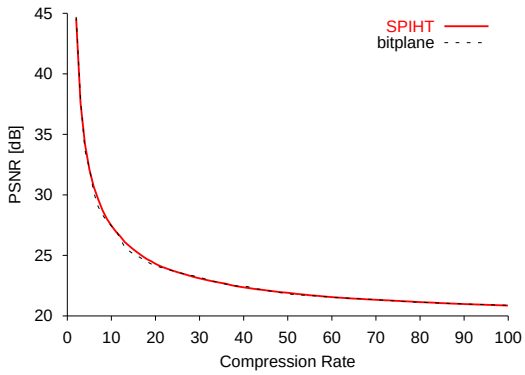
(b) boat



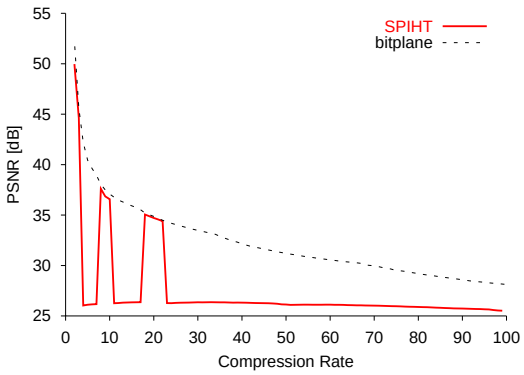
(c) goldhill



(d) lena



(e) mandrill



(f) peppers

Figure 4.9: Comparison of SPIHT and bitplane codecs on 512×512 8-bit greyscale images

4.4 Hybrid Codecs

Analogous to the IFS case in §4.2, a fractal-wavelet compression scheme may be built directly from the development in §2.8. In this case, rather than mapping domain blocks to range blocks, map infinite coefficient trees to each other.

In §2.8, the inverse problem for IFSW was discussed. Again, the collage theorem was presented as a method of approaching the problem. The following coder is a direct implementation of those results. As mentioned in §2.8, for simplicity of implementation, the scaling factors $\alpha_{j,k}^{\lambda}$ are found by employing a least-squares fit.

A simple IFSW compression algorithm

Begin again with a greyscale image of dimension $2^N \times 2^N$ (512×512), and perform a transformation to some wavelet basis. For simplicity, count the tree levels from 0 for a complete (discrete) decomposition. Thus subtrees at the n^{th} level of decomposition consist of $2^n \times 2^n$ blocks of coefficients arranged as in Figure 1.13.

This simple coder will map from one subtree level, n, to the next, $n+1$, although the trees could be offset further. Pick level 3, say, as the parent (domain) level — then the discrete wavelet transform is not performed completely, merely until level 3 has been reached. With these values, the result is a set of domain trees $\{D_m^h, D_m^v, D_m^d\}_{m=1}^{64}$

The encoding process will then first quantize the level 3 image and send it to the decoder. After this encoding[8] has been achieved each range tree (the level 4 subtrees) $\{R_n^h, R_n^v, R_n^d\}_{n=1}^{256}$, needs a mapping from the domain trees (the level 3 subtrees). Following § 2.8, look for maps of the form $\phi_{R_n} : D_m \to R_n \quad R_n = \alpha_n D_m$. There is now a choice of whether to fit the subbands together (at a cost of one quantized α for three subtrees) or separately (more accurate maps). Assuming that the subbands are treated separately, just follow a similar process to that in the spatial IFSM case: for each R_n perform a least-squares fit to find the map ϕ_m for each domain tree D_m and pick the best (i.e., the least error of approximation) one. Once all the maps are found, the list is sent to the decoder.

The decoder is also very similar to the spatial IFSM case. Instead of starting with any image, the level 3 image is first read in. After reading in the list of α_n matched with D_m, the IFSW maps are created, and iterated on this image. When the process converges, the resulting image is an approximation to the original wavelet coefficient tree.

Figure 4.10 shows some examples of this approach to IFSW image coding.

Remarks

The primary disadvantage to this scheme as described is the granularity of the compression rate; changing the parent and child levels by one makes a very large change

[8] The quantized coefficients may be entropy coded for a small rate gain.

(as seen in Figure 4.10) in the rate/distortion result. Of course one would like more fine-grained control of the r/d curve. To address this problem, many refinements of the method have been made, see [28, 35, 8]. Although many gains over the simple method above have been made, the IFSW coders do not perform as well, on general images, as zerotree wavelet methods like SPIHT.

Another significant issue can be the search time of evaluating all of the range and domain pairs, looking for the best fit. There are several simple ways to reduce the time from the *full search* method described above. One of the simplest is to not treat the subbands separately, that is

$$j^h(j,k) = j^v(j,k) = j^d(j,k), \qquad k^h(j,k) = k^v(j,k) = k^d(j,k),$$
$$\alpha^h_{j,k} = \alpha^v_{j,k} = \alpha^d_{j,k}\,.$$

This method reduces the search time and the accuracy of the approximation; it also reduces the storage space for the maps. The hybrid coder presented next uses this approach.

4.4.1 Proposed Hybrid Codec

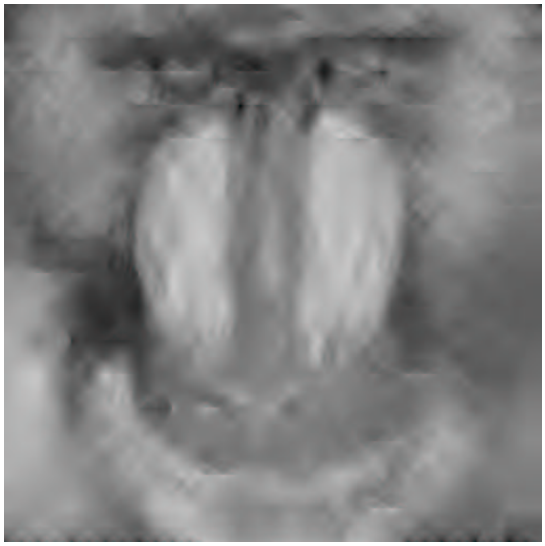
One of the characteristics of zerotree coders (and similarly, schemes like the one presented in §4.3.3) are that many small coefficients may be approximated as zero, especially at low bit-rates. Conversely, a characteristic of the IFSW methods described previously is that all coefficients are potentially non-zero; coefficients at high levels in the tree will be small due to the decay rate of the coefficient trees, but they need not be zero.

From these two observations, a new hybrid coder was conceived. The idea is to use an IFSW approach starting at a high level parent (level 2 or 3, for example) to get a very rough approximation of the image *which includes many high frequency coefficients*. After this rough approximation is made, the remainder of the bit budget[9] is applied to the bitplane method of §4.3.3 is to improve the approximation. Large coefficients may even be ignored in the least-squares fit, in expectation that they will both be well approximated by the bitplane method, and that this will allow a better fit to the small coefficients.

This method can be characterized as a *fractal pre-processing* approach. One issue to consider is that the bitplane (and zerotree) algorithms are designed to approach the optimal PSNR reduction scheme of always reducing the currently largest coefficient error. As such, it can be quite difficult to compete with these algorithms on straight PSNR merits. On the other hand, it has been observed [6]that fractal methods are quite (visually) good at representing some types of image textures, and may result in images of better "visual quality" at the same PSNR. A similar result here would be interesting, even if PSNR was somewhat degraded.

[9] A common term for the number of bits a coder may use.

(a) original image



(b) level 3 parent

(c) level 4 parent

(d) level 5 parent

Figure 4.10: Examples of IFSW compression on the mandrill image using Antonini 9/7 basis (a) original mandrill image (b) level three parent mapped to level four child with rate of 143:1 and PSNR of 19.184 (c) level four parent mapped to level five child with rate of 32:1 and PSNR of 20.384 (d) level five parent mapped to level six child with rate of 7.5:1 and PSNR of 22.628

It should be noted that the balance between IFSW and bitplane, i.e. the number of bits to spend on each algorithm, is neither simple nor does it (due to the "visual quality" effect) lend itself to easy analysis. The operating assumption has been that any given IFSW scheme (within reason) ought to be quite good at *some* point of the rate/distortion curve. For this reason, the IFSW approach was fixed, and entire rate/distortion curves for several images compared. If the premise of this coder holds true, there ought to be a region on the curve that shows improvement.

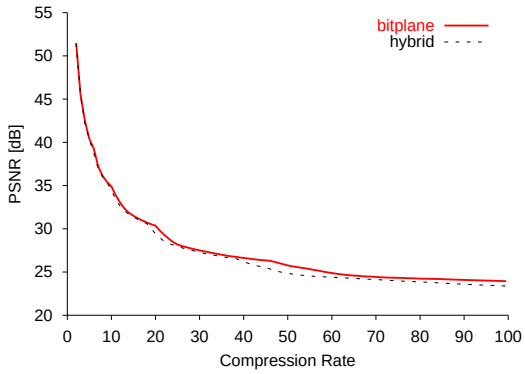
Numerical Results

Figure 4.11 shows a comparison of the bitplane method with this hybrid IFSW plus bitplane method. The wavelet basis used was the Antonini 9/7, and the IFSW maps were from trees of level 3 to trees of level 4. It is interesting to note the discrepancy between the two curves is different for different images. In some cases, the rate/distortion curve is slightly affected, while other images show a more noticeable drop in performance at higher compression rates.

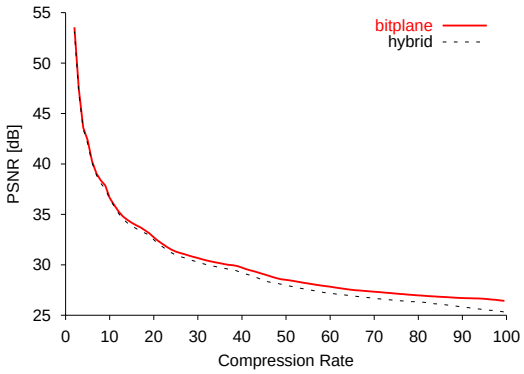
Remarks

Under purely subjective visual comparisons (as against PSNR values) the hoped-for gains were not apparent. In regions of the r/d curves where the PSNR was very different, the hybrid coder images were noticeably inferior as expected. In regions where this was not the case, visual gains, if any, were nearly imperceptible.

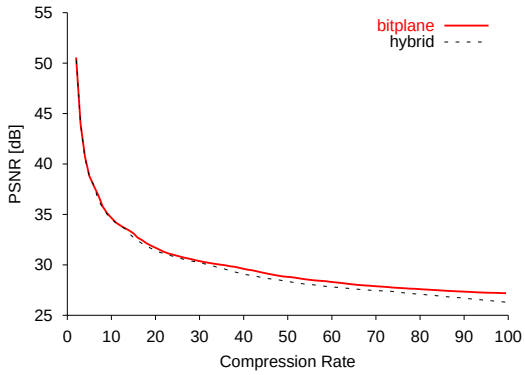
Although data for six images under one 'balance' of IFSW to bitplane cost has been shown, the method has been tried on many more images, and in several variations. Subjective evaluation of the results does show improvements for some images. These gains on some images are balanced out by losses on others, and in any case seem to be small enough as to be irrelevant.



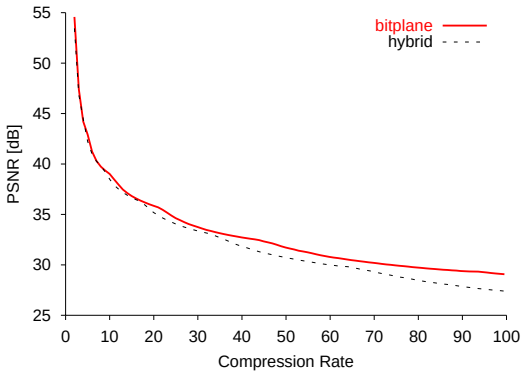
(a) barbara



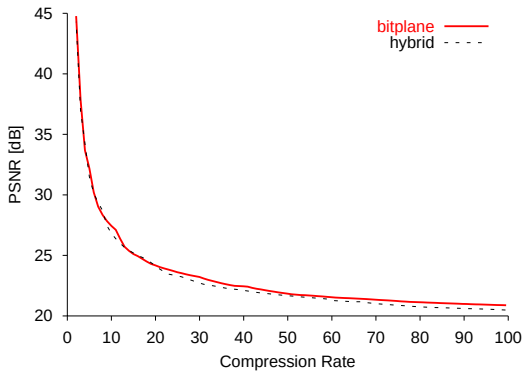
(b) boat



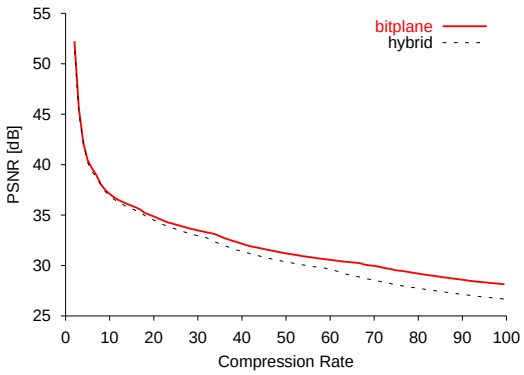
(c) goldhill



(d) lena



(e) mandrill



(f) peppers

Figure 4.11: Comparison of bitplane and hybrid codecs on 512×512 8-bit greyscale images

4.5 Further Investigations

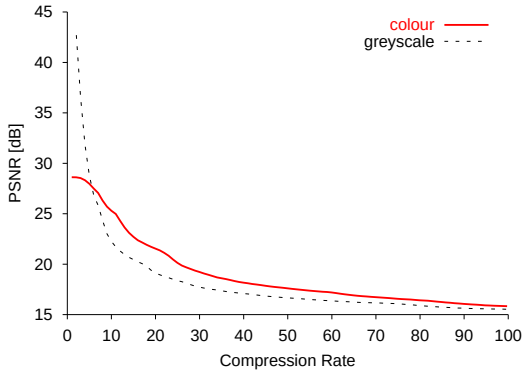
4.5.1 Colour

All of the previously discussed work may be extended to colour images. Recall that an image was defined to have one or more channels — the greyscale images have a single channel and a Red Green Blue (RGB) colour image will have three channels. Images with multiple channels, especially more than three, are often called multi-spectral images. The simplest way to handle colour is to treat each channel just like you would a greyscale image. However, it turns out that there is redundancy in the colour channels (spectral correlation); thus it is important to take advantage of this before coding.

One approach is to map the image to a different colour space before coding, and rely on the colour space to remove the redundancy (de-correlate the pixels). That is the approach used in this work. RGB images are first mapped into the YUV colour space. For a discussion of various colour spaces, see [25]; for applications specifically to compression, see [51, 2].

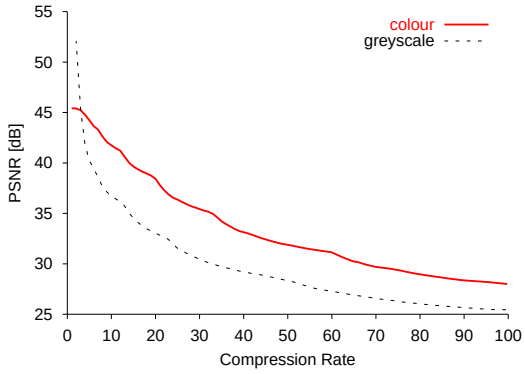
The YUV space may not provide the most effective spectral de-correlation, but it is an important colour space for video coding; YUV was designed for video transmission (and backwards compatibility with black and white TV's), and most video streams are stored in this colour space. From an implementation point of view, YUV is also convenient since it is a linear transformation from RGB space. For these reasons it was used exclusively in this work.

Figure 4.12 shows rate/distortion curves for the bitplane algorithm on colour and greyscale versions of the same image. All encoding parameters are identical other than colour. These results show the performance gain allowed by the redundancy in colour information.



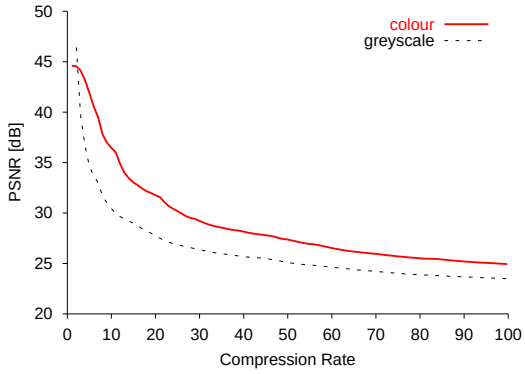
(a) frymire

(b) lena

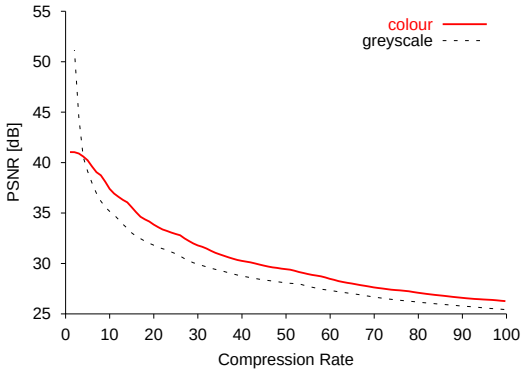


(c) monarch

(d) peppers



(e) sail



(f) tulips

Figure 4.12: Comparison of rate/distortion curves for colour and greyscale versions of the same image

4.5.2 Three-dimensional Image Compression

Video Compression

A video stream can be considered as a sequence of two-dimensional images, or as a three-dimensional image. By breaking the stream into a series of (dyadic length for simplicity) blocks in the time direction, the result is three-dimensional image blocks suitable for application of a three-dimensional version of the bitplane algorithm (§4.3.3). The video stream may be greyscale or, as in the case of the data shown here, colour. Colour streams are handled in the same manner as colour still images. The bitplane algorithm is identical, except that there are 7 subbands rather than 3 as in the two-dimensional case.

Video compression is complicated by a number of factors, not the least of which is the speed at which practical decompression may be performed. Another confounding factor in extending two-dimensional methods to three-dimensional is the fact [60, 39] that humans perceive motion differently than still images. As in the case of still images, only PSNR results are used to quantify distortion. There is no reason to think that this is optimal. However, there are few or no other metrics that have been shown to be superior [57].

In the following, example results are shown for two test sequences. These plots are not rate/distortion curves. Rather, they show the distortion in every frame for a given compression rate. Both data sets were compressed at a rate of about 100:1, which lies within the design goals of the MPEG-2 coder. It would take a truly large amount of data to demonstrate conclusively a performance comparison of any algorithms operating on video streams. No such attempt is made here; these two examples were picked to illustrate some features of the bitplane coder in three dimensions

Kayak Sequence

This sequence is a subsample from one of the Visual Quality Experts Group (VQEG) test sequences. It is a live action sequence, as illustrated by several frames shown in Figure 4.13. Distortion results for Motion Picture Experts Group (MPEG)-2 and a three-dimensional version of the bitplane algorithm are shown in Figure 4.14.

Venus Cubes Sequence

This sequence is a computer generated animation, available at the Renderman Repository. The sequence is problematic for many video compression algorithms due to the sharp lines, similar detail in the cubes, and various scale of the objects. Figure 4.15 shows several frames from the sequence. Figure 4.16 shows the distortion results for the bitplane method and MPEG-2 at a rate of approximately 100:1.

Remarks

It should be noted that the algorithm as stated is quite impractical for video compression tasks, due to the encoding and decoding times needed[10]. On the other hand, the method shows results for a true three-dimensional wavelet approach. There are some advantages to this approach evident in the samples presented.

In the first sequence, "kayak", the the MPEG-2 curve shows 'spikes' where the reference frames (higher quality frames that are used to interpolate/extrapolate to other frames), called I-frames, occur. Overall performance of the three-dimensional bitplane algorithm is quite good, and distortion rates are similar for all frames.

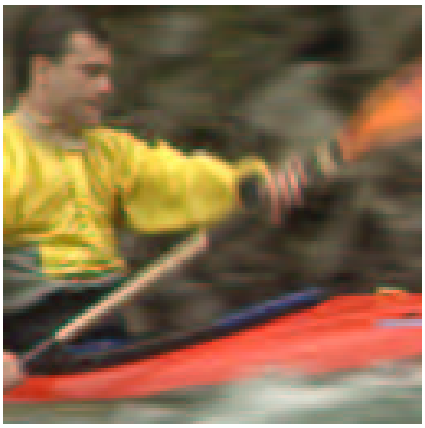
The "Venuscubes" demonstrates a common problem with two-dimensionally oriented coders dealing with motion. The frames near the middle of the sequence (see Figure 4.15-e) are especially difficult for motion detection algorithms. Motion detected in the spinning cubes encroaches on the planet edge, and the motion vectors tend to get lost. At the aforementioned point in the sequence, the MPEG-2 coder suffers a drastic drop in performance for several frames due to this motion detection problem. The three-dimensional bitplane coder, having no motion detection, is not adversely affected.

With the bitplane coder extended to handle three-dimensional data, it would be simple to present results for the hybrid coder of §4.4.1 as well. In light of the lack of improvement for two-dimensional inputs, and the large computation times, doing IFSW on these three-dimensional data sets, this was deemed unnecessary.

Medical Images

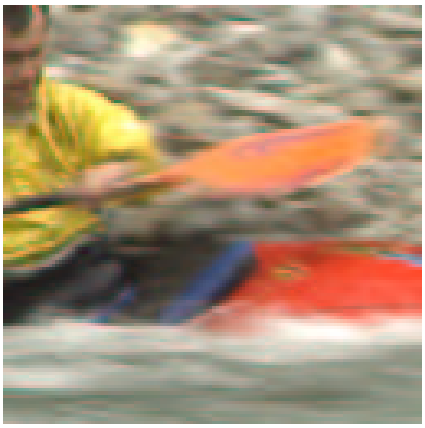
Perhaps a more appropriate (given the computational constraints of video) application of the three-dimensional version of the bitplane algorithm is to medical images. Here the benefit of an embedded coder is that lower resolution approximations may be transmitted first, but high-resolution, high quality versions can also be transmitted. With the large storage requirements of many modern medical images, approaches like this can enable quick review of a image set by reducing the amount of data transferred. When needed, a high quality section of the image may be transferred.

[10]Typical video signals may have 24 or 30 frames/sec. These methods require seconds to minutes/frame on commodity hardware.

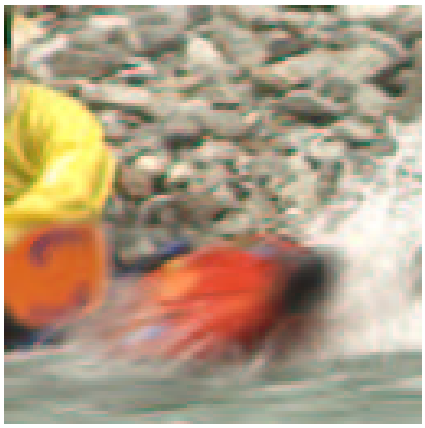


(a) frame 0

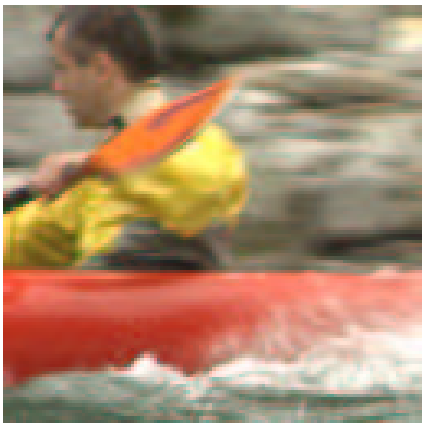
(b) frame 24



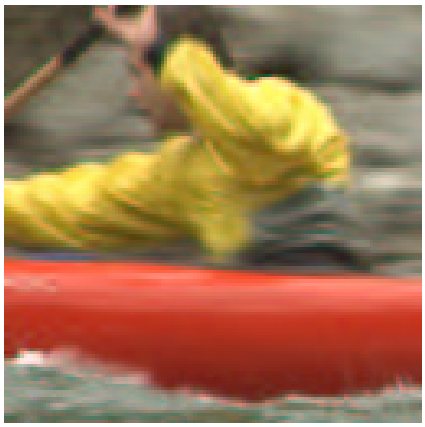
(c) frame 38



(d) frame 45



(e) frame 68



(f) frame 76

Figure 4.13: Several frames from the Kayak sequence

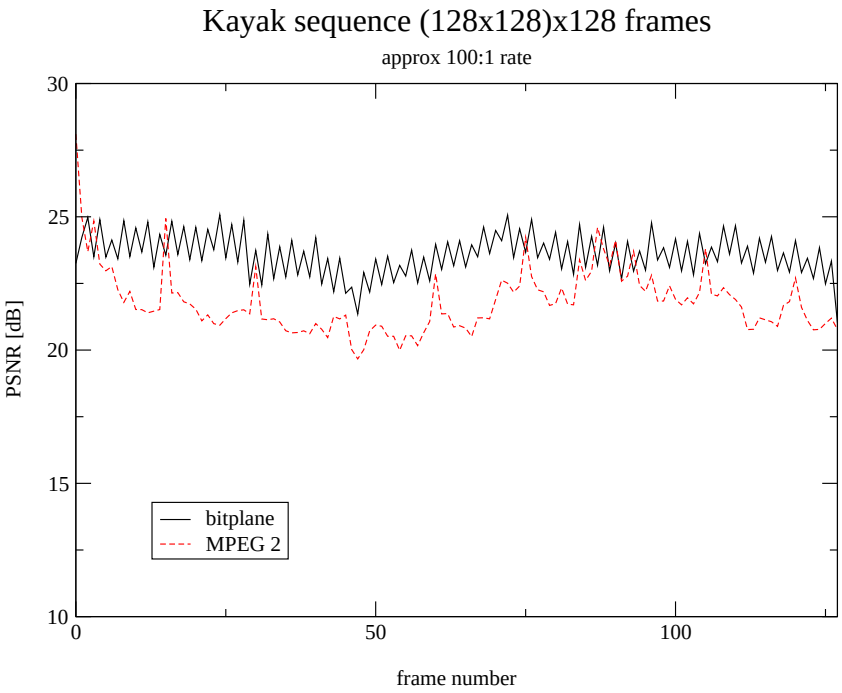
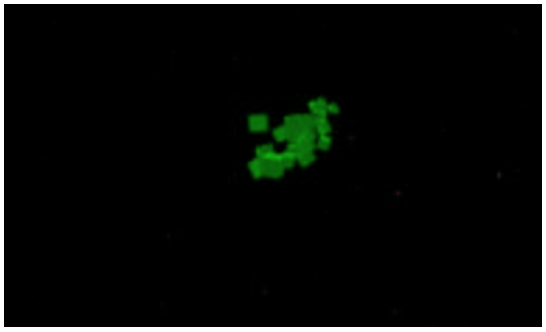
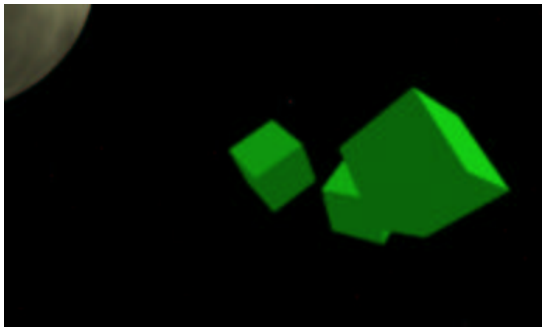


Figure 4.14: Video compression of the ”Kayak” sequence

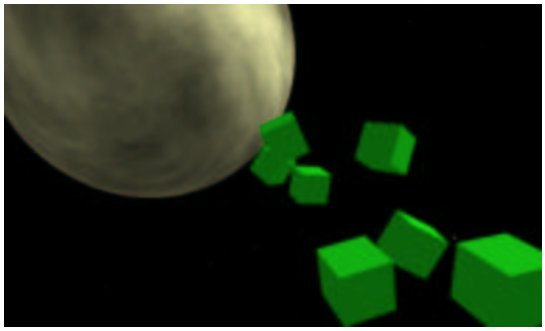


(a) frame 0

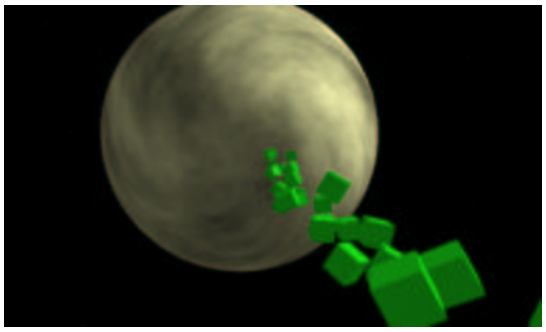
(b) frame 30



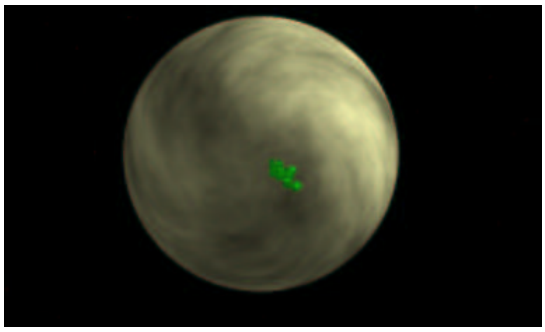
(c) frame 48



(d) frame 65



(e) frame 128



(f) frame 235

Figure 4.15: Several frames from the venuscubes sequence

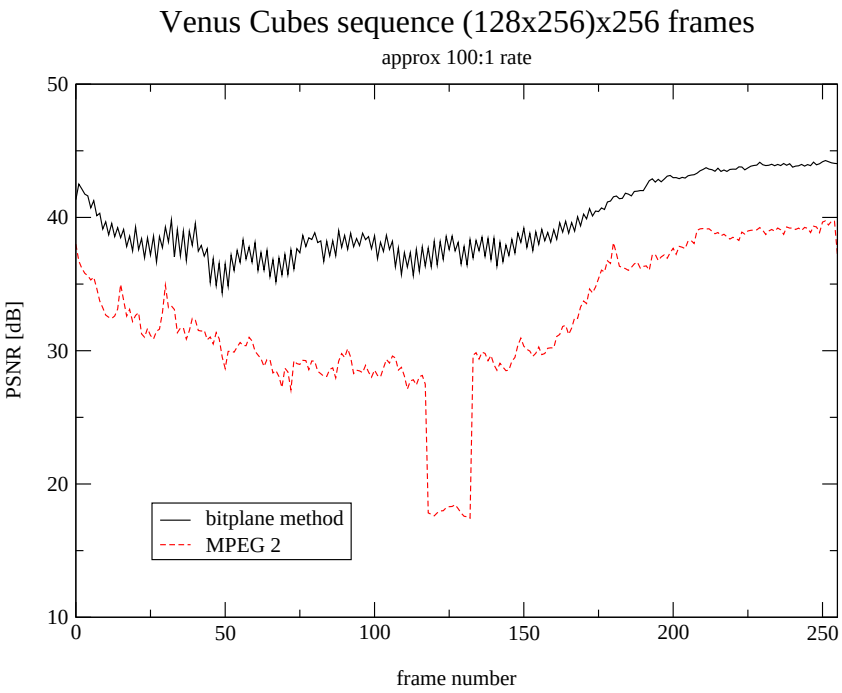


Figure 4.16: Video compression of the ”Venus Cubes” sequence

Chapter 5

Concluding Remarks

This thesis has presented and discussed several areas of image compression. The theoretical background underpinning wavelets, fractal IFSM, and fractal-wavelet IFSW methods was investigated. Additionally, the arithmetic coding method of entropy coding was presented. Two original algorithms for compression of images were detailed. These coders had two specific goals:

1. Investigate the construction of a wavelet based coder with explicit separation of modelling and coding functions. Contrast the performance of this method with standard approaches.

2. Using as a basis the first coder, investigate the construction of a high performance (in a rate/distortion sense) hybrid fractal-wavelet coder.

These goals have been met in Chapter 4. Numerical results on a small but representative set of images were presented to support the discussion. The novel bitplane coder described in §4.3.3 has performance comparable to the SPIHT algorithm, which places it slightly behind current state-of-the-art coders.

In §4.4.1, a new hybrid fractal-wavelet scheme was proposed using the bitplane coder following an IFSW stage. It was hoped that this method would provide visual gains (in the sense of perceivable distortion) with little or no cost in the rate/distortion curve. While performance of the coder is good, the hoped for visual gains were not apparent. In the absence of a compelling visual gain, the extra computation time needed for this method makes it hard to justify.

Extensions of the bitplane method (from §4.3.3) were presented. The compression of colour images was discussed, resulting in significant compression gains (relative to greyscale versions) due to redundancy in the colour channels. Application to three-dimensional images was also addressed, including comparisons with the MPEG-2 video coder on two video streams. The treatment of video as pure three-dimensional data sets has some advantages; motion detection is not used and so cannot create artifacts. Overall strong rate/distortion performance in comparison to the industry

standard MPEG-2 is shown. On the other hand, computation times make this method, as written, impractical for real-time video.

Additional discussion on several points of interest such as three-dimensional medical images are offered. Discussion of the visual effects of biorthogonal wavelet bases is also given; these results are known but often not discussed in the literature.

Future research

The algorithms presented in this thesis have significant potential for future research. For example, it remains to tackle the issue of optimal encoding context for the bitplane coder. Similarly, better bitplane distribution may be possible.

Chapter 4 barely began to scratch the surface of three-dimensional coding issues. Two avenues are open to further research in this area: Approach more specifically the constraints of video coding (a busy area of research today), or concentrate on domain-specific goals of truly three-dimensional data sets such as those constructed by some medical imaging modalities.

Appendix A

Original Images

The original images are reproduced here to avoid duplication throughout the thesis. Figure A.1 shows the 8-bit greyscale image set, and Figure A.2 the colour set. In both cases, all images are 512x512 pixels.

Both greyscale and colour images are reproduced in greyscale in the printed version of this thesis. Images are difficult to reproduce on paper; they will be more easily viewed in the digital version of the thesis, which is available at the University of Waterloo's electronic thesis database or at

`http://links.uwaterloo.ca/~simon .`

(a) barbara

(b) boat

(c) goldhill

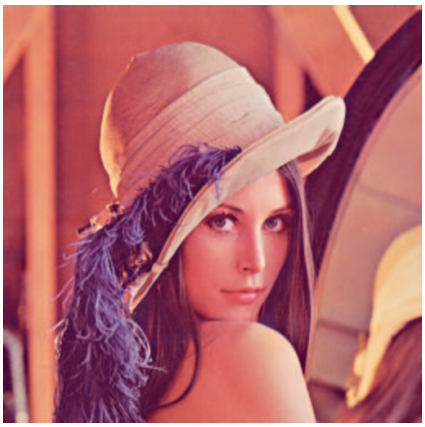
(d) lena

(e) mandrill

(f) peppers

Figure A.1: Original 8-bit greyscale images

(a) frymire



(b) lena

(c) monarch

(d) peppers

(e) sail

(f) tulips

Figure A.2: Original colour images

Appendix B

Software

It was immediately apparent upon beginning the background work for this thesis that several of the optimization operations (such as searching for maps in IFS structures) were going to be computationally expensive. Software like MATLAB can be quite slow for these types of operations, and moving to three dimensions would compound the problem. For these reasons, it was decided that a C++ framework would be useful. Unable to find existing software that met these needs, this author decided to create it. A few core components are briefly introduced here. The software is far from complete as of this writing, but was sufficient for the needs of this thesis, and has been used by other graduate students as well. It is hoped that building on this project will result in a general package useful to the research group as a whole, for wavelet and general image manipulation tasks.

The following discussion assumes some basic familiarity with the C++ language and idioms. A much more detailed description is available with the software itself.

B.1 Arithmetic Coder

A set of arithmetic coding classes was created to allow flexible coding. The package consists of a bit-packing wrapper to iostreams, called "Bitstream", encoding ("Encoder") and decoding ("Decoder") classes, and a model class "Model". The model class has a specialized version, called "BinaryModel", since many operations may be optimized for the binary model case. Implementation is similar to that described in [9, 48]. Essentially, the implementation follows from the discussion of §3.3, with the important distinction that all ranges and frequencies are represented by integer values. This avoids (most) problems with rounding errors, and can be more efficient on some architectures.

The following nearly verbatim code snippet (Figure B.1) illustrates the ease of use this object-based approach allows. This function is used to encode an input stream with local context; that is, the model varies depending on the previous entries.

code B.1.1 Arithmetic coding with local context

```
void
encode(istream &input, ostream &output, size_t context_bits)
{
    //tie input and output to bit-packed streams
    IBitstream ibitstream(input);
    OBitstream obitstream(output);

    //create an encoder
    arithcoder::Encoder coder(obitstream);

    int num_contexts = 1 << context_bits;
    int context_mask= (1 << context_bits) -1;

    //Create an array of contexts that will act as a lookup
    //table.  Each of the contexts is initialized to a default
    // (.5,.5) state. We need num_contexts seperate models.
    vector<arithcoder::BinaryModel> contexts(num_contexts);

    int curr_context=0;
    for(;;) {
        //read in a new bit
        arithcoder::symbol_t symbol=ibitstream.unpack_bit();

        //code it with the current model
        coder.code_symbol(contexts[curr_context],symbol);

        //now update the current context based on latest
        //symbol
        curr_context<<=1;
        curr_context|=symbol;
        curr_context&=context_mask;
        if(ibitstream.eof())
            break; //we are done
    }
    coder.end();
    length=coder.input_bits();
}
```

B.2 Array Package

One of the primary goals of the software package was to make the dimensionality of the data set as transparent as possible, thus operations on two-dimensional or three-dimensional data would use as much of the same code as possible. To this end, a general container class,"Array", was created. Array is templated on data type and dimension. The class is quite similar[1] to the "blitz++" class described in [55]. For efficiency reasons, the methods of template meta-programs [54] and expression templates [53] were introduced by Todd Veldhuizen (author of the aforementioned blitz++ package). The primary advantages of using Array as a container class are as follows:

1. a fairly natural syntax
2. slicing and subarrays in an dimension and direction
3. avoids copying data wherever possible, for efficiency
4. iterators don't have to know the dimensionality of the array, allowing for generic algorithms
5. multiple views into one storage are possible, without copying data

Figure B.1 illustrates the internal structure of the Array class.

The following code snippet (Figure 27) will briefly illustrate the usage of the Array class.

[1] In fact, the software was initiated with the idea of basing it around the blitz++ array. Unfortunately, these techniques are quite cutting-edge and tend to find problems with C++ compilers. Due to long compile times and trouble debugging complicated class structures, it was decided to implement a simpler array class. Some effort has been made recently to move the design closer to blitz++, to allow possible drop-in replacement at a later date.

code B.2.1 Various array operations possible with the Array class

```
//create an NxN  2-d array of doubles
Array<double,2> data(N,N);

// a decimated view that does not copy data
// decimated points to every 2nd (in each dimension)
// entry of data
Array<double,2> decimated(
    array(Range(data.lbound(0),2,data.ubound(0)),
          Range(data.lbound(1),2,data.ubound(1))));

Array<double,2> decim_copy(N/2,N/2);
decim_copy=decimated;
decimated=0;
//now every 2nd entry in data is 0 (and all entries of decmiated)
//but the copy in decim_copy is not touched.

//row3 is a 1d, N length vector holding the 3rd row of data
Array<double,1> row3(data(3,Range((data.lbound(1),2,data.ubound(1)))));

//now to look at iterators, define a function max_magnitude
//which will return the largest (magnitude) value in an array.
template<class T, int N> T
max_magnitude(const Array<T,N> &a)
{
    Array<T,N>::const_iterator i=a.begin();
    T res=*i; ++i;
    for(; i != a.end(); ++i){
        T val = *i;
        val = val < 0 ? -val: val;
        if ( val > res)
            res = val;
    }
    return res;
}

//thus if we have a 2d array
Array<double,2> data(N,N);
//and a 3d array
Array<double,3> data2(N,N,N);

//the following will generate template instances
//for both 2d and 3d arrays, from the above code
double max=max_magnitude(data);
double max2=max_magnitude(data2);
```

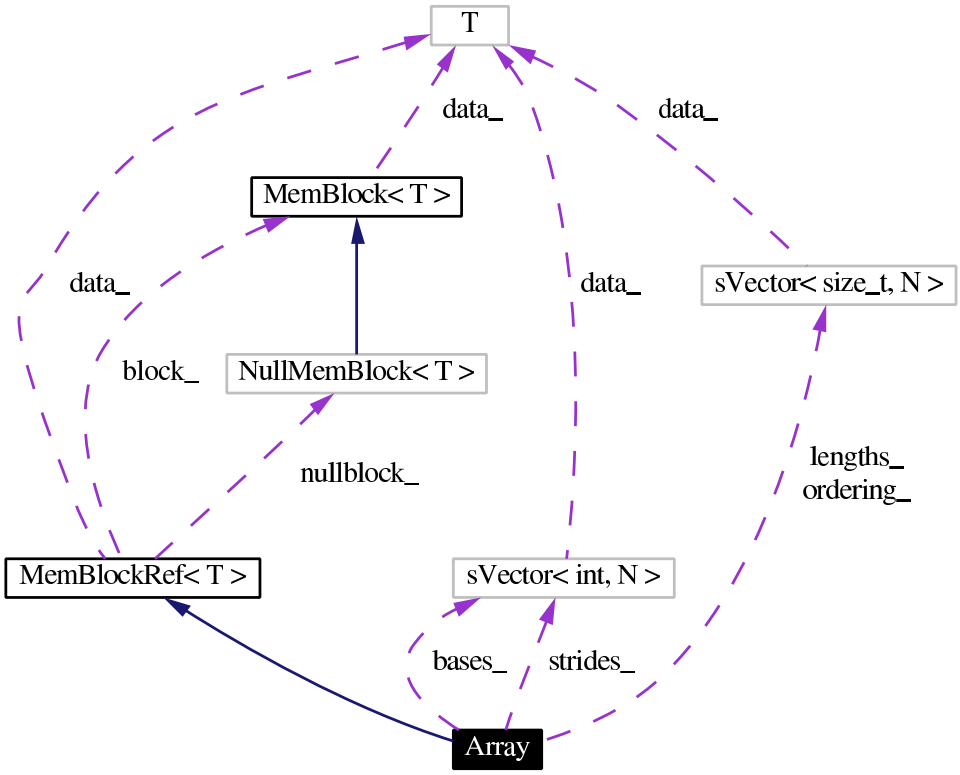


Figure B.1: Structure of the Array<T,N> class

B.3 Wavelet Classes

Wavelet operations are implemented on top of the Array class. Filters are implemented as one-dimensional arrays, and image data may consist of two- or three-dimensional inputs. In the case of colour images/video, there will be a collection of three such arrays. Due to the level of abstraction represented by the Array class, the wavelet transform is essentially identical for all array dimensions (it simply has more indices to iterate over).

Figure B.2 describes the class relationships in this implementation. Solid lines represent class inheritance, while dashed line show membership. Symmetric wavelet inherits its basic structure from the Wavelet class, which is implemented in terms of four Filters, and a buffer. Each filter is a one-dimensional floating point array, Array<Float,1>, as is the buffer. Included in the diagram is the instantiation for Array<T,N> as an Array<Float,1>. By isolating most of the complexity in the Array class, utility classes such as Wavelet can be implemented cleanly.

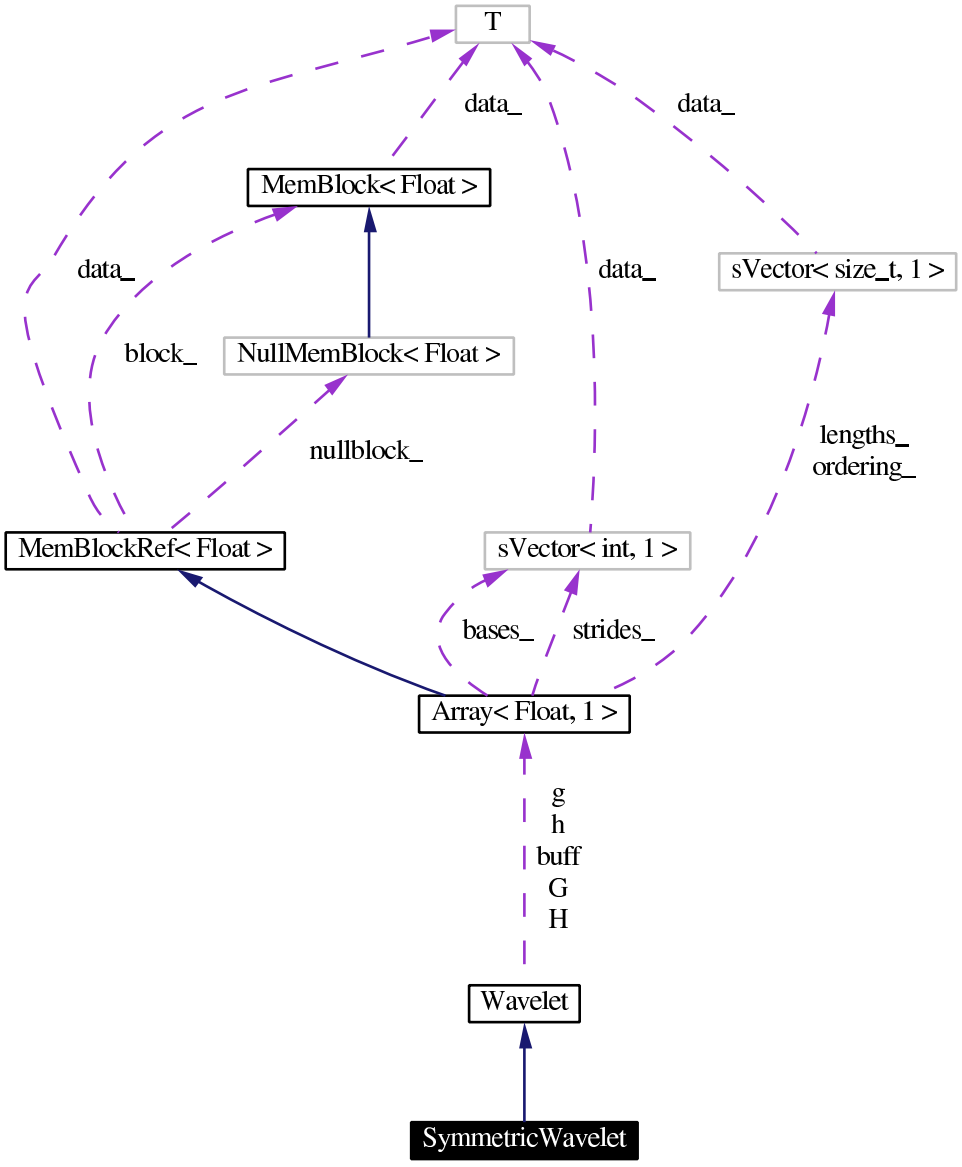


Figure B.2: Class relationships for the symmetric wavelet class

Acronyms

IFS Iterated Function Systems.

IFSM Iterated Function Systems on Grey Level Maps.

IFSW Iterated Function Systems on Wavelet trees.

EZW Embedded Zerotree Coder.

SPIHT Set Partitioning In Hierarchical Trees.

pdf probability density function.

cdf cumulative probability density function.

MRA Multiresolution Analysis.

DWT Discrete Wavelet Transform.

CWT Continuous Wavelet Transform.

CMP Contraction Mapping Principle.

QMF Quadrature Mirror Filter.

FIR Finite Impulse Response.

ASCII American Standard Code for Information Interchange.

PSNR Peak Signal to Noise Ratio.

MSE Mean Squared Error.

JPEG Joint Photographic Experts Group.

MPEG Motion Picture Experts Group.

VQEG Visual Quality Experts Group.

RGB Red Green Blue.

Bibliography

[1] E. H. Adelson, E. Simoncelli, and R. Hingorani. Orthogonal pyramid transforms for image coding. In *Visual Communications and Image Processing II*, volume 845, pages 50–58, 1987.

[2] A. Ahumada and H. Peterson. Luminance-model-based dct quantization for color image compression. In *Proceedings of the SPIE*, volume 1666, pages 365–374, 1992.

[3] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 2(1):205–220, 1992.

[4] S. Banach. Sur les opérations dans les ensembles abstraits et leur application aux équations integrales. *Fundamenta Mathematicae*, 3:133–181, 1922.

[5] M. F. Barnsley. *Fractals Everywhere*. Academic Press Professional, Cambridge, MA, second edition, 1993.

[6] M. F. Barnsley and L. P. Hurd. *Fractal Image Compression*. AK Peters, Wellesley, MA, USA, 1993.

[7] T. P. Barnwell and M. J. T. Smith. Exact reconstruction techniques for tree structured subband coders. *IEEE Trans. Acoust. Speech Signal Process.*, 34:434–441, 1986.

[8] K. U. Barthel, S. Brandau, W. Hermesmeier, and G. Heising. Zerotree wavelet coding using fractal prediction. In *Proceedings ICIP-97 (IEEE International Conference on Image Processing)*, Santa Barbara, CA, USA, 1997.

[9] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice Hall, Englewood Cliffs, 1990.

[10] C. M. Brislawn. Preservation of subband symmetry in multirate signal coding. *IEEE Trans. Signal Processing*, 43(12):3046–3050, December 1995.

[11] C. M. Brislawn. Classification of nonexpansive symmetric extension transforms for multirate filter banks. *Appl. Comput. Harmonic Analysis.*, 3:337–357, 1996.

[12] R. W. Buccigrossi and E. P. Simoncelli. Image compression via joint statistical characterization in the wavelet domain. Technical Report 414, IEEE, Munich, Germany, 1997.

[13] F. W. Campbell. The human eye as an optical filter. In *Proceedings of the IEEE*, volume 56, pages 1009–1014, June 1968.

[14] P. Centore and E. R. Vrscay. Continuity of attractors and invariant measures for iterated function systems. *Canadia Mathematical Bulletin*, 37(3):315–329, 1994.

[15] A. Cohen, I. Daubechies, and P. Vial. Wavelet bases on the interval and fast algorithms. *J. of Appl. and Comput. Harmonic Analysis*, 1:54–81, 1993.

[16] G. V. Cormack and R. N. Horspool. Algorithms for adaptive Huffman codes. *Information Processing Letters*, 18(3):159–165, 1984.

[17] A. Cohen I. Daubechies and J.-C. Feauveau. Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 45:485–560, 1992.

[18] I. Daubechies. *Ten Lectures on Wavelets*. SIAM Press, Philadelphia, PA, 1992.

[19] G. Davos. A wavelet-based analysis fractal image compresion. *IEEE Tran. Image Proc.*, 7:141–154, 1998.

[20] D. Duttweiler and C. Chamzas. Probability estimation in arithmetic and adaptive-huffman entropy coders. *IEEE Transactions on Image Processing*, 4(3), 1995.

[21] K. Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. John Wiley and Sons, West Sussex, England, 1990.

[22] K. Falconer. *Techniques in Fractal Geometry*. John Wiley and Sons, West Sussex, England, 1997.

[23] Y. Fischer, editor. *Fractal Image Compression: Theory and Application*. Springer-Verlag, 1995.

[24] Y. Fischer, editor. *Fractal Image Encoding and Analysis*, NATO ASI Series F 159, New York, 1998. Springer-Verlag.

[25] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison Wesley, 2nd edition, 1997.

[26] B. Forte and E. R. Vrscay. Inverse problem methods for generalized fractal transforms. In Fischer [24].

[27] B. Forte and E. R. Vrscay. Theory of generalized fractal transforms. In Fischer [24].

[28] M. Ghazel and E. R. Vrscay. An effective hybrid fractal-wavelet imagec coder using quadtree patritioning and pruning. In *Proc. Can. Conf. Elect. Comp. Eng.*, Halifax, Nova Scotia, 2000. CCECE.

[29] A. Haar. Zur theorie der orthogonalen funktionensysteme. *Math. Annal.*, 69:331–371, 1910.

[30] H. Hartenstiein. *Topics in Fractal Image Compression and Near-Lossless Image Coding*. PhD thesis, Albert-Ludwigs-Universität Freiburg im Breisgau, 1998.

[31] P. G. Howard and J. S. Vitter. Practical implementations of arithmetic coding. In J. A. Storer, editor, *Image and Text Compression*, pages 85–112, Norwell, MA, 1992. Kluwer Academic Publishers.

[32] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Electronics and Radio Engineers*, 40(9):1098–1101, September 1952.

[33] A. K. Jain. *Fundamentals Of Digital Image Processing*. Prentice-Hall, 1989.

[34] J. Li, P. Cheng, and C. Kuo. On the improvements of embedded zerotree wavelet (ezw) coding. In *Proceedings of the SPIE*, volume 2501, pages 1490–1501. SPIE, May 1995.

[35] J. Li and C.-C. Jay Kuo. Fractal wavelet coding using a rate-distortion constraint. In *Proceedings ICIP-96 (IEEE International Conference on Image Processing)*, volume II, pages 81–84, Lausanne, Switzerland, 1996.

[36] S. Mallat. Multiresolution approximation and wavelet orthonormal bases of fixme l-two. *Transactions of the American Mathematics Society*, 315:69–87, 1989.

[37] S. Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, San Diego, 1998.

[38] B. Mandelbrot. *The Fractal Geometry Of Nature*. Freeman, New York, 1977.

[39] D. Marr. *Vision*. W.H. Freeman and Co., San Fransisco, 1982.

[40] F. Mendivil and E. R. Vrscay. Correspondence between fractal-wavelet transforms and iterated function systems with grey-level maps. In E. Lutton J. Levey-Vehel and C. Tricot, editors, *Fractals in Engineering: From Theory to Industrial Applications*, pages 54–64, London, 1997. Springer-Verlag.

[41] F. Mintzer. Filter for distorion free two-bound multirate filter banks. *IEEE Transactions on Acoutic, Speech, and Signal Processing*, 33(3):626–630, June 1985.

[42] N. Nill. A visual model weighted cosine transform for image compression and quality assesment. *IEEE Trans, Comm.*, pages 551–557, 1985.

[43] G. Øien, R. Hamzaoui, and D. Saupe. On the limitations of fractal image texture coding. In *IEEE Nordic Signal Processing Symposium*, Espoo, Sept. 1996.

[44] A. Oppenheim and R. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall, 1989.

[45] W. K. Pratt. *Digital Image Processing*. Wiley-Interscience, New York, 1978.

[46] A. Said and W. A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6:243–250, 1996.

[47] D. Saupe and R. Hamzaoui. Complexity reduction methods for fractal image compression. In J. M. Blackledge, editor, *Proceedings of the IMA Conference on Image Processing: Mathematical Methods and Applications*, pages 211–229, Oxford, England, 1994.

[48] K. Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 1996.

[49] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[50] J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 41(12):3445–3462, 1993.

[51] Y. Sun and S. Bow. Fast wavelet transform for color image compressio. In *Proceedings of the IEEE International Conference on Image Processing*, pages 541–544, 1996.

[52] P. N. Topiwala, editor. *Wavelet Image and Video Compression*. Kluwer Academic Publishers, 1998.

[53] T. Veldhuizen. Expression templates. *C++ Report*, 7(5):26–31, June 1995. Reprinted in C++ Gems, ed. Stanley Lippman.

[54] T. Veldhuizen. Using C++ template metaprograms. *C++ Report*, 7(4):36–43, May 1995. Reprinted in C++ Gems, ed. Stanley Lippman.

[55] T. Veldhuizen. Arrays in blitz++. In *Proceedings of the 2nd International Scientific Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, Lecture Notes in Computer Science. Springer-Verlag, 1998.

[56] M. Vetterli. Filter banks allowing perfect reconstruction. *Signal Processing*, 10:219–244, 1986.

[57] Video Quality Experts Group (VQEG). Final report from the video quality experts group on the validation of objective models of video quality assessment. Technical report, VQEG, 2000.

[58] E. R. Vrscay. Mathematical theory of generalized fractal transforms and associated inverse problems. In *Proceedings of ImageTech 96 Conference on Multimedia Imaging Technology and Applications*, Atlanta, GA, 1996.

[59] E.R. Vrscay. A generalized class of fractal-wavelet transforms for images representation and compression. *Can. J. Elect. and Comp. Eng.*, 23(1-2):69–83, 1998.

[60] A. Watson and A.J. Ahumada. Model of human visual-motion sensing. *Journal of the Optical Society of America*, 2(2):322–341, 1985.

[61] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

Index