Adaptive Fractal and Wavelet Image Denoising

by

Mohsen Ghazel

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2004

Abstract

The need for image enhancement and restoration is encountered in many practical applications. For instance, distortion due to additive white Gaussian noise (AWGN) can be caused by poor quality image acquisition, images observed in a noisy environment or noise inherent in communication channels. In this thesis, image denoising is investigated. After reviewing standard image denoising methods as applied in the spatial, frequency and wavelet domains of the noisy image, the thesis embarks on the endeavor of developing and experimenting with new image denoising methods based on fractal and wavelet transforms. In particular, three new image denoising methods are proposed: context-based wavelet thresholding, predictive fractal image denoising and fractal-wavelet image denoising. The proposed context-based thresholding strategy adopts localized hard and soft thresholding operators which take in consideration the content of an immediate neighborhood of a wavelet coefficient before thresholding it. The two fractal-based predictive schemes are based on a simple yet effective algorithm for estimating the fractal code of the original noise-free image from the noisy one. From this predicted code, one can then reconstruct a fractally denoised estimate of the original image. This fractal-based denoising algorithm can be applied in the pixel and the wavelet domains of the noisy image using standard fractal and fractal-wavelet schemes, respectively. Furthermore, the cycle spinning idea was implemented in order to enhance the quality of the fractally denoised estimates. Experimental results show that the proposed image denoising methods are competitive, or sometimes even compare favorably with the existing image denoising techniques reviewed in the thesis. This work broadens the application scope of fractal transforms, which have been used mainly for image coding and compression purposes.

Acknowledgements

I would like to thank my co-supervisors: Pr. E.R. Vrscay and Pr. G.H. Freeman for their guidance, support and encouragement throughout the course of my studies and preparing this thesis. I also would like to thank Profs. E. Jernigan, P. Fieguth, E.Y. Yang and W. Kinsner for serving on my examining committee, carefully reading my thesis and providing me with helpful feedback and suggestions.

I would like to acknowledge the financial support from my co-supervisors (grants from the Natural Sciences and Engineering Council of Canada (NSERC)), the Ontario Graduate Scholarships (OGS) as well as the Department of Electrical and Computer Engineering. All support is gratefully acknowledged and appreciated.

My sincere gratitude goes also to the supporting staff of the E&CE Department, everyone was always very patient, helpful and personable. In particular many thanks to Wendy Boles and Wendy Gauthier.

Finally, I wish to thank my family and close friends for their support.

Contents

List of Tables

List of Figures

Chapter 1

Image Degradation and Restoration

The need for image enhancement and restoration is encountered in many practical applications. For instance, distortion due to additive white Gaussian noise (AWGN) can be caused by poor quality image acquisition, images observed in a noisy environment or noise inherent in communication channels. Linear filtering and smoothing operations have been widely used for image restoration because of their relative simplicity. However, since these methods are based upon the assumption that the image signal is stationary and formed through a linear system, their effectiveness is generally acceptable but limited. In reality, real-world images have typically non-stationary statistical characteristics. They are formed through a nonlinear system process where the intensity distribution arriving at the imaging system is the product of the reflectance of the object or the scene of interest and the illumination distribution falling on the scene. There also exist various adaptive and nonlinear image restoration methods that account for the variations in the local statistical characteristics [9, 11, 39, 42, 44, 45, 49, 61, 62, 63, 64]. These methods achieve better enhancement and restoration of the image while preserving high frequency features of the original image such as edges.

Another, seemingly unrelated, problem in signal processing is the need and desire to manipulate, communicate and store large amounts of digital information. This natural demand for data acquisition, coupled with the exponential growth of computer-based information, remote systems and media applications have created tremendous demand for storage. Data compression and more specifically, digital image compression is a viable method for reduction of storage, transmission and manipulation requirements of digital imagery. Digital image compression takes advantage of the relationships existing between pixel values to define a new set of coefficients or parameters which can be used to re-construct an estimate of the original image. In fact most real-world images contain some amount of redundancy that can be removed when the image is stored or transmitted and replaced when it is reconstructed without significant loss of information. Successful compression will result in this transformed image taking up less storage and requiring less time to transmit than the original image.

Recently, many research efforts in the literature have shown that the above two signal processing problems are indeed closely related and lossy image compression methods have been proposed for the purpose of image denoising in several works [12, 13, 14, 15, 16, 60, 69]. The focus of this thesis is to extend the application of wavelet and fractal schemes, which are image compression methods, for the purpose of image denoising and restoration. In particular, the aim is to develop adaptive wavelet and fractal-based image denoising methods, assess and compare their performance.

This chapter is organized as follows: The image denoising problem is first described and the noise statistics are estimated. In section 2, several standard image denoising methods are described and implemented for the purpose of restoring a noisy test image. Some of these methods are spatially based, others are applied in the frequency domain of the noisy image. Section 3 contains a brief outline of some of the more recent developments in the image restoration field of research as well as a brief motivation of the research undertaken in this thesis. A brief outline of the thesis is presented in section 4.

1.1 The Image Denoising Problem

In this section, the image denoising problem will be formulated and some of the preliminary issues such as the assumptions about the noise, estimation of the noise variance and quality assessment criteria of the denoised image will be discussed.

1.1.1 Image Degradation and Restoration

In practice, an image may be degraded by various types and forms of noise. However, the most common type of noise is the additive one. As Figure 1.1 shows, the degradation process is modeled as an additive noise term, $\mathbf{w}$, which operates on an input image, $\mathbf{u}$, to produce a degraded image, $\hat{\mathbf{u}}$. Given this noisy observation, along with some knowledge of the additive noise term, the restoration technique yields an estimate, $\tilde{\mathbf{u}}$, of the original image. The denoised estimate is desired to be as close as possible to original image.

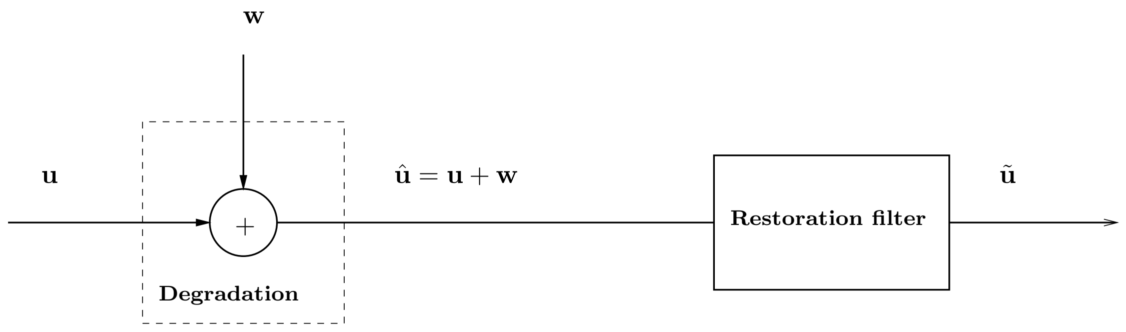


Figure 1.1: The degradation and restoration model for an additive noise process.

1.1.2 Noise Model

The principal source of noise in digital images arise during image acquisition (digitization) or transmission. The performance of imaging sensors is affected by a variety of factors, such as the environmental conditions during image acquisition, and by the quality of the sensing elements themselves. For instance, in acquiring images with a camera, light levels and sensor temperature are major factors affecting the amount of noise in the resulting image. Images are also corrupted during transmission principally due to interference in the channel used for transmission. For example, an image transmitted using a wireless network might be corrupted as a result of lighting or other atmospheric disturbance.

Gaussian Noise

Distortion due to additive white Gaussian noise (AWGN) can be caused by poor quality image acquisition, images observed in a noisy environment or noise inherent in communication channels. However, because of its mathematical tractability in both the spatial and the frequency domains, Gaussian noise models are used frequently in practice. Consequently, throughout this work, the focus will be on restoring digital images that have been corrupted by an additive white Gaussian noise. More specifically the noise, $\mathbf{w}$, is assumed to be an additive wide-sense stationary (WSS) white Gaussian noise (AWGN) process with zero mean and constant variance $\sigma^2_{\mathbf{w}}$, which is formed independently of the original noise-free image. Thus, if an image $\mathcal{I}$ of size $M \times N$ pixels is defined by its gray-level function, $\mathbf{u} = [u_{m,n}]$, $m, n = 1, 2, \ldots, M, N$, then the noisy image $\hat{\mathcal{I}}$ is defined by the noisy gray-level function $\hat{\mathbf{u}} = [\hat{u}_{m,n}]$, $m, n = 1, 2, \ldots, M, N$, as follows:

$$\hat{\mathbf{u}} = \mathbf{u} + \mathbf{w}, \tag{1.1}$$

where $\mathbf{u}$ and $\mathbf{w}$ are statistically independent. In the pixel domain, one has

$$\hat{u}_{m,n} = u_{m,n} + w_{m,n}, \quad m, n = 1, 2, \ldots, M, N, \tag{1.2}$$

where $w_{m,n}, m, n = 1, 2, \ldots, M, N$ are independent and identically distributed (iid) Gaussian random samples with zero mean and variance $\sigma_{\mathbf{w}}^2$, that is

$$w_{m,n} \sim \mathcal{N}(0, \sigma_{\mathbf{w}}^2), \text{ for } m, n = 1, 2, \ldots, M, N. \tag{1.3}$$

Next, the objective of the addressing the image denoising problem is addressed.

1.1.3 The Objective

The ultimate goal of image denoising and restoration techniques is to improve a degraded image in some sense. More specifically, given the noisy observation, $\hat{u}$, the objective of restoration is to obtain an estimate, $\tilde{\mathbf{u}}$, of the original image. The denoised estimate is desired to be as close as possible to original image, in some sense. In general, the more one knows about the noise, the closer the denoised estimate, $\tilde{\mathbf{u}}$, will be to $\mathbf{u}$.

In brief, restoration techniques attempt to suppress the random noise, which has corrupted the image, while preserving the most important visual features of the image, such as edges. The challenge in designing effective image denoising techniques lies in achieving these two competing objectives.

1.1.4 The Test Image

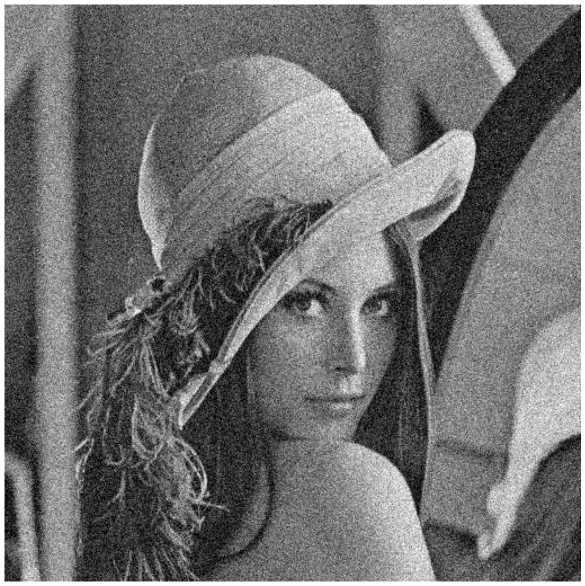
For experimental purposes, unless stated otherwise, the widely known original test image of "Lenna" and its noisy version as corrupted by an AWGN noise with variance $\sigma_{\mathbf{w}}^2 = 25^2$, will be used. These images are illustrated in Figure 1.2. It should be noted here that throughout this thesis, the printed images may have artifacts, such as dithering, halftone, contrast, etc., that are due to the laser printer. An electronic version of this thesis is available from the University of Waterloo library's E-Thesis Database at: http://library.uwaterloo.ca/ETD/etheses.html.

The selection of the noise intensity, $\sigma = 25$, was chosen to ensure that the noise is subjectively significant without overwhelming the original image. In chapter 6, the various image denoising methods of interest will be applied to other test images, corrupted by AWGN noise with different intensity, σ, in order to achieve a better assessment of their performance.

The original noise-free image $\mathcal{I}$

512×512 pixels (8 bits/pixel)

source: http://links.uwaterloo.ca



The noisy image $\hat{\mathcal{I}}$

noise variance: $\sigma_{\mathbf{w}}^2 = 25^2$

RMSE=25.01, PSNR=20.17.

Figure 1.2: The original image of "Lenna" and its noisy version, as corrupted by AWGN noise with noise variance $\sigma_{\mathbf{w}}^2 = 25^2$.

It is important to note here that in practice, the original signal is generally not known, only the distorted one is available. In this case, although the original image is available, it is only used for comparative purposes in order to assess the quality of the denoised image as compared to the original noiseless image. Also in reality, the acquired noisy image has been corrupted by an AWGN noise with unknown noise variance $\sigma_{\mathbf{w}}^2$. Thus, an important first step in solving an image denoising problem involves an accurate and robust estimation of the noise variance, $\sigma_{\mathbf{w}}^2$. Two different methods for estimating the noise variance will be described and implemented next.

1.1.5 Estimation of the Noise Variance $\sigma_{\mathbf{w}}^2$

As mentioned earlier, throughout most of this thesis, a noisy version of the "Lenna" test image which has been degraded by AWGN of variance $\sigma_{\mathbf{w}}^2 = 25^2$, will be used for illustrative purposes. Since in practice the noise variance is generally not known, we first outline how to estimate the noise statistics from the noisy image. The statistics of the AWGN noise are uniquely determined by the noise variance $\sigma_{\mathbf{w}}^2$, or equivalently, its standard deviation $\sigma_{\mathbf{w}}$. In this section, two simple, yet reliable methods for estimating the noise variance from the noisy image, will be presented. The first such method is based in the spatial domain of the noisy image, as discussed next.

A Spatial-Based Method

One method of estimating the noise variance is based on the assumption that an image has many regions of almost uniform intensity and that most changes in these regions of insignificant variations are due to the noise. This assumption is generally valid for many real-world images. The background of a scene is an example of such a region of insignificant variations. Also the noise $\mathbf{w}$ is assumed to have a constant variance $\sigma^2_{\mathbf{w}}$ throughout the image. This is a direct consequence of the fact that the noise is assumed to be a WSS process. The local variance estimates of all window masks of size $m\times m$ pixels, centered at every pixel of the image, are then calculated. The choice of the window size over which to estimate the local variance is important. It needs to be at least 5×5 for reasonable noise estimates, but it should also be small enough to ensure local signal stationarity. Experimentally, it was observed that both 5×5 and 7×7 work well but matter yields better results. Figure 1.3 illustrates the histogram of these local variance estimates. Examining the distribution of the local variance across the entire image and assuming that most of the local 7×7 subregions of the image have insignificant signal variations. The mode (i.e. the most frequent value) of the local variance distribution (histogram) was shown to be a reasonable estimator of the noise variance [39, 45]. This strategy yields the following estimate of the noise variance:

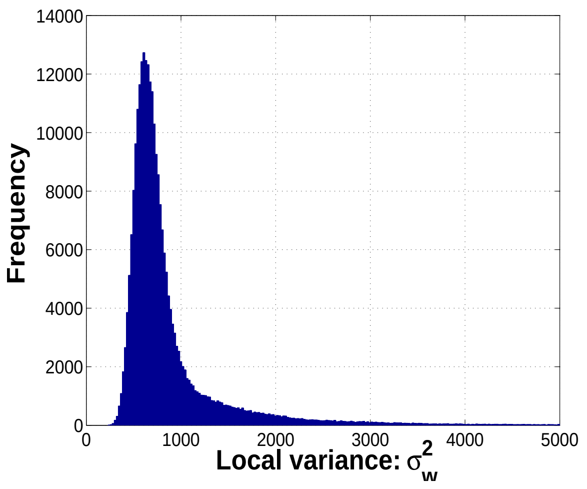


Figure 1.3: The histogram of the local noise variance computed from 7×7 masks of the noisy "Lenna" image.

$$\hat{\sigma}^2_{\mathbf{w}} = 636, \text{ or equivalently } \hat{\sigma}_{\mathbf{w}} = 25.22, \tag{1.4}$$

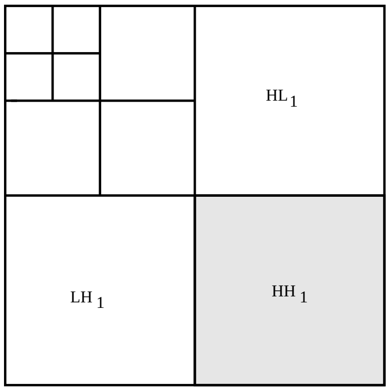
which is relatively close to the true noise standard deviation $\sigma_{\mathbf{w}} = 25$.

Since some of the proposed image denoising methods are applied in the wavelet domain of the noisy image, a wavelet-based method for estimating the noise variance is described next.

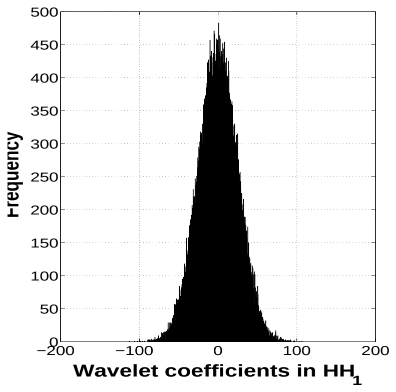
A Wavelet-Based Method

The Discrete Wavelet Transform (DWT) will be discussed in chapter 2. However, here only the basic concepts related to the DWT of an image are used. Recall that the wavelet decomposition of an image is done as follows: In the first level of decomposition, the image is split into four sub-bands, namely HH_1, HL_1, LH_1, and LL_1, as illustrated in Figure 1.4 (a). The HH_1 sub-band gives the diagonal details of the image, the HL_1 subband gives the horizontal features, while the LH_1 represents the vertical structures. The LL_1 subband is the low resolution residual consisting of low frequency components and it is this subband which is further split at higher levels of decomposition. It has been shown that the noise standard deviation $\sigma_{\mathbf{w}}$ can be accurately estimated from the first decomposition level diagonal subband HH_1 by the robust and accurate median estimator [20], as given by

$$\hat{\sigma}_{\mathbf{w}} = \frac{median(|HH_1|)}{0.6745}. \tag{1.5}$$



(a) Wavelet decomposition tree



(b) Histogram of HH_1.

Figure 1.4: The wavelet decomposition tree and a histogram of the coefficient in the first level diagonal subband HH_1.

Although the original image is responsible for a few large amplitude outliers, these few coefficients have little impact on the median operator. Figure 1.4 (b) illustrates the distribution of the wavelet coefficients in the first level diagonal subband, HH_1, of the wavelet coefficients. Note that this distribution is highly symmetric, with zero mean and resembles a Gaussian distribution. Table 1.1 illustrates the statistics as well as the estimate of the noise standard deviation $\sigma_{\mathbf{w}}$. Clearly, for the test image of "Lenna" with true noise standard deviation $\sigma_{\mathbf{w}} = 25$, a very accurate estimate

$$\hat{\sigma}_{\mathbf{w}} = 25.015. \tag{1.6}$$

is obtained. The wavelet-based method yields the best estimate of the noise variance, as compared to the other two estimation methods. This is because the discrete wavelet transform performs a significant degree of localization both spatially and in frequency. Consequently, in the subband HH_1, most of the wavelet coefficients are due to noise. Although, the original image is responsible for a few large amplitude outliers, these few coefficients have little impact on the median operator.

In what follows, this wavelet-based method will be adopted as the method of choice and the noise intensity will be estimated by

$$\sigma_{\mathbf{w}} \approx \hat{\sigma}_{\mathbf{w}} = 25.015. \tag{1.7}$$

$Size(HH_1)$	$Mean(HH_1)$	$Variance(HH_1)$	$Std(HH_1)$	$Median(\|HH_1\|)\|$	$\hat{\sigma}_{\mathbf{w}} = \frac{Median(\|HH_1\|)}{0.6745}$
256×256	0.000	641.598	25.330	17.012	25.015

Table 1.1: The statistics of the diagonal subband of the first decomposition level HH_1. Note that the estimate of the noise standard deviation $\hat{\sigma}_{\mathbf{w}} = 25.015$ is very close to true noise standard deviation $\sigma_{\mathbf{w}} = 25$.

In this section, the objectives involved in solving the image denoising problem were defined and two different reliable methods for estimating the noise variance, $\sigma^2_{\mathbf{w}}$, were presented. Next, the image quality measures, which will be used throughout this thesis to assess the quality of the denoised estimate, are defined.

1.1.6 Image Quality Measures

Image restoration may be viewed as a subjective or objective process. On the one hand, image restoration can be viewed as a heuristic procedure designed to manipulate an image in order to take advantage of the psychological aspects of the human visual system. On the other hand, as a subjective procedure, one needs to define a set of criteria of goodness that yield an optimal estimate of the desired result. However, this generally requires the knowledge of the original image. In this work, image denoising and restoration is viewed as an objective process and standard image quality measures will be used, as defined next.

In the absence of accurate mathematical models for the complete human visual system, there is no reliable standard measure of image quality that is consistent with human perception and that provides a qualitative as well as quantitative measurement. In spite of the lack of such an ideal measure, there are “acceptable” image quality measures that have been consistently used in the literature. One commonly used image quality measure is known as the *Root Mean Squared Error* (*RMSE*). Although it does not always correlate with human perception, the RMSE is often considered as an “good” measure of the fidelity of an image estimate. This measure is defined as follows:

Suppose that the original image $\mathbf{u}$ of size $M \times N$ has been denoised, using an image denoising scheme, and let $\tilde{\mathbf{u}}$ be the denoised estimate. The RMSE between the denoised image and the original image is given by

$$\text{RMSE} \quad = \quad \sqrt{\frac{1}{M \times N} \sum_{i=1}^{M} \sum_{j=1}^{N} (u_{i,j} - \tilde{u}_{i,j})^2}. \tag{1.8}$$

Another related image quality measure is the *Peak Signal to Noise Ratio* (*PSNR*), which is inversely proportional to the RMSE, its units are in *decibels (dB)* and is formally defined by

$$\text{PSNR} \quad = \quad 20 \log_{10}[\frac{255}{\text{RMSE}}] \;\; (dB) \tag{1.9}$$

where 255 is the maximum pixel value for an 8 bits/pixel gray-scale image. These two measures are used consistently throughout this thesis as appropriate measures of image representation quality as well as comparative criteria when comparing the performance of various image denoising schemes.

Clearly, when the original image is not known, one cannot rely on the above quantitative fidelity measures. In that case, the assessment of the denoised image is done subjectively.

Next, a few conventional image denoising methods will be briefly described and and their performance is assessed.

1.2 Standard Image Denoising Methods

The need for noise suppression without significantly degrading the edges and other high frequency components of the image, has thus motivated the development of efficient edge-preserving noise smoothing techniques. Significant progress and development in designing highly effective image denoising techniques have been achieved and reported over the years. In this section, two types of standard image denoising methods that are applied in the spatial and frequency domains of the noisy image, respectively, will be described and implemented.

1.2.1 Spatial Domain Methods

A variety of spatial filters that attempt to suppress noise without corrupting the significant features of the image have been developed over the years, such as [42, 45, 49, 62, 64, 63], just to mention a few. Initially, most of these methods were global and non-adaptive and based on the assumption that the image signal is stationary in nature, which is generally not the case for most real-world images. The evolution of adaptive techniques for image denoising and enhancement began with developing filters that adapt to the local statistics in small sub-regions centered at a sample pixel which is being filtered. In this section, a few conventional spatial-based image denoising methods will be described.

Spatial Mask Filters

Many image enhancement techniques are based on spatial operations performed on local neighborhoods of input pixels. Often, the image is convolved with a finite impulse response filter called *spatial mask*. Here each pixel is replaced by a weighted average of its neighborhood pixels, that is

$$\tilde{u}(m,n) = \sum_{(k,l)\in\mathcal{M}} a(k,l)\hat{u}(m-k,n-l), \tag{1.10}$$

where $\hat{u}(m,n)$ and $\tilde{u}(m,n)$ are the input noisy image and output denoised estimate, respectively, $\mathcal{M}$ is a suitably chosen window mask, and $a(k,l)$ are the filter weights. A common class of spatial averaging filters has all equal weights, which is known as the local averaging (or mean) filter, as discussed next.

The Mean Filter

The local mean filter uses equal weights, $a(k,l)$, to yield a smoothed estimate

$$\begin{aligned}\tilde{u}(m,n) &= \frac{1}{N_{\mathcal{M}}}\sum_{(k,l)\in\mathcal{M}} \hat{u}(m-k,n-l) &(1.11)\\ &= \frac{1}{N_{\mathcal{M}}}\sum_{(k,l)\in\mathcal{M}} u(m-k,n-l)+\bar{w}(m,n), &(1.12)\end{aligned}$$

where $a(k,l)=\frac{1}{N_{\mathcal{M}}}$ and $N_{\mathcal{M}}$ is the number of pixels in the window $\mathcal{M}$ and $\bar{w}(m,n)$ is the spatial average of the noise $w(m,n)$. It is a simple matter to show that $\bar{w}(m,n)$ has zero mean and variance $\frac{\sigma_{\mathbf{w}}^2}{N_{\mathcal{M}}}$. Thus, the noise power is reduced by a factor equal to the number of pixels in the window mask $\mathcal{M}$. If the noiseless image $u(m,n)$ is constant over the window mask $\mathcal{M}$, the spatial averaging results in an improvement in the output signal-to-noise ratio by a factor of $N_{\mathcal{M}}$. This seems to indicate that taking larger windows would be beneficial. However, in practice the size of the window mask $\mathcal{M}$ is limited due to the fact that $u(m,n)$ is not really constant, so for larger windows, spatial averaging introduces a distortion in the form of blurring. A 5×5 mask will be used for the experimental implementation of this filter.

Another spatially-based denoising filter, which is more adaptive than the mean filter, is the Lee filter, which is described next.

The Lee Filter

One of the limitations of conventional linear filtering methods for image denoising is that they are based on the assumption that the image signal is stationary and formed through a linear process. However, such an assumption is generally not valid for most real-world images. It is expected that local characteristics of an image would be more suitable for effective image restoration and enhancement. The Lee filter is such an adaptive local method [49]. It is a local statistics filter that employs local masks whose coefficients are functions of the local signal and noise characteristics. Using the Lee filter approach, at each pixel located (m,n) where the observed noisy pixel value is $\hat{u}(m,n)$, the denoised estimate $\tilde{u}(m,n)$ is obtained from $\hat{u}(m,n)$, as follows:

$$\tilde{u}(m,n) = \alpha_{m,n}\hat{u}(m,n) + \beta_{m,n} \tag{1.13}$$

The parameters $\alpha_{m,n}$ and $\beta_{m,n}$ are chosen to minimize the mean squared estimation error criterion

$$\Delta^2_{m,n} = E[(\mathbf{u} - \alpha_{m,n}\hat{\mathbf{u}} - \beta_{m,n})^2]. \tag{1.14}$$

Setting the partial derivatives of $\Delta^2_{m,n}$ with respect to $\alpha_{m,n}, \beta_{m,n}$ to zero and solving for these coefficients yields

$$\alpha_{m,n} = \frac{\sigma^2_{\mathbf{u}}(m,n)}{\sigma^2_{\hat{\mathbf{u}}}(m,n)} \text{ and } \beta_{m,n} = \frac{\sigma^2_{\mathbf{w}}}{\sigma^2_{\hat{\mathbf{u}}}(m,n)}\bar{\hat{u}}(m,n). \tag{1.15}$$

Since the original signal, $\mathbf{u}$, and the noise signal, $\mathbf{w}$, are assumed to be independent then one has

$$\sigma^2_{\hat{\mathbf{u}}} = \sigma^2_{\mathbf{u}} + \sigma^2_{\mathbf{w}}, \text{ or equivalently } \sigma^2_{\mathbf{u}} = \sigma^2_{\hat{\mathbf{u}}} - \sigma^2_{\mathbf{w}}. \tag{1.16}$$

Thus, the scaling factor, $\alpha_{m,n}$, can be expressed in terms of the statistics of the noisy image as follows:

$$\alpha_{m,n} = \frac{\sigma^2_{\hat{\mathbf{u}}}(m,n) - \sigma^2_{\mathbf{w}}}{\sigma^2_{\hat{\mathbf{u}}}(m,n)}. \tag{1.17}$$

Substituting (1.18) into (1.13) yields the following form of the Lee filter:

$$\tilde{u}(m,n) = \frac{\sigma^2_{\hat{\mathbf{u}}}(m,n) - \sigma^2_{\mathbf{w}}}{\sigma^2_{\hat{\mathbf{u}}}(m,n)} \hat{u}(m,n) + \frac{\sigma^2_{\mathbf{w}}}{\sigma^2_{\hat{\mathbf{u}}}(m,n)} \bar{\hat{u}}(m,n). \tag{1.18}$$

Note that the quantities $\bar{\hat{u}}(m,n)$ and $\sigma^2_{\hat{\mathbf{u}}}(m,n)$ denote, respectively, the local sample mean and variance of $\hat{u}$. These statistics are computed from a local mask centered at (m,n) and consequently they are computed locally and adaptively and thus they are dependent on the pixel (m,n), currently being processed. The choice of the window size over which to estimate the local variance is important. It needs to be at least 5×5 for reasonable noise estimates, but it should also be small enough to ensure local signal stationarity. Lee has found that both 5×5 and 7×7 work well [49]. A 7×7 mask will be used in the implementation of the Lee filter.

Note that the denoised estimate is a weighted sum of the raw noisy observation and its local average, using local variance for the weighting. On the other hand, when the local signal variance is much greater than the constant noise variance, the estimate is close to the observation, i.e. little or no smoothing occurs. On the other hand, when the local variance is entirely attributable to the presence of noise, the estimate is just the local average, and maximum smoothing is performed. Thus, the Lee filter is expected to perform little or no smoothing near edges or high contrast texture regions and extra smoothing in the flat regions of the image.

Denoising methods can also be applied in the frequency domain of the noisy image. Next, two frequency-based image denoising techniques will be described.

1.2.2 Frequency Domain Methods

Frequency domain filters are another way of approaching image restoration and enhancement. One of the main difficulties associated with spatial domain image processing methods is their computational complexity involved in performing the convolution and especially solving the deconvolution problem. Frequency domain methods overcome these problems due to the Fourier convolution property where convolution is transformed into multiplication of the spectra. Since most of the energy of a typical image is concentrated in the low frequencies, and because the energy of the noise is often spread across all frequencies (white noise), frequency-based denoising methods often adopt some form of lowpass filtering to suppress most of the high-frequency components in order to denoise the image. However, this approach is generally not effective because it involves suppressing two distinct types of high frequency components that are randomly mixed within the noisy image. On the one hand, visible edges, which are the significant features of an image from a human viewing perspective, are represented by desirable high-frequency components in the power spectrum of an image. On the other hand, noise is also modeled as high frequency data, representing the undesirable high frequency component that one seeks to eliminate or suppress. Thus, it becomes difficult to suppress the noise without also causing some degree of degradation of the significant features of the image. Consequently, most of the generic frequency-based image denoising methods often result in overly smoothed denoised images where the noise has been suppressed but also edges and other high-frequency features of the image have been blurred.

A non-ideal lowpass filter for image denoising is briefly described next.

The Gaussian Lowpass Filter

The Gaussian lowpass filter (GLF) presents a more realistic alternative to the ideal lowpass filter, which thresholds (sets to zero) all frequency components outside a specified low range of frequencies. The effects of the GLF on the spectrum of an image are similar to those of an ideal lowpass filter in the sense that low frequency components are allowed to pass while higher frequency components are suppressed. The main difference is that the truncation of the higher frequency components is gradual and not sharp, as is the case for the lowpass filter. Consequently, no visible ringing artifacts are observed in the spatial domain of the image. The GLF is represented by the following frequency response:

$$H_G(\Omega) = Ae^{-2\pi^2\sigma^2|\Omega|^2}. \tag{1.19}$$

The GLF still results in some degree of blurring of the original image, something that is expected from a lowpass filtering operation.

Another frequency-based filtering method is the Wiener filter, which is briefly discussed next.

The Wiener Filter

The objective in denoising the simple degradation model, in which a signal $\mathbf{u}$ is corrupted with an AWGN noise $\mathbf{w}$, as given in Eq. (1.1), is to recover the original signal $\mathbf{u}$. One might do so with a linear system with impulse response $\mathbf{h}$ such that its response to the observation is the minimum mean squared error estimate of the signal. Thus, the objective is to find the impulse response $\mathbf{h}$ such that the denoised estimate given by

$$\tilde{\mathbf{u}} = \hat{\mathbf{u}} * \mathbf{h} \tag{1.20}$$

minimizes the mean squared error

$$\Delta^2 = E[|\mathbf{u} - \tilde{\mathbf{u}}|^2] \tag{1.21}$$

The denoised estimate $\hat{\mathbf{u}}$ is the minimum mean squared error estimator of the original signal, $\mathbf{u}$ and the optimal choice of the impulse response, $\mathbf{h}$, is known as the Wiener filter. The frequency response $H(\Omega)$ associated with the Wiener filter is given by

$$H(\Omega) = \frac{P_S(\Omega)}{P_S(\Omega) + P_w(\Omega)} \tag{1.22}$$

where $P_w(\Omega) = \sigma_{\mathbf{w}}^2$ is the global noise variance estimate.

It has been shown that, under certain conditions, the Wiener filter is the optimal globally linear filter for removing AWGN noise [39, 45]. The main problem associated with the Wiener filter is that it uses a model which assumes that the original image and noise are globally stationary. The global stationarity refers to a signal with signal statistics that remain relatively constant throughout the image. Natural images are in general not stationary, in fact it is exactly that non-stationarity which manifests itself in images consisting of regions of relative smoothness and regions of high edge content. The assumption of global stationarity ignores the locally changing nature of the statistics typical in a broad class of images. This is indeed in contrast to the Lee filter outlined above, which uses local statistics from a small fixed window around a pixel of interest to estimate the non-degraded value at that point.

1.2.3 Computational Platform

Throughout this thesis, the numerical computations required to generate the illustrated experimental results were executed using a `Pentium 4, 1.9 GHz.` computer system platform. The C programming language as well as the standard numerical computation software, MATLAB, were generally used to implement these results. The execution times required to produce the experimental results are usually presented below the figures, when a appropriate. It should be noted that the implemented computer programs were not necessarily optimized to yield the fastest execution times.

Next, the various standard image denoising methods are implemented for the purpose of restoring the noisy test image.

1.2.4 Experimental Results

In this section, the standard image denoising methods, described above, will be implemented for the purpose of restoring the noisy test image.

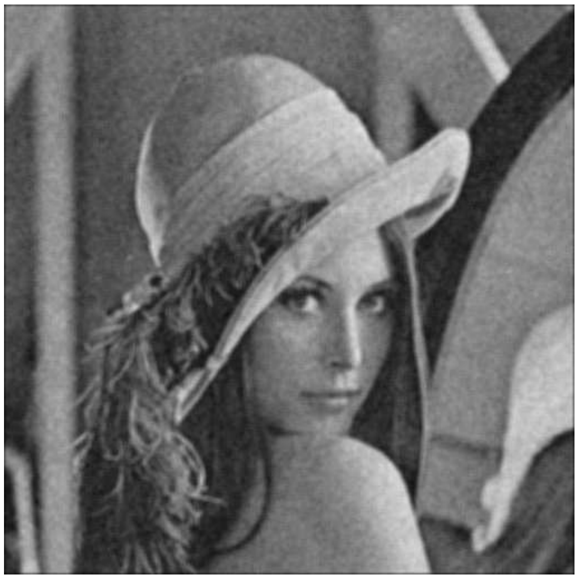
Figure 1.5 illustrates the results obtained by applying four generic image denoising methods, discussed above. In view of these results, the following observations are made:

- The results obtained by the local averaging, mean filter, are highly dependent on the size of the mask, $\mathcal{M}$. On the one hand, a small mask yields sharp yet noisy estimate, on the other hand, larger masks yield over-smoothed yet blurry denoised estimates. Experimentally, it was observed that a 5×5 mask yields the best result for the test image under consideration. Note that the mean filter results in a smoothed and blurry denoised estimate.

- The Lee filter does little or no smoothing near edges and other high contrast texture regions and performs extra smoothing when the signal component is near constant, that is in flat subregions of the image. The Lee filter attempts to adapt itself to the human visual system that is less sensitive to noise near edges but more sensitive to the presence of noise in the flatter subregions of the image. In fact, near edges, the Lee filter performs little or no smoothing, allowing visible noise in close proximity of the edges. However, in flatter subregions, e.g. the background, local signal variance is mostly due to noise and the Lee filter performs a high degree of smoothing through local averaging. In spite of the presence of the noise in high activity subregions of the image, we note that the edges in the Lee filter output remain sharp. This is indeed unlike the case for most lowpass linear filtering and smoothing operations which tend to blur edges.

- The Gaussian filter results in overly smoothed and a rather blurry denoised image. This is the case due to the fast exponential decay of the frequency response of this filter thus resulting in suppressing the higher frequency components of the image, which include noise as well as important high frequency content of the image, such as edges.

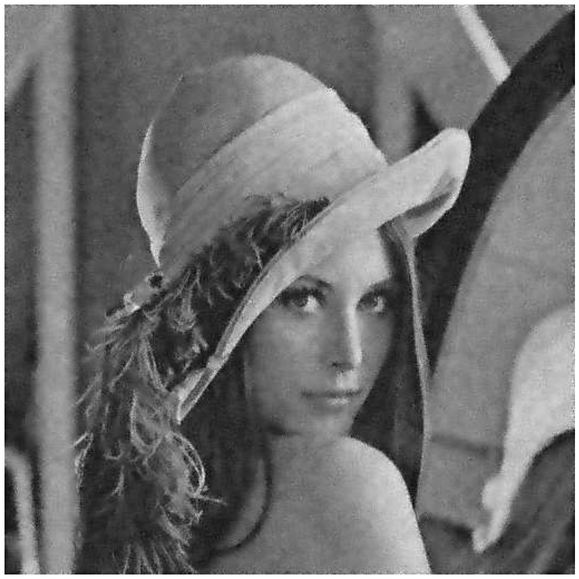
- Finally, the Wiener filter is the optimal globally linear filter for the purpose of removing an additive AWGN noise, in the sense that it minimizes the global mean squared error. This is indeed illustrated by the RMSE and PSNR fidelity measures. However, the problem with the Wiener filter is that it implicitly assumes stationarity of the original signal and the additive noise process, so that descriptive statistics, such as correlation functions, variances and power



(a) Mean filter with 5×5 mask

RMSE=11.01, PSNR=27.29.

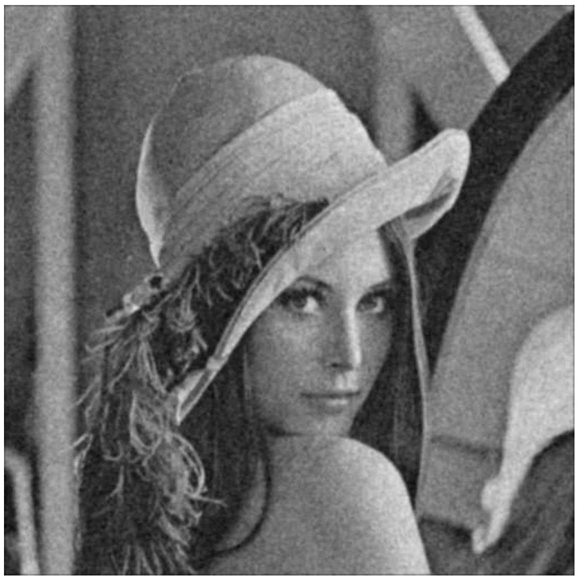
Execution time $\approx$ 2 secs.



(b) Lee filter with 7×7 mask

RMSE=10.73, PSNR=27.52.

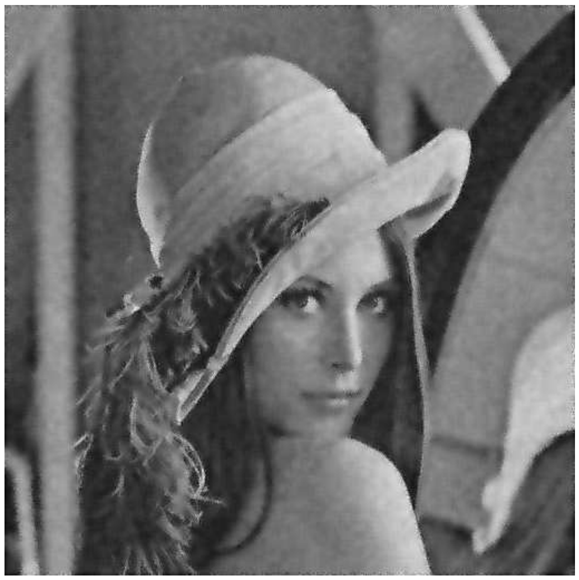
Execution time $\approx$ 3 secs.



(c) Gaussian lowpass filter (GLF)

RMSE=9.72, PSNR=28.38.

Execution time $\approx$ 1 secs.



(d) Wiener filter

RMSE=9.65, PSNR=28.43.

Execution time $\approx$ 2 secs.

Figure 1.5: Denoised estimates of "Lenna", using the various conventional image denoising techniques reviewed in this section.

spectra, remain the same throughout the image. While this assumption generally holds true for the noise, especially in the case of an AWGN noise, it is generally not true for most natural images. The global estimate of the signal statistics tend to underestimate the signal variance near edges and overestimate signal variance in relatively flat regions. This is because one has to choose a single signal variance for both of these vastly different subregions of the image. Consequently, the Wiener filter tends to perform too much smoothing near edges and not enough smoothing in flat subregions of the image. Some blurriness artifacts near edges are visible in the Wiener denoised image. Also, the denoised estimate appears noisy because flatter subregions of the image were not smoothed well enough by this filter and one is generally more aware of the presence of noise in flat regions of the image.

Next, a few more recent developments in adaptive image denoising are briefly outlined.

1.3 Recent Developments in Image Restoration

In the previous section, various standard image denoising methods, as applied in the spatial as well as the frequency domain of the noisy image, have been described and implemented. It should be stated that, although these methods are the most commonly known and used, they are not by any means the most effective ones. Indeed, there have been significant progress in developing highly adaptive, spatial and frequency-based, image denoising methods that perform significantly better than the standard image denoising methods, described above. Some of these recent developments are outlined in this section.

1.3.1 Adaptive Image Denoising Methods

The development of efficient and adaptive image restoration techniques that account for the local statistics has become a rather popular research field and has attracted many researchers from different backgrounds. Research conferences as well as journal issues were dedicated to this subject, such as [42, 64]. An adaptive recursive two-dimensional filtering technique for removing Gaussian noise in images was proposed in [44]. The adaptation was performed with respect to three local image features; edges, spots, and flat regions. Detectors for classifying these three subregions of the images were developed. The proposed filter was shown to perform simultaneous noise suppression and edge enhancement. A recursive and adaptive Wiener filter was proposed in [46]. A Wiener filter which locally estimates the power spectrum at various regions in the image was developed.

A qualitative comparison of edge-preserving smoothing techniques was studied in [11]. Locally adaptive techniques for edge-preserving smoothing were proposed and compared. In [61], a novel method to smooth a signal while preserving preserving discontinuities was presented. This was achieved by repeatedly convolving the signal with a small averaging mask weighted by a measure of the signal continuity at each point. This method was shown to be extremely attractive since edge detection can be performed after only a few iterations, and features extracted from the smoothed signal are correctly localized. In [9], a new nonlinear filter for noise smoothing was introduced. The novel feature of the proposed filter is that it attempts to distinguish between meaningful contours (edges) and noise, so that the image can be smoothed without loss of important details. Many other adaptive image restoration techniques were studied in [39, 62, 63].

Next, the basic idea behind the wavelet-based developed image denoising methods will be briefly described.

1.3.2 Wavelet Thresholding for Image Denoising

Over the past decade, there has also been a new and significant contribution in the image processing literature which lies in the development of wavelet-based methods for the purpose of image denoising. Basic wavelet image restoration techniques are based on thresholding in the sense that each wavelet coefficient of the image is compared to a given threshold; if the coefficient is smaller than the threshold, then it is set to zero, otherwise it is kept or slightly reduced in magnitude. In chapter 3, various wavelet thresholding methods for the purpose of image denoising will be studied and implemented in order to assess and compare their performance.

Another new direction in signal processing literature involves the application of image compression methods for the purpose of image denoising, as briefly discussed next.

1.3.3 Image Denoising using Compression Methods

Recently, many research efforts in the literature have shown that the problems of image compression and denoising are indeed closely related and lossy image compression methods have been proposed for image denoising in several works [12, 13, 14, 15, 16, 60, 69]. The intuition behind using lossy compression for denoising may be explained as follows: A signal typically has structure correlations that a good coder can exploit to yield a concise representation. White noise, however, does not have structural redundancies and thus is not easily compressible. Hence, a good compression method can provide a suitable model for distinguishing between the signal and noise.

The use of traditional image compression methods is the subject of study in this thesis, as motivated next.

1.3.4 Motivation of the Proposed Research

In this thesis, the use of various types of fractal-based image coding techniques, which are lossy image compression methods, for the purpose of image denoising, will be investigated. This research was originally motivated by the simple question: What happens if one simply encodes a noisy image, corrupted by and AWGN noise, using any of the various fractal-based image coding methods? It turns out that one does indeed achieve a varying degree of denoising and enhancement when encoding the noisy image using any of the fractal and fractal-wavelet image compression schemes. The intuition behind this is as follows:

- Purely fractal-based methods, as applied in the spatial domain of the image, exploit local and global self-similarities that are inherent in many classes of real-world images. Natural image structures possess similarities across subregions of the image which can be exploited by fractal image coding methods. However, noisy structures have no resemblance across other parts of the image and, therefore, cannot be accurately encoded using fractal coders. Consequently, encoding a noisy image with a fractal coder results in a good approximation of the natural self-similar structures, whereas the noisy contents cannot be described or reconstructed well by the fractal transform. Hence, fractally encoding a noisy image results in some degree of image denoising.

- Fractal-wavelet methods, as applied in the wavelet domain of the image, exploit redundancies and self-similarities that exist across resolution levels and scales of real-world images. Indeed, natural image structures generally possess similarities across resolution scales of their wavelet coefficients, which normally can exploited by fractal-wavelet coding methods. However, noisy structures have no resemblance across resolution levels and, therefore, cannot be represented well using fractal-wavelet coders. Thus, encoding a noisy image using a fractal-wavelet coder results in good reconstruction of the natural, self-similar structures, whereas the noisy contents cannot be re-generated. Hence, again this explains why one achieves some degree of denoising by simply encoding the noisy image, using any fractal-wavelet scheme.

This initial investigation then led to the question of whether such a simple fractal encoding of the *noisy* image could be used as a starting point to estimate the fractal code of the *noiseless* image from the observed noisy one. These questions will be examined, in detail, in this thesis and the answers will be shown to be in the affirmative.

1.4 Thesis Organization

In chapter 2, a detailed description and implementation of various fractal and wavelet based image coding methods will be provided. These fractal image coding methods will be applied for the purpose of image denoising in later chapters.

Chapter 3 contains a detailed study of the recently developed concept of wavelet thresholding for the purpose of image denoising. A particular attention will be given to implementing a set of standard wavelet thresholding methods, assessing and comparing their performance and investigating ways to improve them.

Chapters 4 and 5 represent the core of this work and they contain detailed investigations of applying fractal and fractal-wavelet methods for the purpose of image denoising, respectively.

In chapter 6, experimental comparisons between the various image denoising schemes studied and developed in this thesis, using different images and noise variances, shall be presented.

Finally, this thesis is concluded in chapter 7, where a summary of the undertaken work and findings is presented and related future work and investigations are proposed.

Chapter 2

Fractal and Wavelet Image Coding

Standard methods for still, noise-free, image coding come in several forms and are applied in various domains. Spatial image coding techniques, as applied in the pixel-domain of the image, are the most basic methods. Fourier transform methods, such as JPEG, have become widely used in practice [39]. However, over the past decade, there has been considerable interest and significant development in fractal and wavelet image coding methods. The wavelet transform has been shown to be perhaps the most efficient domain for the purpose of image compression. Today, the best known image compression techniques are the EZW [68] and the SPIHT [66] algorithms, which are wavelet-based methods. While its performance is not yet comparable to the wavelet transform, the fractal transform has also attracted significant interest and witnessed important developments since its birth in the late 1980s [3, 4, 5, 6]. Originally, the fractal transform was viewed in the signal processing community as a computationally expensive and limited technique that only works when the image exhibits a high degree of self-similarity. Today, significantly faster fractal-based techniques that perform relatively well for most real-world images are available [26, 27, 52, 65]. One of the most important milestones in the fractal transform literature is the novel idea of combining the capabilities of fractal and wavelet transforms to yield what has been known as the fractal-wavelet transform [18, 28, 47, 57, 70]. This hybrid scheme resulted in significant improvement in the performance of fractal-based image compression methods and extended the capabilities of both transforms. Furthermore, as will be seen in this thesis, the application of the fractal transform can be extended to other aspects of signal processing, such as image denoising and enhancement.

In this chapter, a brief review the basics of fractal, wavelet and fractal-wavelet transforms with an emphasis on the recent developments and their applications for the purpose of image coding is presented. In section 1, a brief mathematical framework of fractal image coding is first presented. Various practical issues related to the implementation of fractal image coding are then addressed, the fractal encoding and decoding processes are outlined and finally the fractal coding scheme for the purpose of image representation is implemented. Some of the issues encountered in fractal image compression will also be discussed and recent progress and development in the field of fractal image coding is outlined. In section 2, a brief outline of some of the practical aspects of the implementation of the discrete wavelet transform for the purpose of image coding and describe and implement the SPIHT algorithm, which is a highly effective wavelet-based image compression method. In section 3, an outline of the theory of the fractal-wavelet image transform is given and fractal-wavelet image coding is performed. The advantages of fractal-wavelet image coding as compared to standard image coding are highlighted and some of the recent progress and development in fractal-wavelet image coding are outlined. This chapter will be concluded with an experimental comparison between the various fractal and wavelet image compression methods studied in this chapter and a brief summary and concluding remarks.

2.1 Fractal Image Coding

Fractal image coding techniques are based on the theory of Iterated Function Systems (IFS) founded by Hutchinson [24] and further developed by Barnsley in the early 1980s [3, 4, 5, 6]. The IFS theory is based upon the Banach's Contraction Mapping Principle, which states that a contractive transformation, defined on a complete metric space, possesses a unique fixed point or attractor [3, 4]. For the purpose of image compression, this idea translates into finding an optimal contractive transformation whose attractor closely approximates a given target image. This problem is widely known as the *inverse* problem in the fractal image coding literature. The fractal-based schemes exploit the self-similarities that are inherent in many real-world images for the purpose of encoding an image as a collection of transformations. Hence, a digitized image, which typically requires mega-bytes of storage memory, can be stored as a collection of IFS transformations (parameters) and is easily regenerated or decoded for use or display. The storage of the IFS transformation coefficients generally requires much less memory, resulting in data compression. Iterated function systems were originally introduced to generate globally self-symmetric compact sets and natural images such as the Cantor set, the Sierpinski triangle, and the Spleenwort fern [3, 4, 5, 55]. Due to these restrictions and others, the IFS scheme was initially viewed as little more than a limited scheme for representing a specific class of images, namely those that exhibit a high degree of self-similarity. However, in the late 1980s, Jacquin developed a block-based fractal image compression scheme that exploits local self-similarities within images [43]. Many variations of this scheme have been developed since then [5, 26, 27, 52]. These schemes have shown that the fractal-based approach provides efficient and accurate models for many real-world images, resulting in relatively high compression ratios and good reconstruction fidelity. Although fractal-based schemes are still based on exploiting self-similarities in the spatial domain of images, these self-similarities do not have to be global or highly visible. In fact, most real-world images exhibit some degree of local self-similarity which can be exploited by using fractal-based image compression methods.

In this section, a brief outline of the mathematical framework of fractal image coding is given and various fractal-based schemes for the purpose of image representation are implemented. The main developments in standard fractal image coding will be emphasized. First, the theory of Iterated Function Systems, which represent the cornerstone of fractal image coding, is reviewed.

2.1.1 Mathematical Framework: Iterated Function Systems

An Iterated Function System (IFS) is uniquely described by a set of contractive transformations defined on a complete metric space. Hence, by the contraction mapping theorem, it possesses a unique attractor. For the purpose of image compression, the objective is then to construct an IFS whose attractor approximates a target image. In this section, the contraction mapping principle will be stated its use for the purpose of image compression shall be motivated. The collage theorem, which is closely related to the contraction mapping principle, will also be presented. This theorem provides a method of finding a contractive transformation whose attractor or fixed point closely approximates a given target image or function. The concepts of contractivity and fixed point are first defined.

Definitions: *Let T be a transformation defined on a metric space (Y, d_Y).*

1. *T is said to be contractive if there exists a positive constant $s, 0 \leq s < 1$, such that:*

$$d(T(x), T(y)) \leq s d_Y(x, y), \quad \forall x, y \in Y. \tag{2.1}$$

   The smallest such $s \geq 0$ is called the contractivity factor of T.

2. *A fixed point or attractor of T is a point $\bar{y} \in Y$ that is invariant under the application of T,*

i.e.

$$T(\bar{y}) = \bar{y}. \tag{2.2}$$

The Contraction Mapping Theorem guarantees that a contractive transformation defined on a complete metric space (i.e. a metric space where every Cauchy sequence converges) possesses a unique fixed point or attractor.

Contraction Mapping Theorem: *Let T be a contractive transformation defined on the complete metric space (Y, d_Y). Then T possesses a unique fixed point $\bar{y}$ in Y satisfying the following properties:*

- $T(\bar{y}) = \bar{y}$,
- $\lim_{n\to\infty} d_Y(T^{(n)}(y_0), \bar{y}) = 0$, $\forall y_0 \in Y$.

where

$$T^{(n+1)}(y) = T(T^{(n)}(y)), \quad \textit{for} \quad n = 0, 1, 2, \ldots \tag{2.3}$$

The contraction mapping theorem provides a converging algorithm for the approximation of the attractor $\bar{y}$ of a contractive transformation T. It is important to emphasize the key feature that the fixed point $\bar{y}$ can be closely approximated by iterating the transformation T a few times, starting with any initial seed y_0. In practice, only 10 - 20 iterations are needed for the estimation sequence $\{T^{(n)}(y_0)\}_{n=0}^{\infty}$ to visibly converge within a reasonably small error tolerance. The attractor or fixed point of a contractive transformation often exhibits self-tiling and symmetry characteristics, so it is also called a *fractal.* Hence in fractal image compression, the goal is to approximate a target image by a fractal. However, in practice, unless one carefully and appropriately choose the transformation T, its fixed point may not have any practical use. For the purpose of fractal image representation, for the purpose of image compression and representation, one is mainly interested in the construction of an appropriate contractive transformation whose attractor "closely resembles" a given target image. This is known as the *inverse problem* or the *fractal image coding problem*, and it can be stated in a mathematical framework as follows:

> *Given a target image $\bar{u} \in Y$, construct a contractive transformation T defined on Y whose attractor, $\bar{y}$, closely approximates $\bar{u}$.*

The following theorem is a consequence of the contraction mapping theorem and is known in the IFS literature as the collage theorem [3, 4, 6]. This theorem provides us with a practical and fast way to test for feasible choices of T.

Collage Theorem: *Let $\bar{u}$ be a target image in a complete metric space of images (Y, d_Y). Suppose there exists a contractive transformation T defined on Y with contractivity factor $0 \leq s < 1$ and attractor $\bar{y}$ such that*

$$d_Y(T(\bar{u}), \bar{u}) < \epsilon, \quad \textit{for some given} \;\; \epsilon > 0, \tag{2.4}$$

then

$$d_Y(\bar{u}, \bar{y}) \quad < \quad \frac{\epsilon}{1-s}. \tag{2.5}$$

In other words, if one can find a contractive transformation T that maps the target image $\bar{u}$ close to itself, then the attractor $\bar{y}$ of T will closely approximate the target $\bar{u}$. In view of this theorem, the inverse problem for fractal image compression can be reformulated as a constrained minimization problem:

Given a target image $\bar{u}$, find a contractive transformation T that maps $\bar{u}$ closest to itself.

Hence, solving for the optimal transformation T reduces to solving the following minimization problem for the parameters of T:

$$\text{Minimize:} \quad d_Y(T(\bar{u}), \bar{u}) \qquad \text{subject to:} \;\; T \text{ is contractive.} \tag{2.6}$$

In practice, it turns out that under certain assumptions, such an optimal transformation can easily be obtained by using the least squares optimization [26, 27, 52].

The formulation of this optimization problem conveys the basis of fractal image compression. However, in practice, solving for the parameters of T for a given real-world image is not an easy task. In fact, finding a transformation T, other than the identity transformation, that acts on a real-world image in a global fashion, without introducing a significant amount of distortion may not be possible. Such a transformation would typically be rather complex and may require a comparable amount of bytes to encode as the original image. However, such a complex transformation is not practically interesting, since little or no compression may be gained through its use. The difficulty in solving for such a transformation stems from the fact that it requires that the target image be highly and globally self-similar, a trait that is not exhibited by most real-world images. However, most real-world images may exhibit some degree of local self-similarity, in the sense that some subregions of the image may be "similar" to some other larger subregions. The sense of similarity shall be defined, in more detail, in the next section.

The standard fractal scheme, originally introduced by Jacquin, seeks to exploit these local self-similarities by adopting a block-based approach [43]. In block coding, the target image is partitioned into non-overlapping sub-blocks and "similar" subregions are then matched. This is the basis of fractal-block coding described in detail in the next sections.

2.1.2 Practical Aspects of Fractal Image Coding

Originally, IFS-type methods sought to express a target set or image as a union of shrunken copies of itself. However, most real-world objects are rarely so entirely self-similar. Instead, self-similarity may be exhibited only locally, in the sense that subregions of an image may be self-similar. In the late 1980s, Jacquin developed a block-based fractal image compression scheme that exploits local self-similarities within images [43]. This fractal-based scheme is based on exploiting the inherent local self-similarities in the spatial domain of images. In fact, most real-world images exhibit some degree of local self-similarity which can be exploited by using fractal-based image compression methods. To exploit the local self-similarities within sub-regions of images, the image is subdivided into a pair of simple and uniform partitions of the image: A domain partition of larger sub-blocks, also known as parent sub-blocks and a range partition of smaller sub-blocks, also known as child sub-blocks. A parent sub-block is mapped into its corresponding child sub-block using a geometric mapping, followed by a simple affine transformation, known as the gray-level map. This process is outlined next.

Let $\mathcal{I}$ denote an image of interest as defined by an *image function* $u(x, y)$ supported over a region $X \in \mathbf{R}^2$. Here $x, y \in X$ denote spatial coordinates of a point or pixel of $\mathcal{I}$. Now suppose that there exists a suitable partition $\mathcal{R}$ of X into range sub-blocks R_k so that $X = \cup_k R_k$. For simplicity, the R_k are assumed to be non-overlapping. Also, assume that associated with each sub-block R_k is a larger domain sub-block $D_{i(k)} \subseteq X$ so that $R_k = w_{i,k}(D_{i(k)})$, where $w_{i,k}$ is a one-to-one contraction map in the continuous plane. As illustrated in Figure 2.1, typically, the R_k and $D_{i(k)}$ blocks are square pixel blocks and the $w_{i,k}$ are affine contractions that may also rotate or invert the $D_{i(k)}$. As will be seen, there are eight such possible maps.

Now suppose that the image function $u(R_k)$ supported on R_k is well approximated by a modified copy of the image function $u(D_{i(k)})$ supported on $D_{i(k)}$ as follows:

$$u(R_k) \cong \phi_{i,k}(u(D_{i(k)})) = \phi_{i,k}(u(w_{i,k}^{-1}(R_k))), \tag{2.7}$$

where $\phi_{i,k} : \mathbf{R} \to \mathbf{R}$ is a *gray-level* map that operates on the pixel intensities. Because of the non-overlapping nature of the partition, one may write

$$u(x,y) \cong (Tu)(x,y) = \sum_k \phi_{i,k}(u(w_{i,k}^{-1}(x,y))). \tag{2.8}$$

In other words, the image function u is approximated by a union of spatially-contracted $(w_{i,k})$ and gray-level-distorted $(\phi_{i,k})$ copies of itself. This union of modified copies may be considered as defining a special kind of fractal transform operator T on image functions. If the above approximation is a good one, then the so-called collage distance $\| u - Tu \|$ is small.

Under suitable conditions on the spatial maps $w_{i,k}$ and the gray-level maps $\phi_{i,k}$, the operator T is contractive [29, 30]. From the contraction mapping theorem, there exists a unique fixed point function, $\bar{u}$, such that $\bar{u} = T(\bar{u})$. Furthermore $\bar{u}$ is *attractive*: If one starts with any image function u_0 supported on X and constructs the iteration sequence of functions $u_{n+1} = T(u_n)$, then the sequence u_n converges to $\bar{u}$ as $n \to \infty$. In practice, the sequence converges after a finite number of iterations. Moreover, the collage theorem establishes that if u is "close" to $T(u)$, then u is also "close" to $\bar{u}$, implying that $\bar{u}$ is a good approximation to u. The consequence is that one needs only to store the parameters that define the operator T in order to generate the approximation $\bar{u}$ of u. This is the essence of fractal image coding, as shall be explain next in more detail.

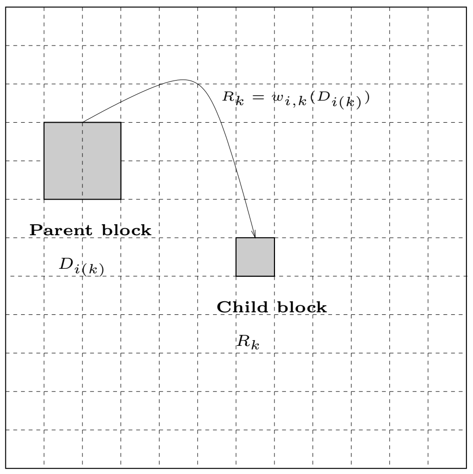


Figure 2.1: Fractal transformation of a domain block $\mathcal{D}_{i(k)}$ into a range block $\mathcal{R}_k$.

2.1.3 Block-Based Fractal Image Coding

A few practical issues left unaddressed in the above discussion, are now addressed. For example, what kind of partitioning for the R_k should be used? Given a *child* or *range block* R_k, how does one determine an "optimal" *parent* or *domain block* $D_{i^*(k)}$, with corresponding gray-level mapping $\phi_{i^*(k),k}$, that produces the best approximation in (2.8)?

Uniform Image Partitioning

For a given fractal resolution, (M, N), the standard fractal scheme adopts a block coding strategy where the target image is first partitioned into non-overlapping uniform sub-blocks as follows:

- As illustrated in Figure 2.2, the target image is subdivided into two different partitions of non-overlapping blocks:
  1. $M \times M$ domain blocks, $\mathcal{D}_i$, $i = 1, 2 \ldots, M^2$,
  2. $N \times N$ range blocks, $\mathcal{R}_k$, $k = 1, 2 \ldots, N^2$, typically $N = 2M$,
- Each child block, $\mathcal{R}_k$, $k = 1, 2, \ldots, N^2$, is then matched to a "similar" parent block $\mathcal{D}_{i^*(k)}$, for some $i^*(k) \in \{1, 2, \ldots, M^2\}$.

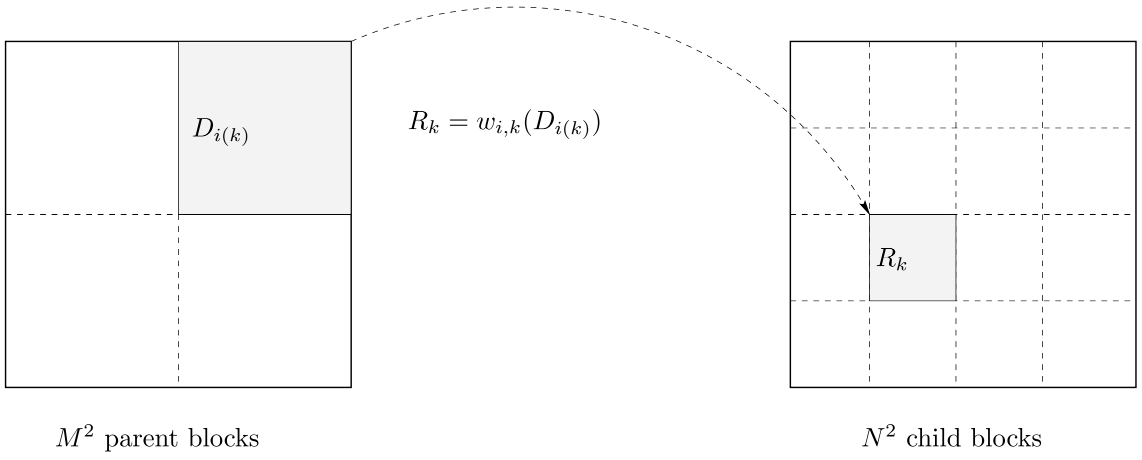


Figure 2.2: Uniform image partitioning for the purpose of fractal image coding.

For each range block R_k, it is assumed that there exists a set or "pool" $\mathcal{D}_i$ of domain blocks $D_i \subset X$. The domain blocks are usually assumed to be twice the length and width (hence four times the area) of the range blocks. Naturally, the larger the domain pool, the better the approximation that can be achieved in (2.8). However, a larger domain pool requires more memory for the storage of indices that specify the locations of optimal domain blocks. From a compression viewpoint, some compromise between domain pool size and fidelity must be established.

The Geometric Mappings w

As illustrated in Figure 2.3, there are eight contractive affine mappings that can transform a square parent block into a smaller square child block – four rotations, a horizontal flipping, a vertical flipping and two diagonal flippings. Any of these mappings can be considered as the composition of a non-rotating affine transformation followed by an isometry. Note that one may consider only a subset of the eight possible isometries are employed. A geometric mapping of domain block D_i to range block R_k is denoted as $w_{ik}^{(m)}$, where $1 \leq m \leq 8$. In most applications, including those in this chapter, the domain blocks are contracted by a factor of four. The action of these geometric

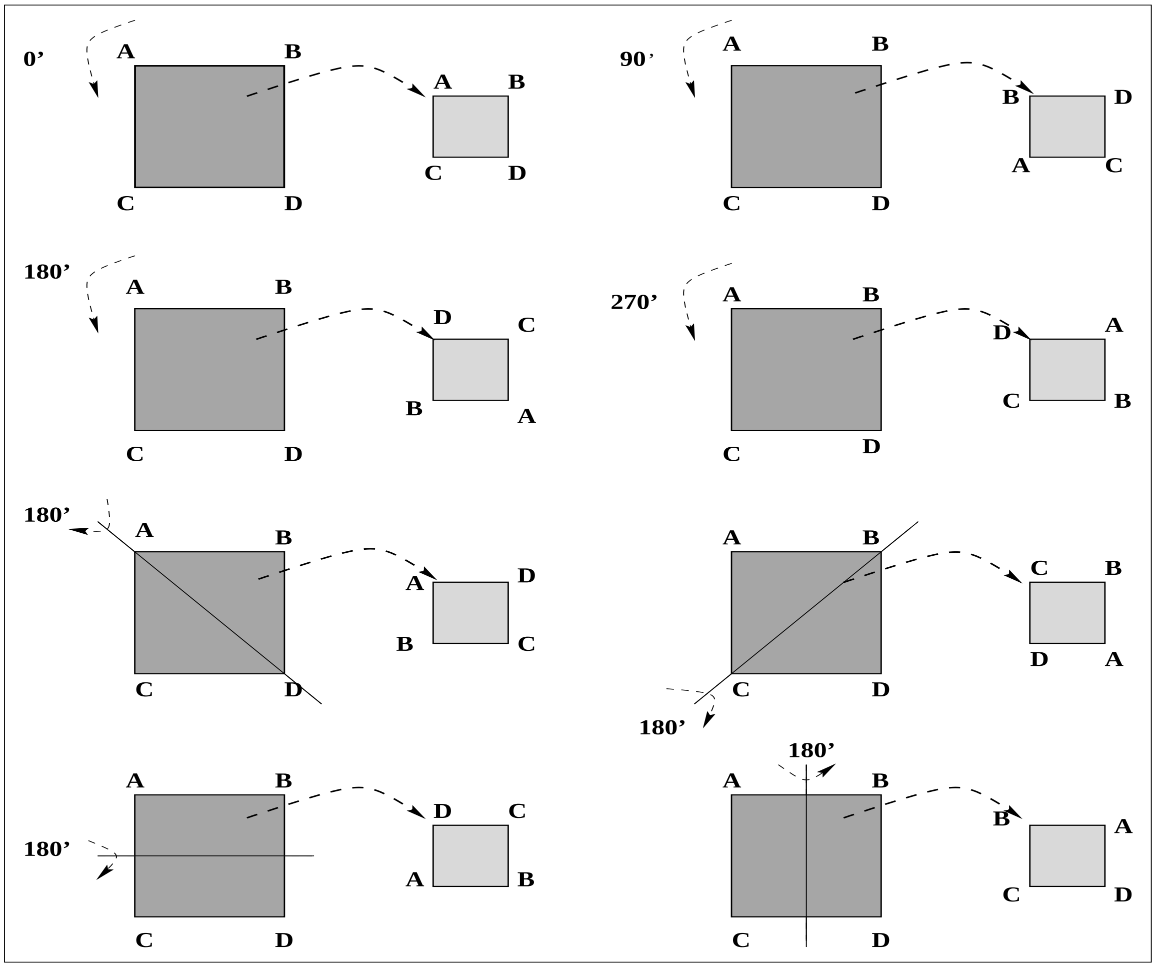


Figure 2.3: There are 8 geometric maps that transform a square parent block into a smaller square child block.

contractions is straightforward when the x and y spatial coordinates are continuous, i.e. real-valued. However, in discrete pixel space, the shrinking of an $2n \times 2n$ parent pixel block to an $n \times n$ child pixel block must be achieved by some kind of reduction or *decimation* procedure. Typically, a sub-block of 2×2 neighboring pixels in the parent block is replaced by a single pixel and the four gray-level values are replaced by their average value. In the discrete case, it is assumed that the maps $w_{ik}^{(m)}$ perform this decimation operation.

Next, the gray-level maps, which are applied to these transformed parent sub-blocks, are defined.

The Gray-level Mappings ϕ

Once the parent blocks D_i are reduced to the same pixel size as the child blocks R_k, a first order linear prediction is performed to estimate the latter from the former using affine transformations of the form

$$\phi(t) = \alpha t + \beta. \tag{2.9}$$

The gray-level map that best maps the gray-level values supported on the (decimated) block D_i to their counterparts on R_k is the map $\phi(t)$ that minimizes the so-called *collage distance* over R_k

$$(\Delta_{ik}^{(m)})^2 = \| \phi(u(w_{ik}^{(m)}(D_i)) - u(R_k) \|_2^2 \,. \tag{2.10}$$

In practice, the $\mathcal{L}^2$ norm (i.e. least squares error) is used so that the optimal gray-level map performs a least-squares fit of the parent-child gray-level data. Let $\{x_j, j = 1, 2, \ldots, n\}$ and $\{y_j, j = 1, 2, \ldots, n\}$, where n is the size of the range block R_k, denote the gray-level values on, respectively, the geometrically transformed (decimated) parent block D_i and the child block R_j, using the geometric mapping $w_{ik}^{(m)}$. Then the minimization of the squared $\mathcal{L}^2$ distance

$$(\Delta_{ik}^{(m)})^2 = \sum_{j=1}^{n} [y_j - (\alpha_{ik}^{(m)} x_j + \beta_{ik}^{(m)})]^2 \tag{2.11}$$

yields a set of linear equations in $\alpha_{ik}^{(m)}$ and $\beta_{ik}^{(m)}$ with solution (assuming a nonzero determinant)

$$\begin{aligned} \alpha_{ik}^{*(m)} &= \frac{n \sum_{j=1}^{n} x_j y_j - \sum_{j=1}^{n} x_j \sum_{j=1}^{n} y_j}{n \sum_{j=1}^{n} x_j^2 - [\sum_{j=1}^{n} x_j]^2} \\ \beta_{ik}^{*(m)} &= \frac{1}{n} \sum_{j=1}^{n} y_j - \alpha_{ik}^{*(m)} \frac{1}{n} \sum_{j=1}^{n} x_j. \end{aligned} \tag{2.12}$$

There is one complication, however, in that the contractivity of the fractal transform operator T is dependent upon the α scaling coefficients. There is no simple relationship between the $\mathcal{L}^2$ contractivity factor of T and the α coefficients because of the local nature of the parent-child mappings. However, in the $\mathcal{L}^\infty$ norm, contractivity is guaranteed if all α satisfy the condition

$|\alpha| < 1$. For this reason, most fractal coding algorithms "clamp" the α coefficients, i.e. $\alpha = \text{sgn}(\alpha)\min(|\alpha|, 1)$. The resulting fractal transform operators T are almost always contractive in $\mathcal{L}^\infty$, hence in $\mathcal{L}^2$, due to the equivalence of the norms in finite pixel space. In [26], the gray-level coefficients, $\alpha_{ik}^{*(m)}$, where truncated at $\sqrt{2}$ with no noticeable effects on the contractivity.

Next, a practical, block-based fractal image coding scheme is presented and implemented for the purpose of fractal image representation of real-world images.

2.1.4 The Standard Fractal Image Coding Scheme

In view of the above definitions and notations, all of the ingredients necessary to outline a block-based fractal image coding scheme, have been defined. This scheme is also applied to perform fractal encoding of a test image.

The Fractal Encoding Algorithm

Supposed that a particular partitioning $\mathcal{R}$ of range blocks R_k, $k = 1, \ldots, N_R$ has been constructed. As outlined above, associated with each block R_k is a pool of domain blocks $\mathcal{D}_k = \{D_{i^*(k)} \subset X\}$. For each range block, R_k, $1 \le k \le N_R$, one seeks the domain block $D_{i^*(k)} \in \mathcal{D}_k$ such that the sub-image $u(D_{i^*(k)})$ best approximates the sub-image $u(R_k)$ after a geometric transformation/decimation. In other words, one need to fin the indices

$$(i^*(k), m^*(k), \alpha_{i^*(k)}^{*(m^*(k))}, \beta_{i^*(k)}^{*(m^*(k))}) = \arg\left[\min_{D_i \in \mathcal{D}_k} \min_{1 \le m \le 8} \| \alpha_{ik}^{*(m)} u(w_{ik}^{(m)}(D_i)) + \beta_{ik}^{*(m)} - u(R_k) \|_2^2\right],$$

where $\alpha_{ik}^{*(m)}$ and $\beta_{ik}^{*(m)}$ denote the least-squares gray-level coefficients associated with the mapping $w_{ik}^{(m)}$.

The *fractal code* associated with the partition $\mathcal{R}$ and the associated domain pools $\mathcal{D}$ is then as follows: For each range block, R_k, $1 \le k \le N_R$, determine

1. The fractal resolution of the uniform partition (M, N),
2. The index $i^*(k)$ of its optimal parent block $D_{i^*(k)} \in \mathcal{D}_k$,
3. The optimal isometry $m^*(k)$.
4. The optimal (least-squares) gray-level coefficient pair $(\alpha_{i^*(k),k}^{*(m^*(k))}, \beta_{i^*(k),k}^{*(m^*(k))})$.

Thus, instead of storing each range block $\mathcal{R}_k$, $1 \le k \le N_R$, one can just store its corresponding fractal code. Generally, and as will be shown later, storing the fractal code requires much less storage memory than storing the original image, hence resulting in significant data compression (i.e. higher than 20:1).

Computational Complexity

For a given fractal resolution, (M, N), there are:

- N^2 child (range) blocks, R_k.
- M^2 parent (domain) blocks, D_i.
- 8 geometric maps, $w^{(m)}$.

For each R_k, $k = 1, 2, \ldots, N^2$, search through all the collection of D_i, $i = 1, 2, \ldots, M^2$, to find the optimal one which minimizes the collage error, as given by Eq. (2.11). In addition, there are 8 ways to map a child block onto a parent block, as illustrated in Figure 2.3. This means one has to compare $8 \times M^2$ to each of the N^2 child blocks, for a total of $8 \times M^2 \times N^2$ child/parent matching tests. For instance, when $(M, N) = (32, 64)$, there are 33554432 tests. Each of these tests requires the decimation step of a parent block, the computation of the gray-level coefficients, as in (2.12) and the collage error, as in (2.11).

Clearly, the standard fractal encoding algorithm, which achieves optimal results for the non-overlapping parent blocks case, is computationally expensive. Many sub-optimal fractal-based image coding methods have recently been proposed the significantly reduce the computation load at the expense of relatively small degradation in the quality of the encoded image have been proposed. Some of these methods will be described in section 2.1.6.

Next, the fractal decoding algorithm, which is significantly simpler and faster than the encoding process, is described.

The Fractal Decoding Algorithm

The fractal code defines a fractal transform operator T that acts on an image function u according to (2.8). Starting with an arbitrary initial image, typically u_0 is chosen to be a blank image. The fractal decoding process can be outlined as follows:

> For each range block, R_k, $1 \leq k \leq N_R$, to obtain the image values of $T(u)$ in R_k, one takes the optimal domain block $D_{i^*(k)}$ and apply the following operations:
>
> 1. decimate the domain block to produce a block of the same size as R_k,

2. apply the appropriate isometry, $1 \leq m^*(k) \leq 8$, to the block,
3. modify the gray-level values of this block according to the appropriate gray-level map
$$\phi(t) = \alpha_{i(k),k}^{*(m(k))} t + \beta_{i(k),k}^{*(m(k))},$$
4. replace the image values in (range) block R_k by those of the above block.

This process is then repeated recursively until it converges to the attracting fixed point within a prescribed tolerance error or after executing a prescribed number of iterations.

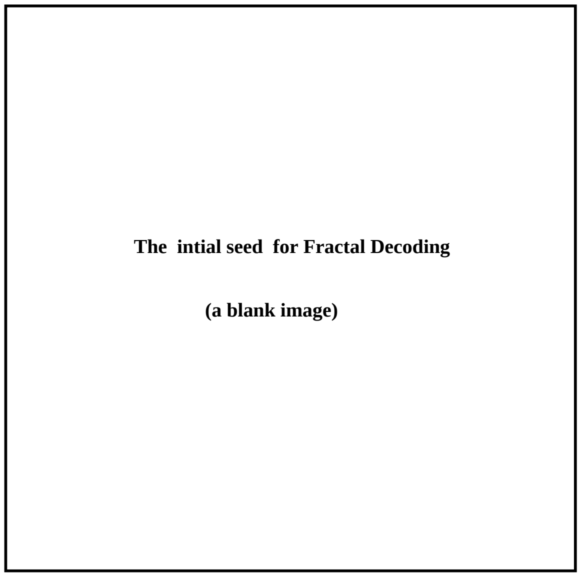
Although the fractal encoder may require a considerable amount of computation in order to match the child blocks to their optimal parent blocks, the decoding process is computationally simple and converges rapidly. This may make the fractal-based schemes attractive alternatives for applications in which the resources available for decoding are considerably less than the resources available for encoding. For instance, in multimedia applications, considerable computational resources may be available for the encoding operation. However, if the decoding is to be done in software, the computational resources available to the decoder may be quite limited.

Next, the fractal representation of the real-world test image of "Lenna" is illustrated.

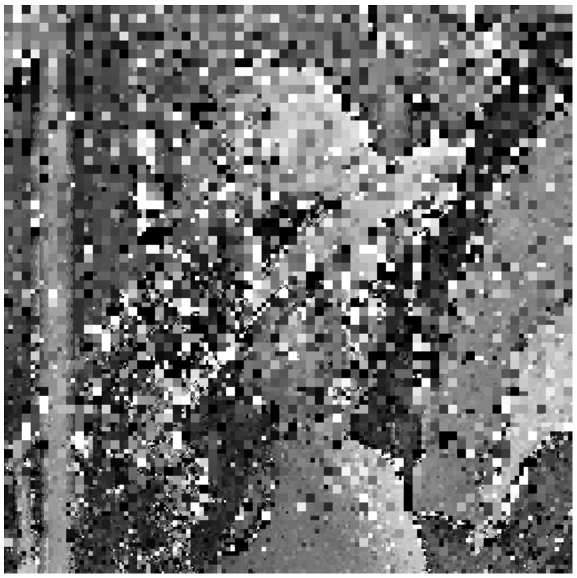
Application to Images

Figure 2.4 illustrates a few iterations of the fractal decoding algorithm. Note how, starting with an initial blank image, a relatively "close" approximation of the fractal representation of the test image of "Lenna", is obtained after only a few iterations. In Figure 2.5, the fractal representations of the test image are shown at various domain-range block resolutions (M, N). The quality of the fractal representations are measured using the RMSE and PSNR fidelity measures. As expected, the quality of the representation increases with M. Clearly, there is a trade-off between the computational complexity of the fractal scheme and the fidelity of the fractal representation of the image.

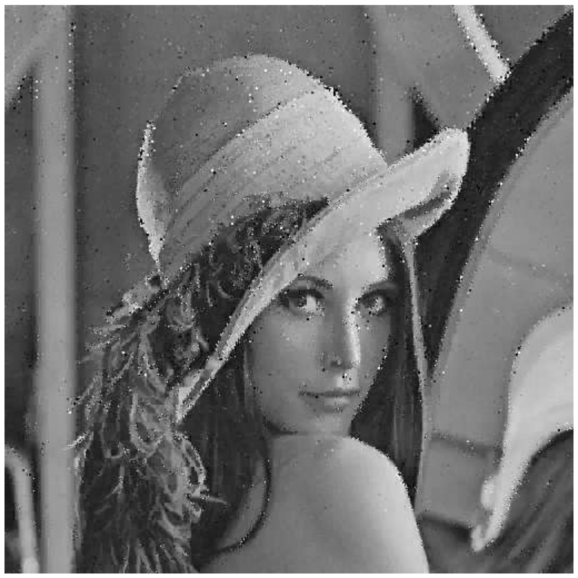
For this fractal image coding implementation, no particular attention was given to assessing the compression capability of the fractal scheme. The fractal code was passed losslessly to the decoder and no quantization of the gray-level parameters was done and hence the compression ratios are not given. Next, a brief review of various issues related to quantizing and storing the fractal coefficients and assessing the compression capability of the above fractal image coding scheme is presented.



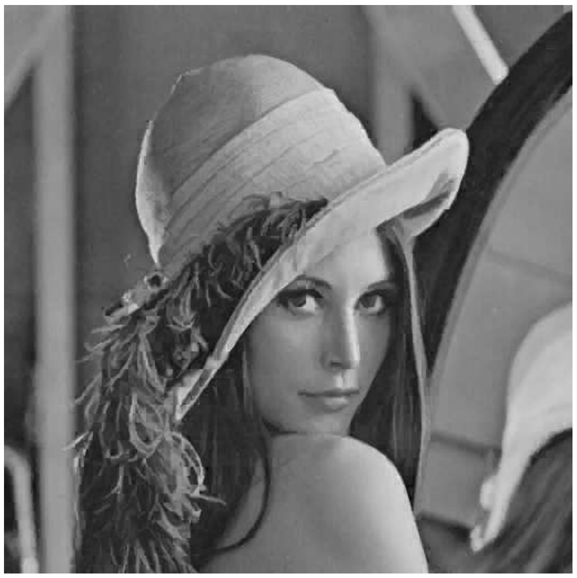
(a) Starting with a blank image



(b) After 1 iteration of T

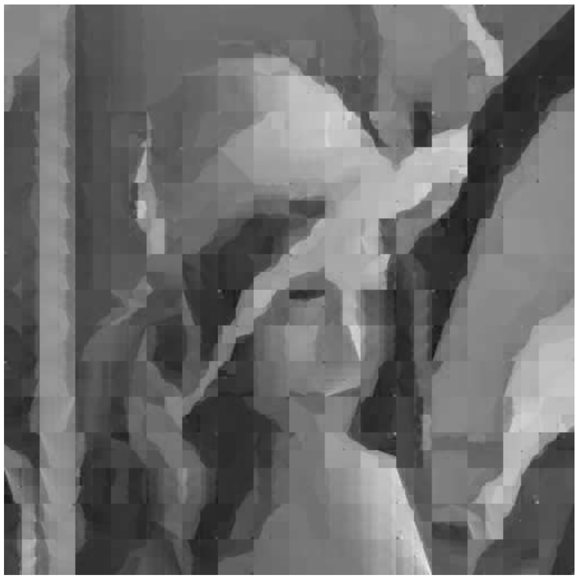


(c) After 5 iterations of T

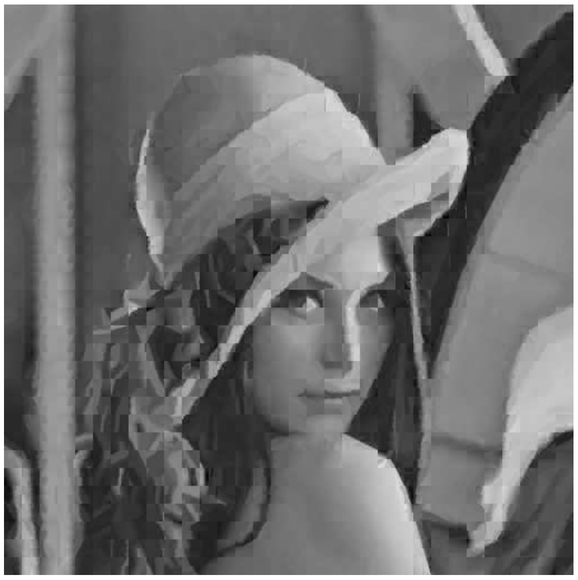


(d) After 10 iterations of T

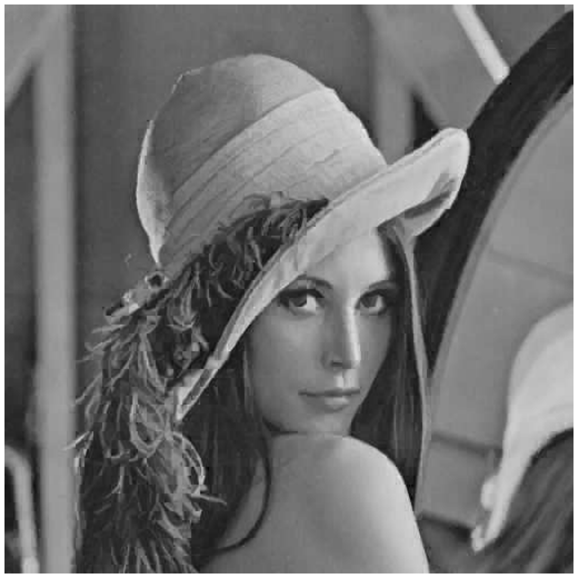
Figure 2.4: Generating a fractal approximation of the target image "Lenna" by iterating the fractal transform T, starting with an initial blank image, using the uniform partitioning fractal scheme with $(M, N) = (32, 64)$.



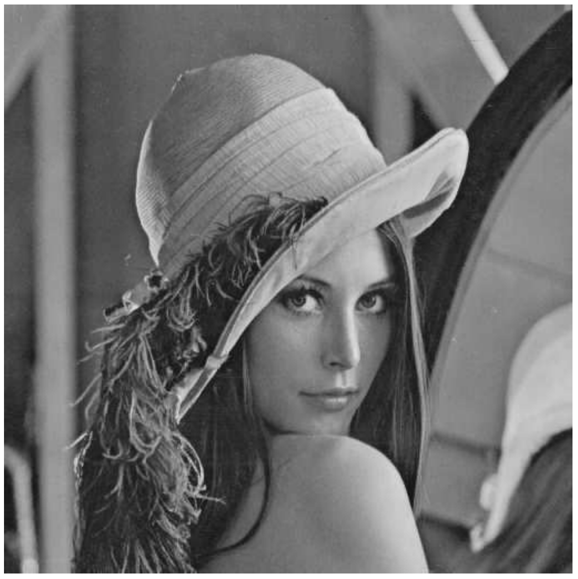
(a) (M,N)=(8,16): Execution time ≈ 105 secs.
RMSE=18.33, PSNR=22.87.



(b) (M,N)=(16,32): Execution time ≈ 309 secs.
RMSE=11.94, PSNR=26.60.



(c) (M,N)=(32,64): Execution time ≈ 1129 secs.
RMSE=7.20, PSNR=31.00.



(d) (M,N)=(64,128): Execution time ≈ 4719 secs.
RMSE=3.39, PSNR=37.53.

Figure 2.5: Fractal representations of the test image of "Lenna" for various resolutions values (M, N), using all 8 geometric maps. Note that the $(M, N) = (64, 128)$ fractal representation is visually identical to the original image.

2.1.5 Standard Fractal Image Compression

Historically, it was found that much less computer memory was needed to store the fractal code than the test image, resulting in data compression. As is the case for any compression algorithm, the goal of fractal image compression is to produce approximations of maximal fidelity subject to constraints on the memory required to store the fractal code. This also involves the question of optimal storage of fractal coefficients in terms of quantization and entropy encoding. In this section, a brief review of the author's previous work in this regard is presented and an assessment of the compression capabilities of the standard fractal scheme is made. The fractal code of an image is first examined.

The Fractal Code

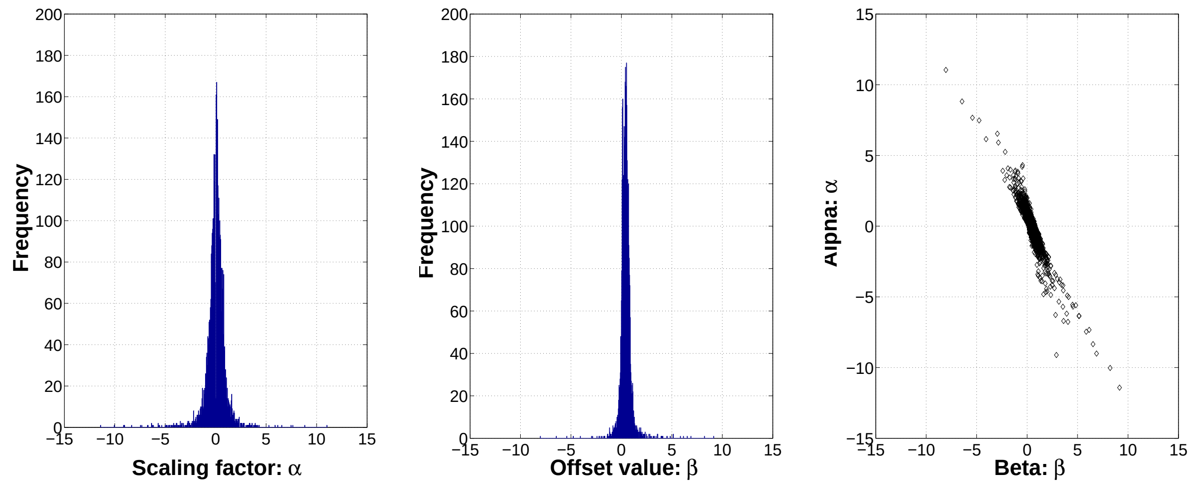
A sample of the fractal coefficients is listed in Table 2.1, and the gray-level coefficients, $\alpha^{*(m^*)}_{i^*(k),k}$ and $\beta^{*(m^*)}_{i^*(k),k}$, are illustrated in Figure 2.6. Clearly, the address of the optimal parent block and the index of the geometric map are discrete integer-valued, so no quantization is required and they can easily be encoded losslessly using a fixed-rate coding method. However, the gray-level coefficients $\alpha^{*(m^*)}_{i^*(k),k}$ and $\beta^{*(m^*)}_{i^*(k),k}$ are real-valued and belong to a continuous range that is "almost" symmetrically clustered around the origin. Thus, these coefficients need to be quantized before they can be encoded. These issues are addressed next.

Child block		Optimal parent block		Grey-level coefficients		Optimal geometric map
k	l	$i^*(k,l)$	$j^*(k,l)$	$\alpha^{*(m^*)}_{i^*j^*,kl}$	$\beta^{*(m^*)}_{i^*j^*,kl}$	$w^{m^*}_{i^*j^*,kl}$
1	1	14	8	0.413	0.425	4
1	2	21	17	0.986	-0.853	7
1	3	4	23	0.120	0.795	8
1	4	1	23	0.891	-0.704	8
...	...	...	...	...	...	...
...	...	...	...	...	...	...
64	64	7	25	0.433	0.109	4

Table 2.1: A sample from the fractal code of "Lenna" for $(M, N) = (32, 64)$.

Quantization of the Gray-level Coefficients

In [33], many issues related to devising efficient quantization and entropy encoding strategies for the fractal transform coefficients in order to achieve compression gain, were investigated. Three quantization strategies for quantizing the gray-level coefficients were studied and implemented. First, a uniform as well as Lloyd-Max probability density function (pdf) optimized quantizers, were designed. This was done by fitting the probability distributions of the gray-level coefficients to a known probability distribution, namely the Laplacian. As illustrated in Figure 2.6 (a)-(b), the distributions of the gray-level coefficients, $\alpha^{*(m^*)}_{i^*(k),k}$ and $\beta^{*(m^*)}_{i^*(k),k}$, are peaked and fast-decaying, resembling a Laplacian probability density function. A statistical justification was provided for such probability fitting using the goodness-of-fit statistics [33]. The gray-level coefficients were also explored for any type of statistical dependence and attempted to exploit such redundancies for the purpose of designing a vector quantizer. In fact, as reflected in Figure 2.6 (c), it was observed that these transform coefficients exhibit a high degree of linear dependence that can be exploited by using an appropriate vector quantizer. The Linde-Buzo-Gray (LBG) vector quantizer was used in order to quantize both gray-level coefficients simultaneously. The linear dependence between the



(a) Histogram of α (b) Histogram of β (c) Plot of α vs. β

Figure 2.6: Histograms of the gray-level coefficients as well the linear dependence between these coefficients.

gray-level coefficients was also exploited in a different way. The highly correlated coefficients were first decorrelated, resulting in a significant reduction in the variation range of at least one of the two variables. The decorrelated variables were then quantized, independently, using appropriate uniform quantizers. The inherent correlation was then re-introduced to obtain the quantizations of the original coefficients. This resulted in a significant compression gain. For each quantization strategy, the Huffman algorithm was used for binary coding of the quantized fractal coefficients, achieving a near-entropy bit rate performance. Figure 2.7 illustrates the fractal representations of "Lenna" using the various proposed quantization strategies. Note that most of these quantization methods yield high quality fractal representations of the test image at relatively high compression ratios.

In this section, it was shown that the standard block-based image compression scheme provides high-quality representations of real-world images at relatively high compression ratios. However, this scheme is not without drawbacks, as it suffers from three main limitations: computational complexity, restrictive uniform partitioning and blockiness artifacts in the fractal representation of the image. Next, an outline some of the recent developments that aim to overcome some of these limitations will be given.

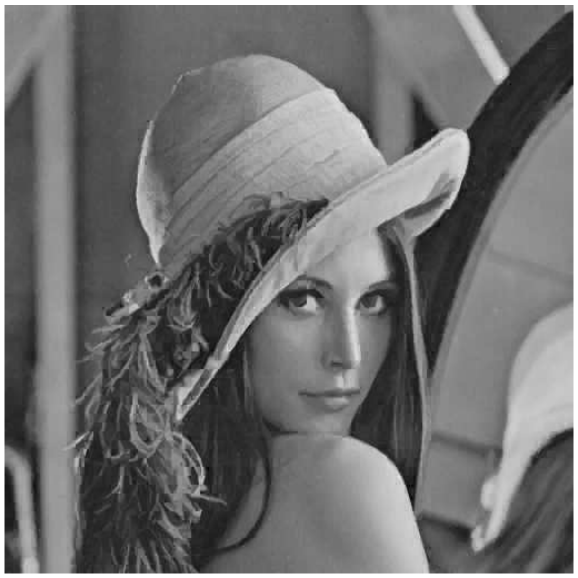
2.1.6 Developments in Fractal Image Coding

Over the years, a variety of highly competitive fractal image compression methods have been developed. Most of these schemes share a common feature, performing some kind of block collage coding as originally introduced by Jacquin [43]. Their variety lies in the different strategies employed to obtain the best possible matching of blocks within an image, subject to constraints. (The books [5, 26, 27, 52], represent excellent surveys of the field. Perhaps the most complete listing of papers on fractal image coding is to be found at the following Leipzig fractal compression website `http://www.informatik.uni-leipzig.de/cgip/`.) These schemes yield results that are comparable, in terms of rate-distortion performance, to some of the most efficient still image compression methods.

Next, an outline of some of the main developments that aim for overcoming some of the limitations of the original fractal image compression scheme will be presented.

Reducing the Computational Complexity

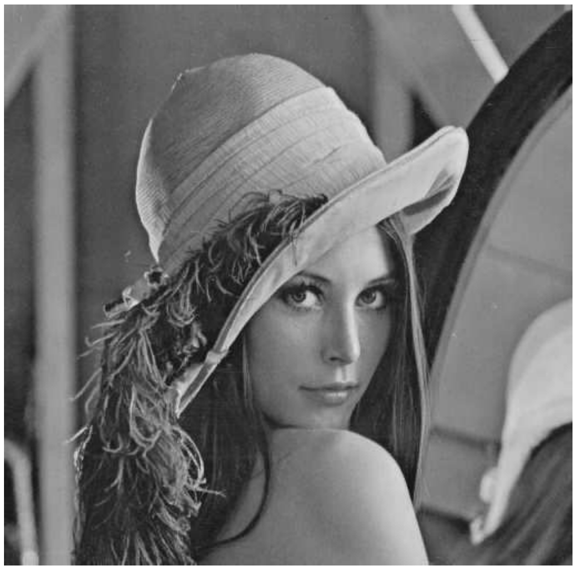
As discussed previously, the standard fractal encoding process adopts an exhaustive search strategy in which all parent sub-blocks as well as all possible geometric maps are tested to match a child block $\mathcal{R}_k$ with its corresponding globally optimal parent block $\mathcal{D}_{i^*(k)}$. This procedure is obviously computationally expensive. Many research efforts have focused on overcoming the computational complexity of the standard fractal scheme. Some investigate region-based image coding methods,



(a) Pdf-optimized uniform quantization

RMSE=8.05, PSNR=30.01, C_R=21:1.

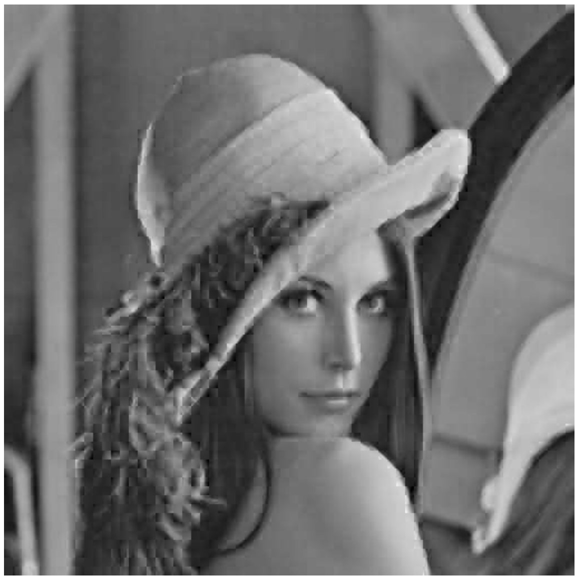
Execution time ≈ 1231 secs.



(b) Pdf-optimized Lloyd-Max quantization

RMSE=7.34, PSNR=30.82, C_R=18:1.

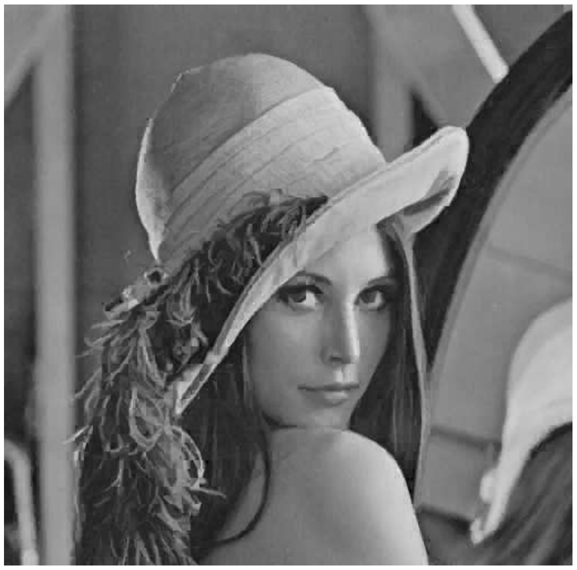
Execution time ≈ 1389 secs.



(c) LBG vector quantization

RMSE=8.80, PSNR=29.24, C_R=22:1.

Execution time ≈ 1534 secs.



(d) Decorrelation and uniform quantization

RMSE=8.42, PSNR=29.63, C_R=24:1.

Execution time ≈ 1423 secs.

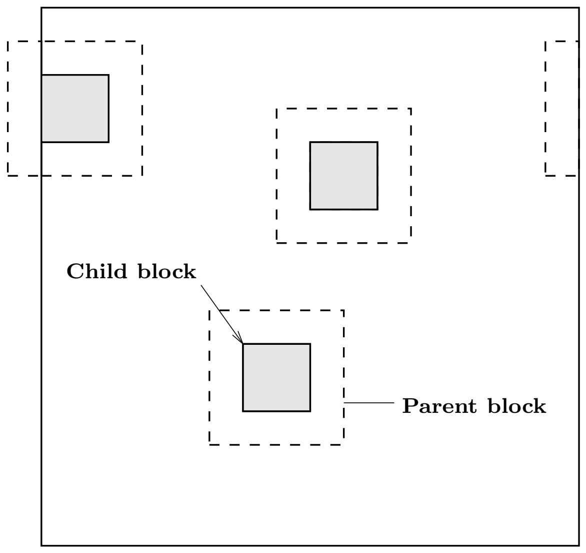
Figure 2.7: Fractal-representations of the test image of "Lenna" using various quantization strategies of the gray-level coefficients.

such as the use of the quadtree-partitioning in [8, 26, 67]. Others proposed combining the fractal transform with other transform methods such as wavelet in [18, 47, 57, 70], to yield a fast fractal-wavelet image coding method. This hybrid fractal-wavelet method will be studied in more details in section 2.3. Recently, classification methods based on the local energy were proposed to speed up the fractal encoding process, such as in [48].

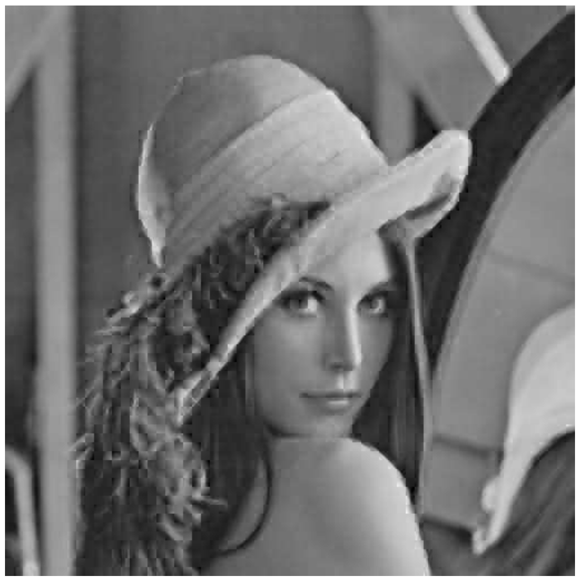
Next, a brief description and practical implementation of a fast, search-free fractal-based image coding scheme, known as the Bath Fractal Transform (BFT) scheme, are presented.

The Bath Fractal Transform (BFT)

The BFT is a hybrid of the standard fractal scheme discussed above [58, 59, 72]. The main difference lies in the fact that the BFT does not require any search or geometric maps (other than the shrinking transformation) in the process of matching child sub-blocks with parent sub-blocks. In fact, as illustrated in Figure 2.8 (a), a child block is simply matched with its "co-centric" parent block. However, this reduction in computational complexity comes at the expense of adding extra parameters to fit the gray-level values. A relatively small degradation in the fidelity of the approximation and reduction in compression ratio are also sacrificed.



(a) Child-parent matching



(b) BFT representation: Execution time ≈ 5 secs. RMSE=10.11, PSNR=28.03, C_R=17.01

Figure 2.8: For BFT coding, each block is matched with its "co-centered" parent block with wrap-around for border sub-blocks.

The place-dependent BFT gray-level maps used in this study have the form:

$$\phi(x, y, f(x, y)) = \alpha f(x, y) + \beta + \gamma_x x + \gamma_y y, \tag{2.13}$$

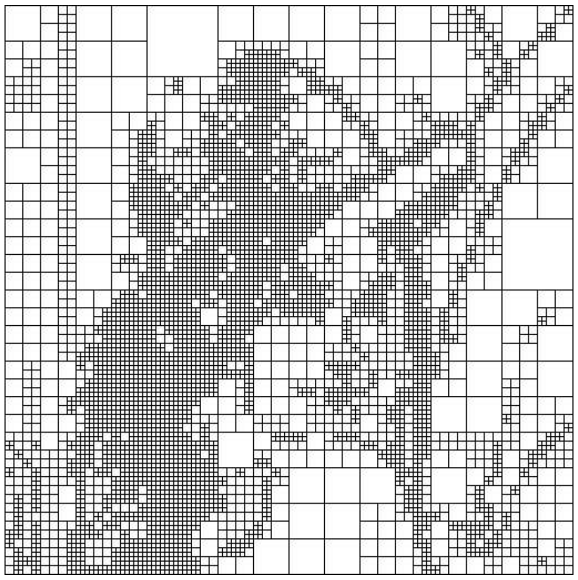
where the final three terms correspond to the addition of a planar or "ramping" term to the fractal term $\alpha f(x, y)$. Similar to the standard fractal scheme, in the case of non-overlapping child sub-blocks, these coefficients are computed using the least squares method. The fractal code of the BFT scheme consists simply of the coefficients:

$$\{\alpha_k, \beta_k, \gamma_{x,k}, \gamma_{y,k}, \quad k = 1, 2, \ldots, N^2\}. \tag{2.14}$$

As was the case for the standard fractal scheme, it was observed that the fractal coefficients corresponding to the BFT scheme still exhibit some degree of linear dependence [33]. These redundancies were exploited for the purpose of obtaining higher compression, and the representation of "Lenna" using the BFT scheme is illustrated in Figure 2.8 (b).

Quadtree Image Partitioning

Another weakness of the standard fractal scheme lies in the fact that it adopts a uniform image-independent square partition of the image. The use of fixed-size partitions may have limitations since there are sub-regions in the image that are difficult to cover using the prescribed resolution or size of the partition. For instance, high detail sub-regions of the image, such as edges, may require a small mesh size to be represented well. On the other hand, there may be sub-regions in the image that can be covered well using larger block sizes, hence resulting in a reduction of the total number of parameters to be stored and an increase in compression of the image. The most common method of adaptive image partitioning is that of quadtrees: Starting with the original image, square pixel sub-blocks are broken down into quadrants in a recursive tree structure. The partitioning, which will vary throughout the image, is terminated when a particular condition is satisfied. Typically, regions of higher image activity, for example edges, will produce partitions of finer resolution, i.e., small block sizes. Consequently, edges are generally represented well in quadtree-based coding schemes, including fractal coding. This process is illustrated nicely in Figure 2.9. Note how certain blocks are subdivided further into four quadrants while others are encoded and not further decomposed. Also, note that the regions of the image that contain "too many" details, such as the eyes and hair and the edges of the hat, are subdivided into a finer level, in some cases the minimum allowable block size is reached. Similarly, Note how relatively flat parts of the image, such as the face and the background, are partitioned more coarsely. This is indeed the essence and the benefit of the quadtree partitioning scheme.


(a) Quadtree image partitioning

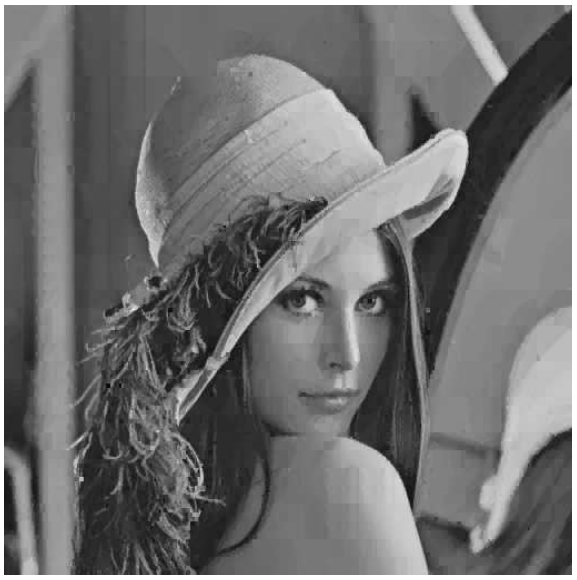

(b) RMSE=6.81, PSNR=31.47, C_R=29:1. Execution time $\approx$ 253 secs.

Figure 2.9: Quadtree partitioning of the image for the purpose of fractal image coding as well the quadtree-based fractal representation of the test image.

The use of quadtrees in fractal coding was originally discussed in detail by Fisher [26], from which much work has been done. Originally, the quadtree-based fractal image coding scheme adopts a collage decomposition criterion. A child sub-block is fractally encoded and the collage error, which describes the goodness of fit, is computed. If the resulting collage error is within a prescribed tolerance, then the child is presumed fractally encoded. However, if the collage error exceeds the prescribed threshold, then the child sub-block is sub-divided into four uncoded child sub-blocks (quadtrees). This process is then repeated until the whole image is partitioned into non-overlapping fractally encoded child sub-blocks.

In [34], the use of the quadtree partitioning scheme for the purpose of fractal image compression was investigated in some detail. Various decomposition strategies were explored and rate distortion curves for fractal image compression schemes using the quadtree partitioning algorithm were generated. It was shown that the variance of the child sub-block is a quadtree decomposition criterion than the collage error, previously proposed for the purpose of quadtree-based fractal image coding. The main advantage of the variance decomposition criterion over the collage one is that the resulting quadtree-based fractal scheme is significantly faster than the one using the collage error. This is because one need not fractally encoded a child block before deciding whether to decompose a child sub-block. In fact, only those child sub-blocks with high (i.e. greater than some threshold) variances are fractally encoded. This quadtree-based fractal scheme using the child variance as a decomposition criterion is outlined next.

A Quadtree-Based Standard Fractal Algorithm:

For each uncoded child sub-block, $\mathcal{R}_k = \{y_{k,1}, y_{k,2}, \dots, y_{k,n}\}$, compute its variance σ_k^2:

$$\sigma_k^2 = \frac{1}{n}\sum_{j=1}^{n}(y_{k,j} - \bar{y}_k)^2, \text{ where } \bar{y}_k = \frac{1}{n}\sum_{j=1}^{n} y_{k,j}. \tag{2.15}$$

- If σ_k^2 is less than some prescribed threshold σ_c^2, then the child sub-block $\mathcal{R}_k$ is encoded using the standard fractal scheme, and marked as fractally coded.
- Otherwise, the child sub-block $\mathcal{R}_k$ is split into four equal sub-blocks (quadtrees) denoted by $\mathcal{R}_{k_1}, \mathcal{R}_{k_2}, \mathcal{R}_{k_3}$ and $\mathcal{R}_{k_4}$ which are then labeled as uncoded.
- Repeat this process until all the child sub-blocks are fractally coded.

One of the advantages of the quadtree-based standard fractal scheme is the ability to generate rate distortion curves by varying the energy criterion threshold σ_c^2. Figure 2.10 illustrates the rate distortion curves for the quadtree-based standard fractal scheme outlined above. The “bumpiness” in these curves may be explained as follows: These rate distortion curves were generated by varying the child-block variance threshold, σ_c^2. For larger values of σ_c^2, the partition of the image is rather coarse. Thus decreasing this threshold is expected to result in many child sub-blocks being partitioned into four quadtrees. This results in relatively significant changes in the rate-distortion quality of the fractal representation. Similarly, for smaller values of σ_c^2, the partition of the image is becoming finer. Thus decreasing this threshold even further results in decomposing some of the larger blocks and yielding an even finer partition of the image. This again results in a relatively significant change in the rate-distortion quality of the fractal representation. For medium range threshold values, the rate distortion quality of the fractal representation appear to vary continuously with the variation of the child block variance threshold, σ_c^2. This explains the steepness of these curves for lower and higher compression ranges.

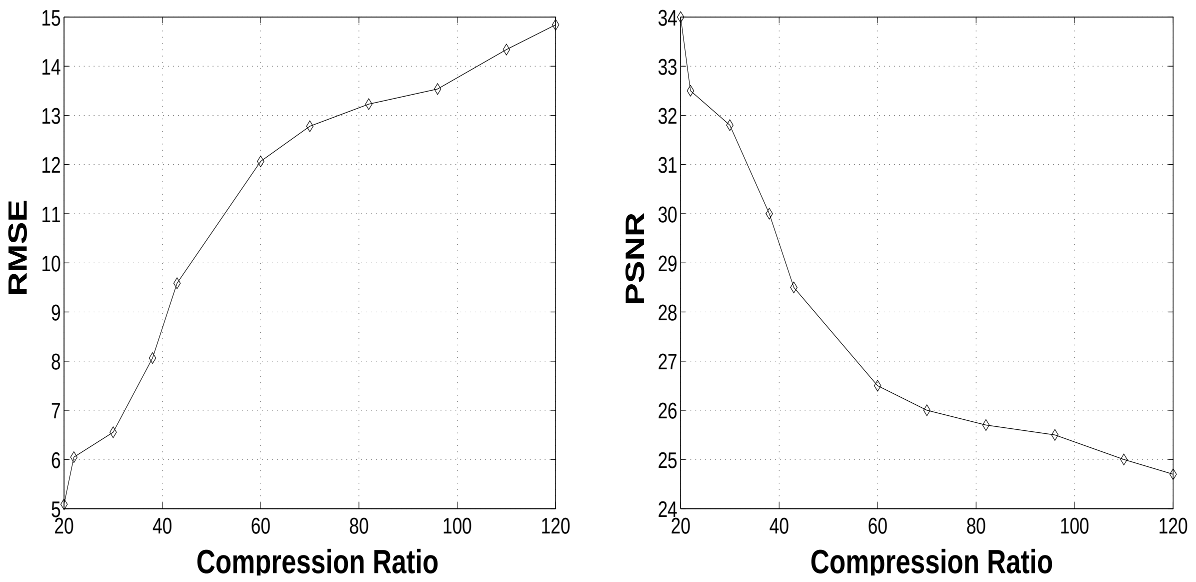


Figure 2.10: Rate distortion curves of the quadtree-based standard fractal scheme, for the test image of "Lenna".

Reducing the Blockiness Artifacts

Standard fractal schemes are based on spatial transformations among the target image sub-blocks; as a result, the reconstructed image generally suffers from disturbing artifacts or blockiness. In fact, as the image is partitioned into blocks and since errors tend to be strongly correlated within a block but generally uncorrelated across neighboring blocks, very distracting artifacts in the fractal representation of an image are often observed. Note that zooming on the fractal representation reveals the blockiness artifacts, as illustrated in Figure 2.11.

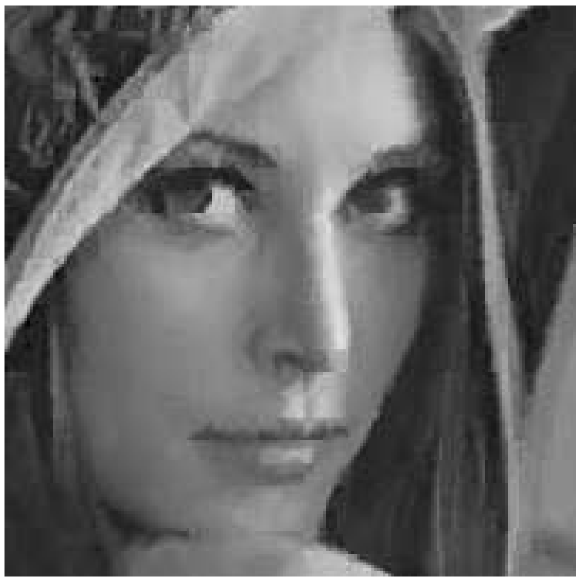


Figure 2.11: Zooming in on the fractal representation reveals the blockiness artifacts.

Fractal-wavelet transforms were introduced in an effort to reduce the blockiness artifacts and computational complexity that are inherent in standard fractal image compression. Fractal-wavelet schemes will be discussed in more detail in section 2.3.

In this section, various fractal-based image coding schemes were described and implemented and some of the significant developments in fractal image compression over the past few years were outlined. Next, a brief discussion of the discrete wavelet transform and its application for the purpose of image coding and compression will be given.

2.2 Discrete Wavelet Image Coding

The wavelet transform has many unique features that have made it a popular method for the purpose of image processing and compression. The wavelet transform performs a high degree of decorrelation between neighboring pixels, and it provides a distinct localization of the image in the spatial as well as the frequency domain. This transform also provides an elegant subband framework in which both high and low frequency components of the image can be analyzed separately. However, one of the major difficulties in wavelet-based coding schemes is that significant wavelet coefficients corresponding to important edge information and other high-frequency content of the signal are often dispersed among a large number of insignificant coefficients. The problem encountered in wavelet-based image coding methods is how to efficiently detect and represent the locations of these significant coefficients without spending most of the allocated bit-budget. Recently, many highly efficient wavelet-based image coders have been developed. In particular, the Embedded Zerotree Wavelet (EZW) [68] and the Set Partitioning in Hierarchical Trees (SPIHT) [66] schemes, which are considered as benchmarks for the performance of wavelet-based image compression methods.

2.2.1 The Discrete Wavelet Transform

There are many wavelet systems that can be used effectively, such as the “Haar”, “Daubeschies”, “Coiflets”, “Symlets”, “Morlets”, “Mexican Hat”, “Meyer” and “Biorthogonal” wavelets [10]. All these wavelet systems have the following three general characteristics:

1. A wavelet system is a set of building blocks to construct or represent a signal or function. It is a two-dimensional expansion set (usually a basis) for some class of one- (or higher) dimensional signals.

2. The wavelet expansion gives a time-frequency localization of the signal. This means most of the energy of the signal is well presented by a few expansion coefficients.

3. The calculation of the coefficients from the signal can be done efficiently. It turns out that many wavelet transforms (the set of expansion coefficients) can be calculated with $O(N)$ operations. This means that the number of floating-point multiplications and additions increase linearly with the length of the signal. More general wavelet transforms require $O(N \log(N))$ operations, the same as for the fast Fourier transform (FFT) [10].

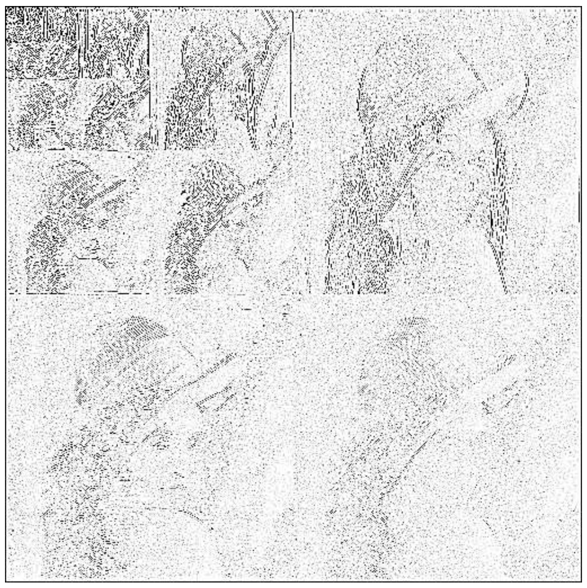
The reader is referred to [10, 19, 53], to mention only a few, for more detailed study of the mathematical foundation and algorithmic implementation of the discrete wavelet transform. There are many software toolboxes for the implementation of the discrete wavelet transform, such as the MATLAB Wavelet Toolbox [56] and WAVELAB [71].

Throughout this thesis, the Daubechies wavelet system [19] of order $N = 8$, denoted as “Db8” is used, when implementing wavelet-based schemes. For the purpose of signal denoising, a smooth wavelet system is generally desired. While there are many wavelet systems that possess varying degree of smoothness and regularity, the selection of the “Db8” wavelet, which possesses the required properties, is somewhat arbitrary. Clearly, one could have chosen any one of the other smooth wavelets, such as “Symlets” or “Coiflets” wavelets.

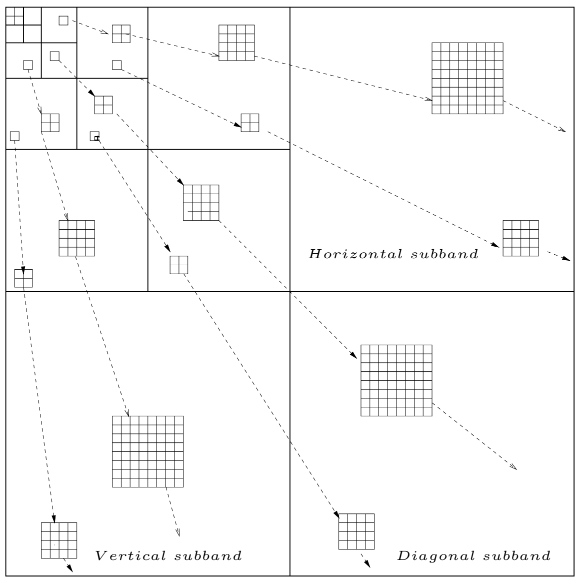
Next, some of the properties of the wavelet decomposition tree are described.

2.2.2 Properties of the Wavelet Decomposition Tree

The wavelet coefficients of an image are often organized in a pyramid structure known as the *wavelet decomposition tree*. This tree is constructed through a recursive four-subband splitting, starting with the original image. This process was applied using the 512×512 test image of “Lenna”, as illustrated in Figure 2.12. This figure also illustrates how the wavelet coefficients of an image are arranged in a spatial orientation tree, also known as the wavelet decomposition tree. The wavelet decomposition tree is divided into three subbands (horizontal, vertical and diagonal), and a number of levels. The wavelet decomposition tree of an image can be decomposed into subtrees that consist of the wavelet coefficients in the same spatial positions for the various wavelet decomposition levels, in the three subbands. A subtree can be rooted anywhere in the spatial orientation tree and the node or root of the subtree is a coefficient identified by a set of coordinates, (i, j), the hierarchical level k and the subband $\lambda \in \{h, v, d\}$, generally denoted as a^{λ}_{kij}. The 2×2 block of pixels in the same spatial location in the next finer level are called children or offspring of a^{λ}_{kij}. The collection of coefficients that are in the same spatial location in all the lower levels are called descendants of



(a) Fully decomposed wavelet tree for "Lenna"



(b) Structure of the wavelet tree

Figure 2.12: The wavelet decomposition tree of an image, using the "Db8" wavelet basis.

a^{λ}_{kij}. Let A^{λ}_{kij} denote the subtree with node a^{λ}_{kij}, $\lambda \in \{h, v, d\}$. Some of the characteristics of the wavelet tree include:

- The coefficients in each subband correspond to edge information in that direction.
- Most of the energy of the image is concentrated in the low frequency components.
- Most importantly, it has also been observed that there is a spatial self-similarity between subbands. That is, the wavelet coefficients in higher level subtrees can be estimated by scaled copies of the wavelet coefficients in lower-level subtrees. This inherent self-symmetry within the wavelet tree motivates the use of fractal-based methods to encode the wavelet coefficients, as will be discussed in the next section.

The DWT has a vast number of applications. In this chapter, the focus is on its usefulness for the purpose of image compression. The use of wavelet-based image compression methods is first discussed.

2.2.3 Wavelet Image Compression

Generic wavelet based image compression techniques exploit the fact that the wavelet transform concentrates most of the energy of the image in a relatively small number of coefficients. The strategy is as follows: An optimal threshold for the coefficients is computed in such a way that a certain percentage of the energy of the image is preserved after compression. Then, coefficients with values below the threshold are deemed to be insignificant and forced to zero, while the rest of the coefficients are quantized and encoded in a refined fashion. For typical images, most of the energy of the image is generally localized in a relatively few coefficients, hence most of the coefficients can be insignificant and discarded, resulting in a some degree of compression. However, more sophisticated wavelet compression techniques can outperform this generic approach. These methods exploit the characteristics and structure of the wavelet decomposition tree in order to locate the significant coefficients.

Locating the Significant Coefficients

The discrete wavelet transform attempts to produce coefficients that are decorrelated with most of the energy of the image localized in a relatively few coefficients, as compared to the spatial distribution of the pixels in the original image. For a typical real-world image, the image is composed of mainly “trends” or relatively smooth areas where neighboring pixels are highly correlated. However, the most important features of the image in terms of the human perception lie in the edges and boundaries of the image. These features have lower cumulative energy than the rest of the image, however they contain perceptual significance that is far greater than their numerical energy contribution to the image. The wavelet transform attempts to separate these two main features of the image and localize them at various scales and in three different subbands. Typically, most of the energy of the image is localized in the lowest frequency components of the image (top left-corner of the wavelet decomposition tree), whereas most of the edge information or high frequency components of the image are scattered in the higher scales of the wavelet decomposition tree. Thus, the fine details or the high frequency components (edges) of the image constitute the most important perceptual characteristics of the image and they are often scattered among a large number of insignificant coefficients. Hence, if not done efficiently, this may represent a problem for wavelet-based image coding methods, as most of the bit budget may be spent in representing and coding the position of those few coefficients corresponding to significant edges or fine details. The challenge in wavelet-based image coding methods is how to efficiently locate these high-information coefficients and representing the positions of the significant wavelet coefficients.

There are many wavelet-based image compression methods, but most of them only differ in the way they locate and encode the significant coefficients. Two of the most efficient wavelet-based image coding methods are the Embedded Zerotrees of Wavelet (EZW) method and the Set Partitioning in Hierarchical Trees (SPIHT) scheme, which are discussed briefly next.

Efficient Wavelet Image Coding Schemes

Over the past decade, many efficient wavelet-based image compression schemes have been developed. Two of the best wavelet image compression schemes, widely known as the Embedded Zerotrees Wavelet (EZW) [68] and the Set Partitioning in Hierarchical Trees (SPIHT) algorithms [66]. In 1993, Shapiro proposed the use of a special structure called *zerotree* for the purpose of locating and encoding the significant wavelet coefficients [68]. The embedded zerotree wavelet algorithm (EZW) is a simple yet remarkably effective image compression algorithm, having the property that the bits in the bit stream are generated in order of importance, yielding a fully embedded code. This highly efficient wavelet-based image compression scheme is based on the following significance hypothesis:

> If a wavelet coefficient at a coarse scale is insignificant with respect to a threshold then all of its descendants are also insignificant.

The embedded code represents a sequence of binary decisions that distinguish an image from the "zero" image.

In 1996, Said and Pearlman [66] proposed an enhanced implementation of the EZW algorithm, known as the Set Partitioning in Hierarchical Trees (SPIHT). Their method is based on the same premises as the EZW algorithm, but with more attention to detail. The public domain version of this coder (which is available from http://www.cipr.rpi.edu/research/SPIHT/) is very fast, and improves the performance of the EZW by 0.3-0.6 dB. Next, the main features of the SPIHT scheme are summarized and its performance is assessed.

The Main Features of the SPIHT Algorithm

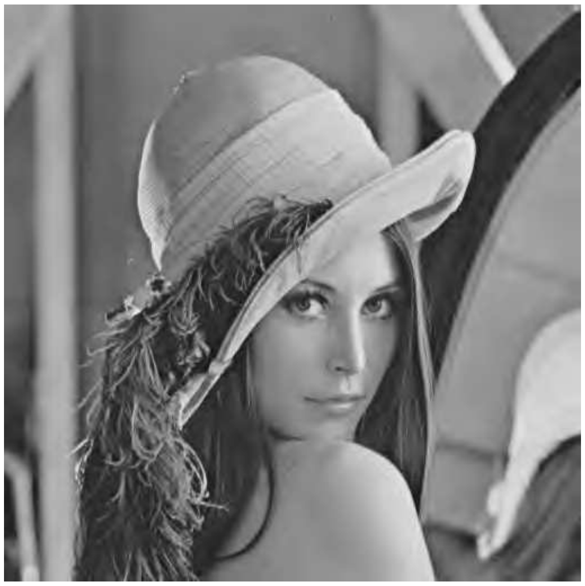
In summary, the SPIHT algorithm partitions the wavelet coefficients into three sets: list of significant pixels, list of significant sets, and list of insignificant sets. By using this structure and conditionally entropy encoding in these symbols, the coder achieves very good rate-distortion performance. In addition, the SPIHT coder also generates an embedded code. Coders that generate embedded codes are said to be have *progressive transmission* or *successive refinement* property. Successive refinement consists of first approximating the image with a few bits of data, and then improving the approximation as more and more information is supplied. An embedded code has the property that for two given bit rates: $R_1 \geq R_2$, the rate R_2 code is a prefix to the rate R_1 code. Such codes are of great practical interest for the following reasons:

- The encoder can easily achieve a precise bit-rate by continuing to output bits until it reaches the desired bit-rate.
- The decoder can cease decoding at any given point, generating an image that is the best representation possible with the decoded number of bits. This is of practical interest in many applications, including broadcast applications where multiple decoders with varying computational, display and bandwidth capabilities attempt to receive the same bit-stream. With an embedded code, each receiver can decode the passing bit-stream according to its particular needs and capabilities.
- Embedded codes are also useful for indexing and browsing, where only a rough approximation is sufficient for deciding whether the image needs to be decoded or received in full. The process of screening images can be sped up considerably by using embedded codes.

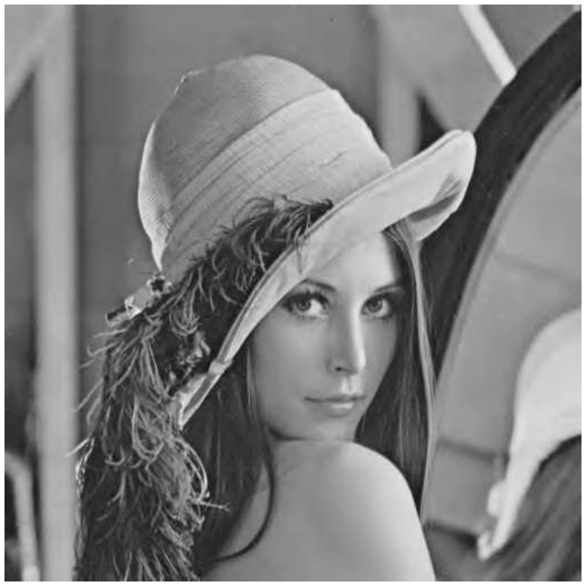
The SPIHT method generates an embedded code by using a bit-slice approach. First the wavelet coefficients of the image are indexed into a one-dimensional array, according to their order of importance. This order places lower frequency bands before higher frequency bands since they have more energy, and coefficients within each band appear in a raster scan order. The bit-slice code is generated by scanning this one-dimensional array, comparing each coefficient with a threshold T. This initial scan provides the decoder with sufficient information to recover the most significant bit slice. In the next pass, new information about each coefficient is refined to a resolution of $\frac{T}{2}$, and the pass generates another bit slice of information. This process is repeated until there are no more slices to code.

The SPIHT algorithm is indeed embedded, progressive and computationally efficient. Figure 2.13 illustrates some typical SPIHT representation of the test image compressed at pre-determined bit-rates as well as the rate distortion performance of the SPIHT method.

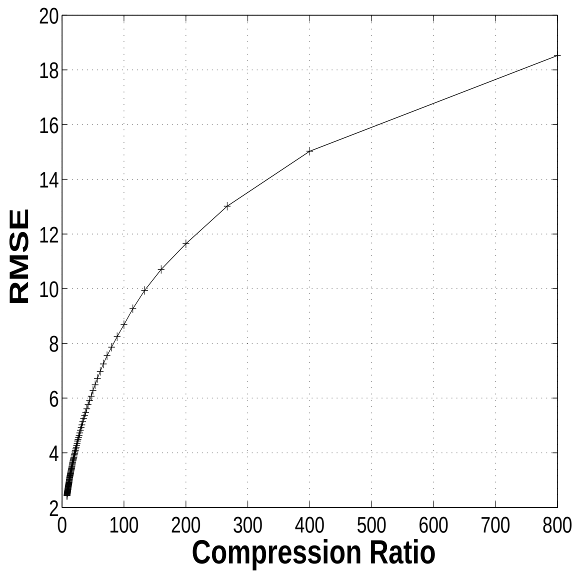
In this section, a brief outline of the practical implementation of the DWT for the purpose of image compression is given. In particular, the main features of the SPIHT method, which is one of the most effective wavelet-based image codec, are described. Next, the hybrid fractal-wavelet scheme which combines the fractal and the wavelet transforms studied so far, is studied.


(a) RMSE = 5.61, PSNR = 33.15 dB, C_R=40:1

Execution time ≈ 55 secs


(b) RMSE = 3.93, PSNR = 36.24 dB, C_R=20:1

Execution time ≈ 51 secs



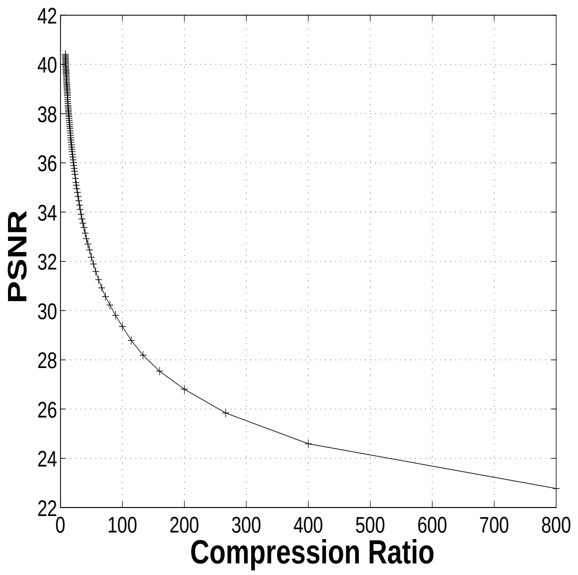


Figure 2.13: Results of the SPIHT compression algorithm for the image of "Lenna": (a)-(b) illustrate SPIHT compressed images and (c)-(d) illustrate the rate distortion performance of the SPIHT method.

2.3 Fractal-Wavelet Image Coding

Standard fractal schemes exploit the self-similarities that are inherent in many real-world images to encode an image as a collection of transformations. As shown earlier in this chapter, these schemes provide efficient and accurate models for many real world images, resulting in relatively high compression ratios for a wide class of images that exhibit some degree of local or global self-similarity. However, fractal-based schemes are not without limitations and much remains to be investigated before such a novel technique is comparable to other image compression technology currently in use. As discussed earlier, some of the disadvantages of the conventional fractal schemes include expensive computational requirements and blockiness artifacts in fractal representations of images. These schemes are based on spatial transformations between sub-blocks of the image; as a result, the reconstructed image generally suffers from disturbing blockiness artifacts. A new class of *fractal-wavelet* (FW) transforms has recently been proposed and investigated [18, 47, 57, 70]. These fractal-wavelet image coding schemes were initially introduced to overcome the above mentioned limitations of the original standard fractal scheme. The FW transform exploits the local self-similarities that are inherent in the wavelet decomposition tree. That is, the wavelet coefficients in higher level subtrees are scaled copies of the wavelet coefficients in lower-level subtrees. As will be shown, the benefits of these fractal-wavelet techniques are numerous and they include; significant reduction in computational complexity, reduction of the blockiness artifacts and significant increase in the rate distortion quality of the fractal representation of the image.

2.3.1 Generalized 2D Fractal-Wavelet Transforms

Fractal-wavelet transforms, discovered independently by a number of researchers ([18, 47, 57, 70] to name only a few), were introduced in an effort to reduce the blockiness and computational complexity that are inherent in fractal image compression. Their action involves a scaling and copying of wavelet coefficient subtrees to lower subtrees, quite analogous to the action of fractal image coders in the spatial domain.

The FW Transform

For the fully decomposed wavelet tree, let $\mathbf{A}_k^h, \mathbf{A}_k^v, \mathbf{A}_k^d$ denote the horizontal, vertical and diagonal sub-blocks of wavelet coefficients at decomposition level k, $0 \leq k \leq K$, respectively. Each of these sub-blocks contains 2^{2k} coefficients; $a^h_{kij}, a^v_{kij}, a^d_{kij}$, respectively. The three collections of blocks

$$\mathbf{A}^h = \bigcup_{k=1}^{K} \mathbf{A}^h_k, \quad \mathbf{A}^v = \bigcup_{k=1}^{K} \mathbf{A}^v_k, \quad \mathbf{A}^d = \bigcup_{k=1}^{K} \mathbf{A}^d_k,$$

comprise the fundamental *horizontal*, *vertical* and *diagonal* subtrees of the coefficient tree, respectively. Now consider any wavelet coefficient a^λ_{kij}, $\lambda \in \{h, v, d\}$ in this matrix and the unique subtree, with this element as its root, this subtree will be denoted by A^λ_{kij}.

The two-dimensional fractal-wavelet transforms involve mappings of "parent" subtrees of wavelet coefficients to lower "child" subtrees. For simplicity in presentation and notation, we consider a particular case in which the roots of all parent quadtrees appear in a given block and the roots of all child quadtrees appear in another given block. Select two integers, the parent and child levels, k_1^* and k_2^*, respectively, with $1 \leq k_1^* < k_2^*$. Then for each possible index $1 \leq i, j \leq 2^{k_2^*}$ define the three sets of affine block transforms:

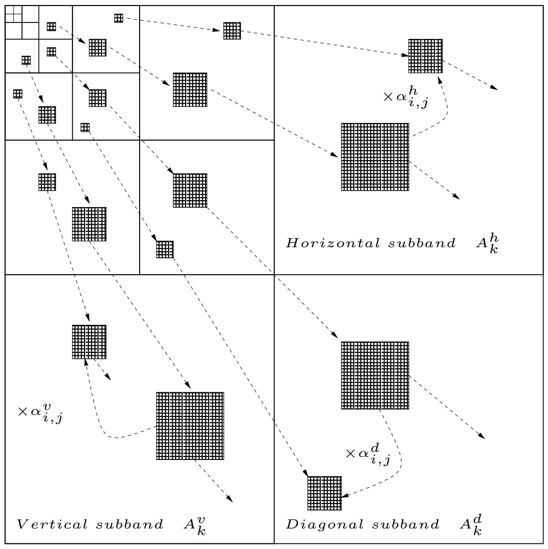


Figure 2.14: The FW transform.

$$\begin{aligned} \mathcal{W}^\lambda_{ij} \quad : \quad & A^\lambda_{k_1^*, i^\lambda(i,j), j^\lambda(i,j)} \rightarrow A^\lambda_{k_2^*, i, j}, \qquad (2.16) \\ & A^\lambda_{k_2^*, i, j} = \alpha^\lambda_{ij} A^\lambda_{k_1^*, i^\lambda(i,j), j^\lambda(i,j)}, \quad \lambda \in \{h, v, d\}. \end{aligned}$$

Note how the child subtrees at level k_2^* are replaced by scaled copies of parent subtrees from level k_1^*. This procedure is illustrated in Figure 2.14. These block transforms will comprise a unique fractal-wavelet (FW) operator $\mathcal{W}$. The use of the indices i^h, j^h, etc. emphasizes that the parent quadtrees corresponding to a given set of child quadtrees $A^h_{k_2^*,i,j}, A^v_{k_2^*,i,j}$ and $A^d_{k_2^*,i,j}$ need not be the same. As well, the scaling coefficients $\alpha^h_{ij}, \alpha^v_{ij}$ and α^d_{ij} are independent.

The "fractal code" associated with the generalized FW operator $\mathcal{W}$ consists of the following:

1. The parent-child index pair (k_1^*, k_2^*), generally $k_2^* = k_1^* + 1$.

2. The wavelet coefficients in blocks $\mathbf{B}_0$, and $\mathbf{A}^\lambda_k$, $\lambda \in \{h, v, d\}$ for $1 \leq k \leq k_2^* - 1$, a total of $4^{k_2^*}$ coefficients.

3. The scaling factors α_{ij}^{λ} and parent block indices, $(i^{\lambda}(i,j),\ j^{\lambda}(i,j))$, for all elements a_{ij}^{λ} in each of the three blocks $A_{k_2^*}^{\lambda}$. The total number of parameters:
   - $3 \times 4^{k_2^*}$ scaling factors,
   - $2 \times 3 \times 4^{k_2^*}$ indices.

It has been shown [57, 70] that, under certain conditions, the fractal-wavelet transform $\mathcal{W}$ is contractive in an appropriate complete metric space (l_2-type square summable sequences) of wavelet coefficients. For the special transform given in Eq. (2.16), contractivity is guaranteed when

$$c_Q = 2^{k_2^* - k_1^*} \max_{\lambda,i,j} |\alpha_{ij}^{\lambda}| \ < \ 1,$$

where $\lambda \in \{h, v, d\}$ and $0 \le i, j \le 2^{k_2^*} - 1$. From the Contraction Mapping Theorem, the condition $c_Q < 1$ guarantees the existence of a unique fixed point of the operator $\mathcal{W}$, that is, a unique wavelet coefficient tree, $\bar{c}$ such that $\mathcal{W}(\bar{c}) = \bar{c}$. Moreover, the wavelet tree $\bar{c}$ may be generated by iteration of $\mathcal{W}$.

The *standard* FW scheme, as described in [18, 47], is a special case of the generalized FW scheme, where it assumes that common parents and common scaling factors are used for the various subbands, that is

$$\begin{aligned} i^h(i,j) &= i^v(i,j) = i^d(i,j) \\ j^h(i,j) &= j^v(i,j) = j^d(i,j) \\ \alpha_{ij}^h &= \alpha_{ij}^v = \alpha_{ij}^d. \end{aligned}$$

In other words, the h, v and d subbands are not treated independently.

Next, a few FW schemes that differ only in whether the three subbands (horizontal, vertical and diagonal) of the wavelet tree are combined together or treated independently, are described and implemented.

2.3.2 Fractal-Wavelet Schemes

In [33, 34], various types of FW schemes were investigated their performance as measured by their corresponding rate distortion curves, was assessed. Only a brief description of these schemes is given here.

As in the case of conventional fractal-based compression methods, there exists a variety of strategies for "optimal" parent block assignment, involving some kind of searching over feasible parent block indices (i', j'). The optimal strategy is to perform a full search of all possible parent blocks within a subband λ in level k_1^*. However, this is expensive from both a computational as well as coding point of view. However, restrictive searches requiring much less computational time often yield good results with relatively small sacrifices in accuracy [33, 34].

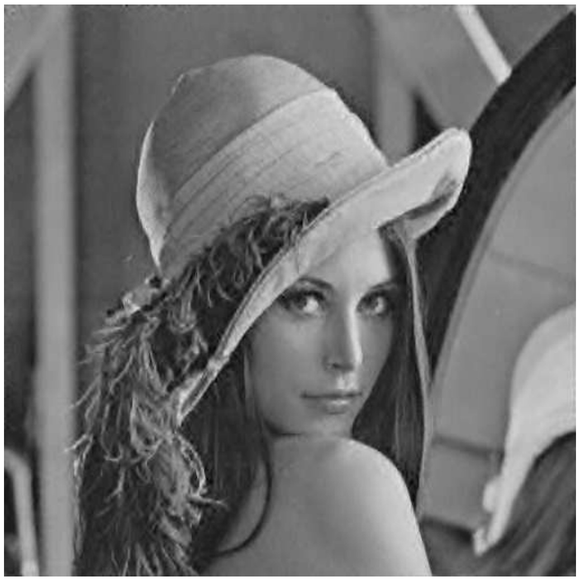
Scheme	α's	Parents	Description
Exhaustive FW-I	$(\alpha^h, \alpha^v, \alpha^d)$	Independent	Generalized FW scheme with independent parent blocks, i.e. the three indices $(i^\lambda(i,j), j^\lambda(i,j))$, and independent scaling coefficients α^λ, $\lambda \in \{h, v, d\}$.
Standard FW-II	$(\alpha^h = \alpha^v = \alpha^d)$	Common	Quite restrictive scheme; the indices $(i^\lambda(i,j), j^\lambda(i,j))$ and the coefficients α^λ are the same for $\lambda \in \{h, v, d\}$.

Table 2.2: The two fractal-wavelet schemes studied here.

Table 2.2 illustrates two FW schemes corresponding to two different ways of choosing the parent subtrees and scaling coefficients for the horizontal, vertical and diagonal subbands. Note that FW-I treats the three subbands independently while FW-II combines the three subbands. These schemes were implemented for the purpose of image representation and compression. The results are illustrated in Figure 2.15. For each scheme, the figure illustrates the approximation at the levels $(k_1^*, k_2^*) = (4, 5)$ and $(k_1^*, k_2^*) = (5, 6)$. For the scaling coefficient, a midriser uniform quantizer, with an appropriate range and number of levels, was used. The results are also presented below each of the images.

The implementation of these fractal-wavelet schemes shows that the application of the fractal-based schemes in the wavelet domain has many advantages, as compared to the conventional spatial-based fractal methods. Some of these benefits include:

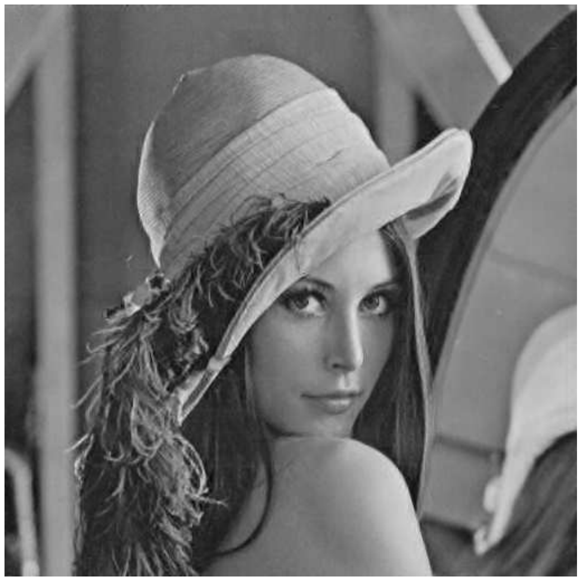
- Significant reduction in computational complexity and encoding time. This is reflected in the reduction of the execution times when comparing Figures 2.5 and 2.15.
- Better approximations at relatively higher compression ratios are also achieved. This can be seen by comparing the results of the standard fractal representations of "Lenna", to those obtained by using the fractal-wavelet schemes. Comparing Figure 2.7 to Figure 2.15, reveals that significant gain in compression as well as approximation fidelity are achieved by applying



FW-I scheme with $(k_1^*, k_2^*) = (4,5)$

RMSE = 10.42, PSNR = 27.78, C_R=47:1.

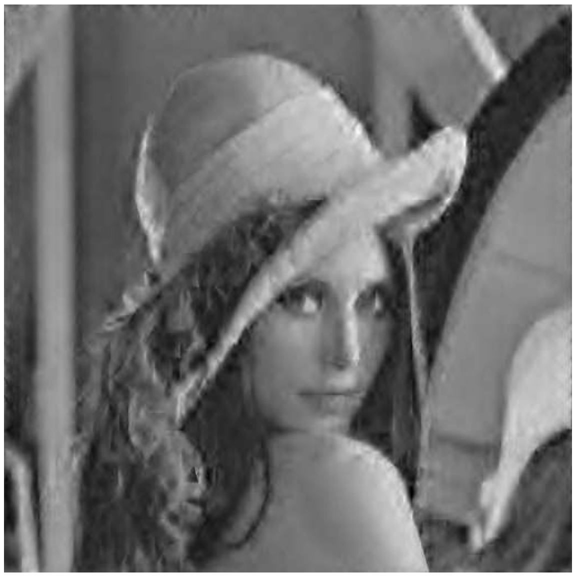
Execution time $\approx$ 43 secs.



FW-I scheme with $(k_1^*, k_2^*) = (5,6)$

RMSE = 6.27, PSNR = 32.20, C_R=11:1.

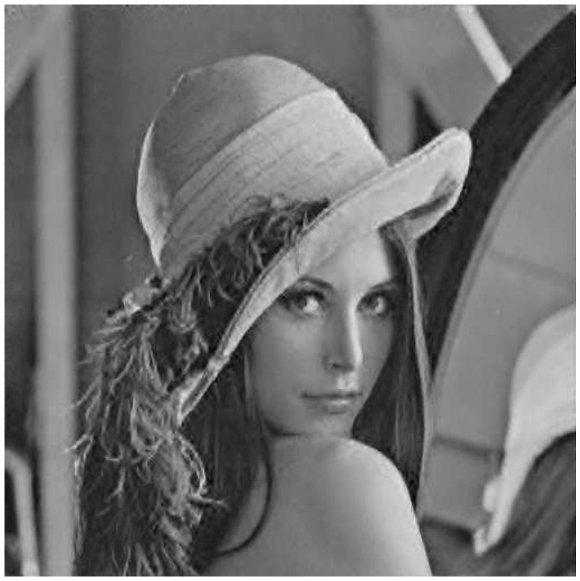
Execution time $\approx$ 87 secs.



FW-II scheme with $(k_1^*, k_2^*) = (4,5)$

RMSE = 12.81, PSNR = 25.98, C_R=112:1.

Execution time $\approx$ 15 secs.



FW-II scheme with $(k_1^*, k_2^*) = (5,6)$

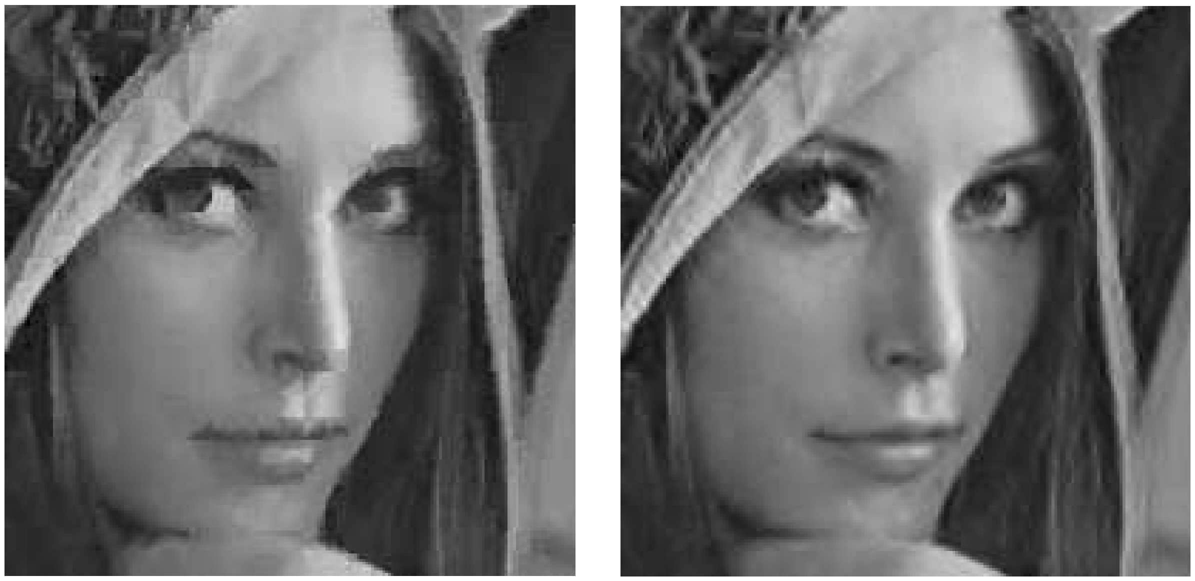
RMSE = 8.04, PSNR = 30.04, C_R=27:1.

Execution time $\approx$ 57 secs.

Figure 2.15: The fractal-wavelet representations of the test image obtained by simply encoding the image using the two FW schemes.

fractal-based schemes in the wavelet domain of the image, especially when the FW-II scheme is used.

- Also, the disturbing blockiness artifacts that are often present in the standard fractal-based schemes approximations are eliminated or significantly reduced when using a smooth wavelet basis with finite support, such as the Daubechies wavelets. As illustrated in Figure 2.16, the blockiness artifacts are no longer apparent in the zoomed fractal-wavelet image approximation. This is, in contrast to the standard fractal approximation of "Lenna" which suffers from apparent blockiness.



(a) Zooming in on the *standard fractal* image representation

(b) Zooming in on the *fractal-wavelet* image representation

Figure 2.16: Zooming in on the fractal and the fractal-wavelet representations of the "Lenna": note that the purely fractal representation suffers from blockiness artifacts while the fractal-wavelet representation shows no blockiness.

However, despite these advantages and others, these fractal-wavelet schemes are not without limitations. For instance, these fractal-wavelet schemes are still rather restrictive in the sense that are constrained by the the three-subband wavelet decomposition tree, when selecting the "parent" and "child" subtrees. This results in a static representation that varies significantly from one resolution level to the next. As illustrated in Figure 2.15, moving from the level $(k_1^*, k_2^*) = (4, 5)$ to the $(k_1^*, k_2^*) = (5, 6)$ level, results in a significant improvement of the fidelity of the approximation, accompanied by a drastic reduction in the compression ratio. Next, some of the recent progress and development in FW image coding is outlined.

2.3.3 Developing Adaptive FW Algorithms

As discussed in the previous section, the original FW schemes are non-adaptive in the sense that they are restricted by the structure of the wavelet decomposition tree in the selection of the child and parent subtrees. The FW resolution is defined by the selection of (k_1^*, k_2^*). As one moves from one resolution to the next, one notices a drastic change in the rate-distortion quality of the FW representation. However, this is not practical since FW representation with intermediate rate distortion quality can be obtained.

Next, an adaptive FW scheme which overcomes some of the limitations of the original FW scheme are presented.

FW Image Coding using Adaptive Partitioning

In many applications, one seeks to obtain the best approximation that yields a certain predetermined bit rate or compression ratio. The parameters involved may be set due to storage or transmission restrictions. Thus, it is often desirable to perform image compression with bit-rate or fidelity constraints.

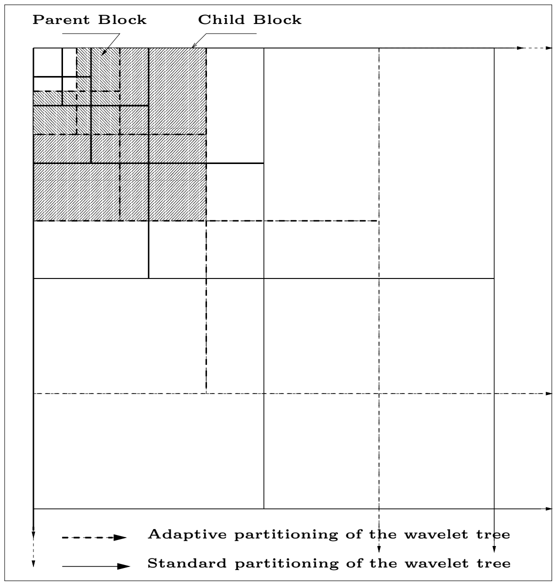

Parent Block
Child Block
Adaptive partitioning of the wavelet tree
Standard partitioning of the wavelet tree

Figure 2.17: Adaptive partitioning of the wavelet coefficients tree for the purpose of FW image coding.

In [33, 37, 38], it was proposed and implemented an algorithm that introduces the essential element of adaptivity to the original fractal-wavelet schemes to yield adaptive fractal wavelet algorithms capable of compressing a target image at a predetermined bit rate. The proposed scheme is less restrictive than the schemes described above, in the sense that the parent and child blocks do not have to be restricted in size or location to the various decomposition level or subbands of the spatial orientation tree, as illustrated in Figure 2.17. This scheme is outlined next.

An Adaptive FW Algorithm:

1. Start at the initial level ($l = l_0$) with corresponding FW resolution: $(k^*_{1,l_0}, k^*_{2,l_0})$:
2. Slide and expand the parent block $A^{\lambda}_{k^*_{1,l}}$ by one pixel, in the direction of $\lambda \in \{h, v, d\}$:
3. Expand each child block $A^{\lambda}_{k^*_{2,l+1}}$, in the direction of $\lambda \in \{h, v, d\}$, by 2 pixels:
4. This process is then extended to the higher levels to cover the entire wavelet decomposition tree.
5. Check if the stopping criterion, such as a prescribed bit rate or distortion, is achieved: if so stop, otherwise set $l = l + 1$ and go to step 2.

Clearly, one can apply the quadtree partitioning algorithm for the purpose of FW image coding using any of the FW schemes described in the previous section, however, the standard FW-II scheme will be mainly use. This scheme is generally preferred for its computational efficiency.

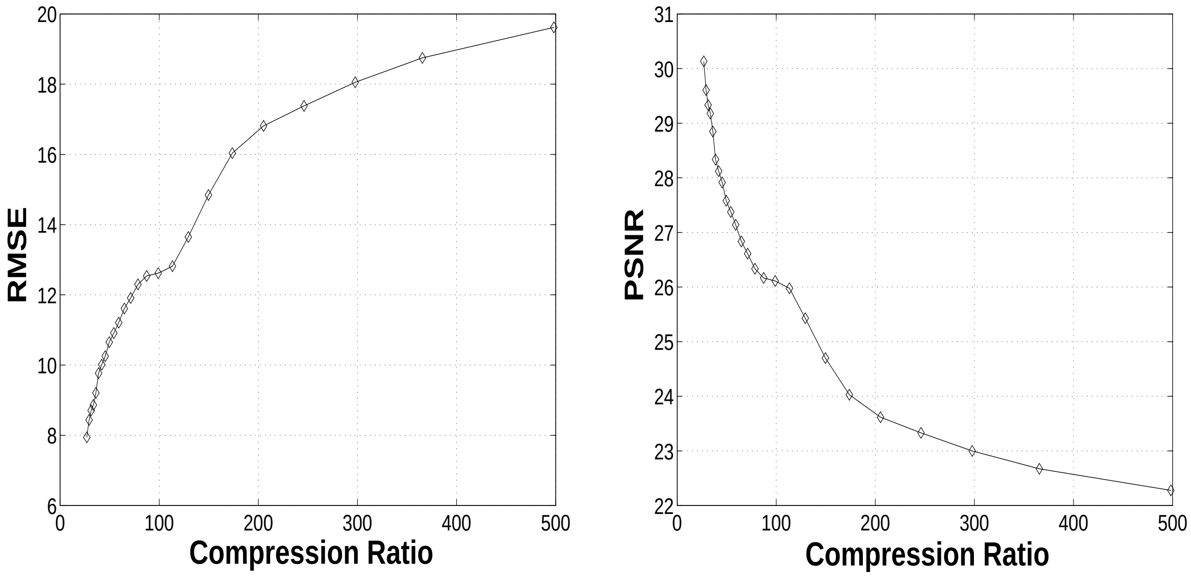


Figure 2.18: Rate distortion curves generated by the adaptive standard FW-II scheme, for the test image of "Lenna".

Some of the benefits of such an adaptive fractal-wavelet scheme include the ability to generate continuous and relatively smooth rate distortion curves for the fractal-wavelet schemes and encode images at a pre-defined bit rate or representation tolerance error. The resulting rate distortion curves for the adaptive fractal-wavelet schemes are illustrated in Figure 2.18. Note how the "block sliding" strategy performs an interpolation between the points corresponding to the $(k_1^*, k_2^*) = (3,4), (4,5)$ and $(5,6)$. The "bumpiness" in these rate-distortion curves may be explained as follows: Recall that this adaptive FW scheme attempts to interpolate between the three standard FW resolutions, $(k_1^*, k_2^*) = (3,4), (4,5)$ and $(5,6)$. The "bump" in the middle of the curves correspond to the middle resolution, $(k_1^*, k_2^*) = (4,5)$. Note that for intermediate resolutions, the partitioning of the wavelet tree does not follow the standard hierarchical quadtree partitioning and hence the blocks of child and parent subtrees contain a mixture of subtrees that originate from the horizontal, vertical and diagonal sub-bands. This is expected to yield FW representations that are somewhat distinct from those obtained using standard FW resolutions. Although this adaptive scheme interpolates reasonably well between the three resolutions, the resulting rate-distortion curves are not smooth at the standard resolutions, $(k_1^*, k_2^*) = (3,4), (4,5)$ and $(5,6)$.

Next, another adaptive FW scheme that is based on quadtree partitioning of the wavelet decomposition tree, will be discussed.

FW Image Coding using Quadtree Partitioning

In [35, 36], the use of the quadtree-partitioning approach for the purpose of partitioning the wavelet tree and performing fractal-wavelet image compression, was proposed. The hierarchical quadtree partitioning scheme of the wavelet domain stems for the spatial quadtree image partitioning algorithm. The main distinction is that while the quadtree image partitioning algorithm seeks to decompose a sub-block that does not satisfy a homogeneity criterion into four smaller quadrants, the hierarchical quadtree partitioning scheme performs similar operation on *subtrees* instead of sub-blocks.

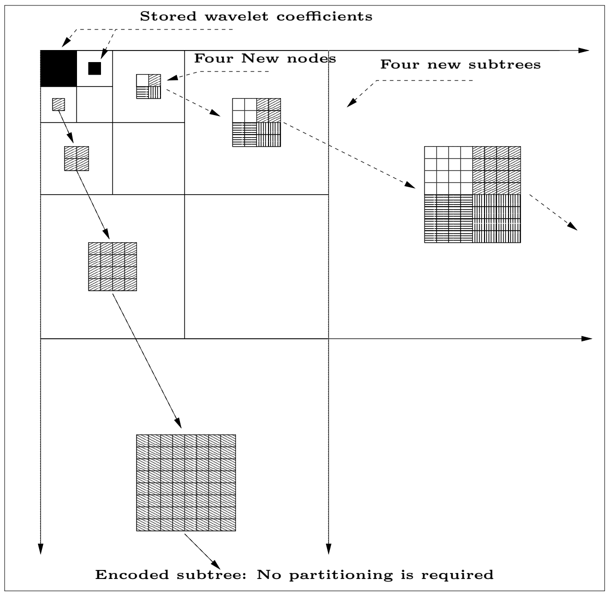

Stored wavelet coefficients
Four New nodes
Four new subtrees
Encoded subtree: No partitioning is required

Figure 2.19: Quadtree partitioning of the wavelet coefficients tree for the purpose of FW image coding.

The hierarchical quadtree partitioning scheme in the wavelet domain can be described as follows:

Consider a subtree of wavelet coefficients, A^{λ}_{kij}, that is rooted at the coefficient a^{λ}_{kij}, $\lambda \in \{h, v, d\}$. The fractal-wavelet based scheme examines such a tree and decides, on the bases of a prescribed criterion, whether or not such a tree should be encoded using a FW scheme. If it turns out that the tree contains "too much" information to be encoded properly at the given level, then the node or root, a^{λ}_{kij}, is stored and the subtree A^{λ}_{kij} is replaced by four subtrees that are rooted at the four children of the original node, a^{λ}_{kij}. This process is illustrated in Figure 2.19.

Various decomposition criteria for the hierarchical quadtree partitioning scheme have been investigated. In particular, the use of the collage error, the variance and the energy of a subtree as decomposition criteria for the quadtree partitioning scheme, was explored. It was found that the energy of the subtree is the best quadtree decomposition criterion [35, 36]. The quadtree-based FW scheme, using the energy of the subtree as the quadtree decomposition criterion, is summarized next.

A Quadtree-Based FW Algorithm:

For each *uncoded* child subtree, $A^{\lambda}_{kij} = \{y_1, y_2, \ldots, y_n\}$, rooted at a^{λ}_{kij}, compute its energy:

$$\mathcal{E}_{kij} = \frac{1}{n} \sum_{m} y_m^2. \tag{2.17}$$

- If $\mathcal{E}_{kij}$ is less than some prescribed threshold $T_{\mathcal{E}}$, then the subtree is encoded using the a FW scheme of choice, and marked the subtree as *fractally coded.*
- Otherwise, the node, a^{λ}_{kij}, of the current subtree is stored and the subtree is decomposed into four new subtrees as follows:
  - Store the node of the subtree and mark it as *stored*,
  - Replace the subtree by the four subtrees that are rooted at its four children, and mark each one of these new subtrees as *uncoded.*
- Continue until all the coefficients are either fractally coded or stored.

Clearly, one can apply the quadtree partitioning algorithm for the purpose of FW image coding using any of the FW schemes described in the previous section. However, it is advantageous to use the standard FW-II scheme due to its computational efficiency. The main advantage of using the energy of the subtree as the quadtree partitioning criterion is that one is performing fast FW image coding. This is the case since only those subtrees with high energy are encoded using the FW scheme. However, when using the collage error as a quadtree decomposition criterion, every

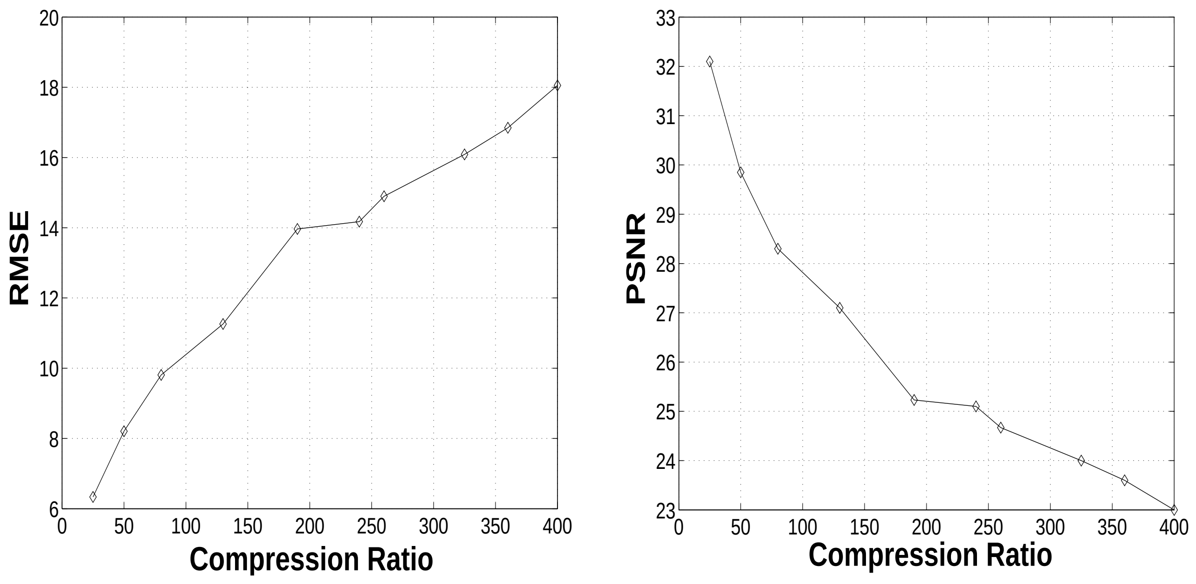


Figure 2.20: Rate distortion curves generated by the quadtree-based standard FW-II scheme, for the test image of "Lenna".

child subtree has to be encoded using the FW scheme before one can decide whether to subdivide it or not.

Some of the advantages of the quadtree-based FW scheme include: making the FW scheme more adaptive to the content of an image, performing FW compression at a pre-determined bit rate or fidelity precision and generating rate distortion curves for the FW coding schemes by varying the energy criterion threshold $\mathcal{E}_T$. Figure 2.20 illustrates the rate distortion curves for the quadtree-based standard FW scheme outlined above.

Similar to the quadtree-based fractal scheme in the pixel domain, the quadtree-based FW scheme yields rate distortion curves that are relatively "bumpy". This bumpiness can be explained as follows: These rate distortion curves were generated by varying the child-subtree energy threshold, $T_{\mathcal{E}}$. On the one hand, for larger values of $T_{\mathcal{E}}$, most child subtrees are encoded using the FW scheme. Thus decreasing this threshold is expected to result in many child subtrees being partitioned into four new subtrees and their roots stores. In turn this results is relatively significant changes in the rate-distortion quality of the FW representation. On the other hand, for smaller values of $T_{\mathcal{E}}$, most of the subtrees are partitioned and their roots stored. Thus decreasing this threshold even further results in decomposing some of the remaining subtrees and storing more wavelet coefficients. This in turn results in a relatively significant change in the rate-distortion quality of the FW representation.

For medium range threshold values, the rate-distortion quality of the FW representation appear to vary continuously with the variation of the child subtree energy threshold, $T_{\mathcal{E}}$. This explains the steepness of these curves for lower and higher compression ranges.

The performance of the adaptive image coding methods discussed in this chapter will be compared next.

2.4 Comparisons and Concluding Remarks

In this chapter, several fractal, wavelet and fractal-wavelet image coding methods for the purpose of image compression were discussed and implemented. Some of the advantages of developing adaptive fractal-based image compression methods include performing content-dependent image compression at at pre-determined bit rates, compression ratios or fidelity precisions and generating rate distortion curves. Generating rate distortion curves for these fractal-based schemes provided a comparison of their performance to each other as well to other image compression methods, such as the SPIHT method.

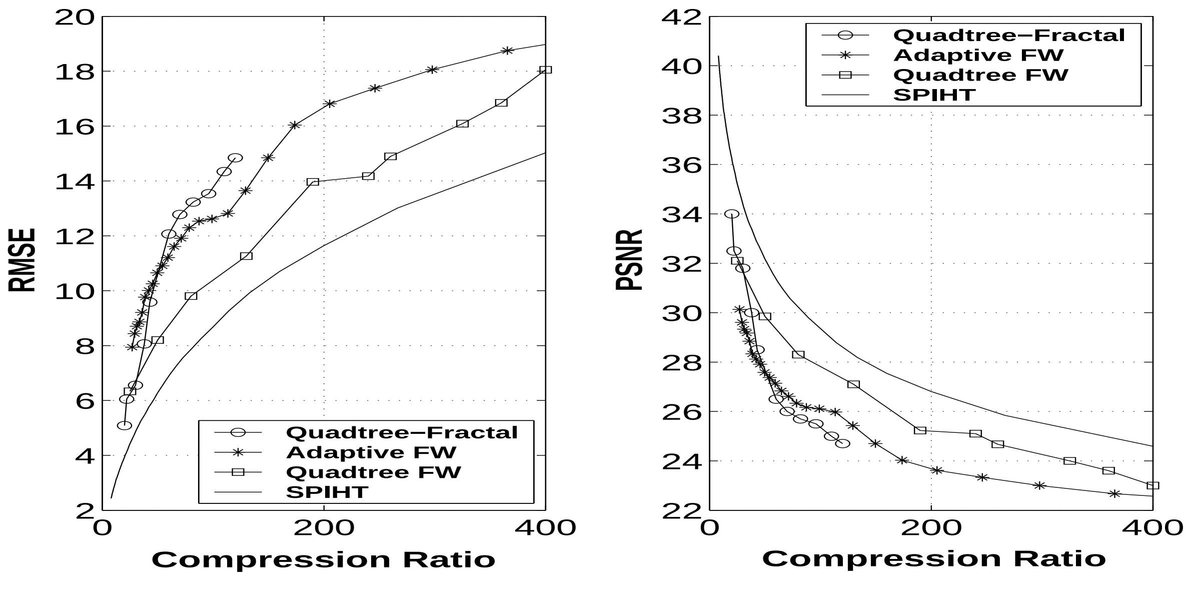


Figure 2.21: Rate distortion curves generated by the various adaptive image compression methods studied in this chapter, namely the quadtree-based standard fractal, the SPIHT, the adaptive FW and the quadtree-based FW schemes.

Figure 2.21 illustrates a comparison between the various adaptive fractal and wavelet-based image compression methods covered in this chapter, namely the quadtree-based standard fractal, the SPIHT, the adaptive FW and the quadtree-based FW schemes. Clearly, the SPIHT performs best. However, when comparing the various fractal-based methods to each other, note that fractal-wavelet based methods perform better than the standard fractal schemes, applied in the spatial domain of the image, for higher compression ratios. However, for lower compression ratios (i.e. less than 50:1), the quadtree-based standard fractal scheme starts to perform better than some of the FW methods.

In this chapter, a brief review of the theory and application of various adaptive fractal and wavelet based image compression methods was presented. Rate distortion curves of these adaptive image compression schemes were generated and their performance was compared. While the SPIHT method performs considerably better than the best fractal-based schemes, fractal-based schemes were shown to be competitive especially at low compression ratios. Algorithms for making fractal-based schemes adaptive were also discussed. The fractal-wavelet schemes perform better than standard fractal schemes, especially for high compression ratios. Furthermore, fractal-wavelet schemes overcome the computational complexity and the disturbing blockiness artifacts that are evident when using the generic spatial-based fractal schemes.

In the following chapters of this thesis, the application of these various fractal and fractal-wavelet based image coding schemes for the purpose of image restoration and enhancement will be investigated.

Chapter 3

Wavelet Image Denoising

As discussed in the last chapter, the wavelet transform has many unique features that has made it a popular method for the purpose of image processing and compression. The wavelet transform performs a high degree of decorrelation between neighboring pixels, and it provides a distinct localization of the image in the spatial as well as the frequency domain. This transform also provides an elegant subband framework in which both high and low frequency components of the image can be analyzed separately. Recently, various wavelet-based methods have been proposed for the purpose of image enhancement and restoration. Basic wavelet image restoration methods are based on thresholding in the sense that each wavelet coefficient of the image is compared to a given threshold; if the coefficient is smaller than the threshold, then it is set to zero, otherwise it is kept or slightly reduced in magnitude. The intuition behind such an approach follows from the fact that the wavelet transform is efficient at energy compaction, thus small wavelet coefficients are more likely due to noise, and large coefficients are generally due to important image features, such as edges. Most of the efforts in the literature have concentrated on developing threshold selection criteria. Originally, Donoho and Johnstone proposed the use of a *universal* threshold uniformly throughout the entire wavelet decomposition tree [20, 21]. Then the use of different thresholds for different subbands and levels of the wavelet tree was found to be more efficient [22, 23, 73]. Some methods of selecting thresholds that are adaptive to different spatial characteristics have recently been proposed and investigated [13, 14, 15, 16]. It was found that such adaptivity in the threshold selection tends to improve the wavelet thresholding performance because it accounts for additional local statistics of the image, such as smooth or edge regions. These observations are consistent with the nature of adaptive processes which account for the local statistics and characteristics of the image. In general, adaptive approaches have been found to be more effective than their global counterparts.

In this chapter, some of the basic wavelet thresholding methods for the purpose of image denoising will be briefly reviewed, implemented and compared. and compare their performance. The use of the idea of cycle spinning for the purpose of reducing the pseudo-Gibbs artifacts that are often evident in the denoised images will also be discussed. Furthermore, the use of a new context-based thresholding strategy that takes the value of the neighboring wavelet coefficients into consideration when thresholding a wavelet coefficient will be proposed and implemented. It will be shown that the use of this proposed adaptive, context-based hard and soft thresholding operators result in an improvement, as compared to the standard hard and soft thresholding operators widely used in the literature.

The layout of this chapter is as follows: In section 1, an brief description of the the wavelet thresholding process is given. Four different standard wavelet thresholding methods for image denoising are described and implemented in section 2. Section 3, contains the the use of the cycle spinning idea for the purpose of reducing some of the artifacts and enhancing the denoised estimates obtained by various wavelet thresholding methods. In section 4, a context-based thresholding strategy is proposed and compared to the conventional hard and soft thresholding operators widely used in the literature. This chapter is concluded in section 5 with a brief summary.

3.1 Wavelet Thresholding for Signal Denoising

In this section, the wavelet thresholding process is first outlined, then the thresholding operators are defined and the selection of the threshold is briefly discussed. This process is then implemented for the purpose of denoising four one-dimensional test signals.

3.1.1 The Wavelet Thresholding Process

Wavelet thresholding for image denoising attempts to remove the noise present in the signal while preserving most of the signal characteristics, regardless of its frequency content. It involves the following steps:

1. Acquire the noisy digital signal.

2. Compute a linear forward discrete wavelet transform of the noisy signal.

3. Perform a non-linear thresholding operation on the wavelet coefficients of the noisy signal.
4. Compute the linear inverse wavelet transform of the thresholded wavelet coefficients.

This simple four-step process is known as wavelet thresholding or shrinkage. A more precise mathematical formulation of the above wavelet denoising procedure is needed. However, first the necessary variables and terms are defined as follows:

- $\mathbf{x}$: the original noise-free digital one or two-dimensional signal which has M samples. In the one-dimensional case, it is denoted by $\mathbf{x} = [x_i], i = 1, 2, \ldots M$ and for the two-dimensional case, without loss of generality, the image is assumed to be square so $\mathbf{x} = [x_{ij}], i, j = 1, 2, \ldots, \sqrt{M}$. In most of the two-dimensional applications in this chapter, the original image is the widely used test image of "Lenna", which is an 8 bits/pixel, gray-scale 512×512 pixels image, so $M = 512^2$. In practice, the original signal is generally not known, only the distorted signal is available.

- $\mathbf{X}$=$\mathcal{DWT}(\mathbf{x})$: the discrete wavelet transform of original signal $\mathbf{x}$, which again depending on the dimension of $\mathbf{x}$ is a one or two-dimensional array of size M. As mentioned earlier, the orthogonal Daubechies wavelet "Db8", chosen for its desirable smoothness properties, will be used throughout this thesis.

- $\mathbf{w}$: an additive white Gaussian noise with zero mean and variance $\sigma_{\mathbf{w}}^2$, which is assumed to have the same size M but independent of the original signal $\mathbf{x}$, so $\mathbf{w}\sim N(0, \sigma_{\mathbf{w}}^2)$. For most of the two dimensional applications, unless stated otherwise, it will be assumed that $\sigma_{\mathbf{w}}$=25.

- $\mathbf{y}$=$\mathbf{x}$+$\mathbf{w}$: the noisy version of the original noise-free signal.

- $\mathbf{Y}$=$\mathcal{DWT}(\mathbf{y})$: the $\mathcal{DWT}$ of the noisy signal $\mathbf{y}$.

- $T(., \lambda)$: the thresholding transformation with threshold λ.

- $\hat{\mathbf{X}} = T(\mathbf{X}, \lambda)$: the thresholded wavelet coefficients obtained after applying the thresholding operator $T(., \lambda)$.

- $\hat{\mathbf{x}} = \mathcal{DWT}^{-1}(\hat{X})$: the denoised version of the noisy image $\mathbf{y}$, which represents an approximation of the original image $\mathbf{x}$, with mean squared error

$$\text{MSE} = E[||\mathbf{x} - \hat{\mathbf{x}}||^2]. \tag{3.1}$$

In view of the above notations, the wavelet denoising process can be summarized as follows:

$$\mathbf{x} \longrightarrow \mathbf{y} = \mathbf{x} + \mathbf{w} \longrightarrow \mathbf{Y} = \mathcal{DWT}(\mathbf{y}) \longrightarrow \hat{\mathbf{X}} = T(\mathbf{Y}, \lambda) \longrightarrow \hat{\mathbf{x}} = \mathcal{DWT}^{-1}(\hat{\mathbf{X}}). \tag{3.2}$$

A block diagram of this process is illustrated in Figure 3.1.

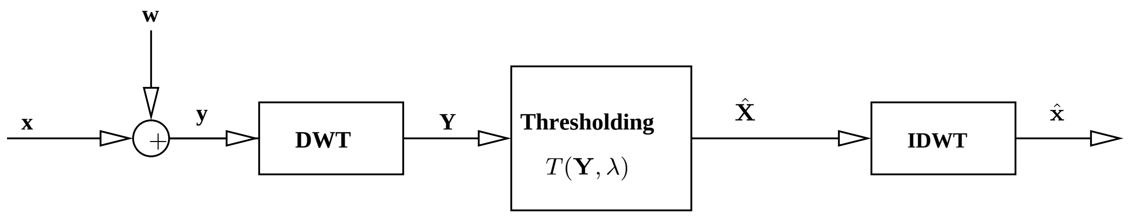


Figure 3.1: The three steps involved in the wavelet denoising process.

In summary, the wavelet denoising problem can be formulated as follows:

Design a thresholding transformation $T(., \lambda)$ with threshold λ such that:

$$\text{MSE} = E[||\mathbf{x} - \hat{\mathbf{x}}||^2] \quad \text{is minimized} \tag{3.3}$$

and the denoised image $\hat{\mathbf{x}}$ satisfies certain criteria, such as smoothness in low activity regions and sharpness of edges.

So far, the wavelet thresholding process is formulated, it remains to describe the two types of the threshold operator $T(., \lambda)$ associated with the threshold λ.

3.1.2 Thresholding Operators

Recall that $T(., \lambda)$ denotes the thresholding operator with corresponding threshold λ. More specifically, in this section the hard thresholding operator $T_h(., \lambda)$ and the soft thresholding operator $T_s(., \lambda)$ will be defined.

The *hard thresholding* operator is defined as:

$$\hat{\mathbf{X}} = T_h(\mathbf{Y}, \lambda) \text{ such that } \hat{x} = T_h(y, \lambda) = \begin{cases} y, & \text{if } |y| \geq \lambda, \\ 0, & \text{otherwise.} \end{cases} \tag{3.4}$$

The *soft thresholding* operator on the other hand is defined as:

$$\hat{\mathbf{X}} = T_s(\mathbf{Y}, \lambda) \text{ such that } \hat{x} = T_s(y, \lambda) = \begin{cases} y - \lambda, & \text{if } y \geq \lambda, \\ y + \lambda, & \text{if } y \leq -\lambda, \\ 0, & \text{otherwise.} \end{cases} \tag{3.5}$$

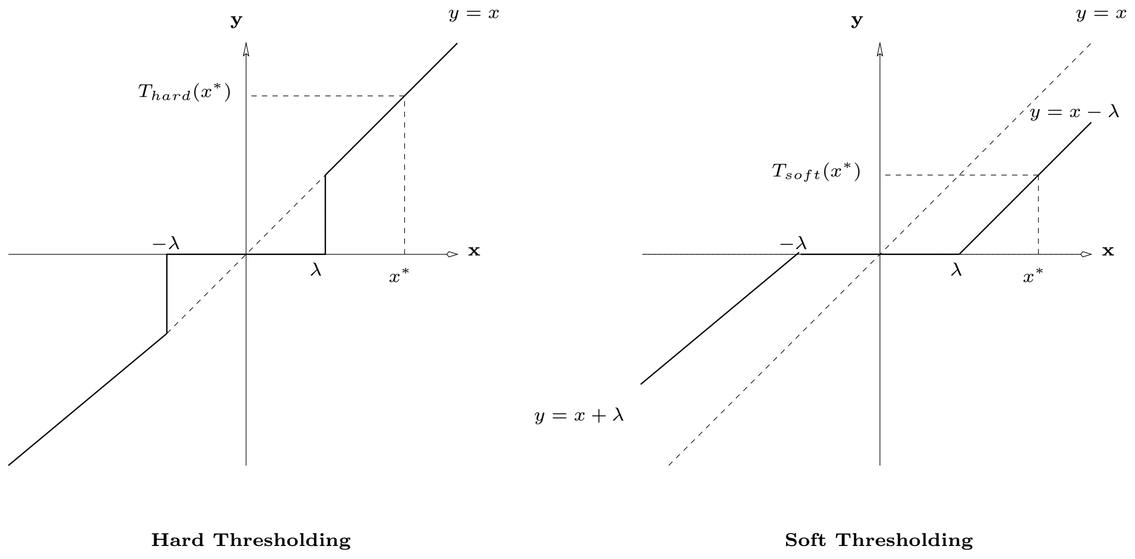


Figure 3.2: Hard and soft thresholding operators as applied on the wavelet coefficients.

The transfer functions of the hard and soft thresholding schemes are illustrated in Figure 3.2. Note that hard thresholding is a "keep or set to zero" procedure and is more intuitively appealing. On the other hand, soft thresholding shrinks coefficients above the threshold in absolute value. While at first sight hard thresholding may seem to be natural, the continuity of soft thresholding has some advantages. Sometimes, pure noise coefficients may pass the hard thresholding and appear as annoying "blips" in the output. However, soft thresholding shrinks these false structures.

Once the thresholding operator $T(., \lambda)$ has been defined, it remains to address the problem of selecting the corresponding threshold, λ.

3.1.3 Threshold Selection

As one may observe, threshold determination is an important question when applying the wavelet thresholding scheme. A small threshold may yield a result close to the input, but the result may be still be noisy. A large threshold on the other hand, produces a signal with a large number of zero coefficients. This leads to an overly smooth signal. Paying too much attention to smoothness generally suppresses the details and edges of the original signal and causes blurring and ringing artifacts.

The Universal Threshold

In this section, the most original threshold known as the *universal threshold* will be introduced and its performance will experimentally explored using a one-dimensional signal example.

Originally, Donoho and Johnstone proposed the use of the universal threshold [20]:

$$\lambda_{univ} = \sqrt{2\ln(M)} \times \sigma_{\mathbf{w}}, \tag{3.6}$$

where M is the signal size and $\sigma_{\mathbf{w}}^2$ is the noise variance. It has been shown that, when using the soft thresholding operator $T_s(.,\lambda)$, with $\lambda = \lambda_{univ}$, the following results hold[21, 22]:

- With high probability, which asymptotically tends to unity as the signal size M increases, the denoised signal $\hat{\mathbf{x}}$ is at least as smooth as the original noise-free signal $\mathbf{x}$, where smoothness is measured by any wide range of smoothness measures.

- Although the universal threshold λ_{univ} was derived for the purpose of soft thresholding, it can also be used for the purpose of hard thresholding. Hard thresholding using the universal thresholding, $T_h(.,\lambda_{univ})$ achieves better estimates in the MSE sense than $T_s(.,\lambda_{univ})$, but it does not guarantee the smoothness property of the denoised signal.

Next, the wavelet thresholding process using the universal threshold will be illustrated using a one-dimensional signal.

Application: A One-Dimensional Example

To investigate the effects of threshold selection, the wavelet thresholding process is applied to four one-dimensional signals commonly used in wavelet literature, namely “Blocks”, “Bumps”, “Doppler” and “HeavySine”. Each, one of these signals was corrupted by an AWGN noise with standard deviation $\sigma_{\mathbf{w}} = 1$. The setup is as follows:

- Apply the hard and soft thresholding operators, $T_h(.,\lambda)$ and $T_s(.,\lambda)$, respectively using the universal threshold. In this example, the original signals have length $M = 2048$ and the noise standard deviation $\sigma_{\mathbf{w}} = 1$ so the universal threshold is given by

$$\lambda_{univ} = \sqrt{2\ln(2048)} \times (1) = 3.905. \tag{3.7}$$

- Vary the threshold λ over the interval $[0, 5]$ with a step size of $\Delta = 0.1$. At each step, the hard and soft thresholding operators are applied to the four test signals using that threshold. The

RMSE fidelity measure of the quality of each denoised signal is then computed by comparing it to the original noise-free signal, which is assumed to be known.

- The above steps are repeated for different orthonormal wavelet bases, namely Haar and Daubechies 2, 4, and 8.

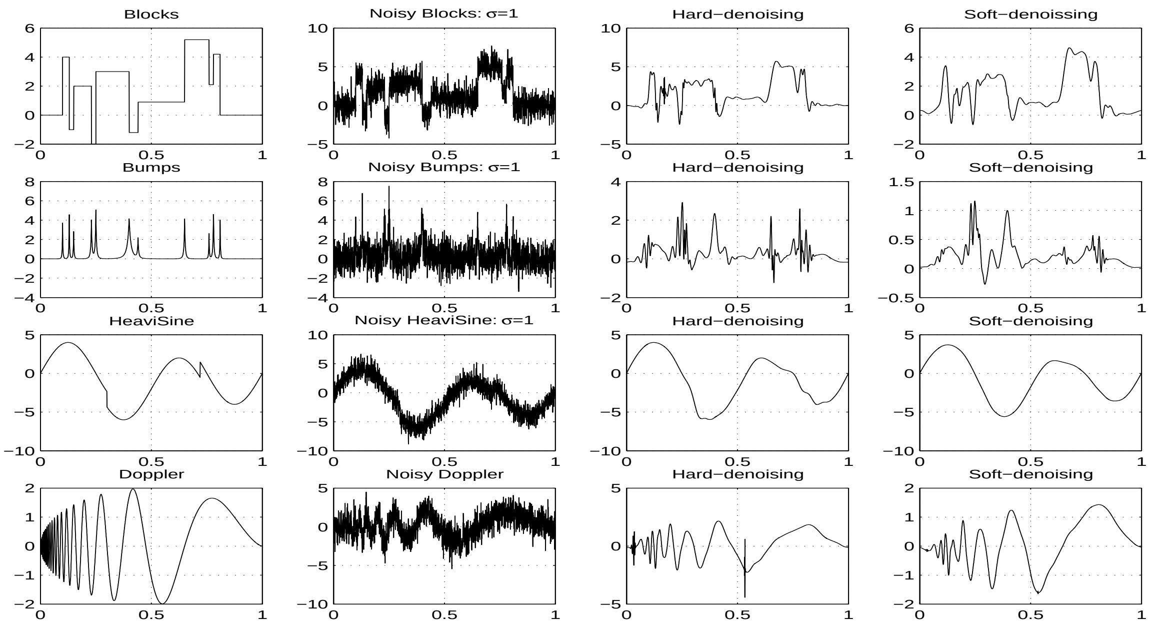
Figure 3.3 (a) illustrates the performance of the soft and hard thresholding operators using the universal threshold for the four test signals. The quality of the denoised signals as a function of the threshold, λ, level are illustrated in Figure 3.3 (b). These quality curves clearly indicate that the universal threshold is not optimal in the RMSE sense for the various signals. In general, the universal threshold tends to be conservatively high resulting in over-smoothing of the signal. This is the case because the derivation of this threshold gives higher priority to ensuring that the denoised estimate is at least as smooth as the original image than to minimizing the mean squared error [20, 21]. Often, this threshold is only useful as a starting value when nothing else is known about the signal characteristics, such as smoothness. One can then test better threshold values depending on the results obtained using the universal threshold. Table 3.1 illustrates the "optimal" thresholds

	Blocks		Bumps		HeavySine		Doppler	
	Hard	Soft	Hard	Soft	Hard	Soft	Hard	Soft
Haar	3.0	1.6	3.1	1.6	3.4	1.6	3.1	1.6
Db2	3.6	1.5	3.1	1.7	3.7	1.9	3.4	1.8
Db4	3.1	1.5	3.1	1.7	3.5	2.0	3.5	1.8
Db8	3.2	1.4	3.4	1.6	3.8	2.0	3.4	1.8

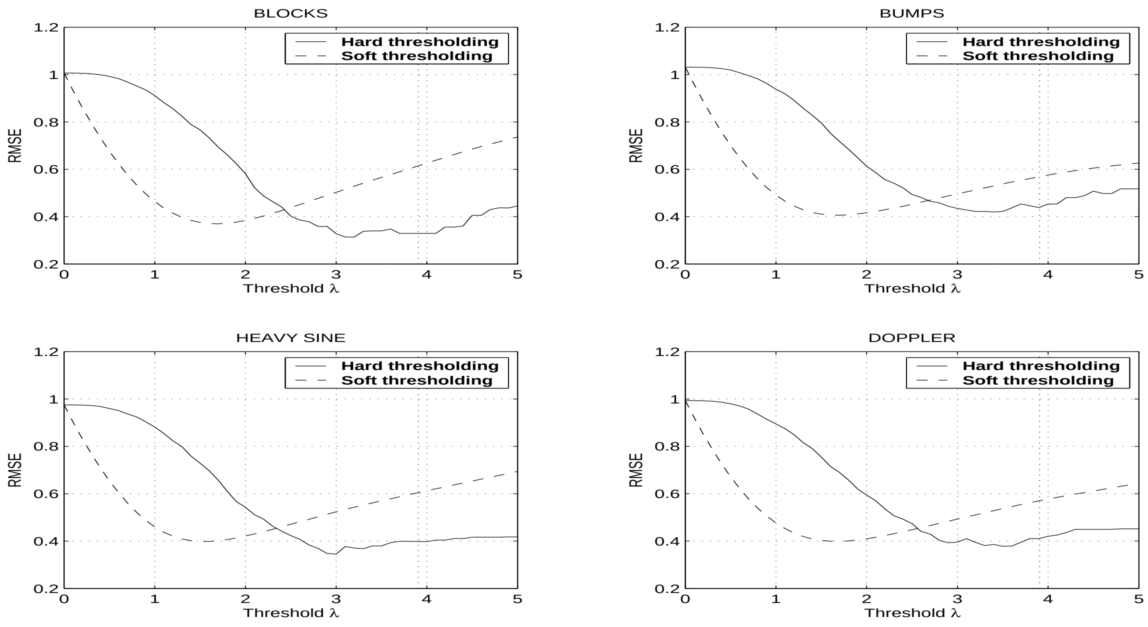
Table 3.1: The optimal thresholds for the four test signals using various wavelet bases. Note that the optimal thresholds are generally lower than the universal threshold $\lambda_{univ} = 3.095$, especially for the soft thresholding scheme. Note also that the optimal threshold for the soft thresholding operator is consistently about half of that corresponding to the hard thresholding operator.

for the various test signals using the different wavelet bases. Again, note that the "optimal" soft and hard thresholds are consistently lower than the universal threshold. It is also interesting to note that the "optimal" soft threshold is consistently about half of the "optimal" hard threshold for the various signals and wavelet bases.

So far, the experiments have been restricted to wavelet thresholding for the purpose of denoising one dimensional signals. Next, these experiments will extended to the two-dimensional case and



(a) Hard and soft thresholding for denoising four commonly used signals: “Blocks”, “Bumps”, “HeavySine” and “Doppler” signals.



(b) The dependence of the quality of the denoised signals on the selection of the threshold λ.

Figure 3.3: One-dimensional wavelet hard and soft thresholding of four noisy signals; “Blocks”, “Bumps”, “HeavySine” and “Doppler” signals, corrupted by an AWGN noise with $\sigma_{\mathbf{w}} = 1$. The “Db8” wavelet basis was used.

illustrate the application of wavelet thresholding for the purpose of image denoising.

3.2 Wavelet Thresholding Methods for Image Denoising

The image is assumed to be corrupted by a random AWGN noise with variance $\sigma_{\mathbf{w}}^2$ during its acquisition or transmission process. The original image of “Lenna” and its noisy version which is corrupted by an AWGN noise with noise standard deviation $\sigma_{\mathbf{w}} = 25$, as illustrated in Figure 1.2, will be used for the experimental results. The underlying concept of wavelet denoising of images is similar to the one-dimensional case. In this section, four standard wavelet thresholding methods will be briefly described, implemented and compared. These technique are VisuShrink, LevelShrink, SureShrink and BayesShrink which differ only in the selection of the threshold λ and the strategy employed in applying the thresholding operator $T(.,\lambda)$.

3.2.1 VisuShrink

The VisuShrink technique consists of applying the soft thresholding operator using the universal threshold:

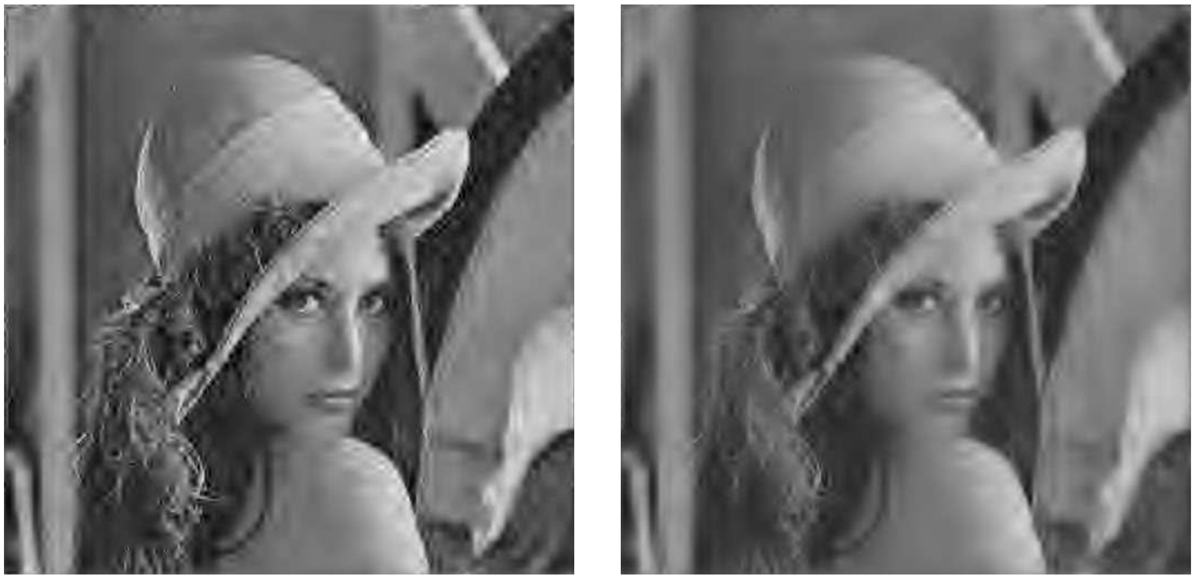
$$\lambda_{univ} = \sqrt{2\ln(M)}\sigma_{\mathbf{w}}, \tag{3.8}$$

as originally proposed by Donoho and Johnstone [20].

The VisuShrink algorithm was implemented for the purpose of restoring and enhancing the noisy image of “Lenna”. Figure 3.4 illustrates the results corresponding to the hard and soft thresholding methods using the universal threshold:

$$\lambda_{univ} = \sqrt{2\ln(512^2)} \times 25 = 124.88. \tag{3.9}$$

Note that VisuShrink is found to yield an overly smoothed estimate, especially in the case of the soft thresholding operator. This is because the universal threshold, λ_{univ}, tends to be too high for large values of M, setting to zero many signal coefficients along with the noise. This illustrates a common limitation of VisuShrink which has been widely reported in the literature [54, 20, 21, 22, 23]. The main feature of VisuShrink is that it guarantees a highly smoothed reconstruction of the noisy image but in doing so it often compromises many of the important features of the image (i.e. edges) by setting the threshold conservatively high. These limitations of VisuShrink are also due to the fact that it fails to adapt to the various types of statistical and structural properties of the wavelet tree. The universal threshold is applied uniformly throughout the wavelet tree. However, the use of different thresholds for different decomposition levels and subbands seems more reasonable.


(a) Hard VisuShrink thresholding
RMSE=12.37, PSNR=26.28.
Execution time ≈ 9 secs.

(b) Soft VisuShrink thresholding
RMSE=15.76, PSNR=24.18.
Execution time ≈ 11 secs.

Figure 3.4: Hard and soft threshold denoised estimates of "Lenna", using the VisuShrink thresholding method with the universal threshold: $\lambda_{universal} = \sqrt{2\ln(512^2)}25 = 124.88$.

Next, the optimality of the universal threshold is explored and it will be shown that the optimal thresholds for soft and hard thresholding, in terms of RMSE and PSNR quality measures, are indeed much lower than the universal threshold.

Exploring the Optimality of the Universal Threshold

In order to further explore the "optimality" of VisuShrink and its adopted universal threshold, the dependence of the quality of the denoised image, as measured by the RMSE and PSNR fidelity measures, on the value of the threshold level is studied. The noisy image of "Lenna", described above was used and the threshold was allowed to span a wide range of values which includes the universal threshold. Figure 3.5 illustrates the results obtained using the hard and soft thresholding methods. Observe that for the given test image, the "optimal" thresholds corresponding to the hard and soft thresholding algorithms are lower than the universal threshold adopted by VisuShrink, especially in the case of soft thresholding.

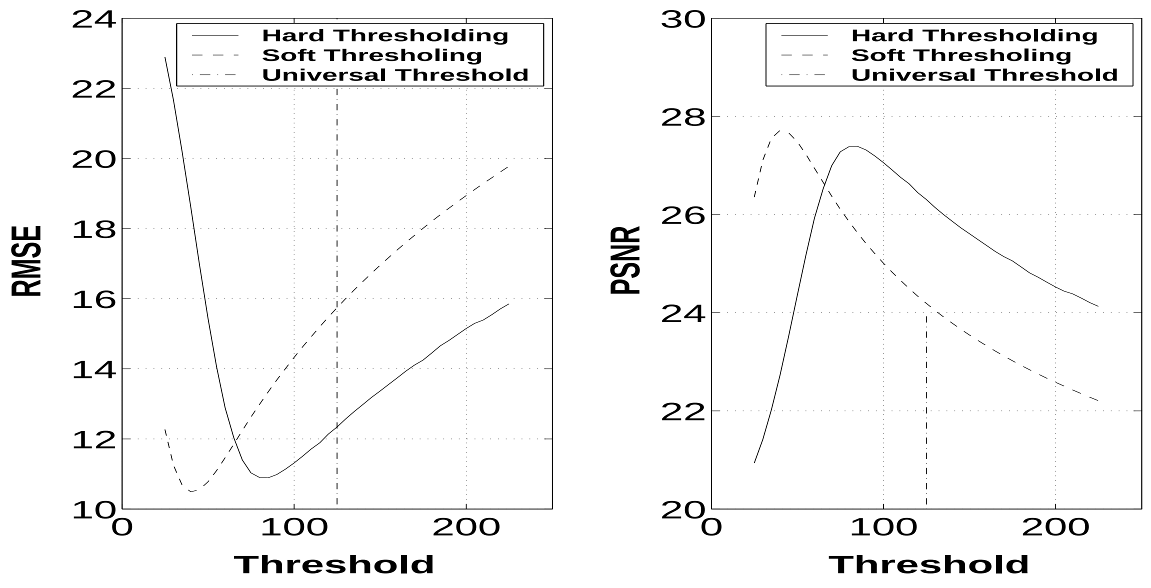


Figure 3.5: The dependence of the quality of the denoised image on the selection of the threshold for hard and soft thresholding, using the noisy image of "Lenna": The universal threshold $\lambda_{univ} = \sqrt{2\ln(512^2)}\times 25 = 124.88$ while the optimal thresholds are $\lambda_{hard} \approx 80$ for hard thresholding and $\lambda_{soft} \approx 40$ for soft thresholding.

Experimentally, it was found that, for the given test image of "Lenna", the optimal thresholds are $\lambda^*_{hard} \approx 80$, for hard thresholding and $\lambda^*_{soft} \approx 40$ for soft thresholding. It is interesting to note again that, similar to the one-dimensional case, the relationship

$$\lambda^*_{soft} \approx \frac{\lambda^*_{hard}}{2}. \tag{3.10}$$

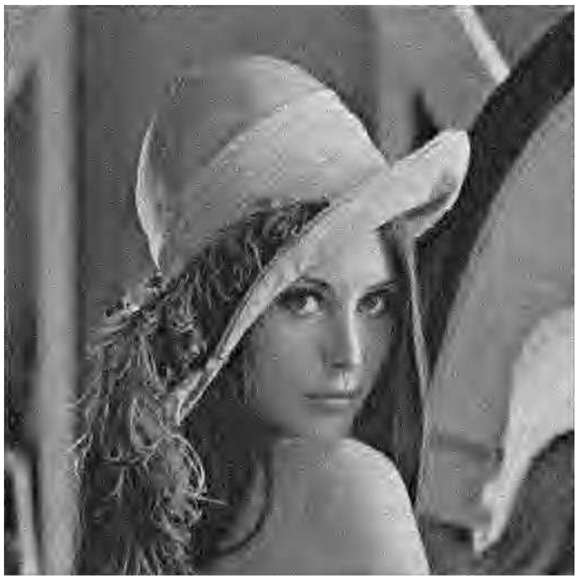
still holds. This relationship between the optimal values of λ^*_{hard} and λ^*_{soft} has been widely reported in the wavelet thresholding literature, although it has yet to be shown to hold analytically [54]. In fact, since many optimal threshold values were derived for the purpose of soft thresholding, it is a common practice to simply set the optimal hard threshold to be twice the optimal soft threshold.

The optimal values of soft and hard thresholds were used to denoise the test image and the results are illustrated in Figure 3.6. Note that the quality of these denoised estimates is better than the results obtained using the universal threshold, which were shown in Figure 3.4.

The limitations of the VisuShrink method can be attributed to the following two fundamental problems associated with this method:

- The universal threshold is conservatively too high resulting in noise-free estimates at the expense of over smoothing of the high frequency contents of the image. The VisuShrink estimates, especially when soft thresholding is used, often exhibit disturbing ringing and blurring artifacts.

- VisuShrink applies the universal threshold uniformly throughout the image without accounting for the local statistics of the various subbands and decomposition levels of the wavelet decomposition tree. Clearly, the use of different thresholds for different levels and subbands seems more reasonable, since it accounts for variation of the local statistics of the wavelet coefficients.


(a) Hard thresholding using: $\lambda^*_{hard} = 80$
RMSE=10.98, PSNR=27.31.
Execution time $\approx$ 8 secs.

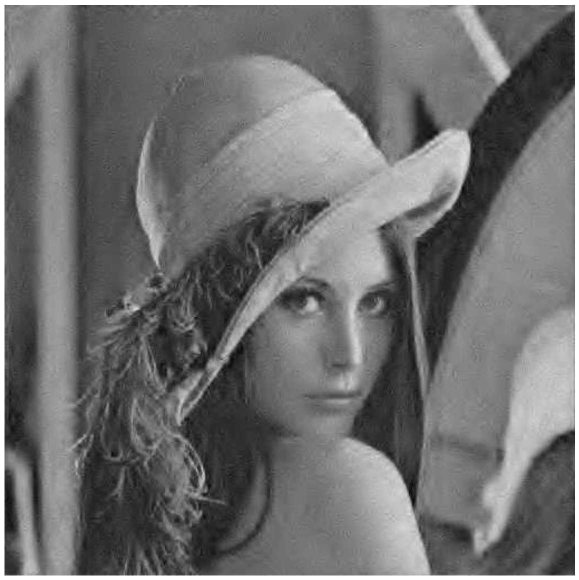

(b) Soft thresholding using: $\lambda^*_{soft} = 40$
RMSE=10.57, PSNR=27.65.
Execution time $\approx$ 10 secs.

Figure 3.6: Hard and soft thresholding denoised estimates of "Lenna" using the optimal thresholds: $\lambda^*_{hard} = 80$ and $\lambda^*_{soft} = 40$.

Next, a level-dependent wavelet thresholding method will be studied. This adaptive thresholding technique accounts for the variability within the wavelet tree structure by using different thresholds for different decomposition levels of the wavelet tree.

3.2.2 LevelShrink

As described in the previous section, VisuShrink adopts the universal threshold to be used uniformly throughout the wavelet decomposition tree of the noisy image. Intuitively, due to the high variability of the wavelet coefficients across different subbands and decomposition levels, it would be more reasonable, and perhaps more efficient, to use different thresholds for different subbands and levels of the wavelet tree. Recently, various methods for selecting thresholds that are adaptive to different spatial and statistical characteristics of the wavelet tree have been investigated [73, 21, 22, 13, 14, 15, 16]. It was found that such adaptivity in the threshold selection tends to improve the wavelet thresholding performance because it accounts for additional local statistics of the image, such as smooth or edge regions. These observations are consistent with the nature of adaptive processes which account for the local statistics and characteristics of the signal. In general, adaptive approaches have shown to be more effective than their global counterparts. In this section, one such simple level-dependent wavelet thresholding technique will be studied.

The level-dependent thresholding algorithm, called LevelShrink, proposes the use of different thresholds for different levels of the wavelet tree. Recall that the horizontal, vertical and diagonal subbands of the wavelet decomposition of an image illustrate distinct but complementary features of the image. In particular, for each decomposition level $j = 1, 2, \ldots, J$, the diagonal subband, HH_j, gives the diagonal details of the image, the horizontal subband, HL_j, gives the horizontal features while the vertical subband LH_j represents the vertical structures. Since the content of the various subbands varies from one level to the next, the use of level-dependent thresholds seems more reasonable than the use of a uniform threshold.

In [73], a level-dependent thresholding algorithm which adopts different thresholds for different levels of the wavelet tree has been proposed to improve the performance of the original wavelet thresholding method, VisuShrink. Instead of using a uniform threshold λ_{univ} throughout the wavelet tree, the level-dependent thresholding method uses different thresholds for different decomposition levels. One particular level-dependent thresholding scheme, called LevelShrink, is to set the threshold at the j^{th} decomposition level of the wavelet tree as follows [73]:

$$\lambda_j = \sqrt{2\ln(M)} \times \sigma_{\mathbf{w}} \times 2^{-(J-j)/2} = \lambda_{univ} \times 2^{-(J-j)/2}, \text{ for } j = 1, 2, \ldots J, \tag{3.11}$$

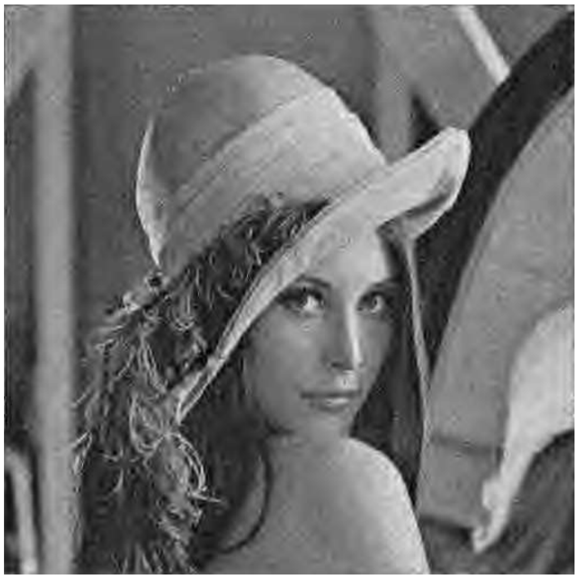
where J is the total number of decomposition levels and j is the scale level where the wavelet coefficient to be thresholded is located.

As illustrated in Table 3.2, this scheme uses larger threshold values for the finer scales decomposition tree and smaller thresholds for the more coarse scales of the wavelet tree. Note that for the highest level, the universal threshold is used. However, for the lower levels, the threshold is gradually scaled down.

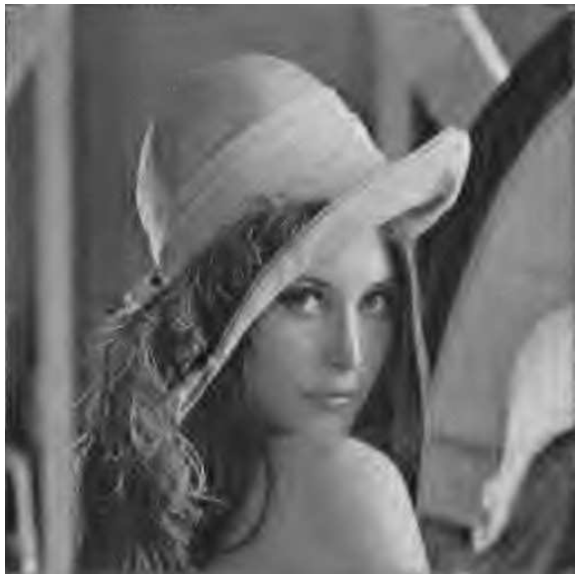
Figure 3.7 illustrates the results obtained by using the hard and soft LevelShrink thresholding the noisy test image of "Lenna". Note that this thresholding scheme yields results that are better than the results obtained by VisuShrink, especially when the hard thresholding scheme is used.

	finer $\longleftarrow$ Wavelet Decomposition Level$\longrightarrow$ coarse								
Level	9	8	7	6	5	4	3	2	1
Threshold	124.88	88.30	62.44	44.15	31.22	22.07	15.61	11.03	7.80

Table 3.2: The optimal thresholds for the various wavelet decomposition levels used by the LevelShrink thresholding scheme.



Hard LevelShrink thresholding
RMSE=10.01, PSNR=28.11.
Execution time $\approx$ 13 secs.



Soft LevelShrink thresholding
RMSE=11.30, PSNR=27.07.
Execution time $\approx$ 15 secs.

Figure 3.7: The denoised estimates obtained by the hard and soft LevelShrink thresholding scheme.

Clearly, the LevelShrink thresholding method is more adaptive than VisuShrink since it adapts to the variability registered within the wavelet tree from one decomposition level to the next by using different thresholds for different levels. However, this level-dependent thresholding method does not account for the inherent variability from one subband to another at the same decomposition level of the wavelet tree. In fact the same threshold is used for the horizontal, vertical and diagonal subbands of the same decomposition level. In reality, as mentioned earlier, these three subbands generally contain different types of details of the image and they should be treated and thresholded differently.

Next, a more adaptive thresholding technique, that adopts different thresholds that vary not only from level to level but also from one subband to another, will be studied.

3.2.3 SureShrink

Donoho and Johnstone developed an adaptive method of selecting a threshold that minimizes the Stein Unbiased Risk Estimator (SURE), which has been known as the SureShrink wavelet thresholding technique [22, 23, 54]. The adaptivity of SureShrink is achieved by choosing distinct thresholds for each subband of each level of the wavelet tree using an efficient recursive process. This thresholding scheme attempts to select thresholds that adapt to the data as well as minimize an estimation of the mean squared error or risk.

Threshold Determination

Let $\mathbf{X}_j^{sub}$ and $\mathbf{Y}_j^{sub}$, represent the wavelet coefficients corresponding to the original noise-free and noisy images, respectively, of size M_j and located in subband $sub \in \{horizontal, vertical, diagonal\}$ and decomposition level $j \in \{1, 2, \ldots, J\}$. To study the impact of the choice of the threshold on the risk, let $R(\mathbf{X}_j^{sub}, \lambda)$ denote the risk of a soft thresholding operator, $T_s(., \lambda)$, calculated with a threshold λ; in other words:

$$R(\mathbf{X}, \lambda) = E[||\mathbf{X}_j^{sub} - T_s(\mathbf{Y}_j^{sub}, \lambda)||^2] \tag{3.12}$$

Since, the original image is generally not known, then $\mathbf{X}_j^{sub}$ is not known and, an estimate $\hat{R}(\mathbf{X}_j^{sub}, \lambda)$ of $R(\mathbf{X}_j^{sub}, \lambda)$ is calculated from $\mathbf{Y}_j^{sub}$ and the best threshold level $\lambda_j^{(sub)*}$ can be estimated by minimizing $\hat{R}(\mathbf{X}_j^{sub}, \lambda)$.

To estimate the risk $R(\mathbf{X}_j^{sub}, \lambda)$, recall that:

$$\hat{\mathbf{X}}_j^{sub} = T_s(\mathbf{Y}_j^{sub}, \lambda), \text{ where } T_s(Y_{j,m}^{sub}, \lambda) = \begin{cases} Y_{j,m}^{sub} - \lambda, & \text{if } Y_{j,m}^{sub} \geq \lambda, \\ Y_{j,m}^{sub} + \lambda, & \text{if } Y_{j,m}^{sub} \leq -\lambda, \\ 0, & \text{otherwise.} \end{cases} \tag{3.13}$$

There are two cases:

1. If $|Y_{j,m}^{sub}| < \lambda$, then the soft thresholding operator sets this coefficient to zero, which produces a risk equal to $|X_{j,m}^{sub}|^2$. Since

$$E[|Y_{j,m}^{sub}|^2] = |X_{j,m}^{sub}|^2 + \sigma_{\mathbf{w}}^2, \tag{3.14}$$

one can estimate

$$|X_{j,m}^{sub}|^2 \approx |Y_{j,m}^{sub}|^2 - \sigma_{\mathbf{w}}^2. \tag{3.15}$$

2. If $|Y_{j,m}^{sub}| \geq \lambda$, the soft thresholding subtracts λ from the amplitude $|Y_{j,m}^{sub}|$. The expected risk is the sum of the noise energy plus the bias introduced by the reduction of the amplitude of $Y_{j,m}^{sub}$ by λ. Thus, the expected risk associated with a wavelet coefficient $|Y_{j,m}^{sub}| < \lambda$ can be estimated by $\sigma^2 + \lambda^2$.

The resulting total risk estimator of $R(\mathbf{X}_j^{sub}, \lambda)$ is

$$\hat{R}(\mathbf{X}_j^{sub}, \lambda) = \sum_{m=1}^{M_j} \Phi(|Y_{j,m}^{sub}|^2) \tag{3.16}$$

where

$$\Phi(|Y_{j,m}^{sub}|^2) = \begin{cases} |Y_{j,m}^{sub}|^2 - \lambda^2, & \text{if } |Y_{j,m}^{sub}| \leq \lambda, \\ \sigma_{\mathbf{w}}^2 + \lambda^2, & \text{if } |Y_{j,m}^{sub}| > \lambda. \end{cases} \tag{3.17}$$

It has been shown that $\hat{R}(\mathbf{X}_j^{sub}, \lambda)$ is an unbiased estimator of $R(\mathbf{X}_j^{sub}, \lambda)$ [54]. This unbiased risk estimator, $\hat{R}(\mathbf{X}_j^{sub}, \lambda)$, is known as the Stein's Unbiased Risk Estimator (SURE).

To find the optimal threshold level λ_j^{sub*} that minimizes the SURE estimator $\hat{R}(\mathbf{X}_j^{sub}, \lambda)$:

1. First, the wavelet coefficients $Y_{j,m}^{sub}$, within each vertical, horizontal and diagonal subband (sub) and each level, $j = 1, 2, \ldots, J$, are sorted in decreasing magnitude order.

2. Now, let $Y_j^{sub(r)}(k),\ k = 1, 2, \ldots, M_j$ be the ordered coefficient of order r, where M_j is the number of coefficients in a subband at decomposition level j.

3. It was shown that the risk estimator is given by [54]:

$$\hat{R}(\mathbf{X}_j^{sub}, \lambda) = \sum_{k=l}^{M_j} |Y_j^{sub(r)}(k)|^2 - (M - l)\sigma^2 + l(\sigma^2 + \lambda^2), \tag{3.18}$$

where l is an index such that:

$$|Y_j^{sub(r)}(l)| \leq \lambda < |Y_j^{sub(r)}(l + 1)|. \tag{3.19}$$

4. Since $\hat{R}(\mathbf{X}_j^{sub}, \lambda)$ is an increasing function of λ, then one must choose

$$\lambda_j^{sub*} = |Y_j^{sub(r)}(l)| \tag{3.20}$$

to minimize $\hat{R}(\mathbf{X}_j^{sub}, \lambda)$.

To find the optimal threshold level λ^*_{sure} that minimizes $\hat{R}(\mathbf{X}_j^{sub}, \lambda)$ it is therefore sufficient to compare the M_j possible values $\{|Y_j^{sub(r)}(l)|\}$, $l = 1, 2, \ldots, M_j$, and choose the value of λ that yields the smallest risk estimator $\hat{R}(\mathbf{X}_j^{sub}, \lambda)$.

Although the SureShrink thresholding method clearly provides an adaptive thresholding strategy, its performance is dependent on estimating the statistics of the wavelet coefficients of the original image from the statistics in the wavelet transform of the noisy image. For instance, the estimation

$$|X_{j,m}^{sub}|^2 \approx |Y_{j,m}^{sub}|^2 - \sigma_{\mathbf{w}}^2. \tag{3.21}$$

may yield a negative estimate of $|X_{j,m}^{sub}|^2$ when $|Y_{j,m}^{sub}|^2 < \sigma_{\mathbf{w}}^2$ which is mathematically inconsistent. This may occur in situations where the wavelet coefficients are sparse. Next, a hybrid approach adopted by the SURE algorithm, which deals with these outlying cases, is described.

Threshold Selection in Sparse Cases

The SURE principle has a drawback in situations of extreme sparsity of the wavelet coefficients. In such cases the noise contributes to the SURE profile through the many coordinates at which the signal is zero or close to zero. The wavelet coefficients of the noisy image at these locations, which correspond mainly to noise, will swamp the information contributed to the SURE profile by the few coordinates where the signal is nonzero. To overcome the limitations in these outlying cases, SureShrink uses a hybrid approach where the threshold is chosen to be λ_j^{sub*} in high activity subbands and the localized universal threshold in sparse subbands.

Although the estimator $\hat{R}(\mathbf{X}_j^{sub}, \lambda)$ of $R(\mathbf{X}_j^{sub}, \lambda)$ is unbiased, in sparse regions of the wavelet decomposition tree, its variance may induce errors leading to an optimal threshold λ_j^{sub*} that is too small. This happens if the signal energy is small relative to the noise energy, that is

$$||\mathbf{X}_j^{sub}||^2 << E[||\mathbf{w}||^2] = M_j \sigma_{\mathbf{w}}^2. \tag{3.22}$$

In this case one must impose another thresholding method, such as the localized universal threshold

$$\lambda_{univ} = \sqrt{2\ln(M_j)} \times \sigma_{\mathbf{w}}, \tag{3.23}$$

computed for the current decomposition level j, in order to remove most of the noise.

However, the original image $\mathbf{x}$ is generally unknown, hence its wavelet transform $\mathbf{X}_j^{sub}$ is also unknown. But, since

$$E[||\mathbf{Y}_j^{sub}||^2] \approx ||\mathbf{X}_j^{sub}||^2 + M_j \sigma_{\mathbf{w}}^2, \tag{3.24}$$

one can estimate

$$||\mathbf{X}_j^{sub}||^2 \approx ||\mathbf{Y}_j^{sub}||^2 - M_j\sigma_w^2 \tag{3.25}$$

This estimate of the local signal energy is compared to the minimum energy level, as defined in [54]

$$\epsilon_{M_j} = \sigma_{\mathbf{w}}^2\sqrt{M_j}[\ln(M_j)]^{\frac{3}{2}}. \tag{3.26}$$

The resulting hybrid threshold adopted by the SureShrink scheme is given by:

$$\lambda_{j,sure}^{sub*} = \begin{cases} \sqrt{2\ln(M_j)}\sigma_w, & \text{if } ||\mathbf{Y}_j^{sub}||^2 - M_j\sigma_w^2 \leq \epsilon_{M_j}, \\ \lambda_j^{sub*}, & \text{if } ||\mathbf{Y}_j^{sub}||^2 - M_j\sigma^2 > \epsilon_{M_j}. \end{cases} \tag{3.27}$$

Next, an implementation of the SureShrink wavelet thresholding scheme for the purpose of denoising images is illustrated.

Experimental Results

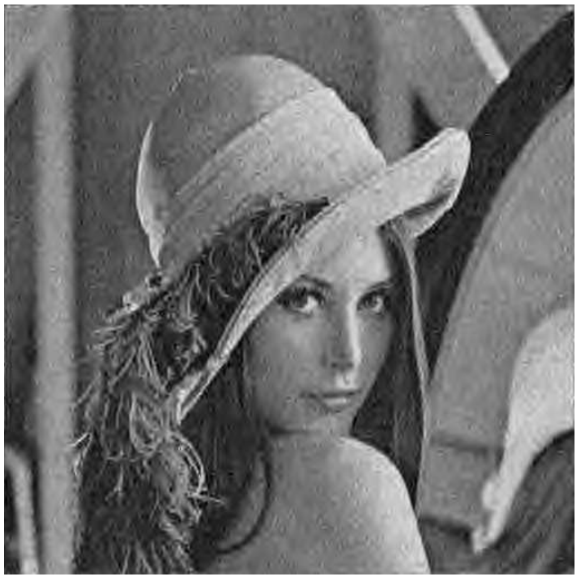
The SureShrink thresholding method was implemented for the purpose of restoring and enhancing the noisy test image of "Lenna". Table 3.3 illustrates the optimal thresholds obtained by using the SureShrink technique. Note that the optimal thresholds vary not only from one level to the next but also from subband to another. This represents an improvement as compared to the LevelShrink thresholding approach where the optimal threshold is the same for the three subbands at each wavelet decomposition level. Again, note that similar to the LevelShrink thresholding strategy, the SureShrink uses larger threshold values at finer scale levels of the wavelet tree and smaller thresholds for more coarse scales. These results illustrate how the universal threshold adopted by the VisuShrink method is far from optimal for most of the decomposition subbands and the decomposition levels, especially the lower ones. The optimal threshold adopted by the SureShrink method is derived for the purpose of soft thresholding. As mentioned earlier, the optimal hard threshold is set to be twice that used by the soft threshold.

Figure 3.8 illustrates the results of denoising the test image using the SureShrink technique, applied on the first five decomposition levels of the wavelet tree. Note that this scheme yields better results as compared to the LevelShrink thresholding technique presented in the previous section. However, this improvement is gained at the expense of an increase in computational complexity. This is evident when comparing the execution times of the hard and soft BayesShrink thresholding schemes to those of the VisuShrink and LevelShrink methods.

	Wavelet Decomposition Level				
Level	9	8	7	6	5
Horizontal	117.74	40.29	23.90	8.95	4.14
Vertical	117.74	30.57	15.21	7.96	3.01
Diagonal	117.74	110.14	28.70	12.28	3.97

Table 3.3: The optimal thresholds for the various wavelet decomposition levels and subbands obtained by the SureShrink soft thresholding scheme, for the first five wavelet decomposition levels.

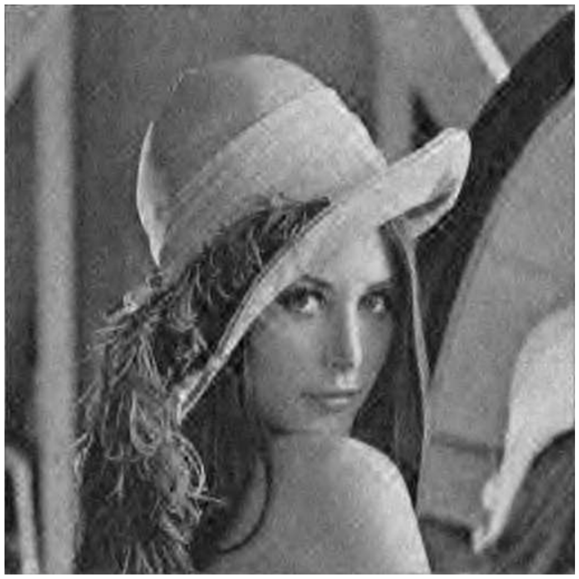
The wavelet thresholding methods discussed so far assume no knowledge of the original signal or its probability distribution. These methods are based on simple and general assumptions, such as the signal is assumed to belong to a wide class of piecewise regular practical signals and real-world images and that the noise is independent of the signal.



Hard SureShrink thresholding

RMSE=9.89, PSNR=28.23.

Execution time $\approx$ 973 secs.



Soft SureShrink thresholding

RMSE=9.96, PSNR=28.16.

Execution time $\approx$ 989 secs.

Figure 3.8: Hard and soft thresholding denoised estimates of “Lenna” using the SureShrink thresholding method applied on the first five decomposition levels of the wavelet tree. Note that the threshold for hard thresholding is twice that of the optimal Sure threshold derived for soft thresholding.

Next, BayesShrink, which is an adaptive wavelet thresholding method that is based on Bayes theory, is described and implemented.

3.2.4 BayesShrink

The BayesShrink wavelet thresholding method [13, 14, 15, 16] adopts Bayesian approach which assumes the knowledge of the probability distribution of the original signal and seeks to optimize the threshold operator $T(., \lambda)$ for the purpose of minimizing the expected risk. In particular, it is assumed that, for the various subbands and decomposition levels, the wavelet coefficients of the original image follow approximately a *Generalized Gaussian Distribution(GGD).* In particular, the wavelet coefficients, $\mathbf{X}_j^{sub}$, of size M_j and located in subband $sub \in \{horizontal, vertical, diagonal\}$ and at decomposition level $j \in \{1, 2, \ldots, J\}$ can be modeled by a Generalized Gaussian Distribution(GGD), which is given by , which is given by

$$GG_{\sigma_{\mathbf{X}_j^{sub}},\beta}(x) = C(\sigma_{\mathbf{X}_j^{sub}}, \beta)e^{-[\alpha(\sigma_{\mathbf{X}_j^{sub}},\beta)|x|]^2}, \text{ for } \infty < x < \infty \text{ and } \beta > 0 \tag{3.28}$$

where

$$\alpha(\sigma_{\mathbf{X}_j^{sub}}, \beta) = \frac{\sqrt{\frac{\Gamma(3/\beta)}{\Gamma(1/\beta)}}}{\sigma_{\mathbf{X}_j^{sub}}}, \text{ and } C(\sigma_{\mathbf{X}_j^{sub}}, \beta) = \frac{\beta\alpha(\sigma_{\mathbf{X}_j^{sub}}, \beta)}{2\Gamma(1/\beta)}. \tag{3.29}$$

The parameter $\sigma_{\mathbf{X}_j^{sub}}$ is the standard deviation and β is the shape parameter. It has been observed [13, 14, 15, 16] that, using a shape parameter β ranging from 0.5 to 1, the distribution of the wavelet coefficients in a subband can be described for a large set of natural images.

Recall that in the wavelet domain, the following relationship holds:

$$\mathbf{Y}_j^{sub} = \mathbf{X}_j^{sub} + \mathbf{W}_j^{sub} \tag{3.30}$$

Due to the independence assumption between the original signal $\mathbf{x}$ and the noise $\mathbf{w}$, the joint distribution of $\mathbf{X}_j^{sub}$ and $\mathbf{W}_j^{sub}$ is the product of the distribution of $\mathbf{X}_j^{sub}$ and $\mathbf{W}_j^{sub}$. The conditional probability distribution of $\mathbf{X}_j^{sub}$, given the observed noisy wavelet coefficients $\mathbf{Y}_j^{sub}$, is called the posterior distribution. This posterior distribution can be used to construct a decision soft thresholding operator $T_s(., \lambda)$ that computes a denoised estimate $\hat{\mathbf{X}}_j^{sub} = T(\mathbf{X}_j^{sub}, \lambda)$ of $\mathbf{X}_j^{sub}$ from the noisy data $\mathbf{Y}_j^{sub}$ by minimizing the Bayes risk.

More specifically, assuming that the noiseless wavelet coefficients, $\mathbf{X}_j^{sub}$ located in subband $sub \in \{h, v, d\}$ and decomposition level $j = 1, 2, \ldots, J$, follows the GGD distribution. Then, β and $\sigma_{\mathbf{X}_j^{sub}}$ are empirically estimated for each subband and try to find the optimal threshold λ^*_{Bayes}

which minimizes the Bayesian risk function, defined as the expected value of the mean square error:

$$\begin{aligned} R(\mathbf{X}_j^{sub}, \lambda) &= E[||\mathbf{X}_j^{sub} - \hat{\mathbf{X}}_j^{sub}||^2] & (3.31) \\ &= E_{\mathbf{X}_j^{sub}}[E_{\mathbf{Y}_j^{sub}}|\mathbf{X}_j^{sub}[(\mathbf{X}_j^{sub} - \hat{\mathbf{X}}_j^{sub})^2]], & (3.32) \end{aligned}$$

where

$$\begin{aligned} \hat{\mathbf{X}}_j^{sub} &= T_s(\mathbf{Y}_j^{sub}, \lambda), & (3.33) \\ \mathbf{Y}_j^{sub}|\mathbf{X}_j^{sub} &\sim N(x, \sigma_{\mathbf{w}}^2) \text{ and} & (3.34) \\ \mathbf{X}_j^{sub} &\sim GGD_{\sigma_{\mathbf{X}_j^{sub}}, \beta}. & (3.35) \end{aligned}$$

For each decomposition level $j \in \{1, 2, \ldots, J\}$ and subband $sub \in \{horizontal, vertical, diagonal\}$, the the optimal threshold λ_j^{sub*} is then given by

$$\lambda_j^{sub*}(\sigma_{\mathbf{X}_j^{sub}}, \beta) = \text{argmin}_{\lambda} R(\mathbf{X}_j^{sub}, \lambda). \tag{3.36}$$

which is a function of the parameters $\sigma_{\mathbf{X}_j^{sub}}$ and β.

Since there is no closed form solution for λ_j^{sub*}, numerical calculation is used to find its value. It was discovered that the threshold value set by

$$\hat{\lambda}_j^{sub*} = \frac{\sigma_{\mathbf{w}}^2}{\sigma_{\mathbf{X}_j^{sub}}} \tag{3.37}$$

is near optimal and indeed very close to λ_j^{sub*} [13, 14, 15, 16]. The estimated threshold $\hat{\lambda}_{Bayes}^* = \frac{\sigma_{\mathbf{w}}^2}{\sigma_{\mathbf{X}_j^{sub}}}$ is not only nearly optimal but also has an intuitive appeal. The normalized threshold, $\frac{\hat{\lambda}_{Bayes}}{\sigma_{\mathbf{w}}}$, is inversely proportional to $\sigma_{\mathbf{X}_j^{sub}}$, the standard deviation of $\mathbf{X}_j^{sub}$, and proportional to $\sigma_{\mathbf{w}}$, the noise standard deviation. This quantity can be interpreted as a noise-to-signal-ratio. There are two limiting cases:

- When $\frac{\sigma_{\mathbf{w}}}{\sigma_{\mathbf{X}_j^{sub}}} << 1$, the signal is much stronger than the noise, $\frac{\hat{\lambda}_j^{sub*}}{\sigma_{\mathbf{w}}}$ is chosen to be small in order to preserve most of the signal and remove some of the noise. This occurs near edges and other high frequency content of the signal where the noise to signal ratio is relatively small. Performing little or no denoising or smoothing in these sub-regions will preserve the sharpness of the image edges.

- On the other hand, when the noise-to-signal ratio is high, i.e. $\frac{\sigma_{\mathbf{w}}^2}{\sigma_{\mathbf{X}_j^{sub}}} >> 1$, the noise dominates and the normalized threshold is chosen to be large to remove the noise which has overwhelmed the signal. Generally, this occurs in flat and low activity sub-regions of the image where extra denoising and smoothing can be performed without degrading the most important features of the image.

Thus, this choice of threshold adapts to both the signal and the noise characteristics as reflected in the parameters $\sigma_{\mathbf{W}}$ and $\sigma_{\mathbf{X}_j^{sub}}$. It is also important to note that this Bayesian thresholding strategy is consistent with the human visual system which is less sensitive to the presence of noise in the vicinity of edges. However, the presence of noise in flat regions of the image is perceptually more noticeable. Exploiting these unique characteristics of the human visual system has been explored in the literature, and the Lee filter, described in chapter 1, is one such example [49].

Parameter Estimation

The GGD parameters, $\sigma_{\mathbf{X}_j^{sub}}$ and β, need to be estimated to compute $\hat{\lambda}^*_{Bayes}$. The parameter β does not explicitly enter into the expression of $\hat{\lambda}_{Bayes}$. Therefore it suffices to estimate directly the signal standard deviation $\sigma_{\mathbf{X}_j^{sub}}$. The observation model is:

$$\mathbf{Y}_j^{sub} = \mathbf{X}_j^{sub} + \mathbf{W}, \tag{3.38}$$

with $\mathbf{X}$ and $\mathbf{W}$ being independent of each other, hence

$$\sigma^2_{\mathbf{Y}_j^{sub}} = \sigma^2_{\mathbf{X}i_j^{sub}} + \sigma^2_{\mathbf{W}}, \tag{3.39}$$

where $\sigma^2_{\mathbf{Y}_j^{sub}}$ is the variance of the observed noisy image $\mathbf{Y}$ and can be computed as follows:

$$\hat{\sigma}^2_{\mathbf{Y}_j^{sub}} \approx \frac{1}{M_j}\sum_m^{M_j}[Y_{j,m}^{sub} - \bar{Y}_j^{sub}]^2, \text{ where } \bar{Y}_j^{sub} =\approx \frac{1}{M_j}\sum_m^{M_j} Y_{j,m}^{sub}. \tag{3.40}$$

Thus $\sigma^2_{\mathbf{X}_j^{sub}}$ can be estimated by re-arranging Eq. (3.39) as follows:

$$\sigma^2_{\mathbf{X}_j^{sub}} = \sigma^2_{\mathbf{Y}i_j^{sub}} + \sigma^2_{\mathbf{W}} \tag{3.41}$$

However, this estimate may be negative, so to avoid these unexpected cases, one may choose:

$$\hat{\sigma}_{\mathbf{X}_j^{sub}} = \sqrt{\max(\hat{\sigma}^2_{\mathbf{Y}_j^{sub}} - \hat{\sigma}^2_{\mathbf{W}}, 0)}. \tag{3.42}$$

The noise variance, $\sigma^2_{\mathbf{W}}$, can be estimated using the wavelet-based method outlined in section 1.1.2.

Once the statistics, $\sigma_{\mathbf{X}_j^{sub}}$ and $\sigma^2_{\mathbf{W}}$, are estimated, the near optimal threshold, $\hat{\lambda}_j^{sub*}$, adopted by the BayesShrink method can be computed as follows:

$$\hat{\lambda}_j^{sub*} = \frac{\hat{\sigma}^2_{\mathbf{W}}}{\hat{\sigma}_{\mathbf{X}_j^{sub}}} \tag{3.43}$$

Note that in the case where $\hat{\sigma}^2_{\mathbf{W}} \geq \hat{\sigma}^2_{\mathbf{Y}_j^{sub}}$, $\hat{\sigma}^2_{\mathbf{X}_j^{sub}}$ is taken to be zero, i.e. $\hat{\lambda}_j^{sub*} \longrightarrow \infty$. Alternatively, in practice, one may choose $\hat{\lambda}_j^{sub*} = \max_{m=1,2,\ldots,M_j}\{|Y_{j,m}^{sub}|\}$, and all coefficients are set to zero.

In summary, the BayesShrink thresholding technique performs soft thresholding with an adaptive, data-driven, subband and level-dependent near optimal threshold given by:

$$\hat{\lambda}_j^{sub*} = \begin{cases} \frac{\hat{\sigma}^2_{\mathbf{W}}}{\hat{\sigma}_{\mathbf{X}_j^{sub}}}, & \text{if } \hat{\sigma}^2_{\mathbf{W}} < \hat{\sigma}^2_{\mathbf{Y}_j^{sub}}, \\ \max_{m=1,2,\ldots,M_j}\{|Y_{j,m}^{sub}|\}, & \text{otherwise}, \end{cases} \tag{3.44}$$

for each subband $sub \in \{h, v, d\}$ and each decomposition level $j = 1, 2, \ldots, J$.

Experimental Results

The BayesShrink thresholding method was implemented for the purpose of denoising the noisy test image of “Lenna”. Table 3.4 illustrates the optimal thresholds obtained by using the BayesShrink technique. Note that the optimal thresholds vary not only from one level to the next but also from one subband to another. Similar to the SureShrink thresholding strategy, the optimal threshold value vary from one wavelet decomposition level to another as from one subband to the next. Note also, similar to the previous two adaptive thresholding methods, BayesShrink uses larger threshold values at finer scale levels of the wavelet tree and smaller thresholds for more coarse scales. Similar to the SureShrink case, the the optimal threshold adopted by the BayesShrink method is derived for the purpose of soft thresholding. The optimal hard threshold is again chosen to be twice that used by the soft threshold.

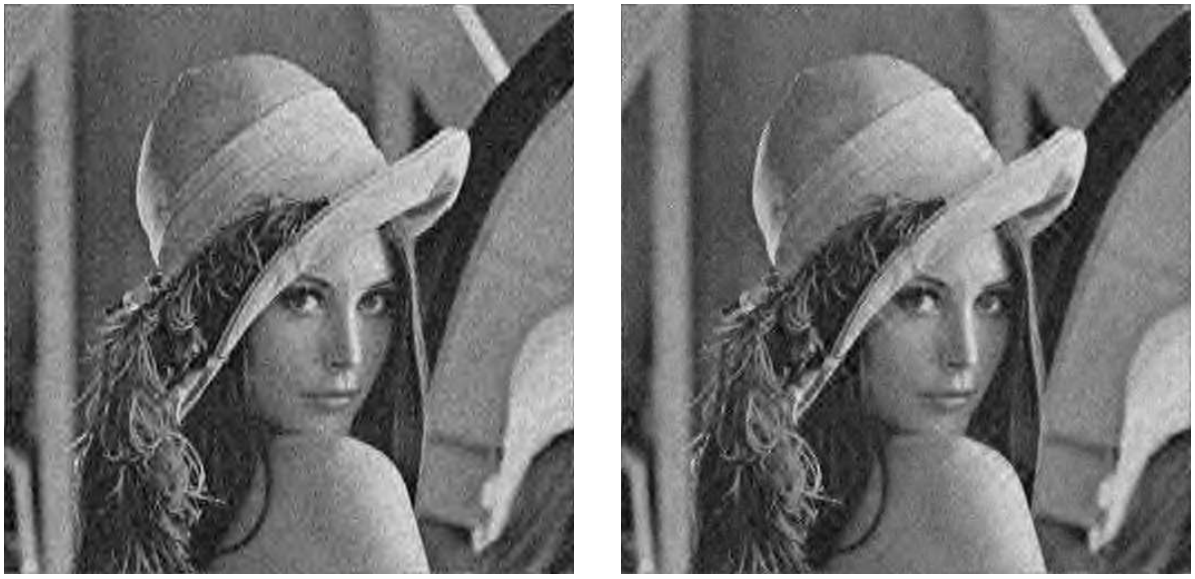
	Wavelet Decomposition Level				
Level	9	8	7	6	5
Horizontal	130.09	43.42	16.43	6.82	3.04
Vertical	77.66	25.90	9.52	3.62	1.18
Diagonal	219.09	55.99	17.92	7.34	2.70

Table 3.4: The optimal thresholds for the various wavelet decomposition levels and subbands used by the BayesShrink soft thresholding scheme for the first five wavelet decomposition levels.

The results obtained by the BayesShrink for the image of “Lenna” is shown in Figure 3.9. The BayesShrink performs better than SureShrink in terms of the RMSE and PSNR fidelity measures. The reconstruction using BayesShrink is smoother and more visually appealing than the one obtained using SureShrink. This not only validates the approximation of the wavelet coefficients to the GGD but also justifies the use of the Bayes threshold:

$$\hat{\lambda}_j^{sub*} = \frac{\hat{\sigma}_{\mathbf{w}}^2}{\hat{\sigma}_{\mathbf{X}_j^{sub}}}. \tag{3.45}$$

which is related to the noise-to-signal ratio. Using this threshold, BayesShrink yields results that are consistent with the human visual system where extra denoising is performed in flat regions of the image and less denoising is performed near edges to preserve the sharpness of the image. Note also that this threshold is independent of the distribution parameter β which makes it more robust as it does not depend on the estimate of the distribution parameters.



Hard BayesShrink thresholding
RMSE=10.07, PSNR=28.07.
Execution time $\approx$ 12 secs.

Soft BayesShrink thresholding
RMSE=9.93, PSNR=28.19.
Execution time $\approx$ 14 secs.

Figure 3.9: Hard and soft threshold denoised estimates of "Lenna" using using the BayesShrink method applied on the first five decomposition levels. Note that the threshold for hard thresholding was chosen to be twice that of the optimal Bayes threshold derived for soft thresholding.

Next, a brief comparison between the four wavelet thresholding methods for image denoising described and implemented in this section is presented.

3.2.5 Comparison Between the Studied Wavelet Thresholding Methods

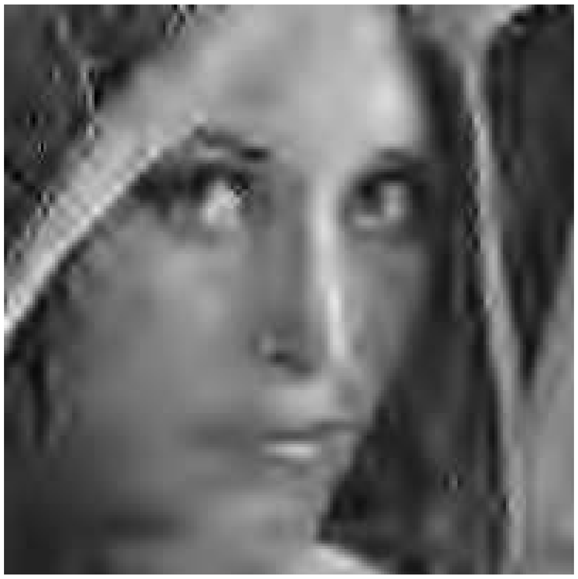
This section is concluded by a brief comparison between the various wavelet thresholding methods studied in this section. Figures 3.10 and 3. 11 summarize the results for the various schemes. The VisuShrink method, which uses the universal threshold uniformly throughout the wavelet tree, yields the worst results. This is expected because the universal threshold tends to be conservatively high resulting in extra smoothing and visible degradation of the edges and sharpness of the image. Also applying the same threshold uniformly throughout the wavelet tree is counter-intuitive since the local statistics of the wavelet tree vary generally from one decomposition level to the next and one subband to another. The LevelShrink thresholding scheme, which uses thresholds that adapt only to the wavelet decomposition level, yields better results than VisuShrink. However, if more adaptive thresholds that vary not only from one decomposition level to the next but also from one subband to another are selected, even better results can be achieved. For each of the adaptive thresholding methods, note that larger threshold values are used at finer scale levels of the wavelet tree and smaller thresholds are used for more coarse scales.

The critical thresholds for the SureShrink and BayesShrink methods were derived for the soft thresholding operator $T_s(., \lambda)$. However, a widely acceptable practice in the literature where the critical threshold for the hard-thresholding operator is taken to be twice that corresponding to soft thresholding was adopted. The BayesShrink technique yields the best results and also adopts a thresholding strategy that not only performs well but it is also intuitively appealing. The quantity $\frac{\hat{\lambda}_j^{sub*}}{\hat{\sigma}_{\mathbf{W}}} = \frac{\hat{\sigma}_{\mathbf{W}}}{\hat{\sigma}_{\mathbf{X}_j^{sub}}}$ can be interpreted as a noise to signal ratio. BayesShrink performs denoising that is consistent with the human visual system that is less sensitive to the presence of noise in the vicinity of edges. However, the presence of noise in flat regions of the image is perceptually more noticeable by the human visual system. BayesShrink performs little denoising in high activity sub-regions to preserve the sharpness of edges but completely denoises the flat sub-parts of the image.

Next, the use of the cycle spinning algorithm in order to improve the quality of the denoised images obtained by various wavelet thresholding methods is illustrated.

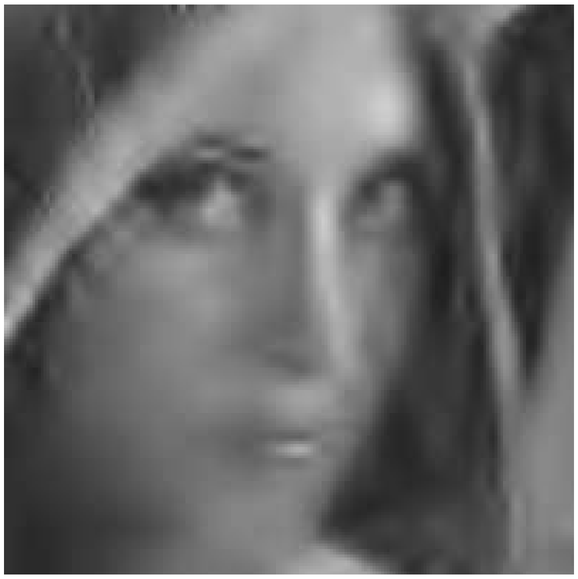
3.3 Improving Wavelet Image Denoising via Cycle Spinning

In spite of the significant developments outlined in the previous section, wavelet thresholding methods are not without limitations. Most notably, denoising with the traditional wavelet transform (orthogonal, maximally decimated) wavelet transforms often exhibit disturbing visual artifacts. In



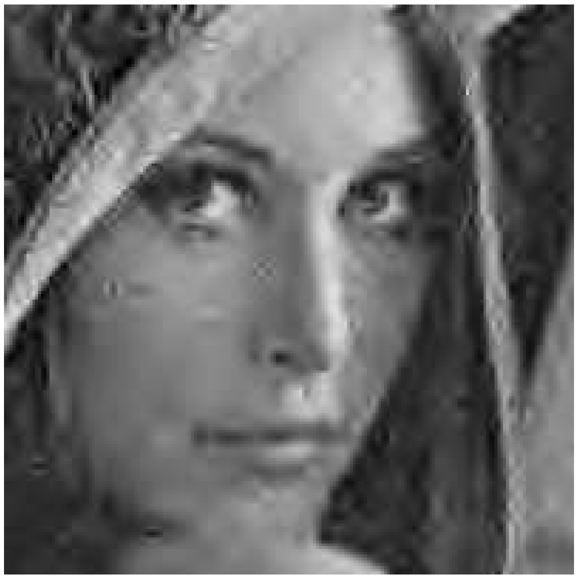
VisuShrink: Hard thresholding

RMSE=12.37, PSNR=26.28.



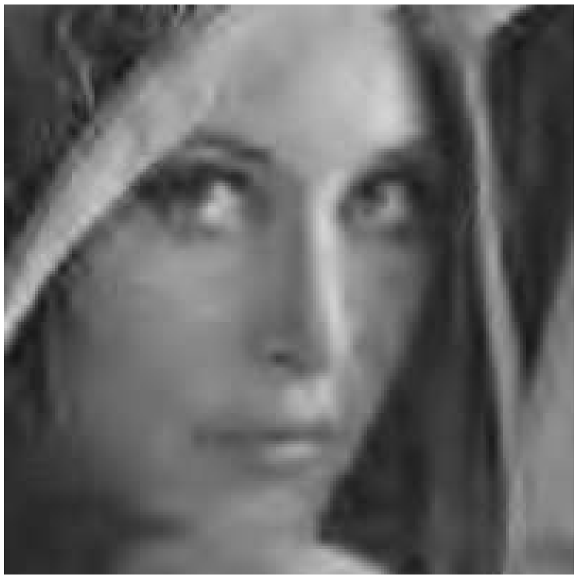
VisuShrink: Soft thresholding

RMSE=15.76, PSNR=24.18.



LevelShrink: Hard thresholding

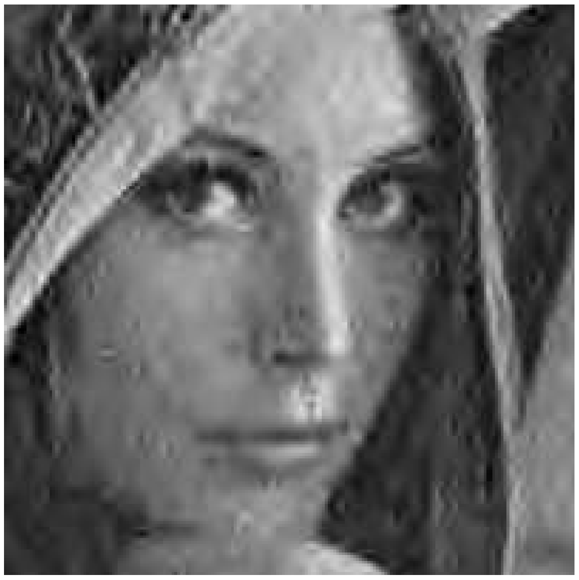
RMSE=10.01, PSNR=28.11.



LevelShrink: Soft thresholding

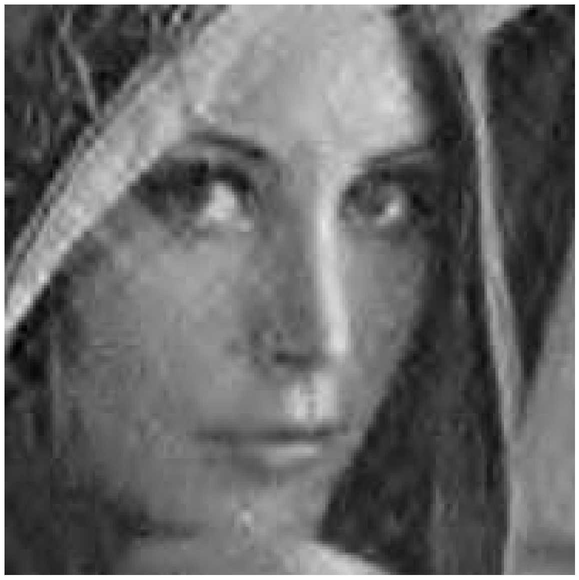
RMSE=11.30, PSNR=27.07.

Figure 3.10: Zooming in on the denoised estimates obtained by VisuShrink and LevelShrink methods reveals the pseudo-Gibbs artifacts.



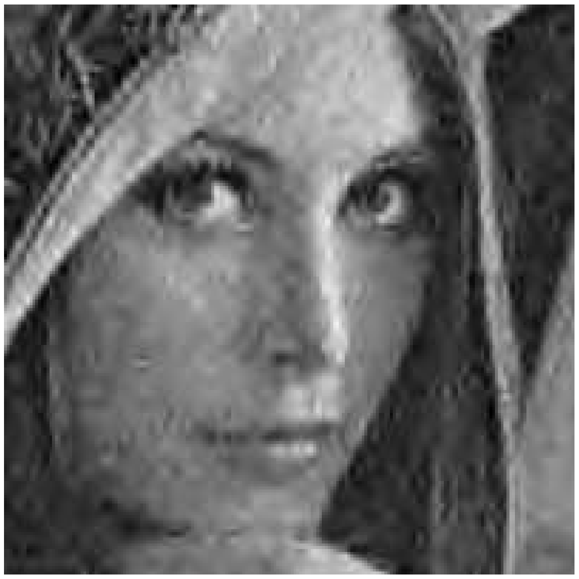
SureShrink: Hard thresholding

RMSE=9.89, PSNR=28.23.



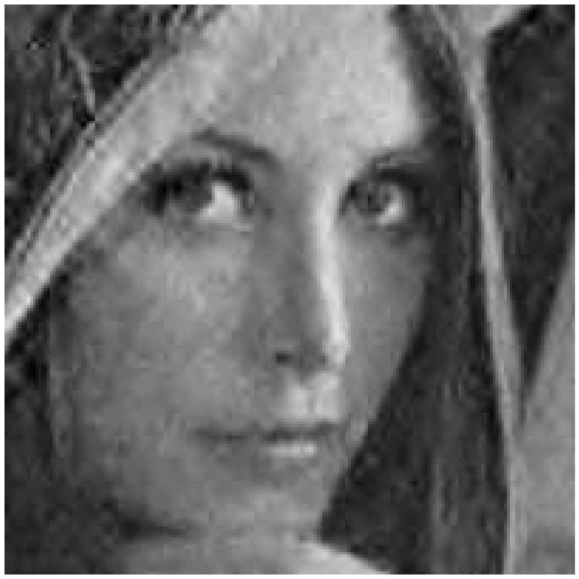
SureShrink: Soft thresholding

RMSE=9.96, PSNR=28.16.



BayesShrink: Hard thresholding

RMSE=10.07, PSNR=28.07.



BayesShrink: Soft thresholding

RMSE=9.93, PSNR=28.19.

Figure 3.11: Zooming in on the denoised estimates obtained by SureShrink and BayesShrink methods reveals the pseudo-Gibbs artifacts.

particular, pseudo-Gibbs phenomena tend to be noticeable in the vicinity of edges. This is mainly due to the lack of translation invariance of the wavelet basis.

As illustrated in Figures 3.10 - 3.11, when zooming on the denoised images obtained by the standard wavelet thresholding methods described in the previous section namely, VisuShrink, LevelShrink, SureShrink and BayesShrink, the pseudo-Gibbs artifacts become quite evident. In this section, the cycle spinning algorithm will be applied in order to improve the performance of these thresholding methods and reduce some of these disturbing artifacts in the denoised images.

The idea of using "cycle spinning" has been previously proposed for the purpose of reducing the pseudo-Gibbs disturbing artifacts that are often present in wavelet-based image reconstruction and denoising [17]. This is performed as follows:

> For a range of shifts, one shifts the image, horizontally or vertically or both, denoises the shifted data using a wavelet thresholding technique of choice, and then unshifts the denoised image. Doing this for each of a range of shifts, and averaging the several results so obtained, produces a reconstruction subject to weaker pseudo-Gibbs phenomena than the thresholding-based denoising using the traditional orthogonal wavelet transform.

This is a consequence of the fact that the discrete wavelet transform is not translation invariant in the case of a periodic signal. In other words, if a periodic signal is shifted, then its wavelet decomposition coefficients are not simply permuted. Mathematical details of this fact have been studied in [53].

The Cycle Spinning Algorithm

In order to formally define this process one needs to introduce the appropriate notations. Consider a noisy image, $\mathbf{y}$ of size $M \times M$. Clearly, there are various ways the image could be shifted; one could shift it horizontally, vertically or both. In general, the results are more sensitive to the total number of shifts rather than to the manner the shifting is performed. Thus, a simple shifting operation that shifts the image horizontally and vertically by the same amount h along its diagonal is adopted. The two-dimensional circular shifting operator D_h is defined as follows:

$$\mathbf{y}^{(h)} = D_h(\mathbf{y}), \tag{3.46}$$

where

$$\mathbf{y}^{(h)} = [y(1 + h \ mod \ M, 1 + h \ mod \ M), \ldots, y(M + h \ mod \ M, M + h \ mod \ M)]. \tag{3.47}$$

and $k \ mod \ M$ represents the remainder when k is divided by M. Note that the shift operator is unitary and

$$(D_h)^{-1} = D_{-h} \tag{3.48}$$

Also, as before the thresholding operator with corresponding threshold λ is denoted by T_λ.

In view of these notations, the cycle spinning algorithm for the purpose of reducing the pseudo-Gibbs artifacts can be outlined as follows:

The Cycle Spinning Algorithm:

For a one-dimensional signal of size M and a given number of shifts $K \leq M$:

1. Initialize the sum signal: $\mathbf{s} = \mathbf{0}$.
2. For each shift h in a range of shifts, $\{0, 1, 2, \ldots, K\}$, repeat:
   - Shift the noisy signal $\mathbf{y}$ by h to obtain the shifted signal $\mathbf{y}^{(h)}$:
   $$\mathbf{y}^{(h)} = D_h(\mathbf{y}) \tag{3.49}$$
   - Compute the discrete wavelet transform, $\mathbf{Y}^{(h)}$ of $\mathbf{y}^{(h)}$:
   $$\mathbf{Y}^{(h)} = \mathcal{DWT}(\mathbf{y}^{(h)}). \tag{3.50}$$
   - Apply the thresholding operator T_λ of choice to obtain the denoised version, $\hat{\mathbf{X}}^{(h)}$ of $\mathbf{Y}^{(h)}$:
   $$\hat{\mathbf{X}}^{(h)} = T_\lambda(\mathbf{Y}^{(h)}). \tag{3.51}$$
   - Take the inverse discrete wavelet transform of $\hat{\mathbf{X}}^{(h)}$ to obtain a denoised version, $\hat{\mathbf{x}}^{(h)}$ of the shifted signal:
   $$\hat{\mathbf{x}}^{(h)} = \mathcal{IDWT}(\hat{\mathbf{X}}^{(h)}). \tag{3.52}$$
   - Now unshift $\hat{\mathbf{x}}^{(h)}$ to obtain a denoised version, $\hat{\mathbf{x}}$, of the original signal:
   $$\hat{\mathbf{x}} = D_{-h}(\mathbf{x}^{(h)}) \tag{3.53}$$
   - Update the sum signal of these estimates:
   $$\mathbf{s} = \mathbf{s} + \hat{\mathbf{x}}. \tag{3.54}$$

3. Compute the average of the above denoised estimates to obtain one denoised signal that has resulted from the above K estimates:

$$\hat{\mathbf{x}}_K = \frac{\mathbf{s}}{K} \tag{3.55}$$

This algorithm can be summarized as follows:

$$\hat{\mathbf{x}}_K = \frac{1}{K}\sum_{h=0}^{K} D_{-h}(\mathcal{IDWT}(T_\lambda(\mathcal{DWT}(D_h(\mathbf{y}))))). \tag{3.56}$$

Since the image is assumed to be periodic with period M, better results can be obtained by using a higher number of shifts $K \in \{0, 1, 2, \ldots, M-1\}$. When $K = M-1$, it is said that total-shift cycle spinning is performed, otherwise only partial-shift cycle spinning is performed. As will be illustrated, the quality of the denoised signal, as measured by the RMSE and PSNR fidelity measures, improves considerably for the first few values of K. However, for larger values of K, no visible gain in achieved by increasing K even further.

Clearly the cycle spinning algorithm may be rather computationally expensive. Indeed, when incorporating this algorithm with K shifts for any denoising method, the computational complexity is multiplied by K times.

Experimental Results

A range of shifts was tested and it found that the quality of the denoised image stabilizes after only a few shifts. Thus, a range of shifts between $K = 1$ and $K = 16$ is selected, that is:

$$1 \leq h \leq K = 16. \tag{3.57}$$

The cycle spinning algorithm was applied to the various thresholding methods, described in the previous section, and the results are illustrated in Figures 3.12 - 3.15. In view of these results, and after comparing the zoomed images in each of these figures to those illustrated in Figures 3.10 - 3.11, where no cycle spinning is performed, it can be concluded that for each of the thresholding methods (hard or soft), the application of the cycle spinning method has resulted in some reduction of the pseudo-Gibbs artifacts and overall improvement of the quality of the denoised images.

In Figure 3.16, the quality of the denoised images as a function of the number of shifts is illustrated for the various wavelet thresholding methods, in order to assess the benefits of using the cycle spinning for the various schemes. Examining this figure, one may conclude that SureShrink and BayesShrink yield comparable results that are better than the results obtained by VisuShrink and relatively better than the results obtained by the LevelShrink thresholding technique. However, in general, the BayesShrink method is preferred due to its simplicity compared to SureShrink, which is computationally expensive. Besides, the BayesShrink method adopts a threshold that is intuitively appealing and consistent with the human visual system. Note also that for most of the methods, the cycle spinning algorithm results in significant improvement of the quality of the image after only a few shifts. After the first few shifts, the quality of the denoised estimate becomes less sensitive to increasing the number of shifts.

In this section, the use of the cycle spinning idea for the purpose of reducing the pseudo-Gibbs artifacts and improving the quality of the denoised estimates obtained by the various wavelet thresholding methods was illustrated. Next, a context-based thresholding strategy that takes into consideration the content of an immediate neighborhood of each wavelet coefficient before thresholding will be proposed and implemented.

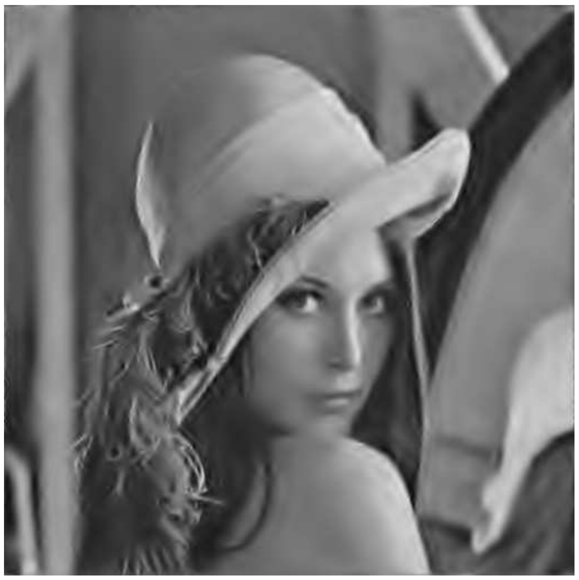
3.4 Context-Based Thresholding for Image Denoising

Recall that all the wavelet thresholding for image denoising methods covered so far adopt the standard hard thresholding operator $T_h(., \lambda)$ and the soft thresholding operator $T_s(., \lambda)$, as defined in (3.4) and (3.5), respectively. The following observations regarding the use of these thresholds are outlined:

- While some of the wavelet thresholding methods studied so far, in particular the LevelShrink, SureShrink and BayesShrink, attempt to employ thresholds that are adaptive to the local characteristics of the signal, they all apply the above hard and soft thresholding operators.

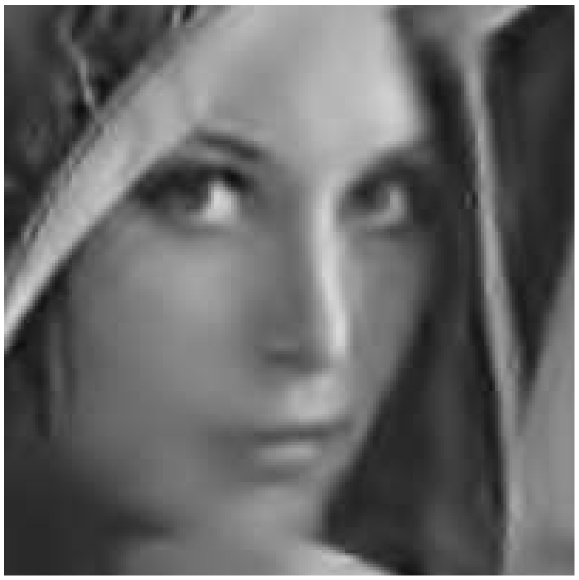
- For a given threshold λ, the hard and soft thresholding operators defined above are global and non-adaptive in nature. They are applied on each wavelet coefficients in the same manner regardless of its location or context. The thresholded coefficient only depends on the value of the noisy coefficients and it is independent of other neighboring or context coefficients.

- While the wavelet transform performs some degree of decorrelation, it is evident that there is still a some degree of redundancies within the wavelet decomposition tree. In fact, natural images structures generally possess similarities across resolution scales of their wavelet coefficients. For instance, wavelet coefficients corresponding to a high activity subregion (such as edges) are often clustered together and copied across the various resolutions and subbands of



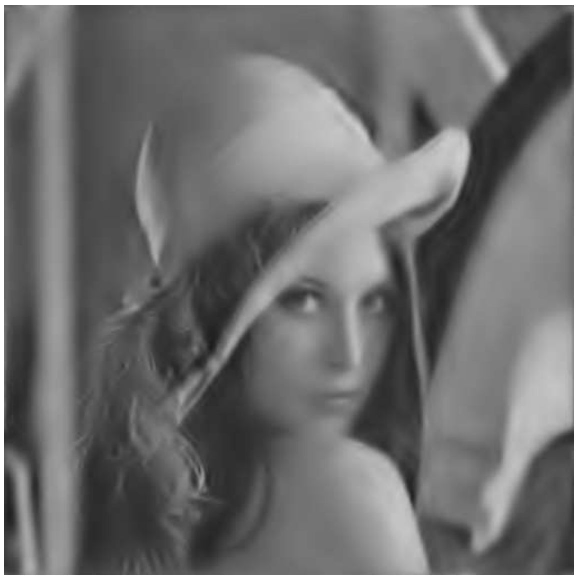
(a) Hard VisuShrink thresholding

RMSE=10.27, PSNR=27.90.



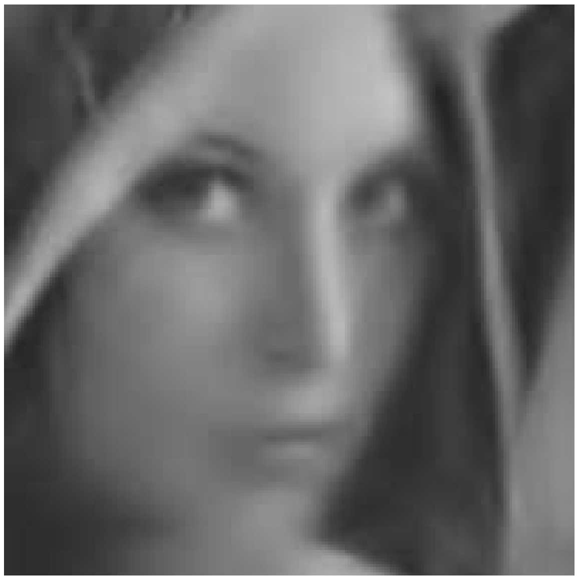
(b) Zooming in on the image in (a)

RMSE=14.94, PSNR=24.64.



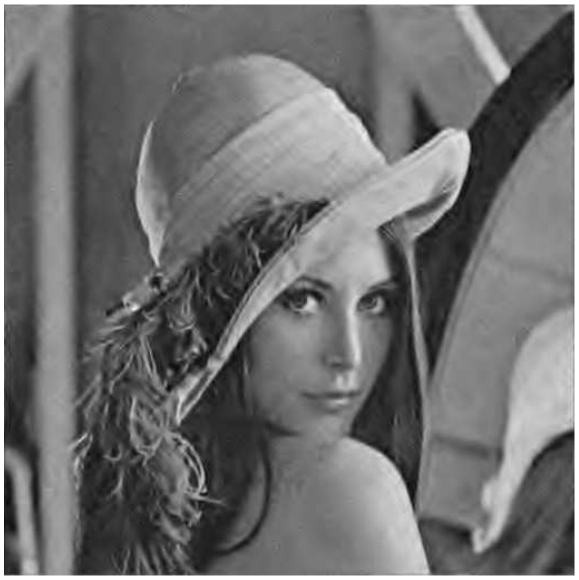
(c) Soft VisuShrink thresholding

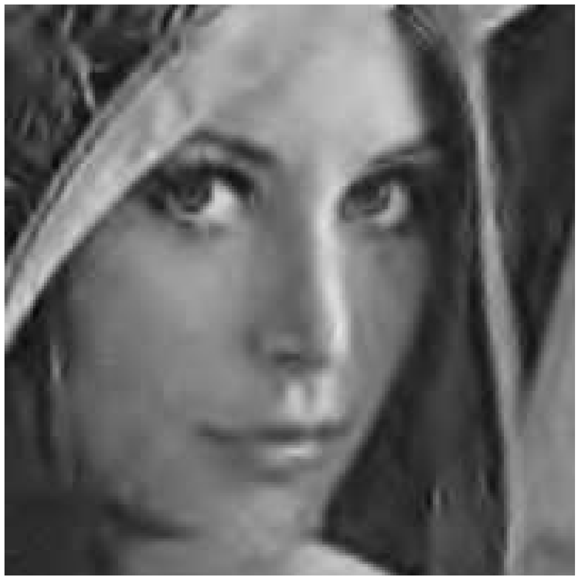
RMSE=14.94, PSNR=24.64.

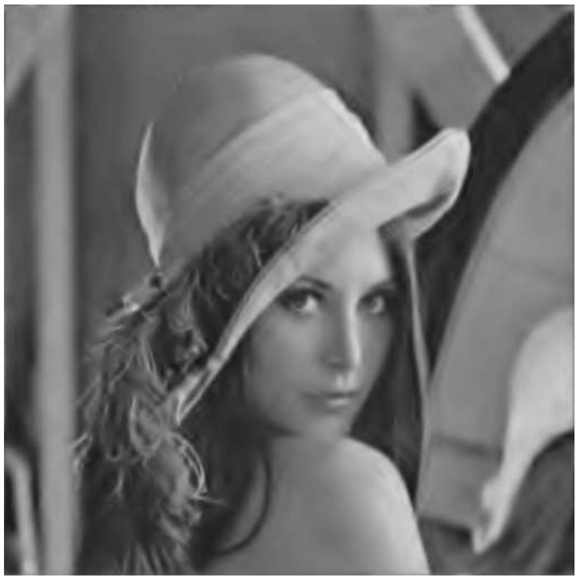


(d) Zooming in on the image in (b)

Figure 3.12: Results of applying the VisuShrink with *hard* and *soft* thresholding using cycle spinning with *K=16* diagonal shifts.


(a) Hard LevelShrink thresholding
RMSE=8.34, PSNR=29.70.


(b) Zooming in on the image in (a)


(c) Soft LevelShrink thresholding
RMSE=10.61, PSNR=27.61.

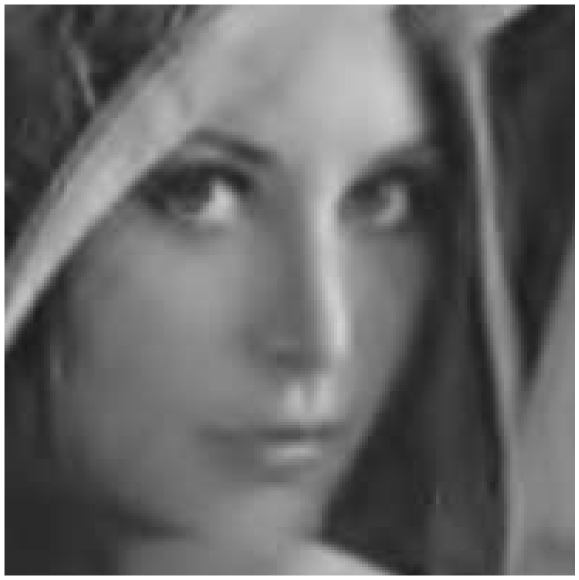
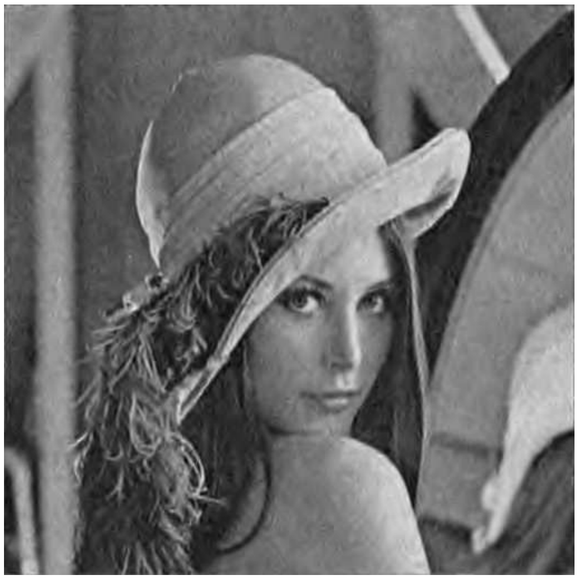
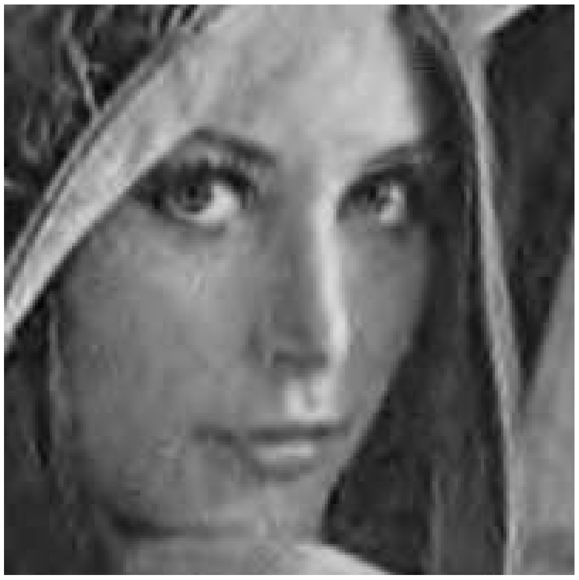
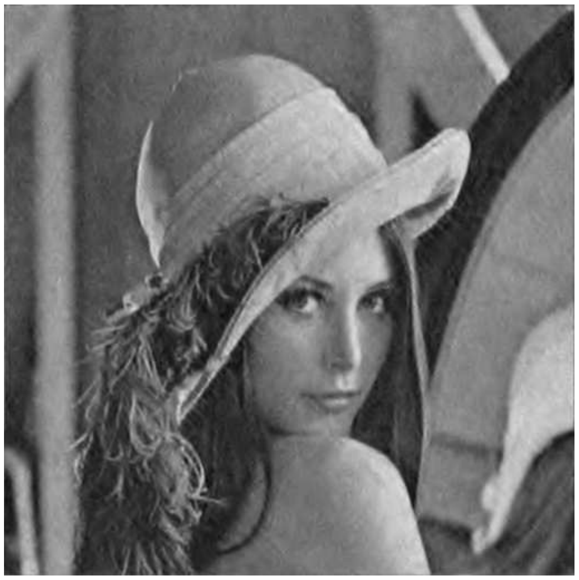

(d) Zooming in on the image in (c)

Figure 3.13: Results of applying the LevelShrink *hard* and *soft* thresholding using cycle spinning with *K=16* diagonal shifts.


(a) Hard SureShrink thresholding
RMSE=8.39, PSNR=29.65.


(b) Zooming in on the image in (a)


(c) Soft SureShrink thresholding
RMSE=8.73, PSNR=29.31.

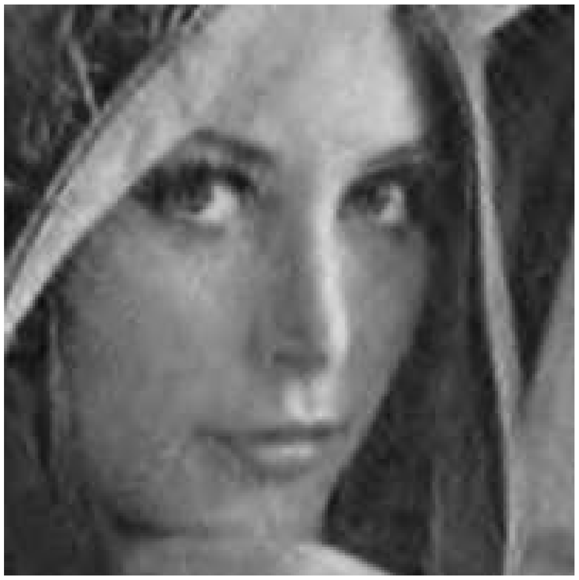
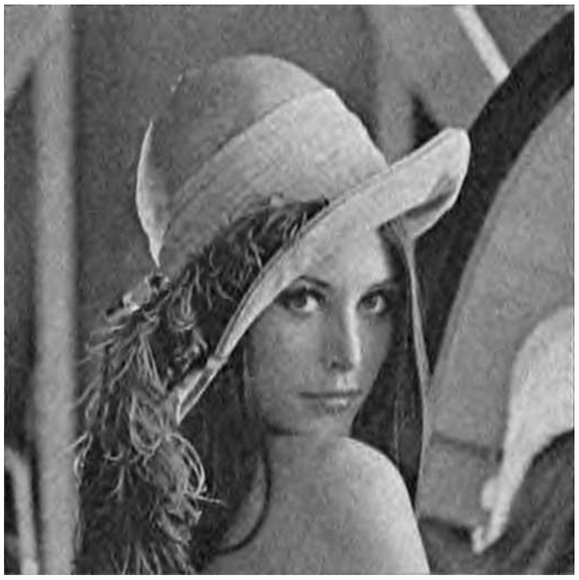
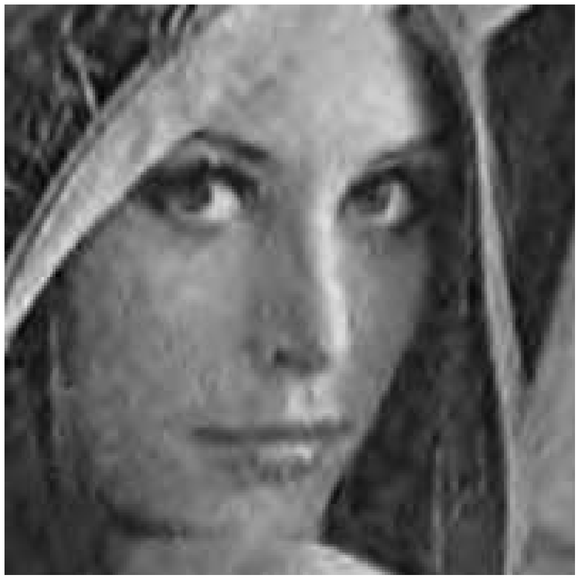

(d) Zooming in on the image in (c)

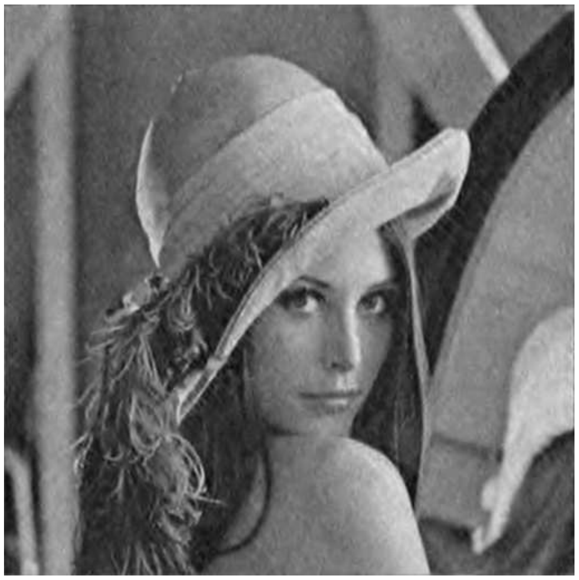
Figure 3.14: Results of applying the SureShrink with *hard* and *soft* thresholding using cycle spinning with *K= 16* diagonal shifts.



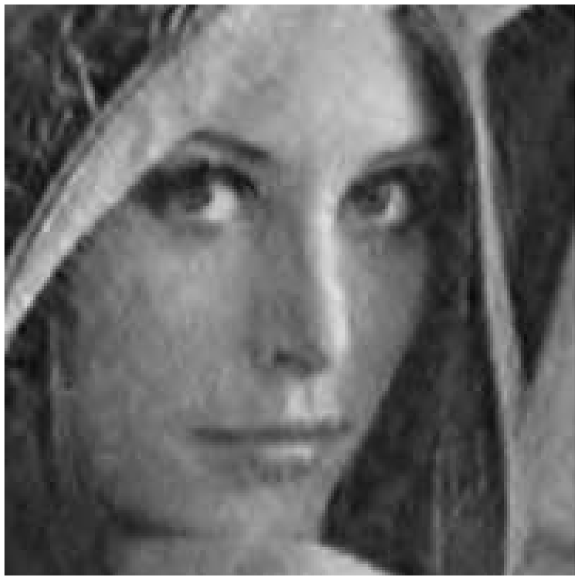
(a) Hard BayesShrink thresholding
RMSE=8.64, PSNR=29.40.



(b) Zooming in on the image in (a)
RMSE=8.66, PSNR=29.38.

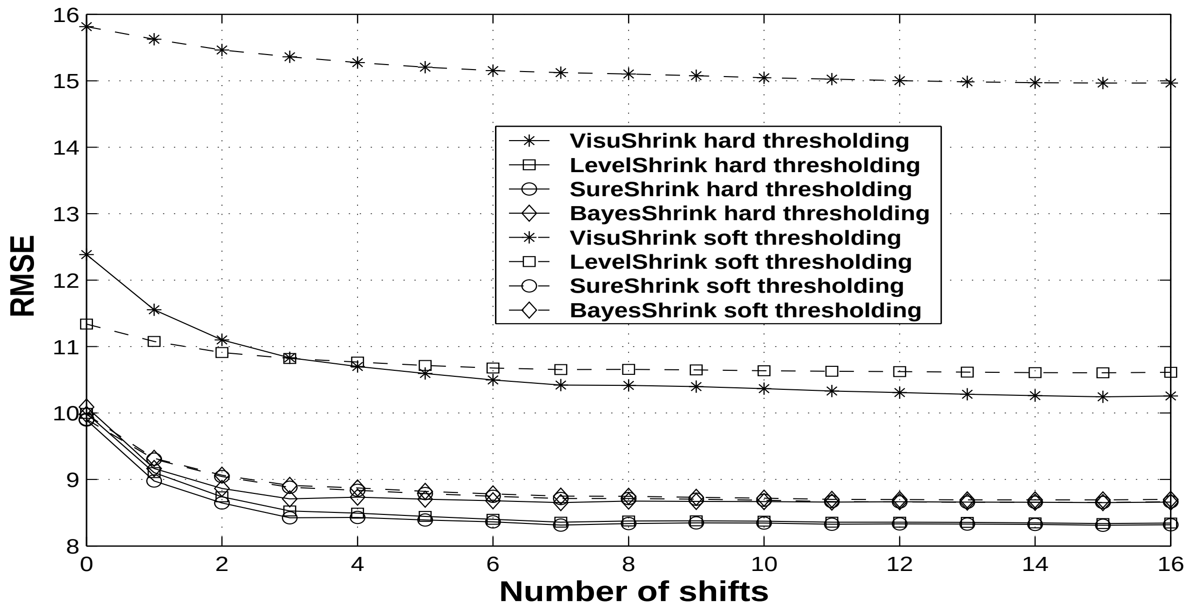


(c) Soft BayesShrink thresholding
RMSE=8.66, PSNR=29.38.



(d) Zooming in on the image in (c)

Figure 3.15: Results of applying the BayesShrink with *hard* and *soft* thresholding using cycle spinning with *K=16* diagonal shifts.



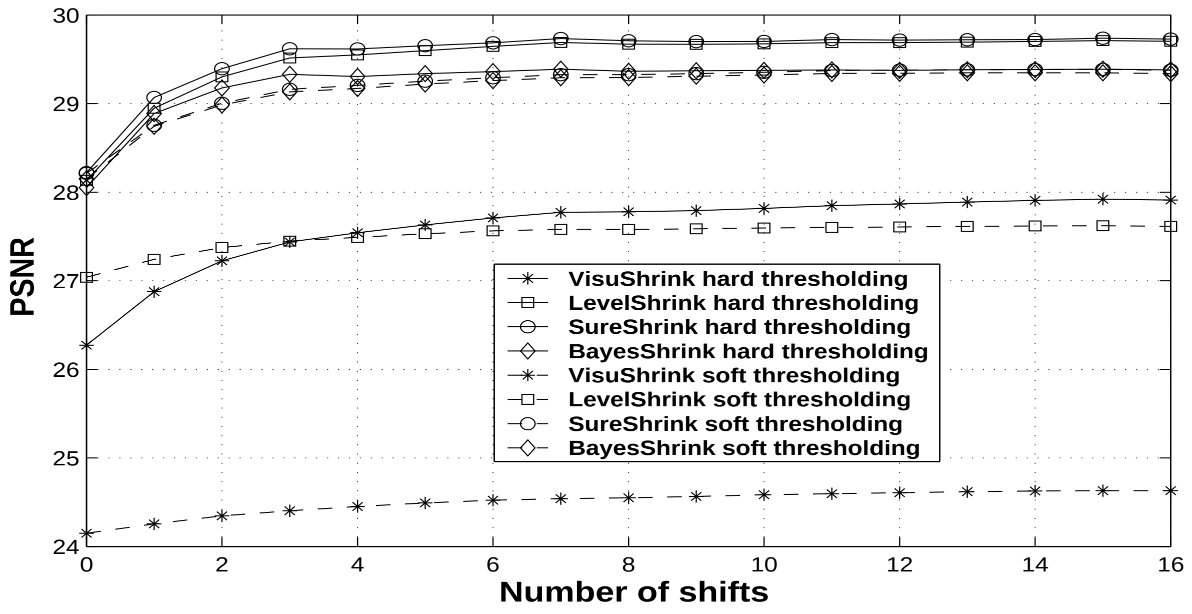


Figure 3.16: Comparison between the various thresholding methods after applying the idea of the cycle spinning with *K=16* shifts.

the wavelet tree, as illustrated in Figure 2.12 (a). Thus, one should expect some degree of dependence among neighboring wavelet coefficients corresponding to high activity subregions of the image. Hence, thresholding these coefficients independently may not be appropriate.

- Doing so may result in zeroing out some of the significant wavelet coefficients in a cluster of correlated wavelet coefficients, resulting in a few significant wavelet coefficients surrounded by many thresholded coefficients, which were set to zero. In the denoised image, this generally results in over-smoothing, blurring and ringing artifacts. These blurring and over-smoothing artifacts are most evident in the case of the VisuShrink and the LevelShrink thresholding methods where the denoised estimates appear overly smooth and blurry.
- It seems more reasonable that when thresholding a wavelet coefficient to take the values of other neighboring coefficients into consideration.

Next, the context-based localized thresholding operators are defined.

3.4.1 Context-Based Thresholding Operators

Many efforts in the literatures have focused on selecting more adaptive thresholds [13, 14, 54, 21, 22, 54, 73]. In particular in [13], an effective, highly spatially adaptive thresholding strategy that selects thresholds that vary from one coefficients to another was proposed. However, all of these methods focus only on making the thresholds adaptive and continue to apply the usual hard and soft thresholding operators defined above. In this section, located thresholding operators that account for the local content characteristics of the wavelet coefficients will be proposed.

As mentioned above, it seems reasonable to take some context of each wavelet coefficient into consideration before thresholding. There are many ways of defining a suitable context of a wavelet coefficient. Employing contexts of wavelet coefficients has been shown to be effective for the purpose of image compression [66, 68]. The use of context-based thresholding was also used for the purpose of deriving threshold values, λ, that are more adaptive. Here a context-based and localized soft and hard thresholding operators are proposed. A simple context, which contains the neighboring wavelet coefficients centered at the coefficient to be thresholded, is considered. That is, for each wavelet coefficient, $y_{i,j}$, its context is defined by the $m \times m$ mask centered at $y_{i,j}$, and denoted by $\mathcal{C}_{m\times m}(y_{i,j})$. For this context, the maximum (in magnitude) value with this mask to be $M_{i,j}$ is defined as follows:

$$M_{i,j} = \max_{(k,l)\in\mathcal{C}_{m\times m}(y_{i,j})} |y_{k,l}|. \tag{3.58}$$

Now for a given threshold λ, consider the following modified, context-based hard and soft thresholding operators:

- The context-based hard thresholding operator is defined as:

$$\hat{\mathbf{X}} = T_h^c(\mathbf{Y}, \lambda) \text{ such that } \hat{x} = T_h^c(y_{i,j}, \lambda) = \begin{cases} y_{i,j}, & \text{if } |y_{i,j}| \geq \lambda \text{ or } M_{i,j} \geq \lambda, \\ 0, & \text{otherwise.} \end{cases} \tag{3.59}$$

- The context-based soft thresholding operator on the other hand is defined as:

$$\hat{\mathbf{X}} = T_s^c(\mathbf{Y}, \lambda) \text{ such that } \hat{x} = T_s^c(y_{i,j}, \lambda) = \begin{cases} y_{i,j} - \lambda, & \text{if } y_{i,j} \geq \lambda, \\ y_{i,j} + \lambda, & \text{if } y_{i,j} \leq -\lambda, \\ y_{i,j} & \text{if } |y_{i,j}| < \lambda \text{ and } M_{i,j} \geq \lambda, \\ 0, & \text{otherwise.} \end{cases} \tag{3.60}$$

These modified, context-based and localized thresholding operators are motivated in the following observations:

1. Note how these thresholds clearly take the values of the neighboring coefficients, located within the defined mask, into consideration before thresholding each wavelet coefficients.

2. For the modified hard thresholding operator, only those wavelet coefficients that are insignificant and also surrounded by insignificant coefficients are set to zero. However, a significant coefficient is kept unchanged if it is located near a significant one.

3. Similarly, for the modified soft thresholding operator, a wavelet coefficient is set to zero if and only if it is insignificant and it all of its neighbors are insignificant. However, an insignificant coefficient that is located near a significant one is left unchanged.

4. Clearly, the issue of selecting the context and its size requires further investigation. Larger masks result in sharper, but noisier estimates, exhibiting more artifacts. Also, defining the context itself requires further investigation. Instead of choosing the neighboring wavelet coefficients, perhaps one could choose a context containing the parent or children of the wavelet coefficient to be investigated as it is usually done in context-based wavelet image

coding such as in [66, 68]. Also, as described above, when an insignificant coefficient is surrounded by a significant one, it is kept unchanged. Clearly, one may decide to alter the value of such a coefficient without setting it to zero. Also, instead of taking the maximum absolute value, $M_{i,j}$, into consideration one may consider other statistics, such as the average or median. These are important issues that are open for investigation and they will be the focus of future research.

Next, the proposed thresholding operators will be implemented for the purpose of image denoising using the various various wavelet thresholding methods.

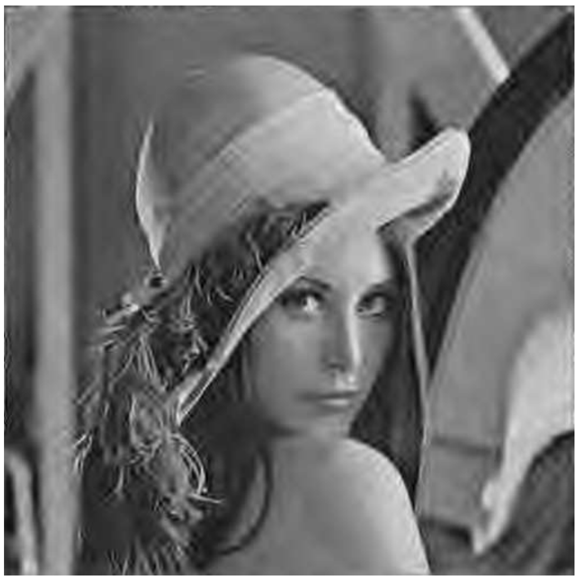
3.4.2 Experimental Results

The above context-based thresholding approach for the purpose of restoring and enhancing the noisy image of "Lenna", was implemented using the various wavelet thresholding methods studied in the previous section. A mask size of 3×3 coefficients was used for the experimental implementations. It was observed that when using larger masks, better quantitative results, as reflected by the RMSE and PSNR measures, may be obtained. However the resulting denoised estimates are not visually preferred since they tend to be noisy and suffer from some disturbing artifacts. This is the case because, for larger masks, more wavelet noisy coefficients are kept unchanged and hence resulting is reconstruction of some of the noise.

Figures 3.17 and 3.18 illustrate the denoised estimates obtained using hard and soft context-based thresholding for VisuShrink, LevelShrink, SureShrink and BayesShrink methods. Also, Table 3.5 summarizes a comparison of the quality of the denoised images obtained by the various thresholding methods using the traditional and the context-dependent thresholding operators. In view of these results it is observed that:

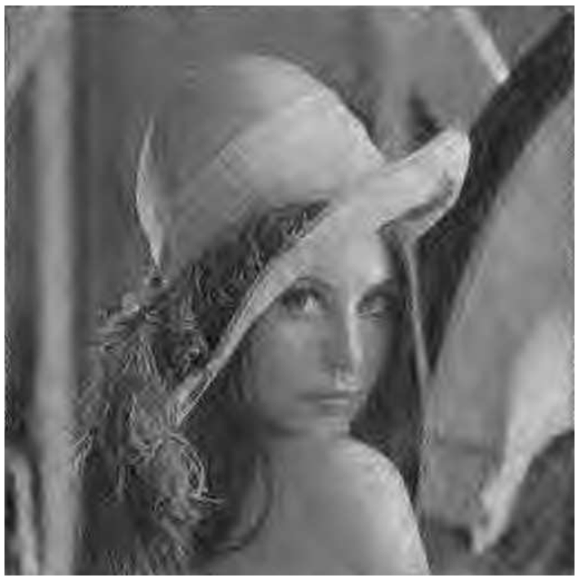
- For all thresholding methods, there is an improvement in the quality of the denoised estimates obtained using the context-based thresholding operators compared to the denoised images obtained by traditional thresholding schemes.

- For the VisuShrink and the LevelShrink thresholding methods, note that the denoised images obtained using the new thresholding strategy appear sharper and less blurry than the images obtained by the traditional thresholding methods. These improvement are also reflected through the RMSE and PSNR fidelity measures, as illustrated in Table 3.5. Note that best result is obtained when using the context-based hard LevelShrink thresholding method.



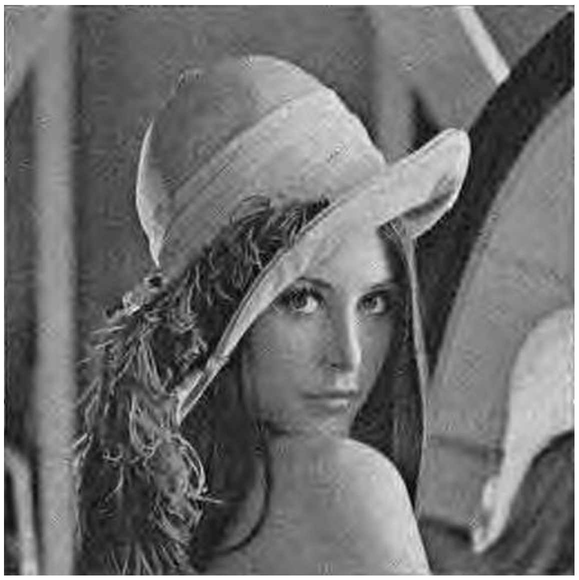
(a) Hard VisuShrink thresholding

RMSE=10.21, PSNR=27.95.



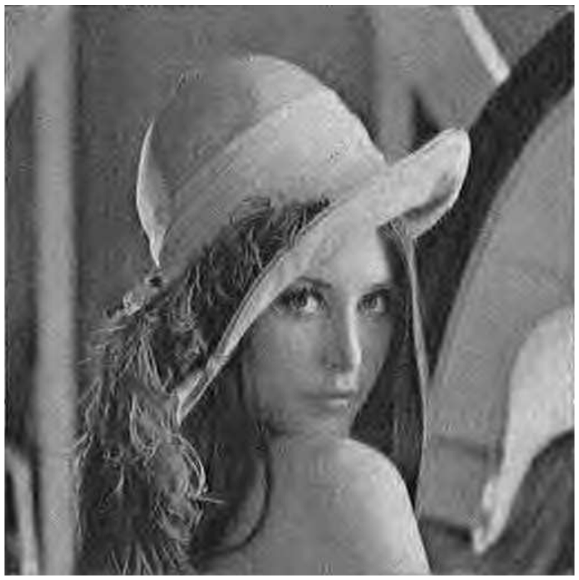
(b) Soft VisuShrink thresholding

RMSE=14.16, PSNR=25.11.



(c) Hard LevelShrink thresholding

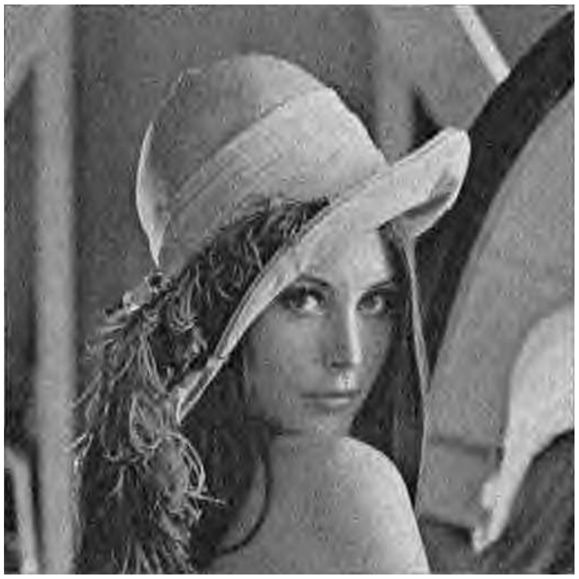
RMSE=9.37, PSNR=28.69.



(d) Soft LevelShrink thresholding

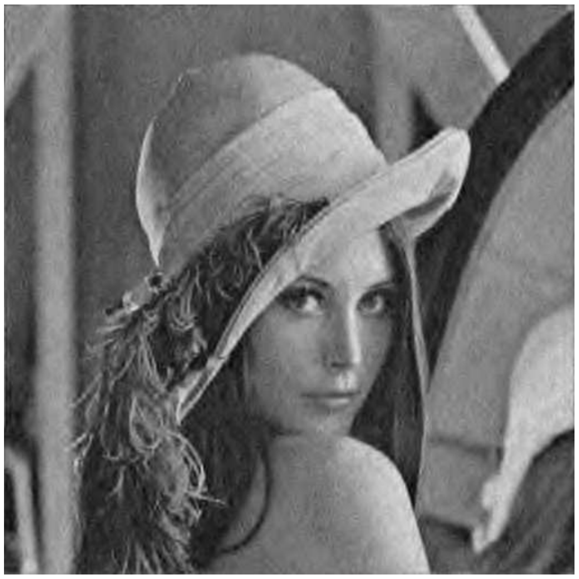
RMSE=10.07, PSNR=28.07.

Figure 3.17: Results of applying the *context-based hard* and *soft* thresholding using 3×3 masks for the VisuShrink and the LevelShrink methods.



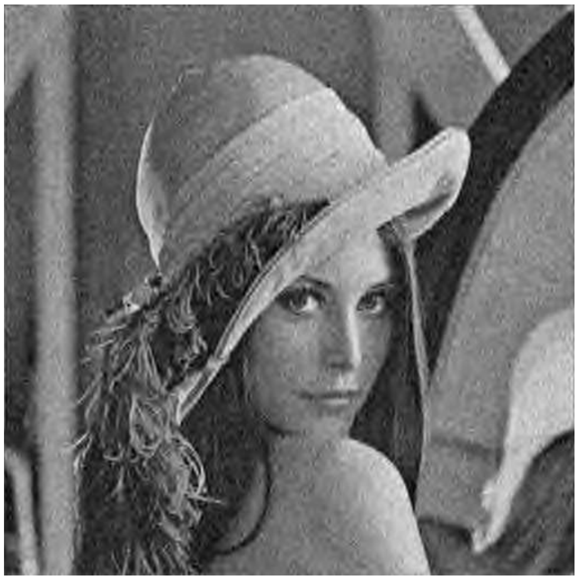
(a) Hard SureShrink thresholding

RMSE=9.66, PSNR=28.43.



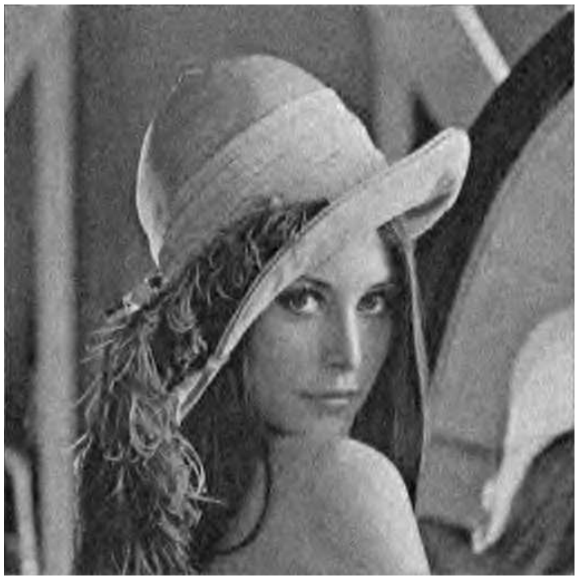
(b) Soft SureShrink thresholding

RMSE=9.01, PSNR=29.03.



(c) Hard BayesShrink thresholding

RMSE=10.02, PSNR=28.11.



(d) Soft BayesShrink thresholding

RMSE=9.02, PSNR=29.02.

Figure 3.18: Results of applying the *context-based hard* and *soft* thresholding using 3×3 masks for the BayesShrink and the SureShrink methods.

	Traditional Thresholding				Context-Based Thresholding			
	Hard		Soft		Hard		Soft	
	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR
VisuShrink	12.37	26.28	15.76	24.18	10.21	27.95	14.16	25.11
LevelShrink	10.01	28.11	11.30	27.07	*9.37*	*28.69*	10.07	28.07
SureShrink	9.89	28.23	9.96	28.16	9.66	28.43	9.01	29.03
BayesShrink	10.07	28.07	9.93	28.19	10.02	28.11	9.02	29.02

Table 3.5: Comparison between the results obtained using the various wavelet thresholding methods, using the traditional as well as the context-based soft and hard thresholding operators.

- For SureShrink and BayesShrink, note that the gain stemming from the use of the context-based thresholding operators is much more visible for the soft thresholding operators than for the hard thresholding one. This is probably because the optimal thresholds corresponding to these methods were originally derived for the purpose of soft thresholding. In fact, using hard thresholding for SureShrink or BayesShrink adopts an ad-hoc method where the optimal hard threshold is taken to be twice the value of the optimal soft threshold for these methods.

- The improvement achieved by the proposed context-based thresholding operators is more evident for the case of the VisuShrink and LevelShrink than it is for the SureShrink and BayesShrink methods. This is probably the case because the "optimal" thresholds for the latter two methods were proposed for thresholding using the conventional soft thresholding operator, as defined in (3.5). Thus, using these thresholds when applying the new thresholds does not result in significant improvement.

Next, the cycle spinning algorithm will be incorporated for the purpose of improving the quality of the denoised estimates obtained by the above context-based thresholding methods.

3.4.3 Enhancing Context-Based Thresholding via Cycle Spinning

The cycle spinning algorithm was incorporated in order to enhance the denoised estimates obtained by the various modified wavelet thresholding methods which apply the new context-based thresholding operators. Figures 3.19 - 3.22 illustrate the results obtained by using $K = 16$ shifts. As before, note that quality of the denoised images is improved by using the cycle spinning methods.

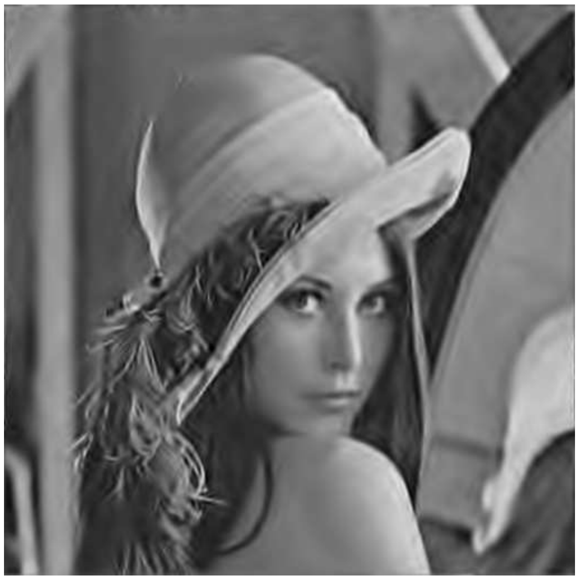
Table 3.6 illustrates the results obtained by using cycle spinning idea when the traditional as well as context-dependent thresholding operators are used, for the various methods. Note that the best overall result is obtained when using the context-based hard thresholding for the LevelShrink method along with the cycle spinning method. Clearly, the use of the proposed context-based thresholding operators yields better results than using the conventional hard and soft thresholding operators before and after incorporating the cycle spinning idea. Again, note that the improvement, resulting from the use of the context-based thresholding, is more evident when using soft thresholding especially in the case of the VisuShrink and the LevelShrink thresholding methods. As explained earlier, this is the case because the latter two methods use "optimal" thresholds that are derived for the purpose of applying the conventional soft thresholding operator. Figures 3.23 and 3.24 also illustrate how the quality of denoised image improves with the number of shifts for the various thresholding methods when using traditional and context-based thresholding. Note that the quality of the denoised estimate improves rapidly for the first few shifts. However, after a relatively small number of shifts, the quality of the image becomes more stable and little further improvement is achieved by increasing the number of shifts even further.

	Traditional Thresholding				Context-Based Thresholding			
	Hard		Soft		Hard		Soft	
	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR	RMSE	PSNR
VisuShrink	10.27	27.90	14.94	24.64	8.87	29.18	12.76	26.01
LevelShrink	8.34	29.70	10.61	27.61	*8.06*	*30.00*	8.71	29.33
SureShrink	8.39	29.65	8.73	29.31	8.38	29.67	8.37	29.67
BayesShrink	8.64	29.40	8.66	29.38	8.71	29.33	8.36	29.69

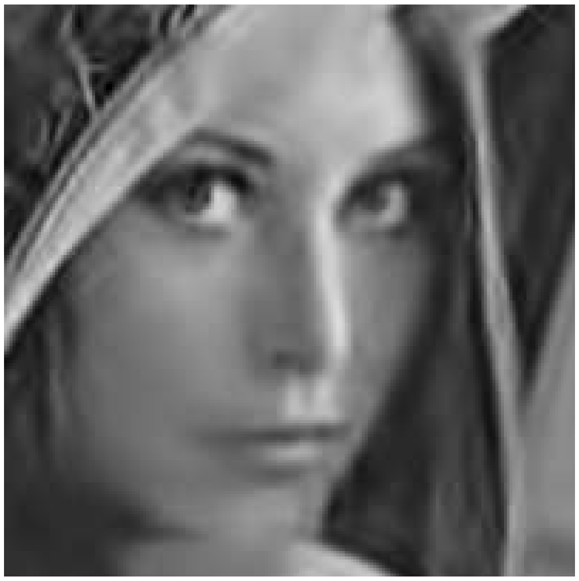
Table 3.6: Comparison between the results obtained by the various wavelet thresholding methods, using traditional and context-based soft and hard thresholding as well the cycle spinning idea with $K = 16$ shifts.

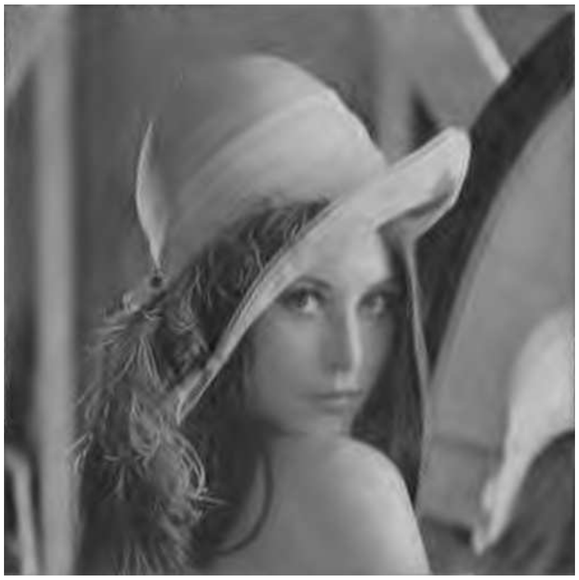
3.5 Summary and Concluding Remarks

In this chapter, a few standard wavelet thresholding methods were reviewed, implemented and compared. These use of the cycle spinning algorithm for the purpose of reducing the Gibbs artifacts that tend to be present in the denoised estimates was illustrated.


(a) Hard VisuShrink thresholding

RMSE=8.87, PSNR=29.18.


(b) Zooming in on the image in (a)


(c) Soft VisuShrink thresholding

RMSE=12.76, PSNR=26.01.

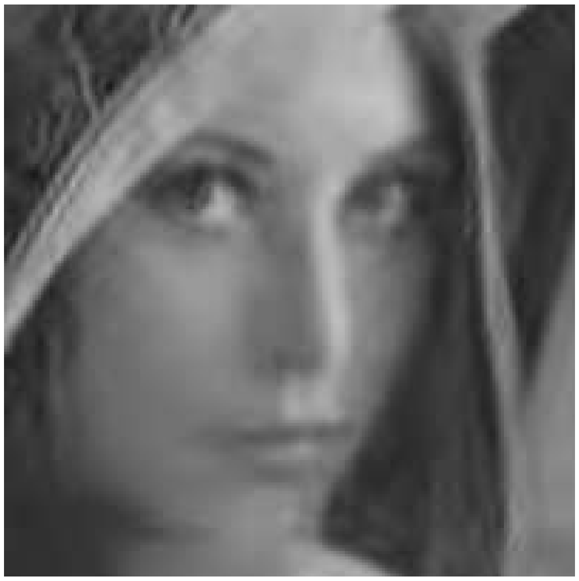
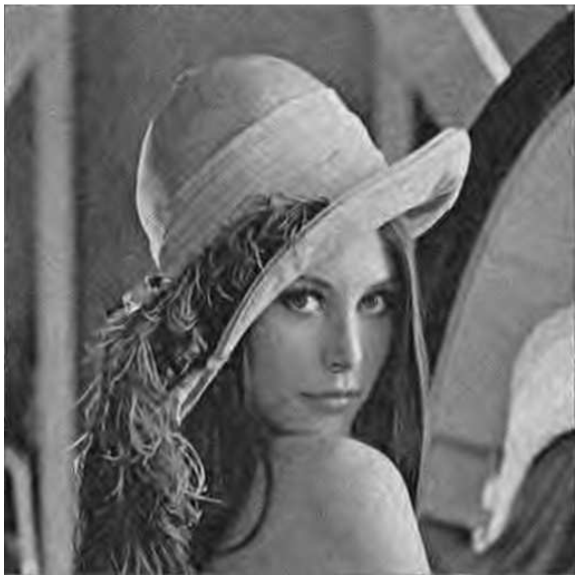
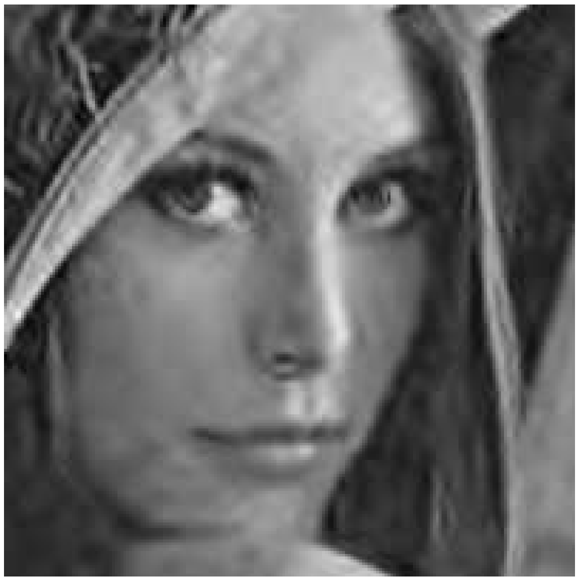
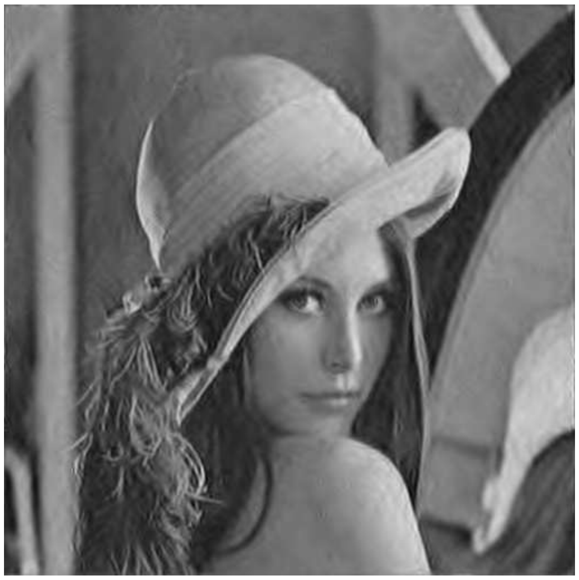

(d) Zooming in on the image in (c)

Figure 3.19: Results of applying the context-based VisuShrink with *hard* and *soft* thresholding using cycle spinning with *K=16* diagonal shifts.


(a) Hard LevelShrink thresholding RMSE=8.06, PSNR=30.00.


(b) Zooming in on the image in (a)


(c) Soft LevelShrink thresholding RMSE=8.71, PSNR=29.33.

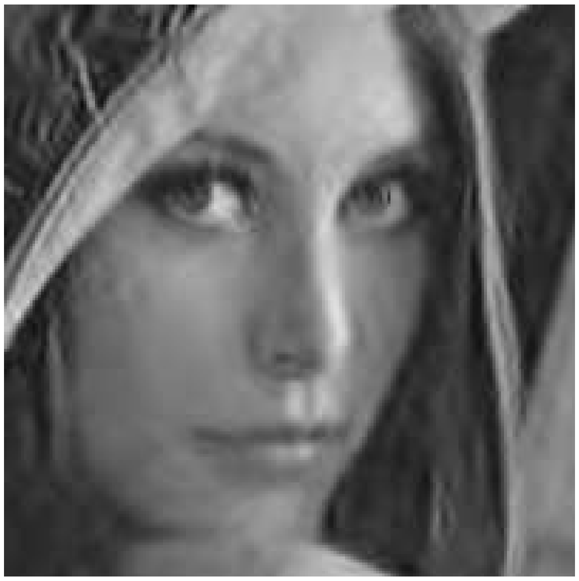
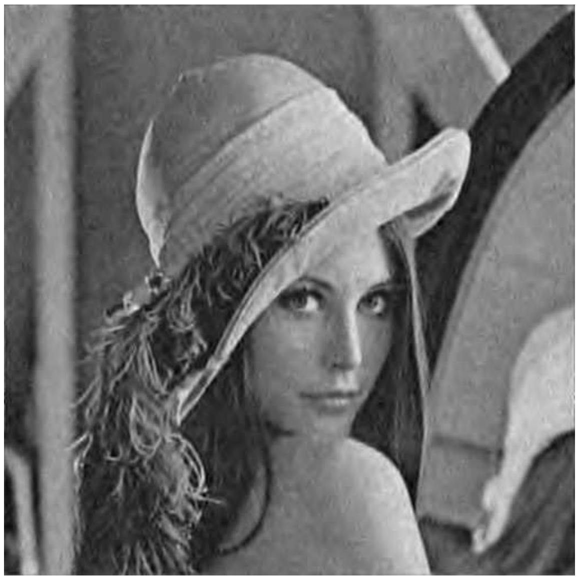
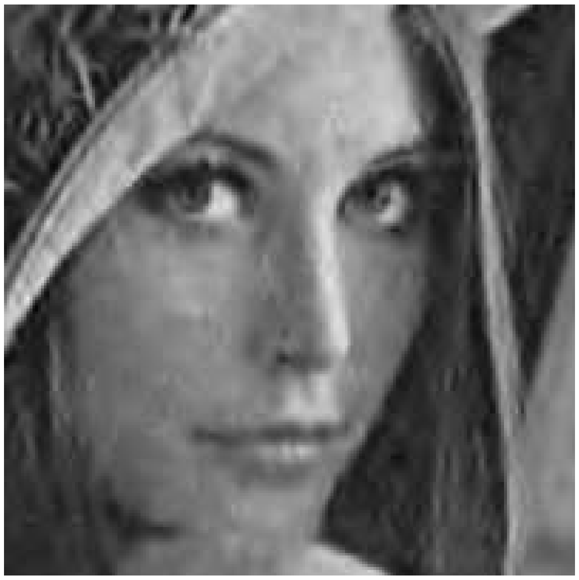
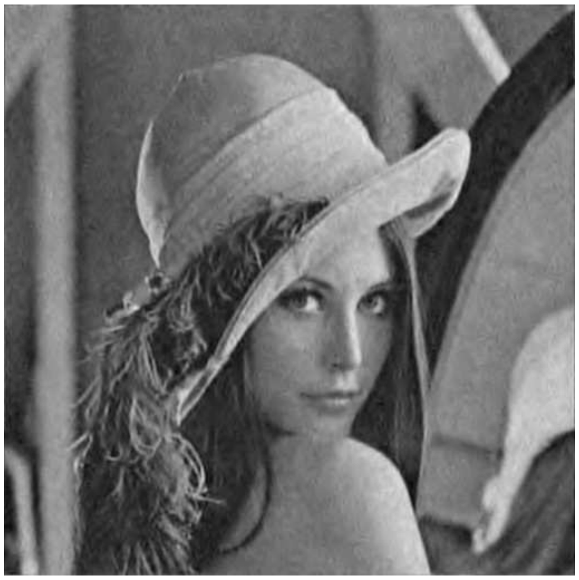

(d) Zooming in on the image in (c)

Figure 3.20: Results of applying the context-based LevelShrink *hard* and *soft* thresholding using cycle spinning with *K=16* diagonal shifts.


(a) Hard SureShrink thresholding
RMSE=8.38, PSNR=29.67.


(b) Zooming in on the image in (a)


(c) Soft SureShrink thresholding
RMSE=8.37, PSNR=29.67.

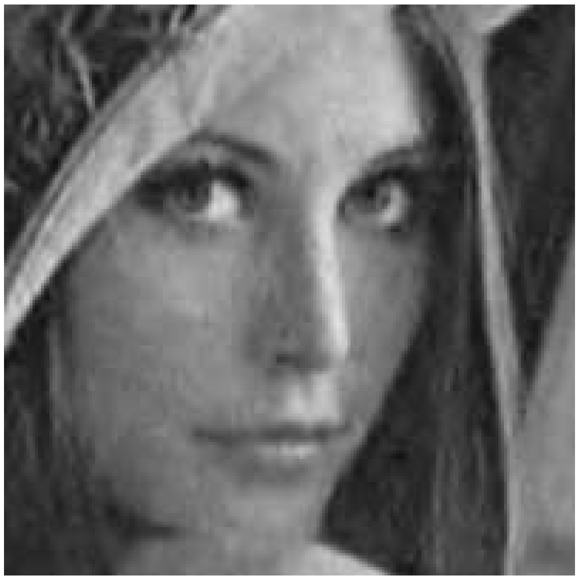
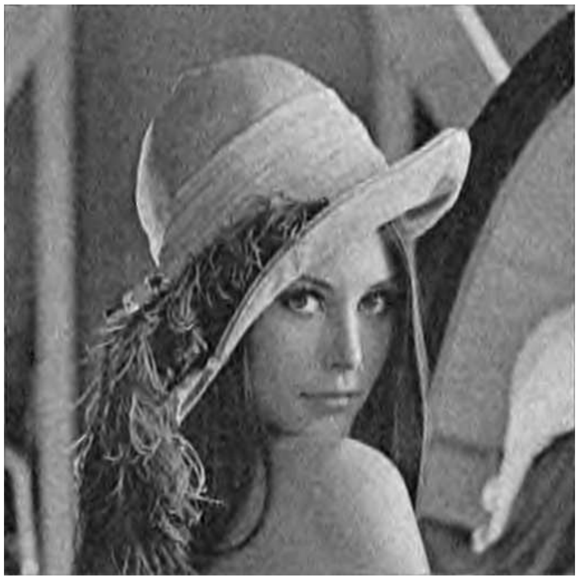
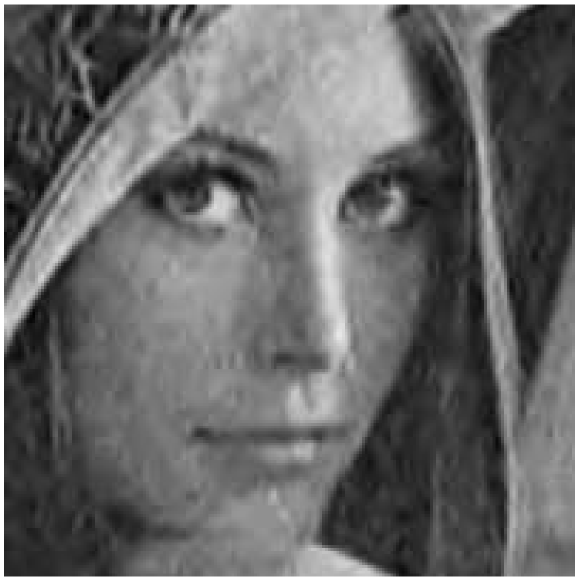

(d) Zooming in on the image in (c)

Figure 3.21: Results of applying the context-based SureShrink with *hard* and *soft* thresholding using cycle spinning with *16* diagonal shifts.



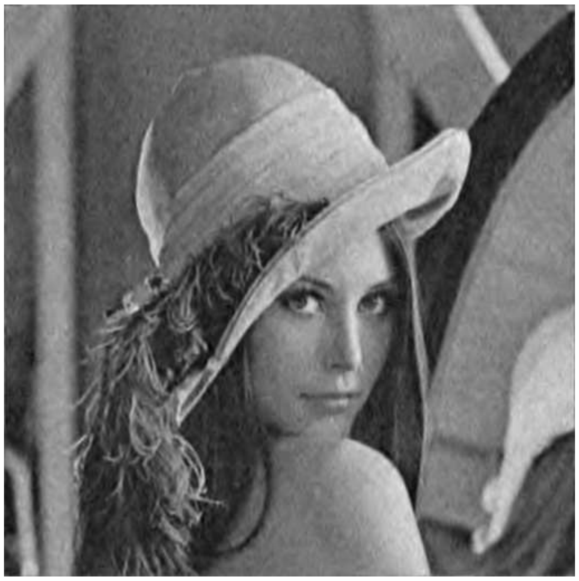
(a) Hard BayesShrink thresholding: K=16

RMSE=8.71, PSNR=29.33.



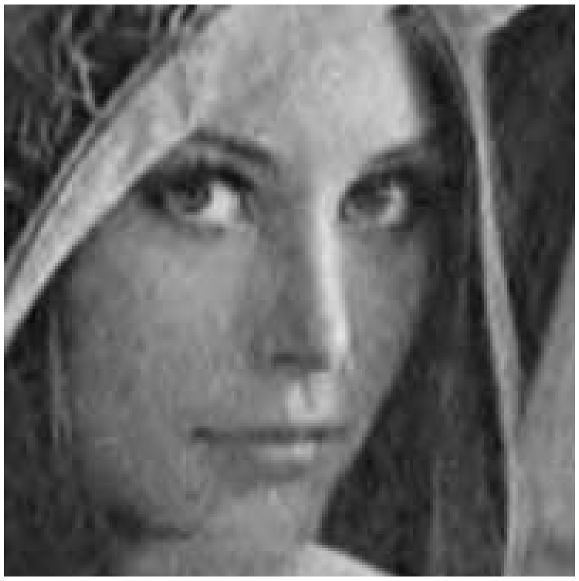
(b) Zooming in on the image in (a)

RMSE=8.36, PSNR=29.69.



(c) Soft BayesShrink thresholding

RMSE=8.36, PSNR=29.69.



(d) Zooming in on the image in (c)

Figure 3.22: Results of applying the context-based BayesShrink with *hard* and *soft* thresholding using cycle spinning with *16* diagonal shifts.

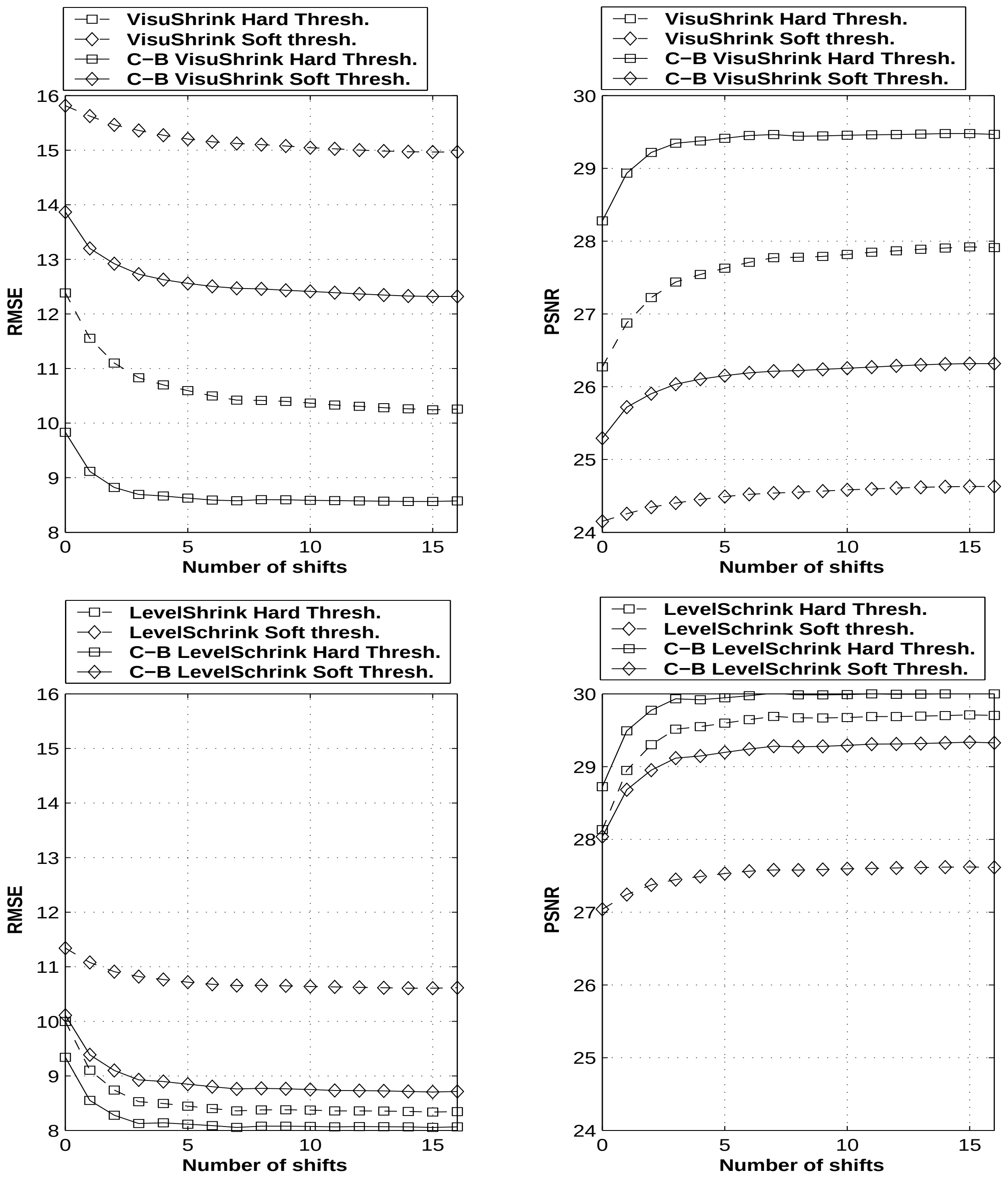


Figure 3.23: Comparison between the conventional and context-based (C-B) VisuShrink and LevelShrink thresholding methods when applying the idea of the cycle spinning with *K=16* shifts.

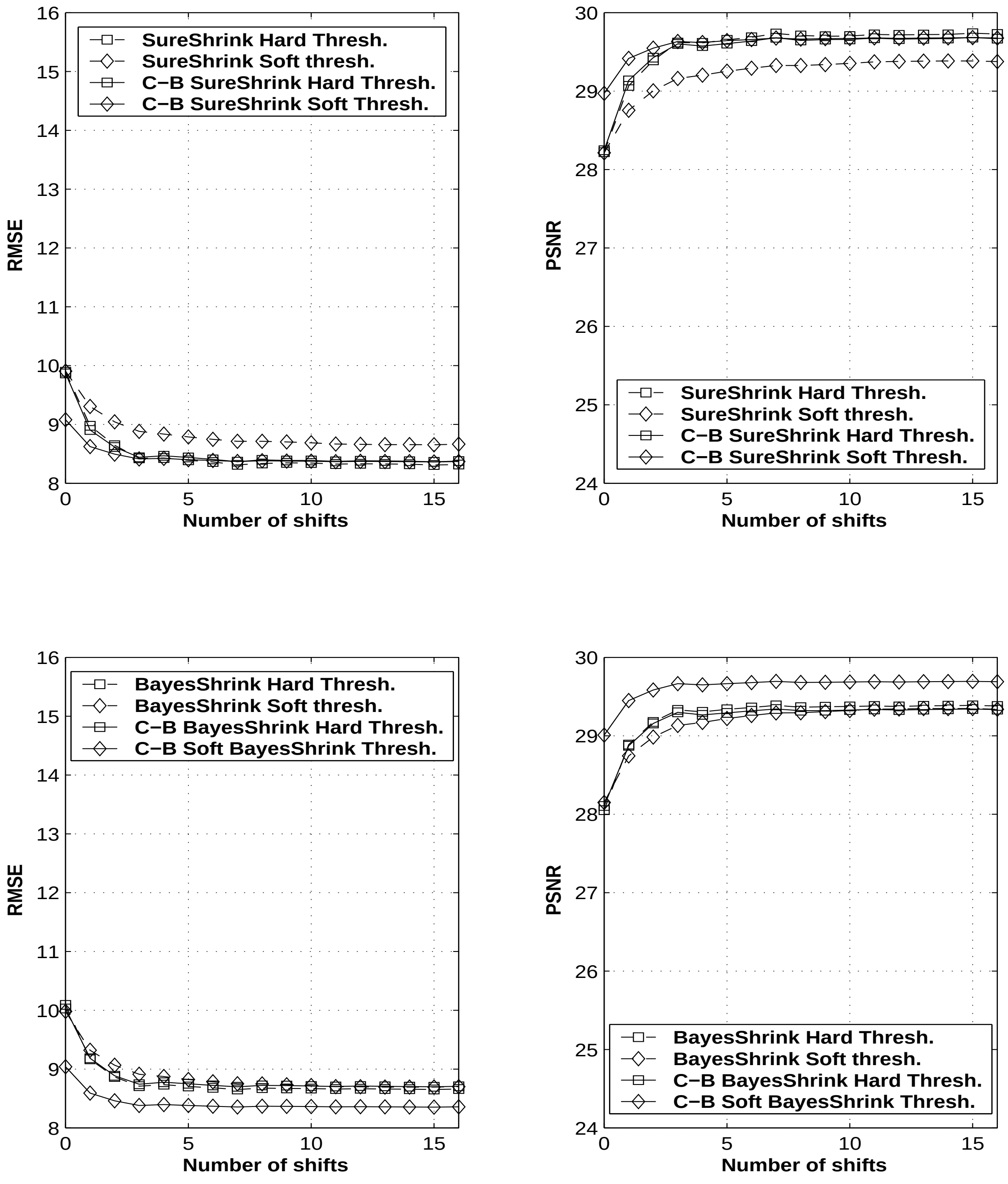


Figure 3.24: Comparison between the conventional and context-based (C-B) SureShrink and BayesShrink thresholding methods when applying the idea of the cycle spinning with *K=16* shifts.

Furthermore, the use of context-based and localized soft and hard thresholding operators that takes into consideration the values of the neighboring coefficient before each wavelet coefficient is thresholded, was proposed and implemented. It was shown that the use of these adaptive thresholding operators is indeed beneficial resulting in an improvement of the quality of the denoised estimates especially for the VisuShrink and LevelShrink thresholding methods. These improvements lie in obtaining denoised estimates that are sharper and less blurry than the results obtained by the traditional thresholding operators. Overall, it was found that the resulting gains are more evident for the soft-thresholding strategy than when the hard thresholding operator is used. The use of the cycle spinning idea with this new thresholding strategy has illustrated the benefits of using these context-based thresholding operators even further and shown that restoration and enhancement of the noisy image can be achieved by combining the use of the context-based thresholding with cycle spinning idea.

In the next chapter, the potential of applying fractal-based image coding methods for the purpose of image denoising will be explored.

Chapter 4

Fractal Image Denoising

In this chapter the potential of applying standard fractal methods for the purpose of image denoising will be explored. As detailed in chapter 2, fractal image coding has received much interest over the past decade, mostly in the context of image compression. However, little or no attention has been given to the use of such fractal-based methods for the purpose of image enhancement and restoration. Indeed, one of the original motivations for this study was the observation that a noisy image is somewhat denoised when it is fractally coded. This led to the question of whether such a simple fractal encoding of the *noisy* image could be used as a starting point to estimate the fractal code of the *noiseless* image, perhaps with some knowledge of the noise, e.g., its variance. One can then use this fractal code to reconstruct a denoised estimate of the original image. This question will be examined in this chapter and the answer will be shown to be in the affirmative.

First, straightforward fractal-based coding of the noisy image is shown to perform rather well as a denoiser. In retrospect, this is not surprising since the (white Gaussian) noise process is not represented well by the (local) linear transform that maps parent blocks to child blocks, hence resulting in noise reduction. Indeed, this is essentially the basis of the local linear minimum mean squared error Lee filter [49] so that fractal coding may be considered to be closely related to Lee filtering. Also, fractal-based schemes exploit local and global self-similarities that are inherent in many classes of real-world images. Natural image structures possess similarities across other parts of the image which can be exploited for fractal image coding. However, noisy structures have no resemblance in other parts of the image and therefore cannot be accurately encoded using fractal coders.

This initial investigative step is followed by proposing a simple, yet effective scheme of predicting the fractal code of the original image from the noisy one. Given a noisy image, it will be shown that its fractal code parameters – in particular, the gray-level map coefficients – can be used to estimate those of its *noiseless* counterpart, assuming that one knows (or can estimate) the variance of the (white Gaussian) noise. This leads to an improvement in the fractal approximation to a target image u. It will be shown that this method performs in a manner similar to that of the human visual system, producing extra smoothing in flat, low activity regions and a lower degree of smoothing near high activity regions, including edges. The use of the cycle spinning idea for the purpose of reducing the blockiness artifacts that are inherent in standard fractal image representations will also be explored.

The layout of this chapter is as follows: First, image denoising through simple fractal coding will be illustrated. In section 2, a theoretical relationship between the fractal code of a noisy image and the fractal code of its noiseless counterpart, will be derived. An outline of a scheme to predict the true fractal code of the noiseless image will also be given. In section 3, the cycle spinning algorithm will be implemented in order to improve the quality of the denoised estimates obtained by the various fractal denoising schemes.

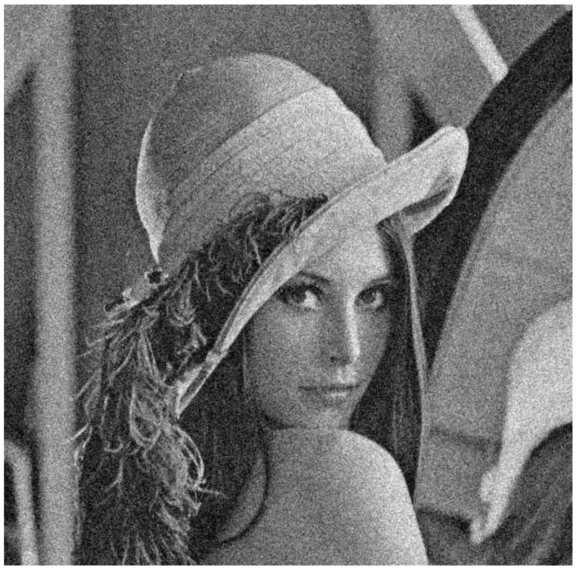
4.1 Image Denoising using Simple Fractal Coding

In this section, the effects of simply encoding a noisy image using a spatially based fractal scheme are investigated. The noisy image used here is the same test image of "Lenna" which has been degraded by an additive white Gaussian noise (AWGN) with standard deviation $\sigma_{\mathbf{w}} = 25$.

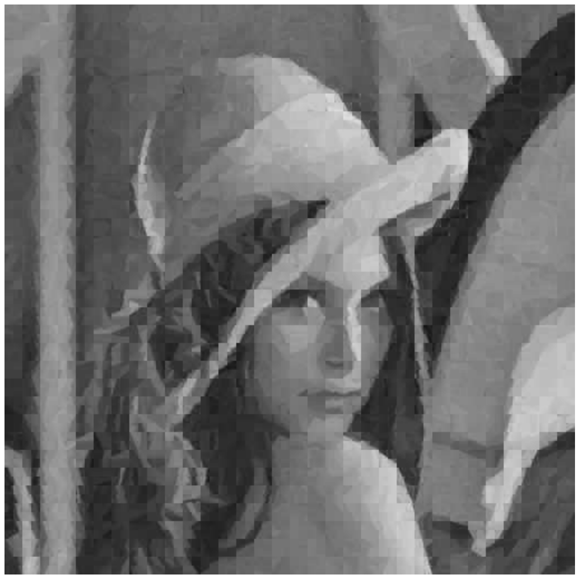
4.1.1 Fractal Coding using Uniform Partitioning

As described in chapter 2, the standard fractal scheme performs a uniform partitioning of the image for the purpose of fractal image coding. This scheme is used to encode the noisy test image of "Lenna" for various fractal resolutions (M, N), and the results are illustrated in Figure 4.1. These results are summarized in the following observations:

- The fractal representation at the resolution $(M, N) = (32, 64)$ is the best, quantitatively, as reflected by the RMSE and PSNR fidelity measures. However, this approximation is clearly not as good as the one obtained by simply fractally encoding the noiseless image, using the same fractal scheme and the same fractal resolution, as illustrated in Figure 2.5 (c). This shows that although a significant degree of restoration and enhancement of the noisy image

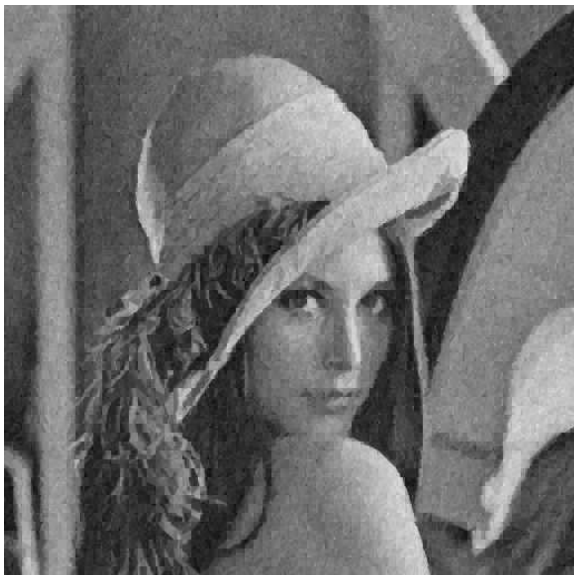

(a) Noisy image: $\sigma_{\mathbf{w}} = 25$

RMSE=25.01, PSNR=20.17.


(b) Fractal encoding: (M,N)=(16,32)

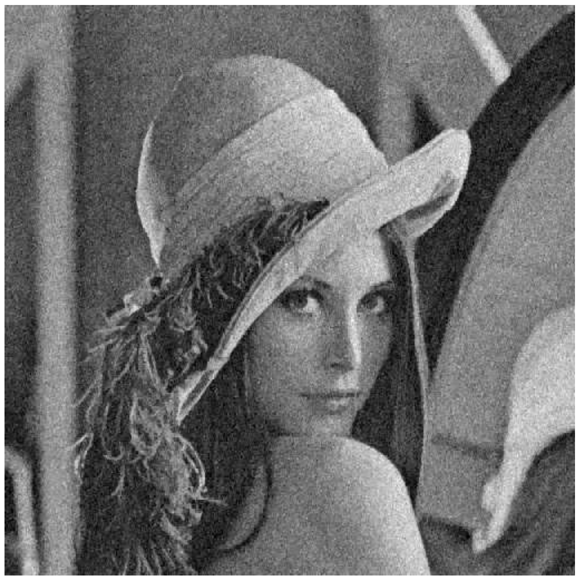
RMSE=13.94, PSNR=25.24.

Execution time $\approx$ 302 secs.


(c) Fractal encoding: (M,N)=(32,64)

RMSE=11.56, PSNR=26.87.

Execution time $\approx$ 1145 secs.


(d) Fractal encoding: (M,N)=(64,128)

RMSE=15.48, PSNR=25.59.

Execution time $\approx$ 4803 secs.

Figure 4.1: (a) The original noisy image with noise standard deviation $\sigma_{\mathbf{w}} = 25$ and (b)-(d) the standard fractal representations of the noisy image of "Lenna" for various fractal resolutions (M, N). Note that the gray-level coefficients were not quantized.

has been achieved, the presence of noise has significantly affected the fractal code, resulting in a less accurate fractal representation of the original image.

- For the lower resolution, $(M, N) = (16, 32)$, most of the noise has been suppressed at the expense of significant blockiness artifacts, over-smoothness and degradation of the sharpness of edges.

- For the higher resolution, $(M, N) = (64, 128)$, the fractal representation appears rather noisy. This is the case because, for this resolution, the child blocks are of size 4×4 pixels and thus one is fitting 16 neighboring pixels in each child block which tend to be more correlated. Hence a better fit can be achieved for small child blocks, resulting in more reconstruction of the noise. Although the fractal approximation appears noisy, some degree of the noise has been suppressed even at this resolution. This is evident when comparing the fractal reconstruction to the original noisy, and it is also reflected by the fidelity measures.

Clearly, there is a trade-off between the fractal resolution and the quality of the fractal representation. A lower resolution results in smoothing most of the noise at the expense of over-smoothing of edges and blockiness artifacts. On the other hand, a higher resolution results in a greater reconstruction of the noise, while preserving the high frequency content (edges) of the image.

One way to exploit this trade-off between the partition size and the reconstruction of the noise is to employ a quadtree partitioning strategy for the purpose of fractal image coding. Hence, the image partitioning is content-dependent. On one hand, the smoother regions of the original noiseless image tend to be dominated by the noise, so one can partition the image coarsely, hence resulting in considerable reduction of the noise. On the other hand, within regions of high activity, such as edges, one can adopt a less coarse (finer) partition hence preserving the sharpness of these features while suppressing some of the noise as well. This process is described next in more detail.

4.1.2 Fractal Coding using Quadtree Image Partitioning

In principle, the human visual system is less sensitive to the presence of noise near edges and other high activity subregions of the image, while noise in flat subregions is more perceivable. Human observers tend to subjectively prefer sharper images with little noise over blurred noiseless images. In fact, psychophysical studies have shown that human observers are less sensitive to random noise variations in the vicinity of strong edges than in constant signal regions, because high local contrast masks the nearby noise [62]. This motivated the investigation of the the use of the quadtree partitioning scheme which allows for adapting the image partitioning to its content by using different fractal resolutions or block sizes for different parts of the image. In particular, one expects that finer resolutions (i.e. smaller sub-blocks) will be used near edges and other high activity areas. From the previous section, these areas will yield noisy fractal representations while preserving the edges well. The presence of an acceptable amount of noise in these high activity subregions of the fractal representation may not be a problem due to the fact that the human visual system is less sensitive to noise near edges. On the other hand, it is expected that a more coarse partitioning will be suitable for flat and low activity regions, resulting in a higher degree of smoothing and denoising.

Quadtree Decomposition Criterion

As described in chapter 2, quadtree-based fractal coding adopts an adaptive, image-dependent partitioning strategy in which square child sub-blocks are either fractally encoded or broken down into four quadrants in a recursive tree structure. Various quadtree decomposition criteria were investigated in [33], and it was found that the child block variance is the optimal decomposition criterion for the purpose of fractal image compression.

For the purpose of fractally encoding and restoring a noisy image, a *signal-to-noise ratio* (SNR) quadtree decomposition criterion will be used. The SNR, γ, of a noise-free child sub-block $\mathbf{Y}$, is computed as

$$\gamma = \frac{\sigma^2_{\mathbf{Y}}}{\sigma^2_{\mathbf{w}}}. \tag{4.1}$$

However, the noise-free child block $\mathbf{Y}$ is not available and only its noisy version, $\hat{\mathbf{Y}}$, defined by

$$\hat{\mathbf{Y}} = \mathbf{Y} + \mathbf{w} \tag{4.2}$$

can be observed. Thus, under the assumption that the original image $\mathbf{u}$ and the noise $\mathbf{w}$ are statistically independent, the following relationship can be established:

$$\sigma^2_{\hat{\mathbf{Y}}} = \sigma^2_{\mathbf{Y}} + \sigma^2_{\mathbf{w}} \text{ or equivalently } \sigma^2_{\mathbf{Y}} = \sigma^2_{\hat{\mathbf{Y}}} - \sigma^2_{\mathbf{w}}. \tag{4.3}$$

Consequently, the SNR γ can be estimated from the noisy child sub-block $\hat{\mathbf{Y}}$ and the noise variance $\sigma^2_{\mathbf{w}}$ as follows:

$$\begin{aligned} \gamma &= \frac{\sigma^2_{\hat{\mathbf{Y}}} - \sigma^2_{\mathbf{w}}}{\sigma^2_{\mathbf{w}}} \\ &= \frac{\sigma^2_{\hat{\mathbf{Y}}}}{\sigma^2_{\mathbf{w}}} - 1. \end{aligned} \tag{4.4}$$

However, in practice, this quantity may be negative when $\sigma^2_{\hat{\mathbf{Y}}} < \sigma^2_{\mathbf{w}}$, thus a more practical estimate of the SNR is given by

$$\gamma = \max\{\frac{\sigma^2_{\hat{\mathbf{Y}}}}{\sigma^2_{\mathbf{w}}} - 1, 0\}. \tag{4.5}$$

Since higher SNR values reflect the existence of edges or other high pixel variability within the sub-block, then a sub-block is partitioned into four quadtrees if its SNR, γ, exceeds a prescribed threshold γ_c. Otherwise, if a sub-block has a small SNR, then it is assumed that it is dominated by noise and fractally encoding this sub-block will result in significant noise reduction, especially for larger block sizes.

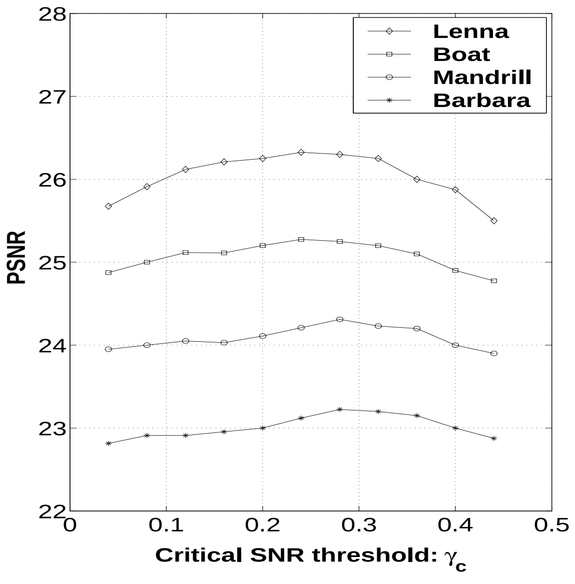


Figure 4.2: Distortion curves illustrating the quality of the fractal representations as a function of the SNR threshold, for the noisy images of "Lenna", "Barbara", "Boat" and "Mandrill", with $\sigma^2_{\mathbf{w}} = 625$. The optimal threshold for "Lenna" is seen to be $\gamma_c \approx 0.25$.

Figure 4.2 illustrates the relationship between the quality of the quadtree-based fractal representations and the signal to noise threshold, γ_c, for the the noisy images of "Lenna", "Barbara", "Boat" and "Mandrill", with noise variance $\sigma^2_{\mathbf{w}} = 625$. Note that there is a trade-off between the quadtree-resolution and the reconstruction of the noise. On the one hand, a higher SNR threshold of γ_c results in coarse quadtree partitioning which, in turn, produces smoother yet coarser fractal representations of the image. On the other hand, a lower SNR threshold yields a finer quadtree partitioning, resulting in finer but noisier fractal representations. For instance, this figure also shows that there is an optimal critical SNR threshold, $\gamma_c \approx 0.25$, for "Lenna". This represents an optimal trade-off between the quadtree-based fractal resolution and the noise reconstruction.

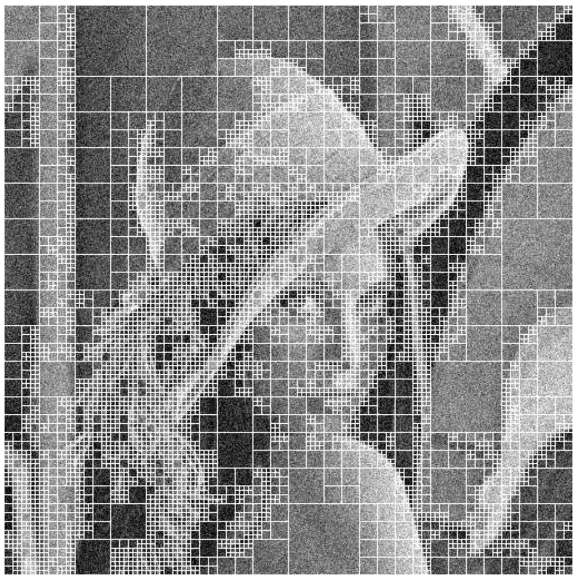
Whenever using the quadtree partitioning algorithm for the purpose of fractal image coding, one has to choose a threshold for the decomposition criterion. In this case, one has to determine the "optimal" value, γ_c, for the SNR threshold decomposition criterion. As illustrated in Figure 4.2, note that quality of the various fractal representations is rather stable and is not highly sensitive to various values of γ_c, for $0 < \gamma_c \leq 0.45$. This is the case especially for the Barbara and Mandrill images. For the image of interest, "Lenna", note that, $25.6 \leq \text{PSNR} \leq 26.3$, whenever $0 < \gamma_c \leq 0.5$, with an optimal value for the SNR threshold is $\gamma_c \approx 0.25$. From Eq. (4.4), this seems to indicate that one partitions a noisy image sub-block $\hat{\mathbf{Y}}$ if

$$\gamma \geq 0.25 \text{ or equivalently } \sigma^2_{\hat{\mathbf{Y}}} \geq 1.25\sigma^2_{\mathbf{w}}. \tag{4.6}$$

This decomposition criterion also seems reasonable for the other test images as well. In general, one expects the optimal value for γ_c to depend on the image and the noise level, so one can, experimentally, generate lookup tables for classes of images and various noise levels. Thus, γ_c, can be viewed as a denoising *fine-tuning* parameter that measures the trade-off between suppressing the noise and reconstructing the high frequency content and important features of the image, such as edges. This is somewhat similar to the idea of using fine-tuning parameters adopted in JPEG, which are obtained from lookup tables. These parameters are used to visually enhance the quality of the image representation.

Experimental Results

Figure 4.3 illustrates the quadtree segmentation of the noisy image of "Lenna" as well as the fractal representation of the image using SNR threshold $\gamma_c = 0.25$. Note that the fractal representation has a relatively high RMSE (or low PSNR) due to the presence of noise near edges – a careful observation of the fractal representation will reveal such noise. However, the edges remain sharp, in contrast to the uniform partitioning case, which results in over smoothing throughout the denoised estimate. Flat regions of the image, such as the shoulder, face and background, are relatively smooth – most of the noise in these regions has been suppressed in the fractal representation. It is important to mention that the quadtree-based fractal representation is indeed visually better than the qualitative measures (i.e. RMSE and PSNR) seem to indicate. This is because, as mentioned previously, most of the remaining noise is localized in the vicinity of edges and other high-frequency content of the image where the human visual system is less sensitive to noise. Hence the noise is less noticeable. However, the quadtree-based fractal representation suffers from blockiness artifacts which is more visible due to the non-uniform nature of the quadtree partitioning.


(a) Quadtree partitioning of the noisy image

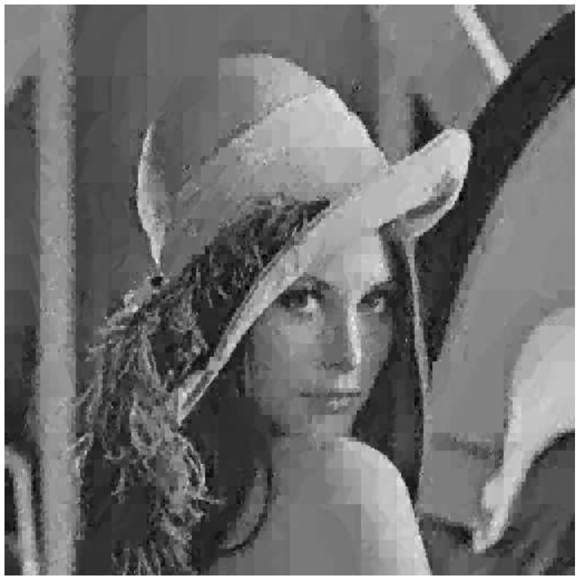

(b) Quadtree-based fractal denoising
RMSE=12.35, PSNR=26.30.
Execution time $\approx$ 257 secs.

Figure 4.3: Standard fractal image restoration using quadtree partitioning using the optimal SNR threshold, $\gamma_c = 0.25$, for the noisy "Lenna" with $\sigma^2_{\mathbf{w}} = 625$.

In summary, the quadtree partitioning based fractal scheme permits more smoothing away from edges and lesser smoothing near edges, resulting in restoring the noisy image without significantly degrading its edges. This scheme also yields results that are consistent with the human visual system which is more sensitive to the presence of noise in flat regions than near edges of the image.

Next, the noisy image is encoded using the search free Bath fractal transform, described in chapter 2.

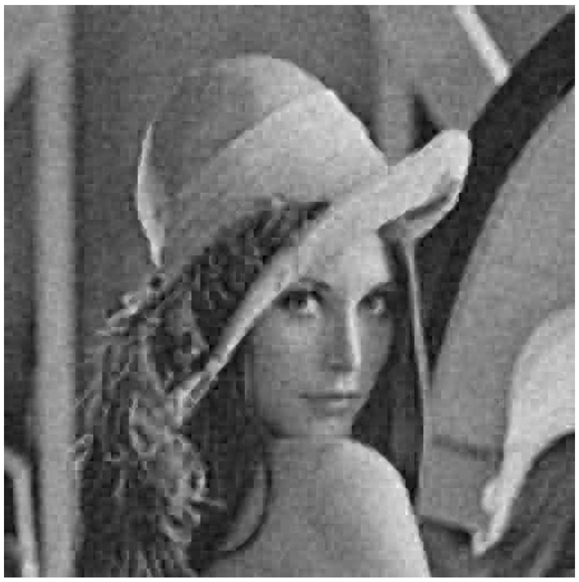
4.1.3 The Bath Fractal Transform

As described in chapter 2, the Bath fractal transform is a search-free fractal scheme that matches each child block with its "co-centric" parent block. This scheme was applied for the purpose of coding the noisy test image of "Lenna" and the results are illustrated in Figure 4.4. Surprisingly, the Bath fractal transform seems to perform almost the same as the standard fractal coding scheme, when using uniform partitioning with $(M, N) = (32, 64)$.

Recall that the Bath fractal transform scheme uses place-dependent gray-level maps of the form

$$\phi(\hat{u}(x,y),x,y) = \alpha\hat{u}(x,y) + \beta + \gamma_x x + \gamma_y y,$$

where the term $\hat{u}(x,y)$ represents the gray-level value of a pixel located at (x,y) in the noisy image. Note that while the gray-level value $\hat{u}(x,y)$ is affected by the additive noise, the location of the pixel, (x,y), is not. Hence, one would expect that the coefficients γ_x, γ_y are not significantly affected by the presence of the noise. This makes the fractal code of the noisy image closer to the fractal code of the noiseless image, resulting in a significantly denoised image. The main advantage of using the BFT scheme is that it is computationally less expensive than the standard fractal scheme. This makes the application of BFT coding scheme for the purpose of image denoising appealing since this scheme is not only computationally inexpensive but also yields denoised estimates that are comparable to the results obtained by exhaustive fractal coding schemes.



Bath fractal coding: RMSE=11.62, PSNR=26.83

Execution time $\approx$ 7 secs.

Figure 4.4: Bath fractal coding of the noisy image of "Lenna".

Next, some of the reasons why fractally encoding a noisy image actually results in significant noise reduction are examined.

4.1.4 Why Does Fractal Coding Result in Denoising?

In the previous section, it was shown that by simply fractally encoding the noisy test image, using any of the fractal-based schemes, one may generally achieve a significant degree of noise reduction. As explained earlier, one of the main reasons for achieving noise reduction through fractal coding is that natural self-similar structures within natural images are generally reconstructed well through fractal coding, whereas the noisy contents cannot be described or approximated well by the fractal transform. In this section, some of the other reasons behind achieving noise reduction through fractal coding are explored.

Fractal Representation of Pure Noise

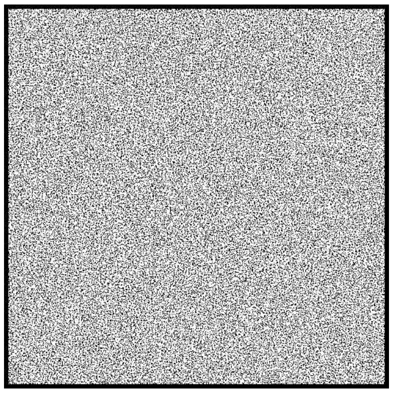
A simple experiment that might help answer the above question involves simply fractally encoding a purely noisy image using a fractal-based scheme. The standard fractal scheme with resolution $(M, N) = (32, 64)$ shall be used. As illustrated in Figure 4.5, a purely noisy image (AWGN) with noise variance $\sigma_{\mathbf{w}}^2 = 25^2$ has been encoded using the standard fractal scheme. The fractally encoded image appears significantly less noisy as compared to the original purely noisy image. This observation is better illustrated through the histogram of the "gray-level" values of the fractally encoded image which are now much closer to zero and occupy a significantly smaller range than the case for the noisy image. This figure also illustrates the gray-level coefficients corresponding to the fractal transform of the pure noise. Note the distribution of the scaling coefficients, α, is bi-modal with all scaling coefficients being non-zero. This can be explained by the fact that when fractal coding a noise-free image, flat regions yield almost constant child blocks which in turn result in zero (or close to zero) scaling coefficients, α. However, a noisy image has no more flat regions, since smooth regions in the original image are now dominated by the noise. Thus, the resulting child blocks in the partition are non-uniform and the corresponding scaling coefficients, α, are going to be non-zero. This explains the shape of the distribution of the scaling coefficients in Figure 4.5 (e).

Another important source of noise reduction lies in the decimation mapping employed by the fractal transform, as discussed next.

Decimation of the Domain Blocks

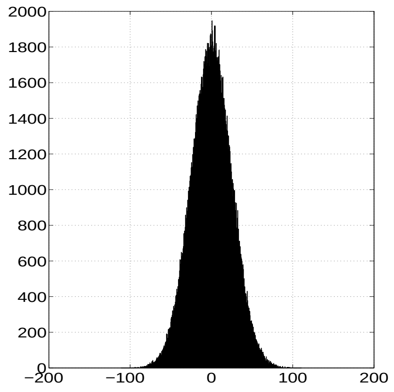
As discussed in detail in section 2.1, when fractally encoding a noisy image, the domain (parent) sub-block, $D_{i(k)}$, is first geometrically transformed into the range (child) sub-block, R_k, through a contractive mapping, $w_{ik}^{(m)}, 1, 2, \ldots, 8$, that involves reducing $D_{i(k)}$ to the same size as R_k and applying one of the eight isometries on the reduced domain sub-block to obtain a decimated and transformed parent sub-block, $\bar{D}_{ik}$. The gray-level values of the transformed domain block, $\bar{D}_{ik}$, are then mapped into the the gray-level values of the range block, R_k, through the gray-level map, ϕ.

In the discrete pixel space, the action of the geometric maps, $w_{ik}^{(m)}$, involve the shrinking of an $2n \times 2n$ parent pixel block to an $n \times n$ child pixel block. This can be achieved by some kind of reduction or *decimation* procedure. Typically, a sub-block of 2×2 neighboring pixels in the parent

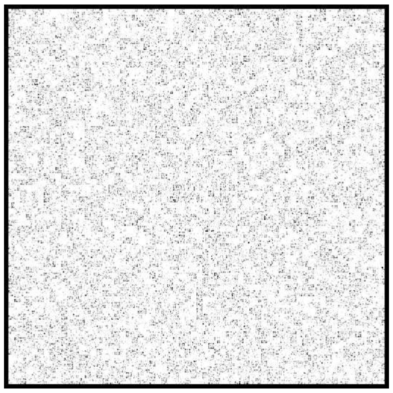


(a) Pure AWGN noise with $\sigma_{\mathbf{w}} = 25$

Negative of the image



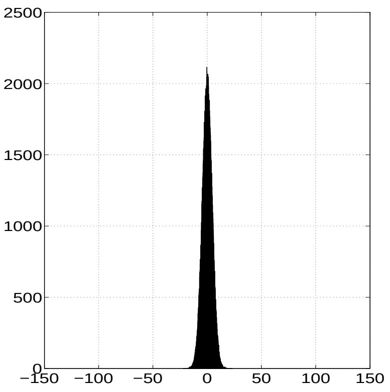
(b) Histogram of the pure noise



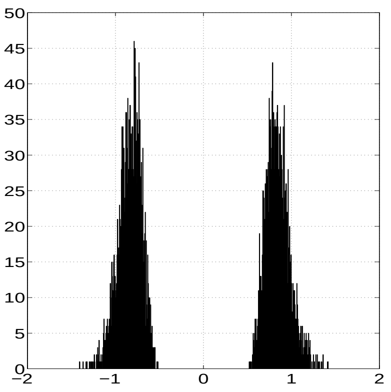
(c) Fractal reconstruction of the noise

Negative of the image

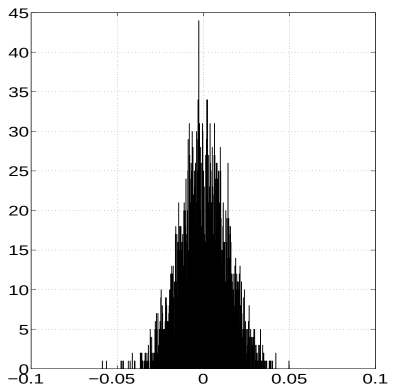
RMSE=5.57, PSNR=33.21



(d) Histogram of the fractally encoded noise.



(e) Histogram of α



(f) Histogram of β.

Figure 4.5: Fractal coding of pure AWGN noise with $\sigma_{\mathbf{w}} = 25^2$: note how fractally encoding the pure AWGN process results in an almost blank image with gray-level values spanning a much smaller range.

block is replaced by a single pixel and the four gray-level values are replaced by their average value. The averaging operation results in reduction of the noise variability, and hence noise suppression. Once the parent sub-block is geometrically transformed, then one is essentially mapping a reduced, smoothed and transformed version, $\bar{D}_{ik}$, of the original parent sub-block, $D_{i(k)}$, into the child sub-block, R_k, using an affine gray-level mapping, $\phi(t) = \alpha t + \beta$. Arguably, it is difficult to reconstruct the random in the unfiltered child block, R_k, from the smoothed and transformed parent sub-block, $D_{i(k)}$, using this simple first order gray-level map, ϕ. The decimation associated with the contractive geometric maps, $w_{ik}^{(m)}$, used in the fractal transform is probably responsible for much of the achieved fractal denoising.

Next, an apparent connection between the idea of fractal coding for image denoising and the Lee filter is explored further.

Connections Between the Fractal Transform and the Lee Filter

Recall that, similar to the fractal transform, the Lee filter also applies an affine point transformation that operates on single pixel values $\hat{u}(m, n)$, as given in Eq. (1.13). From Eq. (1.18), the gray-level coefficients of this transform are determined from the statistics of a 7×7 pixel block that is centered at the pixel (m, n) being processed. As such, the Lee filter may be viewed as a local affine (fractal) transformation of a 7×7 parent block onto the single-pixel child block that lies at its center. This latter action is somewhat reminiscent of the Bath Fractal Transform discussed in chapter 2, in which parent blocks are chosen to be co-centric with child blocks. The main difference is that for the Lee filter, the coefficients $\alpha_{m,n}$ and $\beta_{m,n}$, vary for each pixel. However, for the BFT, these coefficients are the same for all the pixels within the child-block. This establishes some degree of connection between the Bath Fractal Transform, as well as other fractal-based methods, and the Lee filter - a well known standard image denoising method.

Further Observations

In this section, some insights that explained the relatively good performance of fractal-based schemes as image denoising methods were explored. At this point, it is clear that by simply encoding a noisy image using any fractal scheme, one can achieve significant noise reduction. However, the results obtained by the best fractal-based image denoising scheme are surpassed by the results obtained by the standard denoising methods studied in chapter 1, namely the mean, Lee, Gaussian and the Wiener filters. This seems to suggest that fractal-based schemes do not present an alternative to standard image denoising methods. The reason why simply encoding a noisy image using a fractal-based scheme does not yield even better results than it did is mainly due to the following observations:

- Recall that for any spatially based fractal scheme, the fractal transform maps a parent block into a child block through a composition of a gray-level map ϕ and a geometric map w. When encoding a noisy image using a fractal scheme, one is mapping a noisy parent block into a noisy child block and thus some of the noise will indeed be reconstructed. This is indeed evident when applying the standard fractal scheme using a uniform partitioning with a high fractal resolution $(M, N) = (64, 128)$ where the fractal representation appears rather noisy, indicating that most of the noise has been reconstructed by the fractal transform.

- Also, the most negative effects of the noise on the fractal code lies in causing miss-matches between child blocks and their optimal parent blocks. In other words, due to the noise, a child block is not matched with its optimal, in the sense of minimizing the collage error for the original noiseless image, parent block. These miss-matches are the main roots for the relatively inadequate performance of fractal image coding schemes as image denoising methods.

Thus, in conclusion, encoding a noise image using a fractal-based scheme will result in a relatively denoised image where the degree of the reduction of the noise is inversely proportional to the fractal resolution (M, N). For low resolution, extra smoothing at the expense of disturbing blockiness and artifacts and degradation of sharp features of the image. For high resolution, little noise smoothing is performed, resulting in a rather sharp yet noisy fractal representation. The fractal resolution $(M, N) = (32, 64)$ yields the best trade-off between noise reduction and preserving the important features of the original image in the fractal reconstruction. The use of the quadtree-based fractal scheme was also investigated in order to control this trade-off between the partition size, noise and important image features reconstruction in a manner that is consistent with the human visual system. Simply encoding a noisy image by using a fractal-based method does not always result in optimal results that are comparable to the results obtained when applying standard image denoising methods, such as the Lee denoising filter. However, this initial investigation represents only a starting point of the application of fractal-based methods for the purpose of image denoising. Indeed, further investigations that aim for developing more efficient fractal image denoising methods are presented in the coming sections.

Next, a simple method for predicting the fractal code of the original noiseless image from the noisy one will me derived.

4.2 Predicting the Fractal Code of the Noise-free Image

In this section the relationship between a noisy image $\hat{\mathcal{I}}$ and its noiseless counterpart $\mathcal{I}$ is examined, specifically in terms of their respective fractal gray-level map coefficients. This relationship provides a method of estimating the fractal parameters of the noiseless image from those of the noisy image. From the former, a fractal representation of the noiseless image can then be reconstructed [31]. In the discussion that follows, variables and coefficients that correspond to a noisy image will have hats, e.g. $\hat{X}, \hat{Y}$ for the noisy image as opposed to X, Y for the noiseless image.

Before proceeding further, it will be useful to rewrite the least-squares gray-level coefficients in terms of standard statistical quantities. Recall that the optimal least-squared gray-level coefficients are given by

$$\begin{aligned}\alpha^* &= \frac{n\sum_{j=1}^n x_j y_j - \sum_{j=1}^n x_j \sum_{j=1}^n y_j}{n\sum_{j=1}^n x_j^2 - [\sum_{j=1}^n x_j]^2} \\ \beta^* &= \frac{1}{n}\sum_{j=1}^n y_j - \alpha^* \frac{1}{n}\sum_{j=1}^n x_j.\end{aligned} \tag{4.7}$$

An image is considered as a random signal so that the gray-level values $\{x_j, j = 1, 2, \ldots, n\}$ and $\{y_j, j = 1, 2, \ldots, n\}$ in (4.7) can be considered as random samples of the random variables X and Y representing the gray-level distribution of the (decimated) parent block D and child block R, respectively. The least-squares coefficients can then be written as

$$\begin{aligned}\alpha^* &= \frac{Cov(X,Y)}{\sigma^2_{\mathbf{X}}} \\ \beta^* &= E[Y] - \alpha^* E[X],\end{aligned} \tag{4.8}$$

where

$$Cov(X,Y) = \frac{\sum_{j=1}^n x_j y_j}{n} - \frac{\sum_{j=1}^n x_j}{n}\frac{\sum_{j=1}^n y_j}{n}, \tag{4.9}$$

$$\sigma^2_{\mathbf{X}} = \frac{\sum_{j=1}^n x_j^2}{n} - [\frac{\sum_{j=1}^n x_j}{n}]^2, \tag{4.10}$$

$$E[X] = \frac{\sum_{j=1}^n x_j}{n}, \text{ and } E[Y] = \frac{\sum_{j=1}^n y_j}{n}. \tag{4.11}$$

Strictly speaking, the above expressions are approximations to the statistical quantities of the random variables X and Y since they represent (finite) sample statistics. For large n, the sample statistics provide good estimates of the population statistics. The fact that n will not be large in our applications will contribute to errors in estimating the local image statistics and, subsequently, optimal fractal codes for the noiseless images.

4.2.1 Prediction of the Gray-level Coefficients

As above, let X and Y denote the random variables representing the gray-level values in a transformed parent block D and its corresponding child block R, respectively, for the noise-free original image. Also, let $\hat{X}$ and $\hat{Y}$ denote the corresponding gray-level random variables for the noisy image. Recall that the random variable $\hat{X}$ represents the gray-level values of the pixels in the transformed parent block. Various decimation methods have been used in the literature to produce from the parent block a transformed block of the same size as the child block, generally four times smaller. These include: (i) down-sampling by taking every fourth pixel, and (ii) averaging over 2×2 pixel blocks. The averaging operation is preferred and will be used in this study. However, averaging over 2×2 pixel blocks affects the noise variance since

$$\hat{x}_j = x_j + \frac{w_j + w_{j+1} + w_{j+2} + w_{j+3}}{4} = x_j + \bar{w}_{j,4} \tag{4.12}$$

where

$$\bar{w}_{j,4} = \frac{w_j + w_{j+1} + w_{j+2} + w_{j+3}}{4}. \tag{4.13}$$

Since the noise $\mathbf{w}$ is stationary, $\bar{w}_{j,4}$ is independent of the location index j and it is a sample from the averaged random noise

$$\bar{\mathbf{w}}_4 = \frac{\mathbf{w}_1 + \mathbf{w}_2 + \mathbf{w}_3 + \mathbf{w}_4}{4}, \qquad \text{where } \mathbf{w}_i \sim N(0, \sigma_{\mathbf{w}}^2). \tag{4.14}$$

Clearly, $\bar{\mathbf{w}}_4$ is also Gaussian with

$$E[\bar{\mathbf{w}}_4] = 0, \quad Var(\bar{\mathbf{w}}_4) \quad = \quad \frac{\sigma_{\mathbf{w}}^2}{4}. \tag{4.15}$$

Thus, the averaged noise $\bar{\mathbf{w}}_4$ is an AWGN process

$$\bar{\mathbf{w}}_4 \sim N(0, \frac{\sigma_{\mathbf{w}}^2}{4}). \tag{4.16}$$

The relationship between the random variables $\hat{X}$ and X corresponding to a domain block of the noisy and the noiseless image, respectively, can be written as follows:

$$\hat{X} = X + \bar{\mathbf{w}}_4 \Rightarrow E[\hat{X}] = E[X], \text{ since } E[\bar{\mathbf{w}}_4] = 0. \tag{4.17}$$

On the other hand, for the child blocks, no averaging or down-sampling is required, so the relationship between $\hat{Y}$ and Y is given by

$$\hat{Y} = Y + \mathbf{w} \Rightarrow E[\hat{Y}] = E[Y], \text{ again since } E[\bar{\mathbf{w}}] = 0. \tag{4.18}$$

Assuming that the image signal and the noise signal are independent, the variance of the noisy vector $\hat{X}$ is

$$\sigma^2_{\hat{\mathbf{X}}} = \mathrm{Var}(X + \bar{\mathbf{w}}_4) = \sigma^2_{\mathbf{X}} + \frac{\sigma^2_{\mathbf{w}}}{4}, \tag{4.19}$$

where $\sigma^2_{\mathbf{X}}$ is the variance of the noise-free vector X. Also, under the independence assumption between the noise and the image signal as well as the independence between $\bar{\mathbf{w}}_4$ and $\mathbf{w}$, it can be shown that

$$\mathrm{Cov}(\hat{X}, \hat{Y}) = Cov(X + \bar{\mathbf{w}}_4, Y + \mathbf{w}) = Cov(X, Y). \tag{4.20}$$

The independence between $\bar{\mathbf{w}}_4$ and $\mathbf{w}$ can be achieved by insuring that parent block $\hat{X}$ does not overlap with that child block $\hat{Y}$. From (4.8), (4.19) and (4.20), one can express the scaling coefficient $\hat{\alpha}^*$ in terms of the statistics of the noiseless image $\mathcal{I}$ and the noise variance as follows:

$$\hat{\alpha}^* = \frac{Cov(\hat{X}, \hat{Y})}{\sigma^2_{\hat{\mathbf{X}}}} = \frac{Cov(X, Y)}{\sigma^2_{\mathbf{X}} + \frac{\sigma^2_{\mathbf{w}}}{4}} = \frac{\frac{Cov(X,Y)}{\sigma^2_{\mathbf{X}}}}{1 + \frac{\sigma^2_{\mathbf{w}}}{4\sigma^2_{\mathbf{X}}}}. \tag{4.21}$$

From this result and (4.8) it can be shown that

$$\hat{\alpha}^* = \frac{\alpha^*}{1 + \frac{1}{4\gamma}}, \tag{4.22}$$

where the *signal-to-noise ratio*, γ, is defined as

$$\gamma = \frac{\sigma^2_{\mathbf{X}}}{\sigma^2_{\mathbf{w}}}. \tag{4.23}$$

From (4.19), the signal-to-noise can estimated from the noisy image as follows:

$$\gamma = \frac{\sigma^2_{\hat{\mathbf{X}}} - \frac{\sigma^2_{\mathbf{w}}}{4}}{\sigma^2_{\mathbf{w}}} = \frac{\sigma^2_{\hat{\mathbf{X}}}}{\sigma^2_{\mathbf{w}}} - \frac{1}{4}, \tag{4.24}$$

provided that

$$\sigma^2_{\hat{\mathbf{X}}} > \frac{\sigma^2_{\mathbf{w}}}{4}. \tag{4.25}$$

Similarly, from (4.17) and (4.18), the offset β^* is given by

$$\hat{\beta}^* = E[Y] - \hat{\alpha}^* E[X] = E[\hat{Y}] - \hat{\alpha}^* E[\hat{X}]. \tag{4.26}$$

Rearranging the above yields expressions for the the gray-level coefficients, (α^*, β^*), corresponding to the original noiseless image in terms the gray-level coefficients, $(\hat{\alpha}^*, \hat{\beta}^*)$, of the noisy image, as follows:

$$\alpha^* = (1 + \frac{1}{4\gamma})\hat{\alpha}^* \text{ and } \beta^* = E[\hat{Y}] - \alpha^* E[\hat{X}]. \tag{4.27}$$

These relationships will be further explored in the next sections for the purpose of estimating the fractal code of the original image from the noisy image.

Remarks

The following observations are made in view of the above derivations:

- In the absence of noise, i.e. as $\sigma_{\mathbf{w}}^2 \to 0$, then $\gamma \to \infty$ so one has

  $$\alpha^* \to \hat{\alpha}^* \quad \text{and} \quad \beta^* \to \hat{\beta}^*, \tag{4.28}$$

  as expected.

- Near edges or other high activity regions of the image, where the γ is relatively large, one has

  $$\hat{\alpha}^* \approx \alpha^* \quad \text{and} \quad \hat{\beta}^* \approx \beta^*. \tag{4.29}$$

So far, an approach for estimating the gray-level coefficients corresponding to the original noiseless image from those computed from the noisy image has been presented. Next, the problem of child-parent matching assignment will be addressed.

4.2.2 Prediction of the Optimal Parent-Child Matching

The results of the previous section suggest a simple algorithm to fractally denoise an image. First, estimate the variance $\sigma_{\mathbf{w}}^2$ of the noise in the image. Then, fractally encode the noisy image to obtain the noisy gray-level coefficients $(\hat{\alpha}^*, \hat{\beta}^*)$. Use (4.27) to estimate the noise-free gray-level coefficients (α^*, β^*). However, there is one problem: It is not guaranteed that the parent-child assignments of the noisy image $\hat{\mathcal{I}}$ are optimal for the noiseless image $\mathcal{I}$ in the *mean-squares* sense, i.e. that the MSE is minimized.

A method that estimates the optimal collage coding procedure for the noiseless image, which is not available, is proposed next. As before, let X, Y denote the random variables corresponding to the parent and (transformed) child blocks. ($\hat{X}, \hat{Y}$ for the noisy image.) The method is as follows:

For each child block $\hat{Y}_k \in \mathcal{R}$, $1 \leq k \leq N_R$, of the noisy image, the following steps are performed:

1. For each parent block $X_i \in \mathcal{D}_k$, the domain pool of Y_k, consider all possible geometric maps $\omega_{ik}^{(m)}$ and compute the least-squares gray-level gain coefficient, $\hat{\alpha}_{ik}^*$, corresponding to the noisy image. Then use this coefficient, the estimated noise variance and the estimated statistics from the noisy image to estimate the noise-free gray-level coefficients $(\alpha_{ik}^*, \beta_{ik}^*)$ using (4.27) as follows:

$$\alpha_{ik}^* = (1 + \frac{1}{4\gamma})\hat{\alpha}_{ik}^* \text{ and } \beta_{ik}^* = E[\hat{Y}_k] - \alpha^* E[\hat{X}_i]. \tag{4.30}$$

2. For the fractal code of the original noise-free image, one seeks to minimize the collage error, measured in terms of the *mean squares error* of the noiseless image as

$$\Delta_{ik}^{(m)2} = E[(Y_k - (\alpha_{ik}^* X_i + \beta_{ik}^*))^2 | (\alpha_{ik}^*, \beta_{ik}^*)]. \tag{4.31}$$

   Expanding the above quantity yields

$$\Delta_{ik}^{(m)2} = E[Y_k^2] + \alpha_{ik}^{*2} E[X_i^2] - 2\alpha_{ik}^* E[X_i Y_k] - 2\beta_{ik}^* E[Y_k] + 2\alpha_{ik}^* \beta_{ik}^* E[X_i] + \beta_{ik}^{*2}, \tag{4.32}$$

   which can be expressed in terms of the statistics of the noisy image as follows:

$$\begin{aligned}\Delta_{ik}^2 &= (E[\hat{Y}_k^2] - \sigma_{\mathbf{w}}^2) + \alpha_{ik}^{*2}(E[\hat{X}_i^2] - \frac{\sigma_{\mathbf{w}}^2}{4}) - 2\alpha_{ik}^* E[\hat{X}_i \hat{Y}_k] - 2\beta_{ik}^* E[\hat{Y}_k] \\ &\quad + 2\alpha_{ik}^* \beta_{ik}^* E[\hat{X}_i] + \beta_{ik}^{*2}.\end{aligned} \tag{4.33}$$

   provided that

$$E[\hat{Y}_k^2] > \sigma_{\mathbf{w}}^2 \text{ and } E[\hat{X}_i^2] > \frac{\sigma_{\mathbf{w}}^2}{4}. \tag{4.34}$$

   As a result, the collage error for the noiseless image is estimated from statistics of the noisy image.

3. Select the optimal parent block $i^*(k)$ and associated geometric map $w_{i^*(k),k}^{m^*(k)}$ such that

$$\Delta_{i^*(k),k}^{(m^*(k))2} \leq \Delta_{jk}^2 \quad \text{for all} \quad j \neq i^*(k) \text{ and } m \neq m^*(k). \tag{4.35}$$

   The result is an estimated collage-based matching criterion for the original noise-free image.

A method for estimating the fractal code of the noise-free image from that of the noisy image has been derived. This fractal code is expected to be closer to the code of the noiseless image as the variance of the noise decreases, converging to the latter in the limit as $\sigma_{\mathbf{w}} \to 0$. This method will be outlined, as an algorithm, in the next section.

4.2.3 Predicting the Fractal Code of the Original Image from the Noisy Image

In view of the above derivations and discussion, an algorithmic approach for predicting the fractal code for the original noise-free image from the noisy image can be outlined as follows:

For each *uncoded* child block $\hat{Y}_k \in \mathcal{R}$, the range blocks, of the noisy image, the following steps are performed:

1. Compute its energy

$$E[\hat{Y_k}^2] = \frac{1}{n} \sum_{m=1}^{n} \hat{y}_{k,m}^2. \tag{4.36}$$

Then get an estimate of the energy of the corresponding child block in the original noiseless image as follows:

$$\mathcal{E}_{Y_k} = E[{Y_k}^2] = E[\hat{Y_k}^2] - \sigma_{\mathbf{w}}^2. \tag{4.37}$$

In theory, $\mathcal{E}_{Y_k}$ must be positive, however in practice this will not always be the case since $E[\hat{Y}_k^2]$ is computed locally from sub-blocks with a relatively small number of coefficients, so it is possible to encounter cases where

$$E[\hat{Y}_k^2] < \sigma_{\mathbf{w}}^2, \tag{4.38}$$

resulting in negative values for $\mathcal{E}_{Y_k}$. Thus, there are two cases: $E[\hat{Y}^2] > \sigma_{\mathbf{w}}^2$ and $E[\hat{Y}^2] \leq \sigma_{\mathbf{w}}^2$. More specifically, to avoid cases where the estimate of the energy is zero, one should consider the following two cases: $E[\hat{Y}^2] \geq \lambda\sigma_{\mathbf{w}}^2$ and $E[\hat{Y}^2] < \lambda\sigma_{\mathbf{w}}^2$, where the parameter $\lambda > 1$, to be determined experimentally.

2. For each possible parent block $\hat{X}_i \in \mathcal{D}_k$, the domain pool of Y_k, and geometric map, $w_{ik}^{(m)}$, compute:

The least-squares gray-level gain coefficient corresponding to the noisy image

$$\hat{\alpha}_{ik}^* = \frac{Cov(\hat{X}_i, \hat{Y}_k)}{\sigma_{\hat{\mathbf{X}}_\mathbf{i}}^2} \tag{4.39}$$

as shown in Eq. (4.8).

Then compute the signal-to-noise ratio, as predicted from the noisy image as follows:

$$\gamma = \frac{\sigma_{\hat{\mathbf{X}}_i}^2}{\sigma_{\mathbf{w}}^2} - \frac{1}{4}. \tag{4.40}$$

Again, in theory, γ must be positive, however in practice this will not alway be the case since and $\sigma^2_{\hat{\mathbf{X}}_i}$ is computed locally from sub-blocks with a relatively small number of coefficients, so it is possible to encounter cases where

$$\sigma^2_{\hat{\mathbf{X}}_i} < \frac{\sigma^2_{\mathbf{w}}}{4}, \tag{4.41}$$

resulting in negative values for γ values. Thus, again there are two different cases: $\sigma^2_{\hat{\mathbf{X}}_i} \geq \frac{\sigma^2_{\mathbf{w}}}{4}$ and $\sigma^2_{\hat{\mathbf{X}}_i} < \frac{\sigma^2_{\mathbf{w}}}{4}$. More specifically, to avoid cases where the estimate of γ is zero, one should consider the following two cases: $\sigma^2_{\hat{\mathbf{X}}_i} \geq \lambda\frac{\sigma^2_{\mathbf{w}}}{4}$ and $\sigma^2_{\hat{\mathbf{X}}_i} < \lambda\frac{\sigma^2_{\mathbf{w}}}{4}$, for the same parameter, $\lambda > 1$, chosen above.

3. Thus, overall, there are two cases:

   (a) **Case 1**: If $E[\hat{Y}^2] \geq \lambda\sigma^2_{\mathbf{w}}$ and $\sigma^2_{\hat{\mathbf{X}}_i} \geq \lambda\frac{\sigma^2_{\mathbf{w}}}{4}$, then:

   - One has:

$$\mathcal{E}_{Y_k} = E[\hat{Y_k}^2] - \sigma^2_{\mathbf{w}} \geq (\lambda-1)\sigma^2_{\mathbf{w}} > 0 \text{ and } \gamma = \frac{\sigma^2_{\hat{\mathbf{X}}_i}}{\sigma^2_{\mathbf{w}}} - \frac{1}{4} \geq \frac{(\lambda-1)}{4}\sigma^2_{\mathbf{w}} > 0. \tag{4.42}$$

   - Since the current parent sub-block, X_i is fixed, then one is using the same child-parent assignment; $(\hat{Y}_k, \hat{X}_i)$ for the noisy image and (Y_k, X_i) for the noiseless image. Thus, the noise-free gray-level coefficients, $(\alpha^*_{ik}, \beta^*_{ik})$, can be estimated as follows:

$$\begin{aligned} \alpha^*_{ik} &= (1 + \frac{1}{4\gamma})\hat{\alpha}^* \\ \beta^*_{ik} &= E[\hat{Y}_k] - \alpha^* E[\hat{X}_i]. \end{aligned} \tag{4.43}$$

   as shown in (4.27). This yields an estimate of the gray-level coefficients α^*, β^*.

   - Now, use these estimates of α^* and β^* to compute the collage error, measured in terms of the *mean squared error* for the original noiseless image

$$\begin{aligned} \Delta^2_{ik} &= (E[\hat{Y}^2_k] - \sigma^2_{\mathbf{w}}) + \alpha^{*2}_{ik}(E[\hat{X}^2_i] - \frac{\sigma^2_{\mathbf{w}}}{4}) - 2\alpha^*_{ik}E[\hat{X}_i\hat{Y}_k] \\ &\quad -2\beta^*_{ik}E[\hat{Y}_k] + 2\alpha^*_{ik}\beta^*_{ik}E[\hat{X}_i] + \beta^{*2}_{ik}. \end{aligned} \tag{4.44}$$

   as shown in (4.33). Note that in this case it is assumed that $\sigma^2_{\hat{\mathbf{X}}_i} > \lambda\frac{\sigma^2_{\mathbf{w}}}{4}$, then the second term in the above collage error expression is non-negative, since

$$E[\hat{X}^2_i] \geq \sigma^2_{\hat{\mathbf{X}}_i} > \lambda\frac{\sigma^2_{\mathbf{w}}}{4}. \tag{4.45}$$

   This ensures a collage-based matching criterion for the original noise-free image.

Now, the case when the variance of the child-block is small is considered.

(b) **Case 2**: If $E[\hat{Y}_k^2] < \lambda\sigma_{\mathbf{w}}^2$ or $\sigma_{\hat{\mathbf{X}}_i}^2 < \lambda\frac{\sigma_{\mathbf{w}}^2}{4}$, then:

- One may assume that child or parent block is dominated by the noise and the corresponding subregion of the original image is mainly flat and low-activity and contains little relevant information. Thus, it can be argued that it would be beneficial to reduce the magnitude of $\hat{\alpha^*}_{ik}$ since larger values of $\hat{\alpha}_{ik}^*$ would amplify the noise. Thus, $\hat{\alpha}_{ik}^*$ can be reduced to get an estimate of α_{ik}^*, as follows:

$$\alpha_{ik}^* \approx \min(\frac{E[\hat{Y}_k^2]}{\lambda\sigma_{\mathbf{w}}^2}, \frac{\sigma_{\hat{\mathbf{X}}_i}^2}{\lambda\frac{\sigma_{\mathbf{w}}^2}{4}}) \times \hat{\alpha}_{ik}^*. \tag{4.46}$$

- Also, in this case one cannot use the collage error in Eq. (4.33), derived for the noise-free image, because the estimate

$$E[Y_k^2] \approx E[\hat{Y}_k^2] - \sigma_{\mathbf{w}}^2 \text{ or } E[X_i^2] \approx E[\hat{X}_i^2] - \frac{\sigma_{\mathbf{w}}^2}{4} \tag{4.47}$$

may be negative. Thus, in this case one needs to resort to the using the collage error, corresponding to the noisy image

$$\Delta_{ik}^2 = E[\hat{Y}_k^2] + \alpha_{ik}^{*2}E[\hat{X}_i^2] - 2\alpha_{ik}^*E[\hat{X}_i\hat{Y}_k] - 2\beta_{ik}^*E[\hat{Y}_k] + 2\alpha_{ik}^*\beta_{ik}^*E[\hat{X}_i] + \beta_{ik}^{*2}. \tag{4.48}$$

4. Select the optimal parent sub-block $i^*(k)$ and geometric map, $m^*(k)$, such that

$$\Delta_{i^*(k),k}^{(m^*(k))2} \leq \Delta_{i,k}^{(m)2} \quad \text{for all} \quad i \neq i^*(k) \text{ and } m \neq m^*(k). \tag{4.49}$$

In view of the above detailed outline, the proposed predictive fractal image denoising scheme can be summarized as follows:

The Proposed Predictive Fractal Image Denoising Algorithm

Choose the standard fractal scheme and its resolution (M, N), then: For each *uncoded* child sub-block $\hat{Y}_k$, $k = 1, 2, \ldots, N \times N$, do the following:

1. Compute the energy of the child sub-block

$$E[\hat{Y}_k^2] = \frac{1}{n}\sum_{m=1}^{n} \hat{y}_{k,m}^2. \tag{4.50}$$

2. For each possible parent sub-block $\hat{X}_i \in \mathcal{D}_k$, the domain pool of $\hat{Y}_k$, and each possible geometric map, $w_{ik}^{(m)}$, compute:

- Its variance

$$\sigma^2_{\hat{\mathbf{X}}_i} = \frac{1}{n}\sum_{m=1}^{n}(\hat{x}_{i,m} - \bar{x}_i)^2. \tag{4.51}$$

- The least-squares gray-level gain coefficient corresponding to the noisy image:

$$\hat{\alpha}^*_{ik} = \frac{Cov(\hat{X}_i, \hat{Y}_k)}{\sigma^2_{\hat{\mathbf{X}}_\mathbf{i}}} \tag{4.52}$$

Now, consider one of the following two cases:

(a) **Case 1**: If $E[\hat{Y_k}^2] \geq \lambda\sigma^2_{\mathbf{w}}$ and $\sigma^2_{\hat{\mathbf{X}}_i} \geq \lambda\frac{\sigma^2_{\mathbf{w}}}{4}$, then:

- Compute the signal-to-noise ratio, γ, corresponding to the original noise-free image, which can be estimated by

$$\gamma = \frac{\sigma^2_{\hat{\mathbf{X}}_i}}{\sigma^2_{\mathbf{w}}} - 1. \tag{4.53}$$

- Predict the scaling coefficient, α^*, corresponding to the noise-free image as follows:

$$\alpha^*_{ik} = (1 + \frac{1}{4\gamma})\hat{\alpha}^*_{ik}. \tag{4.54}$$

- Compute the gray-level (offset) coefficient, β^*, corresponding to the the noisy image, as given by

$$\beta^*_{ik} = E[\hat{Y}_k] - \alpha^*_{ik}E[\hat{X}_i]. \tag{4.55}$$

- Compute the collage error corresponding to the original noise-free image

$$\begin{aligned}\Delta^2_{ik} &= (E[\hat{Y}^2_k] - \sigma^2_{\mathbf{w}}) + \alpha^{*2}_{ik}(E[\hat{X}^2_i] - \frac{\sigma^2_{\mathbf{w}}}{4}) - 2\alpha^*_{ik}E[\hat{X}_i\hat{Y}_k] \\ &\quad -2\beta^*_{ik}E[\hat{Y}_k] + 2\alpha^*_{ik}\beta^*_{ik}E[\hat{X}_i] + \beta^{*2}_{ik}.\end{aligned} \tag{4.56}$$

(b) **Case 2**: If $E[\hat{Y_k}^2] < \lambda\sigma^2_{\mathbf{w}}$ or $\sigma^2_{\hat{\mathbf{X}}_i} < \lambda\frac{\sigma^2_{\mathbf{w}}}{4}$, then:

- Estimate the scaling coefficient, α^*, corresponding to the noise-free image, from $\hat{\alpha}^*$, corresponding to the noisy image, as follows:

$$\alpha^*_{ik} \approx \min(\frac{E[\hat{Y}^2_k]}{\lambda\sigma^2_{\mathbf{w}}}, \frac{\sigma^2_{\hat{\mathbf{X}}_i}}{\lambda\frac{\sigma^2_{\mathbf{w}}}{4}}) \times \hat{\alpha}^*_{ik}. \tag{4.57}$$

- Compute the gray-level offset coefficient, β^*_{ik}, corresponding to the the noisy image, as given by

$$\beta^*_{ik} = E[\hat{Y}_k] - \alpha^*_{ik}E[\hat{X}_i]. \tag{4.58}$$

- Compute the collage error corresponding to the noisy image, as given by

$$\Delta_{ik}^2 = E[\hat{Y}_k^2] + \alpha_{ik}^{*2} E[\hat{X}_i^2] - 2\alpha_{ik}^* E[\hat{X}_i \hat{Y}_k] - 2\beta_{ik}^* E[\hat{Y}_k] + 2\alpha_{ik}^* \beta_{ik}^* E[\hat{X}_i] + \beta_{ik}^{*2}. \quad (4.59)$$

3. Select the optimal parent sub-block, $i^*(k)$, and geometric map, $m^*(k)$, such that

$$\Delta_{i^*(k),k}^{(m^*(k))2} \leq \Delta_{i,k}^{(m)2} \quad \text{for all} \quad i \neq i^*(k) \text{ and } m \neq m^*(k). \quad (4.60)$$

The predicted fractal code corresponding to the original noise free consisting of

$$\textit{Fractal code} = \{i^*(k), m^*(k), \alpha_{i^*(k)}^{*m^*(k)}, \beta_{i^*(k)}^{*m^*(k)}, \text{ for } k = 1, 2, \ldots, N^2\}. \quad (4.61)$$

which can then be used by the fractal decoder to generate a denoised estimate of the original image.

In view of the above algorithm, the following observations are outlined:

- Clearly, it is straightforward to apply the above predictive algorithm on any of the standard fractal schemes implemented in this chapter for the purpose of image denoising. This predictive fractal denoising scheme will be implemented for the purpose of image denoising, using the exhaustive fractal scheme with $(M, N) = (32, 64)$ and the quadtree-based fractal scheme.

- The selection of the parameter λ in the above algorithm, is based on the interpretation of what is considered as a noise dominated child or parent sub-block and what is considered as a high activity child or parent sub-block. The parameter, λ will be chosen to be 2; to indicate that if the variability of a child block is twice the noise variance, then the child sub-block is dominated by noise. It was observed that $\lambda = 2$ yields better qualitative experimentally results for the noisy test image of “Lenna”.

Next, the issue of quantizing the estimated gray-level coefficients is addressed in order to obtain fractally denoised estimates with gray-level values in the $[0, 255]$, which is the appropriate range for the 8-bits/pixel test image of “Lenna” as well as other test images of interest.

4.2.4 Quantization of the Gray-level Coefficients

In order to guarantee the contractivity of the fractal transform with respect of the $\mathcal{L}_\infty$ for various classes of images, one has to ensure that the absolute value of the quantized scaling parameter α is smaller than unity. In the quantization methods used in this work, the scaling coefficients are restricted to the interval $[-0.99, 0.99]$. Also, adding noise to images often results in many pixels of the noisy image having gray-level values outside the appropriate $[0, 255]$ range for 8-bit gray-scale images. In order to ensure that the gray-level values of the fractal denoised estimate lie within this appropriate range, the following quantization strategy is proposed for the offset coefficient β.

For a quantized scaling coefficient α_q and offset coefficient β_q, it is required that

$$0 \leq \alpha_q x + \beta_q \leq 255 \tag{4.62}$$

for each pixel value x in the parent block. Thus, there are two cases, depending on whether α_q is positive or negative:

1. If $\alpha_q \geq 0$

$$0 \leq x \leq 255 \quad \Rightarrow \quad \beta_q \leq \alpha_q x + \beta_q \leq 255\alpha_q + \beta_q. \tag{4.63}$$

   Thus to ensure that $\alpha_q x + \beta_q$ remains in the $[0, 255]$ range it suffices to have

$$0 \leq \beta_q \text{ and } 255\alpha_q + \beta_q \leq 255, \text{ or equivalently } 0 \leq \beta_q \leq 255(1 - \alpha_q). \tag{4.64}$$

2. If $\alpha_q < 0$

$$0 \leq x \leq 255 \quad \Rightarrow \quad \beta_q \geq \alpha_q x + \beta_q \geq 255\alpha_q + \beta_q. \tag{4.65}$$

   Similarly, to ensure that $\alpha_q x + \beta_q$ remains in the $[0, 255]$ range it suffices to have

$$0 \leq 255\alpha_q + \beta_q \text{ and } \beta_q \leq 255, \text{ or equivalently } -255\alpha_q \leq \beta_q \leq 255. \tag{4.66}$$

This guarantees that the fractal representation of a noisy image will lie within the $[0, 255]$ range provided that the seed image also satisfies this condition. Since one typically uses a blank image, i.e., $u_{ij} = 255$, as the initial seed, the condition is satisfied.

In view of the above derivations, the offset coefficient β is quantized to be within the interval $[\beta_{min}, \beta_{max}]$, depending on the sign of the scaling coefficient α, as follows:

$$[\beta_{min}, \beta_{max}] = \begin{cases} [0, 255(1 - \alpha_q)], & \text{if } \alpha_q \geq 0, \\ [-255\alpha_q, 255], & \text{if } \alpha_q < 0. \end{cases} \tag{4.67}$$

This ensures that the various fractal representations have pixel values within the suitable $[0, 255]$ range. Since achieving high compression ratios is not a priority at this point, then a uniform quantizer with $N_Q = 1024$ quantization levels is used. One of the benefits of insuring that the resulting fractal denoised estimates have pixel values within the suitable $[0, 255]$ range, is that one can use the standard RMSE and PSNR qualitative measures to assess the quality of these representations and display these representations on any standard 8-bit graphic device.

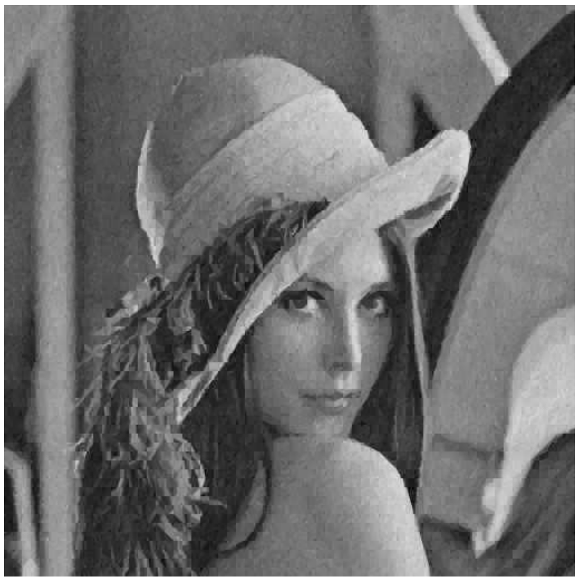
4.2.5 Experimental Results

The uniform as well as quadtree partitioning predictive fractal schemes were implemented for the purpose of enhancing and restoring the noisy test image of “Lenna”. The results are illustrated in Figure 4.6. The following observations are outlined in view of these results:

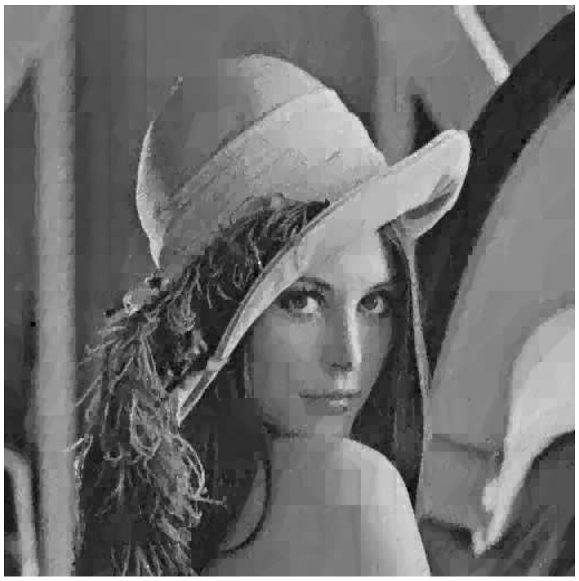
- Clearly, there is clearly a significant improvement of the quality of the fractally denoised estimate, especially when the quadtree partitioning of the image is used. In these fractal representations, most of the noise appears to have been suppressed without blurring the edges or other high frequency components of the image. The zoomed images reflect the disturbing blockiness artifacts in the fractal representations, which is not surprising. Except for a few blockiness artifacts, the quadtree-based fractally denoised estimate appears to have high visual quality. In the next section, ways of reducing these disturbing blockiness artifacts will be addressed. Clearly, the application of the predictive fractal denoising algorithm has resulted in significantly better results than simply fractally encoding the image.

- It is interesting to note that, as illustrated in Figures 4.1 and 4.3, when simply encoding the noisy image, the standard uniform-based fractal coding of the noisy image results in a better denoised estimate (quantitatively) than quadtree-based fractal coding of the noisy image. This is because the quadtree-based fractal coding reconstructs most of the noise in the vicinity of edges and other high-frequency content of the image. However, after applying the proposed fractal code prediction method, the quadtree-based fractal denoising scheme yields a denoised estimate that has a better quality than the one obtained by the uniform partitioning based fractal denoising scheme. This can be explained as follows:

  - When simply encoding the noisy image using the quadtree-based fractal scheme, a signal to noise ratio (SNR) decomposition was applied. The quadtree-based fractal denoising scheme performs a content-based denoising where a high degree of denoising is performed in uniform sub-regions of the image and a lower degree of denoising in performed in the vicinity of edges without compromising their sharpness. Thus, the quadtree denoised image, appears overly smoothed in flat regions and noisy near edges and within high activity sub-regions of the image, which explains why the quadtree denoised estimate has a high RMSE (low PSNR). However, the uniform-based fractal denoising scheme performs a uniform degree of smoothing throughout the image, regardless of its content, and the resulting denoised estimate is smoothed uniformly and contains little or no residual noise. Although the qualitative fidelity measures seem to indicate that uniform-based fractally denoised image is better than the quadtree-based fractally denoised one, visually one may prefer the latter because the presence of noise near edges is less noticeable. However, the quadtree-based fractally denoised image suffers from non-uniform disturbing artifacts.



(a) Predictive fractal denoising
Uniform partitioning: $(M, N) = (32, 64)$
RMSE=10.03, PSNR=28.10.
Execution time $\approx$ 1234 secs.



(b) Predictive fractal denoising
Quadtree partitioning: collage error
RMSE=9.10, PSNR=28.95.
Execution time $\approx$ 1047 secs.

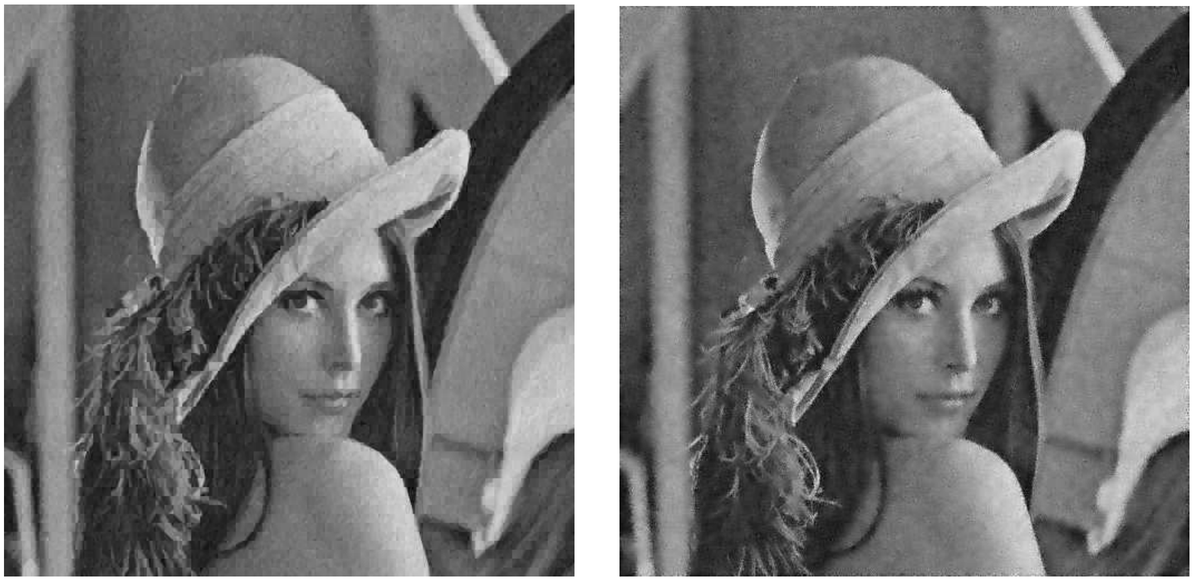
Figure 4.6: The fractal denoised estimates obtained using the proposed predictive fractal denoising algorithm with uniform and quadtree partitioning of the image.

– When using the quadtree-based predictive fractal denoising algorithm, a collage error decomposition criterion was used. A child sub-block is only fractally encoded using the predictive fractal scheme if the resulting collage error is less than a desired error tolerance level. Otherwise, it is split into four quadrant child blocks. By doing so, the predictive fractal denoising scheme performs noise reduction on one hand. On the other hand, the use of the quadtree, with the collage error decomposition criterion, ensures that important features (i.e. edges) of the original image are represented well, by using finer partitioning when necessary, to guarantee a specified fitting error.

Next, the results obtained by the predictive fractal denoising scheme, using uniform partitioning, will be compared to the results obtained by the Lee filter.

4.2.6 Comparison with the Lee Filter

As shown earlier, there is a close relationship between the fractal-based schemes and the Lee filter. Both of these methods attempt to reconstruct a denoised image through affine transformations of sub-blocks of the noisy image into other sub-regions. Thus, the performance of the proposed fractal-based scheme will be compared to the results obtained by the Lee filter.



(a) Predictive fractal denoising
Uniform partitioning: $(M, N) = (32, 64)$
RMSE=10.03, PSNR=28.10.
Execution time $\approx$ 1234 secs.

(b) Lee filter
Local mask: 7×7 pixels
RMSE=10.83, PSNR=27.44.
Execution time $\approx$ 3 secs.

Figure 4.7: Comparison between the fractal-based image method and the Lee filter: (a) the fractal-based image denoising approach using uniform partitioning with 8×8 mask, and (b) the Lee filter, using a 7×7 mask.

Figure 4.7 illustrates results of the proposed fractal-based and Lee filter image denoising methods as applied to the noisy "Lenna" image. For the fractal-based scheme, uniform partitioning was used with 8×8 pixel child blocks was used. The uniform fractal predictive scheme was chosen instead of the quadtree-based one in order to ensure a fair comparison with the Lee filter which uses uniform 7×7 masks to estimate the local statistics of the image.

The proposed fractal denoising method is seen to yield better results both quantitatively, in terms of RMSE and PSNR, as well as qualitatively, in terms of the visual quality of the denoised image. Another advantage of the fractal method is that it yields a representation of the noisy image with pixel values that lie in the range $[0, 255]$ because of the quantization strategy imposed upon the gray-level coefficients, α^* and β^*, as discussed in the previous section. The Lee filter representation of the noisy image will not always satisfy this condition. However, in terms of the computational complexity, the Lee filter is clearly significantly more efficient and faster than the proposed fractal denoising scheme, as reflected by the execution times of the two methods.

Next, the performance of the above algorithm in predicting the fractal code of the test image is assessed.

4.2.7 Examining the Performance of the Predicted Fractal Code

In order to assess the performance of the above predictive fractal scheme, the available original test image of "Lenna" is used to compute its true fractal code. This exact fractal code is then compared to the one predicted by the above predictive scheme, from the noisy test image of "Lenna" with noise intensity, $\sigma = 25$. The results are summarized as follows:

- Figure 4.8 illustrates the distribution of the gray-level coefficients, α and β, corresponding to the original and noisy images as well as the predicted gray-level coefficients. Note that the distribution of the scaling coefficients, α, corresponding to the noisy image is rather distinct from that of the original image. In fact, it looks bi-modal with all scaling coefficients being non-zero. As explained earlier, this is the case because when fractal coding the noise-free image, flat regions yield almost constant child blocks which in turn result in zero (or close to zero) scaling coefficients, α. However, the noisy image has no more flat regions, since smooth regions in the original image are now dominated by the noise. Thus, the resulting child blocks in the partition are non-uniform and the corresponding scaling coefficients, α, are going to be non-zero. The distribution of the predicted scaling coefficients remains bi-modal, although to a lesser degree than that of the noisy image. This is the case because the image has many flat low-activity subregions which correspond to child blocks dominated by noise. As explained in case 2 of the above algorithm, these sub-blocks are simply fractally coded while introducing a minor modification (reduction) of their corresponding scaling coefficients, as given in (4.57). Consequently, the distribution of the predicted scaling coefficients is now closer to that of the

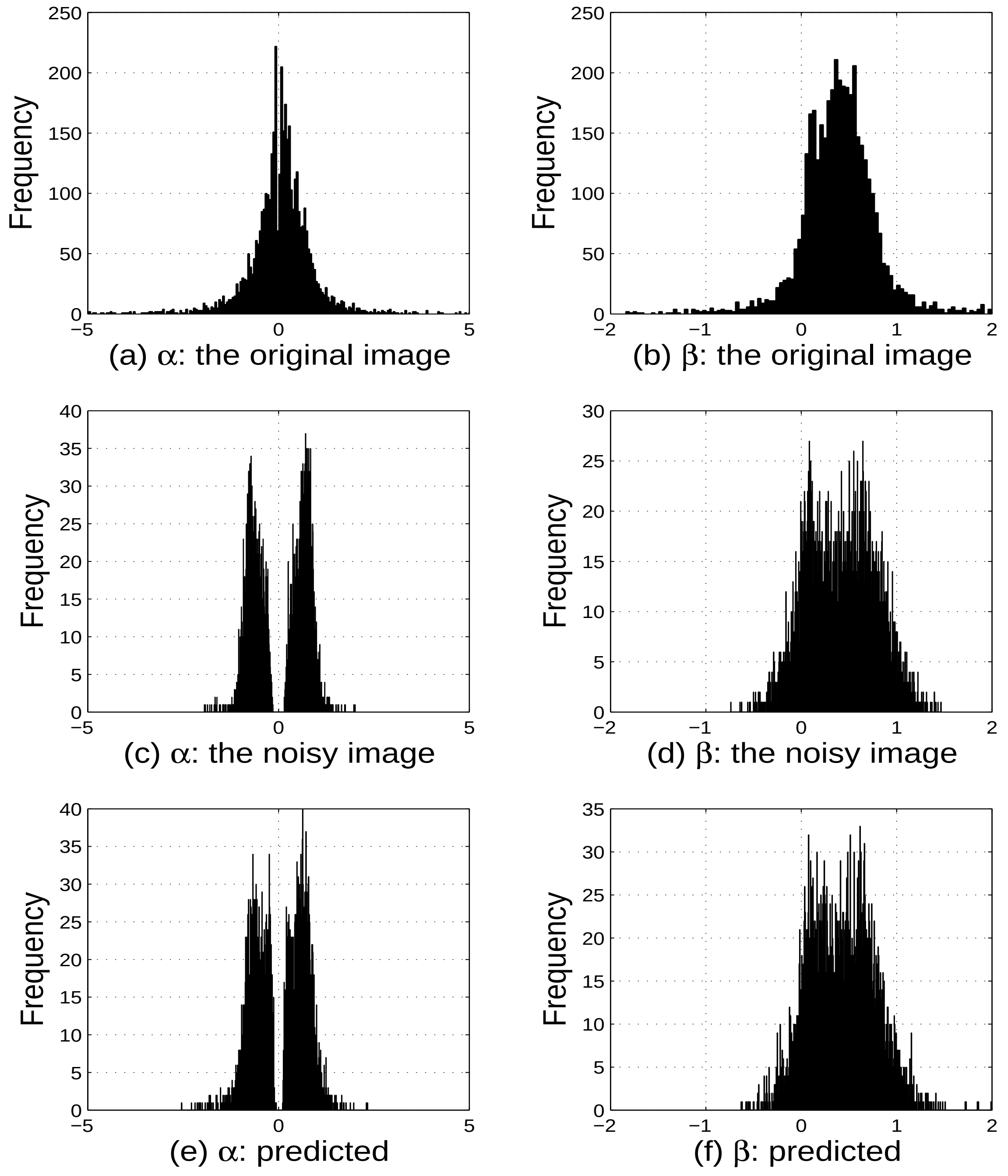
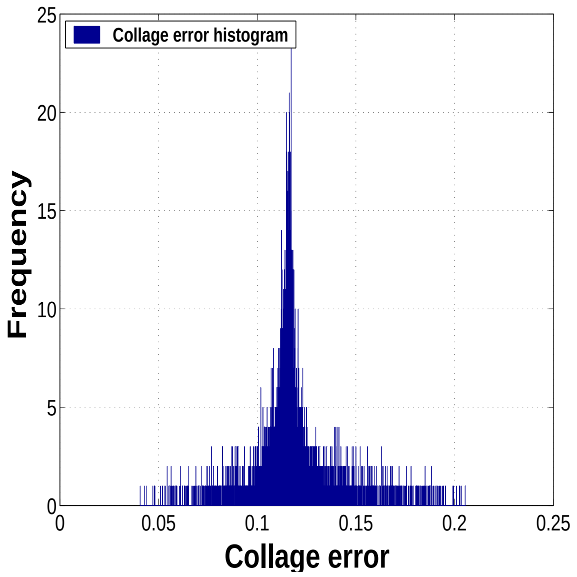
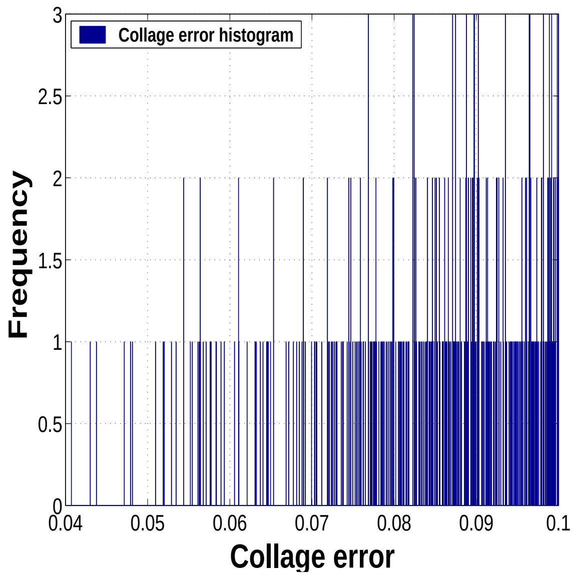


Figure 4.8: Comparison between the gray-level coefficients corresponding to the original image, the noisy image as well as the predicted coefficients. Note that the distributions of the predicted gray-level coefficients resembles those corresponding to the true gray level coefficients.

exact scaling coefficients corresponding to the original image.



(a) Collage error distribution for child sub-block number 900.



Zooming in on the figure in (a)

Figure 4.9: The distribution of the collage error corresponding to a typical child block. Note that there are many sub-optimal parent sub-blocks that yield collage errors that are relatively close to the minimum collage error corresponding to the optimal parent sub-block.

- As for the prediction of the true child-parent assignment maps, it should be noted that one does not need to predict the optimal parent block corresponding to each child block. In fact, as illustrated in Figure 4.9, for each child block there are many good sub-optimal parent blocks that can be chosen instead of the optimal parent block at the expense of a relatively small reduction in the fidelity. Note how for this typical child block, there are a few parent sub-optimal parent blocks that yield collage errors that are relatively close to the minimum collage error obtained from the optimal parent block. This is the subject of a work in progress by Alexander, Tsurumi and Vrscay who have shown that child blocks, especially those corresponding to edges and other high activity subregions, can be matched to many good sub-optimal parent sub-blocks, at the expense of a relatively small degradation in the fractal representation.

- Figure 4.10 illustrates the histogram of the rank of the predicted parent blocks obtained from the noisy image corrupted by AWGN noise with different noise intensity σ. For the chosen

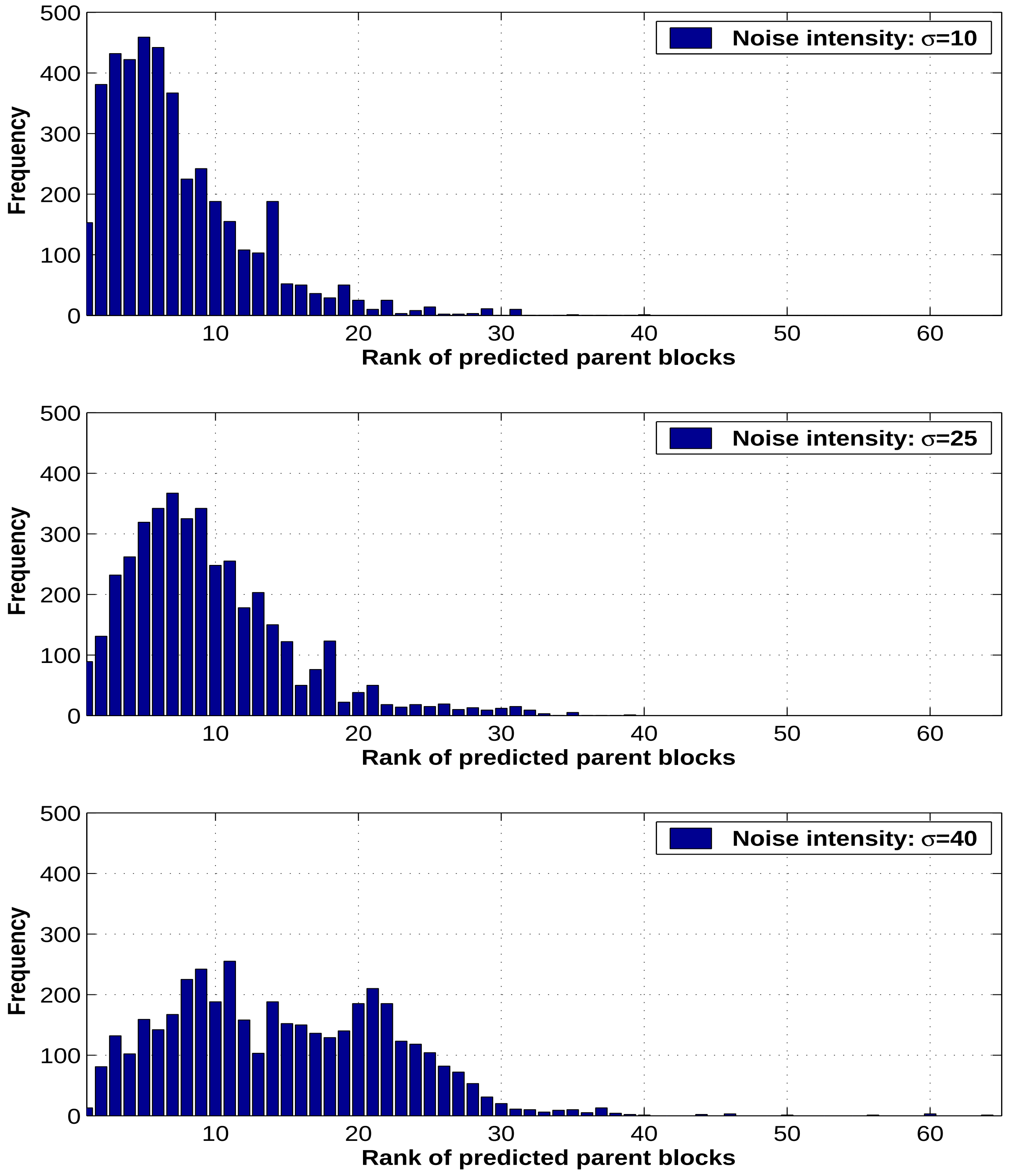
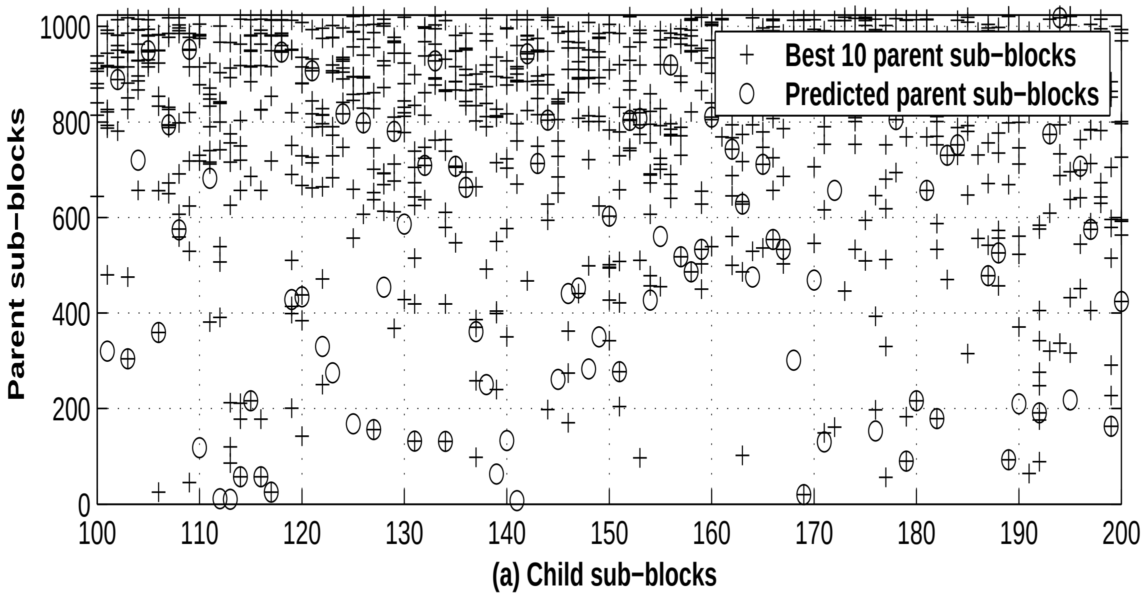


Figure 4.10: The histogram of the rank of the predicted parent blocks for different noise intensity, σ. A rank of 1 means that the optimal parent block was predicted, the rank is 2 when the second best parent block was predicted, etc. Note that for lower noise intensity, most of the predicted parent blocks are among the first few closest parent blocks. However, as the noise intensity increases, many of the predicted parent blocks are further away from the optimal ones.



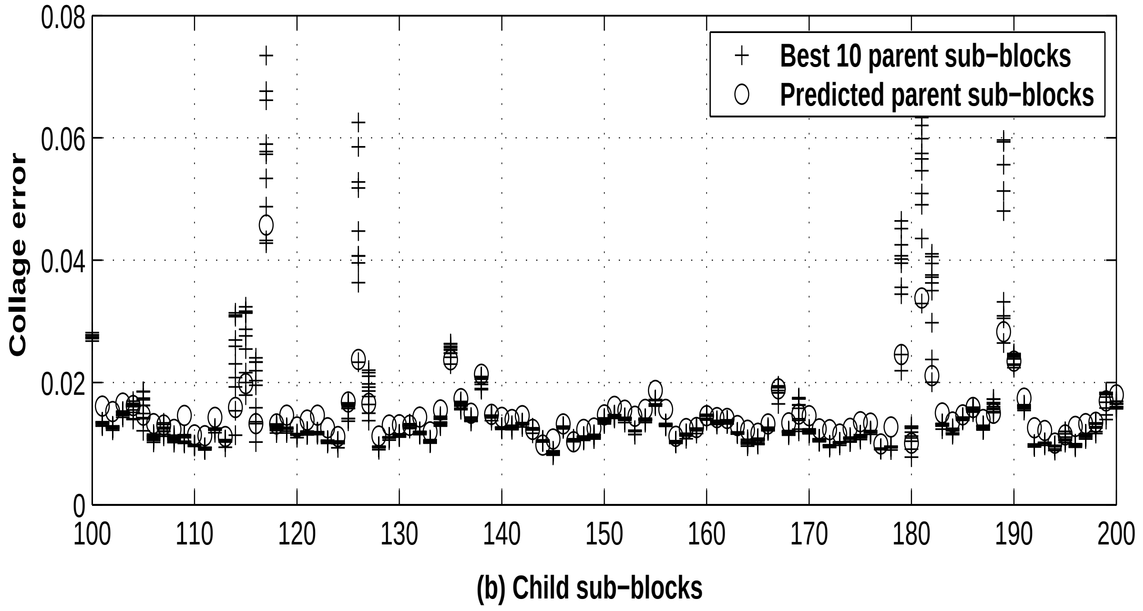


Figure 4.11: (a) Parent-child assignment maps using the best 10 parent blocks corresponding to the original image as well as the predicted parent blocks, for some of the child blocks. (b) The collage errors for the best 10 parent sub-blocks as well as the collage errors corresponding to the predicted parent blocks. It was observed that for about 2857 child blocks, the predicted parent blocks matches one of the best 10 parent sub-blocks, for the noisy test image with $\sigma = 25$.

$(M, N) = (32, 64)$ fractal resolution, all 4096 child blocks were examined and for each child block, all of the 1024 potential parent blocks were tested. For each child block, the predicted parent block is then ranked according to how close it is to best (optimal) parent block, as obtained from the original image. A rank of 1 means a perfect match, while a rank of 2, implies that the predicted parent block is the second best, etc. Note that, for the lower noise intensity, for most of the child blocks, the predicted parent blocks are among the first few closest (sub-optimal) parent blocks. As the noise intensity increases, the rank distribution is shifted to the right indicating that many of the predicted parent blocks are further away from the optimal ones.

- Figure 4.11 (a) illustrates the child-parent assignment maps using the closest 10 parent sub-blocks as well as the predicted child-parent assignment map, corresponding to the noisy test image with $\sigma = 25$. Note that in many cases, the predicted child-parent assignment coincides with one of the closest 10 parent blocks. Part (b) of this figure illustrates the collage errors corresponding to the closest 10 parent blocks and the collage errors corresponding to the predicted parent blocks. The fact that the predicted parent blocks are usually among the first few closest (i.e., near-optimal) parent blocks may indeed be the main reason why the predictive fractal scheme performs reasonably well in predicting a fractal code that is relatively close to the true fractal code of the original image, resulting in a relatively good denoised estimate of the original image.

This completes the examination of the performance of the predicted fractal code.

In this section, an algorithm for the purpose of estimating the fractal code of the original noise-free image from the noisy one was proposed. This fractal denoising scheme was implemented using uniform as well as quadtree partitioning of the noisy image. It was shown that the performance of the uniform based fractal denoising scheme surpasses that of the Lee filter. The quadtree-based fractal denoising scheme performs even better than the uniform based one.

Next, the use of the cycle spinning algorithm in order to improve the fractal denoised estimates is investigated.

4.3 Improving Fractal Image Denoising via Cycle Spinning

In spite of the significant gains achieved by the proposed fractal denoising schemes, the denoised estimate still suffers from disturbing blockiness artifacts that are inherent in fractal-based image representations. In fact, zooming in on any of the fractally denoised estimates, so far generated in this chapter, reveals the disturbing blockiness artifacts in these fractal representations, as illustrated in Figure 4.12. The use of the cycle spinning idea is proposed for the purpose of reducing these artifacts and enhancing the quality of the fractally denoised estimates.

Incorporating the idea of cycle spinning within fractal-based schemes can be performed in a similar fashion to the use of cycle spinning for the purpose of wavelet thresholding and denoising. The main difference is that the standard fractal schemes are performed in the spatial domain of the image and hence the computation of the discrete wavelet transform and its inverse are no longer required. Also, the middle step of wavelet thresholding is now replaced by fractally encoding the noisy image, using any of the spatially-based fractal image denoising schemes developed so far. The initial and final steps of shifting and unshifting the image, respectively, and finally averaging all the unshifted the images to obtain a unique improved denoised image remain the same. In view of this generalization, the fractal-based cycle spinning algorithm can be summarized as follows:

$$\hat{\mathbf{x}}_K = \frac{1}{K}\sum_{h=0}^{K} D_{-h}(FT(D_h(\mathbf{y}))). \tag{4.68}$$

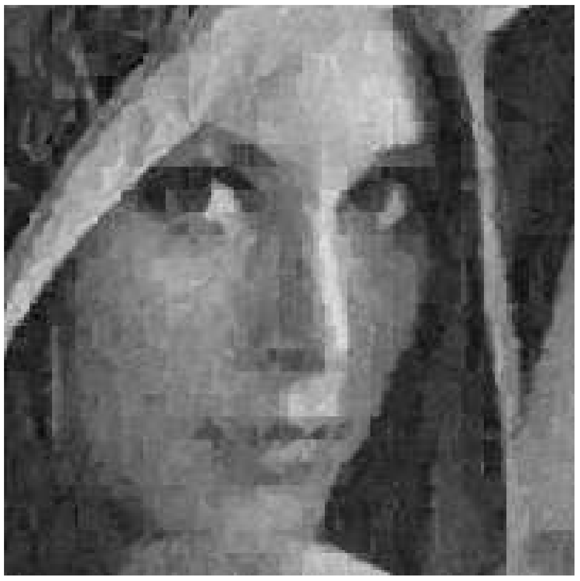
where D_h is the two-dimensional diagonal shifting operator D_h defined in (3.47) and FT is the fractal scheme of choice.

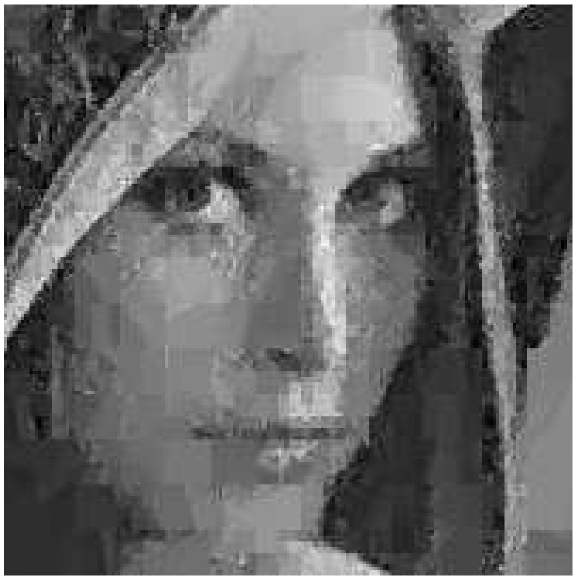
Once again when implementing the cycle spinning algorithm, with K shifts, for the purpose of enhancing the quality of the fractally denoised estimates the computational complexity of the fractal-based schemes increases by K times.

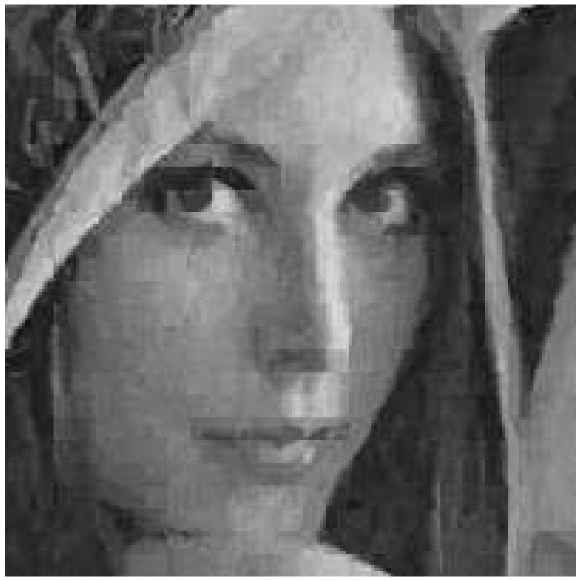
4.3.1 Experimental Results

The cycle spinning algorithm was incorporated in order to improve the performance of some of the fractal image denoising methods developed in this chapter. The results obtained for each of the fractal denoising schemes are outlined as follows:

- Figures 4.13 - 4.15 illustrate the results obtained by applying the above cycle spinning algorithm for the purpose of fractally encoding the noisy image using various fractal-based denoising schemes. The results are summarized in the following observations:
- For all the methods, examining the zoomed fractally denoised images, note that the disturbing blockiness artifacts in the denoised image are significantly reduced or eliminated. This is due to the fact that the various fractally denoised unshifted images are averaged out to obtain one


(a) Standard fractal coding (no maps)


(b) Quadtree-based fractal coding


(c) Predictive fractal denoising
(uniform partitioning)

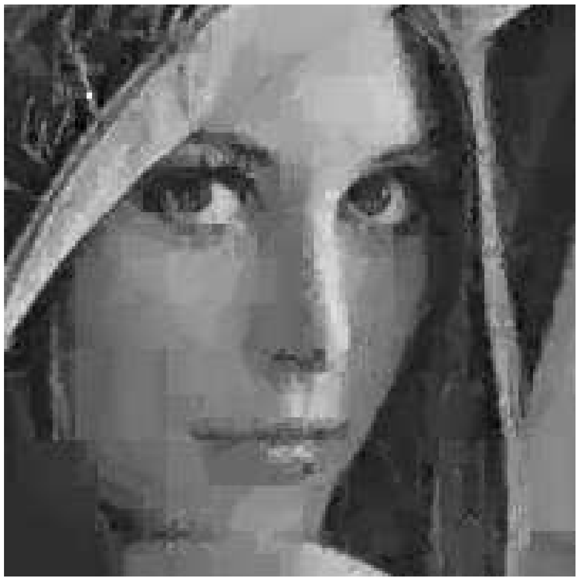

(d) Predictive fractal denoising
(quadtree partitioning)

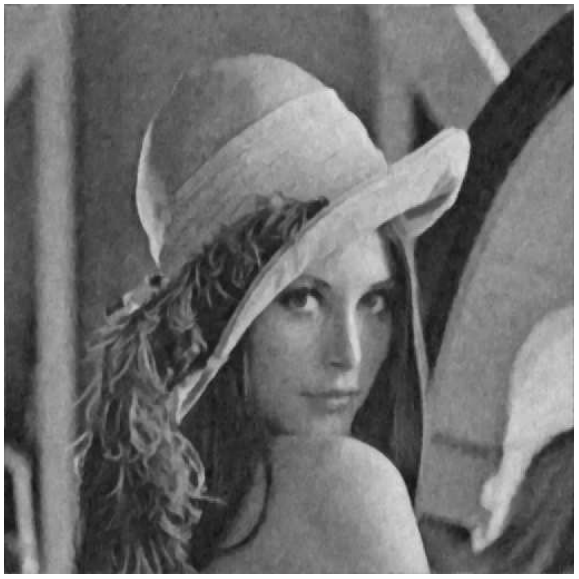
Figure 4.12: Zooming in on the fractally denoised estimates, as obtained by the various fractal schemes, reveals the disturbing artifacts in the fractal representations.

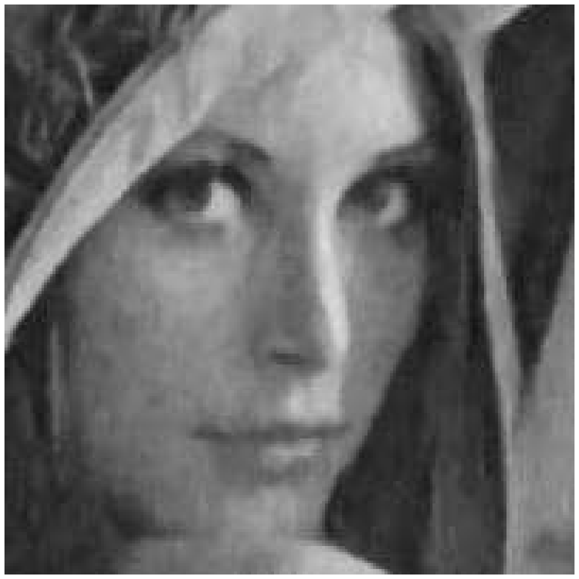
fractally denoised image. Averaging these fractal representations will reduce the variability around the boundaries of the fractal uniform image partition and hence result in significant reduction of the blockiness artifact in the averaged fractal representation.

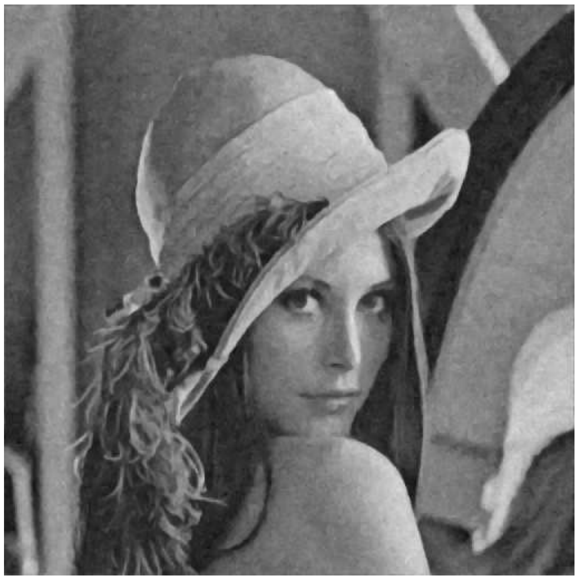
- Figure 4.16 illustrates a comparison between the various fractal-based denoising methods proposed in this chapter, when incorporating the cycle spinning idea. Note that the quadtree-based predictive scheme performs better than the other fractal denoising methods. Also note that the quality of the fractally denoised image stabilizes after only a few shifts, thus achieving fast convergence. This is the case because at the $(M, N) = (32, 64)$, which correspond to an 8×8 child blocks partitioning and 16×16 parent blocks partitioning, if the image is shifted exactly 8 pixels along its diagonal, then the new shifted image will have a partition that is equivalent to the partition of the original image. Thus the fractal codes of the original image and the one that is shifted by 8 pixels are equivalent. Hence, after unshifting the decoded image resulting from the shifted (by 8 pixels) image will be the same as the fractally decoded image from the original (unshifted) image, and hence no new information is gained. This argument can be extended to higher shifts and it can be concluded that the cycle spinning process will converge rapidly when using standard fractal coding and only a relatively small number of shifts is required to improve the quality of the fractal representation and reduce the blockiness artifacts. Performing a higher number of shifts will not degrade the quality of the fractally denoised image but will result in little or no improvement.

4.4 Summary and Concluding Remarks

In this chapter, various fractal-based schemes for image restoration have been developed and implemented. First, the noisy image was simply encoded using the uniform partitioning fractal scheme. It was observed that the standard fractal scheme performs too much smoothing for lower resolutions (M, N), producing a blurring of the edges as well as blockiness artifacts. At higher resolutions (M, N), an insufficient amount of smoothing is performed, resulting in a noisy fractal representation. This competition between quality and resolution can be resolved by using an adaptive partitioning scheme such as quadtrees, which allows different resolutions and block sizes to be used for different parts of an image. It permits a greater degree of smoothing away from edges (larger blocks) and a lesser degree of smoothing near edges (smaller blocks), hence minimizing their degradation. An attempt has been made to gain some insights into some of the reasons why fractally


(a) Standard fractal coding
uniform partitioning with $(M, N) = (32, 64)$
No geometric maps are tested
RMSE=9.04, PSNR=29.01.


(b) Zooming in on the image in (a).


(a) Standard fractal coding
uniform partitioning with $(M, N) = (32, 64)$
all 8 geometric maps are tested
RMSE=8.71, PSNR=29.33.

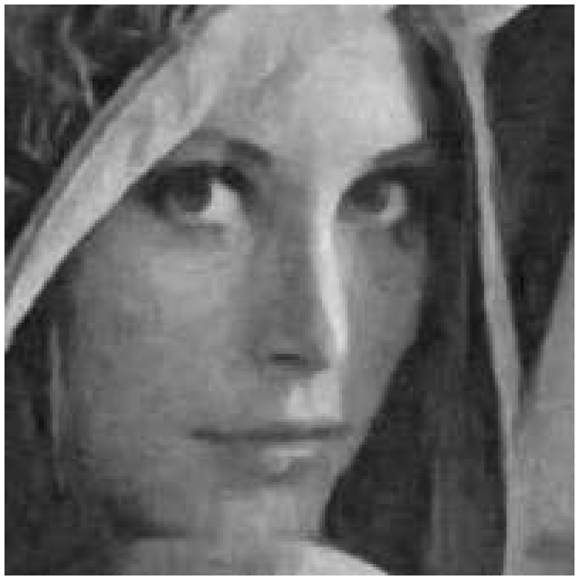
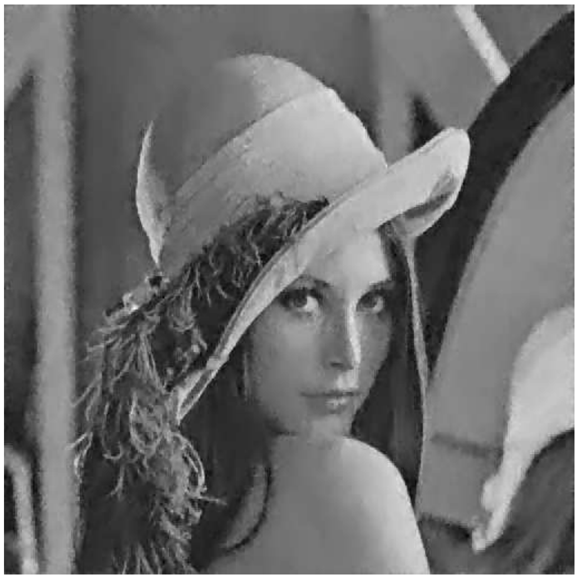
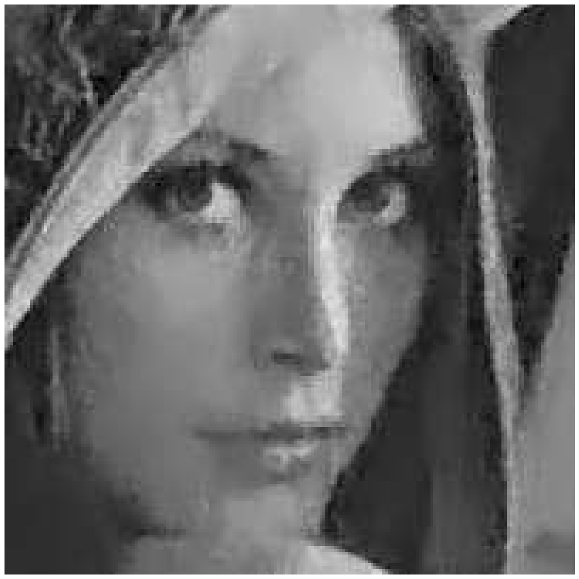
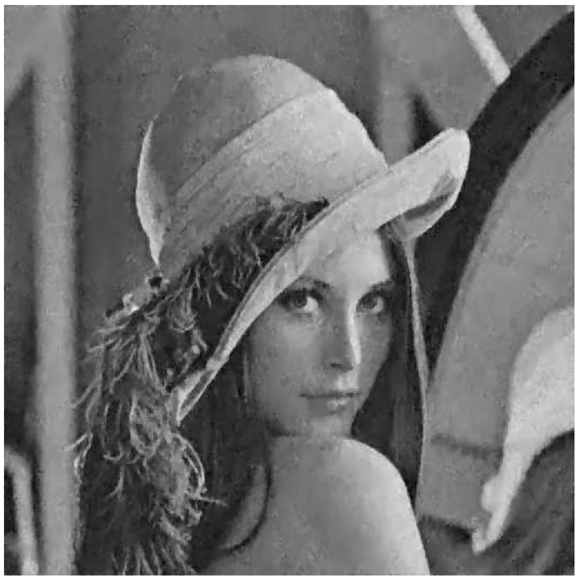

(b) Zooming in on the image in (a).

Figure 4.13: Denoised estimates using the *standard fractal* coding of the noisy image, *without* an well as *with* testing for the 8 geometric maps, when using the cycle spinning algorithm with $K = 16$ diagonal shifts.


(a) Quadtree-based fractal scheme
SNR decomposition criterion
RMSE=8.69, PSNR=29.35.


(b) Zooming in on the image in (a)


(a) Quadtree-based fractal scheme
collage error decomposition criterion
RMSE=8.73, PSNR=29.31.

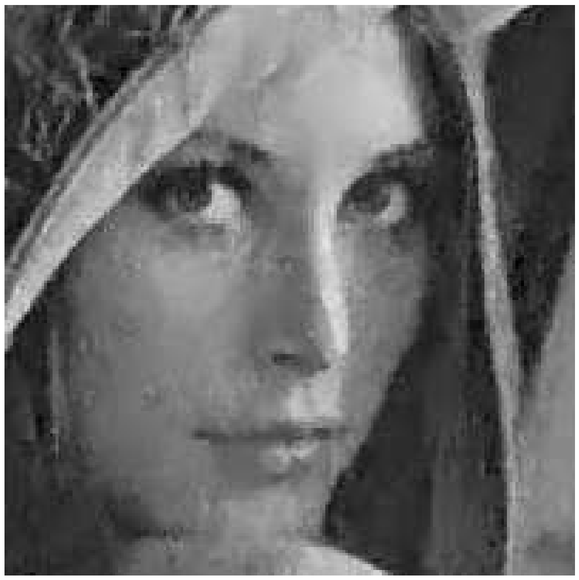
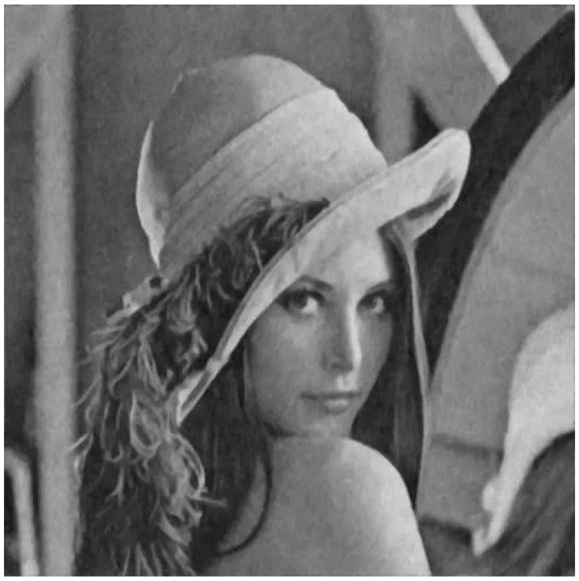
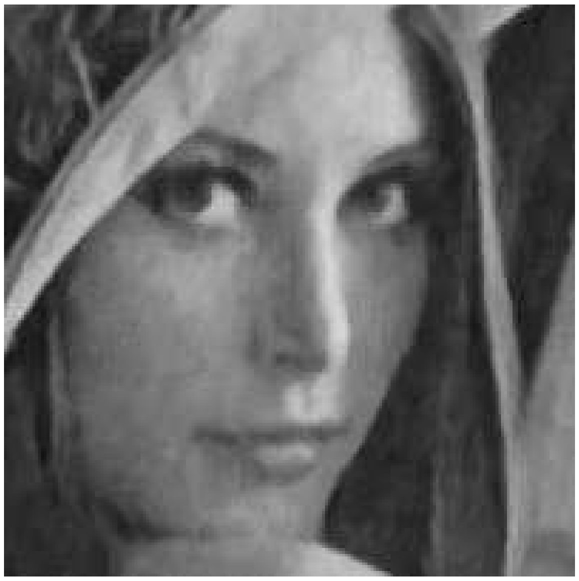

(b) Zooming in on the image in (a).

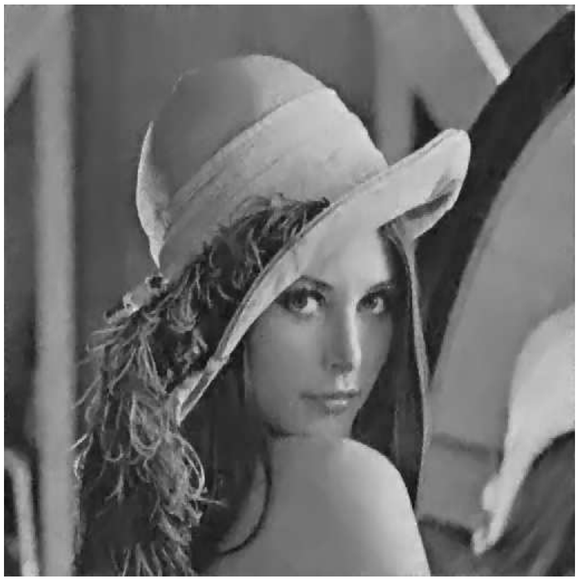
Figure 4.14: Denoised estimates using the *quadtree-based fractal* coding of the noisy image, using *signal-to-noise* (SNR) as well as *collage error* decomposition criterion when incorporating the idea of cycle spinning with $K = 16$ diagonal shifts.



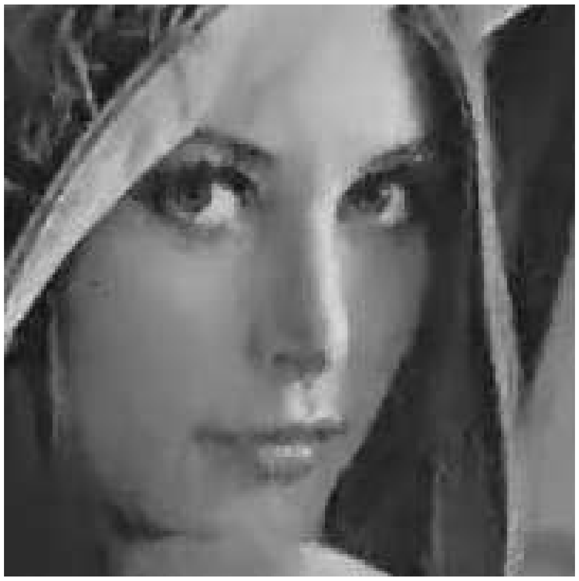
(a) Standard predictive fractal scheme
uniform partitioning with $(M, N) = (32, 64)$
RMSE=8.37, PSNR=29.68.



(b) Zooming in on the image in (a).

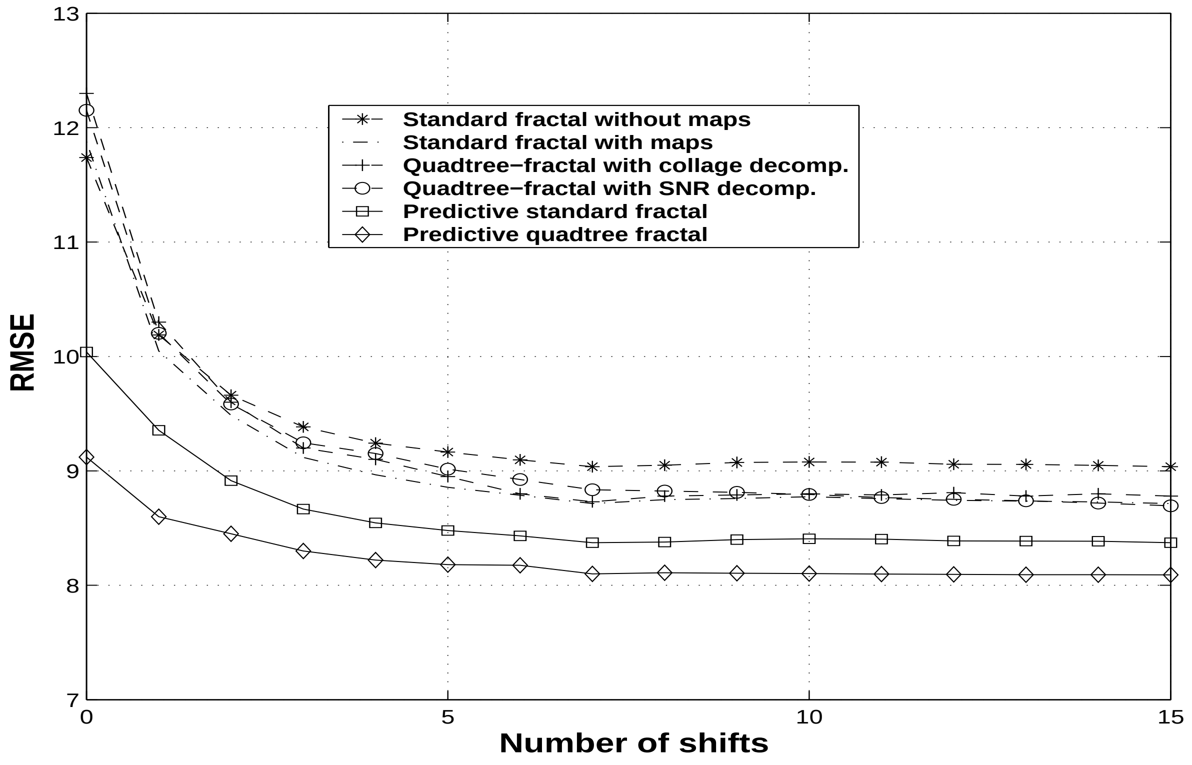


(c) Quadtree-based predictive fractal scheme
collage error decomposition criterion
RMSE=8.09, PSNR=29.97.



(d) Zooming in on the image in (c).

Figure 4.15: Denoised estimates using the *standard* as well as the *quadtree-based* predictive fractal denoising schemes when incorporating the cycle spinning algorithm with $K = 16$ diagonal shifts.



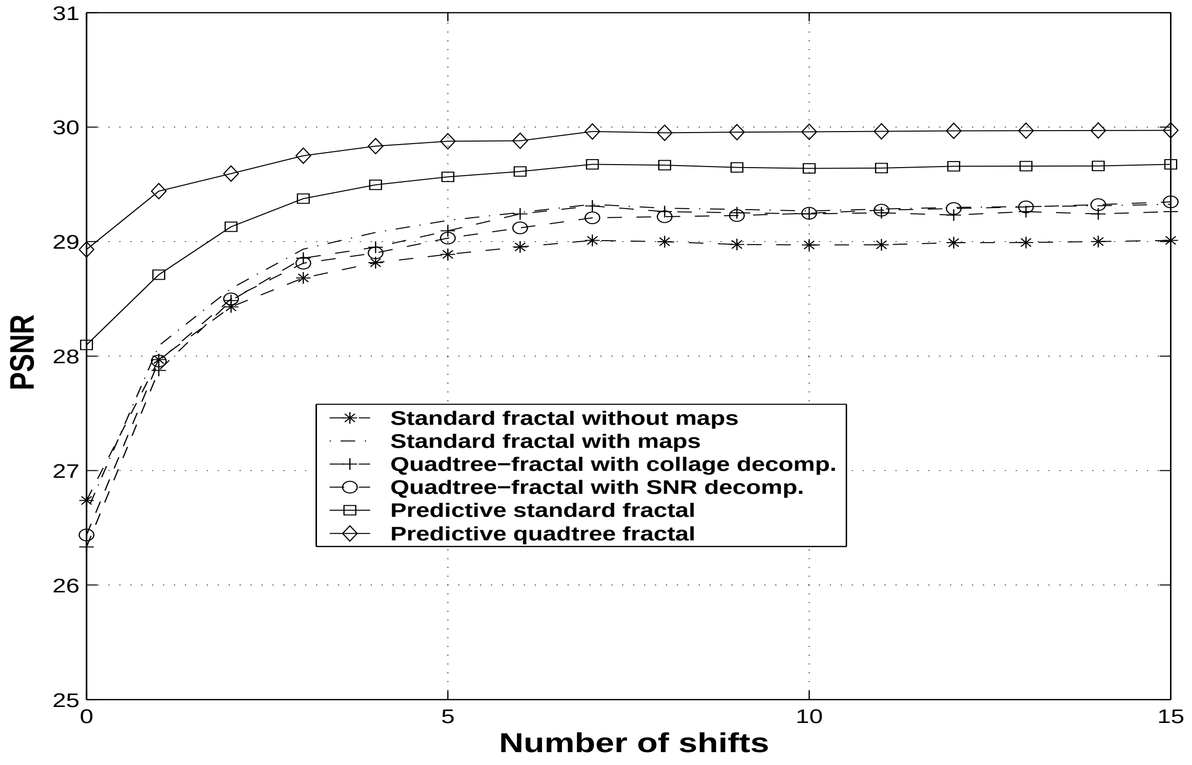


Figure 4.16: Comparison between the improvement achieved by incorporating the cycle spinning idea with K=16 diagonal shifts within the the various fractal-based denoising schemes studied in this chapter.

coding a noisy image leads to a significant degree of noise suppression. The decimation associated with the contractive spatial maps used in the fractal transform is probably responsible for most of the denoising.

A second major result of this chapter lies in the estimation of the fractal code of the noise-free image from the fractal code of the noisy image. However, it is not guaranteed that optimal parent-child pairings of the noisy image will coincide with optimal parent-child pairings of the noiseless image. In an effort to predict the latter, thereby producing a better fractal reconstruction of the noiseless image, a collage-based method employing the noise statistics was proposed. Experiments indicate that the method works very well, improving upon the other fractal-based methods and yielding results that are superior to those obtained by the Lee filter method. Using the quadtree-based fractal predictive scheme, with collage decomposition criterion, performs even better than the standard predictive scheme adopting uniform partitioning of the image. Another possible advantage of the proposed fractal denoising scheme over the Lee filter is that the former can, if desired, perform compression of the noisy image, unlike the latter.

The incorporation of the cycle spinning idea for the purpose of reducing the blockiness artifacts and improving the overall quality of the fractally denoised estimates was also proposed and implemented. It was shown that significant gains are achieved by using the cycle spinning idea at the expense of an increase in the computational complexity.

In this chapter, the task of investigating and developing fractal-based image denoising methods, as applied in the spatial domain of the noisy image, was investigated. This investigative process followed in this chapter has a fractal-wavelet analogue. In particular, the extraction of the fractal code of the original image from noisy one, can be performed for the fractal-wavelet transform. In other words, the predictive fractal denoising algorithm will be extended for the purpose of predicting the fractal-wavelet code of the DWT of the original image from the DWT of the noisy one. This analogous task is treated in the next chapter, in some detail.

Chapter 5

Fractal-Wavelet Image Denoising

In this chapter, the potential of applying various fractal-wavelet (FW) schemes for the purpose of image denoising will be investigated. This is going to be done in an manner that is analogous to the pure fractal-based methods, as detailed in chapter 4. As it was the case for the spatial-based fractal schemes, it will be shown that when the wavelet transform of the noisy image is simply fractally encoded, using any of the fractal-wavelet schemes described in chapter 2, a significant amount of the noise is suppressed. The use of the adaptive as well as the quadtree-based fractal-wavelet schemes for the purpose of image denoising will also be explored.

A simple yet effective method to estimate the fractal-wavelet code of the wavelet transform of the original noise-free image from the statistics of the wavelet transform of the noisy image will be derived. From this fractal-wavelet code, one can then generate a FW denoised estimate of the original noiseless image. This predictive fractal-wavelet image denoising scheme is analogous to the predictive pure fractal image denoising scheme proposed in chapter 4. Even better results can be obtained when using a hybrid fractal-wavelet image denoising scheme that makes use of quadtree partitioning scheme in the wavelet domain as well as adaptive thresholding of the stored wavelet coefficients. It will be shown that this hybrid FW denoising scheme yields estimates that are significantly denoised while preserving the sharpness of the edges and other high frequency features of the image. The denoising and restoration achieved by the proposed hybrid quadtree-based FW denoising scheme was found to be consistent with the human visual system where extra smoothing is performed in flat and low activity regions and a lower degree of smoothing is performed near high frequency components, e.g. edges, of the image.

Furthermore, ways of improving the performance of the proposed fractal-wavelet image denoising methods even further will be investigated. In particular, the cycle spinning algorithm will be applied in order to enhance the fractal-wavelet denoised estimates. Incorporating this idea in the various fractal-wavelet denoising methods results in significant reduction of the pseudo-Gibbs and ringing artifacts that are inherent in a FW denoised estimate and improves its overall visual and quantitative quality considerably.

The layout of this chapter is as follows: First, the potential of applying fractal-wavelet schemes for the purpose of image denoising by simply encoding the noisy images using various FW schemes is investigated. A predictive FW scheme for the purpose of image denoising will be outlined and implemented in section 2. The incorporation of the cycle spinning idea in order to improve the FW denoised estimates, obtained by the various FW schemes is detailed in section 3. A brief summary and conclusions will be given in the last section of this chapter.

5.1 Image Denoising using Fractal-Wavelet Coding

In chapter 2, the potential of applying standard fractal coding as applied in the spatial domain of an image for the purpose of image denoising was investigated. It was shown that significant noise reduction can be achieved by simply fractally encoding a noisy image. In this section, the potential of fractal denoising in the wavelet domain of the noisy image will be investigated. The use of various fractal-wavelet schemes for the purpose of image denoising will be investigated. First, the noisy image is simply encoded using the studied fractal-wavelet schemes. It will be shown that a significant amount of the noise is suppressed. Further improvement in image quality can be achieved by applying the adaptive as well as the quadtree-based fractal wavelet schemes described in chapter 2. Similar to the quadtree-based fractal scheme in the spatial domain, It will be shown that the enhancement achieved by the quadtree-based fractal-wavelet scheme is consistent with the characteristics of the human visual system. Extra denoising is performed in flat and low activity regions and a lower degree of smoothing is performed near high frequency components, hence preserving the sharpness of edges in the image. Similarities that may exist between the fractal-wavelet denoising and the wavelet thresholding techniques will also be explored.

5.1.1 Conventional Fractal-Wavelet Coding for Image Denoising

In chapter 2, various fractal-wavelet schemes were studied and implemented. In particular, two distinct and effective schemes were considered, namely the the exhaustive FW scheme (FW-I) which treats the three subbands independently and the the standard FW scheme (FW-II), which combines the three subbands. These schemes were described in Table 2.2.

The noisy test image used in this chapter is the same standard test image of "Lenna" that has been degraded by an AWGN with standard deviation $\sigma_w = 25$, used throughout this work. These FW schemes have been implemented to encode the noisy image of "Lenna" and the results are illustrated in Figure 5.1 and summarized as follows:

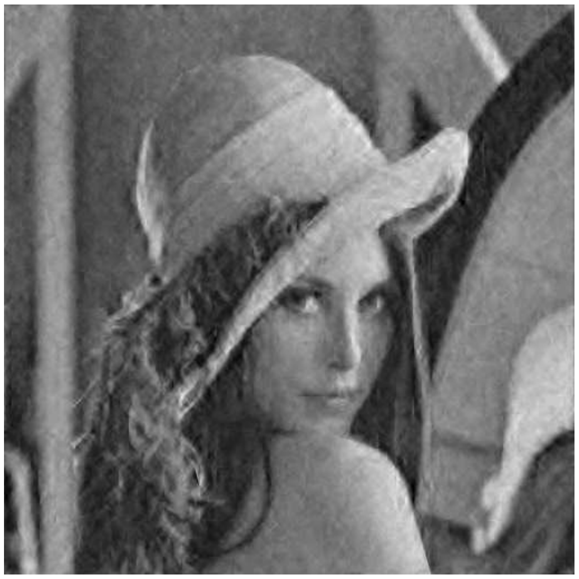
- Note that significant but varying degree of noise reduction has been achieved by the two FW schemes at the $(k_1^*, k_2^*) = (4, 5)$ as well as the $(k_1^*, k_2^*) = (5, 6)$. The best result, as reflected by the RMSE and the PSNR, is obtained when using the Standard FW scheme at the $(k_1^*, k_2^*) = (5, 6)$ resolution.

- At the lower FW resolution, $(k_1^*, k_2^*) = (4, 5)$, both schemes yield overly smoothed denoised estimates where most of the noise has been suppressed but also at the expense of smoothing other high frequency content of the image such as edges. Disturbing ringing and blurring artifacts are also observed throughout the denoised estimate.

- For the higher FW resolution, $(k_1^*, k_2^*) = (5, 6)$, sharper images are obtained but also some degree of residual noise that is still present in the image is observed. This is observed especially when the exhaustive FW-I scheme is used, the denoised estimate appears sharp but visibly noisy.

- Clearly, for both FW schemes, it is observed that there is a consistent trade-off between the resolution and the quality of the FW representation. On one hand, for lower resolution (k_1^*, k_2^*), excessive smoothing is performed which results in blurring of edges and less sharp images with ringing and blurring artifacts. On the other hand, at higher resolution (k_1^*, k_2^*), not enough denoising is performed while the sharpness of the image is preserved, resulting in sharp but noisy images.

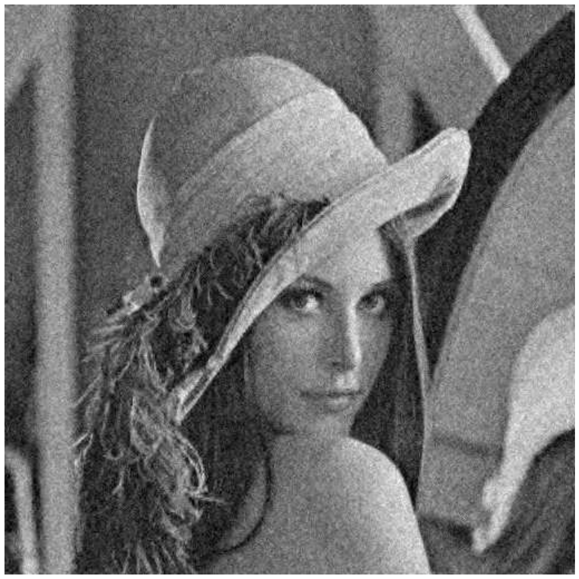
The observed trade-off between the selection of the FW resolution, $(k_1^*, k_2^*) = (4, 5)$, and the quality of the denoised estimate is reminiscent of the trade-off observed between the selection of the threshold, λ, and the quality of the denoised estimate obtained by wavelet thresholding method. This apparent connection between the two methods will be explored in more detail next.



(a) Exhaustive FW-I scheme : $(k_1^*, k_2^*) = (4, 5)$

RMSE =12.01 , PSNR = 26.53.

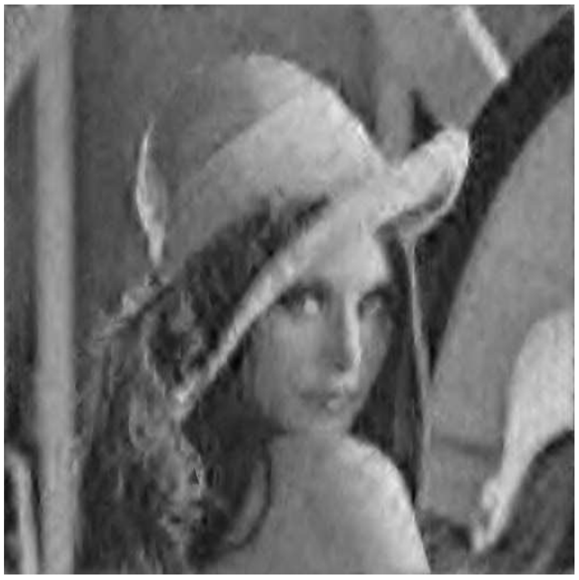
Execution time ≈ 52 secs.



(b) Exhaustive FW-I scheme: $(k_1^*, k_2^*) = (5, 6)$

RMSE = 15.35, PSNR = 24.41.

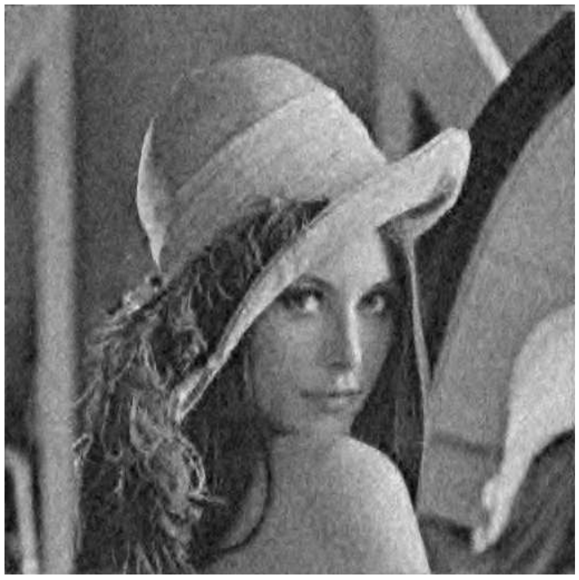
Execution time ≈ 93 secs.



(c) Standard FW-II scheme: $(k_1^*, k_2^*) = (4, 5)$

RMSE = 13.67, PSNR = 25.42.

Execution time ≈ 19 secs.



(d) Standard FW-II scheme: $(k_1^*, k_2^*) = (5, 6)$

RMSE = 11.32, PSNR = 27.05.

Execution time ≈ 65 secs.

Figure 5.1: Denoised estimates obtained by simply encoding the noisy image using the Exhaustive (FW-I) and the Standard (FW-II) schemes. Note that no quantization of the FW scaling coefficients is performed.

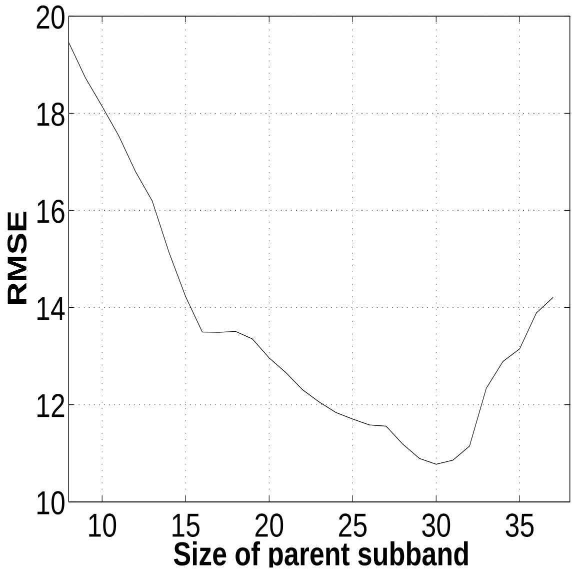
Similarities Between Fractal-Wavelet Denoising and Wavelet Thresholding

In view of the above observations, it can be noted that the trade-off that exists between the FW resolution (k_1^*, k_2^*) and the quality of the FW denoised estimate is rather similar to that observed trade-off between the selection of the value of the threshold λ and the quality of the denoised estimate. For low resolution (k_1^*, k_2^*), the FW encoding method yields overly denoised representation where most of the noise has been suppressed at the expense of smoothing the edges of the image and creating artificial ringing artifacts. This is indeed analogous to the results observed for the wavelet thresholding method when a critical threshold that is too high is used. On the other hand, for higher resolution (k_1^*, k_2^*), FW coding of the noisy image does not perform enough denoising, resulting in sharp but visibly noisy representations. Again, this observation is consistent with the results obtained by the wavelet thresholding method when the used threshold is too low. This establishes an apparent similarity between the FW coding and the wavelet thresholding methods for image denoising, studied in chapter 3. The analogy lies the fact that the selection of the optimal FW resolution (k_1^*, k_2^*) is equivalent to the selection of the optimal critical threshold λ^* for the wavelet thresholding method. In order to achieve denoised images while preserving the sharpness of the image, the wavelet thresholding method seeks to choose an optimal threshold λ^*. Similarly, to achieve the best denoised FW representation of the original image, one seeks to obtain the optimal intermediate resolution between the lower FW resolution $(k_1^*, k_2^*) = (4, 5)$ and the higher FW resolution $(k_1^*, k_2^*) = (5, 6)$.

As seen in chapter 2, there are many different strategies for selecting effective thresholding strategies for the purpose of image denoising. Adaptive thresholds that vary with the subbands and levels of the wavelet tree perform significantly better than uniform thresholding. For the FW coding schemes, different degrees of denoising can be achieved by varying the FW resolution. Next, the selection of the best FW resolution (k_1^*, k_2^*) will be explored by using the adaptive fractal-wavelet scheme described in chapter 2.

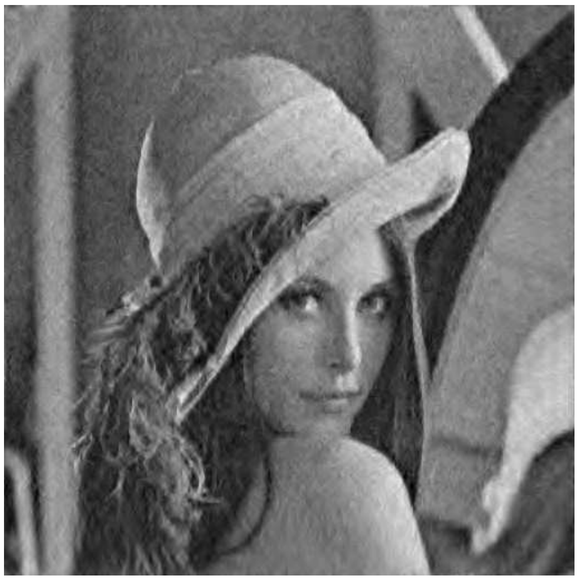
5.1.2 Adaptive Fractal-Wavelet Coding for Image Denoising

Recall that the adaptive fractal-wavelet, scheme described in chapter 2, yields fractal-wavelet representations at any intermediate partition level between any two FW resolutions (k_1^*, k_2^*) and $(k_1^* + 1, k_2^* + 1)$. As seen in the previous section, increasing the resolution from $(k_1^*, k_2^*) = (4, 5)$ to $(k_1^*, k_2^*) = (5, 6)$ results in significantly different results that illustrate a clear trade-off between the noise reduction and the sharpness of the FW representation. The adaptive FW scheme yields FW representations at various intermediate partition levels of the wavelet coefficients. This adaptive FW scheme is less restrictive than the original FW schemes in the sense that the parent and child blocks do not have to be restricted, in size or location, to the various wavelet decomposition subbands and levels of the wavelet tree.



(a) Quality of denoised estimate vs. FW resolution

Optimal FW resolution: 30×30 parent blocks



(b) Best adaptive FW denoised estimate

RMSE = 10.77, PSNR = 27.48.

Execution time $\approx$ 47 secs.

Figure 5.2: (a) The quality of the adaptive FW scheme denoised estimate of "Lenna" for different FW resolutions, and (b) the best FW denoised estimate obtained when the parent blocks contains 30×30 coefficients.

Figure 5.2 (a) illustrates the dependence of the quality of the adaptive FW denoised estimate on the size of the parent sub-block, as the FW resolution varies from $(k_1^*, k_2^*) = (3, 4)$ to $(k_1^*, k_2^*) = (5, 6)$. Note that this figure illustrates the trade-off between the quality of the FW denoised estimate and the FW resolution (k_1^*, k_2^*) quite well. For low FW resolution, the quality of the denoised image is poor mainly because not many wavelet coefficients have been stored and most of the wavelet coefficients are estimated from the relatively small number of stored wavelet coefficients. However, as the FW resolution increases, the quality of the denoised estimate improves until a critical resolution is reached, beyond which the quality starts to degrade. Figure 5.2 (b) illustrates the best representation using adaptive FW scheme with an intermediate level corresponding to the critical resolution with 30×30 parent sub-blocks and 60×60 child sub-blocks. If the FW resolution is increased even further, the quality of the FW denoised estimate starts to get worse because more noisy coefficients are stored and thus more of the noise is reconstructed and the FW reconstruction appear noisier, and hence with higher RMSE and lower PSNR.

Figure 5.3 further illustrates the similarities between the wavelet thresholding method and the adaptive FW scheme. Note that the quality of the FW denoised estimate of the original image depends on the FW resolution, as described by the size of the parent sub-block, in the same manner as the quality of the wavelet thresholding estimate of the original image depends on the selection of the critical threshold. This clearly supports the analogy between the two methods that has been suggested earlier.

A different way to interpret the trade-off between the fractal resolution and the quality of the FW denoised estimate is as a trade-off between the number of wavelet coefficients that are stored, as they are, and the number of those wavelet coefficients that are fractally encoded and estimated during the decoding process from the ones that have been stored and the FW code. One way to adaptively control this trade-off is to use the quadtree-based FW scheme. Next, the use of the quadtree-based fractal-wavelet scheme for the purpose of image denoising and assess its performance will discussed and implemented.

5.1.3 Quadtree-Based Fractal-Wavelet Coding for Image Denoising

In chapter 2, the use of the quadtree-partitioning approach for the purpose of partitioning the wavelet tree and performing fractal-wavelet image coding was described. The hierarchical quadtree partitioning scheme for the purpose of fractal-wavelet image denoising can be described as follows: Consider a subtree of wavelet coefficients, A_{kij}^{sub}, that is rooted at the coefficient a_{kij}^{sub}, $sub \in \{h, v, d\}$. The FW quadtree-based scheme examines such a subtree and decides, on the basis of a prescribed criterion, whether or not such a subtree should be FW encoded. If it turns out that the subtree should not be encoded, then the node or root, a_{kij}^{sub}, is stored and the subtree A_{kij}^{sub} is replaced by four subtrees that are rooted at the four children of the original node, a_{kij}^{sub}. Otherwise the subtree is encoded using the FW scheme of choice.

Various decomposition criteria for the hierarchical quadtree partitioning scheme have been investigated. For the purpose of image denoising, a *signal-to-noise ratio* (SNR) decomposition

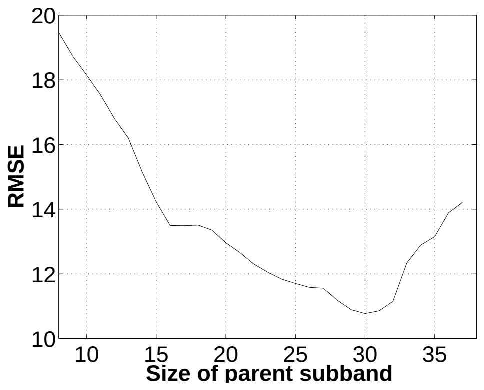
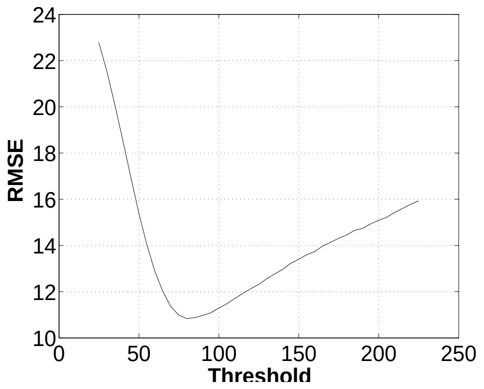
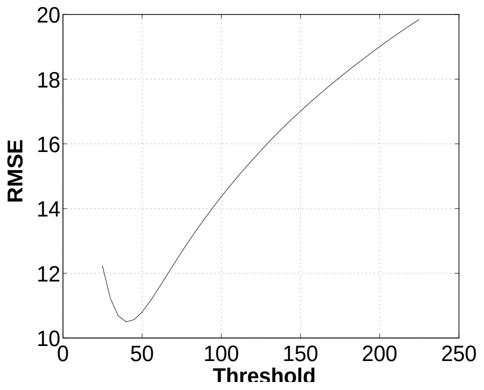
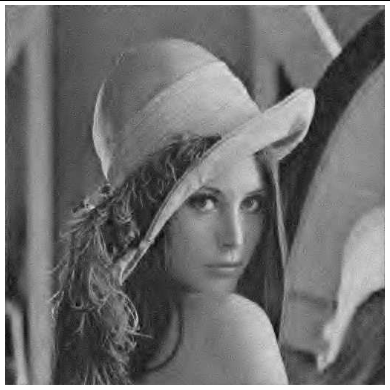
Scheme	Quality curves	Optimal results
Adaptive FW	 (a) Quality vs. Resolution Optimal parent size: 30×30	(b) RMSE = 10.77, PSNR = 27.48.
Hard thresh.	 (c) Quality vs. Threshold Optimal threshold : $\lambda^*_{hard} = 80$	(d) RMSE=10.98, PSNR=27.31.
Soft thresh.	 (e) Quality vs. Threshold Optimal threshold : $\lambda^*_{soft} = 40$	 (f) RMSE=10.57, PSNR=27.65.

Figure 5.3: Comparison between the adaptive FW coding and the VisuShrink wavelet thresholding for image denoising.

criterion is used. The SNR, γ, of a noise-free child subtree $\mathbf{Y} = [\mathbf{A}_{kij}^{sub}]$ is computed as follows:

$$\gamma = \frac{\sigma_{\mathbf{Y}}^2}{\sigma_{\mathbf{w}}^2}. \tag{5.1}$$

However, the noiseless child block, $\mathbf{Y} = [\mathbf{A}_{kij}^{sub}]$, is not available and only its corresponding noisy version, $\hat{\mathbf{Y}} = [\hat{\mathbf{A}}_{kij}^{sub}]$, is observed. Since the noise is assumed to be AWGN and independent of the original image, then the variances, $\sigma_{\hat{\mathbf{Y}}}^2$ and $\sigma_{\mathbf{Y}}^2$, of the noisy and the noise free subtrees, respectively, are related as follows:

$$\sigma_{\hat{\mathbf{Y}}}^2 = \sigma_{\mathbf{Y}}^2 + \sigma_{\mathbf{w}}^2, \tag{5.2}$$

or equivalently

$$\sigma_{\mathbf{Y}}^2 = \sigma_{\hat{\mathbf{Y}}}^2 + \sigma_{\mathbf{w}}^2. \tag{5.3}$$

Thus the SNR, γ, can be computed from the noisy image as follows:

$$\gamma = \frac{\sigma_{\mathbf{Y}}^2}{\sigma_{\mathbf{w}}^2} \approx \frac{\sigma_{\hat{\mathbf{Y}}}^2 - \sigma_{\mathbf{w}}^2}{\sigma_{\mathbf{w}}^2} \approx \frac{\sigma_{\hat{\mathbf{Y}}}^2}{\sigma_{\mathbf{w}}^2} - 1. \tag{5.4}$$

However, this estimate may be negative since $\sigma_{\hat{\mathbf{Y}}}^2$ is computed locally. Thus, a more reasonable estimate of the SNR is as follows:

$$\gamma \approx \max\{\frac{\sigma_{\hat{\mathbf{Y}}}^2}{\sigma_{\mathbf{w}}^2} - 1, 0\}. \tag{5.5}$$

Using this SNR, the quadtree-based FW image denoising scheme can be outlined next.

Quadtree-Based FW Image Denoising Algorithm

For a given prescribed SNR threshold, γ_c:

- For each, uncoded, noisy child subtree, $\hat{\mathbf{Y}} = [\hat{\mathbf{A}}_{kij}^{sub}]$, compute its SNR, as given by Eq. (5.5)

  $$\gamma \approx \max\{\frac{\sigma_{\hat{\mathbf{Y}}}^2}{\sigma_{\mathbf{w}}^2} - 1, 0\}, \tag{5.6}$$

  and compare it to γ_c.

  - If $\gamma \geq \gamma_c$, then store the root of the subtree and partition the subtree into four subtrees rooted at each of the its four children that are marked as *not coded.*
  - Otherwise, the subtree is simply fractally encoded, using any of the fractal-wavelet schemes and marked as *coded.*

As discussed next, the quadtree-based FW scheme is intuitively appealing because it attempts to exploit some of the characteristics of the human visual system.

The Quadtree-Based FW Scheme and the Human Visual System

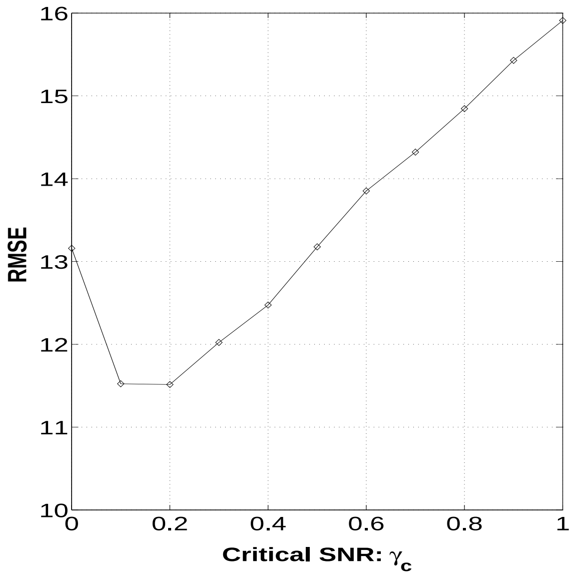
In principle, the human visual system is less sensitive to noise near edges and more sensitive to noise in flatter regions of the image. This motivates the investigation the quadtree partitioning scheme which permits the use of different resolutions or subtree sizes for different parts of the wavelet tree. In particular, subtrees with high signal to noise ratio, γ, generally contain edge information or other high frequency content of the image. Thus, some or most of the significant wavelet coefficients within such a subtree will be stored. On the other hand, subtrees with low signal to noise ratio generally correspond to flat regions of the image. Fractally encoding such a subtree results in significant noise reduction, as was shown earlier. Thus, the quadtree based FW scheme performs little denoising near edges and other high activity regions of the image. However, the presence of an acceptable amount of noise in these high frequency regions of the fractal representation may not be a problem due to the fact that the human visual system is less sensitive to noise near edges. On the other hand, the FW encoding of subtrees with low signal to noise ratio, γ, results in significant noise reduction in flat and low activity regions of the image. These favorable characteristics of the quadtree-based FW scheme will be exploited for the purpose of image denoising while preserving its sharpness.

Experimental Results

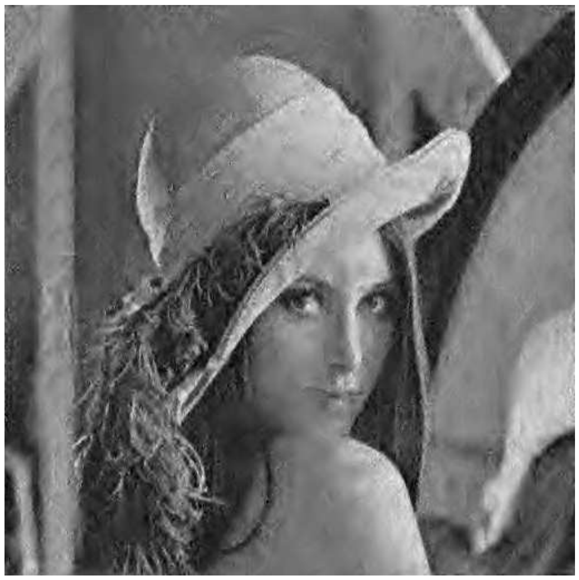
Whenever using the quadtree partitioning algorithm for the purpose of fractal image coding, one has to choose an optimal decomposition criterion. In this case, the "optimal" value, γ_c, for the SNR threshold decomposition criterion has to be determined. Similar to the spatial case, the value was determined experimentally. Figure 5.4 (a) illustrates the dependence of the quality of the quadtree-based FW denoised image on the selection of the prescribed signal-to-noise ratio threshold, γ_c. Note that $\gamma_c \approx 0.2$ yields the best results, which is similar to the critical SNR threshold, $\gamma_c^{'} \approx 0.25$ used for the quadtree-based standard fractal image denoising scheme, as performed in the spatial domain of the noisy image and discussed in detail in section 4.1.2. From Eq. (5.5), this seems to indicate that one needs to partition a subtree $\hat{Y}$ if

$$\sigma^2_{\hat{\mathbf{Y}}} \geq 1.2\sigma^2_{\mathbf{w}}. \tag{5.7}$$

Figure 5.4 (b) illustrates the quadtree-based FW representation of the noisy image corresponding to the optimal critical signal-to-noise ratio threshold, $\gamma_c = 0.2$. Note that quantitatively, the RMSE fidelity measure of the denoised estimate is relatively high. However, visually, the edges of the image appear sharp and most of the background and other low activity areas of the image have been overly smoothed. Taking a closer look, one could still notice the presence of noise near edges and other sharp features of the images. However, naturally the noise is less perceivable and mixed with the other high frequency features of the image.



(a) Quality of denoised estimate vs. SNR optimal SNR: $\gamma_c \approx 0.2$



(b) The best FW-quadtree denoised estimate RMSE = 11.53, PSNR = 26.89. Execution time ≈ 117 secs.

Figure 5.4: (a) The dependence of the quality of the quadtree-based FW denoised image on the selection of the SNR threshold γ_c, and (b) the best quadtree-based FW denoised estimate, obtained using $\gamma_c = 0.2$.

When encoding the wavelet tree of the noisy image using any of the FW schemes, a relatively high number of noisy wavelet coefficients are stored. Thus, during the decoding process, these noisy wavelet coefficients along with the FW code are used to generate the FW estimate. Consequently, the FW estimate will often have some degree of noise. Next, a hybrid FW and wavelet thresholding scheme that denoises the stored wavelet coefficients using a suitable thresholding method before they are stored is proposed.

5.1.4 Hybrid Quadtree-Based Fractal-Wavelet Coding and Thresholding

In this section, the use of a hybrid quadtree-based FW and thresholding scheme that combines the use of FW coding as well as thresholding of stored wavelet coefficients is discussed. Recall that for the various FW schemes, a certain number of wavelet coefficients has to be stored. For instance, for any of the original FW schemes, one has to store $2^{k_2^*} \times 2^{k_2^*}$ wavelet coefficients, when using a FW resolution of (k_1^*, k_2^*). Even more coefficients are stored when the quadtree-based FW is used. However, when encoding the wavelet tree of a noisy image using any of the FW schemes, these stored wavelet coefficients are noisy. During the FW decoding process, these stored wavelet coefficients along with the FW code, will be used to re-generate an estimate of the original image. Consequently, the FW denoised estimate will generally contain some degree of noise. Note that this is quite different from the case of encoding the noisy image using any of the standard fractal schemes which are applied in the spatial domain of the noisy image. Recall that for standard fractal image coding, the fractal decoder generally starts with a blank, noise-free image, and re-generates the fractal estimate of the original image by iterating the fractal code on the initial blank image seed.

Recall that in wavelet thresholding, one often thresholds only the first few highest decomposition levels. Typically, only coefficients at decomposition level j, $6 \leq j \leq 9$ are thresholded, for the 512×512 test image of "Lenna". The reason behind this is that first few lowest decomposition levels, $1 \leq j \leq 5$ contain mainly the low frequency content of the image and thus they are not significantly affected by the noise, which is a high frequency data that is embedded within the high frequency wavelet coefficients of the image. For these reasons, the thresholding process should not be applied at the lower frequency subbands of the wavelet coefficients tree. When applying the fractal-wavelet schemes with fixed levels (k_1^*, k_2^*), one is storing $2^{k_2^*} \times 2^{k_2^*}$ wavelet coefficients that are mainly located at the lowest decomposition levels of the wavelet tree, when k_2^* is small. Thus most of these stored coefficients are not significantly affected by the noise and there is no need to threshold them. It was observed that thresholding these coefficients is counter productive and yields slightly worse results than without any thresholding. However, in the case of the quadtree-based fractal-wavelet scheme described above, the stored wavelet coefficients may come from anywhere in the wavelet decomposition tree. Thus, one would expect that it would be beneficial to threshold the stored wavelet coefficients that originate from high-frequency levels and subbands. Indeed it was observed that this is the case, as will be seen next.

Thresholding Strategy

The hybrid FW quadtree-based thresholding scheme proposes denoising the stored wavelet coefficients by applying a suitable thresholding strategy. The level-dependent thresholding scheme will be used because it has shown to perform well and it is computationally simple and does not require estimating the local statistics of the wavelet tree, as is the case for the SureShrink and BayesShrink. Thus, if a stored wavelet coefficient is positioned in level j of the wavelet decomposition tree of size M, then the level-dependent threshold is given by

$$\lambda_j = \sqrt{2\ln(M)}\sigma_w 2^{-(J-j)/2}, \text{ for } j = 1, 2, \ldots, J. \tag{5.8}$$

The use of soft as well as hard thresholding operators using level-dependent thresholds will be investigated.

Experimental Results

This hybrid wavelet denoising scheme was implemented for the purpose of enhancing the performance of the quadtree-based fractal-wavelet image denoising scheme studied in the previous section. Figure 5.5 illustrates the results obtained by applying the quadtree-based fractal wavelet scheme along with hard and soft thresholding of the stored wavelet coefficients, located at higher frequencies ($j \geq 6$). Comparing these results to those obtained by simply applying the quadtree-based FW scheme, as illustrated in Figure 5.4 (b), note that the use of the proposed thresholding strategy to denoise the stored wavelet coefficients has in fact resulted in a slight improvement of the quality of the FW denoised estimate. This improvement is visually noticeable as the new estimates appear less noisy and exhibit a lesser degree of ringing and blurring artifacts. The RMSE and PSNR fidelity measures also reflect this improvement in the quality of the quadtree-based FW denoised estimates.

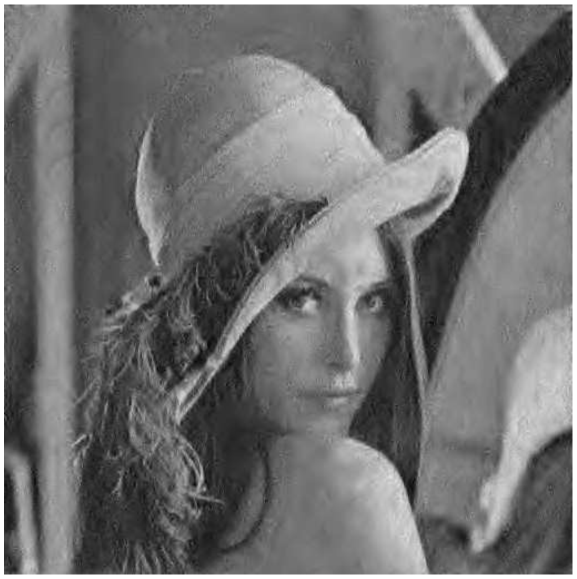
In summary, the quadtree partitioning scheme permits more smoothing away from edges and lesser smoothing near edges, thus achieving the smoothing and denoising of an image without degrading its edges. Also performing some degree of level-dependent thresholding of the stored wavelet coefficient will result in an overall less noisy FW estimate, even in the vicinity of edges.

5.1.5 Observations

So far in this section, it has been shown that, similar to the spatial-based fractal methods discussed in chapter 2, simply encoding a noisy image using any of the conventional, adaptive or quadtree-based fractal-wavelet schemes results in significant noise reduction. Again this is because natural image structures generally possess similarities across resolution scales of their wavelet coefficients, which normally can exploited for fractal-wavelet coding. However, noisy structures have no resemblance across resolution levels and therefore cannot be represented well using fractal-wavelet coders. Thus, encoding a noisy image using a fractal-wavelet coder results in good reconstruction of the natural, self-similar structures, whereas the noisy contents cannot be re-generated. Some of the advantages of using the fractal-wavelet schemes to denoise images include:

- The fractal-wavelet schemes eliminate the blockiness artifacts in the denoised image that are inherent in block-based standard fractal schemes, as applied in the spatial domain of the image. However, especially for lower (k_1^*, k_2^*), the denoised image often suffers from pseudo-Gibbs and ringing artifacts. Ways to significantly reduce or eliminate these artifacts will be studied later in this chapter.

- Fractal-wavelet schemes are also computationally less expensive than their standard fractal counterparts. Also, as was shown in the last section, one may combine fractal-wavelet schemes with other wavelet-based denoising methods, such as wavelet thresholding, to yield a better hybrid fractal-wavelet and thresholding scheme for image denoising.


(a) Hybrid quadtree-based FW: *hard* thresholding
RMSE=11.13, PSNR=27.20.
Execution time $\approx$ 127 secs.

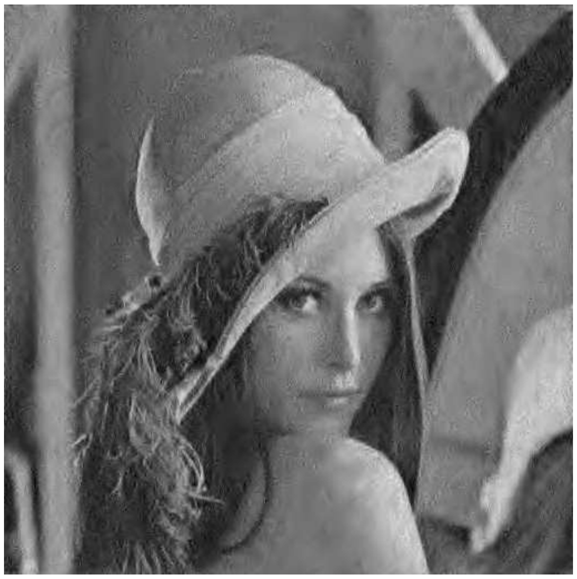

(b) Hybrid quadtree-based FW: *soft* thresholding
RMSE=11.18, PSNR=27.17.
Execution time $\approx$ 132 secs.

Figure 5.5: The results obtained by applying the *hard-thresholding* and *soft-thresholding* of the stored wavelet coefficients for the quadtree-based fractal-wavelet scheme, using the signal-to-noise decomposition threshold $\gamma_c = 0.2$.

Next, a method for predicting the fractal-wavelet code of the original noise-free image from the noisy image will be proposed.

5.2 A Predictive Fractal-Wavelet Image Denoising Technique

In this section, the relationship between the wavelet transform of the noisy image and its noiseless counterpart, will be examined. This relationship will provide a method of estimating the fractal-wavelet code for the wavelet tree of the original noiseless image from the statistics of the noisy image. This FW code is then used to generate a FW denoised estimate of the original noise-free image.

Recall that the FW scheme proceeds as follows: given a fixed set of parent-child level values (k_1^*, k_2^*):

1. For each *uncoded* child quadtree $A^{sub}_{k_2^*,i,j}$, $sub \in \{h, v.d\}$ and $i, j = 1, 2, \ldots, 2^{k_2^*}$:
   - Find the parent quadtree $A^{sub}_{k_1^*,i',j'}$ for which the collage distance associated with that child, namely,
   $$\Delta^{sub}_{i,j,i',j'} = ||A^{sub}_{k_2^*,i,j} - \alpha^{sub}_{i,j,i',j'} A^{sub}_{k_1^*,i',j'}||_2^2, \tag{5.9}$$
   is minimized.
2. Store the wavelet coefficients $\hat{c}_{i,j}$, for $1 \leq i, j \leq 2^{k_1^*}$.

For simplicity of notation let the range subtree of an arbitrary wavelet decomposition tree $R_{i,j} = A^{sub}_{k_2^*,i,j}$ be represented by the vector $\mathbf{y} = [y_1, y_2, \ldots, y_n]$ and the domain subtree $D_{i',j'} = A^{sub}_{k_1^*,i',j'}$ of the same wavelet tree be represented by the by the vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. The determination of the optimal scaling coefficient α is a simple least-squares problem. In practice, the $\mathcal{L}^2$ norm is used so that the optimal scaling map performs a least-squares fit of the parent-child wavelet coefficients data, and the minimization of the squared $\mathcal{L}^2$ (i.e. mean squares) error

$$\Delta^2_{i'(i,j),j'(i,j)} = \frac{1}{n} \sum_{m=1}^{n} [y_m - \alpha x_m]^2 \tag{5.10}$$

yields a single linear equation in α with solution

$$\alpha^* = \frac{\frac{1}{n}\sum_{m=1}^{n} x_m y_m}{\frac{1}{n}\sum_{m=1}^{n} x_m^2}. \tag{5.11}$$

Before proceeding further, it will be useful to rewrite the least-squares scaling coefficients in (5.11) in terms of standard statistical quantities. The wavelet transform of an image can be regarded as a random signal so that, in general, the wavelet coefficients, $\mathbf{x}=\{x_m, m = 1, 2, \ldots, n\}$ and $\mathbf{y}=\{y_m, m = 1, 2, \ldots, n\}$, can be considered as random samples drawn from the random variables X and Y representing the wavelet coefficients distribution of a parent subtree D and its corresponding child subtree R. The least-squares scaling coefficient can then be written as

$$\alpha^* = \frac{E[XY]}{E[X^2]} \tag{5.12}$$

where

$$E[XY] = \frac{1}{n}\sum_{m=1}^{n} x_m y_m, \quad \text{and} \quad E[X^2] = \frac{1}{n}\sum_{m=1}^{n} x_m^2. \tag{5.13}$$

Strictly speaking, the above expressions are approximations to the statistical quantities of the random variables X and Y since they represent (finite) sample statistics. For large n, the sample statistics provide good estimates of the population statistics. The fact that n will not be large in our applications will contribute to errors in estimating the local image statistics and, subsequently, sub-optimal fractal codes for the noiseless images. In what follows, variables and coefficients that correspond to a noisy image will have hats, e.g. $\hat{X}, \hat{Y}$ for the noisy image as opposed to X, Y for the noiseless image.

Next, an expression for the scaling coefficients corresponding to the noisy image will be derived. A relationship between the scaling coefficients corresponding to the noisy and the noise-free images will be established.

5.2.1 Derivation of the Scaling Coefficients for the Noiseless Image

As above, let X and Y denote the random variables representing the the wavelet coefficients values in a parent subtree D and its corresponding child subtree R, respectively, corresponding to the noise-free original image. Also, let $\hat{X}$ and $\hat{Y}$, denote the random variables corresponding to wavelet coefficients of the noisy image.

For the wavelet transform of the noisy image, similarly to (5.12), the least-squares scaling coefficients are given by

$$\hat{\alpha}^* = \frac{E[\hat{X}\hat{Y}]}{E[\hat{X}^2]}, \tag{5.14}$$

where

$$\hat{X} = X + W_X, \quad \hat{Y} = Y + W_Y, \tag{5.15}$$

where W_X and W_Y are independent and identically distributed $N(0, \sigma^2_{\mathbf{w}})$; additive white Gaussian noise (AWGN) processes. Assuming that the image and the noise signals are independent, then the second moment of the noisy random variable $\hat{X}$ is

$$\mathrm{E}[\hat{X}^2] = \mathrm{E}[(X + W_X)^2] = E[X^2] + \sigma^2_{\mathbf{w}}. \tag{5.16}$$

Also, under the independence assumption between the noise and the image signals, it can be shown that

$$\mathrm{E}[\hat{X}\hat{Y}] = E[(X + W_X)(Y + W_Y)] = E[XY]. \tag{5.17}$$

From (5.14), (5.16) and (5.17), the scaling coefficient $\hat{\alpha}^*$ can be expressed in terms of the statistics of the noiseless wavelet transform of the image and the noise variance as follows:

$$\hat{\alpha}^* = \frac{E[XY]}{E[X^2] + \sigma^2_{\mathbf{w}}}. \tag{5.18}$$

Or equivalently,

$$\hat{\alpha}^* = \frac{\frac{E[XY]}{E[X^2]}}{1 + \frac{\sigma^2_{\mathbf{w}}}{E[X^2]}}. \tag{5.19}$$

From this result and (5.12), it can be shown that

$$\hat{\alpha}^* = \frac{\alpha^*}{1 + \frac{1}{\gamma}}, \text{ or equivalently } \alpha^* = (1 + \frac{1}{\gamma})\hat{\alpha}^*, \tag{5.20}$$

where the *signal-to-noise ratio* is defined as

$$\gamma = \frac{E[X^2]}{\sigma^2_{\mathbf{w}}}. \tag{5.21}$$

From Eq. (5.16), the signal-to-noise ratio can be estimated from the statistics of the noisy image, as follows:

$$\gamma = \frac{E[\hat{X}^2]}{\sigma^2_{\mathbf{w}}} - 1. \tag{5.22}$$

provided that:

$$E[\hat{X}^2] > \sigma^2_{\mathbf{w}}. \tag{5.23}$$

Eq. (5.20) provides a relationship between the scaling coefficients, α^* and $\hat{\alpha}^*$, corresponding to the original and the noisy images, respectively. This relationship will be explored in more detail in the next sections.

Remarks

In view of the above derivations, one can make the following observations:

- Examining the signal-to-noise ratio, γ, note that if the energy of the subtree is significantly larger than the noise variance, i.e. $E[X^2] >> \sigma^2_{\mathbf{w}}$, then $\gamma >> 1$ and the content of the subtree can be considered as important. In other words, this subtree contains edge information and other high frequency features of the original image. Thus the noise is insignificant compared to the signal information. In this case one has

  $$\hat{\alpha^*} \approx \alpha^*, \text{ when } \gamma >> 1, \tag{5.24}$$

  In the limit, when the image is noise-free, then one has the following:

  $$\hat{\alpha}^* \to \alpha^*, \text{ as } \gamma \longrightarrow \infty. \tag{5.25}$$

  which is reasonable, as the FW codes of the noisy and the original images will be closer together when the noise is negligible (i.e. small $\sigma^2_{\mathbf{w}}$) and rather distinct for dominant noise (large $\sigma^2_{\mathbf{w}}$).

Next, a method for estimating the the collage error corresponding to the the noiseless image from the noise one is derived.

5.2.2 Derivation of the Collage Error for the Noiseless Image

The results of the previous section suggest an algorithm to fractally denoise an image. First, fractally encode the DWT of the noisy image to obtain the scaling coefficients $\hat{\alpha}^*$. Use (5.3) to estimate the scaling coefficients, α^*, corresponding to the DWT of the noiseless image. There is one problem, however: It is not guaranteed that the parent-child assignments of the wavelet transform of the noisy image are optimal for the wavelet transform of the noiseless image in the mean-squared error sense, i.e. that the MSE is minimized for the original image. A method to ensure that optimal collage coding is being performed for the noiseless image will now be proposed next.

Our objective is to estimate the collage error corresponding to the noise-free wavelet transform of the image as computed from the statistics of the noisy wavelet transform of the image. Recall that the collage error for the DWT of the noiseless image corresponding to the child subtree Y_k, its corresponding parent subtree X_i and resulting scaling coefficient α^*_{ik} is given by

$$\Delta^2_{ik} = E[(Y_k - \alpha^*_{ik} X_i)^2]. \tag{5.26}$$

Expanding the above quantity yields

$$\Delta_{ik}^2 = E[Y_k^2] + \alpha_{ik}^{*2} E[X_i^2] - 2\alpha_{ik}^* E[X_i Y_k].$$

Recall that

$$\hat{X}_i = X_i + W_i, \quad \hat{Y}_k = Y_k + W_k. \tag{5.27}$$

Then, as shown earlier in equations (5.16) and (5.17), the above collage error can be expressed in terms of the statistics of the noisy image as follows:

$$\Delta_{ik}^2 = (E[\hat{Y}_k^2] - \sigma_{\mathbf{w}}^2) + \alpha_{ik}^{*2}(E[\hat{X}_i^2] - \sigma_{\mathbf{w}}^2) - 2\alpha_{ik}^* E[\hat{X}_i \hat{Y}_k]. \tag{5.28}$$

provided that:

$$E[\hat{Y}_k^2] > \sigma_{\mathbf{w}}^2 \text{ and } E[\hat{X}_i^2] > \sigma_{\mathbf{w}}^2. \tag{5.29}$$

This provides an approach to estimate the collage error corresponding to the wavelet transform of a noisy image as computed from the statistics of the wavelet transform of its observed noisy version.

Next, an approach for estimating the FW code corresponding to the noise-free wavelet transform from the noisy wavelet transform is outlined.

5.2.3 Predicting the FW Code of the Original Image from the Noisy Image

In view of the above derivations and discussion, an approach for predicting the fractal-wavelet code for the wavelet transform of the original noise-free image from the wavelet transform of the noisy image is outlined next.

For each *uncoded* child subtree $\hat{Y}_k \in \mathcal{R}$, the range subtrees, of the wavelet transform of the noisy image, the following steps are performed:

1. Compute its energy:

$$E[\hat{Y_k}^2] = \frac{1}{n} \sum_{m=1}^{n} \hat{y}_{k,m}^2. \tag{5.30}$$

2. Then get an estimate of the energy of the corresponding subtree in the wavelet transform of the original noiseless image as follows:

$$\mathcal{E}_{Y_k} = E[{Y_k}^2] = E[\hat{Y_k}^2] - \sigma_{\mathbf{w}}^2. \tag{5.31}$$

In theory, $\mathcal{E}_{Y_k}$ must be positive. However in practice this will not always be the case since $E[\hat{Y}_k^2]$ is computed locally from subtrees with a relatively small number of coefficients, so it is possible to encounter cases where

$$E[\hat{Y}_k^2] < \sigma_{\mathbf{w}}^2 \tag{5.32}$$

resulting in negative values for $\mathcal{E}_{Y_k}$. Thus, there are two cases: $E[\hat{Y}^2] > \sigma_{\mathbf{w}}^2$ and $E[\hat{Y}^2] \leq \sigma_{\mathbf{w}}^2$. More specifically, to avoid cases where the estimate of the energy is zero, one should consider the following two cases: $E[\hat{Y}^2] \geq \lambda\sigma_{\mathbf{w}}^2$ and $E[\hat{Y}^2] < \lambda\sigma_{\mathbf{w}}^2$, where the parameter $\lambda > 1$, to be determined experimentally.

3. For each possible parent subtree $\hat{X}_i \in \mathcal{D}_k$, the domain pool of Y_k, compute the least-squares scaling coefficient, $\hat{\alpha}_{ik}$, corresponding to the DWT of the noisy image

$$\hat{\alpha}_{ik}^* = \frac{E[\hat{X}_i\hat{Y}_k]}{E[\hat{X}_i^2]}, \tag{5.33}$$

as given by (5.14).

Then compute the signal-to-noise ratio, as predicted from the DWT of the noisy image as follows:

$$\gamma \quad = \quad \frac{E[\hat{X_i}^2]}{\sigma_{\mathbf{w}}^2} - 1. \tag{5.34}$$

Again, in theory, γ must be positive, however in practice this will not alway be the case since and $E[\hat{X}_i^2]$ is computed locally from subtrees with a relatively small number of coefficients, so it is possible to encounter cases where

$$E[\hat{X}_i^2] < \sigma_{\mathbf{w}}^2, \tag{5.35}$$

resulting in negative values for γ values. Thus, again there are two cases: $E[\hat{X}_i^2] > \sigma_{\mathbf{w}}^2$ and $E[\hat{X}_i^2] \leq \sigma_{\mathbf{w}}^2$. More specifically, to avoid cases where the estimate of the energy is zero, one should consider the following two cases: $E[\hat{X}_i^2] \geq \lambda\sigma_{\mathbf{w}}^2$ and $E[\hat{X}_i^2] < \lambda\sigma_{\mathbf{w}}^2$, for the same parameter $\lambda > 1$, chosen above. Thus, overall there are two cases, which are treated in detail as follows:

(a) **Case 1**: If $E[\hat{Y}_k^2] \geq \lambda\sigma_{\mathbf{w}}^2$ and $E[\hat{X}_i^2] \geq \lambda\sigma_{\mathbf{w}}^2$, then:

- One has

$$\mathcal{E}_{Y_k} = E[\hat{Y_k}^2] - \sigma_{\mathbf{w}}^2 \geq (\lambda - 1)\sigma_{\mathbf{w}}^2 > 0 \text{ and } \gamma = \frac{E[\hat{X}^2]}{\sigma_{\mathbf{w}}^2} - 1 \geq \lambda - 1 > 0. \tag{5.36}$$

- Now, since the current parent-subtree, X_i is fixed, then one is using the same child-parent assignment (Y_k, X_i) for the noise-free image and $(\hat{Y}_k, \hat{X}_i)$ for the noisy image, then, the scaling coefficient, α^*, corresponding to the wavelet tree of the original image is related to the scaling coefficient, $\hat{\alpha}^*$, corresponding to the noisy image by

$$\alpha^*_{ik} = (1 + \frac{1}{\gamma})\hat{\alpha}^*_{ik}, \tag{5.37}$$

as shown in (5.3). This yields an estimate of the scaling coefficient α^*.

- Now, use this estimate of α^* to compute the collage error, measured in terms of the mean squared error for the wavelet tree corresponding to the original noiseless image given by

$$\Delta^2_{ik} = (E[\hat{Y}^2_k] - \sigma^2_{\mathbf{w}}) + \alpha^{*2}_{ik}(E[\hat{X}^2_i] - \sigma^2_{\mathbf{w}}) - 2\alpha^*_{ik}E[\hat{X}_i\hat{Y}_k]. \tag{5.38}$$

as shown in Eq. (5.28). This insures a collage-based matching criterion for the DWT of the original noise-free image. This completes the first case, the case when the energy of the child-subtree is small is now considered.

(b) **Case 2**: If $E[\hat{Y_k}^2] < \lambda\sigma^2_{\mathbf{w}}$ or $E[\hat{X}^2_i] < \lambda\sigma^2_{\mathbf{w}}$, then:

- One may assume that child or parent subtree is dominated by the noise and the corresponding subregion of the original image is mainly flat and low-activity and contains little relevant information. In [7], it was suggested to reduce the value of $\hat{\alpha}^*$ to get a better estimate of α^*. Indeed, it is beneficial to reduce the magnitude of $\hat{\alpha^*}$ to obtain a distribution that resembles the distribution of the α^*. Besides, larger values of $\hat{\alpha}^*$ would amplify the noise. The following modification is suggested: of $\hat{\alpha^*}$:

$$\alpha^* \approx \min(\frac{E[\hat{Y}^2_k]}{\lambda\sigma^2_{\mathbf{w}}}, \frac{E[\hat{X}^2_i]}{\lambda\sigma^2_{\mathbf{w}}})\hat{\alpha}^*, \tag{5.39}$$

to obtain an estimate of α^*.

- Also, in this case one cannot use the collage error in Eq. (5.28), derived for the noise-free image, because the estimate:

$$E[Y^2_k] \approx E[\hat{Y}^2_k] - \sigma^2_{\mathbf{w}} \text{ or } E[Y^2_k] \approx E[\hat{X}^2_i] - \sigma^2_{\mathbf{w}}, \tag{5.40}$$

may be negative. Thus, in this case one needs to resort to using the collage error, corresponding to the wavelet transform of the noisy image:

$$\Delta^2_{ik} = E[\hat{Y}^2_k] + \alpha^{*2}_{ik}E[\hat{X}^2_i] - 2\alpha^*_{ik}E[\hat{X}_i\hat{Y}_k]. \tag{5.41}$$

4. Select the optimal parent subtree $i^*(k)$ such that

$$\Delta^2_{i^*(k),k} \leq \Delta^2_{i,k} \quad \text{for all} \quad i \neq i^*(k). \tag{5.42}$$

Clearly, one may apply any of the FW schemes discussed in the previous section. However, only the FW-II scheme will be applied because it combines the three subbands, hence resulting in sufficiently large child and parent subtrees and consequently better estimates of the required subtree statistics, such as $E[\hat{X}^2]$ and $E[\hat{Y}^2]$ are obtained.

In view of the above detailed outline, the proposed predictive fractal-wavelet image denoising scheme can be summarized as follows:

The Proposed Predictive FW Image Denoising Algorithm

Choose the standard FW scheme (FW-II) and its resolution (k_1^*, k_2^*), then: For each *uncoded* child subtree $\hat{Y}_k$, $k = 1, 2, \ldots, 2^{k_2^*} \times 2^{k_2^*}$:

1. Compute the energy of the child subtree

$$E[\hat{Y}_k^{\,2}] = \frac{1}{n} \sum_{m=1}^{n} \hat{y}^2_{k,m}. \tag{5.43}$$

2. For each possible parent subtree $\hat{X}_i \in \mathcal{D}_k$, the domain pool of Y_k, compute:

   - The scaling coefficient, $\hat{\alpha}^*$, corresponding to the DWT of the noisy image, as given by (5.14):

$$\hat{\alpha}^*_{ik} = \frac{E[\hat{X}_i \hat{Y}_k]}{E[\hat{X}_i^2]}, \tag{5.44}$$

   - The energy:

$$E[\hat{X}_i^{\,2}] = \frac{1}{n} \sum_{m=1}^{n} \hat{x}^2_{i,m}. \tag{5.45}$$

   Now, consider one of the following two cases:

   (a) **Case 1**: If $E[\hat{Y}_k^{\,2}] \geq \lambda \sigma^2_{\mathbf{w}}$ and $E[\hat{X}_i^{\,2}] \geq \lambda \sigma^2_{\mathbf{w}}$, then:

   - Compute the signal-to-noise ratio, γ, corresponding to the original noise-free image, which can be estimated by

$$\gamma \;=\; \frac{E[\hat{X}_i^{\,2}]}{\sigma^2_{\mathbf{w}}} - 1. \tag{5.46}$$

- Predict the scaling coefficient corresponding to the DWT of the noise-free image as follows:

$$\alpha^* \quad = \quad (1 + \frac{1}{\gamma})\hat{\alpha}^*. \tag{5.47}$$

- Compute the collage error, optimized for the noise-free image, as given by

$$\Delta_{ik}^2 = (E[\hat{Y}_k^2] - \sigma_{\mathbf{w}}^2) + \alpha_{ik}^{*2}(E[\hat{X}_i^2] - \sigma_{\mathbf{w}}^2) - 2\alpha_{ik}^* E[\hat{X}_i \hat{Y}_k]. \tag{5.48}$$

(b) **Case 2**: If $E[\hat{Y_k}^2] < \lambda\sigma_{\mathbf{w}}^2$ or $E[\hat{X_i}^2] < \lambda\sigma_{\mathbf{w}}^2$, then:

- Estimate the scaling coefficient, α^*, corresponding to the DWT of the noise-free image, from $\hat{\alpha}^*$, as follows:

$$\alpha^* \approx \min(\frac{E[\hat{Y}_k^2]}{\lambda\sigma_{\mathbf{w}}^2}, \frac{E[\hat{X}_i^2]}{\lambda\sigma_{\mathbf{w}}^2}) \times \hat{\alpha^*}, \tag{5.49}$$

- Compute the collage error corresponding to the noisy image, as given by

$$\Delta_{ik}^2 = E[\hat{Y}_k^2] + \alpha_{ik}^{*2} E[\hat{X}_i^2] - 2\alpha_{ik}^* E[\hat{X}_i \hat{Y}_k]. \tag{5.50}$$

3. Select the optimal parent subtree, $i^*(k)$, such that:

$$\Delta_{i^*(k),k}^2 \leq \Delta_{i,k}^2 \quad \text{for all} \quad i \neq i(k). \tag{5.51}$$

The predicted fractal-wavelet code corresponding to the original noise free consisting of:

$$\textit{Fractal-Wavelet code} = \{i^*(k), \alpha_{i^*(k)}^*, k = 1, 2, \ldots, 2^{k_2^*} \times 2^{k_2^*}\} \cup \{\hat{c}_{i,j}, \text{ for } 1 \leq i, j \leq 2^{k_2^*}\}. \tag{5.52}$$

which can then be used by the FW decoder to generate a denoised estimate of the original image. In view of the above derivations, it can be observed that:

- Although the above algorithm is outlined for the purpose of using the standard FW-II scheme, it is indeed a straightforward matter to generalize it to other FW schemes, such as the quadtree-based FW scheme. One should use a FW that combines the three subbands in order for the child and parent subtrees to have sufficiently large size. Otherwise, poor estimates of the local statistics may lead to poor results.

- As for the selection of λ, the same value, $\lambda = 2$, will be used for spatial-based fractal predictive scheme. It was observed that this value yields good results.

In this section, an algorithm for predicting and extracting the fractal-wavelet code of the noise-free wavelet transform of the original image from the wavelet transform of the noisy image was proposed.

Next, the above FW predictive scheme is applied for the purpose of denoising the test image and the results are illustrated.

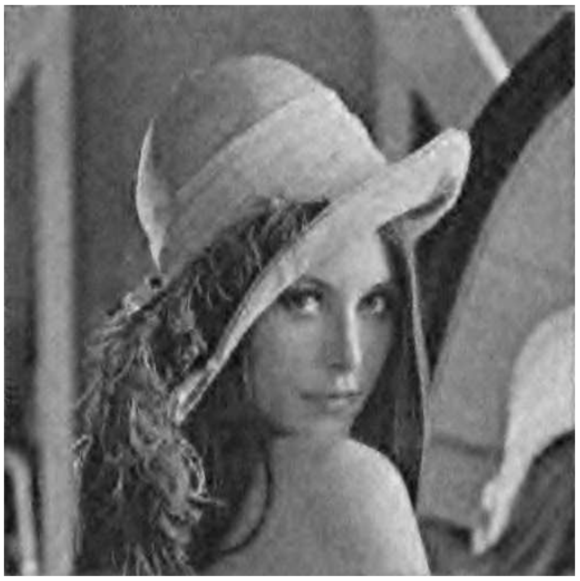
5.2.4 Experimental Results

Figure 5.6 illustrates the results obtained by the predictive fractal-wavelet denoising scheme when the standard FW-II is used with FW resolutions: $(k_1^*, k_2^*) = (5, 6)$ as well as the quadtree-based predictive FW scheme. For the quadtree-based scheme, it was observed that it is beneficial to threshold the stored wavelet coefficients that are located at the highest few decomposition levels, i.e. $5 \leq j \leq 9$. The context-based, level-dependent thresholding strategy discussed in detail in chapter 2, was applied. A collage error decomposition criterion for the quadtree-based FW predictive scheme was also employed. These results are summarized in the following observations:

- In parts (a) and (b), the predictive FW-II denoised estimate and its zoomed version are illustrated. Clearly, the use of the predictive FW scheme has resulted in significant noise reduction. As shown earlier, simply encoding the noisy image using the standard FW scheme at various resolutions results in significant noise reduction. For instance, simply encoding the noisy image using the FW-II scheme with resolution $(k_1^*, k_2^*) = (5, 6)$, results in denoised estimate, as illustrated in Figure 5.1 (d). However, when using the predictive FW-II scheme with the same resolution clearly results in further noise reduction and overall improvement of the denoised estimate. Zooming in on the FW denoised estimate reveals ringing and pseudo-Gibbs artifacts instead of the blockiness artifacts that were evident is standard fractal denoised estimates. Ways of reducing these artifacts will be investigated in the next section.

- Parts (c) and (d) illustrate the results obtained when using the quadtree-based hybrid predictive FW denoising scheme which adopts a level-dependent context-based soft thresholding strategy of the stored wavelet coefficients. Note that, the quadtree-based predictive FW scheme has resulted in further improvement in the quality of the predictive FW denoised estimate. In fact the quality of this hybrid FW denoised estimate is better than the results obtained by any of the wavelet thresholding methods, including SureShrink and BayesShrink which were discussed and implemented in chapter 3.

- When implementing the quadtree-based predictive FW denoising algorithm, a collage error decomposition criterion was used. A child subtree is only fractally encoded using the predictive FW scheme if the resulting collage error is less than a desired error tolerance level. Otherwise, its root is stored and the rest of the subtree is split into four child subtrees rooted at the next decomposition level. By doing so, the predictive fractal denoising scheme performs noise reduction on one hand. On the other hand, the use of the quadtree, with the collage error decomposition criterion, insures that important features (i.e. edges) of the original image are represented well, by storing more significant wavelet coefficients and using finer partitioning when necessary, to fit the subtrees well enough and guarantee a specified fitting error. Consequently, the proposed quadtree-based FW predictive scheme, with collage decomposition criterion, not only denoises the image, it also attempts to recover most of the important features of the original image in order to yield sharp denoised estimates.


(a) Predictive FW-II scheme
$(k_1^*, k_2^*) = (5, 6)$
RMSE = 10.31, PSNR = 27.86
Execution time $\approx$ 69 secs.

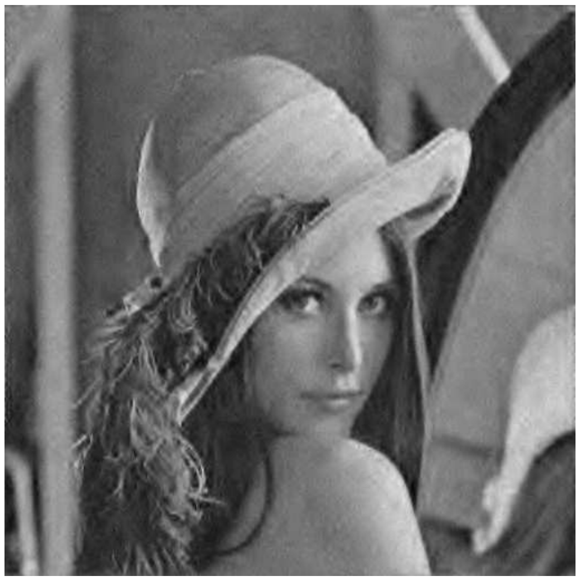

(b) Predictive quadtree-based FW-II scheme
with collage error decomposition criterion
RMSE = 9.24, PSNR = 28.82.
Execution time $\approx$ 178 secs.

Figure 5.6: The denoised estimates obtained by the predictive FW-II with $(k_1, k_2) = (5, 6)$ and the quadtree-based predictive FW-II schemes with collage error decomposition criterion.

Next, the performance of the proposed FW denoising scheme in predicting the FW code of the test image is further examined and assessed.

5.2.5 Examining the Performance of the Predicted FW Code

In this section, the predicted FW code, as obtained from the noisy test image of "Lenna", is compared to the exact FW code obtained from the original noiseless image of "Lenna". The results are similar to those observed when examining the predicted fractal code in the pixel domain, as discussed in some details in section 4.2.7. These observations can be summarized as follows:

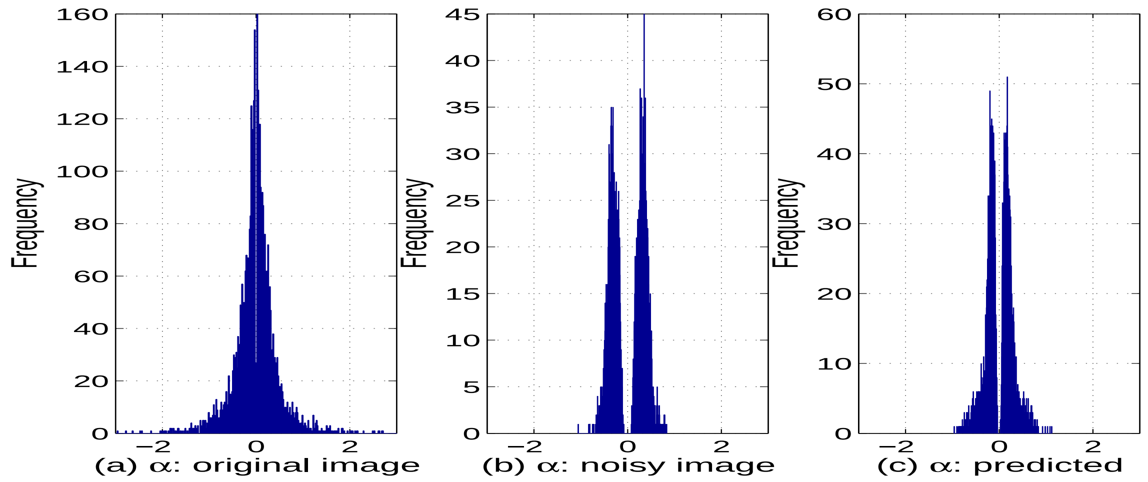
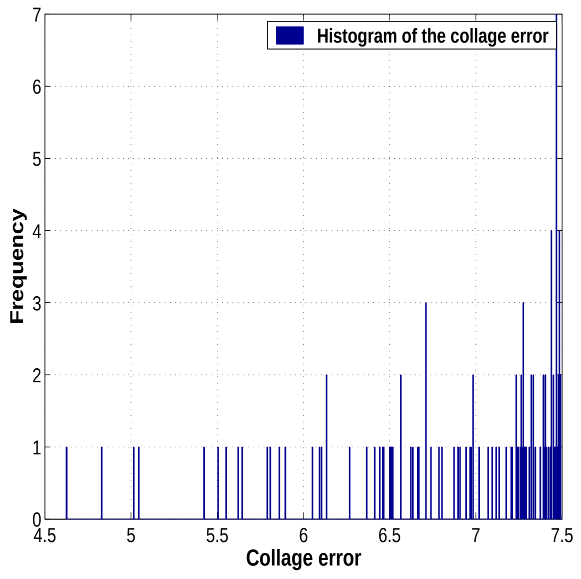
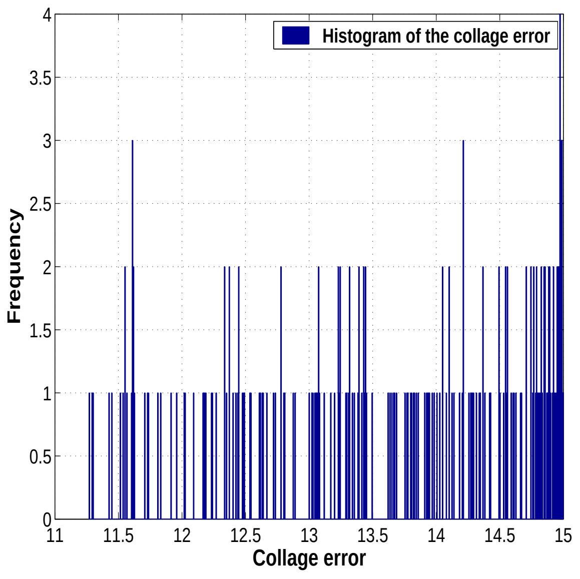


Figure 5.7: Comparison between the scaling coefficients corresponding to the original image, the noisy image as well as the predicted coefficients. Note that the distribution of the predicted scaling coefficients resembles that corresponding to the true scaling coefficients.

- Figure 5.7 illustrates the distribution of the scaling coefficients, α, corresponding to the original and noisy images as well as the predicted scaling coefficients, as obtained using the standard FW-II predictive scheme with $(k_1^*, k_2^*) = (5, 6)$. Note that, once again, the distribution of the scaling coefficients corresponding to the noisy image is rather distinct from that of the original image, as it has a bi-modal with all scaling coefficients being non-zero. The reasons behind this distinct distribution are similar to those explained earlier regarding the shape of the distribution of the gray-level scaling coefficients of the noisy image. The distribution of the predicted scaling coefficients remains bi-modal, although to a lesser degree than that of the noisy image. This is the case because the image many flat low-activity subregions which yields child subtrees dominated by noise. As explained in case 2 of the above algorithms, these subtrees are simply fractally coded while introducing a minor modification (reduction) of their corresponding scaling coefficients, as given in (5.39). Consequently, the distribution of the predicted scaling coefficients is now closer to that of the exact scaling coefficients corresponding to the original image.



Child subtree number 450



Child subtree number 1600.

Figure 5.8: Samples from the distribution of the collage error corresponding to two typical child sub-blocks. Note, for both cases, that there are many sub-optimal parent subtrees that yield collage errors that are relatively close to the minimum collage error corresponding to the optimal parent subtree.

- As in the pixel domain, one does not need to predict the optimal parent subtree corresponding to each child subtree. In fact, as illustrated in Figure 5.8, for each child subtree there are many good sub-optimal parent subtrees that can be chosen instead of the optimal parent subtree with relatively small reduction in the fidelity. Note how for these typical child subtrees, there are a few parent sub-optimal parent subtrees that yield collage errors that are relatively close to the minimum collage error obtained from the optimal parent subtree.

- Figure 5.9 illustrates the histogram of the rank of the predicted parent subtrees. For the chosen $(k_1^*, k_2^*) = (5, 6)$ FW resolution, all 4096 child subtrees were examined and for each child subtree, all of the 1024 potential parent subtrees were tested. For each child subtree, the predicted parent subtree is then ranked according to how close it is to best (optimal) parent subtree, as obtained from the DWT of the original image. A rank of 1 means a perfect match, while a rank of 2, implies that the predicted parent subtree is the second best, etc. Note that most of the predicted parent subtrees are among the first few closest parent subtrees.

- Figure 5.10 (a) illustrates the child-parent subtrees using the closest 10 parent subtrees as

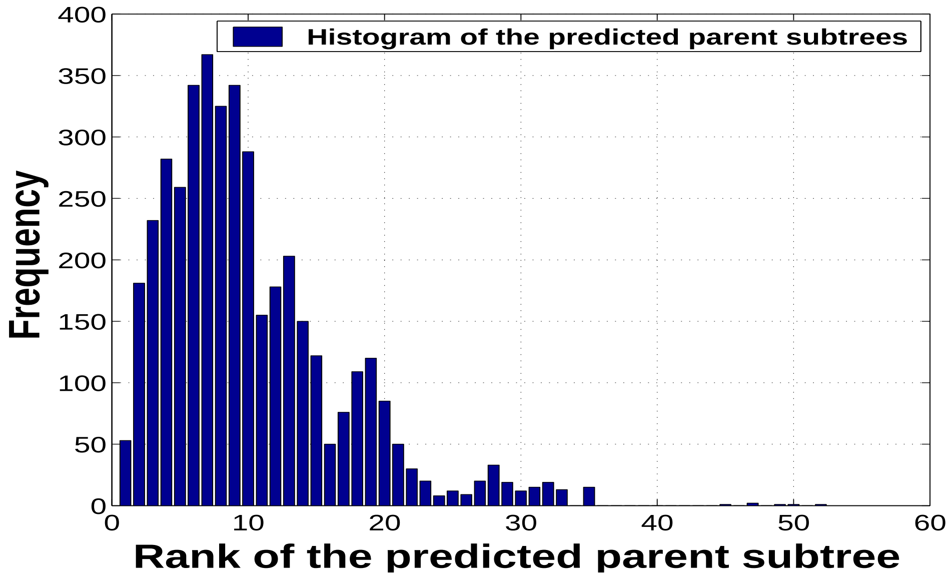
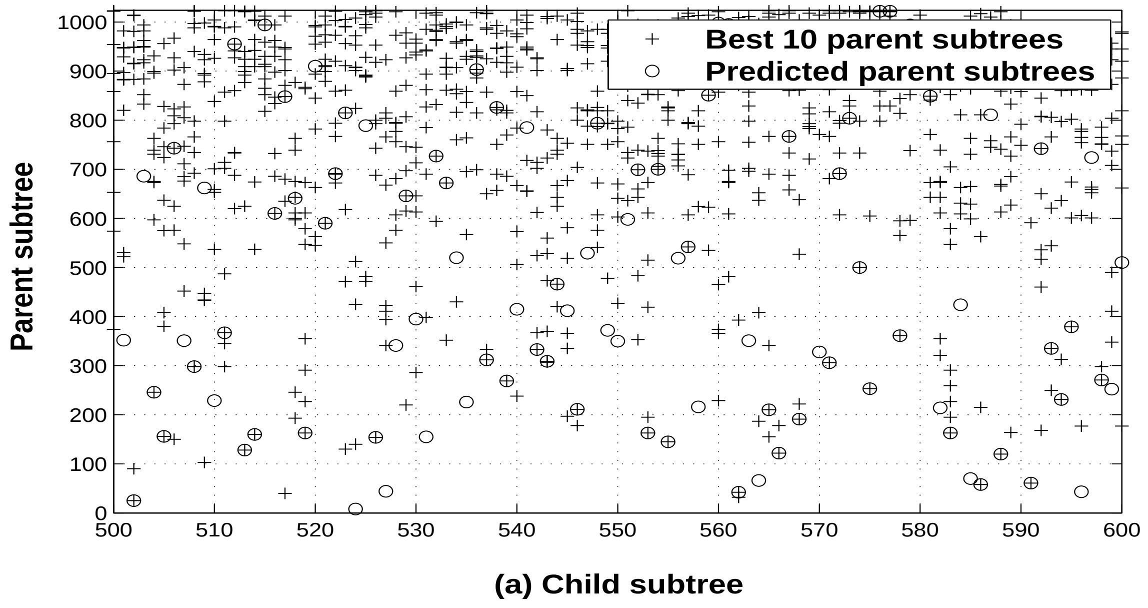


Figure 5.9: The histogram of the rank of the predicted parent subtrees corresponding to the noisy test image with $\sigma = 25$. A rank of 1 means that the optimal parent subtree was predicted, the rank is 2 when the second best parent subtree was predicted, etc. Note that most of the predicted parent subtrees are among the first few closest parent subtrees.

well as the predicted child-parent subtrees assignment map. Note that in many cases, the predicted child-parent assignment coincides with one of the closest 10 parent subtrees. Part (b) of this figure illustrates the collage errors corresponding to the closest 10 parent subtrees and the collage errors corresponding to the predicted parent subtrees. The fact that the predicted parent subtrees are usually among the first few closest (i.e., near-optimal) parent subtrees may indeed be the main reason why the predictive FW scheme performs reasonably well in predicting a fractal code that is close enough to the true fractal code of the original image, resulting in a relatively good denoised estimate of the original image.

This completes the examination of the predicted FW code.

In this section, a predictive fractal-wavelet image denoising method was proposed. It was shown that this scheme performs reasonably well resulting in a significantly restored image while preserving the sharp features of the image, such as edges and other high frequency features of the image. As shown in chapter 4, the spatially-based predictive fractal denoising scheme is competitive with some of the standard spatial-based image denoising schemes, such as the Lee filter. In this chapter, it was also shown that the wavelet-based predictive fractal denoising scheme is competitive with the most efficient wavelet thresholding methods such BayesShrink and SureShrink, especially



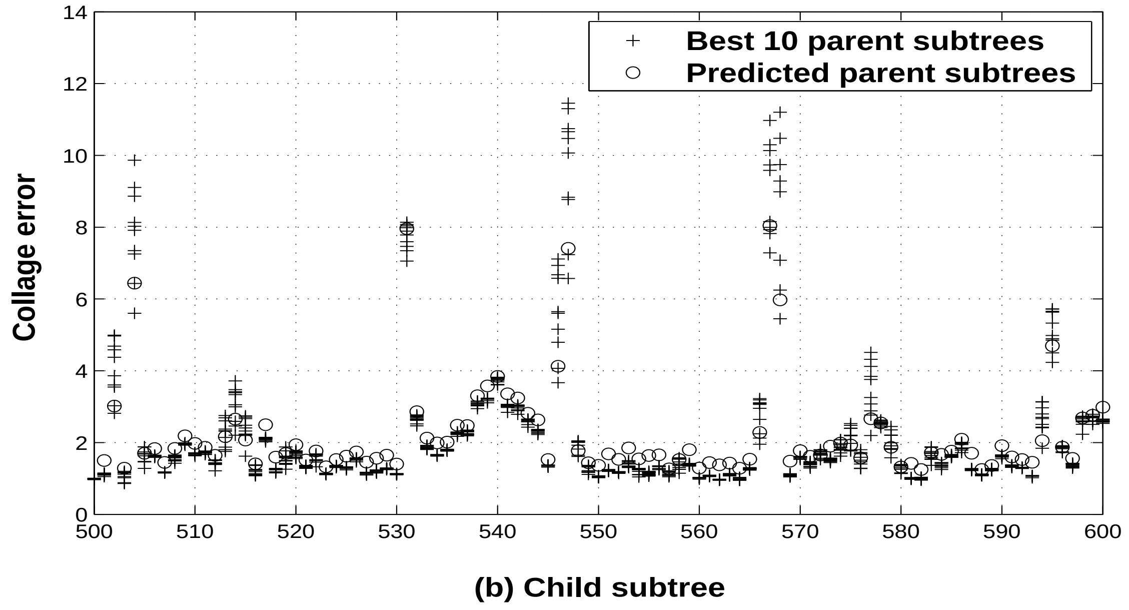


Figure 5.10: (a) Parent-child assignment maps using the best 10 parent subtrees corresponding to the original image as well as the predicted parent subtrees, for some of the child subtrees. (b) The collage errors for the best 10 parent subtrees as well as the collage errors corresponding to the predicted parent subtrees. Note that it was observed that for about 2689 child subtrees, the predicted parent subtrees coincide with one of the best 10 parent subtrees.

when using the quadtree partitioning of the wavelet decomposition tree.

One of the main advantages of the proposed fractal-wavelet denoising scheme is that it is computationally less expensive and significantly faster than the previously proposed standard fractal denoising scheme. The blockiness artifacts in conventional fractal-based representations are also eliminated. However, the FW denoised estimates suffer from other types of artifacts. Similar to most wavelet-based methods, FW representations often suffer from pseudo-Gibbs and ringing artifacts. However, these artifacts are generally less disturbing than the blockiness artifacts in standard fractal representations.

Next, the use of the cycle spinning algorithm for the purpose of enhancing the FW denoised estimates will be illustrated.

5.3 Improving FW Image Denoising using Cycle Spinning

As illustrated in Figures 5.11 and 5.12, the denoised estimates obtained by the various fractal-wavelet schemes exhibit some ringing and pseudo-Gibbs artifacts. Also, when encoding the noisy image using the Exhaustive FW-I scheme, zooming in on the denoised estimate reveals a significant degree of residual noise. In this section, the use of cycle spinning algorithm in order to reduce these artifacts that are inherent in FW representations and improve the overall quality of the FW denoised estimates.

For a range of shifts K, the cycle spinning algorithm for the purpose of fractal-wavelet denoising can be summarized as follows:

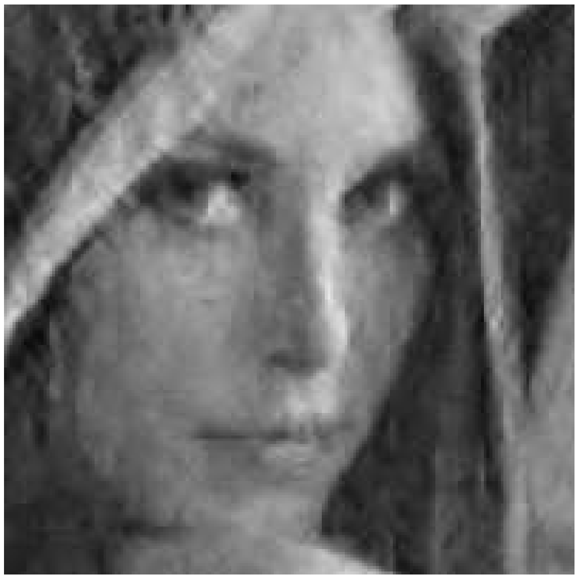
$$\hat{\mathbf{x}}_K = \frac{1}{K}\sum_{h=0}^{K} D_{-h}(\mathcal{IDWT}(FW(\mathcal{DWT}(D_h(\mathbf{y}))))). \tag{5.53}$$

Where D_h still represents the two-dimensional diagonal shifting operator defined in Eq. (3.47).

As before, the use of the cycle spinning algorithm, with K shifts, will increase the computational complexity and execution time by a factor of K.

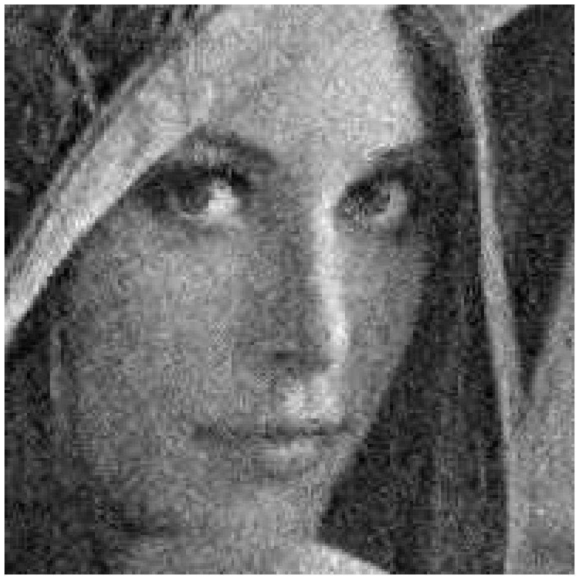
5.3.1 Experimental Results

As illustrated in Figure 5.13 - 5.15, incorporating the cycle spinning idea for the purpose of fractal-wavelet denoising does in fact result in significant improvement of the overall quality of the FW denoised estimates. In particular, significant reduction of the pseudo-Gibbs and ringing artifacts are observed. These improvements are better illustrated through the zoomed images. Note that



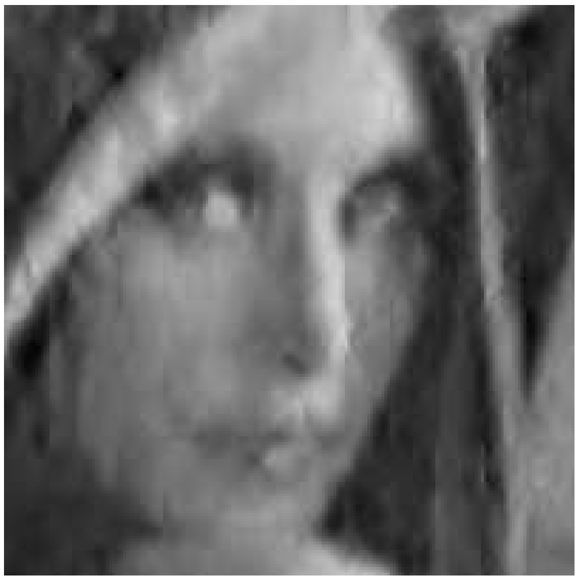
Exhaustive (FW-I) scheme

Level $(k_1^*, k_2^*) = (4, 5)$



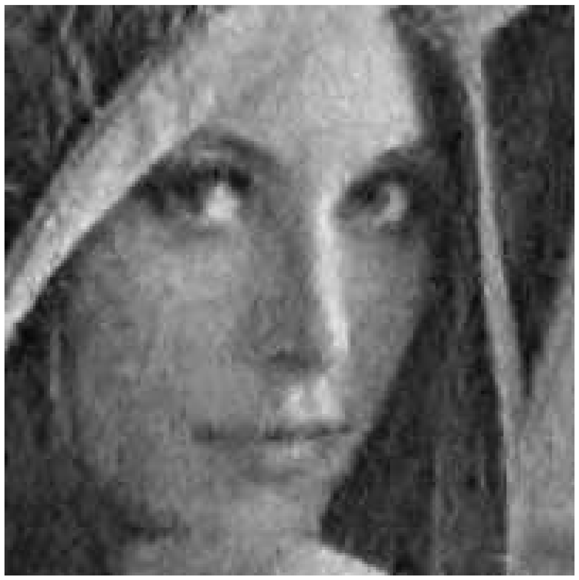
Exhaustive (FW-I) scheme

$(k_1^*, k_2^*) = (5, 6)$



Standard (FW-II) scheme

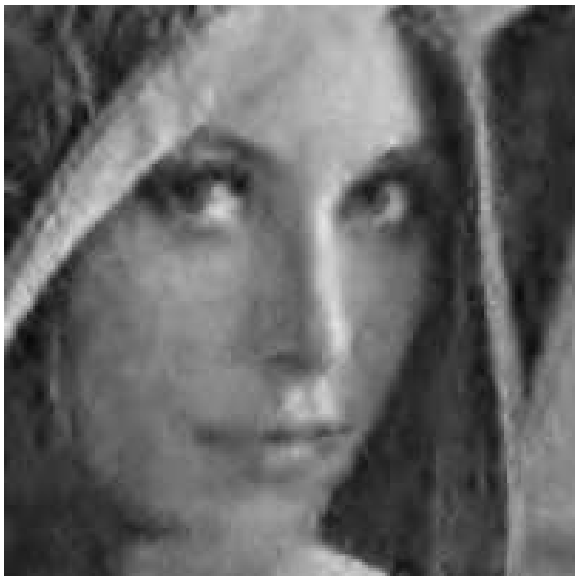
Level $(k_1^*, k_2^*) = (4, 5)$



Standard (FW-II) scheme

$(k_1^*, k_2^*) = (5, 6)$

Figure 5.11: Zooming in on the various fractal-wavelet denoised estimates, using the standard (FW-I) and exhaustive (FW-II) schemes reveals *pseudo-Gibbs and ringing artifacts* that are especially disturbing in the $(k_1^*, k_2^*) = (4, 5)$ resolution.


Predictive Standard (FW-II) scheme
Level $(k_1^*, k_2^*) = (5, 6)$

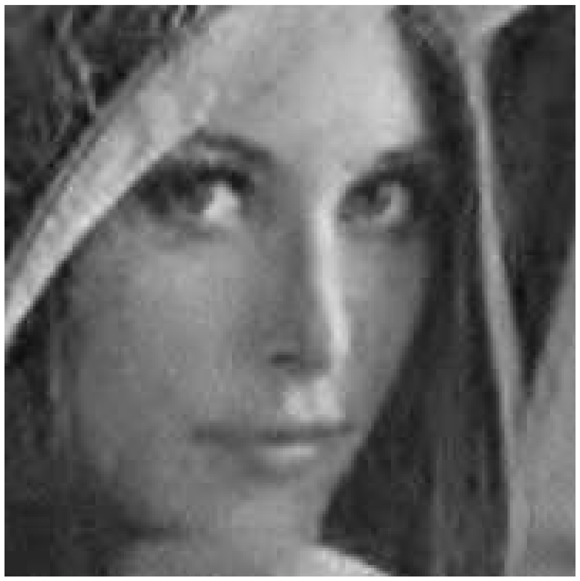
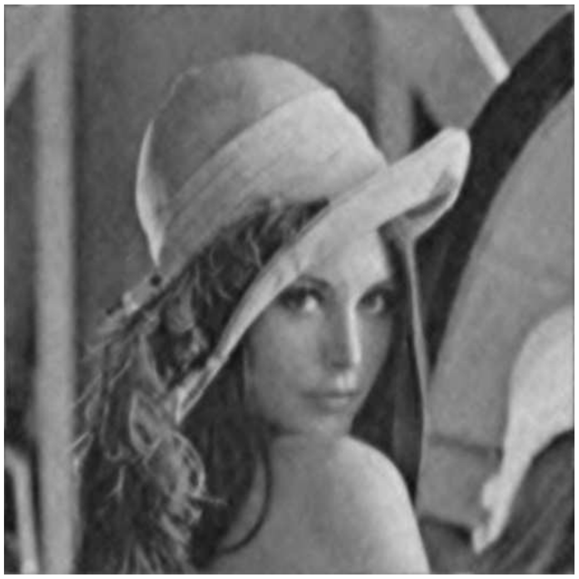
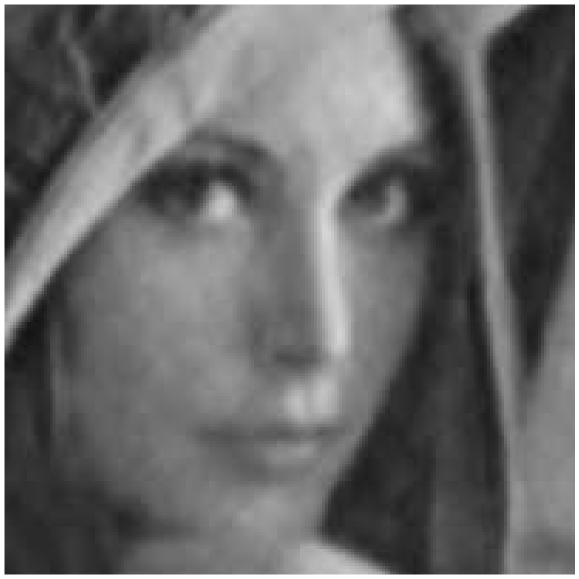

Predictive Quadtree-Based (FW-II) scheme

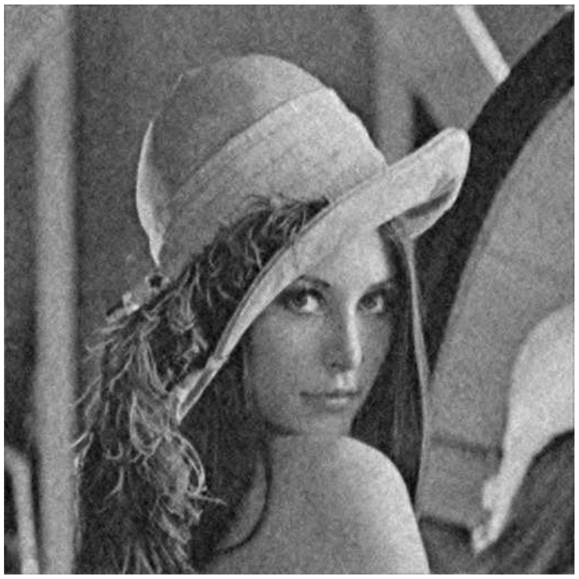
Figure 5.12: Zooming in on the FW denoised estimates, using the predictive FW schemes, reveals less degree of *pseudo-Gibbs and ringing artifacts.*

the enhancement is more significant for the predictive FW schemes than for the simple FW coding schemes. In particular, the predictive FW-II scheme benefits considerably from the use of the cycle spinning idea as most of the ringing and pseudo-Gibbs artifacts in the denoised estimates that have eliminated. Also, the overall quality of the FW-II denoised estimate has been considerably improved. The incorporation of the cycle spinning for the purpose of the quadtree-based FW predictive scheme has also resulted in significant improvement of the denoised estimate, as most of the artifacts have been reduced. This has indeed resulted in the best FW denoised estimate.

Figure 5.16 illustrates a comparison between the various FW denoising methods proposed in this chapter, when incorporating the cycle spinning idea. Note that the quadtree-based predictive scheme performs better than the other FW denoising methods. These curves illustrate the dependence of the quality of the denoised images on the number of shifts. Once again, note that the quality of the FW denoised estimates becomes stable after only a small number of shifts. Thus, only a partial shift of the image is usually needed.


(a) Exhaustive FW-I scheme: $(k_1^*, k_2^*) = (4, 5)$
RMSE=10.52, PSNR=27.69.


(b) Zooming in on the image in (a)


(c) Exhaustive FW-I scheme: $(k_1^*, k_2^*) = (5, 6)$
RMSE=10.21, PSNR=27.95.

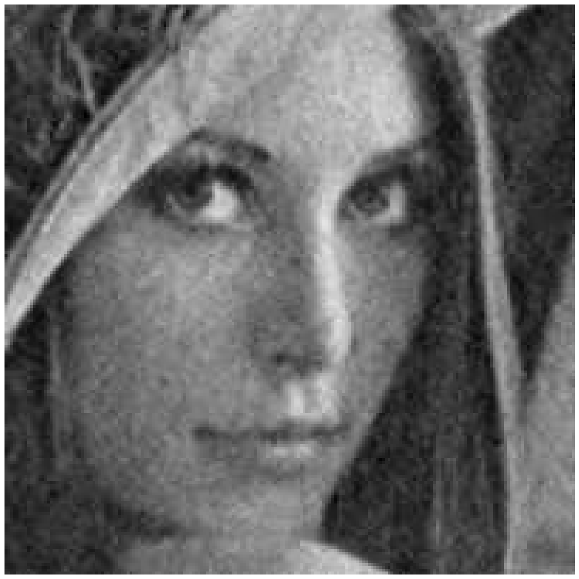
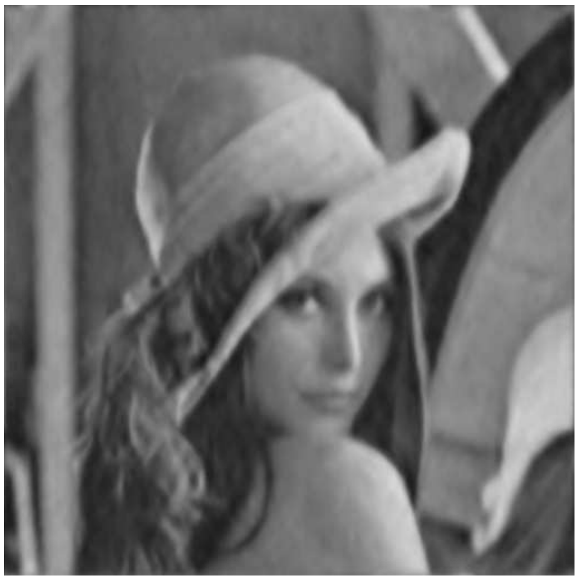
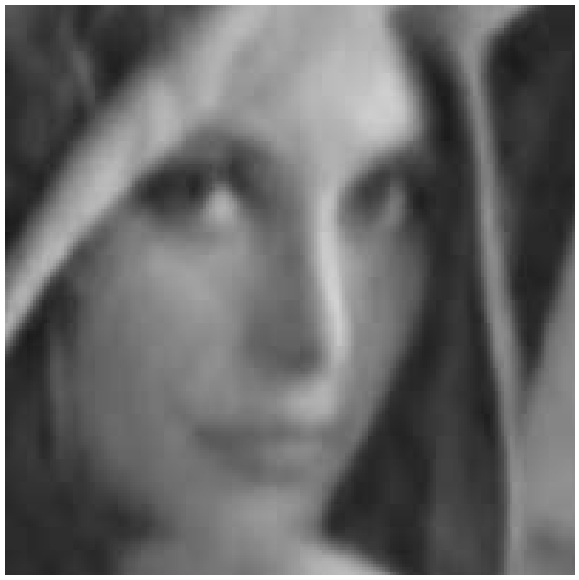
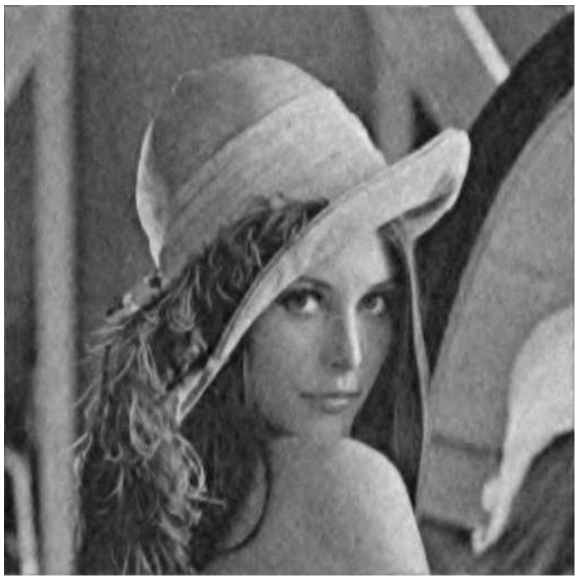

(d) Zooming in on the image in (c)

Figure 5.13: Exhaustive fractal-wavelet (FW-I) coding of the noisy image with $(k_1^*, k_2^*) = (4, 5)$ and $(k_1^*, k_2^*) = (5, 6)$ when incorporating the idea of cycle spinning with $K = 16$ diagonal shifts.


(a) Standard FW-II scheme: $(k_1^*, k_2^*) = (4, 5)$ RMSE=12.51, PSNR=26.18.


(b) Zooming in on the image in (a).


(c) Standard FW-II scheme: $(k_1^*, k_2^*) = (5, 6)$ RMSE=9.07, PSNR=28.97.

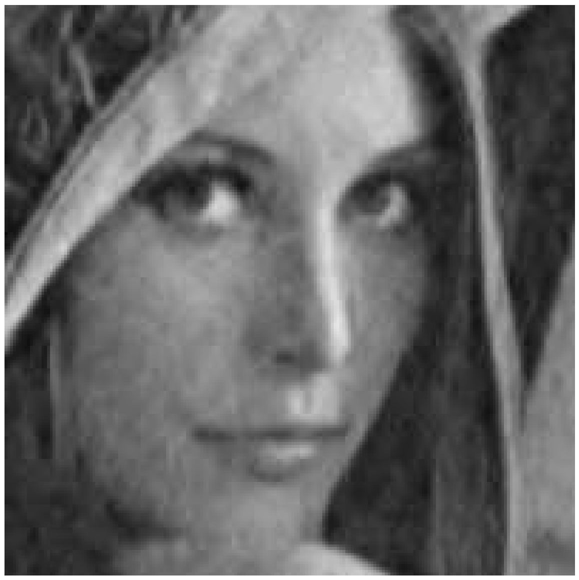
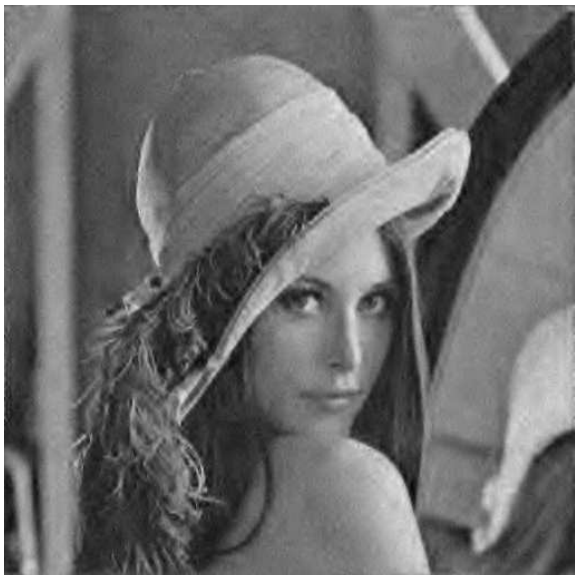
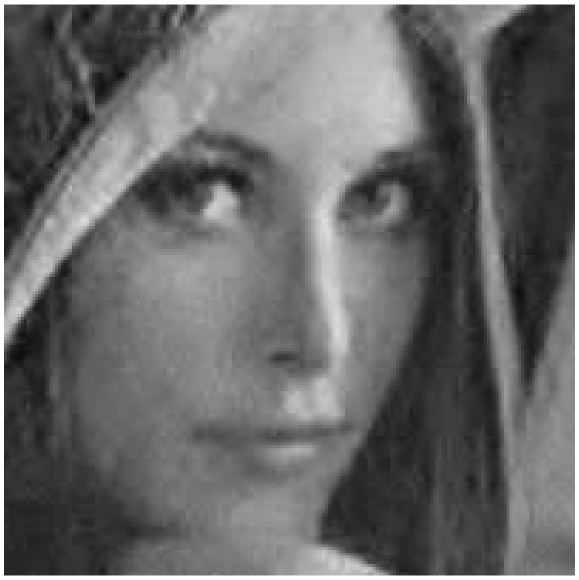

(d) Zooming in on the image in (c).

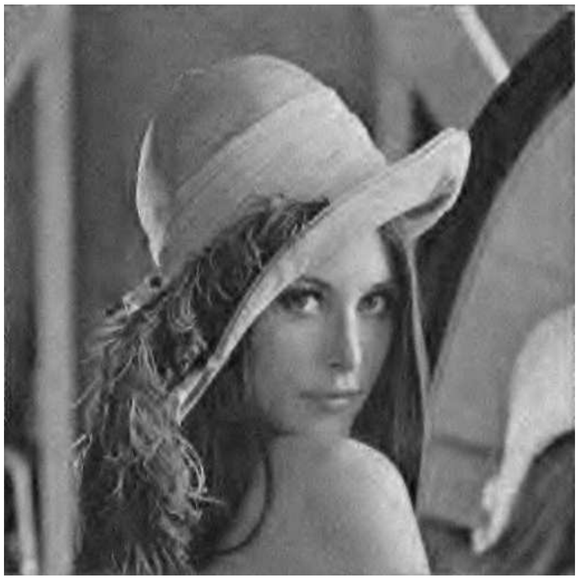
Figure 5.14: Standard fractal-wavelet (FW-II) coding of the noisy image with $(k_1^*, k_2^*) = (4, 5)$ and $(k_1^*, k_2^*) = (5, 6)$ when incorporating the idea of cycle spinning with $K = 16$ diagonal shifts.


(a) Predictive FW-II scheme

Standard with $(k_1^*, k_2^*) = (5, 6)$

RMSE=8.53, PSNR=29.51.


(b) Zooming in on the image in (a).


(c) Predictive FW-II scheme

Quadtree partitioning

RMSE=8.37, PSNR=29.68.

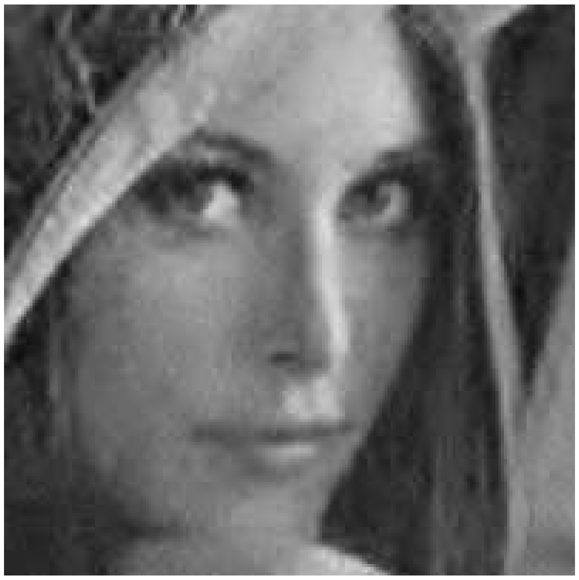
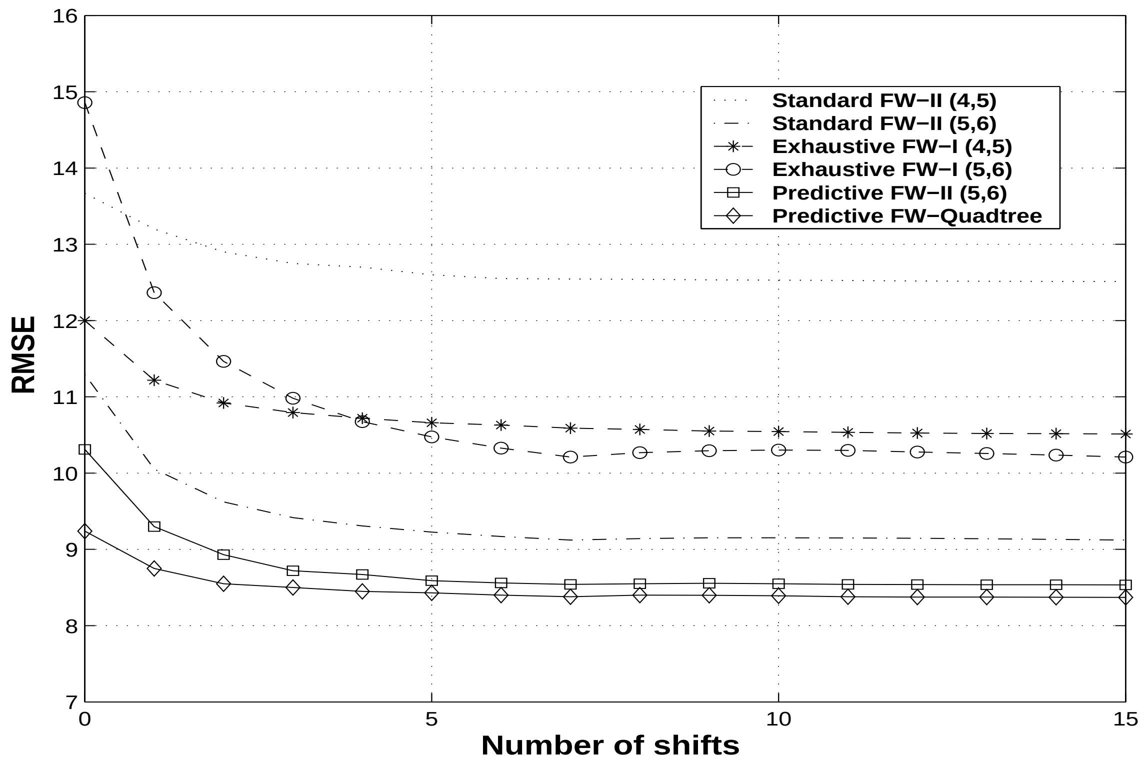

(d) Zooming in on the image in (c).

Figure 5.15: Denoised estimates obtained by the *Standard* and *Quadtree-based* predictive fractal-wavelet (FW-II) denoising schemes when incorporating the idea of cycle spinning with $K = 16$ diagonal shifts.



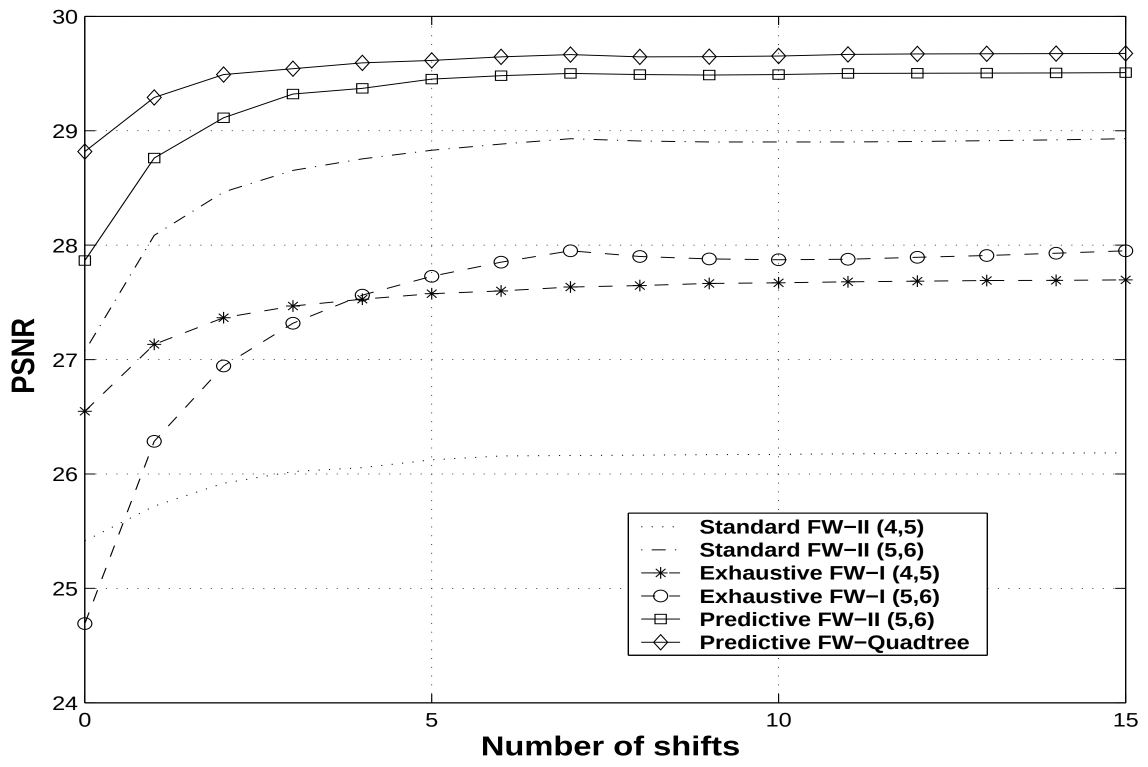


Figure 5.16: Quantitative comparison of the gain achieved by incorporating the cycle spinning idea (K=16 diagonal shifts) within the the various fractal-wavelet based denoising schemes studied in this chapter.

5.4 Summary and Concluding Remarks

In this chapter, the potential of applying various fractal-wavelet schemes for the purpose of image denoising was investigated. It was shown that, as it was the case for the spatial-based fractal schemes, when the wavelet transform of the noisy image is simply fractally encoded, using any of the fractal-wavelet schemes described in chapter 2, a significant amount of the noise is suppressed. The use of the adaptive as well as the quadtree-based fractal-wavelet schemes for the purpose of image denoising was also explored. A relationship between these fractal-wavelet denoising schemes and some of the basic wavelet thresholding methods found in the literature, was established.

A simple yet effective method to estimate the fractal-wavelet code of the original noise-free image from the statistics of the noisy image was proposed. This fractal-wavelet scheme is analogous to the fractal denoising scheme proposed in chapter 2. In particular, even better results can be obtained when using a hybrid fractal-wavelet image denoising scheme that makes use of quadtree partitioning scheme, with collage decomposition criterion, in the wavelet domain as well as adaptive thresholding of the stored wavelet coefficients. This hybrid FW denoising scheme was shown to yield denoised estimates that are significantly denoised while preserving that sharpness of the edges and other high frequency features of the image. The denoising and restoration achieved by the proposed hybrid FW denoising scheme was found to be consistent with the human visual system where extra smoothing is performed in flat and low activity regions and a lower degree of smoothing is performed near high frequency components, e.g. edges, of the image.

The performance of the proposed fractal-wavelet image denoising scheme is compared to the results obtained using the various wavelet thresholding techniques. It was also shown that the proposed scheme yields better results than some of the conventional wavelet thresholding methods, such as SureShrink and BayesShrink described in chapter 3. When comparing the performance of the hybrid FW denoising schemes to the results obtained by using the spatial domain based fractal denoising schemes proposed and described in chapter 4, note that predictive spatial-based fractal schemes yields relatively better results than their fractal-wavelet counterparts. The main advantage of the wavelet-based fractal denoising scheme over the spatial domain fractal denoising scheme is that the former is computationally significantly much less expensive than the latter.

Chapter 6

Additional Experimental Results

So far in this thesis, several image denoising methods were studied and implemented. Some of these methods are spatial-based such as the Lee filter, others are frequency-based, such as the Wiener filter, and the rest are wavelet based such as VisuShrink, LevelShrink, SureShrink and BayesShrink. More importantly, several new wavelet and fractal-based image denoising methods were also developed. In particular, a context-based thresholding strategy that applies hard and soft thresholding operators which take into consideration the values of the neighboring wavelet coefficients before thresholding a wavelet coefficient was proposed. Several fractal-based image denoising schemes were also proposed. In particular, a simple, yet effective methodology the aims for estimating the fractal code of the original noise-free image from the noiseless one was proposed. It was shown that this method can be applied in the spatial domain of the noisy image by using standard fractal schemes or in the wavelet domain of the noisy image by applying fractal-wavelet methods. Throughout this work, one common test image of "Lenna", corrupted by AWGN with noise standard deviation $\sigma_{\mathbf{w}} = 25$, has been used so far in order to assess and compare the performance of the various image denoising methods studied in this thesis.

In this chapter, some of the proposed fractal and wavelet image denoising methods will be applied in order to restore different test images corrupted by AWGN noise with different intensity, $\sigma_{\mathbf{w}}$. The use of these different test images will help us achieve a better assessment of the performance of the proposed image denoising schemes and make a comparison between them.

6.1 Application to Images

The test images are “Lenna”, “Boat”, “Peppers” and “San Francisco”. The three new original (i.e. noiseless) images to be used in this chapter are illustrated in Figure 6.1.

(a) “Boat” (b) “Peppers” (c) “San Francisco”

Figure 6.1: The new original, noise-free, test images to be used in this chapter.

First, an experimental investigation of the advantages of the proposed context-based, localized wavelet thresholding operators is illustrated.

6.1.1 BayesShrink using Context-Based Thresholding

Recall that in chapter 3, the use of context-based, localized hard and soft thresholding operators that take into consideration the values of the neighboring wavelet coefficients when thresholding a wavelet coefficient was proposed. It was shown that the use of this adaptive thresholding strategy with the various wavelet thresholding methods does indeed result in significant gain as compared to the use of the standard hard and soft thresholding operators. These findings were based on the use of a single test image of “Lenna” degraded by AWGN noise with noise intensity, $\sigma_{\mathbf{w}} = 25$. In this section, this proposed thresholding strategy will be applied for the purpose of denoising the various test images which were corrupted by AWGN noise with varying intensity.

Figure 6.2 illustrates the results of denoising four different test images, “Lenna”, “Boat”, “Peppers” and “San Francisco”, which corrupted by AWGN noise with varying intensity; $\sigma_{\mathbf{w}} = 10, 20, 30$ and 40, using the BayesShrink method. Two versions of the BayesShrink scheme were implemented: The original BayesShrink technique which adopts the standard soft thresholding operator and a modified BayesShrink scheme which applies the proposed context-based soft thresholding operator. The results obtained before and after applying the cycle spinning algorithm are presented.

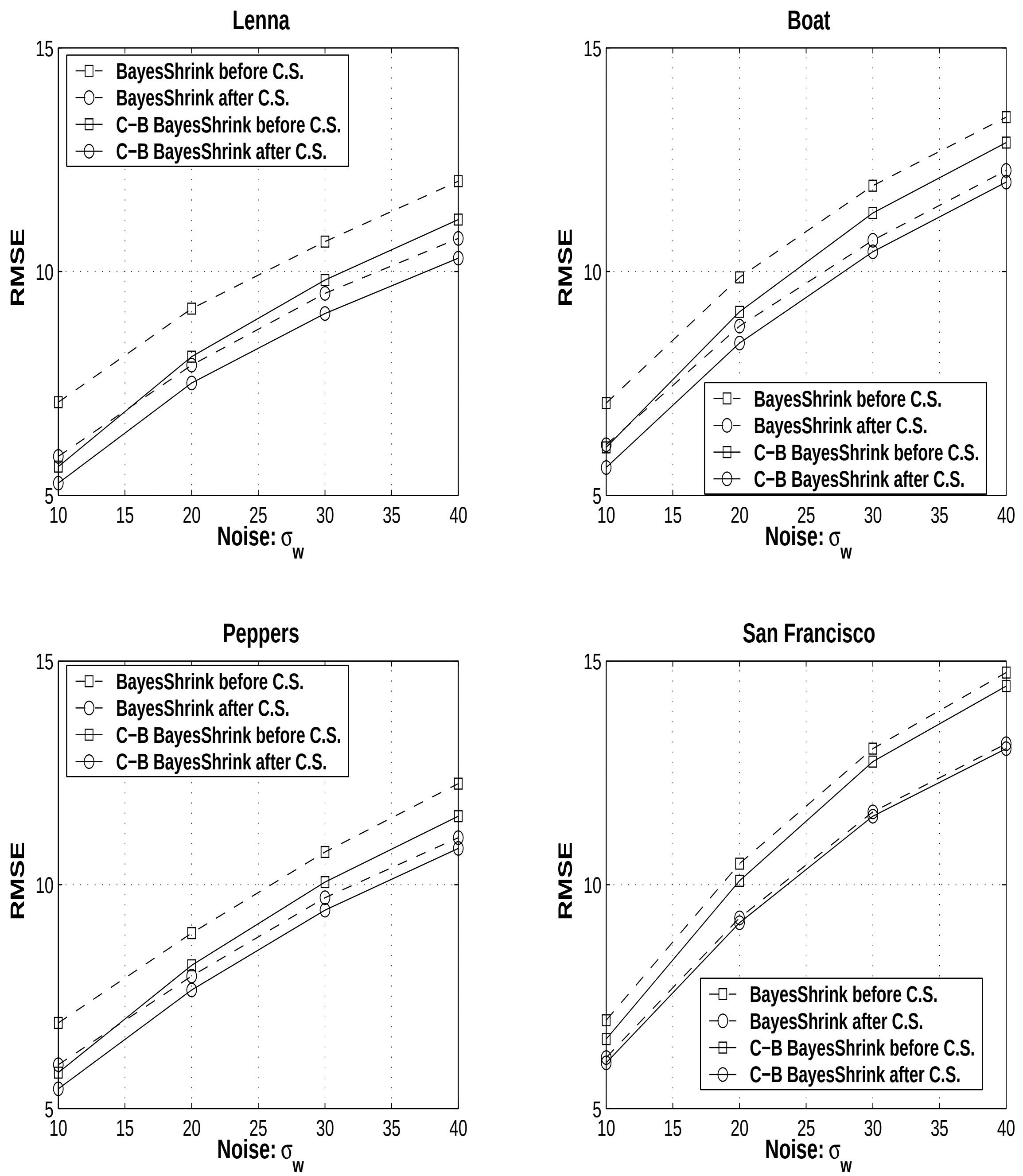


Figure 6.2: Comparison between the results obtained by the BayesShrink method using the conventional and context-based soft thresholding operators, before and after applying the cycle spinning (C.S.) idea with $K = 8$ diagonal shifts, for the various test images and noise intensities. Note that the context-based soft thresholding strategy yields significantly better results than the application of the standard soft thresholding operator.

In view of these experimental results, some observations are outlined as follows:

- The results obtained by the modified BayesShrink scheme which adopts the proposed context-based soft thresholding operator are consistently better than those obtained by the original BayesShrink scheme which uses the standard soft thresholding operator. Indeed, this is the case for all test images and noise intensities.

- This improvement is even more evident before using the cycle spinning idea than after applying this enhancement method. This is probably because the use of the cycle spinning idea has benefited both methods by eliminating or considerably reducing most of the artifacts, hence yielding closer enhanced denoised estimates from both schemes.

- It is important to note that the use of the cycle spinning idea is computationally expensive and hence time consuming. In practice, some applications may not allow for this time complexity and the cycle spinning idea may not be a feasible option. Thus, the fact the proposed thresholding operator yields significantly better results than the standard thresholding operator without applying the cycle spinning idea is of great practical importance and significance.

In this section, it was shown that the advantages of the proposed context-based wavelet thresholding operator observed for the case of the noisy image of “Lenna” holds for other noisy test images and noise intensities as well. In fact incorporating the context-based soft thresholding operator in the BayesShrink technique yields consistently better results than the use of the conventional soft thresholding operator.

Next, the performance of the proposed fractal-based image denoising methods will be assessed.

6.1.2 Predictive Fractal-Based Image Denoising Schemes

In this section, the proposed predictive fractal-based image denoising schemes will be applied for the purpose of image restoration.

Figure 6.3 illustrates the results of applying the predictive standard fractal as well as the fractal-wavelet image denoising schemes, developed in chapters 4 and 5, respectively. Once again, the results obtained before and after applying the cycle spinning algorithm are presented.

In view of these results, we make the following observations:

- The spatial-based fractal predictive image denoising scheme performs consistently better than the fractal-wavelet method, before and after the use of the cycle spinning idea. The reasons

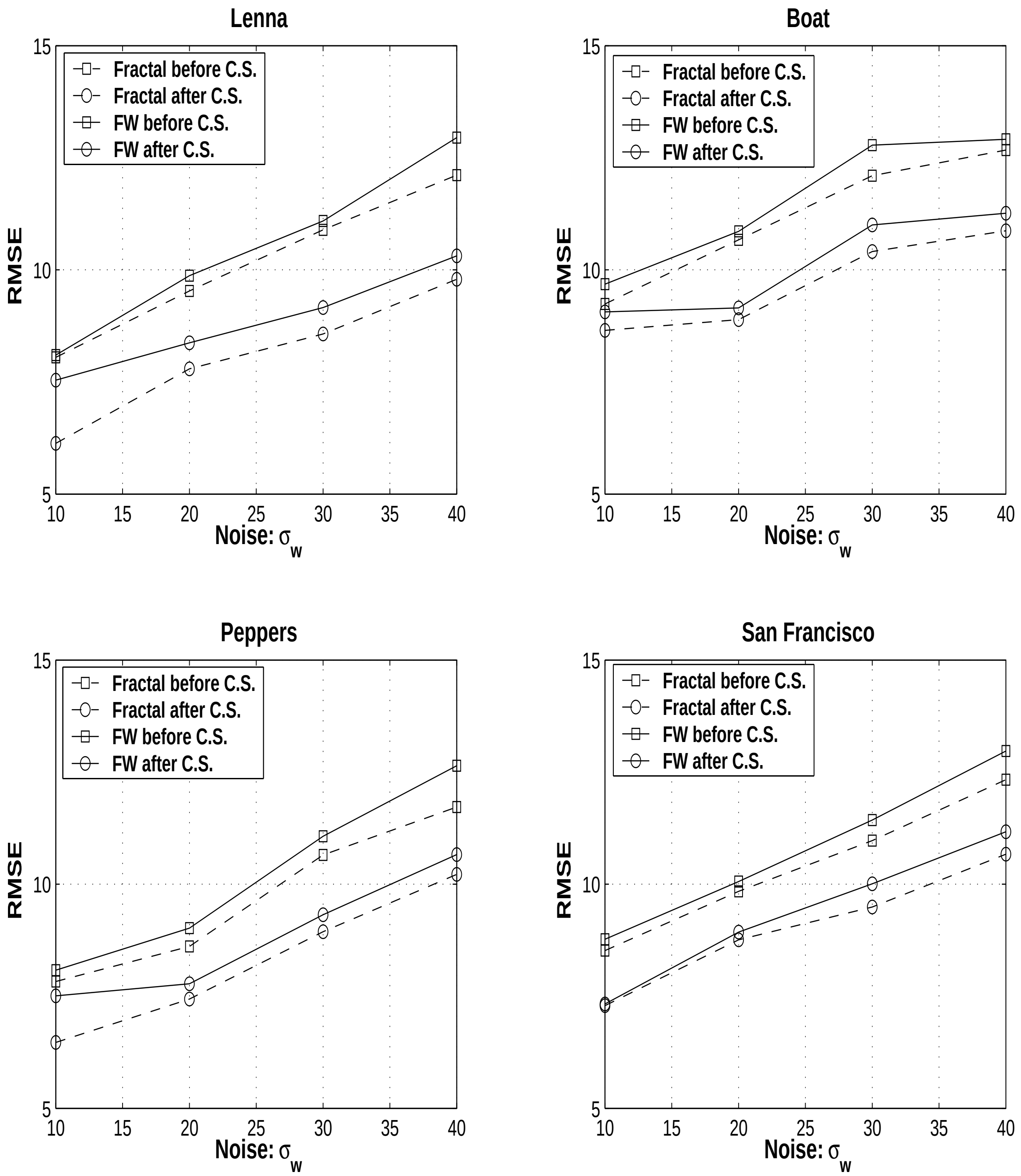


Figure 6.3: Comparison between the results obtained by the *predictive fractal* and *fractal-wavelet* image denoising schemes, before and after applying the cycle spinning (C.S.) idea with $K = 8$ diagonal shifts, for the various test images and noise intensities. Note that standard fractal scheme performs consistently better than the fractal-wavelet method.

for the superior performance of the standard fractal denoising scheme as compared to its fractal-wavelet counterpart may include:

1. The standard fractal predictive scheme employs a set of contractive geometric maps that decimate, through an averaging operation, and geometrically transform, through one of eight isometries, in order to match a child sub-block to its optimal parent sub-block. In fact, as discussed previously, the decimation associated with the contractive spatial maps used in the fractal transform is probably responsible for most of the denoising. However, the fractal-wavelet scheme uses no such smoothing or pre-processing step before fitting a parent subtree to a child subtree. This may indeed be the main reason behind the difference in the performance of the fractal and the fractal-wavelet denoising schemes.

2. Also, the fractal-wavelet scheme stores a set of noisy wavelet coefficients. These noisy wavelet coefficients, along with the FW code, are then used by the FW decoder to estimate the remaining wavelet coefficients, hence resulting in re-distributing the noise among the predicted coefficients. Although the stored wavelet coefficients are indeed located in the lower decomposition levels and scales, which are smoothed by the wavelet filters, the role of these coefficients in redistributing the noise may be significant. When using the quadtree-based fractal-wavelet scheme, a context-based thresholding strategy was applied to all higher scales stored wavelet coefficients. However, when using the standard FW scheme, it was observed that thresholding the stored wavelet coefficients is counterproductive and yields worse results, in terms of RMSE.

- The use of the cycle spinning idea has indeed benefited considerably both fractal schemes, resulting in a almost artifact-free fractal and fractal-wavelet denoised estimates, for the various images and noise levels. This supports the use of the cycle spinning idea for the purpose of enhancing fractal representations. Thus, the cycle spinning idea, which was originally introduced for the purpose of reducing the Gibbs artifacts in wavelet thresholding denoised estimates, has also been shown to be an effective tool of reducing ringing, blurriness and Gibbs artifacts in fractal-wavelet estimates and blockiness artifacts in standard fractal reconstructions. This suggests that the cycle spinning idea can indeed be used for enhancing fractal image representations in applications where the computational complexity and the bit rate are not a major practical concern.

6.1.3 Comparison Between the Proposed Image Denoising Methods

In this section, a comparison between the performance of the various image denoising methods proposed in this thesis will be illustrated. These image denoising methods are the context-based wavelet thresholding, the standard fractal and the fractal-wavelet predictive schemes.

The soft, context-based, thresholding operator was incorporated in the BayesShrink method. The standard fractal and fractal-wavelet (FW-II with $(k_1^*, k_2^*) = (5, 6)$) predictive schemes were also implemented for the purpose of restoring the four noisy test images. The cycle spinning algorithm was used to reduce the artifacts and further enhance the quality of the various denoised estimates. Figure 6.4 - 6.7, illustrate the comparisons between various denoised estimates obtained using the proposed image denoising methods.

In view of these results, some observations are presented as follows:

- For lower noise intensity, $\sigma_{\mathbf{w}}$, the BayesShrink method performs considerably better than the fractal-based methods. However, this is to be expected since the fractal and the fractal-wavelet methods are lossy compression methods, whereas the BayesShrink method is not. In fact, even when encoding the original, noise-free image, using the fractal-based methods, the fractal representation is a lossy one. However, if one attempts to denoise the original noise-free image, i.e. $\sigma_{\mathbf{w}} = 0$, using the BayesShrink method, the resulting estimation should be lossless because the BayesShrink scheme adopts a zero threshold (i.e. $\lambda = 0$), in the absence of any noise. This, it is not surprising that the BayesShrink method performs better than the fractal-based image denoising method for low noise intensities, especially at $\sigma = 10$.

- For larger values of the noise variance, the performance of the BayesShrink method starts to degrade rapidly as the noise variance increases. Under heavy noise, i.e. for $\sigma = 30, 40$, the fractal-based schemes perform better than the BayesShrink method, especially for $\sigma = 40$. In spite of the degradation of the quality of the various denoised estimates, as reflected by the RMSE and PSNR quality measures, few artifacts are observed in the denoised images obtained by the fractal-based or BayesShrink methods. This is mainly due of the use of the cycle spinning idea that does a good job of reducing these artifacts. Perhaps some of the remaining artifacts would be more noticeable when zooming in on these denoised estimates.

- As explained in the previous section, the standard fractal image denoising scheme performs better than the fractal-wavelet counterpart. This scheme also performs better than the BayesShrink method for high noise variance. A significant feature of the standard fractal denoising scheme is that it adopts a quantization strategy of the gray-level coefficients that yields a fractally denoised estimate with pixel values that lie in the range $[0, 255]$. As the noise variance increases, it becomes more difficult for the BayesShrink or the fractal-wavelet methods to yield denoised estimates that satisfy this requirement.

In summary, for most of the test images the BayesShrink method performs better for low noise variance but sharply degrades as the noise variance becomes large. For heavy noise ($\sigma \geq 20$), the fractal-based methods perform consistently better than the BayesShrink method. Also, the standard fractal denoising scheme has been shown to perform consistently better than its fractal-wavelet counterpart.

6.2 Summary and Concluding Remarks

In this section, the proposed wavelet and fractal-based image denoising methods were applied in order to restore and enhance four noisy images degraded by different noise intensities. It was shown that the context-based thresholding strategy performs consistently better than the conventional thresholding operators for the various noisy test images. Also, for most of the test images, the BayesShrink method performs better for lower noise variance but sharply degrades as the noise variance becomes larger. For heavy noise, the fractal-based methods perform consistently better than the BayesShrink method. Also, the standard fractal denoising scheme has been shown to perform consistently better than its fractal-wavelet counterpart.

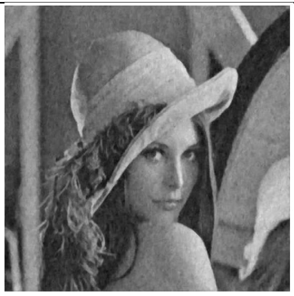
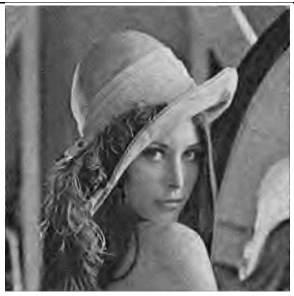
$\sigma_{\mathbf{w}}$	**Context-Based BayesShrink**	**Predictive Fractal Scheme**	**Predictive FW-II Scheme**
10	RMSE=5.27, PSNR=33.70	RMSE=6.13, PSNR=32.38.	RMSE=7.54, PSNR=30.59.
20	RMSE=7.51, PSNR=30.61	RMSE=7.82, PSNR=30.27.	RMSE=8.13, PSNR=29.93.
30	RMSE=9.06, PSNR=28.99	RMSE=8.57, PSNR=29.47.	RMSE=9.16, PSNR=28.89.
40	RMSE=10.30, PSNR=27.87	 RMSE=9.79, PSNR=28.32.	 RMSE=10.31, PSNR=27.86.

Figure 6.4: Denoised estimates using the proposed *BayesShrink context-based thresholding, predictive fractal* and *fractal-wavelet* image denoising schemes, when applying the cycle spinning idea with $K = 8$ shifts, for the noisy test image of "Lenna" with different noise intensity $\sigma_{\mathbf{w}}$.

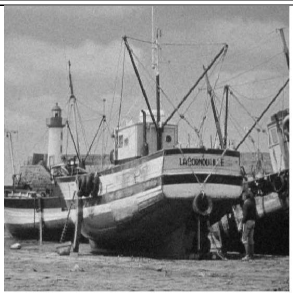
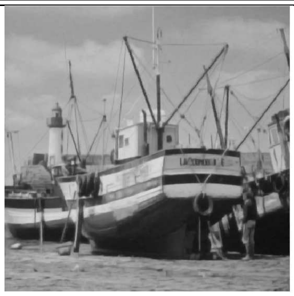
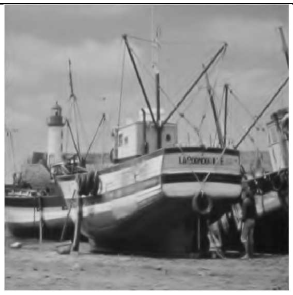
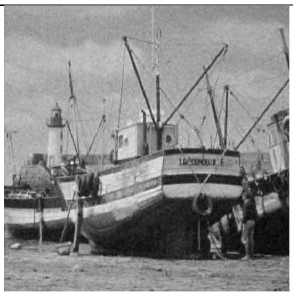
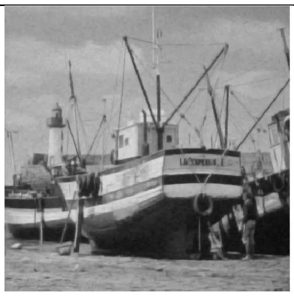
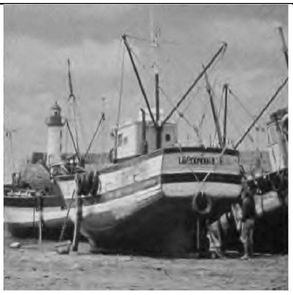
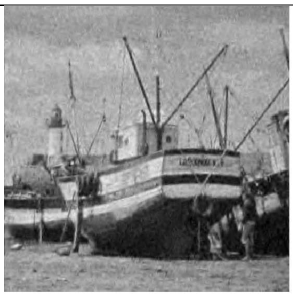
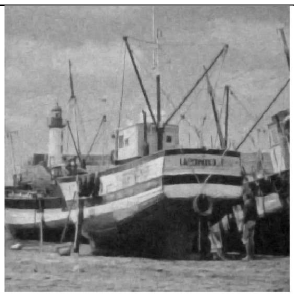
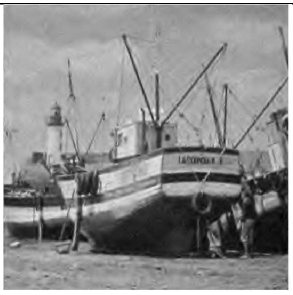
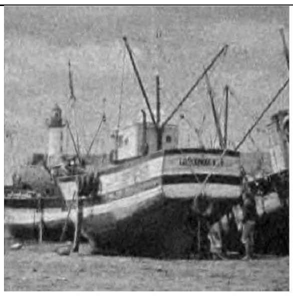
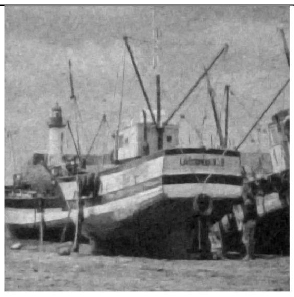
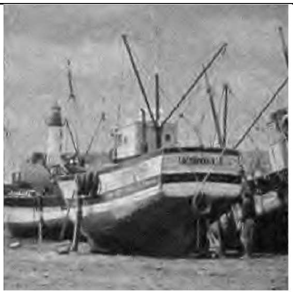
$\sigma_{\mathbf{w}}$	**Context-Based BayesShrink**	**Predictive Fractal Scheme**	**Predictive FW-II Scheme**
10	 RMSE=5.62, PSNR=33.13.	 RMSE=8.65, PSNR=29.39.	 RMSE=9.06, PSNR=28.99.
20	 RMSE=8.40, PSNR=29.65.	 RMSE=8.89, PSNR=29.15.	 RMSE=9.15, PSNR=28.90.
30	 RMSE=10.44, PSNR=27.76.	 RMSE=10.41, PSNR=27.78.	 RMSE=11.00, PSNR=27.31.
40	 RMSE=12.00, PSNR=26.54	 RMSE=11.87, PSNR=26.64.	 RMSE=12.26, PSNR=26.35.

Figure 6.5: Denoised estimates using the proposed *BayesShrink context-based thresholding, predictive fractal* and *fractal-wavelet* image denoising schemes, when applying the cycle spinning idea with $K = 8$ shifts, for the noisy test image of "Boat" with various noise intensity, $\sigma_{\mathbf{w}}$.

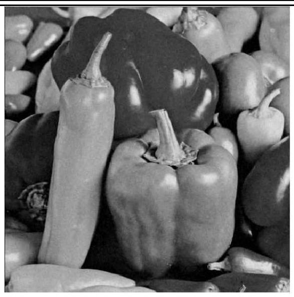
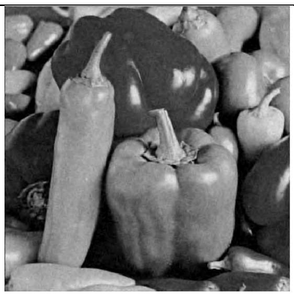
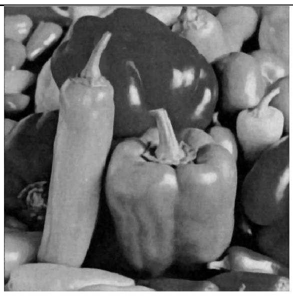
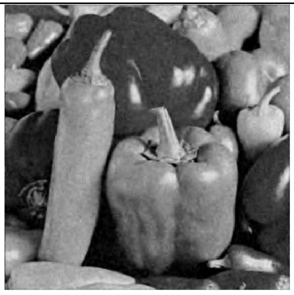
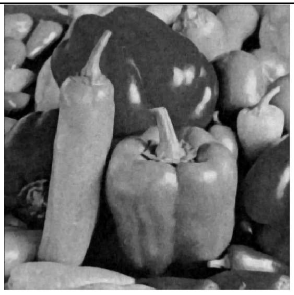
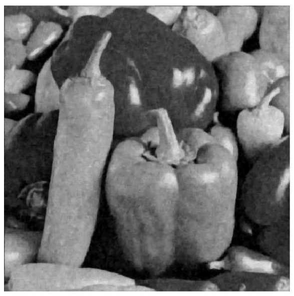
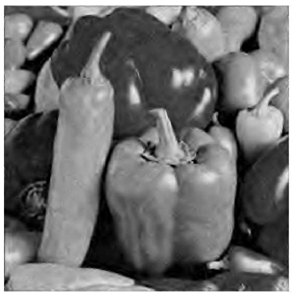
$\sigma_{\mathbf{w}}$	**Context-Based BayesShrink**	**Predictive Fractal Scheme**	**Predictive FW-II Scheme**
10	 RMSE=5.44, PSNR=33.42.	RMSE=6.47, PSNR=31.09.	RMSE=7.51, PSNR=30.61.
20	 RMSE=7.65, PSNR=30.46.	 RMSE=7.44, PSNR=30.70.	RMSE=7.78, PSNR=30.31.
30	 RMSE=9.43, PSNR=28.64.	 RMSE=8.94, PSNR=29.10.	RMSE=9.32, PSNR=28.74.
40	RMSE=10.81, PSNR=27.45.	 RMSE=10.22, PSNR=27.94.	 RMSE=10.66, PSNR=27.57.

Figure 6.6: Denoised estimates using the proposed *BayesShrink context-based thresholding, predictive fractal* and *fractal-wavelet* image denoising schemes, when applying the cycle spinning idea with $K = 8$ shifts, for the noisy test image of "Peppers" with different noise intensity, $\sigma_{\mathbf{w}}$.

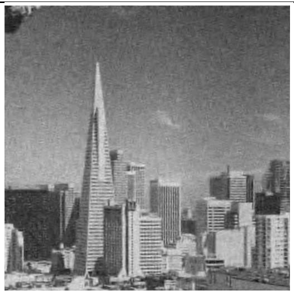
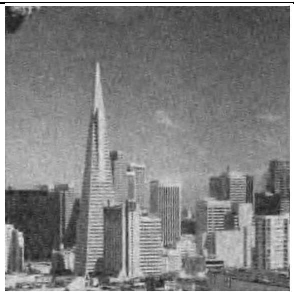
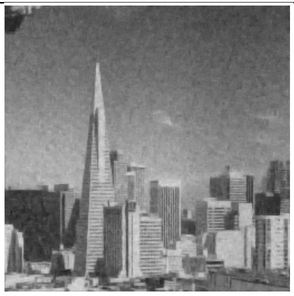
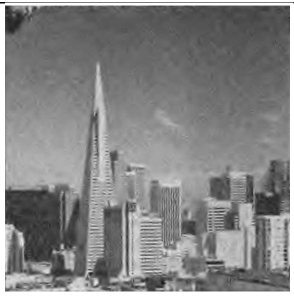
$\sigma_{\mathbf{w}}$	**Context-Based BayesShrink**	**Predictive Fractal Scheme**	**Predictive FW-II Scheme**
10	RMSE=6.02, PSNR=32.54.	RMSE=7.29, PSNR=30.87.	RMSE=7.33, PSNR=30.83.
20	RMSE=9.15, PSNR=28.90.	RMSE=8.76, PSNR=29.28.	RMSE=8.93, PSNR=29.11.
30	 RMSE=11.53, PSNR=26.89.	RMSE=9.49, PSNR=28.59.	RMSE=10.01, PSNR=28.12.
40	 RMSE=13.04, RMSE=25.83.	 RMSE=10.67, PSNR=27.57.	 RMSE=11.17, PSNR=27.17.

Figure 6.7: Denoised estimates using the proposed *BayesShrink context-based thresholding, predictive fractal* and *fractal-wavelet* image denoising schemes, when applying the cycle spinning idea with $K = 8$ shifts, for the noisy test image of "San Francisco" with different noise intensity, $\sigma_{\mathbf{w}}$.

Chapter 7

Summary, Conclusions and Future Work

In this thesis, several novel fractal and wavelet-based image denoising methods were proposed, implemented and assessed. An outline of the major results and contributions are summarized in this chapter. A few related future research directions and problems that stem from this work will also be proposed. First, a brief summary of the thesis is presented.

7.1 Thesis Summary

- In chapter 1, the image denoising problem, encountered when an image is corrupted by an AWGN noise with unknown variance $\sigma_{\mathbf{w}}^2$, was outlined and formulated. The problem of estimating the noise variance from the noisy image was then addressed. Various spatial as well as frequency-based standard image denoising methods were described and implemented. The principle of wavelet thresholding for image denoising was then described. Finally, the research in this thesis was motivated.

- Chapter 2 contains a detailed description and implementation of various fractal and wavelet based image coding methods. In particular, the development of adaptive fractal and fractal-wavelet image compression schemes was outlined. These schemes were then implemented for the purpose of image denoising in later chapters of this work.

- Various basic wavelet thresholding methods for the purpose of image denoising were reviewed, implemented and assessed in chapter 3. The use of the cycle spinning strategy for the purpose of reducing the pseudo-Gibbs artifacts that tend to be present in wavelet-based representations of signals was investigated. The application of context-based thresholding strategies that introduced reasonable modifications to the conventional hard and soft thresholding operators used in the literature was proposed and implemented. These modified thresholding operators take into consideration the content of a specified immediate neighborhood of each wavelet coefficient before thresholding. It was shown that significant improvement in the quality of the denoised estimate is gained, especially in the case of soft thresholding, for the various wavelet thresholding methods.

- In chapters 4 and 5, new fractal-based image denoising techniques in the spatial and the wavelet domains of the noisy image were proposed, implemented and assessed. It was shown that significant noise reduction was gained by simply fractally encoding the noisy image using any of the fractal or fractal-wavelet image coding schemes. Some of the reasons behind achieving image denoising through fractal coding of the noisy image were investigated. Furthermore, a simple, yet effective method of estimating the fractal code of the original noiseless image from the noisy one was outlined. From this predicted fractal code, one can then reconstruct a fractally denoised estimate of the original noiseless image. This was done analogously for the pure fractal schemes, as applied in the spatial domain of the noisy image, and the fractal-wavelet schemes, as applied in the wavelet domain of the noisy image. The use of the cycle spinning algorithm for the purpose of enhancing the fractally denoised estimates was illustrated. It was observed that incorporating this algorithm results in significant reduction of the blockiness and pseudo-Gibbs artifacts that tend to be present in the fractal and the fractal-wavelet denoised estimates, respectively.

- Additional experimental results using different test images and noise intensities were illustrated in chapter 6, in order to achieve a better comparison between the studied and proposed image denoising methods.

7.2 Conclusions

Some of the conclusions that can be drawn from this work include:

- This work broadens the application scope of fractal-based methods. The potential of applying fractal-based methods for the purpose of image denoising has been investigated in detail and fractal-based image denoising schemes were proposed and implemented. Experimental results show that these fractal-based image denoising methods are competitive, or sometimes even compare favorably with standard image denoising methods reviewed in this thesis.

- Much interest has been given, in the literature, to the use of fractal-based coding schemes for the purpose of image compression. However little attention has been given to their possible use in image restoration and enhancement. This work may, in fact, represent the first significant attempt of its kind. It should be stated, however, that other fractal and "multifractal" methods [25] have been shown to be successful in image processing applications, including denoising as well as segmentation, texture analysis, approximation and compression. Over the past decade, much of this work has been done by J. Lévy Véhel and coworkers [40, 50]. The denoising methods developed by this group rely on multifractal methods and more sophisticated methods of analysis (e.g. "2-microlocalization") that are deeply rooted in wavelet theory.

- Fractal denoised estimates may be significantly enhanced using the cycle spinning algorithm that was originally introduced for the purpose of reducing the Gibbs artifacts in wavelet denoised estimates. This enhancement involves reduction of blockiness artifacts in standard fractal representations and Gibbs artifacts in fractal-wavelet estimates.

- Although the proposed fractal-based denoising methods are competitive in performance with some of the standard image denoising methods, they suffer from the enormous computational complexity that is associated with standard fractal techniques. Although many faster fractal-based methods have been proposed, these schemes are only sub-optimal and the reduction of the computational complexity is generally achieved at the expense of significant degradation of the quality of the fractal representation. The computational complexity associated with the exhaustive fractal schemes remains a significant obstacle that has prevented fractal-based signal processing methods from becoming practical and realistic alternatives to the existing signal processing techniques currently in use.

- The use of the localized context-dependent hard and soft thresholding operators have resulted in some improvement in the performance of the various standard wavelet thresholding methods studied in this thesis. This indicates that conventional hard and soft thresholding operators widely used in the wavelet thresholding literature are not optimal even when they use adaptive thresholds. There are two aspects to the adaptivity of the thresholding operators: The first is related to the selected threshold λ where adaptive thresholds perform better than the universal one. The second adaptivity aspect is related to the manner the thresholding operators are applied. In this work, it was shown that better results were achieved by applying adaptive and localized thresholding operators instead of the conventional hard and soft thresholding point operators. While the selection of adaptive thresholds have been investigated in the literature, making the thresholding operators themselves more adaptive seem to have been overlooked.

7.3 Contributions

In this thesis, several important contributions were realized. The main new ideas proposed and implemented in this work can be summarized as follows:

- This work may represent the first serious attempt to investigate the potential of fractal-based image coding methods for the purpose of image enhancement and denoising. Although significant progress and development have been achieved in fractal image coding over the past decade, there has been little published work that focused on investigating the application of fractal-based techniques for other aspects of image processing, other than image compression.
- A simple, yet effective method of estimating the fractal code of the original noiseless image from the noisy image was proposed and implemented. From this predicted fractal code of the original noiseless images, one can then reconstruct a fractally denoised estimate of the original noiseless image. This predictive fractal denoising algorithm was applied in the spatial as well as the wavelet domains of the noisy image.
- Experiments that aimed for developing more adaptive wavelet thresholding strategies were presented. In particular, the application of context-based thresholding strategies that introduced reasonable modifications to the conventional hard and soft thresholding operators was proposed and implemented. These thresholding operators take into consideration the content of a specified immediate neighborhood of each wavelet coefficient before thresholding it. It was shown that significant improvement in the quality of the denoised estimate is gained, especially in the case of soft thresholding, for the various wavelet thresholding methods.
- The cycle spinning idea is not new, however incorporating this approach within the proposed

fractal and fractal-wavelet image denoising schemes for the purpose of improving their performance is a novel idea. This has resulted in significant improvement in the subjective quality of the fractally denoised estimates.

7.4 Future Research Directions

Some of the research directions that may stem from the work presented in this thesis can be outlined as follows:

- It was shown that the use of the quadtree-based fractal and fractal-wavelet predictive schemes for image denoising yields results that are significantly better than using standard fractal and fractal-wavelet schemes. However, whenever using the quadtree partitioning algorithm for the purpose of fractal image coding, one has to choose a threshold for the decomposition criterion. The determination of a reasonable, image independent strategy for selecting such a threshold is still an open question. This threshold can be viewed as a denoising fine-tuning parameter that measures the trade-off between suppressing the noise and reconstructing the high frequency content and important features of the image.

- In practice, one is often constrained with a bit-budget. Thus, developing image denoising methods that only aim for getting the best quality of the denoised image without also paying any attention to the compression ratios and bit-rate limitation may not be very practical. Thus, there a great need to develop effective schemes that perform not only image denoising but also image compression. The Rissanen's Minimum Description Length (MDL) principle has recently been effectively used for the purpose of designing wavelet thresholding methods for the purpose of simultaneous image compression and denoising [13, 14, 15, 16]. The use of the MDL principle may also be applied for the purpose of developing effective fractal-based techniques that are capable of performing simultaneous denoising and compression of noisy images. Fractal-based methods have been shown to be effective lossy image compression methods. In this thesis, it was shown that fractal-based schemes are also effective image denoising methods. Thus, the development of fractal-based joint image compression and denoising would combine these capabilities of the fractal methods. These schemes would allow us to generate rate distortion curves that exhibit the trade-off between the quality of a fractally denoised image and the bit rate required to store this denoised image. Simultaneous image compression and denoising schemes are important in many applications where simultaneous compression and denoising is needed, for instance, when images are acquired from a noisy source and storage or transmission capacity is severely limited, such as in some video coding applications.

- For the context-based thresholding strategy, proposed in chapter 3, the issue of selecting the context and its size requires further investigation. Also, defining the context itself requires further investigation. Instead of choosing the neighboring wavelet coefficients, perhaps one could choose a context containing the parent or children of the wavelet coefficient or other contexts. Also, when an insignificant coefficient is surrounded by a significant one, it is kept unchanged. Clearly, one may decide to alter the value of such a coefficient without setting it to zero. Also, instead of taking the maximum absolute value into consideration one may consider other statistics, such as the average or median. These are important issues that are open for further investigation and will be the focus of future research.

Appendix A

Sample Programs

Most of the experimental results presented in this thesis can be implemented using the following sample programs or variations of these programs.

```
%********************************
%********************************
% Median Flitering of AWGN noise *
%********************************
%********************************
clear all
original=imread('lenna_orig.tif');
original_db=double(original);
noisy=original_db+25*randn(512,512);
noisy_db=double(noisy);
mask=5;
input=noisy_db;
mean_filter0=filter2(ones([mask mask]), input)/(mask*mask);
subplot(1,1,1), image(double(mean_filter0));
colormap(gray(255))
axis off
axis equal
E2=double(mean_filter0)-original_db;
RMSE2=std2(E2);
PSNR2=20*log10(255/RMSE2);
%***********************************
%***********************************
% LEE Flitering of AWGN noise       *
%***********************************
%***********************************
clear all
original=imread('lenna_orig.tif');
original_db=double(original);
noisy=original_db+25*randn(512,512);
noisy_db=double(noisy);
mask=7;
f=noisy_db;
LocalMean=filter2(ones([mask mask]), f)/(mask*mask);
LocalVar=filter2(ones([mask mask]),f.^2)/(mask*mask)-LocalMean.^2;
NoiseVar=25^2;
SignalVar=LocalVar-NoiseVar;
lee_dn=(SignalVar./LocalVar).*f+(LocalMean.*NoiseVar)./LocalVar;
E2=double(lee_dn)-original_db;
RMSE2=std2(E2)
PSNR2=20*log10(255/RMSE2)
colormap(gray(255))
subplot(1,1,1), image(lee_dn);
axis off
axis equal
```

```
orient tall
%*************************************
%*************************************
%Gaussian LPF Flitering of AWGN noise *
%*************************************
%*************************************
clear all
original=imread('lenna_orig.tif');
original_db=double(original);
noisy=original_db+25*randn(512,512);
noisy_db=double(noisy);
mask=7;
f=noisy_db;
h=fspecial('gaussian',[mask,mask],1);
gauss=filter2(h,f);
subplot(1,1,1), image(double(gauss));
colormap(gray(255))
axis off
axis equal
orient tall
E2=double(gauss)-original_db;
RMSE2=std2(E2)
PSNR2=20*log10(255/RMSE2)
%*************************************
%*************************************
% Wiener  LPF Flitering of AWGN noise *
%*************************************
%*************************************
clear all
original=imread('lenna_orig.tif');
original_db=double(original);
noisy=original_db+25*randn(512,512);
noisy_db=double(noisy);
mask=7;
f=noisy_db;
wiener=wiener2(f,[mask,mask]);
subplot(1,1,1), image(wiener);
colormap(gray(255))
axis off
axis equal
E2=original_db-double(wiener);
RMSE2=std2(E2)
PSNR2=20*log10(255/RMSE2)
/*********************************************
/*********************************************/
/* Standard fractal image coding             */
/*    uniform image partitioning: (M,N)      */
/*    input: input_image.dat                 */
/*    output: output_image.out               */
/*********************************************/
/*********************************************/
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#define SIZE    512
#define NMAPS   8
#define MAXITER 20
void getBlock(int index, int n, int n1, int n2, float block[][n]);
void TransBlock(int map, int n, float block[][n], float Tblock[][n]);
void Leastsquares(int n,float x[1024],float y[1024],float coef[2],float *error);
float lena0[SIZE][SIZE], lena[SIZE][SIZE], lena1[SIZE][SIZE];
main()
{
int i, j, iic, jjc, iip, jjp, npixc, npixp, iter;
int nchilds, nparents, ic, jc, ncount, map;
int nrindex[64][64],ncindex[64][64],mapindex[64][64];
float error, errmin, pixel, sum1,sum2,sum, RMSE1, PSNR1;
float childb[8][8], parentb[16][16], Tparentb[16][16];
float alpha,beta,x[1024],y[1024],coef[2],constant[64][64],slope[64][64];
FILE * finput0;
FILE * finput1;
FILE * foutput;
finput0=fopen("input_image.dat","r");
foutput=fopen("output_image.out","w");
nparents=32; // M=32
nchilds=64; // N=64
npixp=SIZE/nparents;
npixc=SIZE/nchilds;
for (i=0;i<SIZE;i++)
{   for (j=0;j<SIZE;j++)
    {  fscanf(finput0, "%f",&pixel);
       lena[i][j]=pixel/255;
       lena1[i][j]=0;
    }
}
// start the encoding process...
for (iic=0;iic<nchilds;iic++) // for each child blocks
{   for (jjc=0;jjc<nchilds;jjc++)
    {   errmin=1000000;
        getBlock(1,npixc,iic, jjc,childb);
        ncount=0;
        for (i=0;i<npixc;i++)
        {   for (j=0;j<npixc;j++)
            {   y[ncount]=childb[i][j];
                ncount++;
            }
        }
        for (iip=0; iip<nparents; iip++) // test all parent blocks
        {   for (jjp=0; jjp<nparents; jjp++)
            {   getBlock(1,npixp,iip,jjp,parentb);
                for (map=0; map<NMAPS; map++) // test all goemetric maps
                {   TransBlock(map, npixp,parentb,Tparentb);
                    ncount=0;
                    for (i=1; i<npixp; i=i+2)
                    {   for (j=1; j<npixp; j=j+2)
                        {    x[ncount]=(Tparentb[i-1][j-1]+Tparentb[i-1][j]+
                                Tparentb[i][j-1]+Tparentb[i][j])/4.0;
                             ncount++;
                        }
                    }
                    Leastsquares(npixc*npixc, x, y, coef, &error);
                    if (error < errmin)
                    {  errmin=error;
                       constant[iic][jjc]=coef[0];
                       slope[iic][jjc]=coef[1];
                       nrindex[iic][jjc]=iip;
                       ncindex[iic][jjc]=jjp;
                       mapindex[iic][jjc]=map;
                    }
                 }
            }
        }
    }
}// end of encoding process
for (iter=0; iter<MAXITER; iter++) // start decoding
{   for (iic=0; iic<nchilds; iic++)
    {   for (jjc=0; jjc<nchilds; jjc++)
        {   iip=nrindex[iic][jjc];
            jjp=ncindex[iic][jjc];
            map=mapindex[iic][jjc];
            getBlock(2,npixp,iip,jjp,parentb);
            TransBlock(map,npixp,parentb,Tparentb); /* transform the block */
            for (i=1; i<npixp; i=i+2)
            {   for (j=1; j<npixp; j=j+2)
                {   ic=(i-1)/2;
                    jc=(j-1)/2;
                    childb[ic][jc] = slope[iic][jjc]*(Tparentb[i][j]+
                    Tparentb[i][j-1]+ Tparentb[i-1][j]+Tparentb[i-1][j-1])/4.0
                    +constant[iic][jjc];
                    lena1[iic*npixc+ic][jjc*npixc+jc]=childb[ic][jc];
                }
            }
        }
    }
}
sum=0;
for (i=0; i<SIZE; i++)
{   for (j=0; j<SIZE; j++)
    {   sum=sum+pow((lena1[i][j]-lena[i][j]),2);
        fprintf(foutput,"%.3f\n", 255*lena1[i][j]);
    }
}
RMSE1=255*sqrt(sum/pow(SIZE,2));
PSNR1=20*log10(255.0/RMSE1);
printf("RMSE = %f\n", RMSE1);
printf("PSNR = %f\n", PSNR1);
fclose(finput0);
fclose(foutput);
}
/*****************************************/
/* This function sets up the child block */
/*****************************************/
void getBlock(int index, int n, int n1, int n2, float block[][n])
{     int i, j;
      if (index==1)
      {  for (i=0; i<n;i++)
              for (j=0; j<n;j++)
                  block[i][j]=lena[n1*n+i][n2*n+j];
              return;
      }
      if (index==2)
      {  for (i=0; i<n;i++)
              for (j=0; j<n;j++)
                  block[i][j]=lena1[n1*n+i][n2*n+j];
         return;
      }
}
/***************************************************************/
/* This function transforms a parent block using one of the 8 */
/* geometric transformations                                   */
/***************************************************************/
void TransBlock(int map, int n, float block[][n], float Tblock[][n])
{    int i, j;
     switch(map)
     {   case(0): for (i=0; i<n;i++)            /* identity map */
                      for (j=0;j<n;j++)
                          Tblock[i][j]=block[i][j];
                  break;
         case(1): for (i=0; i<n;i++) /* 90 degree rotaion */
                      for (j=0;j<n;j++)
                          Tblock[i][j]=block[j][n-1-i];
                  break;
         case(2): for (i=0; i<n;i++)  /* 180 degree rotaion */
                      for (j=0;j<n;j++)
                          Tblock[j][i]=block[n-1-j][n-1-i];
                  break;
         case(3): for (i=0; i<n;i++)   /* 270 degree rotation */
                      for (j=0;j<n;j++)
                          Tblock[j][i]=block[n-1-i][j];
                  break;
         case(4): for (i=0; i<n;i++) /* transpose wrt diagonal */
                      for (j=0;j<n;j++)
                          Tblock[i][j]=block[j][i];
```

```
                 break;
          case(5): for (i=0; i<n;i++)    /* transpose wrt off-diagonal*/
                       for (j=0;j<n;j++)
                           Tblock[i][j]=block[n-1-j][n-1-i];
                   break;
          case(6): for (i=0; i<n;i++)    /* rot. wrt. hori. mid. axis */
                       for (j=0;j<n;j++)
                           Tblock[i][j]=block[n-1-i][j];
                   break;
          case(7): for (i=0; i<n;i++)  /* rot. wrt. vert. mid. axis */
                       for (j=0;j<n;j++)
                           Tblock[i][j]=block[i][n-1-j];
                   break;
          default: printf("fatal error\n");
                       exit(0);
          }
          return;
      }
/******************************/
/*  The Least squares function */
/******************************/
void Leastsquares(int n, float x[1024], float y[1024], float coef[2], float *error)
{    int i, j;
     float s1, s2, s3, s4;
     float xbar, ybar,sum;
     s1=0.0; s2=0.0; s3=0.0;s4=0.0;
     xbar=0.0; ybar=0.0; sum=0.0;
     for (i=0; i<n; i++)
     {   s1=s1+x[i];
 s2=s2+y[i];
 s3=s3+x[i]*y[i];
 s4=s4+pow(x[i],2);
     }
     xbar=s1/n;
     ybar=s2/n;
     if ((s4-pow(s1,2)/n)==0.0)
     {   printf("error in Leastsquares()\n");
         exit(0);
     }
    else
    {   coef[1]=(s3-s1*s2/n)/(s4-pow(s1,2)/n);
        coef[0]=ybar-coef[1]*xbar;
    }
    sum=0.0;
    for (i=0; i<n; i++)
    {   sum=sum+pow((y[i]-coef[0]-coef[1]*x[i]),2);}
    *error=sqrt(sum/n);
     return;
}
/*************************************************************************/
/*************************************************************************/
/*    Bath Fractal Transform (BFT) scheme                                */
/*         input:       image_in.dat                                     */
/*         output:      image_bath.out                                   */
/*************************************************************************/
/*************************************************************************/
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#define SWAP(a1,b1) {float temp=(a1);(a1)=(b1);(b1)=temp;}
#define SIZE    512
#define NMAPS   8
#define MAXITER 25
void getchildBlock(int n, int n1, int n2, float block[][n]);
void getparentBlock(int n,int n1,int n2,float block[][n],float x2[64],float x3[64]);
void Dec_getparentBlock(int n,int n1,int n2,float block[][n],float x2[64],float x3[64]);
float dot_product(int n, float u[n], float v[n]);
void gauss(float a1[4][4],float b1[4], float b2[4]);
float lena[SIZE][SIZE], lena0[SIZE][SIZE], lena1[SIZE][SIZE];
main()
```

```
{
int i, j, iic, jjc, iip, jjp, npixc, npixp, neq, iter;
int nchilds, nparents, ic, jc, ncount,npts;
float error, errmin, pixel, sum, rmse, psnr;
float tot;
float childb[8][8], parentb[16][16];
float x1[64], x2[64], x3[64], y[64];
float constant[64][64];
float x_loc[64][64],y_loc[64][64], greyc[64][64];
float ones[64], a[4][4], b[4], coef[4];
FILE * finput;
FILE * foutput1;
FILE * foutput2;
finput=fopen("lenna0.dat","r");
foutput1=fopen("lenab.dat","w");
foutput2=fopen("lenab.out","w");
nchilds=64;
nparents=32;
npixp=SIZE/nparents;
npixc=SIZE/nchilds;
for (i=0;i<SIZE;i++)
{    for (j=0;j<SIZE;j++)
     {   fscanf(finput, "%f",&pixel);
         lena[i][j]=pixel/255.0;
         lena0[i][j]=0.0;
         lena1[i][j]=0.0;
      }
    }
for (iic=0;iic<nchilds;iic++) // for each child block
{   for (jjc=0;jjc<nchilds;jjc++)
    {    getchildBlock(npixc,iic, jjc,childb);
         ncount=0;
         for (i=0;i<npixc;i++)
         {   for (j=0;j<npixc;j++)
             {   y[ncount]=childb[i][j];
                 ones[ncount]=1.0;
                 ncount++;
             }
         }
         getparentBlock(npixp,iic,jjc,parentb,x2,x3); // get the co-centric parent block
         ncount=0;
         for (i=1; i<npixp; i=i+2)
         {   for (j=1; j<npixp; j=j+2)
             {    x1[ncount]=(parentb[i-1][j-1]+parentb[i-1][j]+
                            parentb[i][j-1]+parentb[i][j])/4.0;
                  ncount++;
             }
          }
          npts=npixc*npixc;
          a[0][0]=dot_product(npts,x1,x1);
  a[0][1]=dot_product(npts,x1,x2);
  a[0][2]=dot_product(npts,x1,x3);
    a[0][3]=dot_product(npts,x1,ones);
  a[1][0]=dot_product(npts,x1,x2);
  a[1][1]=dot_product(npts,x2,x2);
  a[1][2]=dot_product(npts,x2,x3);
  a[1][3]=dot_product(npts,x2,ones);
  a[2][0]=dot_product(npts,x1,x3);
  a[2][1]=dot_product(npts,x2,x3);
  a[2][2]=dot_product(npts,x3,x3);
  a[2][3]=dot_product(npts,x3,ones);
  a[3][0]=dot_product(npts,x1,ones);
  a[3][1]=dot_product(npts,x2,ones);
  a[3][2]=dot_product(npts,x3,ones);
  a[3][3]=dot_product(npts,ones,ones);
          b[0]=dot_product(npts,x1,y);
  b[1]=dot_product(npts,x2,y);
```

```
          b[2]=dot_product(npts,x3,y);
  b[3]=dot_product(npts,ones,y);
          gauss(a, b, coef);
  greyc[iic][jjc]=coef[0];
          x_loc[iic][jjc]=coef[1];
  y_loc[iic][jjc]=coef[2];
  constant[iic][jjc]=coef[3];
          error=0.0;
  for (i=0;i<npts;i++)
          {   for (j=0;j<npts;j++)
              {   error=error+pow(y[ncount]-coef[0]*x1[ncount]-coef[1]*x2[ncount]
        -coef[2]*x3[ncount]-coef[3],2);
      }
          }
  error=sqrt(error/npts);
     }
}
for (iter=0; iter<MAXITER; iter++)   // start the decoding process
{   for (iic=0; iic<nchilds; iic++)
    {   for (jjc=0; jjc<nchilds; jjc++)
        {   Dec_getparentBlock(npixp,iic,jjc,parentb, x2, x3);
            ncount=0;
            for (i=1; i<npixp; i=i+2)
            {   for (j=1; j<npixp; j=j+2)
{   ic=(i-1)/2;
                    jc=(j-1)/2;
                    childb[ic][jc] =greyc[iic][jjc]*(parentb[i][j]+
                    parentb[i][j-1]+ parentb[i-1][j]+parentb[i-1][j-1])/4.0
                    +x_loc[iic][jjc]*x2[ncount]+y_loc[iic][jjc]*x3[ncount]
    +constant[iic][jjc];
                    ncount++;
                    lena1[iic*npixc+ic][jjc*npixc+jc]=childb[ic][jc];
                }
            }
        }
    }
    for (i=0; i<SIZE; i++)
    {   for (j=0; j<SIZE; j++)
        {   lena0[i][j]=lena1[i][j];}
    }
}
sum=0;
tot=0;
for (i=0; i<SIZE; i++)
{   for (j=0; j<SIZE; j++)
    {   sum=sum+pow((lena1[i][j]-lena[i][j]),2);
        fprintf(foutput1,"%.3f\n", 255.0*lena1[i][j]);
    }
}
rmse=255.0*sqrt(sum/pow(SIZE,2));
psnr=20*log10(255.0/rmse);
printf(" RMSE = %f\n ", rmse);
printf(" PSNR = %f\n", psnr);
fclose(finput);
fclose(foutput1);
fclose(foutput2);
}
/****************************************/
/* This function sets up the child block */
/****************************************/
void getchildBlock(int n, int n1, int n2, float block[][n])
{    int i, j;
     for (i=0; i<n;i++)
     {   for (j=0; j<n;j++)
         {   block[i][j]=lena[n1*n+i][n2*n+j];}
     }
     return;
```

```
}
/*****************************************/
/* This function sets up the parent block */
/*****************************************/
void getparentBlock(int n,int n1,int n2,float block[][n],float x2[64],float x3[64])
{
int i, j, m1, m2, ncount;
switch(n1)
{   case(0): if (n2<63)
             {  m1=0; m2=n2*n/2;}
             else
             {  m1=0; m2=(n2-1)*n/2;}
             break;
    case(63):if (n2<63)
             {  m1=(n1-1)*n/2; m2=n2*n/2;}
             else
             {  m1=(n1-1)*n/2; m2=(n2-1)*n/2;}
             break;
    default : break;
}
switch(n2)
{
    case(0): if (n1<63)
             {  m1=n1*n/2; m2=0;}
             else
             {  m1=(n1-1)*n/2; m2=0;}
             break;
    case(63):if (n1<63)
             {  m1=n1*n/2; m2=(n2-1)*n/2;}
             else
             {  m1=(n1-1)*n/2; m2=(n2-1)*n/2;}
             break;
    default : break;
}
if ((n1>0)&&(n1<63)&&(n2>0)&&(n2<63))
   {m1=n1*n/2-n/4; m2=n2*n/2-n/4;}
for (i=0; i<n;i++)
{   for (j=0; j<n;j++)
    {   block[i][j]=lena[m1+i][m2+j];}
}
ncount=0;
for (i=0; i<n; i=i+2)
{   for (j=0; j<n; j=j+2)
    {   x2[ncount]=(m2+j+0.5)/512.0;
        x3[ncount]=(m1+i+0.5)/512.0;
        ncount=ncount+1;
    }
}
return;
}
/*****************************************/
/* This function sets up the parent block */
/*****************************************/
void Dec_getparentBlock(int n,int n1,int n2,float block[][n],float x2[64],float x3[64])
{
int i, j, m1, m2, ncount;
switch(n1)
{   case(0): if (n2<63)
             {  m1=0; m2=n2*n/2;}
             else
             {  m1=0; m2=(n2-1)*n/2;}
             break;
    case(63):if (n2<63)
             {  m1=(n1-1)*n/2; m2=n2*n/2;}
             else
             {  m1=(n1-1)*n/2; m2=(n2-1)*n/2;}
             break;
    default : break;
}
switch(n2)
{
    case(0): if (n1<63)
             {  m1=n1*n/2; m2=0;}
             else
             {  m1=(n1-1)*n/2; m2=0;}
             break;
    case(63):if (n1<63)
             {  m1=n1*n/2; m2=(n2-1)*n/2;}
             else
             {  m1=(n1-1)*n/2; m2=(n2-1)*n/2;}
             break;
    default : break;
}
if ((n1>0)&&(n1<63)&&(n2>0)&&(n2<63))
   {m1=n1*n/2-n/4; m2=n2*n/2-n/4;}
for (i=0; i<n;i++)
{   for (j=0; j<n;j++)
    {   block[i][j]=lena0[m1+i][m2+j];}
}
ncount=0;
for (i=0; i<n; i=i+2)
{   for (j=0; j<n; j=j+2){
        x2[ncount]=(m2+j+0.5)/512.0;
        x3[ncount]=(m1+i+0.5)/512.0;
        ncount=ncount+1;
    }
}
return;
}
/*******************************************************************/
/* The Gaussian function solves the system of linear equations: Ax=b */
/*******************************************************************/
void gauss(float a1[4][4],float b1[4], float b2[4])
{
int indxc[4], indxr[4], ipiv[4];
int i, icol, irow, j, k, l, ll,m,n;
float big, dum, pivinv, a[4][4],b[4][1];
n=4;
m=1;
for (i=0;i<n;i++)
for (j=0;j<n;j++)
{   a[i][j]=a1[i][j];}
for (i=0;i<n;i++)
{   indxc[i]=0;
    indxr[i]=0;
    ipiv[i]=0;
    b[i][0]=b1[i];
}
for (j=0; j<n; j++)
    ipiv[j]=0;
for (i=0; i<n; i++)
{   big=0.0;
    for (j=0; j<n; j++)
        if (ipiv[j] != 1)
           for (k=0; k<n; k++) {
               if (ipiv[k] == 0){
                  if (fabs(a[j][k]) >= big){
                     big=fabs(a[j][k]);
                     irow=j;
                     icol=k;
                   }
               } else if (ipiv[k]>1) printf("GAUSS:  singular Matrix-2\n");
           }
++(ipiv[icol]);
if (irow != icol){
   for (l=0; l<n;l++) SWAP(a[irow][l],a[icol][l]);
   for (l=0;l<m;l++)  SWAP(b[irow][l],b[icol][l]);
}
indxr[i]=irow;
indxc[i]=icol;
if (a[icol][icol]==0.0) printf("GAUSS:  singular Matrix-2\n");
pivinv=1.0/a[icol][icol];
```

```
a[icol][icol]=1.0;
for (l=0; l<n; l++) a[icol][l] *= pivinv;
for (l=0; l<m; l++) b[icol][l] *= pivinv;
for (ll=0; ll<n;ll++)
    if (ll != icol) {
       dum=a[ll][icol];
       a[ll][icol]=0.0;
       for (l=0;l<n;l++) a[ll][l] -= a[icol][l]*dum;
       for (l=0;l<m;l++) b[ll][l] -= b[icol][l]*dum;
    }
}
for (l=n-1;l>=0;l--){
    if (indxr[l] != indxc[l])
       for (k=1;k<=n;k++)
           SWAP(a[k][indxr[l]],a[k][indxc[l]]);
}
for (l=0;l<n;l++) b2[l]=b[l][0];
return;
}
/****************************************************************************/
/* This function computes the dot product betwwen two vecors */
/****************************************************************************/
float dot_product(int n, float u[n], float v[n])
{
float sum;
int i;
sum=0.0;
for (i=0;i<n;i++) {
    sum=sum+u[i]*v[i];
}
return(sum);
}
/*****************************************************************************/
/*****************************************************************************/
/*  Quadtree Fractal Image Coding:                                          */
/*         Variance decomposition criterion                                 */
/*         Starting at (M,N)=(2,4)                                          */
/*         Ending at (M,N)=(68,132)                                         */
/*         Input: input_image.dat                                           */
/*         Output:output_image.out                                          */
/*****************************************************************************/
/*****************************************************************************/
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#define SIZE    512
#define MAXITER 20
void GetBlock(int n, int n1, int n2, float block[][n]);
float Uniform(int level, float low, float up, float x);
void Leastsquares(int n, float x[n], float y[n], float coef[2], float *error);
void Exhaustive(int npixc,int iic,int jjc,int indexp[2], float coef[2],float *error);
void Split(int npixc, int iic, int jjc);
void Statistics(int npixc,int iic,int jjc,float min_var,
               float *mean,float *variance, int *status);
float Entropy(int size, int level, float low, float up, float vector[size][]);
float lena[SIZE][SIZE], lena0[SIZE][SIZE], lena1[SIZE][SIZE];
float partition[SIZE+1][SIZE+1];
main()
{
int levels,skip,i,j,iic,jjc,iip,jjp,npixc,npixp,iter,npixc1;
int nchilds,nparents,ic,jc,ncount,status,index[2];
int index1[4][4],index2[8][8],index3[16][16];
int index4[32][32],index5[64][64],index6[128][128];
int iindex1[4][4],iindex2[8][8],iindex3[16][16];
int iindex4[32][32],iindex5[64][64],iindex6[128][128];
int jindex1[4][4],jindex2[8][8],jindex3[16][16];
int jindex4[32][32],jindex5[64][64],jindex6[128][128];
int nsplit,repeat,last,number,nchilds0,nparents0,count,level;
int level1,level2,level3,level4,level5,level6;
float errmin,error,threshold,pixel,sum,rmse,psnr,distance,s,c;
float childb[128][128],parentb[256][256],coef[2];
float mean,variance,ave,constant6[128][128];
float constant1[4][4],constant2[8][8],constant3[16][16];
float constant4[32][32],constant5[64][64],coefficients[128*128][2];
float slope1[4][4],slope2[8][8],slope3[16][16];
float slope4[32][32],slope5[64][64],slope6[128][128];
float threshold0,c_ratio,entropy,low,high,bit_rate;
FILE * finput1;
FILE * foutput1;
finput1=fopen("input_image.dat","r");
foutput1=fopen("output_image.out","w");
nparents0=2; //M=2
nchilds0=4; //N=4
threshold0=1000;
level=256;
low=-5.0;
high=5.0;
sum=0.0;
for (i=0;i<SIZE;i++)
{   for (j=0;j<SIZE;j++)
    {    fscanf(finput1, "%f",&pixel);
         lena[i][j]=pixel;
         lena1[i][j]=0;
         lena0[i][j]=0;
      }
}
for (repeat=0;repeat<1;repeat++)
{  threshold=threshold0-repeat*100;
   for (i=0;i<SIZE/4;i++)
   {   for (j=0;j<SIZE/4;j++)
       {   index6[i][j]=2;
           index5[i/2][j/2]=2;
           index4[i/4][j/4]=2;
           index3[i/8][j/8]=2;
           index2[i/16][j/16]=2;
           index1[i/32][j/32]=2;
       }
   }
   for (i=0;i<SIZE;i++)
   {   for (j=0;j<SIZE;j++)
       {   lena0[i][j]=0.0;
           lena1[i][j]=0.0;
       }
   }
npixc=SIZE/nchilds0;
for (i=0;i<=SIZE;i++)
{   for (j=0;j<=SIZE;j++)
    {   if ((i%npixc)==0 || (j%npixc)==0)
        {  partition[i][j]==0.0;}
        else
        {  partition[i][j]=255.0;}
    }
}
level1=level2=level3=level4=level5=level6=0;
ncount=0;
last=0;
nsplit=0;
printf("Start the encoding process\n");
for (count=1;count<=6;count++)
{   nchilds=pow(2,count-1)*nchilds0;
    nparents=pow(2,count-1)*nparents0;
    npixc=SIZE/nchilds;
    npixp=SIZE/nparents;
    printf("Encoding: working on layer number : %d\n",count);
    for (iic=0;iic<nchilds;iic++)
    {    for (jjc=0;jjc<nchilds;jjc++)
         {   if (count==1)
{  Statistics(npixc,iic,jjc,threshold,&mean,&variance,&status);
   if (status==0)
   {  index1[iic][jjc]=0;
      Exhaustive(npixc,iic,jjc,index,coef,&error);
      iindex1[iic][jjc]=index[0];
      jindex1[iic][jjc]=index[1];
      slope1[iic][jjc]=coef[0];
      constant1[iic][jjc]=coef[1];
      coefficients[ncount][0]=coef[0];
      coefficients[ncount][1]=coef[1];
      ncount++;
   }
   if (status==1)
   {  index1[iic][jjc]=1;
      Split(npixc,iic,jjc);
      nsplit++;
   }
}
if (count==2 && index1[iic/2][jjc/2]==1)
{  Statistics(npixc,iic,jjc,threshold,&mean,&variance,&status);
   if (status==0)
   {  index2[iic][jjc]=0;
      Exhaustive(npixc,iic,jjc,index,coef,&error);
      iindex2[iic][jjc]=index[0];
      jindex2[iic][jjc]=index[1];
      slope2[iic][jjc]=coef[0];
      constant2[iic][jjc]=coef[1];
      coefficients[ncount][0]=coef[0];
      coefficients[ncount][1]=coef[1];
      ncount++;
   }
   if (status==1)
   {  index2[iic][jjc]=1;
      Split(npixc,iic,jjc);
      nsplit++;
   }
}
if (count==3 && index2[iic/2][jjc/2]==1)
{  Statistics(npixc,iic,jjc,threshold,&mean,&variance,&status);
   if (status==0)
   {   index3[iic][jjc]=0;
       Exhaustive(npixc,iic,jjc,index,coef,&error);
       iindex3[iic][jjc]=index[0];
       jindex3[iic][jjc]=index[1];
       slope3[iic][jjc]=coef[0];
       constant3[iic][jjc]=coef[1];
       coefficients[ncount][0]=coef[0];
       coefficients[ncount][1]=coef[1];
       ncount++;
   }
   if (status==1)
   {  index3[iic][jjc]=1;
      Split(npixc,iic,jjc);
      nsplit++;
   }
}
if (count==4 && index3[iic/2][jjc/2]==1)
{  Statistics(npixc,iic,jjc,threshold,&mean,&variance,&status);
   if (status==0)
   {   index4[iic][jjc]=0;
       Exhaustive(npixc,iic,jjc,index,coef,&error);
       iindex4[iic][jjc]=index[0];
       jindex4[iic][jjc]=index[1];
       slope4[iic][jjc]=coef[0];
       constant4[iic][jjc]=coef[1];
       coefficients[ncount][0]=coef[0];
       coefficients[ncount][1]=coef[1];
```

```
                    ncount++;
                }
                if (status==1)
                {  index4[iic][jjc]=1;
                   Split(npixc,iic,jjc);
                   nsplit++;
                }
             }
             if (count==5 && index4[iic/2][jjc/2]==1)
             {  Statistics(npixc,iic,jjc,threshold,&mean,&variance,&status);
                if (status==0)
                {   index5[iic][jjc]=0;
                    Exhaustive(npixc,iic,jjc,index,coef,&error);
                    iindex5[iic][jjc]=index[0];
                    jindex5[iic][jjc]=index[1];
                    slope5[iic][jjc]=coef[0];
                    constant5[iic][jjc]=coef[1];
                    coefficients[ncount][0]=coef[0];
                    coefficients[ncount][1]=coef[1];
                    ncount++;
                }
                if (status==1)
                {  index5[iic][jjc]=1;
                   Split(npixc,iic,jjc);
                   nsplit++;
                }
             }
             if (count==6 && index5[iic/2][jjc/2]==1)
             {  index6[iic][jjc]=0;
                Exhaustive(npixc,iic,jjc,index,coef,&error);
                iindex6[iic][jjc]=index[0];
                jindex6[iic][jjc]=index[1];
                slope6[iic][jjc]=coef[0];
                constant6[iic][jjc]=coef[1];
                coefficients[ncount][0]=coef[0];
                coefficients[ncount][1]=coef[1];
                ncount++;
                last++;
             }
         }
    }
}
number=ncount;
entropy=Entropy(number,level,low,high,coefficients);
for (iter=0;iter<MAXITER;iter++)
{
for (count=1;count<=6;count++)
{   nchilds=pow(2,count-1)*nchilds0;
    nparents=pow(2,count-1)*nparents0;
    npixc=SIZE/nchilds;
    npixp=SIZE/nparents;
    for (iic=0;iic<nchilds;iic++)
    {    for (jjc=0;jjc<nchilds;jjc++)
         {   skip=1;
             if (count==1 && index1[iic][jjc]==0)
             {   iip= iindex1[iic][jjc];
                 jjp= jindex1[iic][jjc];
                 s=slope1[iic][jjc];
                 c=constant1[iic][jjc];
                 skip=0;
             }
             if (count==2 && index2[iic][jjc]==0)
             {   iip= iindex2[iic][jjc];
                 jjp= jindex2[iic][jjc];
                 s=slope2[iic][jjc];
                 c=constant2[iic][jjc];
                 skip=0;
             }
             if (count==3 && index3[iic][jjc]==0)
             {   iip= iindex3[iic][jjc];
                 jjp= jindex3[iic][jjc];
                 s=slope3[iic][jjc];
                 c=constant3[iic][jjc];
                 skip=0;
             }
             if (count==4 && index4[iic][jjc]==0)
             {   iip= iindex4[iic][jjc];
                 jjp= jindex4[iic][jjc];
                 s=slope4[iic][jjc];
                 c=constant4[iic][jjc];
                 skip=0;
             }
             if (count==5 && index5[iic][jjc]==0)
             {   iip= iindex5[iic][jjc];
                 jjp= jindex5[iic][jjc];
                 s=slope5[iic][jjc];
                 c=constant5[iic][jjc];
                 skip=0;
             }
             if (count==6 && index6[iic][jjc]==0)
             {   iip= iindex6[iic][jjc];
                 jjp= jindex6[iic][jjc];
                 s=slope6[iic][jjc];
                 c=constant6[iic][jjc];
                 skip=0;
             }
             if (skip==0)
             {  for (i=0; i<npixp;i++)
                {   for (j=0; j<npixp;j++)
                    {   parentb[i][j]=lena0[iip*npixp+i][jjp*npixp+j];}
                }
                for (i=1; i<npixp; i=i+2)
                {   for (j=1; j<npixp; j=j+2)
                    {   ic=(i-1)/2;
                        jc=(j-1)/2;
                        ave=(parentb[i][j]+parentb[i][j-1]+
                             parentb[i-1][j]+parentb[i-1][j-1])/4.0;
                        childb[ic][jc]=s*ave+c;
                        lena1[iic*npixc+ic][jjc*npixc+jc]=s*ave+c;
                    }
                }
             }
        }
    }
} /** end of the count loop...****/
for (i=0; i<SIZE; i++)
{  for (j=0; j<SIZE; j++)
   {   lena0[i][j]=lena1[i][j];}  /* copy lena1 into lena0 and iterate */
}
} /*** end of the iteration loop ***/
printf("\n");
sum=0;
for (i=0; i<SIZE; i++)
{   for (j=0; j<SIZE; j++)
    {   sum=sum+pow((lena1[i][j]-lena[i][j]),2);
        fprintf(foutput1, "%f\n", 255*lena1[i][j]);
    }
}
rmse=255.0*sqrt(sum/pow(SIZE,2));
psnr=20*log10(255.0/rmse);
c_ratio=(float)(pow(2,21))/(nsplit+number-last+number*10+
2*(level1+level2*2+level3*3+level4*4+level5*5+level6*6));
bit_rate=8.0/c_ratio;
```

```
printf("\n");
printf("_____________________________________________________\n");
printf(" Iteration = %d\n with variance threshol = %.2f\n", repeat, threshold);
printf("The RMSE = %f\n ", rmse);
printf("The PSNR = %f\n ", psnr);
printf("The Compression Ratio = %f\n ", c_ratio);
printf("\n");
printf("==================================================================\n");
}  /**** end of the repeat loop ****/
fclose(finput1);
fclose(finput2);
fclose(foutput1);
fclose(foutput2);
}
/*********************************************************************/
/* This function sets up the child/parent block                     */
/*********************************************************************/
void GetBlock(int n, int n1, int n2, float block[][n])
{
int i, j;
for (i=0; i<n;i++)
{   for (j=0; j<n;j++)
    {   block[i][j]=lena[n1*n+i][n2*n+j];}
}
return;
}
/*********************************************************************/
/*  This is a Uniform quatizer that quatizes the scaling coefficients   */
/*  into the specified number of levels on the given range.          */
/*********************************************************************/
float  Uniform(int level, float low, float up, float x)
{
int num;
float q,step;
step=(up-low)/level;
if (x<=low)
{  q=low+step/2.0;}
if (x>=up)
   {q=up-step/2.0;}
if (x>low && x<up)
{  num=(int)(level*(x-low)/(up-low));
   q=low+((float)(num)+0.5)*step;
}
return(q);
}
/**********************************************************************/
/*  The Least squares function                                        */
/**********************************************************************/
void Leastsquares(int n, float x[n], float y[n], float coef[2], float *error)
{
int i,j;
float s1,s2, s3, s4;
float xbar, ybar,sum;
float s,c;
s1=0.0; s2=0.0; s3=0.0;s4=0.0;
xbar=0.0; ybar=0.0; sum=0.0;
for (i=0; i<n; i++)
{ s1=s1+x[i];
  s2=s2+y[i];
  s3=s3+x[i]*y[i];
  s4=s4+pow(x[i],2);
}
xbar=s1/n;
ybar=s2/n;
if ((s4-pow(s1,2)/n)==0.0)
{   s=0;
    c=ybar;
    printf("ZERO DETERMINANT IN LEASTSQUARES\n");
}
else
{   s=(s3-s1*s2/n)/(s4-pow(s1,2)/n);
    c=ybar-s*xbar;
}
coef[0]=s;
coef[1]=c;
sum=0.0;
for (i=0; i<n; i++)
{   sum=sum+pow((y[i]-(coef[0]*x[i]+coef[1])),2);}
*error=sqrt(sum/n);
return;
}
/************************************************************************/
/* This function finds the closest parent block to a child block          */
/* and returns:                                                           */
/* The address of the parent-block                                        */
/* The optimum coefficients                                               */
/* The collage error                                                      */
/************************************************************************/
void Exhaustive(int npixc,int iic,int jjc,int indexp[2],float coef[2],float *error)
{
int  i,j,iip,jjp,ncount,nparents,npixp;
float childb[npixc][npixc], y[npixc*npixc], parentb[2*npixc][2*npixc], x[npixc*npixc];
float err,errmin,coefficients[2];
ncount=0;
errmin=100000000;
for (i=0; i<npixc;i++)
{   for (j=0; j<npixc;j++)
    {   childb[i][j]=lena[iic*npixc+i][jjc*npixc+j];}
}
for (i=0;i<npixc;i++)
{   for (j=0;j<npixc;j++)
    {   y[ncount]=childb[i][j];
        ncount++;
    }
}
npixp=2*npixc;
nparents=SIZE/npixp;
for (iip=0; iip<nparents; iip++)
{   for (jjp=0; jjp<nparents; jjp++)
    {   for (i=0; i<npixp;i++)
        {   for (j=0; j<npixp;j++)
            {   parentb[i][j]=lena[iip*npixp+i][jjp*npixp+j];}
        }
        ncount=0;
        for (i=1; i<npixp; i=i+2)
        {   for (j=1; j<npixp; j=j+2)
            {   x[ncount]=(parentb[i-1][j-1]+parentb[i-1][j]+
                           parentb[i][j-1]+parentb[i][j])/4.0;
                ncount++;
            }
        }
        Leastsquares(npixc*npixc, x, y, coefficients, &err);
        if (err < errmin)
        {  errmin=err;
           coef[0]=coefficients[0];
           coef[1]=coefficients[1];
           indexp[0]=iip;
           indexp[1]=jjp;
           *error=err;
        }
    }
}
return;
}
/*********************************************************************/
/*   this function splits a quadtree into four quadtrees              */
```

```
/*********************************************************************/
void Split(int npixc, int iic, int jjc)
{
int i,shift1,shift2;
shift1=iic*npixc;
shift2=jjc*npixc;
for (i=0;i<=npixc;i++)
{   partition[shift1+i][shift2+npixc/2]=0.0;
    partition[shift1+npixc/2][shift2+i]=0.0;
}
return;
}
/*****************************************************************************/
/*   this functions computes the statistics(mean and variance) of a child block*/
/*   and returns 1 if the variance is greater than some defined THRESHOLD      */
/*****************************************************************************/
void Statistics(int npixc,int iic,int jjc,float min_var,
               float *mean,float *variance, int *status)
{
int i,j;
float block[npixc][npixc], sum;
float ave, var;
for (i=0; i<npixc;i++)
{   for (j=0; j<npixc;j++)
    {   block[i][j]=lena[iic*npixc+i][jjc*npixc+j];}
}
sum=0.0;
for (i=0;i<npixc;i++)
{   for (j=0;j<npixc;j++)
    {   block[i][j]=255.0*block[i][j];
        sum=sum+block[i][j];
    }
}
*mean=sum/(npixc*npixc);
ave=sum/(npixc*npixc);;
sum=0.0;
for (i=0;i<npixc;i++)
{  for (j=0;j<npixc;j++)
   {   sum=sum+pow((block[i][j]-ave),2);}
}
*variance=sum/(npixc*npixc);
var=sum/(npixc*npixc);
if (var>min_var)
{   *status=1;}
else
{   *status=0;}
return;
}
/*********************************************************************/
/*  This is function computes the entropy of a set of quatized values   */
/*  to be used for compression ratio...                                 */
/*********************************************************************/
float Entropy(int size, int level, float low, float up, float vector[size][2])
{     int i,j,num,freq[512];
      float sum,x,step;
      step=(up-low)/level;
      for (i=0;i<level;i++)
          freq[i]=0;
      for (i=0;i<size;i++)
      {   for (j=0;j<2;j++)
          {   x=vector[i][j];
              if (x<=low)
              {  freq[0]=freq[0]+1;}
              if (x>=up)
              {  freq[level-1]=freq[level-1]+1;}
              if (x>low && x<up)
              {  num=(int)(level*(x-low)/(up-low));
                 freq[num]=freq[num]+1;
```

```
            }
        }
    }
    sum=0.0;
    for (i=0;i<level;i++)
    {   if (freq[i]>0)
        {  sum=sum-((float)(freq[i])/(2.0*size))
               *(log10((float)(freq[i])/(2.0*size)))/log10(2.0);}
    }
    return(sum);
}
/*****************************************************************/
/*****************************************************************/
/*    Exhaustive Fractal-Wavelet scheme (FW-I)                   */
/*****************************************************************/
/*****************************************************************/
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#define SIZE    512
main()
{
int i,j,ii,jj,lk1,lk2,lkk1,lkk2,k1,k2,k,lx1,lx2,nsize,block;
int iindexh[32][32], iindexv[32][32],iindexd[32][32];
int jindexh[32][32], jindexv[32][32],jindexd[32][32];
int size1,size2,m,n,lx,l1,l2,iindex,jindex;
int nq,iter;
float psnr,rmse,pixel,sum,sum1,sum2,err,errormin,C[512*512];
float Ak2ij[85], Ak1ij[85], wcoef0[512][512],wcoef[512][512],wcoef2[512][512];
float alpha, alph, alphah[32][32],alphav[32][32],alphad[32][32];
FILE * finput0;
FILE * finput1;
finput0=fopen("lena_wv_in0.dat","r");
foutput1=fopen("lena_wv_out.out","w");
for (i=0;i<SIZE;i++)
{   for (j=0;j<SIZE;j++)
    {    fscanf(finput0, "%f",&pixel);
         wcoef0[i][j]=pixel;
         wcoef[i][j]=pixel;
         wcoef2[i][j]=0.0;
    }
}
k1=4;// k1*=4;
k2=5;//k2*=5;
lkk1=pow(2,k1);
lkk2=pow(2,k2);
for (block=1; block<=3; block++)
{    for (i=0; i<pow(2,k2); i++)
     {   for (j=0; j<pow(2,k2); j++)
         {    lk2=lkk2;
              lx2=0;
              size2=1;
              if (block==1)
                 Ak2ij[lx2]=wcoef[i][lk2+j];
              if (block==2)
                 Ak2ij[lx2]=wcoef[i+lk2][j];
              if (block==3)
                 Ak2ij[lx2]=wcoef[i+lk2][j+lk2];
              for (k=k2;k<=7;k++)
              {   lk2=lk2+pow(2,k);
                  size2=2*size2;
                  for (m=0; m<size2; m++)
                  {   for (n=0; n<size2; n++)
                      {   lx2=lx2+1;
                          if (block==1)
                             Ak2ij[lx2]=wcoef[size2*i+m][lk2+size2*j+n];
                          if (block==2)
                             Ak2ij[lx2]=wcoef[lk2+size2*i+m][size2*j+n];
                          if (block==3)
```

```
                    Ak2ij[lx2]=wcoef[lk2+size2*i+m][lk2+size2*j+n];
             }
          }
     }
     nsize=lx2+1;
     errormin=100000.0;
     for (ii=0;ii<pow(2,k1); ii++)
     {   for (jj=0; jj<pow(2,k1); jj++)
         {   lk1=lkk1;
             size1=1;
             lx1=0;
             if (block==1)
                Ak1ij[lx1]=wcoef[ii][lk1+jj];
             if (block==2)
                Ak1ij[lx1]=wcoef[lk1+ii][jj];
             if (block==3)
                Ak1ij[lx1]=wcoef[lk1+ii][lk1+jj];
             for (k=k1; k<7; k++)
             {   lk1=lk1+pow(2,k);
                 size1=2*size1;
                 for (m=0; m<size1; m++)
                 {  for (n=0; n<size1; n++)
                    {   lx1=lx1+1;
                        if (block==1)
                           Ak1ij[lx1]=
                               wcoef[size1*ii+m][lk1+size1*jj+n];
                        if (block==2)
                           Ak1ij[lx1]=
                               wcoef[lk1+size1*ii+m][size1*jj+n];
                        if (block==3)
                           Ak1ij[lx1]=
                               wcoef[lk1+size1*ii+m][lk1+size1*jj+n];
                    }
                 }
             }
             sum1=0.0;
             sum2=0.0;
             for (lx=0; lx<nsize; lx++)
             {    sum1=sum1+Ak2ij[lx]*Ak1ij[lx];
                  sum2=sum2+Ak1ij[lx]*Ak1ij[lx];}
             alph=sum1/sum2;
             sum=0.0;
             for (lx=0; lx<nsize; lx++)
             {   sum=sum+pow((Ak2ij[lx]-alph*Ak1ij[lx]),2); }
             err=sum/nsize;
             if (err<=errormin)
             {  errormin=err;
                alpha=alph;
                iindex=ii;
                jindex=jj;
             }
         }
     }
     if (block==1)
     {   alphah[i][j]=alpha;
         iindexh[i][j]=iindex;
         jindexh[i][j]=jindex;
     }
     if (block==2)
     {   alphav[i][j]=alpha;
         iindexv[i][j]=iindex;
         jindexv[i][j]=jindex;
     }
     if (block==3)
     {   alphad[i][j]=alpha;
         iindexd[i][j]=iindex;
         jindexd[i][j]=jindex;
     }
}
```

```
     }
}
for (i=0;i<lkk2;i++)
    for (j=0;j<lkk2;j++)
        { wcoef2[i][j]=wcoef[i][j];}
for (block=1; block<=3; block++)
{    size2=1;
     lk1=lkk1;
     lk2=lkk2;
     for (k=k2;k<=8;k++)
     {   for (i=0; i<pow(2,k2);i++)
         {   for (j=0; j<pow(2,k2);j++)
             {    if (block==1)
                  {   ii=iindexh[i][j];
                      jj=jindexh[i][j];
                      alpha=alphah[i][j];}
                  if (block==2)
                  {   ii=iindexv[i][j];
                      jj=jindexv[i][j];
                      alpha=alphav[i][j]; }
                  if (block==3)
                  {   ii=iindexd[i][j];
                      jj=jindexd[i][j];
                      alpha=alphad[i][j]; }
                  for (m=0; m<size2; m++)
                  {   for (n=0; n<size2; n++)
                      {   if (block==1)
                             wcoef2[i*size2+m][lk2+j*size2+n]=
                             alpha*wcoef2[ii*size2+m][lk1+jj*size2+n];
                          if (block==2)
                             wcoef2[lk2+i*size2+m][j*size2+n]=
                             alpha*wcoef2[lk1+ii*size2+m][jj*size2+n];
                          if (block==3)
                             wcoef2[lk2+i*size2+m][lk2+j*size2+n]=
                             alpha*wcoef2[lk1+ii*size2+m][lk1+jj*size2+n];
                      }
                  }
             }
         }
         size2=2*size2;
         lk1=lk1+pow(2,k-1);
         lk2=lk2+pow(2,k);
     }
}
sum=0.0;
for (i=0;i<512;i++)
{   for (j=0;j<512;j++)
    {   fprintf(foutput1, "%.5f\n", wcoef2[i][j]);
        sum=sum+pow((wcoef0[i][j]-wcoef2[i][j]),2);
    }
}
rmse=sqrt(sum)/512.0;
psnr=20*log10(255.0/rmse);
printf("RMSE = %.5f\n",rmse);
printf("PSNR = %.5f\n",psnr);
fclose(foutput1);
fclose(finput0);
}
/****************************************************************************/
/****************************************************************************/
/*   Standard Fractal-Wavelet scheme (FW-II)                                */
/****************************************************************************/
/****************************************************************************/
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#define SIZE    512
main()
{
```

```
int i,j,ii,jj,lk1,lk2,lkk1,lkk2,k1,k2,k,lx1,lx2,nsize,block;
int iindexh[32][32],jindexh[32][32],nq;
int size1,size2,m,n,lx,l1,l2,kk1,kk2,iindex,jindex,kmax;
float alph2,rmse,pixel,sum,sum1,sum2,sum3,err,errormin;
float Ak2ij[3*85], Ak1ij[3*85], wcoef0[512][512],wcoef[512][512],wcoef2[512][512];
float alpha, alph, alphah[32][32],psnr;
FILE * finput0;
FILE * foutput1;
finput0=fopen("lena_wv_in0.dat","r");
foutput1=fopen("lena_wv_out.out","w");
for (i=0;i<SIZE;i++)
{   for (j=0;j<SIZE;j++)
    {    fscanf(finput0, "%f",&pixel);
         wcoef0[i][j]=pixel;
         wcoef[i][j]=pixel;
         wcoef2[i][j]=0.0;
    }
}
k1=4; //k1*=4
k2=5; //k2*=5
kmax=9;
lkk1=pow(2,k1);
lkk2=pow(2,k2);
kk1=0;
for (i=k2; i<kmax; i++)
{    kk1=kk1+pow(2,2*(i-k2));}
for (i=0; i<pow(2,k2); i++)
{   for (j=0; j<pow(2,k2); j++)
    {    lk2=lkk2;
         lx2=0;
         size2=1;
         Ak2ij[lx2]=wcoef[i][lk2+j];
         Ak2ij[lx2+kk1]=wcoef[i+lk2][j];
         Ak2ij[lx2+2*kk1]=wcoef[i+lk2][j+lk2];
         for (k=k2;k<=7;k++)
         {   lk2=lk2+pow(2,k);
             size2=2*size2;
             for (m=0; m<size2; m++)
             {   for (n=0; n<size2; n++)
                 {   lx2=lx2+1;
                     Ak2ij[lx2]=wcoef[size2*i+m][lk2+size2*j+n];
                     Ak2ij[lx2+kk1]=wcoef[lk2+size2*i+m][size2*j+n];
                     Ak2ij[lx2+2*kk1]=wcoef[lk2+size2*i+m][lk2+size2*j+n];
                  }
              }
         }
         nsize=3*(lx2+1);
         errormin=1000000000.0;
         for (ii=0;ii<pow(2,k1); ii++)
         {   for (jj=0; jj<pow(2,k1); jj++)
             {   lk1=lkk1;
                 size1=1;
                 lx1=0;
                 Ak1ij[lx1]=wcoef[ii][lk1+jj];
                 Ak1ij[lx1+kk1]=wcoef[lk1+ii][jj];
                 Ak1ij[lx1+2*kk1]=wcoef[lk1+ii][lk1+jj];
                 for (k=k1; k<7; k++)
                 {   lk1=lk1+pow(2,k);
                     size1=2*size1;
                     for (m=0; m<size1; m++)
                     {  for (n=0; n<size1; n++)
                        {   lx1=lx1+1;
                            Ak1ij[lx1]= wcoef[size1*ii+m][lk1+size1*jj+n];
                            Ak1ij[lx1+kk1]=wcoef[lk1+size1*ii+m][size1*jj+n];
                            Ak1ij[lx1+2*kk1]=wcoef[lk1+size1*ii+m][lk1+size1*jj+n];
                        }
                     }
                 }
                 sum1=0.0;
                 sum2=0.0;
                 sum3=0.0;
                 for (lx=0; lx<nsize; lx++)
                 {    sum1=sum1+Ak2ij[lx]*Ak1ij[lx];
                      sum2=sum2+Ak1ij[lx]*Ak1ij[lx];
                 }
                 alph=sum1/sum2;
                 sum=0.0;
                 for (lx=0;lx<nsize;lx++)
                 {   sum=sum+pow((Ak2ij[lx]-alph*Ak1ij[lx]),2); }
                 err=sum/nsize;
                 if (err<=errormin)
                 {  errormin=err;
                    alpha=alph;
                    iindex=ii;
                    jindex=jj;
                 }
             }
         }
         alphah[i][j]=alpha;
         iindexh[i][j]=iindex;
         jindexh[i][j]=jindex;
    }
}
for (i=0;i<lkk2;i++)
    for (j=0;j<lkk2;j++)
    {   wcoef2[i][j]=wcoef[i][j];}
size2=1;
lk1=lkk1;
lk2=lkk2;
for (k=k2;k<=8;k++)
{   for (i=0; i<pow(2,k2);i++)
    {   for (j=0; j<pow(2,k2);j++)
        {   ii=iindexh[i][j];
            jj=jindexh[i][j];
            alpha=alphah[i][j];
            for (m=0; m<size2; m++)
            {   for (n=0; n<size2; n++)
                {   wcoef2[i*size2+m][lk2+j*size2+n]=
                             alpha*wcoef2[ii*size2+m][lk1+jj*size2+n];
                    wcoef2[lk2+i*size2+m][j*size2+n]=
                             alpha*wcoef2[lk1+ii*size2+m][jj*size2+n];
                    wcoef2[lk2+i*size2+m][lk2+j*size2+n]=
                             alpha*wcoef2[lk1+ii*size2+m][lk1+jj*size2+n];
                }
            }
        }
     }
     size2=2*size2;
     lk1=lk1+pow(2,k-1);
     lk2=lk2+pow(2,k);
}
printf(" decoding process done...\n");
sum=0.0;
for (i=0;i<512;i++)
{    for (j=0;j<512;j++)
    {   fprintf(foutput1, "%.5f\n", wcoef2[i][j]);
        sum=sum+pow((wcoef0[i][j]-wcoef2[i][j]),2);
    }
}
rmse=sqrt(sum)/512.0;
psnr=20*log10(255.0/rmse);
printf("RMSE = %.5f\n",rmse);
printf("PSNR = %.5f\n",psnr);
printf("program completed successfully\n");
}
```

```
/************************************************************************/
/************************************************************************/
/*        Adaptive FW-II scheme                                         */
/************************************************************************/
/************************************************************************/
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#define SIZE    512
void  uniform(int sizeb2, int level, float low, float up, float alpha[][128],
float q[][128], float *entropy);
main()
{
int i,j,ii,jj,lk1,lk2,lkk1,lkk2,k1,k2,k,lx1,lx2,nsize,block;
int iindexh[128][128],jindexh[128][128],nq,shift;
int size1,size2,m,n,lx,l1,l2,kk1,kk2,iindex,jindex,kmax;
int sizeb2,sizeb1,count,ncount,k10,k20,num1, num2, num3, lx3;
int nsize1,nsize2;
int row, col, nshift;
float rmse,pixel,sum,sum1,sum2,err,errormin,bit_rate;
float Ak2ij[3*350], Ak1ij[3*350], wcoef[512][512],wcoef2[512][512];
float alpha, alph, alphah[128][128],psnr,c_ratio,entropy,low,high;
float alphaq[128][128],wcoef1[128][128],wcoefq[128][128],ent_a,ent_w;
float y1[350], y2[350], y3[350];
FILE * finput;
FILE * foutput;
finput=fopen("lena_wv_in0.dat","r");
foutput=fopen("rate_adapt.out","w");
for (i=0;i<SIZE;i++)
{    for (j=0;j<SIZE;j++)
    {    fscanf(finput, "%f",&pixel);
         wcoef[i][j]=pixel;
         wcoef2[i][j]=0.0;
    }
}
k10=3;
k20=4;
nshift=24;
kmax=9;
for (ncount=0;ncount<=nshift;ncount++)
{   if (ncount<8)
    {  k1=3;
       k2=4;
    }
    if (ncount>= 8 && ncount <24)
    {  k1=4;
       k2=5;
    }
    if (ncount==24)
    {  k1=5;
       k2=6;
    }
    shift=ncount;
    sizeb1=pow(2,k10)+shift;
    sizeb2=pow(2,k20)+2*shift;
    printf("shift = %d\n",shift);
    printf("sizeb1=%d    sizeb2=%d\n",sizeb1,sizeb2);
    for (i=0;i<SIZE;i++)
        for (j=0;j<SIZE;j++)
            wcoef2[i][j]=0.0;
lkk1=pow(2,k10)+shift;
lkk2=2*sizeb1;
for (i=0; i<sizeb2; i++)
{   for (j=0; j<sizeb2; j++)
    {    num1=0;
         num2=0;
         num3=0;
         nsize1=0;
         nsize2=0;
         lk2=lkk2;
         lx1=0;
         lx2=0;
         lx3=0;
         size2=1;
         count=0;
         y1[lx1]=wcoef[i][lk2+j];
         y2[lx2]=wcoef[lk2+i][j];
 y3[lx3]=wcoef[lk2+i][lk2+j];
         for (k=k2;k<=7;k++)
 {   count++;
     lk2=lk2+pow(2,count)*sizeb1;
             size2=2*size2;
             for (m=0; m<size2; m++)
     {   for (n=0; n<size2; n++)
                 {   if ((lk2+size2*j+n)<512)
     {  lx1=lx1+1;
                        y1[lx1]=wcoef[size2*i+m][lk2+(size2*j+n)];
                     }
                     else
                     {  lx1=lx1+1;
                        y1[lx1]=wcoef[size2*i+m][256+(lk2+size2*j+n)%256];
                     }
                     if ((lk2+size2*i+m)<512)
                     {  lx2=lx2+1;
                        y2[lx2]=wcoef[lk2+size2*i+m][size2*j+n];
                     }
                     else
                     {  lx2=lx2+1;
                        y2[lx2]=wcoef[256+(lk2+size2*i+m)%256][size2*j+n];
                     }
                     if ((lk2+size2*i+m)<512 && (lk2+size2*j+n)<512)
                     {  lx3=lx3+1;
                        y3[lx3]=wcoef[lk2+size2*i+m][lk2+size2*j+n];
                     }
                     if ((lk2+size2*i+m)>=512 && (lk2+size2*j+n)<512)
                     {  lx3=lx3+1;
                        y3[lx3]=wcoef[256+(lk2+size2*i+m)%256][lk2+size2*j+n];
                     }
                     if ((lk2+size2*i+m)<512 && (lk2+size2*j+n)>=512)
                     {  lx3=lx3+1;
                        y3[lx3]=wcoef[lk2+size2*i+m][256+(lk2+size2*j+n)%256];
                     }
                     if ((lk2+size2*i+m)>=512 && (lk2+size2*j+n)>=512)
                     {  lx3=lx3+1;
                        y3[lx3]=wcoef[256+(lk2+size2*i+m)%256][256+(lk2+size2*j+n)%256];
                     }
                 }
             }
         }
         num1=lx1+1;
         num2=lx2+1;
         num3=lx3+1;
         for (k=0;k<num1;k++)
             Ak2ij[k]=y1[k];
         for (k=0;k<num2;k++)
             Ak2ij[num1+k]=y2[k];
         for (k=0;k<num3;k++)
             Ak2ij[num1+num2+k]=y3[k];
         nsize1=num1+num2+num3;
         errormin=100000000000.0;
         for (ii=0;ii<sizeb1; ii++)
       {   for (jj=0; jj<sizeb1; jj++)
             {   lk1=lkk1;
                 size1=1;
                 lx1=0;
                 lx2=0;
                 lx3=0;
                 count=0;
                 Ak1ij[lx1]=wcoef[ii][lk1+jj];
                 Ak1ij[num1+lx2]=wcoef[lk1+ii][jj];
```

```
                 Ak1ij[num1+num2+lx3]=wcoef[lk1+ii][lk1+jj];
                 for (k=k1; k<=6; k++)
 {   lk1=lk1+pow(2,count)*sizeb1;
                     count++;
                     size1=2*size1;
                     if (k<6)
                     {  row=size1;
                        col=size1;
                        for (m=0; m<row; m++)
                        {  for (n=0; n<col; n++)
                           {   lx1=lx1+1;
                               lx2=lx2+1;
                               lx3=lx3+1;
                               if ((lk1+size2*jj+n)<512)
                               {  Ak1ij[lx1]=wcoef[size1*ii+m][lk1+size1*jj+n];}
                               else
     {  Ak1ij[lx1]=wcoef[size1*ii+m][256+(lk1+size1*jj+n)%256];}
                               if ((lk1+size2*ii+m)<512)
     {   Ak1ij[lx2+num1]=wcoef[lk1+size1*ii+m][size1*jj+n];}
                               else
     {   Ak1ij[lx2+num1]=wcoef[256+(lk1+size1*ii+m)%256][size1*jj+n];}
                               if ((lk1+size2*ii+m)<512 && (lk1+size2*jj+n)<512)
     {  Ak1ij[num1+num2+lx3]=wcoef[lk1+size1*ii+m][lk1+size1*jj+n];}
                               if ((lk1+size2*ii+m)>=512 && (lk1+size2*jj+n)<512)
     {  Ak1ij[num1+num2+lx3]=wcoef[256+(lk1+size1*ii+m)%256][lk1+size1*jj+n];}
                               if ((lk1+size2*ii+m)<512 && (lk1+size2*jj+n)>=512)
     {  Ak1ij[num1+num2+lx3]=wcoef[(lk1+size1*ii+m)][256+(lk1+size1*jj+n)%256];}
                               if ((lk1+size2*ii+m)>=512 && (lk1+size2*jj+n)>=512)
     {   Ak1ij[num1+num2+lx3]=wcoef[256+(lk1+size1*ii+m)%256][256+(lk1+size1*jj+n)%256];}
                           }
                        }
             }
                 }
                 nsize=nsize1;
         sum1=0.0;
                sum2=0.0;
                for (lx=0; lx<nsize; lx++)
                 {    sum1=sum1+Ak2ij[lx]*Ak1ij[lx];
                      sum2=sum2+Ak1ij[lx]*Ak1ij[lx];
 }
                 alph=sum1/sum2;
                 sum=0.0;
                 for (lx=0; lx<nsize; lx++)
                {   sum=sum+pow((Ak2ij[lx]-alph*Ak1ij[lx]),2); }
                 err=sum/nsize;
                 if (err<=errormin)
                 {  errormin=err;
                    alpha=alph;
                    iindex=ii;
                    jindex=jj;
                 }
             }
         }
   alphah[i][j]=alpha;
  iindexh[i][j]=iindex;
  jindexh[i][j]=jindex;
    }
}
for (i=0;i<lkk2;i++)
    for (j=0;j<lkk2;j++)
          wcoef2[i][j]=wcoef[i][j];
size2=1;
lk1=lkk1;
lk2=lkk2;
count=0;
for (k=k2;k<=8;k++)
{   for (i=0; i<sizeb2;i++)
    {   for (j=0; j<sizeb2;j++)
        {   ii=iindexh[i][j];
```

```
            jj=jindexh[i][j];
            alpha=alphah[i][j];
            for (m=0; m<size2; m++)
            {   for (n=0; n<size2; n++)
                {   if ((lk2+j*size2+n)<512)
                    {   wcoef2[i*size2+m][lk2+j*size2+n]=
                              alpha*wcoef2[ii*size2+m][lk1+jj*size2+n];}
                    if ((lk2+i*size2+m)<512)
                    {   wcoef2[lk2+i*size2+m][j*size2+n]=
                              alpha*wcoef2[lk1+ii*size2+m][jj*size2+n];}
                    if ((lk2+i*size2+m)<512 && (lk2+j*size2+n)<512)
                    {   wcoef2[lk2+i*size2+m][lk2+j*size2+n]=
                              alpha*wcoef2[lk1+ii*size2+m][lk1+jj*size2+n];}
                 }
             }
         }
     }
     count++;
     size2=2*size2;
     lk1=lk1+pow(2,count-1)*sizeb1;
     lk2=lk2+pow(2,count)*sizeb1;
}
sum=0.0;
for (i=0;i<512;i++)
{   for (j=0;j<512;j++)
    {   sum=sum+pow((wcoef[i][j]-wcoef2[i][j]),2);
        fprintf(foutput,"%f\n", wcoef2[i][j]);
    }
}
rmse=sqrt(sum)/512.0;
psnr=20*log10(255.0/rmse);
printf("RMSE = %.5f\n",rmse);
printf("PSNR = %.5f\n",psnr);
fprintf(foutput1, "%d   %.5f   %.5f   %.5f\n",ncount, rmse, psnr);
} /* end of the ncount loop */
}
/*******************.....Uniform quantizer...**************************/
void  uniform(int sizeb2, int level, float low, float up, float alpha[][128],
float q[][128], float *entropy)
{
int i,j,size,num,freq[512];
float sum,x,vector[4096],step;
size=sizeb2;
for (i=0;i<size;i++)
    for (j=0;j<size;j++)
    {     vector[i*size+j]=alpha[i][j];}
step=(up-low)/level;
for (i=0;i<level;i++)
    freq[i]=0;
for (i=0;i<size;i++)
{    for (j=0;j<size;j++)
     {   x=alpha[i][j];
         if (x<=low)
            {q[i][j]=low+step/2.0;
             freq[0]=freq[0]+1;}
         if (x>=up)
            {q[i][j]=up-step/2.0;
             freq[level-1]=freq[level-1]+1;}
         if (x>low && x<up)
         {  num=(int)(level*(x-low)/(up-low));
            freq[num]=freq[num]+1;
            q[i][j]=low+((float)(num)+0.5)*step;
         }
     }
}
sum=0.0;
for (i=0;i<level;i++)
```

```
{   if (freq[i]>0)
    {   sum=sum-(freq[i]/pow(size,2))*(log10(freq[i]/pow(size,2)))/log10(2.0);}
}
printf("entropy=%f\n",sum);
*entropy=sum;
return;
}
/***********************************************************************/
/***********************************************************************/
/*      Quadtree-Based FW-II scheme      */
/***********************************************************************/
/***********************************************************************/
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#define SIZE     512
#define EPSILON  0.0001
#define INFINITY 1000000000
float Uniform(int level, float low, float up, float x);
float Entropy(int size, int level, float low, float up, float vector[512*512]);
float Max_Min(int n, float buffer[][512], float *max,float *min);
main()
{
int iindex[SIZE][SIZE],jindex[SIZE][SIZE],store[SIZE][SIZE];
int kk1,i,j,ii,jj,lk1,lk2,lkk1,lkk2,k1,k2,k,lx1,lx2,nsize,block;
int size1,size2,m,n,lx,l1,l2,iind,jind,k1_min,k1_max,k10,k20,iter;
int repeat,last,count_s,count_w,level_s,level_w;
int level1,level2,level3,level4;
int npoints,num,num_3,num_4,num_5,num_6,image;
float bit_rate,entropy_w,entropy_s,temp,rmse,pixel,sum,sum1,sum2,err,errormin;
float threshold,threshold_f,threshold_0,energy,psnr,low_w,high_w,low_s,high_s,c_ratio;
float wcoef1[SIZE][SIZE],alpha[SIZE][SIZE],wcoef[SIZE][SIZE],Ak2ij[1500],Ak1ij[1500];
float wave[SIZE*SIZE],scaling[SIZE*SIZE],wcoef2[SIZE][SIZE],alphaq[SIZE][SIZE];
float prune,min,max,alph,alpha1,sumh,sumv,sumd,energyh,energyv,energyd,step;
FILE * finput;
FILE * foutput;
finput1=fopen("lenna_wv0_in.dat","r");
foutput1=fopen("rate_fw_qd.out","w");
sum=0.0;
for (i=0;i<SIZE;i++)
{   for (j=0;j<SIZE;j++)
    {   fscanf(finput1, "%f",&pixel);
        wcoef[i][j]=pixel;
    }
}
level_s=65;
low_s=-3;
high_s=3;
level_w=256;
Max_Min(SIZE,wcoef,&max,&min);
low_w=min;
high_w=max;
printf("For the chosen image : Range = [%.2f, %.2f]\n",min,max);
printf("=================================================================\n");
printf("\n");
printf("Enter the NUMBER OF DATA POINTS to be generated >\n");
scanf("%d", &npoints);
printf("Enter the INITIAL ENERGY CRITERION THRESHOLD >\n");
scanf("%f", &threshold_0);
printf("Enter the INCREMENTAL STEP > \n");
scanf("%f", &step);
printf("Enter the minimum energy threshold for pruning\n");
printf("Typically, we chose the pruning threshold = 25\n");
printf("But other values may be chosen >\n");
scanf("%f",&prune);
printf("\n");
printf("=================================================================\n");
printf("\n");
/*******************************************************/
```

```
/* Set the minimum quadtree layer (k1*,k2*)=(3,4)        */
/* Set the maximum quadtree layer (k1*,k2*)=(6,7)        */
/*****************************************************/
k1_min=3;
k1_max=6;
/*****************************************************/
/* Start the encoding process, and repeat as many as the */
/* number of requested data points, starting at the     */
/*  specified initial threshold.                        */
/*****************************************************/
for (repeat=0;repeat<npoints;repeat++)
{   threshold=threshold_0+repeat*step; /* increment the energy threshold */
    printf("Data point number : %d, Energy threshold = %.2f\n",repeat+1,threshold);
    sum=0.0;
    /*  initialization */
    for (i=0;i<SIZE;i++)
    {   for (j=0;j<SIZE;j++)
        {   wcoef1[i][j]=0.0;
            wcoef2[i][j]=0.0;
            iindex[i][j]=0;
            jindex[i][j]=0;
            store[i][j]=2;
            alpha[i][j]=0;
            alphaq[i][j]=0;
        }
    }
    /* setup the sizes initial sizes ...(k1*,k2*)=(3,4)....*/
    k10=3;
    k20=4;
    lkk1=(int)pow(2,k10);
    lkk2=2*(int)pow(2,k10);
/************************************************************/
/*  count_s : the number of fractally encoded subtrees (roots)  */
/*  count_w : the number of stored wavelet coefficients         */
/*  level(i): the number of subtrees encoded on the i_th layer  */
/*  last    : the number of subtrees encoded on the last layer  */
/************************************************************/
    count_s=0;
    count_w=0;
    last=0;
    level1=level2=level3=level4=0;
    num_3=num_4=num_5=num_6=0;
/******************************************************************/
/* store and quantize the initial set of wavelet coefficients, up to */
/* the initial quadtree level (k1*,k2*)=(3,4)                       */
/******************************************************************/
    for (i=0;i<lkk2;i++)
    {   for (j=0;j<lkk2;j++)
        {   wcoef1[i][j]=wcoef[i][j];
            temp=wcoef[i][j];
            wcoef2[i][j]=Uniform(level_w,low_w,high_w,temp);
            store[i][j]=1;
            wave[count_w]=wcoef[i][j];
            count_w++;
        }
    }
/*************************************************************************/
/*  start the quadtree partitioning scheme, starting at the initial level */
/*  (k1*,k2*)=(3,4), until the final level (k1*,k2*)=(6,7)              */
/*************************************************************************/
for (k1=k1_min;k1<=k1_max;k1++)
{   k2=k1+1;
    lkk1=0;
    lkk2=0;
    for (i=0; i<k1; i++)
    {    lkk1=lkk1+pow(2,i);}
    lkk1=lkk1+1;
    lkk2=lkk1+pow(2,k1);
    kk1=0;
    for (i=k2; i<9; i++)
    {    kk1=kk1+pow(2,2*(i-k2));}
    for (i=0; i<pow(2,k2); i++)
    {   for (j=0; j<pow(2,k2); j++)
        { if (store[i][lkk2+j]==2)
          { lk2=lkk2;
            lx2=0;
            size2=1;
            Ak2ij[lx2]=wcoef[i][lk2+j];
            Ak2ij[lx2+kk1]=wcoef[i+lk2][j];
            Ak2ij[lx2+2*kk1]=wcoef[i+lk2][j+lk2];
            for (k=k2;k<=7;k++)
            {   lk2=lk2+pow(2,k);
                size2=2*size2;
                for (m=0; m<size2; m++)
                {   for (n=0; n<size2; n++)
                    {   lx2=lx2+1;
                        Ak2ij[lx2]=wcoef[size2*i+m][lk2+size2*j+n];
                        Ak2ij[lx2+kk1]=wcoef[lk2+size2*i+m][size2*j+n];
                        Ak2ij[lx2+2*kk1]=wcoef[lk2+size2*i+m][lk2+size2*j+n];
                     }
                 }
            }
            nsize=3*(lx2+1);
            errormin=INFINITY;
/*********************************************************************/
/*  Compute the energy of each subtree (h,v,d) and set the energy to the */
/*  maximum of the three values                                      */
/*********************************************************************/
            sumh=0.0;
            sumv=0.0;
            sumd=0.0;
            for (m=0;m<nsize/3;m++)
            {   sumh=sumh+pow(Ak2ij[m],2);}
            for (m=nsize/3;m<2*nsize/3;m++)
            {   sumv=sumv+pow(Ak2ij[m],2);}
            for (m=2*nsize/3;m<nsize;m++)
            {   sumd=sumd+pow(Ak2ij[m],2);}
            energyh=sqrt(sumh);
            energyv=sqrt(sumv);
            energyd=sqrt(sumd);
            energy=energyh;
            if (energyv>energy)
               energy=energyv;
            if (energyd>energy)
               energy=energyd;
/*********************************************************************/
/*  If the energy is less than the threshold, then encode the subtree    */
/*  and store the fractal code                                       */
/*********************************************************************/
            if (energy<threshold || k1==k1_max)
            {  store[i][lkk2+j]=0;
               store[lkk2+i][j]=0;
               store[lkk2+i][lkk2+j]=0;
               size2=1;
               lk2=lkk2;
               for (k=k2;k<=7;k++)
               {   lk2=lk2+pow(2,k);
                   size2=2*size2;
                   for (m=0; m<size2; m++)
                   {   for (n=0; n<size2; n++)
                       {   store[size2*i+m][lk2+size2*j+n]=3;
                           store[lk2+size2*i+m][size2*j+n]=3;
                           store[lk2+size2*i+m][lk2+size2*j+n]=3;
                       }
                   }
               }
               for (ii=0;ii<pow(2,k1); ii++)
             {   for (jj=0; jj<pow(2,k1); jj++)
                   {   lk1=lkk1;
                       size1=1;
                       lx1=0;
                       Ak1ij[lx1]=wcoef[ii][lk1+jj];
                       Ak1ij[lx1+kk1]=wcoef[lk1+ii][jj];
                       Ak1ij[lx1+2*kk1]=wcoef[lk1+ii][lk1+jj];
                       for (k=k1; k<7; k++)
                       {   lk1=lk1+pow(2,k);
                           size1=2*size1;
                           for (m=0; m<size1; m++)
                           {  for (n=0; n<size1; n++)
                              {   lx1=lx1+1;
                                  Ak1ij[lx1]= wcoef[size1*ii+m][lk1+size1*jj+n];
                                  Ak1ij[lx1+kk1]=wcoef[lk1+size1*ii+m][size1*jj+n];
                                  Ak1ij[lx1+2*kk1]=wcoef[lk1+size1*ii+m][lk1+size1*jj+n];
                              }
                           }
                       }
                       sum1=0.0;
                       sum2=0.0;
                       for (lx=0; lx<nsize; lx++)
                       {    sum1=sum1+Ak2ij[lx]*Ak1ij[lx];
                            sum2=sum2+Ak1ij[lx]*Ak1ij[lx];
                       }
                       alph=Uniform(level_s,low_s,high_s,sum1/sum2);
                       sum=0.0;
                       for (lx=0; lx<nsize; lx++)
                       {   sum=sum+pow((Ak2ij[lx]-alph*Ak1ij[lx]),2); }
                       err=sum/nsize;
                       if (err<=errormin)
                       {  errormin=err;
                          alpha1=alph;
                          iind=ii;
                          jind=jj;
                       }
                   }
               }
/**************************************************************/
/*   If the scaling coefficient is too small (quantized to zero)    */
/*   or the energy of the subtree is insignificant, then prune the  */
/*   subtree, and keep track of which subtrees have been pruned     */
/*   and where                                                      */
/**************************************************************/
               if (fabs(alpha1)<EPSILON || energy<=prune)
               {  if (k1==3)
                     num_3++;
                  if (k1==4)
                     num_4++;
                  if (k1==5)
                     num_5++;
                  if (k1==6)
                     num_6++;
                  alpha1=0.0;
               }
               alpha[i][lkk2+j]=alpha1;
               alphaq[i][lkk2+j]=alpha1;
               iindex[i][lkk2+j]=iind;
               jindex[i][lkk2+j]=jind;
               scaling[count_s]=alpha1;
               count_s++;
               if (k1==k1_max)
               {   last++;}
               if (k1==3)
                  level1++;
               if (k2==4)
                  level2++;
               if (k1==5)
                  level3++;
               if (k1==6)
                  level4++;
            }
/*******************************************************************/
/*  If the energy is higher than the threshold, then store the node  */
/*  and replace the subtree by four subtrees rooted at the four      */
/*  children of the node                                             */
/*******************************************************************/
            else
            {   store[i][lkk2+j]==1;
                wcoef1[i][lkk2+j]=wcoef[i][lkk2+j];
                temp=wcoef[i][lkk2+j];
                wcoef2[i][lkk2+j]=Uniform(level_w,low_w,high_w,temp);
                wave[count_w]=wcoef[i][lkk2+j];
                count_w++;
                wcoef1[lkk2+i][j]=wcoef[lkk2+i][j];
                temp=wcoef[lkk2+i][j];
                wcoef2[lkk2+i][j]=Uniform(level_w,low_w,high_w,temp);
                wave[count_w]=wcoef[lkk2+i][j];
                count_w++;
                wcoef1[lkk2+i][lkk2+j]=wcoef[lkk2+i][lkk2+j];
                temp=wcoef[lkk2+i][lkk2+j];
                wcoef2[lkk2+i][lkk2+j]=Uniform(level_w,low_w,high_w,temp);
                wave[count_w]=wcoef[lkk2+i][lkk2+j];
                count_w++;
            }
         }
      }
   }
}
/*********************************************************/
/*  The ENCODING process is DONE                         */
/*********************************************************/
/**********************************************************/
/* Compute and display the entropies of the scaling and the */
/* stored wavelet coefficients                              */
/**********************************************************/
entropy_s=Entropy(count_s,level_s,low_s,high_s,scaling);
entropy_w=Entropy(count_w,level_w,low_w,high_w,wave);
printf("Entropies:\n");
printf("The scaling coefficients entropy = %f\n", entropy_s);
printf("The wavelet coefficients entropy = %f\n", entropy_w);
printf("\n");
printf("The number of scaling coefficients = %d\n", count_s);
printf("The number of wavelet coefficients = %d\n", count_w);
printf("\n");
/*********************************************************/
/*  Start the DECODING process                           */
/*********************************************************/
for (k1=k1_min;k1<=k1_max;k1++)
{   k2=k1+1;
    lkk1=0;
    lkk2=0;
    for (i=0; i<k1; i++)
    {    lkk1=lkk1+pow(2,i);}
    lkk1=lkk1+1;
    lkk2=lkk1+pow(2,k1);
    size2=1;
    lk1=lkk1;
    lk2=lkk2;
    for (k=k2;k<=8;k++)
    {   for (i=0; i<pow(2,k2);i++)
        {   for (j=0; j<pow(2,k2);j++)
            {  if (store[i][lkk2+j]==0)
               {   ii=iindex[i][lkk2+j];
                   jj=jindex[i][lkk2+j];
                   alph=alphaq[i][lkk2+j];
                   for (m=0; m<size2; m++)
```

```
                    {   for (n=0; n<size2; n++)
                        {   wcoef2[i*size2+m][lk2+j*size2+n]=
                              alph*wcoef2[ii*size2+m][lk1+jj*size2+n];
                            wcoef2[lk2+i*size2+m][j*size2+n]=
                              alph*wcoef2[lk1+ii*size2+m][jj*size2+n];
                            wcoef2[lk2+i*size2+m][lk2+j*size2+n]=
                              alph*wcoef2[lk1+ii*size2+m][lk1+jj*size2+n];
                        }
                    }
                }
            }
        }
        size2=2*size2;
        lk1=lk1+pow(2,k-1);
        lk2=lk2+pow(2,k);
    }
}
/****************************************************/
/* Compute the RMSE and the PSNR of the decoded         */
/* wavelet tree.                                        */
/****************************************************/
sum=0.0;
for (i=0;i<SIZE;i++)
{   for (j=0;j<SIZE;j++)
    {    sum=sum+pow((wcoef[i][j]-wcoef2[i][j]),2);}
}
rmse=sqrt(sum)/SIZE;
psnr=20*log10(255.0/rmse);
/****************************************************/
/*  Count how many subtrees have been pruned           */
/*  these need to be removed from the fractal code     */
/*  depending on which layer such subtrees were pruned */
/****************************************************/
num=num_3+num_4+num_5+num_6;
sum=2*(3*num_3+4*num_4+5*num_5+6*num_6)+num*entropy_s;
c_ratio=pow(2,21)/((entropy_w+1)*count_w-pow(2,(k1_min+1))*pow(2,(k1_min+1))+
count_s*entropy_s+2*(level1*3+level2*4+level3*5+level4*6)+count_s-last-sum+num);
bit_rate=8.0/c_ratio;  /* Bit-Rate */
printf("\n");
printf("The number of zero-scaling coefs = %d\n",num);
printf("RMSE = %.5f\n",rmse);
printf("PSNR = %.5f\n",psnr);
printf("Compression Ratio = %.5f\n",c_ratio);
printf("Bit Rate = %.5f\n",bit_rate);
printf("\n")
printf("===============================================================\n");
threshold,rmse,psnr,c_ratio,bit_rate);
}  /* END of the repeat loop */
fclose(foutput1);
fclose(finput1);
printf("The program finished successfully\n");
}
/*_________________________ END OF MAIN()_______________________________*/
/*******************************************************************/
/*  This is a Uniform quatizer that quatizes the scaling coefficients   */
/*  into the specified number of levels on the given range.            */
/*******************************************************************/
float  Uniform(int level,float low,float up,float x)
{
int num;
float q,step;
step=(up-low)/level;
if (x<=low)
{  q=low+step/2.0;}
if (x>=up)
   {q=up-step/2.0;}
if (x>low && x<up)
```

```
{  num=(int)(level*(x-low)/(up-low));
   q=low+((float)(num)+0.5)*step;
}
return(q);
}
/**********************************************************************/
/*  This is function computes the entropy of a set of quatized values   */
/*  to be used for compression ratio...                                 */
/**********************************************************************/
float Entropy(int size,int level,float low,float up,float vector[512*512])
{
int i,j,num,freq[1024];
float sum,x,step;
step=(up-low)/level;
for (i=0;i<level;i++)
    freq[i]=0;
for (i=0;i<size;i++)
{   x=vector[i];
    if (x<=low)
       {freq[0]=freq[0]+1;}
    if (x>=up)
       {freq[level-1]=freq[level-1]+1;}
    if (x>low && x<up)
    {  num=(int)(level*(x-low)/(up-low));
       freq[num]=freq[num]+1;
    }
}
sum=0.0;
for (i=0;i<level;i++)
{   if (freq[i]>0)
    {  sum=sum-((float)(freq[i])/(size))*(log10((float)(freq[i])/(size)))/log10(2.0);}
}
return(sum);
}
/******************************************************************/
/*  This function computes the max and minimum values in an array.  */
/******************************************************************/
float Max_Min(int n, float buffer[][512],float *max,float *min)
{
int i,j;
float mx,mn;
mx=buffer[0][0];
mn=buffer[0][0];
for (i=1;i<n;i++)
{   for (j=1;j<n;j++)
    {   if (buffer[i][j]>=mx)
        {  mx=buffer[i][j]; }
        if (buffer[i][j]<=mn)
        {  mn=buffer[i][j]; }
    }
}
*max=mx;
*min=mn;
return;
}
%***************************************************************************
%***************************************************************************
%  VisuShrink wavelet threshold scheme
%       Performs hard and soft thresholding using the universal threshold.
%***************************************************************************
%***************************************************************************
clear all
load lenna0.dat;
for i = 1:512
    lena0(i,1:512)=lenna0(512*(i-1)+1:i*512)';
end
noisy_im=lena0+25*randn(512,512);
SIZE=512;
noise_std=25;
```

```
num_levels=9;
max_levels=9;
dwtmode('per');
X=noisy_im;
for i=1:9
    ii=2^i;
    jj=512/ii;
    [CA,CH,CV,CD] = dwt2(X,'sym8');
    coef(1:512/ii,1:512/ii)=CA;
    coef(512/ii+1:2*512/ii,1:512/ii)=CH;
    coef(1:512/ii,512/ii+1:2*512/ii)=CV;
    coef(512/ii+1:2*512/ii,512/ii+1:2*512/ii)=CD;
    clear X;
    X=CA;
    clear CA;
    clear CH;
    clear CV;
    clear CD;
end
thr=sqrt(2*log(SIZE*SIZE))*noise_std;
for i=1:512
    for j=1:512
        if (abs(coef(i,j))>thr)
           coef_hdn(i,j)=coef(i,j);
        else
           coef_hdn(i,j)=0.0;
        end
        if (coef(i,j)>thr)
           coef_sdn(i,j)=coef(i,j)-thr;
        end
        if (coef(i,j)<-thr)
            coef_sdn(i,j)=coef(i,j)+thr;
        end
        if (abs(coef(i,j))<thr)
            coef_sdn(i,j)=0.0;
        end
    end
end
coef=coef_hdn;
for level=1:num_levels
    jj=2^(max_levels-num_levels+level-1);
    CA=coef(1:jj,1:jj);
    CH=coef(jj+1:2*jj,1:jj);
    CV=coef(1:jj,jj+1:2*jj);
    CD=coef(jj+1:2*jj,jj+1:2*jj);
    X=idwt2(CA,CH,CV,CD,'sym8');
    coef(1:2*jj,1:2*jj)=X;
    if (level<num_levels)
       clear X;
    else
       X0=X;
    end
    clear CA;
    clear CH;
    clear CV;
    clear CD;
end
clear X;
X_hd=X0;
coef=coef_sdn;
for level=1:num_levels
    jj=2^(max_levels-num_levels+level-1);
    CA=coef(1:jj,1:jj);
    CH=coef(jj+1:2*jj,1:jj);
    CV=coef(1:jj,jj+1:2*jj);
    CD=coef(jj+1:2*jj,jj+1:2*jj);
    X=idwt2(CA,CH,CV,CD,'sym8');
    coef(1:2*jj,1:2*jj)=X;
    if (level<num_levels)
       clear X;
    else
      X0=X;
    end
    clear CA;
    clear CH;
    clear CV;
    clear CD;
end
X_sd=X0;
lena_hd=X_hd;
lena_sd=X_sd;
D=lena_hd-lena0;
rmse=sqrt(sum(sum(D.^2)))/512;
psnr=20*log10(255.0/rmse);
fprintf(1,'===================================\n');
fprintf(1,'VISUSHRINK HARD THRESHOLDING\n');
fprintf(1,'The RMSE = %d\n', rmse);
fprintf(1,'The PSNR = %d\n', psnr);
fprintf(1,'\n');
fprintf(1,'===================================\n');
D=lena_sd-lena0;
rmse=sqrt(sum(sum(D.^2)))/512;
psnr=20*log10(255.0/rmse);
fprintf(1,'===================================\n');
fprintf(1,'VISUSHRINK SOFT THRESHOLDING\n');
fprintf(1,'The RMSE = %d\n', rmse);
fprintf(1,'The PSNR = %d\n', psnr);
fprintf(1,'\n');
fprintf(1,'===================================\n');
colormap(gray(255))
subplot(1,2,1), image(lena_hd);
axis off
axis equal
subplot(1,2,2), image(lena_sd);
axis off
axis equal
colormap(gray(255))
%*************************************************************************
%*************************************************************************
%   LevelShrink Wavelet thresholding scheme
%*************************************************************************
%*************************************************************************
% clear all variables...
clear all
load lenna0.dat;
for i = 1:512
    lena0(i,1:512)=lenna0(512*(i-1)+1:i*512)';
end
noisy_im=lena0+25*randn(512,512);
%*******************************
SIZE=512;
noise_std=25;
num_levels=9;
max_levels=9;
dwtmode('per');
% setup the sizes
%*******************************
X=noisy_im;
for level=1:num_levels
    ii=2^level;
    jj=SIZE/ii;
    [CA,CH,CV,CD]=dwt2(X,'sym8');
    coef(1:SIZE/ii,1:SIZE/ii)=CA;
    coef_hdn(1:SIZE/ii,1:SIZE/ii)=CA;
    coef_sdn(1:SIZE/ii,1:SIZE/ii)=CA;
    thr=noise_std*sqrt(2*log(SIZE*SIZE))*2^((num_levels-level+1-num_levels)/2.0);
    thr_sd=thr;
    thr_hd=thr;
    % The horizontal subband
    coef(SIZE/ii+1:2*SIZE/ii,1:SIZE/ii)=CH;
    for i=1:jj
        for j=1:jj
```

```
            if (abs(CH(i,j))>thr_hd)
               CH_hdn(i,j)=CH(i,j);
            else
               CH_hdn(i,j)=0.0;
            end
            if (CH(i,j)>thr_sd)
               CH_sdn(i,j)=CH(i,j)-thr_sd;
            end
            if (CH(i,j)<-thr_sd)
               CH_sdn(i,j)=CH(i,j)+thr_sd;
            end
            if (abs(CH(i,j))<thr_sd)
               CH_sdn(i,j)=0.0;
            end
        end
    end
    coef_hdn(SIZE/ii+1:2*SIZE/ii,1:SIZE/ii)=CH_hdn;
    coef_sdn(SIZE/ii+1:2*SIZE/ii,1:SIZE/ii)=CH_sdn;
    % The vertical subband
    coef(1:SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CV;
    for i=1:jj
        for j=1:jj
            if (abs(CV(i,j))>thr_hd)
               CV_hdn(i,j)=CV(i,j);
            else
               CV_hdn(i,j)=0.0;
            end
            if (CV(i,j)>thr_sd)
               CV_sdn(i,j)=CV(i,j)-thr_sd;
            end
            if (CV(i,j)<-thr_sd)
               CV_sdn(i,j)=CV(i,j)+thr_sd;
            end
            if (abs(CV(i,j))<thr_sd)
               CV_sdn(i,j)=0.0;
            end
        end
    end
    coef_hdn(1:SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CV_hdn;
    coef_sdn(1:SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CV_sdn;
    % The diagonal subband
    coef(SIZE/ii+1:2*SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CD;
    for i=1:jj
        for j=1:jj
            if (abs(CD(i,j))>thr_hd)
               CD_hdn(i,j)=CD(i,j);
            else
               CD_hdn(i,j)=0.0;
            end
            if (CD(i,j)>thr_sd)
               CD_sdn(i,j)=CD(i,j)-thr_sd;
            end
            if (CD(i,j)<-thr_sd)
               CD_sdn(i,j)=CD(i,j)+thr_sd;
            end
            if (abs(CD(i,j))<thr_sd)
               CD_sdn(i,j)=0.0;
            end
         end
    end
    coef_hdn(SIZE/ii+1:2*SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CD_hdn;
    coef_sdn(SIZE/ii+1:2*SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CD_sdn;
    clear X;
    X=CA;
    clear CA;
    clear CH;
    clear CH_hdn;
    clear CH_sdn;
    clear CV;
    clear CV_hdn
    clear CV_sdn;
    clear CD;
    clear CD_hdn;
    clear CD_sdn;
    clear Y;
end
coef=coef_hdn;
for level=1:num_levels
    jj=2^(max_levels-num_levels+level-1);
    CA=coef(1:jj,1:jj);
    CH=coef(jj+1:2*jj,1:jj);
    CV=coef(1:jj,jj+1:2*jj);
    CD=coef(jj+1:2*jj,jj+1:2*jj);
    X=idwt2(CA,CH,CV,CD,'sym8');
    coef(1:2*jj,1:2*jj)=X;
    if (level<num_levels)
       clear X;
    else
       X0=X;
    end
    clear CA;
    clear CH;
    clear CV;
    clear CD;
end
X_hd=X0;
coef=coef_sdn;
for level=1:num_levels
     jj=2^(max_levels-num_levels+level-1);
     CA=coef(1:jj,1:jj);
     CH=coef(jj+1:2*jj,1:jj);
     CV=coef(1:jj,jj+1:2*jj);
     CD=coef(jj+1:2*jj,jj+1:2*jj);
     X=idwt2(CA,CH,CV,CD,'sym8');
     coef(1:2*jj,1:2*jj)=X;
     if (level<num_levels)
        clear X;
     else
        X0=X;
     end
     clear CA;
     clear CH;
     clear CV;
     clear CD;
end
X_sd=X0;
lena_hd=X_hd;
lena_sd=X_sd;
D=lena_hd-lena0;
rmse=sqrt(sum(sum(D.^2)))/512;
psnr=20*log10(255.0/rmse);
fprintf(1,'====================================\n');
fprintf(1,'LevelShrink Hard thresholding \n');
fprintf(1,'The RMSE = %d\n', rmse);
fprintf(1,'The PSNR = %d\n', psnr);
fprintf(1,'\n');
fprintf(1,'====================================\n');
D=lena_sd-lena0;
rmse=sqrt(sum(sum(D.^2)))/512;
psnr=20*log10(255.0/rmse);
fprintf(1,'====================================\n');
fprintf(1,'LevelShrink Soft thresholding \n');
fprintf(1,'The RMSE = %d\n', rmse);
fprintf(1,'The PSNR = %d\n', psnr);
fprintf(1,'\n');
fprintf(1,'====================================\n');
colormap(gray(255))
subplot(1,2,1), image(lena_hd);
axis off
axis equal
subplot(1,2,2), image(lena_sd);
axis off
axis equal
```

```
colormap(gray(255))
fprintf(1,'The program completed successfully\n');
%****************************************************************************
%****************************************************************************
%  SureShrink wavelet threshold scheme
%****************************************************************************
%****************************************************************************
% clear all variables...
clear all
load lenna0.dat;
for i = 1:512
    lena0(i,1:512)=lenna0(512*(i-1)+1:i*512)';
end
noisy_im=lena0+25*randn(512,512);
%*******************************
SIZE=512;
noise_std=25;
num_levels=4;
max_levels=9;
dwtmode('per');
% setup the sizes
%*******************************
X=noisy_im;
for level=1:num_levels
    ii=2^level;
    jj=SIZE/ii;
    N=jj^2;
    [CA,CH,CV,CD]=dwt2(X,'sym8');
    coef(1:SIZE/ii,1:SIZE/ii)=CA;
    coef_hdn(1:SIZE/ii,1:SIZE/ii)=CA;
    coef_sdn(1:SIZE/ii,1:SIZE/ii)=CA;
    % The horizontal subband
    coef(SIZE/ii+1:2*SIZE/ii,1:SIZE/ii)=CH;
    for j=1:jj
        Y((j-1)*jj+1:j*jj)=CH(j,1:jj)';
    end
    Y=sort(abs(Y));
    Y=Y(jj*jj:-1:1);
    sure_min=1000000000000000;
    ssum=Y*Y';
    for k=1:N
        t=Y(k);
        s=Y(k:N)*(Y(k:N))';
        sure=s-(N-k)*noise_std^2+k*(noise_std^2+t^2);
        if (sure<sure_min)
           tmin=t;
           sure_min=sure;
        end
    end
    epsilon=noise_std^2*sqrt(N)*(log(N))^(3.0/2);
    if (ssum-N*noise_std^2<=epsilon)
       thr_hdn=noise_std*sqrt(2*log(N));
       thr_sdn=noise_std*sqrt(2*log(N));
    else
       thr_sdn=tmin;
       thr_hdn=2*tmin;
    end
    CH_hdn=zeros(jj,jj);
    CH_sdn=zeros(jj,jj);
    for i=1:jj
        for j=1:jj
            if (abs(CH(i,j))>thr_hdn)
               CH_hdn(i,j)=CH(i,j);
            else
               CH_hdn(i,j)=0.0;
            end
            if (CH(i,j)>thr_sdn)
               CH_sdn(i,j)=CH(i,j)-thr_sdn;
            end
            if (CH(i,j)<-thr_sdn)
               CH_sdn(i,j)=CH(i,j)+thr_sdn;
            end
            if (abs(CH(i,j))<thr_sdn)
               CH_sdn(i,j)=0.0;
            end
        end
    end
    coef_hdn(SIZE/ii+1:2*SIZE/ii,1:SIZE/ii)=CH_hdn;
    coef_sdn(SIZE/ii+1:2*SIZE/ii,1:SIZE/ii)=CH_sdn;
    % The vertical subband
    coef(1:SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CV;
    for j=1:jj
        Y((j-1)*jj+1:j*jj)=CV(j,1:jj)';
    end
    Y=sort(abs(Y));
    Y=Y(jj*jj:-1:1);
    sure_min=10000000000000000;
    s=Y*Y';
    for k=1:N
        t=Y(k);
        s=Y(k:N)*(Y(k:N))';
        sure=s-(N-k)*noise_std^2+k*(noise_std^2+t^2);
        if (sure<sure_min)
           tmin=t;
           sure_min=sure;
        end
    end
    epsilon=noise_std^2*sqrt(N)*(log(N))^(3.0/2);
    if (ssum-N*noise_std^2<=epsilon)
       thr_hdn=noise_std*sqrt(2*log(N));
       thr_sdn=noise_std*sqrt(2*log(N));
    else
       thr_sdn=tmin;
       thr_hdn=2*tmin;
    end
    CV_hdn=zeros(jj,jj);
    CV_sdn=zeros(jj,jj);
    for i=1:jj
        for j=1:jj
            if (abs(CV(i,j))>thr_hdn)
               CV_hdn(i,j)=CV(i,j);
            else
               CV_hdn(i,j)=0.0;
            end
            if (CV(i,j)>thr_sdn)
               CV_sdn(i,j)=CV(i,j)-thr_sdn;
            end
            if (CV(i,j)<-thr_sdn)
               CV_sdn(i,j)=CV(i,j)+thr_sdn;
            end
            if (abs(CV(i,j))<thr_sdn)
               CV_sdn(i,j)=0.0;
            end
        end
    end
    coef_hdn(1:SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CV_hdn;
    coef_sdn(1:SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CV_sdn;
    % The diagonal subband
    coef(SIZE/ii+1:2*SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CD;
    for j=1:jj
       Y((j-1)*jj+1:j*jj)=CD(j,1:jj)';
    end
    Y=sort(abs(Y));
    Y=Y(jj*jj:-1:1);
    sure_min=100000000000000000;
    ssum=Y*Y';
    for k=1:N
        t=Y(k);
        s=Y(k:N)*(Y(k:N))';
        sure=s-(N-k)*noise_std^2+k*(noise_std^2+t^2);
        if (sure<sure_min)
           tmin=t;
           sure_min=sure;
        end
    end
    epsilon=noise_std^2*sqrt(N)*(log(N))^(3.0/2);
    if (ssum-N*noise_std^2<=epsilon)
       thr_hdn=noise_std*sqrt(2*log(N));
       thr_sdn=noise_std*sqrt(2*log(N));
    else
       thr_sdn=tmin;
       thr_hdn=2*tmin;
    end
    CD_hdn=zeros(jj,jj);
    CD_sdn=zeros(jj,jj);
    for i=1:jj
        for j=1:jj
            if (abs(CD(i,j))>thr_hdn)
               CD_hdn(i,j)=CD(i,j);
            else
               CD_hdn(i,j)=0.0;
            end
            if (CD(i,j)>thr_sdn)
               CD_sdn(i,j)=CD(i,j)-thr_sdn;
            end
            if (CV(i,j)<-thr_sdn)
               CD_sdn(i,j)=CD(i,j)+thr_sdn;
            end
            if (abs(CD(i,j))<thr_sdn)
               CD_sdn(i,j)=0.0;
            end
        end
    end
    coef_hdn(SIZE/ii+1:2*SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CD_hdn;
    coef_sdn(SIZE/ii+1:2*SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CD_sdn;
    clear X;
    X=CA;
    clear CA;
    clear CH;
    clear CH_hdn;
    clear CH_sdn;
    clear CV;
    clear CV_hdn
    clear CV_sdn;
    clear CD;
    clear CD_hdn;
    clear CD_sdn;
    clear Y;
    clear x;
    clear sure_h;
    clear sure_v;
    clear sure_d;
end
coef=coef_hdn;
for level=1:num_levels
    jj=2^(max_levels-num_levels+level-1);
    CA=coef(1:jj,1:jj);
    CH=coef(jj+1:2*jj,1:jj);
    CV=coef(1:jj,jj+1:2*jj);
    CD=coef(jj+1:2*jj,jj+1:2*jj);
    X=idwt2(CA,CH,CV,CD,'sym8');
    coef(1:2*jj,1:2*jj)=X;
    if (level<num_levels)
       clear X;
    else
       X0=X;
    end
    clear CA;
    clear CH;
    clear CV;
    clear CD;
end
X_hd=X0;
coef=coef_sdn;
for level=1:num_levels
    jj=2^(max_levels-num_levels+level-1);
    CA=coef(1:jj,1:jj);
    CH=coef(jj+1:2*jj,1:jj);
    CV=coef(1:jj,jj+1:2*jj);
    CD=coef(jj+1:2*jj,jj+1:2*jj);
    X=idwt2(CA,CH,CV,CD,'sym8');
    coef(1:2*jj,1:2*jj)=X;
    if (level<num_levels)
       clear X;
    else
       X0=X;
   end
   clear CA;
   clear CH;
   clear CV;
   clear CD;
end
X_sd=X0;
lena_hd=X_hd;
lena_sd=X_sd;
D=lena_hd-lena0;
rmse=sqrt(sum(sum(D.^2)))/512;
psnr=20*log10(255.0/rmse);
fprintf(1,'===================================\n');
fprintf(1,'\n');
fprintf(1,'SureShrink - Hard thresholding...\n');
fprintf(1,'The RMSE = %d\n', rmse);
fprintf(1,'The PSNR = %d\n', psnr);
fprintf(1,'\n');
fprintf(1,'===================================\n');
D=lena_sd-lena0;
rmse=sqrt(sum(sum(D.^2)))/512;
psnr=20*log10(255.0/rmse);
fprintf(1,'===================================\n');
fprintf(1,'\n');
fprintf(1,'SureShrink - Soft thresholding...\n');
fprintf(1,'The RMSE = %d\n', rmse);
fprintf(1,'The PSNR = %d\n', psnr);
fprintf(1,'\n');
fprintf(1,'===================================\n');
colormap(gray(255))
subplot(1,2,1), image(lena_hd);
axis off
axis equal
subplot(1,2,2), image(lena_sd);
axis off
axis equal
colormap(gray(255))
fprintf(1,'The program completed successfully\n');
%*****************************************************************************
%*****************************************************************************
%   BayesShrink wavelet thresholding scheme....
%*****************************************************************************
%*****************************************************************************
% clear all variables...
clear all
noise_std=25;
load sanfran512.dat;
lenna0=sanfran512;
for i = 1:512
    lena0(i,1:512)=lenna0(512*(i-1)+1:i*512)';
end
noisy_im=lena0+noise_std*randn(512,512);
%******************************
SIZE=512;
num_levels=4;
max_levels=9;
dwtmode('per');
% setup the sizes
%******************************
X=noisy_im;
for level=1:num_levels
    ii=2^level;
```

```
jj=SIZE/ii;
N=jj^2;
[CA,CH,CV,CD]=dwt2(X,'sym8');
coef(1:SIZE/ii,1:SIZE/ii)=CA;
coef_hdn(1:SIZE/ii,1:SIZE/ii)=CA;
coef_sdn(1:SIZE/ii,1:SIZE/ii)=CA;
% The horizontal subband
coef(SIZE/ii+1:2*SIZE/ii,1:SIZE/ii)=CH;
for j=1:jj
    Y((j-1)*jj+1:j*jj)=CH(j,1:jj)';
end
var_y=var(Y);
sigma_x=sqrt(max((var_y-noise_std^2),0));
if (sigma_x==0)
   thr_sdn=max(abs(Y));
else
   thr_sdn=noise_std^2/sigma_x;
end
thr_hdn=2*thr_sdn;
CH_hdn=zeros(jj,jj);
CH_sdn=zeros(jj,jj);
for i=1:jj
    for j=1:jj
        if (abs(CH(i,j))>thr_hdn)
            CH_hdn(i,j)=CH(i,j);
        else
            CH_hdn(i,j)=0.0;
        end
        if (CH(i,j)>thr_sdn)
           CH_sdn(i,j)=CH(i,j)-thr_sdn;
        end
        if (CH(i,j)<-thr_sdn)
           CH_sdn(i,j)=CH(i,j)+thr_sdn;
        end
        if (abs(CH(i,j))<thr_sdn)
           CH_sdn(i,j)=0.0;
        end
    end
end
coef_hdn(SIZE/ii+1:2*SIZE/ii,1:SIZE/ii)=CH_hdn;
coef_sdn(SIZE/ii+1:2*SIZE/ii,1:SIZE/ii)=CH_sdn;
%  The vertical subband
coef(1:SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CV;
for j=1:jj
    Y((j-1)*jj+1:j*jj)=CV(j,1:jj)';
end
var_y=var(Y);
sigma_x=sqrt(max((var_y-noise_std^2),0));
if (sigma_x==0)
   thr_sdn=max(abs(Y));
else
   thr_sdn=noise_std^2/sigma_x;
end
thr_hdn=2*thr_sdn;
CV_hdn=zeros(jj,jj);
CV_sdn=zeros(jj,jj);
for i=1:jj
    for j=1:jj
        if (abs(CV(i,j))>thr_hdn)
            CV_hdn(i,j)=CV(i,j);
        else
            CV_hdn(i,j)=0.0;
        end
        if (CV(i,j)>thr_sdn)
           CV_sdn(i,j)=CV(i,j)-thr_sdn;
        end
        if (CV(i,j)<-thr_sdn)
           CV_sdn(i,j)=CV(i,j)+thr_sdn;
        end
        if (abs(CV(i,j))<thr_sdn)
```

```
                CV_sdn(i,j)=0.0;
             end
         end
     end
     coef_hdn(1:SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CV_hdn;
     coef_sdn(1:SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CV_sdn;
     % The diagonal subband
      coef(SIZE/ii+1:2*SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CD;
     for j=1:jj
         Y((j-1)*jj+1:j*jj)=CD(j,1:jj)';
     end
     var_y=var(Y);
     sigma_x=sqrt(max((var_y-noise_std^2),0));
     if (sigma_x==0)
        thr_sdn=max(abs(Y));
     else
        thr_sdn=noise_std^2/sigma_x;
     end
     thr_hdn=2*thr_sdn;
     CD_hdn=zeros(jj,jj);
     CD_sdn=zeros(jj,jj);
     for i=1:jj
         for j=1:jj
             if (abs(CD(i,j))>thr_hdn)
                CD_hdn(i,j)=CD(i,j);
             else
                CD_hdn(i,j)=0.0;
             end
             if (CD(i,j)>thr_sdn)
                CD_sdn(i,j)=CD(i,j)-thr_sdn;
             end
             if (CV(i,j)<-thr_sdn)
                CD_sdn(i,j)=CD(i,j)+thr_sdn;
             end
             if (abs(CD(i,j))<thr_sdn)
                CD_sdn(i,j)=0.0;
             end
         end
     end
     coef_hdn(SIZE/ii+1:2*SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CD_hdn;
     coef_sdn(SIZE/ii+1:2*SIZE/ii,SIZE/ii+1:2*SIZE/ii)=CD_sdn;
     clear X;
     X=CA;
     clear CA;
     clear CH;
     clear CH_hdn;
     clear CH_sdn;
     clear CV;
     clear CV_hdn
     clear CV_sdn;
     clear CD;
     clear CD_hdn;
     clear CD_sdn;
     clear Y;
end
coef=coef_hdn;
for level=1:num_levels
     jj=2^(max_levels-num_levels+level-1);
     CA=coef(1:jj,1:jj);
     CH=coef(jj+1:2*jj,1:jj);
     CV=coef(1:jj,jj+1:2*jj);
     CD=coef(jj+1:2*jj,jj+1:2*jj);
     X=idwt2(CA,CH,CV,CD,'sym8');
     coef(1:2*jj,1:2*jj)=X;
     if (level<num_levels)
        clear X;
     else
        X0=X;
     end
     clear CA;
     clear CH;
     clear CV;
     clear CD;
```

```
end
X_hd=X0;
coef=coef_sdn;
for level=1:num_levels
    jj=2^(max_levels-num_levels+level-1);
    CA=coef(1:jj,1:jj);
    CH=coef(jj+1:2*jj,1:jj);
    CV=coef(1:jj,jj+1:2*jj);
    CD=coef(jj+1:2*jj,jj+1:2*jj);
    X=idwt2(CA,CH,CV,CD,'sym8');
    coef(1:2*jj,1:2*jj)=X;
    if (level<num_levels)
       clear X;
    else
       X0=X;
   end
   clear CA;
   clear CH;
   clear CV;
   clear CD;
end
X_sd=X0;
lena_hd=X_hd;
lena_sd=X_sd;
D=lena_hd-lena0;
rmse=sqrt(sum(sum(D.^2)))/512;
psnr=20*log10(255.0/rmse);
fprintf(1,'====================================\n');
fprintf(1,'Bayes Hard Thresholding...\n');
fprintf(1,'The RMSE = %d\n', rmse);
fprintf(1,'The PSNR = %d\n', psnr);
fprintf(1,'\n');
fprintf(1,'====================================\n');
D=lena_sd-lena0;
rmse=sqrt(sum(sum(D.^2)))/512;
psnr=20*log10(255.0/rmse);
fprintf(1,'====================================\n');
fprintf(1,'Bayes Soft Thresholding...\n');
fprintf(1,'The RMSE = %d\n', rmse);
fprintf(1,'The PSNR = %d\n', psnr);
fprintf(1,'\n');
fprintf(1,'====================================\n');
colormap(gray(255))
subplot(2,2,1), image(lena_hd);
axis off
axis equal
subplot(2,2,2), image(lena_sd);
axis off
axis equal
colormap(gray(255))
fprintf(1,'The program completed successfully\n');
%*******************************************************************
%===================================================================
```

Bibliography

[1] F. Abramovich, T. Sapatinas, and B. W. Silverman, "Wavelet thresholding via a Bayesian approach," *J. R. Statist. Soc.,* vol. 60, pp. 725-749, 1998.

[2] M.F. Barnsley, *Fractals Everywhere.* New York: Academic Press, 1988.

[3] M.F. Barnsley, and S. Demko, "Iterated function systems and the global construction of fractals," *Proc. Roy. Soc. Lond.,* vol. A399, pp. 243-275, 1985.

[4] M.F. Barnsley, V. Ervin, D. Hardin, and J. Lancaster, "Solution of an inverse problem for fractals and other sets," *Proc. Nat. Acad. Sci. USA.,* vol. 83, pp. 1975-1977, 1985.

[5] M.F. Barnsley, and L.P. Hurd, *Fractal Image Compression.*, Massachusetts: A.K. Peters, Wellesley, 1993.

[6] M. Barnsley, and A.D. Sloan, "A better way to compress images", *BYTE Magazine,* pp. 215-223, 1998.

[7] K.U. Barthel, H.L. Cycon, and D. Marpe, "Image denoising using fractal and wavelet-based methods", *SPIE Proc.* vol. 5266, pp 10-18, 2003.

[8] K. Belloulata, and J. Konrad, "Fractal image compression with region-based functionality," *IEEE Trans. Image Processing,* vol. 11, no. 4, pp. 351-362, 2002.

[9] K.B. Boussaid, and A. Beghdadi, "A new image smoothing method based on a simple model of spatial processing in the early stages of human vision", *IEEE Trans. Image Processing,* vol. 9, no. 2, pp. 220-226, 2000.

[10] C. S. Burrus, R. A. Gopinath, and H. Guo, *Introduction to Wavelets and Wavelet Transforms:* A Primer Prentice Hall. New Jersey, 1997.

[11] T.G. Campbell, and J.M.H Du Buf, "A quantitative comparison of edge-preserving smoothing techniques", *IEEE Trans. Sig. Proc.,* vol. 21, pp. 289-301, 1990.

[12] A. Chambrolle, R.A. DeVore, N. Lee, B. J. Lucier, "Nonlinear wavelet image processing: variational problems, compression, and noise removal through wavelet shrinkage," *IEEE Trans. Image Processing,* vol. 7, no. 3, pp. 319-335, 1998.

[13] S.G. Chang, B. Yu, and Martin Vetterli, "Spatially adaptive wavelet thresholding with context modeling for image denoising," *IEEE Trans. on Image Proc.*, vol. 9, no. 9, pp. 1522-1531, 2000.

[14] S.G. Chang, B. Yu, and Martin Vetterli, "Adaptive image thresholding for image denoising and compression," *IEEE Trans. on Image Proc.*, vol. 9, no. 9, pp. 1532-1546, 2000.

[15] S.G. Chang, B. Yu, and Martin Vetterli, "Bridging compression to wavelet thresholding as a denoising method," in *Proc. Conf. Information Sciences Systems*, Baltimore, MD, pp. 568-573, 1997.

[16] S.G. Chang, B. Yu, and Martin Vetterli, "Image denoising via lossy compression and wavelet thresholding", in *Proc. IEEE Int. Conf. Image Processing,* vol. 1, pp. 604-607, 1997.

[17] R.R. Coifman, and D.L. Donoho, "Translation-invariant denoising," in A. Antoniadis, and G. Oppenheim, editors, Wavelets and Statistics, vol. 103 of Springer Lecture Notes in Statistics, pp. 125-150, New York, Springer-Verlag 1995.

[18] G. Davis, "A wavelet-based analysis fractal image compression," *IEEE Trans. Image Processing,* vol. 7, pp. 141-154, 1998.

[19] I. Daubechies, *Ten Lectures on Wavelets.* SIAM Press, Philadelphia, 1992.

[20] D.L. Donoho, "Nonlinear wavelet methods for recovery of signals, densities, and spectra from indirect and noisy data," in *Proc. of Symposia in Applied Mathematics*, vol., 00, pp. 173-205, AMS, 1993.

[21] D.L. Donoho, "Denoising and soft-thresholding," *IEEE Trans. Infor. Theory*, vol. 41, pp. 613-627, 1995.

[22] D.L. Donoho, and I.M. Johstone, "Ideal spatial adaptation via wavelet shrinkage," *Biometrika,* vol. 81, pp. 425-455, 1994.

[23] D.L. Donoho, and I.M. Johstone, "Adapting to unknown smoothness via wavelet shrinkage," *Journal of the American Statistical Assoc.*, vol. 90, no. 432, pp. 1200-1224, 1995.

[24] J. Hutchinson, "Fractals and self-similarity," *Indiana Univ. J. Math.* vol. 30, pp. 713-747, 1981.

[25] C.J.G. Evertesz, and B.B. Mandelbrot, "Multifractal measures," in *Chaos and Fractals: New Frontiers of Science*, H.-O. Peitgen, H. Jürgens, and D. Saupe. New York: Springer Verlag, 1994.

[26] Y. Fisher, Editor, *Fractal Image Compression, Theory and Application.* New York: Springer-Verlag, 1995.

[27] Y. Fisher, Editor, *Fractal Image Encoding and Analysis, NATO ASI Series F 159.* New York: Springer Verlag, 1998.

[28] B. Forte, and E.R. Vrscay, "Theory of generalized fractal transforms," in *Fractal Image Encoding and Analysis*, Y. Fisher, Ed., *NATO ASI Series F 159*, New York: Springer Verlag, 1998.

[29] B. Forte, and E.R. Vrscay, "Solving the inverse problem for function/image approximation using iterated function systems I: Theoretical basis," *Fractals*, vol. 2, pp. 325-334, 1994.

[30] B. Forte, and E.R. Vrscay, "Solving the inverse problem for function/image approximation using iterated function systems II: Algorithm and computations," *Fractals*, vol. 2, pp. 335-346, 1994.

[31] M. Ghazel, G.H. Freeman, and E.R. Vrscay, "Fractal image denoising," *IEEE Trans. Image Processing,* vol. 12, no. 12, pp. 1560-1578, 2003.

[32] M. Ghazel, G.H. Freeman, and E.R. Vrscay, "Fractal-wavelet image denoising," *Proc. IEEE Int. Conf. on Image Proc. (ICIP),* Rochester, New York, pp. 836-839, 2002.

[33] M. Ghazel, "Adaptive fractal image compression in the spatial and the wavelet domain," Master's of Electrical and Computer Engineering degree thesis, University of Waterloo, April 1999.

[34] M. Ghazel, and E.R. Vrscay, "Generating rate distortion curves for fractal image compression schemes using adaptive and quadtree partitioning algorithms," a *research report,* Department of Applied Mathematics, University of Waterloo, August 1999.

[35] M. Ghazel, and E.R. Vrscay, "Adaptive fractal and wavelet image coding using quadtree partitioning," Proc. *20th Biennial Symposium on Information Theory*, Kingston, ON, May, 2000.

[36] M. Ghazel, and E.R. Vrscay, "An effective hybrid fractal-wavelet image coder using quadtree partitioning and pruning," Proc. *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, Halifax, NS, May 7-11, 2000,

[37] M. Ghazel, E.R. Vrscay, and A.K. Khandani, "Adaptive fractal-wavelet image compression schemes," Proc. *1999 Canadian Workshop on Information Theory,* Kingston, ON, June 15-18, 1999.

[38] M. Ghazel, E.R. Vrscay, and A.K. Khandani, "An interpolative scheme for fractal image compression in the wavelet domain," Proc. *8th International Conference on Computer Analysis of Images and Patterns (CAIP),* Ljubljana, Slovenia, September 1-3, 1999.

[39] R.C. Gonzalez, and R.E. Woods, *Digital image processing,* New Jersey: Prince-Hall, Inc. 2002.

[40] B. Guiheneuf, and J. Lévy Véhel, "2-Microlocal analysis and applications in signal processing," INRIA Rocquencourt, 1997.

[41] *Proc. of the IAM Workshop on "Fractals in Multimedia"*, held at the Institute for Mathematics and Applications, U. of Minnessota, Jan. 14-17, 2001.

[42] *IEEE Trans. Image Processing,* Special Issue on Nonlinear Image Processing, vol. 5, no. 6, June 1996.

[43] A. Jacquin, "Image coding based on a fractal theory of iterated contractive image transformations," *IEEE Trans. Image Processing,* vol. 1, pp. 18-30, 1992.

[44] K. Rank, and R. Unbehauen, "An adaptive recursive 2-D filter for removal of Gaussian noise in images," *IEEE Trans. Image Processing,* vol. 1, no. 3, pp. 431-436, 1992.

[45] E. Jernigan, *SYDE 775 Lecture Notes with Problems*, Courseware Solutions, University of Waterloo, 2000.

[46] R. Adriaanse, and E. Jernigan, "Recursive adaptive Winer filtering," Proc. *CCECE*, Vol. II, pp. 496-499, 1997.

[47] H. Krupnik, D. Malah, and E. Karnin, "Fractal representation of images via the discrete wavelet transform," *Proc. IEEE Conv. EE.*, Tel-Aviv, March 7-9, 1995.

[48] C.K. Lee, and W.K. Lee, "Fast fractal image block coding based on local variances," *IEEE Trans. Image Processing,* vol. 7, no. 6, pp. 888-891, 1998.

[49] J.S. Lee, " Digital image enhancement and noise filtering by use of local statistics", *IEEE PAMI*, vol. 2, no. 2, pp. 165-168, 1980.

[50] J. Lévy Véhel, "Introduction to the multifractal analysis of images," in *Fractal Image Encoding and Analysis*, Y. Fisher, Ed., *NATO ASI Series F 159*, New York: Springer Verlag, 1998.

[51] J. Lévy Véhel, and B. Guiheneuf, "Multifractal image denoising," INRIA Rocquencourt, 1997.

[52] N. Lu, *Fractal Imaging.* New York: Academic Press, 1997.

[53] S.G. Mallat, "A theory for multiresolution signal decomposition: The wavelet representation," *IEEE Trans. PAMI,* vol. 11, no. 7, pp. 674-693, 1989.

[54] S.G. Mallat, *A Wavelet Tour of Signal Processing,* New York: Academic Press, 1998.

[55] B. Mandelbrot, *The Fractal Geometry of Nature.* San Francisco, Freeman, 1983.

[56] *Matlab 7: The Language of Technical Computing,* The Mathworks Worldwide, http://www.mathworks.com/, 2004.

[57] F. Mendivil, and E.R. Vrscay, "Correspondence between fractal-wavelet transforms and Iterated Function Systems", Proc. *Fractals in Engineering, from Theory to Industrial Applications,* J. Lévy Véhel, E. Lutton, and C. Tricot, Eds., New York, Springer-Verlag, pp. 54-64, 1997.

[58] D.M. Monro, "A hybrid fractal transform," in *Proc. ICASSP*, vol. 5, pp. 162-172, 1993.

[59] D.M. Monro, and F. Dudbridge, "Fractal block coding of images," *Electron. Lett.,* vol. 28, pp. 1053-1054, 1992.

[60] B.S. Natarajan, "Filtering random noise from deterministic signals via data compression," *IEEE Trans. Image Processing,* vol. 43, no. 11, pp. 2595-2605, 1995.

[61] P. Saint-Marc, J.S. Chen, and G. Medioni, "Adaptive smoothing: a general tool for early vision", *IEEE Trans. Image Processing*, vol. 13, no. 6, pp. 514-529, 1991.

[62] R. Pinter, editor, *Nonlinear Vision.* New York: *CRC,* 1992.

[63] I. Pitas, and A. N. Venetsanopoulos, *Nonlinear Digital Filters.* New York: Kluwer Academic, 1990.

[64] *Proceedings, Canadian Conference on Electrical and Computer Engineering*, vol. 2, St. Johns, NFLD, June 1997. New York: Kluwer Academic, 1990.

[65] M. Ruhl, and H. Hartenstein, "Optimal fractal coding is NP-hard," in *Proc. IEEE Data Compression Conference,* J. Storer, and M. Cohn, Eds., Snowbird, Utah, 1997.

[66] A. Said, and W.P. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circ. Sys. Video Tech.,* vol. 6, pp. 243-250, 1996.

[67] D. Saupe, and S. Jacob, "Variance-based quadtrees in fractal image compression," *Electron. Lett.,* vol. 33, no. 1, pp. 46-48, 1997.

[68] J. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Sig. Proc.*, 41, pp. 3445-3462, 1993.

[69] D. G. Sheppard, A. Belgin, M. S. Nadar, B. R. Hunt, and M. W. Marcellin, "A vector quantizer for image restoration," *IEEE Trans. Image Processing,* vol. 7, no. 1, pp. 119-124, 1998.

[70] E.R. Vrscay, "A generalized class of fractal-wavelet transforms for image representation and compression," *Can. J. Elect. Comp. Eng.* vol. 23, no. 1-2, pp. 69-84, 1998.

[71] *Wavelab: A Library of Matlab Routines,* http://www-stat.stanford.edu/w̃avelab/, 1994.

[72] S.J. Woolley, and D.M. Monro, "Rate distortion performance of fractal transforms for image compression," *Fractals,* vol. 2, pp. 395-398, 1994.

[73] S. Zhong, and V. Cherkassky, "Image denoising using wavelet thresholding and model selection," *Proc. IEEE Int. Conf. on Image Proc. (ICIP),* Vancouver, BC, Sept. 2000.