FRACTAL CODING OF DISPLACED FRAME DIFFERENCE SIGNALS

by

Nino Ferrario

B.A.Sc. Simon Fraser University, 1993

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE
in the School
of
Engineering Science

SIMON FRASER UNIVERSITY
July 1997

APPROVAL

Name: Nino Ferrario

Degree: Master of Applied Science

Title of thesis : FRACTAL CODING OF DISPLACED FRAME DIFFERENCE SIGNALS

Examining Committee: Dr. Colombo Bolognesi, Chairman

Dr. Jacques Vaisey
Associate Professor, Engineering Science, SFU
Senior Supervisor

Dr. Paul Ho
Associate Professor, Engineering Science, SFU
Supervisor

Dr. John Jones
Associate Professor, Engineering Science, SFU
Supervisor

Dr. Mehrdad Saif
Associate Professor, Engineering Science, SFU
Examiner

Date Approved: 17 / July / 97

Abstract

Much of the work on fractal video coding has focused on 3-D fractal block coding and coding of original frames using previously reconstructed frames and variations thereof. This thesis investigates the possibility of using fractal image coding techniques for direct coding of displaced frame difference signals. Both standard affine transform and orthogonal basis iterated function system (OBIFS) coders are considered. These coders are generalized for video coding by introduction of alternative domain pool sources. It is shown that the use of these alternative domain pool sources provides much better performance than classical fractal coding of the displaced frame difference signals alone. Extensive simulations are also performed to quantify the effects that various transformation parameters have on coding performance. Finally, the objective and subjective coding performance of the best affine transform and orthogonal basis IFS coders are compared to that of a reference discrete cosine transform (DCT) coder for two standard video sequences.

The simulation results indicate that affine-transform-based fractal coders are feasible for direct fractal coding of displaced frame difference signals. The peak signal-to-noise ratio (PSNR) performance of these coders was as good or better than the reference DCT-based coder for the two sequences tested. The OBIFS coders, however, are not feasible for direct fractal coding of displaced frame difference signals. The reference DCT-based coder provided much better performance for all sequences tested.

Acknowledgements

I would like to thank my senior supervisor, Dr. Jacques Vaisey, for his helpful suggestions, guidance and financial support throughout my research. In addition, I would like to thank Dr. Paul Ho and Dr. John Jones for serving on my supervisory committee. I am also indebted to the Natural Sciences and Engineering Research Council of Canada and Simon Fraser University for the financial support they have provided me during the course of my thesis work.

Contents

List of Figures

List of Abbreviations

BFOS	Breiman, Friedman Olshen and Stone
DCT	discrete cosine transform
DECIM_BY_AVG	decimation by averaging
DFD	displaced frame difference
HV	horizontal-vertical
IFS	iterated function system
JPEG	Joint Photographic Experts Group
KLT	Karhunen-Loeve transform
LIFS	local iterated function system
MC	motion compensation
MPEG	Moving Pictures Experts Group
NO_CONTRACTION	no contraction
OBIFS	orthogonal basis iterated function system
PCM	pulse code modulation
PCO	previously coded original frame
PCR	previously coded residual image
PSNR	peak signal-to-noise ratio
RBC	residual image being coded
SUBSAMP	subsampling

Chapter 1

Introduction

New digital applications and services are emerging all the time. These include high definition television (HDTV), videoconferencing, videotelephony, multimedia and improved and enhanced cable services. Accomodating these applications and services using digital information in its raw form would require the use of an extremely high bandwidth channel; especially in the case of images and video. Consequently, efficiently utilizing the communication channel bandwidth requires that the digital information be significantly compressed.

To this end, research in the areas of image and video compression has been ongoing and widespread. Currently, all the major image coding standards: the Joint Photographic Experts Group (JPEG) standard (Pennebaker and Mitchell 1993) for still images and the Moving Pictures Experts Group (MPEG) (LeGall 1991), H.261 (Liou 1991) and H.263 (Rijkse 1995) standards for full-motion video are based on the use of the discrete cosine transform (DCT). Nevertheless, new algorithms and improvements to existing algorithms are appearing all the time.

A particular coding technique which has now reached a certain level of maturity is fractal image coding. Work in this area was originally stimulated by Barnsley (1988).

Fractal image coders differ from traditional coding techniques in the manner in which they exploit redundancies in still images and video sequences. Classical fractal

image coders approximate an original image as the fixed point of a contractive transformation. Images are decoded by iterating the encoding transformation on any arbitrary initial image. Generalized fractal coders have also been developed which relax the contractivity constraint thus making higher fidelity encodings possible. Fractal image coders have shown good performance for coding of still images. This thesis investigates their use for video coding.

1.1 Thesis Objective

The objective of this work was to investigate the feasibility of using fractal image coding techniques for direct coding of displaced frame difference (DFD) signals. The work was probing and experimental in nature. The ultimate goal was to provide both quantitative and qualitative results on the ability of fractal image coders to directly code displaced frame difference signals. To this end, we focused on image sequences with moderate to relatively high motion so that the DFD signals contained enough energy to make the analysis useful and enlightening; the video coders implemented were designed with these moderate to high energy DFD signals in mind. The targeted bit rates were approximately 100 kbps (kilobits per second) and below; the actual bit rates obtained varied from approximately 20 kbps to 120 kbps depending on the video sequence.

Two classes of fractal coders were considered in the investigation: standard affine and orthogonal basis iterated function system (OBIFS) coders. The investigation was done without regard to encoder complexity but only in terms of objective and subjective performance measurements. The performance of the best affine transform and OBIFS coders was compared to that of a reference DCT-based coder using two standard video sequences.

1.2 Thesis Motivation

The two main reasons for undertaking this work were:

- Current video compression standards, MPEG, H.261 and H.263, employ hybrid coding techniques consisting of motion compensated prediction followed by discrete cosine transform coding of the resulting displaced frame difference signals. It is well known that the rate-distortion performance of the DCT is excellent for sources with high correlation coefficients. The DCT is used in the JPEG standard for this very reason; the inter-pixel correlation in a block of contiguous samples is generally high for still images. On the other hand, displaced frame difference signals typically have low correlation coefficients, but the DCT is still used to encode them.

  Fractal image coders have shown good rate-distortion performance for compression of still images. Some authors (Barthel et al. 1994) (Barthel and Voyé 1994) have reported results superior to JPEG over a wide range of bit rates for compression of the standard **Lena** image.

  Because of the good performance shown by fractal coders for coding of still images, we thought it both meaningful and instructive to investigate the performance of fractal coders when used for direct coding of DFD signals.

- All of the work to date on fractal video coding has focused on either 3-D fractal block coding or fractal coding of original frames using the previously reconstructed frame as a domain pool source and variations thereof. There have been, to the author's knowledge, no published results on direct fractal coding of displaced frame difference signals. This work thus contributes to the study of digital video coding using fractal techniques.

1.3 Thesis Contributions

The main contributions of this thesis can be summarized as follows:

- An empirical study on direct fractal coding of displaced frame difference signals is presented.
- Thorough descriptions and analysis of all transformation parameters are provided; the effects of all transformation parameters on coding performance are quantified.
- Both classes of coders considered are generalized for video coding by introduction of alternative domain pool sources. A scheme for efficient quantization and entropy coding of the resulting transformation parameters is presented. The resulting coders provide much better performance than classical fractal coding of the DFD signals alone.
- Objective and subjective performance results comparing the best affine transform and OBIFS coders to a reference DCT-based coder are presented.

1.4 Thesis Outline

Chapter 2 provides an overview of image and video coding fundamentals. The concept of redundancy is introduced and several common image compression techniques are described. Overviews of quantization, bit allocation and entropy coding are also presented along with detailed descriptions of their use in this work. The chapter concludes with a detailed discussion of motion compensation, a key component of all video coding systems implemented.

Chapter 3 outlines the mathematical theory underlying all classical fractal image coding algorithms. The fractal image model is detailed and relevant mathematics describing the image encoding and decoding procedure are provided. Finally, a discussion of practical fractal image coding algorithms for encoding and decoding real

world images is presented; current applications of these algorithms to video coding are described.

Chapter 4 presents a detailed description and analysis of all fractal video coding algorithms implemented in this work. All aspects of the coding algorithms are described. This includes encoder and decoder specifications, parameter specifications, quantization considerations, and descriptions of the output bit streams.

Chapter 5 presents extensive simulation results obtained by quantifying the effects of all transformation parameters (for both classes of coders) on coding performance; the **pongi** and **foreman** video sequences were used for these simulations. The best affine transform and OBIFS coders are selected for final comparison to a reference DCT-based video coding system.

Chapter 6 presents and discusses the simulation results obtained by comparing the performance of the affine transform and OBIFS coders selected from Chapter 5 to a reference DCT-based coder. The standard **carphone** and **salesman** video sequences are used for this comparison. A conclusion is drawn on the feasibility of using fractal image coding techniques for direct coding of DFD signals.

Chapter 7 summarizes the main contributions of this thesis and suggests possibilities for future work.

Chapter 2

Image and Video Coding Fundamentals

This chapter provides an overview of image and video coding fundamentals. The aim of the chapter is to acquaint the reader with many of the basic techniques used in image and video compression, many of which were used during the course of this work. As a secondary objective, the chapter provides a framework against which the fractal coding methods described later can be compared.

2.1 Redundancy

The goal of any data compression system is to find an efficient representation of a source signal, while at the same time maintaining the signal quality. To achieve such a goal, source coders attempt to eliminate or reduce source redundancy. This redundancy manifests itself in two distinct forms, statistical redundancy and subjective redundancy or irrelevancy (Netravali and Haskell 1988).

Statistical redundancy refers to information that can be removed from the source signal without introducing any loss of information. The original signal can be reconstructed exactly from the encoded bit stream. Source coders that remove only

statistical redundancy are called lossless coders.

Subjective redundancy or irrelevancy, deals with information that can be removed from the source signal without introducing any perceived distortion. Such information is deemed irrelevant since it is not perceivable by a human and therefore not needed to convey the signal information. In practical systems, source coders do introduce perceptable distortion; the objective being to minimize the amount of perceived distortion added for a given bit rate constraint.

The removal of subjective redundancy is a lossy process; information is lost and the original signal can no longer be reconstructed exactly. Source coders that remove subjective redundancy are called lossy coders.

The next section describes three common techniques for exploiting image redundancies.

2.2 Common Compression Techniques

2.2.1 Predictive Coding

Pulse code modulation (PCM) (Jayant and Noll 1984) is a very simple and basic compression technique. In a PCM system, each image sample is coded (quantized) independently of all other samples. Such a coding system is memoryless and does not exploit any correlations that exist between samples. Better performance can be achieved if the correlation between image samples is exploited during the coding process. Predictive coding (Gersho and Gray 1992) is a technique that exploits these inter-sample correlations.

Predictive coders attempt to minimize the amount of additional information required to specify each new image sample. This is done by coding (quantizing) only the difference between the current image sample and a prediction of the current sample based on past reconstructed samples. The prediction is usually formed as a weighted linear combination of image samples in the immediate vicinity of the current sample

being processed. The difference or prediction error represents the additional information needed to specify the new image sample. Because images are locally highly correlated, the sequence of prediction errors will generally have a lower variance than the original image samples. This implies that at the same rate, a lower distortion can be achieved, or conversely, for the same level of distortion, a lower rate is required.

2.2.2 Transform Coding

Transform coders (Clarke 1985), like predictive coders, exploit the inter-pixel correlation that exists between image samples. In a typical transform coding system, the input image is partitioned into rectangular blocks of dimension $M \times N$ where usually $M = N = 8$. Each block is then transformed via a unitary transform to produce a set of $M \times N$ transform coefficients. The i'th coefficients from each block are then grouped to form a set of $M \times N$ sources. Some type of bit allocation algorithm is then used to allocate a fixed quota of bits amongst the $M \times N$ sources. Quantizers are then designed at the corresponding rates and the coefficients are quantized to produce an encoded bit stream. At the decoder, the inverse transform is applied to the quantized coefficients to produce a reconstruction of the original image.

The objective of transform coding is two-fold. First, it is desired to concentrate most of the block energy in as few transform coefficients as possible, thereby minimizing the number of coefficients that have to be quantized. Compaction of the block energy also allows the coefficients to be ordered according to the contribution each makes to the total block energy. During the bit allocation process, more bits can be allocated to those coefficients that contribute most to the total energy while fewer bits are allocated to the less energetic coefficients; allocating more bits to the quantization of more energetic coefficients yields a lower overall average distortion. The second goal of transform coding is the decorrelation of block image samples. It is desirable to remove all of the correlation that exists between image samples in a block and produce a set of uncorrelated transform coefficients. A transform achieving

these two objectives will produce a set of transform coefficients that if quantized at a given rate, R, will yield a lower overall average distortion than the original image samples quantized at the same rate.

For stationary sources, the Karhunen-Loève transform (KLT) is the unique, optimal transform that achieves both of the above objectives; the basis vectors of the KLT are the eigenvectors of the source covariance matrix. The KLT is not, however, a practical transform for image coding because it is image-dependent and must be re-computed for every image due to variations in image statistics. This source dependency also implies that the eigenvectors must be stored or transmitted to the decoder. The KLT is not widely used for image coding applications; the discrete-cosine transform (DCT) is used extensively instead.

The DCT is a data-independent transform whose performance approaches that of the KLT for first-order autoregressive sources with correlation coefficients approaching one (Jayant and Noll 1984). The DCT performs very well in still image coding applications because still images[1] can be locally modeled by such sources. For this reason, as well as the existence of many fast algorithms for its computation, the DCT has been made an integral part of all the current still image and video coding standards.

For an image block, $f(i,j)$, with dimensions $N \times N$, the two-dimensional DCT, $F(u,v)$, $u,v = 0,1,\ldots,N-1$ is given by:

$$F(u,v) = C(u)C(v)\,\frac{2}{N}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f(i,j)cos(\frac{\pi u(2i+1)}{2N})cos(\frac{\pi v(2j+1)}{2N})$$

where

$$C(m) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } m = 0 \\ 1 & \text{otherwise} \end{cases}$$

[1]This is not the case for the displaced frame difference signals that are produced by motion compensation.

2.2.3 Subband Coding

Subband coding (Jayant and Noll 1984) (Gersho and Gray 1992) is a more general form of transform coding. However, subband coders differ from transform coders in that subband coders filter the entire image being coded rather than simply filtering $N \times N$ image blocks. This is advantageous in that the reconstructed images are smooth without any blocking artifacts. In general, both coding methods can be classified as signal decomposition techniques as they both decompose the source signal into a set of individual components each of which is coded separately.

Subband coders split the two-dimensional image frequency spectrum into separate frequency bands. They do this by passing the original image through a bank of linear filters to produce a set of subimages. Like transform coding, it is desired to have energy compaction and the ability to code the subbands separately; the number of bits allocated to quantizing each subimage can then be adapted according to the perceptual importance of the subimage.

Still images have most of their total energy concentrated in the low-pass subimages. The remaining subimages contain additional high-frequency information needed to reproduce the original image exactly. The unequal distribution of subimage energies allows the subimages to be prioritized so that more bits are allocated to the coding of high energy subimages while fewer bits are allocated to the lower energy subimages. This prioritization is useful since it is generally the case that the highest energy subimages are the most perceptually important and therefore should be coded at the highest rate; in some cases, the low energy subimages need not be coded at all.

The current state-of-the-art still-image coding algorithm is an algorithm based on subband coding techniques (Said and Pearlman 1996). This algorithm is based on embedded zero-trees of wavelet coefficients (Shapiro 1993). It has shown rate-distortion performance far superior to JPEG at lower rates.

2.3 Quantization

The previous section described three common image compression techniques. The outputs of all these algorithms are sets of real numbers that must be discretized if they are to be transmitted or stored digitally. The process of discretizing a set of real numbers is called quantization.

2.3.1 Scalar Quantization

A scalar quantizer (Gersho and Gray 1992), Q, is a many-to-one mapping from $\Re \rightarrow C = \{y_1, y_2, \ldots, y_N\}$, the y_i's are called output points. A scalar quantizer, Q, assigns to each $x \in \Re$ an element y_i from the set C. The index i of the selected output point is transmitted or stored. For a quantizer, Q, to be optimal, it must satisfy both the nearest neighbor condition and the centroid condition.

Given the set C of output points, the nearest neighbor condition governs the assignment of an output point y_i to an input point x. It states that:

$$Q(x) = y_i \quad \text{if} \quad d(x, y_i) < d(x, y_j) \quad \forall \; j = 1, \ldots, N, \quad j \neq i$$

Here $d(x, y_i)$ is the distortion incurred by representing the input point x by the output point y_i. The nearest neighbor condition defines an optimal partitioning of the real line for the given set C in the sense of minimizing the overall average distortion.

The centroid condition governs how the output points are selected given a partition of the real line. Let $\boldsymbol{X}$ be a continuous random variable that we wish to quantize. Given a partition of the real line into intervals $R_i = (x_{i-1}, x_i]$, $i = 1, \ldots, N$, the output points, y_i are given by:

$$y_i = \text{centroid}(R_i) = \min_y E[d(\boldsymbol{X}, y) | \boldsymbol{X} \in R_i]$$

This equation states that the optimal output point y_i is that value of $y \in \Re$ that minimizes the average distortion, $E[d(\boldsymbol{X}, y)]$, for $\boldsymbol{X} \in R_i$. For the special case where $d(.,.)$ is the mean-squared error distortion measure, the output points are given by:

$$y_i = E[\, \boldsymbol{X} | \, \boldsymbol{X} \in R_i]$$

The centroid condition defines an optimal set C of output points for a given partition of the real line in the sense of minimizing the overall average distortion. The intervals $(x_{i-1}, x_i]$ are called decision regions or nearest neighbor cells.

There are two basic types of scalar quantizers, uniform and non-uniform. Uniform quantizers are characterized by decision regions of constant width, Δ. Uniform quantizers can further be classified as midtread or midrise. Midtread quantizers have a zero output level while midrise quantizers do not. Non-uniform quantizers have decision regions of varying widths with regions of high probability mass consisting of many short cells and regions of low probability mass consisting of a few wider cells.

In designing a scalar quantizer to discretize a continuous random variable $\boldsymbol{X}$, we have to find the output points y_i and the decision regions $(x_{i-1}, x_i]$ so that the overall average distortion, $D = E[d(\, \boldsymbol{X}, Q(\boldsymbol{X}))]$, is minimized subject to a constraint on the number of output levels N or the entropy of the output indices i. A quantizer, Q, that minimizes the overall average distortion subject to a constraint on the number of output levels, N, is called a Lloyd-Max quantizer. Such a quantizer is optimal in the sense that no other quantizer, Q, with N levels or fewer can yield a lower overall average distortion. However, there may be another quantizer with lower output entropy that can yield a lower average distortion. For this to be possible, the entropy of the output indices from the Lloyd-Max quantizer must be greater than the maximum entropy as specified by the entropy constraint. It has been shown that if a constraint is placed on the output entropy of the quantization indices, a uniform quantizer with a large number of output levels is nearly optimal (Farvardin and Modestino 1984).

2.3.2 Vector Quantization

Unlike scalar quantizers which quantize individual samples, vector quantizers (VQ's) (Gersho and Gray 1992) quantize a vector of samples. A vector quantizer, Q, is a

many-to-one mapping from $\Re^k \rightarrow C = \{\boldsymbol{y_1}, \ldots, \boldsymbol{y_N}\}$, $\boldsymbol{y_i} \in \Re^k$. The set, C, is called a codebook and the $\boldsymbol{y_i}$'s are called code vectors. A vector quantizer, Q, assigns to each input vector, $\boldsymbol{x} \in \Re^k$, an element $\boldsymbol{y_i}$ from the set C. The index i of the selected code vector is transmitted or stored.

As is the case for scalar quantizers, necessary conditions for the optimality of a vector quantizer, Q, are the nearest neighbor condition and the centroid condition. These conditions are just straightforward generalizations of the one-dimensional case and will not be stated here.

Vector quantizers, like scalar quantizers, are designed in practice using a vector generalization of Lloyd's iterative design algorithm (Gersho and Gray 1992).

2.4 Entropy Coding

The goal of entropy coding (Gersho and Gray 1992)(Jayant and Noll 1984) is to minimize the amount of information required to represent a discrete data source such as quantizer output indices. The process of entropy coding is lossless so the original data set can be reconstructed exactly.

Let $\boldsymbol{X}$ be a discrete random variable that can assume any value from the set $S = \{0, 1, \ldots, N-1\}$ where N is finite. Furthemore, let $p_i = P[\boldsymbol{X}{=}X_i]$, $X_i \in S$. The zero'th order entropy of $\boldsymbol{X}$, denoted by $H(\boldsymbol{X})$, is defined as:

$$H(\boldsymbol{X}) = -\sum_{i=0}^{N-1} p_i \log_2 p_i$$

$H(\boldsymbol{X})$ is a measure of the average amount of information contained in $\boldsymbol{X}$. More precisely, $H(\boldsymbol{X})$ represents the minimum number of bits required to losslessly encode $\boldsymbol{X}$, if each symbol is coded independently. Huffman coding (Gersho and Gray 1992) is the most widely used lossless coding technique.

The basic idea behind Huffman coding is to assign symbols that occur with high probability to short codewords and symbols that occur with low probability to long

codewords in such a way that the average codeword length in bits/symbol is minimized. The codewords are selected in such a way that no codeword is the prefix of any other codeword; this ensures that the code is uniquely decodable. It can be shown that using Huffman coding on individual symbols, the average codeword length will lie within 1 bit of the zero'th order entropy of the source. The average codeword length (in a binary code) will be equal to $H(\boldsymbol{X})$ only if all the p_i are powers of 1/2.

In any practical implementation of a Huffman coder, the encoder and decoder must be synchronized. This means that both the encoder and decoder must have a copy of the same Huffman table. Synchronization can be achieved in several ways. First, a fixed Huffman table can be used for all source signals so that a Huffman table need only be generated once and stored at the decoder. This method will work fine as long as the statistics of the data being coded do not change very much. Secondly, if the number of symbols is not too large, frequency information specifying the number of times each symbol occurs can be sent to the decoder. Finally, the probabilities of occurrence of each symbol can be approximated using a model of the source data and a Huffman code designed based on this model. The parameters of the model can be sent to the decoder with minimal overhead information, and the decoder can reconstruct the same Huffman table as was used at the encoder. Of course, for this method to work, accurate models of the source data are required. This latter approach was adopted in this work, and thus its use will be detailed in Section 2.4.1.

A second technique used to minimize the amount of information required to represent a discrete data set is run-length coding (Jayant and Noll 1984). Run-length coding exploits the inter-sample correlation that exists between elements of a sequence of discrete data samples. For example, denote by $\{X_i\}_{i=1}^{N}$ a sequence of discrete data samples. Instead of coding each X_i independently, run-length coders exploit consecutive repetitions (runs) of the same symbol by specifying the length of the run followed by the symbol in the run. Run-length coding is used in transform coding systems to exploit runs of 0's that occur after quantization of the transform coefficients.

2.4.1 Data Modeling

It was stated in Section 2.4 that a model of the source data could be used to synchronize the encoder and decoder with minimal overhead information in practical applications of Huffman coding. The model models the probability distribution of the real-valued data to be quantized and entropy coded; it allows the probabilities needed by the Huffman coding algorithm to be estimated and a Huffman code designed based on these probabilities. We now describe in detail the use of this model.

The following notation will be used in this discussion:

- $\{X_i\}_{i=1}^{M}$ will denote a set of real numbers corresponding to a set of data that must be quantized and entropy coded using a Huffman code; without loss of generality, it is assumed that the mean, μ, of the original data set has already been removed so that $\sum_{i=1}^{M} \frac{X_i}{M} = 0$.
- $\mu_{|x|} = \sum_{i=1}^{M} \frac{|X_i|}{M}$ will denote the mean-absolute-value of the data set.
- σ_x will denote the standard deviation of the data set.
- $(x_{i-1}, x_i], \quad i = 1, \ldots, N$ will denote the decision regions of a quantizer, Q, to be used for quantizing the data set.

Consider the probability density function (pdf) of a continuous random variable $\boldsymbol{X}$ with zero mean and standard deviation σ defined by:

$$p(x) = [\frac{\nu\,\eta(\nu,\sigma)}{2\Gamma(\frac{1}{\nu})}]\, e^{-(\eta(\nu,\sigma)|x|)^{\nu}} \tag{2.1}$$

where

$$\eta(\nu,\sigma) = \frac{1}{\sigma}[\frac{\Gamma(\frac{3}{\nu})}{\Gamma(\frac{1}{\nu})}]^{\frac{1}{2}} \tag{2.2}$$

and

$$\Gamma(\,x\,) = \int_0^{\infty} t^{x-1}\, e^{-t}\, dt. \tag{2.3}$$

$p(x)$ is called a generalized Gaussian pdf. It is completely parameterized by ν and σ. Figure 2.1 displays $p(x)$ for different values of ν ($\sigma = 1.0$).

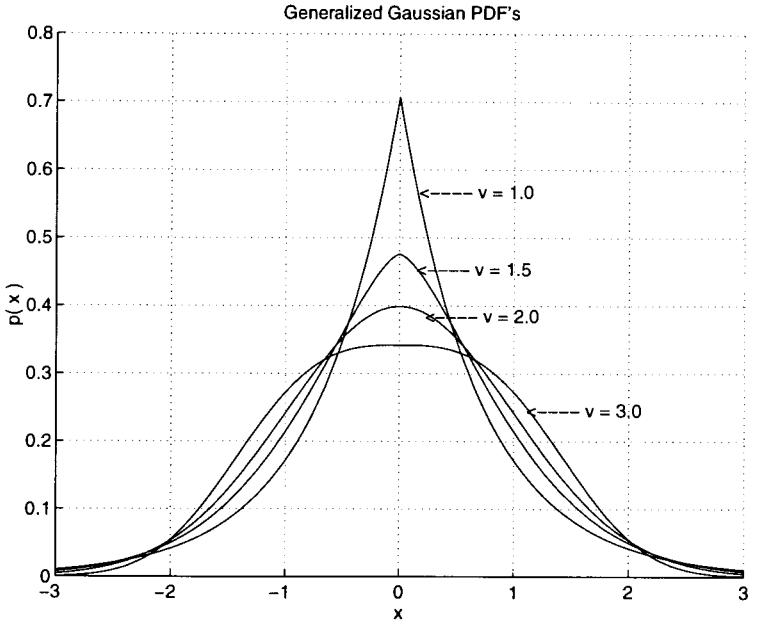

Figure 2.1: Generalized Gaussian PDF

It can be seen from the figure that as ν gets larger the pdf becomes broader; as ν gets smaller the pdf becomes peakier. Two special cases occur when $\nu = 1.0$ (Laplacian density) and $\nu = 2.0$ (Gaussian density). Because the pdf shape can be controlled by changing ν, the pdf can be used to model the distributions of source data that have the general shapes shown in Figure 2.1. Published work (Birney and Fischer 1995) describes the use of the generalized Gaussian pdf for modeling of DCT and subband image data for the purposes of quantization and entropy coding. Given a data set $\{X_i\}_{i=1}^{M}$, $p(x)$ will model the data set in the sense that the mean-absolute-value and variance computed using $p(x)$ will equal the mean-absolute-value and variance of the actual data set.

For a given set of data $\{X_i\}_{i=1}^{M}$, the corresponding $p(x)$ is found by solving (Birney and Fischer 1995)

$$\nu = F^{-1}[\frac{\mu_{|x|}}{\sigma_x}] \tag{2.4}$$

where

$$F(\alpha) = \frac{\Gamma(\frac{2}{\alpha})}{\sqrt{\Gamma(\frac{1}{\alpha})\Gamma(\frac{3}{\alpha})}}. \tag{2.5}$$

Solving Eq. 2.4 directly is very difficult; in practice, we can use a lookup table of ν values, $\{\nu_i\}_{i=1}^{L}$, and find ν_i such that

$$|F(\nu_i) - \frac{\mu_{|x|}}{\sigma_x}| \tag{2.6}$$

is minimized. A 2^{16}-point lookup table of ν values spanning 0.1 to 5.0 was used in this work.

We now have a function $p(x)$ modeling the probability distribution of our data set. However, we want to design a Huffman code for the output indices of the quantizer, Q, used to quantize our data set; therefore, we need to know the probability of occurrence of each quantizer output index. These probabilities can be estimated using the model by numerically integrating $p(x)$ over the decision regions $(x_{i-1}, x_i]$, $i = 1, \ldots, N$ of the quantizer. It will be shown later in this thesis that the resulting Huffman code designed using these estimated probabilities is nearly optimal; the difference in the output rate compared to a Huffman code designed using the actual quantizer output index probabilities is negligible. Also, the only information needed by the decoder to reconstruct the same Huffman code is the index i of the optimal ν_i and the standard deviation of the data set.

2.5 Bit Allocation

The bit allocation problem is one of allocating a fixed quota of bits amongst a set of N sources. Many different algorithms are available to perform this bit assignment

including the Breiman, Friedman, Olshen, and Stone (BFOS) (Riskin 1991) bit allocation algorithm, the Greedy algorithm (Gersho and Gray 1992) and allocations based on high rate assumptions (Gersho and Gray 1992). Here we describe the BFOS bit allocation algorithm as it was used in this work.

The basic bit allocation problem can be formulated as follows:

- Let the N sources be represented by $\boldsymbol{X}_i$, $i = 1, \ldots, N$ and let $|\boldsymbol{X}_i|$ denote the number of data points in the set $\boldsymbol{X}_i$.
- Let R be the target rate (bits/data point) specified for quantization of the $\boldsymbol{X}_i$.
- Let Q_j^i be the set of quantizers used to quantize $\boldsymbol{X}_i$, $i = 1, \ldots, N$, $j = 1, \ldots, M_i$.
- Let $B = \sum_{k=1}^{N} p_k b_k$ where $p_k = \frac{|\boldsymbol{X}_k|}{\sum_{j=1}^{N} |\boldsymbol{X}_j|}$ and b_k is the average rate in bits/data point required to quantize the set $\boldsymbol{X}_k$. B depends on the quantizers selected to quantize each $\boldsymbol{X}_k$.
- Let $D = \sum_{k=1}^{N} p_k d_k$ where p_k is defined as above and d_k is the average distortion incurred in quantizing data points in the set $\boldsymbol{X}_k$ using a given quantizer. D also depends on the quantizers selected to quantize each $\boldsymbol{X}_k$.

The bit allocation problem is to minimize D subject to the constraint that $B \leq R$. The solution provided by the BFOS algorithm is as follows:

The BFOS algorithm is given as input a rate-distortion table for each of the N sources. For source $\boldsymbol{X}_i$, this table contains M_i entries and specifies the average rate and distortion incurred in quantizing $\boldsymbol{X}_i$ using Q_j^i, $j = 1, \ldots, M_i$. With N sources and M_i quantizers per source, there are a total of $\prod_{i=1}^{N} M_i$ different quantizer assignments each yielding a different overall average distortion, D_j, and average rate, R_j, $j = 1, \ldots, \prod_{i=1}^{N} M_i$. These points (R_j, D_j) comprise the operational distortion rate function of the N sources. The BFOS algorithm traces out the convex hull of the operational distortion rate function. For a given target rate, R, it assigns quantizers

to each source such that $B \leq R$, and this allocation is such that no other allowable quantizer assignments with $B \leq R$ will yield a lower overall average distortion.

2.6 Motion Compensation (MC)

A digital video signal can be viewed as a sequence of still images called frames. Because successive frames typically exhibit high temporal correlation, instead of coding each frame independently (intraframe coding), only the difference between the current frame and a prediction of the current frame formed from the previous frame needs to be coded (interframe coding). The underlying assumption is that the difference (residual) image will have a lower information content and thus can be coded with fewer bits. The problem then becomes one of identifying those regions in the previous frame (or frames) that most closely resemble the regions in the current frame which are to be coded. Motion compensation is a procedure that accomplishes this objective.

2.6.1 Standard Block Motion Compensation

Standard block motion compensation computes both motion vectors and residual images. Block motion compensation is basically a block-matching process. It is based on the assumption that all pixels within a block move with the same translational motion so that each block in the current frame is just a translated version of some block in the previous frame. This assumption is generally not valid, and hence there will often be a significant difference between a block in the current frame and its optimal prediction based on a block from the previous frame. These differences or prediction errors comprise the additional information required to specify the current block.

The standard block motion compensation procedure is as follows. The n'th frame

in a digital video sequence is partitioned into non-overlapping square blocks of dimension $N \times N$[2]. Refering to Figure 2.2[3], let Ω_n and Ω_{n-1} be the n'th and $(n-1)$'th frames in the digital video sequence. Furthermore, let the block labelled **a** in the n'th frame be the block currently being processed. Assume this block is centered at pixel coordinates (i, j). The previous frame, Ω_{n-1}, is searched for the $N \times N$ block that matches block **a** most closely. The quality of the match is usually measured using either the mean squared error or the mean absolute error distortion measure. Once the optimal matching block has been found, block **b** in the figure, the residual block and motion vector corresponding to block **a** can be computed.

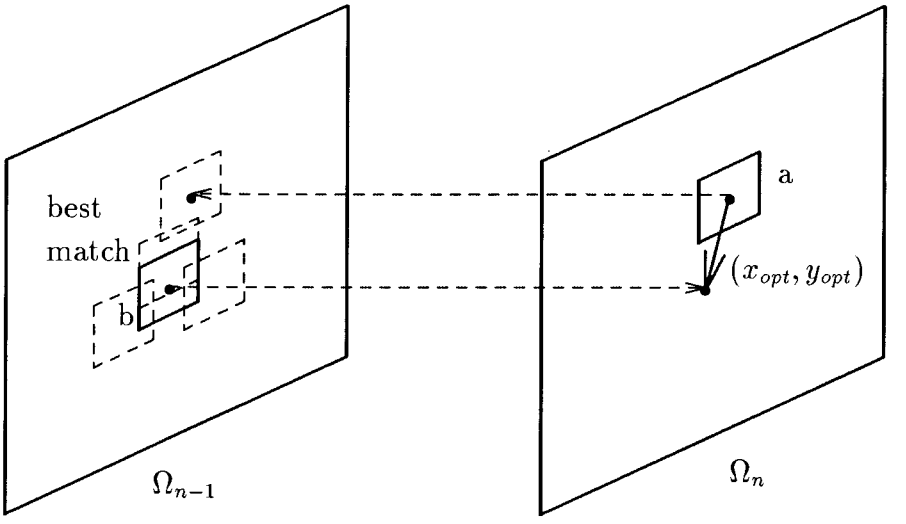

Figure 2.2: Standard Block Motion Compensation

If we assume that block **b** is centered at pixel coordinates (m,n), the motion vector is computed as:

$$x_{opt} = m - i, \; y_{opt} = n - j$$

This vector represents the displacement from the block being processed to the optimal matching block in the previous frame. The residual or displaced frame difference (DFD) is then computed as the pixel to pixel difference between the block being

[2] N will be called the MC block size
[3] Figure courtesy of Jacques Vaisey.

processed (block **a**) and the optimal matching block (block **b**). These motion vectors and residual blocks are computed for each $N \times N$ block in the partitioned frame.

The set of all motion vectors for blocks in Ω_n is called the motion field for frame n. The image consisting of all the optimal matching blocks for Ω_n is called the prediction image.

In practice, the search region over which a matching block in the previous frame is sought is limited to a windowed region surrounding the block being processed. In the case of the above example, an optimal matching block might be sought for all offset vectors (x, y) such that $-N \leq y \leq N$ and $-N \leq x \leq N$;i.e., for each (x, y), the difference between $\Omega_n(i, j)$ and $\Omega_{n-1}(i + x, j + y)$ would be computed. This type of search strategy is known as full search; it was the search strategy used throughout this work. Other search strategies exist in the literature that provide tradeoffs between search time and the quality of matching blocks found (Liu and Zaccarin 1993).

2.6.2 Overlapped Windowed Block Motion Compensation

The previous section described standard block motion compensation. It was seen that standard block motion compensation is a form of memoryless predictor in which each block in the current partitioned frame, Ω_n, is predicted by a block in the previous frame. The prediction of each block is memoryless; it is done independently without regard to neighboring blocks. Consequently, if two neighboring blocks have differing motion vectors, there will be a discontinuity in the pixel intensities at the boundary of these two blocks in the prediction image. These discontinuities also appear in the DFD signal which is to be coded. If block coding algorithms are used to code the DFD signal, the discontinuities will not present a problem so long as the block size used is the same as the MC block size. If these block sizes are different, the discontinuities will appear within the block to be coded. Such discontinuities introduce high-frequency energy into the block and can result in decreased coding performance (Ohta and Nogaki 1993). This is particularly relevant to fractal coding as fractal coders make

use of spatially contracted domain blocks when coding an image in order to exploit the inter-pixel correlation that exists between blocks of image samples at different scales. These domain blocks are usually larger than the MC block size and will therefore contain the discontinuities. Since fractal coders approximate blocks of contiguous image pixels using suitably transformed domain blocks, a discontinuity in the middle of a domain block will certainly adversely affect the quality of the approximation.

A new technique called overlapped windowed motion compensation (Auyeung et al. 1992) (Orchard and Sullivan 1994) (Watanabe and Singhal 1991) has been introduced to try and improve coding performance when there is such a motion edge inside of a block. This process is similar to the standard block motion compensation procedure with the exception that enlarged optimal matching blocks are windowed and overlapped in forming the prediction image. Overlapped windowed motion compensation, as implemented in this thesis, can be formulated as follows: As in standard block motion compensation, the current frame, Ω_n, is partitioned into non-overlapping square blocks of dimension $N \times N$. The optimal matching blocks for each of the $N \times N$ blocks are found in exactly the same manner as for the standard block motion compensation case, and the motion vectors are computed in the same way. The difference lies in the formation of the prediction image. Consider once again Figure 2.2 and suppose that block **b** is the optimal $N \times N$ matching block for block **a**. Let block **c** be the $2N \times 2N$ block consisting of block **b** and all pixels within $\frac{N}{2}$ samples of block **b**. Block **c** is represented in Figure 2.3 by the dotted line; block **b** is represented by the solid line.

Note that if there are not $\frac{N}{2}$ image samples around block **b**, i.e., if block **b** is located near an image border, the corresponding pixels in block **c** are set to zero.

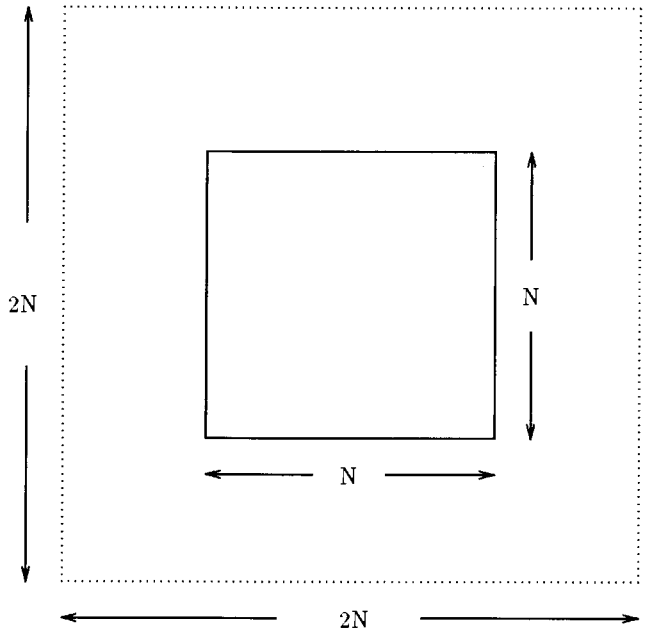

Figure 2.3: Example of Enlarged Optimal Matching Block

Block **c** is then windowed using a window function $w(x, y)$ and placed in the prediction image so that block **b** overlays block **a**. A typical window function is the $\sin^2$ window given below and shown graphically in Figure 2.4.

$$w(x,y) = \sin^2 \frac{\pi(x+0.5)}{N} \sin^2 \frac{\pi(y+0.5)}{N}, \quad x, y = 0, \ldots, N-1$$

The $\sin^2$ window satisfies the perfect reconstruction condition. This means that if blocks of constant pixel intensity are windowed and summed as to be described shortly, the resulting block of pixels will remain at the same intensity.

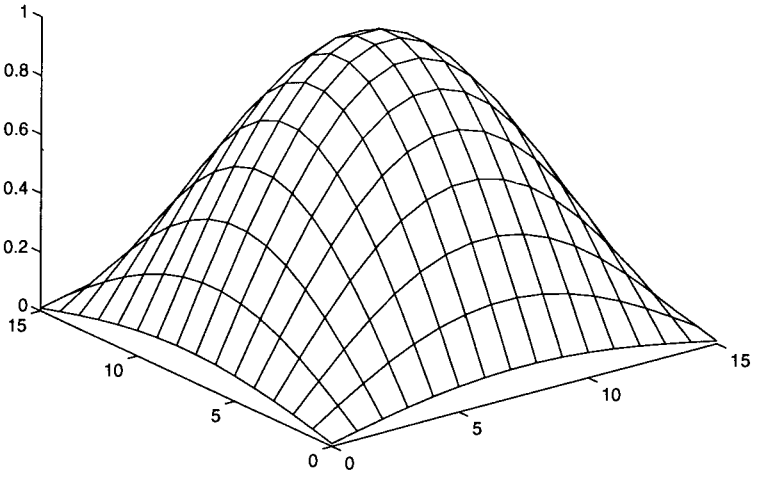

Figure 2.4: Sin^2 Window

Because the enlarged windowed optimal matching blocks are used in forming the prediction image, the blocks will be overlapping when placed in the prediction image in the manner described above. Pixels in the overlapped regions are summed producing an averaging effect and thereby reducing discontinuities at the $N \times N$ block boundaries. With the exception of $N \times N$ blocks on the borders of the image, all pixels in the prediction image are generated by summing the weighted contributions of four pixels. These pixels come from the enlarged windowed optimal matching block of the $N \times N$ block in which the pixel lies and the enlarged windowed optimal matching blocks of three neighboring $N \times N$ blocks.

As an implementation issue, portions of the enlarged windowed optimal matching blocks corresponding to pixels $\frac{N}{2}$ samples or less from the image borders of the prediction image are not windowed because these pixels do not possess enough neighboring blocks to form a proper pixel average.

Published results (Auyeung et al. 1992) indicate that overlapped windowed block motion compensation not only computes smoother prediction images than the standard block motion compensation technique, but the resulting DFD signals also generally have lower energies. Figures 2.5 and 2.6 depict two DFD signals, one computed using overlapped windows and the other computed using the standard block motion compensation technique. It is evident that the overlapped windowed block motion compensation technique produces a much smoother DFD signal with lower average energy per pixel (57.2 versus 66.9).

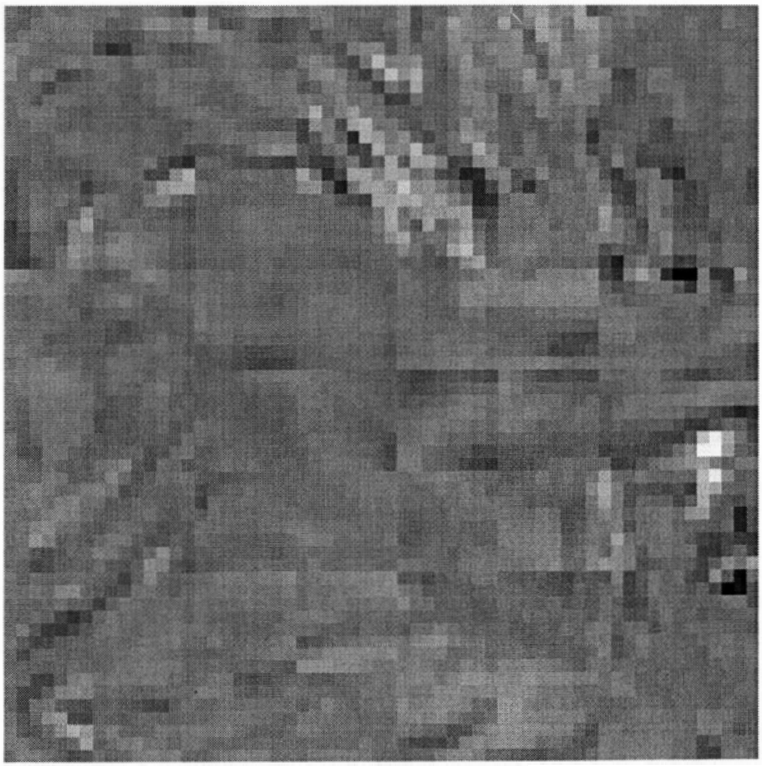

Figure 2.5: DFD Signal: Standard Block Motion Compensation

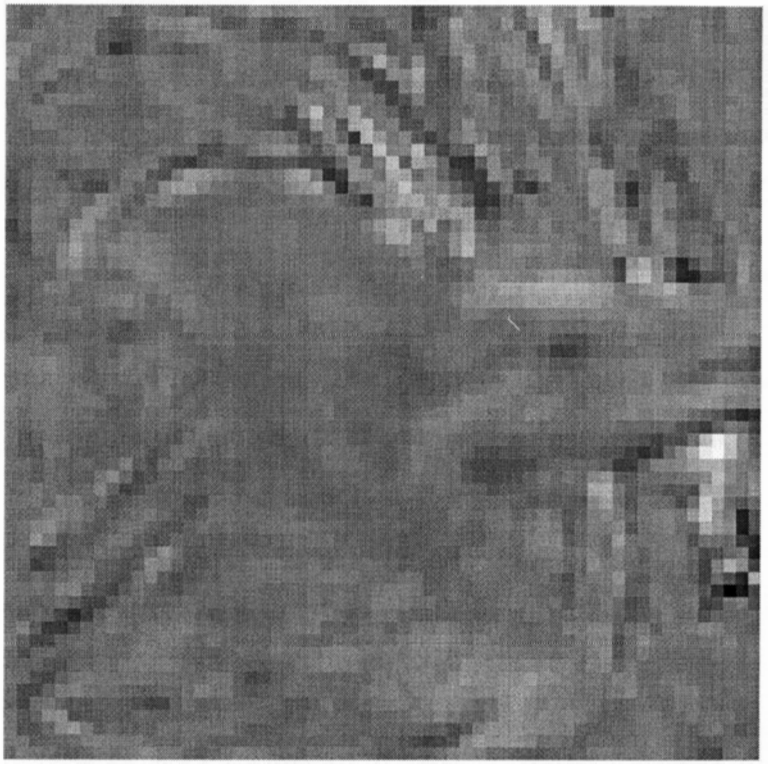

Figure 2.6: DFD Signal: Overlapped Windowed Block Motion Compensation

2.7 Performance Measures

The two most commonly used objective performance measures are the mean squared error (MSE) and the mean absolute error (ABS). The mean squared error between two images Ω_1 and Ω_2 is defined as

$$d_{MSE}(\Omega_1, \Omega_2) = \frac{1}{MN} \sum_{x=1}^{N} \sum_{y=1}^{M} [\Omega_2(x,y) - \Omega_1(x,y)]^2,$$

where $\Omega_1(x,y)$ and $\Omega_2(x,y)$ are the pixel intensities at spatial coordinates (x,y) in images Ω_1 and Ω_2 respectively and both images are of dimension $M \times N$.

The peak signal-to-noise ratio (PSNR) is often used to express the mean squared error logarithmically. The PSNR is defined as

$$PSNR = 10 \log \frac{(2^r - 1)^2}{d_{MSE}},$$

where r is the image pixel resolution. The PSNR is used because it is independent of the energy in the original image and thus provides a normalized measure that makes comparison of PSNR measurements across images meaningful. Such a measure is desirable because different images have different associated energies.

The mean absolute error is defined as

$$d_{ABS}(\Omega_1, \Omega_2) = \frac{1}{MN} \sum_{x=1}^{N} \sum_{y=1}^{M} |\Omega_2(x,y) - \Omega_1(x,y)|, \tag{2.7}$$

where Ω_1, Ω_2, M and N are defined as above.

Objective performance measures such as the ones described above do not always correlate well with perceived image quality. Subjective tests should also be performed when comparing compression algorithms as the perceived effects of PSNR differences are in general image-dependent. Typically, a 1 dB PSNR difference is perceivable by a human viewer.

Chapter 3

Fractal Image Coding

The previous chapter described several popular image compression techniques and the manner in which they exploit image redundancies. This chapter presents the theoretical foundation upon which classical fractal image coding techniques are based.

Classical fractal image coders exploit correlations that exist between blocks of image samples at different scales. Fractal coders assume that images belong to a special class of images characterized by a fractal model. The images in this class all possess the property of self-similarity or more generally, piecewise self-transformability. This class of images is uniquely associated with a class of non-linear contractive transformations in the sense that each transformation uniquely specifies an image in the class.

Under the assumption that images to be encoded can be satisfactorily described by a fractal model, the fractal image coding problem is one of finding that image in the class that most closely resembles the image to be encoded. The transformation associated with the best class image is stored as a representation of the original image. If the information required to completely specify the encoding transformation is less than the raw information content of the original image, the image has been effectively compressed. The encoding transformation is called a fractal code.

Fractal source coders are lossy. The reconstructed image is only an approximation

of the original image. The quality of the approximation is dependent on how well the original image is characterized by the fractal model.

We begin this chapter by describing the mathematics of classical fractal source coding. The chapter concludes with an overview of practical fractal image coding algorithms.

3.1 Mathematical Background

This section defines concepts and notation that are required for a logical and thorough treatment of classical fractal image compression. Excellent treatments of this material can be found in (Fisher 1995a) (Barnsley 1988) (Barnsley and Hurd 1993) (Lundheim 1995).

The first concept needed is the notion of a metric space. Metric spaces are important because real world images are elements of such a space.

Definition 1 (Metric Space) *A metric space, (X, d) consists of a set X and a real-valued function d called a metric*[1] *that defines the distance between any two points $x, y \in X$.*

The points x and y refered to in this definition depend on the nature of the space X. For example, if $X = \{x \mid 0 \leq x \leq 1\}$, then points in X can be individual real numbers in $[0, 1]$ or they may be subsets of X (i.e., intervals of the form $[a, b]$ or (a, b) where $a \geq 0$, $b \leq 1$, and $a \leq b$). As an example, if the function $d : X \times X \rightarrow \Re$ is defined by $d(x, y) = |x - y|$, then the pair (X, d) is a metric space.

Transformations defined on a metric space need to be constructed to encode real world images.

Definition 2 (Contractive Transformation) *Let (X, d) be a metric space. Let τ be a transformation defined on X. τ is called a contractive transformation if there exists a positive constant, $s < 1$ such that*

[1] Technically, to be a metric, d must satisfy certain axioms (Barnsley 1988).

$$d(\tau(x), \tau(y)) \leq s \cdot d(x, y) \quad \forall \; x, y \in X \; . \tag{3.1}$$

s is called the contraction factor of τ.

In a more general sense, the smallest $s > 0$ satisfying Eq. 3.1 is called the Lipschitz constant of τ.

If a transformation τ is expressed as the composition of two or more transformations, the contractivity of τ is just the product of the contractivities of each of the transformations in the composition. For example, if $\tau = w_1 \circ w_2$ and s_1 and s_2 are the contractivites of w_1 and w_2 respectively, the contractivity of τ is $s_1 s_2$. This property is important in that even though individual transformations in the composition may not be contractive, the overall transformation will be contractive if the product of the individual Lipschitz constants is less than one.

Intuitively, contractive transformations reduce the distance between points in a metric space X. For example, if a metric space (X, d) is defined by

$$X = \{(x, y) \mid 0 \leq x \leq 1, \; 0 \leq y \leq 1\}$$

$$d\,(\,(x_1, y_1),\,(x_2, y_2)\,) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

and

$$\tau \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

then

$$d\,(\tau(x_1, y_1), \tau(x_2, y_2)) \leq 0.5 \cdot d\,(\,(x_1, y_1),\,(x_2, y_2)\,) \; .$$

In this example, τ is a contractive transformation with contractivity 0.5.

The transformation τ defined above is an example of an affine transformation in $\Re^2$. In general, affine transformations can be defined in any n-dimensional space.

Definition 3 (Affine Transformation) *An affine transformation, w, is a function that consists of a linear term* **A** *and a translational term* **b**. *Mathematically,*

$$w(x) = \mathbf{A}x + \mathbf{b} \tag{3.2}$$

where x is a point in the domain of w.

For example, if $w : \Re^3 \rightarrow \Re^3$, then **A** would be a 3×3 matrix and **b** would be a 3×1 column vector. Affine transformations can be used to rotate, reflect, scale, skew, or move a set of points within a space X. Affine transformations are very common in fractal image compression.

For the purposes of image compression, transformations formed from the union of individual contractive transformations are required. This union of transformations forms an iterated function system.

Definition 4 (Iterated Function System) *Let (X, d) be a complete[2] metric space. Let $w_i : i = 1, \ldots, N$ be a set of contractive transformations defined on X. Let $s_i : i = 1, \ldots, N$ be the contractivities of $w_1, w_2, \ldots, w_N$ respectively. Then, the metric space X along with the $w_i : i = 1, \ldots, N$ form an iterated function system (IFS).*

A transformation W which maps subsets of X into subsets of X can be associated with any IFS. This transformation W is given by:

$$W = \bigcup_{i=1}^{N} w_i \,. \tag{3.3}$$

The contractivity of W is $\max\{s_i | i = 1, \ldots, N\}$. Eq. 3.3 states that if $A \in X$, then $W(A) = w_1(A) \bigcup w_2(A) \bigcup \ldots \bigcup w_N(A)$. In other words, applying w_1 to A produces a set A_1, applying w_2 to A produces a set A_2, and so on for each w_i. $W(A)$ is then just the union of the sets A_1,A_2,...,A_N. The order in which the w_i are applied to the set

[2]This is a technicality. In the context of image coding, the space of images is complete. For a definition of completeness, see Barnsley (1988).

A does not matter because each w_i is acting independently on the set A; each w_i is applied to the entire set A.

The following two theorems form the basis of classical fractal image compression algorithms. They suggest how contractive transformations of the general form Eq. 3.3 can be used to encode and decode images.

Theorem 1 (Contraction Mapping Fixed Point Theorem) (Barnsley 1988) *Let (X, d) be a complete metric space. Let τ be a contractive transformation defined on X. Then, for any arbitrary initial point $x_o \in X$, there exists a unique point $x_\infty \in X$ (called the fixed point or attractor of τ) to which the sequence*

$$x_o,\ \tau(x_o),\ \tau(\tau(x_o)) = \tau^2(x_o),\ \ldots,\ \tau^n(x_o),\ \ldots$$

converges. Stated mathematically,

$$\lim_{n\to\infty} \tau^n(x_o) \ = \ x_\infty \, . \tag{3.4}$$

Furthermore, x_∞ is left unchanged when operated on by τ: i.e.,

$$\tau(x_\infty) = x_\infty \, . \tag{3.5}$$

This theorem is fundamental to fractal image compression. It states that every contractive transformation τ on a complete metric space X uniquely determines some unique point $x_\infty \in X$ regardless of the choice of x_o. This means that given τ, x_∞ can be determined, and τ can be said to encode x_∞. However, in the context of image coding, the *inverse problem* is what is of interest.

Definition 5 (Inverse Problem) *Given x_∞ (the image to be encoded), find a contractive transformation τ such that x_∞ is the attractor of τ.*

If such a τ can be found, the Contraction Mapping Fixed Point Theorem guarantees that x_∞ can be regenerated by iterating τ on any arbitrary initial point x_o.

Furthermore, if the information necessary to completely specify τ is less than the information required to specify x_∞,τ can be used to compress images. The only problem that remains is finding the transformation τ corresponding to the given image x_∞ that is to be encoded.

Barnsley's Collage Theorem (Barnsley 1988), a derivative of the Contraction Mapping Fixed Point Theorem, suggests how to go about solving this problem.

Theorem 2 (Collage Theorem) (Barnsley 1988)
Let (X,d) be a complete metric space. Let $\psi \in X$ and $\varepsilon \geq 0$. If a contractive transformation $W = \bigcup_{i=1}^{N} w_i$ can be found, such that

$$d\ (\ \psi,\ \bigcup_{i=1}^{N} w_i(\psi))\ \leq\ \varepsilon\ , \tag{3.6}$$

then,

$$d\ (\ \psi,\ x_\infty)\ \leq\ \varepsilon(1-s)^{-1}\ , \tag{3.7}$$

where s is the contractivity of W and x_∞ is the attractor of W.

The Collage theorem states that if a sequence of contractive transformations $w_i : i = 1,\ldots,N$ can be found such that the distance between ψ (the image to be encoded) and $W(\psi)$ (called the collage of ψ) is less than some threshold, ε, then there is a guarantee that the distance between ψ and the attractor of the IFS W will be no greater than $\varepsilon(1-s)^{-1}$. The attractor of W is an approximation of the original image ψ. Theoretically, this approximation can be made as accurate as desired by finding an appropriate IFS W. If $W(\psi) = \psi$, the original image (ψ) can be reconstructed exactly.

Although designing an IFS such that $W(\psi) = \psi$ for any image ψ is always possible[3], the resulting IFS W will consist of a very large number of transformations, thereby resulting in little or no compression of the image. For this reason, in practical image compression applications, the goal is to find an IFS W such that

[3]Just define mappings w_i each of which maps the entire image onto a different pixel.

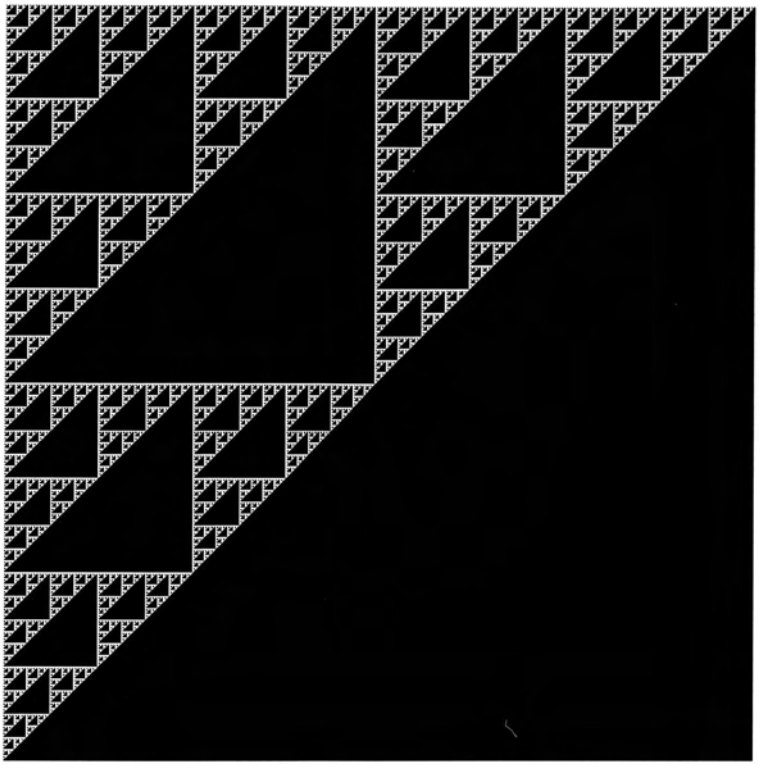

Figure 3.1: Sierpinski Gasket Image

$W(\psi) \approx \psi$. In this way, $x_\infty \approx \psi$, which means that the attractor of the IFS W will only be an approximation of the original image.

Images that can be encoded perfectly using an IFS W all possess, to varying degrees, the property of self-similarity. The image of the Sierpinski Gasket in Figure 3.1 illustrates this property.

The entire image consists of copies of itself, but at different scales. This means, for example, that if the Gasket image were to be contracted both horizontally and vertically by a factor of 2, the resulting image would once again be a Sierpinski Gasket. This is the property of self-similarity that allows for the extremely efficient encoding of images using an IFS.

The Sierpinski Gasket image can be encoded perfectly using only three contractive, affine transformations (Barnsley 1988). When these transformations are applied to the Sierpinski Gasket image, the union of the resulting images is once again a Sierpinski

Gasket. From the earlier discussion, d [Sierpinski Gasket, W(Sierpinski Gasket)]= 0 which implies that the attractor of the IFS W is the Sierpinski Gasket.

Real-world images do not exhibit the type of global self-similarity evident in the Sierpinski Gasket image. Furthermore, the transformations w_i in an IFS must be applied to the entire image. These two factors make designing an IFS that can encode a general real-world image sufficiently well, while at the same time compressing the image, almost impossible. As a result, local iterated function systems[4] (Barnsley and Hurd 1993) (Fisher 1995a) were introduced, an extension of the general IFS idea.

Local iterated function systems differ from the general IFS in that the w_i are not restricted to operating on an entire image; they can be applied to only parts of the image. To encode an image using an LIFS, parts of the image are approximated by transformations applied to other sections of the image with the condition that after all transformations have been applied, the entire image has been covered (approximated). Local iterated function systems exploit image redundancy through this property of piecewise self-transformability. All practical fractal image coding algorithms known to the author are based on LIFS and generalizations thereof.

3.2 Fractal Coding Algorithms

3.2.1 Still Image Coding

All practical fractal coding algorithms known to the author used for the purposes of encoding and decoding real world images can be described using the block diagram shown in Figure 3.2.

Given an input image Ω, the first step in the encoding process is to partition Ω into N disjoint blocks R_i called range blocks; i.e., $\Omega = \cup_{i=1}^{N} R_i$. Generally the range blocks are square although any shape can be used. For each R_i, a domain block

[4]The definitions and theorems provided in this section can also be applied to an LIFS (also called partitioned iterated function system).

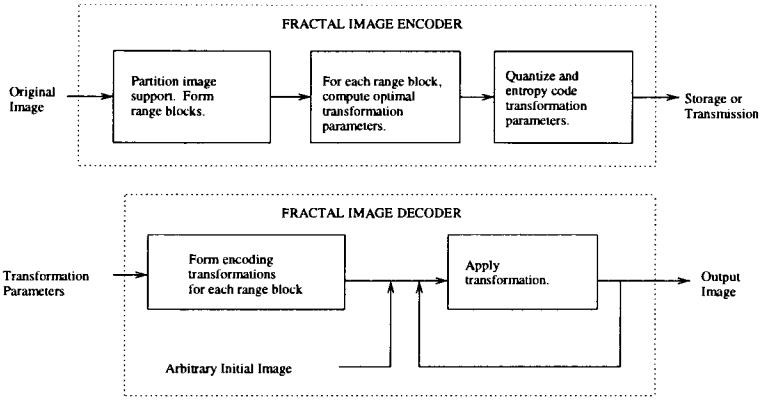

Figure 3.2: Fractal Coder Block Diagram

D_i (all of whose dimensions are larger than R_i) and a contractive transformation w_i must be found such that the distortion incurred in aproximating R_i by $w_i(D_i)$ is as small as possible. The forms of the transformations w_i are defined a priori so that minimization of the distortion simply involves computing the optimal transformation parameters.

Adaptive partitioning algorithms are often used so that if the distortion is larger than a pre-defined threshold, the R_i are further divided and transformations are sought for the smaller range blocks. This partitioning continues until either a satisfactory transformation is found or the smallest allowable range block size is reached. The compression rate achievable is inversely proportional to the number of range blocks coded.

Because of their simple structure, the most common transformations, w_i, are affine transformations that approximate a range block as the linear combination of a transformed domain block and a constant offset term. A key component of the domain block transformation is a spatial contraction operation that shrinks the domain block D_i to the same dimensions as R_i. Spatially contracted domain blocks are required to ensure that the w_i are spatially contractive. The set of all spatially contracted domain blocks for range blocks of a given size will be refered to in this thesis as a domain pool for the range block; the source of the domain blocks (in this case, the

still image being coded) will be called a domain pool source. Using this terminology, we can think of a range block R_i being approximated by a transformation $w_i(D_i)$, where here D_i is a spatially contracted domain block taken from the domain pool; the spatial contraction operation has been removed from w_i and is implicit in the domain pool construction.

Transformations of this type have been generalized to relax the contractivity constraint by introducing fixed or image independent blocks in the encoding tansformation. Fractal coders of this type are called generalized fractal coders in the literature (Gharavi-Alkkansari and Huang 1994).

The final step in the encoding process involves quantization and entropy coding of the parameters needed to describe each w_i. The type of quantizer and entropy code used (if any) depends on the distribution of the transformation parameters. Efficient representations are required if good rate-distortion performance is to be achieved.

The basic core of all fractal still image coding algorithms has been described. Specific algorithms are available in the literature (Fisher 1995b) (Fisher and Menlove 1995) (Barthel et al. 1994) (Gharavi-Alkkansari and Huang 1994).

3.2.2 Fractal Video Coding

As mentioned earlier, much of the work on fractal video coding has focused on

- 3-D fractal block coding
- fractal coding of original frames using the previously reconstructed frame as a domain pool source and variations thereof

There have been numerous papers published on the 3-D fractal block coding approach (Barakat and Dugelay 1996) (Barthel, Ruhl and Voyé 1996) (Lazar and Burton 1994) (Li, Novak and Forchheimer 1993). The method is a straightforward extension of the 2-D case. Instead of partitioning an image into range blocks, a slab of frames

(time dimension) is partitioned into range cubes and suitable transformations are sought for these cubes in the same manner as described in the previous section.

The second method is again a generalization of the 2-D still image coding approach. The algorithm is more flexible because there is no requirement that the encoding transformation be contractive, since domain blocks are obtained from the previously reconstructed frame; i.e., the domain blocks are available a priori and do not have to generated iteratively at the decoder as in the classical 2-D still image coding approach. In addition, decoding is non-iterative and requires only a single iteration. This method can also be used in conjunction with motion compensation to encode only those range blocks in the current frame for which a suitable matching block has not been found. Implementations of this approach (and variations) are described in (Fisher, Rogovin and Shen 1994) (Gharavi-Alkkansari and Huang 1996) (Hürtgen and Büttgen 1993) (Paul and Hayes 1994).

A variation of the second method is described in (Wilson, Nicholls and Monro 1994). In this method original frames are coded independently using a higher-order affine-like transformation but no domain block searching. This general algorithm and its variants have been used for real-time video coding.

We conclude by noting that video sequences, unlike still images, allow for the use of more generalized fractal coders. This is due to the fact that in encoding a given frame the encoding algorithm has access to all previously reconstructed frames as well as any frames that can be derived from them. Consequently, there is no contractivity constraint on the encoding transformation as the domain blocks used in the transformation are available a priori at the decoder. Removing all constraints from the encoding transformation makes higher fidelity encodings possible. Coders of this type are not fractal in the classical sense; however, they do exploit the property of piecewise self-transformability among video sequence frames and thus they are still called fractal coders.

The remainder of this thesis investigates another possible approach to fractal video coding exploiting both classical and generalized fractal coding methods. Specifically, we investigate direct fractal coding of displaced frame difference signals.

Chapter 4

Fractal Coding of DFD Signals

This chapter presents a detailed description and analysis of all fractal video coding algorithms implemented in this work. The two classes of coders considered are the affine transform and OBIFS coders. Simulation results quantifying the performance of these coders for direct fractal coding of displaced frame difference signals are presented in Chapter 5. The chapter begins by presenting the proposed fractal video coding model.

4.1 Proposed Fractal Video Coding Model

Figure 4.1 illustrates the fractal video coding model proposed for this work. The basic features of the model are a motion compensation block, a fractal image encoder block and a switch that allows one of three possible images to be selected as a domain pool source. All fractal video coders implemented in this work comprise the fractal image encoder block.

The fractal image encoder block has two inputs, the DFD signal to be encoded and the image to be used as a domain pool source, and one output, the reconstructed DFD signal. For a given video sequence, the DFD signal generated depends on the motion compensation algorithm used. For a given DFD signal, the quality of the

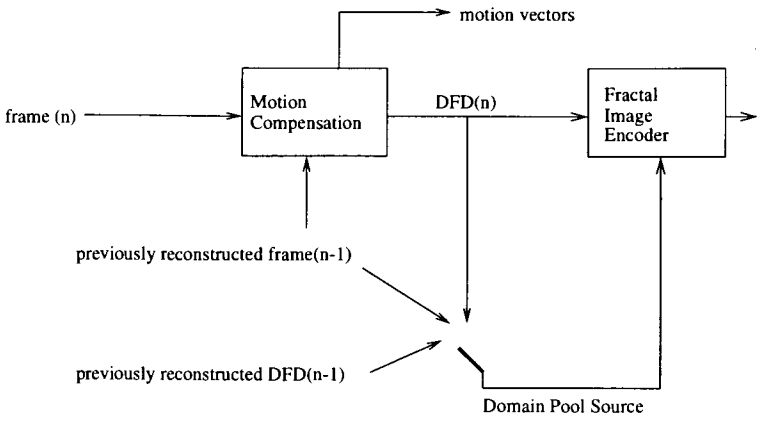

Figure 4.1: Fractal Video Coding Model

reconstructed DFD signal depends on the domain pool source selected. Furthermore, the domain pool source selected has consequences for the encoding transformation and resulting parameter distributions. In general, for a given video sequence, the quality of the reconstructed DFD signal depends on both the motion compensation algorithm and the domain pool source selected. All of these issues will be discussed and analyzed later.

It should also be noted at this time that the mean-squared-error was used as the distortion measure throughout this work.

4.2 Affine-Transform-Based Fractal Coders

This section presents a detailed description and analysis of the affine-transform-based fractal coders implemented in this work. All descriptions are made with respect to the proposed fractal video coding model and are an extension of the general overview presented in Section 3.2.

4.2.1 Domain Pool Construction

Affine-transform-based fractal coders approximate a range block R_i as the linear combination of a transformed spatially contracted domain block taken from the domain

pool of R_i and a constant offset term[1]. With reference to Figure 4.1, we note that there are three possible domain pool sources: the residual[2] image being coded (RBC), the previously coded residual image (PCR) and the previously coded original frame (PCO). The RBC domain pool source corresponds to classical fractal coding of the DFD signal. The PCR and PCO domain pool sources are alternative domain pool sources that place no constraints on the encoding transformation and therefore make higher fidelity encodings possible.

The spatial contraction operators that can be used in forming a domain pool depend on the domain pool source. Specifically, if the domain pool source is the RBC, the spatial contraction operator must be spatially contractive. The PCR and PCO domain pool sources place no constraints on the spatial contraction operator as the domain blocks are available a priori at the decoder. These concepts will be made clearer shortly.

The three spatial contraction operators considered in this work were decimation by a factor of two in each dimension[3] Furthermore, a factor of two is most often used in published work. by averaging (DECIM_BY_AVG) (Lundheim 1995), subsampling (SUBSAMP) (Lundheim 1995) and no contraction (NO_CONTRACTION); these operators were used because they are the most often used in published work. These operators (as implemented) can be described mathematically using the following notation. Let

- Ω_{D_i} denote a $2N \times 2M$ domain block with upper left corner at coordinates $(1, 1)$.
- D_i denote the $N \times M$ domain block resulting from applying the indicated spatial contraction operator to Ω_{D_i}.
- $1 \leq x \leq N$ and $1 \leq y \leq M$

[1]The domain pool can be made adaptive (Jacquin 1992), but to ensure the best coding results, the entire domain pool was used in this work for coding each R_i

[2]Displaced frame difference signals are also called residual images.

[3]Any factor could be used, but there is no reason why any other factor should give better results.

The DECIM_BY_AVG operator generates each pixel in D_i by averaging over four neighboring pixels in Ω_{D_i}. Mathematically,

$$D_i(x,y) = \frac{1}{4}[\Omega_{D_i}(2x-1,2y-1) + \Omega_{D_i}(2x-1,2y) + \Omega_{D_i}(2x,2y-1) + \Omega_{D_i}(2x,2y)].$$

The SUBSAMP operator generates each pixel in D_i by simply taking every second pixel both horizontally and vertically from Ω_{D_i},

$$D_i(x,y) = \Omega_{D_i}(2x-1,2y-1).$$

The NO_CONTRACTION operator is the identity operator. No spatial contraction is performed. The $N \times M$ domain pool block D_i is obtained by directly extracting an $N \times M$ block from the domain pool source.

Of the three spatial contraction operators implemented, only the DECIM_BY_AVG and SUBSAMP operators are spatially contractive; therefore, if the domain pool source is the RBC, only these operators can be used in forming the domain pool.

Given a domain pool source and an appropriate spatial contraction operator, the domain pool construction procedure can now be described. Without loss of generality, we assume that we are constructing a domain pool for range blocks of dimension $N \times M$. Furthermore, it is assumed that the spatial contraction operator being used is DECIM_BY_AVG or SUBSAMP.

The construction begins by positioning a window of dimension $2N \times 2M$[4] in the upper left corner of the domain pool source (coordinates $(1,1)$). The spatial contraction operator is then applied to that section of the domain pool source covered by the window and the resulting block (provided it is not a zero-intensity block) is placed in the domain pool . The window is then moved horizontally by a displacement Δx and the above procedure is repeated. When the window can no longer be moved horizontally without extending outside the domain pool source boundaries, it is repositioned at coordinates $(current\_row + \Delta y, 1)$ where Δy is the vertical window displacement.

[4]If the spatial contraction operator was NO_CONTRACTION, the window size would be $N \times M$.

When the window can no longer be moved horizontally or vertically without extending outside the domain pool source boundaries, the construction is complete. This procedure is repeated for all allowable range block sizes.

In this work, as is the case in most published work, only one spatial contraction operator was used per coder in forming the domain pool[5]. Therefore, given an $N \times M$ domain pool source, an $L \times R$ window (whose dimensions will depend on the range block size and spatial contraction operator used) and horizontal and vertical grid displacements Δx and Δy, the maximum number of domain blocks in the domain pool, N_d, can be computed as

$$N_d = [\frac{M-R}{\Delta x} + 1][\frac{N-L}{\Delta y} + 1]$$

This implies that $\lceil \log_2(N_d) \rceil$ [6] bits are required to specify which domain block was used in the encoding of a given range block (assuming no entropy coding).

4.2.2 Partitioning Algorithms

Adaptive partitioning algorithms are used to try and improve image fidelity by allowing larger range blocks for which a satisfactory transformation has not been found to be split into smaller blocks and transformations sought for these smaller blocks. It is desired to find a partitioning algorithm that maximizes the number of large range blocks coded while at the same time maintaining image fidelity.

The three partitioning algorithms considered in this work were standard quadtree (Fisher 1995b), horizontal-vertical (HV) (Fisher and Menlove 1995) and partial quadtree (Barthel and Voyé 1994). Each of these algorithms (as implemented) will be described assuming that an $N \times N$ range block R_i is being encoded. Specifically, we consider $N = 8$, as 8×8 range blocks (parent blocks) were the largest range blocks allowed; the smallest range blocks allowed were 4×4 range blocks (child blocks). Larger range

[5]There is no reason why the use of more than one spatial contraction operator should yield better results at the same rate.

[6]$\lceil x \rceil$ denotes the smallest integer greater than x.

blocks were not considered as it is very difficult to find good matches for such range blocks even when coding still images.

The following notation will be used in the algorithm descriptions. Let

- T_{split} denote the splitting-threshold
- $w_i(D_i)$ denote the minimum-distortion approximation of R_i
- $d(R_i, w_i(D_i))$ denote the distortion incurred in approximating R_i by $w_i(D_i)$

The quadtree algorithm simply splits R_i into 4 disjoint 4×4 range blocks if $d(R_i, w_i(D_i)) > T_{split}$. Each of these 4×4 range blocks is then encoded separately as no further splitting is allowed.

If $d(R_i, w_i(D_i)) > T_{split}$, the HV partitioning algorithm proceeds in the following manner. R_i is temporarily split vertically into 2 disjoint 8×4 blocks and the variance of these blocks is computed. Let these variances be denoted by σ_1^2 and σ_2^2. Next, R_i is temporarily split horizontally into 2 disjoint 4×8 blocks and the variance of these blocks is also computed. Denote these variances by σ_3^2 and σ_4^2. The final decision on how to split R_i is made by comparing the sum of the computed variances. R_i is split vertically if $\sigma_1^2 + \sigma_2^2 < \sigma_3^2 + \sigma_4^2$; otherwise it is split horizontally. Each 4×8 or 8×4 block can be split further if the distortion incurred in approximating the block is greater than T_{split}. However, the manner in which these blocks are split is known a priori because the smallest allowable range block side dimension is 4. This implies that if the blocks are split they will be split into 2 disjoint 4×4 blocks; no further splitting is allowed.

In the partial quadtree algorithm, the distortion incurred in approximating R_i by $w_i(D_i)$ is computed over each quadrant (4×4 blocks). If 2 or more quadrants have distortions less than T_{split}, the transformation w_i is stored and the remaining quadrants for which the distortion is greater than T_{split} are coded separately. If fewer than 2 quadrants have distortions less than T_{split}, the transformation w_i is abandoned

and R_i is split into 4 disjoint 4×4 blocks each of which is coded separately; no further splitting is allowed.

The overhead required to specify how a range is split depends on the particular partitioning algorithm used. The quadtree algorithm has the least amount of overhead requiring only one bit to indicate whether or not the 8×8 range block was split. The HV algorithm is slightly more complex and requires two bits of overhead; one bit to indicate whether or not the 8×8 range block was split, and if the block was split, one bit to indicate whether the split was horizontal or vertical. The partial quadtree algorithm requires the most overhead; one bit is required to indicate whether or not the 8×8 range block was split, and if the block was split, 4 bits are required to indicate the nature of the split. To see where this 4-bit overhead comes from, consider that with 4 quadrants, there is one way of selecting four 4×4 blocks; there are 4 ways of selecting three 4×4 blocks; there are 6 ways of selecting two 4×4 blocks. Therefore,

$$\lceil \log_2(1 + 4 + 6) \rceil = 4$$

bits.

4.2.3 Encoding Transformations

In all of the affine-transform-based fractal coders implemented, range blocks were classified into one of three classes with the encoding transformation dependent on the block class. In this section, the encoding transformations and quantization considerations for the uniform and non-uniform block classes will be described. The third range block class is trivial and will be defined in Section 4.2.4 when the complete encoding procedure is outlined.

To simplify this discussion, we introduce the following vector notation. Let

- $\mathbf{r}$ denote the range block R_i to be encoded
- $\hat{\mathbf{r}}$ denote an approximation of $\mathbf{r}$

- $\{\mathbf{d}_j\}_{j=1}^{L}$ denote the domain pool for the range block $\mathbf{r}$
- $\mathbf{o}$ denote a constant vector of all ones
- T_{mse} denote the pre-defined mean-squared-error threshold

The vector dimensions are equal to the number of pixels in the block they represent. Throughout this section, we will assume that the range block R_i being encoded has dimensions $N \times M$. Furthermore, we define $K = MN$; this implies that $\mathbf{r} \in \Re^K$. The dimensions of all other vectors can be deduced from the dimension of $\mathbf{r}$.

Uniform

A range block $\mathbf{r}$ is classified as a uniform block if its variance is less than or equal to T_{mse}. Uniform blocks were approximated by their mean, μ_r, i.e.,

$$\hat{\mathbf{r}} = \mu_r \mathbf{o} \tag{4.1}$$

By approximating a uniform block by its mean, we are assured that the resulting mean-squared-error will be less than T_{mse}.

After encoding all uniform range blocks $\mathbf{r}$ in the DFD signal, we are left with a set of real numbers (the range block means) that have to be quantized. Therefore, consider what happens when we quantize $\gamma = \mu_r$ using a quantizer Q. Let $Q(\gamma) = \gamma - \Delta\gamma$ where $\Delta\gamma$ is the quantization error incurred in quantizing γ using the quantizer Q. The quantized approximation of $\mathbf{r}$, $\hat{\mathbf{r}}_q$, can be written as

$$\hat{\mathbf{r}}_q = Q(\gamma)\mathbf{o} = (\gamma - \Delta\gamma)\mathbf{o}. \tag{4.2}$$

Let $\mathbf{e}$ be the error vector resulting by approximating $\mathbf{r}$ by $\hat{\mathbf{r}}_q$. Mathematically,

$$\mathbf{e} = \mathbf{r} - \hat{\mathbf{r}}_q. \tag{4.3}$$

Substituting for $\hat{\mathbf{r}}_q$ using Eq. 4.2, we get

$$\mathbf{e} = (\mathbf{r} - \gamma\mathbf{o}) + \Delta\gamma\mathbf{o}. \tag{4.4}$$

The bracketed term in this equation is the collage error, $\mathbf{e}_c$, for uniform range blocks $\mathbf{r}$; the second term is the error due to quantization of the range block mean. The squared error, $\mathbf{e}^T\mathbf{e}$, can thus be expressed mathematically as

$$\mathbf{e}^T\mathbf{e} = \mathbf{e}_c^T\mathbf{e}_c + (\Delta\gamma)^2\mathbf{o}^T\mathbf{o}. \tag{4.5}$$

Dividing by K, we obtain the final expression for the mean-squared-error incurred in approximating a uniform range block $\mathbf{r}$

$$\frac{\mathbf{e}^T\mathbf{e}}{K} = \frac{\mathbf{e}_c^T\mathbf{e}_c}{K} + (\Delta\gamma)^2. \tag{4.6}$$

Let $D_\gamma = (\Delta\gamma)^2$ represent that part of the mean-squared-error attributed to quantization; D_γ will be used later when the quantization and entropy coding strategies implemented in this work are described.

Non-uniform

A range block $\mathbf{r}$ is classified as a non-uniform block if its variance is greater than T_{mse}. Each non-uniform block is approximated as a linear combination of a transformed spatially contracted domain block and a constant offset term. The transformations applied to the spatially contracted domain blocks are called isometries; these transformations do not alter the pixel intensities of the block; rather they simply shuffle the pixels within a block. There are eight possible ways of mapping an $N \times N$ block Ω onto itself without scrambling the pixels. These are described below using the terminology adopted by Jacquin (1992). In each case, $1 \leq x \leq N$, $1 \leq y \leq N$.

1. Identity:

   $$\Omega(x, y) = \Omega(x, y)$$

2. Orthogonal Reflection About Mid-Vertical Axis:

   $$\Omega(x, y) = \Omega(x, N - y + 1)$$

3. Orthogonal Reflection About Mid-Horizontal Axis:

   $$\Omega(x, y) = \Omega(N - x + 1, y)$$

4. Orthogonal Reflection About First Diagonal ($y = x$):

$$\Omega(x, y) = \Omega(y, x)$$

5. Orthogonal Reflection About Second Diagonal ($y = N - x + 1$):

$$\Omega(x, y) = \Omega(N - y + 1, N - x + 1)$$

6. Rotation Around Center of Block ($+90°counter - clockwise$):

$$\Omega(x, y) = \Omega(y, N - x + 1)$$

7. Rotation Around Center of Block ($+180°counter - clockwise$):

$$\Omega(x, y) = \Omega(N - x + 1, N - y + 1)$$

8. Rotation Around Center of Block ($+90°clockwise$):

$$\Omega(x, y) = \Omega(N - y + 1, x)$$

If the spatially contracted domain block is rectangular, only isometries 1, 2, 3, and 7 can be used as only these isometries map rectangular blocks onto rectangular blocks of the same dimensions. Therefore, for a square range block 3 bits are needed to specify which of the 8 isometries was used; for a rectangular range block only 2 bits are required.

The two encoding transformations considered for encoding non-uniform blocks can be expressed mathematically as

$$\hat{\mathbf{r}} = \alpha_1 \mathbf{d} + \beta_1 \mathbf{o} \tag{4.7}$$

and

$$\hat{\mathbf{r}} = \alpha_2(\mathbf{d} - \mu_d \mathbf{o}) + a_o \mu_d \mathbf{o} + \beta_2 \mathbf{o} \tag{4.8}$$

where $\mathbf{d}$ is a transformed spatially contracted domain block from the domain pool of $\mathbf{r}$, μ_d is the mean of $\mathbf{d}$ and α_1, α_2, β_1, β_2, and $a_o \in \Re$. a_o is a constant specified by the encoding algorithm.

Eq. 4.7 is the standard affine transformation used in fractal image coding; it will be refered to as the standard affine transformation throughout the remainder of this thesis. A sufficient condition for this transformation to be contractive is that $|\alpha_1| < 1.0$. However, in practice (Fisher 1995b), slightly larger values are used and the overall image transformation still converges (see Section 3.1). Values of α_1 in the range [-1.1, 1.1] were used in this work. Note that this contractivity condition only applies if the domain pool source is the RBC. α_1 is unconstrained if the domain pool source is the PCR or the PCO.

Eq. 4.8 was introduced by (Barthel et al. 1994) as a means of decorrelating the α_2 and β_2 parameters (this will be explained later). It is called a modified luminance transformation in the literature, and it will be refered to as such throughout the remainder of this thesis. This transformation is used for a very specific purpose in this work. Its use is restricted to the PCR and PCO domain pool sources, and thus, there are no constraints on any of the transformation parameters. It is not used for the RBC domain pool source because the distributions of the resulting transformation parameters are such that its use is not required (this will be explained later).

Both transformations span the same space and therefore produce the same distortion in the absence of quantization. However, the two transformations differ in the parameter distributions they generate; the importance of this observation to this work will be made clear in Section 5.2.2.

Given a non-uniform range block $\mathbf{r}$ and a transformed spatially contracted domain block $\mathbf{d}$, we want to find α and β such that

$$J(\alpha, \beta) = \| \mathbf{r} - \hat{\mathbf{r}} \|^2 \tag{4.9}$$

is minimized.

Using the orthogonality principle of optimal least squares estimation, the α, β pair minimizing Eq. 4.9 can be found by solving the following two equations for α and β

$$(\mathbf{r} - \hat{\mathbf{r}})^T \mathbf{d} = \mathbf{0} \tag{4.10}$$

and

$$(\mathbf{r} - \hat{\mathbf{r}})^T \mathbf{o} = \mathbf{0}. \tag{4.11}$$

where $\mathbf{0}$ is the zero-vector.

Substituting Eq. 4.7 into Eqs. 4.10 and 4.11 and solving for α_1 and β_1 we find that the optimal α_1, β_1 pair for the standard affine transformation is given by

$$\alpha_1 = \frac{\mathbf{r}^T\mathbf{d} - K\mu_d\mu_r}{\mathbf{d}^T\mathbf{d} - K\mu_d^2} \tag{4.12}$$

and

$$\beta_1 = \mu_r - \alpha_1\mu_d \tag{4.13}$$

where K is the dimension of $\mathbf{r}$, μ_d is the mean of $\mathbf{d}$ and μ_r is the mean of $\mathbf{r}$.

Substituting Eq. 4.8 into Eqs. 4.10 and 4.11 and solving for α_2 and β_2 we find that the optimal α_2, β_2 pair for the modified luminance transformation is given by

$$\alpha_2 = \frac{\mathbf{r}^T\mathbf{d} - K\mu_d\mu_r}{\mathbf{d}^T\mathbf{d} - K\mu_d^2} \tag{4.14}$$

and

$$\beta_2 = \mu_r - a_o\mu_d \tag{4.15}$$

where K, a_o, μ_d and μ_r are as described previously.

It can be seen that $\alpha_1 = \alpha_2$. However, β_1 and β_2 differ; the optimal β_1 for the standard affine transformation depends on α_1 whereas the optimal β_2 for the modified luminance transformation has no dependance on α_2. Both optimal β values depend on the range block and spatially contracted domain block means.

Given an encoding transformation and a non-uniform range block $\mathbf{r}$ to encode, Eq. 4.9 is minimized for all possible transformed spatially contracted domain blocks in the domain pool of $\mathbf{r}$. The domain block $\mathbf{d}$ with associated isometry, and the corresponding α, β pair yielding the minimum distortion form the optimal encoding transformation for $\mathbf{r}$.

The resulting α and β values for each non-uniform range block $\mathbf{r}$ must next be quantized. Consider what happens when α and β are quantized using quantizers, Q_α and Q_β. Let $Q_\alpha(\alpha) = \alpha - \Delta\alpha$ and $Q_\beta(\beta) = \beta - \Delta\beta$ where $\Delta\alpha$ and $\Delta\beta$ are the quantization errors incurred in quantizing α and β respectively. The quantized approximation of $\mathbf{r}$, $\hat{\mathbf{r}}_q$, for the standard affine transformation can be written as

$$\begin{aligned} \hat{\mathbf{r}}_q &= Q_\alpha(\alpha_1)\mathbf{d} + Q_\beta(\beta_1)\mathbf{o} \\ &= (\alpha_1 - \Delta\alpha_1)\mathbf{d} + (\beta_1 - \Delta\beta_1)\mathbf{o} \end{aligned} \tag{4.16}$$

and for the modified luminance transformation,

$$\begin{aligned} \hat{\mathbf{r}}_q &= Q_\alpha(\alpha_2)(\mathbf{d} - \mu_d\mathbf{o}) + a_o\mu_d\mathbf{o} + Q_\beta(\beta_2)\mathbf{o} \\ &= (\alpha_2 - \Delta\alpha_2)(\mathbf{d} - \mu_d\mathbf{o}) + a_o\mu_d\mathbf{o} + (\beta_2 - \Delta\beta_2)\mathbf{o}. \end{aligned} \tag{4.17}$$

Let $\mathbf{e}$ be as defined in Eq. 4.3. Substituting for $\hat{\mathbf{r}}_q$ using Eqs. 4.16 and 4.17, we get that

$$\mathbf{e} = (\mathbf{r} - \alpha_1\mathbf{d} - \beta_1\mathbf{o}) + (\Delta\alpha_1\mathbf{d} + \Delta\beta_1\mathbf{o}) \tag{4.18}$$

for the standard affine transformation, and

$$\mathbf{e} = [\mathbf{r} - \alpha_2(\mathbf{d} - \mu_d\mathbf{o}) - a_o\mu_d\mathbf{o} - \beta_2\mathbf{o}] + [\Delta\alpha_2(\mathbf{d} - \mu_d\mathbf{o}) + \Delta\beta_2\mathbf{o}] \tag{4.19}$$

for the modified luminance transformation.

The first term in each of the above expressions is the collage error, $\mathbf{e}_c$, resulting from encoding the non-uniform range block $\mathbf{r}$ with the corresponding encoding transformation. The second term is the error due to quantization of the α and β parameters.

Using the fact that the optimal α, β pair satisfy Eqs. 4.10 and 4.11, the final mean-squared-error incurred in aproximating a non-uniform range block $\mathbf{r}$ by $\hat{\mathbf{r}}_q$ can be expressed mathematically as

$$\frac{\mathbf{e}^T\mathbf{e}}{K} = \frac{(\Delta\alpha_1)^2\mathbf{d}^T\mathbf{d}}{K} + 2\mu_d\Delta\alpha_1\Delta\beta_1 + (\Delta\beta_1)^2 \tag{4.20}$$

for the standard affine transformation and as

$$\frac{\mathbf{e}^T\mathbf{e}}{K} = (\Delta\alpha_2)^2\sigma_d^2 + (\Delta\beta_2)^2 \tag{4.21}$$

for the modified luminance transformation where σ_d^2 is the variance of $\mathbf{d}$ and all other parameters are as previously defined.

Let $D_\alpha = (\Delta\alpha_2)^2\sigma_d^2$ and $D_\beta = (\Delta\beta_2)^2$ represent that part of the mean-squared-error due solely to quantization of the α_2 and β_2 parameters when the modified luminance transformation is used as the encoding transformation; D_α and D_β will be used later when the quantization and entropy coding strategies implemented in this work are described.

This section can be summarized by noting that the final distortion incurred in quantizing a range block $\mathbf{r}$ is made up of the sum of two separate components. The first component, the collage error, is due to mismatches between the assumed fractal model and the actual DFD signal being coded. Even in the absense of quantization, the collage error will never be zero unless the DFD signal being coded can be exactly described by our proposed fractal model. This is a characteristic feature of most fractal coding agorithms; there is a lower limit to the minimum distortion that can be achieved even in the absence of quantization. The second component of the final distortion is due to quantization of the transformation parameters. This component

can be minimized (at a given rate) by the use of efficient quantization strategies. These strategies depend on the distributions of the encoding parameters; the strategies implemented in this work will be described in Section 5.2.2 when the parameter distributions are analyzed.

4.2.4 Encoding Procedure

This section outlines the complete encoding procedure for all affine-transform-based fractal coders implemented in this work. The proposed fractal video coding model shown in Figure 4.1 will once again be used as our reference.

Let the video sequence to be encoded be represented by $\{\Omega_0, \Omega_1, \ldots, \Omega_N\}$ where Ω_i is the i'th frame in the video sequence, and $N+1$ is the total number of frames.

For all affine-transform-based fractal coders implemented, the following encoder parameters were specified a priori:

- the motion compensation algorithm, including MC block size and search size
- the domain pool source with associated spatial contraction operator and horizontal and vertical grid displacements Δx, Δy
- the partitioning algorithm
- the encoding transformation to be used for encoding non-uniform blocks
- the mean-squared-error threshold, T_{mse}

Given that the above parameters have been specified, the encoding procedure can be described. The encoding procedure can be divided into two parts: DFD signal generation and fractal encoding of the resulting DFD signal.

DFD signal generation pertains to the motion compensation block of the proposed fractal video coding model. At start-up, the motion compensation block takes as input

Ω_0 and Ω_1[7] and produces at its output a DFD signal and a set of motion vectors. Subsequent actions depend on the domain pool source. If the domain pool source is the RBC or the PCO, the output DFD signal is encoded. However, if the domain pool source is the PCR, the output DFD signal, which is the first one generated, is not coded; rather it is used as a domain pool source for the next DFD signal output by the motion compensation block. This issue only occurs on start-up as it is required to produce a first residual image if the PCR is the domain pool source. Under steady state conditions, the motion compensation block will take as input the current original frame and the previously reconstructed original frame and it will produce at its output a DFD signal to be encoded and a set of motion vectors.

The second part of the encoding procedure is fractal encoding of the output DFD signal. Independent of the partitioning algorithm, the largest allowable range block size (as noted earlier) was 8×8, and the smallest allowable range block size was 4×4. The first step in the encoding process is to produce a domain pool for each of the allowable range block sizes using the specified domain pool source and spatial contraction operator. The DFD signal is then partitioned into disjoint 8×8 range blocks R_i and for each R_i the following actions are taken:

- If the average energy-per-pixel is less than T_{mse}, R_i is considered coded as the motion compensation algorithm has found a good match. In this case, simply approximating R_i by a block of zero-intensity yields a mean-squared-error less than T_{mse}. It should be noted that this step was only performed for the largest allowable range blocks (8×8 blocks).

- If a good match has not been found by the motion compensation algorithm, R_i is classified as uniform or non-uniform and coded accordingly. R_i is classified as uniform if its variance is less than T_{mse}; otherwise, it is classified as non-uniform. If R_i is classified as uniform, it is simply approximated by its mean. If

[7]For all initial simulations, original frames were used. When final comparisons were made to a reference DCT-based coder (later), coded frames were used at start-up.

it is classified as non-uniform, Eq. 4.9 is minimized for all possible combinations of isometries and spatially contracted domain blocks from the domain pool of R_i. If the resulting minimum distortion is less than T_{mse}, R_i is considered coded; otherwise, R_i is split according to the specified partitioning algorithm and the entire process is repeated.

After all range blocks R_i have been encoded, there are three sets of parameters that must be quantized before an output bit stream can be produced; namely, the set of all γ values called absorb[8] parameters, the set of all α values called alpha parameters and the set of all β values called beta parameters. The quantization and entropy coding strategies for these parameters will be described in Section 5.2.2.

4.2.5 Decoding Procedure

The previous section discussed the encoding procedure for affine-transform-based fractal coders. This section outlines the decoding procedure.

The decoder has the following information available a priori:

- the motion compensation algorithm and MC block size
- the domain pool source and associated spatial contraction operator
- the horizontal and vertical grid displacements Δx, Δy
- the order in which the R_i were encoded; this ordering never changes and is defined a priori
- the partitioning algorithm
- the form of all encoding transformations
- all allowable isometries

[8]Adopting the terminology of Jacquin (1992) who called them absorption parameters.

Furthermore, if the transformation parameters are quantized and entropy coded, the decoder will have knowledge of the quantization and entropy coding strategies used. If the transformation parameters are not quantized, knowledge of the above information is sufficient to reconstruct the coded DFD signal.

The exact decoding procedure depends on the domain pool source. If the domain pool source is the RBC, decoding is governed by the contraction mapping fixed point theorem. Specifically, we start with an initial zero-intensity image Ω_o, and iterate the encoding transformation $W = \bigcup_{i=1}^{N} w_i$ on Ω_o. Since the decoder knows the general form of each range block transformation, w_i, reformulating the transformations is trivial. The procedure simply consists of acquiring the necessary transformation parameters from the encoded output file or bit stream (if quantized coders are used). The transformations, w_i, are then applied to Ω_o just as in the encoding procedure. 20 iterations were used to reconstruct encoded DFD signals when the domain pool source was the RBC.

If the domain pool source is the PCR or PCO, decoding is non-iterative and requires only a single iteration as all domain blocks are available a priori.

The resulting reconstructed DFD signal is then added to the previously reconstructed original frame (using the motion vectors) to form an approximation of the current frame. This procedure is repeated until the entire video sequence has been decoded.

4.3 Orthogonal Basis IFS Coders

The previous section considered the use of affine-transform-based fractal coders for direct fractal coding of DFD signals. Range blocks, $\mathbf{r}$, were approximated as a linear combination of a transformed spatially contracted domain block, $\mathbf{d}$, and a fixed block, $\mathbf{o}$. It was mentioned that using such coders the collage error could not be reduced to zero (even in the absence of quantization) unless the DFD signal being encoded could be exactly described by the corresponding fractal model.

In this section, a higher-dimensional encoding transformation is presented for direct fractal coding of DFD signals. This transformation has the property that in the absence of quantization, the collage error is zero. These coders thus have the potential for higher fidelity encodings as in the absence of quantization perfect reconstruction is achieved. The resulting class of coders are called orthogonal basis IFS (OBIFS) coders; they were first introduced by Vines (1995).

To simplify the discussions in this section, the following notation will be used throughout:

- Ω will denote the DFD signal being encoded
- an $N \times N$ range block R_i being encoded will be denoted by the column vector $\mathbf{r}_i$ where $K = N \times N$ and $\mathbf{r}_i \in \Re^K$

4.3.1 Overview

Orthogonal basis IFS coders are essentially transform coders in which most of the basis vectors are generated from the image being encoded. With reference to our proposed fractal video coding model, there are three possible sources for these basis vectors: the RBC, the PCR and the PCO.

Orthogonal basis IFS coders approximate a range block, $\mathbf{r}_i$, as linear combination of K basis vectors.

$$\mathbf{r}_i = \sum_{i=1}^{N_d} \alpha_i \mathbf{b}_i + \sum_{j=1}^{N_f} \beta_j \mathbf{f}_j$$

The basis vectors comprise two distinct sets: a set of N_d image-dependent basis vectors $\{\mathbf{b}_i\}_{i=1}^{N_d}$ and a set of N_f fixed (image-independent) basis vectors $\{\mathbf{f}_j\}_{j=1}^{N_f}$ where $K = N_d + N_f$. The image dependent basis vectors are generated from the specified domain pool source (to be explained later). Furthermore, the basis vectors are made orthonormal so that computation of the α_i and β_j is simplified. It is important to note that if the RBC is being used as the domain pool source at least one fixed basis vector is required to ensure that the encoding transformation is non-linear and therefore possesses a non-zero fixed point (if contractive).

As in transform coding, the main goal of the orthogonal basis IFS idea is to find a set of basis vectors that result in maximum energy compaction so that each range block can be approximated by as few basis vectors as possible.

In implementing orthogonal basis IFS coders, the following issues arise:

- selecting the fixed basis vectors $\mathbf{f}_j$
- generating a set of desirable image dependent basis vectors $\mathbf{b}_i$
- allocating bits, quantizing, and entropy coding the transform coefficients

These issues, as they pertain to this work, are addressed in subsequent sections.

4.3.2 Basis Generation Methods

In this work, only one fixed basis vector, $\mathbf{f}_1$, was used. This vector was the dc-vector given by

$$\mathbf{f}_1 = \frac{1}{\sqrt{K}}[1, 1, \ldots, 1] \tag{4.22}$$

where $\| \mathbf{f}_1 \| = 1$. It is used to represent the dc component of a block. Generally, any vectors can be used as fixed basis vectors; it was decided to use only $\mathbf{f}_1$ so that a direct comparison could be made to a reference DCT-based coder.

To generate the remaining $K-1$ image-dependent basis vectors, $\mathbf{b}_i$, two algorithms (basis generation methods) were considered. The first algorithm known as the covariance method was introduced by Vines (1995). The second algorithm, introduced by the author for comparative purposes, will be called the centroid method. This algorithm is a variant of an algorithm proposed by Vines (1995). Both algorithms attempt to generate, from a specified domain pool source, a set of $K-1$ basis vectors $\mathbf{b}_i$ that will most efficiently represent the set of range blocks to be encoded.

Let $\{\mathbf{r}_i\}_{i=1}^{M}$ be the set of range blocks obtained by partitioning the DFD signal, Ω, to be encoded. Independent of the basis generation method used, the first step is to remove from each $\mathbf{r}_i$ their projection onto $\mathbf{f}_1$. Thus, a new set of vectors $\{\mathbf{r}_i^{(1)}\}_{i=1}^{M}$ are produced where

$$\mathbf{r}_i^{(1)} = \mathbf{r}_i - [\mathbf{r}_i^T \mathbf{f}_1]\mathbf{f}_1 \tag{4.23}$$

At this point, the covariance and centroid algorithms proceed in different manners. The covariance algorithm proceeds as follows:

1. For each $\mathbf{r}_i^{(1)}$, compute the following

$$\sum_{j=1, j\neq i}^{M} |(\mathbf{r}_i^{(1)})^T \mathbf{r}_j^{(1)}| \tag{4.24}$$

   and select as the optimal basis vector direction for $\mathbf{b}_1$ that $\mathbf{r}_i^{(1)}$ for which the above quantity is maximized; store this vector as $\mathbf{t}_1$.

2. Remove from $\{\mathbf{r}_i^{(1)}\}_{i=1}^{M}$ their projection onto $\mathbf{t}_1$; a new set of vectors $\{\mathbf{r}_i^{(2)}\}_{i=1}^{M}$ is produced.

3. Repeat the above steps with $\{\mathbf{r}_i^{(2)}\}_{i=1}^{M}$ and so on until a final set $\{\mathbf{r}_i^{(K-1)}\}_{i=1}^{M}$

of vector is produced; at this time, the optimal basis vector directions for $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{K-1}$ will all be determined.

Based on the above algorithm description, we note that the covariance method selects at each stage that vector in the set which has the maximum correlation with all other vectors. The output of the above algorithm is a set of $K-1$ orthogonal direction vectors $\mathbf{t}_j$, $j = 1, \ldots, K-1$. The way these direction vectors are used to form a set of basis vectors will be described shortly after the centroid algorithm is outlined.

The centroid algorithm proceeds as follows:

1. For a set of vectors $\{\mathbf{r}_i^{(1)}\}_{i=1}^{M}$, the corresponding optimal basis vector direction, $\mathbf{t}_1$ is computed as

$$\frac{1}{M}\sum_{i=1}^{M}\mathbf{r}_i^{(1)} \tag{4.25}$$

   $\mathbf{t}_1$ is simply that vector, $\mathbf{c}$, that minimizes $\sum_{i=1}^{M} \| \mathbf{r}_i^{(1)} - \mathbf{c} \|^2$.

2. Remove from $\{\mathbf{r}_i^{(1)}\}_{i=1}^{M}$ their projection onto $\mathbf{t}_1$; a new set of vectors $\{\mathbf{r}_i^{(2)}\}_{i=1}^{M}$ is produced.

3. Repeat the above steps with $\{\mathbf{r}_i^{(2)}\}_{i=1}^{M}$ and so on until a final set $\{\mathbf{r}_i^{(K-1)}\}_{i=1}^{M}$ of vector is produced; at this time, the optimal basis vector directions for $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{K-1}$ will all be determined.

The centroid algorithm selects at each stage the mean of the vectors in the set. The output of the centroid algorithm is also a set of direction vectors $\mathbf{t}_j$, $j = 1, \ldots, K-1$.

Independent of the basis generation method used, we are left with a set of $K-1$ direction vectors $\mathbf{t}_j$. Representing these direction vectors directly would require a very large number of bits (K^2 times the number of bits required to represent each component); therefore, to efficiently represent the direction vectors $\mathbf{t}_j$, domain blocks

(from the domain pool) are sought which most closely resemble (to be explained shortly) each $\mathbf{t}_j$. The overhead required to specify this encoding transformation is thus $(K-1)N_d$ bits where N_d was defined in Section 4.2.1. The overhead simply consists of specifying which domain blocks best approximate the direction vectors.

Continuing on, the next step involved in generating the basis is the formation of a domain pool. The domain pool is constructed as described in Section 4.2.1.

Given a domain pool consisting of domain blocks, $\mathbf{d}$, generated from the specified domain pool source, the basis generation procedure continues as follows:

1. Starting with $\mathbf{t}_1$, find that vector, $\mathbf{d}$, in the domain pool which has the largest component in the direction of $\mathbf{t}_1$; i.e., find $\mathbf{d}$ such that

$$\frac{\mathbf{d}^T\mathbf{t}_1}{\| \mathbf{t}_1 \|} \tag{4.26}$$

   is maximized. The corresponding optimal vector $\mathbf{d}$ is the basis vector $\mathbf{b}_1$. $\mathbf{d}$ is removed from the domain pool so that it will not be selected again.

2. Repeat step 1 for $\mathbf{t}_2, \mathbf{t}_3, \ldots, \mathbf{t}_{K-1}$.

After this stage of the basis generation procedure, we are left with a fixed basis vector, $\mathbf{f}_1$, and $K-1$ image dependent basis vectors $\mathbf{b}_j$, $j = 1, \ldots, K-1$. The standard Gram-Schmidt orthogonalization algorithm was then applied to the ordered set of vectors $[\mathbf{f}_1, \mathbf{b}_1, \ldots, \mathbf{b}_{K-1}]$ to produce a complete set of orthonormal basis vectors. Let $\mathbf{A}$ denote the matrix whose columns are the resulting set of orthonormal basis vectors.

4.3.3 Quantization and Entropy Coding

Given that Ω has been partitioned into disjoint range block $\mathbf{r}_i \in \Re^K$ and given the matrix $\mathbf{A}$, the quantization and entropy coding strategy utilized will now be described. The method implemented was based on the JPEG standard (Pennebaker and Mitchell

1993) for quantizing and entropy coding the DCT transform coefficients. This method was selected because it is a very efficient method for quantizing and entropy coding transform coefficients, and it allows a direct comparison to a reference DCT-based coder to be made.

The quantization process begins by computing the transform coefficients, $\mathbf{x}_i$, for each $\mathbf{r}_i$. Since $\mathbf{A}$ is orthonormal, the $\mathbf{x}_i$ are easily computed as

$$\mathbf{x}_i = \mathbf{A}^T \mathbf{r}_i. \tag{4.27}$$

The i'th coefficients from each $\mathbf{x}_i$ are then grouped into a set of K sources. For each of the K sources, the following actions are taken:

- Let $\Delta_{min} = 1$ and $\Delta_{max} = 128$. For each Δ in the range Δ_{min} to Δ_{max} in increments of 1, quantize the source coefficients with a uniform quantizer of step size Δ. For an input point x, the quantized output point, $Q(x)$, is the integer part of

$$\frac{[x + \frac{\Delta}{2}]}{\Delta}. \tag{4.28}$$

- For each quantizer designed, compute the rate as the entropy of the output quantization indicies and the distortion as the average distortion incurred in quantizing the source coefficients. Form a corresponding rate-distortion table.

After performing the above steps on all K sources, the BFOS algorithm is performed to determine the optimal quantizer step sizes for each of the K sources based on a specified target rate; this is not quite optimal as the run-length coding procedure (to be described shortly) was not included here. Note that all the step sizes are constrained to be integers as in the JPEG standard; furthermore, all the quantized coefficients are integers. The values specified for Δ_{min} and Δ_{max} are not part of the JPEG standard; they were used in all simulations to allow a wide range of rates to be achieved.

At this point we have a set of quantized transform coefficients corresponding to each $\mathbf{r}_i$; denote these quantized transform coefficient vectors by $\mathbf{x}_i^q$ where all the components of these vectors are integers. The entropy coding method adopted is that proposed in the JPEG standard; all coefficients are treated as AC coefficients and entropy coded accordingly.

Adopting the same terminology used in the JPEG standard, we define the following terms:

- RRRR represents the length of a run of zeros before a non-zero coefficient is reached; the maximum run length is 15.
- SSSS represents the number of additional bits required to represent a non-zero coefficient; SSSS values ranging from 1 to 9 were used in this implementation.
- EOB denotes end-of-block; it is used to specify that all remaining coefficients in the block are zero

The SSSS values also represent categories of transform coefficients where the n'th category, $n = 1, \ldots, 9$, consists of the ordered set of integers

$$[-(2^n - 1), \ldots, -2^{n-1}, 2^{n-1}, \ldots, 2^n - 1]$$

and n is the number of bits required to specify an integer in the set (the n'th category contains 2^n integers). For example, for $n = 3$ the n'th category consists of the ordered set of integers

$$[-7, \ldots, -4, 4, \ldots, 7].$$

There are 8 elements in the above set; therefore, to specify any particular element, 3 bits are required.

Using the terminology defined above, the basic features of the entropy coding procedure will now be described. The goal of the algorithm is to produce an entropy

code for so-called RUN-SIZE symbols where a RUN-SIZE symbol is obtained from RRRR and SSSS. A RUN-SIZE symbol can be thought of as an eight-bit number with the upper four bits comprising RRRR, and the lower four bits comprising SSSS. These RUN-SIZE symbols are generated for each block of quantized transform coefficients $\mathbf{x}_i^q$, and a Huffman code designed using the actual frequency of occurrence of each symbol.

Suppose we are processing a particular vector $\mathbf{x}_i^q$ of quantized transform coefficients. The basic algorithm for constructing RUN-SIZE symbols for such a vector proceeds as follows. Start from the first coefficient and move through the vector until a non-zero coefficient is reached. The number of zero coefficients before reaching the non-zero coefficient is RRRR. SSSS is obtained by determining in which of the nine categories the non-zero coefficient belongs. The values of RRRR and SSSS thus define the RUN-SIZE symbol; following the RUN-SIZE symbol, SSSS additional bits are required to specify exactly which coefficient in the selected category the non-zero coefficient corresponds to. The algorithm proceeds in this manner until the end of the vector is reached.

There are two special cases in the above procedure that must be described. First, the maximum run length allowed is 15; therefore, if a run of more than 15 zeros occurs, the zeros are processed as runs of length 15 followed by a zero. This implies RRRR equals 15, SSSS equals 0, and no additional bits are required after the RUN-SIZE symbol. The second special case occurs when all the remaining coefficients in the vector are zero. In this case, RRRR equals 0, SSSS equals 0 and no additonal bits are required after the RUN-SIZE symbol. This RUN-SIZE symbol is denoted EOB as described earlier.

The above procedure is repeated for all vectors $\mathbf{x}_i^q$ of quantized transform coefficients. A histogram of all the generated RUN-SIZE symbols is then formed and a Huffman code is designed for these symbols. The output bit stream thus consists (for each RUN-SIZE symbol in the block) of the codeword for the RUN-SIZE symbol

followed by any additional bits required to exactly specify the non-zero coefficient terminating the run of zeros.

The following overhead was required to implement this algorithm:

- $\lceil \log_2(\Delta_{max} - \Delta_{min} + 1) \rceil$ bits to specify the optimal step size for each of the K sources
- the frequency of occurrence of each of the RUN-SIZE symbols

The frequency information was output in the following way. If the frequency of occurance of a RUN-SIZE symbol was non-zero a 1 was output followed by a 13 bit number representing the frequency information; otherwise, a 0 was output.

4.3.4 Encoding Procedure

This section outlines the complete encoding procedure for all OBIFS coders implemented in this work. The proposed fractal video coding model shown in Figure 4.1 will once again be used as our reference.

For all OBIFS coders implemented, the following encoder parameters were specified a priori:

- the motion compensation algorithm, including MC block size and search size
- the domain pool source with associated spatial contraction operator and horizontal and vertical grid displacements Δx, Δy
- the fixed-basis vectors (only one was used)
- the basis generation method
- the mean-squared-error threshold, T_{mse}

Given that the above parameters have been specified, the encoding procedure can be described. The encoding procedure can be divided into two parts: DFD signal generation and fractal encoding of the resulting DFD signal.

The DFD signal generation part was already described in Section 4.2.4. The second part of the encoding procedure is fractal encoding of the output DFD signal. The first step in the encoding process is to generate the set of orthonormal basis vectors (matrix **A**) as described in Section 4.3.2; 8×8 range blocks R_i were used throughout. For each R_i, the average energy-per-pixel is computed; if it is less than T_{mse}, R_i is set to a block of zero-intensity (such a block will be coded by the EOB symbol). These are blocks for which the motion compensation algorithm has found a good match. All blocks R_i are then transformed using the matrix **A** to form a set of transform coefficients. Finally, the coefficients are quantized and entropy coded using the method described in Section 4.3.3.

At this time it should be noted that if the domain pool source is the RBC, decoding is governed by the contraction mapping fixed point theorem. This implies that the domain pool from which the final set of orthonormal basis vectors are formed must be generated iteratively at the decoder. For this to occur, it is required that the encoding transformation be contractive. Preliminary simulations run on two DFD signals showed the encoding transformations were not contractive; i.e., the decoded image did not possess a true fixed point. In general, there is no guarantee that the encoding transformations will be contractive (Vines 1995) and, the contractivity condition is very difficult to check during the encoding as it depends on the orthonormal basis vectors generated. For these reasons, the RBC was not considered as a possible domain pool source for the OBIFS class of coders.

4.3.5 Decoding Procedure

The previous section discussed the encoding procedure for OBIFS coders. This section outlines the decoding procedure.

The decoder has the following information available a priori:

- the motion compensation algorithm and MC block size

- the domain pool source and associated spatial contraction operator
- the horizontal and vertical grid displacements Δx, Δy
- the order in which the R_i were encoded; this ordering never changes and is defined a priori
- the basis generation method
- the fixed-basis vectors (only one was used)

From the encoded output files (bit streams), the optimal step sizes for each source, the frequency information needed to reconstruct the RUN-SIZE symbols Huffman table and the indices of the domain blocks needed to reconstruct the basis vectors can be determined. Using this information the orthonormal basis is generated and the Huffman table constructed.

As mentioned earlier, we considered only the PCR and PCO domain pool sources with this class of coders; using these domain pool sources, decoding is non-iterative and requires only a single iteration as all domain blocks used in generating the basis are available a priori.

Once the basis is generated, decoding simply involves decoding each of the RUN-SIZE symbols and reconstructing the quantized transform coefficients. The range blocks, $\mathbf{r}_i$ are then reconstructed as $\mathbf{r}_i = \mathbf{A}\mathbf{x}_i$ where $\mathbf{A}$ is the matrix whose columns are the orthonormal basis vectors, and $\mathbf{x}_i$ is the vector of reconstructed transform coefficients.

The resulting reconstructed DFD signal is then added to the previously reconstructed original frame (using the motion vectors) to form an approximation of the current frame. This procedure is repeated until the entire video sequence has been decoded.

Chapter 5

Simulation Results and Analysis

This chapter presents the results of extensive simulations that were performed to quantify the effects that various parameters of the affine and OBIFS class of coders have on coding performance. Based on these results, the best coders in each class are selected for comparison to a reference DCT-based coder. The two video sequences used to perform these simulations were the **pongi** and **foreman** video sequences.

5.1 Source Descriptions

Pongi is a relatively high motion sequence of two men playing ping-pong. There is considerable motion as the camera pans horizontally back and forth following the ball. The background consists of a stationary region, a detailed poster on the back wall and a few spectators watching the game. The first eleven original frames (frames $0 - 10$) were used; each frame had dimensions 240×360, and the frame rate was 30 frames per second.

Foreman is a low to moderate motion sequence of a construction worker standing in front of a building and talking. The motion is concentrated around the man's head, shoulders and mouth as he talks. To increase the amount of motion in this sequence, the first 50 original frames were decimated temporally by 5. The resulting frames

were then enumerated $0-10$. Each frame had dimensions 144×176 and the resulting frame rate after decimation was 6 frames per second.

Typical frames from each of these sequences are shown in Figures 5.1 and 5.2.

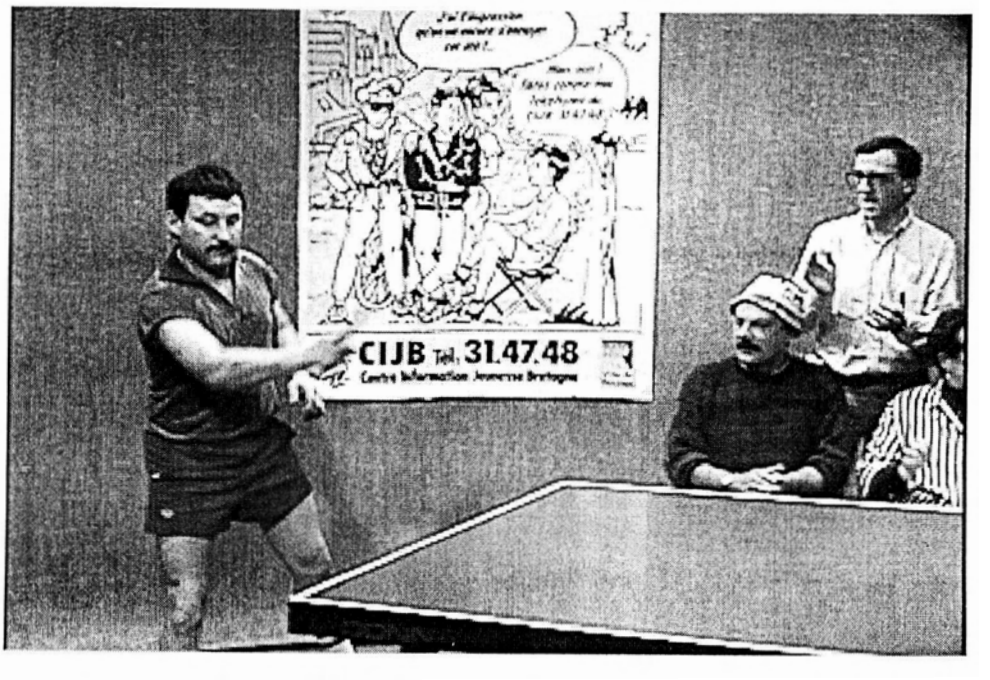

Figure 5.1: Pongi Sequence

Figure 5.2: Foreman Sequence

To avoid bombarding the reader with many duplicate results, results are only provided for the **pongi** sequence. Results are provided for the **foreman** sequence only if they provide additional insights beyond what can be deduced from the **pongi** results. In addition, PSNR results are only provided for the last 6 frames (frames $5 - 10$) of the video sequences. This is done to ensure that the results displayed are steady-state results truly indicative of the parameter being investigated; i.e., any effects due to the use of original frames at start-up are eliminated. The PSNR results represent the PSNR calculated between the original video sequence frame and a reconstruction of that frame using the encoded DFD signal.

5.2 Affine-Transform-Based Coders

This section quantifies through simulation the effects that various parameters of the affine-transform-based fractal coders have on coding performance. The **pongi** and **foreman** video sequences introduced in Section 5.1 were used for this investigation. Based on the simulation results, quantization and entropy coding strategies were defined, and a representative best set of completely quantized coders was selected for comparison to a reference DCT-based coder.

Unless otherwise stated, the following is assumed:

- the standard block-based motion compensation algorithm is used; the MC block size is 8 and the search window over which an optimal matching blocks is sought is ± 15 (as is standard in MPEG, H.261)
- the standard affine transformation is used for encoding non-uniform range blocks
- the standard quadtree partitioning algorithm is used (any partitioning algorithm could have been chosen while investigating other parameters)
- all parameters are unquantized as initially their distributions are unknown
- the mean-squared-error threshold, T_{mse}, is equal to 40; this value was chosen so that there would be a large fraction of blocks that would have to be coded; i.e., we did not want the motion compensation algorithm to find good matches for all the blocks

5.2.1 Domain Pool Source Related Results

We begin this empirical investigation by quantifying the effect that the spatial contraction operator has on coding performance. The results are illustrated in Figures 5.3, 5.4 and 5.5 for the RBC, PCR and PCO domain pool sources.

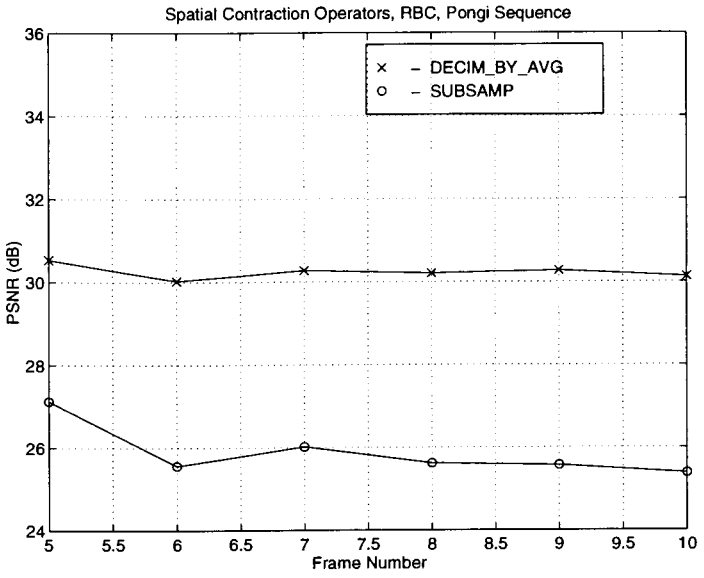

Figure 5.3: Pongi: Spatial Contraction Operator Comparison, RBC

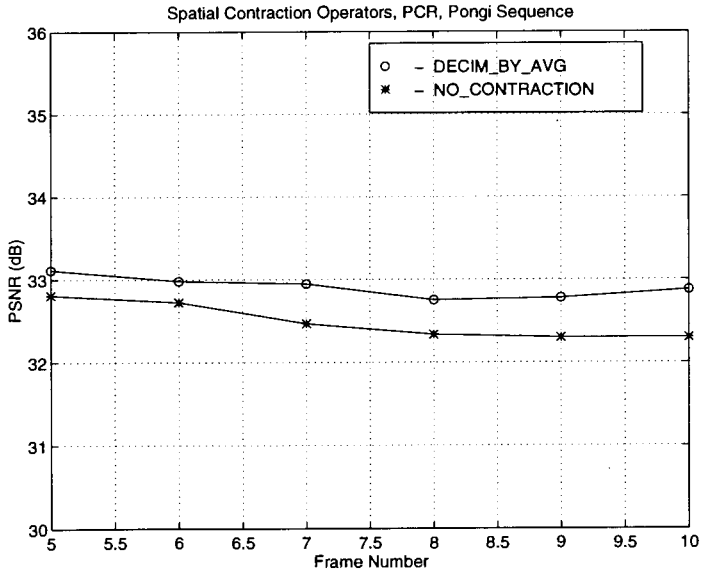

Figure 5.4: Pongi: Spatial Contraction Operator Comparison, PCR

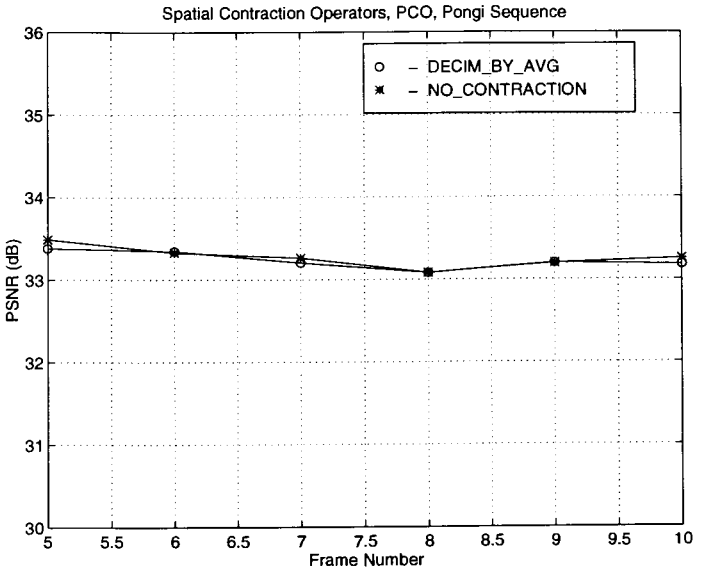

Figure 5.5: Pongi: Spatial Contraction Operator Comparison, PCO

These results indicate that the DECIM_BY_AVG spatial contraction operator provides much better performance than the SUBSAMP operator for the RBC domain pool source. This result is not surprising as the DECIM_BY_AVG operator acts as a low-pass filter and combines more block information when forming the spatially contracted domain block; each pixel in the spatially contracted domain block is obtained using a four pixel average. The SUBSAMP operator performs straight decimation, and thus results in aliasing. The DECIM_BY_AVG spatial contraction operator is most often used in practice for coding of still images; the SUBSAMP operator is rarely used. The SUBSAMP operator was not investigated any further for either of the PCR and PCO domain pool sources.

The results in Figure 5.4 indicate that for the PCR domain pool source, the DECIM_BY_AVG spatial contraction operator provides marginally better performance than the NO_CONTRACTION operator. For the PCO domain pool source, the DECIM_BY_AVG and NO_CONTRACTION operators both provide equivalent performance.

Combining these results, Figure 5.6 illustrates the effect of the selected domain pool source on the coding performance for the DECIM_BY_AVG spatial contraction operator.

It is very clear from these results that the PCR and PCO domain pool sources provide much better performance; this is due to the fact that there are no constraints on the encoding transformations. The performance of the PCO domain pool source is marginally better than that of the PCR. This is probably due to the fact that the PCR domain pool source generally has fewer domain blocks in its domain pool as zero-intensity blocks are not included. These zero-intensity blocks can be quite common as they are produced whenever a good match has been found by the motion compensation algorithm (see Section 4.2.4). Subjectively, the 3 dB PSNR difference is noticeable (slightly) in the reconstructed image frames.

We investigate next the effect that the horizontal and vertical grid displacements,

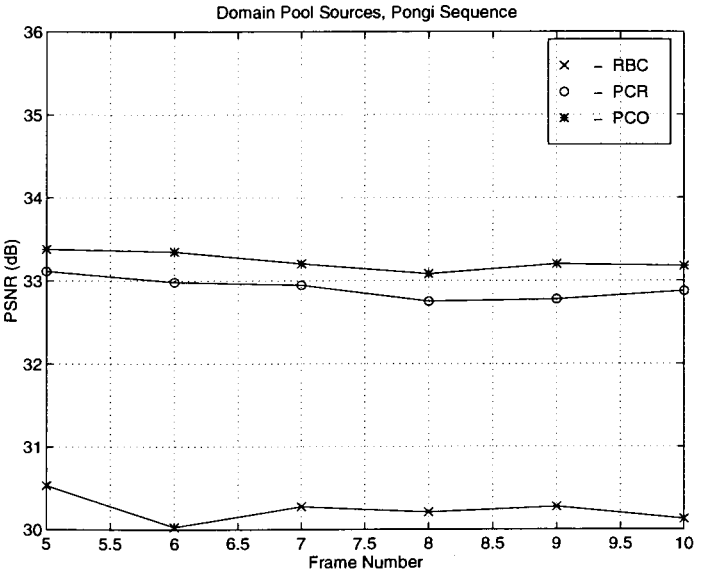

Figure 5.6: Pongi: Domain Pool Source Comparison

Δx and Δy, used in the domain pool construction procedure have on coding performance. Intuitively, it was expected that for smaller values of (Δx, Δy) the performance would be much better as smaller values of (Δx, Δy) result in a larger number of domain blocks in the domain pool thus increasing the likelihood of finding a satisfactory encoding transformation. Nevertheless, quantifying the performance loss due to the use of a smaller sized domain pool is important since specifying the domain block used for encoding a given non-uniform range block requires a large number of bits, approximately half of the total number of bits used to represent the encoding transformation.

Figures 5.7, 5.8 and 5.9 display the results obtained for horizontal and vertical grid displacements $(\Delta x, \Delta y)$ of $(4,4)$, $(8,8)$ and $(16,16)$. The results are displayed for the RBC, PCR and PCO domain pool sources.

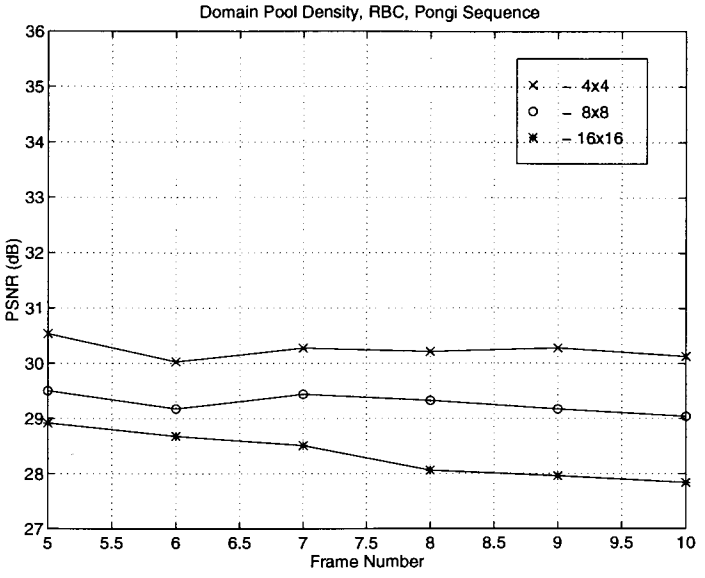

Figure 5.7: Pongi: Domain Pool Density Comparison, RBC

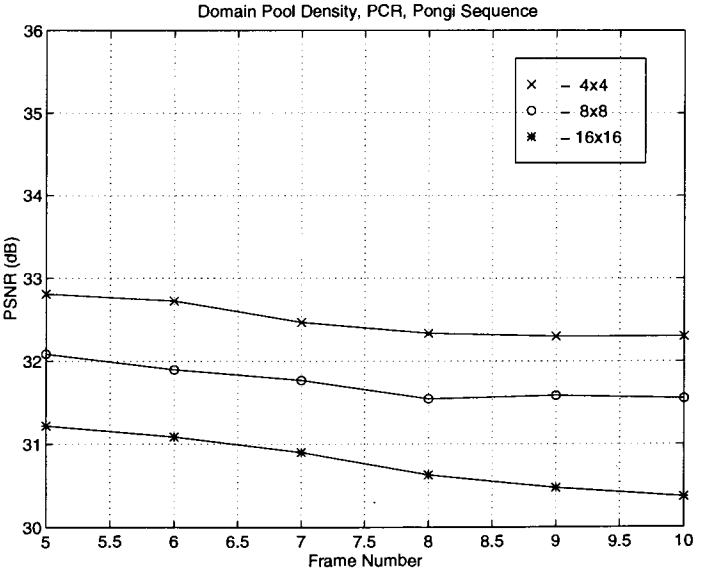

Figure 5.8: Pongi: Domain Pool Density Comparison, PCR

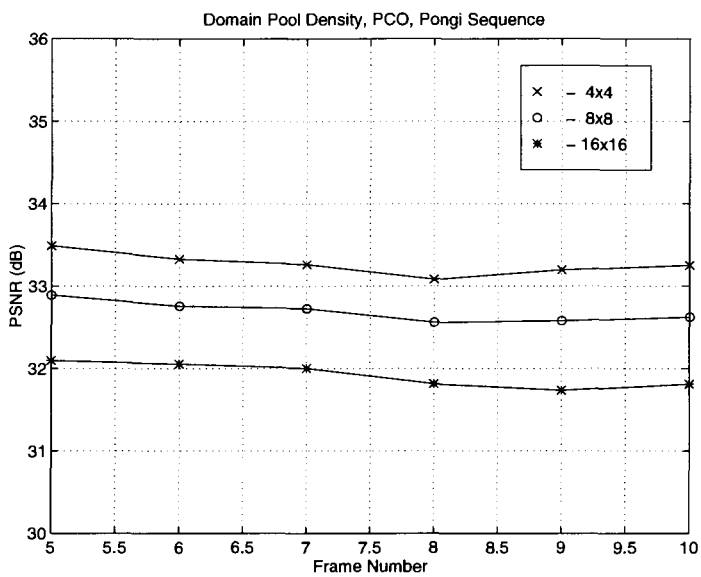

Figure 5.9: Pongi: Domain Pool Density Comparison, PCO

As expected, the PSNR decreases as the horizontal and vertical grid displacements increase. The actual PSNR loss will in general depend on the DFD signal being coded; for the **pongi** sequence displayed, the PSNR loss was approximately 1 dB for each doubling of $(\Delta x, \Delta y)$. The PSNR loss was less than 0.5 dB for the **foreman** sequence.

Based on the results presented thus far, the following general conclusions can be made:

- the DECIM_BY_AVG spatial contraction operator yields the best PSNR performance for all domain pool sources
- the PCR and PCO domain pool sources widely outperform the RBC domain pool source in PSNR terms
- reducing the domain pool size reduces the PSNR

5.2.2 Parameter Distribution Results

We now turn our attention to the parameter distributions; namely, we investigate the distributions of the alpha, beta and absorb parameters and describe the quantization and entropy coding strategies that were used to represent them. The parameter distributions are dependent on the domain pool source, thus we consider all three domain pool sources in turn.

Figures 5.10, 5.11 and 5.12 display the distributions of the alpha, beta and absorb parameters for the RBC domain pool source.

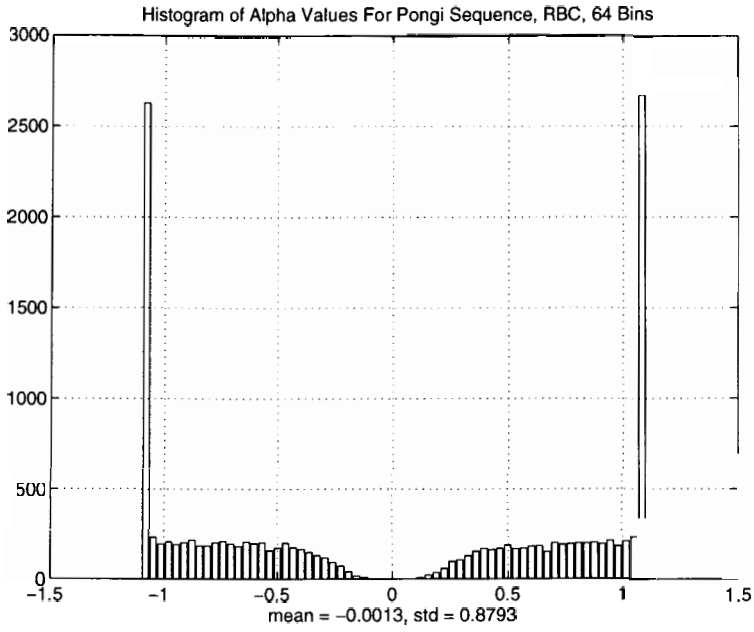

Figure 5.10: Pongi: Histogram of Alpha Values, RBC, 64 Bins

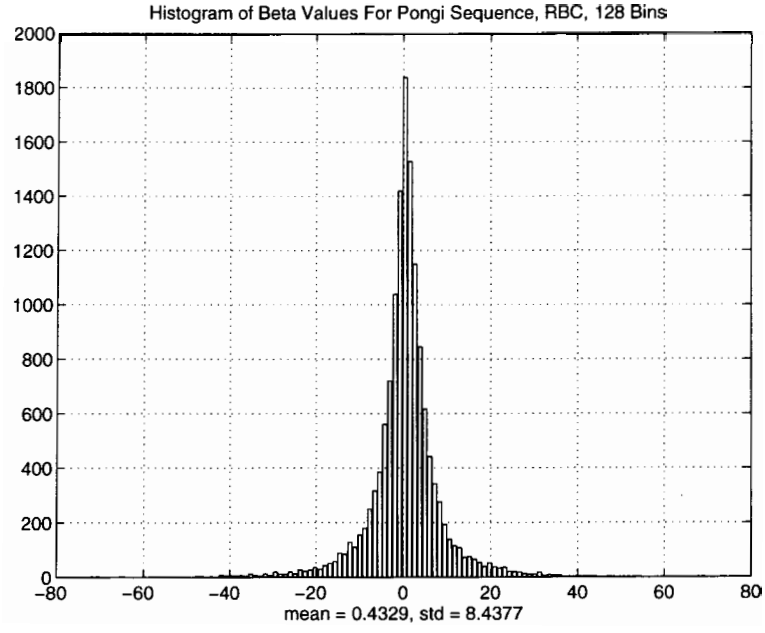

Figure 5.11: Pongi: Histogram of Beta Values, RBC, 128 Bins

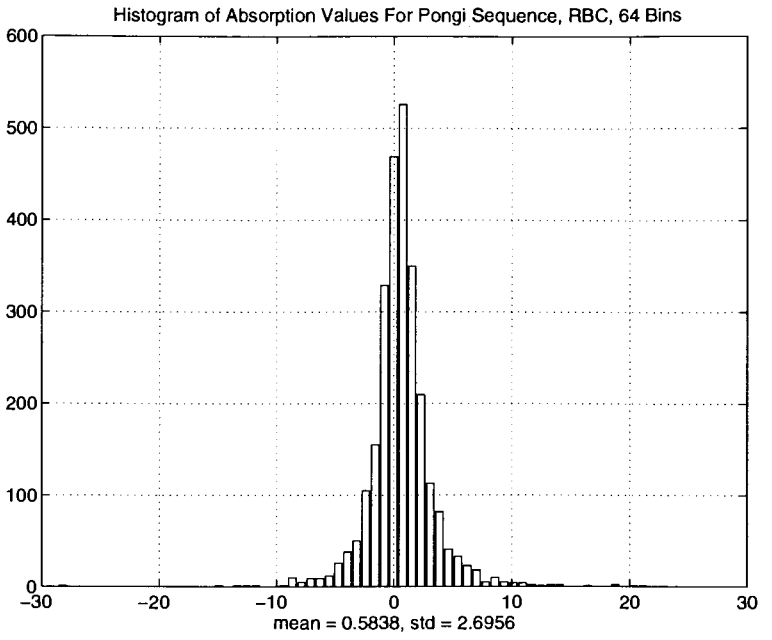

Figure 5.12: Pongi: Histogram of Absorb Values, RBC, 64 Bins

Very noticeable from Figure 5.10 is the large number of α values concentrated at ± 1.1, the maximum and minimum allowable α values. The form of this distribution indicates that larger α values would improve the PSNR performance for the RBC domain pool source, but the contractivity requirement prevents larger α values from being used. The same type of distribution was observed for **foreman**. Due to the fixed nature of this distribution (the distribution shape remained constant with most of the α values concentrated at the minimum and maximum allowable values), fixed Lloyd-max quantizers were used to quantize the α values, and fixed Huffman tables were used for entropy coding of the resulting output quantization indicies. The quantizers were designed using Lloyd's iterative design algorithm; all the α values generated from coding frames $0 - 10$ of **pongi** and **foreman** were used as a training set. Both 16 and 32 output level quantizers were designed with corresponding Huffman tables; two quantizers were designed so that different target rates could be used for representing the α values (needed later on).

Quantization of the absorb and beta parameters will be described shortly as their

distributions are very similar to those observed for the PCR and PCO domain pool sources.

Figures 5.13 and 5.14 display the alpha parameter distributions for the PCR and PCO domain pool sources. Figure 5.15 displays the beta parameter distribution for the PCO domain pool source. The absorb parameters for the PCR and PCO domain pool sources, and the beta parameter of the PCR domain domain pool source are not shown as these distributions are very similar in shape to the corresponding distributions for the RBC domain pool source.

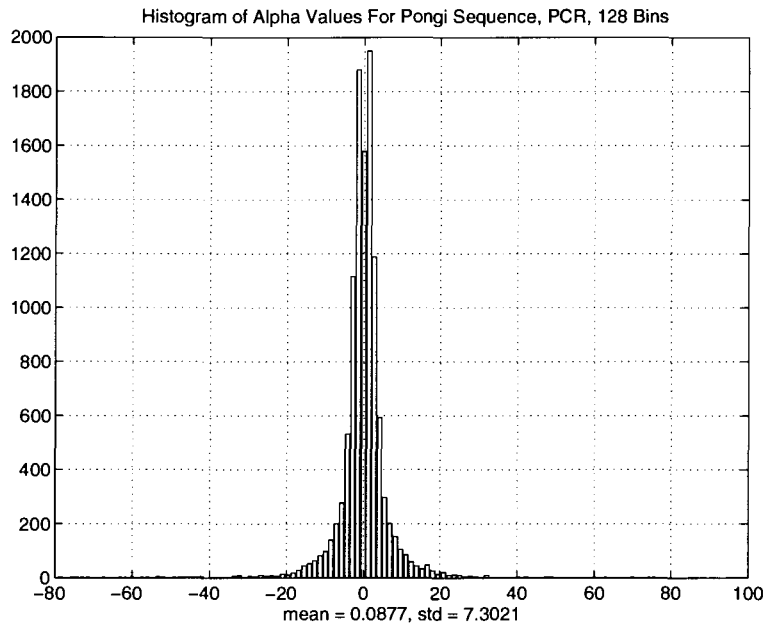

Figure 5.13: Pongi: Histogram of Alpha Values, PCR, 128 Bins

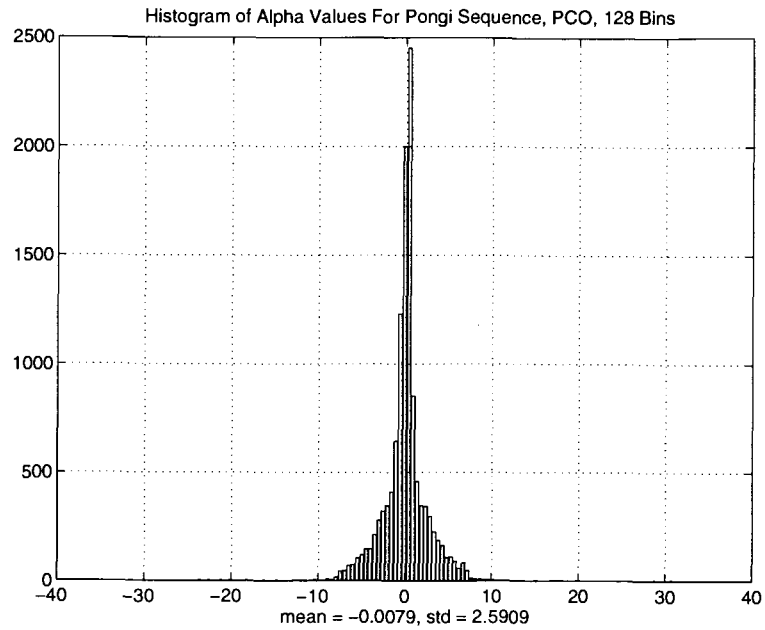

Figure 5.14: Pongi: Histogram of Alpha Values, PCO, 128 Bins

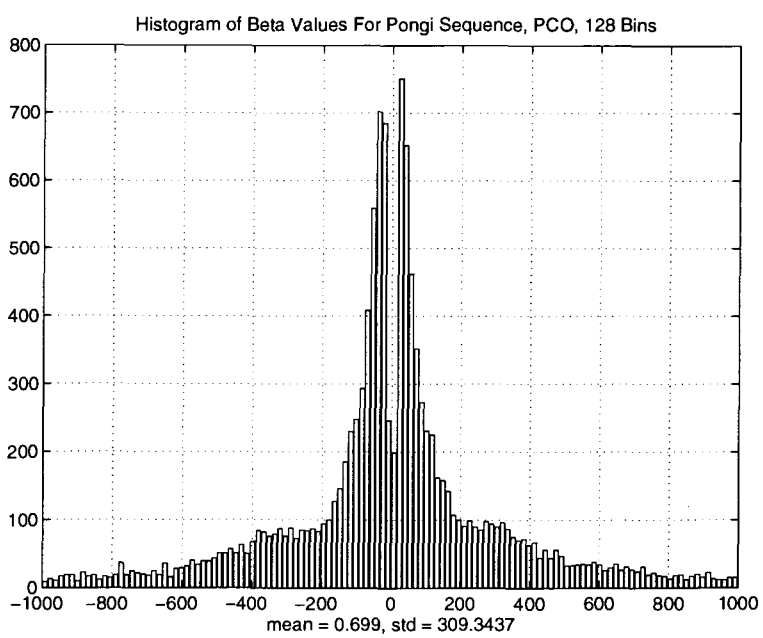

Figure 5.15: Pongi: Histogram of Beta Values, PCO, 128 Bins

The alpha parameter distributions are very similar in shape to the absorb and beta parameter distributions observed thus far. The beta parameter distribution for the PCO domain pool has a very wide range; the standard deviation of this distribution is over 300. This same characteristic was observed for the **foreman** sequence. Efficiently quantizing a set of data with such a distribution would be very difficult as the standard deviation is so large. Therefore, to alter the distribution of the beta parameters while leaving the approximation error of the encoding transformation unchanged, the modified luminance transformation was used. Figures 5.16, 5.17 and 5.18 display the resulting beta parameter distributions for values of a_o equal to 0.5, 0.2 and 0.0. The alpha parameter distributions are left unchanged when the modified luminance transformation is used.

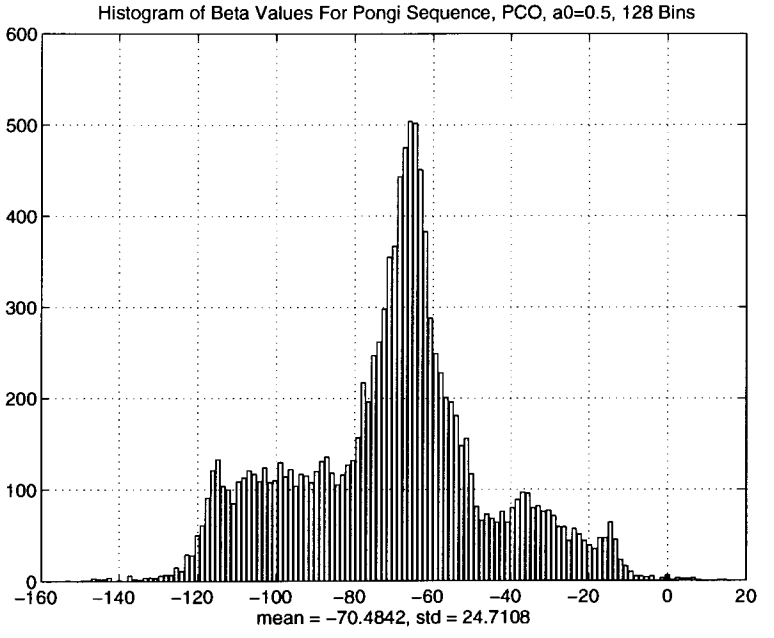

Figure 5.16: Pongi: Histogram of Beta Values, PCO, a0 = 0.5, 128 Bins

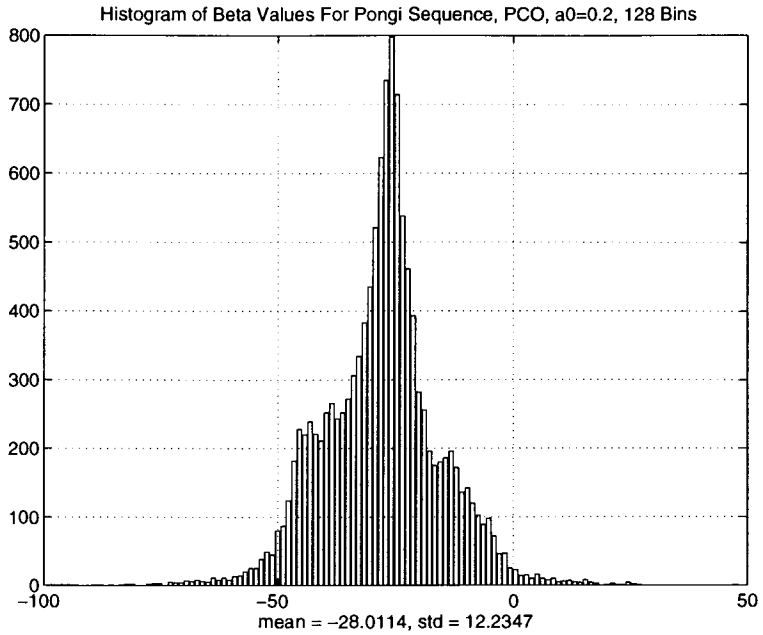

Figure 5.17: Pongi: Histogram of Beta Values, PCO, a0 = 0.2, 128 Bins

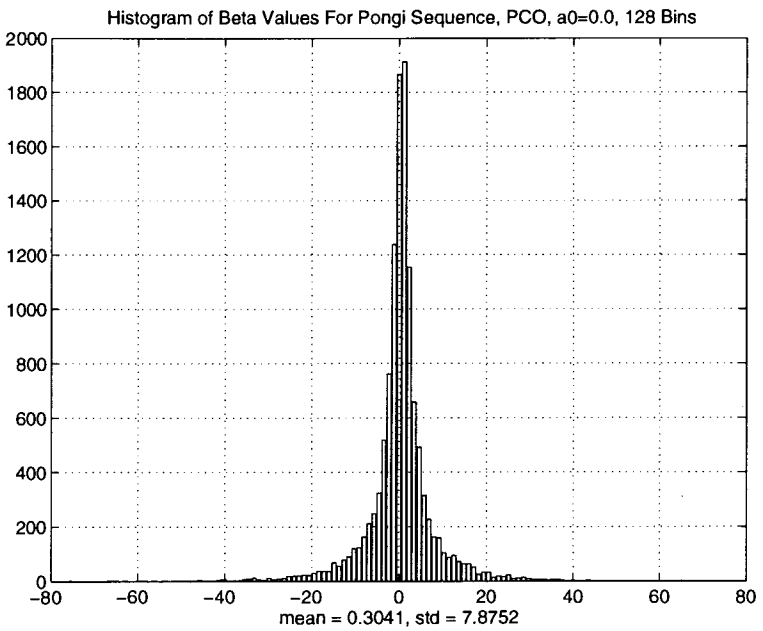

Figure 5.18: Pongi: Histogram of Beta Values, PCO, a0 = 0.0, 128 Bins

It is very clear from these figures that reducing the value of a_o reduces the standard deviation of the beta parameter distribution; $a_o = 0.0$ providing the best result. Using the modified luminance transformation with the PCR domain pool source leaves the beta parameter distribution relatively unchanged; therefore, it was decided to use this transformation with the PCR domain pool source also. The resulting distribution is illustrated in Figure 5.19.

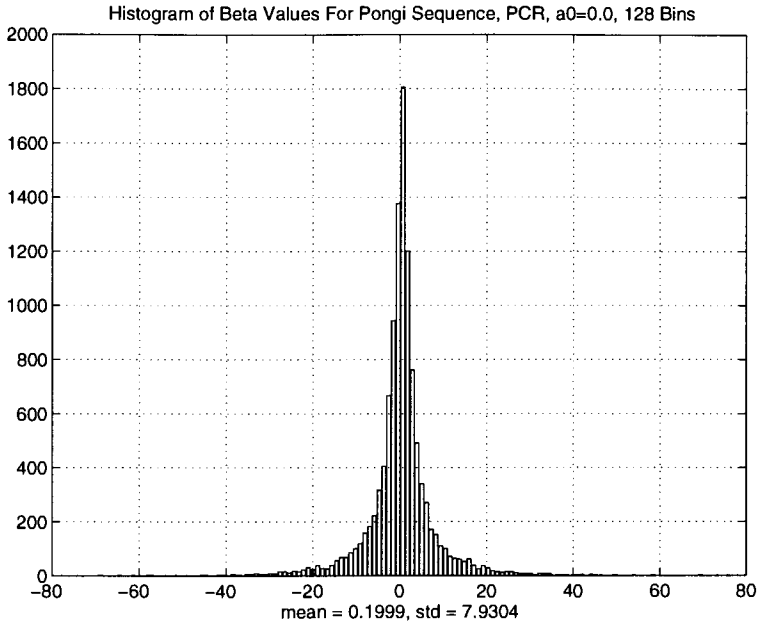

Figure 5.19: Pongi: Histogram of Beta Values, PCR, a0 = 0.0, 128 Bins

Based on the observed parameter distributions, the following conclusions were made:

- the modified luminance transformation should be used for the PCO domain pool source; it will also be used with the PCR domain pool source
- the resulting alpha, beta and absorb parameter distributions for the PCO and PCR domain pool sources and the beta and absorb parameter distributions for the RBC domain pool source all have distribution shapes that can be modeled using a generalized Gaussian pdf

Based on the latter conclusion, it was decided to quantize the above sources using an entropy constrained scalar quantizer and then efficiently design a Huffman code using the generalized Gaussian pdf model of the data. The entropy constrained scalar quantizer would simply be a uniform quantizer with a large number of output levels, N.

To investigate the above method, a large training set of alpha, beta and absorb parameters was obtained by encoding frames $0 - 10$ of **pongi** and **foreman** for both the PCR and PCO domain pool sources.

The following experiment was then performed. For each parameter and for a given number of quantizer output levels, N, a rate-distortion function was generated for that parameter. The computed rate was the average number of bits (average codeword length) required to represent that parameter using a Huffman code designed as described in Section 2.4.1; the parameter distribution is modeled using a generalized Gaussian pdf and the quantizer output index probabilities required to design the Huffman code are obtained by numerically integrating the model over the bins specified by the quantizer. The average distortion is just the average distortion incurred in quantizing the parameters. The rate-distortion functions for all three parameter distributions are shown in Figures 5.20 to 5.22. Note that for a given N, the part of the rate-distortion function we are interested in is that portion starting where the distortion is minimum and extending downwards (increasing distortion).

A characteristic feature of these rate distortion functions is that for a given average rate, a quantizer, Q_1, with N_1 output levels will always yield a lower average distortion than a quantizer, Q_2, with N_2 output levels, if N_1 is greater than N_2.

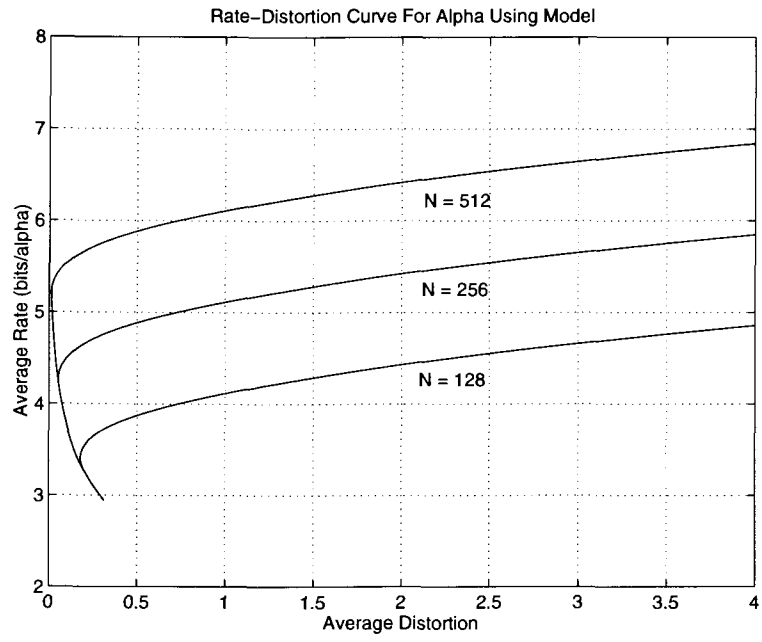

Figure 5.20: Rate Distortion Function For Alpha Parameter

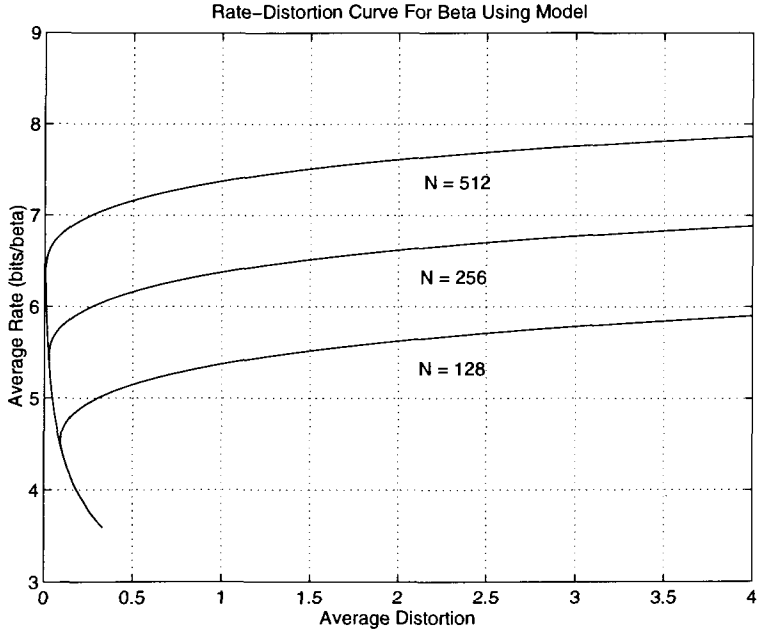

Figure 5.21: Rate Distortion Function For Beta Parameter

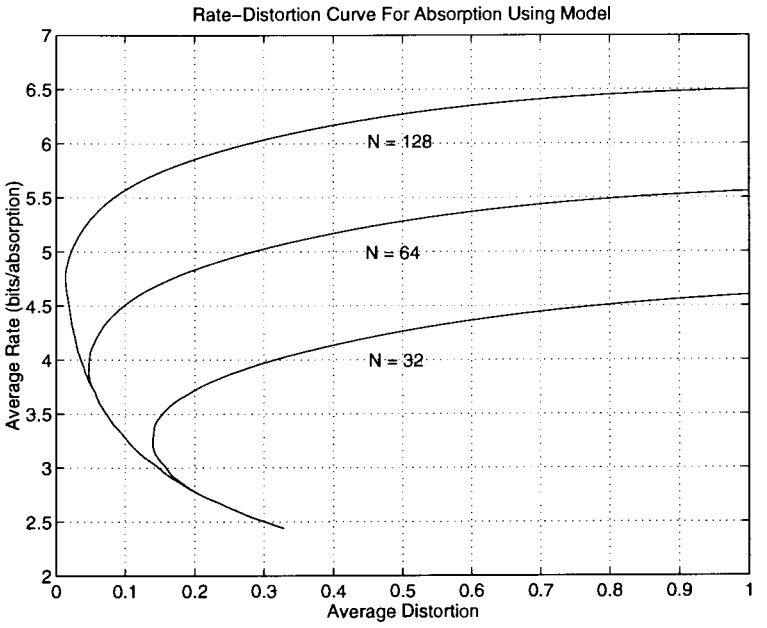

Figure 5.22: Rate Distortion Function For Absorb Parameter

The rate was also computed using frequency information obtained directly from the actual parameter distributions. Both of the computed rates were then compared to the true source entropy. The difference between the computed rates and the true source entropy is called the redundancy; ideally, we would like the redundancy to be zero. The redundancies are displayed in Figures 5.23 through 5.25 for a particular value of N over that portion of the rate-distortion curve where the distortion is minimum and extends downwards (increasing distortion).

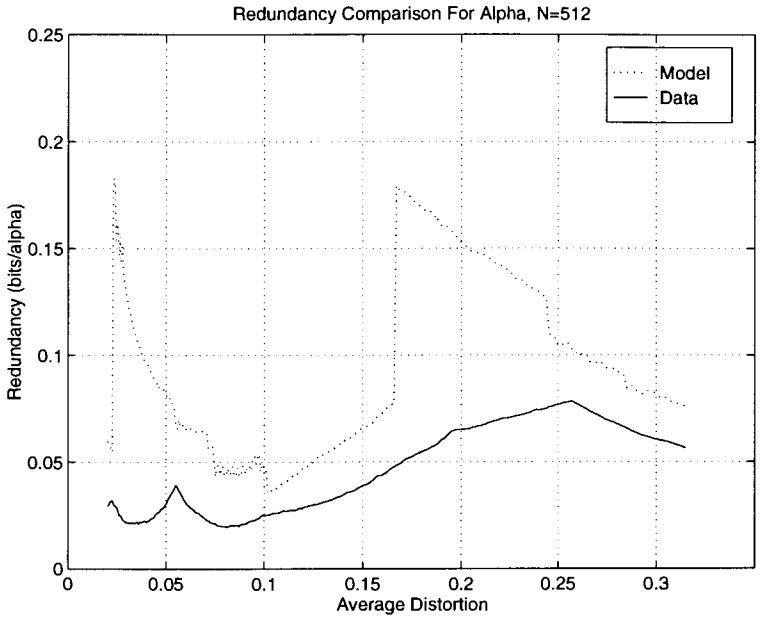

Figure 5.23: Redundancy Comparison For Alpha, N = 512

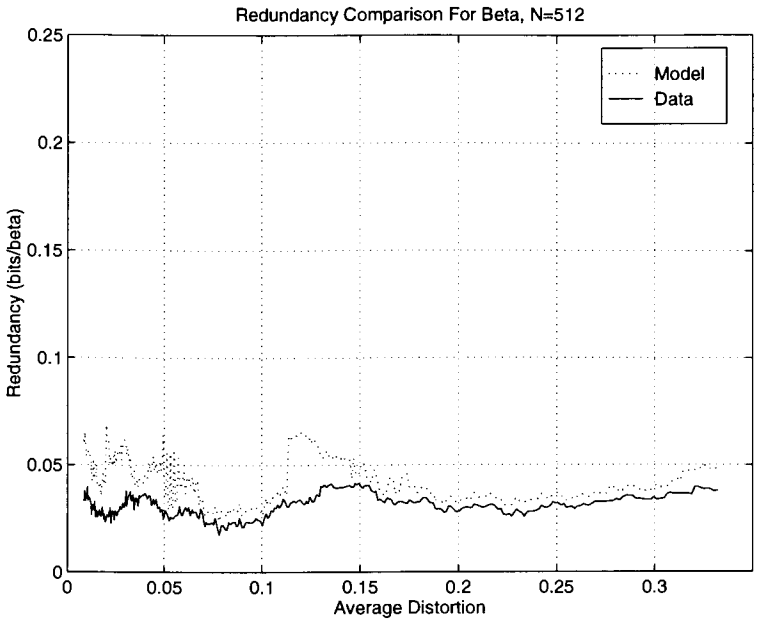

Figure 5.24: Redundancy Comparison For Beta, N = 512

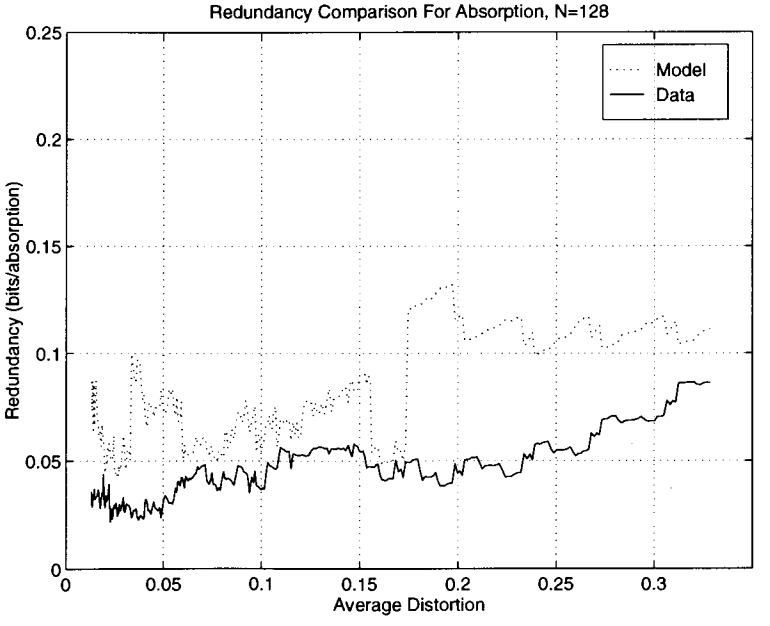

Figure 5.25: Redundancy Comparison For Absorb Parameter, N = 128

From these figures, it can be concluded that by using $N = 512$ for alpha, $N = 512$ for beta and $N = 128$ for absorb, the redundancy is no more than 0.2 bits per parameter value; therefore, the use of entropy constrained scalar quantizers and Huffman codes designed using a model of the data sets is justified. Based on the observed parameter distributions, it was decided to constrain the α values to ± 20 and the β values to ± 60 in the quantized coders; the absorb parameters were not constrained as the spread of the parameters was much narrower.

The actual implemented quantization and entropy coding strategy for the alpha, beta and absorb parameters of the PCR and PCO domain pool sources and the beta and absorb parameters of the RBC domain pool source can now be described. For each parameter distribution (data set), the following steps were performed:

1. Compute and remove the mean from the data set; find that value in the resulting data set farthest from zero, denote this value by x_{max}.

2. Following the procedure described in Section 2.4.1 determine the model parameters for the data set with mean removed.

3. Set the minimum step size to $\Delta_{min} = \frac{2x_{max}}{N}$ where N is the number of quantizer output levels used in quantizing the particular data set. This minimum step size ensures that the quantizer completely spans the data, thereby ensuring that the rate-distortion function generated later will be relatively well behaved; i.e., the BFOS algorithm will start at a point on the rate-distortion curve where the rate is maximum and the distortion is minimum. This is also the reason that the α and β parameters were constrained to ± 20 and ± 60 respectively.

4. Set the step size increment to $\Delta_{inc} = \frac{\Delta_{min}}{4}$.

5. Construct a rate-distortion table to be used by the BFOS algorithm.

6. Run the BFOS algorithm to optimally quantize the three data set parameters at a given specified rate.

We now elaborate on the last two steps. Starting from Δ_{min} with step size increment Δ_{inc} and given a data set with an associated uniform quantizer of N output levels, the rate and distortion are computed for a specified fixed number of quantizers to be designed for that data set. For example, for the α and β parameter distributions 250 quantizers were designed while for the absorb parameter distribution 50 quantizers were designed. The rate was calculated as the entropy of the quantizer output indices where the probabilities of occurrence of each quantizer output index are obtained by numerically integrating the data set model over the bins specified by the quantizer. The distortion function used for a given data set was the average of the actual distortions due to quantization of the particular data set parameter incurred in approximating a range block. These are the D_α, D_β and D_γ expressions derived in Section 4.2.3. The D_β and D_γ distortion functions are simply the quantization errors. The D_α distortion function weights the quantization errors by the variance of the domain blocks used in the particular encoding transformation. Use of these distortion functions minimizes the actual distortion incurred in approximating a given range block rather than simply minimizing the quantization errors.

Given the complete set of rate-distortion functions for each parameter and a specified target rate, the BFOS algorithm is run to determine the optimal quantizer assignments.

It is important to keep in mind that the data sets are quantized with mean removed; therefore, during the decoding process, the mean must be added back to the quantized value to reconstruct an estimate of the orignal parameter value.

The quantization and entropy coding strategy implemented has the following overhead:

- the mean and standard deviation of each data set were quantized using a step size of 0.005 and output using 15 bits (fewer bits could have been used)
- the optimal step size for each data set was quantized using a step size of 0.001 and output using 15 bits (fewer bits could have been used)

- the ν model parameter for each data set was output using 16 bits

Using this information, the decoder can reconstruct the exact same Huffman tables that were used at the encoder.

Also pertaining to encoding transformation parameter distributions are the distributions of the isometry indices. A typical distribution is shown in Figure 5.26 for the RBC domain pool source. Similar distributions were observed for all domain pool sources.

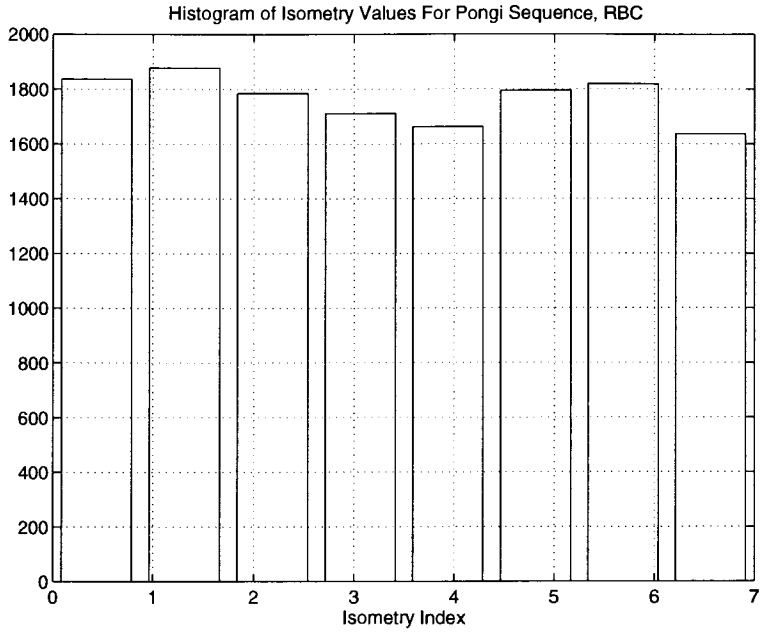

Figure 5.26: Pongi: Distribution of Isometries, RBC

Due to the uniform nature of the isometry index distributions, no entropy coding was performed.

5.2.3 Motion Compensation Results

The next parameter to be examined is the motion compensation algorithm used to generate the DFD signals. Both the standard and overlapped windowed block-based motion compensation methods were examined for the RBC, PCR and PCO domain

pool sources. Figures 5.27, 5.28 and 5.29 illustrate the comparative results for all three domain pool sources.

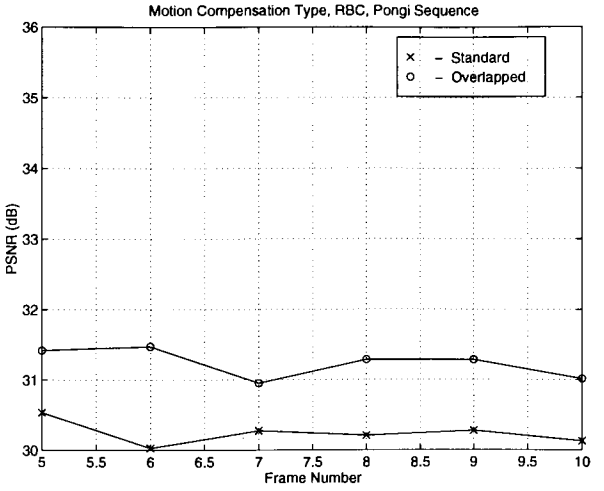

Figure 5.27: Pongi: Motion Compensation Type Comparison, RBC

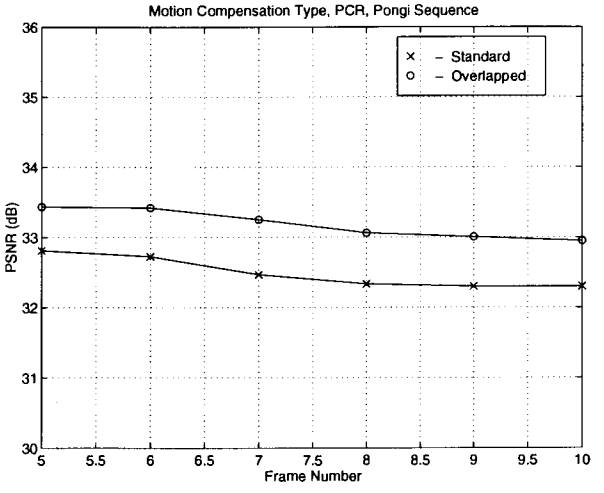

Figure 5.28: Pongi: Motion Compensation Type Comparison, PCR

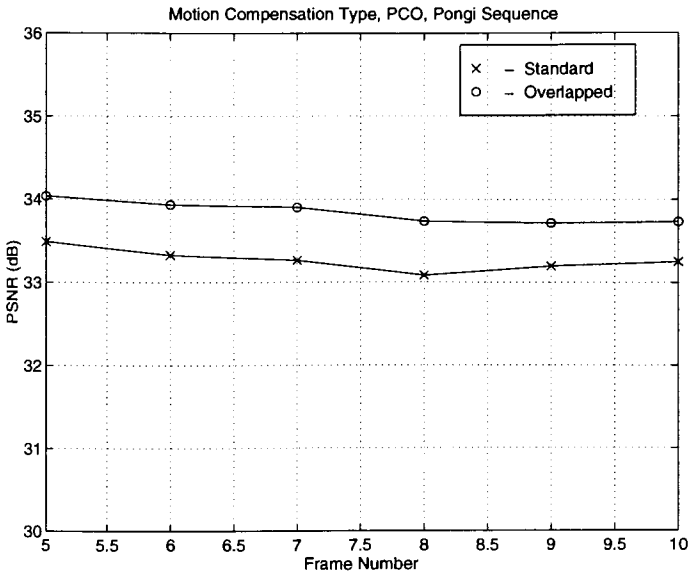

Figure 5.29: Pongi: Motion Compensation Type Comparison, PCO

In all three cases, the use of overlapped windowed block motion compensation, which produces smoother, lower energy DFD signals, provided better performance; similar results were observed for **foreman**. It should be noted that the shape of the resulting parameter distributions is not altered by use of the overlapped windowed motion compensation algorithm.

Table 5.1 lists the standard deviations of all parameter distributions for both standard block-based and overlapped windowed block motion compensation algorithms for each of the domain pool sources. The parameter distributions were obtained by encoding frames $0-10$ of the **pongi** and **foreman** video sequences using the corresponding domain pool source.

The results indicate that, in general, the standard deviation of the alpha parameter distributions is lower when the overlapped windowed motion compensation algorithm is used. The standard deviations of the beta and absorb parameter distributions are left relatively unchanged.

Table 5.1: Effect of MC Type on Parameter Distributions

Sequence	DPS	MC Type	σ_α	σ_β	σ_γ
Pongi	PCO	Standard	4.73	7.88	2.71
Pongi	PCO	Windowed	3.92	7.50	2.89
Pongi	PCR	Standard	7.31	7.93	2.69
Pongi	PCR	Windowed	4.72	7.56	2.69
Foreman	PCO	Standard	7.98	6.91	3.18
Foreman	PCO	Windowed	3.90	7.55	3.14
Foreman	PCR	Standard	4.62	6.83	3.23
Foreman	PCR	Windowed	4.17	7.21	3.10

5.2.4 Partitioning Algorithm Results

The next parameter investigated was the partitioning algorithm. Complete quantized coders using overlapped windowed block motion compensation were used for this investigation. It was desired to investigate the PSNR versus bit rate tradeoff with all other parameters apart from the partitioning algorithm being equal. To this end, the target rate specified for quantization of the alpha, beta and absorb parameters was set at 4.5 for the standard quadtree and HV partitioning algorithms; a target rate of 3.5 was used for the partial quadtree algorithm as the overhead required to specify the partitioning was observed (through simulation) to be approximately 1 bit more than that required for the standard quadtree and HV algorithms when the split information was entropy coded.

Simulations were run over frames $0 - 10$ of **pongi** and **foreman** for all three domain pool sources. The average PSNR and the average bits-per-pixel (bpp) were obtained by averaging the PSNR's and bit rates over frames $5 - 10$ (steady-state) in each simulation. Tables 5.2 and 5.3 summarize the results obtained.

Table 5.2: Partitioning Algorithm Comparison For Pongi

DPS	Partitioning Type	Avg. PSNR (dB)	Avg. Bpp
RBC	Quadtree	31.17	0.50
RBC	Partial Quadtree	31.10	0.51
RBC	HV	31.05	0.50
PCR	Quadtree	33.44	0.45
PCR	Partial Quadtree	33.34	0.46
PCR	HV	33.36	0.44
PCO	Quadtree	33.82	0.44
PCO	Partial Quadtree	33.55	0.46
PCO	HV	33.67	0.43

Table 5.3: Partitioning Algorithm Comparison For Foreman

DPS	Partitioning Type	Avg. PSNR (dB)	Avg. Bpp
RBC	Quadtree	33.38	0.29
RBC	Partial Quadtree	33.40	0.33
RBC	HV	33.33	0.28
PCR	Quadtree	34.49	0.27
PCR	Partial Quadtree	34.53	0.30
PCR	HV	34.39	0.26
PCO	Quadtree	34.58	0.27
PCO	Partial Quadtree	34.36	0.31
PCO	HV	34.39	0.26

The results indicate that all partitioning algorithms yield essentially equivalent results; for the same average bit rate, the average PSNR is the same and vice-versa. Based on this observation, the standard quadtree partitioning algorithm, with its minimal overhead, was selected as the prefered partitioning algorithm.

Finally, we investigated the PSNR loss that results as the target rate specified for quantization of the alpha, beta and absorb parameters is varied. Two DFD signals obtained from the **pongi** sequence were used for this investigation. The results are displayed in Tables 5.4 and 5.5 for the PCR and PCO domain pool sources respectively.

The results are normalized such that zero PSNR loss corresponds to a target rate of 5.0; all PSNR changes are quoted with respect to the PSNR at the target rate of 5.0. The end rate is the actual rate obtained using the quantization and entropy coding strategy implemented. Futhermore, there were approximately 600 α, β values and 140 γ (absorb) values in each of the DFD signals.

Table 5.4: PSNR Change as a Function of Target Rate, PCR

Target Rate	End Rate	PSNR Change
5.0	4.97	0.0
4.5	4.37	-0.05
4.0	3.96	-0.10
3.5	3.51	-0.23
3.0	3.00	-0.57
2.5	2.53	-1.22

Table 5.5: PSNR Change as a Function of Target Rate, PCO

Target Rate	End Rate	PSNR Change
5.0	4.98	0.0
4.5	4.39	-0.07
4.0	4.04	-0.13
3.5	3.50	-0.30
3.0	2.84	-1.05
2.5	2.55	-1.69

The results indicate that a target rate of 3.5 is sufficient for quantization of the alpha, beta and absorb parameters.

5.2.5 Summary

Based on all the results presented in this section, affine-transform-based fractal coders with the following parameters were selected for comparison to a reference DCT-based coder:

- overlapped windowed motion compensation algorithm
- standard quadtree partitioning algorithm
- PCR and PCO domain pool sources
- DECIM_BY_AVG spatial contraction operator
- modified luminance transformation for encoding non-uniform range blocks
- entropy constrained scalar quantizers for quantizing the α ($N = 512$), β ($N = 512$) and γ ($N = 128$) values; BFOS algorithm for optimal quantizer assignments; generalized Gaussian pdf modeling of the parameter distributions for Huffman code design

In summary, it should be understood by the reader that fractal coders are heuristic in nature; their ability to encode a particular type of signal can only be determined through extensive simulation. The simulations in this section were performed in order to obtain some general insight on the effects that various parameters of the affine-transform-based coders have on the coding performance so that logical decisions could be made on what parameters were more likely to produce superior PSNR results.

5.3 OBIFS Coders

This section quantifies through simulation the effects that various parameters of the OBIFS coders have on coding performance. The **pongi** and **foreman** video sequences introduced in Section 5.1 were used for this investigation. Based on the simulation results, a set of coders was selected for comparison to a reference DCT-based coder.

Unless otherwise stated, the following is assumed:

- the standard block-based motion compensation algorithm is used; the MC block size is 8 and the search window over which an optimal matching block is sought is ± 15 (as is standard in MPEG, H.261)

- $(\Delta x, \Delta y) = (2, 2)$
- the target rate specified for **pongi** is 0.4; the target rate specified for **foreman** is 0.25
- the spatial contraction operator is NO_CONTRACTION (either could have been used as shown later)
- the mean-squared-error threshold, T_{mse}, is equal to 40 (for the same reasons as described earlier for the affine transform case)

Fully quantized coders have to be considered because using the PCR and PCO domain pool sources in the absence of quantization will result in perfect signal reconstruction; recall that the RBC domain pool source was not considered for the OBIFS class of coders because preliminary simulations showed the encoding transformation to be non-contractive.

5.3.1 Basis Generation Method Related Results

The first parameter investigated was the basis generation method. Figures 5.30 and 5.31 display the results obtained for the PCR and PCO domain pool sources.

The results indicate that the covariance method provides marginally better performance for both domain pool sources. Similar results were observed for **foreman**. The centroid method was used in all subsequent simulations because the encoding time is much shorter, and its performance is nearly equivalent to that of the covariance method.

Combining these results Figure 5.32 compares the PCR and PCO domain pool sources using the centroid basis generation method.

It can be seen that the PCO domain pool source provides slightly better performance over most of the sequence.

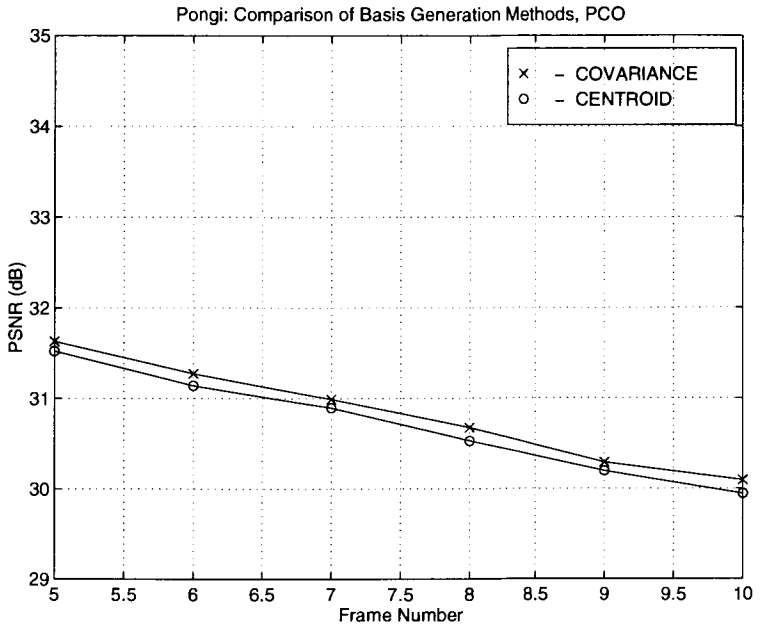

Figure 5.30: Pongi: Basis Generation Method Comparison, PCO

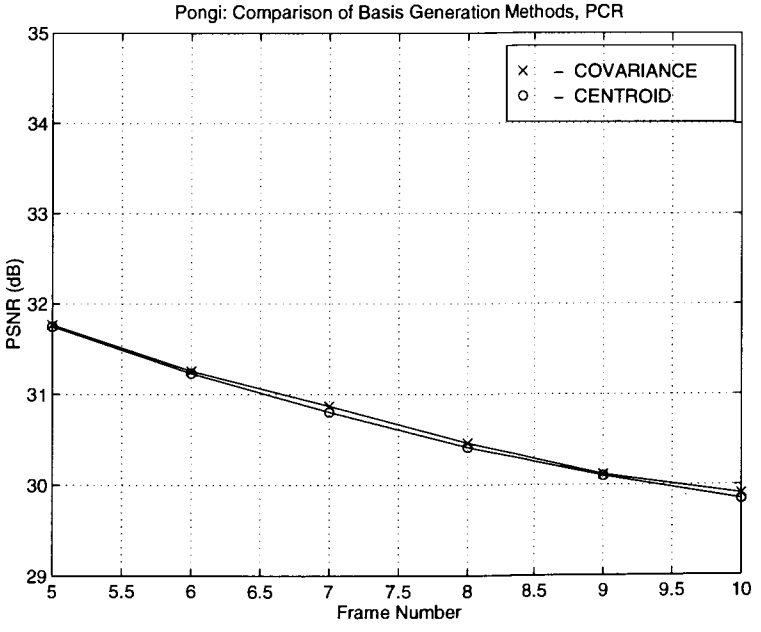

Figure 5.31: Pongi: Basis Generation Method Comparison, PCR

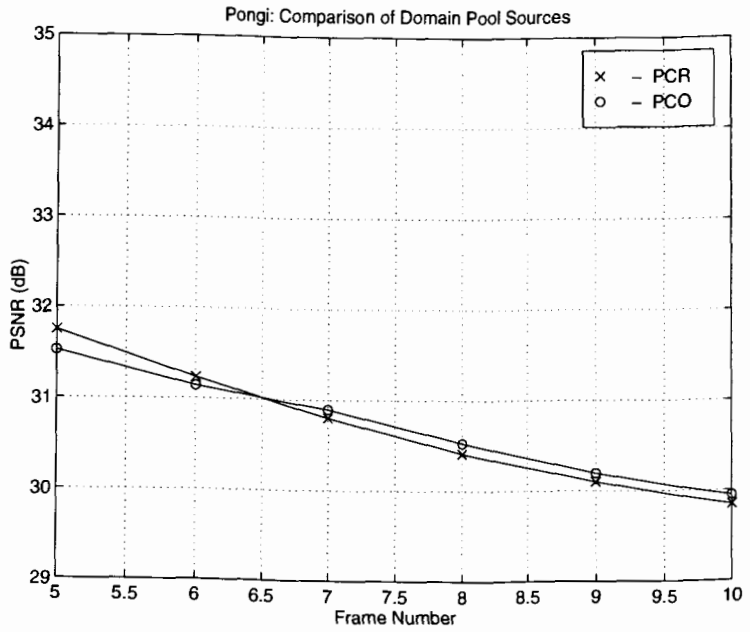

Figure 5.32: Pongi: Domain Pool Source Comparison

5.3.2 Motion Compensation Results

Figures 5.33 and 5.34 summarize the effect that the motion compensation algorithm has on the coding performance for the PCR and PCO domain pool sources.

It is very clear that use of the overlapped windowed block motion compensation algorithm provides superior performance; similar results were observed for the **foreman** sequence. This is again due to the smoother, lower energy DFD signal produced by the overlapped windowed block motion compensation algorithm.

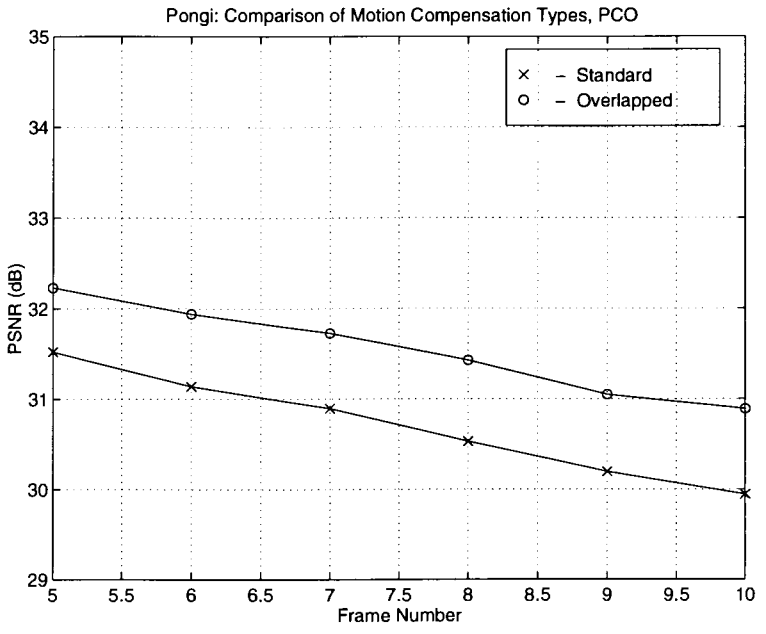

Figure 5.33: Pongi: Motion Compensation Type Comparison, PCO

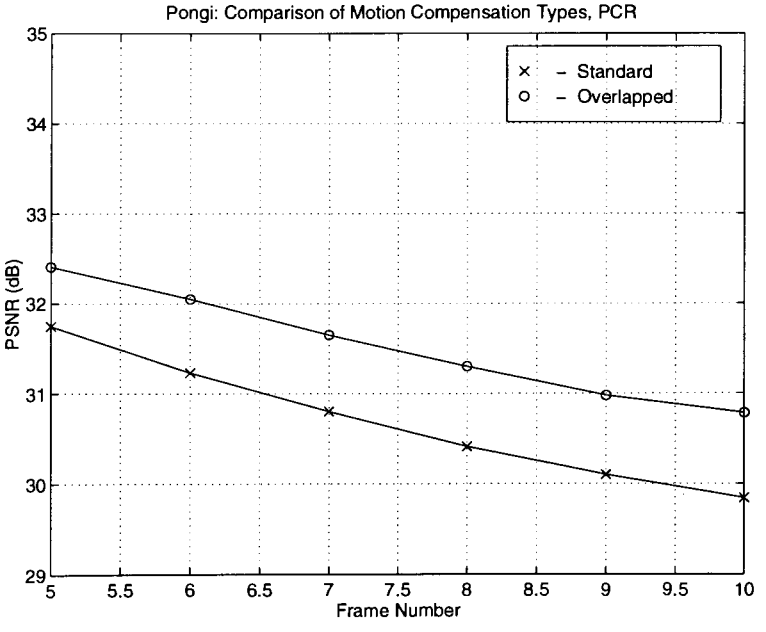

Figure 5.34: Pongi: Motion Compensation Type Comparison, PCR

5.3.3 Spatial Contraction Operator Results

Finally, the effect of the spatial contraction operator on coding performance was investigated. There is no reason to believe that either of the operators considered, DECIM_BY_AVG and NO_CONTRACTION, should yield different results. The results are presented in Figures 5.35 and 5.36 for the **pongi** and **foreman** sequence respectively.

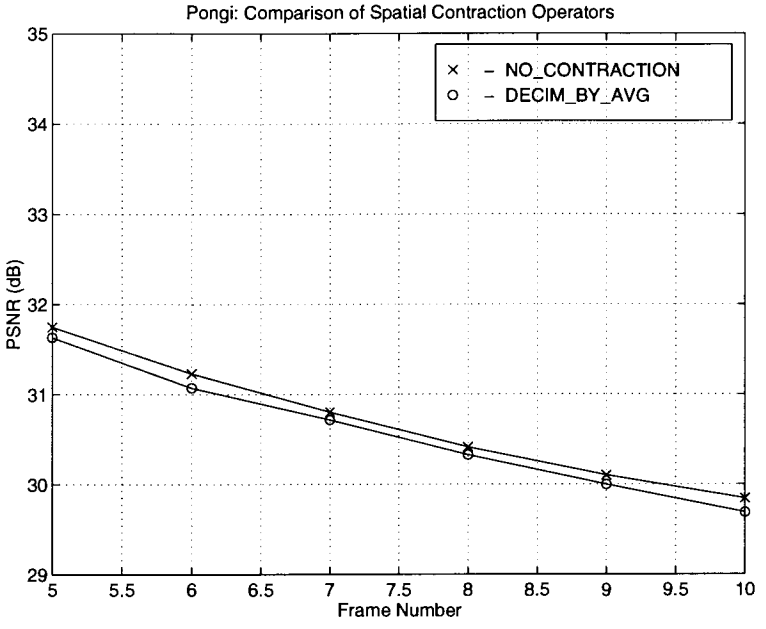

Figure 5.35: Pongi: Spatial Contraction Operator Comparison

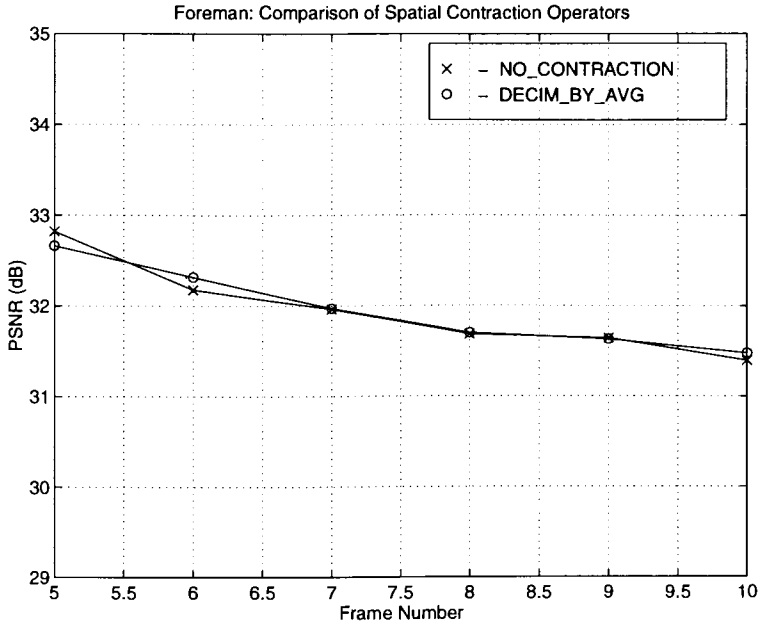

Figure 5.36: Foreman: Spatial Contraction Operator Comparison

As expected, both operators yield essentially the same performance.

5.3.4 Summary

Based on the results presented in this section, OBIFS coders with the following parameters were selected for comparison to a reference DCT-based coder:

- overlapped windowed motion compensation algorithm
- covariance basis generation method
- PCR and PCO domain pool sources
- DECIM_BY_AVG spatial contraction operator
- quantization and entropy coding as per JPEG

Either spatial contraction operator could have been used as both produced equivalent results. The DECIM_BY_AVG spatial contraction operator was selected to be consistent with the affine-transform-based fractal coders.

Chapter 6

Final Results

This section presents and discusses the simulation results obtained by comparing the performance of the affine transform and OBIFS coders selected from Chapter 5 to a reference DCT-based coder.

6.1 Source Descriptions

The **carphone** and **salesman** video sequences were used for comparing the objective and subjective performance of the various coders.

Carphone is a low to moderate motion sequence of a man sitting in a moving car. To increase the amount of motion in this sequence, the first 100 frames were decimated temporally by 5; the resulting frames were enumerated $0-19$. Each frame had dimensions 144×176, and the resulting frame rate after decimation was 6 frames per second.

Salesmen is a moderate motion sequence of a man sitting at a desk holding an object in his right hand. The background is somewhat detailed and the motion is concentrated in the man's right arm and head as he moves the object up and down. The amount of motion in this sequence was also increased by decimating the first 60 frames temporally by 4; the resulting frames were enumerated $0-15$. Each frame had

dimensions 288 × 352, and the resulting frame rate after decimation was 7.5 frames per second.

Typical frames from each of these two sequences are shown in Figures 6.1 and 6.2

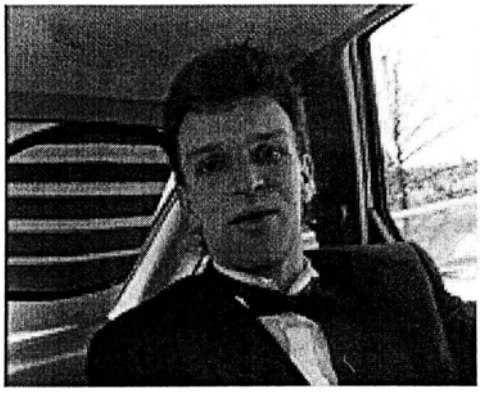

Figure 6.1: Carphone Sequence

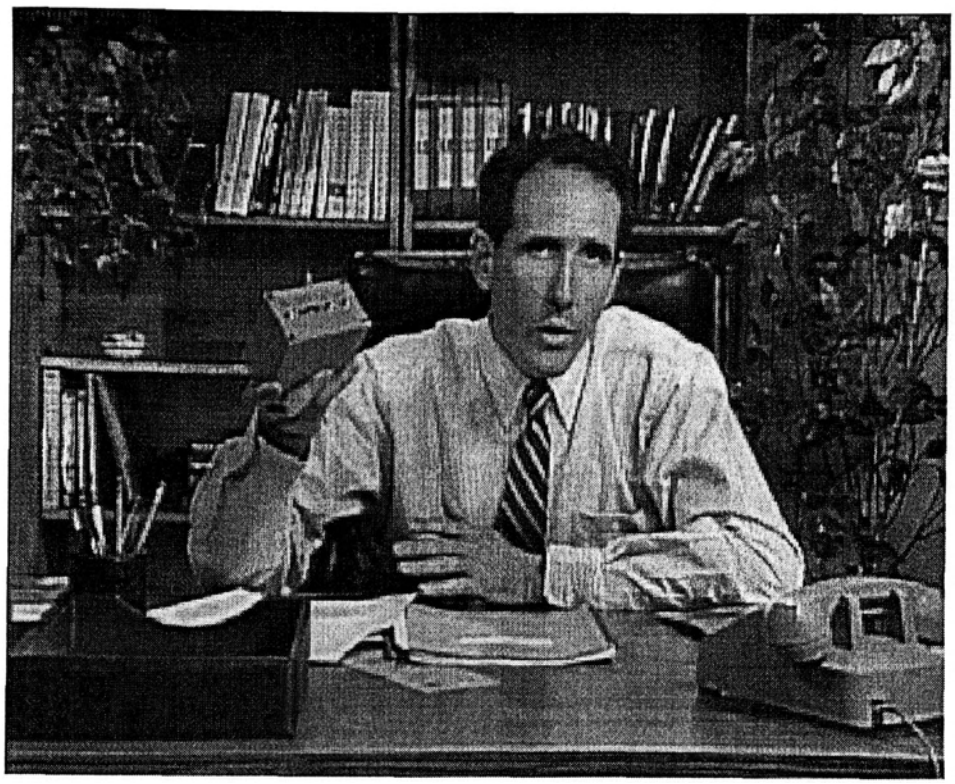

Figure 6.2: Salesman Sequence

6.2 Results

The reference DCT-based coder implemented for comparison utilizes the exact same quantization and entropy coding strategy described in Section 4.3.3 for OBIFS coders. The only difference is that the DCT coefficients are zig-zag scanned in the standard DCT zig-zag scan order. The functionality of the DCT coder was verified by implementing a still image version of the coder and comparing the results to a JPEG coder for coding of the standard **Lena** image (approximately 0.5 dB better than JPEG at 0.6 bpp). Finally, the overlapped windowed block motion compensation algorithm was used for the DCT coder as it was used for all other coders as well.

We now summarize the fractal coders that were selected in the previous chapter

as being the best representative coders in their corresponding classes. For the affine transform class of coders, we used the overlapped windowed block motion compensation algorithm, the DECIM_BY_AVG spatial contraction operator, the standard quadtree partitioning algorithm and both the PCR and PCO domain pool sources. For the OBIFS class of coders, we used the overlapped windowed block motion compensation algorithm, the covariance basis generation method, the DECIM_BY_AVG spatial contraction operator and the PCR and PCO domain pool sources. The coders were quantized and entropy coded as described in the previous chapter.

The mean-squared-error threshold, T_{mse} was kept constant at 50.0 to eliminate this variable from the comparison and to allow our target bit rates to be achieved. Similarly, 3.5 bits was used as the target rate for quantization of the α, β and γ values as this was shown previously to be sufficient. Furthermore, 16×16 blocks were used for performing motion compensation and the search size was fixed at ± 15. The use of 16×16 blocks increases the amount of energy in the DFD signals.

All PSNR measurements were calculated between the original frames and the reconstruction of those frames using the encoded DFD signals. All bit rate measurements exclude the bits required to code the motion vectors.

In addition, it was ensured that the first DFD signal coded was the same for both the PCR and PCO domain pool sources. For this reason, all results are presented from frame 1 to the sequence end. Also, the first 2 frames from each sequence were coded at approximately 32 dB using JPEG.

The aim of this final set of simulations was to encode the selected video sequences at two different rates so that some general conclusions could be made on the feasibility of using fractal video coding techniques for direct coding of DFD signals.

The affine transform and OBIFS fractal coders were each compared individually to the reference DCT-based coder at two separate rates. We begin by presenting the results obtained when comparing the affine-transform-based fractal coders with the DCT-based coder; it should be noted that to make the comparison fair, zero intensity

blocks were coded using a single bit as was done for the affine transform coders.

Since the bit rate of the affine transform coders can not be finely controlled (the algorithm is adaptive in nature and therefore the rate is sequence dependent), the **carphone** and **salesman** sequences were first coded using the affine transform coders. The average rates were then computed in each case, and the DCT coder was used to code the same sequences at those average rates. In this way, we compare the two coding methods at the same average rate.

To generate a higher rate coding, a horizontal and vertical grid displacement of 4 was used for **carphone** and 8 was used for **salesman**. A larger value was used for **salesman** because the image is twice the size; using a value of 8 implies that the number of domain blocks in the domain pool is the same for both sequences.

To produce a lower rate coding, no adaptive partitioning was performed; i.e., all range blocks were coded at the 8×8 block size.

The **carphone** sequence was coded at average rates of approximately 0.26 and 0.13 bpp; the **salesman** sequence was coded at approximate average rates of 0.15 and 0.09 bpp. The results obtained are presented in Figures 6.3 to 6.6.

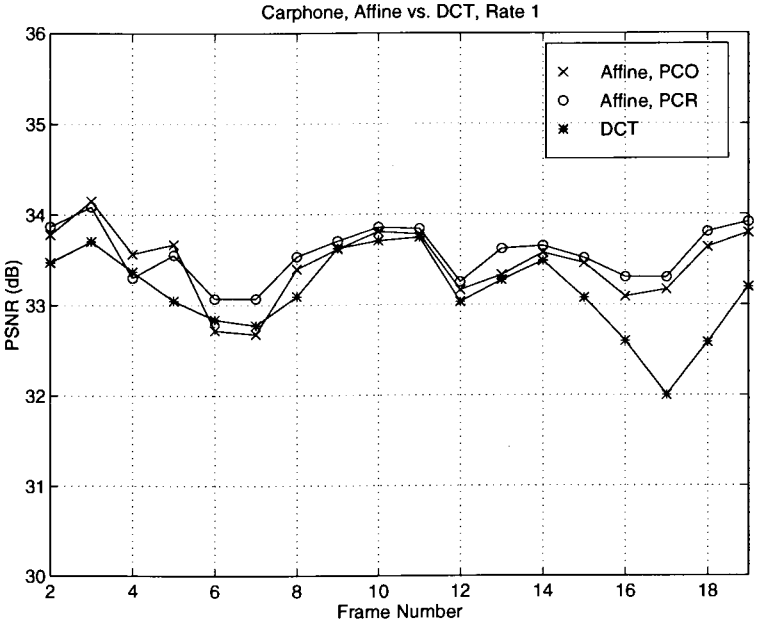

Figure 6.3: Carphone, Affine, 0.26 bpp Average Rate

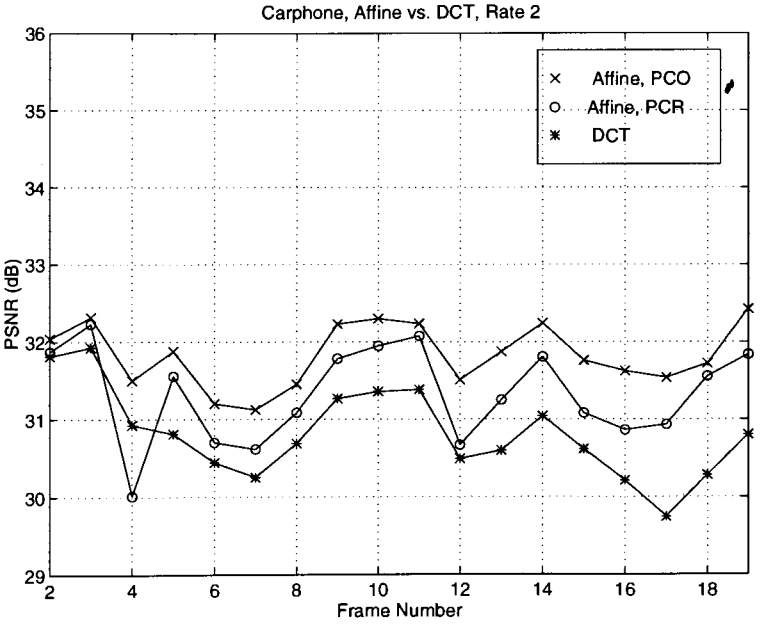

Figure 6.4: Carphone, Affine, 0.13 bpp Average Rate

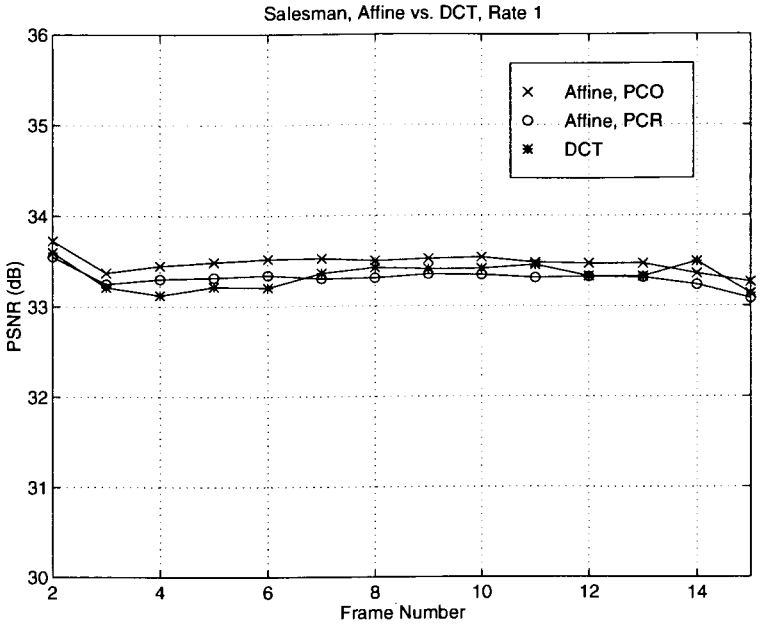

Figure 6.5: Salesman, Affine, 0.15 bpp Average Rate

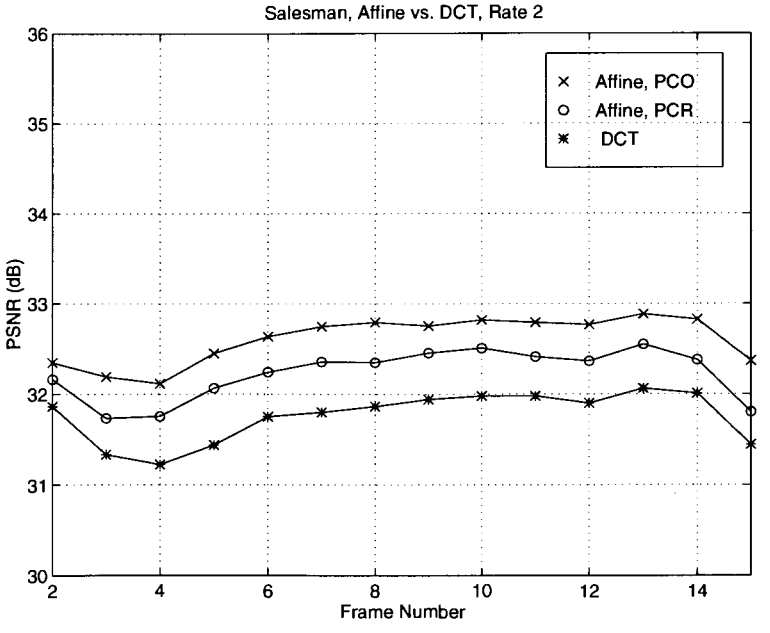

Figure 6.6: Salesman, Affine, 0.09 bpp Average Rate

Based on these results, the following general conclusions can be made:

- For **carphone** at 0.26 bpp average rate and **salesman** at 0.15 bpp average rate, the affine transform coders perform as well or better than the DCT coder over all frames. Subjectively, both coders exhibit blocking artifacts, contouring (noisy type regions of different intensity), and blurring of regions of motion.

- At the lower rates of 0.13 bpp for **carphone** and 0.09 bpp for **salesman**, the best affine transform coder performs up to 1 dB better than the DCT coder over all frames. Subjectively, the DCT coder exhibits much more contouring and granular type noise than does the affine transform coder. Blocking artifacts are more noticeable for both coders.

- The PCO domain pool source always yields results as good or better than the PCR domain pool source; the PSNR gain being more noticeable at the lower rates.

The OBIFS coders were next compared to the reference DCT-based coder. In comparing these two coders, the quantization and entropy coding strategies were identical; the only difference being the basis vectors. Again the coders were compared at the same average rates.

Horizontal and vertical grid displacements of 2 were used for **carphone** while 4 was used for **salesman** for the same reasons described earlier.

The **carphone** sequence was coded at an average rate of approximately 0.26 bpp. The **salesman** sequence was coded at average rates of approximately 0.16 bpp and 0.11 bpp. The results are presented in Figures 6.7 to 6.9.

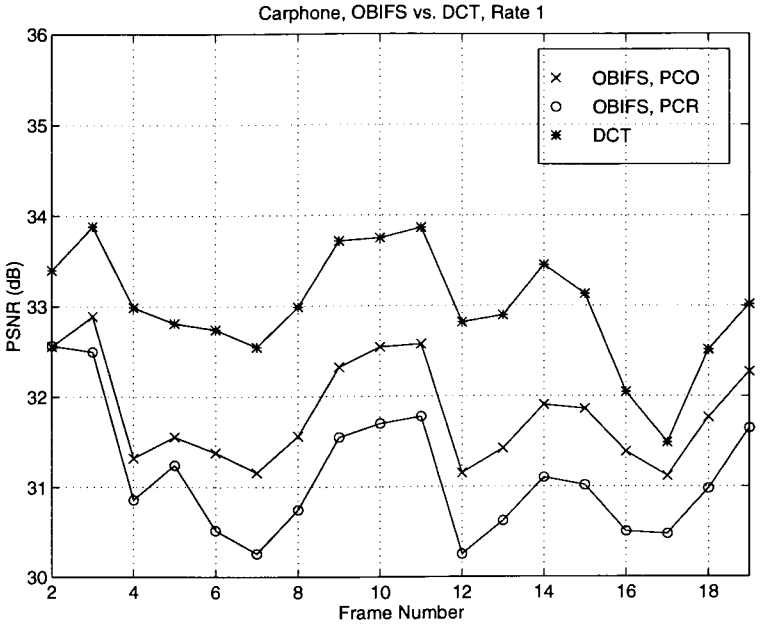

Figure 6.7: Carphone, OBIFS, 0.26 bpp Average Rate

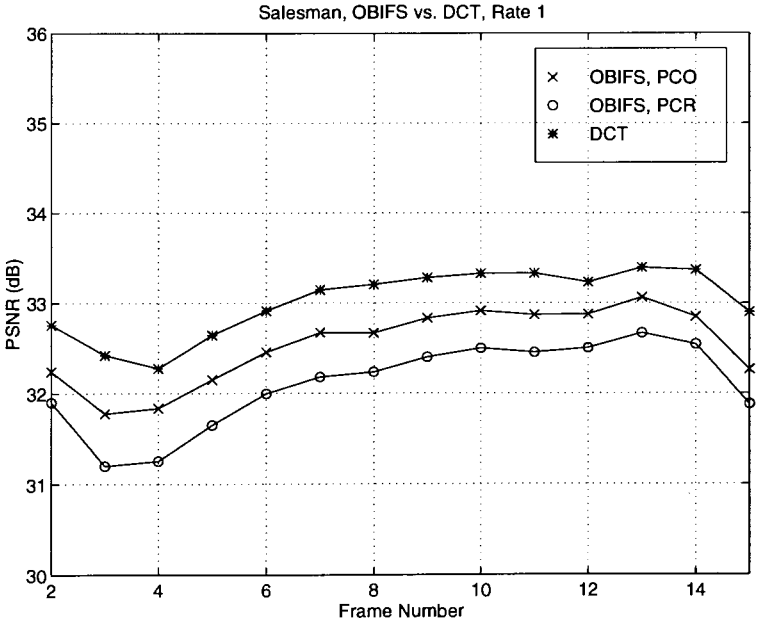

Figure 6.8: Salesman, OBIFS, 0.16 bpp Average Rate

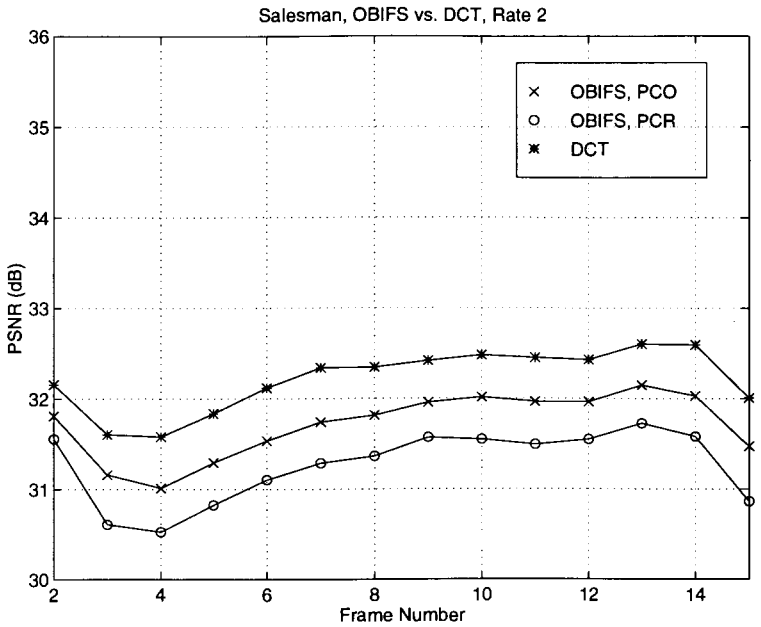

Figure 6.9: Salesman, OBIFS, 0.11 bpp Average Rate

Based on these results, the following general conclusions can be made:

- The DCT coder performs between $0.5-2$ dB better than the best OBIFS coders over all frames for both sequences. Subjectively, the OBIFS coders exhibit a mosquito-type noise spread all over the regions of motion; the DCT coders exhibit more blocking and contouring.

- The PCO domain pool source provides on average approximately 0.5 dB improvement over the PCR domain pool source over all frames for both sequences.

Chapter 7

Conclusion

This thesis investigated the feasibility of using fractal image coding techniques for direct fractal coding of displaced difference signals.

The contributions of the thesis are three-fold:

- An empirical study on direct fractal coding of displaced frame difference signals was presented. Both standard affine and OBIFS coders were considered in the investigation. Quantitative and qualitative results were provided for coding of moderate to high energy DFD signals.
- Thorough descriptions and analysis of all transformation parameters were provided; the effects of all transformation parameters on coding performance were quantified.
- Both classes of coders considered were generalized for video coding by introduction of alternative domain pool sources. A scheme for efficient quantization and entropy coding of the resulting transformation parameters was presented. The resulting coders provided much better performance than classical fractal coding of the DFD signals alone.

Based on the results of this investigation, it can be concluded that affine-transform-based fractal coders are feasible for direct fractal coding of moderate to high energy

DFD signals. For the two video sequences considered, the objective performance of the affine-transform-based fractal coders was as good or better than the reference DCT-based coder for the two sequences tested. It was observed that the PSNR difference increased at lower rates. The OBIFS coders produced objective results $0.5 - 2$ dB worse than the reference DCT-based coder for both sequences tested.

Further work should be done to verify the above conclusions using a larger set of video sequences. Furthermore, the methods considered in this thesis should be compared against other fractal video coding methods.

References

Auyeung, C., James Kosmach, Michael Orchard and Tino Kalafatis. (1992). Overlapped Block Motion Compensation. *Proc. of SPIE Conf. on Visual Communications and Image Processing.* Vol. 1818, 561-571.

Barakat, M. and J.L. Dugelay. (1996). Image Sequence Coding Using 3-D I.F.S. *Proc. of IEEE International Conference on Image Processing.* Vol. I, 141-144.

Barnsley, Michael F. (1988). *Fractals Everywhere.* Toronto: Harcourt Brace Jovanovich.

Barnsley, Michael F. and Lyman P. Hurd. (1993). *Fractal Image Compression.* Wellesley, Massachusetts: AK Peters Ltd.

Barthel, K.U., J. Schüttemeyer, T. Voyé and P. Noll. (1994). A New Image Coding Technique Unifying Fractal and Transform Coding. *IEEE Int. Conf. on Image Processing.*

Barthel, K.U. and T. Voyé. (June). Adaptive Fractal Coding in the Frequency Domain. *Proceedings of International Workshop on Image Processing: Theory, Methodology, Systems, and Applications.* Budapest.

Barthel, K.U., Gerhard Ruhl and Thomas Voyé. (1996). Combining Wavelet and Fractal Coding For 3-D Video Coding. *Proc. of IEEE International Conference on Image Processing.* Vol. I, 181-184.

Birney, K. A. and T. R. Fischer. (1995). On the modelling of DCT and subband image data for Compression. *IEEE Trans. on Image Processing.* Vol. 4, No. 2, 186-193.

Bochez, K., M. Kaneko and H. Harashima. (1996). Fractal-like Video Coding with Weighted Summation. *Proc. of SPIE Conf. on Visual Communications and Image Processing.* Vol. 2727, 1377-1384 .

Clarke, R. (1985). *Transform Coding of Images.* New York: Academic Press.

Farvardin, N. and James W. Modestino. (1984). Optimum Quantizer Performance for a Class of Non-Gaussian Memoryless Sources. *IEEE Transactions on Information Theory.* Vol. IT-30, No. 3, 485-497.

Fisher, Y., D. Rogovin, and T. P. Shen. (1994). Fractal (Self-VQ) Encoding of Video Sequences. *Proc. of SPIE Conf. on Visual Communications and Image Processing.*

Fisher, Y. (1995). Mathematical Background. Chapter 2 in *Fractal Image Compression: Theory and Application.* Editor: Yuval Fisher. New York: Springer Verlag.

Fisher, Y. (1995). Fractal Image Compression with Quadtrees. Chapter 3 in *Fractal Image Compression: Theory and Application.* Editor: Yuval Fisher. New York: Springer Verlag.

Fisher, Y. and S. Menlove (1995). Fractal Encoding with HV Partitions. Chapter 6 in *Fractal Image Compression: Theory and Application.* Editor: Yuval Fisher. New York: Springer Verlag.

Gersho, A. and R. M. Gray (1992). *Vector Quantization and Signal Compression.* Series in Communications and Information Theory. Kluwer Academic Publishers.

Gharavi-Alkansari, M. and Thomas S. Huang (1994). Fractal-Based Techniques for a Generalized Image Coding Method. *Proc. of IEEE International Conference on Image Processing.* Vol. III, 122-126.

Gharavi-Alkansari, M. and Thomas S. Huang (1996). Fractal Video Coding By Matching Pursuit. *Proc. of IEEE International Conference on Image Processing.* Vol. I, 157-159.

Hürtgen, B. and P. Büttgen (1993). Fractal approach to low rate video coding. *SPIE Visual Communications and Image Processing.* Vol. 2094, 120-131.

Jacquin, Arnaud E. (1992). Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations. *IEEE Trans. Image Processing.* Vol. 1, 18-30.

Jayant, N. and P. Noll (1984). *Digital Coding of Waveforms, Principles and Applications to Speech and Audio.* Prentice Hall.

Lazar, M.S., and L. T. Burton (1994). Fractal Block Coding of Digital Video. *IEEE Transactions on Circuits and Systems for Video Technology.* Vol. 4, No. 3, 297-308.

LeGall, D. (1991, April). MPEG: A video compression standard for multimedia applications. *Communications of the ACM.* Vol. 34, No. 4, 47–58.

Li, H., M. Novak and R. Forchheimer (1993). Fractal-based image sequence compression scheme. *Optical Engineering.* Vol. 32, No. 7, 1588-1595.

Liou, M. (1991, April). Overview of the p×64 video coding standard. *Communications of the ACM.* Vol. 34, No. 4, 60–63.

Liu, B. and André Zaccarin (1993). New Fast Algorithms for the Estimation of Block Motion Vectors. *IEEE Trans. on Circuits and Systems for Video Technology.* Vol. 3, No. 2, 148-157.

Lundheim, L. (1995). A Discrete Framework for Fractal Signal Modeling. Chapter 7 in *Fractal Image Compression: Theory and Application.* Editor: Yuval Fisher. New York: Springer Verlag.

Netravali, A. and B. Haskell (1988). *Digital Pictures, Representation and Compression.* Applications of Communication Theory. Plenum.

Ohta, M. and Satoshi Nogaki (1993). Hybrid Picture Coding with Wavelet Transform and Overlapped Motion-Compensated Interframe Prediction Coding. *IEEE Trans. on Signal Processing.* Vol. 41, No. 12, 3416-3424.

Orchard, M. T. and G. J. Sullivan. (1994). Overlapped Block Motion Compensation: An Estimation-Theoretic Approach. *IEEE Trans. on Image Processing.* Vol. 3, No. 5, 693-699.

Paul, B. and M. H. Hayes (1994). Fractal-Based Compression of Motion Video Sequences. *IEEE Int. Conf. on Image Processing.* Vol. I, 755-759.

Pennebaker, W. B. and J. L. Mitchell (1993). *JPEG Still Image Compression Standard.* New York: Van Nostrand Reinhold.

Rijkse, K. (1995). ITU Standardization of Very Low Bit-Rate Video Coding Algorithms. *Signal Processing: Image Communication.* Vol. 7, No. 4-6, 553-565.

Riskin, E. A. (1991). Optimal bit allocation via the generalized BFOS algorithm. *IEEE Transactions on Information Theory.* Vol. 37, No. 2, 400-402.

Said, A. and W. A. Pearlman (1996). A New, Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees. *IEEE Trans. on Circuits and Systems for Video Technology.* Vol. 6, No. 3, 243-250.

Shapiro, J. (1993). Embedded Image Coding Using Zerotrees of Wavelet Coefficients. *IEEE Transactions on Signal Processing.* Vol, 41, No. 12, 3445-3462.

Vines, G. (1995). Orthogonal Basis IFS. Chapter 10 in *Fractal Image Compression: Theory and Application.* Editor: Yuval Fisher. New York: Springer Verlag.

Watanabe, H. and Sharad Singhal (1991). Windowed Motion Compensation. *Proc. of SPIE Conf. on Visual Communications and Image Processing.* Vol. 1605, 582-589.

Wilson, D.L., J. A. Nicholls and D. M. Monro (1994). Rate Buffered Fractal Video. *IEEE Int. Conf. on Acoustics, Speech and Signal Processing.*