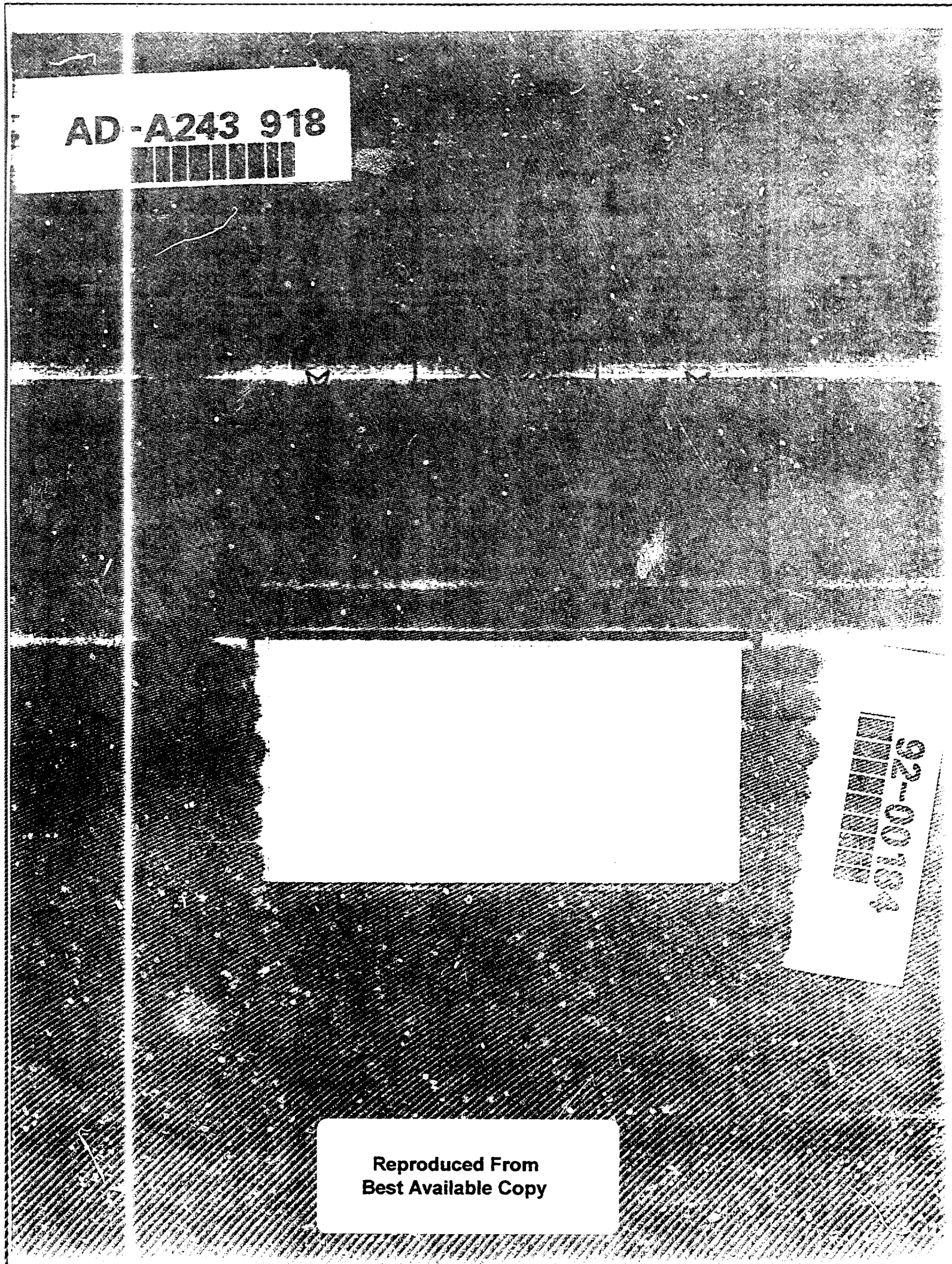


AFIT/~~EN~~GE/ENG/91D-50

IMAGE SEGMENTATION USING AFFINE WAVELETS

THESIS

Steven E. Smiley
Captain, USAF
AFIT/~~EN~~GE/ENG/91D-50

AFIT/EN/ENG/91D-50

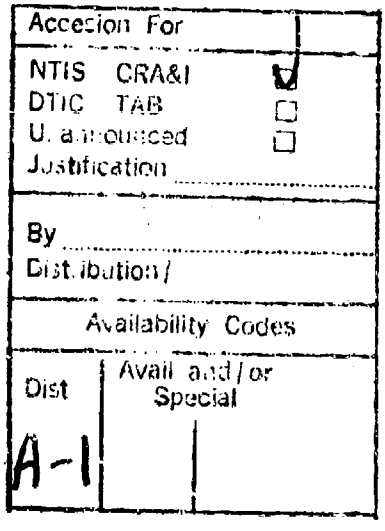
IMAGE SEGMENTATION USING AFFINE WAVELETS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Steven E. Smiley, BS
Captain, USAF

December 12, 1991



Acknowledgments

I would like to start with the basics. I am grateful for the love and support of my wife Sara. While I was at school working on this project she was taking care of the home front and filling in for my absence. If it had not been for you I could not have put forth my best effort on this work.

Next I would like to thank the members of my thesis support team who gave me the necessary strategies, equipment, and encouragement to finish this project. To Major Steve Rogers for giving me the guidance and knowledge upon which I formulated my thesis topic and for his endless enthusiasum. Thank you for having confidence in me. To Drs. Matthew Kabrisky, Dennis Quinn, and Greg Warhola for being a source of knowledge and a sounding board for the many ideas and questions which arose during the process of completing this thesis. To Stewart Laing for showing me that there is always more than one way to solve a problem and for his continued friendship, even after spending so many hours working together.

Thanks also to my sponsors, Kevin Willey and Edward Zelnio from the Automatic Target Recognition Group, Aeronautical Systems Division Wright-Patterson Air Force Base Ohio. Their efforts to provide support and data were beneficial to the success of this thesis.

Steven E. Smiley

Table of Contents

List of Figures

AFIT/EN/ENG/91D-50

Abstract

This thesis discusses the use of the multiresolution representation and Radial Basis Function (RBF) neural networks to segment both FLIR and SAR imagery. The multiresolution approximation coefficients are used as features into the RBF network which learns to distinguish between different cultural and natural regions or objects. The wavelets used are Mallat's spline wavelet and Daubechies' compactly supported wavelets. Additionally, this thesis provides an explanation of wavelets in a tutorial manner. It introduces wavelet theory and discusses two different approaches to generating the multiresolution or wavelet representation.

IMAGE SEGMENTATION USING AFFINE WAVELETS

I. Introduction

1.1 Introduction

Application of the wavelet transform in the pattern recognition process has shown promising results, in particular, the wavelet's use in processing images. This work pursues the development of a system to robustly segment Synthetic Aperture Radar (SAR) images and Forward Looking Infrared (FLIR) images using orthonormal wavelets.

1.2 Statement of Problem

A system which can robustly segment SAR and FLIR imagery does not yet exist. This work explores the ability of a multiresolution representation to segment natural and cultural features in SAR and FLIR images. The wavelet transform will be the primary mathematical basis for the decomposition of images, while the Multiresolution Decomposition will be the main analytical tool.

1.3 Background

"In 1830 about 300 technical and scientific journals were in circulation. Today there are over 60,000 journals and 2.5 million articles per year throughout the world in over 50 languages."[35:2] A growing problem each passing year is our inability to manage the abundance of information we produce. One approach to this problem is the automation of data processing systems with the ability to accomplish pattern recognition.

There are many systems in existence today which can identify or classify, with a high degree of accuracy, individual non-connected elements. One such system can recognize individually printed numbers, another system can recognize non-connected words (words spoken

one at a time with an accentuated pause between each word). Although these results are impressive, in practice most pattern recognition tasks require the recognition of an element which is connected to other elements. One of the unsolved problems in pattern recognition is how to isolate or segment the elements of interest from other elements of interest or from the rest of the "background" clutter. A reliable segmentation system in conjunction with an existing classification system could be employed in many applications. Such a system in the US Postal Service would be able to process nearly all hand-written addresses automatically, without resorting to a human operator. Unfortunately no such system is yet fully capable of this segmentation task and much of our mail today is still processed by a human operator as far as segmenting the individual letters and numbers of the addresses on the envelope is concerned.

The United States Air Force is today pursuing research in the area of image processing with the goal of reliably detecting, recognizing, and classifying targets. The fundamental problem with target detection is segmentation or finding possible targets in highly cluttered images[2:2].

> The concept of separating one object from another is as common to humans as breathing. We constantly manipulate our environment to suit our needs, whether we physically change the environment (actually move or remove objects within our environment) or simply change the way we perceive or observe the environment.
>
> Often, we cannot physically alter the environment in which a target resides, short of destroying the target and its surroundings. Therefore, we attempt to observe the target in its surroundings. We must separate the target from the rest of the image to observe or detect the target. Distinguishing targets from background is referred to as segmentation.[2:2]

This thesis will justify further application of wavelets in the process of segmenting images, specifically segmenting between natural items such as trees, fields, and shadows, in SAR imagery as well as man made objects in FLIR imagery.

1.4 Scope

This research will show that cluttered imagery can be segmented using elements of a multiresolution representation. This multiresolution representation is produced with respect to dilations and translations of a mother wavelet as is the case when using an affine

wavelet transform. The resulting coefficients are then processed using standard thresholding techniques and neural network techniques to accomplish the segmentation.

This research will not define the theoretical limits on the use of the wavelet nor will it attempt to show the use of all possible combinations of "mother wavelets". This study uses a very small subset of possible "mother wavelets" and demonstrates their usefulness in the segmentation of images.

1.5 Summary of Current Knowledge

1.5.1 Pattern Recognition and Segmentation Pattern recognition of images is, in general, composed of three steps. First, images are segmented into regions of interest. Second, features are extracted from these regions of interest. These features might include such things as length to width ratios of the regions or perhaps the average pixel value of a region. The third and final step is to classify these regions of interest into some predetermined categories. These categories could be tanks, trucks, background clutter or some other appropriate category depending on the types of images being used.

Segmentation is the process of subdividing an image into its constituent parts or objects. Segmentation is one of the most important steps in automated image analysis. It is at this step that objects or other entities of interest are extracted from an image for further processing such as description and recognition [15:331].

Algorithms for segmentation are generally based on one of two basic properties of gray values: discontinuity and similarity. Discontinuity algorithms partition an image based on abrupt changes in gray scale. The approach of similarity algorithms is aimed at thresholding, region growing, and region splitting [15:331]. Recent work at AFIT makes use of similarity algorithms for image segmentation. Albert L'Homme used Gabor coefficients [20] and Joseph Brickey used fractal dimension to [3] segment high resolution SAR imagery in their thesis research. This thesis demonstrates that segmentation of both SAR and FLIR imagery can be accomplished using the affine wavelet coefficients.

1.6 General Approach

The proposed system is composed of three stages of processing. The first stage generates a multiresolution representation of a SAR image. The second stage extracts features from a particular level of the multiresolution representation. The third stage used a radial

basis function artificial neural network, which accomplishes the segmentation based on the multiresolution features.

1.7 Objectives

The specific objectives of this research are to answer the following questions:

- How is the Wavelet transform related to other types of signal or image analysis tools?
- How is the Multiresolution Representation obtained or calculated for a signal or image?
- Do the multiresolution coefficients provide values which can be used to separate natural and man-made regions within both SAR and FLIR imagery?
- Which set of coefficients should be used as the features?
- Can the Radial Basis Function (RBF) artificial neural network be trained to autonomously segment SAR and FLIR imagery using the wavelet coefficients as features?
- Will the RBF neural network segmentation using the multiresolution coefficients generalize to all areas of an image? If so, will it also generalize to additional images not used in network training?

1.8 Materials and Equipment

The image processing equipment consists of SPARC stations which currently reside at the Model Based Vision (MBV) Laboratory and the those at AFIT on Wright Patterson AFB. The SAR and FLIR imagery is supplied by the same lab. The numerical analysis support is provided by a member of the AFIT Department of Mathematics and Statistics.

1.9 Sequence of Presentation

Chapter 1 is a general introduction to the problem and an approach to its solution. Chapter 2 provides a review of literature which is relevant to segmentation, wavelets, and radial basis function artificial neural networks. Chapter 3 is a tutorial on the Wavelet transform and the multiresolution representation. Chapter 4 explains the algorithm used to generate the multiresolution approximations and contains some pictorial examples of the multiresolution approximations of various images. Note that Chapters 3 and 4 were jointly written with Capt John (Stewart) Laing [19] and it is highly recommended that the reader become very familiar with chapters 3 and 4 before proceeding on to chapter 5. This provides

a common vocabulary which makes Chapter 5 on experimental application and results more understandable. Chapter 6 contains conclusions regarding this research and recommendation for further research in this area.

II. Literature Review

2.1 Introduction

This literature review is undertaken to show that the wavelet transform is an excellent ongoing area of research as it relates to the pattern recognition process and, in particular, to image segmentation. Additionally, it demonstrates the utility of radial basis function artificial neural networks in the area of pattern recognition.

2.2 Scope of Review

This review covers a small subset of the available literature. It provides simple justification for research on the wavelet transform as a means of segmenting images.

2.3 Background

Pattern recognition is in general composed of three steps. First, images are segmented into regions of interest. Second, features are extracted from these regions of interest (these features might include such things as length to width ratios of the regions or perhaps the average pixel value of a region.). The final step is classification of the regions of interest into some predetermined categories. These categories could be tanks, trucks, background clutter or some other appropriate category depending on the types of images being used.

The inherent difficulty of these three tasks has caused many researchers to look to the biological visual system as a model. This is due to the fact that in general biological visual systems (human, cat, dog, spider, etc) are vastly superior to any constructed by man, thus far. The study of mammalian visual systems has provided insight into the decomposition and processing of visual stimuli. A great deal of recent research has been conducted to artificially mimic the biologically based decomposition and processing.

One numerical method which can be employed in the pattern recognition process which may show some promise is based on the discrete wavelet transform. The discrete wavelet transform is a representation of an arbitrary function having finite energy as the superposition of a set of functions known as wavelets[1:2297].

2.4 *Method of treatment and organization*

This literature review is organized into three specific areas of research as they relate to pattern recognition and image processing in general. The first area is biological research, which has been used as the basis for some image processing approaches. This includes the use of Gabor functions, as they have been used in image coding and pattern classificatioı The second area deals with the wavelet transform and its usage, thus far, in pattern recognition. The final area will cover the use of artificial neural networks in pattern classification and, in particular, the use of radial basis functions.

2.5 *Neurophysiological Research*

2.5.1 Theories on Building Perception David Hubel has shown that the visual cortex is functionally subdivided into columns of neurons which respond to similar input. He has also shown that different neurons in the visual cortex respond to different types of input stimulus [17].

David Hubel's exploration of the " transformation of the retinal image into a perception" [17:54] was carried out on the brains of adult cats. The cats were anesthetized and faced toward a wide screen 1.5 meters away. On the screen various patterns of white light were projected. As a pattern of light appeared a microelectrode was inserted into a portion of the cat's visual system. This provided a method of recording the response of individual neurons to a particular light pattern. The correlation or comparison of the light patterns and the neuronal output gave rise to the following generalizations about the mammalian visual system.

- Orientation is an important factor to neuronal response. Some of the neurons responded only to lines at certain angles as the cat faced the screen. Additionally, a large number of neurons in the Lateral Geniculate responded to differences in illumination intensity rather than the amount of total intensity. The cortical neurons showed vigorous response to slow downward movements and a lesser response to upward movements. They showed no response to side-to-side motion [17:2].
- A large number of cortical neurons are considered "simple" and respond depending on the orientation and position of the shape with regard to the cells receptive field. "Complex" cells also responded strongly to edges and bars and their associated orientation but they were not as discriminating as to the exact position of the stimulus.[17:59,60]

These results suggest that mammalian visual perception *is built from small pieces of the whole*[17].

2.5.2 The Gabor Function Approximates the 2D Visual Receptive Field Responses Once research of visual perception had shown evidence of this parts-to-whole relationship, other researchers attempted to quantify some of the intermediate representations. The work of Jones and Palmer showed one form of the processing that is done on 2-dimensional(2D) images in the mammalian visual system. A graphical representation of this processing is very similar to the graph of a set of mathematical functions known as Gabor functions [18:1180].

Fourteen adult cats were anesthetized and shown various visual stimulus by way of an oscilloscope screen. The cats were shown an illuminated dot on a dark screen and a dark dot on an illuminated screen for various intervals of time. Simultaneously microelectrodes were recording the neuronal responses from the visual cortex. These sets of data were later compared using a process called "reverse correlation"[18:1180] and produced a 2D image which compared with little error to the 2D Gabor functions.

2.6 Image Processing using The Gabor Transform

2.6.0.1 Finding Optimal Gabor Coefficients Using the Gabor functions as a model of a portion of human visual processing, John Daugman showed that, because of the intrinsic redundancy, 2D images can be coded using the 2D Gabor transform. He has been able to code images into a more compact form and thus reduce the amount of data to be transmitted for image reconstruction. Daugman has also shown the usefulness of the Gabor transform for image analysis and image segmentation. The Gabor transform "extracts locally windowed 2D spectral information concerning form and texture without sacrificing information about 2D location or more global spatial relationships, as does a Fourier transform."[11]

The main thrust of Daugman's article [11] was to show the utility of a neural network for obtaining the Gabor coefficients to represent an image with a set of Gabor functions. Since the 2D discrete Gabor transform is not an orthonormal set of functions, it is computationally intensive to directly calculate the coefficients for an optimal approximation to an image. Daugman's neural network approach is used to find an optimal set of coefficients which produces an optimal match to the original image.

2.6.0.2 FLIR Image Segmentation Using Gabor Coefficients During his thesis research, Kevin Ayer segmented targets from non-targets in a Forward Looking Infrared (FLIR) image using Gabor functions[2]. Ayer determined the Gabor transform of FLIR images and obtained multiple sets of Gabor coefficients. Each set of coefficients represented a particular orientation of interest. The Gabor coefficients constituted a correlation coefficient of a particular Gabor function as it relates to the image of interest. From the Gabor coefficients he was able to segment the image into regions of interest. He then used conventional pattern recognition techniques to classify a specific region of interest using the Gabor coefficients as his features.

2.6.1 SAR Image Segmentation Using Gabor Coefficients Albert L'Homme has constructed a system to segment SAR imagery using the Gabor transform[20]. Utilizing a small subset of possible orientations and a constant modulation of Gabor functions he generated a set of Gabor coefficients. He found that the bandwidth of the Gabor functions to be more significant in segmentation than the orientation parameters. These coefficients were then processed using a radial basis function self-organizing neural network to segment a SAR image. He was able to segment with measures of accuracy up to 92% as compared to hand segmented imagery.

2.6.2 Wavelets in image processing As described in the previous section, image coding, segmentation, and feature extraction have been done using Gabor functions. These functions are a subset of a larger class of functions known as wavelets [22:2098]. According to Mallat [22], [25], [1], the wavelet transform can provide a multiresolution representation of an arbitrary function having finite energy. This representation allows for localization of frequency content of an image, and provides a tool for texture and edge discrimination. Both texture and edge detection can be an effective tool in pattern recognition.

Christopher Heil has contributed to understanding the mathematical definition of wavelets. Heil explains that separable Hilbert Spaces, in particular $\mathbf{L^2(R)}$, possess an orthonormal basis. The major benifit of an orthonormal basis is that it provides a decomposition for a Hilbert space such that if $\{e_n\}$ is an orthonormal basis for $\mathbf{H}$ then every $x \in \mathbf{H}$ can be written

$$x = \sum_n \langle x, e_n \rangle e_n$$

This does not guarantee that the basis set can be found or that when found it will be convenient to work with[16:147].

"Frames are an alternative to orthonormal basis sets. By giving up the requirements of orthogonality and uniqueness of decomposition we allow much more freedom in our choice of "basic vectors", while still retaining the ability to decompose the space."[16:147] If $\{x_n\}$ is a frame, then every $x \in \mathbf{H}$ can be written

$$x = \sum_n c_n x_n$$

in such a way that scalars are computable, and the series converges to x.

Frames fall into two general categories called Weyl-Heisenberg(W-H) frames and affine frames. The W-H frames are composed of discrete modulations and translations of a single function, known as a "mother wavelet". The affine frames are composed of discrete dilations and translations of the "mother wavelet" [16:147-159]. The Gabor transform mentioned above falls into the W-H category and doesn't form an orthonormal basis set.

2.6.2.1 Orthonormal Wavelets Daubechies has revealed the form of a set of wavelets which are orthonormal [8]. These orthonormal sets can be used to exactly reconstruct a function from its wavelet coefficients with the classical expansion method formula. This provides a straightforward means of exact reconstruction of an image rather than an optimal reconstruction. The wavelet coefficients should prove useful in some areas of pattern recognition [22]. Daubechies has also provided orthonormal wavelets with a compact support meaning the wavelet has nonzero values on a finite interval.

2.7 Artificial Neural Network Radial Basis Function Classifiers

Artificial neural networks constitute one class of architecture for parallel distributed processing systems. This architecture follows from what is currently known and hypothesized about the mammalian nervous system. A large number of papers have been recently published with respect to neural network techniques and tools which can be applied to pattern recognition problems [33:28].

In Dan Zahirniak's thesis [37] on the characterization of radar signals, he explains the concepts and architecture of RBF neural networks. The citations following each section are the original literature sources used by Zahirniak.

Neural network classifiers are categorized as either a Hyperplane Classifier, an Exemplar Classifier, a Probabilistic Classifier or a Kernel Classifier. These categories are dependent on the method used by the network to accomplish classification [21:47-63]. The most

common neural network classifier is the multilayer perceptron. Using a single hidden layer where each node uses a sigmoidal function to calculate its output from a sum of the product of its inputs and their associated weights, the perceptron is a hyperplane classifier. This type of classifier is based on the property that any multivariate function can be approximated with a finite superposition of sigmoidal functions.

The Radial Basis Function (RBF) departs from the traditional McCuilough and Pitts neuron and falls into the Kernel Classifiers category. The Kernel Classifier is characterized by:

- The overlapping radial basis functions create a complex receptive-field decision region over the feature space as shown in Figure 2.1.
- The basic premise of the network is that any multivariate function can be reasonably approximated using a linear combination of radial basis functions with their centers on or near data points[29:143-167] [28:978-980].

Additionally, the RBF network architecture consists of establishing a single hidden layer, with nodes in the hidden layer transforming inputs to outputs using a radial basis function [37].

2.7.1 Localized Receptive Fields The same type of neural network architecture discussed by Zahirniak in the previous subsection is explored by Moody and Darken in [26]. They mention that locally-tuned overlapping receptive fields (radial basis functions) are known data structures in biological nervous systems. These same receptive field have plasticity which is comparable to the development of orientation selective cells in the visual cortex. The exact learning of the RBF network is described as a two stage hybrid process where the lower layer field centers and field width is determined by a self-organizing manner and the amplitudes of the node is determined by a supervised LMS rule. This provides for faster learning since only the output weights are calculated using an error term. Moody and Darken apply the receptive field network to predicting the Mackey-Glass differential delay equation time series. Their results demonstrate that the receptive field network achieves comparable prediction accuracy to a multilayer perceptron using a gradient descent learning algorithm in significantly less time, on the order of 1000 times faster [26].

In [27] Nowlan describes the difference between hard and soft learning algorithms for RBF networks.

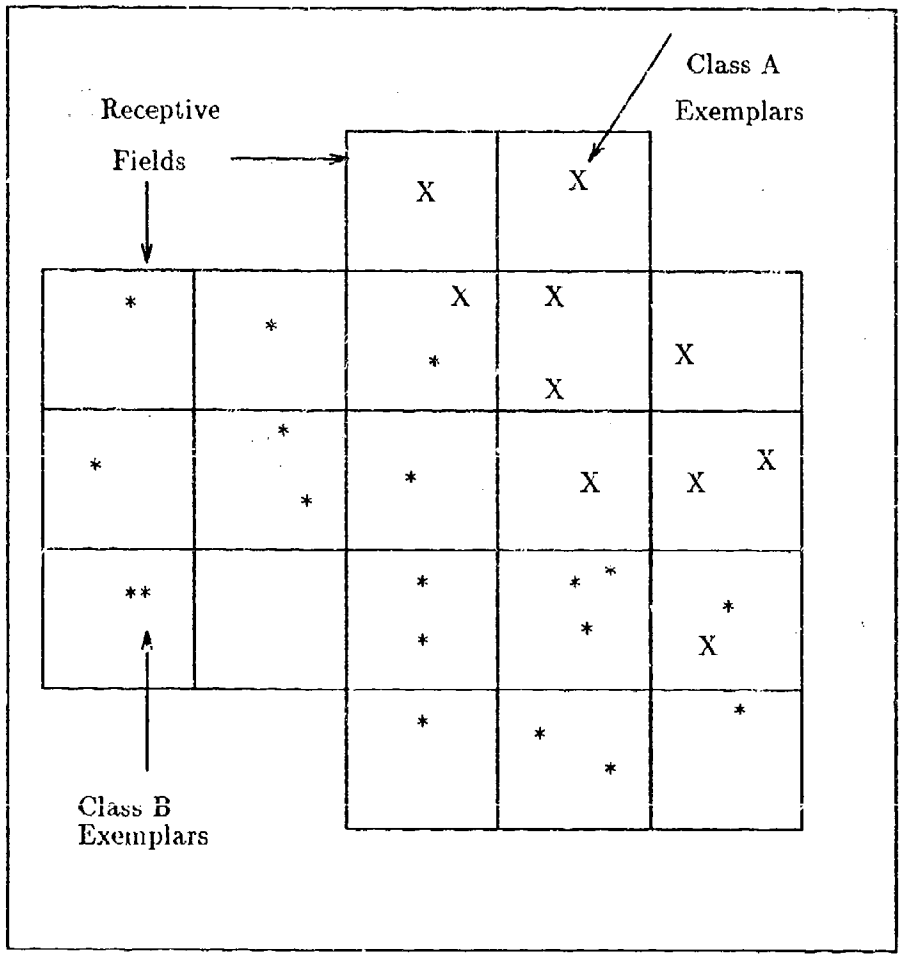


Figure 2.1. Kernel Classifier Network Decision Regions[37:12]

- The Hard learning algorithm requires updating weights on a winner take all criteria. Only the winning node is updated and all other node are unaffected.
- The Soft learning algorithm requires updating weights based on their proportionality to the present vector's input strength. All of the nodes are updated. The amount of the update is proportional to their response to the input vector.

Additionally, Nowlan describes various methods for placing the centers of the radial basis functions in the feature space. These method include:

- K-means center selection with an adjustment to the size of the RBF to reduce a smoother interpolation.
- Cluster centers are assigned based on the closest mean then the mean is recalculated based on the average of the samples within its class.

Nowlan tested an RBF network using Soft and Hard algorithms against each other as well as against a multilayer perceptron network using a Least Mean Square (LMS) learning algorithm. He utilized two types of data for this comparison. The first type was hand drawn digits and the second type of data was human speech in the form of digitized versions of the first and second formant frequencies of 10 vowels from multiple speakers. In both cases the RBF network using soft learning algorithms were able to outperform the same network using Hard algorithms.

2.8 Conclusions

Pattern recognition tasks of segmentation, feature extraction, and classification are in general very difficult. Research into the mammalian visual system has given some insight and provided new avenues of approach to these tasks. The wavelet representation provides varied views of data depending on the dilation of the basis set or wavelet used. The use of wavelets can provide a multiresolution representation of visual data. This provides a method of representing redundancies in visual data as Daugman and L'Homme found. This representation may provide useful features to the biologically motivated radial basis neural network. The network provides a means of using the wavelet representation to segment imagery. This thesis explains and applies these two relatively new tools to the segmentation task.

III. Theory of Wavelet Analysis

This chapter was co-authored with John (Stewart) Laing and exists in his thesis in duplicate [19].

3.1 Introduction

Signal analysis seeks to discover the information content of signals needed for applications such as pattern recognition and signal coding. One approach is to transform a mathematical representation of the signal into a domain of interest. A simple example is a coordinate transformation which maps a function, such as a circle, from Cartesian coordinates to polar coordinates. A circle represented by $x^2 + y^2 = r^2$ in Cartesian space is now more easily expressed by $\rho = r$ in polar space. The coordinates x and y or ρ and θ provide alternate representations of the circle.

Another example of this kind of transform analysis is the Fourier series expansion. If $f(x)$ is a continuous function on the interval $[-\frac{T}{2}, \frac{T}{2}]$ and $f(-\frac{T}{2}) = f(\frac{T}{2})$,

$$f(x) = \sum_n c_n e^{\frac{jn2\pi x}{T}} \tag{3.1}$$

where $j^2 = -1$, and n is an integer. The Fourier series expansion of a function requires the generation of coefficients, c_n.

$$c_n = \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) e^{\frac{-jn2\pi x}{T}} dx \tag{3.2}$$

These coefficients are the amplitude and phase of each member of the Fourier series basis set needed to reconstruct the original function. In continuous form, Equation 3.2 becomes the Fourier Transform.

$$F(\xi) = \int_{-\infty}^{+\infty} f(x) e^{-j2\pi\xi x} dx \tag{3.3}$$

It maps one dimensional signals from the time domain to the frequency domain and can be extended to map two dimensional images from the space domain to the spatial-frequency domain. From another point of view, the transform projects the original signal or image onto the space spanned by the exponential basis set, $\{e^{j2\pi\xi nx} | n \text{ is an integer}\}$, for all integers n. In this paper we will denote this set with the symbol E_n.

Unfortunately, the Fourier Transform representation gives no information as to the location of the frequency characteristics in the original signal. This is due to the fact that the basis set E_n has infinite support. Therefore, any abrupt changes in the time domain require contributions from the entire frequency domain. The Fourier Transform might indicate that high frequencies are present in the signal, but it does not indicate where in time that range of frequencies are significant. In images, edges or lines are areas of high spatial frequency. A Fourier Transform of an image with edges would provide evidence of high spatial frequencies but would not indicate where in the image the edges could be found. Finding the location of unique spectral characteristics can be extremely useful as a feature set in applications such as pattern recognition and signal coding [11, 24].

Therefore, we need an extra variable in the target or transform domain. In other words, we need a transformation that maps a signal to the time/frequency domain or an image to the space/spatial-frequency domain. The Windowed Fourier Transform (WFT) is such an transformation.

$$WF_f(\omega, \tau) = \int_{-\infty}^{+\infty} \mathrm{w}(t - \tau) e^{-j\omega t} f(t) dt \tag{3.4}$$

where $\mathrm{w}(\bullet)$ is the window function. This transformation uses the window to localize the analysis of time and frequency on the signal. However, because the window size is fixed, no sharper resolution in time can be provided. Due to the uncertainty principle, it is impossible for this basis set to have arbitrarily high resolution in both time and frequency [10, 34]. Even the Gabor Transform, a WFT whose Gaussian shaped window gives the best compromise, still falls prey to the uncertainty principle. Additionally, because the window width is fixed sharp discontinuities in the time signal are spread across many Fourier coefficients.

One answer to the time/frequency resolution problem is the Wavelet Transform[1]. It allows variations in the size of the window effectively trading resolution in time for resolution in frequency. The collection of its coefficients, similar to the Fourier Transform, is a projection of the original signal or image onto the space spanned by its basis set. The wavelet basis set is made up of variations in the translation and dilation of a mother wavelet function just as the $\{E_n\}$ is made up of variations in the frequency of the complex exponential function.

This chapter provides the basics for understanding wavelet analysis. It presents the Wavelet Transforms of both continuous and discrete signals. We discuss Multiresolution

[1]Another approach to the time/frequency resolution problem is that of Time-Frequency Distributions [14, 6]

Wavelet Analysis both in terms of successive projections onto a wavelet basis set and successive lowpass and bandpass filtering in the Fourier domain. Finally, we address the extension of Multiresolution Wavelet Analysis to two dimensions.

3.2 Notation

The following notation will be used throughout this document.

- $\mathbf{Z}$ denotes the set of integers.
- $\mathbf{R}$ denotes the set of real numbers.
- $\mathbf{R}^+$ denotes the set of positive real numbers.
- $\mathbf{L^2(R)}$ denotes the space of measurable, square integrable, one dimensional, real-valued functions $f(x)$, such that

$$\int_{-\infty}^{\infty} |(f(x))|^2 dx < \infty \tag{3.5}$$

- $\mathbf{L^2(R^2)}$ denotes the space of measurable, square integrable, real-valued, two dimensional functions $f(x,y)$, such that

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} |f(x,y)|^2 dx dy < \infty \tag{3.6}$$

- For $f, g \in \mathbf{L^2(R)}$ the inner product of f with g is defined as

$$\langle f, g \rangle = \int_{-\infty}^{+\infty} g(x) f(x) dx \tag{3.7}$$

- For $f, g \in \mathbf{L^2(R)}$ the convolution of f with g is defined as

$$[f * g](x) = \int_{-\infty}^{+\infty} f(\alpha) g(x - \alpha) d\alpha \tag{3.8}$$

- For $f, g \in \mathbf{L^2(R)}$ the correlation of f with g is defined as

$$[f \star g](x) = \int_{-\infty}^{+\infty} f(\alpha) g(\alpha - x) d\alpha \tag{3.9}$$

- $\mathbf{P}_n$ denotes the projection operator on $\mathbf{L^2(R)}$ such that for any $f \in \mathbf{L^2(R)}$

$$\mathbf{P}f = \sum_n \langle f, \phi_n \rangle \phi_n \tag{3.10}$$

where $\{\phi_n\}$ is a complete basis set and $n \in \mathbf{Z}$.[2]

3.3 The Continuous Wavelet Transform

The basis functions in wavelet analysis, $\{\psi_{ab}\}$, are derived from a single function called the mother wavelet, $\psi(x)$. It acts as the window in the Wavelet Transform whose size is varied by the dilation parameter, $a \in \mathbf{R}^+$. Like the Windowed Fourier Transform, it has a translation parameter, $b \in \mathbf{R}$.

$$\psi_{ab}(x) = \frac{1}{\sqrt{a}} \psi(\frac{x-b}{a}) \tag{3.11}$$

The $\frac{1}{\sqrt{a}}$ term normalizes the energy of each basis function. Figure 3.1 shows dilated and translated versions of a mother wavelet.[3] The function in the middle is the prototype function where $b = 0$ and $a = 1$. The function to the right is translated by $b = 15$ and dilated by $a = \frac{1}{2}$. And finally, the function to the left is translated by $b = -20$ and dilated by $a = 2$. All such possible dilations and translations of the mother wavelet, $\psi(\frac{x-b}{a})$ make up the elements of the set $\{\psi_{ab}\}$.

This basis set provides narrow windows for small a isolating discontinuities in time that are spread over a broad range of frequencies and wide windows for large a that have better frequency resolution. The Continuous Wavelet Transform for a real mother wavelet ψ is [14:7]

$$W_f(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{+\infty} f(x) \psi(\frac{x-b}{a}) dx, a \in \mathbf{R}^+, b \in \mathbf{R} \tag{3.12}$$

With this transform, a wavelet coefficient is obtained for each dilation and translation of the mother wavelet.

If the Fourier Transform of the mother wavelet, $\psi(x)$, denoted by $\Psi(\omega)$, satisfies the condition that

$$c = \int_0^{+\infty} |\Psi(\omega)|^2 / |\omega| d\omega < \infty \tag{3.13}$$

---

[2]The relationship of this basis set ϕ_n to the mother wavelet $\psi(x)$ is discussed in Section 3.6 of this chapter.

[3]Laplacian of the Gaussian $\psi(x) = \frac{2}{\sqrt{3}} \pi^{-\frac{1}{4}} (1 - x^2) e^{-\frac{x^2}{2}}$.

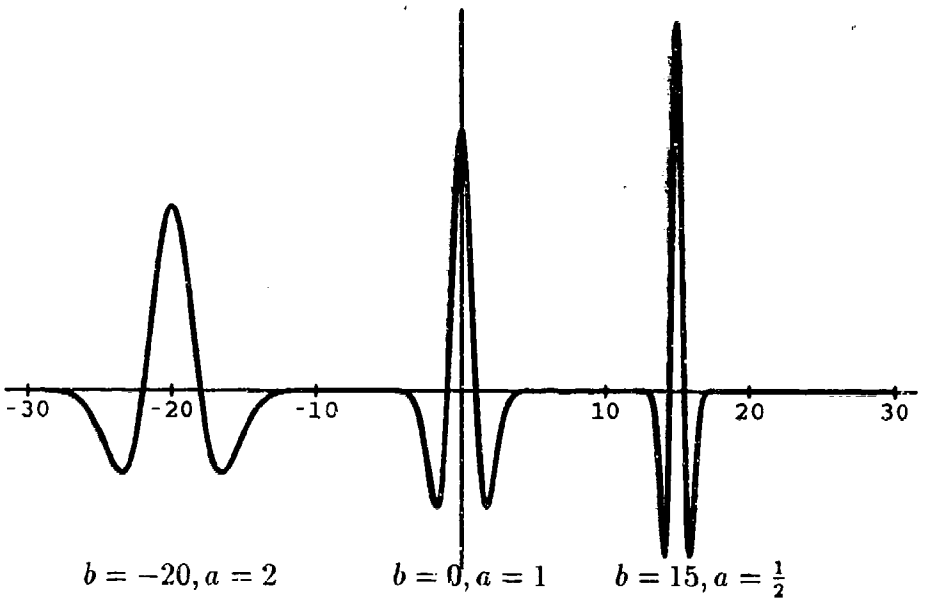


Figure 3.1. A Typical Mother Wavelet

which requires that $\Psi(0) = 0$ [4], an inversion transform exists and is given by [14:8]

$$f(x) = c^{-1} \frac{1}{\sqrt{a}} \int_{-\infty}^{+\infty} \int_{0}^{+\infty} \psi(\frac{x-b}{a}) W_f(a,b) \frac{dadb}{a^2} \tag{3.14}$$

The wavelet transform pair given in Equations 3.12 and 3.14 are analogous to the Fourier Transform pair of Equations 3.1 and 3.3. As the dilation parameter a varies, the window width of function $\psi(\frac{x-b}{a})$ varies. Since small values of a correspond to small window widths, a varies inversely with the frequency detectable within the window. Therefore, the wavelet transform isolates time discontinuities or abrupt changes in time at the expense of low frequency resolution at high frequencies. In many applications, the important information content of the signal is contained in the quick transitions of the signal in time. For this reason, the Wavelet Transform can be quite useful.

Because the windows overlap when the parameters (a, b) are varied continuously, the Wavelet Transform is highly redundant. Therefore, it is possible to evaluate it with a discrete set of basis functions in much the same way that the Fourier expansion of Equation 3.1 represents a signal with a set of discrete exponentials. The time/frequency plane evaluating grids are shown in Figure 3.2 for uniform time-frequency sampling associated with the Windowed Fourier Transform and the nonuniform sampling of the Wavelet Transform. Each dot in the

[4] The ω in the denominator of Equation 3.13 requires that $\Psi(\omega)$ vanishes as ω approaches zero.

lattice indicates the localization in the time/frequency plane of one resolution cell, showing the center of the time window and corresponding bandpass filter. In this figure, we can see

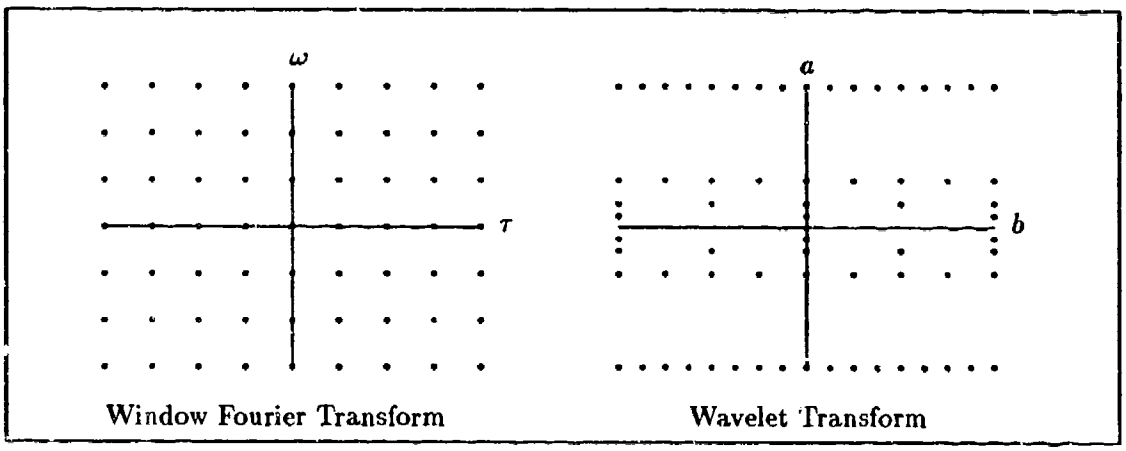


Figure 3.2. Time/Frequency Window Localization Lattice [9:41]

that the fixed window widths of the Windowed Fourier Transform have a fixed resolution in time and frequency; whereas, the variable window widths of the Wavelet Transform provide variable resolution in time and frequency. The clustering of grid dots at the origin along the a^{-1} axis of the Wavelet Transform time/frequency lattice indicate the low time resolution or localization of low frequencies; whereas, the denseness of grid dots parallel to the shift axis, b, at high frequencies (large a^{-1}) indicates the higher time resolution or localization of higher frequencies.

3.4 The Wavelet Transform with Discrete Wavelets

It is sometimes convenient to use a mother wavelet whose discrete translations and dilations form an orthonormal basis [7]. For this case, the discretized basis set $\{\psi_m^n\}$ where $m, n \in \mathbf{Z}$ is defined as

$$\psi_m^n(x) = \alpha^{-\frac{m}{2}} \psi(\alpha^{-m} x - n\beta) \tag{3.15}$$

where $\alpha > 1$ and $\beta > 0$ [14:11]. In this chapter, we use the dyadic interval defined to be $\alpha = 2$ and $\beta = 1$. For the dyadic case, Equation 3.15 becomes

$$\psi_m^n(x) = 2^{-\frac{m}{2}}\psi(2^{-m}x - n) \tag{3.16}$$

Using this form of the mother wavelet in Equation 3.12 yields the Wavelet Transform with a discrete wavelet basis.

$$W_f(m,n) = 2^{-\frac{m}{2}}\int_{-\infty}^{+\infty}\psi(2^{-m}x - n)f(x)dx \tag{3.17}$$

To check this, consider the Fourier Expansion given in Equation 3.1. We can represent any function, $f \in \mathbf{L^2(R)}$ as

$$f(x) = \sum_n c_n\psi_n(x) \tag{3.18}$$

where ψ_n is the n^{th} element of an orthonormal basis for $\mathbf{L^2(R)}$. Equation 3.18 can also be thought of as the reconstruction of $f(x)$ from its coefficients $\{c_n\}$ in terms of the orthonormal basis $\{\psi_n\}$. The inner product, $c_n = \langle f, \psi_n\rangle$, gives the coefficient of the n^{th} term in the basis. Just as any vector r in three dimensional Euclidian space can be expanded in a set of mutually orthogonal unit vectors x, y, and z in the form $r = a_1x + a_2y + a_3z$, we can expand any function $f \in \mathbf{L^2(R)}$ in a set of mutually orthogonal unit vectors $\{\psi_n\}$ in the form $f = \sum_n c_n\psi_n$. If we multiply both sides of Equation 3.18 by any term ψ_m for $m \in \mathbf{Z}$ and integrate, we get

$$\int_{-\infty}^{\infty}\psi_m(x)f(x)dx = \sum_n c_n\int_{-\infty}^{\infty}\psi_n(x)\psi_m(x)dx \tag{3.19}$$

But, because of the orthonormality of the set $\{\psi_n\}$ we know that

$$\int_{-\infty}^{\infty}\psi_m(x)\psi_n(x)dx = \delta_{nm} \tag{3.20}$$

where δ_{nm} is the Kronecker's symbol, and is defined as 0 if $m \neq n$ and 1 if $m = n$. Therefore, all the terms in the summation of Equation 3.19 are zero except the one in which $n = m$. Thus,

$$\int_{-\infty}^{\infty}f(x)\psi_m(x)dx = c_m \tag{3.21}$$

is the integral form we need to find the coefficient of the m^{th} basis element, c_m. Written another way, Equation 3.21 becomes a continuous transform with an orthonormal basis that maps $f(x) \rightarrow T_f(m)$.

$$T_f(m) = \int_{-\infty}^{+\infty} f(x)\psi_m(x)dx \tag{3.22}$$

Now, we can insert the orthonormal wavelet basis set $\{\psi_m^n\}$ of Equation 3.16 into Equation 3.22 and get the Wavelet Transform of Equation 3.17. To reconstruct the original signal, we perform a generalized Fourier series expansion (see Equation 3.18) with the coefficients obtained with Equation 3.17 and our basis set $\{\psi_m^n\}$.

$$f(x) = \sum_m \sum_n W_f(m,n)\psi(x)_m^n \tag{3.23}$$

The next hurdle in wavelet analysis is to determine the most appropriate mother wavelet for a specific application. Presently, the appropriateness of a specific mother wavelet is determined experimentally. We first try to match the characteristic shape of the mother wavelet with the characteristics of the function under analysis. For a more complete discussion of this issue, see Fastman [14].

3.5 Multiresolution Analysis

In section 3.3, The Continuous Wavelet Transform, we said that the Wavelet Transform uses a variable length window to examine the function. Increasing window lengths correspond to successively coarser scales or resolutions (in time or space) of the function. Therefore, wavelet analysis is sometimes referred to as multiresolution analysis. In this section, we will describe each resolution level as the projection of the function onto the basis set made up of all shifts of a scaling function (not a wavelet) at a fixed dilation or scale. Multiresolution analysis represents a signal as a series of successive projections, each of which approximates the original signal at a different level of resolution [4, 30]. Here, 'level' corresponds to a particular dilation of the scaling function. A more intuitive view is that of successive low pass filtering of the signal with filters of narrower and narrower bandwidth representing the signal with less and less detail. The filter is related to, and can be derived from, the scaling function. Both views will be discussed in the following subsections.

3.6 *Multiresolution with Projections*

The projection operator $\mathbf{P}f$ projects a function f onto a basis set $\{\phi_n\}$ (see Section 3.2, Notation). For mathematical convenience we consider a scaling function $\phi(x)$ whose translations and dilations form an orthonormal basis. This is possible according to Stephane Mallat's Theorem 1 which states:

> Let $(\mathbf{V}_{2^j})_{j\in Z}$ be a multiresolution approximation of $\mathbf{L^2(R)}$. There exists a unique function $\phi(x) \in \mathbf{L^2(R)}$, called a *scaling function*, such that if we set $\phi_{2^j}(x) = 2^j\phi(2^jx)$ for $j \in \mathbf{Z}$, (the dilation of $\phi(x)$ by 2^j), then
>
> $$(\sqrt{2^{-j}}\phi_{2^j}(x - 2^{-j}n))_{n\in\mathbf{Z}} \tag{3.24}$$
>
> is an orthonormal basis of $\mathbf{V}_{2^j}$

[23:676]; see [23:690] for proof. In Mallat's theorem, $\mathbf{V}_{2^j}$ is a vector subspace of $\mathbf{L^2(R)}$ whose basis set is the *scaling function* $\phi(x)$. In being consistent with our earlier notation, where Mallat uses j to denote level or scale we use the integer m and the integer n to denote shift. One property of Mallat's set, $\{\phi_m^n\}$, is that each element is identical in shape to every other element but differs in height by a power of two and differs in relevant width by a power of two. This is known as the dyadic case. Figure 3.3 shows a rectangular scaling function dilated three times. With an orthonormal scaling function dilated and translated dyadically, we can use Mallat's discrete projection operator

$$\hat{\mathbf{A}}_{2^m} f(x) = \left(2^{-m}\sum_{n\in\mathbf{Z}}\langle f, \phi_{2^m}(\bullet - 2^{-m}n)\rangle\phi_{2^m}(x - 2^{-m}n)\right)_{m\in\mathbf{Z}} \tag{3.25}$$

which generates an approximation of the original function at a level of resolution 2^m. The set of inner products

$$\{\langle f, \phi_{2^m}(\bullet - 2^{-m}n)\rangle\}_{m,n\in\mathbf{Z}} \tag{3.26}$$

characterizes an approximation of f at scale m. In Mallat's terminology, A_{2^m} projects $f \in \mathbf{L^2(R)}$ onto the subspace V_{2^m}. For notational convenience, we now drop the subscript 2 and rewrite V_m for V_{2^m}.

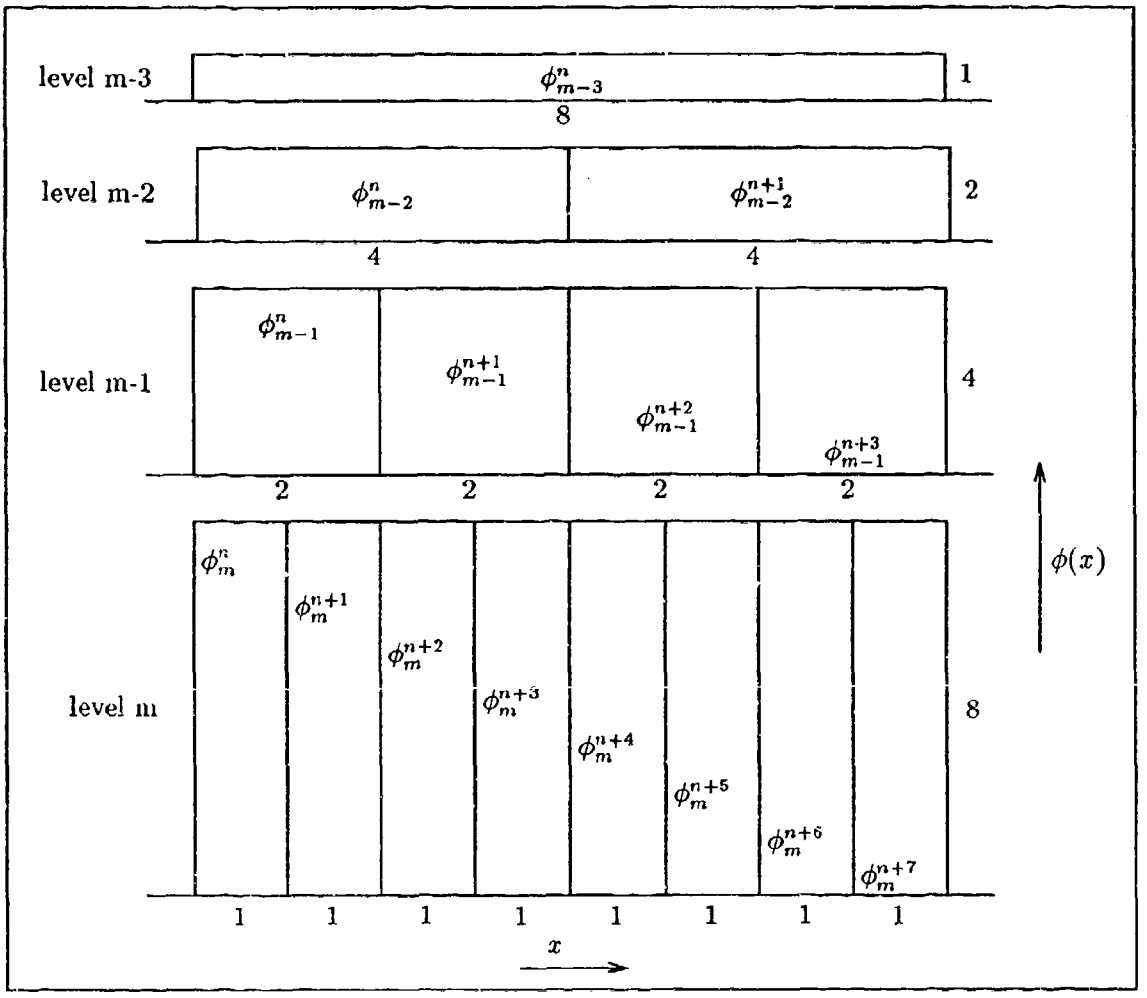


Figure 3.3. A Rectangular Scaling Function Dyadically Scaled

The family of subspaces V_m created by successively coarser approximations of $\mathbf{L^2(R)}$ has the property that

$$\cdots \subset V_{m-2} \subset V_{m-1} \subset V_m \subset V_{m+1} \subset V_{m+2} \subset \cdots \tag{3.27}$$

That is, each resolution approximation of $\mathbf{L^2(R)}$ is contained in (is a subset of) the next higher resolution approximation. Because a physical sampling device samples at a finite rate, any signal, f, is represented at its finest level of resolution by $A_{m_o}f$. For reference, we

choose $m_o = 0$. Then for a finite number of resolutions, M, we have

$$V_{-(M-1)} \subset V_{-(M-2)} \subset \cdots \subset V_{-1} \subset V_0 \tag{3.28}$$

Since $A_m f \in V_m$, each approximation of coarser resolution $A_{m-1}f$ can be derived from its parent projection of finer resolution $A_m f$.

The difference between two adjacent scales, m and $m-1$, given by

$$D_{m-1}f = A_m f - A_{m-1}f \tag{3.29}$$

is called the detail signal at scale $m-1$. It contains the details in the signal f that are lost during the projection from level m to level $m-1$. The detail signal, $D_{m-1}f$, is the result of projecting f onto the basis set of a vector space, O_{m-1}, which is orthogonal to V_{m-1}, with the projection operator D_{m-1}. Analogous to the projection Equation 3.25, this operator is described in terms of a basis set[5] ψ_m^n which spans the space O_m.

$$D_m f(x) = \left(2^{-m} \sum_{n \in \mathbf{Z}} \langle f, \psi_{2^m}(\bullet - 2^{-m}n) \rangle \psi_{2^m}(x - 2^{-m}n) \right)_{m \in \mathbf{Z}} \tag{3.30}$$

Equation 3.30 generates the difference between approximations. It is characterized by the set of inner products

$$\{ \langle f, \psi_{2^m}(\bullet - 2^{-m}n) \rangle \}_{m,n \in \mathbf{Z}} \tag{3.31}$$

This is just Equation 3.17 written as an inner product. Thus, the mother wavelet, $\psi(x)$, generates a basis set, $\{\psi_m^n\}$, of the vector space O_m. Figure 3.4 shows an example of a mother wavelet dilated and translated dyadically. It follows from Equation 3.29 that the sum of all the detail signals and the coarsest approximation equals the original signal.

$$f(x) = D_{-1}f + \ldots + D_{-(M-1)}f + A_{-(M-1)}f \tag{3.32}$$

Equation 3.32 is the Wavelet Decomposition of $f(x)$.

[5]Here, $\psi(x)$ is the particular mother wavelet associated with the scaling function, $\phi(x)$ used in Equation 3.25. Some researchers derive the ϕ given a ψ, and others derive the ψ given a ϕ [9]. In this thesis, we use previously derived ϕ,ψ pairs [23] [8].

3.7 Multiresolution with Filters

An alternate view of the multiresolution approximations is that of filtering the image with a set of low pass filters with successively narrower bandwidth. The inner products of Equation 3.26 are convolutions evaluated at the point $2^{-m}n$ (see section 3.2, Notation).

$$\langle f, \phi_{2^m}(\bullet - 2^{-m}n)\rangle = \int_{-\infty}^{+\infty} f(x)\phi_{2^m}(x - 2^{-m}n)dx = [(f * \phi_{2^m}(-\bullet)](2^{-m}n) \qquad (3.33)$$

An alternative approach uses correlations where the argument of ϕ is reversed (see section 3.2, Notation).

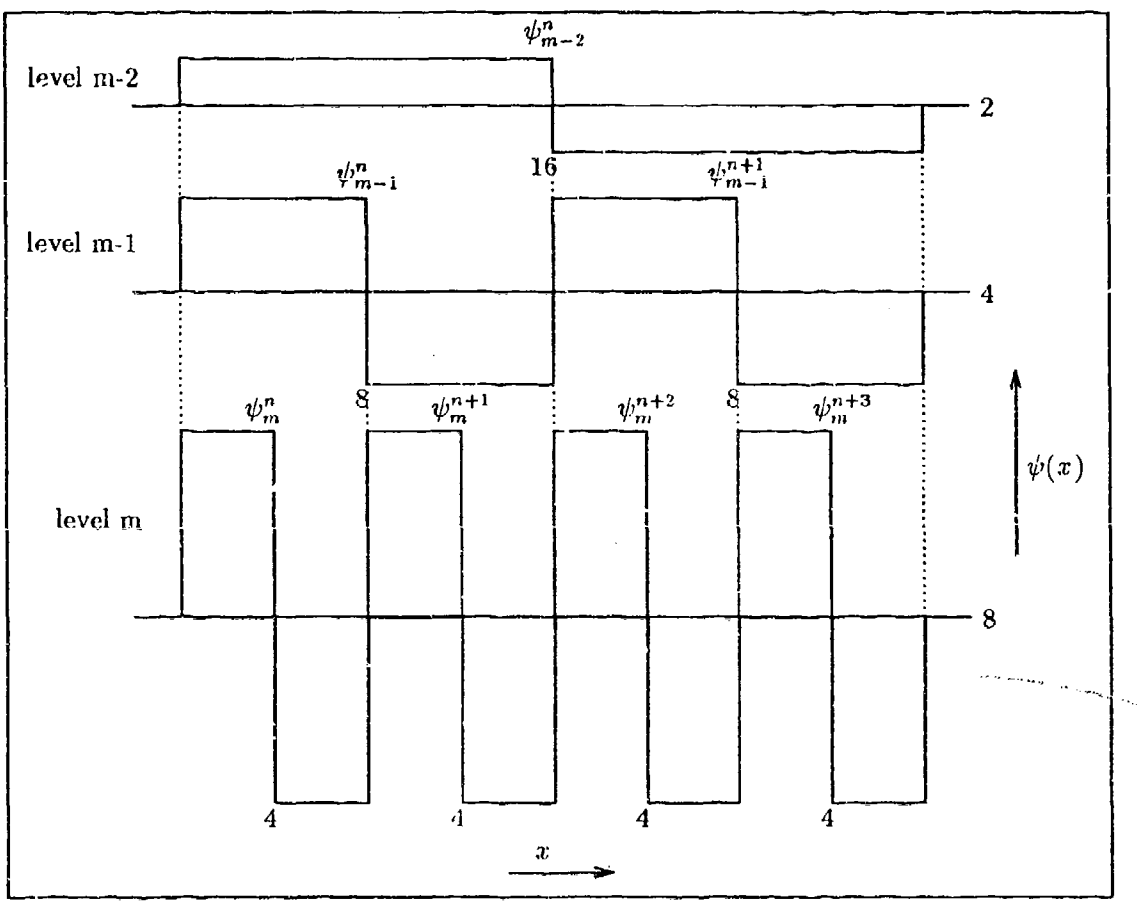


Figure 3.4. A Haar Mother Wavelet Function Dyadically Scaled

$$\langle f, \phi_{2^m}(\bullet - 2^{-m}n)\rangle = \int_{-\infty}^{+\infty} f(x)\phi_{2^m}(2^{-m}n - x)dx = [(f \star \phi_{2^m}(x)](2^{-m}n) \qquad (3.34)$$

Convolution and correlation are interchangeable. We choose convolution for consistency with current wavelet literature. Of course every good electrical engineer recognizes convolution as multiplication in the Fourier domain

$$[f * g](x) \leftrightarrow F(\omega)G(\omega) \qquad (3.35)$$

where F and G are the Fourier Transforms of f and g respectively. The Fourier Transform, $\Phi(\omega)$, of the scaling function, $\phi(x)$, is a low pass filter with a specific bandwidth. The Fourier Transform of each successively wider scaling function (dilated by a power of 2) will also be a low pass filter, but with a bandwidth smaller than that of the previous scale or level. This operation of successive low pass filtering produces "smoothed" versions or approximations of the original signal. Each version contains less information or detail than its predecessor. In the case of images, each approximation is "blurred" by the amount of high spatial-frequency information that is filtered out. Finally, the lowest or coarsest level approximation occurs when all frequencies have been filtered out and only the dc component of the signal remains.

In multiresolution analysis, we are primarily concerned with the information contained in the difference between levels of resolution. In the case of filters, the difference between two lowpass filters whose bandwidths vary by a power of two is a bandpass filter with a bandwidth of one octave. This bandpass filter is provided by the Fourier Transform, $\Psi(\omega)$, of the wavelet function, $\psi(x)$. We can express the inner products of Equation 3.31 as the convolution of the signal with the wavelet function evaluated at $2^{-m}n$ as we did in Equation 3.33.

$$\langle f, \psi_{2^m}(\bullet - 2^{-m}n)\rangle = [f * \psi_{2^m}(-\bullet)](2^{-m}n) \qquad (3.36)$$

Figure 3.5 shows a typical scaling function, $\phi(x)$, and the corresponding low pass filter, $\Phi(f)$, its Fourier Transform. Here f denotes frequency measured in Hertz, not the function f used previously. Figure 3.6 shows the wavelet function, $\psi(x)$, which corresponds to the scaling function of Figure 3.5. It also shows the bandpass filter, $\Psi(f)$, the Fourier Transform of $\psi(x)$. These filters,$\Psi(f)$ and $\Phi(f)$ correspond to the same level of resolution or scale. Superpositioning them, creates the lowpass filter of the next higher level of resolution. Similarly, adding the next bandpass filter will create the next lowpass filter and so on. Therefore, any signal or image can be decomposed into a set of signals or images each containing a

one octave bandwidth of the original signal or image. In this manner, we can construct a bank of bandpass filters from a mother wavelet for the purpose of wavelet decomposition. Furthermore, if we choose our mother wavelet to be orthonormal, the resulting bandpass filters will completely cover the frequency plane such that the information content of each signal or image in the decomposition is unique. A major advantage to the filtering approach as opposed to the projection approach is the decrease in computational time complexity of the decomposition process. Using a Fast Fourier Transform (FFT), the scale and wavelet coefficients are computed in $O(n\log(n))$ time. Alternately, using spatial convolution when the size of the filter functions are much smaller than the length of the signal $O(n)$ time is required, where n is the number of samples in the signal.

3.8 Two Dimensional (2D) Wavelet Transform

The Wavelet Transform can be extended from one dimension (1D) to n dimensions, $n > 1$. For image processing, we need a 2D Wavelet Transform to map images from the space domain to the space/spatial-frequency domain. Mallat's Theorem 1 is valid for $\mathbf{L}^2(\mathbf{R}^2)$ and there exists a scaling function $\Phi(x,y)$ whose dilations and translations are an orthonormal basis for $\mathbf{L}^2(\mathbf{R}^2)$ [22:682]. The symbol Φ is used here for consistency with referenced material and should not be confused with the Fourier Transform of ϕ denoted previously with this symbol. The $\Phi(x,y)$ can be a separable or a inseparable function. We will discuss the separable case in which $\Phi(x,y)$ is written as a product of two identical 1D scaling functions.

$$\Phi(x,y) = \phi(x)\phi(y) \tag{3.37}$$

For the separable case, the multiresolution projection approximations of the image at level m can be obtained from the following set of inner products

$$\mathbf{A}_m f(x,y) = \left(2^{-m} \sum_{n_1 \in \mathbf{Z}} \sum_{n_2 \in \mathbf{Z}} \langle f, \phi_m(\bullet - 2^{-m}n_1)\phi_m(\bullet - 2^{-m}n_2) \rangle \phi_m(x - 2^{-m}n_1)\phi_m(y - 2^{-m}n_2) \right)_{m \in \mathbf{Z}} \tag{3.38}$$

Here we use the same m and n in both x and y since we dilate and shift equally in both dimensions. However, in the more general case x and y could be shifted and dilated independently.

We obtain the detail image just as in the 1D case in Equation 3.31. The detailed image at resolution m is equal to the orthogonal projection of the 2D function on the orthonormal

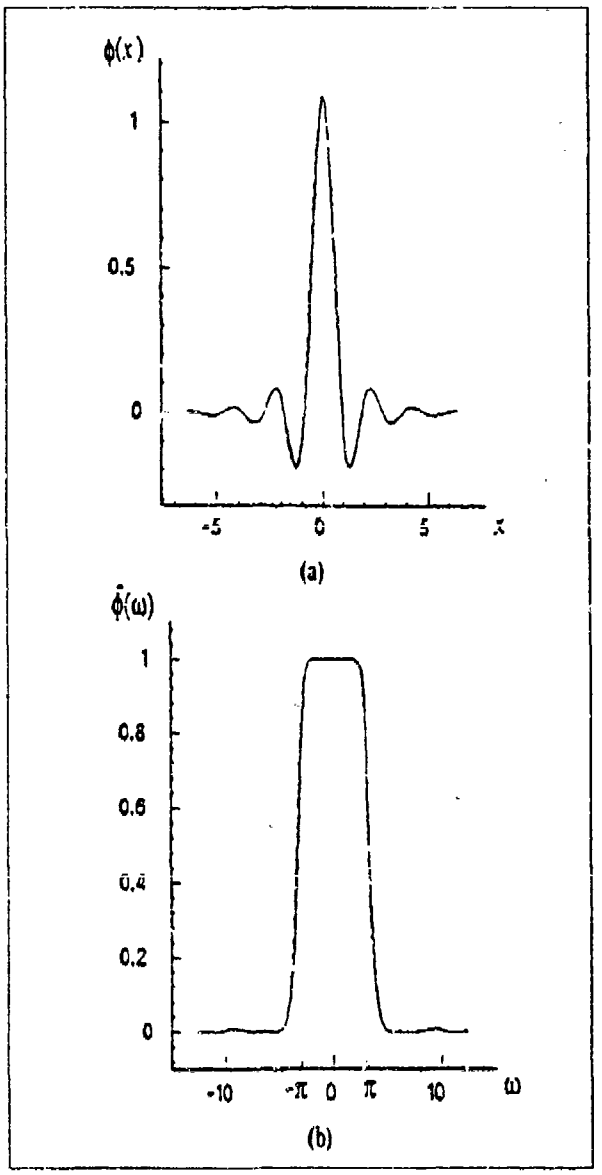


Figure 3.5. Typical Scaling Function and its Fourier Transform [23:677]

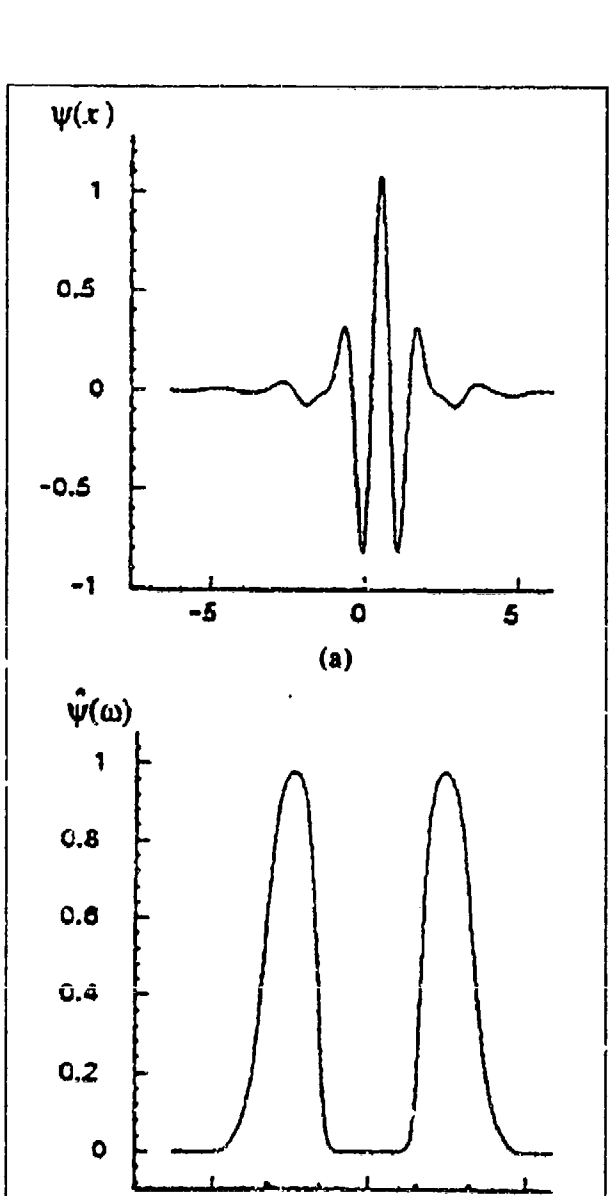


Figure 3.6. Typical Wavelet Function and its Fourier Transform [23:677]

complement, O_m, of V_m. The orthonormal basis of O_m is composed of the three wavelet basis functions $\Psi_j^1(x,y)$, $\Psi_j^2(x,y)$, $\Psi_j^3(x,y)$ which we construct from the 1D scaling function, ϕ, and its corresponding wavelet, ψ [23:683]. The symbol Ψ is used here for consistency with referenced material and should not be confused with the Fourier Transform of ψ denoted previously with this symbol.

$$\Psi_m^1(x,y) = \phi_m(x)\psi_m(y) \tag{3.39}$$

$$\Psi_m^2(x,y) = \psi_m(x)\phi_m(y) \tag{3.40}$$

$$\Psi_m^3(x,y) = \psi_m(x)\psi_m(y) \tag{3.41}$$

There is one detail projection for each of the three wavelet bases. Applying Equation 3.31 to each yields [23:684]

$$D_m^1 f(x,y) = \left(2^{-m} \sum_{n_1 \in \mathbf{Z}} \sum_{n_2 \in \mathbf{Z}} \langle f, \Psi_{2^m}^1(\bullet - 2^{-m}n_1, \bullet - 2^{-m}n_2)\rangle \Psi_{2^m}^1(x - 2^{-m}n_1, y - 2^{-m}n_2)\right)_{m \in \mathbf{Z}} \tag{3.42}$$

$$D_m^2 f(x,y) = \left(2^{-m} \sum_{n_1 \in \mathbf{Z}} \sum_{n_2 \in \mathbf{Z}} \langle f, \Psi_{2^m}^2(\bullet - 2^{-m}n_1, \bullet - 2^{-m}n_2)\rangle \Psi_{2^m}^1(x - 2^{-m}n_1, y - 2^{-m}n_2)\right)_{m \in \mathbf{Z}} \tag{3.43}$$

$$D_m^3 f(x,y) = \left(2^{-m} \sum_{n_1 \in \mathbf{Z}} \sum_{n_2 \in \mathbf{Z}} \langle f, \Psi_{2^m}^3(\bullet - 2^{-m}n_1, \bullet - 2^{-m}n_2)\rangle \Psi_{2^m}^1(x - 2^{-m}n_1, y - 2^{-m}n_2)\right)_{m \in \mathbf{Z}} \tag{3.44}$$

The image can then be completely represented at any level of resolution $m-1$ by summing $A_m f$ and $D_m^i f$ for $i = 1,2,3$. Figure 3.7 shows an approximation of the locations of the corresponding lowpass and bandpass filters for the 2D wavelet decomposition in the 2D frequency domain. This figure demonstrates the spatial orientation of each bandpass filter. The filter formed by $\Psi^1(\omega_x, \omega_y)$ is oriented horizontally, $\Psi^2(\omega_x, \omega_y)$ vertically, and $\Psi^3(\omega_x, \omega_y)$ diagonally. In many image processing applications it is desirable to obtain a representation which is not only a space/spatial-frequency representation but also is sensitive to specific orientations. Although Mallat generates three orientations as represented by the three detail signals of Equations 3.42 through 3.44, recent work by Cohen and Schlenker at AT&T Bell Laboratories suggest more are possible [5].

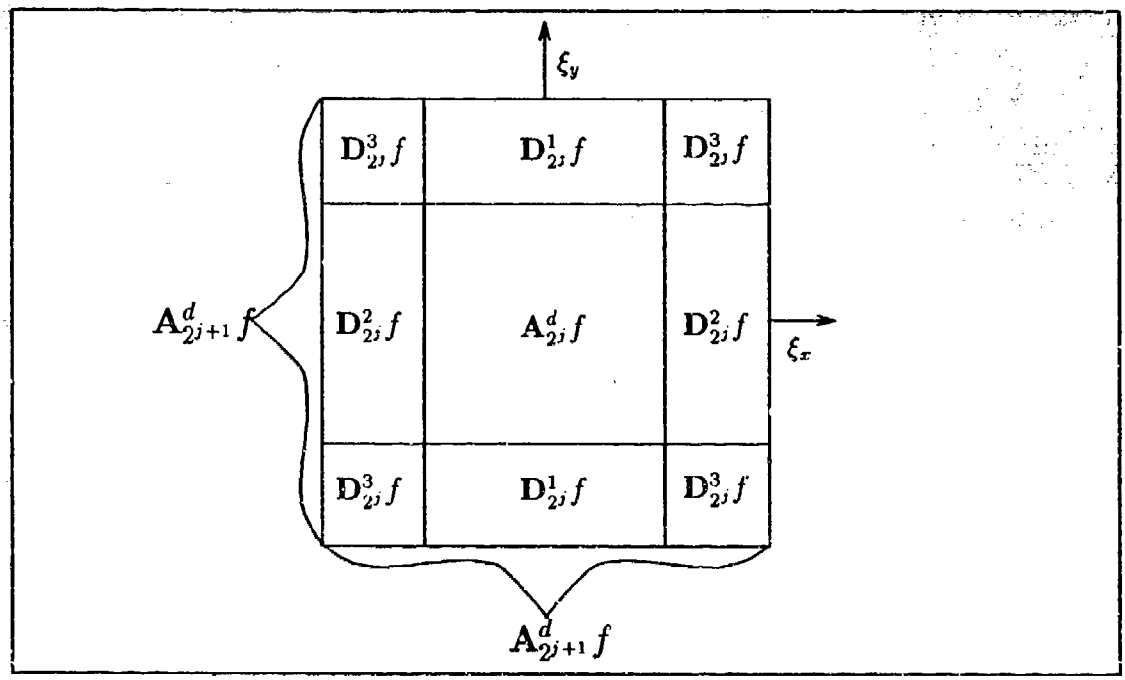


Figure 3.7. Orientation of Wavelet Decomposition Filters in the Fourier Domain [14:65]

3.9 Conclusion

The predominate tool in signal analysis for the past three decades has been the Windowed Fourier Transform. It provides a representation of signals in the time/frequency domain. However, this transform uses a constant size window; thus, it provides only a fixed resolution of the location of the frequency characteristics of a signal in the time domain. A new engineering tool, the Wavelet Transform, provides an alternative by using multiple sized windows effectively trading resolution in time for resolution in frequency for applications in which localization of frequency characteristics in time is more important for high frequencies.

IV. Multiresolution Analysis Algorithms

4.1 Introduction

This chapter discusses two different approaches to using wavelets in multiresolution analysis. It is the result of a combined effort with John (Stewart) Laing and exists in duplicate in his thesis [19]. The first approach uses the scaling function $\phi(x)$ associated with a mother wavelet $\psi(x)$ to decompose an image into successive V_m and W_m space projections where V_m and W_m are vector spaces in $\mathbf{L^2(R)}$ (see Chapter III) and are orthogonal compliments of each other in the next larger space V_{m+1}[1]. The second approach uses a set of quadrature mirror filters H and G constructed from a mother wavelet and its associated scaling function to decompose a signal or image into sets of coefficients. These coefficients characterize the V and W space projections. Following the discussion of each approach, we include implementation examples in support of the theoretical explanations.

4.2 Multiresolution with Approximations

This section discusses our implementation of multiresolution decomposition using the Haar wavelet bases. First it defines the Haar function as an orthogonal wavelet basis suitable for multiresolution decomposition. Then, it explains our implementation of decomposition. Finally, we provide an example decomposition using our decomposition program.

4.2.1 V space, W space, and Haar basis. In one dimension, the Haar mother wavelet is defined as follows:

$$\psi(x) = \begin{cases} 1 & \text{if } 0 \le x < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \le x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

The one dimensional scaling function that corresponds to the Haar mother wavelet is defined as follows.

$$\phi(x) = \begin{cases} 1 & \text{if } 0 \le x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

[1] In this chapter, the symbol W_m replaces the symbol O_m used in Chapter III, Section 3.3.

The two dimensional scaling function, $\Phi(x,y)$, is the product of $\phi(x)$ and $\phi(y)$, where $\Phi(x,y)$ is a two dimensional rectangular function. In general, Φ is scaled by an amount proportional to the length of its interval of support, I, where its values are non-zero. In the dyadic case, the length of the interval of support is given by

$$\{|I_m^n| = 2^m\}_{m,n\in\mathbf{Z}} \tag{4.3}$$

for the shift n and the level m. We use the convention that level 0 is the finest resolution level. This means that the projection in the V_0 space represents the image at its original sample density. In this case, the shift interval for the ϕ and ψ functions is

$$|I_0^n| = 1 \tag{4.4}$$

which is equal to the sample size of the image, one pixel. The scale factor is, therefore, $\frac{1}{\sqrt{2^m}}$. Now, we can write an expression for the one dimensional ϕ with the proper scale factor as follows

$$\phi_m^n(x) = \begin{cases} \frac{1}{\sqrt{2^m}} & \text{if } x \in I_m^n \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

From Equation 4.5, we build a two dimensional scaling function with the product mentioned above as follows

$$\Phi_m^n(x,y) = \begin{cases} 2^{-m} & x,y \in I_m^n \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

Therefore, our convention allows us to easily derive the size of ϕ in terms of its interval of support from 2^{-m}, where m is the level of resolution. As mentioned above, the finest resolution level corresponds to $m = 0$ and is contained in the vector space V_0. The maximum resolution level is also easily found. This is done by finding $\log_2(N)$ where N is the size of the NxN image under analysis. For example, if the image is 512x512, the largest Φ that will fit completely on the image is 512x512. Since the size of Φ is related to the level by 2^{-m}, we find m by taking $\log_2(N)$. In this example, that would be $\log_2(512) = 9$. Therefore, all contributing levels of resolution range from zero to nine, where level zero is the finest resolution and level nine is the coarsest. Though level zero is exactly the original image, we will continue to consider it for programming convenience.

The projection on the vector space V_m of the image $f(x,y)$ or the approximation of

the image at the m^{th} level of resolution is characterized by the set of coefficients, $\{c_m^n\}$ where

$$c_m^n = < \Phi_m^n, f > \tag{4.7}$$

Then, the projection is given by

$$A_m f(x,y) = \sum_n c_m^n \Phi_m^n(x,y) \tag{4.8}$$

Given that the orthogonal complement in V_{m-1} of the vector space V_m is W_m, which means that $W_m = V_{m-1} - V_m$, we can find the projection of the image onto the vector space W_m from Equation 3.29. It is possible to calculate the wavelet coefficients, d_m^n, that characterize the projection into the orthogonal vector space W_m in a manner similar to Equation 4.7 using

$$d_m^n = \langle \Psi_m^n, f \rangle \tag{4.9}$$

where $\Psi(x,y) = \psi(x)\psi(y)$ But this is not necessary since we can find the projections $D_f(m)$ more directly from Equation 3.29

4.2.2 Haar Transform Program The data flow diagram in Figures 4.1 and 4.2 shows the operation of the Wavelet Decomposition program, *wave*. This program, is written in the ANSI standard C programming language. It reads in an image from an ASCII file and writes its output to ASCII files; the Φ coefficients, the projections in V space, and the projections in W space. The number of files produced is determined by the size of the input image to be decomposed. For example, the image of Lenna shown in Figure 4.3 has a resolution of 480x512 pixels. Therefore, ten files each will be produced for the Φ coefficients, the V space projections, and the W space projections. The Φ coefficients are calculated by taking the inner product of the appropriate level Φ and the image, Equation 4.7. The projections of the input image onto the V space are found by multiplying the Φ basis by the Φ coefficients, Equation 4.8. Then, the projections in the W space are found from the difference of V space projections at adjacent levels, Equation 3.30. The source code for the *wave* program is made up of ten files. They are provided in their entirety in Appendix A.2.

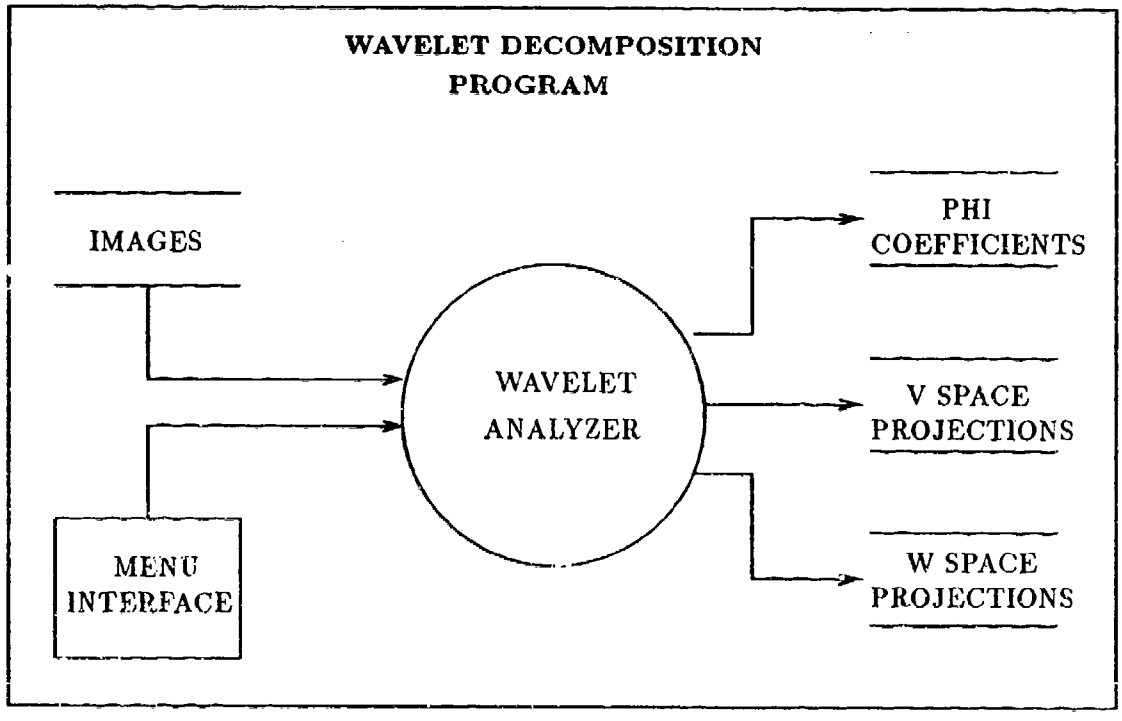


Figure 4.1. Dataflow Diagram of the Wavelet Decomposition Program, First Level

4.2.3 An Example Decomposition We subjected a 480x512 sampled image of Lenna to the Haar transform program and printed her projections in the V spaces and the W spaces for resolution levels one through nine according to the convention established above (See Figure 4.4 through 4.15).

The W space projections are made viewable by adding 255 to their gray scale values and dividing the sum by two. This process centered the values about 128 instead of zero. The low energy contained in the W space projections is as expected, since it represents only that part of the image which correlates to the ψ of the corresponding level. In other words, only small amounts of the whole image lie in the scale bandwidth of the corresponding scale of ψ at that level of resolution. The projection onto $W_1 = V_0 - V_1$ space showed only the high frequency information, changes that occurred within the Haar interval of support or a 2x2 pixel area. This is seen in Figures 4.10 through 4.15 in which six projections onto the W spaces are shown. On the other hand, the V space projections get progressively blurrier with larger m, corresponding to coarser levels of resolution. They represent all frequencies

of the image from the dc component, V_9, to the current level. All V space projections of coarser resolution are contained in a V space projection of finer resolution, smaller m (See Figure 4.4 through 4.9).

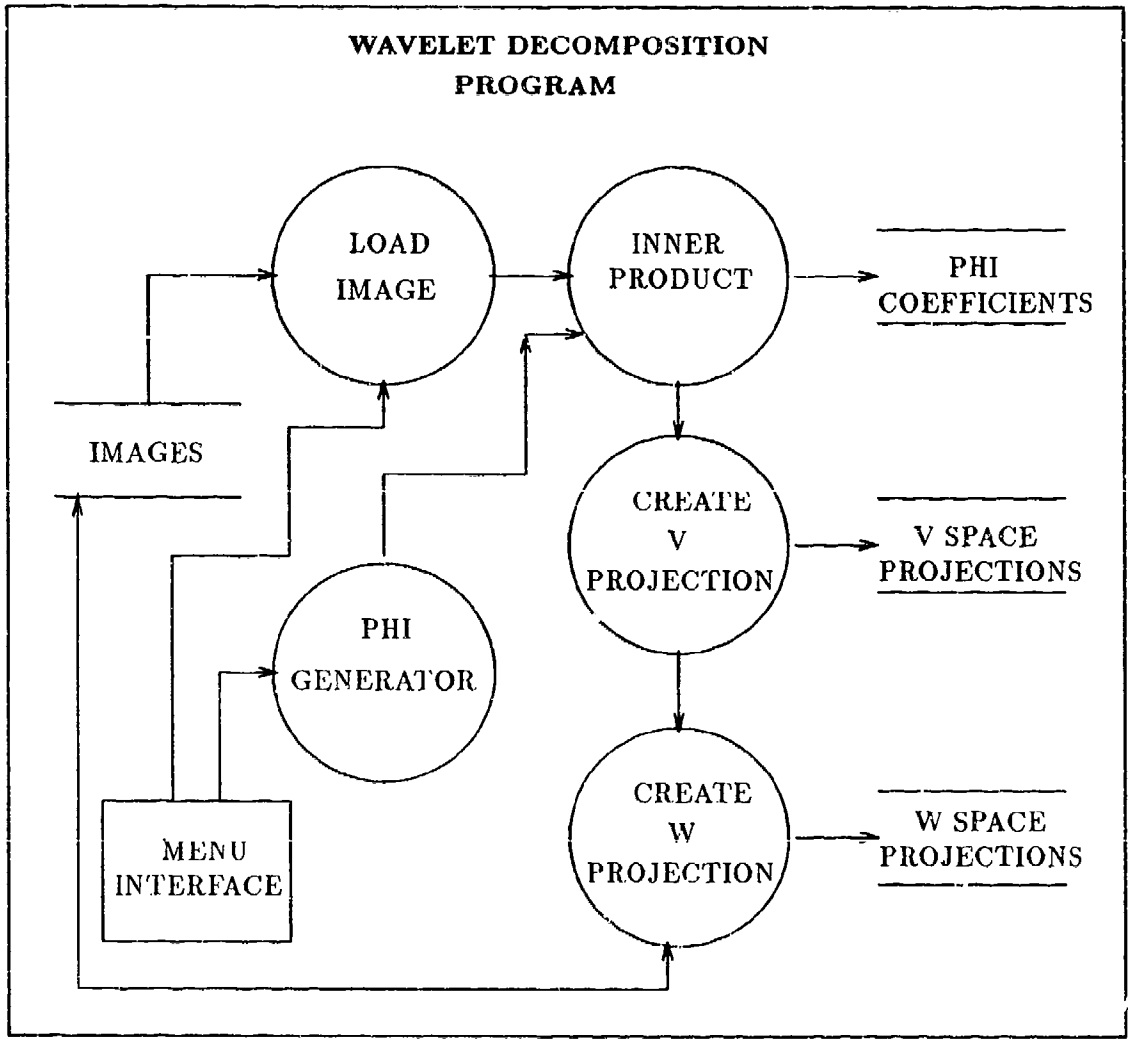


Figure 4.2. Dataflow Diagram of the Wavelet Decomposition Program, Second Level

Figure 4.3. Projection of Lenna onto V_0

Figure 4.4. Projection of Lenna onto V_1

Figure 4.5. Projection of Lenna onto V_2

Figure 4.6. Projection of Lenna onto V_3

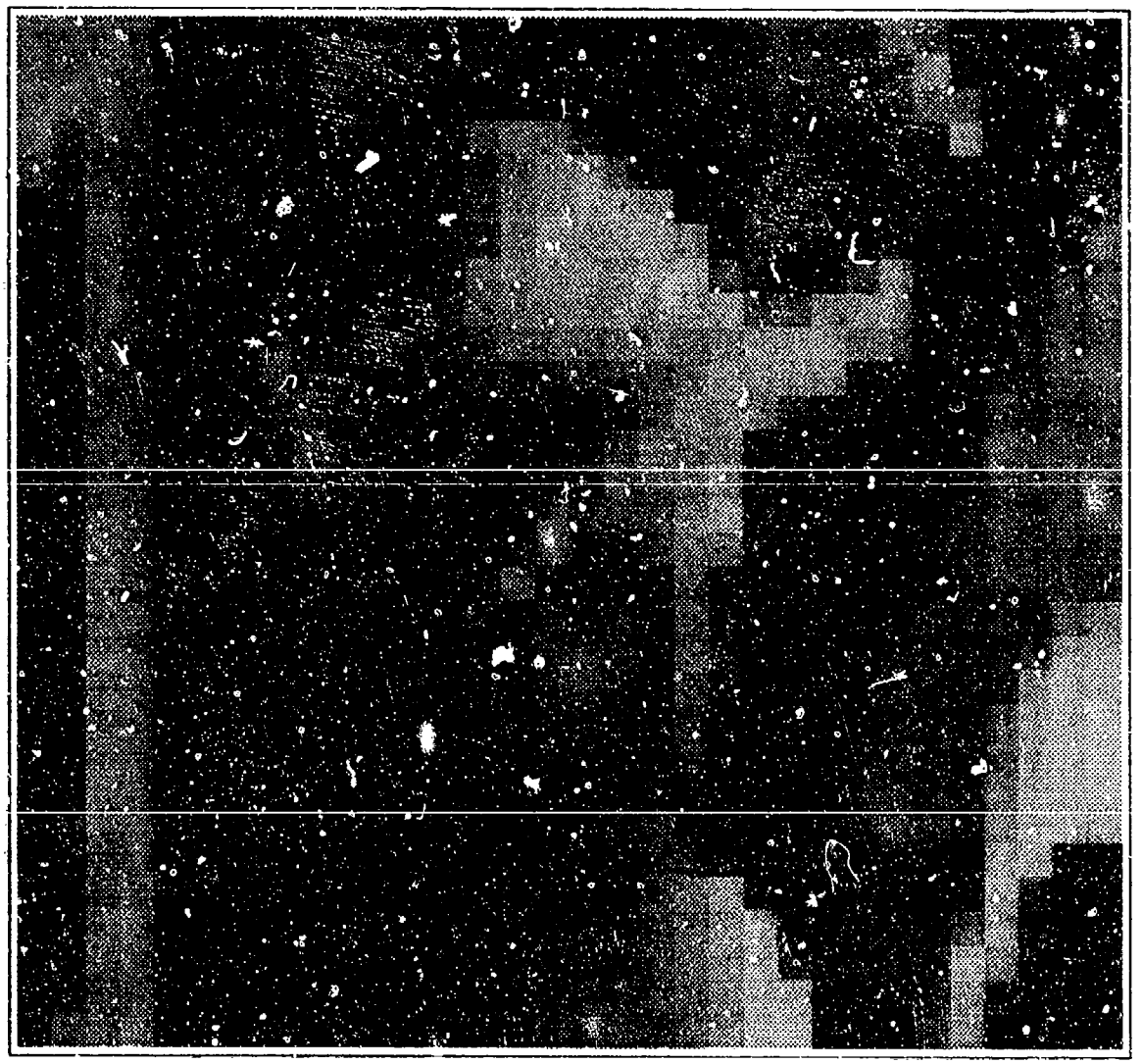


Figure 4.7. Projection of Lenna onto V_4

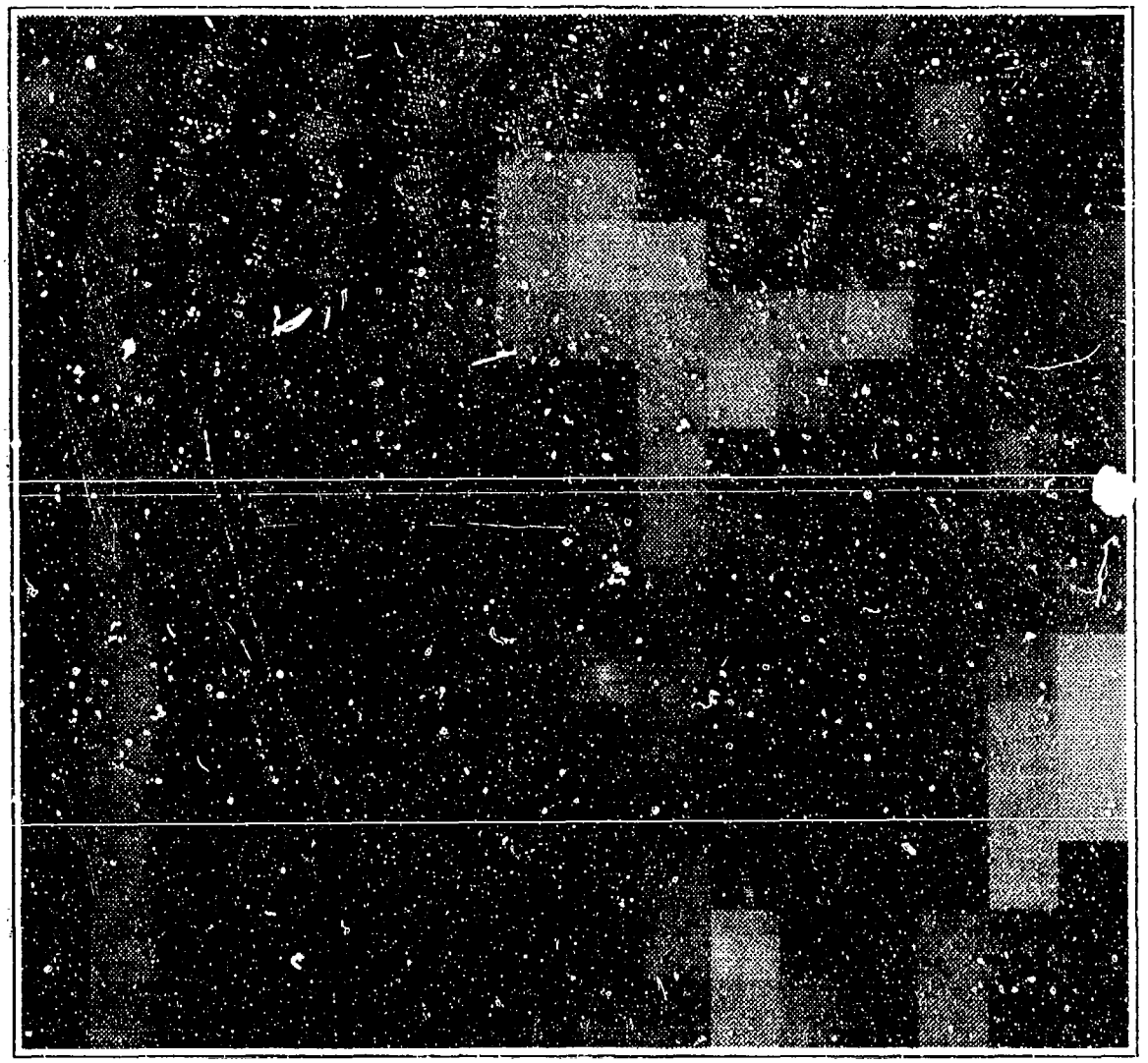


Figure 4.8. Projection of Lenna onto V_5

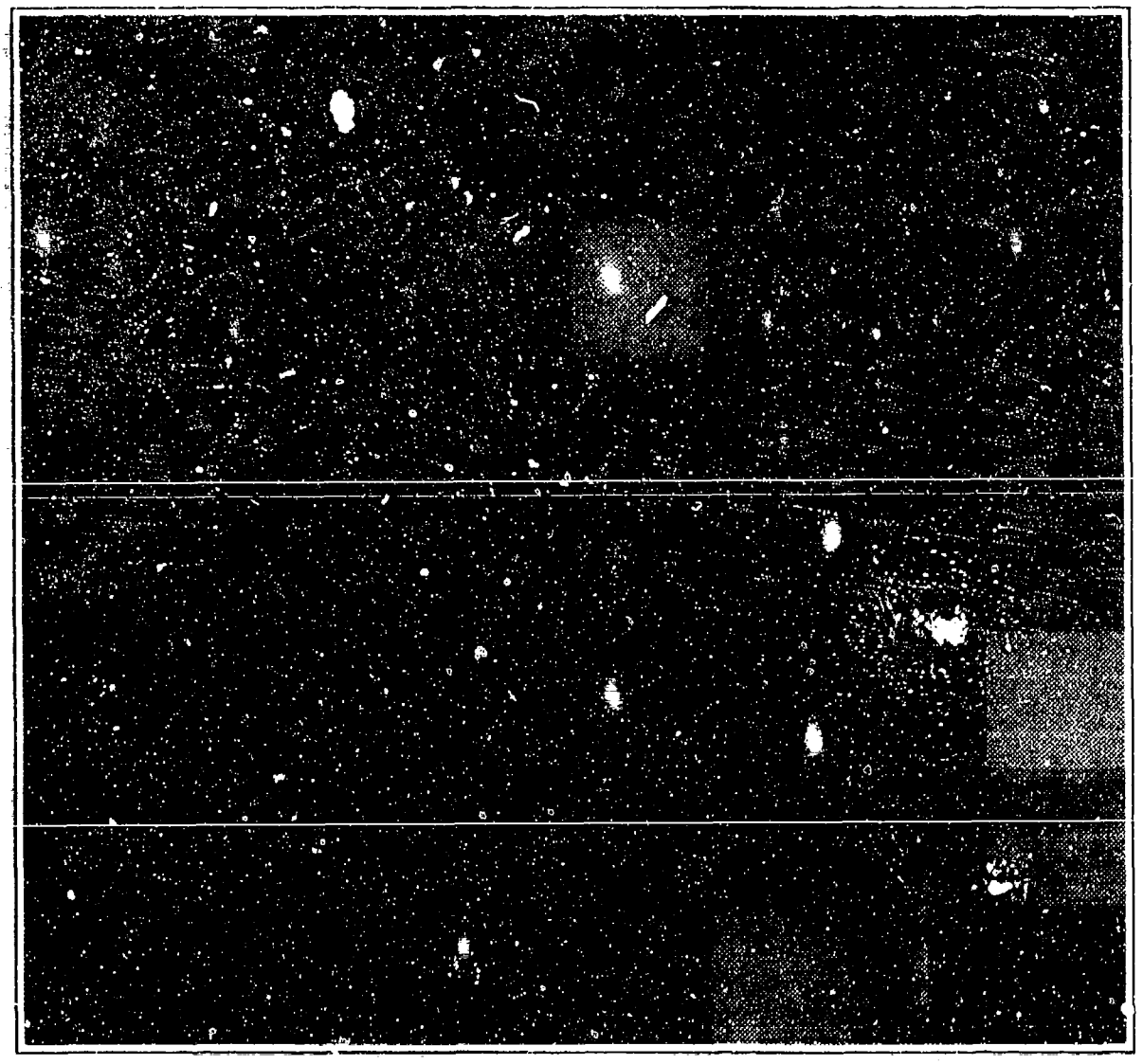


Figure 4.9. Projection of Lenna onto V_6

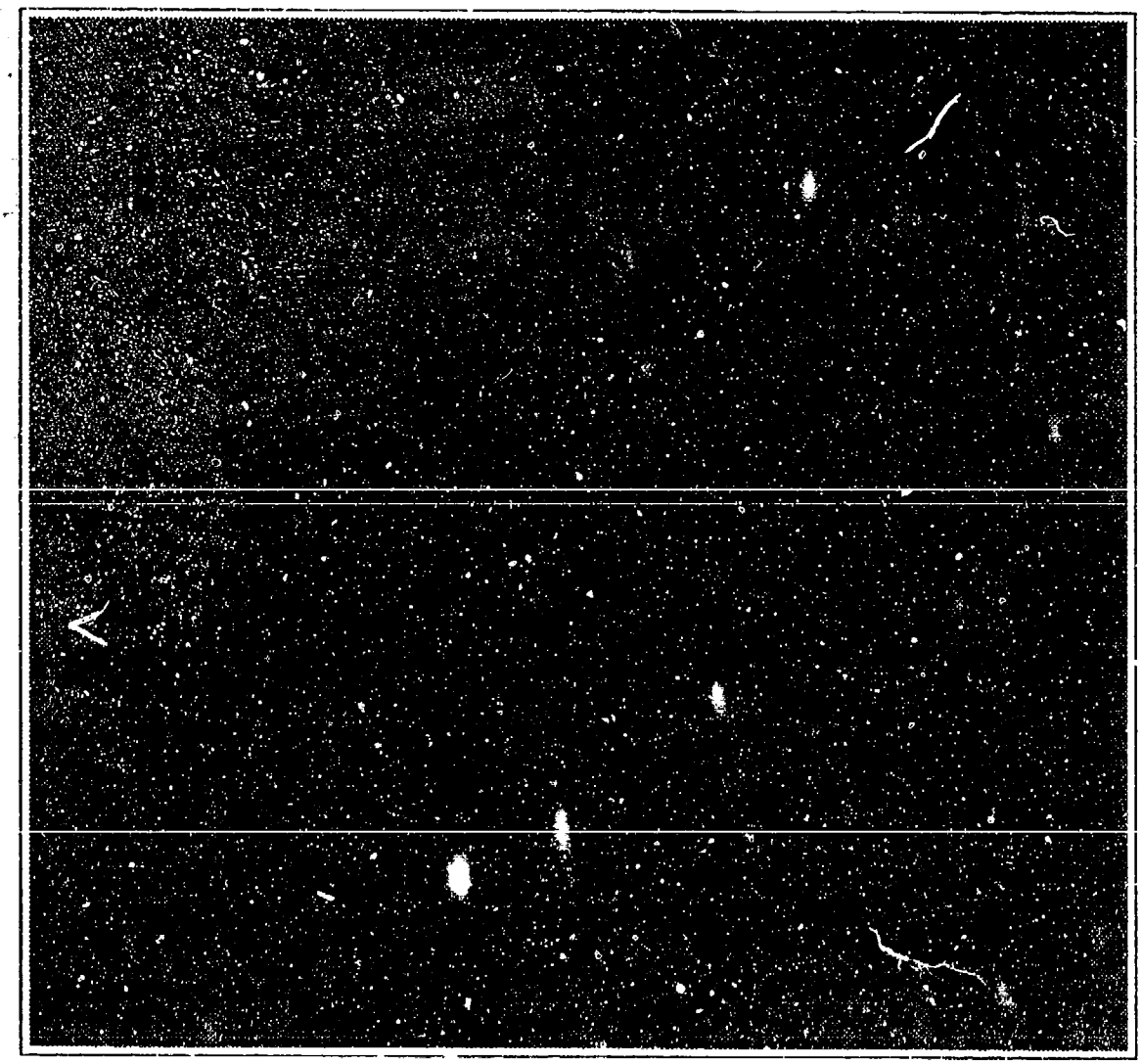


Figure 4.10. Projection of Lenna onto W_1

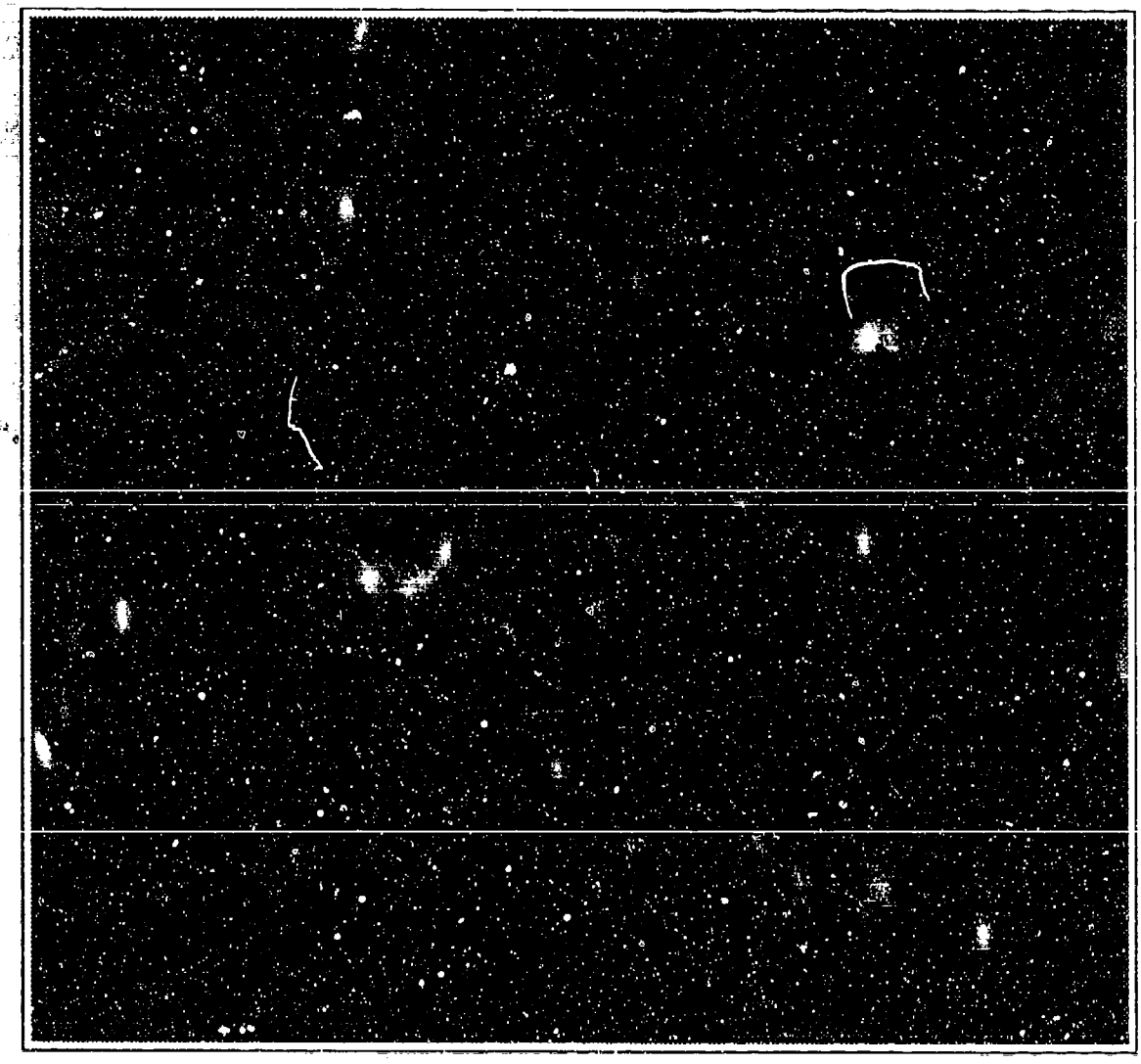


Figure 4.11. Projection of Lenna onto W_2

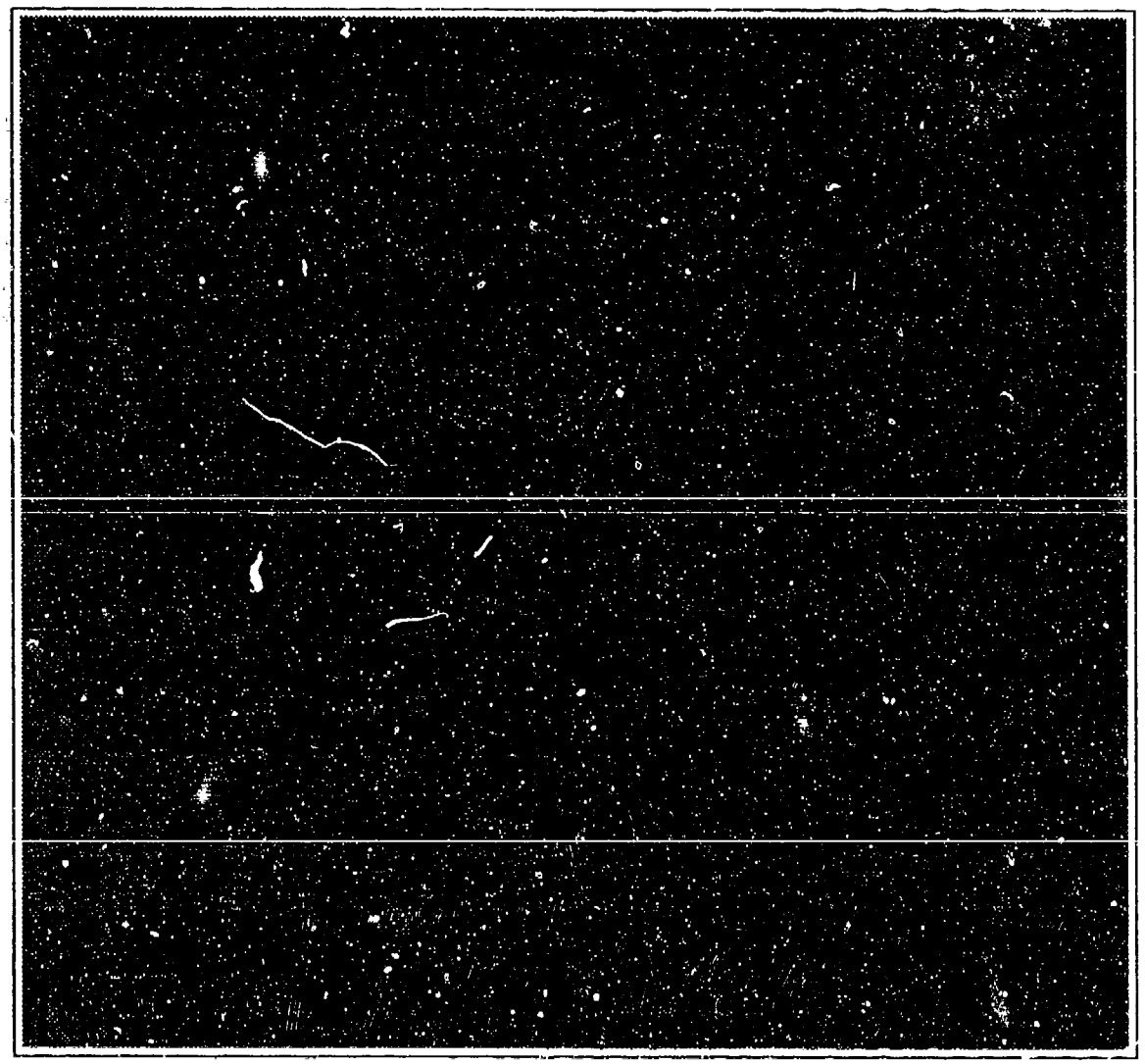


Figure 4.12. Projection of Lenna onto W_3

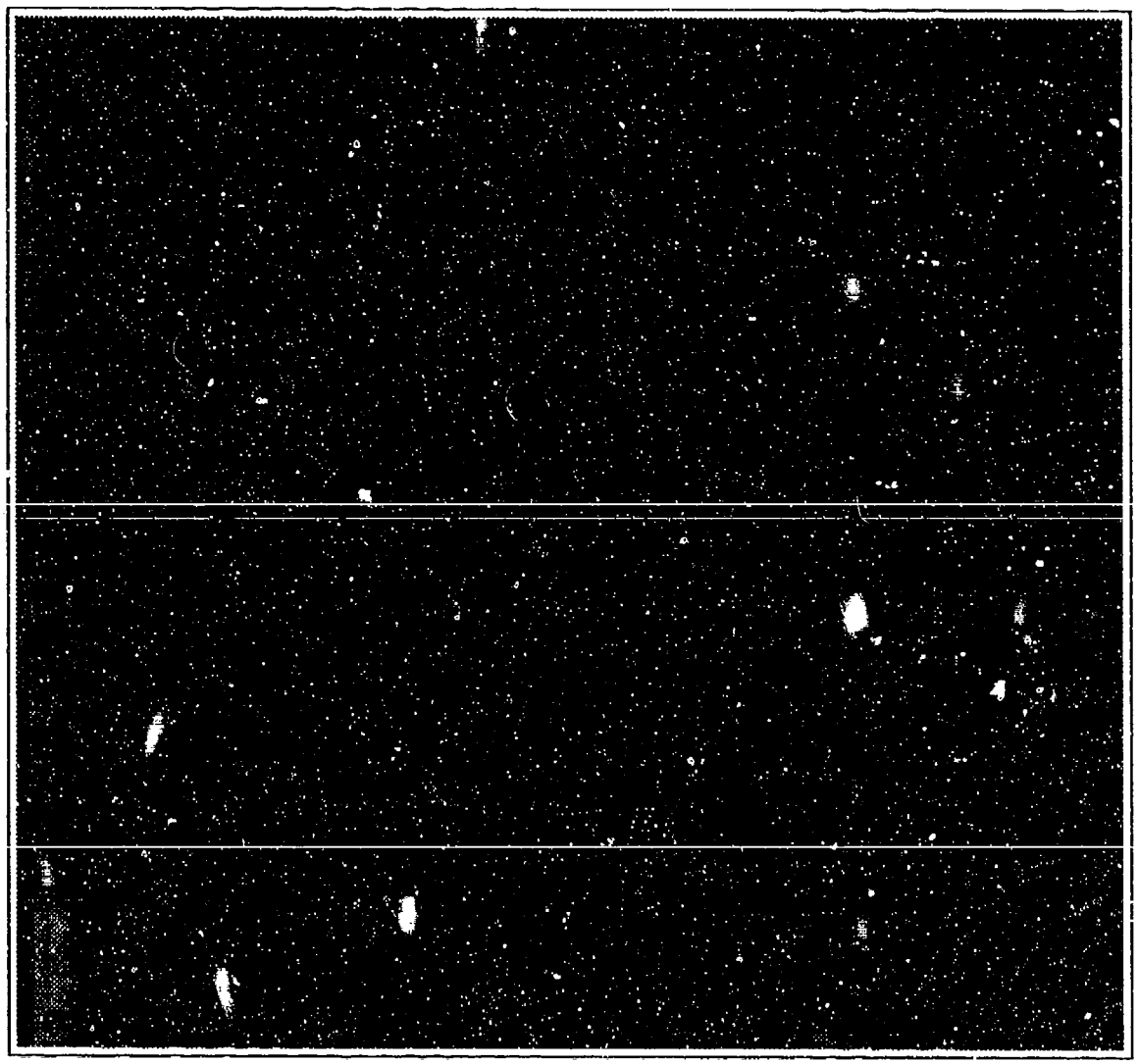


Figure 4.13. Projection of Lenna onto W_4

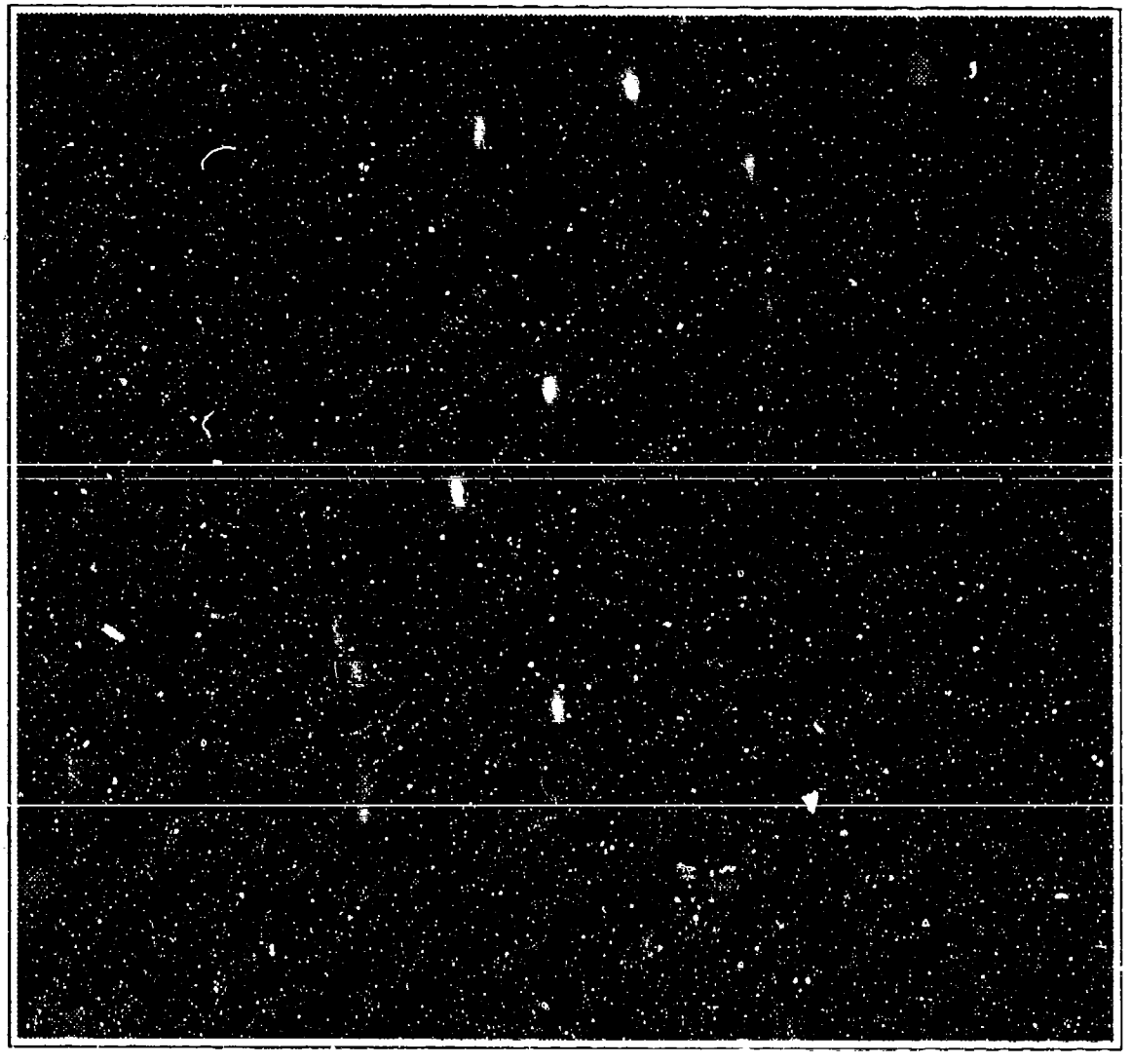


Figure 4.14. Projection of Lenna onto W_5

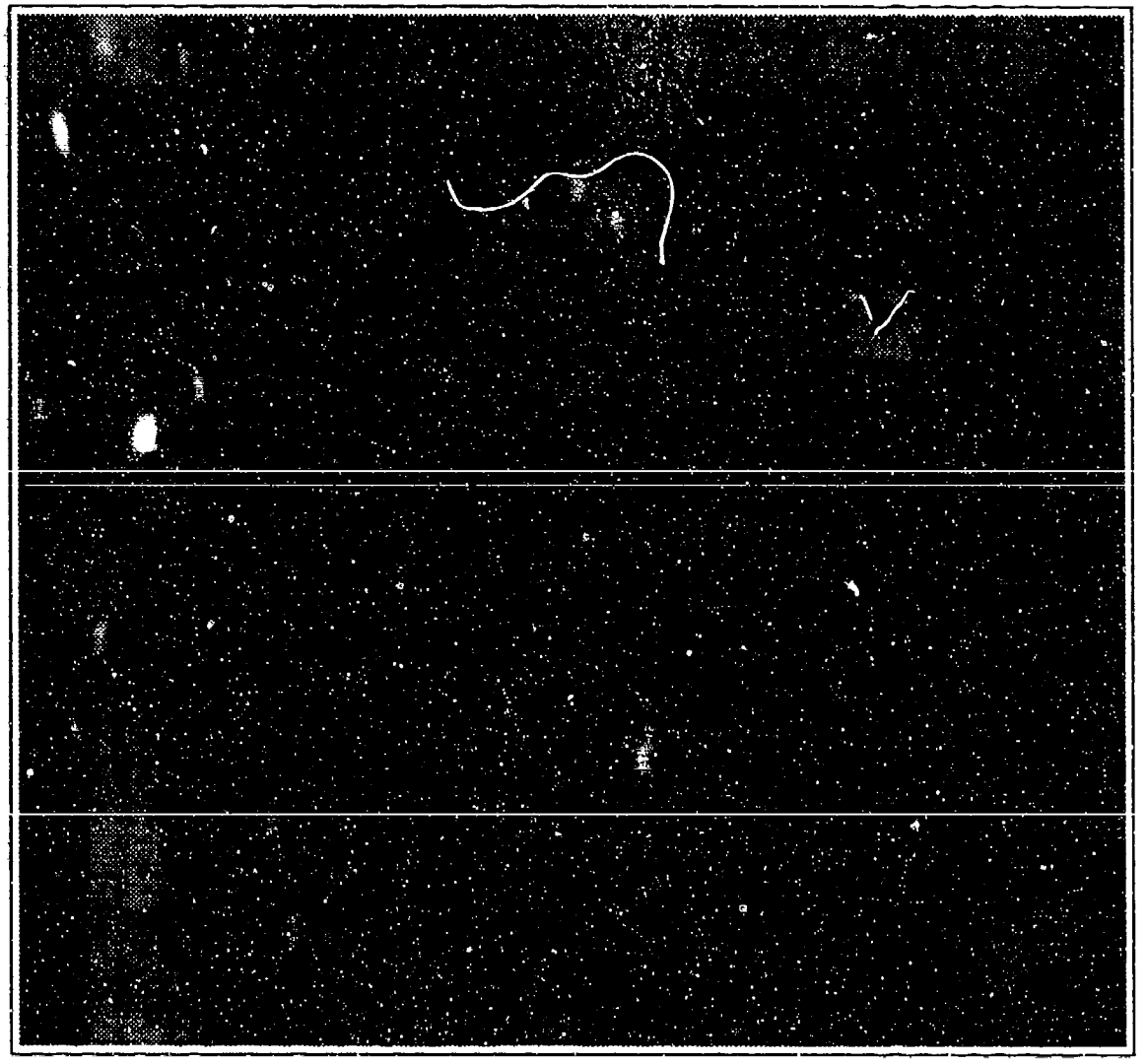


Figure 4.15. Projection of Lenna onto W_6

4.2.4 Histograming To view the histogram of grey scale values of the projected images, the Khoros signal and image processing system developed at the University of New Mexico [31]. Figures 4.16 and 4.17 show the resulting histograms of the original Lenna image and the first three levels of the V and W projections. These results show how the V space projections contain a wide variety of grey scale levels compared to the W projections. Therefore, the W space projections would be a good choice of representation from which to code and compress the original image.

4.2.5 Thresholding The histograms discussed above provide a good measure of the grey scale values that are important to the information content of the image. For example, the histogram of the W_1 projection shown in Figure 4.10 shows that most of the information content of the image, the essence of Lenna, is contained in a relatively small number of pixels in a small range of grey scale values to either side of grey scale value 128. To isolate this information from the vast amount of data required to represent the entire 512x512 image, we developed a routine called *threshold* to eliminate or zero out the large number of pixels in the grey scale range around the value 128. Our routine also binarizes the remaining grey scale values. If a grey scale value falls within the thresholding window, it is set to white or 255, and if a grey scale value is outside the threshold window, it is set to black or 0. Figures 4.18, 4.19, and 4.20 shows the results of executing the *threshold* program on the first three levels of W space projections. These figures demonstrate the edge detection capability of a Multiresolution Wavelet Decomposition. These images were produced by chosing to eliminate all grey scale values between 131 and 125. The *threshold* routine, whose source code is listed in the Appendix F.2, allows the user to select the upper and lower bounds of grey scale values for thresholding.

4.3 Multiresolution with Filters

This section briefly reviews Mallat's multiresolution approximation algorithm [23:677]. It also expands on selected areas of his paper that are vague or incorrect. Because the theory of multiresolution analysis is covered in Chapter II of this thesis, we begin here with the specifics of Mallat's algorithm. The specific equations referenced in this section are taken directly from Mallat's paper [23].

4.3.1 Multiresolution Decomposition In Mallat's Equation (10) [23:677], he gives the "orthogonal projection" of a signal $f(x)$ onto a scale space, V, of an arbitrary level of

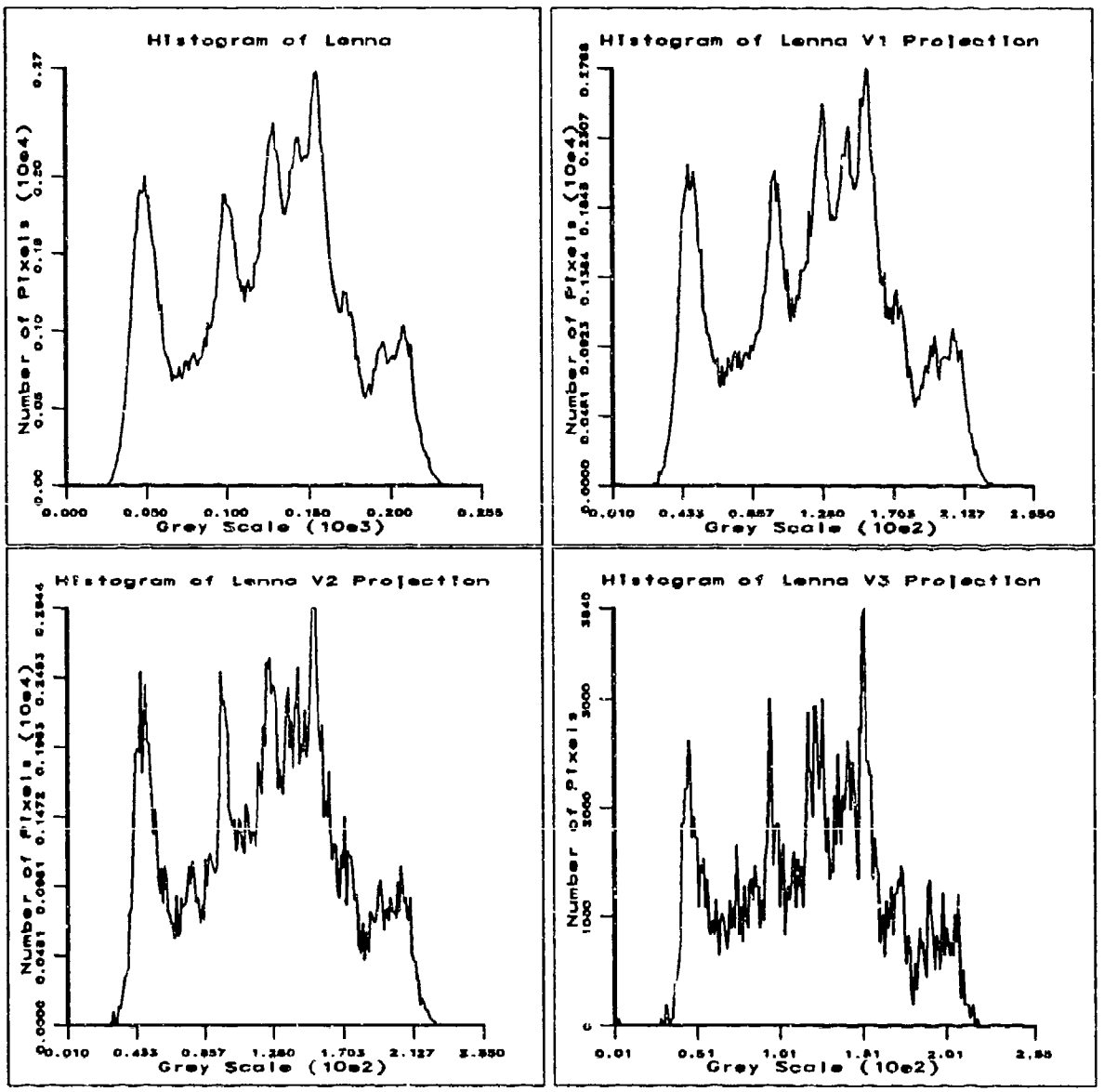


Figure 4.16. Histograms of Lenna's Original Image and V_1 through V_3 Projections

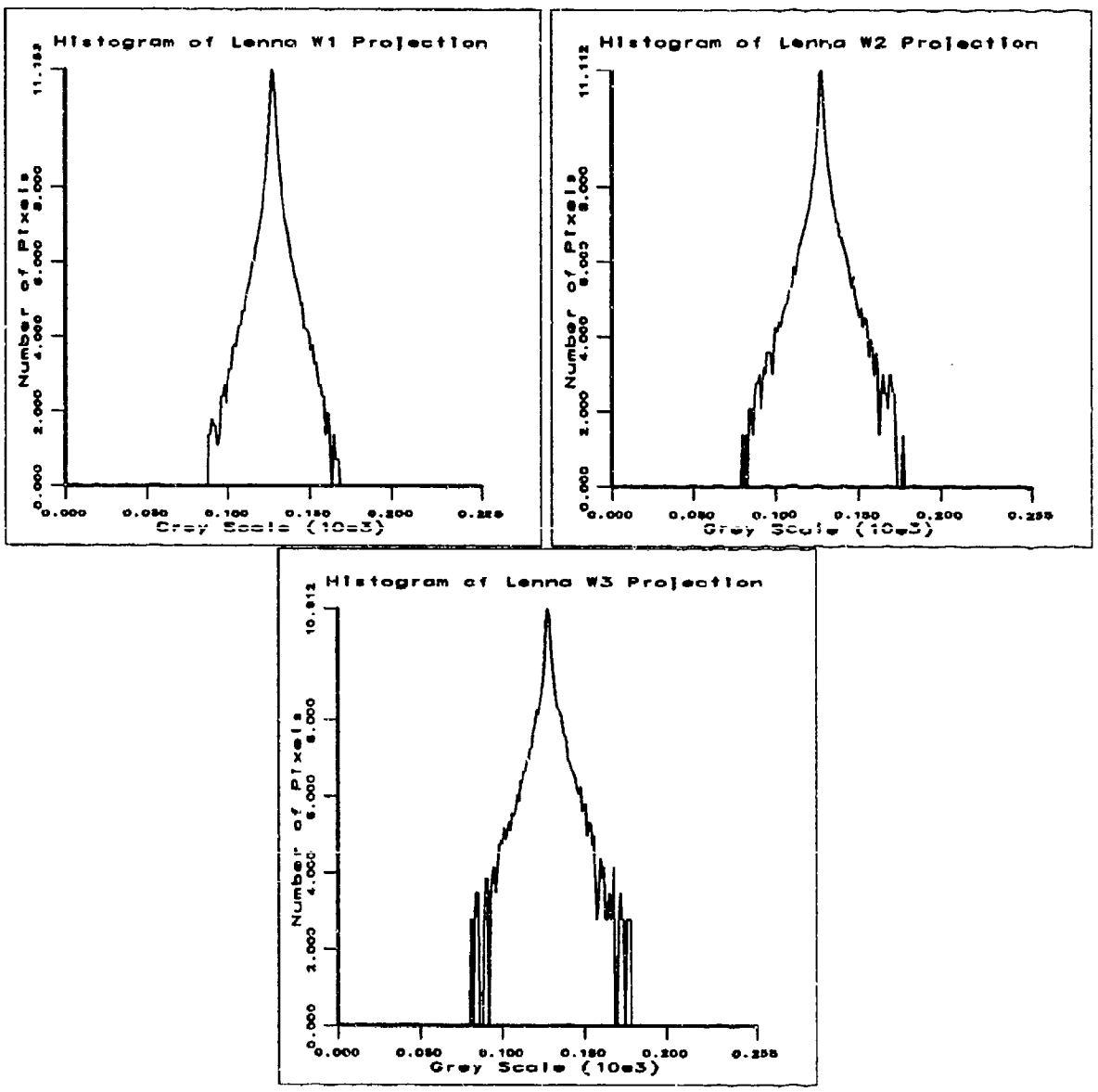


Figure 4.17. Histograms of Lenna's W_1, W_2, and W_3 Projections with the Number of Pixels Logged

Figure 4.18. Lenna's W_1 Projection Thresholded

Figure 4.19. Lenna's W_2 Projection Thresholded

Figure 4.20. Lenna's W_3 Projection Thresholded

resolution, 2^j for $j \in \mathbf{Z}$ as

$$\mathbf{A}_{2^j} f(x) = 2^{-j} \sum_{n=-\infty}^{+\infty} \langle f, \phi_{2^j}(\bullet - 2^{-j}n) \rangle \phi_{2^j}(x - 2^{-j}n), \forall f \in \mathbf{L}^2(\mathbf{R}) \tag{4.10}$$

Then in Equation (11) [23:677], he adds a superscript d to his notation indicating that the inner product of this equation is a "discrete approximation" of $f(x)$ at the given level of resolution. Mallat's Equation (11) is just that inner product.

$$\mathbf{A}^d_{2^j} f = \{\langle f, \phi_{2^j}(\bullet - 2^{-j}n) \rangle\}_{n \in \mathbf{Z}} \tag{4.11}$$

The discrete set of inner products in Equation 4.11 is the set of scaling function coefficients previously given in this thesis in Equation 4.7 as c^n_m where n corresponds to Mallat's n and m corresponds to his j. From this point on in his paper, Mallat refers to this set of inner products as "the image". While his explanation is easy to miss, it is true that he treats a discretely sampled signal or image as being equivalent to these coefficients at the finest level of resolution without ever taking the inner product. In other words, he considers the sampling process of the original analog signal or image to be an approximation of that signal or image at the finest level of resolution, sample density, allowable by the sampling device (ie. digitizer or scanner). He treats this set of samples as equivalent to the scaling function coefficients at the finest level of resolution, $j = 0$. We have adopted his convention, but include here a brief explanation that considers the digitally sampled signal or image as the projection of the original analog signal or image onto the scale space, $\mathbf{V}_{2^j}$, where $j = 0$ as the finest level of resolution corresponding to the sample density of our input data. This approach would add two steps to Mallat's algorithm — one at the beginning to perform the inner product with $\phi_{2^0}(x - n)$ and one at the end to perform the discrete sum that projects the reconstructed scaling function coefficients onto the scale space at level $j = 0$. Performing the inner product of Equation 4.11 via convolution the level $j = 0$ scale coefficients are

$$\mathbf{A}^d_1 f = \{(f * \phi_1(-\bullet))(n)\}_{n \in \mathbf{Z}} \tag{4.12}$$

for one dimension and

$$\mathbf{A}^d_1 f = \{(f * \phi_1(-\bullet) * \phi_1(-\bullet))(n, m)\}_{n,m \in \mathbf{Z}} \tag{4.13}$$

for two dimensions. Obtaining the scale space projection from these coefficients at the end of reconstruction is just as straight forward if we think of $\phi(x)$ as a discretely sampled function with k samples. For illustration, replace the continuous variable x with the discrete variable k. Then, inserting Equation 4.12 into projection Equation 4.10 yields

$$\mathbf{A}_1 f(k) = \sum_{n=-\infty}^{\infty} (\mathbf{A}_1^d f)(n)\phi_1(k-n) \tag{4.14}$$

which is the rectangle approximation of the Riemann integral of the convolution

$$((\mathbf{A}_1^d f)(n) * \phi_1(n))(k) \tag{4.15}$$

Using Equation 4.15 as the final step in our multiresolution reconstruction program, we obtain the discrete multiresolution approximation of the original signal. The two dimensional form of Equation 4.15 using the discrete variables k and l in place of the continuous variables x and y respectively is

$$\mathbf{A}_1 f(k,l) = ((\mathbf{A}_1^d f)(n,m) * \phi_1(n) * \phi_1(m))(k,l) \tag{4.16}$$

Because these extra steps add no additional accuracy to Mallat's multiresolution analysis algorithm, we omit them as he did. However, their explanation provides a clearer transition from the theory discussed earlier in this thesis to the implementation described in this chapter.

In his Equation (15) [23:677], Mallat introduces the "discrete filter", H, "whose impulse response is given by", $h(n)$. In this thesis, we will refer to $h(n)$ as a *response function* and refer to H as a *filter*. Mallat shows in the one dimensional case that the set of scale coefficients $\mathbf{A}_{2^j}^d f$ at resolution level j can be found by convolving the response function $h(n)$ with the set of scale coefficients $\mathbf{A}_{2^{j+1}}^d f$ at the previous level of resolution $j+1$ and evaluating the result at even values of the argument n. Our interpretation of his Equation (16) [23:678] is

$$\mathbf{A}_{2^j}^d f = \{(\mathbf{A}_{2^{j+1}}^d f * \tilde{h})(2n)\}_{j,n\in\mathbf{Z}} \tag{4.17}$$

where $\tilde{h}(n) = h(-n)$. After this point, Mallat frequently uses the upper and lower case 'H' interchangeably even though the operation clearly calls for a space domain convolution, not a convolution in the frequency domain. Equation 4.17 describes the decomposition of a set

of scale coefficients at level $j+1$ into the set of scale coefficients at the next coarser level of resolution j. The detail that is lost in the multiresolution transformation is described by the wavelet coefficients which are in Mallat's notation $\mathbf{D}_{2^j} f$. These coefficients are found by way of a similar multiresolution transform using another filter, G, whose response function is $g(n)$. This transform is given by Mallat's equation (28) [23:681] and is interpreted as

$$\mathbf{D}_{2^j} f = \{(\mathbf{A}^d_{2^{j+1}} f * \tilde{g})(2n)\}_{j,n,k \in \mathbf{Z}} \tag{4.18}$$

where $\tilde{g}(n) = g(-n)$. The filters G and H have the following relationship [23:681]

$$g(n) = (-1)^{1-n} h(1-n) \tag{4.19}$$

Notice that the $h(n)$ and $g(n)$ are reflected about $n = 0$ and shifted relative to each other. Even though the convolution operation occurs for all shifts, it is very important to maintain the relative shift of $g(n)$ with respect to $h(n)$. In other words, these response functions must be defined to have a relative offset of one, as shown in Equation 4.19, for whatever convolution routine is used.

Now, armed with a set of response functions, $h(n)$ and $g(n)$, Equations 4.17 and 4.18 can be implemented iteratively to decompose the scale coefficients of a signal at the finest level of resolution into the scale coefficients and detail coefficients at each level of resolution. Because the number of scale coefficients diminishes by a power of two at each iteration, the extent of this decomposition is limited by the size of the response functions. For example, a signal, $f(x)$, with 128 discrete samples decomposed with response functions, $h(n)$ and $g(n)$, that have 11 samples each can produce scale and detail coefficients, $\mathbf{A}^d_{2^j} f$ and $\mathbf{D}_{2^j} f$, for four levels of resolution. At the fourth level, the scale coefficient contains only eight elements which is not enough to meaningfully convolve with the eleven element response functions.

The response function $h(n)$ and its lowpass filter H that correspond to the cubic spline mother wavelet of Figure 3.6 are shown in Figure 4.22. Using Equation 4.19, we derived the response function $g(n)$ from $h(n)$. It is plotted along with its highpass filter G in Figure 4.22. From these plots, it is apparent that H is a low pass filter which smooths the signal and G is a high pass filter which captures the details lost in the smoothing process. The algorithm given by Equations 4.17 and 4.18 is diagramed in Figure 4.21 which is redrawn from [23:681].

4.3.2 Two Dimensional Multiresolution Decomposition The two dimensional case is a natural extension from one dimension. Equations 3.38, 3.42, 3.43, and 3.44 give the scale and detail coefficients. These correspond to Mallat's Equations (39) through (40) [23:684]. Our interpretation of these equations when the response functions $h(n)$ and $g(n)$ are incorporated is as follows:

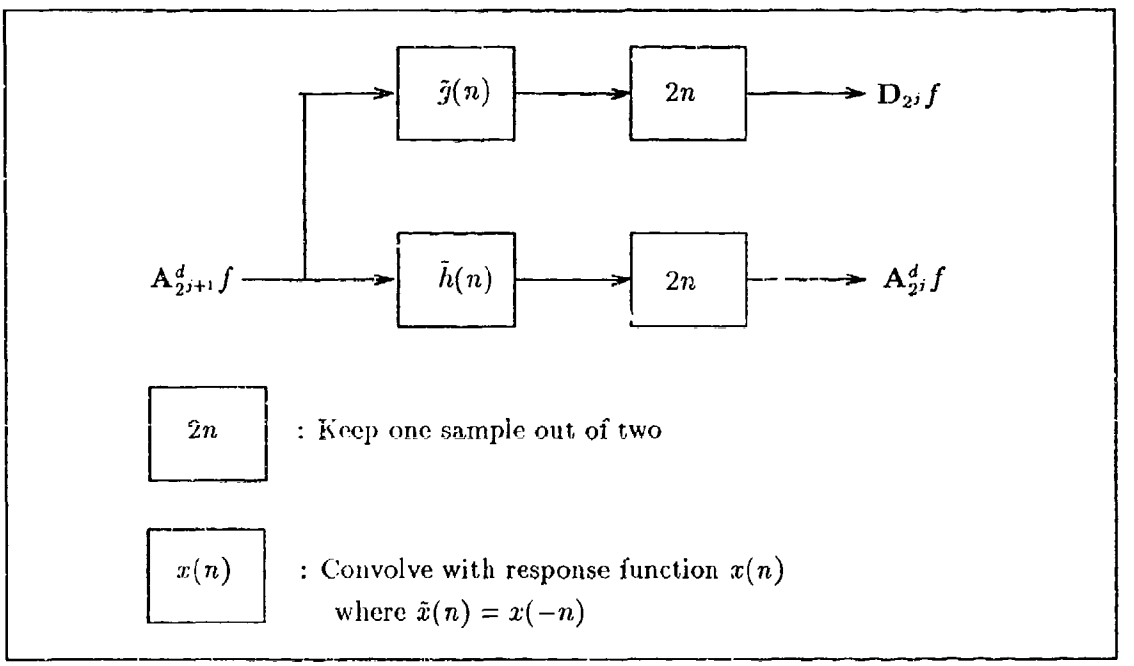


Figure 4.21. One Dimensional Multiresolution Decomposition [23:681]

$$\mathbf{A}^d_{2^j} f = (\mathbf{A}^d_{2^{j+1}} f)(k,l) * h(k) * h(l))(2n, 2m) \tag{4.20}$$

$$\mathbf{D}^1_{2^j} f = (\mathbf{A}^d_{2^{j+1}} f)(k,l) * \tilde{h}(k) * \tilde{g}(l))(2n, 2m) \tag{4.21}$$

$$\mathbf{D}^2_{2^j} f = (\mathbf{A}^d_{2^{j+1}} f)(k,l) * \tilde{g}(k) * \tilde{h}(l))(2n, 2m) \tag{4.22}$$

$$\mathbf{D}^3_{2^j} f = (\mathbf{A}^d_{2^{j+1}} f)(k,l) * \tilde{g}(k) * \tilde{g}(l))(2n, 2m) \tag{4.23}$$

for $j, k, l, m, n \in \mathbf{Z}$ where $f(x,y) \in \mathbf{L}^2(\mathbf{R}^2)$). The scale coefficients, $\mathbf{A}^d_{2^j} f$, become successively smoother versions of themselves and the details that are lost in smoothing are captured in the three sets of detail coefficients, $\mathbf{D}^1_{2^j} f$, , $\mathbf{D}^2_{2^j} f$, and $\mathbf{D}^3_{2^j} f$. Each of these sets of detail coefficients represents an orientation as shown in Figure 3.7.

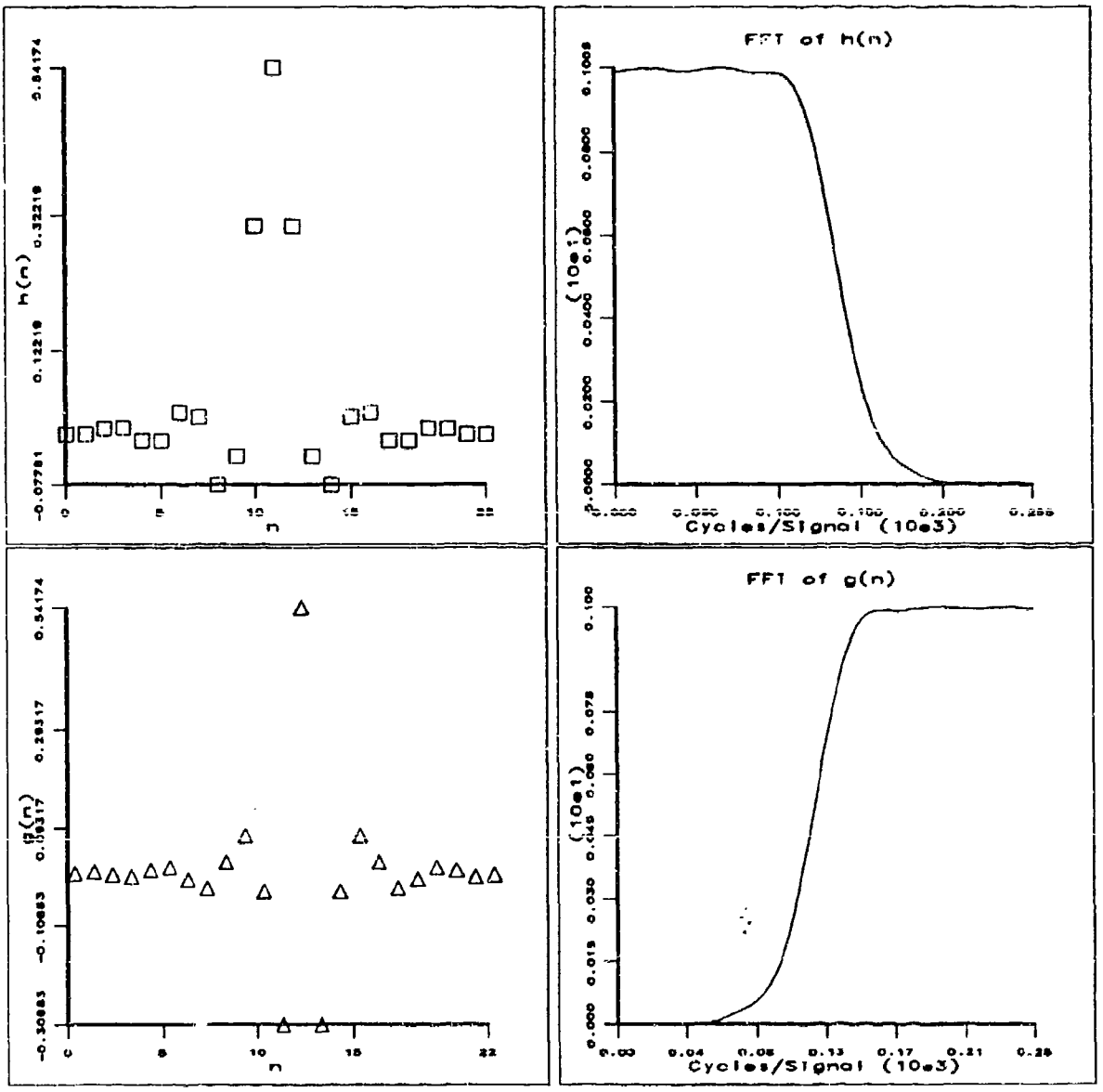


Figure 4.22. Response and Filter Functions Based on Cubic Spline Wavelet

In Equations 4.20 through 4.23, separate discrete variables k and l are used to emphasize that the response functions $h(n)$ and $g(n)$ operate on rows and columns independently. This emphasis plays an important role in understanding the mistake in Mallat's Figure 12 [23:685] which diagrams the two dimensional decomposition algorithm. There is an inconsistency between the text and the figure that we resolve in the following manner. First, we correct in **boldface** the text in paragraph A, first subparagraph, fifth sentence to read

> We first convolve the **cols** of $\mathbf{A}^d_{2^{j+1}} f$ with a one-dimensional filter, retain every other row, conlvolve the **rows** of the resulting signals with another one-dimensional filter and retain every other column.

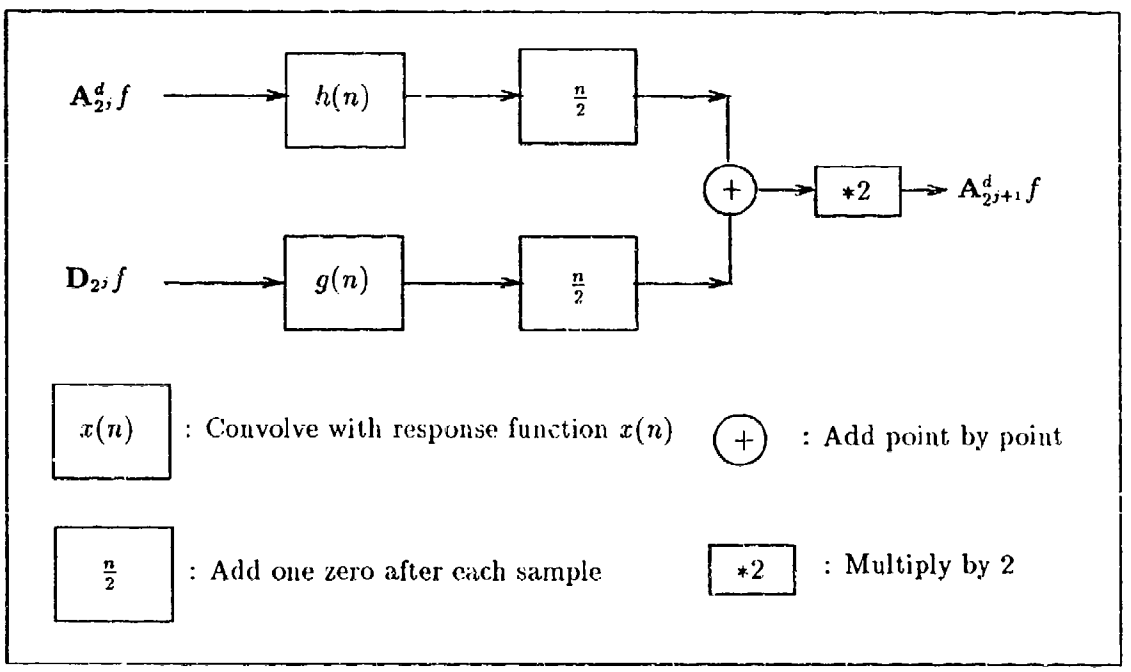


Figure 4.23. One Dimensional Multiresolution Reconstruction [23:682]

Next we correct his Figure 12 exchanging the words 'columns' and 'rows' at the top of the diagram. To understand why these corrections are necessary, consider the independent nature of the one dimensional convolutions performed on rows and columns. In the decomposition process, the rows/columns and respective $h(n)/g(n)$ convolution pairs must be the same as in the reconstruction process. In other words, the reconstruction and decomposition processes

must be mirrors of each other. Figure 4.24 diagrams the algorithm given by the pyramidal transforms of Equations 4.20 through 4.23. Figure 4.24 is Mallat's Figure 12 [23:685] redrawn and corrected.

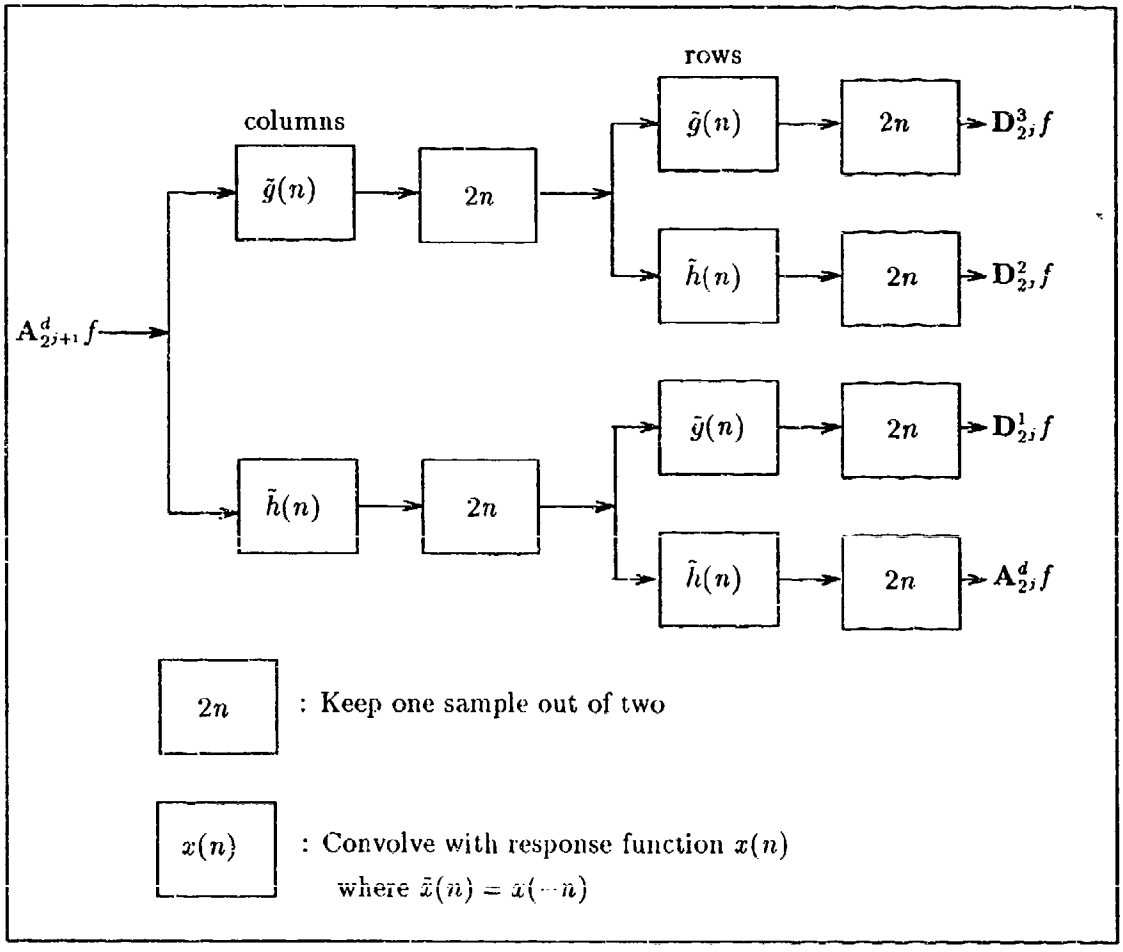


Figure 4.24. Two Dimensional Multiresolution Decomposition [23:685]

4.3.3 Multiresolution Reconstruction In his Equation (32) [23:682], Mallat shows that the scale coefficients at any level $j+1$ can be reconstructed from the scale and detail coefficients from the adjacent level j. Our interpretation of this equation is

$$\mathbf{A}^d_{2^{j+1}}f = 2((\mathbf{A}^d_{2^j}f)(\frac{k}{2}) * h(k))(n) + 2((\mathbf{D}_{2^j}f)(\frac{k}{2}) * g(k))(n) \tag{4.24}$$

This equation is implemented by inserting zeroes between each sample of $\mathbf{A}^d_{2^j} f$ and $\mathbf{D}_{2^j} f$ and convolving the results with h(n) and g(n) respectively. Finally, the convolution results are added point by point. The factor of two comes from the way Mallat normalizes his response function and is not necessary if implementing a Daubechies response function as given in [9]. Figure 4.23 diagrams the algorithm of Equation 4.24. This figure is redrawn from Figure 7 in [23:682].

4.3.4 Two Dimensional Multiresolution Reconstruction The reconstruction of a function $f(x,y) \in \mathbf{L}^2(\mathbf{R}^2)$ from the coefficients obtained by using Equations 4.20 through 4.23 is a natural extension of the one dimensional reconstruction. We apply the same notation extended to two dimensions. Again, we use the discrete variables k and l for row and column operations respectively. It is important for the rows/columns and $h(n)/g(n)$ reconstruction convolution pairs to match the decomposition convolution pairs. In other words, the reconstruction must be a mirror of the decomposition. This point is illustrated in Equation 4.25. For the two dimensional case, the reconstruction equation is:

$$\begin{aligned}
\mathbf{A}^d_{2^{j+1}} f \;=\; & 4((\mathbf{A}^d_{2^j} f)(\frac{k}{2},\frac{l}{2}) * h(k) * h(l))(n,m) + \\
& 4((\mathbf{D}^1_{2^j} f)(\frac{k}{2},\frac{l}{2}) * h(k) * g(l))(n,m) + \\
& 4((\mathbf{D}^2_{2^j} f)(\frac{k}{2},\frac{l}{2}) * g(k) * h(l))(n,m) + \\
& 4((\mathbf{D}^3_{2^j} f)(\frac{k}{2},\frac{l}{2}) * g(k) * g(l))(n,m) \qquad (4.25)
\end{aligned}$$

where $n,m \in \mathbf{Z}$.

A row of zeroes is inserted between each row before the columns of each coefficient set is convolved with the designated response function. Then, a column of zeroes is inserted between each column before the rows are convolved with the designated response function. Finally the convolution results are added. Again the factor, this time four, is for normalization of the $h(n)$ for the cubic spline as derived by Mallat and is not necessary if implementing Daubechies $h(n)$'s [9]. Figure 4.25 diagrams equation 4.25. This figure is adapted from Figure 13 in [23:686].

At any level of resolution, the scale or detail coefficients can be projected onto the scale or detail spaces respectively by using the general form of Equations 4.15 and 4.16 given here

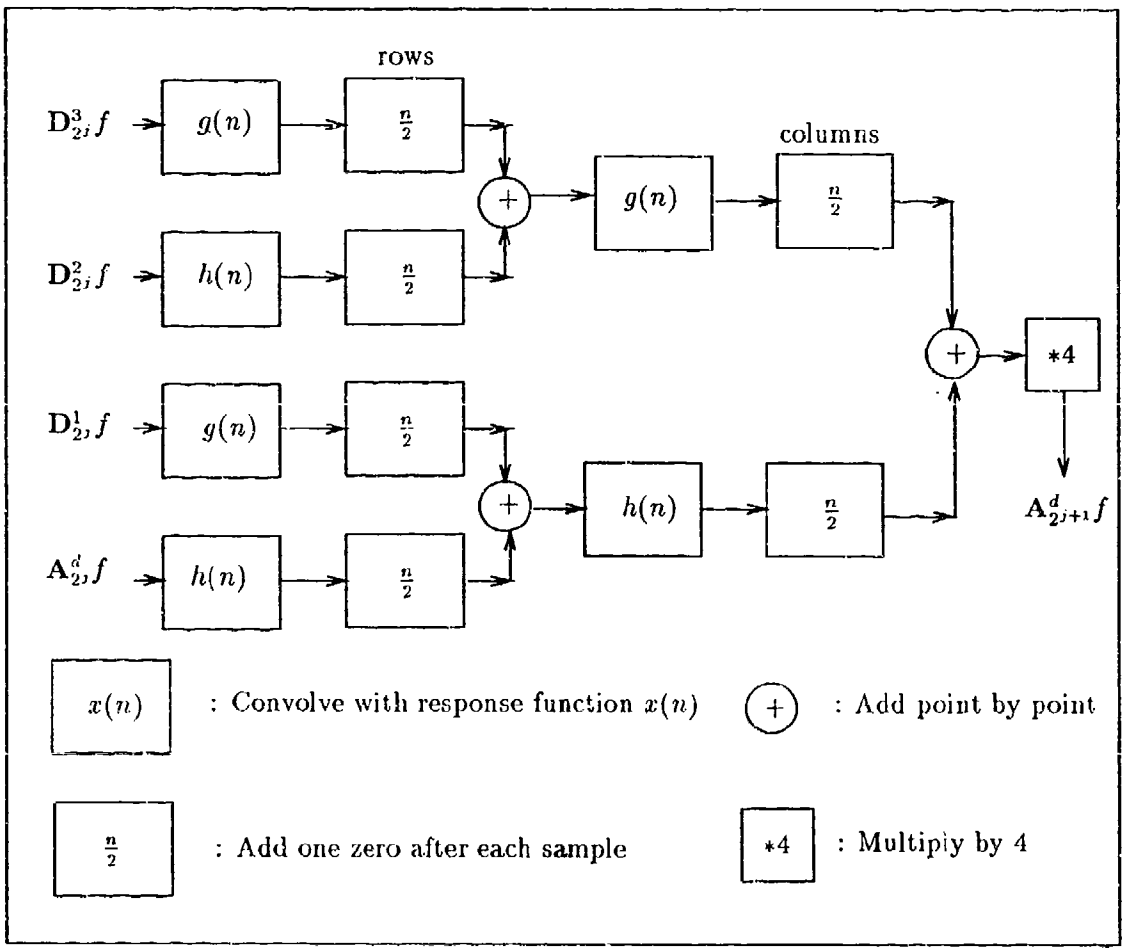


Figure 4.25. Two Dimensional Multiresolution Reconstruction [23:686]

in Equation 4.26 for the one dimensional case and in Equation 4.27 for the two dimensional case.

$$\mathbf{A}_{2^j} f = ((\mathbf{A}^d_{2^j})(n) * \phi_{2^j}(n))(k) \tag{4.26}$$

$$\mathbf{A}_{2^j} f = ((\mathbf{A}^d_{2^j})(n,m) * \phi_{2^j}(n) * \phi_{2^j}(m))(k,l) \tag{4.27}$$

4.3.5 Fine Points Of The Implementation of the Algorithm This section will address some of the more subtle problems which we encountered in the implementation of the multiresolution algorithm. Readers interested in implementing this algorithm, take heed.

4.3.5.1 Missing Coefficients in the Reconstruction The Multiresolution Algorithm promises an exact reconstruction can be accomplished from the retained coefficients of the decomposition process. The number of coefficients of the approximation $\mathbf{A}^d_{2^j} f$ plus the number of coefficients of the detail $\mathbf{D}_{2^j} f$ should be equal to the number of samples of the original signal or image. Since we generate the coefficients with the shift, multiply, and sum process, there are always more coefficients than the original number of samples. The number of resulting coefficients is equal to the number of samples of the original signal plus the number of elements in the filter. We discard the least important coefficients, those that border the image or signal. This results in an inexact reconstruction of the border or edge of the signal. This can be a significant problem since the decomposition process results in an increasingly smaller number of coefficients. Thus, a border error at the fifth level with respect to two coefficients will result in a reconstruction error spread over 64 samples of the original signal. Mallat suggests the border problem can be reduced by making the original signal symmetric with regard to the first and last sample or in the 2D case make the image symmetric with respect to the horizontal and vertical borders[23:681]. This process eliminates the border problem completely if the filter is symmetric and the reconstruction is accomplished with the same assumed border symmetry as in the decomposition. If the filter is asymmetric the problem may only be alleviated by padding the image with enough extra elements to retain the extra convolution coefficients.

4.3.5.2 Convolution Methods There are two main methods of accomplishing convolution. The first is to calculate the so called "convolution sum" using a shift multiply and sum routine. The second is to take the Fourier Transform of the two functions, multiply them point by point, and take the inverse Fourier Transform. The first method is normally considered slower. It has a time complexity of $\mathbf{O}(N^2)$ assuming that the functions to be

convolved are the same size. The Fourier Transform method used with the Fast Fourier Transform (FFT) has a time complexity of O(NlogN). In the multiresolution algorithm, the filters used are normally a fraction of the size of the signal or image of interest. This enables us to reduce the time complexity of the shift multiply and sum routine to approximately O(N). Therefore, we have chosen the shift, multiply, and sum method. However, our investigation of the Fourier Transform method revealed some interesting points of the application at hand, which we include for the benefit of the reader in the following section.

4.3.5.3 Numerical Recipies in C Convolution Routine The convolution routine in Numerical Recipies in C is a function called *convlv*. The interface to this function requires the response function have an odd number of values m and be stored in an array in "wrap around order". Wrap around order as shown in Figure 4.26 requires those elements of the response function greater than or equal to zero on the discrete time (sample) axis to reside in that order in the first positions in the input response array, "respns". Those response elements less than zero on the discrete time (sample) axis must be stored in the same order in the last positions in the response array. If the same variable name is used more than once to hold the response array input to convlv, it must be reset each time the procedure is called. This is due to the fact that the response array is altered each time convlv is called. While these are fine points in the use of the convolution routine, they must be exactly followed for successful convolutions using Numerical Recipies in C.

4.3.5.4 Problems Encountered Using the Khoros System All of the images used in the decomposition analysis were composed of integer grey scale values between 0 and 255. They exist in a floating point format to obtain the needed accuracy in the decomposition and reconstruction algorithm. We visually evaluate the results of the reconstruction with the Khoros image processing system provided by the University of New Mexico [31]. The first reconstructed images viewed in this system appeared to be much darker than the original image. After analyzing the resulting floating point values of the reconstructed image we discovered that zero gray scale values in the original image corresponded to small negative values in the reconstructed image. Inherent in the Khoros display system is a normalization process which compresses the dynamic range of the rest of the image to accommodate the negative numbers. To produce a more visually acceptable reconstruction, we set all values less than zero to zero and greater than 255 to 255.

4.3.6 Examples The Multiresolution Decomposition decomposes an image into a lower resolution approximation and three detail signals. This process is iterated to obtain successively lower, coarser, resolution approximations and details. This section along with the following diagrams will demonstrate this process and provide additional insight into the frequency content of these approximation and detail signals.

Figures 4.28-4.30 show the detail coefficients from a decomposition of an original image made up of two rectangular boxes. We chose this image for its pristine vertical and horizontal high frequency content, edges. These detail signals are thresholded and binarized using our *threshold* program discussed previously. T[illegible]e figures illustrate the edge detection

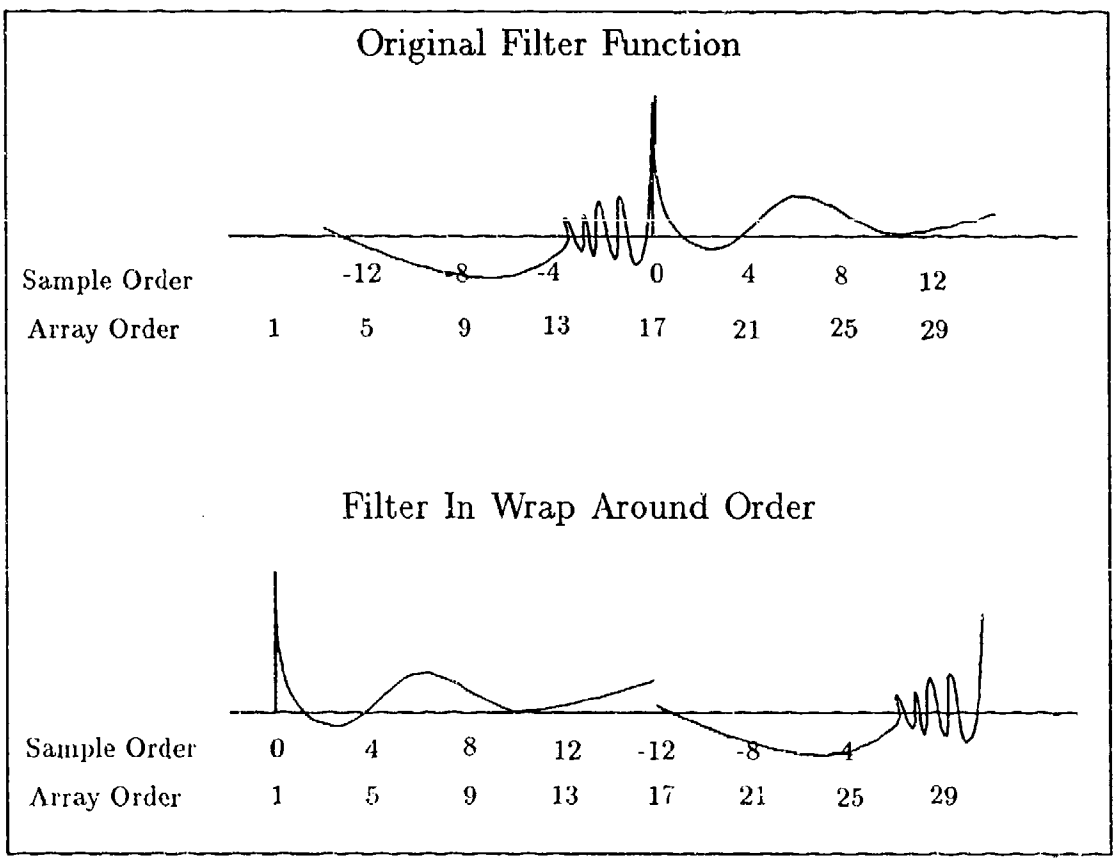


Figure 4.26. Wrap Around Order for the Convlv.c Procedure

capability of multiresolution wavelet analysis and the orientation selectivity of the different detail signals. The magnitude of the Fast Fourier Transform of the wavelet detail coefficients

in Figure 4.32, demonstrates how well this orientation selectivity is accomplished. The original image, two rectangular boxes, is also shown in the figure. These plots illustrate how the frequency content of each detail signal is localized in terms of orientation. The $\mathbf{D}_{2^j}^1 f$ coefficients contain the horizontal high frequency information, the $\mathbf{D}_{2^j}^2 f$ coefficients contain the vertical high spatial-frequency information, and the $\mathbf{D}_{2^j}^3 f$ coefficients contain the higher angular frequency information of the original image. In this figure, we arbitrarily chose level $j = -4$ for documentation convenience. All levels of resolution are shown to have this orientation selective characteristic as diagramed in Figure 3.7.

Figures 4.33-4.39 illustrate the main facets of the multiresolution decomposition and reconstruction process. The original image, 512x512 Lenna, is given in Figure 4.33 for a comparison with the various results of multiresolution process. Figure 4.34 is the reconstructed Lenna from a 5 level decomposition. The successively coarser approximations $\mathbf{A}_{2^j}^d$ of Lenna are shown in Figure 4.35 on the top of the page. Notice the reduction in size as a result of the down sampling from the original Lenna Figure 4.33 (level 0) to the first approximation (level 1) in the upper left corner of Figure 4.35. The bottom of Figure 4.35 shows the series of reconstructed approximation $\mathbf{A}_{2^j}^d$ of Lenna. The final reconstruction (level 0) is found in Figure 4.34. The coarsest approximation of Lenna, a 16x16 image, is shown in Figure 4.39. This level 5 approximation along with the detail coefficients found in Figures 4.36-4.39 are used to accomplish the reconstruction. Note that these coefficients have been thresholded to make the orientation specific frequency content viewable.

4.4 Conclusion

This chapter evaluates two methods of multiresolution analysis. It demonstrates only the decomposition capability of the projection method, although reconstruction is possible. Basically, the V and W space projections at some arbitrary coarse level of decomposition are added point by point. The result is then added to the W space projections at the next finer level of resolution. This process is iterated until the finest level approximation is reached resulting in the final reconstruction. We elected not to pursue this technique due to the computational overhead associated with the projection of every set of the decomposed coefficients onto the V and W spaces for addition. Instead, we chose to implement the reconstruction with the second method of multiresolution analysis described in this chapter, using Quadrature Mirror Filters (QMF). In this method, the sets of scale and wavelet coefficients

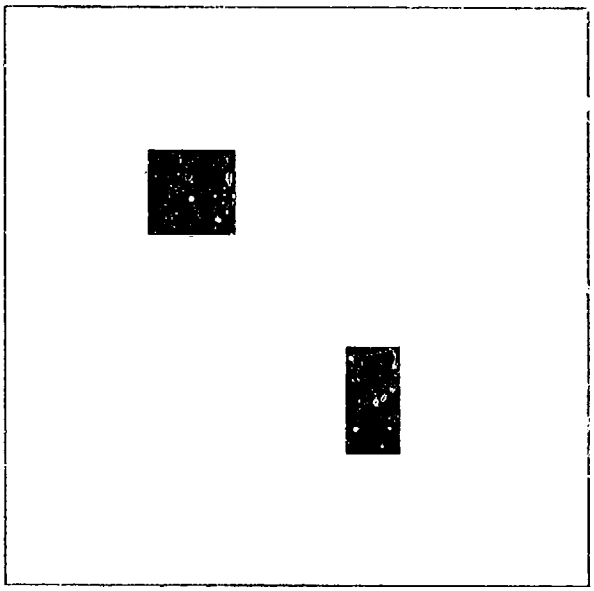


Figure 4.27. Original Image of Boxes (Reduced 58%)

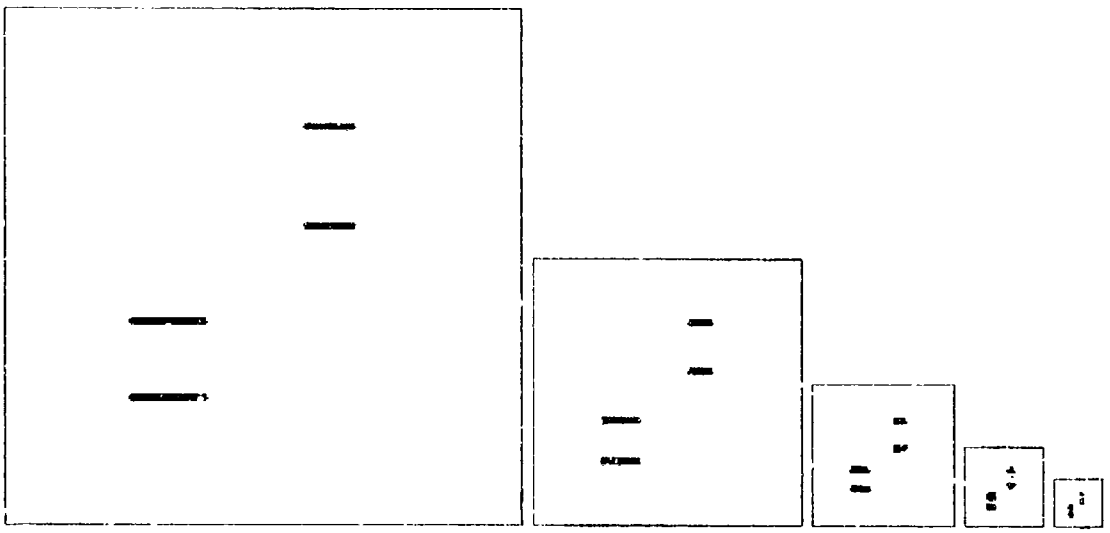


Figure 4.28. Horizontal Multiresolution Detail Coefficients of Boxes (Reduced 25%)

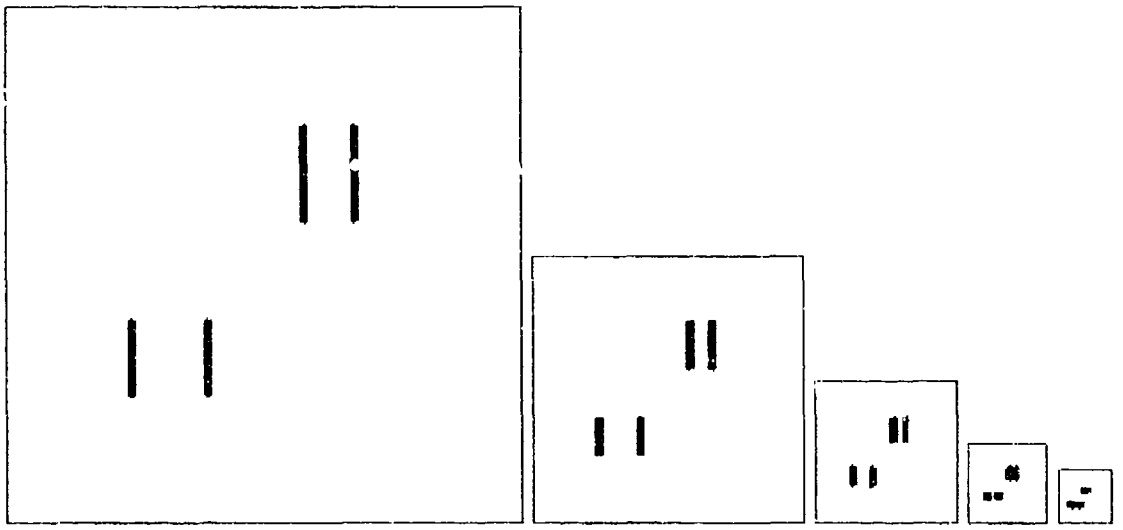


Figure 4.29. Vertical Multiresolution Detail Coefficients of Boxes (Reduced 25%)

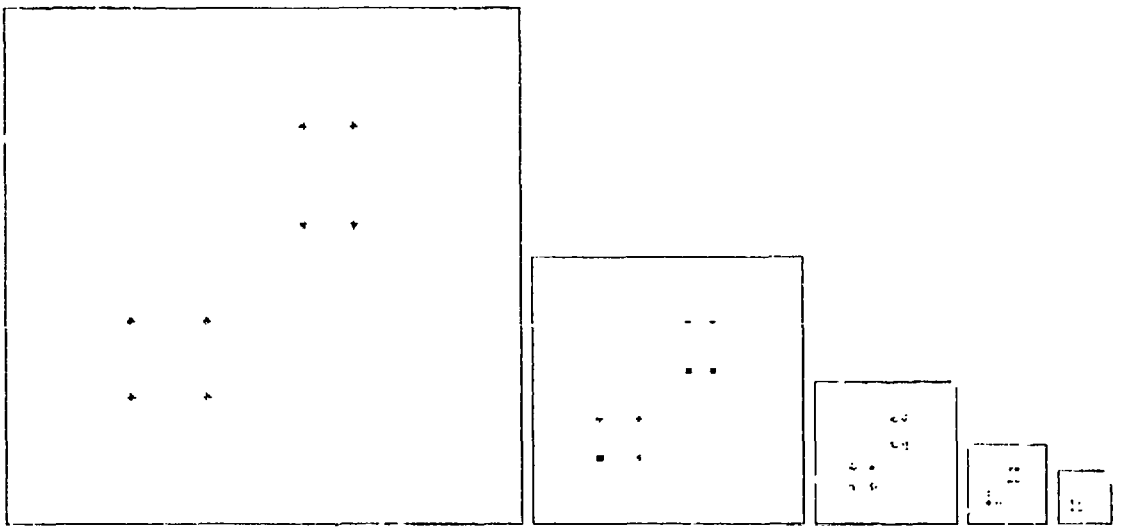


Figure 4.30. Angular Multiresolution Detail Coefficients of Boxes (Reduced 25%)

Figure 4.31. Coarsest Approximation of Boxes Used for Reconstruction (Reduced 25%)

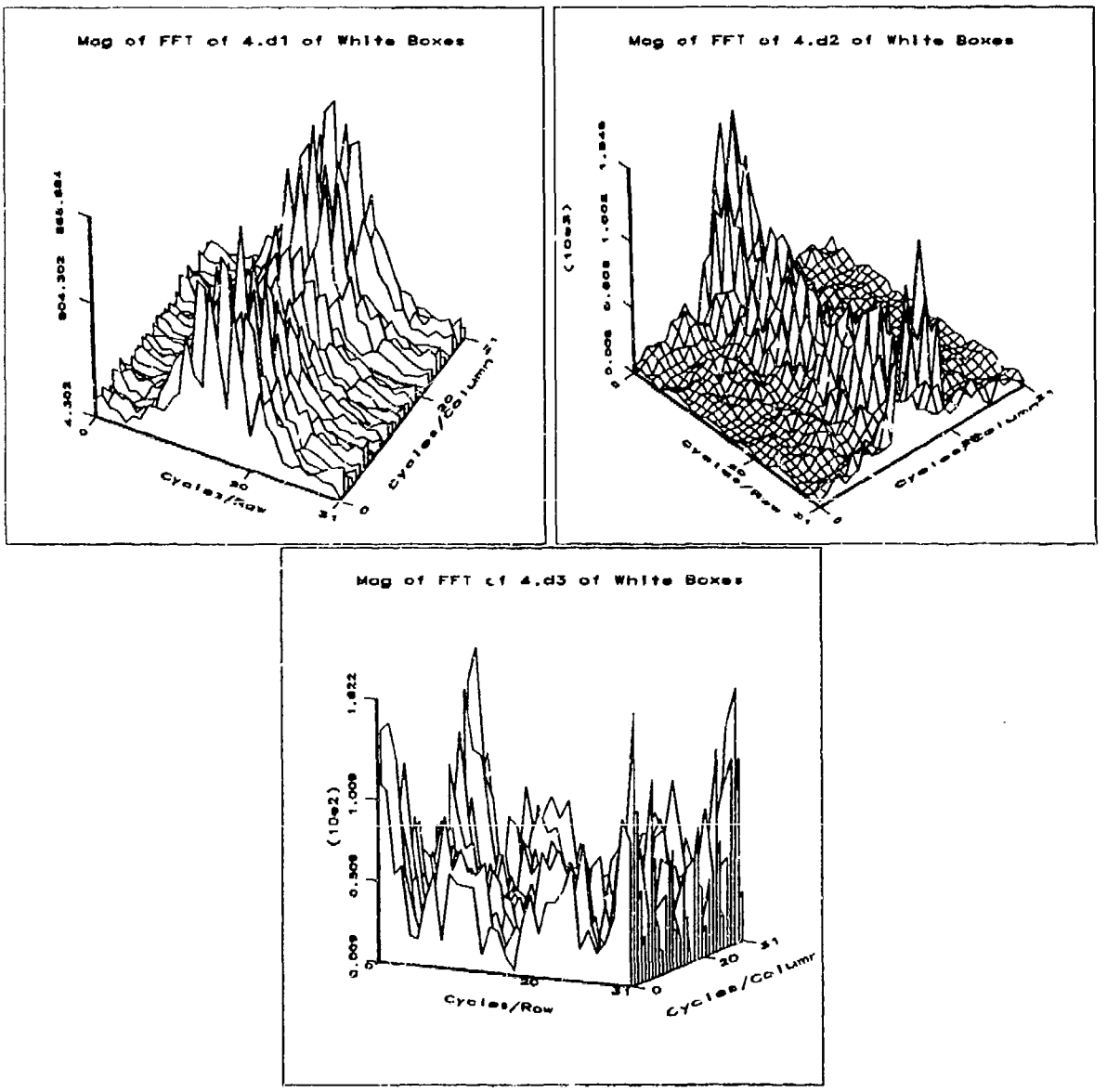


Figure 4.32. Frequency Support of Detail Signals Of The Cubic Spline Wavelet

Figure 4.33. Original Image of Lenna (Reduced 2%)

Figure 4.34. Reconstructed Image of Lenna Using the Spline Wavelet (Reduced 2%)

Figure 4.35. Multiresolution Decomposition/Reconstruction Approximations of Lenna Using the Cubic Spline Wavelet (Actual Size)

Figure 4.36. Horizontal Multiresolution Detail Coefficients of Lenna (Reduced 25%)

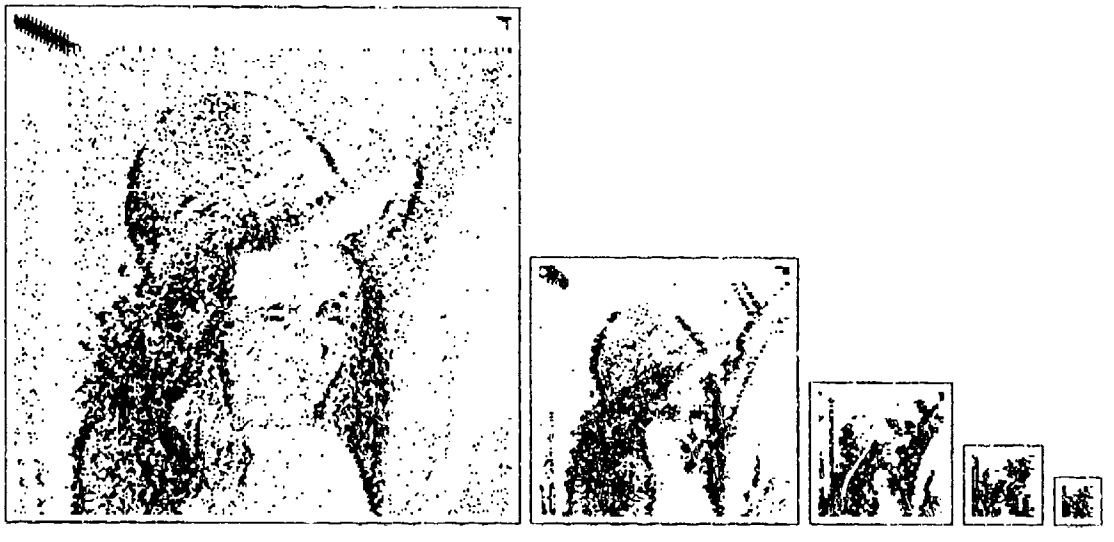


Figure 4.37. Vertical Multiresolution Detail Coefficients of Lenna (Reduced 25%)

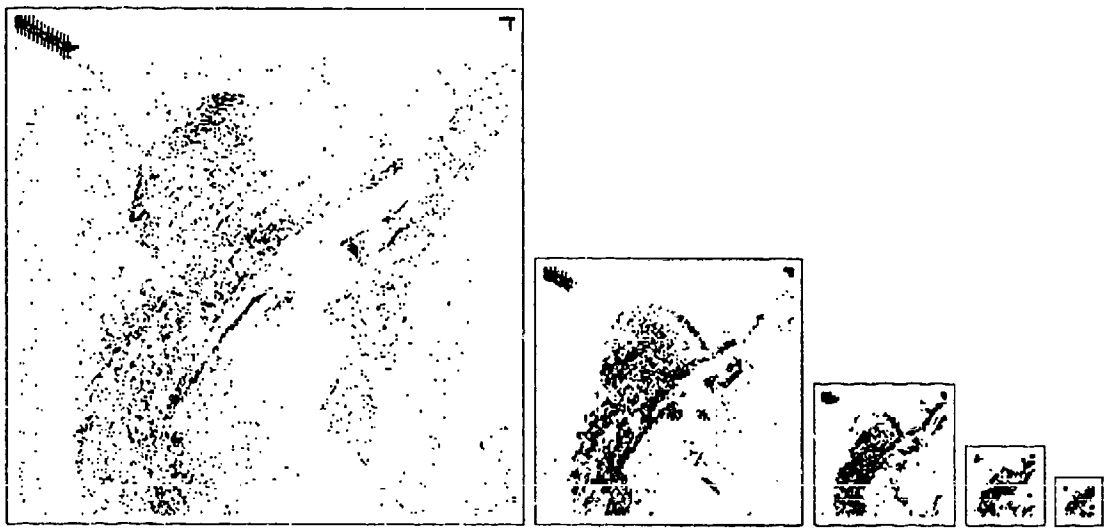


Figure 4.38. Angular Multiresolution Detail Coefficients of Lenna (Reduced 25%)

Figure 4.39. Coarsest Approximation of Lenna Needed for Reconstruction (Reduced 25%)

get logarithmically smaller with coarser levels of resolution. Moreover, the algorithm does not require that the coefficients be projected at each level of resolution. For these reasons, we use the QMF method as the tool for analyzing the data in this thesis.

V. *Experimental Application and Results*

5.1 Introduction

This chapter explains the approach and methodology used to segment Synthetic Aperture Radar (SAR) imagery and Forward Looking InfraRed (FLIR) using the multiresolution approximation representation. This chapter first reviews the objectives of this thesis research. Second, a description of the overall system approach is given. Third, criteria for multiresolution feature selection is explored. Finally, segmentation results using different wavelets are given.

5.2 Overview

Thus far, the background of a wavelet representation and the basics of artificial neural networks have been explained with an emphasis on their usefulness as an analysis tool. The purpose of this research is to use these tools to segment SAR imagery, specifically, to segment different homogeneous regions (trees, fields, and shadow) from one another. The goal in general is then to determine which set of the many multiresolution coefficients will provide an adequate set of features to enable the radial basis function network to accomplish segmentation. The following sections address the application of these tools to the segmentation problem.

5.3 Methodology

5.3.1 Introduction This section is composed of five parts. The first part gives a brief overview of the total segmentation system. The second part explains the method of selecting wavelet coefficients. The third part explains the rationale for using receptive fields. The fourth part explains the procedures for training the RBF neural network. The fifth and final part provides information regarding the SAR and FLIR imagery.

5.3.2 Approach The diagram in figure 5.1 represents the overall segmentation system. The first task is to take the SAR data which is stored in 884 format and convert it to an ascii format. This is accomplished in four steps. The first is to use `rd884.c`, which converts the data from 884 data format to a raw floating point complex pair format. Second, the program `logb.c` is used to convert the complex data to a raw byte image file with values between 0 and 255. Both of these programs were supplied by Sandia Laboratories and are included in the Appendix. The third step uses the Khoros system (provided to the Model Based Vision Laboratory by the University of New Mexico) to convert from byte format to

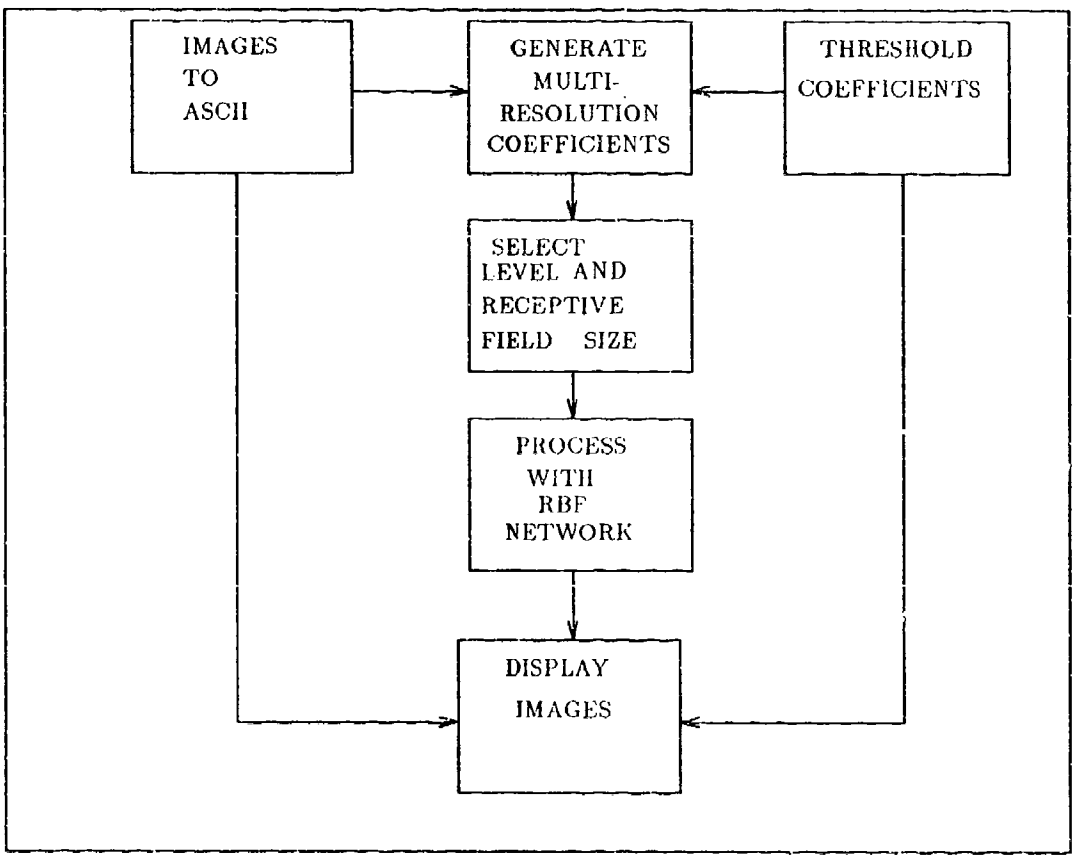


Figure 5.1. Block Diagram of Segmentation System

labeled ASCII format. The final step of stripping off the labels is accomplished by the the `matrixtoascii.c` program.

The image file is then processed by the `wave2` program which utilizes Mallat's algo-

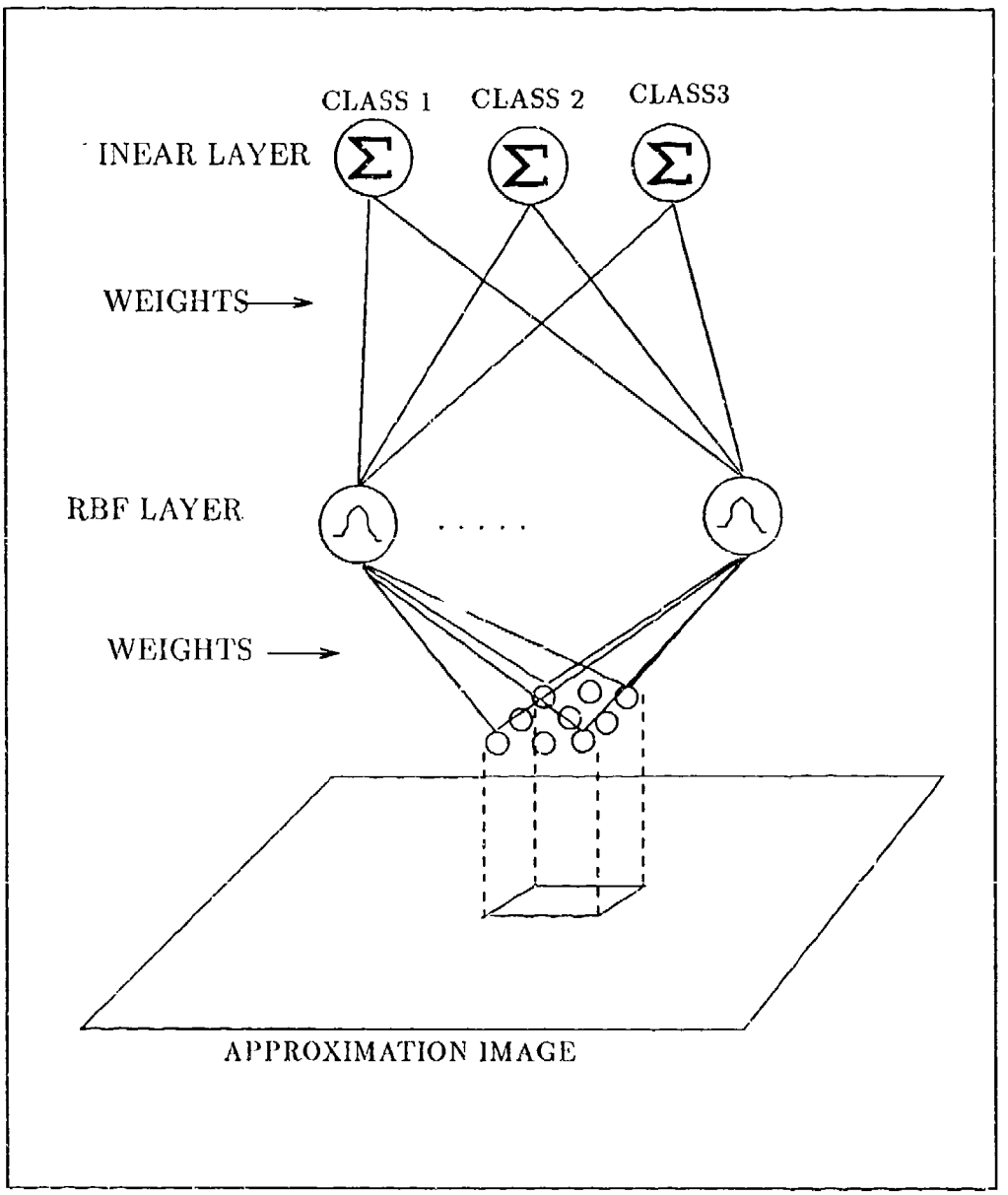


Figure 5.2. Segmentation System Architecture

Level	Detail Signal Cycles/Object	Approximation Signal Cycles/Object	Smallest Detectable Change
0	N/A	0-1024	1 foot
1	1024-512	0-512	2 feet
2	512-256	0-256	4 feet
3	256-128	0-128	8 feet
4	128-64	0-64	16 feet
5	64-32	0-32	32 feet
6	32-16	0-16	64 feet
7	16-8	0-8	128 feet
8	8-4	0-4	256 feet
9	4-2	0-2	512 feet
10	2-1	0-1	1024 feet

Figure 5.3. Frequency Content of Multiresolution Levels

rithm. Details of the `wave2` program can be found in the Appendix. One of the resulting wavelet coefficient files (note that these files are formatted as if they were each an image) is selected based on the size and frequency content of the objects or regions to be segmented. The file which is selected is then partitioned into small overlapping receptive fields centered about one coefficient value. Initially a small subset of the image is used to train the network on the different regions. Following network training the entire image is fed through the artificial neural network and each coefficient is classified as a particular type of region and is given a particular gray scale value. The resulting segmented image can then be viewed, using Khoros, for comparison with the original image. A diagram of the total system architecture in shown in Figures 5.1 and 5.2.

5.3.3 Selection of Wavelet Coefficients The multiresolution representation, as developed in Chapters 3 and 4, can be viewed as filtering an image with a set of low pass filters with successively narrower bandwidths. The information lost due to this smoothing process is retained in the detail signal. This process of approximation will result in a set of successively lower resolution representations of the original image. Figure 5.3 shows the frequency content of the various resolution levels which correspond to a 2048X2048 original image. The

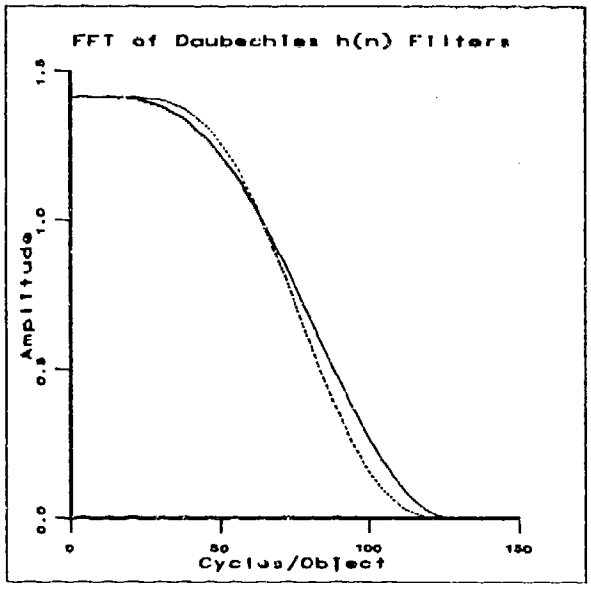


Figure 5.4. Characteristics of Various Daubechies H Filters. Daub2(solid), Daub3(dots)

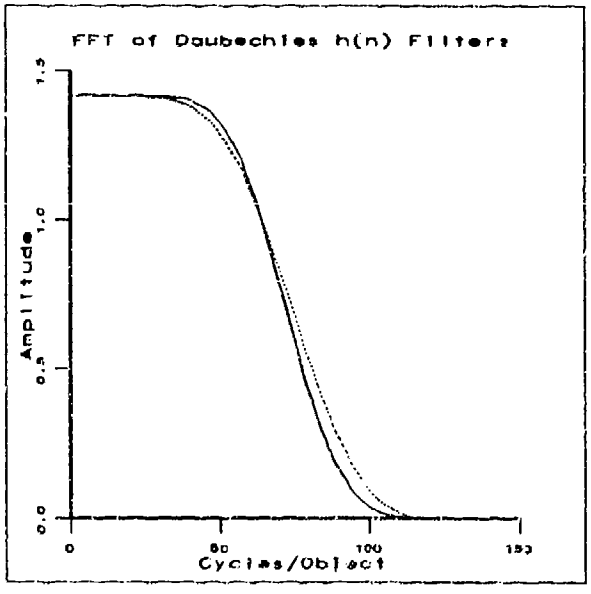


Figure 5.5. Characteristics of Various Daubechies H Filters. Daub4(dots), Daub6(solid)

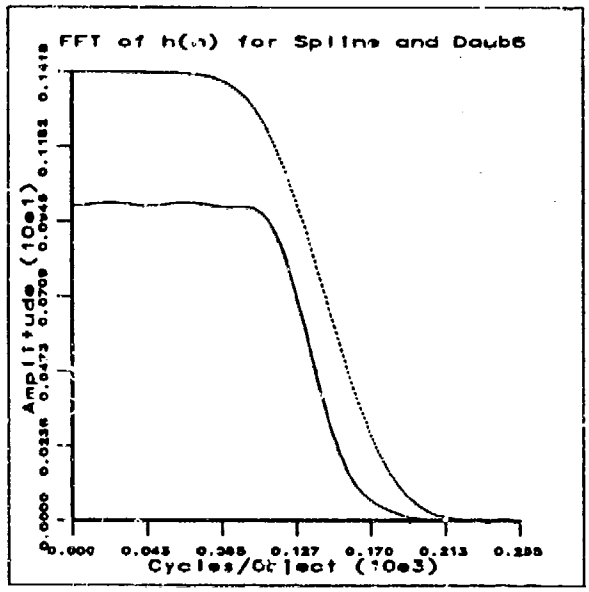


Figure 5.6. Comparison of The Spline (solid) and Daubechies (dots) H Filters

highest possible frequency in a 2048X2048 image is 1024 cycles/object which corresponds to an object which is 2 feet (2 pixels) wide. This correlation of pixel size to the actual size of an object is possible due to the use of SAR imagery. Because the effective view of SAR imagery is a "God's eye view", and the camera angle or source angle doesn't create a skewed view, the density of pixels per area is constant. For example, normally a camera produces pictures with a non-uniform density of pixels to actual area in a scene. Those pixels which correspond to near objects in the scene cover less area than those pixels which correspond to objects further away. One pixel might correspond to 1 foot by 1 foot resolution of the scene for near objects, while another pixels may correspond to a 10 foot by 10 foot area for objects further away in the scene. These non-uniform density of pixels to area effects are not found in SAR imagery.

The assignment of cycles/object to a particular level is a direct result of the filtering process and the down sampling required by Mallat's algorithm. Figures 5.4 and 5.5 show the roll-off of various Daubechies H filters which represent the use of different wavelet basis sets. Figure 5.6 compares the roll-off of a Daubechies-6 H filters with that of Mallat's cubic

Level	Image Dimension	Total Number of Elements
0	2048X2048	4194304
1	1024X1024	1048576
2	512X512	262144
3	256X256	65536
4	128X128	16384
5	64X64	4096
6	32X32	1024
7	16X16	256
8	8X8	64
9	4X4	8
10	2X2	4

Figure 5.7. Dimension of Coefficient Files a Different Levels

spline H filter. Note that the higher the order of the Daubechies H filter, the more rapid the roll-off, but the bandwidth of the filter remains nearly constant. Figure 5.8 shows the changing band widths of the same filter with the down sampling. These lower resolution approximations also contain only one-fourth as many coefficient values as the previous level. Figure 5.7 shows the reduced number of approximation coefficients.

The result of this process is to suppress objects in an image which have higher frequency content (smaller objects or edges) at each lower resolution level.

5.3.4 Choosing The Receptive Field Size Once a particular level has been selected, features for the radial basis network must be chosen. The overlapping receptive field or window mentioned in the previous section dictates the number of features used to classify a single coefficient value. This can be seen in Figure 5.9 which represents a distribution of wavelet coefficients of some image at an arbitrary level. The center value in the 3X3 receptive field is what the network is attempting to classify. It may be impossible to distinguish between classes based on one coefficient value as would be the case for any of the small white blocks in Figure 5.9. If we utilize the 8 nearest neighboring coefficient values, of any particular coefficient value, it becomes clearer as to which class each individual coefficient

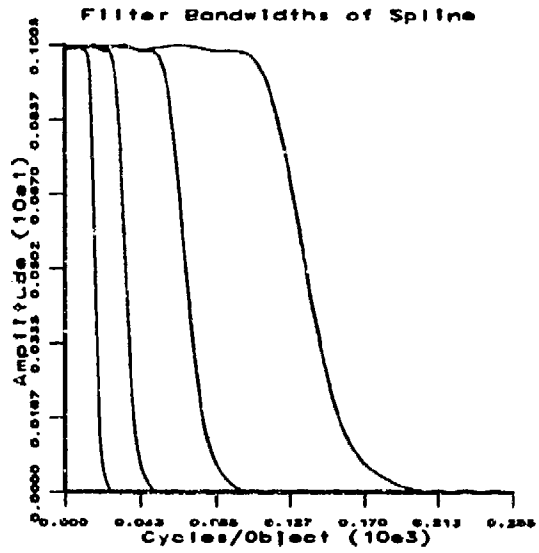


Figure 5.8. Filter Band Widths With Down Sampling

belongs. This method does has difficulty classifying blocks near the dashed boundary line in Figure 5.9 which is between the two different types of regions. If we reduce the size of the receptive field to take care of the boundary problem we reduce our ability to distinguish members of the homogeneous classes. The Multiresolution Representation may have the needed information to deal with this problem.

Depending on our original choice of levels, the Multiresolution Representation can provide levels of greater detail of these ambiguous regions. For example, the segmentation of the homogeneous regions (tree, field, and shadow) were done at level 4 using a 3X3 receptive field. The 3X3 receptive field contains 9 coefficients or in other words generates 9 features for each coefficient value. Each coefficient value at level 4 is represented at level 3 (a higher resolution representation) by four coefficient values and at level 2 by eight coefficient values. These additional levels of detail may be useful in defining more accurately the bounding edge between adjacent regions. The limitation on the number of training vectors did not allow for explicit testing of this theory while segmenting the homogeneous regions, but higher levels of resolution were utilized in segmenting the roads from the SAR imagery.

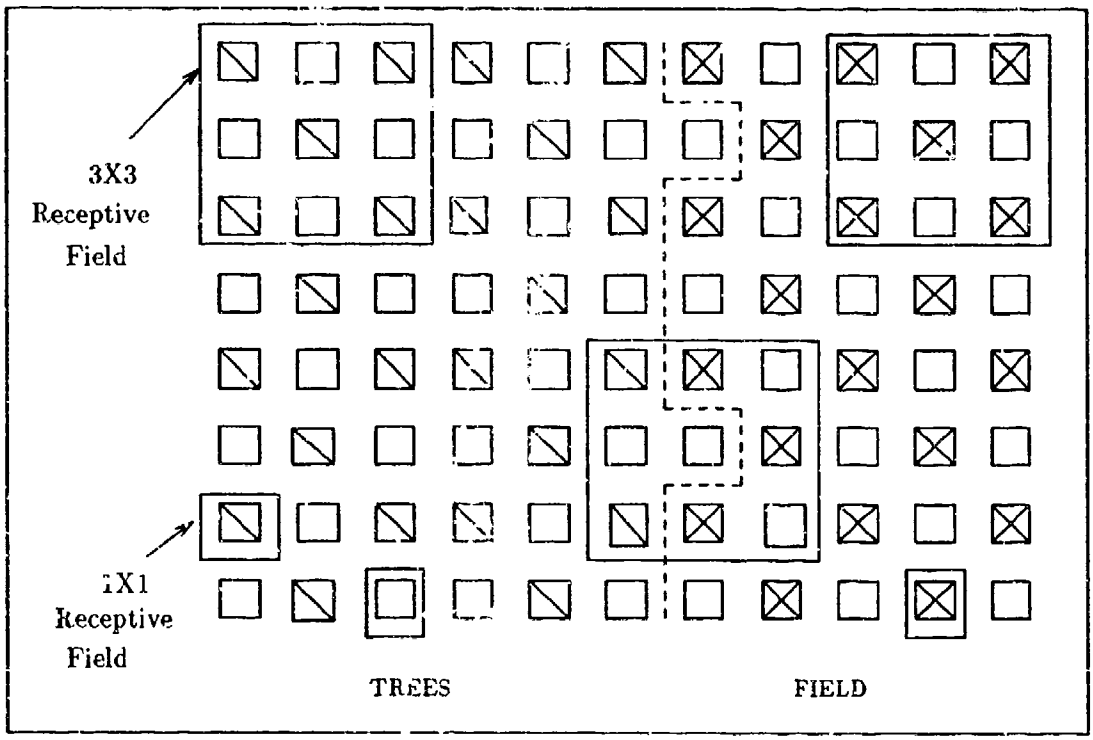


Figure 5.9. Receptive Field Sizes For Segmentation

5.3.5 Data For Training The Radial Basis Function Network The neural network classifier is trained by presenting to it samples of classified data known as truth data. Once this training process is completed, the network then attempts to classify or categorize any presented data into one of the previously learned classes. This subsection discusses the total amount of training data which can be used and its relationship to the number of training samples, the number of features per sample, and the number of classes. The specific learning algorithms used by the RBF network can be found in [37].

There are two main rules-of-thumb known as Foley's Rule and Cover's Rule. Foley's Rule deals with the relationship between the number of features per training vector and the number of total training vectors. There should be at least three times as many training vectors per class as there are features per vector. Thus a 3X3 receptive field would have nine features per vector requiring at least 27 training vectors per class. Cover's Rule states

that if the total number of training vectors for a two class problem is less than two times the number of features, the error rate on the train set will always be very nearly zero with-out regard for the distribution of data [32:60]. This indicates that training results (percent error) on a sparse set of training data will not necessarily reflect the result of the test data. The network will "memorize" the training data but generalization to the test data will be poor.

The RBF neural network used in this research is constrained to having no more than 200 training vectors. Each of the vectors may contain up to 100 components or features [36]. Nearly all of the training was done to meet or exceed the rules given above to provide a network classifier capable of robust generalization.

5.3.6 Imagery

5.3.6.1 Synthetic Aperture Radar Imagery The 1 foot by 1 foot resolution SAR imagery data used in this research was supplied by MIT/Lincoln Laboratories via the Model Based Vision Laboratory at Wright Patterson AFB. Although the full polarimetric data was available, only the horizontal/horizontal polarization was used. Mission 85 is almost entirely composed of regions of trees, field, and shadow. All of the SAR imagery was histogram normalized before processing and display.

- mission 85, pass 5, frame 30 (m85p5f30hh)
- mission 85, pass 5, frame 29 (m85p5f29hh)
- mission 85, pass 5, frame 28 (m85p5f28hh)
- mission 85, pass 5, frame 27 (m85p5f27hh)

5.3.6.2 Forward Looking Infrared Imagery The FLIR data used in this thesis was provided via the Model Based Vision Laboratory by Texas Instruments (TI) Corp. The data is a series of high-resolution, 8-bit, FLIR images. The collection of data involved moving and stationary targets during both day and night. All of the imagery was taken from a height of approximately 1000 feet along a previously defined flight vector [12:1].

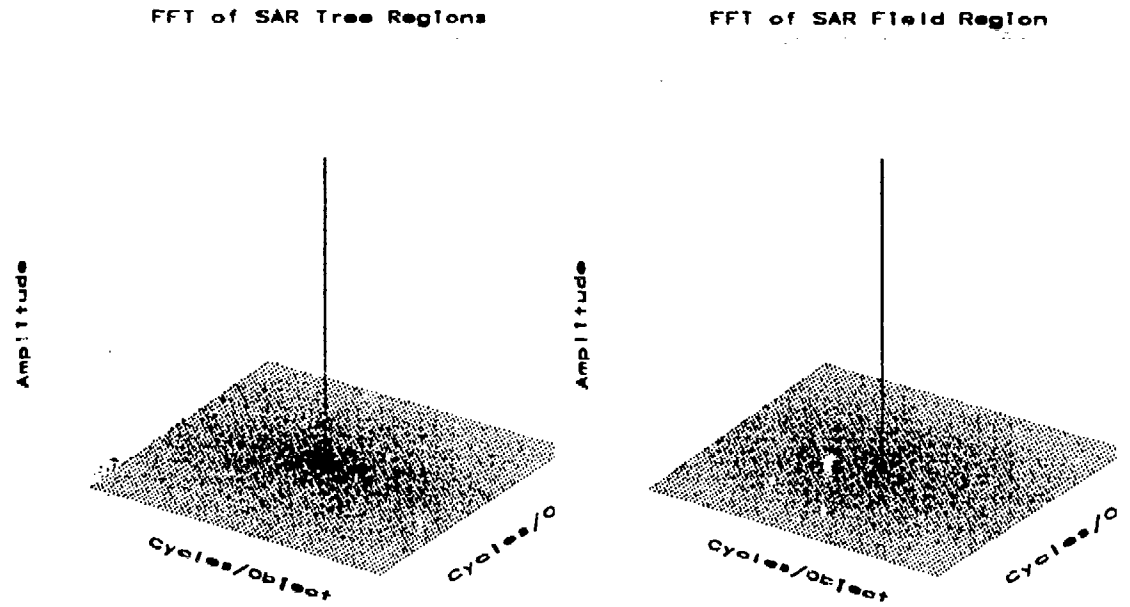


Figure 5.10. FFT of a 64X64 Region of Trees And Field

5.4 Segmentation of Homogeneous Regions

5.4.1 Selection of Wavelet Coefficients Recalling that we want to segment the tree, field, and shadow regions shown in Figure 5.11 from one another, it is necessary to find the frequency content of the field and trees. The first analysis is to simply look at the multiresolution approximations of the the SAR imagery and notice any obvious differences between the fields, trees, and shadow at a particular level. Figure 5.11 is a section of mission 85 with large sections of trees, field and shadow. Figure 5.12 shows the approximation from levels 1 through 4. It is easily seen that the level 4 representation (smallest) of the image had a greater visual contrast between the field and trees than any other level. The next step is to investigate the frequency content of the tree and field regions. This can be accomplished by taking the Fourier transform of a small area of trees and a small area field. In Figure 5.10 both magnitude plots look some what similar. If we ignore the DC component (center spike), the only significant difference is that the trees have more energy concentrated at the center of the frequency plane (lower frequencies) than do the field. These lower frequencies

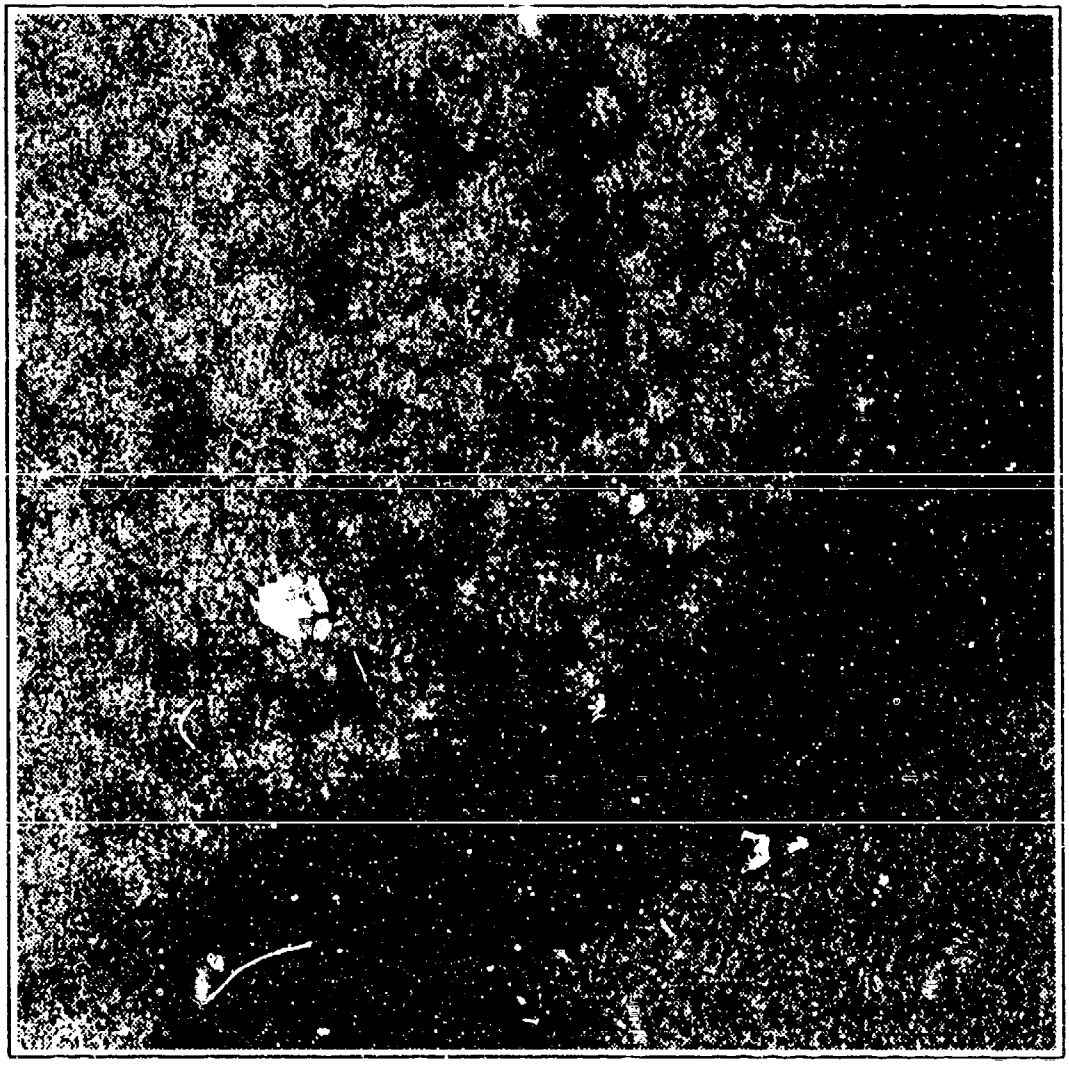


Figure 5.11. 512X512 Sample of SAR Imagery (Reduced 22%)

are between 1 and 2 cycles/object taken from a 64X64 image sample. This indicates the size of the tree clusters to be between 32 and 64 feet long. This reinforces the previous observation that the level 4 approximation would provide a good set of features.

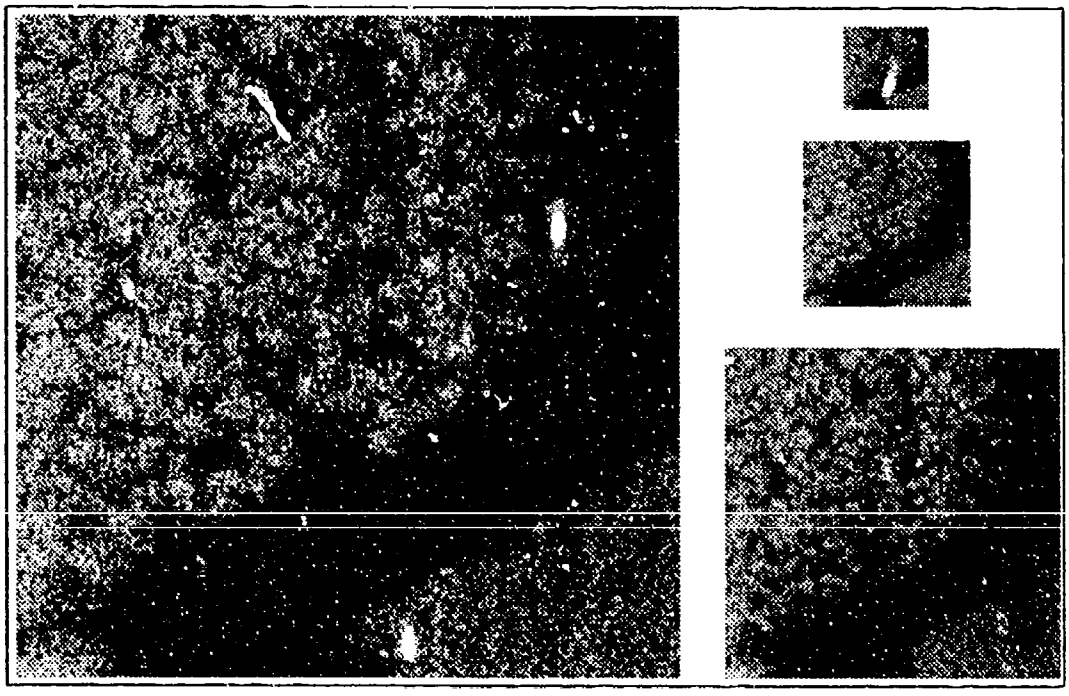


Figure 5.12. 512X512 SAR Imagery Multiresolution Approximations Using The Cubic Spline Wavelet (Actual Size)

5.4.2 Results A small subsection of mission 85 is shown at full scale, 72 pixels per inch, along with the segmentation results in Figure 5.13. Figure 5.14 is a reduced version of the entire 2048X2048 mission 85 and will be the baseline for comparison with the following images segmented using different wavelets. The correctness of the segmentation is somewhat arbitrary. That is to say, if 10 people were to segment m85 by hand there would be 10 different results and all of the 10 would be correct. Additionally, if a tactical planner where to hand segment mission 85 he or she might segment it very differently than the 10 test cases. It is more appropriate in these expiremental results to let the reader judge from his perspective if the results are useful for his purposes. Figures 5.15, 5.16, 5.17, and 5.18

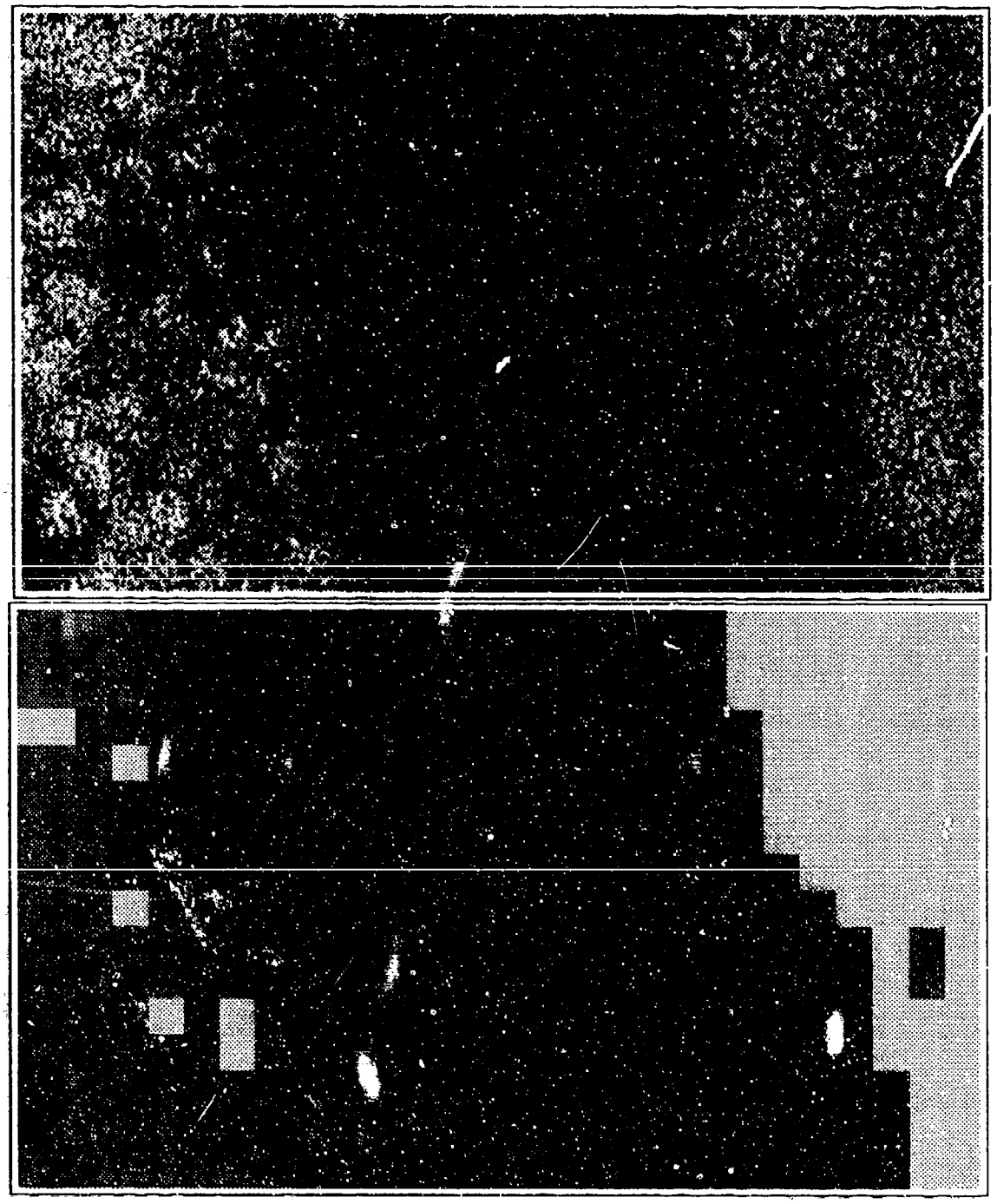


Figure 5.13. Homogeneous Regions of Interest and Their Segmentation (Actual Size)

show the segmentation of homogeneous regions as accomplished by the spine wavelet and Daubechies 2, 3, and 6 respectively. Notice in Figure 5.15 small regions of trees throughout the image are identified and segmented as trees. This is are the case of the other three segmentations. Because the filter characteristic of the Cubic Spline and the Daubechies wavelets are very similar, it is not surprising that the resulting segmentations are also similar.

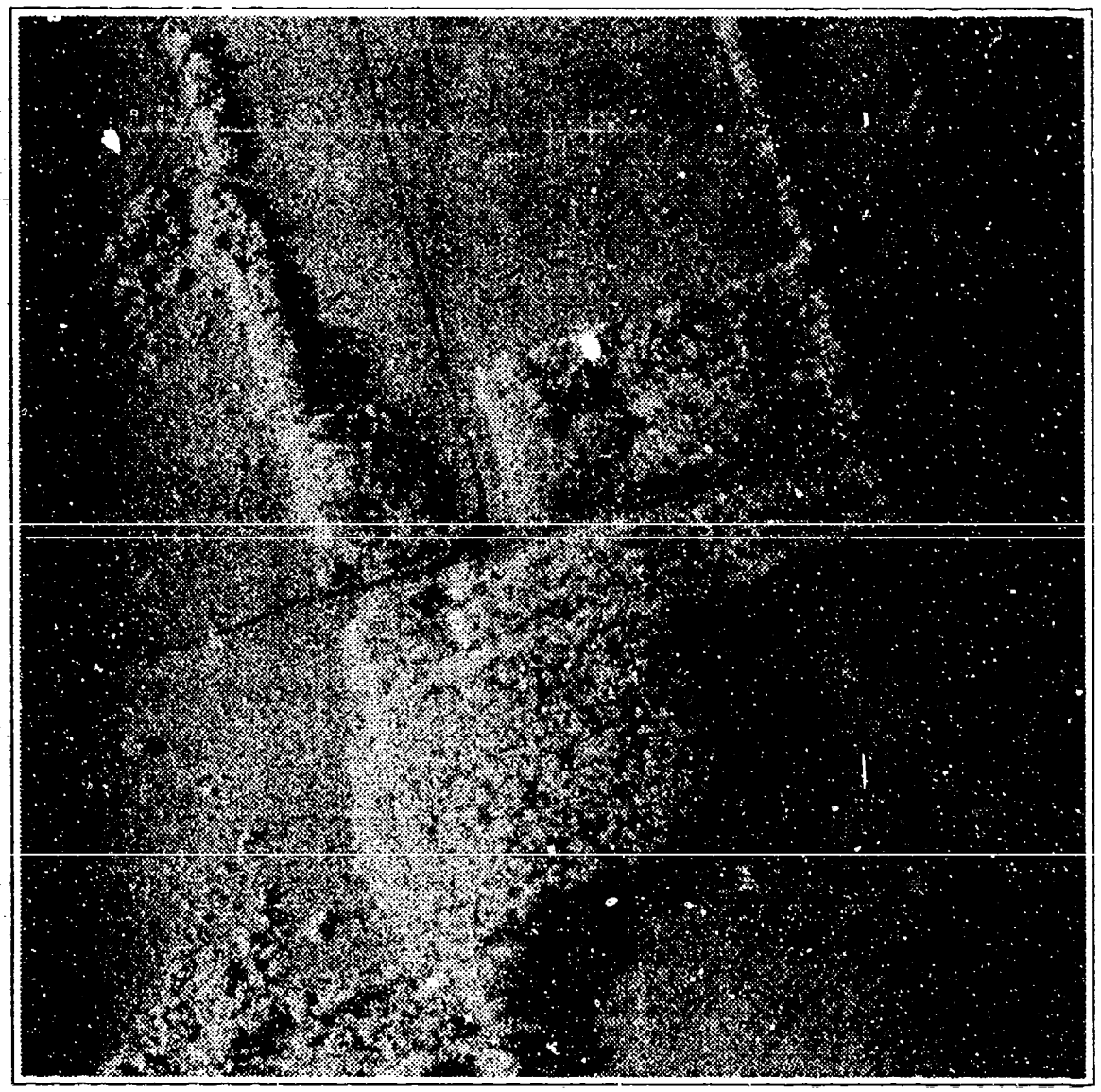


Figure 5.14. Entire Mission 85 2048X2048 SAR Image at 1/16 scale

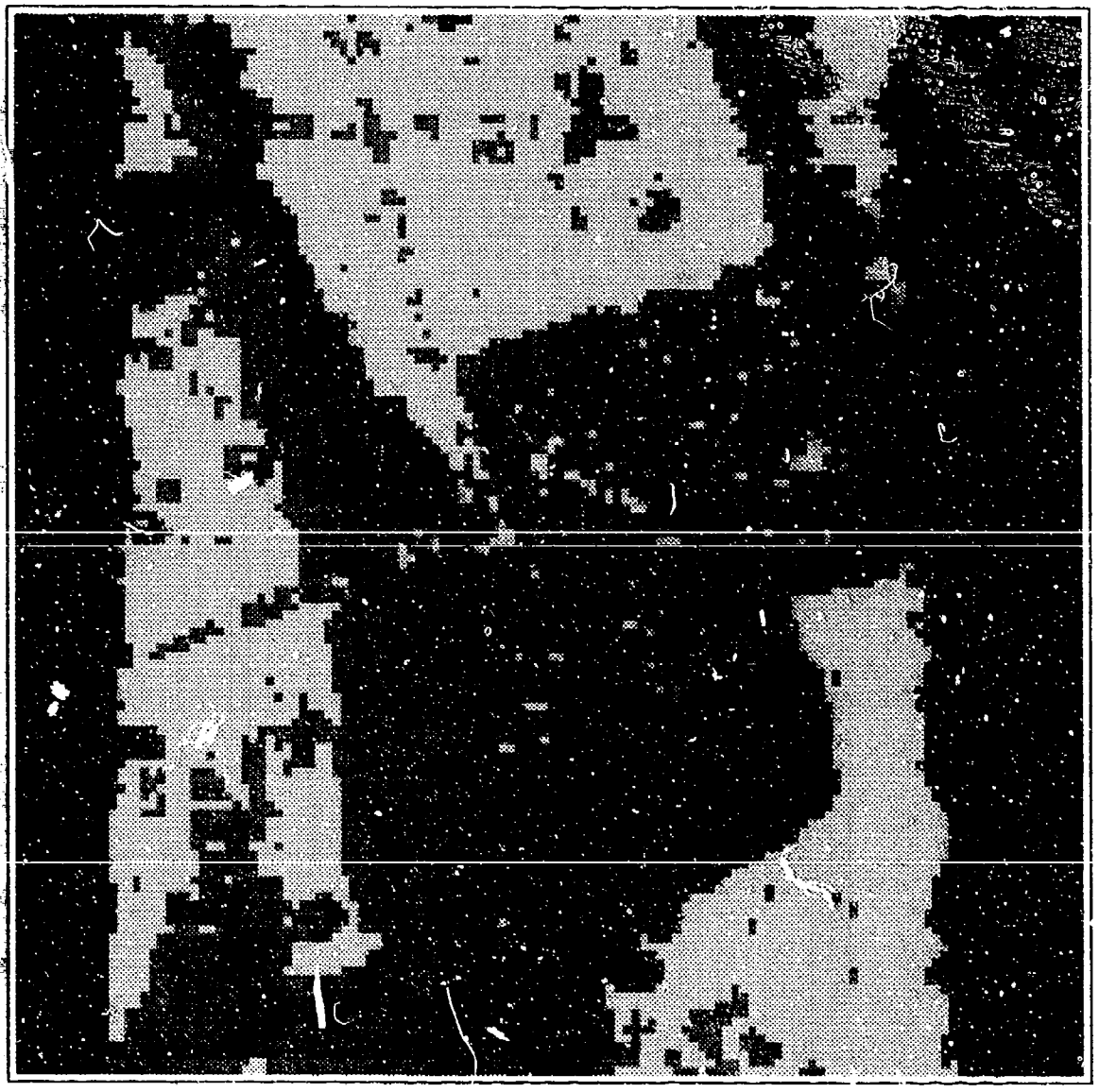


Figure 5.15. Spline Wavelet Segmentation Of Mission 85 SAR Image at 1/16 scale

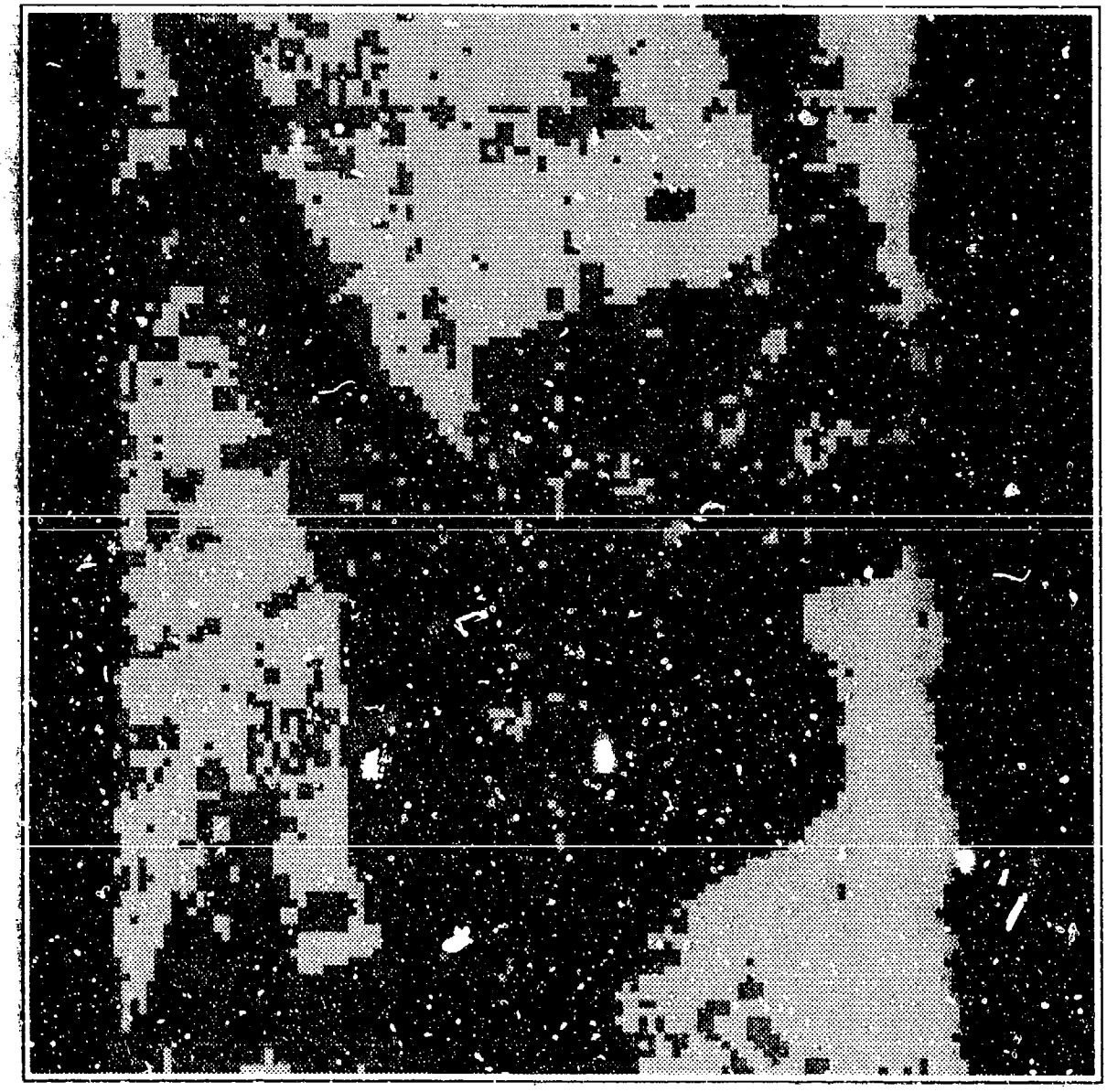


Figure 5.16. Daubechies 2 Wavelet Segmentation Of Mission 85 SAR Image at 1/16 scale

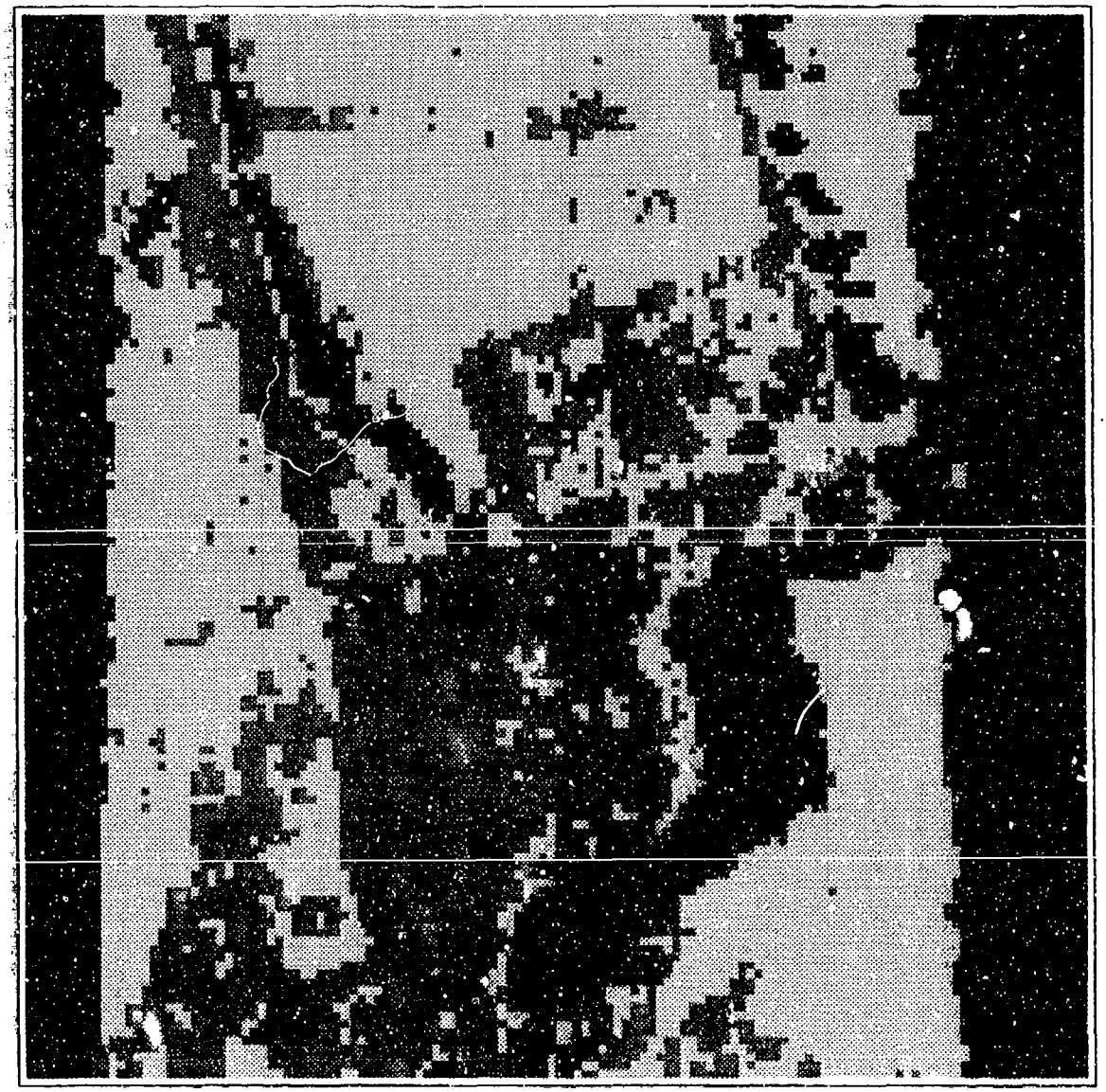


Figure 5.17. Daubechies 3 Wavelet Segmentation Of Mission 85 SAR Image at 1/16 scale

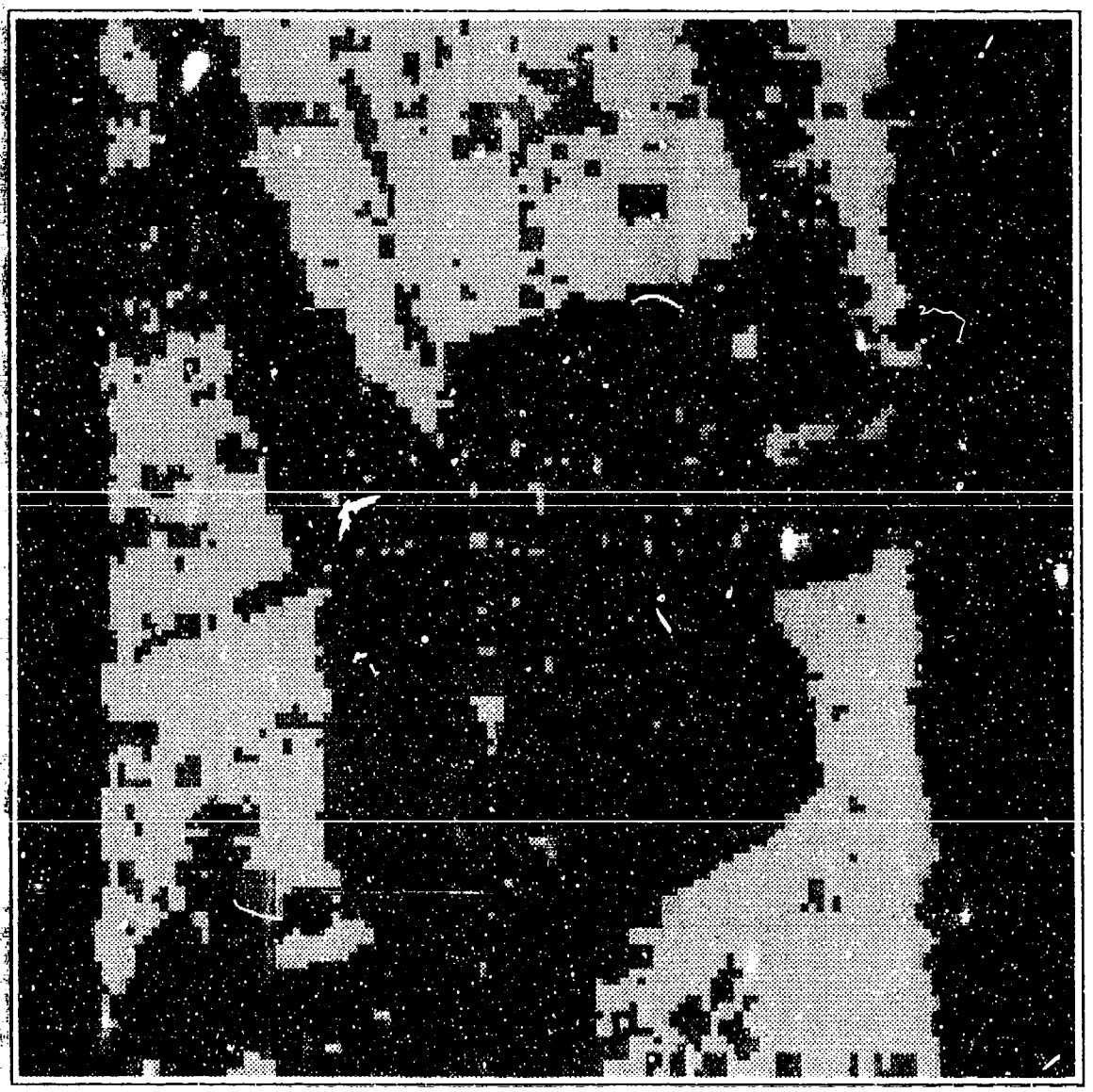


Figure 5.18. Daubechies 6 Wavelet Segmentation Of Mission 85 SAR Image at 1/16 scale

5.5 *Segmentation of Roads*

5.5.1 Selection of Wavelet Coefficients and Receptive Field Size The segmentation of roads utilizing the RBF network was done in the same manner as the segmentation of the homogeneous regions differing only in the approximation level and receptive field size. The section of SAR imagery used to segment roads ,Figure 5.19, came from the upper left hand corner mission 85 (see figure 5.14). The width of the road was measured to be between 8 and 12 feet across. This indicates the the first level approximation coefficients can provide a good set of features to accomplish the segmentation. The first attempt at segmentation done using a 3X3 receptive field. This resulted in finding two distinct edges instead of a solid road. The next attempt was done using a 5X5 receptive field. Using a 5X5 receptive did not provide a perfect segmentation but it was able to segment the road as single object rather than two edges. To provide better results some median filtering was done on the segmented road image. Median filtering is done by moving a window along the image and replacing the pixel which is being processed with the average pixel value in the window. This process eliminates some types of noise, but preserves edge information.

5.5.2 Results Figure 5.19 is the original section of the image without any scaling. Figure 5.20 shows the segmentation done by the radial basis function network using features from level 1. Figure 5.21 is the same segmentation after a median filter was passed over it. The road is admittedly slimmer but stands out as a distinct object.

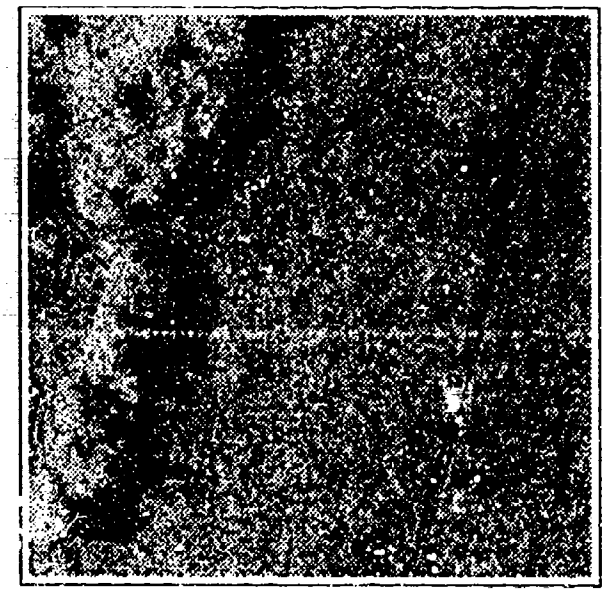


Figure 5.19. Section of M85 With Roads at Full Scale

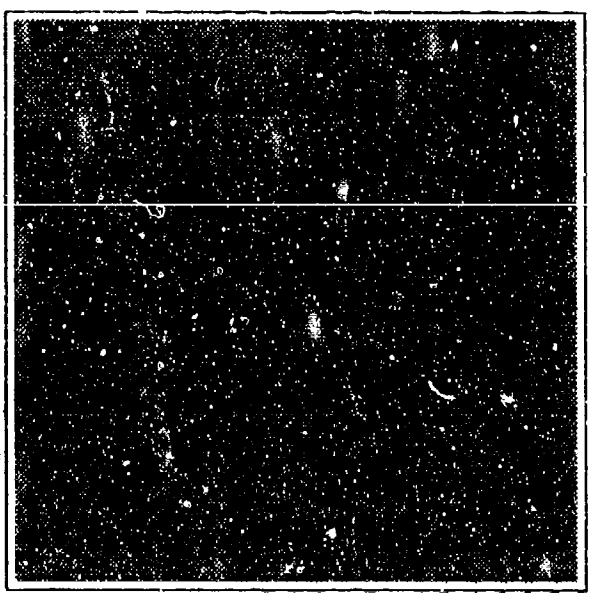


Figure 5.20. Segmentation of Roads from M85

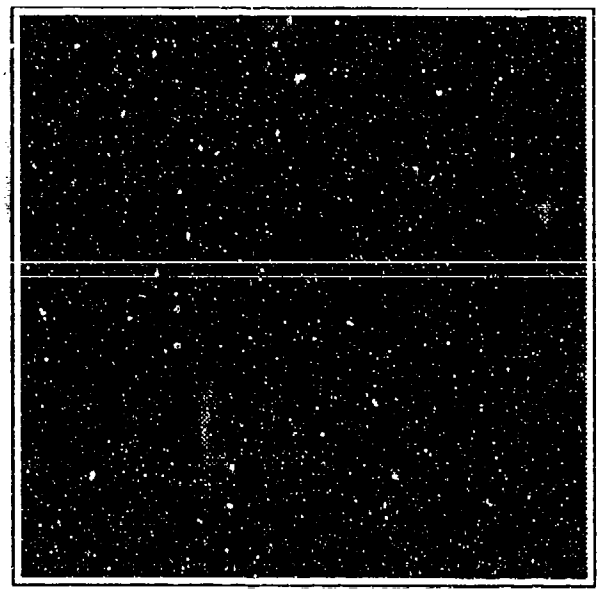


Figure 5.21. Segmentation of Roads Using A 1X3 Median Filter

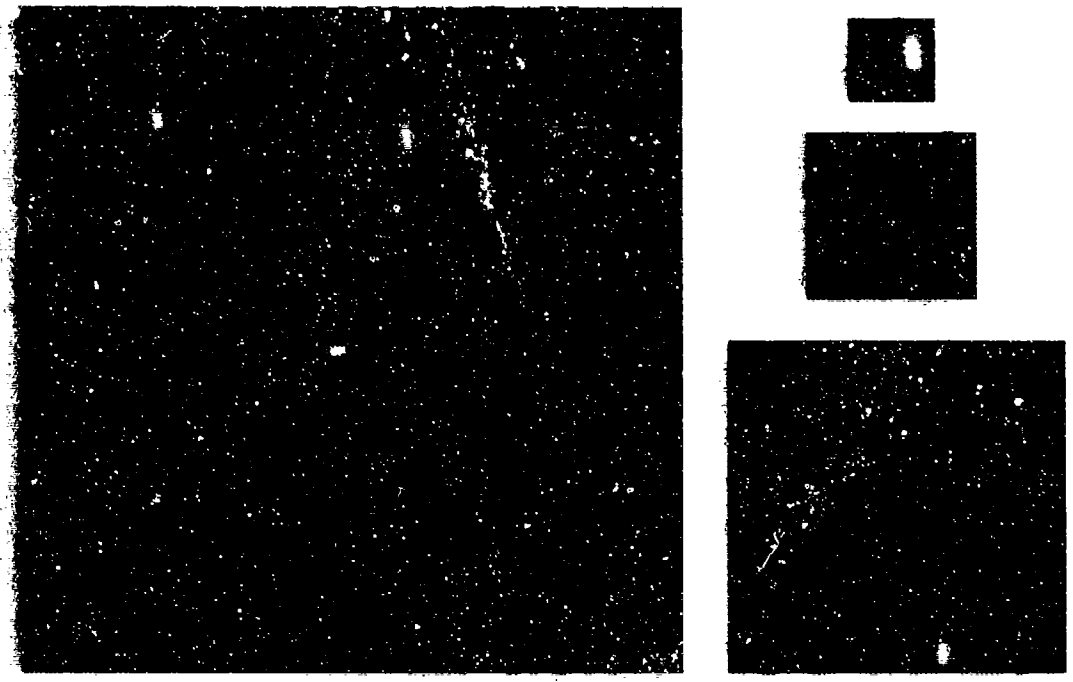


Figure 5.22. FLIR Imagery Multiresolution Approximations Using The Cubic Spline Wavelet (Actual Size)

5.6 Segmentation of Man Made Objects in FLIR Imagery

5.6.1 Selection of Wavelet Coefficients The segmentation of the FLIR imagery utilizing the RBF network was done in the same manner as the segmentation of the homogeneous regions and the roads differing only in the approximation level and receptive field size. The selection of which wavelet coefficient set to use in segmenting smaller man made objects out of an image is a more complicated decision than choosing a set for large homogeneous regions. The decision depends on the size of the features chosen. For example, one could choose to use an entire tank as a feature. This would require a large enough receptive field to cover the entire tank. An alternative approach would use smaller portions of the tank as features. This choice would require a receptive field the size of track wheels or perhaps the size of the engine housing. The correct set of approximation coefficients would then be the set which contains information of at least the size of the features of interest. I chose to use smaller portions of the tank for my features. The size of the features then drove the decision

to use the set of level 1 approximation coefficients (see Figure 5.22). A 3X3 receptive field was large enough to extract the features of interest.

5.7 Results

The results of the segmentation process are shown for six FLIR images along side the original images in Figures 5.23, 5.24, and 5.25. Each of FLIR images contains a tank at different ranges at the center of image. Since range data is available and the size of an object of interest can be determined at a particular range, a less than perfect segmentation such as Figure 5.24 can be useful.

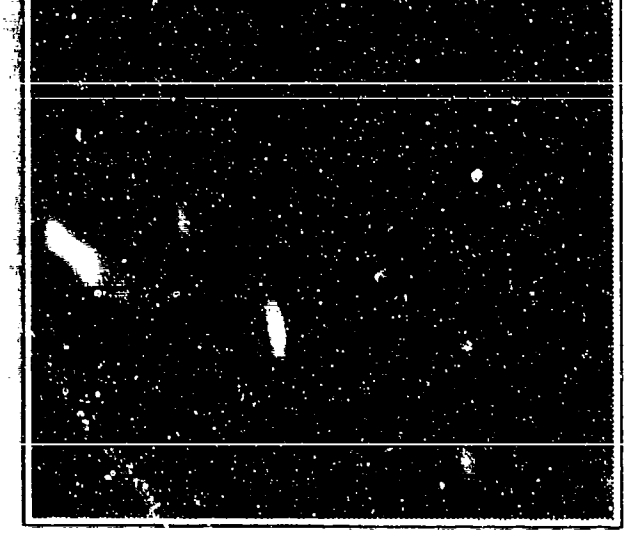


Figure 5.23. FLIR Segmentation at Range 1170 and 1230 Yards

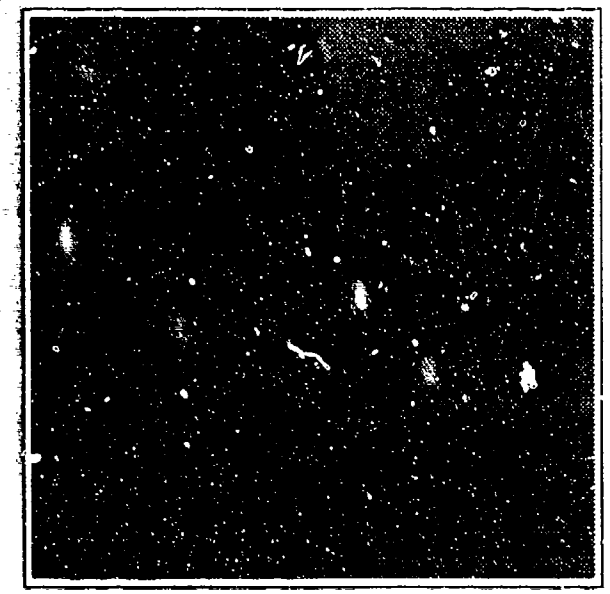


Figure 5.24. FLIR Segmentation at Range 1290 and 1360 Yards

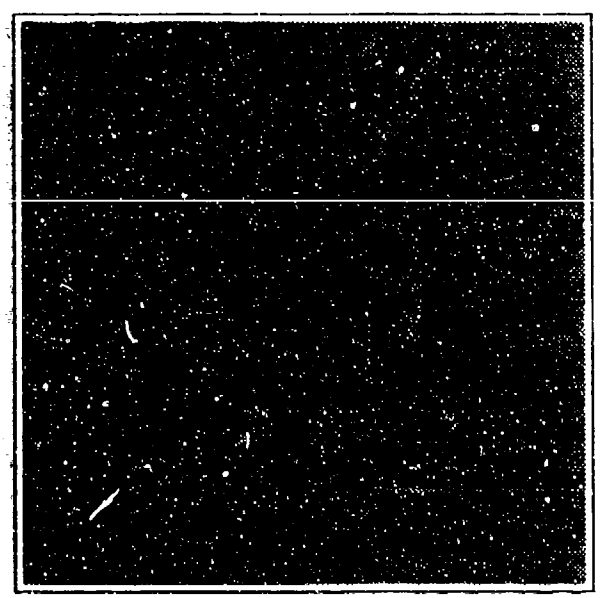


Figure 5.25. FLIR Segmentation at Range 1430 and 1480 Yards

5.8 Conclusions

Segmentation, feature extraction, and classification are the three primary tasks which encompass the pattern recognition process. These task are in general very difficult to accomplish and require large amounts of time and computer processing. In an attempt to provide new approaches to processing and to improve our ability to autonomously recognize patterns, research into biological visual systems has been pursued. This biologically based research has suggested that visual perception is constructed from small pieces of the whole with each small piece having information regarding the frequency content of a small region of the entire image. This approach has been used successfully to segment natural and man-made regions in both SAR and FLIR imagery.

The predominant tool used in image analysis to generate spatial-frequency representation of images has been the 2D Windowed Fourier Transform. It provides a representation of images in the space domain/spatial-frequency domain. However, because this transform utilizes constant sized windows; it obtains a constant resolution in space. A new analysis tool, the Wavelet Transform, provides an alternative by using multiple sized windows. These multiple sized windows trade resolution in space for resolution in spatial-frequency. These space/spatial-frequency representation of small regions of the image have been processed by a Radial Basis Function (RBF) artificial neural network to determine the difference between natural homogeneous regions (trees, fields, and shadow) and man-made objects (tanks and roads).

The Multiresolution Decomposition provides a representation of both SAR and FLIR imagery which shows similarities in homogeneous regions and provides features for segmentation. The level four approximation coefficients can accurately segment trees from fields. This is clearly seen in Figures 5.15,5.16, 5.17 and 5.18 which show the segmentation of naturally occurring homogeneous regions. The higher resolution approximations provided adequate features to segment man made objects. The dirt roads found in the SAR imagery (Figure 5.21) and the tanks found in the FLIR imagery (5.23,5.24, and 5.25) were segmented from the surrounding clutter.

5.9 Summary

The various sets of coefficients which constitute the wavelet representation from a multiresolution analysis provide a broad enough representation to segment both natural and man made objects of varying size from surrounding clutter in both SAR and FLIR imagery.

VI. Conclusions and Recommendations

6.1 Introduction

This thesis research investigated the use of wavelets and artificial neural networks to segment high resolution Synthetic Aperture Radar (SAR) and Forward Looking InfraRed (FLIR) imagery. The specific objective was to find a reliable method to segment or separate naturally occurring regions such as tree, fields, and shadows as well as man-made objects such as tanks and roads.

6.2 Major Findings

The objective of this thesis research was to answer six questions.

- How is the Wavelet transform related to other types of signal or image analysis tools?
- How is the Multiresolution Representation obtained or calculated for a signal or image?
- Do the multiresolution coefficients provide values which can be used to separate natural and man-made regions within both SAR and FLIR imagery?
- Which set of coefficients should be used as the features?
- Can the Radial Basis Function (RBF) artificial neural network be trained to autonomously segment SAR and FLIR imagery using the wavelet coefficients as features?
- Will the RBF neural network segmentation using the multiresolution coefficients as features, generalize to all areas of an image? If so, will it also generalize to additional images not used in network training?

The results of this thesis research have provided a basis for answering the above questions.

- The Wavelet transform trades resolution in space for resolution in spatial/frequency in its analysis of images. This concept and the mathematical basis for wavelet analysis are fully explained in a tutorial manner in Chapter 3.

- There are various methods of generating a multiresolution representation. Chapter 4 explains two methods of producing this representation. Additionally, the Appendix of this document contains the code to accomplish a Wavelet transform.
- The multiresolution approximation coefficients were sufficiently different for both natural regions and man-made objects to accomplish a rudimentary segmentation. Preliminary tests produced segmentation of regions and objects using standard thresholding techniques.
- The approximation coefficients proved to be more useful than the detail coefficients. The different naturally occurring regions in SAR imagery contained multiple orientations and the three orientations of the detail could not be used directly as a means of separating regions. The bandwidths of the various levels of the approximation coefficients provided the decisive element in region segmentation.
- The Radial Basis Function artificial neural network provided a tool which could segment both natural regions and man-made object from surrounding regions and clutter in SAR and FLIR imagery using multiresolution approximation coefficients as features.
- The RBF network was able to generalize to different geographical regions of SAR imagery and to different FLIR images within a series of images of the same object at different ranges.

Additionally, this thesis has provides an excellent introduction and analysis of the multiresolution mathematics and a system to generate the multiresolution representations.

6.3 Recommendations

There are some additional areas for further research regarding the multiresolution representation of imagery and its utility in the pattern recognition process. The RBF network is another area where further study can yield benefits. The following suggestion are made with these two areas in mind.

- The small number of training vectors in Zahirniak's RBF network was a limiting factor on the size of receptive fields which were used. An RBF network without this training vector limitation could provide for using larger receptive fields.
- The Boundary Contour System using detail coefficient as proposed by J. S. Laing in his thesis [19] could provide a means of better segmentation of roads and other man-made objects SAR and FLIR imagery.
- The design of a system which would use both the detail coefficients and the approximation coefficients could provide a means of using both the similarity and discontinuity approaches simultaneously in image segmentation.
- The wavelet representation of image data should be applied to the task of classification or categorization of man-made objects.

6.4 Summary

The ultimate aim of this research was obtained. The wavelet representation of image data is useful for segmentation. The wavelet coefficients should also be explored for their application to other tasks in the pattern recognition process.

Appendix A. *Multiresolution Analysis Using Projections*

A.1 System Description of the WAVE Program

The following is a list of functions which comprise the *wave* program:

1. `main_wave.c` - The main driver program for wave.
2. `loadimage.c` - A routine to load the include image from an ascii data file.
3. `phi_gen_haar.c` - A routine that builds a new Φ for each level of the decomposition.
4. `inner_prod.c` - A routine to perform the inner product and obtains the Φ coefficients. It generates one file for each level of decomposition with the suffix `.phicoef.`.
5. `v_projection.c` - A routine that finds the projection of the include image on the space V_m where m is the current level of decomposition. It generates one file for each level of decomposition with the suffix `.v_project.`.
6. `w_projection.c` - A routine that finds the projection of the include image onto the space W_m orthogonal to the V_m space where m is the current level of decompostion. It generates one file for each level of decomposition with the suffix `.w_project.`.
7. `makefile` - A makefile that is used to compile and link the source code to make an executable file.
8. `jsmacros.h` - An include file that contains macros we found useful in our programming environment. This file must be present in the directory where compilation takes place (See Appendix F.1 for listing).
9. `macros.h` - An include file that we borrowed from G. Tarr. It contains addition macros used throughout our code. It also must be present in the directory where the compilation takes place (See Appendix F.1 for listing).
10. `stewmath.h` - An include file containing some math routines specific to our program. It must be present in the directory where complilation takes place (See Appendix F.2 for listing).

Typing *"make"* at the command prompt in any directory with all of the above files present will create the appropriate object code and an executable file called *wave* that may be executed by typing *"wave"* at the command prompt.

A.2 Haar Wavelet Analysis Software

A.2.1 Listing of MAIN-WAVE.C

```
/*****************************************************************************/
/*****************************************************************************/
/***            WAVELET ANALYZER MAIN PROGRAM DRIVER                       **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  09 April 91                                                       */
/*                                                                           */
/*  VERSION: 1.0                                                             */
/*                                                                           */
/*  NAME:  main-wave.c                                                       */
/*                                                                           */
/*  DESCRIPTION:  This program performs a multiresolution wavelet analysis   */
/*  of an input image with a wavelet from its internal library chosen        */
/*  interactively by the user. It handles the menu interface with the        */
/*  user and drives the subroutines that take inputs, analyzes, and          */
/*  produces output. Currently only the Haar Wavelet is available for this   */
/*  program.                                                                 */
/*                                                                           */
/*  FILES READ: NONE                                                         */
/*                                                                           */
/*  FILES WRITTEN:  NONE                                                     */
/*                                                                           */
/*  HEADERS USED:   <stdio.h>, "macros.h", "jsmacros.h"                      */
/*                                                                           */
/*  CALLING PROGRAMS:   NONE                                                 */
/*                                                                           */
/*  PROGRAMS CALLED: imageload.c, innerprod.c, phi_gen_haar.c,               */
/*                   phi_gen_pl.c, vproj.c, wproj.c                          */
/*                                                                           */
/*  AUTHOR: Steve Smiley and J. Stewart Laing                                */
/*                                                                           */
/*  HISTORY: Initial Version; adapted from phiv1.c and haarv1.c              */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include <stdio.h>
#include "macros.h"
#include "jsmacros.h"
#include "stewmath.h"

int_array    loadimage();
float_array  phi_gen_haar();
int_array    inner_prod();
int_array    v_projection();
int_array    w_projection();

/************************/
```

```
/*  MAIN PROGRAM BODY  */
/***********************/
void main(argc, argv)
   int  argc;
   char *argv[];
{
/************************/
/* initialize variables */
/************************/
int         i, wavelet_type, level, maxlevel;
int_array   image, phi_coef, v_image, lastv_image, w_image;
float_array phi;
char        filename[64], load;
/************************************/
/*  load image to be analyzed       */
/************************************/
if(argc != 4 && argc != 1){
   printf("Usage: wave <filename> <# of Rows> <# of Cols>\n");
   exit(0);
}
image = loadimage(filename, argc, argv);
maxlevel = LOG2(image.ROW);

   /************************************/
   /* This section performs the wavelet*/
   /* analysis on the image according  */
   /* to the value of wavelet_type.    */
   /************************************/

        loopi(maxlevel){

        /************************************/
        /*  generate phi for haar           */
        /************************************/

        phi = phi_gen_haar(i);

        printf("\n Level %d phi generated.\n", i);

        /*************************************************/
        /*  perform inner product to get phi coeficients  */
        /*************************************************/

phi_coef = inner_prod(image, phi, i, filename);
        printf("\n I have created and strored the Level %d", i);
        printf(" inner_product coeficients.\n");

        /*********************************/
        /*  generate V space projections  */
        /*********************************/

lastv_image = v_image;
        v_image = v_projection(image, phi, phi_coef, i, filename);
        printf("\n I have created and stored the Level %d", i);
        printf(" V projection.\n", level);

        /*********************************/
        /*  generate W space projections  */
        /*********************************/

        if (i == 1) w_image = w_projection(image, v_image, i, filename);
if (i > 1) w_image = w_projection(lastv_image, v_image,  ,
           filename);
      }

/* THE END */
}
```

A.2.2 Listing of LOADIMAGE.C

```
/*****************************************************************************/
/*****************************************************************************/
/***            WAVELET ANALYZER LOADIMAGE ROUTINE                         ***/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  10 April 91                                                       */
/*                                                                           */
/*  VERSION: 1.0                                                             */
/*                                                                           */
/*  NAME:  loadimage.c                                                       */
/*                                                                           */
/*  DESCRIPTION:  This routine loads an image into an array whose name is    */
/*  specified by the user interactively. It is intended to be used as a      */
/*  subroutine for the WAVELET ANALYZER PROGRAM.                             */
/*                                                                           */
/*  FILES READ: One file specified by the user.                              */
/*                                                                           */
/*  FILES WRITTEN:  NONE                                                     */
/*                                                                           */
/*  HEADERS USED:   <stdio.h>, "macros.h", <stdlib.h>, "jlmacros.h"          */
/*                                                                           */
/*  CALLING PROGRAMS:   main-wave.c                                          */
/*                                                                           */
/*  PROGRAMS CALLED:   NONE                                                  */
/*                                                                           */
/*  AUTHOR: Steve Smiley and J. Stewart Laing                                */
/*                                                                           */
/*  HISTORY: Initial Version                                                 */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include    <stdlib.h>
#include    <stdio.h>
#include    "macros.h"
#include    "jsmacros.h"

/************************/
/*  FUNCTION BODY       */
/************************/

int_array loadimage(infilename, argc, argv)
   char   *infilename[64];
   int    argc;
   char   *argv[];
{
   /************************/
   /* initialize variables */
   /************************/

   int       i,j;
   FILE      *infile;
   int_array image;

   /*******************************************/
   /*  create array to hold the incoming image  */
   /*******************************************/
   if(argc == 1){
      printf("\n\n\n  Input the size of the image (ROW COLUMN):>");
      scanf("%d %d", &image.ROW, &image.COL);
      printf("  \n\n Input filename of image to be analyzed:>");
      scanf("%s", infilename);
   }
```

```
   else {
      sprintf(infilename, "%s", argv[1]);
      sscanf(argv[2], "%d", &image.ROW);
      sscanf(argv[3], "%d", &image.COL);
   }
   CREATE_MATRIX_ROW(image.array, image.ROW, int);
   CREATE_MATRIX_COL(image.array, image.ROW, image.COL, int);
   /*************************************/
   /*  load image to be analyzed        */
   /*************************************/

   OPEN_FILE (infile, infilename, "The wavelet analyzer");
   loopij(image.ROW, image.COL){
         fscanf(infile,"%d", &image.array[i][j]);
   }
      printf("\n ** The image %s has been loaded for processing. **\n\n\n",
         infilename);
   return image;
}
```

A.2.3 Listing of PHI_GEN_HAAR.C

```
/*****************************************************************************/
/*****************************************************************************/
/***      WAVELET ANALYZER ROUTINE TO GENERATE THE PHI FOR HAAR            ***/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  11 April 91                                                       */
/*                                                                           */
/*  VERSION: 1.0                                                             */
/*                                                                           */
/*  NAME:  phi_gen_haar.c                                                    */
/*                                                                           */
/*  DESCRIPTION:  This routine generates the phi function for a particular   */
/*  level of resolution. It is represented as an array whose size depends    */
/*  on the level requested by the calling function.                          */
/*                                                                           */
/*  FILES READ: NONE                                                         */
/*                                                                           */
/*  FILES WRITTEN:  NONE                                                     */
/*                                                                           */
/*  HEADERS USED:   <stdio.h>, "macros.h", <stdlib.h>, "jlmacros.h"          */
/*                                                                           */
/*  CALLING PROGRAMS:   main-wave.c                                          */
/*                                                                           */
/*  PROGRAMS CALLED:   NONE                                                  */
/*                                                                           */
/*  AUTHOR: Steve Smiley and J. Stewart Laing                                */
/*                                                                           */
/*  HISTORY: Initial Version                                                 */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include    <stdlib.h>
#include    <stdio.h>
#include    "macros.h"
#include    "jsmacros.h"

/************************/
/*  FUNCTION BODY       */
/************************/
```

```
float_array phi_gen_haar(level)
   int level;
{
   /************************/
   /* initialize variables */
   /************************/
   int i,j, phisize;
   float_array phi;
   /*******************************************/
   /*  create array to hold phi               */
   /*******************************************/
   phisize = 1;
   for(i=0; i < level; ++i) phisize *= 2;
   phi.ROW =  phisize;
   phi.COL =  phisize;
   CREATE_MATRIX_ROW(phi.array, phi.ROW, float);
   CREATE_MATRIX_COL(phi.array, phi.ROW, phi.COL, float);
   /**************/
   /*  build phi */
   /**************/
   loopij(phi.ROW,phi.COL) phi.array[i][j] = 1.0/(float)phisize;
   return phi;
}
```

A.2.4 Listing of INNER_PROD.C

```
/*****************************************************************************/
/*****************************************************************************/
/***    ROUTINE TO PERFORM INNER PRODUCT FOR WAVELET ANALYZER             *****/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  11 April 91                                                       */
/*                                                                           */
/*  VERSION: 1.0                                                             */
/*                                                                           */
/*  NAME:  inner_prod.c                                                      */
/*                                                                           */
/*  DESCRIPTION:  This routine performs the inner product between the phi    */
/*  and the image at any valid level as requested by the caller.             */
/* It is intended as a subroutine for the WAVELET ANALYZER PROGRAM.          */
/*                                                                           */
/*  FILES READ: NONE.                                                        */
/*                                                                           */
/*  FILES WRITTEN:  A file will be generated each time the                   */
/*  routine is called. The name of the file will depend on the input         */
    image filename, the type of wavelet used, and the level of resolution.  */
                                                                             */
/*  HEADERS USED:   <stdio.h>, "macros.h", <stdlib.h>, "jlmacros.h",         */
/*                  <string.h>                                               */
/*                                                                           */
/*  CALLING PROGRAMS:   main-wave.c                                          */
/*                                                                           */
/*  PROGRAMS CALLED:   NONE                                                  */
/*                                                                           */
/*  AUTHOR:  Steve Smiley and J. Stewart Laing                               */
/*                                                                           */
/*  HISTORY: Initial Version                                                 */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/

/************************/
```

```
/*  DECLARATION SECTION  */
/***********************/

#include    <stdlib.h>
#include    <stdio.h>
#include    "macros.h"
#include    "jsmacros.h"
#include    <string.h>

/*************************/
/*  FUNCTION BODY       */
/*************************/

int_array inner_prod(image, phi, level, filename)
   int_array   image;
   float_array phi;
   int         level;
   char        filename[64];
{
int       i, j, phisize;
int_array phi_coef;
FILE      *outfile;
char      coeffile[64];
float     product;

   /************************************************/
   /*  create a matrix to hold the phi coeficients */
   /************************************************/

   phisize = 1;
   for(i=0; i < level; ++i) phisize *= 2;
   phi_coef.ROW = image.ROW/phisize;
   phi_coef.COL = image.COL/phisize;
   CREATE_MATRIX_ROW(phi_coef.array, phi_coef.ROW, int);
   CREATE_MATRIX_COL(phi_coef.array, phi_coef.ROW, phi_coef.COL, int);
   /*printf("\nphi_coef matrix sucessfully created.\n");*/

   /************************************************************/
   /*  perform inner product <image, phi> to get coeficients */
   /************************************************************/

   loopij(image.ROW, image.COL){
         product = phi.array[i%phisize][j%phisize] * (float)image.array[i][j];
         phi_coef.array[i/phisize][j/phisize] += (int)product;
   }

   /*************************************************/
   /* write the phi coeficient array out to a file      */
   /*************************************************/

   sprintf(coeffile, "%s.phicoef.%d", filename, level);

   CREATE_FILE(outfile, coeffile, "WAVELET ANALYZER")

   loopij(phi_coef.ROW,phi_coef.COL)
         fprintf(outfile, "%d\n", phi_coef.array[i][j]);

  printf("\n The level %d phi_coeficients have been stored in a file",level);
  printf(" called: %s\n", coeffile);
  return phi_coef;
}
```

A.2.5 Listing of V_PROJECTION.C

```
/*************************************************************************/
/*************************************************************************/
/***   ROUTINE TO PERFORM THE V_PROJECTION FOR WAVELET ANALYZER        *****/
/*************************************************************************/
/*************************************************************************/
/*  DATE:  15 April 91                                                   */
```

```
/*                                                                          */
/*  VERSION: 1.0                                                            */
/*                                                                          */
/*  NAME:  v_projection.c                                                   */
/*                                                                          */
/*  DESCRIPTION:  This routine performs the inner product between the phi   */
/*  and phi coeficient of the image at any valid level as requested by      */
/*  the caller.                                                             */
/*  It is intended as a subroutine for the WAVELET ANALYZER PROGRAM.        */
/*                                                                          */
/*  FILES READ: NONE.                                                       */
/*                                                                          */
/*  FILES WRITTEN:  A file will be generated each time the routine is       */
/*  routine is called. The name of the file will depend on the input        */
/*  mage filename, the type of wavelet used, and the level of resolution.  */
/*                                                                          */
/*  HEADERS USED:   <stdio.h>, "macros.h", <stdlib.h>, "jlmacros.h",        */
/*                  <string.h>                                              */
/*                                                                          */
/*  CALLING PROGRAMS:   main-wave.c                                         */
/*                                                                          */
/*  PROGRAMS CALLED:   NONE                                                 */
/*                                                                          */
/*  AUTHOR: Steve Smiley and J. Stewart Laing                               */
/*                                                                          */
/*  HISTORY: Initial Version                                                */
/*                                                                          */
/****************************************************************************/
/****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include    <stdlib.h>
#include    <stdio.h>
#include    "macros.h"
#include    "jlmacros.h"
#include    <string.h>
#include    <math.h>

/************************/
/*  FUNCTION BODY       */
/************************/

int_array v_projection(image, phi, phi_coef, level, filename)
   int_array    image, phi_coef;
   float_array         phi;
   char          filename[64];
{
int_array  v_image;
int     i, j, phisize;
FILE     *outfile;
char      vprojfile[64];

 v_image.ROW = image.ROW;
 v_image.COL = image.COL;

CREATE_MATRIX_ROW(v_image.array, v_image.ROW, int);
CREATE_MATRIX_COL(v_image.array, v_image.ROW, v_image.COL, int);

phisize = (int)pow(2.0, (double)level);

printf("The phisize is %d\n", phisize);

 sprintf(vprojfile, "%s.v_project.%d", filename, level);

   CREATE_FILE(outfile, vprojfile, "WAVELET ANALYZER")

   loopij(v_image.ROW,v_image.COL){
      v_image.array[i][j] = (int)((phi.array[i%phisize][j%phisize])*
      ((float)phi_coef.array[i/phisize][j/phisize]));
```

```
      fprintf(outfile,"%d\n", v_image.array[i][j]);
   }
/*****************************************************/
/* write the v projection array out to a file        */
/*****************************************************/

   printf("\n The level %d V projections have been stored in a file",level);
   printf(" called: %s\n", vprojfile);
   return v_image;
   }
```

A.2.6 Listing of W_PROJECTION.C

```
/*****************************************************************************/
/*****************************************************************************/
/***   ROUTINE TO PERFORM THE W_PROJECTION FOR WAVELET ANALYZER        *****/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  15 April 91                                                       */
/*                                                                           */
/*  VERSION: 1.0                                                             */
/*                                                                           */
/*  NAME:  w_projection.c                                                    */
/*                                                                           */
/*  DESCRIPTION:  This routine calculates the W space projections by         */
/*  performing a point for point subtraction with the two adjacent           */
/*  V space projections.  It is intended as a subroutine for the             */
/*  WAVELET ANALYZER PROGRAM.                                                */
/*                                                                           */
/*  FILES READ: NONE.                                                        */
/*                                                                           */
/*  FILES WRITTEN:  A file will be generated each time the routine is        */
/*  routine is called. The name of the file will depend on the input         */
/*  image filename, the type of wavelet used, and the level of resolution.  */
/*                                                                           */
/*  HEADERS USED:   <stdio.h>, "macros.h", <stdlib.h>, "jlmacros.h",         */
/*                  <string.h>                                               */
/*                                                                           */
/*  CALLING PROGRAMS:   main-wave.c                                          */
/*                                                                           */
/*  PROGRAMS CALLED:   NONE                                                  */
/*                                                                           */
/*  AUTHOR: Steve Smiley and J. Stewart Laing                                */
/*                                                                           */
/*  HISTORY: Initial Version                                                 */
/*                                                                           */
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include    <stdlib.h>
#include    <stdio.h>
#include    "macros.h"
#include    "jsmacros.h"
#include    <string.h>
#include    <math.h>

/*************************/
/*  FUNCTION BODY       */
/*************************/

int_array w_projection(lastv_image, v_image, level, filename)
   int_array    lastv_image, v_image;
```

```
   int          level;
   char         filename[64];
{
int_array  w_image;
int        i, j, phisize;
FILE       *outfile;
char       wprojfile[64];
 w_image.ROW = v_image.ROW;
 w_image.COL = v_image.COL;
CREATE_MATRIX_ROW(w_image.array, w_image.ROW, int);
CREATE_MATRIX_COL(w_image.array, w_image.ROW, w_image.COL, int);
sprintf(wprojfile, "%s.w_project.%d", filename, level);
CREATE_FILE(outfile, wprojfile, "WAVELET ANALYZER")
loopij(w_image.ROW,w_image.COL){
      w_image.array[i][j] = lastv_image.array[i][j] - v_image.array[i][j];
/*      w_image.array[i][j] += 255;
      w_image.array[i][j] /= 2;*/
 /*****************************************************/
 /* write the w projection array out to a file        */
 /*****************************************************/
      fprintf(outfile,"%d\n", w_image.array[i][j]);
}
   printf("\n The level %d W projections have been stored in a file",level);
   printf(" called: %s\n", wprojfile);
   return w_image;
   }
```

A.2.7 Listing of JSMACROS.H (See Appendix F.1)

A.2.8 Listing of MACROS.H (See Appendix F.1)

A.2.9 Listing of STEWMATH.H (See Appendix F.1)

A.2.10 Listing of MAKEFILE

```
# Makefile routine for the WAVE program by Laing and Smiley.
OBJS = main-wave.o loadimage.o phi_gen_haar.o inner_prod.o \
       v_projection.o w_projection.o
wave: $(OBJS)
@echo "linking ..."
cc $(OBJS) -o wave -lm
main-wave.o: main-wave.c
cc -c main-wave.c
loadimage.o: loadimage.c
cc -c loadimage.c
phi_gen_haar.o: phi_gen_haar.c
cc -c phi_gen_haar.c
inner_prod.o: inner_prod.c
cc -c inner_prod.c
v_projection.o: v_projection.c
cc -c v_projection.c
w_projection.o: w_projection.c
cc -c w_projection.c
```

Appendix B. *Multiresolution Analysis Using Filters*

B.1 2D System Description

The following is a list of functions which comprise the *wave2* program.

1. `main_wave.c` - The main driver program for wave.
2. `loadimage.c` - A routine to load the input image from an ascii data file.
3. `decompose.c` - A routine that controls the decomposition.
4. `reconstruct.c` - A routine that controls the reconstruction.
5. `filters.c` - A routine that provides the coefficient values of the $h(n)$ and $g(n)$ response functions.
6. `convolve.c` - A routine that controls the convolutions for decomposition.
7. `reconvolve.c` - A routine that controls the convolutions for reconstruction.
8. `spconvlv.c` - A routine that performs the spatial convolutions.
9. `makefile` - A makefile that is used to compile and link the source code to make an executable file.
10. `jsmacros.h` - An include file that contains macros we found useful in our programming environment. This file must be present in the directory where compilation takes place (See Appendix F.2 for listing).
11. `stewmath.h` - An include file containing some math routines specific to our program. It must be present in the directory where complilation takes place (See Appendix F.2 for listing).
12. `nrutil.c` - Source code that contains utility macros for dynamic memory allocation (See Appendix F.2 for listing).

Typing *"make"* at the command prompt in any directory with all of the above files present will create the appropriate object code and an executable file called *wave2* that may be executed by typing *"wave2"* at the command prompt.

The intended input to the program is a 2D image in raw ascii format in which each sample of the image is stored in a file, one number per line. For example, an image that is 512x512 samples will consist of 262,144 lines each with one decimal integer number representing the grey scale value of that sample. The grey scale values range from 0 to 255. The output of the program are ascii files representing the scale and detail wavelet coefficients in floating point format. For an in depth explanation of the these coefficients and the algorithm, see the author's theses. The algorithm implemented in this program is taken from a paper by Stephan Mallat. The paper is referenced in the authors theses. Be aware that we found some printing mistakes in the paper which are addressed in our theses. The program was developed on Sun sparcstation 2's. But, it should compile on any system with an ansi standard C compiler. To compile the program, type make at the command prompt with the default directory set to the current directory. Object files will then be created and linked into an executable file called wave2. Then to run the program, type wave2 at the command prompt. A menu should appear first with four choices. If not done at the command line entry into the program, a file must be loaded from the current directory before either decomposition or reconstruction can be executed. Once a file is loaded the Decomposition can be selected. Then the Reconstruction can be selected. The Reconstruction portion depends on files generated by the Decomposition portion. But, it is not necessary to run the Decomposition during the same session as the Reconstruction as long as the Decompostion was run in a prior session and the files still reside in the current directory. An alternate way to start the program is to type wave2 followed by the name of the input file and its size. The size of the input file must be a power of two and is defined to be the length along one dimension of the sampled image. At this time the largest file used is a 512 by 512 sampled image. It is possible to specify the path to an input file that is not in the current directory iether relative

to the current directory or absolutely from the root. However, if this is done, the output files will be sent to that same directory. To review the usage of wave2 is

```
command prompt: wave2 [infilename] [size]
```

The infilename and size are optional but if the infilename is given its size along one dimension of the square power of two sampled image must be given as well.

Also, only one file may be input in any one session. This fact is not obvious from the program menu, so be aware. If you try to select the Load image option from the main menu after you have already loaded a file, the result has not been fully characterized. In other words, we haven't tried to figure out what would happen. This menu option is provided as an alternative to specifying the file on the command line.

The filters available are presently limited to the some of the Daubechies wavelets and the Cubic Spline wavelet. But it is a simple process to add new filters to the filters c program in the same fasion as those already included. To generate the H and G filters, see our theses for references.

B.2 2D Multiresolution Wavelet Analysis Software

B.2.1 Listing of MAIN-WAVE.C

```
/*************************************************************************/
/*************************************************************************/
/***            WAVELET ANALYZER MAIN PROGRAM DRIVER                   **/
/*************************************************************************/
/*************************************************************************/
/*  DATE:  09 April 91, 18 June 91
    VERSION: 2.0
    NAME:  main-wave.c
    DESCRIPTION:  This program performs a multiresolution wavelet analysis
    of an input image with a wavelet from its internal library chosen
    interactively by the user. It handles the menu interface with the
    user and drives the subroutines that take input, analyze, and produce
    output. The the wavelet decomposition algorithm is a pyramid algorithm
    proposed by Stephan Mallat in A Theory for Multiresolution Signal
    Decomposition: The Wavelet Representation published in IEEE Trans.
    on Pattern Anal. and Machine Intel. July 89. The algorithm uses a pair
    of mirror filters derived from the scaling function, phi(x). The user
    may enter the intended input image file from the command line following
    the calling command 'wave' or the user may wait to be prompted for
    the input file name and size after starting the program with the same
    command. In any case, additional images may be entered for processing
    by selecting the appropriate option from the program's main menu.
    FILES READ: NONE  (A subroutine reads the input files.)
```

```
    FILES WRITTEN:  NONE (Subroutines write out the saved data in files.)
    HEADERS USED:   <stdio.h>, "jsmacros.h", "stewmath.h"
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED: imageload.c, reconstruct.c, decompose.c
    AUTHOR: Steve Smiley and J. Stewart Laing
    HISTORY: Initial Version; adapted from phiv1.c and haarv1.c
    Version 2.0 was a rewrite to change the basic algorithm from the using
    inner products to using the Mallat algorithm referenced above.
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include <stdio.h>
#include "jsmacros.h"
#include "stewmath.h"

int_array    loadimage();
void         reconstruct();
void         decompose();

/************************/
/*  MAIN PROGRAM BODY  */
/************************/

void main(argc, argv)
   int  argc;
   char *argv[];
{
/************************/
/* initialize variables */
/************************/

int         selection;
int_array   image, *imagepointer = &image;
char        filename[64];

/************************************/
/*  load image to be analyzed       */
/************************************/

if(argc != 3 && argc != 1){
   printf("Usage: wave <filename> <# of Rows> <# of Cols>\n");
   exit(0);
   }

if(argc == 3){
   image = loadimage(filename, argc, argv);
   /*printf("returned from loadimage"); fflush(stdout);*/
   }

do {

      /************************************/
      /*  display menu                    */
      /************************************/

   printf("\n\n    MAIN MENU\n\n");
   printf("      1  =  Load a new image from disk.\n");
   printf("      2  =  Perform Wavelet Decomposition.\n");
   printf("      3  =  Perform Wavelet Reconstruction.\n");
   printf("      4  =  Exit Program.\n\n");
   printf("   Enter an integer (1-4):");

   scanf("%d", &selection);
```

```
   if (selection == 4) break;        /* Quit program */
   argc = 1;
   if (selection == 1) image = loadimage(filename, argc, argv);
   else if (selection == 2) decompose(imagepointer, filename);
   else if (selection == 3) reconstruct(imagepointer,
      filename);
   else {
      printf(" \n\n Just enter an integer from 1 to 4 and");
      printf("press return. \n");
      }
   } while (selection != 4);
/* THE END */
}
```

B.2.2 Listing of LOADIMAGE.C

```
/*****************************************************************************/
/*****************************************************************************/
/***            WAVELET ANALYZER LOADIMAGE ROUTINE                         **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  10 April 91
    VERSION: 1.1
    NAME:  loadimage.c
    DESCRIPTION:  This routine loads an image into an array whose name is
    specified by the user interactively. It is intended to be used as a
    subroutine for the wave2 program.
    FILES READ: One file specified by the user.
    FILES WRITTEN:  NONE
    HEADERS USED:   <stdio.h>, "jsmacros.h"
    CALLING PROGRAMS:   main-wave.c
    PROGRAMS CALLED:   NONE
    AUTHOR: Steve Smiley and J. Stewart Laing
    HISTORY: Version 1.1 was changed to accept square matrices
             only.
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/
#include    <stdio.h>
#include    "jsmacros.h"
int         **imatrix();
void        free_imatrix();

/*************************/
/*  FUNCTION BODY       */
/*************************/
int_array loadimage(infilename, argc, argv)
   char   *infilename[64];
   int    argc;
   char   *argv[];
{
   /*************************/
   /* initialize variables */
```

```
   /************************/

   int       i,j;
   FILE      *infile;
   int_array image;
   /*****************************************/
   /*  create array to hold the incoming image  */
   /*****************************************/
   if(argc == 1){
      printf("\n\n Input filename of image to be analyzed:");
      scanf("%s", infilename);
      printf("\n\n Input the number of Rows in the square matrix");
      printf("\n data file. (The number must a power of 2):");
      scanf("%d", &image.ROW);
      image.COL = image.ROW;
   }
   else {
      sprintf(infilename, "%s", argv[1]);
      sscanf(argv[2], "%d", &image.ROW);
      image.COL = image.ROW;
   }
   image.array = imatrix(1, image.ROW, 1, image.COL);
   /*************************************/
   /*  load image to be analyzed        */
   /*************************************/

   OPEN_FILE (infile, infilename, "The wavelet analyzer");
   loopij(image.ROW, image.COL)
         fscanf(infile,"%d", &image.array[i+1][j+1]);
      CLOSE_FILE (i, infilename, "The Wavelet analyzer", infile)
      printf("\n ** The image %s has been loaded for processing. **\n\n\n",
         infilename);
   return image;
}
```

B.2.3 Listing of DECOMPOSE.C

```
/*****************************************************************************/
/*****************************************************************************/
/***          WAVELET DECOMPOSITION SUBROUTINE                             **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  19 June 91

    VERSION: 1.0

    NAME:  decompose.c

    DESCRIPTION:  This subroutine is intended to be part of a Wavelet
    analyzing program called "wave". The algorithm used is discussed in
    the description of the main driver module called "main-wave.c.
    Data is passed by reference from the main driver module. The data is
    in ascii format arranged in a square matrix whose dimensions are a
    power of 2. This requirement has not only made programming more
    convenient but is required by the convolution routine from Numerical
    Recipes in C: The Art of Scientific Computing.

    FILES READ. NONE  (Passed by reference from the caller.)

    FILES WRITTEN:  Four coefficient files at each level of analysis.
                    The file names begin with the input image filename
                    and end with an extension of the form ".nXm" where
                    n is an integer that represents the level, X is one
    of the letters 'c' or 'd' to represent phi
    or psi coefficients respectively, and m is
```

```
    an integer 1, 2, or 3 that represents the
    orientation verticle, horizontal, or angular
    repsectively.
    HEADERS USED:   <stdio.h>, "jsmacros.h"
    CALLING PROGRAMS:   main-wave.c
    PROGRAMS CALLED: convolve.c, filters.c, nrutil.c
    AUTHOR: Steve Smiley and J. Stewart Laing
    HISTORY: Initial Version.
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include <stdio.h>
#include "jsmacros.h"

void        convolve();
void        filters();
float       *vector();
float       **matrix();
void        free_vector();
void        free_matrix();
int         **imatrix();

/************************/
/*  MAIN PROGRAM BODY  */
/************************/

void decompose(imagepointer, infilename)
   int_array *imagepointer;
   char      infilename[];
{

/************************/
/* declare variables    */
/************************/

int           i, j, k, maxlevel, wavelet_type;
float_vector  h_of_n,  h_of_nflipo, g_of_n, g_of_nflipo, phi, phiflipo;
float_vector  phiflipc, *phiflipcpointer = &phiflipc;
float_vector  *h_of_npointer = &h_of_n, *h_of_nflipopointer = &h_of_nflipo;
float_vector  *g_of_npointer = &g_of_n, *g_of_nflipopointer = &g_of_nflipo;
float_vector  *phipointer = &phi, *phiflipopointer = &phiflipo;
float_array   c_coef, d1_coef, d2_coef, d3_coef;
float_array   *c_coefpointer= &c_coef,*d1_coefpointer= &d1_coef;
float_array   *d2_coefpointer= &d2_coef,*d3_coefpointer= &d3_coef;
float_array   temp, *temppointer = &temp;
FILE          *outfile;
char          filename[64], wave_code[64];
int_array     newimage, *newimagepointer = &newimage;

/************************/
/* allocate memory      */
/************************/

temp.ROW = imagepointer->ROW;
temp.COL = imagepointer->COL;
temp.array = matrix(1, temp.ROW, 1, temp.COL);
loopij(temp.ROW,temp.COL) temp.array[i+1][j+1] = 0.0;
c_coef.ROW = imagepointer->ROW;
c_coef.COL = imagepointer->COL;
c_coef.array = matrix(1, c_coef.ROW, 1, c_coef.COL);
loopij(c_coef.ROW,c_coef.COL) c_coef.array[i+1][j+1] = 0.0;
d1_coef.ROW = imagepointer->ROW;
d1_coef.COL = imagepointer->COL;
```

```
d1_coef.array = matrix(1, d1_coef.ROW, 1, d1_coef.COL);
loopij(d1_coef.ROW,d1_coef.COL) d1_coef.array[i+1][j+1] = 0.0;
d2_coef.ROW = imagepointer->ROW;
d2_coef.COL = imagepointer->COL;
d2_coef.array = matrix(1, d2_coef.ROW, 1, d2_coef.COL);
loopij(d2_coef.ROW,d2_coef.COL) d2_coef.array[i+1][j+1] = 0.0;
d3_coef.ROW = imagepointer->ROW;
d3_coef.COL = imagepointer->COL;
d3_coef.array = matrix(1, d3_coef.ROW, 1, d3_coef.COL);
loopij(d3_coef.ROW,d3_coef.COL) d3_coef.array[i+1][j+1] = 0.0;
newimage.ROW = imagepointer->ROW;
newimage.COL = imagepointer->COL;
newimage.array = imatrix(1, newimage.ROW, 1, newimage.COL);
loopij(newimage.ROW,newimage.COL) newimage.array[i+1][j+1] = 0;

h_of_n.vector = vector(1,imagepointer->ROW*2);
loopi(imagepointer->ROW*2) h_of_n.vector[i+1] = 0.0;
g_of_n.vector = vector(1,imagepointer->ROW*2);
loopi(imagepointer->ROW*2) g_of_n.vector[i+1] = 0.0;
h_of_nflipo.vector = vector(1,imagepointer->ROW*2);
loopi(imagepointer->ROW*2) h_of_nflipo.vector[i+1] = 0.0;
g_of_nflipo.vector = vector(1,imagepointer->ROW*2);
loopi(imagepointer->ROW*2) g_of_nflipo.vector[i+1] = 0.0;
phi.vector = vector(1,2*imagepointer->ROW);
loopi(imagepointer->ROW*2) phi.vector[i+1] = 0.0;
phiflipo.vector = vector(1,2*imagepointer->ROW);
loopi(imagepointer->ROW*2) phiflipo.vector[i+1] = 0.0;
phiflipc.vector = vector(1,2*imagepointer->ROW);
loopi(imagepointer->ROW*2) phiflipc.vector[i+1] = 0.0;

/************************************/
/*  display menu                    */
/************************************/

printf("\n\n   DECOMPOSITION MENU\n\n");
printf("      1  =  Piece-wise Constant.(N/A)\n");
printf("      2  =  Piece-wise Linear.(N/A)\n");
printf("      3  =  Daubechies N=2.\n");
printf("      4  =  Daubechies N=3.\n");
printf("      5  =  Daubechies N=4.\n");
printf("      6  =  Daubechies N=5.\n");
printf("      7  =  Daubechies N=6.\n");
printf("      8  =  Daubechies N=7.\n");
printf("      9  =  Daubechies N=8.\n");
printf("      10  =  Daubechies N=9.\n");
printf("      11  =  Daubechies N=10.\n");
printf("      12  =  Splines.\n");
printf("      13  =  Morlet.(N/A)\n");
printf("\n  Enter an integer 1-13: ");

scanf("%d", &wavelet_type);

/* error handling for invalid input */

if (wavelet_type < 3 || wavelet_type > 13) {
   printf("\nYou have chosen an Invalid Wavelet type or");
   printf("\nthis type is not currently available.");
   } /* end if */
else {

   /*********************************************/
   /* Set wave_code for use in output filenames. */
   /*********************************************/

   if (wavelet_type == 3) sprintf(wave_code, "db2");
```

```
   if (wavelet_type == 4) sprintf(wave_code, "db3");
   if (wavelet_type == 5) sprintf(wave_code, "db4");
   if (wavelet_type == 6) sprintf(wave_code, "db5");
   if (wavelet_type == 7) sprintf(wave_code, "db6");
   if (wavelet_type == 8) sprintf(wave_code, "db7");
   if (wavelet_type == 9) sprintf(wave_code, "db8");
   if (wavelet_type == 10) sprintf(wave_code, "db9");
   if (wavelet_type == 11) sprintf(wave_code, "db0");
   if (wavelet_type == 12) sprintf(wave_code, "spl");

   /*****************************/
   /* Generate Phi and Filters */
   /*****************************/

   filters (wavelet_type,h_of_npointer,g_of_npointer,phipointer);
   flipo(phipointer, phiflipopointer);

   h_of_nflipopointer = h_of_npointer;
   g_of_nflipopointer = g_of_npointer;
loopij(imagepointer->ROW,imagepointer->COL)
   temppointer->array[i+1][j+1] = (float)imagepointer->array[i+1][j+1];

   /*****************************************************************/
   /* Call convolution routine and save the coefficient arrays for */
   /* each level of analysis.                                       */
   /*****************************************************************/

   maxlevel = LOG2(imagepointer->ROW); /* Calculate the highest level */
   k=1;
   loopk(maxlevel){
      if (temp.ROW >= h_of_n.length){ /* image has to be bigger than filter */
         printf("\nPerforming convolution with filters, level");
 printf("%d...", k+1);
 convolve(temppointer, h_of_nflipopointer, g_of_nflipopointer,
 c_coefpointer, d1_coefpointer,d2_coefpointer,d3_coefpointer);
 sprintf(filename, "%s.%d.c.%s", infilename, k+1, wave_code);
         CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
         loopij(c_coef.ROW,c_coef.COL)
            fprintf(outfile, "%f\n", c_coef.array[i+1][j+1]);
         CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile)
         sprintf(filename, "%s.%d.d1.%s", infilename, k+1,wave_code);
         CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
         loopij(d1_coef.ROW,d1_coef.COL)
            fprintf(outfile, "%f\n", d1_coef.array[i+1][j+1]);
         CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile)
         sprintf(filename, "%s.%d.d2.%s", infilename, k+1,wave_code);
         CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
         loopij(d2_coef.ROW,d2_coef.COL)
            fprintf(outfile, "%f\n", d2_coef.array[i+1][j+1]);
         CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile)
         sprintf(filename, "%s.%d.d3.%s", infilename, k+1,wave_code);
         CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
         loopij(d3_coef.ROW,d3_coef.COL)
            fprintf(outfile, "%f\n", d3_coef.array[i+1][j+1]);
         CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile)
         temp.ROW = c_coef.ROW;
         temp.COL = c_coef.COL;
         loopij(temp.ROW,temp.COL) temp.array[i+1][j+1]=c_coef.array[i+1][j+1];
      } /* end if */
   } /* end loop */
} /* end else */
```

```
/* free memory */
free_matrix(temp.array, 1, temp.ROW, 1, temp.COL);
free_matrix(c_coef.array, 1, c_coef.ROW, 1, temp.COL);
free_matrix(d1_coef.array, 1, d1_coef.ROW, 1, d1_coef.COL);
free_matrix(d2_coef.array, 1, d2_coef.ROW, 1, d2_coef.COL);
free_matrix(d3_coef.array, 1, d3_coef.ROW, 1, d3_coef.COL);
free_vector(h_of_n.vector,1,imagepointer->ROW*2);
free_vector(g_of_n.vector,1,imagepointer->ROW*2);
free_vector(phi.vector,1,imagepointer->ROW*2);
free_vector(phiflipo.vector,1,imagepointer->ROW*2);
free_vector(phiflipc.vector,1,imagepointer->ROW*2);
/* THE END */
}
```

B.2.4 Listing of RECONSTRUCT.C

```
/*****************************************************************************/
/*****************************************************************************/
/***            WAVELET RECONSTRUCTION SUBROUTINE                          **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  2 July 91
  VERSION: 2.0 (uses spconvlv)
  NAME:  reconstruct.c
  DESCRIPTION:  This subroutine is intended to be part of a Wavelet
  analyzing program called "wave2". The algorithm used is discussed in
  the description of the main driver module called "main-wave.c.
  It controls the portion of the program that reconstructs a previously
  decomposed image using Mallat's multiresolution algorithm referenced
  in the description of the calling program, "main-wave.c".
  FILES READ:      Four coefficient files at each level of analysis.
                   The file names begin with the input image filename
                   and end with an extension of the form ".nXm" where
                   n is an integer that represents the level, X is one of
                   the letters 'c' or 'd' to represent phi or psi coef-
                   ficients respectively, and m is an integer 1, 2, or 3
                   that represents the orientation verticle, horizontal,
                   or angular repsectively.
  FILES WRITTEN:  One file with the extension ".rec".
  HEADERS USED:   <stdio.h>, "jsmacros.h"
  CALLING PROGRAMS:   main-wave.c
  PROGRAMS CALLED:  filters.c, reconvolve.c, nrutil.c
  AUTHOR: Steve Smiley and J. Stewart Laing
  HISTORY: Initial Version.
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include <stdio.h>
#include "jsmacros.h"

void    filters();
void    reconvolve();
float   *vector();
float   **matrix();
void    free_vector();
void    free_matrix();
```

```
int     **imatrix();
void    free_imatrix();

/************************/
/*  MAIN PROGRAM BODY  */
/************************/

void reconstruct(imagepointer,infilename)
   int_array *imagepointer;
   char      infilename[];
{

/*************************/
/* declare variables     */
/*************************/

int           i, j, k, l, maxlevel, wavelet_type;
float_vector  h_of_n, h_of_nflipo, h_of_nflipc, g_of_n;
float_vector  g_of_nflipo, g_of_nflipc, phi, phiflipc;
float_vector  *h_of_npointer = &h_of_n, *g_of_npointer = &g_of_n;
float_vector  *h_of_nflipopointer = &h_of_nflipo;
float_vector  *g_of_nflipopointer = &g_of_nflipo;
float_vector  *h_of_nflipcpointer = &h_of_nflipc;
float_vector  *g_of_nflipcpointer = &g_of_nflipc;
float_vector  *phipointer = &phi, *phiflipcpointer = &phiflipc;
float_array   c_coef, d1_coef, d2_coef, d3_coef;
float_array   *c_coefpointer= &c_coef,*d1_coefpointer= &d1_coef;
float_array   *d2_coefpointer= &d2_coef,*d3_coefpointer= &d3_coef;
float_array   temp, *temppointer = &temp;
int_array     newimage, *newimagepointer = &newimage;
FILE          *outfile, *infile;
char          filename[64], wave_code[64];
float          holder[64];

/*************************/
/* allocate memory       */
/*************************/

temp.ROW = imagepointer->ROW;
temp.COL = imagepointer->COL;
temp.array = matrix(1, temp.ROW, 1, temp.COL);
loopij(temp.ROW,temp.COL) temp.array[i+1][j+1] = 0.0;
newimage.ROW = imagepointer->ROW;
newimage.COL = imagepointer->COL;
newimage.array = imatrix(1, newimage.ROW, 1, newimage.COL);
loopij(newimage.ROW,newimage.COL) newimage.array[i+1][j+1] = 0.0;
c_coef.ROW = imagepointer->ROW;
c_coef.COL = imagepointer->COL;
c_coef.array = matrix(1, c_coef.ROW, 1, c_coef.COL);
loopij(c_coef.ROW,c_coef.COL) c_coef.array[i+1][j+1] = 0.0;
d1_coef.ROW = imagepointer->ROW;
d1_coef.COL = imagepointer->COL;
d1_coef.array = matrix(1, d1_coef.ROW, 1, d1_coef.COL);
loopij(d1_coef.ROW,d1_coef.COL) d1_coef.array[i+1][j+1] = 0.0;
d2_coef.ROW = imagepointer->ROW;
d2_coef.COL = imagepointer->COL;
d2_coef.array = matrix(1, d2_coef.ROW, 1, d2_coef.COL);
loopij(d2_coef.ROW,d2_coef.COL) d2_coef.array[i+1][j+1] = 0.0;
d3_coef.ROW = imagepointer->ROW;
d3_coef.COL = imagepointer->COL;
d3_coef.array = matrix(1, d3_coef.ROW, 1, d3_coef.COL);
loopij(d3_coef.ROW,d3_coef.COL) d3_coef.array[i+1][j+1] = 0.0;

h_of_n.vector = vector(1,imagepointer->ROW*2);
loopi(imagepointer->ROW*2) h_of_n.vector[i+1] = 0.0;
g_of_n.vector = vector(1,imagepointer->ROW*2);
loopi(imagepointer->ROW*2) g_of_n.vector[i+1] = 0.0;
phi.vector = vector(1,2*imagepointer->ROW);
```

```
loopi(imagepointer->ROW*2) phi.vector[i+1] = 0.0;
phiflipc.vector = vector(1,2*imagepointer->ROW);
loopi(imagepointer->ROW*2) phiflipc.vector[i+1] = 0.0;
h_of_nflipo.vector = vector(1,imagepointer->ROW*2);
loopi(imagepointer->ROW*2) h_of_nflipo.vector[i+1] = 0.0;
g_of_nflipo.vector = vector(1,imagepointer->ROW*2);
loopi(imagepointer->ROW*2) g_of_nflipo.vector[i+1] = 0.0;
h_of_nflipc.vector = vector(1,imagepointer->ROW*2);
loopi(imagepointer->ROW*2) h_of_nflipc.vector[i+1] = 0.0;
g_of_nflipc.vector = vector(1,imagepointer->ROW*2);
loopi(imagepointer->ROW*2) g_of_nflipc.vector[i+1] = 0.0;
/************************************/
/*  display menu                    */
/************************************/
   printf("\n\n   RECONSTRUCTION MENU\n\n");
   printf("      1  =  Piece-wise Constant.(N/A)\n");
   printf("      2  =  Piece-wise Linear.(N/A)\n");
   printf("      3  =  Daubechies N=2.\n");
   printf("      4  =  Daubechies N=3.\n");
   printf("      5  =  Daubechies N=4.\n");
   printf("      6  =  Daubechies N=5.\n");
   printf("      7  =  Daubechies N=6.\n");
   printf("      8  =  Daubechies N=7.\n");
   printf("      9  =  Daubechies N=8.\n");
   printf("      10  =  Daubechies N=9.\n");
   printf("      11  =  Daubechies N=10.\n");
   printf("      12  =  Splines.\n");
   printf("      13  =  Morlet.(N/A)\n");
   printf("   Enter an integer (1-13):");
   scanf("%d", &wavelet_type);
   if(wavelet_type < 1 || wavelet_type > 13 )
      printf("\nYou have chosen an invalid wavelet or");
      printf("\nit is not currently available.");
      }
   else {
      /*********************************************/
      /* Set value of wave_code for input filename */
      /*********************************************/
      if (wavelet_type == 3) sprintf(wave_code, "db2");
      if (wavelet_type == 4) sprintf(wave_code, "db3");
      if (wavelet_type == 5) sprintf(wave_code, "db4");
      if (wavelet_type == 6) sprintf(wave_code, "db5");
      if (wavelet_type == 7) sprintf(wave_code, "db6");
      if (wavelet_type == 8) sprintf(wave_code, "db7");
      if (wavelet_type == 9) sprintf(wave_code, "db8");
      if (wavelet_type == 10) sprintf(wave_code, "db9");
      if (wavelet_type == 11) sprintf(wave_code, "db0");
      if (wavelet_type == 12) sprintf(wave_code, "spl");
      /*****************************/
      /* Generate Phi and Filters */
      /*****************************/
      filters(wavelet_type,h_of_npointer,g_of_npointer,phipointer);
/*****************************************/
/*          flip the filters             */
/*****************************************/

loop1j(h_of_npointer->length)
```

```
  holder[h_of_npointer->length +1 -j]= h_of_npointer->vector[j];
loop1j(h_of_npointer->length)
  h_of_npointer->vector[j] = holder[j];
loop1j(g_of_npointer->length)
  holder[g_of_npointer->length +1 -j]= g_of_npointer->vector[j];
loop1j(g_of_npointer->length)
  g_of_npointer->vector[j] = holder[j];

h_of_nflipcpointer= h_of_npointer;
g_of_nflipcpointer= g_of_npointer;


     /*****************************************************************/
     /* Call reconvolution routine to reconstruct from coursest phi  */
     /* coefficients and all of the psi coefficients.                 */
     /*****************************************************************/
     maxlevel = LOG2(imagepointer->ROW);/*Calculate the highest level*/

     temp.ROW = 1; temp.COL = 1;
   do {              /* make sure image is bigger than filter */
     temp.ROW *=2;
     temp.COL *=2;
     --maxlevel;
     } while (temp.ROW < h_of_n.length/2);
   c_coef.ROW = temp.ROW; c_coef.COL = temp.COL;
   d1_coef.ROW = temp.ROW; d1_coef.COL = temp.COL;
   d2_coef.ROW = temp.ROW; d2_coef.COL = temp.COL;
   d3_coef.ROW = temp.ROW; d3_coef.COL = temp.COL;
   l = 1;
   for(k=maxlevel;k>0;--k){
    /* for(k=maxlevel;k==maxlevel;--k){ */
         if(l == 1){
            sprintf(filename, "%s.%d.c.%s", infilename, k,wave_code);
            OPEN_FILE(infile, filename, "The Wavelet Analyzer")
            loopij(c_coef.ROW,c_coef.COL)
               fscanf(infile, "%f\n", &c_coef.array[i+1][j+1]);
    CLOSE_FILE(i, filename, "The Wavelet Analyzer", infile)
            l = 0;
            } /* end if */
         else {
            c_coef.ROW = temp.ROW;
            c_coef.COL = temp.COL;
            loopij(c_coef.ROW,c_coef.COL) c_coef.array[i+1][j+1] =
               temp.array[i+1][j+1];
    } /* end else */
         sprintf(filename, "%s.%d.d1.%s", infilename, k,wave_code);
         OPEN_FILE(infile, filename, "The Wavelet Analyzer")
         loopij(d1_coef.ROW,d1_coef.COL)
            fscanf(infile, "%f\n", &d1_coef.array[i+1][j+1]);
         CLOSE_FILE(i, filename, "The Wavelet Analyzer", infile)
         sprintf(filename, "%s.%d.d2.%s", infilename, k,wave_code);
         OPEN_FILE(infile, filename, "The Wavelet Analyzer")
         loopij(d2_coef.ROW,d2_coef.COL)
            fscanf(infile, "%f\n", &d2_coef.array[i+1][j+1]);
         CLOSE_FILE(i, filename, "The Wavelet Analyzer", infile)
         sprintf(filename, "%s.%d.d3.%s", infilename, k,wave_code);
         OPEN_FILE(infile, filename, "The Wavelet Analyzer")
         loopij(d3_coef.ROW,d3_coef.COL)
            fscanf(infile, "%f\n", &d3_coef.array[i+1][j+1]);
         CLOSE_FILE(i, filename, "The Wavelet Analyzer", infile)
```

```
        printf("\nPerforming reconvolution with filters, level %d...", k);
        reconvolve(temppointer, h_of_nflipcpointer, g_of_nflipcpointer,
    c_coefpointer, d1_coefpointer, d2_coefpointer,
    d3_coefpointer);
        if(wavelet_type == 12)
          loopij(temp.ROW,temp.COL) temp.array[i+1][j+1] *= 4;
        sprintf(filename, "%s.%d.c.%s.rec", infilename,k-1,wave_code);
        CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
        loopij(temp.ROW,temp.COL)
           fprintf(outfile, "%f\n", temp.array[i+1][j+1]);
        CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile)
        } /* end loop */
   } /* end else */
/* free memory */
free_matrix(temp.array, 1, temp.ROW, 1, temp.COL);
free_imatrix(newimage.array, 1, newimage.ROW, 1, newimage.COL);
free_matrix(c_coef.array, 1, c_coef.ROW, 1, c_coef.COL);
free_matrix(d1_coef.array, 1, d1_coef.ROW, 1, d1_coef.COL);
free_matrix(d2_coef.array, 1, d2_coef.ROW, 1, d2_coef.COL);
free_matrix(d3_coef.array, 1, d3_coef.ROW, 1, d3_coef.COL);
/* THE END */
}
```

B.2.5 Listing of FILTERS.C

```
/*****************************************************************************/
/*****************************************************************************/
/***                    WAVELET H&G FILTER SUBROUTINE                      **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  20 June 91
  VERSION: 2.0
  NAME:  filters.c
  DESCRIPTION:  This subroutine is intended to be part of a Wavelet
  analyzing program called "wave2". The algorithm used is discussed in
  the description of the main driver module called "main-wave.c.
  This routine provides the caller with the discrete points of a pair of
  response functions previously derived and hard coded corresponding to
  the type of wavelet desired. Also, the scaling function,
  phi(x) is provided for the purpose of generating the phi
  coefficients at level zero.
  FILES READ: NONE
  FILES WRITTEN:  (Passed by reference back to the caller.)
  HEADERS USED:   <stdio.h>, "jsmacros.h"
  CALLING PROGRAMS:   decompose.c, nrutil.c
  PROGRAMS CALLED: NONE
  AUTHOR: Steve Smiley and J. Stewart Laing
  HISTORY: Version 2  altered filters.c for spatial convolution from the
  Fourier convolution used in version 1.
*/
/*****************************************************************************/
/*****************************************************************************/
/*************************/
/*  DECLARATION SECTION  */
/*************************/
#include <stdio.h>
```

```
#include "jsmacros.h"
/************************/
/*  MAIN PROGRAM BODY  */
/************************/
void filters (wavelet_type,h_of_npointer,g_of_npointer,phipointer)
   int           wavelet_type;
   float_vector *h_of_npointer, *g_of_npointer, *phipointer;
{
/*****************************************************************************/
/* The response functions of the H and G filters are evaluated at the */
/*  negative of the argument. i.e. g(n)=g(-n) and h(n)=h(-n)          */
/*****************************************************************************/
if (wavelet_type == 1){
printf("\nThis selection not currently available.");
}
if (wavelet_type == 2){
printf("\nThis selection not currently available.");
}
if (wavelet_type == 3){
   h_of_npointer->vector[4] = .482962; /* h(0)*/
   h_of_npointer->vector[5] = .836516; /* h(1)*/
   h_of_npointer->vector[6] = .224143; /* h(2)*/
   h_of_npointer->vector[7] =  -.129409; /* h(3)*/
   h_of_npointer->vector[1] = 0.0;      /* h(-3)*/
   h_of_npointer->vector[2] = 0.0;     /* h(-2)*/
   h_of_npointer->vector[3] = 0.0;      /* h(-1)*/
   h_of_npointer->length = 7;
   g_of_npointer->vector[4] =   .836516; /* g(0)*/
   g_of_npointer->vector[5] = -.482962; /* g(1)*/
   g_of_npointer->vector[6] =  0.0; /* g(2)*/
   g_of_npointer->vector[7] =  0.0; /* g(3)*/
   g_of_npointer->vector[1] =  0.0; /* g(-3)*/
   g_of_npointer->vector[2] =  -.129409; /* g(-2)*/
   g_of_npointer->vector[3] =  -.224143; /* g(-1)*/
   g_of_npointer->length = 7;
   phipointer->vector[1] = 0.032348658; /* phi(0)*/
   phipointer->vector[2] = 1.302567547; /* phi(1)*/
   phipointer->vector[3] = -0.334912635; /* phi(2)*/
   phipointer->vector[4] = 0.0000000001; /* phi(3)*/
   phipointer->vector[5] = 0.0000000001; /* phi(-3)*/
   phipointer->vector[6] = 0.0000000001; /* phi(-2)*/
   phipointer->vector[7] = 0.0000000001; /* phi(-1)*/
   phipointer->length = 7;
}
if (wavelet_type == 4){
   h_of_npointer->vector[6] = 0.332670553; /* h(0)*/
   h_of_npointer->vector[7] = 0.806891509; /* h(1)*/
   h_of_npointer->vector[8] = 0.459877502; /* h(2)*/
   h_of_npointer->vector[9] = -0.135011020; /* h(3)*/
   h_of_npointer->vector[10] = -0.085441274; /* h(4)*/
   h_of_npointer->vector[11] =  0.035226292; /* h(5)*/
   h_of_npointer->vector[1] =  0.0; /* h(-5)*/
   h_of_npointer->vector[2] =  0.0; /* h(-4)*/
   h_of_npointer->vector[3] =  0.0; /* h(-3)*/
   h_of_npointer->vector[4] =  0.0; /* h(-2)*/
   h_of_npointer->vector[5] =  0.0; /* h(-1)*/
     h_of_npointer->length = 11;
   g_of_npointer->vector[5] = 0.806891509; /* g(0)*/
   g_of_npointer->vector[7] = 0.332670553; /* g(1)*/
```

```
    g_of_npointer->vector[8] = 0.0; /* g(2)*/
    g_of_npointer->vector[9] = 0.0; /* g(3)*/
    g_of_npointer->vector[10] = 0.0; /* g(4)*/
    g_of_npointer->vector[11] = 0.0; /* g(5)*/
    g_of_npointer->vector[1] = 0.0; /* g(-5)*/
    g_of_npointer->vector[2] = 0.459877502; /* g(-4)*/
    g_of_npointer->vector[3] = -0.135011020; /* g(-3)*/
    g_of_npointer->vector[4] = -0.085441274; /* g(-2)*/
    g_of_npointer->vector[5] = 0.035226292; /* g(-1)*/
      g_of_npointer->length = 11;

    phipointer->vector[1] = 0.001129175; /* phi(0)*/
    phipointer->vector[2] = 1.285632059; /* phi(1)*/
    phipointer->vector[3] = -0.386241412; /* phi(2)*/
    phipointer->vector[4] = 0.095244687; /* phi(3)*/
    phipointer->vector[5] = 0.004229018; /* phi(4)*/
    phipointer->vector[6] = 0.000000001; /* phi(5)*/
    phipointer->vector[7] = 0.0000000001; /* phi(-5)*/
    phipointer->vector[8] = 0.0000000001; /* phi(-4)*/
    phipointer->vector[9] = 0.0000000001; /* phi(-3)*/
    phipointer->vector[10] = 0.0000000001; /* phi(-2)*/
    phipointer->vector[11] = 0.0000000001; /* phi(-1)*/
    phipointer->length = 11;
}

if (wavelet_type == 5){
    h_of_npointer->vector[8] = 0.230377813; /* h(0)*/
    h_of_npointer->vector[9] = 0.714846571; /* h(1)*/
    h_of_npointer->vector[10] = 0.630880768; /* h(2)*/
    h_of_npointer->vector[11] = -0.027983769; /* h(3)*/
    h_of_npointer->vector[12] = -0.187034812; /* h(4)*/
    h_of_npointer->vector[13] = 0.030841382; /* h(5)*/
    h_of_npointer->vector[14] = 0.032883012; /* h(6)*/
    h_of_npointer->vector[15] = -0.010597402; /* h(7)*/
    h_of_npointer->vector[1] =  0.0; /* h(-7)*/
    h_of_npointer->vector[2] =  0.0; /* h(-6)*/
    h_of_npointer->vector[3] =  0.0; /* h(-5)*/
    h_of_npointer->vector[4] =  0.0; /* h(-4)*/
    h_of_npointer->vector[5] =  0.0; /* h(-3)*/
    h_of_npointer->vector[6] =  0.0; /* h(-2)*/
    h_of_npointer->vector[7] =  0.0; /* h(-1)*/
      h_of_npointer->length = 15;

    g_of_npointer->vector[8] = 0.714846571; /* g(0)*/
    g_of_npointer->vector[9] = 0.230377813; /* g(1)*/
    g_of_npointer->vector[10] = 0.0; /* g(2)*/
    g_of_npointer->vector[11] = 0.0; /* g(3)*/
    g_of_npointer->vector[12] = 0.0; /* g(4)*/
    g_of_npointer->vector[13] = 0.0; /* g(5)*/
    g_of_npointer->vector[14] = 0.0; /* g(6)*/
    g_of_npointer->vector[15] = 0.0; /* g(7)*/
    g_of_npointer->vector[1] = 0.0; /* g(-7)*/
    g_of_npointer->vector[2] = -0.010597402; /* g(-6)*/
    g_of_npointer->vector[3] = 0.032883012; /* g(-5)*/
    g_of_npointer->vector[4] = 0.030841382; /* g(-4)*/
    g_of_npointer->vector[5] = -0.187034812; /* g(-3)*/
    g_of_npointer->vector[6] = -0.027983769; /* g(-2)*/
    g_of_npointer->vector[7] = 0.630880768; /* g(-1)*/
      g_of_npointer->length = 15;

    phipointer->vector[1] = 0.000041362; /* phi(0)*/
    phipointer->vector[2] = 1.010495941; /* phi(1)*/
    phipointer->vector[3] = -0.039093761; /* phi(2)*/
```

```
    phipointer->vector[4] = 0.041834300; /* phi(3)*/
    phipointer->vector[5] = -0.012011135; /* phi(4)*/
    phipointer->vector[6] = -0.001294973; /* phi(5)*/
    phipointer->vector[7] = 0.000021869; /* phi(6)*/
    phipointer->vector[8] = 0.000000001; /* phi(7)*/
    phipointer->vector[9] = 0.0000000001; /* phi(-7)*/
    phipointer->vector[10] = 0.0000000001; /* phi(-6)*/
    phipointer->vector[11] = 0.0000000001; /* phi(-5)*/
    phipointer->vector[12] = 0.0000000001; /* phi(-4)*/
    phipointer->vector[13] = 0.0000000001; /* phi(-3)*/
    phipointer->vector[14] = 0.0000000001; /* phi(-2)*/
    phipointer->vector[15] = 0.0000000001; /* phi(-1)*/
    phipointer->length = 15;
}
if (wavelet_type == 6){
printf("\nThis selection not currently available.");
}
if (wavelet_type == 7){
    h_of_npointer->vector[12] = 0.111540743; /* h(0)*/
    h_of_npointer->vector[13] = 0.494623890; /* h(1)*/
    h_of_npointer->vector[14] = 0.751133908; /* h(2)*/
    h_of_npointer->vector[15] = 0.315250352; /* h(3)*/
    h_of_npointer->vector[16] = -0.226264694; /* h(4)*/
    h_of_npointer->vector[17] = -0.129766868; /* h(5)*/
    h_of_npointer->vector[18] =  0.097501606; /* h(6)*/
    h_of_npointer->vector[19] = 0.027522866; /* h(7)*/
    h_of_npointer->vector[20] = -0.031582039; /* h(8)*/
    h_of_npointer->vector[21] = 0.000553842; /* h(9)*/
    h_of_npointer->vector[22] = 0.004777257; /* h(10)*/
    h_of_npointer->vector[23] = -0.001077301; /* h(11)*/
    h_of_npointer->vector[1] =  0.0; /* h(-11)*/
    h_of_npointer->vector[2] =  0.0; /* h(-10)*/
    h_of_npointer->vector[3] =  0.0; /* h(-9)*/
    h_of_npointer->vector[4] =  0.0; /* h(-8)*/
    h_of_npointer->vector[5] =  0.0; /* h(-7)*/
    h_of_npointer->vector[6] =  0.0; /* h(-6)*/
    h_of_npointer->vector[7] =  0.0; /* h(-5)*/
    h_of_npointer->vector[8] =  0.0; /* h(-4)*/
    h_of_npointer->vector[9] =  0.0; /* h(-3)*/
    h_of_npointer->vector[10] =  0.0; /* h(-2)*/
    h_of_npointer->vector[11] =  0.0; /* h(-1)*/
    h_of_npointer->length = 23;

    g_of_npointer->vector[12] = -0.494623890; /* g(0)*/
    g_of_npointer->vector[13] = 0.115407434; /* g(1)*/
    g_of_npointer->vector[14] = 0.0; /* g(2)*/
    g_of_npointer->vector[15] = 0.0; /* g(3)*/
    g_of_npointer->vector[16] = 0.0; /* g(4)*/
    g_of_npointer->vector[17] = 0.0; /* g(5)*/
    g_of_npointer->vector[18] = 0.0; /* g(6)*/
    g_of_npointer->vector[19] = 0.0; /* g(7)*/
    g_of_npointer->vector[20] = 0.0; /* g(8)*/
    g_of_npointer->vector[21] = 0.0; /* g(9)*/
    g_of_npointer->vector[22] = 0.0; /* g(10)*/
    g_of_npointer->vector[23] = 0.0; /* g(11)*/
    g_of_npointer->vector[1] = 0.0; /* g(-11)*/
    g_of_npointer->vector[2] = 0.001077301; /* g(-10)*/
    g_of_npointer->vector[3] = 0.004777257; /* g(-9)*/
    g_of_npointer->vector[4] = -0.000553842; /* g(-8)*/
    g_of_npointer->vector[5] = -0.031582039; /* g(-7)*/
    g_of_npointer->vector[6] = -0.027522866; /* g(-6)*/
```

```
   g_of_npointer->vector[7] = 0.097501606; /* g(-5)*/
   g_of_npointer->vector[8] = 0.129766868; /* g(-4)*/
   g_of_npointer->vector[9] = -0.226264694; /* g(-3)*/
   g_of_npointer->vector[10] = -0.315250352; /* g(-2)*/
   g_of_npointer->vector[11] = 0.751133908; /* g(-1)*/
     g_of_npointer->length = 23;
   phipointer->vector[1] = 0.000018901; /* phi(0)*/
   phipointer->vector[2] = 0.474401220; /* phi(1)*/
   phipointer->vector[3] = 0.807783651; /* phi(2)*/
   phipointer->vector[4] = -0.376153951; /* phi(3)*/
   phipointer->vector[5] =  0.137747794; /* phi(4)*/
   phipointer->vector[6] = -0.024343102; /* phi(5)*/
   phipointer->vector[7] = -0.003162779; /* phi(6)*/
   phipointer->vector[8] = 0.001579497; /* phi(7)*/
   phipointer->vector[9] = 0.000017680; /* phi(8)*/
   phipointer->vector[10] = -0.000001908; /* phi(9)*/
   phipointer->vector[11] = 0.000000002; /* phi(10)*/
   phipointer->vector[12] = 0.000000001; /* phi(11)*/
   phipointer->vector[13] = 0.0000000001; /* phi(-11)*/
   phipointer->vector[14] = 0.0000000001; /* phi(-10)*/
   phipointer->vector[15] = 0.0000000001; /* phi(-9)*/
   phipointer->vector[16] = 0.0000000001; /* phi(-8)*/
   phipointer->vector[17] = 0.0000000001; /* phi(-7)*/
   phipointer->vector[18] = 0.0000000001; /* phi(-6)*/
   phipointer->vector[19] = 0.0000000001; /* phi(-5)*/
   phipointer->vector[20] = 0.0000000001; /* phi(-4)*/
   phipointer->vector[21] = 0.0000000001; /* phi(-3)*/
   phipointer->vector[22] = 0.0000000001; /* phi(-2)*/
   phipointer->vector[23] = 0.0000000001; /* phi(-1)*/
     phipointer->length = 23;
}
if (wavelet_type == 8){
printf("\nThis selection not currently available.");
}
if (wavelet_type == 9){
printf("\nThis selection not currently available.");
}
if (wavelet_type == 10){
printf("\nThis selection not currently available.");
}
if (wavelet_type == 11){
printf("\nThis selection not currently available.");
}
if (wavelet_type == 12){
   h_of_npointer->vector[13] = 0.542; /* h(0)*/
   h_of_npointer->vector[14] = 0.307; /* h(1)*/
   h_of_npointer->vector[15] = -0.035; /* h(2)*/
   h_of_npointer->vector[16] = -0.078; /* h(3)*/
   h_of_npointer->vector[17] =  0.023; /* h(4)*/
   h_of_npointer->vector[18] =  0.030; /* h(5)*/
   h_of_npointer->vector[19] =  -0.012; /* h(6)*/
   h_of_npointer->vector[20] =  -0.013; /* h(7)*/
   h_of_npointer->vector[21] =  0.006; /* h(8)*/
   h_of_npointer->vector[22] = 0.006; /* h(9)*/
   h_of_npointer->vector[23] = -0.003; /* h(10)*/
   h_of_npointer->vector[24] = -0.002; /* h(11)*/
   h_of_npointer->vector[25] = 0.0; /* h(12)*/
   h_of_npointer->vector[1] = 0.0; /* h(-12)*/
   h_of_npointer->vector[2] = -0.002; /* h(-11)*/
   h_of_npointer->vector[3] = -0.003; /* h(-10)*/
```

```
h_of_npointer->vector[4] =  0.006; /* h(-9)*/
h_of_npointer->vector[5] =  0.006; /* h(-8)*/
h_of_npointer->vector[6] = -0.013; /* h(-7)*/
h_of_npointer->vector[7] = -0.012; /* h(-6)*/
h_of_npointer->vector[8] =  0.030; /* h(-5)*/
h_of_npointer->vector[9] =  0.023; /* h(-4)*/
h_of_npointer->vector[10] = -0.078; /* h(-3)*/
h_of_npointer->vector[11] = -0.035; /* h(-2)*/
h_of_npointer->vector[12] =  0.307; /* h(-1)*/
h_of_npointer->length = 25;

g_of_npointer->vector[13] = -0.307; /* g(0)*/
g_of_npointer->vector[14] = 0.542; /* g(1)*/
g_of_npointer->vector[15] = -0.307; /* g(2)*/
g_of_npointer->vector[16] = -0.035; /* g(3)*/
g_of_npointer->vector[17] = 0.078; /* g(4)*/
g_of_npointer->vector[18] = 0.023; /* g(5)*/
g_of_npointer->vector[19] = -0.030; /* g(6)*/
g_of_npointer->vector[20] = -0.012; /* g(7)*/
g_of_npointer->vector[21] = 0.013; /* g(8)*/
g_of_npointer->vector[22] = 0.006; /* g(9)*/
g_of_npointer->vector[23] = -0.006; /* g(10)*/
g_of_npointer->vector[24] = -0.003; /* g(11)*/
g_of_npointer->vector[25] = 0.002; /* g(12)*/
g_of_npointer->vector[1] = 0.0; /* g(-12)*/
g_of_npointer->vector[2] = 0.0; /* g(-11)*/
g_of_npointer->vector[3] = 0.002; /* g(-10)*/
g_of_npointer->vector[4] = -0.003; /* g(-9)*/
g_of_npointer->vector[5] = -0.006; /* g(-8)*/
g_of_npointer->vector[6] = 0.006; /* g(-7)*/
g_of_npointer->vector[7] = 0.013; /* g(-6)*/
g_of_npointer->vector[8] = -0.012; /* g(-5)*/
g_of_npointer->vector[9] = -0.030; /* g(-4)*/
g_of_npointer->vector[10] = 0.023; /* g(-3)*/
g_of_npointer->vector[11] = 0.078; /* g(-2)*/
g_of_npointer->vector[12] = -0.035; /* g(-1)*/
g_of_npointer->length = 25;

phipointer->vector[1] = 0.5385; /* phi(0)*/
phipointer->vector[2] = -0.2106; /* phi(1)*/
phipointer->vector[3] =  0.04319; /* phi(2)*/
phipointer->vector[4] =  0.01334; /* phi(3)*/
phipointer->vector[5] =  0.00738; /* phi(4)*/
phipointer->vector[6] =  -0.00324; /* phi(5)*/
phipointer->vector[7] =  0.00030; /* phi(6)*/
phipointer->vector[8] =  -0.00012; /* phi(7)*/
phipointer->vector[9] = 0.00001; /* phi(8)*/
phipointer->vector[10] =  0.0000000001; /* phi(9)*/
phipointer->vector[11] = 0.000000001; /* phi(10)*/
phipointer->vector[12] = 0.000000001; /* phi(11)*/
phipointer->vector[13] = 0.0000000001; /* phi(-11)*/
phipointer->vector[14] = 0.0000000001; /* phi(-10)*/
phipointer->vector[15] = 0.0000000001; /* phi(-9)*/
phipointer->vector[16] = 0.00001; /* phi(-8)*/
phipointer->vector[17] = -0.00012; /* phi(-7)*/
phipointer->vector[18] = 0.00030; /* phi(-6)*/
phipointer->vector[19] = -0.00324; /* phi(-5)*/
phipointer->vector[20] = 0.00738; /* phi(-4)*/
phipointer->vector[21] = 0.01334; /* phi(-3)*/
phipointer->vector[22] = 0.04319; /* phi(-2)*/
phipointer->vector[23] = -0.02106; /* phi(-1)*/
```

```
    phipointer->length = 23;
}
if (wavelet_type == 13){
printf("\nThis selection not currently available.");
}
if (wavelet_type > 13 || wavelet_type < 1)
   printf("\nYou have chosen an invalid selection.");
/* THE END */
}
```

B.2.6 Listing of CONVOLVE.C

```
/*****************************************************************************/
/*****************************************************************************/
/***              WAVELET CONVOLUTION SUBROUTINE                           **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  19 June 91
  VERSION: 1.0
  NAME:  convolve.c
  DESCRIPTION:  This subroutine is intended to be part of a Wavelet
  analyzing program called "wave". The algorithm used is discussed in
  the description of the main driver module called "main-wave.c.
  Data is passed by reference from the decomposition subroutine. Data is
  in ASCII format arranged in a square matrix whose dimensions are a
  power of 2. This requirement has not only made programming more
  convenient but is required by the convolution routine from Numerical
  Recipes in C: The Art of Scientific Computing.
  FILES READ: NONE  (Passed by reference from the caller.)
  FILES WRITTEN:  (Passed by reference back to the caller.)
  HEADERS USED:   <stdio.h>, "jsmacros.h"
  CALLING PROGRAMS:   decompose.c, nrutil.c
  PROGRAMS CALLED: needs nr library, libnr.a
  AUTHOR: Steve Smiley and J. Stewart Laing
  HISTORY: Initial Version.
*/
/*****************************************************************************/
/*****************************************************************************/

/************************/
/*  DECLARATION SECTION  */
/************************/

#include <stdio.h>
#include "jsmacros.h"

float   *vector();
float   **matrix();
void    free_vector();
void    free_vector();
void    spconvlv();
/**********************/
/*  MAIN PROGRAM BODY  */
/**********************/

void convolve (datainpointer,  h_of_npointer ,g_of_npointer, c_coefpointer,
   d1_coefpointer, d2_coefpointer, d3_coefpointer)
   float_array *datainpointer;
   float_vector *h_of_npointer, *g_of_npointer;
   float_array  *c_coefpointer,*d1_coefpointer,*d2_coefpointer,*d3_coefpointer;
{
```

```
/************************/
/* declare variables    */
/************************/

int           i, j;
float_vector  rowin,rowout,colin,colout,response;
float_array   temp;
FILE *outfile;
char filename[64];

/********************/
/* allocate memory  */
/********************/

temp.array = matrix(1, datainpointer->ROW, 1, datainpointer->COL);
loopij(datainpointer->ROW,datainpointer->COL) temp.array[i+1][j+1] = 0.0;
rowin.vector = vector(1,2*datainpointer->COL);
loopi(2*datainpointer->COL) rowin.vector[i+1] = 0.0;
rowout.vector = vector(1,4*datainpointer->COL);
loopi(datainpointer->COL*4) rowout.vector[i+1] = 0.0;
colin.vector = vector(1,2*datainpointer->ROW);
loopi(2*datainpointer->ROW) colin.vector[i+1] = 0.0;
colout.vector = vector(1,4*datainpointer->ROW);
loopi(datainpointer->ROW*4) colout.vector[i+1] = 0.0;
response.vector = vector(1,2*datainpointer->ROW);
loopi(datainpointer->ROW*2) response.vector[i+1] = 0.0;

rowin.length = 2*datainpointer->COL;
colin.length = 2*datainpointer->ROW;

/************************/
/* perform convolution */
/************************/

printf("\nConvovling rows with h(-n)...");
loopi(datainpointer->ROW){ /* convolve rows with h(-n) */
   loopj(datainpointer->ROW*2){
      response.vector[j+1] = h_of_npointer->vector[j+1];
                                    }
   loopj(datainpointer->COL) rowin.vector[j+1] = datainpointer->array[i+1][j+1];
   spconvlv(rowin.vector,rowin.length,response.vector,h_of_npointer->length,1,
      rowout.vector);
   loopj(datainpointer->COL/2) temp.array[i+1][j+1] = rowout.vector[2*(j+1)];
   } /* downsample by selectiny even cols */


printf("\nConvovling cols with h(-n)...");
loopi(datainpointer->COL/2){ /* convolve cols with h(-n) */
   loopj(datainpointer->ROW*2)
      response.vector[j+1] = h_of_npointer->vector[j+1];
   loopj(datainpointer->ROW) colin.vector[j+1] = temp.array[j+1][i+1];
   spconvlv(colin.vector,colin.length,response.vector,h_of_npointer->length,1,
      colout.vector);
   loopj(datainpointer->ROW/2) c_coefpointer->array[j+1][i+1] = colout.vector[2*(j+1)];
   } /* downsample by selecting even rows */


printf("\nConvovling cols with g(-n)...");
loopi(datainpointer->COL/2){ /* convolve cols with g(-n) */
   loopj(datainpointer->ROW*2)
      response.vector[j+1] = g_of_npointer->vector[j+1];
   loopj(datainpointer->ROW) colin.vector[j+1] = temp.array[j+1][i+1];
   spconvlv(colin.vector,colin.length,response.vector,g_of_npointer->length,1,
      colout.vector);
```

```
   loopj(datainpointer->ROW/2) d1_coefpointer->array[j+1][i+1] = colout.vector[2*(j+1)];
   }


printf("\nConvovling rows with g(-n)...");
loopi(datainpointer->ROW){ /* convolve rows with g(-n) */
   loopj(datainpointer->ROW*2)
      response.vector[j+1] = g_of_npointer->vector[j+1];
   loopj(datainpointer->COL) rowin.vector[j+1] = datainpointer->array[i+1][j+1];
   spconvlv(rowin.vector,rowin.length,response.vector,g_of_npointer->length,1,
      rowout.vector);
   loopj(datainpointer->COL/2) temp.array[i+1][j+1] = rowout.vector[2*(j+1)];
   }


printf("\nConvovling cols with h(-n)...");
loopi(datainpointer->COL/2){ /* convolve cols with h(-n) */
   loopj(datainpointer->ROW*2)
      response.vector[j+1] = h_of_npointer->vector[j+1];
   loopj(datainpointer->ROW) colin.vector[j+1] = temp.array[j+1][i+1];
   spconvlv(colin.vector,colin.length,response.vector,h_of_npointer->length,1,
      colout.vector);
   loopj(datainpointer->ROW/2) d2_coefpointer->array[j+1][i+1] = colout.vector[2*(j+1)];
   }


printf("\nConvovling cols with g(-n)...");
loopi(datainpointer->COL/2){ /* convolve cols with g(-n) */
   loopj(datainpointer->ROW*2)
      response.vector[j+1] = g_of_npointer->vector[j+1];
   loopj(datainpointer->ROW) colin.vector[j+1] = temp.array[j+1][i+1];
   spconvlv(colin.vector,colin.length,response.vector,g_of_npointer->length,1,
      colout.vector);
   loopj(datainpointer->ROW/2) d3_coefpointer->array[j+1][i+1] = colout.vector[2*(j+1)];
   }


   /* reset row and col indeces. */
c_coefpointer->ROW = datainpointer->ROW/2;
c_coefpointer->COL = datainpointer->COL/2;
d1_coefpointer->ROW = datainpointer->ROW/2;
d1_coefpointer->COL = datainpointer->COL/2;
d2_coefpointer->ROW = datainpointer->ROW/2;
d2_coefpointer->COL = datainpointer->COL/2;
d3_coefpointer->ROW = datainpointer->ROW/2;
d3_coefpointer->COL = datainpointer->COL/2;
/* free memory */
free_matrix(temp.array, 1, datainpointer->ROW, 1,
datainpointer->COL);
free_vector (rowin.vector,1,2*datainpointer->ROW);
free_vector (rowout.vector,1,4*datainpointer->ROW);
free_vector (colin.vector,1,2*datainpointer->ROW);
free_vector (colout.vector,1,4*datainpointer->ROW);
free_vector (response.vector,1,2*datainpointer->ROW);
/* THE END */
}
```

B.2.7 Listing of RECONVOLVE.C

```
/*************************************** ***********************************/
```

```
/*****************************************************************************/
/***              WAVELET RECONVOLUTION SUBROUTINE                         **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  2 July 91
  VERSION: 1.0
  NAME:  reconvolve.c
  DESCRIPTION:  This subroutine is intended to be part of a Wavelet
  analyzing program called "wave2". The algorithm used is referenced in
  the description of the main driver module called "main-wave.c.
  Data is passed by reference from the reconstruction subroutine. Data is
  in ascii format arranged in a square matrix whose dimensions are a
  power of 2. This requirement has not only made programming more
  convenient but is required by the convolution routine from Numeric
  Recipes in C: The Art of Scientific Computing.
  FILES READ: NONE  (Passed by reference from the caller.)
  FILES WRITTEN:  NONE (Passed by reference back to the caller.
  HEADERS USED:   <stdio.h>, "jsmacros.h"
  CALLING PROGRAMS:   reconstruct.c
  PROGRAMS CALLED: NONE
  AUTHOR: Steve Smiley and J. Stewart Laing
  HISTORY: Initial Version.
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include <stdio.h>
#include "jsmacros.h"

float   *vector();
float   **matrix();
void    free_vector();
void    free_matrix();

/************************/
/*  MAIN PROGRAM BODY  */
/************************/

void reconvolve(dataoutpointer,h_of_npointer,g_of_npointer,c_coefpointer,
   d1_coefpointer, d2_coefpointer, d3_coefpointer)
   float_array *dataoutpointer;
   float_vector *h_of_npointer, *g_of_npointer;
   float_array  *c_coefpointer,*d1_coefpointer,*d2_coefpointer,*d3_coefpointer;
{

/************************/
/* declare variables    */
/************************/

int           i, j;
float_vector  rowin,rowout,colin,colout, response;
float_array   temp,temp1,temp2,temp3,temp4;
char          filename[64];
FILE          *outfile;

/*********************/
/* allocate memory  */
/*********************/

temp.ROW = c_coefpointer->ROW*2;
temp.COL = c_coefpointer->COL*2;
temp.array = matrix(1, temp.ROW, 1, temp.COL);
loopij(temp.ROW,temp.COL) temp.array[i+1][j+1] = 0.0;
```

```
temp1.ROW = c_coefpointer->ROW*2;
temp1.COL = c_coefpointer->COL*2;
temp1.array = matrix(1, temp1.ROW, 1, temp1.COL);
loopij(temp1.ROW,temp1.COL) temp1.array[i+1][j+1] = 0.0;
temp2.ROW = c_coefpointer->ROW*2;
temp2.COL = c_coefpointer->COL*2;
temp2.array = matrix(1, temp2.ROW, 1, temp2.COL);
loopij(temp2.ROW,temp2.COL) temp2.array[i+1][j+1] = 0.0;
temp3.ROW = c_coefpointer->ROW*2;
temp3.COL = c_coefpointer->COL*2;
temp3.array = matrix(1, temp3.ROW, 1, temp3.COL);
loopij(temp3.ROW,temp3.COL) temp3.array[i+1][j+1] = 0.0;
temp4.ROW = c_coefpointer->ROW*2;
temp4.COL = c_coefpointer->COL*2;
temp4.array = matrix(1, temp4.ROW, 1, temp4.COL);
loopij(temp4.ROW,temp4.COL) temp4.array[i+1][j+1] = 0.0;
rowin.vector = vector(1,temp.ROW*2);
loopi(temp.ROW*2) rowin.vector[i+1] = 0.0;
rowout vector = vector(1,temp.ROW*4);
loopi(temp.ROW*4) rowout.vector[i+1] = 0.0;
colin.vector = vector(1,temp.COL*2);
loopi(temp.COL*2) colin.vector[i+1] = 0.0;
colout.vector = vector(1,4*temp.COL);
loopi(temp.COL*4) colout.vector[i+1] = 0.0;
response.vector = vector(1,temp.COL*2);
loopi(temp.COL*2) response.vector[i+1] = 0.0;

rowin.length = 4*c_coefpointer->COL;
colin.length = 4*c_coefpointer->ROW;
dataoutpointer->ROW = c_coefpointer->ROW*2;
dataoutpointer->COL = c_coefpointer->COL*2;

/********************* **/
/* perform convolution */
/***********************/

printf("\nConvovling cols of c_coef with h(n)...");
loopi(c_coefpointer->COL){
   loopj(c_coefpointer->ROW)
      colin.vector[2*(j+1)] = c_coefpointer->array[j+1][i+1];
   loopj(colin.length)
      response.vector[j+1]=h_of_npointer->vector[j+1];
   spconvlv(colin.vector,colin.length,response.vector,
      h_of_npointer->length,1,colout.vector);
   loopj(c_coefpointer->ROW*2)
      temp1.array[j+1][i+1] = colout.vector[j+1];
   } /* zeros are added between each row before convolution */

printf("\nConvovling cols of d1_coef with g(n)...");
loopi(d1_coefpointer->COL){
   loopj(d1_coefpointer->ROW) colin.vector[2*(j+1)] =
      d1_coefpointer->array[j+1][i+1];
   loopj(colin.length)
      response.vector[j+1]=g_of_npointer->vector[j+1];
   spconvlv(colin.vector,colin.length,response.vector,
      g_of_npointer->length,1,colout.vector);
   loopj(d1_coefpointer->ROW*2) temp2.array[j+1][i+1] = colout.vector[j+1];
     } /* zeros are added between each row before convolution */

printf("\nConvovling cols of d2_coef with h(n)...");
loopi(d2_coefpointer->COL){
   loopj(d2_coefpointer->ROW) colin.vector[2*(j+1)] =
      d2_coefpointer->array[j+1][i+1];
   loopj(colin.length)
```

```
      response.vector[j+1]=h_of_npointer->vector[j+1];
   spconvlv(colin.vector,colin.length,response.vector,
      h_of_npointer->length,1,colout.vector),
   loopj(d2_coefpointer->ROW*2)
       temp3.array[j+1][i+1] = colout.vector[j+1];
   } /* zeros are added between each row before convolution */
printf("\nConvovling cols of d3_coef with g(n)...");
loopi(d3_coefpointer->COL){
   loopj(d3_coefpointer->ROW) colin.vector[2*(j+1)] =
      d3_coefpointer->array[j+1][i+1];
   loopj(colin.length)
      response.vector[j+1]=g_of_npointer->vector[j+1];
   spconvlv(colin.vector,colin.length,response.vector,
      g_of_npointer->length,1,colout.vector);
     loopj(d3_coefpointer->ROW*2)
     temp4.array[j+1][i+1] = colout.vector[j+1];
   } /* zeros are added between each row before convolution */
/* Add temp arrays for col convolutions */
loopij(temp.ROW, temp.COL)
   temp.array[i+1][j+1] = temp1.array[i+1][j+1] + temp2.array[i+1][j+1];
loopij(temp1.ROW, temp1.COL)
   temp1.array[i+1][j+1] = temp3.array[i+1][j+1] +
                                                  temp4.array[i+1][j+1];
   /* sprintf(filename, "temp");
   CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
   loopi(dataoutpointer->ROW/2)
      fprintf(outfile, "%f\n", temp.array[i+1][128]);
   CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile)
   sprintf(filename, "temp1");
   CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
   loopi(dataoutpointer->ROW/2)
      fprintf(outfile, "%f\n", temp1.array[i+1][128]);
   CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile) */
printf("\nConvovling rows with h(n)...");
loopi(dataoutpointer->ROW){
   loopj(dataoutpointer->COL/2) rowin.vector[2*(j+1)] = temp.array[i+1][j+1];
   loopj(rowin.length) response.vector[j+1]=h_of_npointer->vector[j+1];
   spconvlv(rowin.vector,rowin.length,response.vector,
                                    h_of_npointer->length,1,rowout.vector);
   loopj(dataoutpointer->ROW) temp2.array[i+1][j+1] = rowout.vector[j+1];
   } /* zeros are added between each col before convolution */
printf("\nConvovling rows with g(n)...");
loopi(dataoutpointer->ROW){
   loopj(dataoutpointer->COL/2) rowin.vector[2*(j+1)] = temp1.array[i+1][j+1];
   loopj(colin.length) response.vector[j+1]=g_of_npointer->vector[j+1];
   spconvlv(rowin.vector,rowin.length,response.vector,
                                    g_of_npointer->length,1,rowout.vector);
   loopj(dataoutpointer->ROW) temp3.array[i+1][j+1] = rowout.vector[j+1];
   } /* zeros are added between each row before convolution */
   /* sprintf(filename, "temp2");
   CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
   loopi(dataoutpointer->ROW)
      fprintf(outfile, "%f\n", temp2.array[i+1][128]);
   CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile)
   sprintf(filename, "temp3");
   CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
   loopi(dataoutpointer->ROW)
```

```
      fprintf(outfile, "%f\n", temp3.array[i+1][128]);
   CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile) */
/* Add temp arrays to get resulting dataout */
loopij(dataoutpointer->ROW,dataoutpointer->COL)
   dataoutpointer->array[i+1][j+1] = temp2.array[i+1][j+1] +
                                                  temp3.array[i+1][j+1];
   /* sprintf(filename, "dataout");
   CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
   loopi(dataoutpointer->ROW)
      fprintf(outfile, "%f\n", dataoutpointer->array[i+1][128]);
   CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile) */

/*loopij(dataoutpointer->ROW,dataoutpointer->COL)
   printf("dataoutpointer->array[%d][%d]=%f\n",i+1,j+1,
      dataoutpointer->array[i+1][j+1]);*/
   /* reset row and col indeces. */
d1_coefpointer->ROW = dataoutpointer->ROW;
d1_coefpointer->COL = dataoutpointer->COL;
d2_coefpointer->ROW = dataoutpointer->ROW;
d2_coefpointer->COL = dataoutpointer->COL;
d3_coefpointer->ROW = dataoutpointer->ROW;
d3_coefpointer->COL = dataoutpointer->COL;
/* free memory */
free_matrix(temp.array, 1, c_coefpointer->ROW*2, 1,c_coefpointer->COL);
free_matrix(temp1.array, 1, c_coefpointer->ROW*2, 1,c_coefpointer->COL);
free_matrix(temp2.array, 1, c_coefpointer->ROW*2, 1,c_coefpointer->COL);
free_matrix(temp3.array, 1, c_coefpointer->ROW*2, 1,c_coefpointer->COL);
free_matrix(temp4.array, 1, c_coefpointer->ROW*2, 1,c_coefpointer->COL);
free_vector(rowin.vector, 1, 4*dataoutpointer->COL);
free_vector(rowout.vector, 1, 8*dataoutpointer->COL);
free_vector(colin.vector, 1, 4*dataoutpointer->COL);
free_vector(colout.vector, 1, 8*dataoutpointer->COL);
}
```

B.2.8 Listing of SPCONVLV.C

```
/*************************************************************************/
/*************************************************************************/
/***            WAVELET  SPACIAL CONVOLUTION SUBROUTINE                ***/
/*************************************************************************/
/*************************************************************************/
/*  DATE:  26 July 91
  VERSION: 1.0
  NAME:  spconvlv.c
  DESCRIPTION: This subroutine will do a convolution of two time
  signals in the time domain by means of a shift-multiply-sum method.
  This program intended use is to replace the convlv() subroutine
  now being used in the wavelet convolve.c and reconvolve.c portions
  of the wave2 program.
  FILES READ: NONE  (Passed by reference from the caller.)
  FILES WRITTEN:  (Passed by reference back to the caller.)
  HEADERS USED:   <stdio.h>", "jsmacros.h"
  CALLING PROGRAMS:   decompose.c
  PROGRAMS CALLED: nrutil.c
  AUTHOR: Steve Smiley and J. Stewart Laing
```

```
  HISTORY: Initial Version.
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include <stdio.h>
#include "jsmacros.h"

float   *vector();
void    free_vector();
void    free_vector();

/************************/
/*  MAIN PROGRAM BODY  */
/************************/

void spconv1v (input, input_length, filter, filter_length ,dumby,output)
   float   *input, *output, *filter;
   int             input_length, filter_length, dumby;
{

/*************************/
/* declare variables    */
/*************************/

int           i, j;
float         *temp, *temp2;

/*********************/
/* allocate memory  */
/*********************/

CREATE_FLOAT_VECTOR(temp,1,2*input_length);
loop1i(2*input_length) temp[i] = 0.0;

CREATE_FLOAT_VECTOR(temp2,1,2*input_length);
loop1i(2*input_length) temp2[i] = 0.0;

/*********************************/
/* diagnostic print statements   */
/*********************************/

 /* printf("\n  filter length is %d", filter_length);
   printf("\n  input length is %d", input_length); */
loop1i(2*input_length)
      output[i] = 0.0;


/*********************************/
/* load first level coefficients */
/*********************************/

 loop1i(input_length/2){
      temp[i + filter_length -1] = input[i];

   /*printf("\n i= %d, temp[i + filter_length/2]
= %f", i, temp[i + filter_length/2]); */
          }

/*******************************************************/
/*  fill in both ends of vector with flip of image    */
/*******************************************************/
loop1i(filter_length -1){
  temp[filter_length - i] = temp[filter_length + 1];
  temp[filter_length -1 + input_length/2 + i] =
temp[filter_length -1 + input_length/2 - i];
             }
```

```
/*********************************/
/* convolution of signal         */
/*********************************/
loop1i(input_length/2 + filter_length -1){
     loop1j(filter_length)
      temp2[i] += temp[i+j-1]*filter[j];
              }

/************************************************/
/* load proper convolution coefficients         */
/************************************************/
loop1i(input_length/2)
   output[i] = temp2[filter_length/2 + i];
free_vector(temp,1,2*input_length);
free_vector(temp2,1,2*input_length);

/*  printf("\n i = %d,output=%f",i, output[i]); */
 }
```

B.2.9 Listing of NRUTIL.C (See Appendix F.2) [13]

B.2.10 Listing of JSMACROS.H (See Appendix F.2)

B.2.11 Listing of STEWMATH.H (See Appendix F.2)

B.2.12 Listing of MAKEFILE

```
# Makefile routine for the wave2 program by Laing and Smiley.
 DEFLAGS = -g
OBJS = main-wave.o loadimage.o filters.o convolve.o spconvlv.o\
        decompose.o reconstruct.o reconvolve.o nrutil.o
spwave2: $(OBJS)
@echo "linking ..."
cc $(OBJS) -o wave2 $(DEFLAGS) -lm
main-wave.o: main-wave.c
cc -c $(DEFLAGS) main-wave.c
loadimage.o: loadimage.c
cc -c $(DEFLAGS) loadimage.c
filters.o: filters.c
cc -c $(DEFLAGS) filters.c
spconvlv.o: spconvlv.c
cc -c $(DEFLAGS) spconvlv.c
convolve.o: convolve.c
cc -c $(DEFLAGS) convolve.c
reconvolve.o: reconvolve.c
cc -c $(DEFLAGS) reconvolve.c
decompose.o: decompose.c
cc -c $(DEFLAGS) decompose.c
resonstruct.o: reconstruct.c
cc -c $(DEFLAGS) reconstruct.c
nrutil.O: nrutil.c
cc -c $(DEFLAGS) nrutil.c
```

B.3 1D System Description

The following is a list of functions which comprise the *wave1* program.

1. `main_wave1.c` - The main driver program for wave.
2. `loadsignal.c` - A routine to load the input signal from an ascii data file.
3. `decompose1.c` - A routine that controls the decomposition.
4. `reconstruct1.c` - A routine that controls the reconstruction.
5. `filters.c` - A routine that provides the coefficient values of the $h(n)$ and $g(n)$ response functions (See Appendix B.2 for listing).
6. `convolve1.c` - A routine that controls the convolutions for decomposition.
7. `reconvolve1.c` - A routine that controls the convolutions for reconstruction.
8. `spconvlv.c` - A routine that performs the spatial convolutions (See Appendix B.2 for listing).
9. `makefile` - A makefile that is used to compile and link the source code to make an executable file.
10. `jsmacros.h` - An include file that contains macros we found useful in our programming environment. This file must be present in the directory where compilation takes place (See Appendix F.2 for listing).
11. `stewmath.h` - An include file containing some math routines specific to our program. It must be present in the directory where complilation takes place (See Appendix F.2 for listing).
12. `nrutil.c` - Source code that contains utility macros for dynamic memory allocation (See Appendix F.2 for listing).

Typing *"make"* at the command prompt in any directory with all of the above files present will create the appropriate object code and an executable file called *wave1* that may be

executed by typing *"wave1"* at the command prompt.

The intended input to the program is a 1d signal in raw ascii format in which each sample of the signal is stored in a file, one number per line. For example, a signal that is 512 samples will consist of 512 lines each with one decimal integer number representing the value of that sample. The output of the program are ascii files representing the scale and detail wavelet coefficients in floating point format. For an in depth explanation of the these coefficients and the algorithm, see the author's theses. The algorithm implemented in this program is taken from a paper by Stephan Mallat. The paper is referenced in the authors theses. Be aware that we found some printing mistakes in the paper which are addressed in our theses. The program was developed on Sun sparcstation 2's. But, it should compile on any system with an ansi standard C compiler. To compile the program, type make at the command prompt with the default directory set to the current directory. Object files will then be created and linked into an executable file called wave1. Then to run the program, type wave1 at the command prompt. A menu should appear first with four choices. If not done at the command line entry into the program, a file must be loaded from the current directory before either decomposition or reconstruction can be executed. Once a file is loaded the Decomposition can be selected. Then the Reconstruction can be selected. The Reconstruction portion depends on files generated by the Decomposition portion. But, it is not necessary to run the Decomposition during the same session as the Reconstruction as long as the Decompostion was run in a prior session and the files still reside in the current directory. An alternate way to start the program is to type wave1 followed by the name of the input file and its size. The size of the input file must be a power of two. At this time the largest file used is a 512 sampled signal. It is possible to specify the path to an input file that is not in the current directory either relative to the current directory or absolutely from the root. However, if this is done, the output files will be sent to that same directory. To review the usage of wave1 is

```
command prompt: wave1 [infilename] [size]
```

The infilename and size are optional but if the infilename is given its size along one dimension of the power of two sampled signal must be given as well.

Also, only one file may be input in any one session. This fact is not obvious from the program menu, so be aware. If you try to select the Load signal option from the main menu after you have already loaded a file, the result has not been fully characterized. In other words, we haven't tried to figure out what would happen. This menu option is provided as an alternative to specifying the file on the command line.

The filters available are presently limited to the some of the Daubechies wavelets and the Cubic Spline wavelet. But it is a simple process to add new filters to the filters.c program in the same fasion as those already included. To generate the H and G filters, see our theses for references.

B.4 1D Multiresolution Wavelet Analysis Software

B.4.1 Listing of MAIN-WAVE1.C

```
/*****************************************************************************/
/*****************************************************************************/
/***            WAVELET ANALYZER MAIN PROGRAM DRIVER                       **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  09 April 91, 18 June 91, 16 August 91
    VERSION: 3.0
    NAME:  main-wave1.c
    DESCRIPTION:  This program performs a multiresolution wavelet analisys
    of an input signal with a wavelet from its internal library chosen
    interactively by the user. It handles the menu interface with the
    user and drives the subroutines that take input, analyze, produce
    output. The the wavelet decomposition algorithm is a pyramid algorithm
    proposed by Stephan Mallat in A Theory for Multiresolution Signal
    Decomposition: The Wavelet Representation published in IEEE Trans.
    on Pattern Anal. and Machine Intel. July 89. The algorithm uses a pair
    of mirror filters derived from the scaling function, phi(x). The user
    may enter the intended input signal file from the command line following
    the calling command 'wave1' or the user may wait to be prompted for
    the input file name and size after starting the program with the same
    command.
    FILES READ: NONE  (A subroutine reads the input files.)
    FILES WRITTEN:  NONE (Subroutines write out the saved data in files.)
    HEADERS USED:   <stdio.h>, "jsmacros.h", "stewmath.h"
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED: signalload.c, reconstruct1.c, decompose1.c
    AUTHOR: Steve Smiley and J. Stewart Laing
    HISTORY: Initial Version; adapted from phiv1.c and haarv1.c
```

```
    Version 2.0 was a rewrite to change the basic algorithm from the using
    inner products to using the Mallat algorithm referenced above.
    Version 3.0 adapted the two dimensional program for one
    dimensional signals.
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include <stdio.h>
#include "jsmacros.h"
#include "stewmath.h"

int_vector    loadsignal();
void         reconstruct();
void         decompose();

/************************/
/*  MAIN PROGRAM BODY  */
/************************/

void main(argc, argv)
   int  argc;
   char *argv[];
{
/*************************/
/* initialize variables */
/************************/

int         selection;
int_vector   signal, *signalpointer = &signal;
char        filename[64];

/*************************************/
/*  load image to be analyzed        */
/*************************************/

if(argc != 3 && argc != 1){
   printf("Usage: wave1 <filename> <# of Samples>\n");
   exit(0);
   }

if(argc == 3){
   signal = loadsignal(filename, argc, argv);
   /*printf("returned from loadimage"); fflush(stdout);*/
   }

do {

      /************************************/
      /*  display menu                    */
      /************************************/

   printf("\n\n    MAIN MENU\n\n");
   printf("      1  =  Load a new signal from disk.\n");
   printf("      2  =  Perform Wavelet Decomposition.\n");
   printf("      3  =  Perform Wavelet Reconstruction.\n");
   printf("      4  =  Exit Program.\n\n");
   printf("   Enter an integer (1-4):");

   scanf("%d", &selection);

   if (selection == 4) break;       /* Quit program */

   argc = 1;

   if (selection == 1) signal = loadsignal(filename, argc, argv);

   else if (selection == 2) decompose(signalpointer, filename);

   else if (selection == 3) reconstruct(signalpointer,
```

```
      filename);
   else {
      printf(" \n\n Just enter an integer from 1 to 4 and");
      printf("press return. \n");
      }
   } while (selection != 4);
/* THE END */
}
```

B.4.2 Listing of LOADSIGNAL.C

```
/*****************************************************************************/
/*****************************************************************************/
/***            WAVELET ANALYZER LOADIMAGE ROUTINE                         **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  10 April 91, 16 August 91
    VERSION: 2.0
    NAME:  loadsignal.c
    DESCRIPTION:  This routine loads an signal into an vector whose name is
    specified by the user interactively. It is intended to be used as a
    subroutine for the wave1 program.
    FILES READ: One file specified by the user.
    FILES WRITTEN:  NONE
    HEADERS USED:   <stdio.h>, <stdlib.h>, "jsmacros.h"
    CALLING PROGRAMS:   main-wave1.c
    PROGRAMS CALLED:   nrutil.c
    AUTHOR: Steve Smiley and J. Stewart Laing
    HISTORY: Version 1.1 was changed to accept square matrices
             only.
     Version 2.0 changed the two dimensional program to
     accept only one dimensional signals. The new
     executable is called wave1 vs wave2 for the old
     one.
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include    <stdio.h>
#include    "jsmacros.h"

int         *ivector();
void        free_ivector();

/*************************/
/*  FUNCTION BODY       */
/*************************/

int_vector loadsignal(infilename, argc, argv)
   char   *infilename[64];
   int    argc;
   char   *argv[];
{

   /*************************/
   /* initialize variables */
   /*************************/

   int       i,j;
   FILE      *infile;
```

```
   int_vector signal;
   /*******************************************/
   /*  create vector to hold the incoming signal  */
   /*******************************************/
   if(argc == 1){
      printf("\n\n Input filename of singal to be analyzed:");
      scanf("%s", infilename);
      printf("\n\n Input the number of Samples in the signal");
      printf("\n data file. (The number must a power of 2):");
      scanf("%d", &signal.length);
   }
   else {
      sprintf(infilename, "%s", argv[1]);
      sscanf(argv[2], "%d", &signal.length);
   }
   signal.vector = ivector(1, signal.length);
   /*************************************/
   /*  load signal to be analyzed       */
   /*************************************/

   OPEN_FILE (infile, infilename, "The wavelet analyzer");
   loop1i(signal.length)
         fscanf(infile,"%d", &signal.vector[i]);
      CLOSE_FILE (i, infilename, "The Wavelet analyzer", infile)
      printf("\n ** The signal %s has been loaded for processing. **\n\n\n",
         infilename);
   return signal;
}
```

B.4.3 Listing of DECOMPOSE1.C

```
/*****************************************************************************/
/*****************************************************************************/
/***            WAVELET DECOMPOSITION SUBROUTINE                           **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  19 June 91, 16 August 91
    VERSION: 2.0
    NAME:  decompose1.c
    DESCRIPTION:  This subroutine is intended to be part of a Wavelet
    analyzing program called "wave1". The algorithm used is discussed in
    the description of the main driver module called "main-wave1.c.
    Data is passed by reference from the main driver module. The data is
    in ascii format arranged in a vector whose dimension is a
    power of 2. This requirement has not only made programming more
    convenient but is required by the convolution routine from Numerical
    Recipes in C: The Art of Scientific Computing.
    FILES READ: NONE  (Passed by reference from the caller.)
    FILES WRITTEN:  Two coefficient files at each level of analysis.
                    The file names begin with the input signal filename
                    and end with an extension of the form ".nX" where
                    n is an integer that represents the level, X is one
    of the letters 'c' or 'd' to represent phi
    or psi coefficients respectively.
    HEADERS USED:   <stdio.h>, "jsmacros.h"
    CALLING PROGRAMS:   main-wave1.c
    PROGRAMS CALLED: convolve1.c, filters.c, nrutil.c
    AUTHOR: Steve Smiley and J. Stewart Laing
```

```
    HISTORY: Initial Version.
    Version 2.0 no longer uses the Fourier domain filtering. Now
    only spactial convolution is done. Also, this version was
    adapted from the two dimensional version 1.0.
*/
/*****************************************************************************/
/*****************************************************************************/

/************************/
/*  DECLARATION SECTION  */
/************************/

#include <stdio.h>
#include "jsmacros.h"

void        convolve();
void        filters();
float       *vector();
void        free_vector();
int         *ivector();

/************************/
/*  MAIN PROGRAM BODY  */
/************************/

void decompose(signalpointer, infilename)
   int_vector *signalpointer;
   char       infilename[];
{

/************************/
/* declare variables    */
/************************/

int           i, j, k, maxlevel, wavelet_type;
float_vector  h_of_n,  h_of_nflipo, g_of_n, g_of_nflipo, phi, phiflipo;
float_vector  phiflipc, *phiflipcpointer = &phiflipc;
float_vector  *h_of_npointer = &h_of_n, *h_of_nflipopointer = &h_of_nflipo;
float_vector  *g_of_npointer = &g_of_n, *g_of_nflipopointer = &g_of_nflipo;
float_vector  *phipointer = &phi, *phiflipopointer = &phiflipo;
float_vector   c_coef, d_coef;
float_vector   *c_coefpointer= &c_coef,*d_coefpointer= &d_coef;
float_vector   temp, *temppointer = &temp;
FILE          *outfile;
char          filename[64], wave_code[64];
int_vector     newsignal, *newsignalpointer = &newsignal;

/************************/
/* allocate memory      */
/************************/

temp.length = signalpointer->length;
temp.vector = vector(1, temp.length);
loop1i(temp.length) temp.vector[i] = 0.0;
c_coef.length = signalpointer->length;
c_coef.vector = vector(1, c_coef.length);
loop1i(c_coef.length) c_coef.vector[i] = 0.0;
d_coef.length = signalpointer->length;
d_coef.vector = vector(1, d_coef.length);
loop1i(d_coef.length) d_coef.vector[i] = 0.0;
newsignal.length = signalpointer->length;
newsignal.vector = ivector(1, newsignal.length);
loop1i(newsignal.length) newsignal.vector[i] = 0;
h_of_n.vector = vector(1,signalpointer->length*2);
loop1i(signalpointer->length*2) h_of_n.vector[i] = 0.0;
g_of_n.vector = vector(1,signalpointer->length*2);
loop1i(signalpointer->length*2) g_of_n.vector[i] = 0.0;
h_of_nflipo.vector = vector(1,signalpointer->length*2);
loop1i(signalpointer->length*2) h_of_nflipo.vector[i] = 0.0;
g_of_nflipo.vector = vector(1,signalpointer->length*2);
```

```
loop1i(signalpointer->length*2) g_of_nflipo.vector[i] = 0.0;
phi.vector = vector(1,2*signalpointer->length);
loop1i(signalpointer->length*2) phi.vector[i] = 0.0;
phiflipo.vector = vector(1,2*signalpointer->length);
loop1i(signalpointer->length*2) phiflipo.vector[i] = 0.0;
phiflipc.vector = vector(1,2*signalpointer->length);
loop1i(signalpointer->length*2) phiflipc.vector[i] = 0.0;
/*************************************/
/*  display menu                     */
/*************************************/

printf("\n\n   DECOMPOSITION MENU\n\n");
printf("      1  =  Piece-wise Constant.(N/A)\n");
printf("      2  =  Piece-wise Linear.(N/A)\n");
printf("      3  =  Daubechies N=2 \n");
printf("      4  =  Daubechies N=3.\n");
printf("      5  =  Daubechies N=4.\n");
printf("      6  =  Daubechies N=5.\n");
printf("      7  =  Daubechies N=6.\n");
printf("      8  =  Daubechies N=7.\n");
printf("      9  =  Daubechies N=8.\n");
printf("      10  =  Daubechies N=9.\n");
printf("      11  =  Daubechies N=10.\n");
printf("      12  =  Splines.\n");
printf("      13  =  Morlet.(N/A)\n");
printf("\n  Enter an integer 1-13: ");

scanf("%d", &wavelet_type);

/* error handling for invalid input */

if (wavelet_type < 3 || wavelet_type > 13) {
   printf("\nYou have chosen an Invalid Wavelet type or");
   printf("\nthis type is not currently available.");
   } /* end if */
else {

   /*********************************************/
   /* Set wave_code for use in output filenames. */
   /*********************************************/

   if (wavelet_type == 3) sprintf(wave_code, "db2");
   if (wavelet_type == 4) sprintf(wave_code, "db3");
   if (wavelet_type == 5) sprintf(wave_code, "db4");
   if (wavelet_type == 6) sprintf(wave_code, "db5");
   if (wavelet_type == 7) sprintf(wave_code, "db6");
   if (wavelet_type == 8) sprintf(wave_code, "db7");
   if (wavelet_type == 9) sprintf(wave_code, "db8");
   if (wavelet_type == 10) sprintf(wave_code, "db9");
   if (wavelet_type == 11) sprintf(wave_code, "db0");
   if (wavelet_type == 12) sprintf(wave_code, "spl");

   /***************************/
   /* Generate Phi and Filters */
   /***************************/

   filters (wavelet_type,h_of_npointer,g_of_npointer,phipointer);
   flipo(phipointer, phiflipopointer);

   h_of_nflipopointer = h_of_npointer;
   g_of_nflipopointer = g_of_npointer;

loop1i(signalpointer->length)
   temppointer->vector[i] = (float)signalpointer->vector[i];

   /*****************************************************/
```

```
   /* Call convolution routine and save the coefficient vectors for */
   /* each level of analysis.                                       */
   /*****************************************************************/
   maxlevel = LOG2(signalpointer->length); /* Calculate the highest level */
   k=1;
   loopk(maxlevel){
      if (temp.length >= h_of_n.length){ /* signal has to be bigger than filter */
         printf("\nPerforming convolution with filters, level");
 printf("%d...", k+1);
 convolve(temppointer, h_of_nflipopointer, g_of_nflipopointer,
 c_coefpointer, d_coefpointer);
 sprintf(filename, "%s.%d.c.%s", infilename, k+1, wave_code);
         CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
         loop1i(c_coef.length) fprintf(outfile,"%f\n",c_coef.vector[i]);
         CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile)
         sprintf(filename, "%s.%d.d.%s", infilename, k+1,wave_code);
         CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
         loop1i(d_coef.length)fprintf(outfile,"%f\n",d_coef.vector[i]);
         CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile)
         temp.length = c_coef.length;
         loop1i(temp.length) temp.vector[i] = c_coef.vector[i];
      } /* end if */
   } /* end loop */
} /* end else */
/* free memory */
free_vector(temp.vector, 1, temp.length);
free_vector(c_coef.vector, 1, c_coef.length);
free_vector(d_coef.vector, 1, d_coef.length);
free_vector(h_of_n.vector,1,signalpointer->length*2);
free_vector(g_of_n.vector,1,signalpointer->length*2);
free_vector(phi.vector,1,signalpointer->length*2);
free_vector(phiflipo.vector,1,signalpointer->length*2);
free_vector(phiflipc.vector,1,signalpointer->length*2);
/* THE END */
}
```

B.4.4 Listing of RECONSTRUCT1.C

```
/*****************************************************************************/
/*****************************************************************************/
/***            WAVELET RECONSTRUCTION SUBROUTINE                          **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  2 July 91, 16 August 91
  VERSION: 3.0
  NAME:  reconstruct1.c
  DESCRIPTION:  This subroutine is intended to be part of a Wavelet
  analyzing program called "wave1". The algorithm used is discussed in
  the description of the main driver module called "main-wave1.c.
  It controls the portion of the program that reconstructs a previously
  decomposed signal using Mallat's multiresolution algorithm referenced
  in the description of the calling program, "main-wave1.c".
  FILES READ:     Four coefficient files at each level of analysis.
                  The file names begin with the input signal filename
                  and end with an extension of the form ".nX" where
                  n is an integer that represents the level, X is one of
                  the letters 'c' or 'd' to represent phi or psi coef-
                  ficients respectively.
```

```
  FILES WRITTEN:  One file with the extension ".rec".
  HEADERS USED:   <stdio.h>, "jsmacros.h"
  CALLING PROGRAMS:   main-wave1.c
  PROGRAMS CALLED:  filters.c, reconvolve1.c, spconvlv.c, nrutil.c
  AUTHOR: Steve Smiley and J. Stewart Laing
  HISTORY: Initial Version.
  Version 2.0 is adapted to use the spatial correlation and not
  the Fourier convolution.
  Version 3.0 adapted the two dimensional program to handle only
  1 dimensional signals. The command is wave1 vs wave2.
*/
/*************************************************************************/
/*************************************************************************/

/************************/
/*  DECLARATION SECTION  */
/************************/

#include <stdio.h>
#include "jsmacros.h"

void    filters();
void    reconvolve();
float   *vector();
void    free_vector();
int     *ivector();
void    free_ivector();

/************************/
/*  MAIN PROGRAM BODY  */
/************************/

void reconstruct(signalpointer,infilename)
   int_vector *signalpointer;
   char      infilename[];
{

/************************/
/* declare variables    */
/************************/

int           i, j, k, l, maxlevel, wavelet_type;
float_vector  h_of_n, h_of_nflipo, h_of_nflipc, g_of_n;
float_vector  g_of_nflipo, g_of_nflipc, phi, phiflipc;
float_vector  *h_of_npointer = &h_of_n, *g_of_npointer = &g_of_n;
float_vector  *h_of_nflipopointer = &h_of_nflipo;
float_vector  *g_of_nflipopointer = &g_of_nflipo;
float_vector  *h_of_nflipcpointer = &h_of_nflipc;
float_vector  *g_of_nflipcpointer = &g_of_nflipc;
float_vector  *phipointer = &phi, *phiflipcpointer = &phiflipc;
float_vector   c_coef, d_coef,
float_vector   *c_coefpointer= &c_coef,*d_coefpointer= &d_coef;
float_vector   temp, *temppointer = &temp;
int_vector     newsignal, *newsignalpointer = &newsignal;
FILE          *outfile, *infile;
char          filename[64], wave_code[64];
float          holder[64];

/************************/
/* allocate memory      */
/************************/

temp.length = signalpointer->length;
temp.vector = vector(1, temp.length);
loop1i(temp.length) temp.vector[i] = 0.0;
newsignal.length = signalpointer->length;
newsignal.vector = ivector(1, newsignal.length);
```

```
loop1i(newsignal.length) newsignal.vector[i] = 0.0;
c_coef.length = signalpointer->length;
c_coef.vector = vector(1, c_coef.length);
loop1i(c_coef.length) c_coef.vector[i] = 0.0;
d_coef.length = signalpointer->length;
d_coef.vector = vector(1, d_coef.length);
loop1i(d_coef.length) d_coef.vector[i] = 0.0;
h_of_n.vector = vector(1,signalpointer->length*2);
loop1i(signalpointer->length*2) h_of_n.vector[i] = 0.0;
g_of_n.vector = vector(1,signalpointer->length*2);
loop1i(signalpointer->length*2) g_of_n.vector[i] = 0.0;
phi.vector = vector(1,2*signalpointer->length);
loop1i(signalpointer->length*2) phi.vector[i] = 0.0;
phiflipc.vector = vector(1,2*signalpointer->length);
loop1i(signalpointer->length*2) phiflipc.vector[i] = 0.0;
h_of_nflipo.vector = vector(1,signalpointer->length*2);
loop1i(signalpointer->length*2) h_of_nflipo.vector[i] = 0.0;
g_of_nflipo.vector = vector(1,signalpointer->length*2);
loop1i(signalpointer->length*2) g_of_nflipo.vector[i] = 0.0;
h_of_nflipc.vector = vector(1,signalpointer->length*2);
loop1i(signalpointer->length*2) h_of_nflipc.vector[i] = 0.0;
g_of_nflipc.vector = vector(1,signalpointer->length*2);
loop1i(signalpointer->length*2) g_of_nflipc.vector[i] = 0.0;
/************************************/
/*  display menu                    */
/************************************/
   printf("\n\n   RECONSTRUCTION MENU\n\n");
   printf("      1  =  Piece-wise Constant.(N/A)\n");
   printf("      2  =  Piece-wise Linear.(N/A)\n");
   printf("      3  =  Daubechies N=2.\n");
   printf("      4  =  Daubechies N=3.\n");
   printf("      5  =  Daubechies N=4.\n");
   printf("      6  =  Daubechies N=5.\n");
   printf("      7  =  Daubechies N=6.\n");
   printf("      8  =  Daubechies N=7.\n");
   printf("      9  =  Daubechies N=8.\n");
   printf("      10  =  Daubechies N=9.\n");
   printf("      11  =  Daubechies N=10.\n");
   printf("      12  =  Splines.\n");
   printf("      13  =  Morlet.(N/A)\n");
   printf("   Enter an integer (1-13):");
   scanf("%d", &wavelet_type);
   if(wavelet_type < 1 || wavelet_type > 13 ){
      printf("\nYou have chosen an invalid wavelet or");
      printf("\nit is not currently available.");
      }
   else {
      /*****************************************/
      /* Set value of wave_code for input filename */
      /*****************************************/
      if (wavelet_type == 3) sprintf(wave_code, "db2");
      if (wavelet_type == 4) sprintf(wave_code, "db3");
      if (wavelet_type == 5) sprintf(wave_code, "db4");
      if (wavelet_type == 6) sprintf(wave_code, "db5");
      if (wavelet_type == 7) sprintf(wave_code, "db6");
      if (wavelet_type == 8) sprintf(wave_code, "db7");
      if (wavelet_type == 9) sprintf(wave_code, "db8");
      if (wavelet_type == 10) sprintf(wave_code, "db9");
```

```
      if (wavelet_type == 11) sprintf(wave_code, "db0");
      if (wavelet_type == 12) sprintf(wave_code, "spl");
      /****************************/
      /* Generate Phi and Filters */
      /****************************/
      filters(wavelet_type,h_of_npointer,g_of_npointer,phipointer);
/*****************************************/
/*            flip the filters           */
/*****************************************/

loop1j(h_of_npointer->length)
  holder[h_of_npointer->length +1 -j]= h_of_npointer->vector[j];
loop1j(h_of_npointer->length)
  h_of_npointer->vector[j] = holder[j];
loop1j(g_of_npointer->length)
  holder[g_of_npointer->length +1 -j]= g_of_npointer->vector[j];
loop1j(g_of_npointer->length)
  g_of_npointer->vector[j] = holder[j];

h_of_nflipcpointer= h_of_npointer;
g_of_nflipcpointer= g_of_npointer;


      /*****************************************************************/
      /* Call reconvolution routine to reconstruct from coursest phi   */
      /* coefficients and all of the psi coefficients.                 */
      /*****************************************************************/
      maxlevel = LOG2(signalpointer->length);/*Calculate the highest level*/

      temp.length = 1;
   do {              /* make sure signal is bigger than filter */
      temp.length *=2;
      --maxlevel;
      } while (temp.length < h_of_n.length/2);
   c_coef.length = temp.length;
   d_coef.length = temp.length;
   l = 1;
   for(k=maxlevel;k>0;--k){
         if(l == 1){
            sprintf(filename, "%s.%d.c.%s", infilename, k,wave_code);
            OPEN_FILE(infile, filename, "The Wavelet Analyzer")
            loop1i(c_coef.length)
               fscanf(infile, "%f\n", &c_coef.vector[i]);
    CLOSE_FILE(i, filename, "The Wavelet Analyzer", infile)
            l = 0;
            } /* end if */
         else {
            c_coef.length = temp.length;
            loop1i(c_coef.length) c_coef.vector[i] = temp.vector[i];
    } /* end else */
         sprintf(filename, "%s.%d.d.%s", infilename, k,wave_code);
         OPEN_FILE(infile, filename, "The Wavelet Analyzer")
         loop1i(d_coef.length)
            fscanf(infile, "%f\n", &d_coef.vector[i]);
         CLOSE_FILE(i, filename, "The Wavelet Analyzer", infile)
         printf("\nPerforming reconvolution with filters, level %d...", k);
         reconvolve(temppointer, h_of_nflipcpointer, g_of_nflipcpointer,
    c_coefpointer, d_coefpointer);
         if(wavelet_type == 12)
```

```
          loop1i(temp.length) temp.vector[i] *= 2;
        sprintf(filename, "%s.%d.c.%s.rec", infilename,k-1,wave_code);
        CREATE_FILE(outfile, filename, "The Wavelet Analyzer")
        loop1i(temp.length)
           fprintf(outfile, "%f\n", temp.vector[i]);
        CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile)
        } /* end loop */
   } /* end else */
/* free memory */
free_vector(temp.vector, 1, temp.length);
free_ivector(newsignal.vector, 1, newsignal.length);
free_vector(c_coef.vector, 1, c_coef.length);
free_vector(d_coef.vector, 1, d_coef.length);
/* THE END */
}
```

B.4.5 Listing of FILTERS.C (See Appendix B.2)

B.4.6 Listing of CONVOLVE1.C

```
/*****************************************************************************/
/*****************************************************************************/
/***              WAVELET CONVOLUTION SUBROUTINE                           **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  19 June 91, 16 August 91
  VERSION: 2.0
  NAME:  convolve1.c
  DESCRIPTION:  This subroutine is intended to be part of a Wavelet
  analyzing program called "wave1". The algorithm used is discussed in
  the description of the main driver module called "main-wave1.c".
  Data is passed by reference from the decomposition subroutine. Data is
  in ascii format arranged in a vector whose dimension is a
  power of 2. This requirement has not only made programming more
  convenient but is required by the convolution routine from Numerical
  Recipes in C: The Art of Scientific Computing.
  FILES READ: NONE  (Passed by reference from the caller.)
  FILES WRITTEN:  (Passed by reference back to the caller.)
  HEADERS USED:   <stdio.h>, "jsmacros.h"
  CALLING PROGRAMS:   decompose1.c
  PROGRAMS CALLED:  spconvlv.c, nrutil.c
  AUTHOR: Steve Smiley and J. Stewart Laing
  HISTORY: Initial Version.
  Version 2.0 was adapted from the two dimensional version 1.0
  to handle one dimensional signals. It does not use the Forier
  space filtering indicated above.
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include <stdio.h>
#include "jsmacros.h"

float   *vector();
void    free_vector();
```

```
void    spconvlv();

/************************/
/*  MAIN PROGRAM BODY  */
/************************/

void convolve(datainpointer,  h_of_npointer ,g_of_npointer, c_coefpointer,
   d_coefpointer)
   float_vector *datainpointer;
   float_vector *h_of_npointer, *g_of_npointer;
   float_vector  *c_coefpointer,*d_coefpointer;
{
/************************/
/* declare variables    */
/************************/

int           i, j;
float_vector  vectin,vectout,response;
float_vector   temp;
FILE *outfile;
char filename[64];

/********************/
/* allocate memory  */
/********************/

temp.vector = vector(1, datainpointer->length);
loop1i(datainpointer->length) temp.vector[i] = 0.0;
vectin.vector = vector(1,2*datainpointer->length);
loop1i(2*datainpointer->length) vectin.vector[i] = 0.0;
vectout.vector = vector(1,4*datainpointer->length);
loop1i(datainpointer->length*4) vectout.vector[i] = 0.0;
response.vector = vector(1,2*datainpointer->length);
loop1i(datainpointer->length*2) response.vector[i] = 0.0;

vectin.length = 2*datainpointer->length;

/************************/
/* perform convolution */
/************************/

printf("\nConvovling signal with h(-n)...");
loop1j(datainpointer->length*2)
   response.vector[j] = h_of_npointer->vector[j];
loop1j(datainpointer->length)
   vectin.vector[j] = datainpointer->vector[j];
spconvlv(vectin.vector,vectin.length,response.vector,
   h_of_npointer->length,1,vectout.vector);
loop1j(datainpointer->length/2)
   c_coefpointer->vector[j] = vectout.vector[2*j];
    /* downsample by selectiny even cols */

printf("\nConvovling signal with g(-n)...");
loop1j(datainpointer->length*2)
      response.vector[j] = g_of_npointer->vector[j];
loop1j(datainpointer->length)
   vectin.vector[j] = datainpointer->vector[j];
spconvlv(vectin.vector,vectin.length,response.vector,
   g_of_npointer->length,1,vectout.vector);
loop1j(datainpointer->length/2)
   d_coefpointer->vector[j] = vectout.vector[2*j];

   /* reset signal indeces. */

c_coefpointer->length = datainpointer->length/2;
d_coefpointer->length = datainpointer->length/2;

/* free memory */
```

```
free_vector(temp.vector, 1, datainpointer->length);
free_vector (vectin.vector,1,2*datainpointer->length);
free_vector (vectout.vector,1,4*datainpointer->length);
free_vector (response.vector,1,2*datainpointer->length);
/* THE END */
}
```

B.4.7 Listing of RECONVOLVE1.C

```
/*****************************************************************************/
/*****************************************************************************/
/***              WAVELET RECONVOLUTION SUBROUTINE                         **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  2 July 91, 16 August 91
  VERSION: 2.0
  NAME:  reconvolve1.c
  DESCRIPTION:  This subroutine is intended to be part of a Wavelet
  analyzing program called "wave1". The algorithm used is referenced in
  the description of the main driver module called "main-wave1.c".
  Data is passed by reference from the reconstruction subroutine. Data is
  in ascii format arranged in a vector whose dimension is a
  power of 2. This requirement has not only made programming more
  convenient but is required by the convolution routine from Numeric
  Recipes in C: The Art of Scientific Computing.
  FILES READ: NONE  (Passed by reference from the caller.)
  FILES WRITTEN:  NONE (Passed by reference back to the caller.
  HEADERS USED:   <stdio.h>, "jsmacros.h"
  CALLING PROGRAMS:   reconstruct1.c
  PROGRAMS CALLED: spconvlv.c, nrutil.c
  AUTHOR: Steve Smiley and J. Stewart Laing
  HISTORY: Initial Version.
  Version 2.0 adapted from 1.0 allows only one dimensional
  signals to be decomposed. It does not use Fourier filtering.
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include <stdio.h>
#include "jsmacros.h"

float   *vector();
void    free_vector();

/************************/
/*  MAIN PROGRAM BODY  */
/************************/

void reconvolve(dataoutpointer,h_of_npointer,g_of_npointer,
   c_coefpointer,d_coefpointer)
   float_vector *dataoutpointer;
   float_vector *h_of_npointer, *g_of_npointer;
   float_vector  *c_coefpointer,*d_coefpointer;
{

/**********************/
/* declare variables    */
/**********************/

int          i, j;
```

```
float_vector  vectin,vectout, response;
float_vector   temp,temp1;
char          filename[64];
FILE          *outfile;

/*******************/
/* allocate memory  */
/*******************/

temp.length = c_coefpointer->length*2;
temp.vector = vector(1, temp.length);
loop1i(temp.length) temp.vector[i] = 0.0;
temp1.length = c_coefpointer->length*2;
temp1.vector = vector(1, temp1.length);
loop1i(temp1.length) temp1.vector[i] = 0.0;
vectin.vector = vector(1,temp.length*2);
loop1i(temp.length*2) vectin.vector[i] = 0.0;
vectout.vector = vector(1,4*temp.length);
loop1i(temp.length*4) vectout.vector[i] = 0.0;
response.vector = vector(1,temp.length*2);
loop1i(temp.length*2) response.vector[i] = 0.0;

vectin.length = 4*_coefpointer->length;
dataoutpointer->length = c_coefpointer->length*2;

/***********************/
/* perform convolution */
/***********************/

printf("\nConvovling c_coef with h(n)...");
loop1j(c_coefpointer->length) vectin.vector[2*j] = c_coefpointer->vector[j];
loop1j(vectin.length) response.vector[j]=h_of_npointer->vector[j];
spconvlv(vectin.vector,vectin.length,response.vector,
                                h_of_npointer->length,1,vectout.vector);
loop1j(c_coefpointer->length*2) temp.vector[j] = vectout.vector[j];
/* zeros were added between each row before convolution */

printf("\nConvovling d_coef with g(n)...");
loop1j(d_coefpointer->length) vectin.vector[2*j] = d_coefpointer->vector[j];
loop1j(vectin.length) response.vector[j]=g_of_npointer->vector[j];
spconvlv(vectin.vector,vectin.length,response.vector,
                                g_of_npointer->length,1,vectout.vector);
loop1j(d_coefpointer->length*2) temp1.vector[j] = vectout.vector[j];
/* zeros are added between each row before convolution */

/* Add temp vectors  */

loop1i(dataoutpointer->length)
   dataoutpointer->vector[i] = temp.vector[i] + temp1.vector[i];

   /* reset vector indeces. */

d_coefpointer->length = dataoutpointer->length;

/* free memory */

free_vector(temp.vector, 1, c_coefpointer->length*2);
free_vector(temp1.vector, 1, c_coefpointer->length*2);
free_vector(vectin.vector, 1, 4*dataoutpointer->length);
free_vector(vectout.vector, 1, 8*dataoutpointer->length);
}
```

B.4.8 Listing of SPCONVLV.C (See Appendix B.2)

B.4.9 Listing of NRUTIL.C (See Appendix F.2) [13]

B.4.10 Listing of JSMACROS.H (See Appendix F.2)

B.4.11 Listing of STEWMATH.H (See Appendix F.2)

B.4.12 Listing of MAKEFILE

```
# Makefile routine for the wave1 program by Laing and Smiley.
 DEFLAGS = -g
OBJS = main-wave1.o loadsignal.o filters.o convolve1.o spconvlv.o\
        decompose1.o reconstruct1.o reconvolve1.o nrutil.o
spwave2: $(OBJS)
@echo "linking ..."
cc $(OBJS) -o wave1 $(DEFLAGS) -lm
main-wave1.o: main-wave1.c
cc -c $(DEFLAGS) main-wave1.c
loadsignal.o: loadsignal.c
cc -c $(DEFLAGS) loadsignal.c
filters.o: filters.c
cc -c $(DEFLAGS) filters.c
spconvlv.o: spconvlv.c
cc -c $(DEFLAGS) spconvlv.c
convolve1.o: convolve1.c
cc -c $(DEFLAGS) convolve1.c
reconvolve1.o: reconvolve1.c
cc -c $(DEFLAGS) reconvolve1.c
decompose1.o: decompose1.c
cc -c $(DEFLAGS) decompose1.c
resonstruct1.o: reconstruct1.c
cc -c $(DEFLAGS) reconstruct1.c
nrutil.0: nrutil.c
cc -c $(DEFLAGS) nrutil.c
```

Appendix C. *Software for Utilities*

C.1 Description of Utilities

The following is a list of the software utilities used in this thesis. It includes Header files and subroutines that are found in much of the software listed in earlier appendices and command line programs that filter individual files.

1. `jsmacros.h` - A header file containing macros used widely in the software written for this thesis.
2. `macros.h` - A header file containing some macros used early on in the software of this thesis.
3. `stewmath.h` - A header file containing an integer math routine to take the base 2 logarithm of an integer number.
4. `asift.c` - A program that converts an input file of float ASCII values, one per line, to integer ASCII values, one per line. The values are clipped at a minimum value of 0 and a maximum of 255. After conversion and before clipping, the absolute value of each number is taken.
5. `daub.c` - A program used to generate $g(n)$, $\phi(x)$, and $\psi(x)$ given an $h(n)$. All $h(n)$ values are hard coded and must be entered before compilation. Other input is interactive.
6. `epsview.c` - A program that converts an input file from ASCII format in which each line holds an integer number to hex with an Ecapsulated PostScript header.
7. `matrixtoascii.c` - A program that converts a Khoros ASCII output to a file that has one integer per line [31]. This program strips off the matrix coordinates of the values.
8. `nrutil.c` - A set of utilities provided by *Numverical Recipies in C* used in this thesis mostly for dynamic memory allocation [13].

9. `threshold.c` - A program that thresholds an input file of ASCII values eliminating a user specified window of minimum and maximum values. All values inside the window are set to 255 and all values outside the window are set to 0. This creates a black and white binary representation of the input file.

10. `rd834.c` - A program to convert raw SAR imagery to a complex format.

11. `logb.c` - A program to convery complex format SAR imagery to a byte format scaled between 0-255 grey scales and ready for consumption by the Khoros system.

12. `extract.c` - This program generates data vectors from multiresolution decomposition coefficient files for use in Dan Zahirniak's Radial Basis Function neural network.

13. `center.c` - This program generates data vectors from multiresolution decomposition coefficient files for use in Dan Zahirniak's Radial Basis Function neural network. The training data is extracted from large blocks or areas of the image.

14. `normalizeA.c` - This program will normalize a set of data vectors used in the RBF network.

C.2 Utility Software

C.2.1 Listing of ISMACROS.C

```
/*******************************************************
     Convenient Macros for WAVE program
*******************************************************/
/*** MACROS ***/

#define CREATE_MATRIX_ROW(A,B,C) A = (C **)calloc(B, sizeof(C *))
#define DELETE_MATRIX_ROW(A,C) free((C *) A)

#define CLOSE_FILE(A,B,C,D) if((A=fclose(D)) == EOF) { \
        printf(strcat(C,":file may already be closed - %s.\n"),B); }

#define CREATE_MATRIX_COL(A,B,C,D) for(i=0; i<B; ++i) A[i] = (D *) \
        calloc(C, sizeof(D))
#define DELETE_MATRIX_COL(A,B,D) for(i=0; i<B; ++i) free((D *) A[i])

#define CREATE_VECTOR(A,B,C) A = (C *)calloc(B, sizeof(C))
#define DELETE_VECTOR(A) free(A)

#define loop1i(A) for(i=1;i<=A;i++)
#define loop1j(A) for(j=1;j<=A;j++)
#define loop1l(A) for(l=1;l<=A;l++)
#define loop1k(A) for(k=1;k<=A;k++)
#define loop1ij(A,B) for(i=1;i<=A;i++) for(j=1;j<=B;j++)
#define loop1kl(A,B) for(k=1;k<=A;k++) for(l=1;l<=B;l++)
```

```
#define CREATE_FLOAT_VECTOR(A,B,C) A = vector(B,C)
#define CREATE_INT_VECTOR(A,B,C) A = ivector(B,C)
#define CREATE_DOUBLE_VECTOR(A,B,C) A = dvector(B,C)
#define CREATE_FLOAT_MATRIX(A,B,C,D,E) A = matrix(B,C,D,E)
#define CREATE_INT_MATRIX(A,B,C,D,E) A = imatrix(B,C,D,E)
#define CREATE_DOUBLE_MATRIX(A,B,C,D,E) A = dmatrix(B,C,D,E)



struct int_array {
   int **array;
   int ROW, COL;
};
typedef struct int_array int_array;
struct float_array {
   float **array;
   int ROW, COL;
};
typedef struct float_array  float_array;
struct phi_array {
   float **array;
   int ROW, COL;
   int intervals;
};
typedef struct phi_array phi_array;
struct float_vector {
   float *vector;
   int length;
};
typedef struct float_vector float_vector;
```

C.2.2 Listing of MACROS.C

```
/*****************************************************
Convenient Macros for Perceptron Package by Capt Greg Tarr
*****************************************************/
/*** MACROS ***/
#ifdef LEO
#define REAL_FMT "%g"
#else
#define REAL_FMT "%lg"
#endif
#ifdef NEXT
#undef REAL_FMT
#define REAL_FMT "%lf"
#endif
#define Boolean int
#define False 0
#define True  1
/** Dominant Sensor Definitions **/
#define SINGLE 0
#define FLIR 1
#define RNG  2
/** Mask Definitions **/
#define OFF 0.0
#define ON  1.0
char      junk_response[256];
```

```
#define skip_line(A) fgets(junk_response, 256, A)
#define skip_line gets(junk_response)
#define rloopi(A) for(i=(A)-1;i>=0;i--)
#define rloopj(A) for(j=(A)-1;j>=0;j--)
#define rloopk(A) for(k=(A)-1;k>=0;k--)
#define rloopl(A) for(l=(A)-1;l>=0;l--)
#define rloopm(A) for(m=(A)-1;m>=0;m--)
#define rloopn(A) for(n=(A)-1;n>=0;n--)
#define rloopp(A) for(p=(A)-1;p>=0;p--)
#define rloopij(A,B) for(i=(A)-1;i>=0;i--) for(j=(B)-1;j>=0;j--)
#define loopi(A) for(i=0;i<A;i++)
#define loopj(A) for(j=0;j<A;j++)
#define loopk(A) for(k=0;k<A;k++)
#define loopl(A) for(l=0;l<A;l++)
#define loopm(A) for(m=0;m<A;m++)
#define loopn(A) for(n=0;n<A;n++)
#define loopp(A) for(p=0;p<A;p++)
#define loopij(A,B) for(i=0;i<A;i++) for(j=0;j<B;j++)
#define loopkl(A,B) for(k=0;k<A;k++) for(l=0;l<B;l++)
#define MALLOC(A,B,C,D) if((A=(C *)malloc((B)*sizeof(C)))==NULL) { \
  fprintf(stderr, strcat(D,": insufficient memory\n") ); \
  exit(-1); }
#define CREATE_FILE(A,B,C) if((A=fopen(B,"w")) == NULL) { \
          printf(strcat(C,": can't open for writing - %s.\n"),B); \
  exit (-1); }
#define OPEN_FILE(A,B,C) if((A=fopen(B,"r")) == NULL) { \
          printf(strcat(C,": can't open for reading - %s.\n"),B); \
  exit (-1); }
#define idx(I,J,N) (I)*(N)+(J)

/** All of these are dependent on the definition of "layer" **/
#define MAX_INPUTS                50
#define MAX_NODES                 50
#define MAX_H1_NODES              50
#define MAX_H2_NODES              50
#define MAX_OUTPUTS               50

#define MAX_VECTORS             1000

#define WTS_TYPE_MSF 2 /* new weights file */
#define WTS_TYPE_1   1 /* new weights file */
#define WTS_TYPE_0   0 /* old weights file */

#define TRAIN     0
#define TEST      1

#define THREE_LAYER    3
#define TWO_LAYER      2
```

C.2.3 Listing of STEWMATH.C

```
/*   This is a collection of functions for Convenience */

/***********
  LOG2 takes the log base two of an integer and returns an integer.
***********/

int  LOG2(x)
   int  x;
{
int y = 0;
   while (x/2 > 0){
      x /= 2;
      y++;
      }
```

```
      return y;
}

/* The following is not used in WAVE */
void flipo(invectorpointer,outvectorpointer)
   float_vector *invectorpointer, *outvectorpointer;
{
int  i;
int  map;
outvectorpointer->length = invectorpointer->length;
outvectorpointer->vector[1] = invectorpointer->vector[1];
map = invectorpointer->length - 2;
loopi(invectorpointer->length - 1){
   outvectorpointer->vector[i+2] = invectorpointer->vector[i+2+map];
   map -= 2;
   }
}
void flipc(invectorpointer,outvectorpointer)
   float_vector *invectorpointer, *outvectorpointer;
{
int i;
loopi(invectorpointer->length/2 + 1)
   outvectorpointer->vector[invectorpointer->length/2 +1 - i] =
   invectorpointer->vector[i+1];
outvectorpointer->length = invectorpointer->length;
}
```

C.2.3.1 Listing of ASIFT.C

```
/*****************************************************************/
/*****************************************************************/
/***            FLOAT TO INTEGER CLIP AND SIFT PROGRAM           **/
/*****************************************************************/
/*****************************************************************/
/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  asift.c
    DESCRIPTION:  This program converts the numbers from an input file in which
    each number is on a separate line from float to integer. This process also
    takes the absolute value and clips the values to stay between a minimum
    value of 0 and a maximum value of 255.
    FILES READ: One file specified by the user.
    FILES WRITTEN:  One file specified by the user.
    HEADERS USED:   <stdio.h>,  "jsmacros.h" , "macros.h", "stewmath.h",
                    <math.h>
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:   NONE
    AUTHOR: J. Stewart Laing and Steve Smiley
    HISTORY: Initial Version
*/
/*****************************************************************/
/*****************************************************************/

/************************/
/*  DECLARATION SECTION  */
/************************/

#include <stdio.h>
#include "macros.h"
#include "jsmacros.h"
```

```
#include "stewmath.h"
#include <math.h>

/************************/
/*  MAIN PROGRAM BODY  */
/************************/

void main(argc, argv)
   int argc;
   char *argv[];
{

/************************/
/* initialize variables */
/************************/

float_array basis, coef, proj, temp;
int i, j, k, l, level, size, shift=1, newi, newj,newint;
char basisfile[64], coeffile[64];
FILE *infile1, *infile2, *outfile;

/************************/
/* test parameters     */
/************************/

if(argc != 4 && argc != 1){
   printf("Usage: threshold <filename> <# of rows> <# of Cols>\n");
   exit(0);
}

/*****************************/
/*      PROMPT USER          */
/*****************************/

if(argc == 1){

printf("\n Enter the name of the coefficient file>>");
scanf("%s", coeffile);
printf("\n Enter the size of the NXN coefficen  array>>");
scanf("%d", &coef.ROW);
}

else{
    sprintf(coeffile, "%s", argv[1]);
    sscanf(argv[2], "%d", &coef.ROW);
    sscanf(argv[3], "%d", &coef.COL);
 }

/*************************************/
/*  create a matrix to hold the image */
/*************************************/

coef.COL = coef.ROW;
CREATE_MATRIX_ROW(coef.array, coef.ROW, float);
CREATE_MATRIX_COL(coef.array, coef.ROW, coef.COL, float);

/************************/
/*  open input file     */
/************************/

 OPEN_FILE (infile1, coeffile, "The projection program");
   loopij(coef.ROW, coef.COL)
         fscanf(infile1,"%f", &coef.array[i][j]),
  CLOSE_FILE (i, coeffile, "The projection program ", infile1)
  printf("\n ** The image %s has been loaded for processing. **\n\n\n",
         coeffile);

/*********************************/
/*     OUTPUT PROJECTION         */
/*********************************/
```

```
CREATE_FILE(outfile, "sifted", "The Projection Program")
loopij(coef.ROW, coef.COL){

       newint = abs((int)(coef.array[i][j]));
       if (newint > 255) newint = 255;
       if (newint < 0)  newint = 0;
       fprintf(outfile,"%d\n", newint);
     }
  printf("The projection file has been completed\n");
}
```

C.2.4 Listing of DAUB.C

```
/*****************************************************************************/
/*****************************************************************************/
/***              WAVELET GENERATOR PROGRAM                                **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  daub.c
    DESCRIPTION:  This program generates the g(n), phi(x), and psi(x) from
    a given h(n). The values of the h(n) are hard coded and must be set
    before comilation. Depth of recursion and type of wavelet are chosen
    by the user interactively.
    FILES READ: NONE
    FILES WRITTEN:  one file each for g(n), phi(x), and psi(x)
    HEADERS USED:   <stdio.h>,  "jsmacros.h" , "macros.h"
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:   NONE
    AUTHOR: J. Stewart Laing   and Steve Smiley
    HISTORY: Initial Version
*/
/*****************************************************************************/
/*****************************************************************************/

#include <stdio.h>
#include "macros.h"
#include "jsmacros.h"

float H(N,n)
   int N,n;
{
if(N == 2){
   if(n == 0) return .4829629131;
   if(n == 1) return .8365163037;
   if(n == 2) return .2241438680;
   if(n == 3) return -.1294095226;
   else return 0.0;
   }
if(N == 3){
   if(n == 0) return .3326705530;
   if(n == 1) return .8068915093;
   if(n == 2) return .4598775021;
   if(n == 3) return -.1350110200;
   if(n == 4) return -.0854412739;
   if(n == 5) return .0352262919;
   else return 0.0;
   }
if(N == 4){
   if(n == 0) return .2303778133;
```

```
    if(n == 1) return .7148465706;
    if(n == 2) return .6308807679;
    if(n == 3) return -.0279837694;
    if(n == 4) return -.1870348117;
    if(n == 5) return -.0308413818;
    if(n == 6) return .0328830117;
    if(n == 7) return -.0105974018;
    else return 0.0;
    }
  if(N == 5){
    if(n == 0) return .1601023980;
    if(n == 1) return .6038292698;
    if(n == 2) return .7243085284;
    if(n == 3) return .1384281459;
    if(n == 4) return -.2422948871;
    if(n == 5) return -.0322448696;
    if(n == 6) return .0775714938;
    if(n == 7) return -.0062414902;
    if(n == 8) return -.0125807520;
    if(n == 9) return .0033357253;
    else return 0.0;
    }
  if(N == 6) {
    if(n == 0) return .115407434;
    if(n == 1) return .4946238904;
    if(n == 2) return .7511339080;
    if(n == 3) return .3152503517;
    if(n == 4) return -.2262646940;
    if(n == 5) return -.1297668676;
    if(n == 6) return .0975016056;
    if(n == 7) return .0275228655;
    if(n == 8) return -.0315820393;
    if(n == 9) return .0005538422;
    if(n == 10) return .0047772575;
    if(n == 11) return -.0010773011;
    else return 0.0;
    }
  if(N == 7) {
    if(n == 0) return .0778520541;
    if(n == 1) return .3965393194;
    if(n == 2) return .7291320908;
    if(n == 3) return .4697822874;
    if(n == 4) return -.1439060039;
    if(n == 5) return -.2240361850;
    if(n == 6) return .0713092193;
    if(n == 7) return .0806126092;
    if(n == 8) return -.0380299369;
    if(n == 9) return -.0[illegible]5745416;
    if(n == 10) return .[illegible]09986;
    if(n == 11) return .0[illegible]295780;
    if(n == 12) return -.0018016407;
    if(n == 13) return .0003537138;
    else return 0.0;
    }
  if(N == 8) {
    if(n == 0) return .0544156422;
    if(n == 1) return .3128715909;
    if(n == 2) return .6756307363;
    if(n == 3) return .5863546837;
    if(n == 4) return -.0158291053;
    if(n == 5) return -.2840155430;
    if(n == 6) return .0004724856;
    if(n == 7) return .1287474266;
    if(n == 8) return -.0173693010;
    if(n == 9) return -.0440882539;
    if(n == 10) return .0139810279;
```

```
   if(n == 11) return .0087460940;
   if(n == 12) return -.0048703530;
   if(n == 13) return -.0003917404;
   if(n == 14) return .0006754494;
   if(n == 15) return -.0001174768;
   else return 0.0;
   }
if(N == 9) {
   if(n == 0) return .0380779474;
   if(n == 1) return .2438346746;
   if(n == 2) return .6048231237;
   if(n == 3) return .6572880781;
   if(n == 4) return .1331973858;
   if(n == 5) return -.2932737833;
   if(n == 6) return -.0968407832;
   if(n == 7) return .1485407493;
   if(n == 8) return .0307256815;
   if(n == 9) return -.0676328291;
   if(n == 10) return .0002509471;
   if(n == 11) return .0223616621;
   if(n == 12) return -.0047232048;
   if(n == 13) return -.0042815037;
   if(n == 14) return .0010476469;
   if(n == 15) return .0002303858;
   if(n == 16) return -.0002519632;
   if(n == 17) return .0000393473;
   else return 0.0;
   }
if(N == 10) {
   if(n == 0) return .0266700579;
   if(n == 1) return .1881768001;
   if(n == 2) return .5272011889;
   if(n == 3) return .6884590395;
   if(n == 4) return .2811723437;
   if(n == 5) return -.2498464243;
   if(n == 6) return -.1959462744;
   if(n == 7) return .1273693403;
   if(n == 8) return .0930573646;
   if(n == 9) return -.0713941472;
   if(n == 10) return -.0294575368;
   if(n == 11) return .0332126741;
   if(n == 12) return .0036065536;
   if(n == 13) return -.0107331755;
   if(n == 14) return .0013953517;
   if(n == 15) return .0019924053;
   if(n == 16) return -.0006858567;
   if(n == 17) return -.0001164669;
   if(n == 18) return .0000935887;
   if(n == 19) return -.0000132642;
   else return 0.0;
   }

else {
   printf("\nError: Invalid choice of N");fflush(stdout);
   return 0.0;
   }
}

float G(N,n)
   int N,n;
{
int i,sign=1;
for(i=1;i<=abs(1-n);i++) sign *= -1;
return (sign*H(N,1-n));
}

float new(N,l,x)
```

```
   int N,l,x;
{
int n;
float temp = 0.0;
if (l <= 0){
   if (x == 0) return 1.0;
   else return 0.0;
   }
else {
   for (n=0;n<=2*N-1;++n) temp += H(N,n) * new(N, l-1, 2*x-n);
   return (1.414212562*temp);
   }
}
void main()
{
int i,l,N,j;
float temp,temp_sum=0.0;
FILE *outfile;
char filename[64];
printf("\nInput N corresponding to the desired Daubeshies");
printf(" Wavelet: ");
scanf("%d", &N);
printf("\nInput depth of recursion l = ");
scanf("%d", &l);
printf("\nWorking...");
sprintf(filename,"daub%d.phi", N);
CREATE_FILE(outfile, filename, "The Daub routine")
for(i=0; i<=(2*N-1); ++i) fprintf(outfile, "%.9f\n",new(N,l,i));
CLOSE_FILE(i, filename, "The Daub routine", outfile);
sprintf(filename,"daub%d.h", N);
CREATE_FILE(outfile, filename, "The Daub routine")
for(i=0; i<=2*N-1; ++i) fprintf(outfile, "%.9f\n",H(N,i));
CLOSE_FILE(i, filename, "The Daub routine", outfile);
sprintf(filename,"daub%d.g", N);
CREATE_FILE(outfile, filename, "The Daub routine")
for(i=1; i>=2-2*N; --i) fprintf(outfile, "%.9f\n",G(N,i));
CLOSE_FILE(i, filename, "The Daub routine", outfile);
printf("\n");
sprintf(filename,"daub%d.psi", N);
CREATE_FILE(outfile, filename, "The Daub routine")
printf("psi interval of support is %d  %d\n",(1-((2*N)-1))/2,(1+((2*N)-1))/2);
for(j=(1-((2*N)-1))/2; j<=(1+((2*N)-1))/2; ++j){
      temp_sum =0.0;
      for(i=1; i>=2-(2*N); --i){
      temp_sum  +=  G(N,i)*new(N,l,((2*j)-i));
                }
  fprintf(outfile, "%.9f\n",1.414212562*temp_sum);
   }
CLOSE_FILE(i, filename, "The Daub routine", outfile);
printf("\n");
}
```

C.2.5 Listing of EPSVIEW.C

```
/*************************************************************************/
/*************************************************************************/
/***                  ROUTINE TO VIEW IMAGES FOR WAVELET ANALYZER      *****/
/*************************************************************************/
/*************************************************************************/
/*  DATE:  15 April 91                                                   */
```

```
/*                                                                          */
/*  VERSION: 1.0                                                            */
/*                                                                          */
/*  NAME: epsview.c                                                         */
/*                                                                          */
/*  DESCRIPTION:  This routine performs the inner product between the phi   */
/*  and phi coeficient of the image at any valid level as requested by      */
/*  the caller.                                                             */
/* It is intended as a subroutine for the WAVELET ANALYZER PROGRAM.         */
/*                                                                          */
/*  FILES READ: NONE.                                                       */
/*                                                                          */
/*  FILES WRITTEN:  A file will be generated each time the routine is       */
/*  routine is called. The name of the file will depend on the input        */
/*  mage filename, the type of wavelet used, and the level of resolution.  */
/*                                                                          */
/*  HEADERS USED:   <stdio.h>, "macros.h", <stdlib.h>, "jlmacros.h",        */
/*                  <string.h>                                              */
/*                                                                          */
/*  CALLING PROGRAMS:   main-wave.c                                         */
/*                                                                          */
/*  PROGRAMS CALLED:   NONE                                                 */
/*                                                                          */
/*  AUTHOR: Steve Smiley and J. Stewart Laing                               */
/*                                                                          */
/*  HISTORY: Initial Version                                                */
/*                                                                          */
/****************************************************************************/
/****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include    <stdlib.h>
#include    <stdio.h>
#include    "macros.h"
#include    "jsmacros.h"
#include    <string.h>
#include    <math.h>

/************************/
/*  FUNCTION BODY       */
/************************/

/*imageview(image)
    int-array    image;

{*/
int   i, j;
FILE *fopen(), *infile, *outfile;
char   infilename[64], viewfile[64], psfile[64];
int_array image;

void main(argc, argv)
   int argc;
   char *argv[];
{
 if(argc != 4 && argc != 1){
    printf("Usage: hist <filename> <# of rows> <# of Cols>\n");
    exit(0);
 }

if(argc == 1){
   printf("  \n\n Input filename of image to be viewed:>"); fflush(stdout);
   scanf("%s", infilename);
   printf("\n\n  Input the size of the image (ROW COLUMN):>");
   scanf("%d %d", &image.ROW, &image.COL);
 }
```

```
 else{
    sprintf(infilename, "%s", argv[1]);
    sscanf(argv[2], "%d", &image.ROW);
    sscanf(argv[3], "%d", &image.COL);
 }
   CREATE_MATRIX_ROW(image.array, image.ROW, int);
   CREATE_MATRIX_COL(image.array, image.ROW, image.COL, int);
   OPEN_FILE(infile, infilename, "epsview.c")
 loopij(image.ROW, image.COL){
            fscanf(infile, "%3u\n", &image.array[i][j]);
 }
sprintf(psfile, "%s.eps", infilename);
CREATE_FILE(outfile, psfile, "epsview.c")
   fprintf(outfile,"%%!PS-Adobe-2.0 EPSF-1.2\n");
   fprintf(outfile,"%%%%BoundingBox: 0 0 %d %d \n", image.ROW, image.COL);
   fprintf(outfile,"%%%%Creator: Imageview by Laing & Smiley\n");
   fprintf(outfile,"%%%%Title: %s.eps\n", infilename);
   fprintf(outfile,"%%%%EndComments\n");
   fprintf(outfile,"gsave\n");
   fprintf(outfile,"/picstr %d string def\n",image.ROW);
   fprintf(outfile,"0 0 translate\n");
   fprintf(outfile,"%d %d scale\n",image.ROW, image.COL);
   fprintf(outfile,"%d %d 8 [%d 0 0 %d 0 0]\n", image.ROW, image.COL,
 image.ROW, image.COL);
   fprintf(outfile,"{ currentfile picstr readhexstring pop}\n");
   fprintf(outfile,"image\n");
   loopij(image.ROW, image.COL){
       if(image.array[i][j] <= 15) fprintf(outfile, "0%x\n", image.array[i][j]);
       if(image.array[i][j] > 15)  fprintf(outfile,"%2x\n", image.array[i][j]);
        }

    fprintf(outfile,"showpage");
/*****************************************/
/*  call the showpage from unix          */
/*****************************************/
printf("\nI have created a postscript file called: %s\n\n", psfile);
fflush(stdout);
/*sprintf(viewfile, "csh -c pageview /tmp_mnt/home/scgraph/en/ge/ssmiley/thesis/C-code/Develop
/heximage.ps\n");
printf("%s", viewfile); fflush(stdout);
system(viewfile); */
}
```

C.2.6 Listing of MATRIXTOASCII.C

```
/*************************************************************************/
/*************************************************************************/
/***            KHOROS ASCII STRIPPER                                  **/
/*************************************************************************/
/*************************************************************************/
/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  matrixtoascii.c
    DESCRIPTION:  This program strips the matrix coordinates from an ASCII
    file output by the Khoros image processing system.
```

```
    FILES READ: One file specified by the user.
    FILES WRITTEN:  One file with the suffix .ascii added.
    HEADERS USED:   <stdio.h>,  "jsmacros.h", <stdlib.h>, <string.h>,
    <math.h>, "macros.h"
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:   NONE
    AUTHOR: Steve Smiley
    HISTORY: Initial Version
*/
/*************************************************************************/
/************   *********************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include    <stdlib.h>
#include    <stdio.h>
#include    "macros.h"
#include    "jsmacros.h"
#include    <string.h>
#include    <math.h>

main()
{

FILE  *infile, *outfile;
char   infilename[64], psfile[64], element[24], num[20];
int    i, j, holdi, hold2;
int_array image;


printf("  \n\n Input filename of image to be cleaned:>");
   scanf("%s", infilename);
   printf("\n\n  Input the size of the image (ROW COLUMN):>");
   scanf("%d %d", &image.ROW, &image.COL);
   CREATE_MATRIX_ROW(image.array, image.ROW, int);
   CREATE_MATRIX_COL(image.array, image.ROW, image.COL, int);

OPEN_FILE(infile, infilename, "matrixtoascii.c")
 while(*element != '#') fscanf(infile, "%c", element);
 loopij(image.ROW, image.COL){
    fscanf(infile, "%c", element);
    while(*element != '=') fscanf(infile, "%c", element);
       fscanf(infile, "%3d", &image.array[i][j]);
        }


sprintf(psfile, "%s.ascii", infilename);
CREATE_FILE(outfile, psfile, "matrix.c")
   loopij(image.ROW, image.COL){
       fprintf(outfile, "%d\n", image.array[i][j]);
        }
}
```

C.2.7 Listing of NRUTIL.C

```
#include <malloc.h>
#include <stdio.h>
void nrerror(error_text)
```

```
char error_text[];
{
void exit();
fprintf(stderr,"Numerical Recipes run-time error...\n");
fprintf(stderr,"%s\n",error_text);
fprintf(stderr,"...now exiting to system...\n");
exit(1);
}


float *vector(nl,nh)
int nl,nh;
{
float *v;
v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
if (!v) nrerror("allocation failure in vector()");
return v-nl;
}
int *ivector(nl,nh)
int nl,nh;
{
int *v;
v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
if (!v) nrerror("allocation failure in ivector()");
return v-nl;
}
double *dvector(nl,nh)
int nl,nh;
{
double *v;
v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
if (!v) nrerror("allocation failure in dvector()");
return v-nl;
}


float **matrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
int i;
float **m;
m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
if (!m) nrerror("allocation failure 1 in matrix()");
m -= nrl;
for(i=nrl;i<=nrh;i++) {
m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
if (!m[i]) nrerror("allocation failure 2 in matrix()");
m[i] -= ncl;
}
return m;
}
double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
int i;
double **m;
m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
if (!m) nrerror("allocation failure 1 in dmatrix()");
m -= nrl;
for(i=nrl;i<=nrh;i++) {
m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
```

```
if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
m[i] -= ncl;
}
return m;
}

int **imatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
int i,**m;

m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
if (!m) nrerror("allocation failure 1 in imatrix()");
m -= nrl;

for(i=nrl;i<=nrh;i++) {
m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
if (!m[i]) nrerror("allocation failure 2 in imatrix()");
m[i] -= ncl;
}
return m;
}


float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
int oldrl,oldrh,oldcl,oldch,newrl,newcl;
{
int i,j;
float **m;

m=(float **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(float*));
if (!m) nrerror("allocation failure in submatrix()");
m -= newrl;

for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

return m;
}


void free_vector(v,nl,nh)
float *v;
int nl,nh;
{
free((char*) (v+nl));
}

void free_ivector(v,nl,nh)
int *v,nl,nh;
{
free((char*) (v+nl));
}

void free_dvector(v,nl,nh)
double *v;
int nl,nh;
{
free((char*) (v+nl));
}


void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
int nrl,nrh,ncl,nch;
{
int i;

for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
free((char*) (m+nrl));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
```

```
double **m;
int nrl,nrh,ncl,nch;
{
int i;
for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
free((char*) (m+nrl));
}
void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
int nrl,nrh,ncl,nch;
{
int i;
for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
free((char*) (m+nrl));
}


void free_submatrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
free((char*) (b+nrl));
}


float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
int nrl,nrh,ncl,nch;
{
int i,j,nrow,ncol;
float **m;
nrow=nrh-nrl+1;
ncol=nch-ncl+1;
m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
if (!m) nrerror("allocation failure in convert_matrix()");
m -= nrl;
for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;
return m;
}


void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
int nrl,nrh,ncl,nch;
{
free((char*) (b+nrl));
}
```

C.2.8 Listing of THRESHOLD.C

```
/*****************************************************************************/
/*****************************************************************************/
/***              THRESHOLDER                                              ***/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  threshold.c
    DESCRIPTION:  This program thresholds an array of values. A window is
    chosen interactively by the user. All values inside the window are set
    to 255 (white) and all values outside the threshold are set to 0 (black).
    FILES READ: One file specified by the user.
    FILES WRITTEN:  One file with the suffix .thresh added.
```

```
    HEADERS USED:   <stdio.h>,  "jsmacros.h", <stdlib.h>, "macros.h"
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:   NONE
    AUTHOR: J. Stewart Laing   and  Steve Smiley
    HISTORY: Initial Version
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include    <stdlib.h>
#include    <stdio.h>
#include    "macros.h"
#include    "jsmacros.h"

/************************/
/*  FUNCTION BODY       */
/************************/

void main(argc, argv)
   int argc;
   char *argv[];
{
   /************************/
   /* initialize variables */
   /************************/

   int       i,j;
   FILE      *infile, *outfile;
   int_array image;
   int       upthresh, downthresh;
   char      infilename[64], threshfile[64];

 /************************/
/* test parameters      */
/************************/

if(argc != 4 && argc != 1){
   printf("Usage: threshold <filename> <# of rows> <# of Cols>\n");
   exit(0);
}

/**************************************/
/*  prompt for parameters if not input */
/**************************************/

if(argc == 1){
   printf("\n\n\n  Input the size of the image (ROW COLUMN):>");
   scanf("%d %d", &image.ROW, &image.COL);
   printf("  \n\n Input filename of image to be histogramed:>"); fflush(stdout);
   scanf("%s", infilename);
 }

/*****************************************/
/*  use parameters given on command line  */
/*****************************************/

 else {
    sprintf(infilename, "%s", argv[1]);
    sscanf(argv[2], "%d", &image.ROW);
    sscanf(argv[3], "%d", &image.COL);
 }

/**************************************/
/*  create a matrix to hold the image */
/**************************************/
```

```
 CREATE_MATRIX_ROW(image.array, image.ROW, int);
 CREATE_MATRIX_COL(image.array, image.ROW, image.COL, int);
/************************/
/*  open input file     */
/************************/
 OPEN_FILE (infile, infilename, "The thresholder")
/*******************************************************/
/*  prompt user for upper and lower threshold values */
/*******************************************************/
 printf(" \n\n Input upper threshold:>");
   scanf("%d", &upthresh);
 printf(" \n\n Input lower threshold:>");
   scanf("%d", &downthresh);
/*******************************************************/
/*  Create file to output the thresholded array for use.*/
/*******************************************************/
sprintf(threshfile, "%s.thresh", infilename);
CREATE_FILE(outfile, threshfile, "The Thresholder")
/*******************************************************/
/*  This part actually inputs the file, thresholds the  */
/*  grey scale values, and writes out either a 255 for  */
/*  white if it is between the up and down thresh values*/
/*  and a 0 if it is outside this window.               */
/*******************************************************/
   loopij(image.ROW, image.COL){
       fscanf(infile, "%d\n", &image.array[i][j]);
       if((image.array[i][j] >= downthresh) &&
          (image.array[i][j] <= upthresh)) image.array[i][j] = 255;
       else image.array[i][j] = 0;
       fprintf(outfile, "%d\n", image.array[i][j]);
     }
/*****************************************************/
/*  Tell the user where the output file is located.  */
/*****************************************************/
printf("\n Thresholded and binarized image created and saved in: %s\n\n", threshfile);
/* THE END */
}
```

C.2.9 Listing of RD884.C

```
/* Program to read ADTS 8-mm tapes on a SUN.
                * * * * * * Notice * * * * * *
This material was prepared as an account of work sponsored by the
United States Government.  Neither the United States, nor the
Department of Energy, nor the Department of Defense, nor any of
their employees, nor any of their contractors, subcontractors, or
their employees, makes any warrenty, expressed or implied, or
assumes any legal liability or responsibility for the accuracy,
completeness, or usefulness of any information, apparatus, product,
or process disclosed, or represents that its use would not infringe
upon privately owned rights.

author : Thomas D. Sullivan
 Sandia National Laboratories
 Division - 9133
 Albuquerque, NM  87185-5800
 May 10, 1991
```

```
compile : cc -o rd884 rd884.c
usage   : rd884 log_file < adts_file > file_of_complex_floats

log_file : output file of formatted header1 and header2 from the
   adts file. Default for log_file is stderr.

adts_file : input file in 8-8-4 ATRWG format.

file_of_complex_floats : output file of calibrated complex
 floating point pairs.

Transfering file from tape to disk.

dd if=/dev/nrst0 of=adts_file ibs=C156

Positioning tape to beginning of file 6.

mt -f /dev/nrst0 asf 5

*/
#include <stdio.h>
#include <string.h>
char *strstr();

main(argc,argv)
int argc;
char *argv[];
{
FILE *fp;
int ic,il,ip,irec=0,line,pol,frame,offset;
char *hdr2,*phdr2,line_header[9],line_trailer[3],c,e,i,q;
float *cbuf,cal_factor;
/* Declaration of variables in header 1 */
char country[5];
char lab[5];
char date[9];
char data_name[9];
long nrecords;
long header2_len;
long reclen;
long pixent;
long ibytes;
long mbytes;
long ebytes;
long nsamps;
long nlines;
long data_value_type;
long aux_data_type;
long header2_fmt;
long lines_per_rec;

/* get log_file name */
  if(argc > 1) {
     if((fp = fopen(argv[1],"w")) == NULL ) {
fprintf(stderr,"Cannot open output log file %s\n",argv[1]);
exit(1);
     }
  }
  else
     fp = stderr;

  for(;;) {
/* Position to start of next header 1 */
  fseek(stdin,irec*reclen,0) ;
/* Read and echo header1 */
  if ( fgets(country,5,stdin) == NULL ) break;
  fgets(lab,5,stdin);
  fgets(date,9,stdin);
  fgets(data_name,9,stdin);
  scanf("%d%d%d%d%d%d%d%d%d%d%d%d%d",&nrecords,&header2_len,
    &reclen,&pixent,&ibytes,&mbytes,&ebytes,&nsamps,&nlines,
```

```
    &data_value_type,&aux_data_type,&header2_fmt,&lines_per_rec);
  fprintf(fp,"       HEADER-1 CONTENTS\n\n");
  fprintf(fp,"%5s%5s : country/laboratory of origin\n",country,lab);
  fprintf(fp,"%10s : date recorded (YY MM DD)\n",date);
  fprintf(fp,"%10s : data name\n",data_name);
  fprintf(fp,"%10d : records in this file\n",nrecords);
  fprintf(fp,"%10d : bytes in header two\n",header2_len);
  fprintf(fp,"%10d : bytes per record\n",reclen);
  fprintf(fp,"%10d : entries per image sample\n",pixent);
  fprintf(fp,"%10d : bytes per integer entry\n",ibytes);
  fprintf(fp,"%10d : bytes per mantissa of an entry\n",mbytes);
  fprintf(fp,"%10d : bytes per exponent of an entry\n",ebytes);
  fprintf(fp,"%10d : image samples per image line\n",nsamps);
  fprintf(fp,"%10d : image lines per image\n",nlines);
  fprintf(fp,"%10d : imagery data type\n",data_value_type);
  fprintf(fp,"%10d : auxiliary data type\n",aux_data_type);
  fprintf(fp,"%10d : format of header two\n",header2_fmt);
  fprintf(fp,"%10d : image line(s) per tape record\n",lines_per_rec);
  fprintf(fp,"\n");

  if ( header2_len != 0 ) {
/* Position to start of header2. Read and echo */
  fseek(stdin,(irec+1)*reclen,0);
/* Allocate space for header2 */
  if((hdr2 = (char *) malloc(header2_len)) == NULL ) {
fprintf(stderr,"Out of memory for header 2.\n");
exit(1);
  }
  fread(hdr2,sizeof(char),header2_len,stdin);
  for(il = 0 ; il < header2_len ; il++) {
    c = hdr2[il];
    if(c == '\r') c = '\n';
    fputc(c,fp);
  }
  fprintf(fp,"\n");

/* scan header2 for calibration factor */
  if((phdr2 = strstr(hdr2,"SIGMA_0_TO_RCS      ")) == NULL ) {
fprintf("Problems with header 2 cal_factor.\n");
exit(1);
  }
  phdr2 += 34;
  sscanf(phdr2,"%f",&cal_factor);
/* Return allocated space for header 2. No longer needed. */
  free(hdr2);

/* Allocate space for floating_point complex array. */
  if((cbuf = (float *) malloc(2*sizeof(float)*nsamps)) == NULL ) {
fprintf(stderr,"Out of memory for complex array.\n");
exit(1);
  }

  offset = nrecords - nlines ; /* start of data */

#ifdef PRT
  fprintf(fp,"Offset = %d\n",offset);
  fprintf(fp,"Calfac = %f\n",cal_factor);
#endif
  cal_factor /= 4096.0;
  } /* if (header2_len */

  for(il = 0 ; il < nlines ; il++) {
   fseek(stdin,(il+offset+irec)*reclen,0);
/* Read and unpack line header */
  fread(line_header,sizeof(char),9,stdin) ;
  line = ((int)(line_header[1] & 037) << 8) + (line_header[2] & 0377);
```

```
  pol  = (line_header[1] & 0140) >> 5;
  frame = line_header[2] & 0377;
#ifdef PRT
  for(ic=0;ic<9;ic++) fprintf(fp,"%2.2x ",line_header[ic]);
  fprintf(fp,"\n");
  fprintf(fp,"line %4d, pol %4d, frame %4d\n",line,pol,frame);
#endif
/* Read data and convert to complex */
  for(ip = 0 ; ip < nsamps ; ip++) {
   e=getc(stdin);
   i=getc(stdin);
   q=getc(stdin);
   cbuf[2*ip]   = (float)((long)i<<(e&017))*cal_factor;
   cbuf[2*ip+1] = (float)((long)q<<(e&017))*cal_factor;
  }
 fwrite(cbuf,sizeof(float),2*nsamps,stdout);
 }
 iisc += nrecords ;
 } /* for(;;)  */
 exit(0);
}

char *strstr(str1,str2)
char *str1,*str2;
{
 int i,str1len,str2len;

 str1len=strlen(str1);
 str2len=strlen(str2);

 for(i=0 ; i<str1len-str2len ; i++)
    if (strncmp(str1+i,str2,str2len)==0)
       return( str1+i );

 return(NULL);
}
```

C.2.10 Listing of LOGB.C

```
/* Reads a raw floating point complex data file. Outputs a
log magnitude byte image.

                * * * * * * Notice * * * * * *

This material was prepared as an account of work sponsored by the
United States Government.  Neither the United States, nor the
Department of Energy, nor the Department of Defense, nor any of
their employees, nor any of their contractors, subcontractors, or
their employees, makes any warrenty, expressed or implied, or
assumes any legal liability or responsibility for the accuracy,
completeness, or usefulness of any information, apparatus, product,
or process disclosed, or represents that its use would not infringe
upon privately owned rights.

        author : Thomas D. Sullivan
                 Sandia National Laboratories
                 Division - 9133
                 Albuquerque, NM  87185-5800
                 May 3, 1991


  Sets PMAX dBsm at 255 and 0 at PRNG dB below PMAX
compile: cc -o logb logb.c -lm
usage: logb < in_file > out_file
*/
#include <stdio.h>
#include <math.h>
```

```
#define PMAX 30
#define PRNG 60

main(argc,argv)
int argc;
char *argv[];
{
float scale,offst,tmp,in[1024];
int actual,i;
unsigned char out[512];

scale = 2550.0/PRNG ;
offst = scale *(PRNG-PMAX)/10.0 ;

while(actual = fread(in,sizeof(float),1024,stdin)) {
for(i=0;i<actual/2;i++) {
 tmp = in[2*i]*in[2*i] + in[2*i+1]*in[2*i+1];
 if(tmp > 0.0) {
 tmp = scale * log10(tmp) + offst;
 if (tmp > 255.0)
     tmp = 255.0;
 else if (tmp < 0.0)
     tmp = 0.0;
 }
    out[i] = tmp;
}
fwrite(out,1,actual/2,stdout);
}
}
```

C.2.11 Listing of EXTRACT.C

```
/***************************************************************************/
/***************************************************************************/
/***                     EXTRACT BLOCKS PROGRAM                          **/
/***************************************************************************/
/***************************************************************************/
/*  DATE:  12 Sept 91
    VERSION: 1.0
    NAME:  extract.c
    DESCRIPTION: This program generates data vectors from multiresolution
    decomposition coefficient files for use in Dan Zahirniak's Radial
    Basis Function neural network.
    FILES READ: Detail signal files of various levels and a lower level
    multiresolution approximation file. A vector file containing locations
    on the original image of desired feature extraction locations.
    FILES WRITTEN:  A file called block.# will be generated.
    HEADERS USED:   <stdio.h>, "macros.h", "jsmacros.h", "stswmath.h"
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED: normalize.c
    AUTHOR: Steve Smiley
    HISTORY: Initial Version
*/
/***************************************************************************/
/***************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include <stdio.h>
#include "macros.h"
```

```
#include "jsmacros.h"
#include "stewmath.h"
#include "nrutil.h"

void    normalizeA();


/**********************/
/*  MAIN PROGRAM BODY  */
/**********************/

main()
{

/**********************/
/* declare variables  */
/**********************/

float_vector   coefficient, temp;
float_array    temps,holder;
int           **coordinates;
int           i, j, k, l,m, osample_size,nsample_size, maxlevel, ROW_position;
int           COL_position,aROW,aCOL,number_of_coordinates;
int           oROW, oCOL,nROW,nCOL, oimage_size,nimage_size;
int           level,image_size, class,sizer=1,dsize,counter;
float         biggest;
char          infilename[64], infilename2[64];
char          done='n';
char          filename[64],tempname[64],wave_type[10];
char          image_name[34], output[64],coef_type[5],outputfile[64];
FILE          *outfile, *infile, *infile2;

coordinates = imatrix(0, 200, 0, 2);

/**************************/
/*  User Input Parameters  */
/**************************/

 printf("\n\n Input the name of the file of the decomposed \nimage:");
      scanf("%s", infilename);

    loopi(64)
      tempname[i] = infilename[i];

    loopi(64)
      if (infilename[i] == '.') tempname[i] = ' ';

     sscanf(tempname, "%s %d %d %s %s", image_name,
&oimage_size, &level,coef_type, wave_type);

 printf("\n\n Input the number of sample rows in the area of interest");
      printf("\n(The number must a power of 2):");
      scanf("%d", &osample_size);

   /* Read in vectors of feature directions  */

   printf("\nInput the name of the vector file:");
   scanf("%s", infilename2);

    OPEN_FILE(infile2, infilename2, "The Center Program")
       fscanf(infile2, "%d", &number_of_coordinates);

       loopij(number_of_coordinates, 2)
           fscanf(infile2, "%d", &coordinates[i][j]);

      CLOSE_FILE(i, filename, "The Center Program", infile2)

 printf("\n\n Input the class number (integer):");
       scanf("%d", &class);

 printf("\nInput the name of the output file: ");
 scanf("%s",outputfile);
```

```
   sprintf(output, "%s.%d",outputfile,class);
   CREATE_FILE(outfile, output, "The Center Program")
/*************************************************************************/
/*  change sizes from original image to image sizes of level of interest  */
/*************************************************************************/
maxlevel = LOG2(oimage_size);
if (maxlevel > 5) maxlevel = 5;
loopi(level)
sizer *=2;
nsample_size = osample_size/sizer;
nimage_size = oimage_size/sizer;
/************************/
/* allocate memory      */
/************************/

temps.ROW = nsample_size;
temps.COL = nsample_size;
temps.array = matrix(1, temps.ROW, 1, temps.COL);
loop1ij(temps.ROW,temps.COL) temps.array[i][j] = 0.0;
holder.ROW = nimage_size;
holder.COL = nimage_size;
holder.array = matrix(1, holder.ROW, 1, holder.COL);
loop1ij(holder.ROW,holder.COL) holder.array[i][j] = 0.0;


/************************************************/
/* Read in sampled areas and store into arrays  */
/************************************************/

   OPEN_FILE(infile, infilename, "The Center Program")
     loop1ij(nimage_size, nimage_size)
          fscanf(infile, "%f", &holder.array[i][j]);

   CLOSE_FILE(i, filename, "The Center Program", infile)
   normalizeA(holder.array,holder.ROW,holder.COL);


/* begin data extraction */
loopm(number_of_coordinates){
  oROW = coordinates[m][0];
  oCOL = coordinates[m][1];
nROW = oROW/sizer; if(oROW%sizer != 0) nROW +=1;
nCOL = oCOL/sizer; if(oCOL%sizer != 0) nCOL +=1;
printf("%d %d %d %d\n", nROW,nCOL,nsample_size,nimage_size);
/* Cut put the vectors to a file */


   loopij(nsample_size,nsample_size){
     ROW_position = nROW + i;
     COL_position = nCOL + j;
     loopkl(5,5){
     aROW = k + ROW_position - 2;
     aCOL = l + COL_position - 2;
     if(aROW > nimage_size) aROW = nimage_size;
     if(aCOL > nimage_size) aCOL = nimage_size;
     if(aROW < 1) aROW = 1;
     if(aCOL < 1) aCOL = 1;
     fprintf(outfile, "%f  ", holder.array[aROW][aCOL]*3);
```

```
      if(k == 4 && l == 4)
        fprintf(outfile, "\n%d\n", class);
                          }
                    }
          }

}
```

C.2.19 Listing of NORMALIZEA.C

```
/*****************************************************************************/
/*****************************************************************************/
/***                      NORMALIZE AN ARRAY PROGRAM                       **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  15 Aug 91
    VERSION: 1.0
    NAME:  normalizeA.c
    DESCRIPTION: This program will normalize a set of data vectors used
    in the RBF network.
    FILES READ: Detail signal files of various levels and a lower level
    multiresolution approximation file.
    FILES WRITTEN:  A file called vector.# will be generated
    HEADERS USED:   <stdio.h>, "macros.h", "jsmacros.h", "stewmath.h"
    CALLING PROGRAMS:   vector or sampler.c
    PROGRAMS CALLED: NONE
    AUTHOR: Steve Smiley & Stewart Laing
    HISTORY: Initial Version
                                                                          */
/*****************************************************************************/
/*****************************************************************************/
/*  DECLARATION SECTION  */
/*************************/
#include <stdio.h>
#include "macros.h"
#include "jsmacros.h"
#include <math.h>

normalizeA(array_ptr,ROW,COL)
float **array_ptr;
int    ROW,COL;
{
/************************/
/* declare variables    */
/************************/
float biggest;
int   i,j;

/*****************************************/
/*  Find the largest value in the vector    */
/*****************************************/
biggest = 0.0;

loop1ij(ROW,COL)
    if(fabs((double)array_ptr[i][j]) > biggest) biggest
= fabs((double)array_ptr[i][j]);

/*****************************************/
/*   Normalize vector                    */
/*****************************************/

loop1ij(ROW,COL)
```

```
   array_ptr[i][j] = array_ptr[i][j]/biggest;
}
```

C.2.13 Listing of CENTER.C

```
/*****************************************************************************/
/*****************************************************************************/
/***                        CENTER BLOCKS PROGRAM                          **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  23 Aug 91
    VERSION: 1.0
    NAME:  center.c
    DESCRIPTION: This program generates data vectors from multiresolution
    decomposition coefficient files for use in Dan Zahirniak's Radial
    Basis Function neural network.  The training data is extracted
    from large blocks or areas of the image.
    FILES READ: Detail signal files of various levels and a lower level
    multiresolution approximation file.
    FILES WRITTEN:  A file called block.# will be generated.
    HEADERS USED:   <stdio.h>, "macros.h", "jsmacros.h", "stewmath.h"
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED: normalize.c
    AUTHOR: Steve Smiley
    HISTORY: Initial Version
*/
/*****************************************************************************/
/*****************************************************************************/

/*************************/
/*  DECLARATION SECTION  */
/*************************/

#include <stdio.h>
#include "macros.h"
#include "jsmacros.h"
#include "stewmath.h"
#include "nrutil.h"

void    normalizeA();


/************************/
/*  MAIN PROGRAM BODY  */
/************************/

main()
{

/************************/
/* declare variables    */
/************************/

float_vector   coefficient, temp;
float_array    temps,holder;
int           i, j, k, l, osample_size,nsample_size, maxlevel, ROW_position;
int           COL_position,aROW,aCOL;
int           oROW, oCOL,nROW,nCOL, oimage_size,nimage_size;
int           level,image_size, class,sizer=1,dsize,counter;
float         biggest;
char          infilename[64];
char          filename[64],tempname[64],wave_type[10];
char          image_name[34], output[64],coef_type[5],outputfile[64];
FILE          *outfile, *infile;
```

```
/*****************************/
/*  User Input Parameters  */
/*****************************/
 printf("\n\n Input the name of the file of the decomposed \nimage:");
      scanf("%s", infilename);
    loopi(64)
      tempname[i] = infilename[i];
    loopi(64)
      if (infilename[i] == '.') tempname[i] = ' ';
     sscanf(tempname, "%s %d %d %s %s", image_name, &oimage_size,
&level,  coef_type, wave_type);

 printf("\n\n Input the number of sample Rows in the area of interest");
      printf("\n(The number must a power of 2):");
      scanf("%d", &osample_size);
 printf("\n\n Input the pixel coordinates of the top left hand\n
corner of the region of interest in the original image (row col):");
         scanf("%d %d", &oROW, &oCOL);
printf("\n\n Input the class number (integer):");
       scanf("%d", &class);
/*****************************************************************************/
/*  change sizes from original image to image sizes of level of interest  */
/*****************************************************************************/
maxlevel = LOG2(oimage_size);
if (maxlevel > 5) maxlevel = 5;
loopi(level)
sizer *=2;
nROW = oROW/sizer; if(oROW%sizer != 0) nROW +=1;
nCOL = oCOL/sizer; if(oCOL%sizer != 0) nCOL +=1;
nsample_size = osample_size/sizer;
nimage_size = oimage_size/sizer;
printf("%d %d %d %d\n", nROW,nCOL,nsample_size,nimage_size);
/*************************/
/* allocate memory      */
/*************************/
holder.ROW = nimage_size;
holder.COL = nimage_size;
holder.array = matrix(1, holder.ROW, 1, holder.COL);
loop1ij(holder.ROW,holder.COL) holder.array[i][j] = 0.0;
temps.ROW = nsample_size;
temps.COL = nsample_size;
temps.array = matrix(1, temps.ROW, 1, temps.COL);
loop1ij(temps.ROW,temps.COL) temps.array[i][j] = 0.0;

/*****************************************/
/* Read in sampled areas and store into arrays  */
/*****************************************/
   OPEN_FILE(infile, infilename, "The Sampler Program")
     loop1ij(nimage_size, nimage_size)
          fscanf(infile, "%f", &holder.array[i][j]);

   CLOSE_FILE(i, filename, "The Wavelet Analyzer", infile)
   normalizeA(holder.array,holder.ROW,holder.COL);
printf("\nInput the name of the output file: ");
scanf("%s",outputfile);
```

```
   sprintf(output, "%s.%d",outputfile,class);
  /* sprintf(output, "trainblock.%d",class); */
   CREATE_FILE(outfile, output, "The Center Program")
   loopij(nsample_size nsample_size){
     ROW_position = nROW + i;
     COL_position = nCOL + j;
     loopkl(3,3){
     aROW = k + ROW_position - 1;
     aCOL = l + COL_position - 1;
     if(aROW > nimage_size) aROW = nimage_size;
     if(aCOL > nimage_size) aCOL = nimage_size;
     if(aROW < 1) aROW = 1;
     if(aCOL < 1) aCOL = 1;
     fprintf(outfile, "%f  ", holder.array[aROW][aCOL]*3);
     if(k == 2 && l == 2)
       fprintf(outfile, "\n%d\n", class);
                              }
                    }

}
```

Bibliography

1. Antonini, et al. "Image Coding Using Vector Quantization in the Wavelet Transform Domain." *Proceedings of IEEE International Conference in ASSP*. 2297–2300. 1990.
2. Ayer, Kevin W. *Gabor Transforms For Forward Looking Infrared Image Segmentation*. MS thesis, AFIT/GEO/ENG/89D-1, Air Force Institute of Technology, 1989.
3. Brickey, Joseph L. *Fractal Geometry Segmentation of High Resolution Polarimetric Synthetic Aperture Radar Data*. MS thesis, AFIT/GE/ENG/90D, School of Engineering, Air Force Institute of Technology, December 1990.
4. Burt, Peter J. and Edward H. Adelson. "The Laplacian Pyramid as a Compact Image Code," *IEEE Transactions on Communications*, *COM-31*(4):532–540 (April 1983).
5. Cohen, A. and J. M. Schlenker. "Compactly Supported Bidimensional Wavelet Bases with Hexagonal Symmetry." AT&T Bell Laboratories, Preprint, 1991.
6. Cohen, I. "Time-Frequency Distributions - A Review," *Proceedings of the IEEE* (July 1989).
7. Combes, J. and others. *Time-Frequency Methods and Phase Space* (2 Edition), 21–37. Berlin: Springer-Verlag, 1989.
8. Daubechies, Ingrid. "Orthonormal Bases of Compactly Supported Wavelets," *Communications on Pure and Applied Mathematics*, *41*:909–996 (1988).
9. Daubechies, Ingrid. "Orthonormal Bases of Wavelets with Finite Support - Connection with Discrete Filters." AT&T Bell Laboratories, Preprint, 1990.
10. Daugman, John G. "Uncertainty Relation for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two-Dimensional Visual Cortical Filters," *Journal of Optical Society of America*, 1160–1169 (July 1985).
11. Daugman, John G. "Complete Discrete 2-D Gabor Transforms by Neural Networks for Image Analysis and Compression," *IEEE Transactions on Acoustics Speech and Signal Processing*, *36*(7):1169–1179 (July 1988).
12. et al, Chris Conatser. *Digital IR Recording System*. Final Program Summary 361203, Texas Instruments, December 1982.
13. et al., William H. Press. *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge University Press, 1988.
14. Fastman, Inc. *The Wavelet Handbook*. Technical Report, Defence Advanced Research Projects Agency, 1991 (AD-B151 677).
15. Gonzalez, Rafael C. and Paul Wintz. *Digital Image Processing* (Second Edition). Addison-Wesley Publishing Company, 1987.

16. Heil, Christopher. "Wavelets and Frames," *Signal Processing, Part I: Signal Processing Theory*, 147–160 (1990).

17. Hubel, David H. "The Visual Cortex of The Brain," *Scientific American*, *209*(5):54–62 (November 1963).

18. Jones, et al. "The Two-Dimensional Spatial Structure of Simple Receptive Fields in the Striate Cortex of Cats," *Journal of Neurophysiology*, *58*:1187–1211 (December 1987).

19. Laing, John S. *Analysis of Visual Illusions Using Multiresolution Wavelet Decomposition Based Models*. MS thesis, AFIT/GE/ENG/91D-34, AFIT, December 1991.

20. L'Homme, Albert P. *Gabor Filters and Neural Networks for Segmentation of Synthetic Aperture Radar Imagery*. MS thesis, GE/ENG/90D-31, Air Force Institute of Technology, December 1990.

21. Lippmann, Richard P. "Pattern Classification Using Neural Networks," *IEEE Communications Magazine*, 47–63 (November 1989).

22. Mallat, Stephane G. "Multifrequency Channel Decompositions of Images And Wavelet Models," *ASSP*, *37*(12):2091–2109 (December 1989).

23. Mallat, Stephane G. "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *11*(7):674–693 (July 1989).

24. Mallat, Stephane G. "Zero-Crossings of a Wavelet Transform," *IEEE Transactions on Information Theory*, *37*(4):1019–1033 (July 1991).

25. Mallat, Stephane G. and Sifen Zhong. *Complete Signal Representation With Multiscale Edges*. Technical Report 483, Courant Institute, December 1989.

26. Moody, John and Christian Darken. "Learning with Localized Receptive Fields." *Proceedings of the 1988 Connectionist Models Summer School*. 133–143. 1988.

27. Nowlan, Steven J. *Max Likelihood Competition in RBF Networks*. Technical Report CRG-TR-90-2, Department of Computer Science, University of Toronto, 1990.

28. Poggio, Thomaso and F. Girosi. "Regularization Algorithms for Learning Equivalent to Multilayer Networks," *Science*, *247*:978–982 (February 1990).

29. Powell, M.J.D. and others. *Algorithms for Approximation*. Oxford: Clarendon, 1987.

30. Ranganath, Surendra. "Image Filtering Using Multiresolution Representations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *13*(5):426–440 (May 1991).

31. Rasure, John and Danielle Argiro. *Khoros User's Manual*. University of New Mexico, khoros@chama.unm.edu, May 1991.

32. Rogers, Steven K. and Mathew Kabrisky. *An Introduction to Biological Artifical Neural Networks for Pattern Recognition, TT4*. SPIE Optical Engineering Press, 1991.

33. Roth, Michael W. "Survey of Neural Network Technology for Automatic Target Recognition," *IEEE Transactions on Neural Networks*, *1*(1):28–43 (March 1990).

34. Shapiro, H. S. and others. "Uncertainty Principles for Basis in $L^2(R)$." Prometheus Inc., Preprint, 1991.

35. Tou, Julius T. and Rafael C. Gonzales. *Pattern Recognition Principles*. Massachusetts: Addison-Westly, 1974.

36. Zahirniak, Daniel, "Discussions with Dan Zahirniak." Personal Interview, June 1991.

37. Zahirniak, Daniel R. *Characterization of Radar Signals Using Neural Networks*. MS thesis, Air Force Institute of Technology, 1990.

Vita

Steven Smiley was born is San Jose, California on August 29, 1960. He obtained his Bachelors of Science Degree from Brigham Young University in 1986. His major was Electrical Engineering. He entered the United States Air Force on Jan 16, 1986. He obtained his commission through Officers Training School (OTS) at Lackland AFB on 31 March 1987. He Served from 16 April 1987 to 18 April 1990 at the 2nd Communication Squadron located at Buckley ANGB. He obtained his Masters of Science Degree from the Air Force Institute of Technology (AFIT) on 12 December 1991. He is currently serving at Electronic Systems Division (ESD) at Hanscom AFB Mass.

Permanent address: 2997 Holiday Dr. Ct.
Morgan Hill CA 95037

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1991	Master's Thesis

4. TITLE AND SUBTITLE
IMAGE SEGMENTATION USING AFFINE WAVELETS

5. FUNDING NUMBERS

6. AUTHOR(S)
Steven E. Smiley, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GE/~~ENS~~ENG/91D-50

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Model Based Vision Laboratory
Wright Laboratories
Wright Patterson AFB OH

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This thesis discusses the use of the multiresolution representation and Radial Basis Function (RBF) neural networks to segment both FLIR and SAR imagery. The multiresolution approximation coefficients are used as features into the RBF network which learns to distinguish between different cultural and natural regions or objects. The wavelets used are Mallat's spline wavelet and Daubechies' compactly supported wavelets. Additionally, this thesis provides an explanation of wavelets in a tutorial manner. It introduces wavelet theory and discusses two different approaches to generating the multiresolution or wavelet representation.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Wavelets, Multiresolution Analysis, Image Segmentation	208
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL