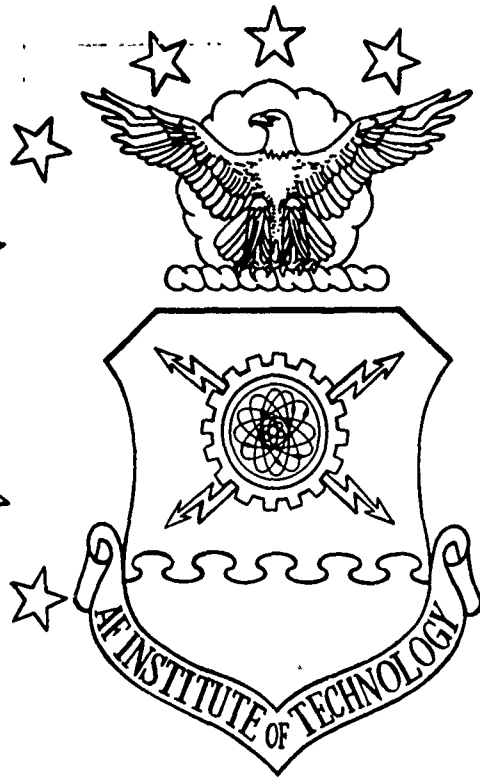


DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/91D-34

ANALYSIS OF VISUAL ILLUSIONS USING MULTIRESOLUTION WAVELET DECOMPOSITION BASED MODELS

THESIS

John S. Laing
Captain, USAF

AFIT/GE/ENG/91D-34

Approved for public release; distribution unlimited

AFIT/GE/ENG/91D-34

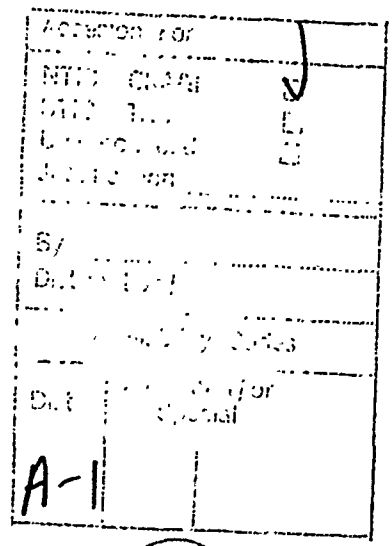
ANALYSIS OF VISUAL ILLUSIONS USING MULTIRESOLUTION WAVELET DECOMPOSITION BASED MODELS

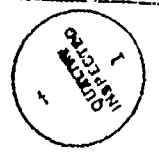
THESIS

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

John S. Laing, B.S.E.E., M.B.A.
Captain, USAF

December, 1991





Approved for public release; distribution unlimited

Acknowledgments

My thesis committee and others deserve thanks for their enthusiasm and technical assistance. First, I want to thank my thesis advisor, Maj Steve Rogers, for his guidance and uplifting support. Much thanks is due to Dr. Matthew Kabrisky, a committee member, for sharing his broad wealth of knowledge, experience, and wisdom in Biology and Engineering. As committee members, Capt Dennis Ruck and Dr. Mark Oxley, helped make this thesis the success I feel it is by providing technical reviews at each iteration of the document. Others that have taken a great deal of interest in our work and provided invaluable brainstorming stimulation are Capt Greg Warhola, Dr. Dennis Quinn, and Dr. Bruce Suter. These professors were instrumental in developing my understanding of the theory and application of Wavelet mathematics. Also deserving of thanks are Lt Col Phil Amburn, Capt John Brunderman, and Capt Greg Tarr for their unselfish assistance with the development of the graphics software that was necessary for demonstrating and researching the temporal effects of the analysis. A special thanks goes to my partner in developing the wavelet based application software that is the basis of this work, Capt Steve Smiley.

Finally, and most importantly, my deepest and most sincere thanks goes to my fiancée, *Leslie Bohler*, whose patience and gentle support has given me the confidence to pursue this degree successfully.

John S. Laing

Table of Contents

List of Figures

Abstract

This thesis provides alternatives to the explanation that spatial filtering is responsible for the perception of illusory contours in the Kanisza Triangle illusion. Specifically, we use a Multiresolution Wavelet Decomposition to divide an image into spatial-frequency bands that are used as inputs to three biologically motivated models. The thesis includes a brief tutorial of Wavelet theory and an in-depth explanation of our implementation of recently published algorithms for Multiresolution Wavelet Analysis. The first model is based on the saccadic movements of the human eye. It demonstrates the importance of the high spatial-frequency content of an image in the formulation of the illusion. The second model is based on the serial architecture of the data transmission channel between the retina and the visual cortex of the brain. It demonstrates the importance of low temporal-frequency characteristics of the build-up of the visual world model. The third model considers only the high spatial-frequency content of the image. It consists of lateral excitation networks that serve to simulate the local high spatial-frequency energy interactions that contribute to illusory contours.

ANALYSIS OF VISUAL ILLUSIONS USING MULTIRESOLUTION WAVELET DECOMPOSITION BASED MODELS

I. Introduction

1.1 Background

By today's computer standards, the individual processing elements of the human brain process information extremely slowly, yet its capabilities as a whole far outpace even the fastest supercomputers. The secret of its success has eluded researchers for decades. Somehow the brain manages to reduce vast amounts of environmental data, discarding unimportant details. This concept was best expressed by the poet William Blake when he wrote, "If the doors of perception were cleansed everything would appear to man as it is, infinite." Perception creates in the brain a model of the world using only a small part of the infinity of information contained in reality. But, sometimes the lack of a complete picture causes ambiguities which the brain perceives incorrectly. One such misperception is visual illusion. The hope is that by studying the brain's failures we can gain better insight into its function. To this end, we seek the best mathematical model for the visual system that explains illusions.

1.2 Problem

This paper proposes a thesis in which three models of the human visual system are based on a relatively new mathematical theory, Wavelets. The models are specifically designed to study spatial and spatial/temporal visual illusions. The thesis develops the algorithms and software necessary to decompose two dimensional images of visual illusions in

terms of wavelet bases. The thesis research includes experiments involving manipulations of the decomposed image based on current knowledge and conjecture of possible human visual system processing. Included in the evaluation of the resulting images is a comparison with previous work in this area.

1.3 Assumptions

The assumption on which this thesis depends is theoretical in nature. We assume that human cerebral processing includes some type of spatial frequency and spatial orientation selectivity. The choice of wavelet analysis as the method of decomposition is based on this assumption. In fact, the purpose of proposing a wavelet model of the visual system is to test this assumption through the evaluation of visual illusions with that model. The assumption of frequency and orientation selectivity in the brain is motivated by a deeper assumption that the observed behavior of the visual cortex of the cat and the monkey discussed in Chapter II is a good indication of the behavior of the human visual cortex.

1.4 Scope

This effort is limited to the following:

- A description of the mathematical theory of Wavelet Analysis.
- A description of the proposed visual models based on Wavelet Decomposition, modification of the decomposed illusions, and Wavelet Reconstruction.
- An analysis of the results produced by processing various static illusions with the model and comparison with previous work in the area of visual illusions.
- The software source code used to implement the models along with adequate documentation.

It is not the charter of this thesis to attempt to completely explain visual illusions. Such an explanation would require philosophical and psychological examination. This type of

examination is left to experts in those realms. The thesis is limited to a study of visual illusions based solely on engineering analysis.

1.5 Standards

All software written for this thesis is in the ANSI standard C programming language. All source code employs structured programming techniques such that the code may be easily modified and maintained by future research efforts. The code is compileable on any computer system that possesses an ANSI standard C compiler. We use standard image processing techniques to modify the decomposed images. A further explanation of these techniques may be found in an image processing text such as the one by Gonzalez [15].

1.6 Approach/Methodology

First, this thesis provides a written description of Wavelet Analysis. This description contains the information necessary for a reader with a background in Electrical Engineering to comprehend the mathematical basis for the proposed model. Next, research focuses on the development of the software that performs the first part of the processing required, that of Multiresolution Wavelet Decomposition of two dimensional images and one dimensional signals. This software is based on an algorithm developed by Mallat [28]. The bulk of the effort and the heart of the research lies in the development and use of the proposed models to discover the appropriate modifications of the decomposed image necessary to explain the illusion. This task is the original effort of the thesis and requires considerable experimentation. The evaluation criteria for these modifications is comparison with the original illusion. Finally, the thesis provides documentation of the results and a complete description of the models. It includes an evaluation of the results as evidence of the correctness of the models as possible engineering explanations of visual illusions.

1.7 Conclusion

The body of this thesis is logically divided to take the reader smoothly from background to supporting theory to application. Chapter II is a review of the works of Ginsburg and Oberndorf investigating low spatial-frequency contributions to visual perception. Chapters III and IV provide the theory of Wavelet Analysis and its application in Multiresolution Analysis respectively. These two chapters represent a collaborative effort with Steven Smiley. The same material can be found in the corresponding chapters of his masters thesis, "Image Segmentation Using Affine Wavelets" [41]. Next, Chapter V compares the low spatial-frequency approximation provided by the Multiresolution Wavelet Decomposition to the results of Ginsburg and Oberndorf [14, 32]. Chapters VI, VII, and VIII each present the methodology and results of the three models used to further analyze the Kanisza Triangle illusion in terms of spatial and temporal-frequency characteristics. Finally, Chapter IX summarizes the results and makes recommendations for future research.

II. Literature Review

2.1 Introduction

A model which shows promise in suggesting a type of processing that may occur in the brain is based on a relatively new mathematical theory, wavelets. Though others have mathematically explored illusions [19, 30, 25], this paper addresses two approaches which establish a relevant background for future research using wavelet analysis, Fourier and Gabor filtering. In two parts, it examines Ginsburg's research with Fourier analysis [14] and Oberndorf's research with Gabor analysis [32]. Each of these sections includes a brief discussion of the advances in the field of physiology which led these researchers in their choice of analysis.

2.2 Fourier Analysis

Ginsburg's use of Fourier filtering, grew out of a need to reduce the vast amount of data presented to the sensors of the human visual system. The eyes use over 110,000,000 rods and about 6,000,000 cones to see objects at 100 brightness levels and about 50 different colors [14:10]. He attempted to find the appropriate range of physical properties that would satisfactorily model reality. He chose filtering as a means of excluding what he considered the redundant details of a scene. Since this was an early attempt at explaining human perception in terms of a physical process, it was necessary to use a scientifically accepted and understood method of filtering. Fourier filtering provided such a tool.

The concept of a Fourier filter is quite simple. All the objects that make up an image can be characterized in terms of their spatial frequency. Larger objects have a low spatial-frequency while small objects and fine details have a high spatial-frequency. A Fourier filter can exclude any unwanted or unneeded range of spatial-frequencies in much the same way that we tune a radio or television to a specific channel or frequency. Filters can also alter the spatial-frequency content of images.

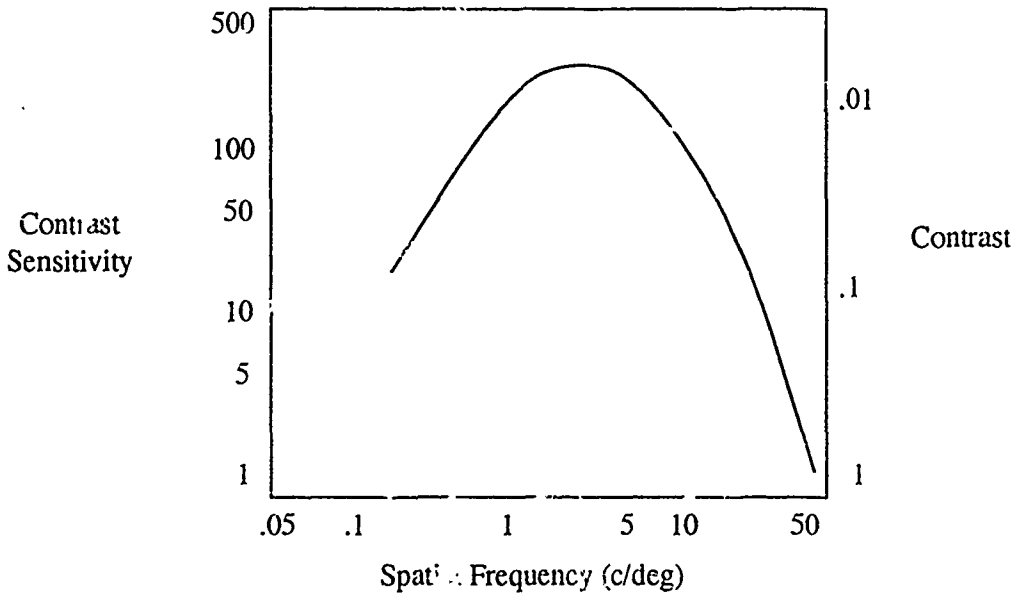


Figure 1. Typical Contrast Sensitivity Function [14:136]

To find which spatial-frequencies to exclude and which to keep, Ginsburg turned to known biological data about the contrast sensitivity of the visual system. As it turns out, the human eye has a range of spatial frequencies, bandwidth, for which it needs less contrast to discern objects than spatial frequencies outside of that range. Mostly, it is the low frequency characteristics, the form and shape of objects, that need the least illumination for discernment. To test this, turn down the contrast knob on a television and note that the finer details are the first to drop out of view. Figure 1 illustrates the characteristic shape of the contrast sensitivity curve as a function of spatial-frequency. This graph illustrates the range of spatial-frequencies to which the visual system is the most sensitive or which require the least contrast to discern.

Ginsburg fashioned a filter with a spatial frequency bandwidth that approximates the contrast sensitivity of the visual system (Figure 2). He reasoned that visual illusion is a consequence of filtering out certain high frequency details about objects at one or more stages of processing between the eyes and the visual cortex of the brain. The high frequency and low frequency information may then be transmitted independently to the perception

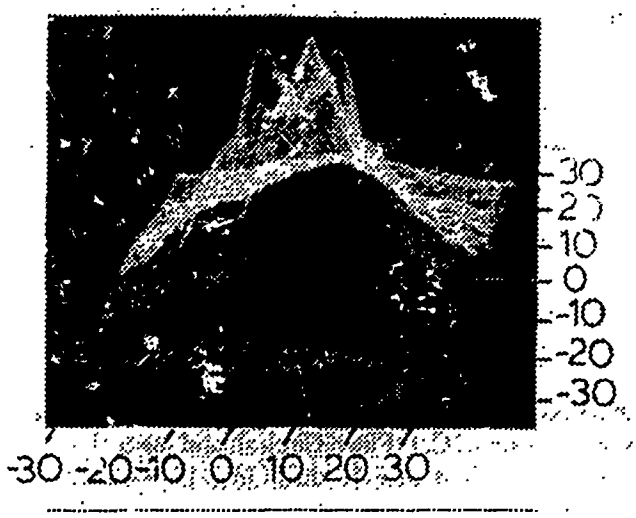
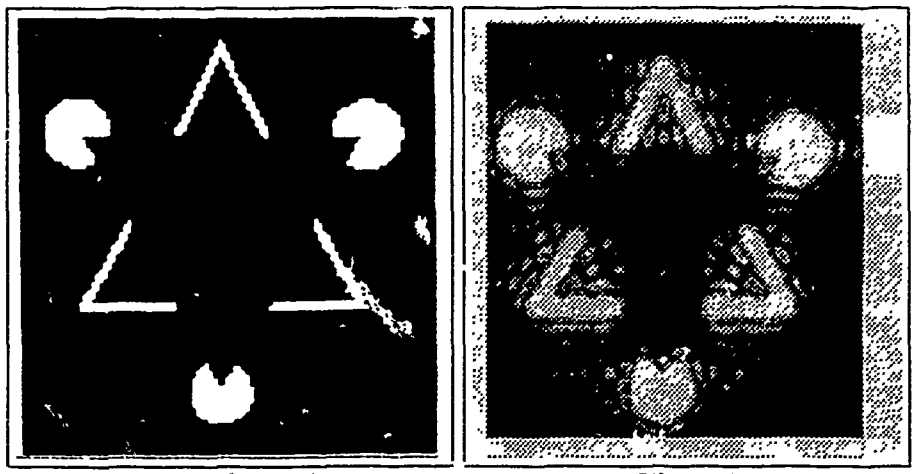


Figure 2. Ginsburg's 2D Low Pass Filter Based on the Contrast Sensitivity Function [14:141]

forming areas of the brain where they are recombined creating the awareness of a coherent scene. Ginsburg's thesis is that the visual system's form and shape recognizer, which uses the lower frequency information of an image, carries more weight in perception than the visual system's edge detector, which uses only high frequency information. Figure 3 [14:225] is a filtered version of the well known Kanizsa triangle next to the unfiltered original. The edges of the illusory triangle in the original image are even more strongly suggested in the filtered image. Also, the blurring effect in the filtered image is stronger outside of the illusory triangle matching the tendency to perceive the interior of the original triangle to be darker than the surrounding area.

While addressing only part of the misperception that leads to a visual illusion, Ginsburg's work lends credence to the view of the brain as an information processor that can be modeled mathematically. It is not clear, however, what other filtering or processing occurs to create the illusion as we see it. For example, how does the relative proximity and orientation of the objects in the image effect the illusion? By Ginsburg's own admission, further research is required [14:66].



Unfiltered Filtered

Figure 3. Results of Using Ginsburg's Contrast Sensitivity Based Filter [14:225]

2.3 Gabor Analysis

Ginsburg was not alone in his search for an explanation of visual illusion by means of filtering. In the same year that Ginsburg's dissertation was published, 1978, Ozawa independently duplicated Ginsburg's results [33]. The next break came in 1987 when new biologically measured data became available. Jones and Palmer gathered data from the visual cortex of a cat [24]. They demonstrated that the impulse response of the visual cortex, an early level of information processing done in the brain, closely resembles a two dimensional Gabor filter [11, 12]. Figure 4 shows the impulse response map of simples cells in the visual cortex of a cat as measured by Jones and Palmer, and the best-fitting two dimensional Gabor functions. The figure also shows how well the Gabor functions model the cells by demonstrating that the difference is nothing more than background noise.

The Jones and Palmer results have led to a host of image processing experiments which use Gabor filters to model the visual system. In 1990, Oberndorf, a masters student at the Air Force Institute of Technology, tested the Gabor theory on visual illusions [32]. Inspired by the notion that columnar groupings of cells in the visual cortex share a frequency response [24, 21, 23], he proposed a Gabor filter tuned to 1.32 octaves to model the processing effect

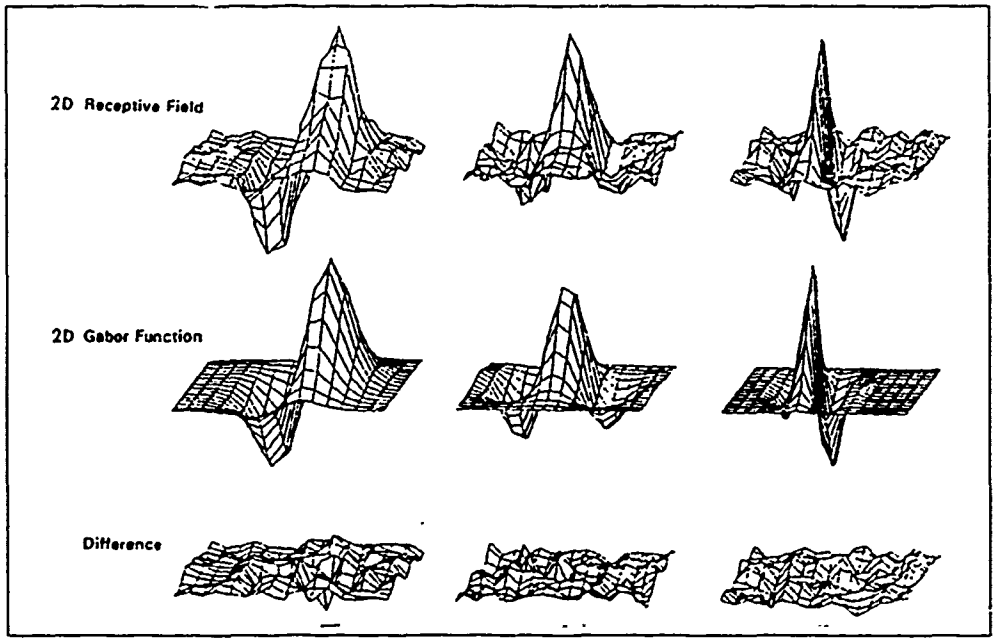


Figure 4. The Recptive Field Profile and the Gabor Function [9:1174]

of each grouping of simple cells [32:36]. When Oberndorf summed these filters, the total effect exhibited the characteristics of the contrast sensitivity data that inspired Ginsburg. Figure 5 is a plot of the low pass filter made from a summation of narrowly tuned Gabor filters. The small ripples in the top of the filter correspond to each of the peaks of the individual Gabor filters. The results of using this filter on the Kanizsa triangle is shown in Figure 6. The energy difference between the illusory triangle and its surrounding area is even more distinguishable than in Ginsburg's results (Figure 3). This improvement is a strong indication that the Gabor analysis is a step in the right direction.

The basis for this success lies in the ability of the Gabor analysis to discriminate a range of frequencies at a particular location. The Fourier filter, on the other hand, can only be applied to the entire scene. Therefore, if the desire is to isolate certain frequency characteristics by location, cell groupings, then the Gabor is a good choice. Going beyond Oberndorf's work, it might also be useful to isolate the local directional characteristics, orientation, in the image. Mallat shows that this is not only possible but desirable in many

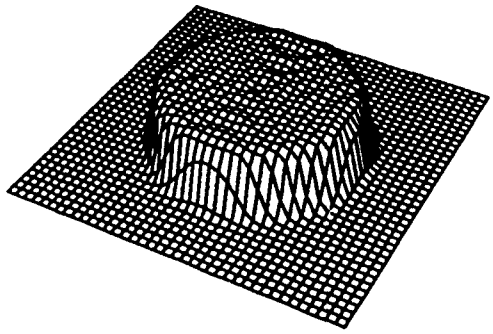


Figure 5. Low Pass Filter Created with Narrowly-Tuned Gabors [32:37]

Figure 6. Oberndorf's Result Using Gabor Filtering [32:37]

engineering applications [27]. He also shows that with wavelet analysis the Gabor filter can be made orientation as well as frequency selective. In wavelet analysis, the Gabor filter retains its location and orientation selectivity and gains a dilation parameter. This parameter is used to isolate the desired spatial-frequency characteristics in certain locations in the image. With this tool, it may be possible to isolate and emphasize the illusory contours of the image. So, the next logical step in research of visual illusions, started by Ginsburg and carried on by Oberndorf, is to analyze various illusions with wavelets.

2.4 Conclusion

A better understanding of visual illusions promises to provide insight into the processing taking place in the visual cortex of the brain. Ginsburg advanced this effort by applying Fourier filtering techniques to some visual illusions including the Kanizsa triangle. His results indicate that there may be some preattentive processing occuring in the visual system that separates the low frequency characteristics and enhances them with respect to the high frequency characteristics. Thus, the perception forming areas of the brain receive biased data. Oberndorf took this idea a step farther by applying the property of location sensitivity inherent in Gabor filtering. Each individual Gabor was made to emulate a columnar grouping of simple cells in the visual cortex as suggested by Jones and Palmer. Oberndorf's results indicate the approach is basically sound. The next logical step is to apply the Gabor filters in varying dilations and orientations. This can be done by extending the Gabor into a class of mathematical functions called wavelets. These functions are the filters that will determine the combination of frequencies and orientations that cause the illusion to appear. If the analysis is successful, it will form the basis of a mathematical model of the visual system.

III. Theory of Wavelet Analysis

This chapter was co-authored with Steven Smiley and exists in his thesis in duplicate [41].

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 $f(-\frac{T}{2}) = f(\frac{T}{2})$,

$$f(x) = \sum_n c_n e^{\frac{jn2\pi x}{T}} \tag{1}$$

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

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

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 2 becomes the Fourier Transform.

$$F(\xi) = \int_{-\infty}^{+\infty} f(x) e^{-j2\pi\xi x} dx \tag{3}$$

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 [9, 29].

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.

$$WF_f(\omega, \tau) = \int_{-\infty}^{+\infty} \mathrm{w}(t - \tau) e^{-j\omega t} f(t) dt \tag{4}$$

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 [8, 40]. 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 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.
- $\mathbf{L}^2(\mathbf{R})$ denotes the space of measurable, square integrable, one dimensional, real-valued functions $f(x)$, such that

$$\int_{-\infty}^{\infty} |(f(x))|^2 dx < \infty \tag{5}$$

[1] Another approach to the time/frequency resolution problem is that of Time-Frequency Distributions [10, 4]

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

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}|f(x,y)|^2dxdy < \infty \tag{6}$$

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

$$\langle f,g\rangle = \int_{-\infty}^{+\infty} g(x)f(x)dx \tag{7}$$

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

$$[f*g](x) = \int_{-\infty}^{+\infty} f(\alpha)g(x-\alpha)d\alpha \tag{8}$$

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

$$[f*g](x) = \int_{-\infty}^{+\infty} f(\alpha)g(\alpha-x)d\alpha \tag{9}$$

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

$$\mathbf{P}f = \sum_n \langle f,\phi_n\rangle\phi_n \tag{10}$$

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

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

translation parameter, $b \in \mathbf{R}$.

$$\psi_{ab}(x) = \frac{1}{\sqrt{a}}\psi(\frac{x-b}{a}) \tag{11}$$

The $\frac{1}{\sqrt{a}}$ term normalizes the energy of each basis function. Figure 7 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 $\{\psi_{ab}\}$.

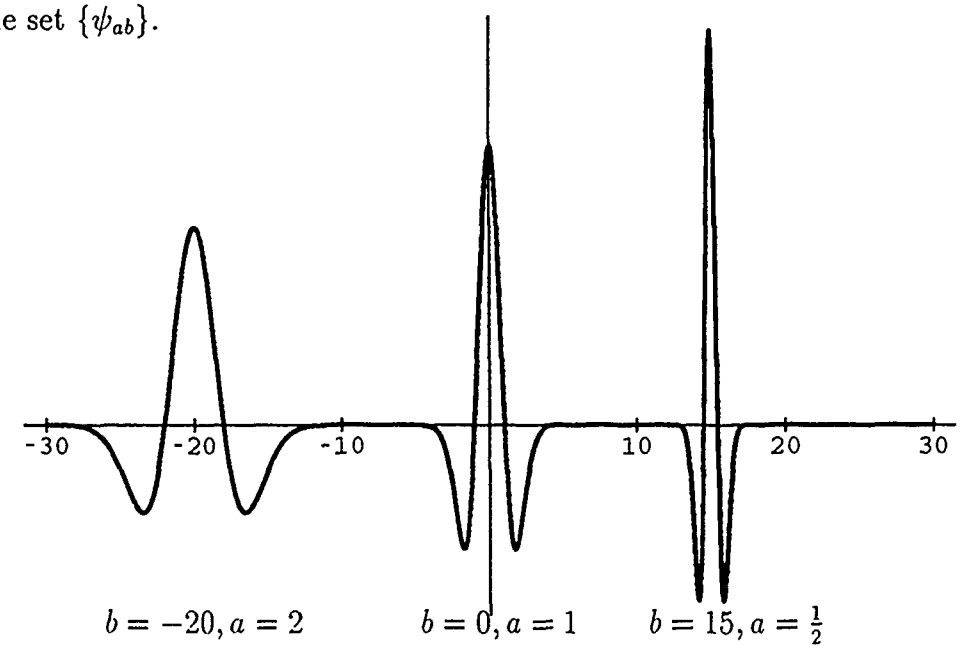


Figure 7. A Typical Mother Wavelet

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 [10:7]

$$W_f(a,b) = \frac{1}{\sqrt{a}}\int_{-\infty}^{+\infty} f(x)\psi(\frac{x-b}{a})dx, a \in \mathbf{R}^+, b \in \mathbf{R} \tag{12}$$

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

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

$$c = \int_0^{+\infty} |\Psi(\omega)|^2/|\omega| d\omega < \infty \tag{13}$$

which requires that $\Psi(0) = 0$ [4], an inversion transform exists and is given by [10:8]

$$f(x) = c^{-1} \frac{1}{\sqrt{a}} \int_{-\infty}^{+\infty} \int_0^{+\infty} \psi(\frac{x-b}{a}) W_f(a,b) \frac{dadb}{a^2} \tag{14}$$

The wavelet transform pair given in Equations 12 and 14 are analogous to the Fourier Transform pair of Equations 1 and 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 1 represents a signal with a set of discrete exponentials. The time/frequency plane evaluating grids are shown in Figure 8 for uniform time-frequency sampling associated with the Windowed Fourier Transform and the nonuniform sampling of the Wavelet Transform. Each dot in the 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 that the fixed window widths of the Windowed Fourier Transform have a fixed resolution in

[4] The ω in the denominator of Equation 13 requires that $\Psi(\omega)$ vanishes as ω approaches zero.

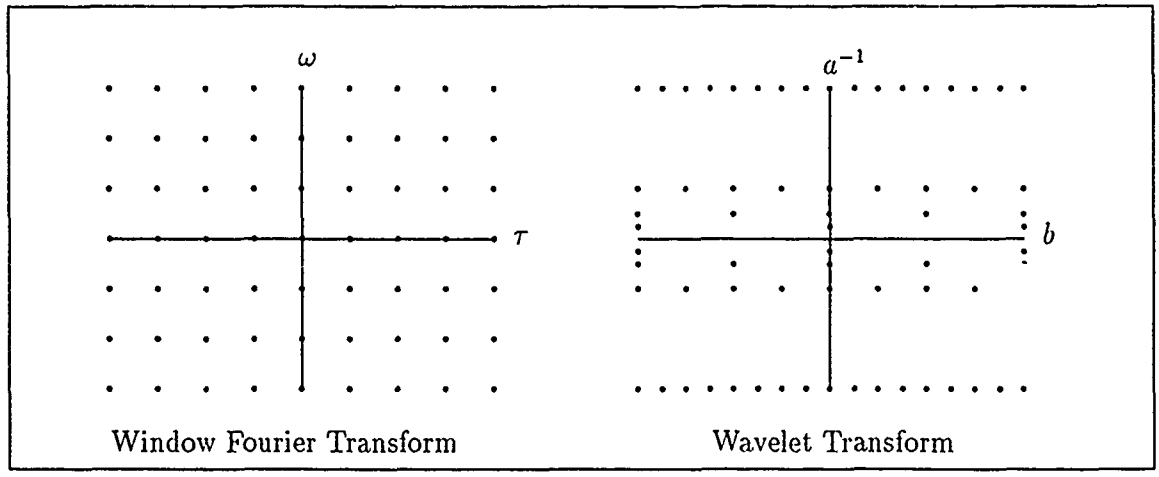


Figure 8. Time/Frequency Window Localization Lattice [7:41]

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 [5]. For this case, the discretized basis set $\{\psi_m^n\}$ where $m, n \in \mathbf{Z}$ is defined as

$$\psi_m^n(x) = \alpha^{-\frac{m}{2}} \psi(\alpha^{-m} x - n\beta) \tag{15}$$

where $\alpha > 1$ and $\beta > 0$ [10:11]. In this chapter, we use the dyadic interval defined to be $\alpha = 2$ and $\beta = 1$. For the dyadic case, Equation 15 becomes

$$\psi_m^n(x) = 2^{-\frac{m}{2}}\psi(2^{-m}x - n) \tag{16}$$

Using this form of the mother wavelet in Equation 12 yields the Wavelet Transform with a discrete wavelet basis.

$$W_f(m,n) = 2^{-\frac{m}{2}}\int_{-\infty}^{+\infty}\psi(2^{-m}x - n)f(x)dx \tag{17}$$

To check this, consider the Fourier Expansion given in Equation 1. We can represent any function, $f \in \mathbf{L}^2(\mathbf{R})$ as

$$f(x) = \sum_n c_n\psi_n(x) \tag{18}$$

where ψ_n is the n^{th} element of an orthonormal basis for $\mathbf{L}^2(\mathbf{R})$. Equation 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 $f \in \mathbf{L}^2(\mathbf{R})$ 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 18 by any term ψ_m for $m \in \mathbf{Z}$ and integrate, we get

$$\int_{-\infty}^{\infty}\psi_m(x)f(x)dx = \sum_n c_n\int_{-\infty}^{\infty}\psi_n(x)\psi_m(x)dx \tag{19}$$

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

$$\int_{-\infty}^{\infty}\psi_m(x)\psi_n(x)dx = \delta_{nm} \tag{20}$$

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 19 are zero except the one in which $n = m$.

Thus,

$$\int_{-\infty}^{\infty} f(x)\psi_m(x)dx = c_m \tag{21}$$

is the integral form we need to find the coefficient of the m^{th} basis element, c_m. Written another way, Equation 21 becomes a continuous transform with an orthonormal basis that maps $f(x) \rightarrow T_f(m)$.

$$T_f(m) = \int_{-\infty}^{+\infty} f(x)\psi_m(x)dx \tag{22}$$

Now, we can insert the orthonormal wavelet basis set $\{\psi_m^n\}$ of Equation 16 into Equation 22 and get the Wavelet Transform of Equation 17. To reconstruct the original signal, we perform a generalized Fourier series expansion (see Equation 18) with the coefficients obtained with Equation 17 and our basis set $\{\psi_m^n\}$.

$$f(x) = \sum_m \sum_n W_f(m,n)\psi(x)_m^n \tag{23}$$

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 [10].

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 [2, 36]. 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 1.2, 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 $\mathbf{L}^2(\mathbf{R})$. There exists a unique function $\phi(x) \in \mathbf{L}^2(\mathbf{R})$, 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
>
> $$(\sqrt{2^{-j}}\phi_{2^j}(x - 2^{-j}n))_{n\in\mathbf{Z}} \qquad (24)$$
>
> is an orthonormal basis of $\mathbf{V}_{2^j}$

[28:676]; see [28:690] for proof. In Mallat's theorem, $\mathbf{V}_{2^j}$ is a vector subspace of $\mathbf{L}^2(\mathbf{R})$ 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 9 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

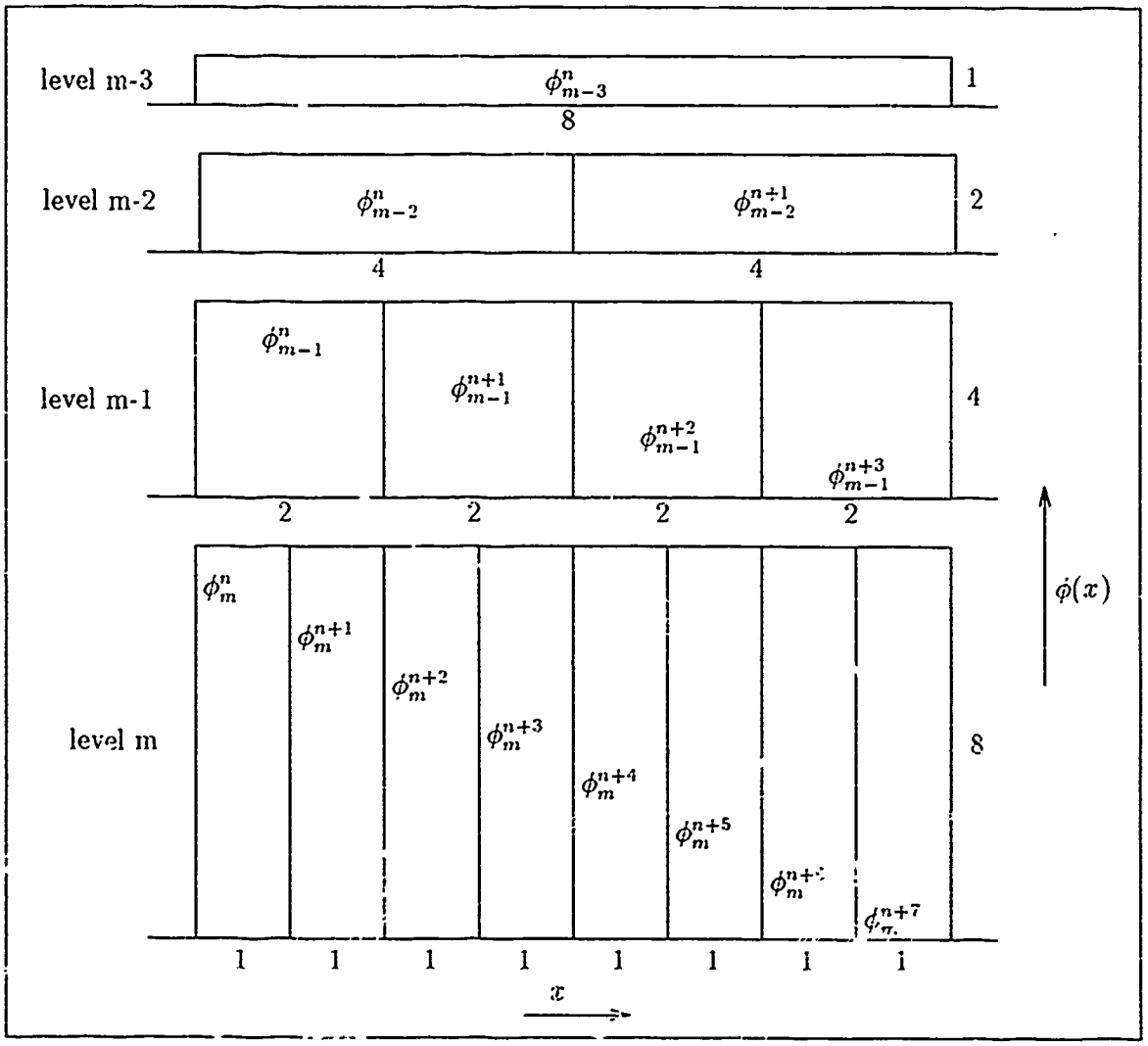


Figure 9. A Rectangular Scaling Function Dyadically Scaled

$$\mathbf{A}_{2^m} f(x) = \left(2^{-m} \sum_{n \in \mathbf{Z}} \langle f, \phi_{2^m}(\bullet - 2^{-m} n) \rangle \phi_{2^m}(x - 2^{-m} n) \right)_{m \in \mathbf{Z}} \tag{25}$$

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

$$\{ \langle f, \phi_{2^m}(\bullet - 2^{-m} n) \rangle \}_{m,n \in \mathbf{Z}} \tag{26}$$

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

$$\cdots \subset V_{m-2} \subset V_{m-1} \subset V_m \subset V_{m+1} \subset V_{m+2} \subset \cdots \tag{27}$$

That is, each resolution approximation of $\mathbf{L}^2(\mathbf{R})$ 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

$$V_{-(M-1)} \subset V_{-(M-2)} \subset \cdots \subset V_{-1} \subset V_0 \tag{28}$$

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

$$D_{m-1}f = A_m f - A_{m-1}f \tag{29}$$

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 25, this operator is

described in terms of a basis set[5] ψ_m^n which spans the space O_m.

$$D_m f(x) = \left(2^{-m} \sum_{n \in Z} \langle f, \psi_{2^m}(\bullet - 2^{-m}n) \rangle \psi_{2^m}(x - 2^{-m}n) \right)_{m \in Z} \tag{30}$$

Equation 30 generates the difference bet veen approximations. It is characterized by the set of inner products

$$\{ \langle f, \psi_{2^m}(\bullet - 2^{-m}n) \rangle \}_{m,n \in Z} \tag{31}$$

This is just Equation 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 10 shows an example of a mother wavelet dilated and translated dyadically. It follows from Equation 29 that the sum of all the detail signals and the coarsest approximation equals the original signal.

$$f(x) = D_{-1}f + \ldots + D_{-(M-1)}f + A_{-(M-1)}f \tag{32}$$

Equation 32 is the Wavelet Decomposition of $f(x)$.

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 26 are convolutions evaluated at the point $2^{-m}n$ (see section 3.2, Notation).

$$\langle f, \phi_{2^m}(\bullet - 2^{-m}n) \rangle = \int_{-\infty}^{+\infty} f(x) \phi_{2^m}(x - 2^{-m}n) dx = [(f * \phi_{2^m}(-\bullet)](2^{-m}n) \tag{33}$$

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

[5]Here, $\psi(x)$ is the particular mother wavelet associated with the scaling function, $\phi(x)$ used in Equation 25. Some researchers derive the ϕ given a ψ, and others derive the ψ given a ϕ [7]. In this thesis, we use previously derived ϕ,ψ pairs [28] [6].

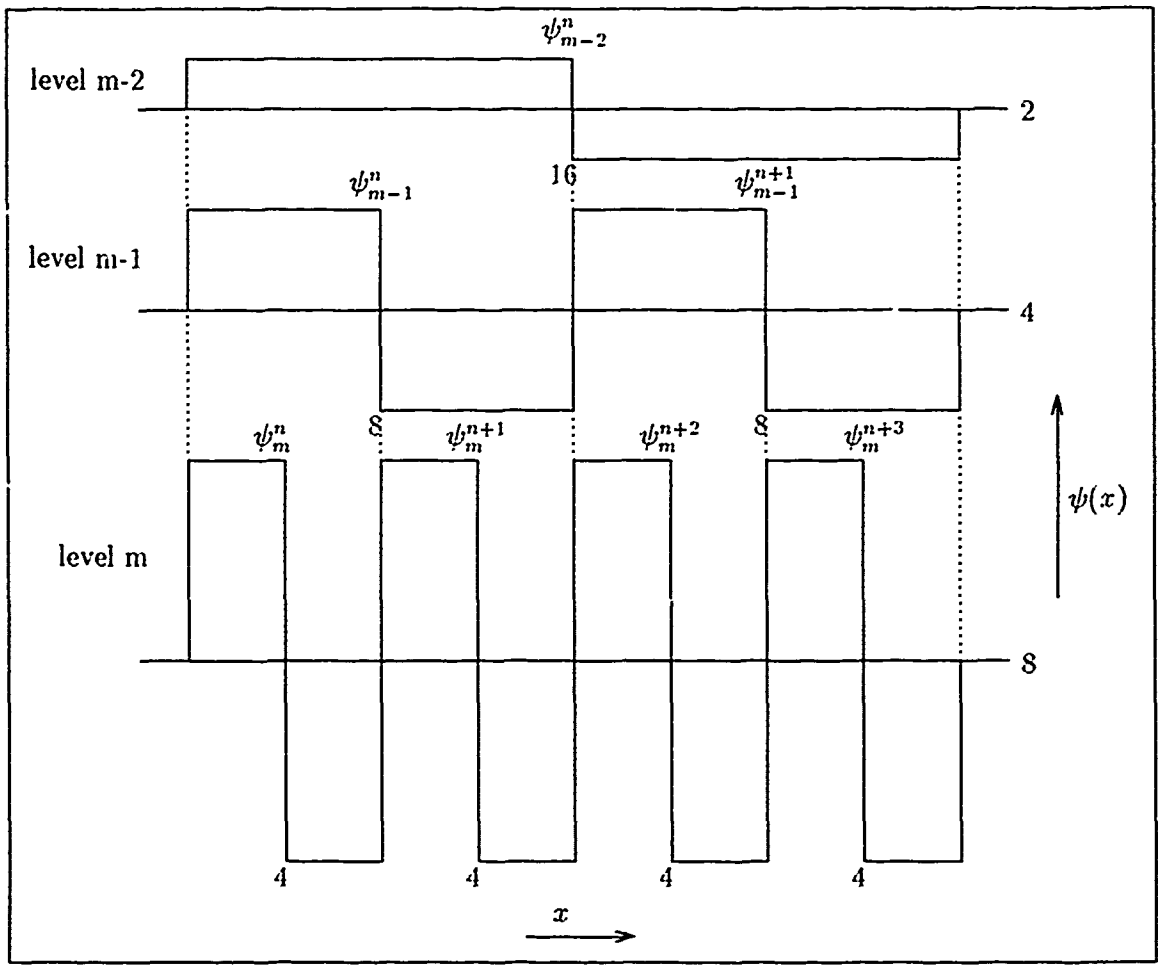


Figure 10. A Haar Mother Wavelet Function Dyadically Scaled

$$\langle f, \phi_{2^m}(\bullet - 2^{-m}n)\rangle = \int_{-\infty}^{+\infty} f(x)\phi_{2^m}(2^{-m}n - x)dx = [(f * \phi_{2^m}(x)](2^{-m}n) \qquad (34)$$

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

$$[f * g](x) \leftrightarrow F(\omega)G(\omega) \qquad (35)$$

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 31 as the convolution of the signal with the wavelet function evaluated at $2^{-m}n$ as we did in Equation 33.

$$\langle f, \psi_{2^m}(\bullet - 2^{-m}n)\rangle = [f * \psi_{2^m}(-\bullet)](2^{-m}n) \tag{36}$$

Figure 11 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 12 shows the wavelet function, $\psi(x)$, which corresponds to the scaling function of Figure 11. 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 $\mathbf{L^2(R^2)}$ and there exists a scaling function $\Phi(x, y)$ whose dilations and translations are an orthonormal basis for $\mathbf{L^2(R^2)}$ [27:682]. 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.

$$\Phi(x, y) = \phi(x)\phi(y) \tag{37}$$

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

$$\mathbf{A}_m f(x, y) = \left(2^{-m} \sum_{n_1 \in Z} \sum_{n_2 \in Z} \langle f, \phi_m(\bullet - 2^{-m} n_1)\phi_m(\bullet - 2^{-m} n_2)\rangle \phi_m(x - 2^{-m} n_1)\phi_m(y - 2^{-m} n_2) \right)_{m \in Z} \tag{38}$$

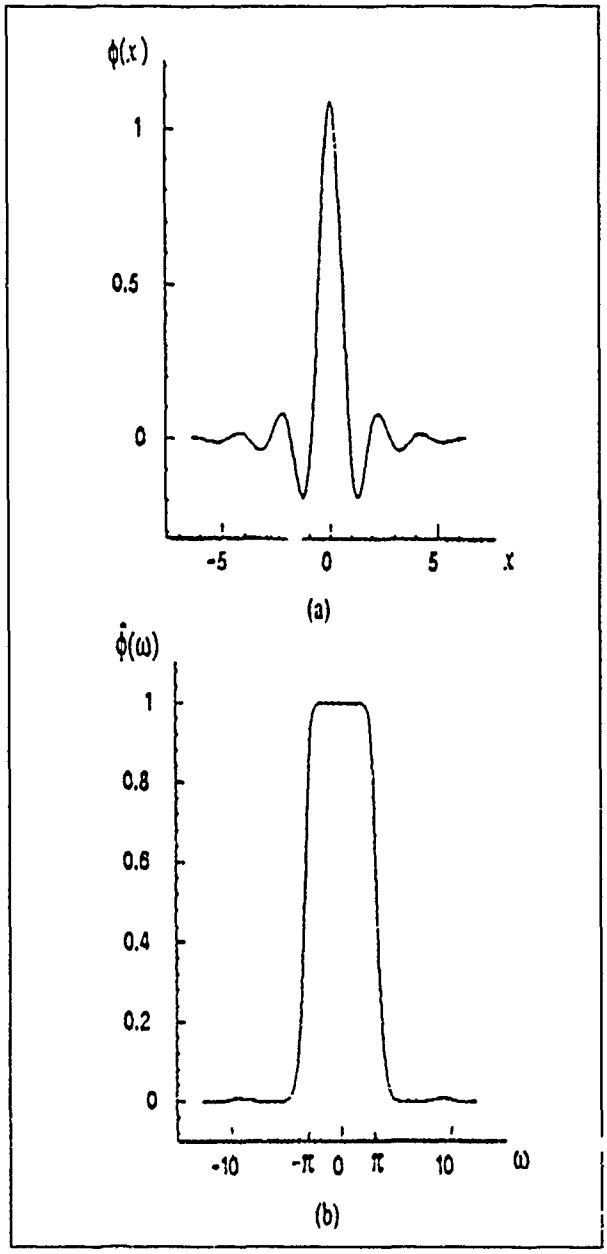


Figure 11. Typical Scaling Function and its Fourier Transform [28:677]

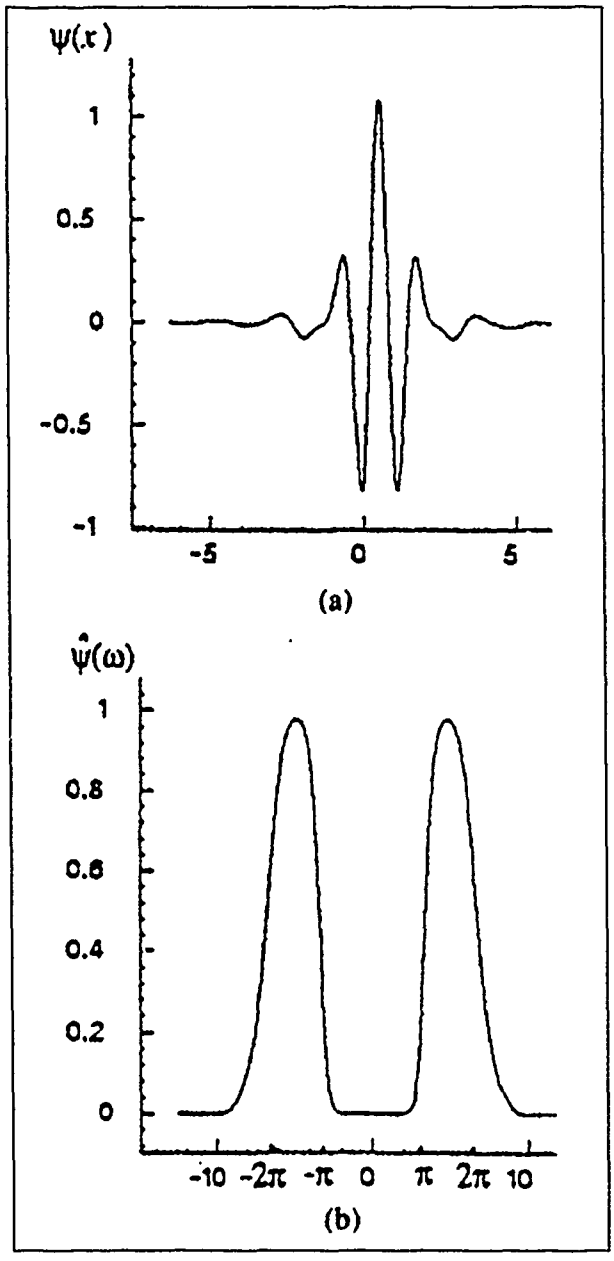


Figure 12. Typical Wavelet Function and its Fourier Transform [28:677]

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 31. The detailed image at resolution m is equal to the orthogonal projection of the 2D function on the orthonormal 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 [28:683]. 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.

$$\Psi_m^1(x,y) = \phi_m(x)\psi_m(y) \tag{39}$$

$$\Psi_m^2(x,y) = \psi_m(x)\phi_m(y) \tag{40}$$

$$\Psi_m^3(x,y) = \psi_m(x)\psi_m(y) \tag{41}$$

There is one detail projection for each of the three wavelet bases. Applying Equation 31 to each yields [28:684]

$$D_m^1 f(x,y) = \left(2^{-m} \sum_{n_1 \in Z} \sum_{n_2 \in Z} \langle f, \Psi_{2^m}^1(\bullet - 2^{-m}n_1, \bullet - 2^{-m}n_2)\rangle \Psi_{2^m}^1(x - 2^{-m}n_1, y - 2^{-m}n_2)\right)_{m \in Z} \tag{42}$$

$$D_m^2 f(x,y) = \left(2^{-m} \sum_{n_1 \in Z} \sum_{n_2 \in Z} \langle f, \Psi_{2^m}^2(\bullet - 2^{-m}n_1, \bullet - 2^{-m}n_2)\rangle \Psi_{2^m}^1(x - 2^{-m}n_1, y - 2^{-m}n_2)\right)_{m \in Z} \tag{43}$$

$$D_m^3 f(x,y) = \left(2^{-m} \sum_{n_1 \in Z} \sum_{n_2 \in Z} \langle f, \Psi_{2^m}^3(\bullet - 2^{-m}n_1, \bullet - 2^{-m}n_2)\rangle \Psi_{2^m}^1(x - 2^{-m}n_1, y - 2^{-m}n_2)\right)_{m \in Z} \tag{44}$$

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 13 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 42 through 44, recent work by Cohen and Schlenker at AT&T Bell Laboratories suggest more are possible [3].

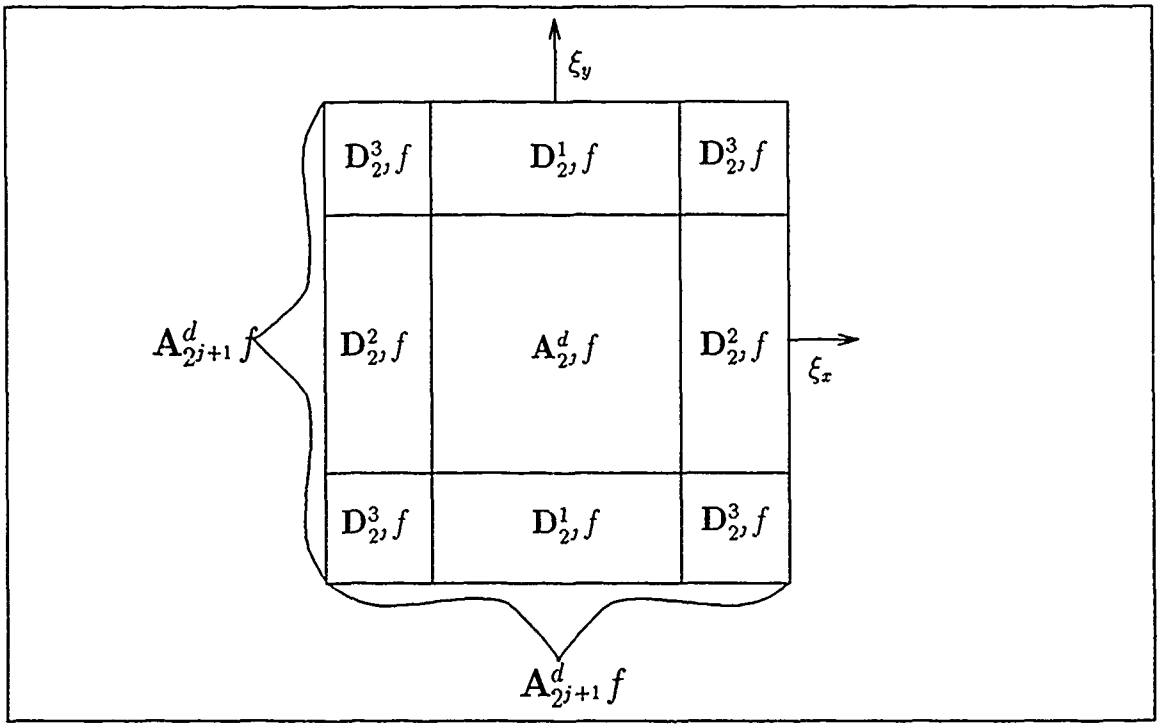


Figure 13. Orientation of Wavelet Decomposition Filters in the Fourier Domain [10:65]

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 Steven Smiley and exists in duplicate in his thesis [41]. 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 $\mathbf{L}^2(\mathbf{R})$ (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:

$$\psi(x) = \begin{cases} 1 & \text{if } 0 \leq x < \frac{1}{2} \\ -1 & \text{if } \frac{1}{2} \leq x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{45}$$

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

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

$$\phi(x) = \begin{cases} 1 & \text{if } 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{46}$$

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

$$\{|I_m^n| = 2^m\}_{m,n \in \mathbf{Z}} \tag{47}$$

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

$$|I_0^n| = 1 \tag{48}$$

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

$$\phi_m^n(x) = \begin{cases} \frac{1}{\sqrt{2^m}} & \text{if } x \in I_m^n \\ 0 & \text{otherwise} \end{cases} \tag{49}$$

From Equation 49, we build a two dimensional scaling function with the product mentioned above as follows

$$\Phi_m^n(x, y) = \begin{cases} 2^{-m} & x, y \in I_m^n \\ 0 & \text{otherwise} \end{cases} \tag{50}$$

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

$$c_m^n =< \Phi_m^n, f > \tag{51}$$

Then, the projection is given by

$$A_m f(x,y) = \sum_n c_m^n \Phi_m^n(x,y) \tag{52}$$

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 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 51 using

$$d_m^n = \langle \Psi_m^n, f \rangle \tag{53}$$

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 29

4.2.2 Haar Transform Program The data flow diagram in Figures 14 and 15 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

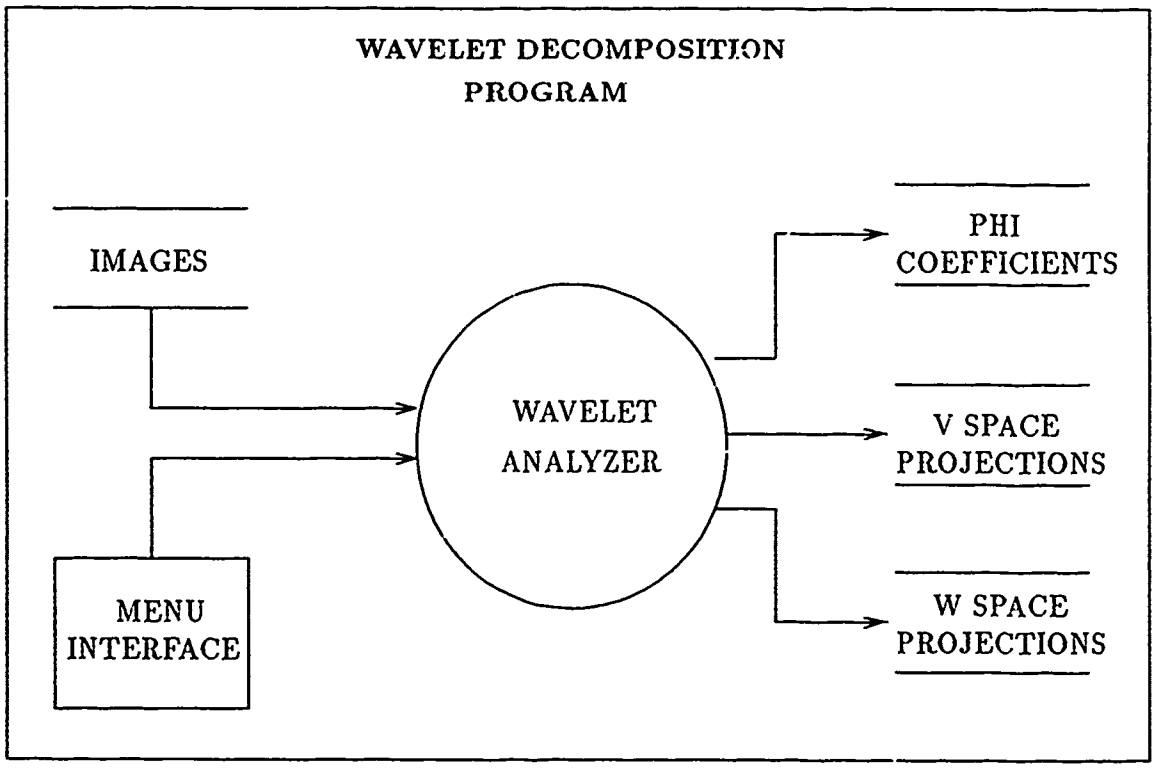


Figure 14. Dataflow Diagram of the Wavelet Decomposition Program, First Level

be decomposed. For example, the image of Lenna shown in Figure 16 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 51. The projections of the input image onto the V space are found by multiplying the Φ basis by the Φ coefficients, Equation 52. Then, the projections in the W space are found from the difference of V space projections at adjacent levels, Equation 30. The source code for the *wave* program is made up of ten files. They are provided in their entirety in Appendix A.2.

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 17 through 28). The W space projections are made viewable by adding 255 to

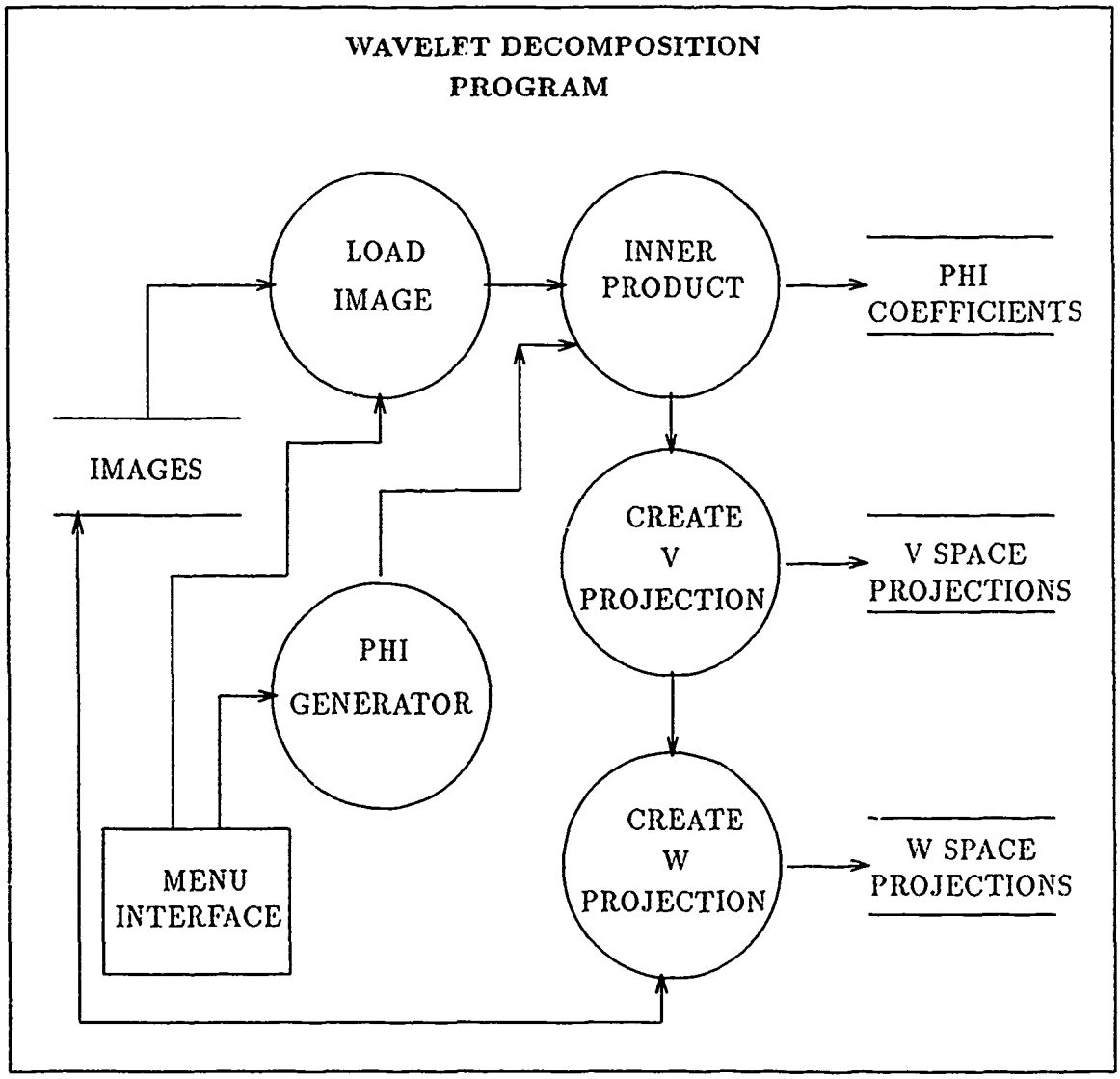


Figure 15. Dataflow Diagram of the Wavelet Decomposition Program, Second Level

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

Figure 16. Projection of Lenna onto V_0

Figure 17. Projection of Lenna onto V_1

Figure 18. Projection of Lenna onto V_2

Figure 19. Projection of Lenna onto V_3

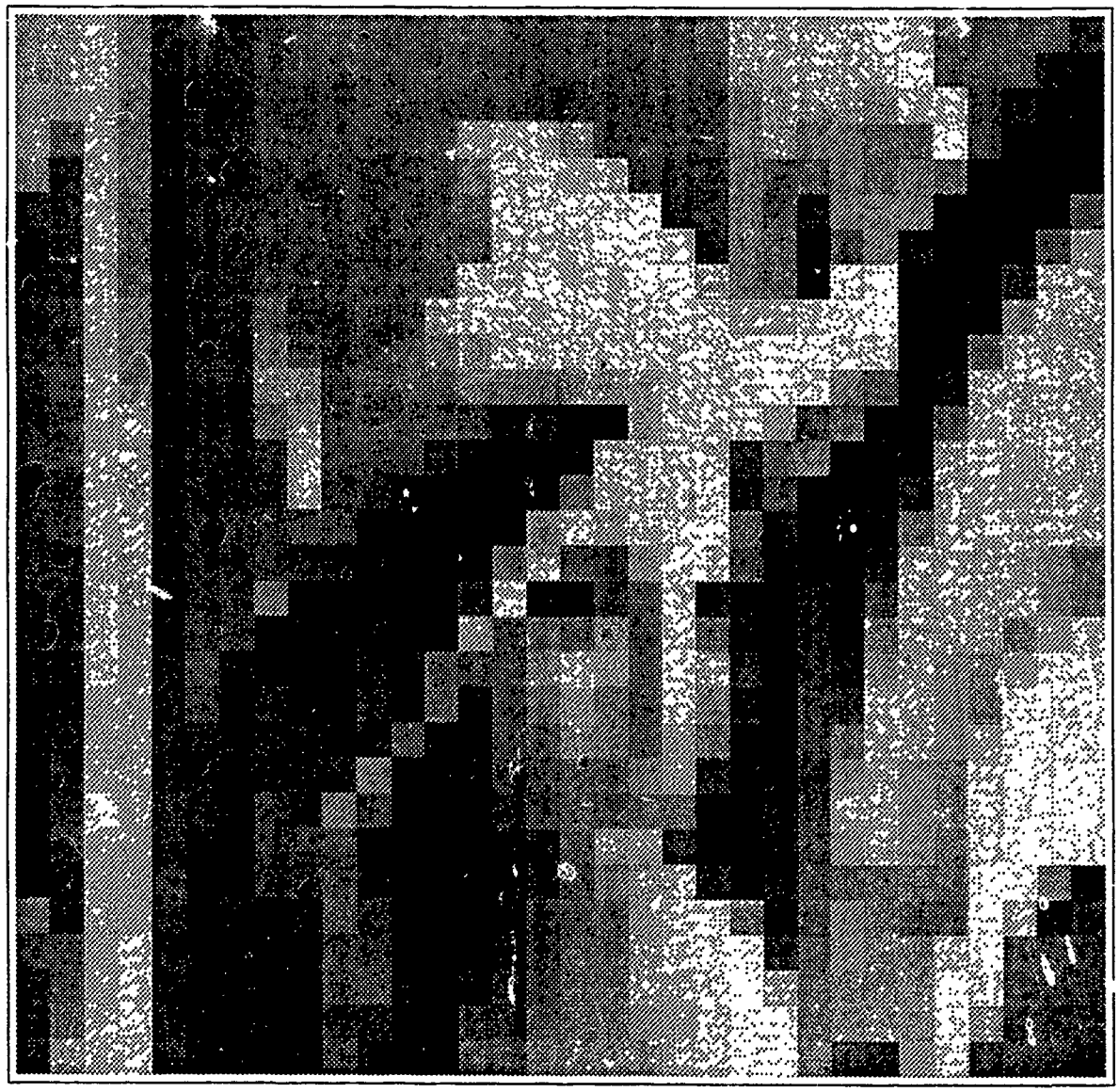


Figure 20. Projection of Lenna onto V_4

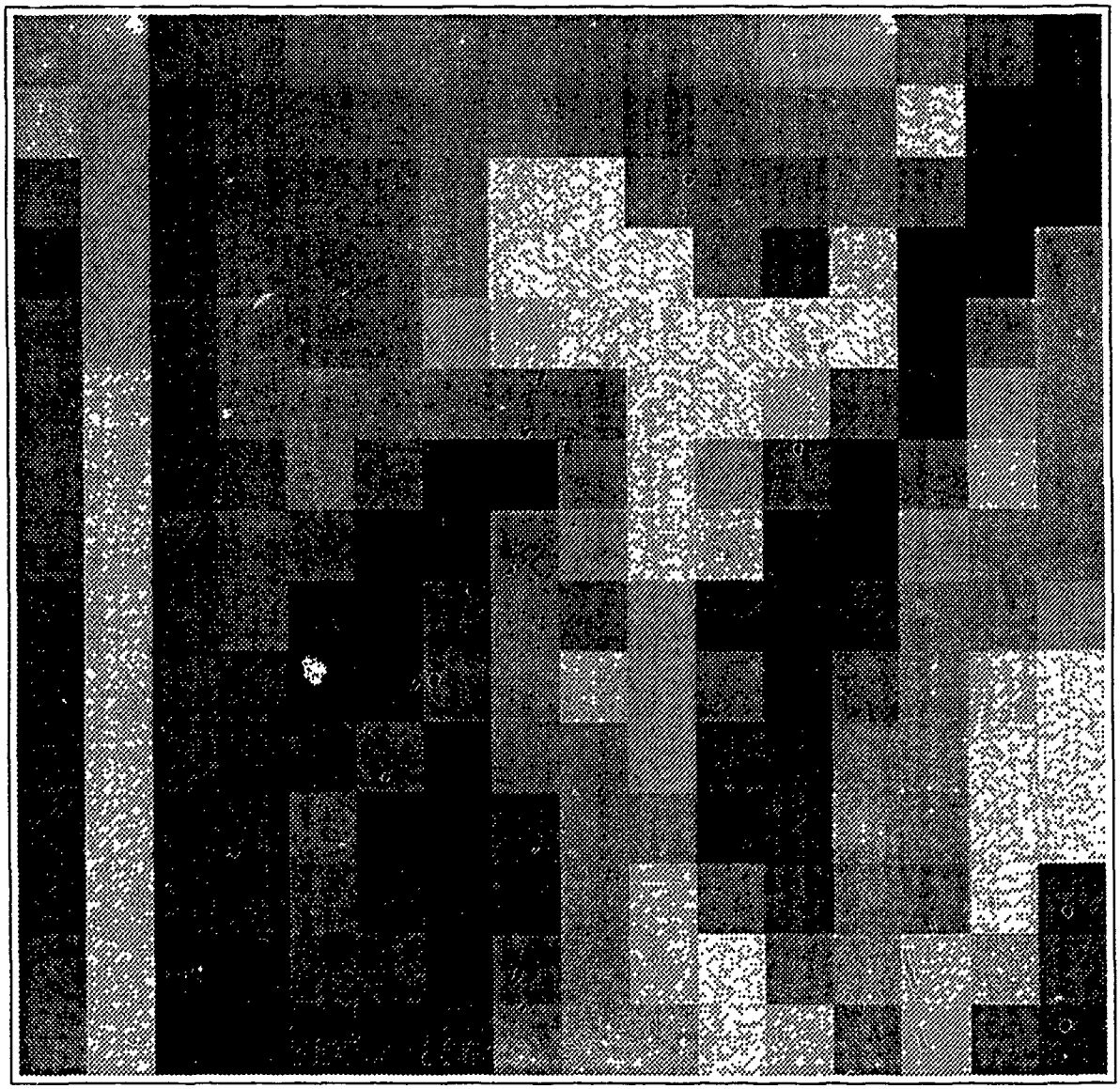


Figure 21. Projection of Lenna onto V_5

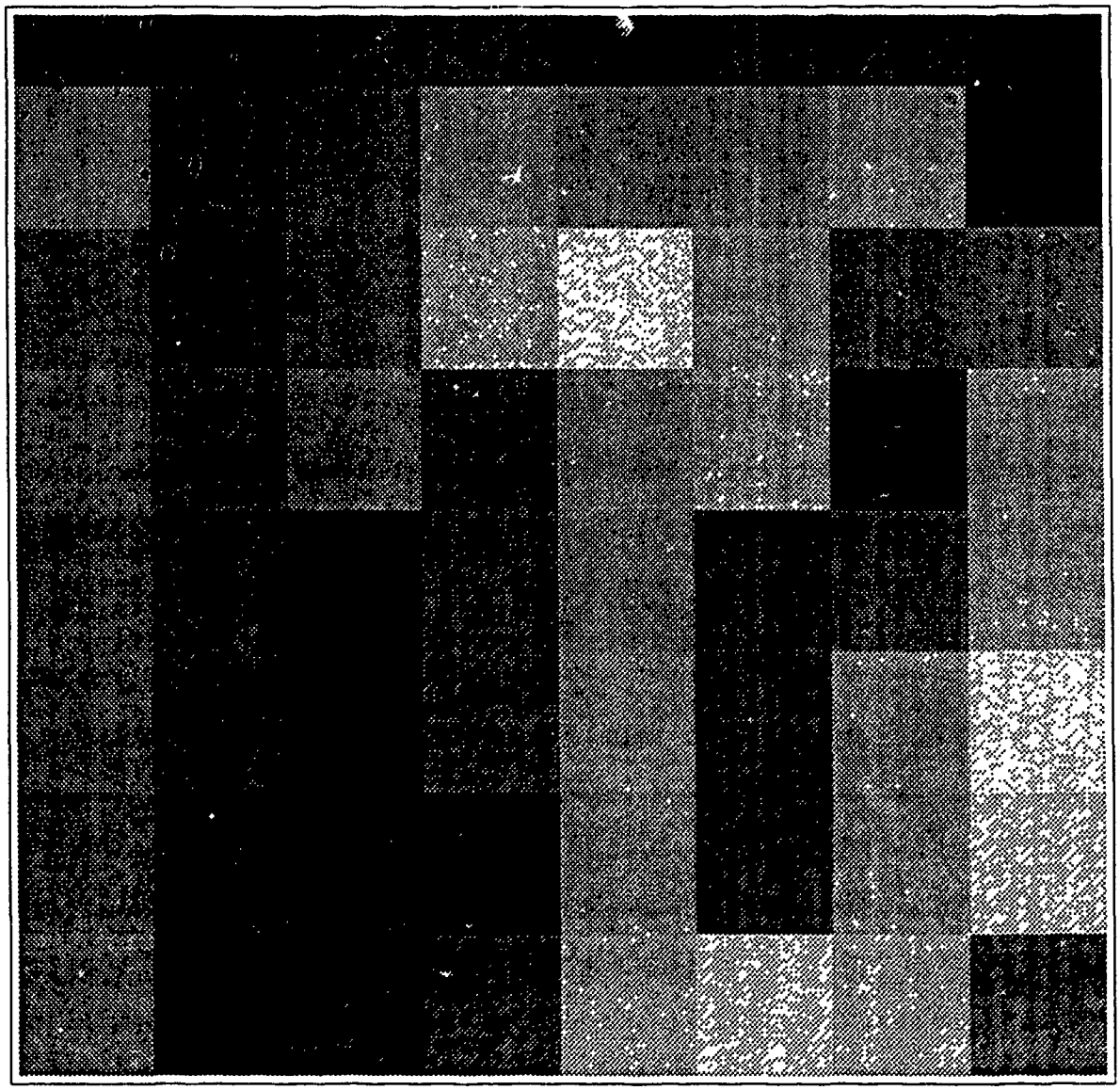


Figure 22. Projection of Lenna onto V_6

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 23 through 28 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 17 through 22).

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 [37]. Figures 29 and 30 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 23 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 31, 32, and 33 shows the results of executing the *threshold* program on the first three levels of W space projections. These figures demonstrate the edge detection capability

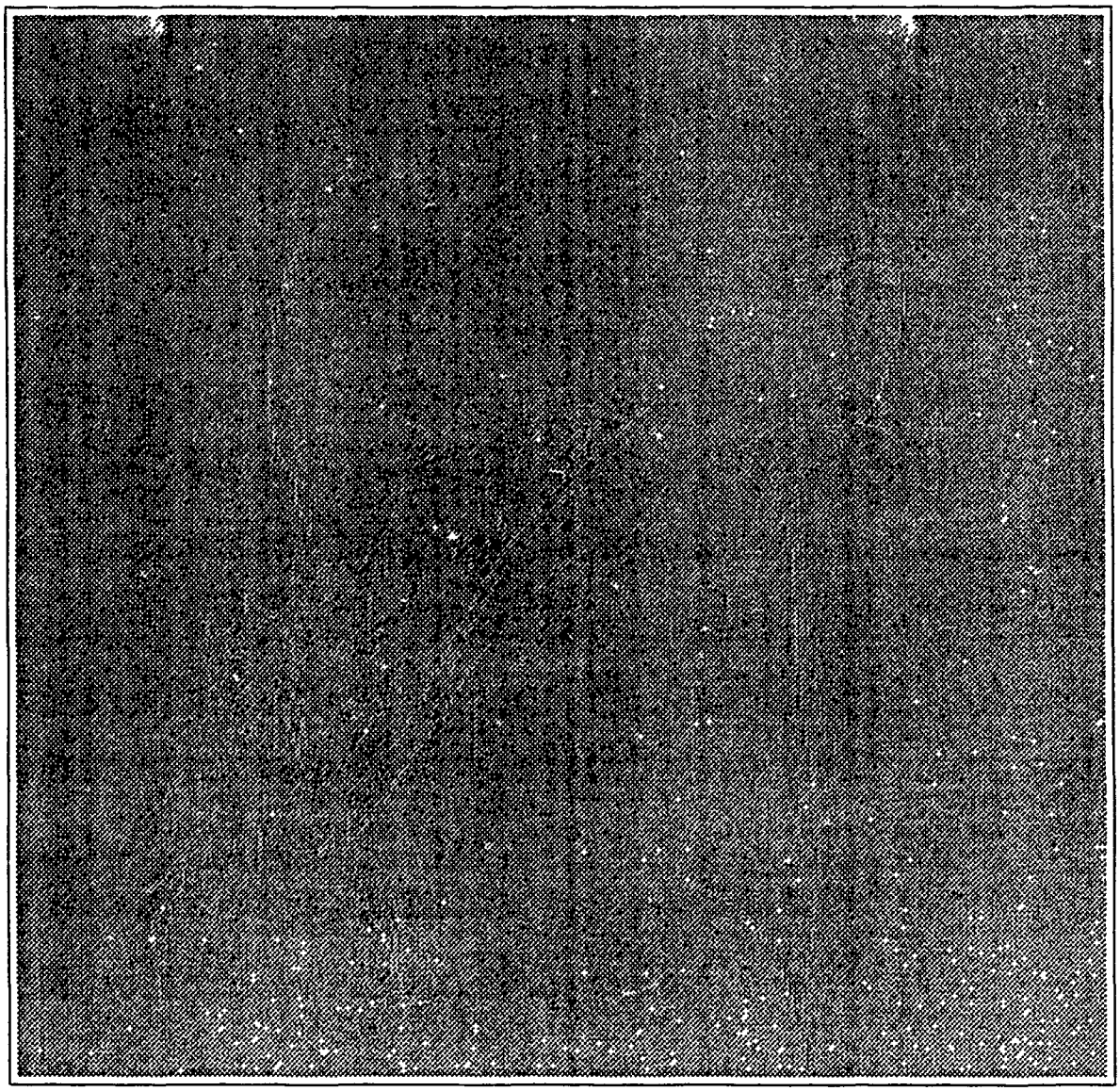


Figure 23. Projection of Lenna onto W_1

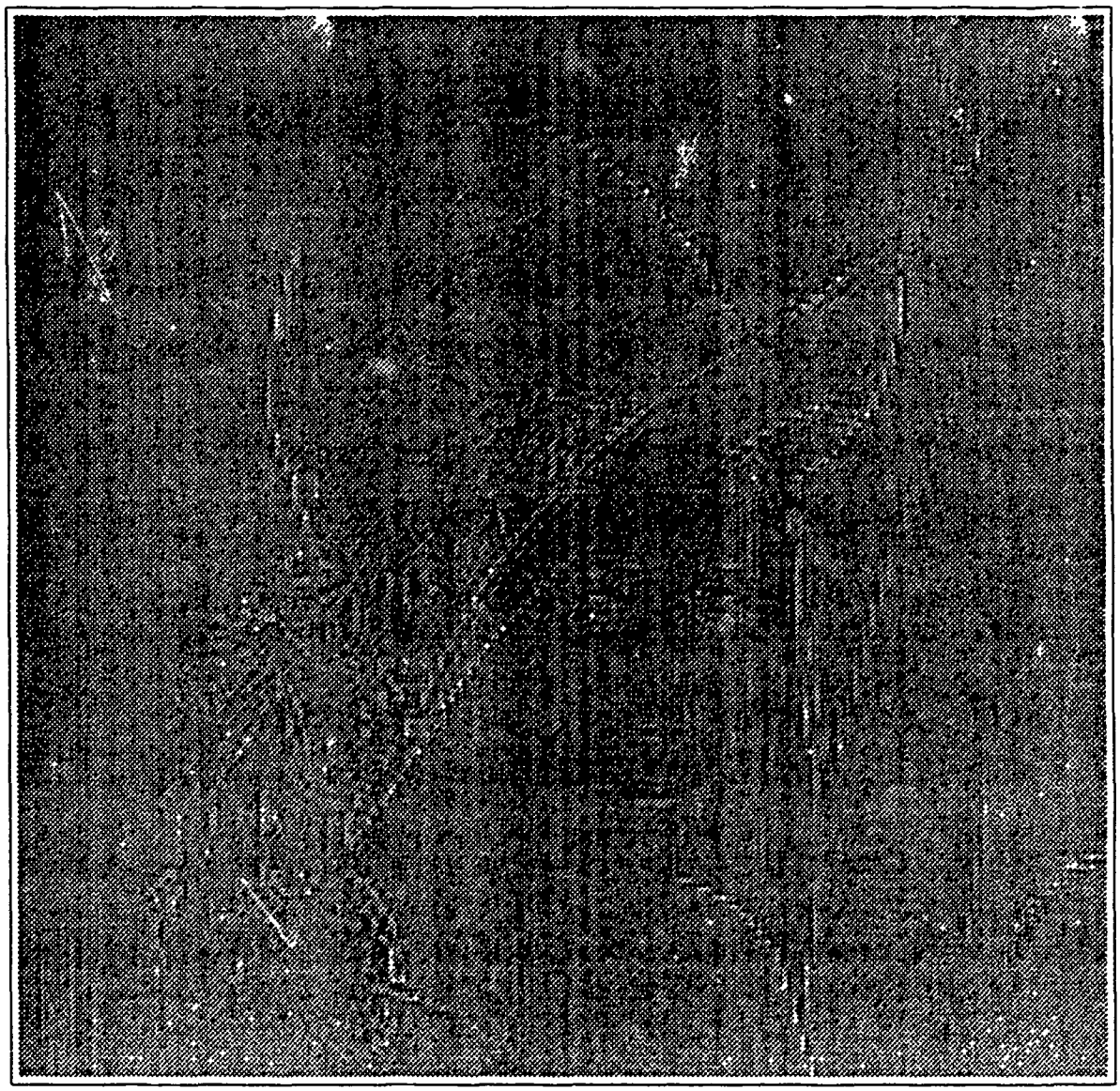


Figure 24. Projection of Lenna onto W_2

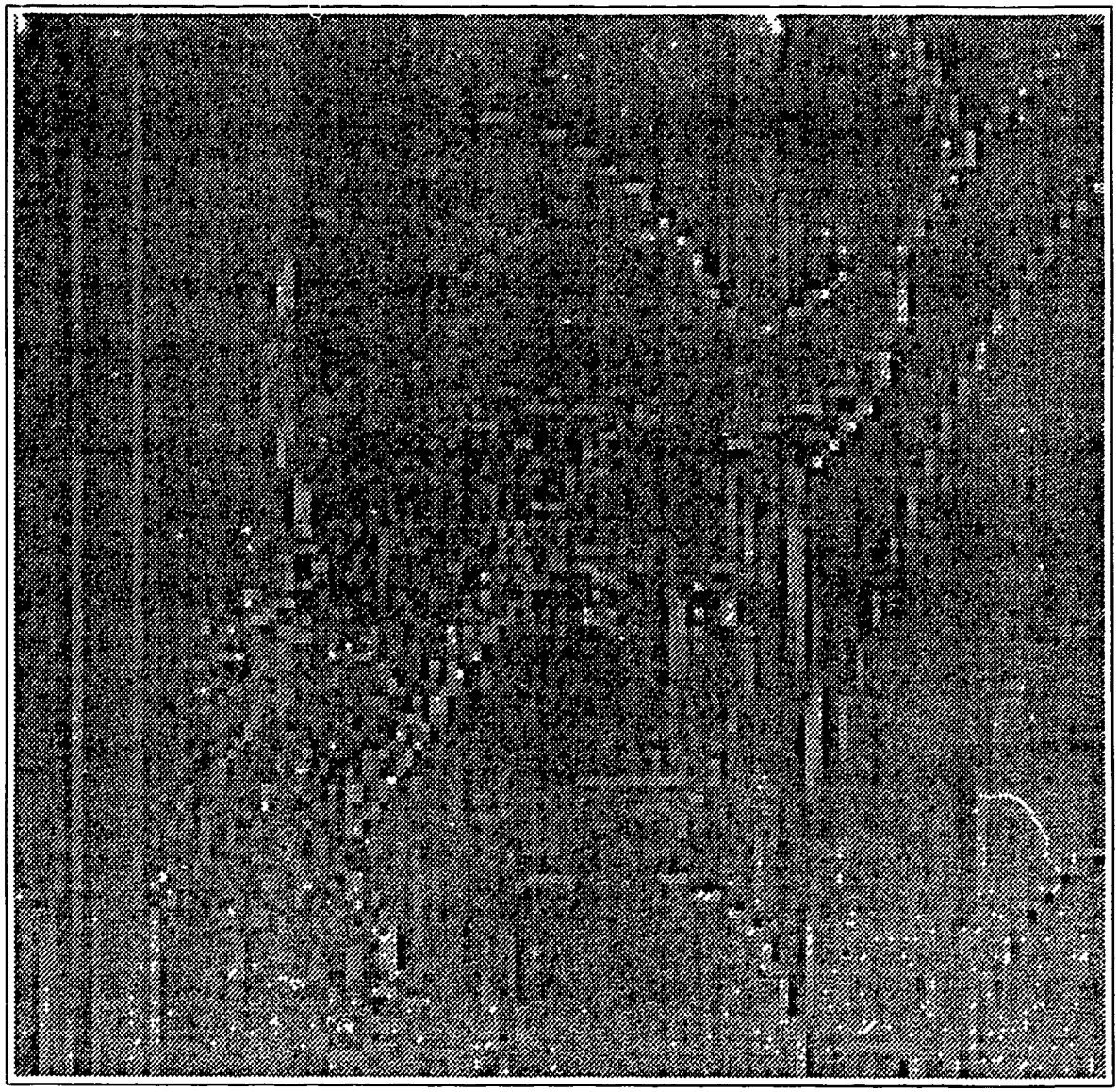


Figure 25. Projection of Lenna onto W_3

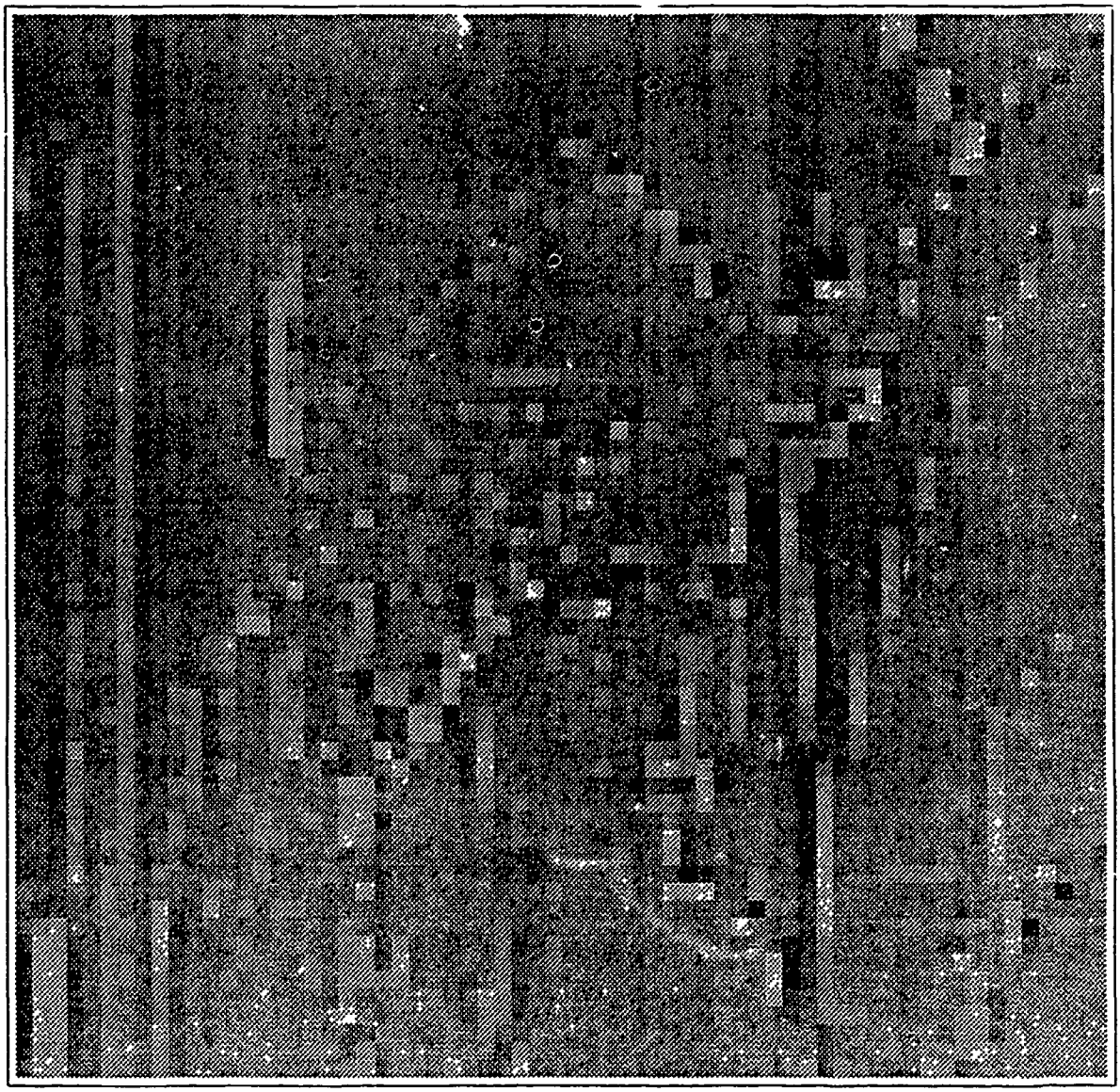


Figure 26. Projection of Lenna onto W_4

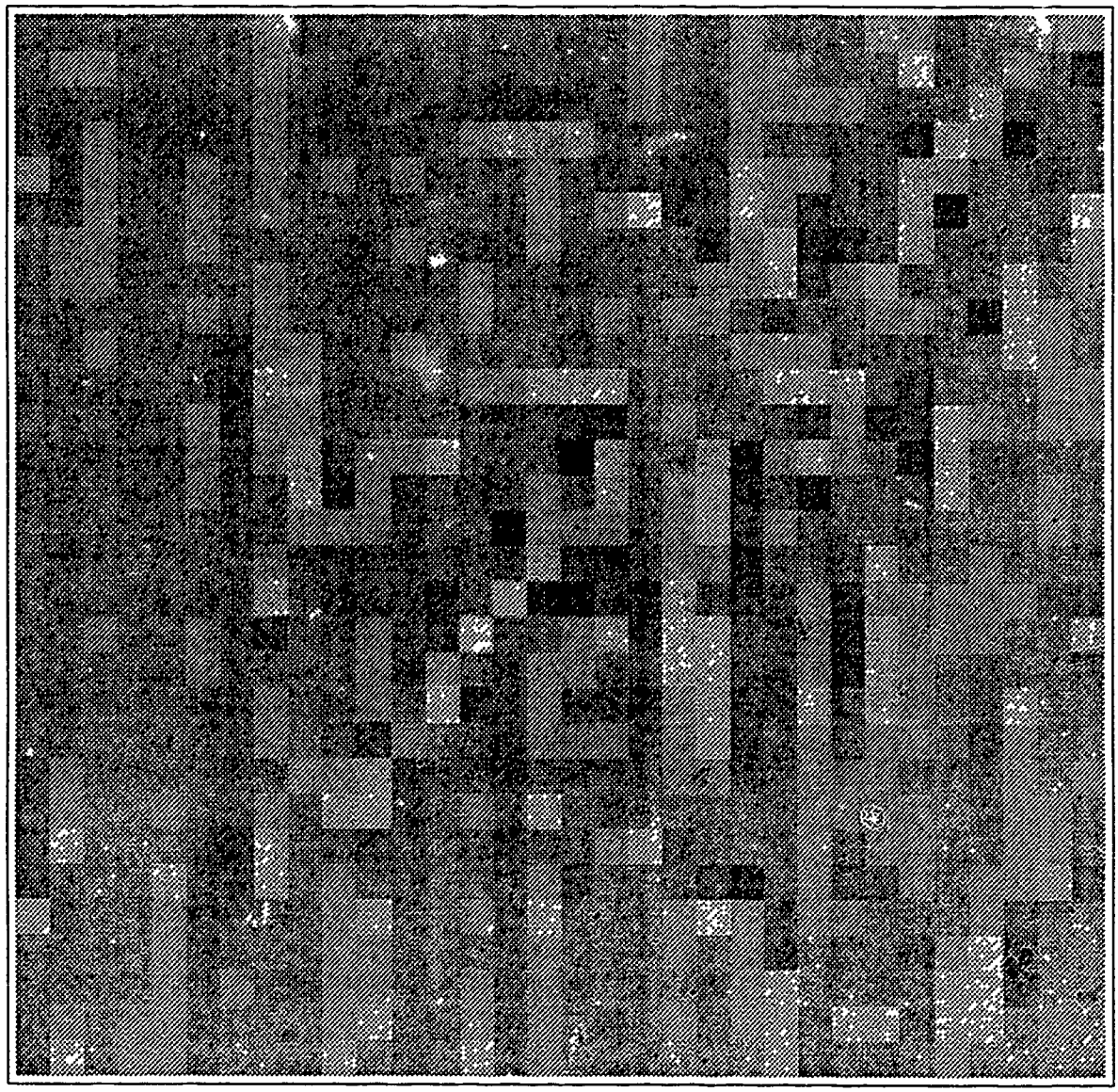


Figure 27. Projection of Lenna onto W_5

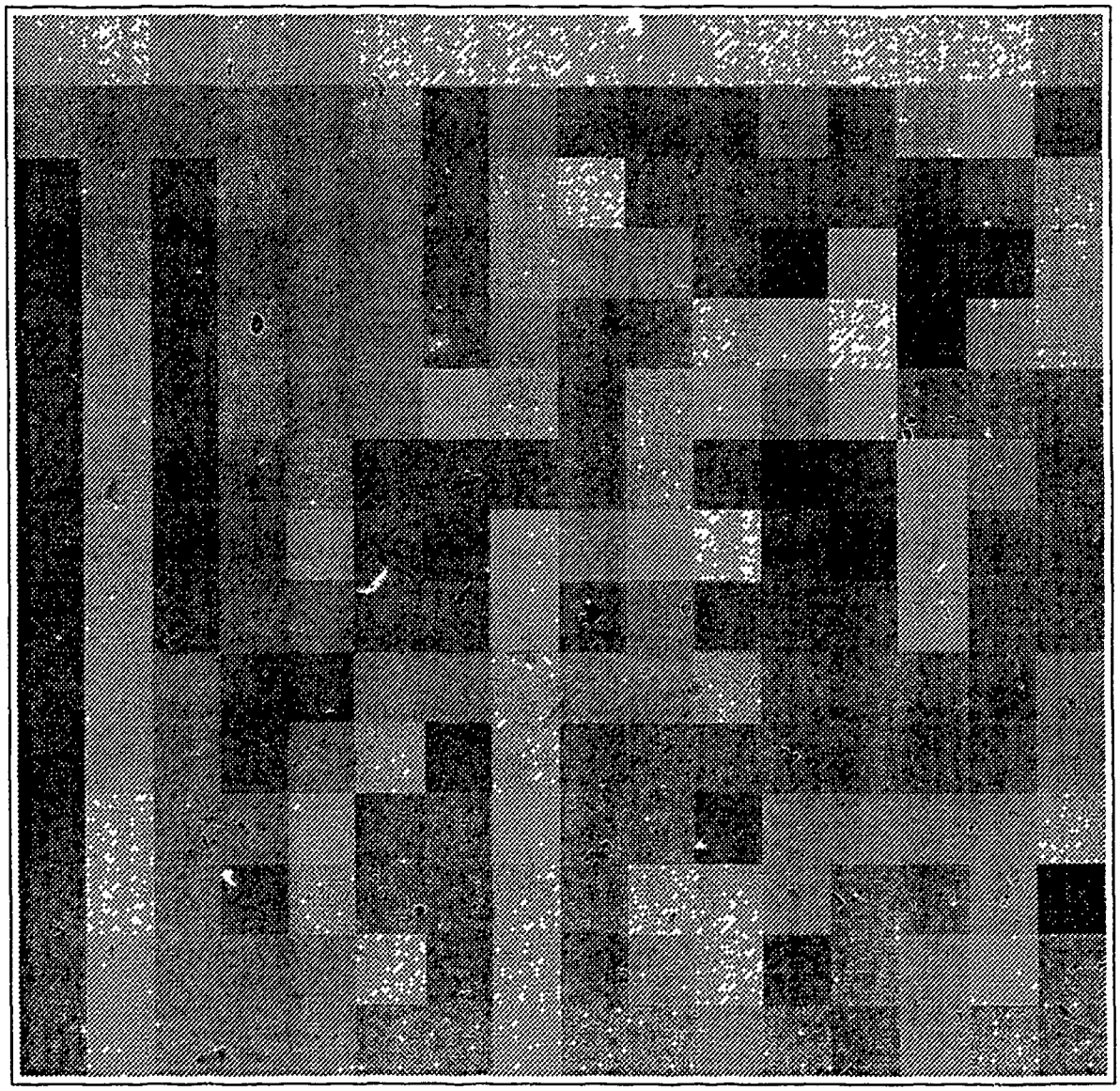


Figure 28. Projection of Lenna onto W_6

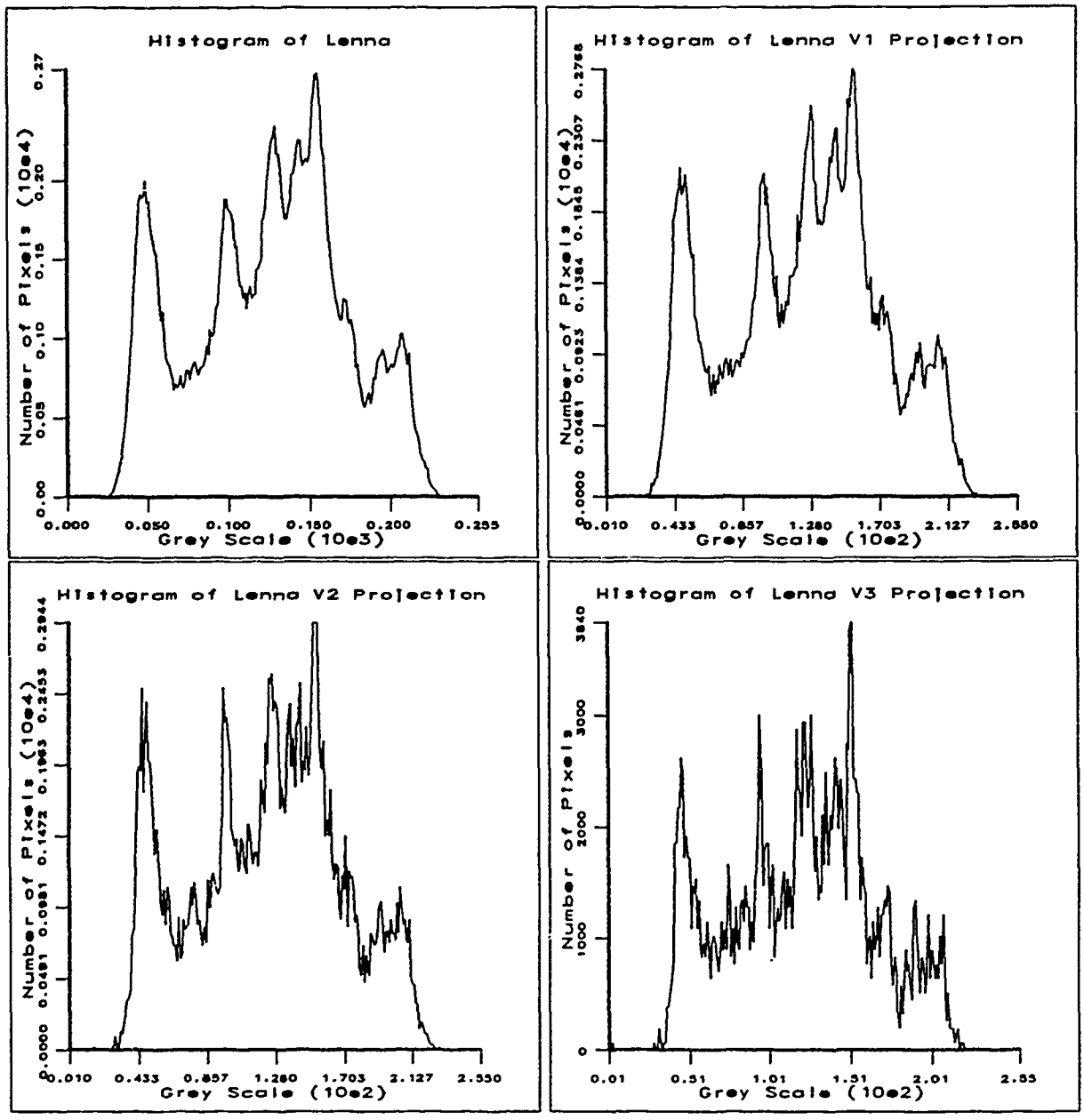


Figure 29. Histograms of Lenna's Original Image and V_1 through V_3 Projections

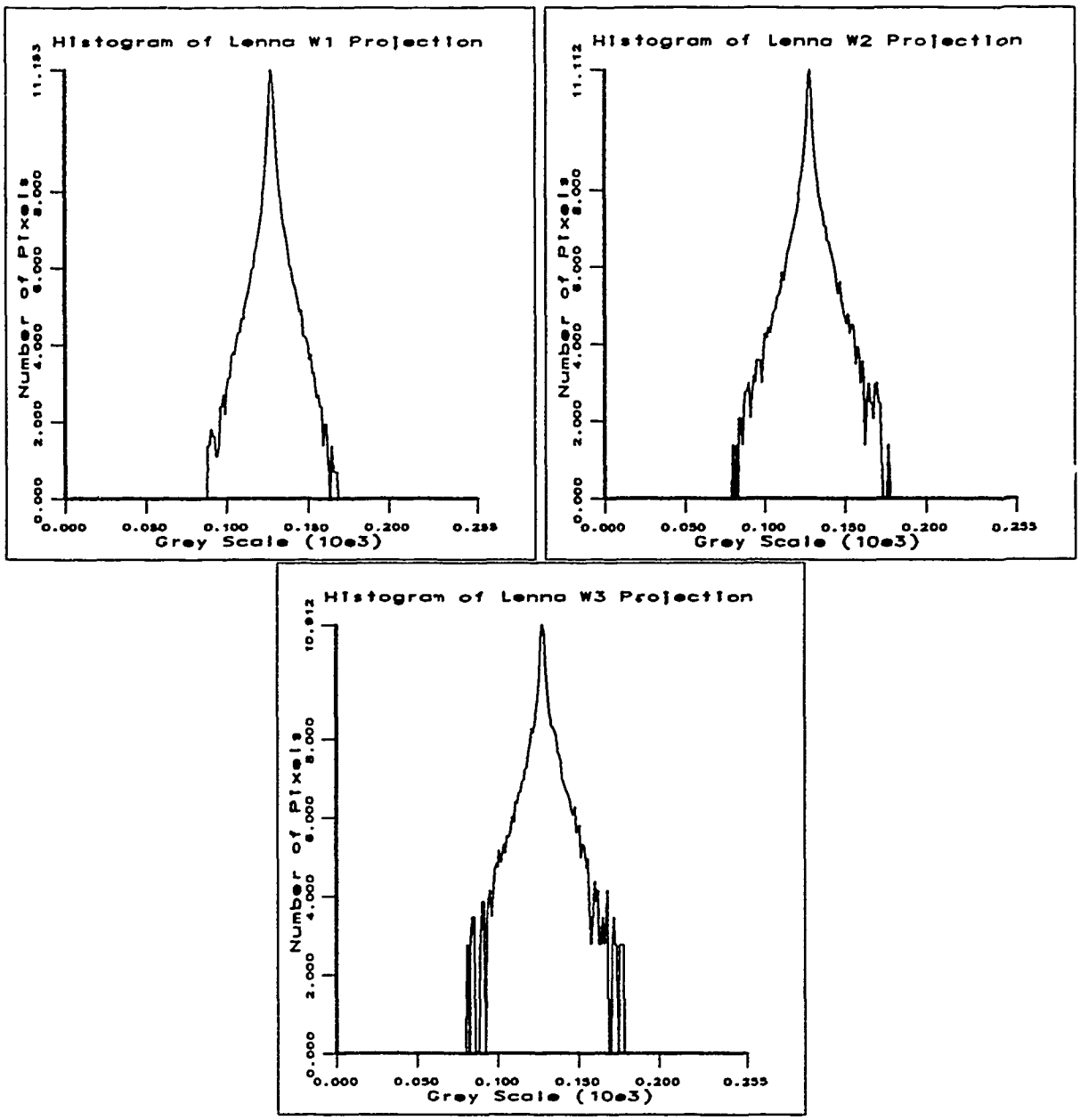


Figure 30. Histograms of Lenna's W_1, W_2, and W_3 Projections with the Number of Pixels Logged

Figure 31. Lenna's W_1 Projection Thresholded

Figure 32. Lenna's W_2 Projection Thresholded

Figure 33. Lenna's W_3 Projection Thresholded

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 [28:677]. 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 [28].

4.3.1 Multiresolution Decomposition In Mallat's Equation (10) [28:677], he gives the "orthogonal projection" of a signal $f(x)$ onto a scale space, V, of an arbitrary level of resolution, 2^j for $j \in \mathbf{Z}$ as

$$\mathbf{A}_{2^j} f(x) = 2^{-j} \sum_{n=-\infty}^{+\infty} \langle f, \phi_{2^j}(\bullet - 2^{-j}n)\rangle \phi_{2^j}(x - 2^{-j}n), \forall f \in \mathbf{L}^2(\mathbf{R}) \tag{54}$$

Then in Equation (11) [28:677], 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.

$$\mathbf{A}^d_{2^j} f = \{\langle f, \phi_{2^j}(\bullet - 2^{-j}n)\rangle\}_{n \in \mathbf{Z}} \tag{55}$$

The discrete set of inner products in Equation 55 is the set of scaling function coefficients previously given in this thesis in Equation 51 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 even 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 55 via convolution the level $j = 0$ scale coefficients are

$$\mathbf{A}_1^d f = \{(f * \phi_1(-\bullet))(n)\}_{n \in \mathbf{Z}} \tag{56}$$

for one dimension and

$$\mathbf{A}_1^d f = \{(f * \phi_1(-\bullet) * \phi_1(-\bullet))(n, m)\}_{n,m \in \mathbf{Z}} \tag{57}$$

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 56 into projection Equation 54 yields

$$\mathbf{A}_1 f(k) = \sum_{n=-\infty}^{\infty} (\mathbf{A}_1^d f)(n) \phi_1(k - n) \tag{58}$$

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

$$((\mathbf{A}_1^d f)(n) * \phi_1(n))(k) \tag{59}$$

Using Equation 59 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 59 using the discrete variables k and l in place of the continuous variables x and y respectively is

$$\mathbf{A}_1 f(k,l) = ((\mathbf{A}_1^d f)(n,m) * \phi_1(n) * \phi_1(m))(k,l) \tag{60}$$

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) [28:677], 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) [28:678] is

$$\mathbf{A}_{2^j}^d f = \{(\mathbf{A}_{2^{j+1}}^d f * \tilde{h})(2n)\}_{j,n \in \mathbf{Z}} \tag{61}$$

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 61 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) [28:681] and is interpreted as

$$\mathbf{D}_{2^j} f = \{(\mathbf{A}_{2^{j+1}}^d f * \tilde{g})(2n)\}_{j,n,k \in \mathbf{Z}} \tag{62}$$

where $\tilde{g}(n) = g(-n)$. The filters G and H have the following relationship [28:681]

$$g(n) = (-1)^{1-n} h(1-n) \tag{63}$$

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 63, for whatever convolution routine is used.

Now, armed with a set of response functions, $h(n)$ and $g(n)$, Equations 61 and 62 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, $\mathbf{A}_2^d f$ and $\mathbf{D}_2 f$, 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 12 are shown in Figure 35. Using Equation 63, we derived the response function $g(n)$ from $h(n)$. It is plotted along with its highpass filter G in Figure 35. 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 61 and 62 is diagramed in Figure 34 which is redrawn from [28:681].

4.3.2 Two Dimensional Multiresolution Decomposition The two dimensional case is a natural extension from one dimension. Equations 38, 42, 43, and 44 give the scale and detail coefficients. These correspond to Mallat's Equations (39) through (40) [28:684]. Our interpretation of these equations when the response functions $h(n)$ and $g(n)$ are incorporated

is as follows:

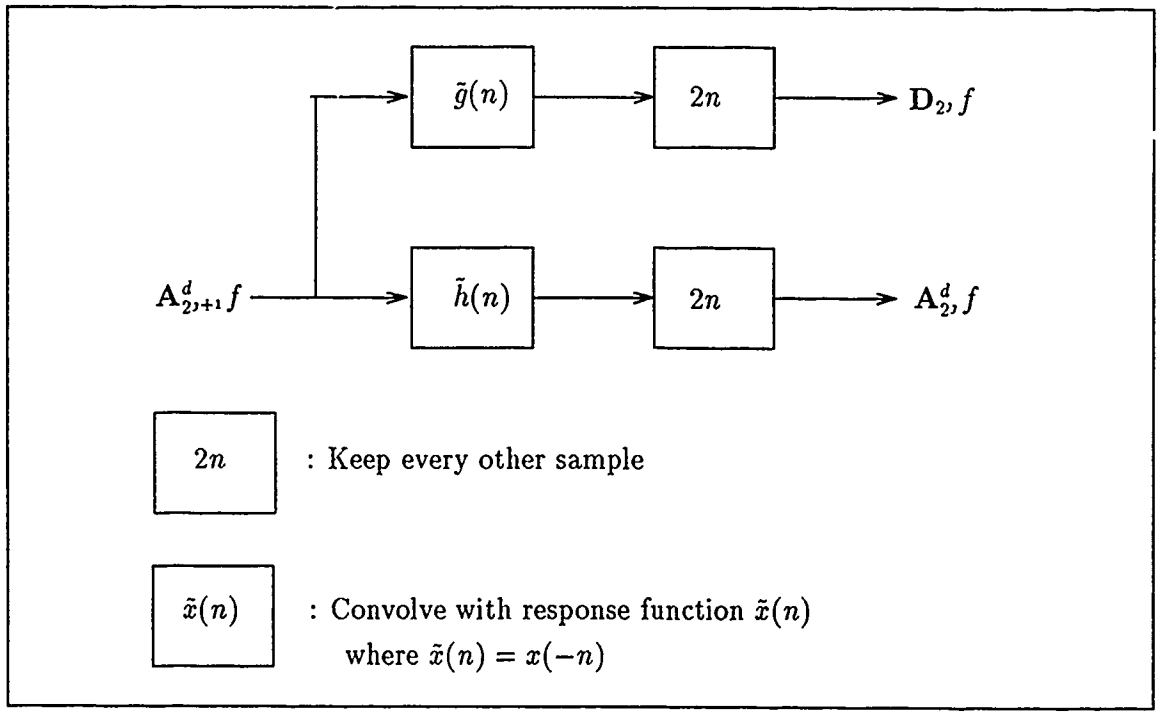


Figure 34. One Dimensional Multiresolution Decomposition [28:681]

$$\mathbf{A}^d_{2^j} f = (\mathbf{A}^d_{2^{j+1}} f)(k,l) * h(k) * h(l))(2n, 2m) \tag{64}$$

$$\mathbf{D}^1_{2^j} f = (\mathbf{A}^d_{2^{j+1}} f)(k,l) * \tilde{h}(k) * \tilde{g}(l))(2n, 2m) \tag{65}$$

$$\mathbf{D}^2_{2^j} f = (\mathbf{A}^d_{2^{j+1}} f)(k,l) * \tilde{g}(k) * \tilde{h}(l))(2n, 2m) \tag{66}$$

$$\mathbf{D}^3_{2^j} f = (\mathbf{A}^d_{2^{j+1}} f)(k,l) * \tilde{g}(k) * \tilde{g}(l))(2n, 2m) \tag{67}$$

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 13.

In Equations 64 through 67, 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

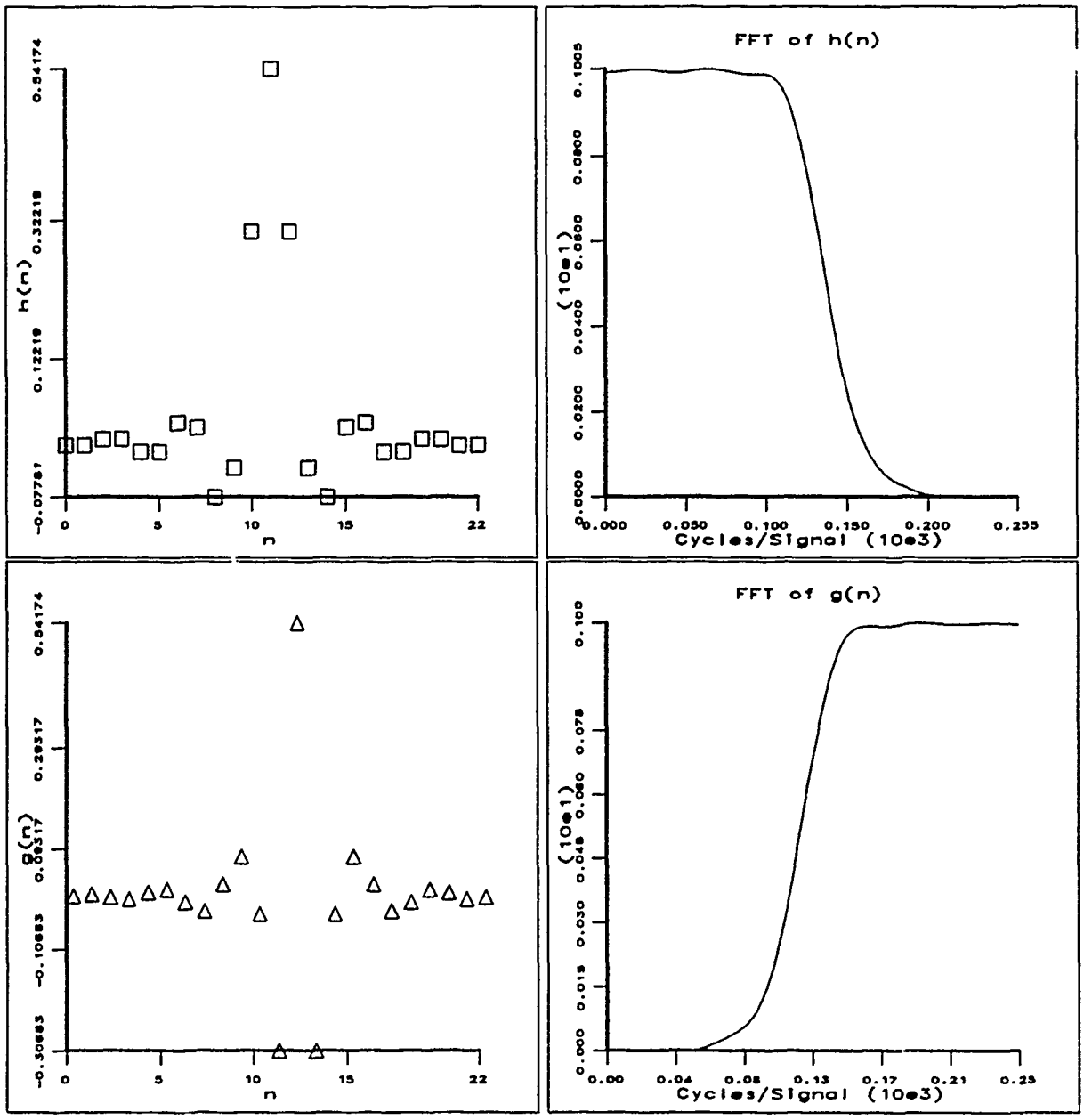


Figure 35. Response and Filter Functions Based on Cubic Spline Wavelet

emphasis plays an important role in understanding the mistake in Mallat's Figure 12 [28:685] 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.

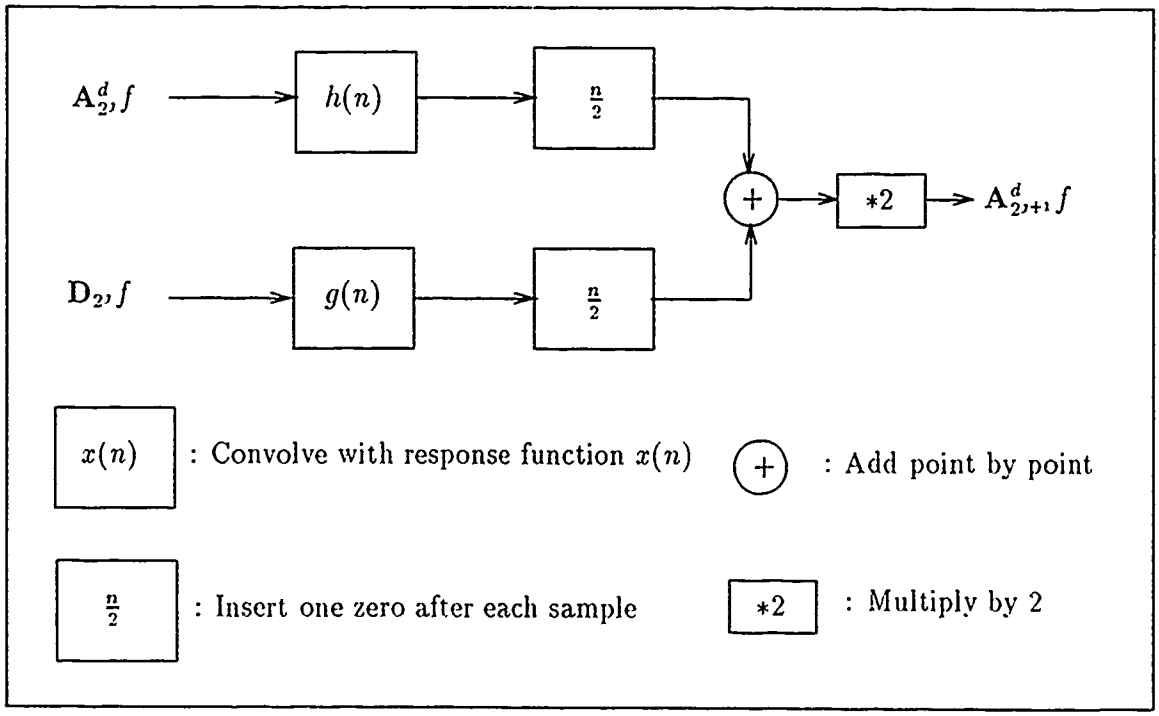


Figure 36. One Dimensional Multiresolution Reconstruction [28:682]

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 37 diagrams the algorithm given by the pyramidal transforms of Equations 64 through 67. Figure 37 is Mallat's Figure 12 [28:685] redrawn and corrected.

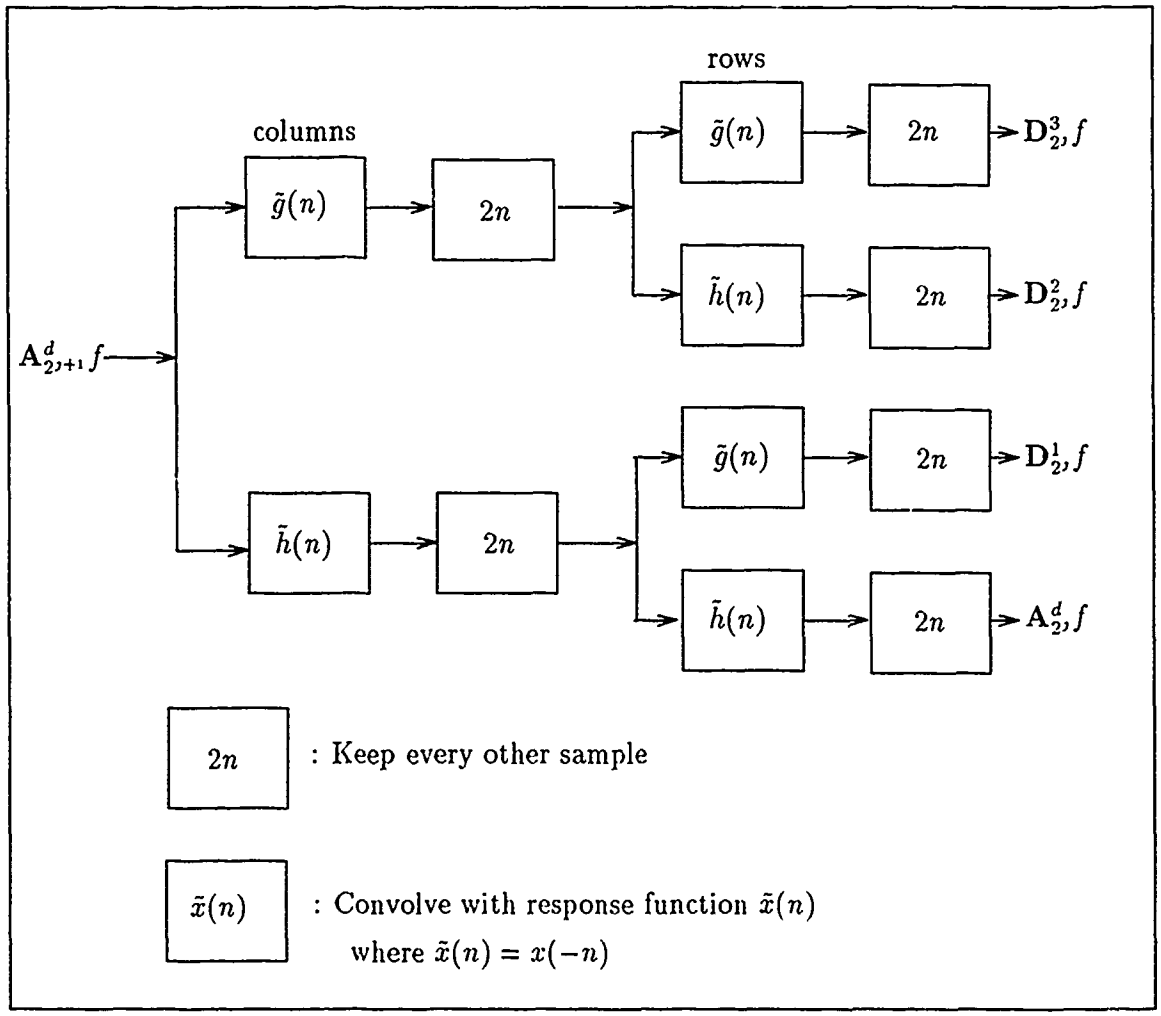


Figure 37. Two Dimensional Multiresolution Decomposition [28:685]

4.3.3 Multiresolution Reconstruction In his Equation (32) [28:682], Mallat shows that the scale coefficients at any level $j+1$ can be reconstructed from the scale and detail coeffi-

cients from the adjacent level j. Our interpretation of this equation is

$$\mathbf{A}^d_{2^{j+1}} f = 2((\mathbf{A}^d_{2^j} f)(\frac{k}{2}) * h(k))(n) + 2((\mathbf{D}_{2^j} f)(\frac{k}{2}) * g(k))(n) \tag{68}$$

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 [7]. Figure 36 diagrams the algorithm of Equation 68. This figure is redrawn from Figure 7 in [28:682].

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 64 through 67 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 69. For the two dimensional case, the reconstruction equation is:

$$\begin{aligned}
\mathbf{A}^d_{2^{j+1}} f \;=\; & 4((\mathbf{A}^d_{2^j} f)(\frac{k}{2},\frac{l}{2}) * h(k) * h(l))(n,m) + \\
& 4((\mathbf{D}^1_{2^j} f)(\frac{k}{2},\frac{l}{2}) * h(k) * g(l))(n,m) + \\
& 4((\mathbf{D}^2_{2^j} f)(\frac{k}{2},\frac{l}{2}) * g(k) * h(l))(n,m) + \\
& 4((\mathbf{D}^3_{2^j} f)(\frac{k}{2},\frac{l}{2}) * g(k) * g(l))(n,m)
\end{aligned} \tag{69}$$

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 [7]. Figure 38 diagrams equation 69. This figure is adapted from Figure 13 in [28:686].

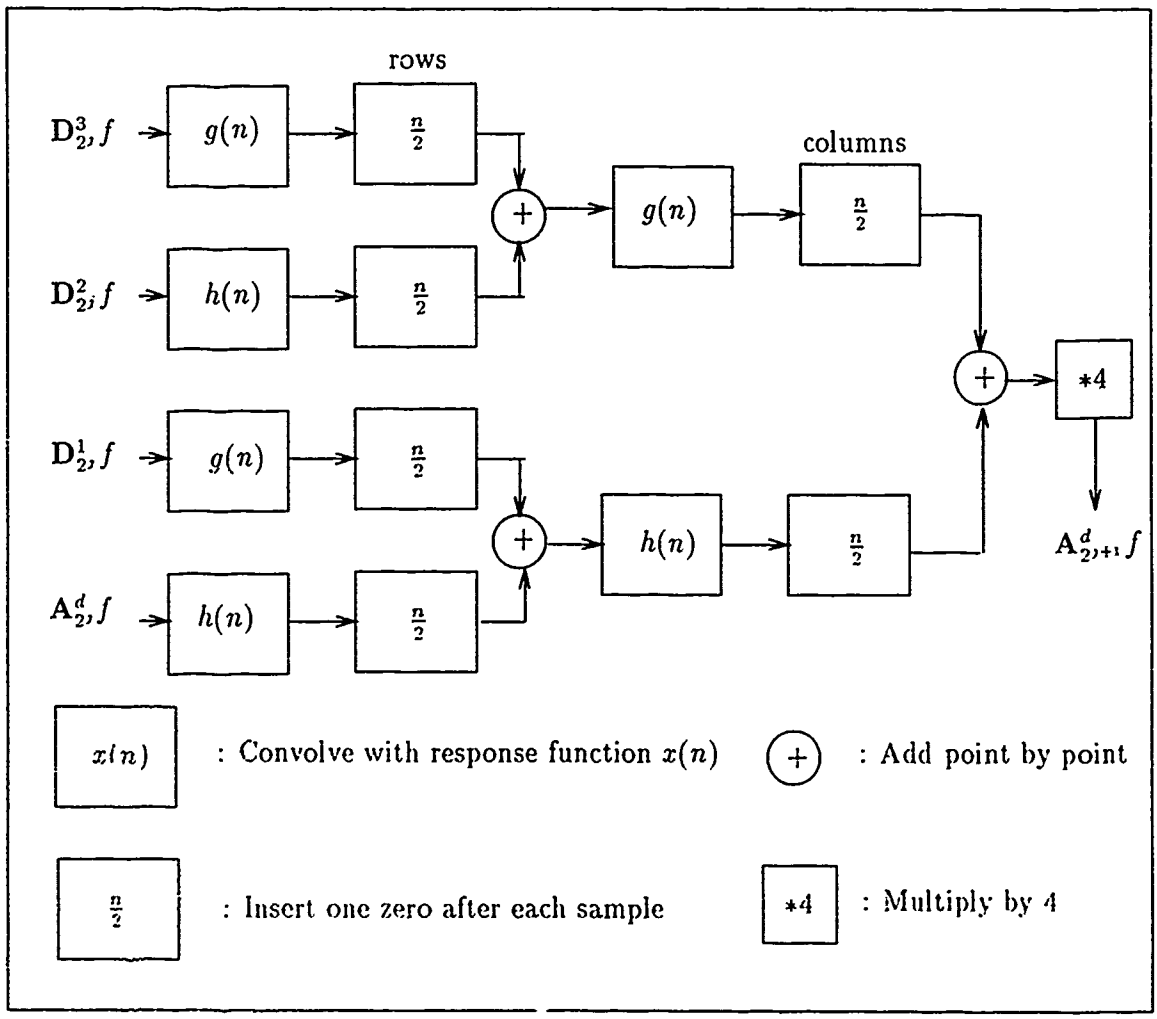


Figure 38. Two Dimensional Multiresolution Reconstruction [28:686]

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 59 and 60 given here in Equation 70 for the one dimensional case and in Equation 71 for the two dimensional case.

$$\mathbf{A}_{2^j} f = ((\mathbf{A}^d_{2^j})(n) * \phi_{-j}(n))(k) \tag{70}$$

$$\mathbf{A}_{2^j} f = ((\mathbf{A}^d_{2^j})(n,m) * \phi_{2^j}(n) * \phi_{2^j}(m))(k,l) \tag{71}$$

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[28:681]. 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 $O(N^2)$ 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 39 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.

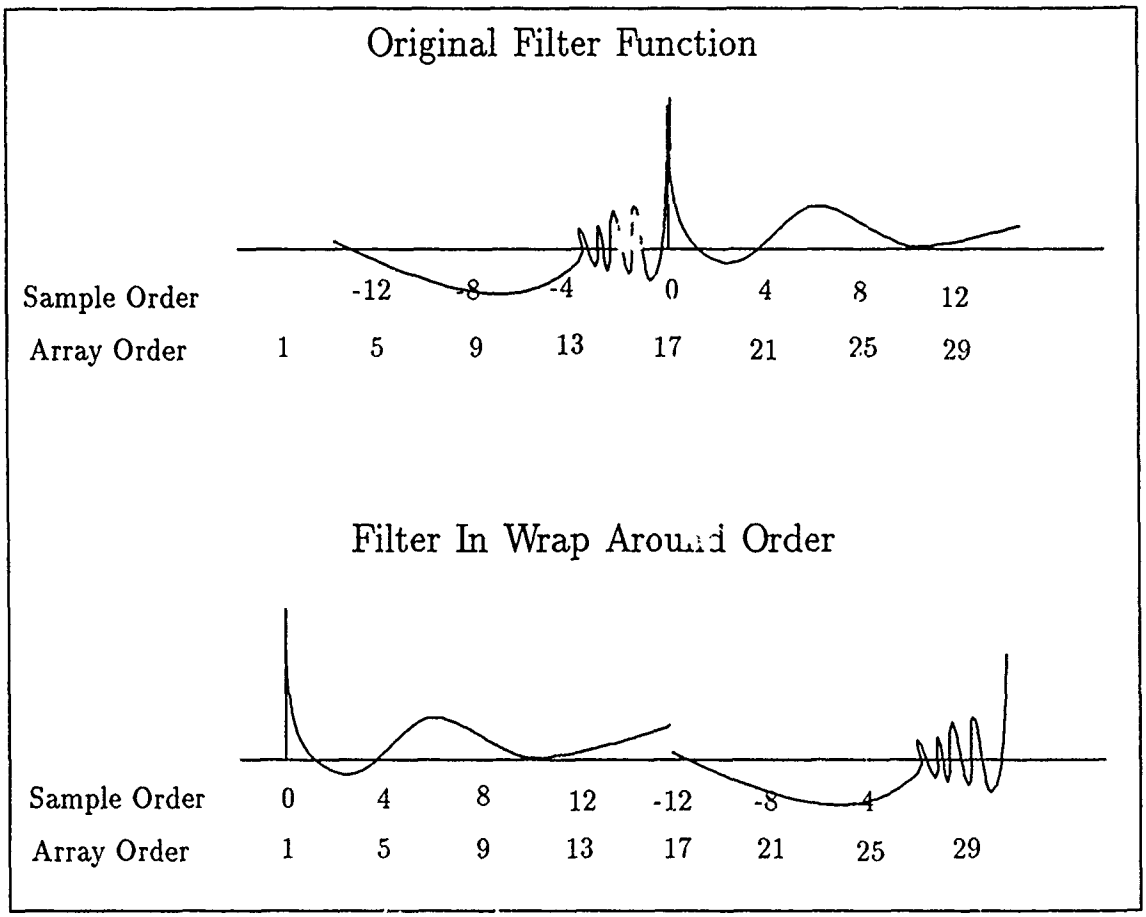


Figure 39. Wrap Around Order for the Convlv.c Procedure

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 [37]. 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 41-43 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. These figures illustrate the edge detection 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 45, 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 13.

Figures 46-52 illustrate the main facets of the multiresolution decomposition and reconstruction process. The original image, 512x512 Lenna, is given in Figure 46 for a comparison with the various results of multiresolution process. Figure 47 is the reconstructed Lenna from a 5 level decomposition. The successively coarser approximations $\mathbf{A}_{2^j}^d$ of Lenna are shown in Figure 48 on the left side of the page. Notice the reduction in size as a result of the down

sampling from the original Lenna Figure 46 (level 0) to the first approximation (level 1) in the upper left corner of Figure 48. The right side of Figure 48 from top to bottom shows the series of reconstructed approximation $\mathbf{A}_{2_j}^d$ of Lenna. The final reconstruction (level 0) is found in Figure 47. The coarsest approximation of Lenna, a 16x16 image, is found in the center of Figure 48. This level 5 approximation along with the detail coefficients found in Figures 49-52 are used to accomplish the reconstruction. Note that these coefficients have been thresholded to make the orientation specific frequency content viewable.

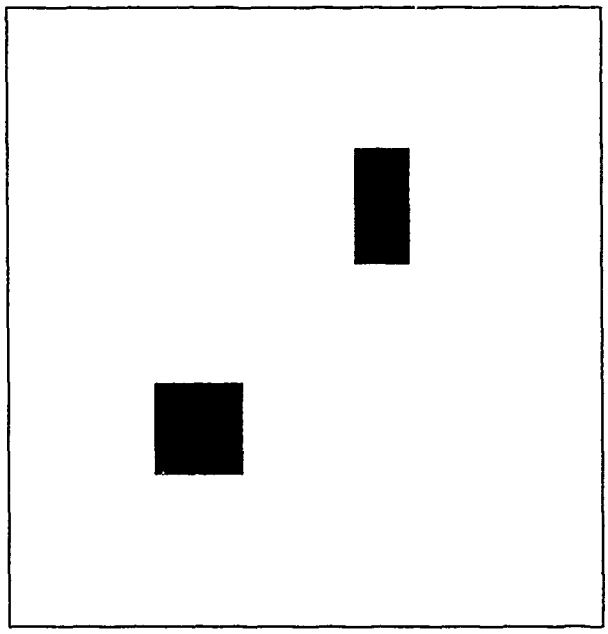


Figure 40. Original Image of Boxes (Reduced 58%)

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

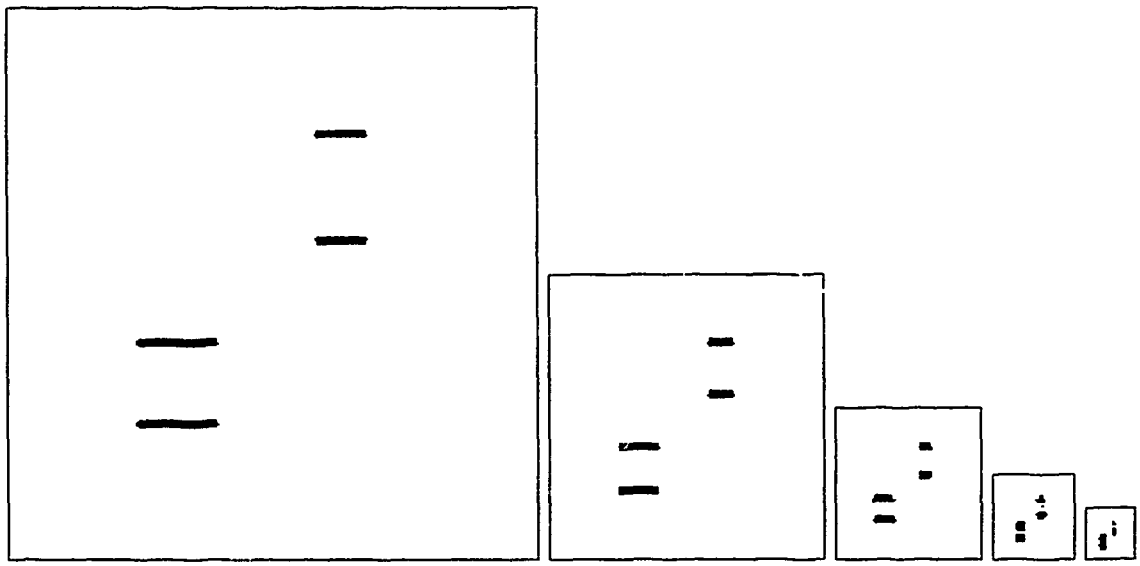


Figure 41. Horizontal Multiresolution Detail Coefficients of Boxes (Reduced 25%)

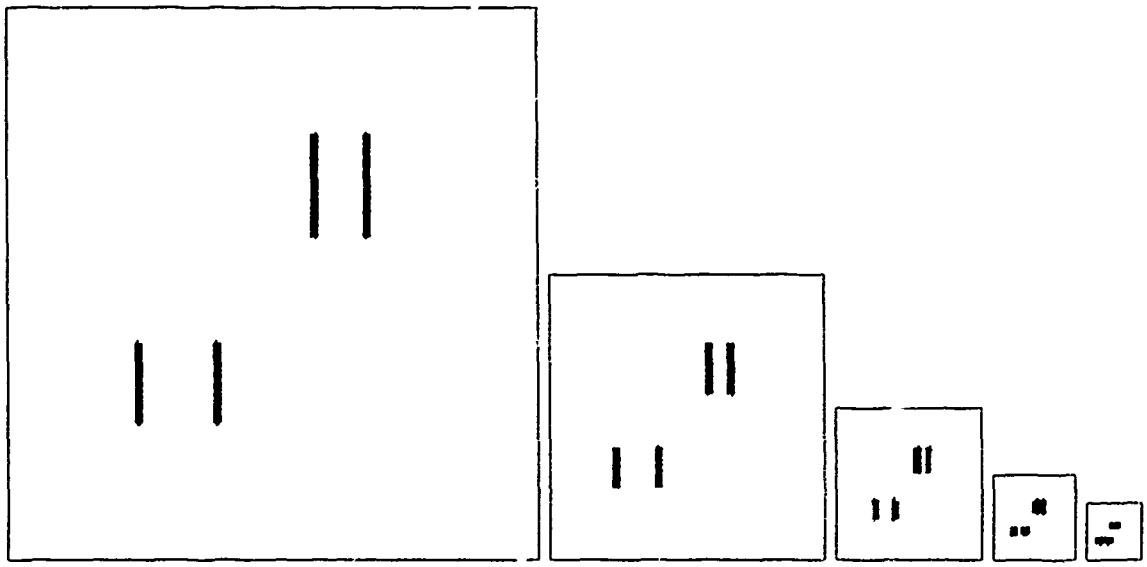


Figure 42. Vertical Multiresolution Detail Coefficients of Boxes (Reduced 25%)

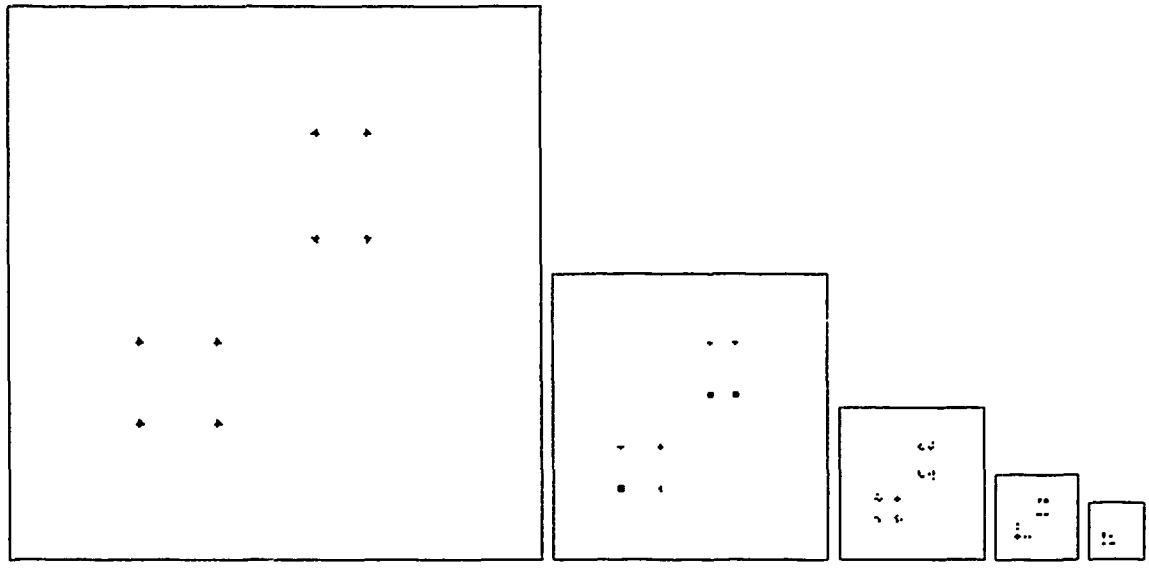


Figure 43. Angular Multiresolution Detail Coefficients of Boxes (Reduced 25%)

Figure 44. Coarsest Approximation of Boxes Used for Reconstruction (Reduced 25%)

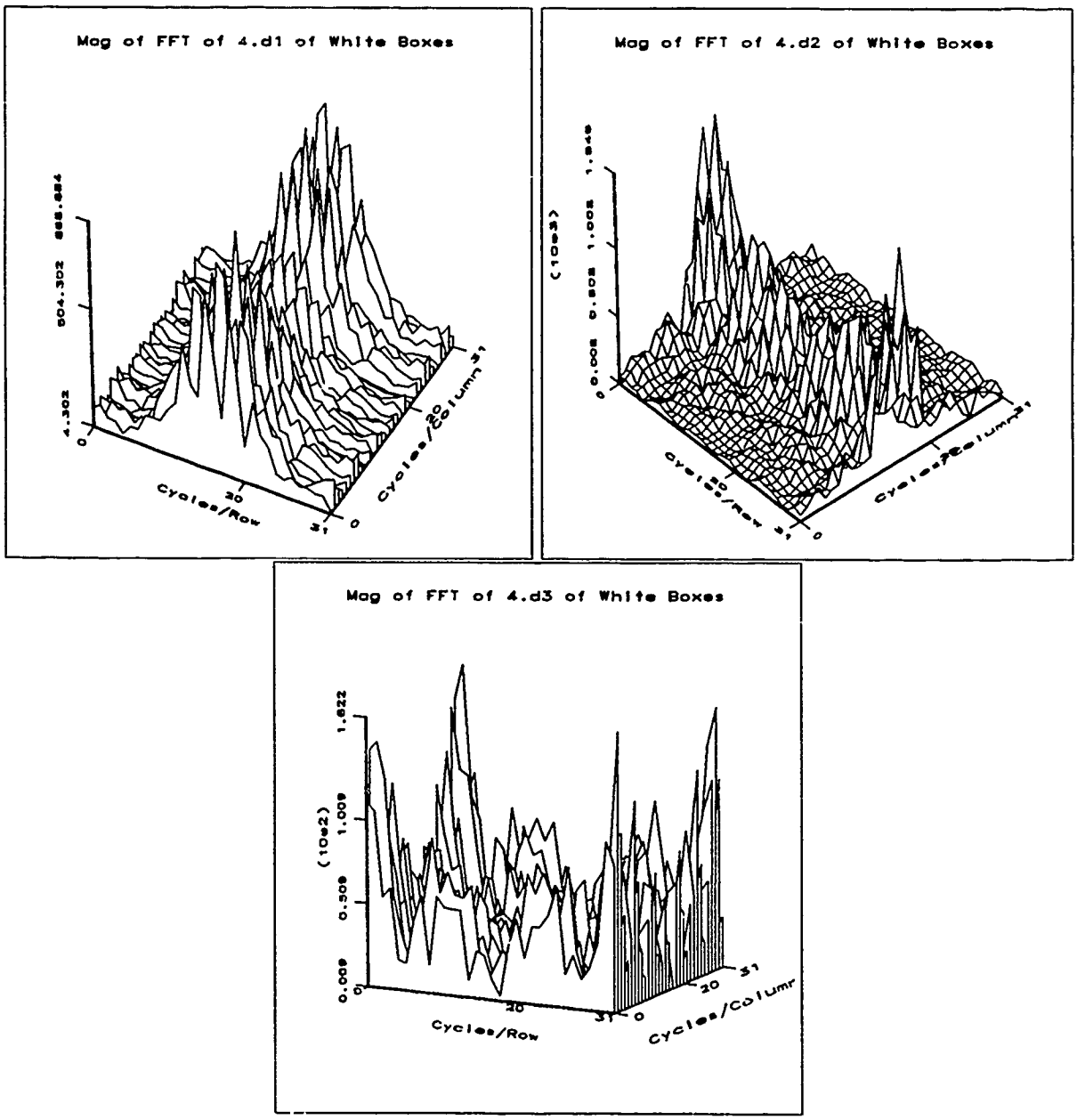


Figure 45. Frequency Support of Detail Signals Of The Cubic Spline Wavelet

Figure 46. Original Image of Lenna (Reduced 2%)

Figure 47. Reconstructed Image of Lenna Using the Spline Wavelet (Reduced 2%)

Figure 48. Multiresolution Decomposition/Reconstruction Approximations of Lenna Using the Cubic Spline Wavelet (Actual Size)

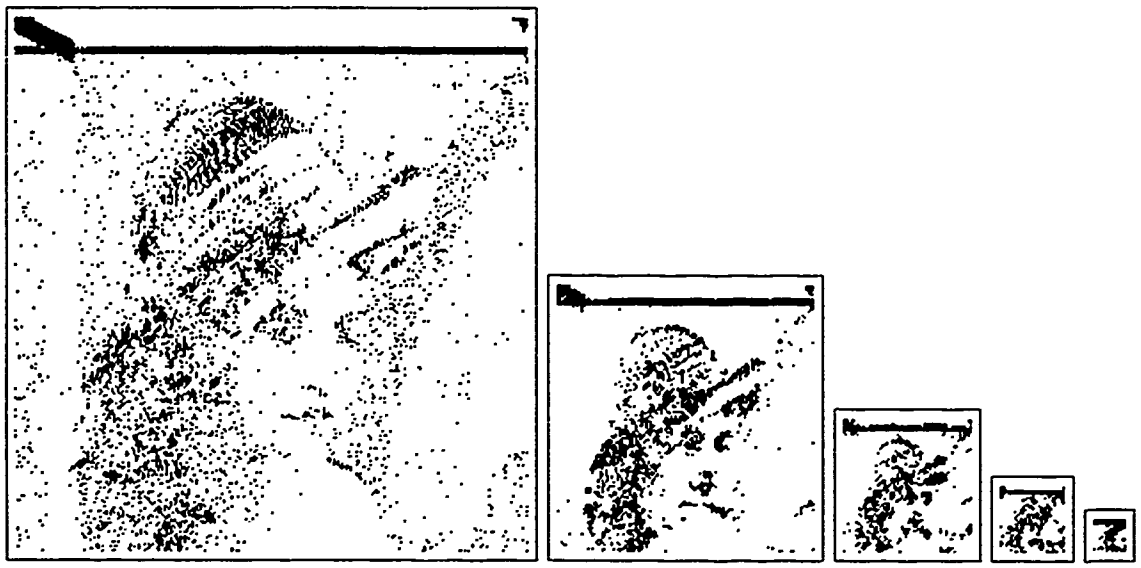


Figure 49. Horizontal Multiresolution Detail Coefficients of Lenna (Reduced 25%)

Figure 50. Vertical Multiresolution Detail Coefficients of Lenna (Reduced 25%)

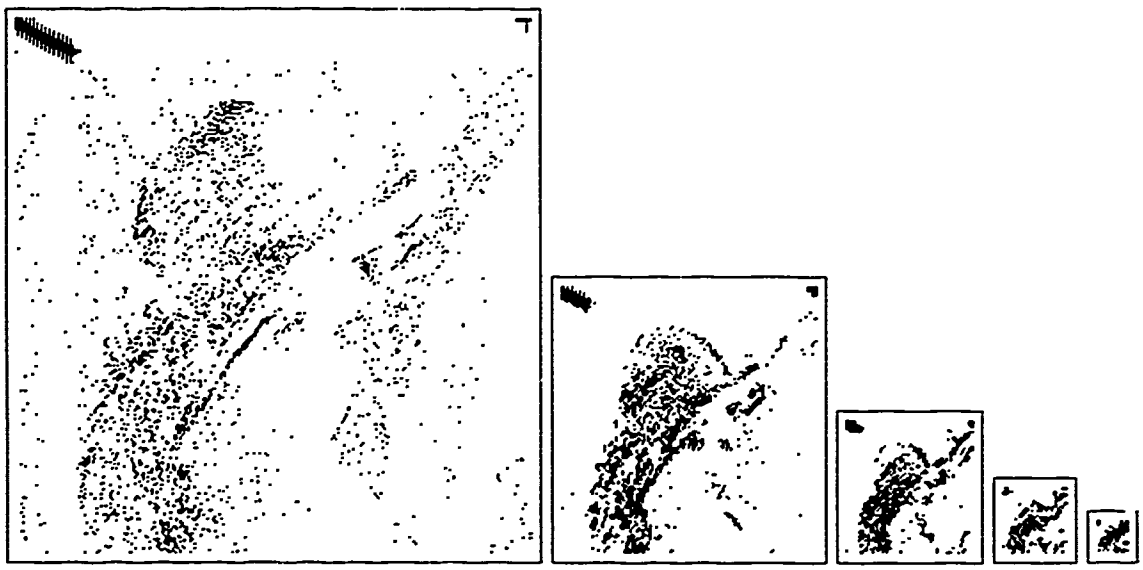


Figure 51. Angular Multiresolution Detail Coefficients of Lenna (Reduced 25%)

Figure 52. Coarsest Approximation of Lenna Needed for Reconstruction (Reduced 25%)

the computational overhead associated with the projection of every set of the decomposed coefficients onto the V^j and W^j 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 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. *Preliminary Results*

With the Multiresolution Wavelet Decomposition at our disposal, we have a tool with which we can analyze an image in a way similar to the works of Ginsburg [14] and Oberndorf [32]. That analysis is the subject of this chapter.

5.1 Review of Multiresolution Wavelet Decomposition

The multiresolution decomposition system breaks the input image out into bands of information each consisting of one octave of spatial frequency on a logarithmic scale. Each band is further decomposed into three sets of wavelet coefficients representing horizontal, vertical, and angular orientations (See Figure 13). The decomposition also produces a set of scaling coefficients for each level of resolution that represents the information contained in all of the lower frequency bands below that level. These scaling coefficients form an approximation of the original input image at successively coarser resolution for lower and lower frequency bands. Each of these coefficients can also be described as the result of subtracting a band of information at all orientations from the scale coefficients that represent one octave finer resolution. Due to the down sampling involved with the decomposition process, the number of subbands or the depth of the decomposition is limited by the size or the interval of support of the filter $h(n)$. For example, using a filter with an interval of support of 23, a 512x512 sampled input image can be decomposed into five levels of resolution where the fifth level of decomposition consists of three sets of wavelet coefficients and one set of scale coefficients each containing 16x16 values. Since the size of the rows and columns is greater than the size of the filter, no further decomposition would be meaningful. The reconstruction reverses the process combining the coarsest level scale coefficients with all band information to rebuild the original image.

5.2 *Methodology*

In the Kanisza Triangle illusion, two anomalies occur: 1. Illusory contours seem to appear forming the distinct impression of a triangle and 2. The relative intensity within the area bounded by the illusory contours appears exaggerated relative to the background intensity of the image [38]. Both Ginsburg and Oberndorf addressed these effects by means of lowpass filtering the image. "It is the energy differences between the various areas ("physical intensity distribution"[13:65]) of the anomaly which suggest how data is being forwarded to the areas of the brain where concept formulation (i.e. object recognition) is taking place."[32:34]. Since the wavelet approximations (scale coefficients) are successively lowpass filtered versions of the original image, it is appropriate to compare one of these approximations to their results. To figure out which approximation contains the proper frequency range, we must characterize the levels of resolution in terms of the spatial-frequencies they contain. In our analysis, the dimension of the original image is 512x512 pixels (see Figure 53). Thus, the sample rate is 512 pixels per object in both the horizontal and vertical directions. Therefore, due to the Nyquist criteria, the highest spatial-frequency detectable in this representation is 256 cycles per object. The process of down sampling by two at each level of the decomposition produces successively coarser approximations of the original image such that the highest spatial-frequency detectable is reduced by one octave on a log scale (reduced by a power of 2). Figure 54 shows the maximum spatial-frequency contained in each level of approximation.

For comparison with Ginsburgs results, Oberndorf chose a lowpass filter containing spatial-frequencies out to 16 cycles/object [32:36]. From the table in Figure 54, we can see that 16 cycles per object corresponds to the fourth approximation. But, this approximation contains 32x32 samples which looks relatively sharp at that level. Therefore, we need to expand it to the scale of the original image (512x512) for comparison. The program *expd* performs this function (see Appendix F.2 for source listing). Given an input filename and the expansion factor, this program expands the image using a two dimensional cubic spline interpolation [35:104]. The results of performing this expansion on the level four approxima-

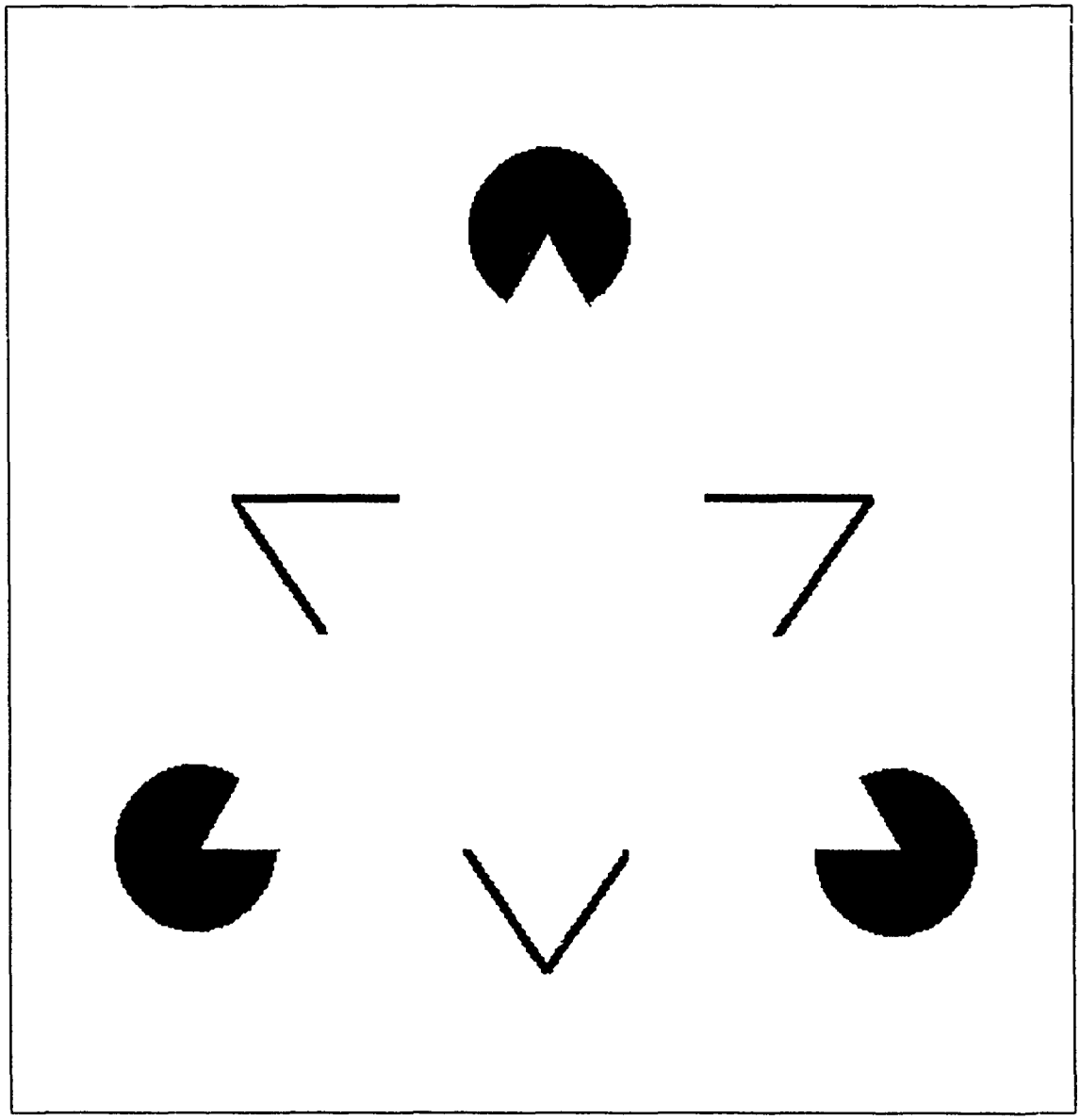


Figure 53. Kanisza Triangle Illusion

Level	Detail Coefficients	Scale Coefficients
	cycles/object	cycles/object
1	128 - 256	0 - 128
2	64 - 128	0 - 64
3	32 - 64	0 - 32
4	16 - 32	0 - 16
5	8 - 16	0 - 8
6	4 - 8	0 - 4
7	2 - 4	0 - 2
8	1 - 2	0 - 1

Figure 54. Relative Spatial-Frequency Range of Each Level of Approximation

tion of the original Kanisza of Figure 53, is depicted in figure 55. While the energy is spread out from the objects in the original image, it does not spread out more in a direction away from the illusory figure as in Oberndorf's results (reproduced in figure 6).

5.3 Conclusion

This comparison leads to the conclusion that Oberndorf's Gabor Lowpass filter possesses characteristics not found in the filtering process of the Multiresolution Wavelet Decomposition. Along this line of reasoning, an obvious difference between the Gabor filtering and the decomposition filtering is the ringing associated with the sharp cutoff of the Gabor filter. It might be this ringing which spreads the energy to help define the illusory contours and the apparent contrast in sensitivity. If this is true, it suggests the spatial filtering process of the brain is also characterized by ringing. On the other hand, it may be that the spatial filtering alone is not enough to cause the perception of the illusion. Rather, it might be that spatial filtering is combined with some other cerebral processing. After all, experimentation has uncovered the functions of only small parts of the cerebral complex.

In the remainder of the thesis, we process the scale coefficients in three biologically motivated ways to analyze the Kanisza Triangle illusion. The first method uses the principle of the saccadic fixation of the human eye. It builds individual frames of the field of view in

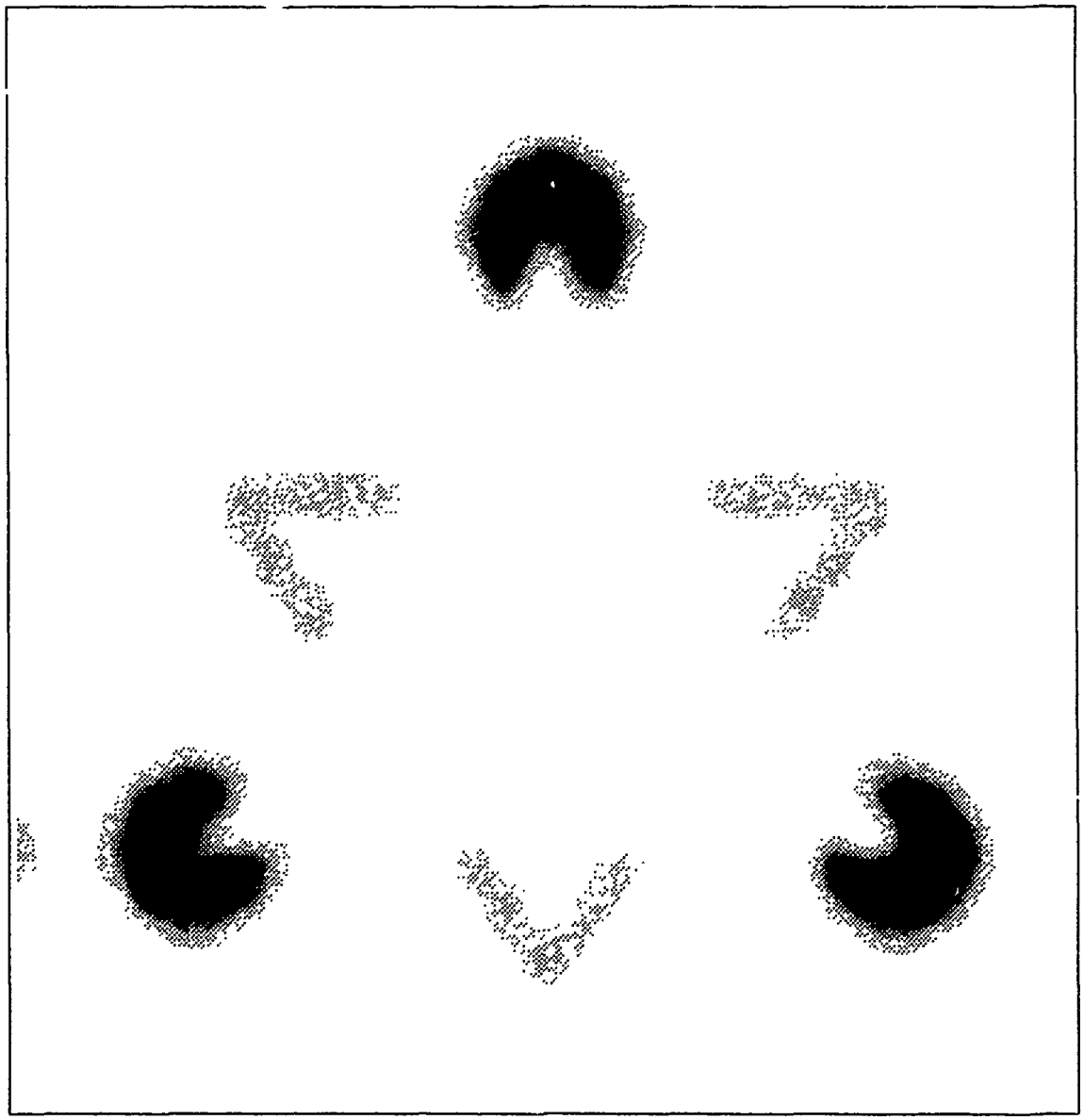


Figure 55. The Kanisza Triangle Approximated at Level Four

which each frame represents a single saccadic fixation. In the second method, we perform a one dimensional Multiresolution Wavelet Decomposition in the time domain in which each set of coefficients generated from the two dimensional decomposition is sampled in time. We use this method to include temporal processing. Finally, the last method uses the scale coefficients as inputs to a Boundary Contour System Model. We use this last method to investigate the local interaction of high spatial-frequency energy.

VI. Building a World Model

The next three chapters each present a unique model of some part of the Human Visual System and the corresponding analysis of visual illusions using that model. In this chapter, we present the first of the three, a model of the human retina, and the results of using the Kanisza Triangle as the input image (see Figure 53). The major component and heart of all three systems is the Multiresolution Wavelet Decomposition which we described in detail in previous chapters. Here, we explain its application as a front end to the model of the human retina.

6.1 Methodology

The human retina contains a random arrangement of photoreceptors called rods and cones. Here, we will consider only the cones for their use in high acuity vision. The density of cones on the retina is higher in the center than at the edges. This distribution of cells has been characterized by an 1894 experiment [39:441] and is redrawn in Figure 56. The figure shows how density distribution causes the receptive field of the retina to sample a stimulus image with high acuity or resolution at the center and progressively lower resolution toward the perephery. We use the different levels of scale coefficients from the Multiresolution Wavelet Decomposition to simulate this resolution distribution. The retina based model constructs an image with high resolution only in the center and successively coarser resolution toward the perephery by strategically placing the scale coefficients of the appropriate resolution. Figure 58 shows how each set of coefficients is located on the constructed image emulating a single fixation of the retina at the center of the image. The program *fbuild* builds the individual frames. A flow diagram for an example input of four levels of resolution is shown in Figure 59. The source code for *fbuild* is contained in Appendix C.2. To create this figure, we first decomposed the Kanisza Triangle with the Multiresolution Wavelet Decomposition program *wave2* (see Chapter IV). Then, to simulate a single fixation of the retina, *fbuild* performs three basic functions: 1. It extracts the appropriate subset of coefficients from the

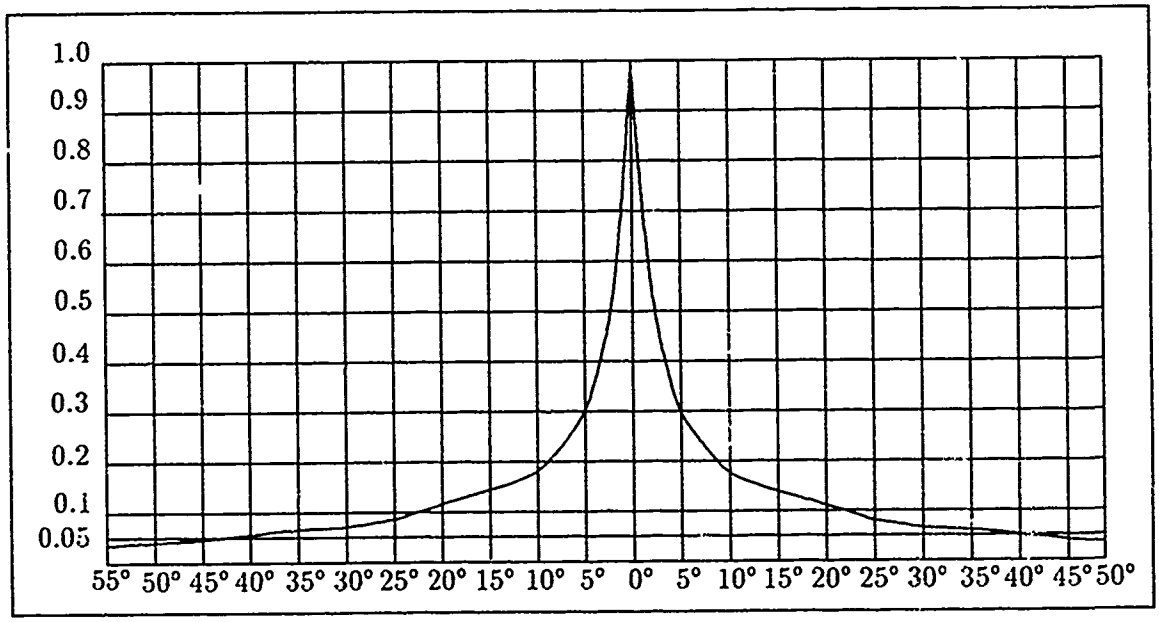


Figure 56. Relative Acuity of Vision Curve [39:441]

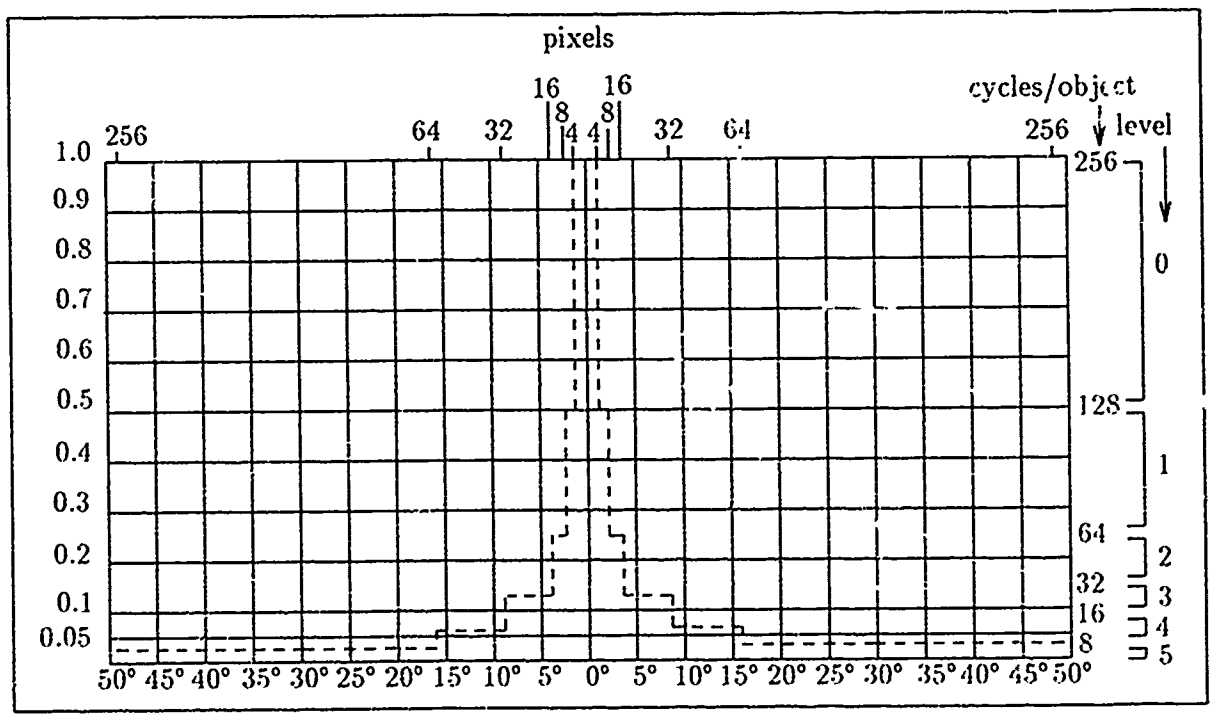


Figure 57. Artificial Relative Acuity of Vision for Model

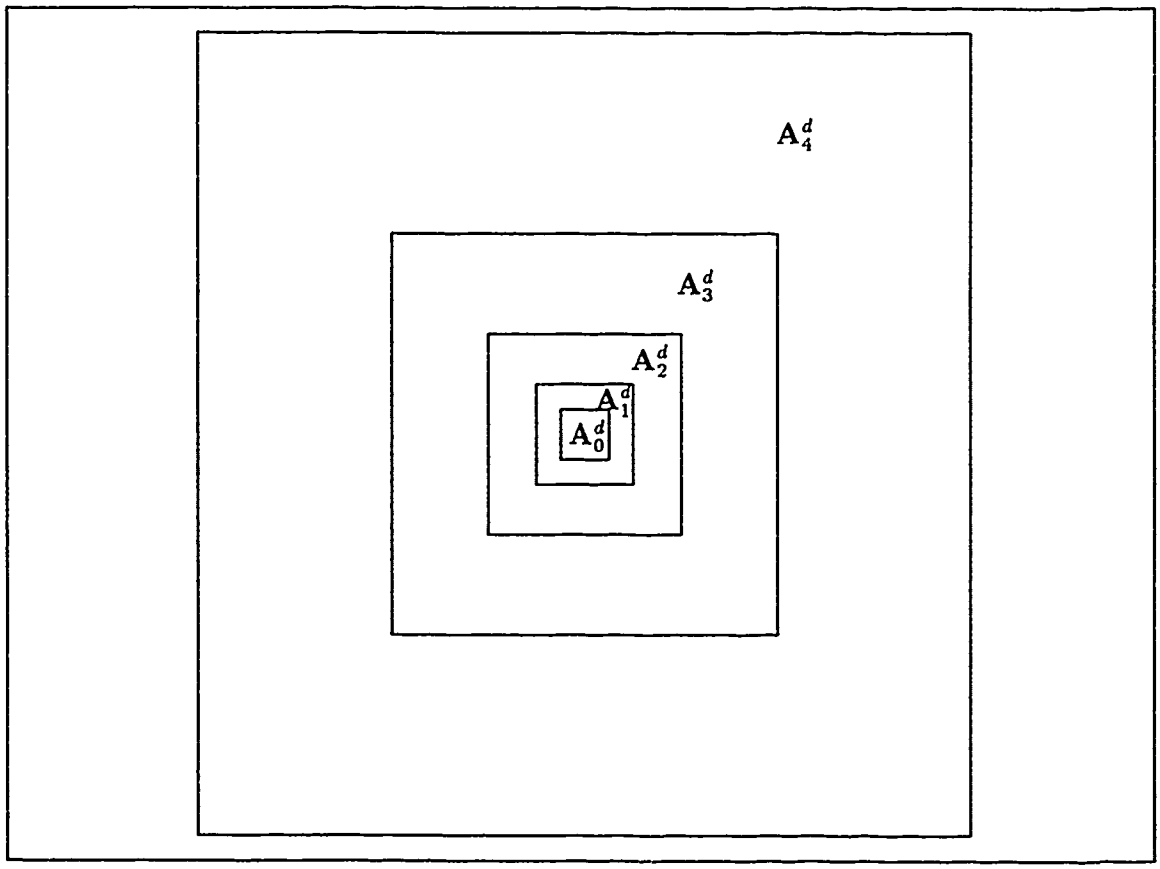


Figure 58. Multiresolution Fixation Map

scale coefficients at each level of resolution, 2. It expands each extraction to the scale of the original image using a two dimensional cubic spline interpolation [35:194], and 3. It places the expanded exserts in the proper location in the constructed simulation. Figure 57 shows the artificial relative acuity emulated with this method. The piecewise constant shape of this curve causes the blocky appearance in Figure 60. We translate degrees of the perceptive field in Figure 56 to the sample space of the 512x512 sampled image by choosing the viewing distance to be 3.18 inches. Therefore, one degree of visual angle is equivalent to four pixels across the image. While the density distribution of Figure 57 is not an exact match with that of Figure 56, it provides roughly the same exponential shape. The myopic view of the image provided by Figure 61 demonstrates the first step in building a perceptual model of the image in which the entire picture seems to be in focus. The composite high resolution

model is due to the saccadic movements of the human eye in which the eye fixates on one highlight of the visual field after another in rapid succession [20:821]. With each movement, the photoreceptors sample the visual field forming a series of myopic frames similar to the one shown in Figure 61. Figure 61 is an example of the output of this program. Because the brain supresses inputs from the visual field during the saccades, the observer is unaware of the movement. The effect of this biological process is to create the perception of a high resolution "motion picture" of the field of vision.

We emulate this perceptual model with a series of these myopic frames assuming that the brain provides a short-term memory to maintain a certain number of these frames which compose the world model at a perceptual instant in time. Therefore, we have chosen a finite number of frames to represent the composite image. Figure 60 shows a composite set of frames which have been combined in one image for static presentation. For simplicity, we do not incorporate other normally imperceptable eye movements such as 1. The "continuous tremor" at 30 to 80 cycles per second due to successive muscular contractions which serves to increase the overall resolution of the visual perception, 2. The "slow drift" of the eyes, and 3. The "flicking" movements that recenter the point of fixation in the receptive field after the "slow drift" has taken place [20:820]. To limit the total number of fixations or frames in our composite image, we consider that the eye needs only fixate where the high spatial frequency information is located in the stimulus image[1]. Therefore, a manually generated set of fixations locations based on the locations of most likley high spatial frequency energy contribution to the precept are used to generate a series of frames with the *fbuild* program. If all the frames are taken together, the result is a composite image as in Figure 60.

6.2 Conclusion

The observation that led to this approach is that the illusion does not appear when the viewer forces fixation at one point in the image; thus, eliminating the saccades of the eye.

[1]This principle has been used by Mallat and others for image coding and compression applications. [27, 26, 1, 42]

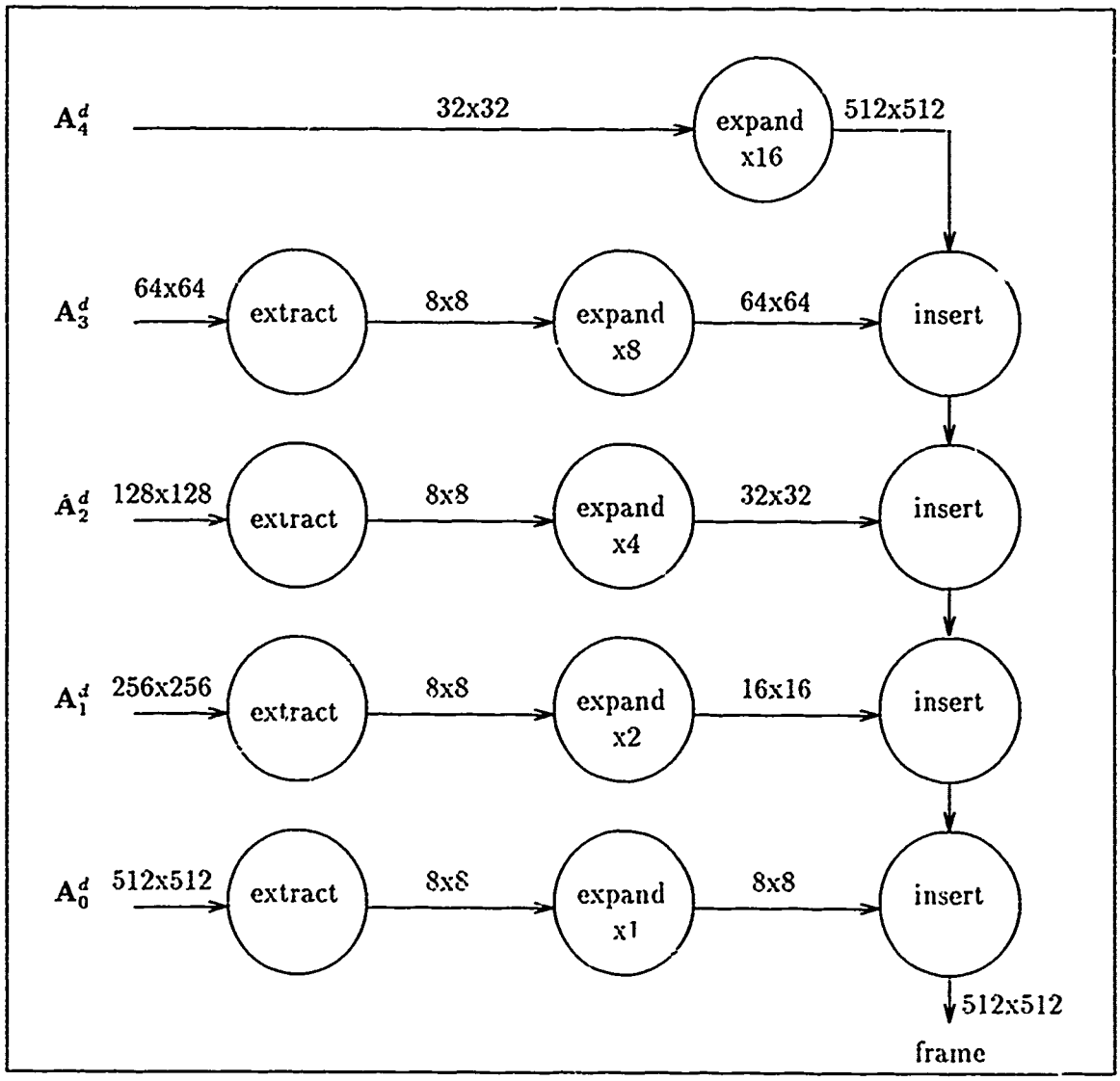


Figure 59. Data Flow of the *fbuild* Program for a 512x512 Input Image

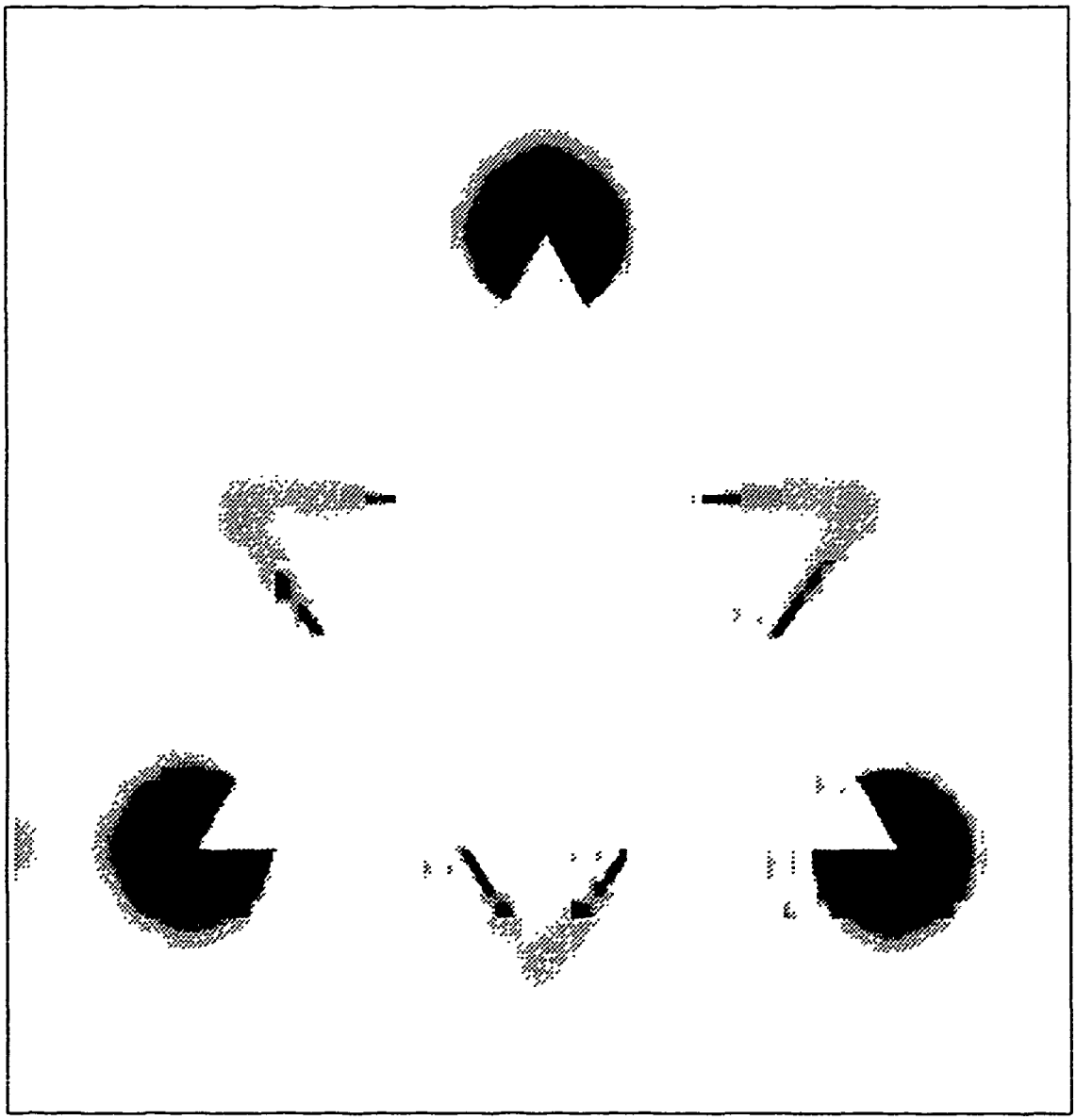


Figure 60. Composite Representation Including 34 Fixation Points

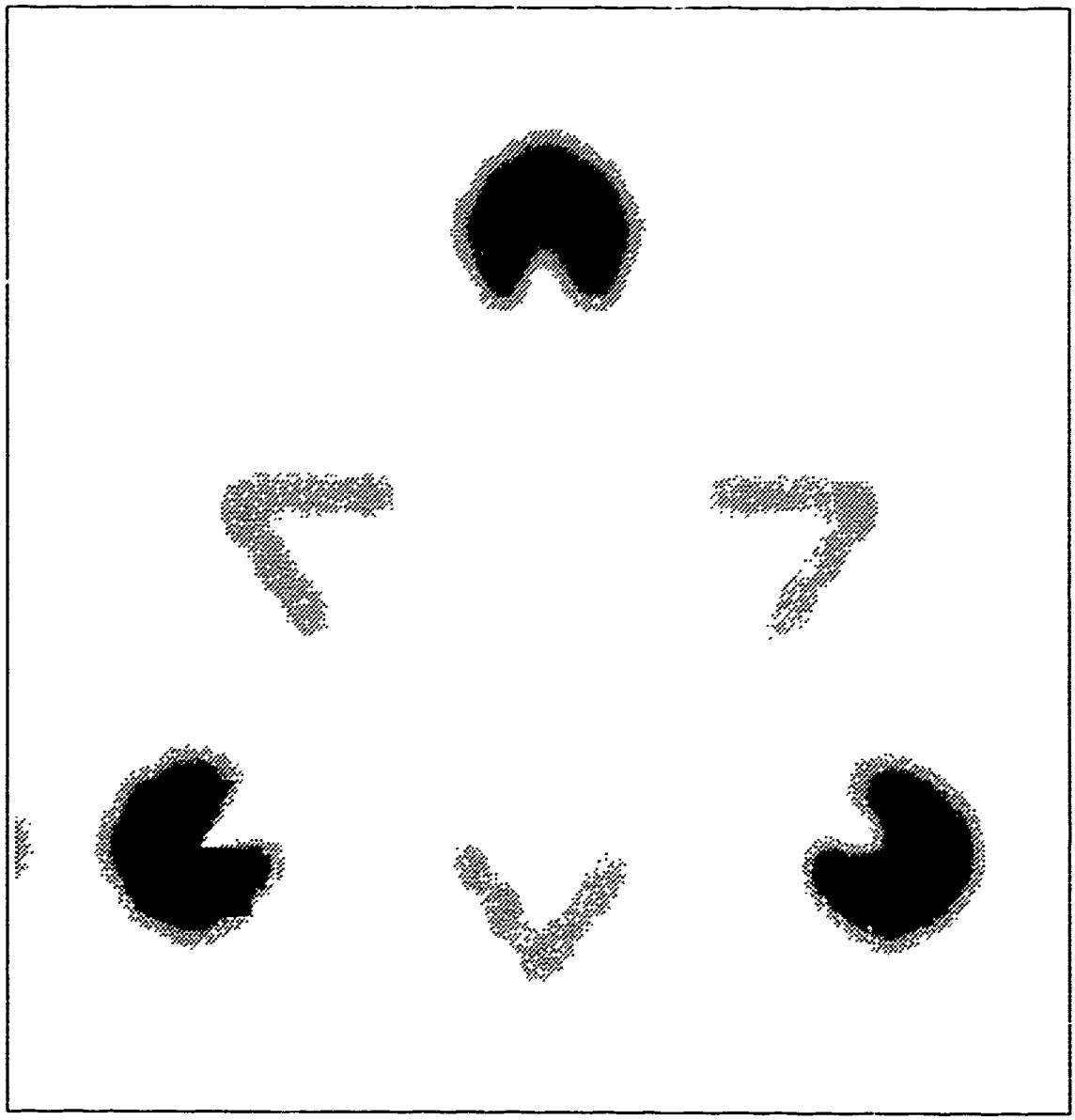


Figure 61. One Myopic Fixation on the Lower Left Packman of the Kanisza Triangle

This suggests that the saccadic movement is necessary for the formulation of the illusion. Compared to the previous results of Figure 55, the low spatial-frequency spreading in Figure 60 is found now only in the areas not replaced by high resolution information. While this does not discount the importance of the "ringing" in the Ginsburg and Oberndorf results, it does emphasize the necessity of incorporating the saccadic movement as a contributor to the perceptual formulation of the illusion. Moreover, the failure of the Ginsburg and Oberndorf results to cause the suggestive contours to appear distinctly, leads us to conclude that high spatial-frequency energy local to the contours is required for the perception. Indeed, our frame built model illustrates how the quality of the illusion is enhanced by adding high spatial-frequency information along the suggestive contours. The reader may wonder how many fixation points are necessary to produce an illusion of "good quality". Figure 62 illustrates the dramatic effect of using only 15 different fixation points along the suggestive contours. We believe these results demonstrate the correctness of considering the high spatial-frequency information local to the contours, which is the goal of our Boundary Contour Model described in Chapter VIII. An additional implication is that if the location and rate of fixation is controllable, it may be possible to eliminate the illusion by drawing the attention of the viewer away from the illusory contours. To test this hypothesis, one would need to build a version of the composite image using fixation points that are outside the locality of the contributory objects. The result would be overwriting more of the low spatial-frequency spreading in effect balancing the energy spread. But the number of fixation points required to produce this effect must be determined. The number would depend in some way on how many points can be "remembered" by the concept forming areas of the brain which is beyond the scope of this thesis. However, considering visual information transmission from the sensing device, retina, to the brain to be a serial process, we can think of the perceptual model as one built up over time. This brings us to the temporal aspect of perception which we explore by first considering each fixation as a frame in time. So as not to incorporate too many variables at once, the next chapter considers spatial filtering and temporal filtering dropping the fixation phenomenon.

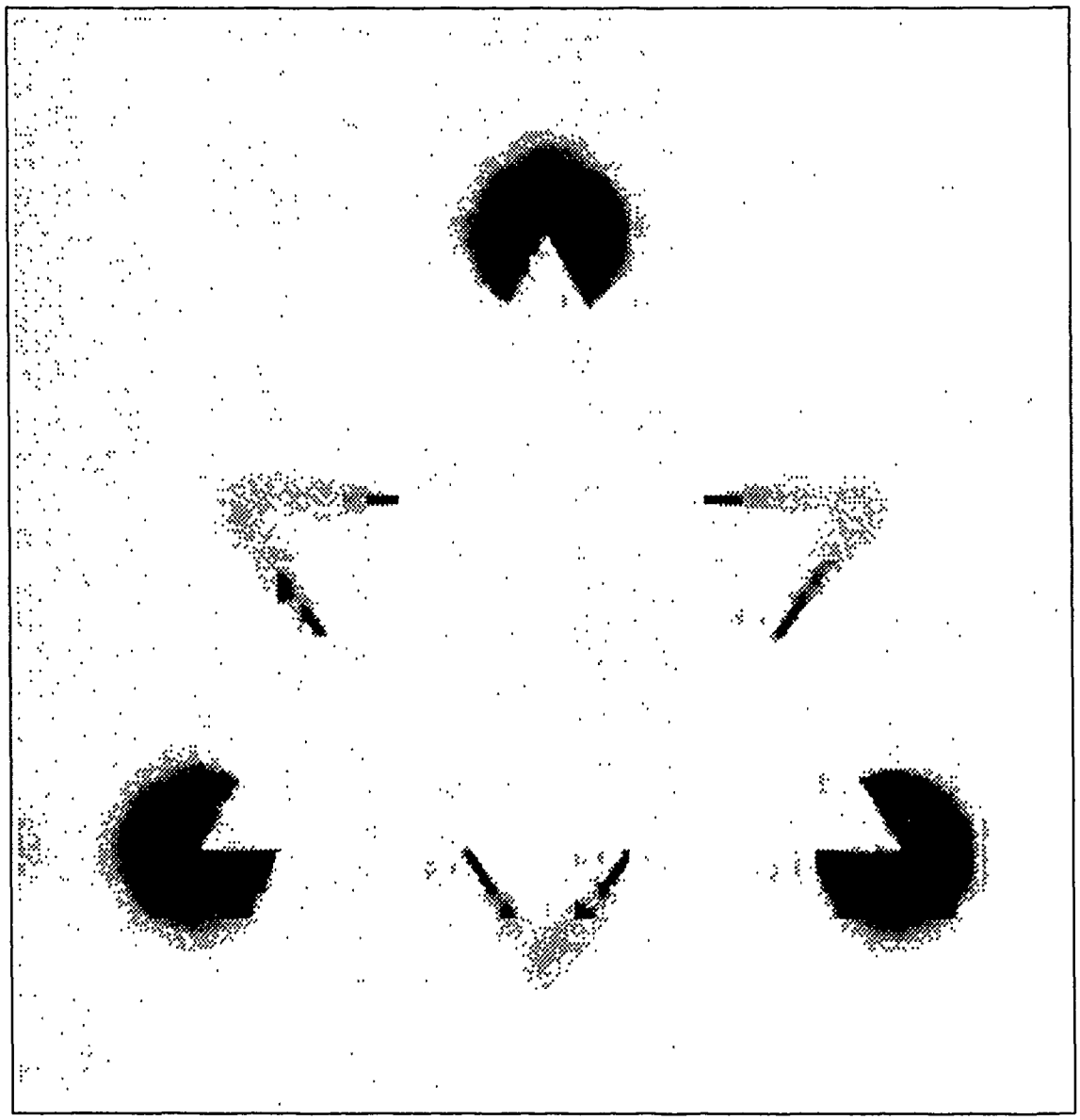


Figure 62. Composite Respresentation Using 15 Fixation Points

VII. A Spatial-Temporal Model

The second method of analysis is designed specifically for spatial-temporal illusions in which the human percept of temporally and spatially discrete stimulation is continuous motion. A popular example of this type of illusion is the "Phi Phenomenon" in which a small dot is illuminated in one location in the perceptive field for a period of time [31:108]. Then, after a specified delay, the dot is again illuminated but in a different location in the perceptive field. If the product of the delay between illuminations and the distance between the two locations is small enough, the dot will appear to move continuously between the two locations. The distance limit can be characterized by a range or bandwidth of spatial-frequencies and the delay can be characterized by a range or bandwidth of temporal-frequencies. Therefore, there exists some fundamental spatial-temporal bandwidth product limit above which the illusion is perceived. The two dimensional Multiresolution Wavelet Decomposition explained in Chapter III, demonstrated in Chapter IV, and used in Chapters V and VI provides the needed spatial-frequency channels to analyze this illusion and a one dimensional version of this tool provides the temporal-frequency channels. With this composite three dimensional method of channel decomposition, we can control the spatial-temporal bandwidth of a displayed image.

For consistency with the two dimensional analysis of Chapters V and VI, the specific illusion used here is an animated version of the Kanisza Triangle devised for this thesis. Figure 64 shows discrete variations of the original Kanisza Triangle. When animated with the *kanmov* program as frames of a motion picture, the suggestive contours appear to displace continuously (see Appendix D.2 for source listing).

7.1 Methodology

Marr recognized that edges in space occur in all octaves of spatial-frequency since they represent discontinuity in space [30]. Lewis and Knowles took this idea one step further implementing it in the time dimension, "Motion or change between frames produces 'edges'

in time" [26:397]. In the same way that filtering our high spatial-frequencies "blurs" an image in space, filtering out high temporal-frequencies "blurs" motion across time. The Multiresolution Wavelet Decomposition approach to performing this "time blurring" constitutes decomposing each pixel's one dimensional time signal (one sample per frame) into successively coarser approximations. Rebuilding the frames from a coarser time approximation has the effect of smoothing out the perceived motion emulating or aiding the perception of continuous motion from discretely changing frames.

Figure 64 shows frames 2, 5, 9, 13, 17, 21, 25, 29, and 32 of the 32 frames used in this analysis. In the interest of computational time complexity we use a 256x256 sampled version

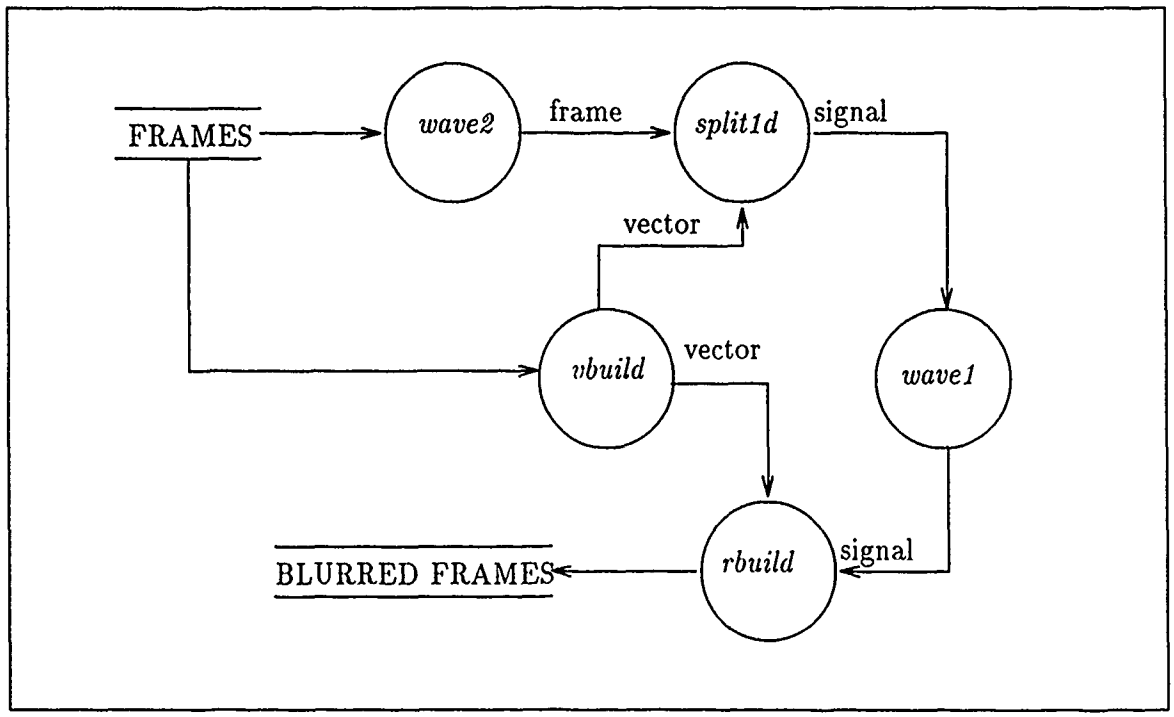


Figure 63. Flow Diagram of the Spatial-Temporal Blurring System

of the original Kanisza Triangle. Each frame is altered such that the illusory contours appear to bend deforming the suggestive triangle. The data flow diagram in Figure 63 shows the individual steps in the three dimensional spatial-temporal decomposition. The process circles represent individual programs, whose source code is listed in Appendix D.2. First, the *wave2*

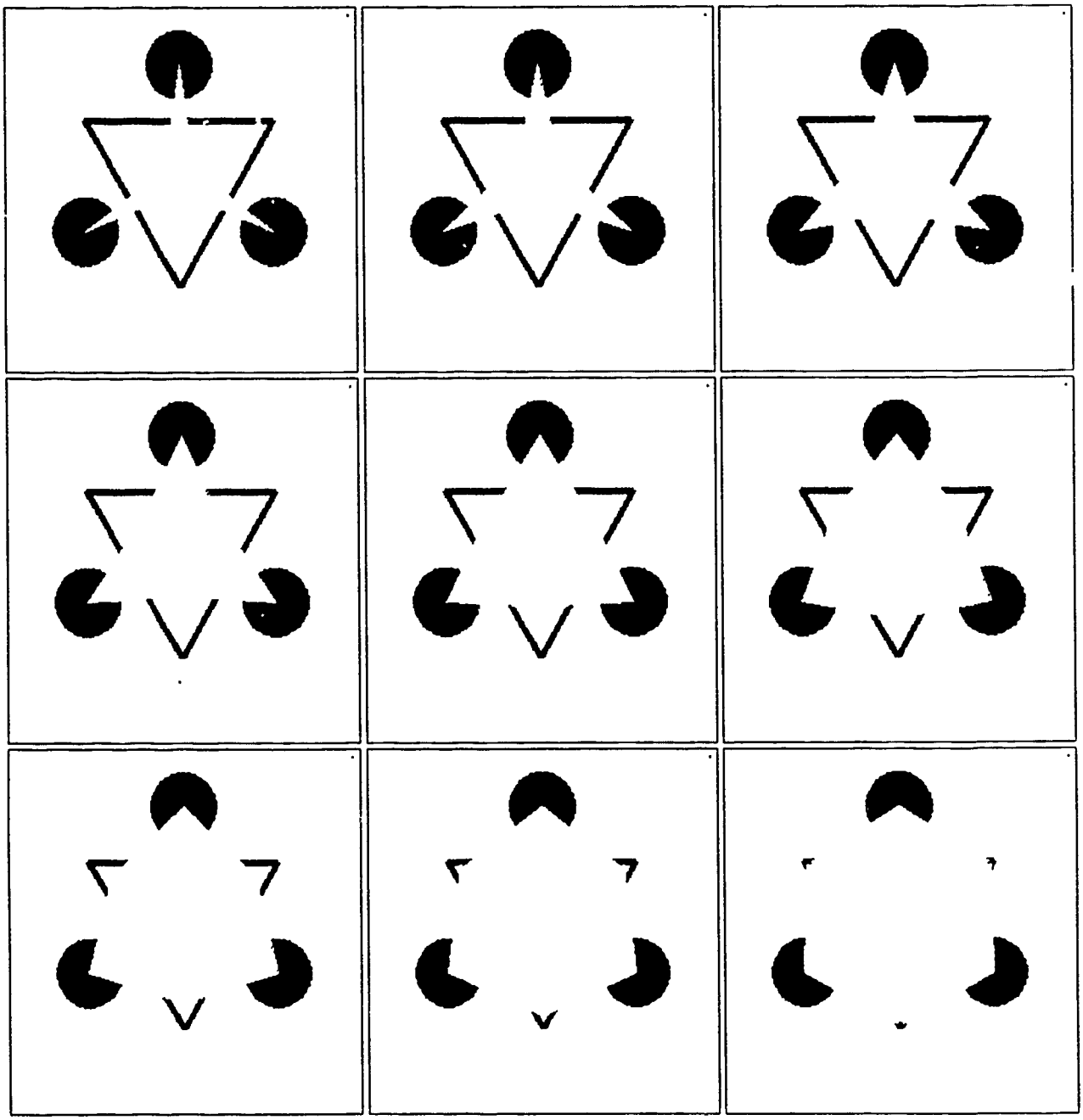


Figure 64. Frames of Moving Kanisza Triangle Illusion

program described in Chapter IV decomposes each frame into several levels of multiresolution approximation. Next, the *vbuild* program reduces the potential number of one dimensional time signals by finding all the pixels which change in value over the 32 frames. The output of this program is a set of vectors which identify these pixels. These vectors are used by the *strip1d* program to build a one dimensional signal for each of the pixels that change. Then the *wave1* program performs a Multiresolution Wavelet Decomposition on each signal. Finally, the *rbuild* program rebuilds the frames given the vectors from the *vbuild* program and a selected combination of spatially decomposed frames and temporally decomposed signals output from the *wave2* and *wave1* programs respectively. The *wave1* program used here is a modified version of the *wave1* program described in Chapter IV. The original source code is listed in Appendix B.4 and the modified portions are listed in Appendix D.2. The *rbuild* program expands the time decomposed time signals to the original 32 samples using a cubic spline interpolation [35:104].

For consistency with the 2D Wavelet decompositions used in Chapters V and VI, we use the cubic spline wavelet for spatial decomposition. However, in the time domain, due to its size (32 samples), the filters corresponding to the cubic spline wavelet would limit us to only one meaningful level of resolution in the decomposition. Therefore, for the decomposition in time, we use a Daubechies 2 wavelet whose corresponding filters have four values each. These choices allow decomposition to three levels of resolution in time and four levels of resolution in space. Thus, there are 20 possible combinations of spatial-temporally blurred sets of frames, levels zero through four in space and zero through three in time. Figure 65 shows the nine frames of Figure 64 using the original frames in space and the first level approximation in time. Figure 66 shows the same frames with the original frames in space and the second level approximation in time. Figure 67 shows the original frames in space with the third level in time. Finally, Figure 68 represents the same frames from the fourth level approximation in space and the third level approximation in time. In this last result, the fourth level spatial approximation of each frame is expanded to a 256x256 sample scale with

the *expd* program (see Appendix F.2 for source listing). Figure 68 represents the expanded frames for comparison to other figures.

For the purpose of viewing the effects of "time blurring" dynamically, we animated each set of 32 frames on a Silicon Graphics workstation with the *tblur* program.

7.2 Conclusion

The original motivation for performing this analysis was to see if the static illusion persisted when the speed of animation was such that the eye did not have time to saccade enough points on the image to produce the illusion. Assuming the time between saccades to be approximately 100 msec [20], and that only one fixation per object in the image is necessary to form the illusory contours, then since there are seven objects in the image, 700 msec would be required to produce the illusion on each frame. Therefore, using the Silicon Graphics workstation with a frame update rate of approximately 20 frames per second, there is not enough time for the eye to saccade and fixate seven times. But, the animation produced the illusory triangle distinctly suggesting that more than judiciously placed high spatial-frequency processing provided by fixations and overall low spatial-frequency processing is at work to produce the illusion.

In keeping with our charter of investigating the contributions of various frequency specific processing, consider the temporal-frequency information in the series of frames of an animated scene. The brain may process this information in such a way as to suppress high temporal-frequencies in effect smoothing the motion to give the eye more time to fixate and saccade around the scene. Figures 65 through 67 show the effects of progressively reducing the high temporal-frequency content of the scene. The range of spatial variation is somewhat reduced. It might be that this reduction in spatial variation allows the brain more time to process the spatial-frequency information required to produce the illusion. All in all, one observation is consistent across spatial and temporal processing - that is that less high frequency information is required to form the illusory contours. While low spatial-frequency information is required of the entire image to provide the overall form, high spatial-frequency

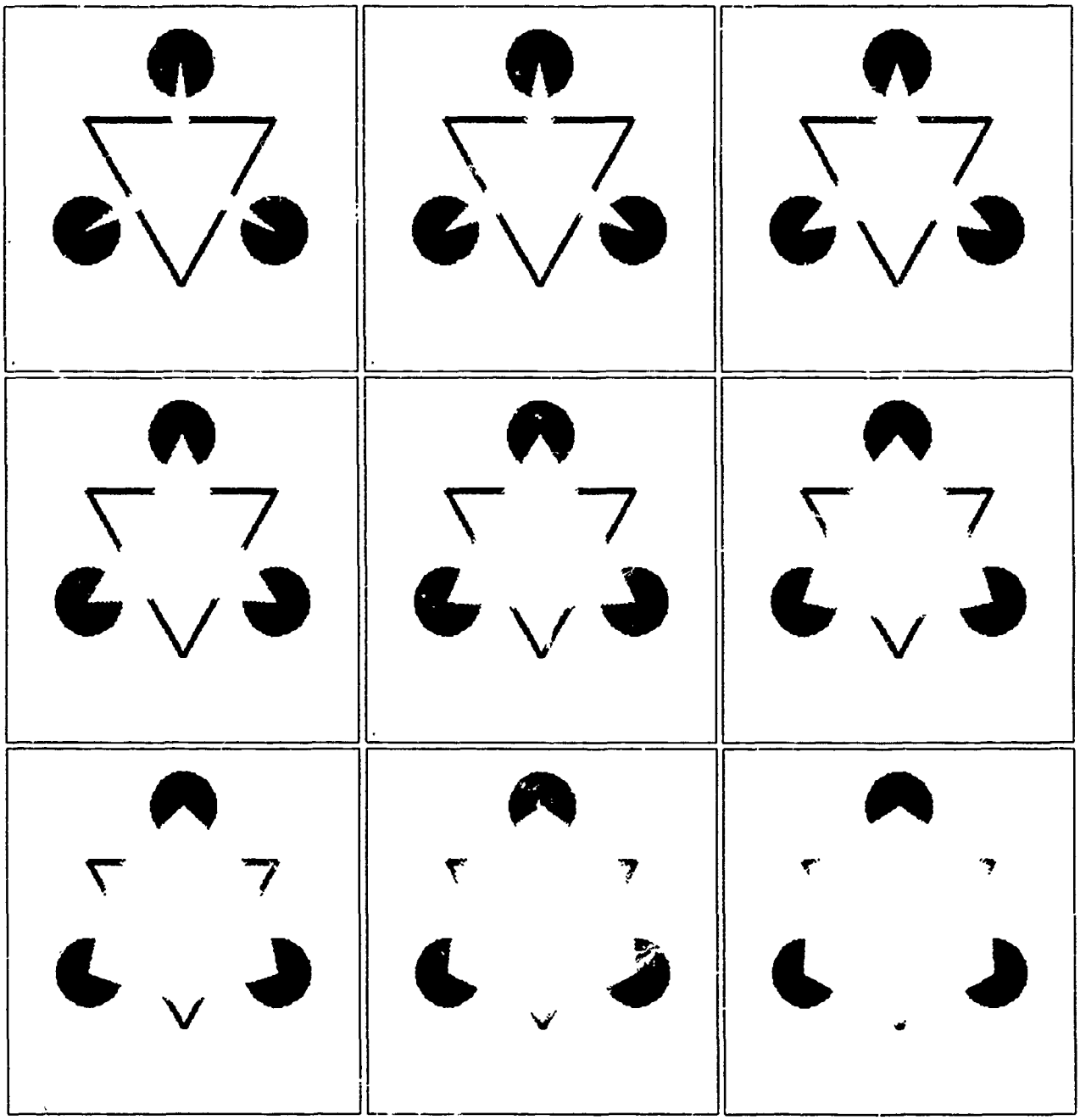


Figure 65. Frames of Kanisza Triangle Using Level 1 Decomposition in Time

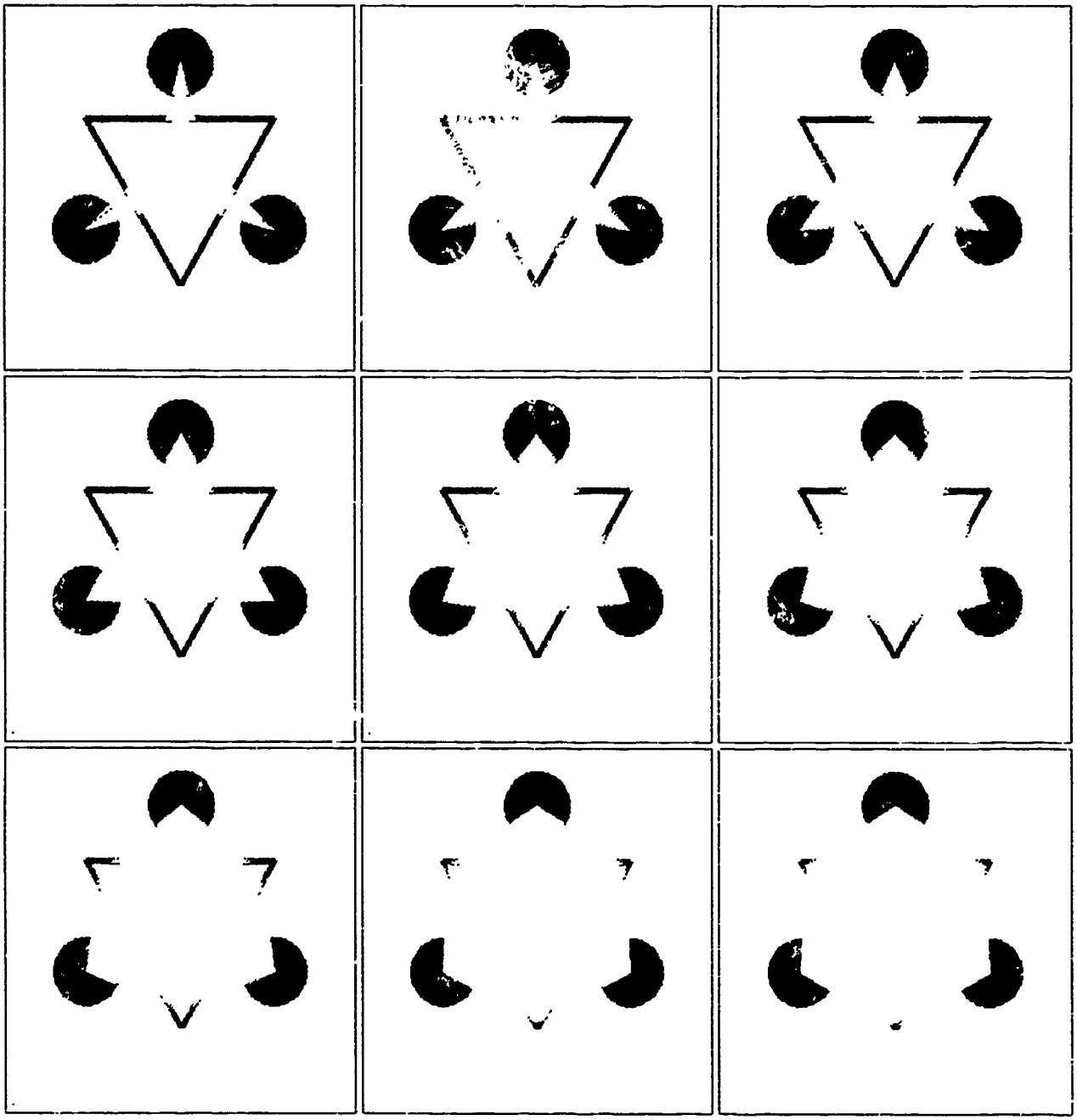


Figure 66. Frames of Kanisza Triangle Using Level 2 Decomposition in Time

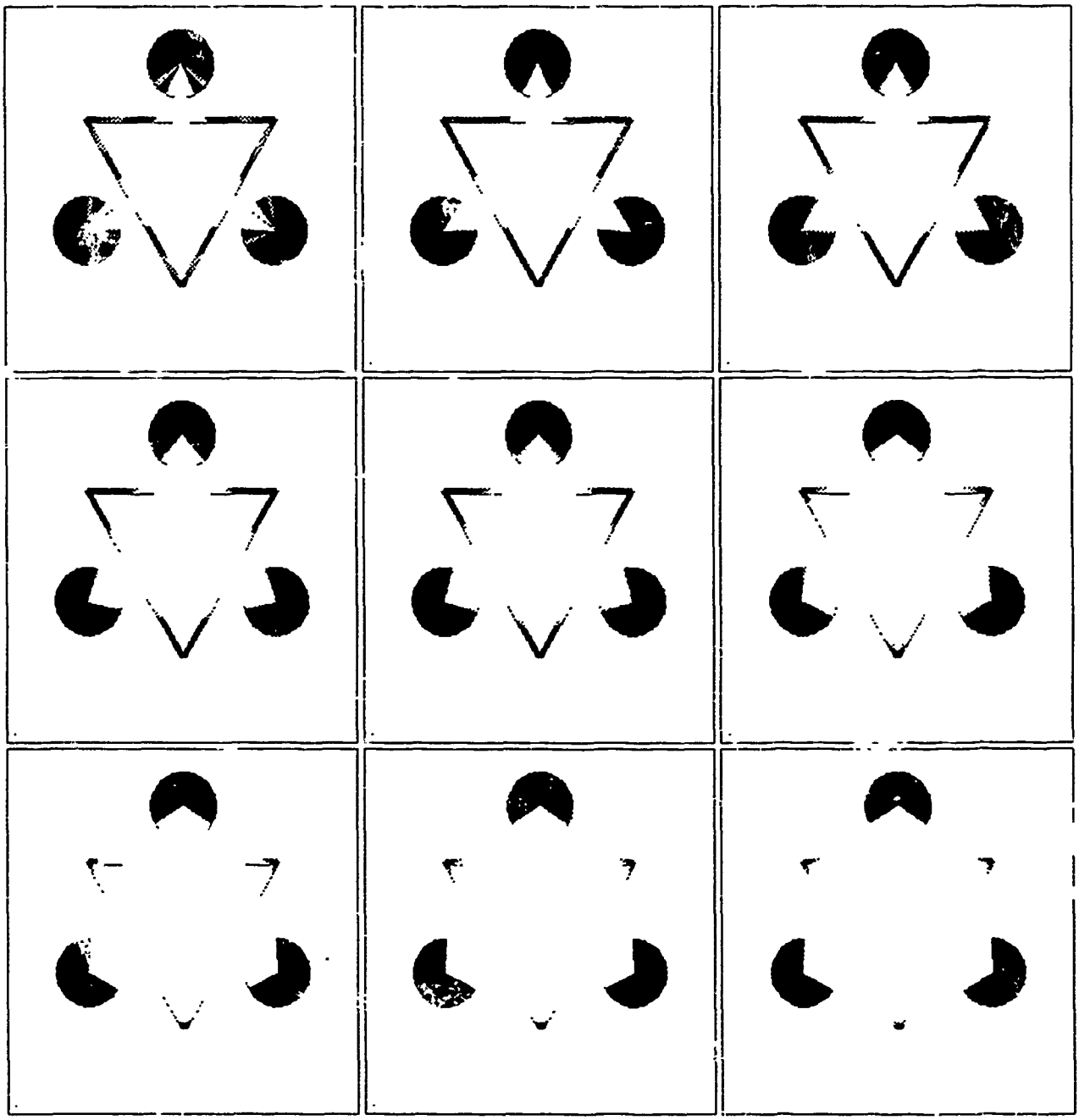


Figure 67. Frames of Kanisza Triangle Using Level 3 Decomposition in Time

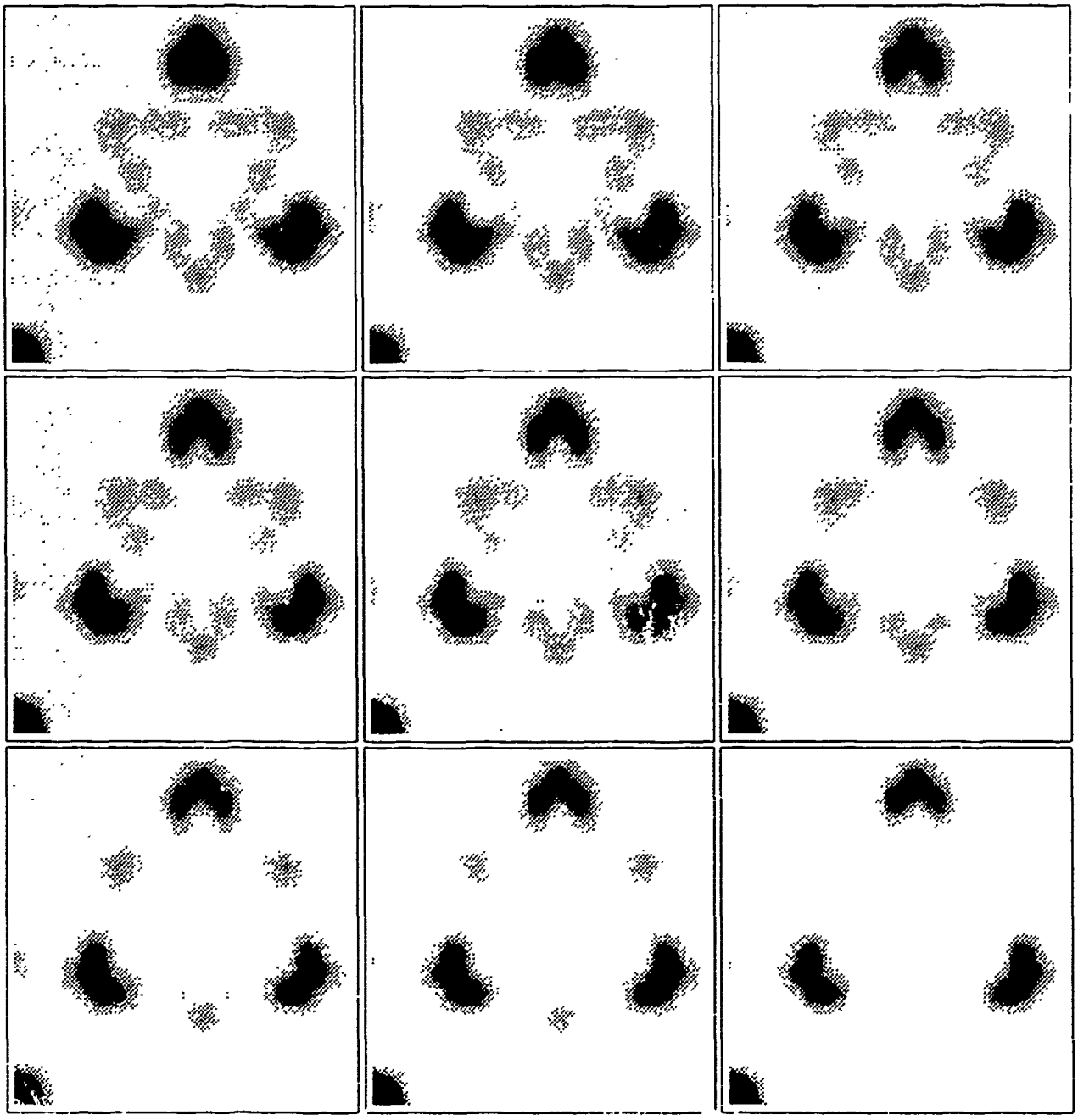


Figure 68. Frames of Kanisza Triangle Using Level 4 in Space and Level 3 in Time

information is only necessary in specific locations of contributory energy around the illusory contours. In much the same way that low temporal-frequency information is required to give the general perception of motion, high temporal-frequency information is required to sharpen that perception. Furthermore, there may be a fundamental trade-off between resolution in time and resolution in space that determines the space/time bandwidth product envelope within which the illusion is perceived [34].

VIII. A Boundary Contour Model

To investigate the importance of high spatial-frequency processing to the perception of illusory contours, we developed a boundary contour model. This model is a simplified version of the first stage of the Grossberg's Boundary Contour System (BCS) [18, 16, 17]. The model described here is designed only to demonstrate the contribution of the multi-orientation, high spatial-frequency output from the Multiresolution Wavelet Decomposition to the perception of illusory contours.

8.1 Methodology

The Boundary Contour Model (BCM) uses as input the detail wavelet coefficients provided by the *wave2* program which performs a Multiresolution Wavelet Decomposition (see Chapter IV). The BCM as illustrated in Figure 69 performs a one dimensional lateral excitation on either the rows or columns of these coefficients. The purpose of this network is

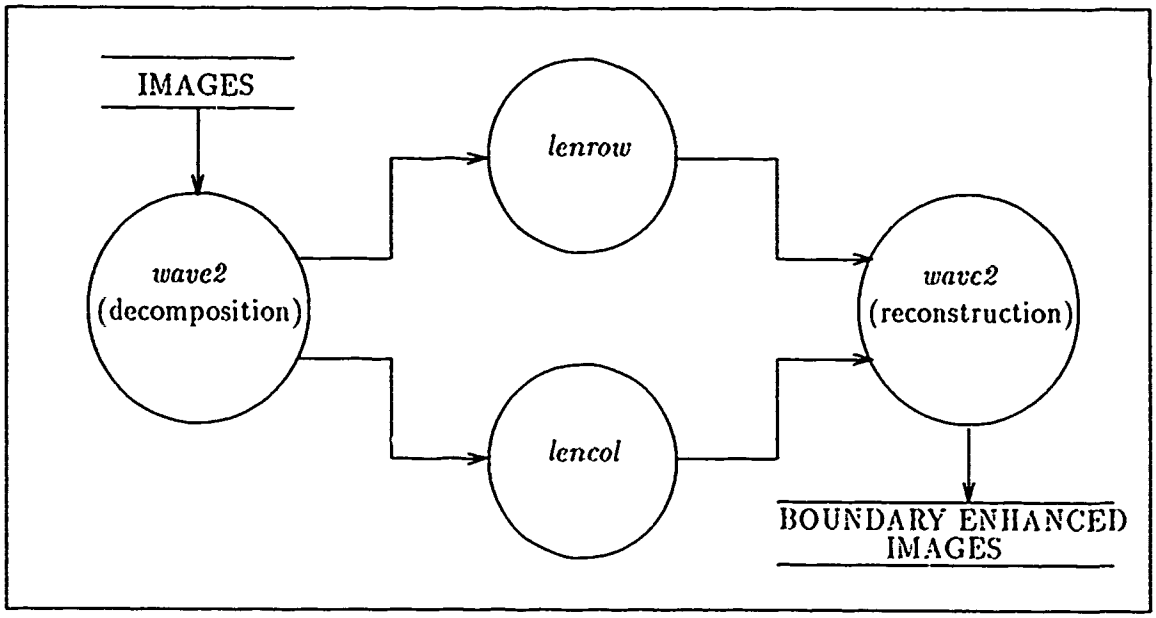


Figure 69. Data Flow of the Boundary Contour Model

to spread the energy along the dimension of excitation. The wavelet coefficients provided by the Mallat Multiresolution Wavelet Decomposition isolate the energy content of each spatial-frequency band or level of resolution in one of three spatial orientations: horizontal, vertical, and angular. We perform lateral excitation along the horizontal direction to spread the horizontal wavelet coefficients in the horizontal dimension of the two dimensional coefficient array. Likewise, the energy of the vertical wavelet coefficients are spread vertically and the energy of the angular coefficients are spread in both dimensions, vertically and horizontally. The lateral excitation process is performed in such a way as to maintain the total energy of each set of wavelet coefficients at a given orientation and level of resolution. This is necessary to ensure accurate reconstruction in the final step. Equation 72 gives the one dimensional lateral excitation algorithm performed by the *lenrow* and the *lencol* programs which operate on rows and columns respectively.

$$w_i = \left(\sum_{p=-P}^{-1} 2^p J_{i+p} + \sum_{p=1}^{P} 2^{-p} J_{i+p} \right) - \left(\sum_{p=-P}^{-1} 2^p + \sum_{p=1}^{P} 2^{-p} \right) J_i \qquad (72)$$

for $i, p \in \mathbf{Z}$, where w_i is the output value of the i^{th} input coefficient, J_i, after lateral excitation, and P is the maximum extent of the excitation window in both the positive and negative directions from J_i. The first term enclosed in parentheses is the surrounding receptive field

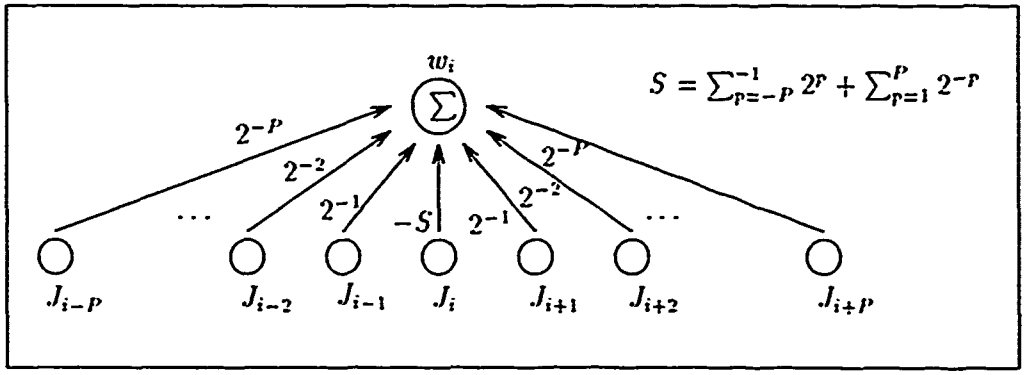


Figure 70. Lateral Excitation Network of Equation 72

of the i^{th} cell. The second term is the center of the receptive field and constitutes the

amount of the i^{th} input cell, J_i, that subtracts from the i^{th} output cell, w_i. Software source code for these programs is listed in Appendix E.2. The third and final step in the BCM is to reconstruct the image with the *wave2* program using the affected wavelet coefficients. This was done by simply substituting the output of the lateral excitation networks for the decomposed wavelet coefficients before selecting the reconstruction option from the main menu of the *wave2* program.

8.2 Conclusions

Figure 71 shows the resulting reconstruction of the Kanisza Triangle in which only the fourth level detail wavelet coefficients are affected by lateral excitation. Figure 73 shows the same figure in which all levels of detail wavelet coefficients are affected by lateral excitation from level one through level four. These results are satisfying and surprising. They are satisfying because the energy tends to spread in such a way as to aid the outline of the illusory triangle and they are surprising because the appearance of the energy spread so closely resembles Oberndorf's results (see Figure 72).

The goal of this analysis is to test the hypothesis that specific spatial-frequency channels, within a specific orientation, may contain spatial receptive fields that together determine the response of a single spatial element centered in that field. The result of such a response network would be a new set of orientation specific, spatial-frequency specific response elements whose energy is determined by the weighted sum of their respective receptive field. If this is true, it would go a long way in explaining the perception of illusory contours which lie between or in line with two or more regions of like orientation. Consider, for example, the base of the illusory Kanisza Triangle (see Figure 53). This contour is oriented horizontally and lies in line with the horizontal edges of the contributing objects. It may be that the perception forming area of the brain receives signals in the location and orientation of this contour stimulated by the high spatial frequency energy of the edges whose orientation is horizontal.

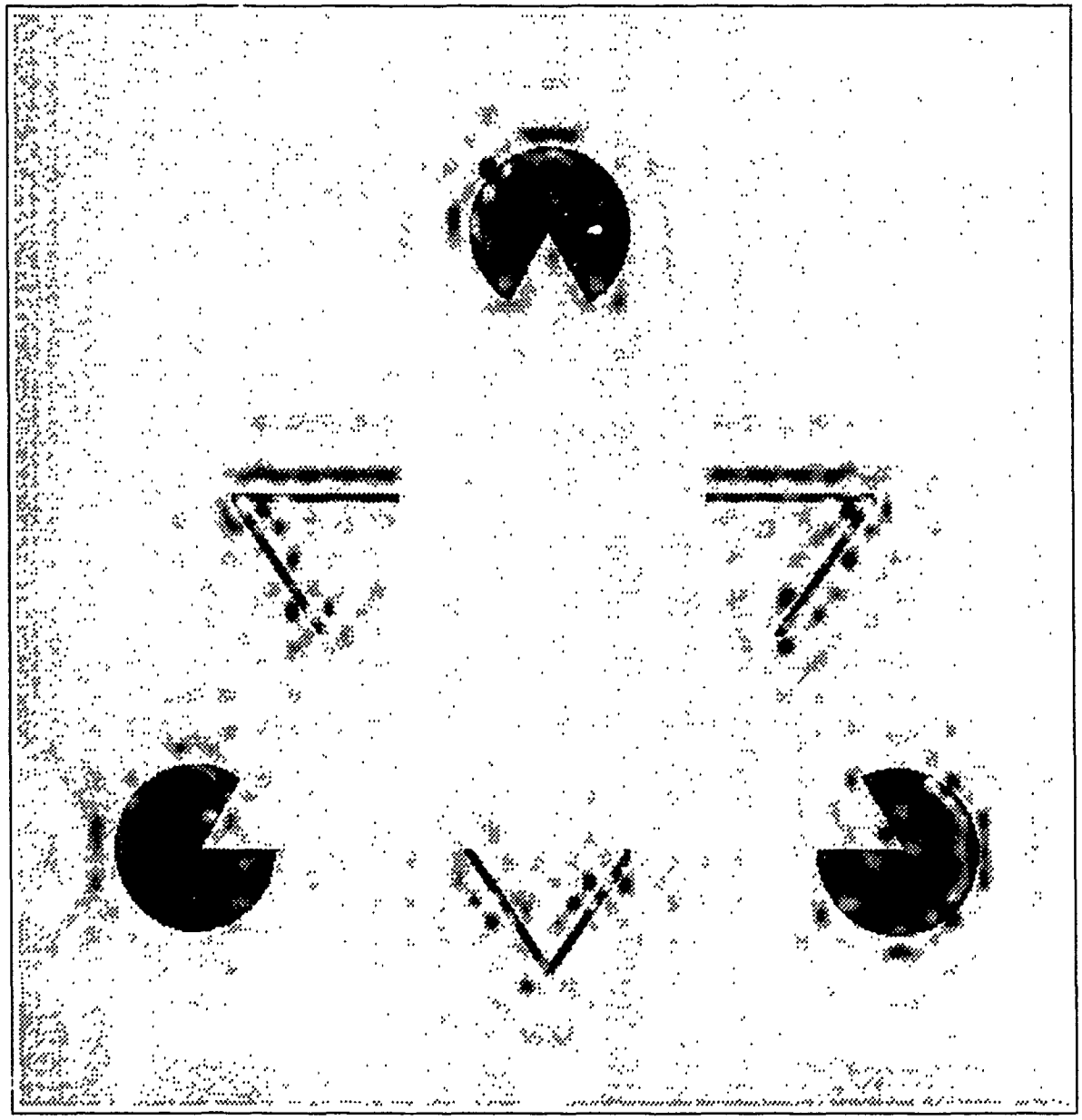


Figure 71. Output of Boundary Contour Model Using Only Level 4 Detail Coefficients

Figure 72. Oberndorf's Results Using a Gabor Low Pass Filter [32]

Biological experiments performed on the visual cortex of the cat and monkey revealed at least two distinct catagories of cells, simple cells and complex cells [23, 22]. In these experiments, microelectrodes measured the impulse response of the respective cells. Simple cells were found to respond only to visual stimulus within a specific spatial-frequency band and orientation and complex cells were found to be orientation independent suggesting that each complex cell has a receptive field of simple cells that themselves respond to a specified band of spatial-frequencies and orientations. One explanation is that an intermediate layer exists between these simple and complex cells. One that responds to specific spatial-frequencies and orientations in the desired locations. Another explanation is that the perception forming areas of the brain or, some intermediate stage which may reside in another location in the cortex receives as input a receptive field of simple cells. Since so little is known of the interconnections of the brain, it is not unreasonable to make these hypotheses.

The wavelet detail coefficients provide the required spatial-frequency and orientation selectivity to simulate simple cell response. Considering the type of network connections known to be possible in the cortex, we apply a lateral excitation network to these coefficients and reconstruct the original image using the excited values. The results shown in Figure 71 strongly suggest illusory perception is aided by such a network. Taking the excitation process to all bands simultaneously as shown in Figure 73 only degrades the suggestion of

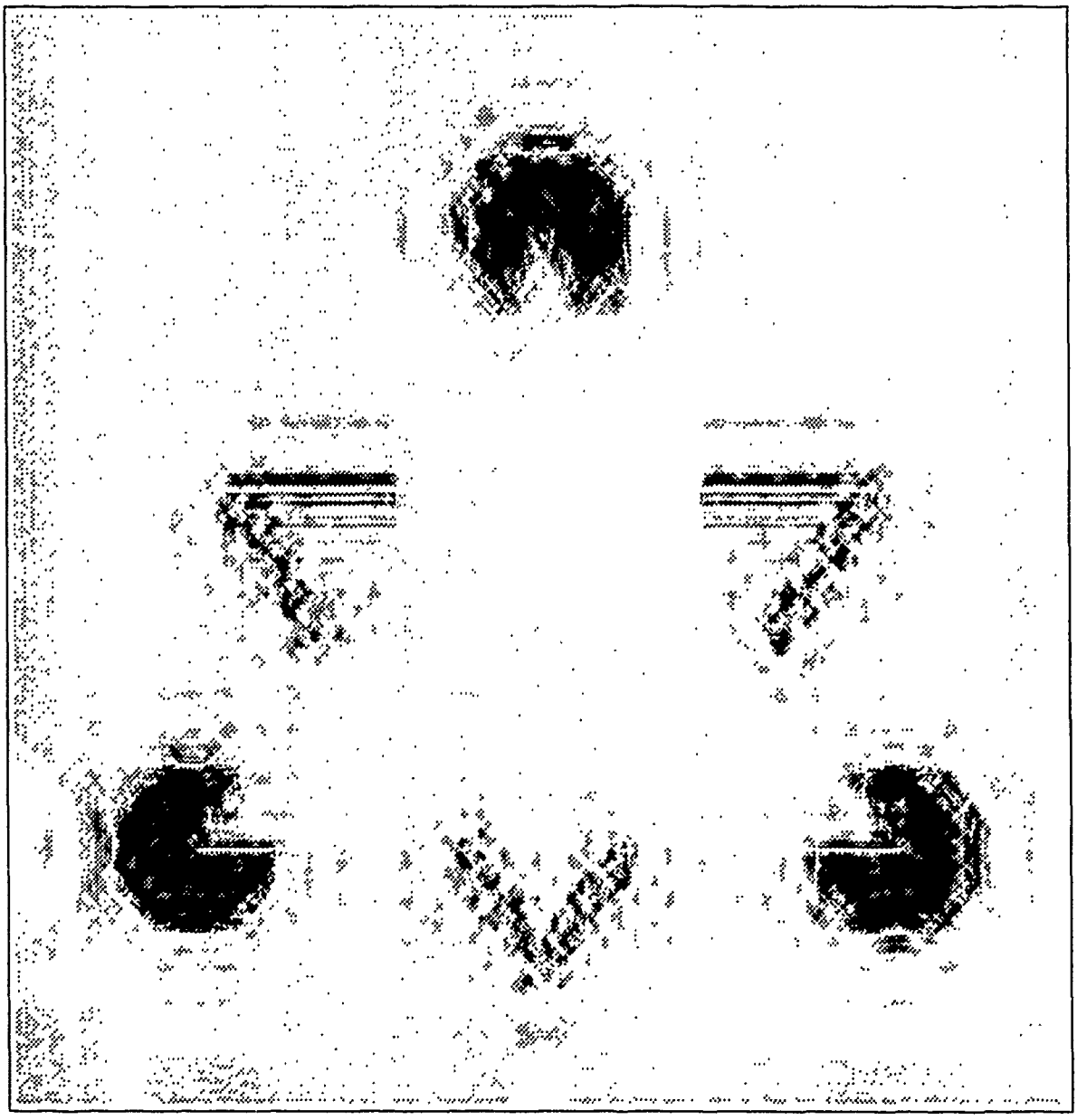


Figure 73. Output of Boundary Contour Model Using Levels 1-4 Detail Coefficients

the contours through the distortion of the high spatial-frequency characteristics needed to terminate the contours. In our results, only the excitation of the lowest frequency band, shown in Figure 71, maintains the edges of the contributory objects distinctly. To obtain this result, the level four detail coefficients were excited and used in the reconstruction. This level corresponds to a spatial-bandwidth of 16 to 32 cycles per object (see Figure 54). This spatial-bandwidth corresponds to highest frequencies passed by both Gisburg's [14] and Oberndorf's [32] low pass filters.

The Boundary Contour Model described in this chapter, while extremely simple, is much like the first stage of the Grossberg Boundary Contour System (BCS), "Competition I, On-Center-Off-Surround Interaction within Each Orientation" [18:169]. In the BCS, Grossberg includes two more major stages and some refinements. The second stage is "Competition II, Push-Pull Opponent Processes Between Orientation at Each Position" and the third stage is "Oriented Cooperation: Statistical Gates" [18:169-170]. While the BCM proposed here is not iterative nor does it incorporate a feedback loop as does the BCS, it does demonstrate the edge enhancing potential of performing lateral excitation within a local receptive field in a specific spatial-frequency and orientation bandwidth. Furthermore, this work is the first of its kind to display the output of a contour enhancing model that still provides the complete range of spatial-frequencies in an algorithmically sound manner, Wavelet Multiresolution Reconstruction. As a matter of fact, the wavelet coefficients may prove to be a good input souce for the BCS taking the place of Grossberg's oval dipole receptive fields.

IX. Conclusions/Recommendations

9.1 Introduction

By investigating possible human perceptual processing of the Kanisza Triangle illusion, this thesis provides some insight into the workings of the human visual system. Ginsburg investigated low spatial-frequency biasing in visual perception formulation. Oberndorf went a step farther with his application of the location sensitivity inherent in Gabor filtering. His results support Ginsburg's thesis that low spatial-frequency information is important in the perception of illusory contours. In this thesis, we further explore frequency contributions to percepts by considering high spatial-frequencies as well as low spatial-frequencies and then incorporating temporal-frequencies. Due to a characteristic of the Wavelet Transform to effectively trade resolution in time or space for resolution in temporal-frequency or spatial-frequency respectively, we use a Multiresolution Wavelet Decomposition in the place of Gabor filtering. The results of this decomposition are approximations and detail coefficients that represent the spatial and temporal bands of frequency information which provide input into our biologically motivated models of visual system processing.

9.2 Preliminary Results

Before implementing our three visual system models, we stop to compare our low spatial-frequency representation of the image output from the Multiresolution Wavelet Decomposition as a coarse approximation of the original image with the results of Ginsburg and Oberndorf. This comparison leads to the conclusion that Oberndorf's Gabor Lowpass filter possesses characteristics not found in the filtering process of the Multiresolution Wavelet Decomposition. The primary difference between the Fourier and Gabor filtering and the decomposition filtering is the ringing associated with the sharp cutoff of the Ideal Fourier and Gabor filters. This ringing seems to be the cause of the energy spread observed in their results. If the ringing indeed aids the percept, it suggests the spatial filtering process of the brain is also characterized by ringing. On the other hand, it may be that some cerebral

processing in addition to spatial filtering is required to cause illusory percepts. It is likely that the results of spatial-frequency processing is fused with some other cerebral processing (e.g. temporal-frequency processing). After all, there is much unknown in the current understanding of the processing that takes place in the cerebral complex.

9.3 Building a World Model

We use the multiresolution approximations as input to a model of the visual system based on the known operation of saccadic eye movements and retinal processing. The observation that led to this approach is that the illusion seems to break up when the viewer forces fixation at one point in the image; thus, eliminating the saccadic movements of the eye. Since retinal fixation between saccades is necessary for the retina to process high spatial-frequency information from the field of vision, this observation suggests that high spatial-frequency information is somehow critical in the perception of the illusory contours. Compared to the previous results of Figure 55, the low spatial-frequency spreading in Figure 60 is found now only in the areas not replaced by high resolution information. While this does not discount the importance of the "ringing" in the Ginsburg and Oberndorf results, it does emphasize the necessity of incorporating the high spatial-frequency bands in the analysis of the illusion. This frame-built model illustrates how the quality of the illusion is enhanced by adding high spatial-frequency information in selected locations along the suggestive contours.

Two areas of investigation naturally follow from the above observations: 1. Whether manipulation of high spatial-frequency information helps to enhance or diminish the illusion, and 2. Whether the temporal-frequency information across the frames of the world model contributes to or is necessary to produce the illusion. Therefore, we developed a spatial-temporal model and a boundary contour model.

9.4 A Spatial-Temporal Model

Considering the temporal-frequency information across a series of frames in an animated scene, we perform a Multiresolution Wavelet Decomposition in the time dimension

and then use the coarse approximations of the time signals of each pixel in the set of frames to rebuild a version of the frames that is "blurred in time". The brain may process temporal-frequency information in such a way as to suppress high temporal-frequencies smoothing the motion giving the eye more time to fixate enough locations around the scene to produce the illusion. This suggests that the Ginsburg proposal of low spatial-frequency biasing applies to temporal-frequency as well. The question is how fast can the motion be before there is not enough time to process high spatial-frequencies. Obviously, the slower the motion the more spatial processing can take place. If relatively more spatial processing is necessary to produce the illusion, more high temporal-frequencies must be filtered out. Therefore, each illusion depending on its relative spatial complexity has a fundamental space-time bandwidth envelope in which it is perceived.

9.5 A Boundary Contour Model

The Boundary Contour Model takes a closer look at the possible contributions of the spatial-frequency components of a scene to the perception of illusory contours. It demonstrates the potential of using the wavelet detail coefficients as input to the Grossberg Boundary Contour System (BCS). It simualates a non-iterative version of the first stage of the BCS, Off-Center-Off-Surround network within each orientation. The results show the same kind of energy spread found in Oberndorf's results but for a totally different reason. In Figure 71 the energy spread is caused by a lateral excitation network in which the receptive fields lies within specified localities, spatial-frequency bandwidth, and orientations; whereas, Oberndorf's energy spread is caused by the ringing characteristic of a highly tuned low-pass filter. The ocurrence of lateral excitation networks in the central nervous system is well known. Therefore, it presents a more plausible explanation of energy spreading than does ringing which is not easily characterized. But, the most gratifying aspect of the result of Figure 71 is that it provides all spatial-frequency components found in the original image in effect fusing the suggestive contour data on top of a distinctly recognizable figure.

9.6 Recommendations

This thesis lays the ground work for a whole new realm of study – that of spatial/temporal-frequency processing in the human cerebral complex. The next logical step in this area would be to implement Grossberg's Boundary Contour System with the detail coefficients output from the Multiresolution Wavelet Decomposition [18]. Such a composite boundary finding system could be implemented in a highly parallel architecture. Its output would be extremely useful in object segmentation applications in which incomplete boundaries are segmented from background textures. The Wavelet based BCS would "connect the dots" and give distinct form to objects of interest [41].

In this thesis, the temporal-frequencies are processed in much the same way as the spatial-frequency processing of earlier work [14, 32]. This new comparison of the time domain to the space domain suggests that there exist contours across space and time. It may be possible using the BCS to isolate and characterize such contours. This thesis provides the tools to do just that.

The whole field of spatial-temporal image processing is basically untouched for applications in pattern recognition, texture segmentation, and feature extraction. The spatial/temporal models proposed here are applicable and will provide a method of improving feature sets used in these application areas by bluring in space and time.

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, i,
           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    "jsmacros.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.                                                             */
/*                                                                           */
/*  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 author's 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

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. The proper usage of *wave2* is as follows:

```
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 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
    Recipies 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 coarsest 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")
         loop1j(d3_coef.ROW,d3_coef.COL)
            fscanf(infile, "%f\n", &d3_coef.array[i+1][j+1]);
         CLOSE_FILE(1, 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] = 1.302557547; /* 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
  Recipies 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
  Recipies 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 spconvlv (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 - 1];
            }

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

```
/* 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) [35]

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.0: 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 author's 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. The usage of *wave1* is as follows:

```
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 analysis
    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 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
    Recipies 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 spatial 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, spconv1v.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 coarsest 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
  Recipies 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 Fourier
  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
  Recipies 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*c_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) [35]

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 to Build a World Model*

C.1 System Description of the FBUILD Program

This program requires as input the output of the "*wave2*" program. When running the "*wave2*" program, any available wavelet may be used. However, the filenames of the approximation images required as input to the "fbuild" program must have the wavelet code suffix stripped off before running the the "*fbuild*" program. In the case of the anlaysis of Chapter VI of the author's thesis, we used the cubic spline wavelet in the "wave2" program. Therefore, the ".spl" had to be removed from the end of each of the approximation files whose names were "wkanisza.512.1.c.spl" through "wkanisza.512.4.c.spl".

Three parameters are adjustable before compile time: 1. The level or depth that "fbuild" uses to build the frames, 2. The window size in pixels that "fbuild" uses to determine the spatial extent of information taken from each level for each fixation point, and 3. the number of fixation points used to build the composite image. These parameters may be changed in the declaration section of the fbuild.c file in the `#define` statements.

To run the program, type at the command line the following:

```
command prompt: fbuild <filename> <size>
```

The filename and its size are optional entries on the command line. If not used, "*fbuild*" will prompt the user for these items.

The "fbuild" program was written for the masters thesis of J. Stewart Laing. It was used in the evaluation executed in Chapter VI of that thesis for the Air Force Institute of Technology. The author has no intention of maintaining this program or enhancing it in any way.

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

1. `fbuild.c` - The main program for fbuild.
2. `futil.c` - Utility functions written specifically for the fbuild program.

3. `nrutil.c` - Utiltity functions *Numerical Recipies in C* (See Appendix F.2 for listing) [35]. decomposition.

4. `spline.c` - A routine from it Numerical Recipies in C used in the cubic spline expansion [35].

5. `splint.c` - A routine from *Numerical Recipies in C* used to perform cubic spline interpolations [35].

6. `splin2.c` - A routine from *Numerical Recipies in C* used in two dimensional cubic spline interpolations [35].

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.2 for listing).

9. `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 *fbuild* that may be executed by typing *"fbuild"* at the command prompt.

C.2 FBUILD Program Software

C.2.1 Listing of FBUILD.C

```
/*************************************************************************/
/*************************************************************************/
/***              FRAME BUILDER MAIN PROGRAM DRIVER                    **/
/*************************************************************************/
/*************************************************************************/
/*  DATE:  14 Aug 91
    VERSION: 1.0
```

```
    NAME:  fbuild.c
    DESCRIPTION:  This program builds frames for a model of the human visual
    system based on the characteristic of the human eye to scan and fixate
    to build a world model for use in the brain. The program uses as input
    data the approximation images from a wavelet multiresolution decompostion.
    In begins with the approximation at a given level of resolution and builds
    a frame based on given fixation coordinates in the original image.
    FILES READ: approximation images as generated by the wave2
    program with the wavelet type suffix stripped off.
    FILES WRITTEN:  The output file is one frame with the suffix .frm
    HEADERS USED:   <stdio.h>, "jsmacros.h", "stewmath.h",
    <math.h>
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:    futil.c, nrutil.c
    AUTHOR: J. Stewart Laing
    HISTORY: Initial Version.
*/
/*************************************************************************/
/*************************************************************************/

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

#include <stdio.h>
#include "jsmacros.h"
#include "stewmath.h"
#include <math.h>
#define  WIDTH     4
#define  DEPTH     4
#define  FRAMES    15

void  extract();
void  expand1();
void  insert();
float **matrix();
void  free_matrix();
float  *vector();
int    *ivector();
void  free_vector();
void  free_ivector();

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

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

int          i,j,k,l,m, fwidth, depth, x, y, *X, *Y, row, col, offset;
char         infilename[64], filename[64];
float_array  a, d1, d2, d3, d, exp, frame;
FILE         *infile, *outfile;
float        **sub, ceil;

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

if(argc != 3 && argc != 1){
   printf("Usage: fbuild <filename> <N for NxN array of original image>\n");
   exit(0);
   }
```

```
if(argc == 1){
   printf("\n\n\n  Input the size of the image (N for NxN array):");
   scanf("%d", &frame.ROW);
   printf("  \n\n Input filename of image to be histogramed:>"); fflush(stdout);
   scanf("%s", infilename);
 }
 else {
    sprintf(infilename, "%s", argv[1]);
    sscanf(argv[2], "%d", &frame.ROW);
 }
/*******************/
/* Allocate Memory */
/*******************/
fwidth = WIDTH;
depth = DEPTH;
exp.ROW = fwidth*(int)(pow(2.0,(double)depth));
a.ROW = 2*frame.ROW;
a.COL = a.ROW;
a.array = matrix(1, a.ROW, 1, a.COL);
loop1ij(a.ROW,a.COL) a.array[i][j] = 0.0;
frame.ROW *= 2;
frame.COL = frame.ROW;
frame.array = matrix(1, frame.ROW, 1, frame.COL);
loop1ij(frame.ROW,frame.COL) frame.array[i][j] = 0.0;
exp.COL = exp.ROW;
exp.array = matrix(1, exp.ROW, 1, exp.COL);
loop1ij(exp.ROW,exp.COL) exp.array[i][j] = 0.0;
sub = matrix(1, 2*fwidth, 1, 2*fwidth);
loop1ij(2*fwidth,2*fwidth) sub[i][j] = 0.0;
/*d.ROW = frame.ROW/(int)(pow(2.0,(double)depth));
d.COL = d.ROW;
d.array = matrix(1, d.ROW, 1, d.COL);
loop1ij(d.ROW, d.COL) d.array[i][j] = 0.0;
d1.ROW = frame.ROW/(int)(pow(2.0,(double)depth));
d1.COL = d.ROW;
d1.array = matrix(1, d1.ROW, 1, d1.COL);
loop1ij(d1.ROW, d1.COL) d1.array[i][j] = 0.0;
d2.ROW = frame.ROW/(int)(pow(2.0,(double)depth));
d2.COL = d2.ROW;
d2.array = matrix(1, d2.ROW, 1, d2.COL);
loop1ij(d2.ROW, d2.COL) d2.array[i][j] = 0.0;
d3.ROW = frame.ROW/(int)(pow(2.0,(double)depth));
d3.COL = d3.ROW;
d3.array = matrix(1, d3.ROW, 1, d3.COL);
loop1ij(d3.ROW, d3.COL) d3.array[i][j] = 0.0;*/
/*****************************************/
/*  prompt user for fixation coordinate: */
/*****************************************/
/*   printf("\n\n Input the number of coordinates in memory:");
   scanf("%d",&l);
X = ivector(1, l);
Y = ivector(1, l);
loop1i(l) {
   printf("\n\n Input col coordinate X%d:",i);
   scanf("%d", &X[i]);col = X[i];
   printf("\n\n Input row coordinate Y%d:",i);
   scanf("%d", &Y[i]);row = Y[i];
   }
*/
l=FRAMES;
```

```
X = ivector(1, l); Y = ivector(i, l);
sprintf(filename, "col15.x");
OPEN_FILE(infile, filename, "The Frame Builder")
loop1i(l) fscanf(infile, "%d\n", &X[i]);
CLOSE_FILE(i, filename, "The Frame Builder", infile)
sprintf(filename, "row15.y");
OPEN_FILE(infile, filename, "The Frame Builder")
loop1i(l) fscanf(infile. "%d\n", &Y[i]);
CLOSE_FILE(i, filename, "The Frame Builder", infile)

/****************************************/
/*  prompt user for fixation threshold   */
/****************************************/

/*   printf("\n\n Input the threshold for fixation(float):");
   scanf("%f",&ceil);*/

/* set size of arrays for reading */
a.ROW = (frame.ROW/2)/(int)(pow(2.0,(double)depth));
a.COL = a.ROW;
offset = frame.ROW/4;

/* read in coarsest approximation and details */

/*printf("\nreading...\n");fflush(stdout);
sprintf(filename, "%s.%d.c", infilename, depth);
OPEN_FILE(infile, filename, "The Frame Builder")
loop1ij(a.ROW,a.COL) fscanf(infile, "%f\n", &a.array[i][j]);
CLOSE_FILE(i, filename, "The Frame Builder", infile)*/

/*printf("\nreading...\n");fflush(stdout);
sprintf(filename, "%s.%d.d1", infilename, depth);
OPEN_FILE(infile, filename, "The Frame Builder")
loop1ij(a.ROW,a.COL) fscanf(infile, "%f\n", &d1.array[i][j]);
CLOSE_FILE(i, filename, "The Frame Builder", infile)
printf("\nreading...\n");fflush(stdout);
sprintf(filename, "%s.%d.d2", infilename, depth);
OPEN_FILE(infile, filename, "The Frame Builder")
loop1ij(a.ROW,a.COL) fscanf(infile, "%f\n", &d2.array[i][j]);
CLOSE_FILE(i, filename, "The Frame Builder", infile)
printf("\nreading...\n");fflush(stdout);
sprintf(filename, "%s.%d.d3", infilename, depth);
OPEN_FILE(infile, filename, "The Frame Builder")
loop1ij(a.ROW,a.COL) fscanf(infile, "%f\n", &d3.array[i][j]);
CLOSE_FILE(i, filename, "The Frame Builder", infile)*/

/* find fixation vectors */

/*X = ivector(1, d.COL*d.ROW);
Y = ivector(1, d.ROW*d.COL);
l=0;
loop1ij(d.ROW, d.COL){
   d.array[i][j] = d1.array[i][j]+d2.array[i][j]+d3.array[i][j];
   if(d.array[i][j] > ceil) {
      if((i< 31) && (j<31)){
         l++;
         X[l] = i;printf("\nX[%d]=%d",l,X[l]);fflush(stdout);
         Y[l] = j;printf("\nY[%d]=%d",l,Y[l]);fflush(stdout);
         }
      }
   }*/
printf("\nThere are %d fixation points", l);fflush(stdout);

/* expand coarsest approx to form background of frame */
```

```
/*printf("\nexpanding...\n");fflush(stdout);
expand1(a.array, a.ROW, frame.ROW, frame.array);*/

/* read in already expanded approximation of coarsest approx
   to form background of frames. */

printf("\nreading...\n");fflush(stdout);
sprintf(filename, "%s.%d.c.exp", infilename, depth);
OPEN_FILE(infile, filename, "The Frame Builder")
loop1ij(frame.ROW/2,frame.COL/2)
   fscanf(infile, "%f\n", &frame.array[i+offset][j+offset]);
CLOSE_FILE(i, filename, "The Frame Builder", infile)

/* begin iteration */

for(k=depth-1;k>=0;k--){

   /* read in next approx. */

   printf("\nreading...\n");fflush(stdout);
   a.ROW *= 2; a.COL *= 2;
   sprintf(filename, "%s.%d.c", infilename, k);
   OPEN_FILE(infile, filename, "The Frame Builder")
   loop1ij(a.ROW,a.COL)
      fscanf(infile, "%f\n", &a.array[i+offset][j+offset]);
   CLOSE_FILE(i, filename, "The Frame Builder", infile)

   loop1m(1){

      /* extract area of approx. */

      printf("\nextracting...\n");fflush(stdout);
      x = ((X[m]/(fwidth*2))*(fwidth*2))+(fwidth);
      y = ((Y[m]/(fwidth*2))*(fwidth*2))+(fwidth);
      x = (x-1)/(int)(pow(2.0,(double)k))+1;
      y = (y-1)/(int)(pow(2.0,(double)k))+1;
      x -= (fwidth-1);
      y -= (fwidth-1);

/*      X[m] = 2*(X[m]-1)+1;
      Y[m] = 2*(Y[m]-1)+1;
*/
      extract(a.array, fwidth*2, x+offset, y+offset, sub);

      /* expand to scale of frame */

      printf("\nexpanding...\n");fflush(stdout);
      exp.ROW = fwidth*2*(frame.ROW/2)/a.ROW;
      exp.COL = exp.ROW;
      printf("\narea=%dx%d",exp.ROW,exp.COL);fflush(stdout);
      printf("\nvector=%d",m);fflush(stdout);
      expand1(sub, fwidth*2, exp.ROW, exp.array);

      /* overwrite new approx onto frame */

      printf("\ninserting...\n");fflush(stdout);
      x = (int)(pow(2.0,(double)k))*(x-1)+1;
      y = (int)(pow(2.0,(double)k))*(y-1)+1;
      insert(exp.array, exp.ROW, x+offset, y+offset,
     frame.array);

      }/* end m loop*/

   /* recenter coordinates */

/*   loop1i(1){
      X[i] += fwidth/2;
      Y[i] += fwidth/2;
      }*//* end i loop */

   }/* end k loop */

/* write frame to a file */
```

```
printf("\nwriting...\n");fflush(stdout);
sprintf(filename, "%s.frm.%d%d", infilename, row, col);
CREATE_FILE(outfile, filename, "The Frame Builder")
loop1ij(frame.ROW/2,frame.COL/2)
   fprintf(outfile, "%f\n", frame.array[i+offset][j+offset]);
CLOSE_FILE(i, filename, "The Frame Builder", outfile)

/* free memory */

exp.ROW = fwidth*(int)(pow(2.0,(double)depth));
a.ROW = frame.ROW;
a.COL = a.ROW;
exp.COL = exp.ROW;
free_matrix(a.array,1,a.ROW,1,a.COL);
free_matrix(exp.array,1,exp.ROW,1,exp.COL);
free_matrix(frame.array,1,frame.ROW,1,frame.COL);
free_matrix(sub,1,2*fwidth,1,2*fwidth);

/* THE END */
}
```

C.2.2 Listing of FUTIL.C

```
/*****************************************************************************/
/***                        FRAME BUILDER UTILITY SUBROUTINE                **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  14 Aug 91
    VERSION: 1.0
    NAME:  futil.c
    DESCRIPTION:  This program provides the utilities for cubic
    spline interpolation based expansion of the images used int
    the fbuild program.
    FILES READ: NONE
    FILES WRITTEN: NONE
    HEADERS USED:   <stdio.h>, "jsmacros.h", "stewmath.h",
    <math.h>
    CALLING PROGRAMS: fbuild.c
    PROGRAMS CALLED:  nrutil.c, spline.c, splint.c, splin2.c
    AUTHOR: J. Stewart Laing
    HISTORY: Initial Version.
*/
/*****************************************************************************/
/*****************************************************************************/

#include <math.h>
#include "jsmacros.h"
#include <stdio.h>
float **matrix();
float *vector();
void free_vector();
void free_matrix();

void extract(in, window, X, Y, out)
   float **in, **out;
   int   X, Y, window;
{
int i,j;

for(i=Y; i<Y+window; i++)
   for(j=X; j<X+window; j++) out[i-Y+1][j-X+1] = in[i][j];

}
```

```
void expand(in, small, big, out)
   float **in, **out;
   int   small, big;
{
int i,j,k,l, factor;
factor = big/small;
loop1ij(small,small)
   loop1kl(factor,factor) out[factor*(i-1)+k][factor*(j-1)+l] = in[i][j];
}
void insert(in, window, X, Y, out)
   float **in, **out;
   int   window, X, Y;
{
int i,j;
for(i=Y; i<Y+window; i++)
      for(j=X; j<X+window; j++) out[i][j] = in[i-Y+1][j-X+1];
}
void expand0(in, small, big, out)
   int small, big;
   float **in, **out;
{
int i,j,k,l, factor;
float *tab, **yp;
void splin2(), spline();
tab = vector(1, small);
yp = matrix(1, small, 1, small);
factor = big/small;
loop1i(small) tab[i] = factor*i;
loop1i(small) spline(tab, in[i], small, 1.0e30, 1.0e30, yp[i]);
loop1ij(small,small)
   loopkl(factor, factor)
      splin2(tab,tab,in,yp,small,small,(float)(factor*i-k),
 (float)(factor*j-l),&out[factor*i-k][factor*j-l]);

free_vector(tab, 1, small);
free_matrix(yp, 1, small, 1, small);
}
void expand2(in, small, out)
   int small;
   float **in, **out;
{
int i,j,k,l, factor;
float **yp, *tmp, *ptmp, *tab;
void spline(), splint();
yp = matrix(1, small, 1, small);
tab = vector(1, small);
tmp = vector(1, small);
ptmp = vector(1, small);
loop1i(small) tab[i] = 2*i;
loop1i(small) spline(tab, in[i], small, 1.0e30, 1.0e30, yp[i]);
loop1ij(small, small){
   out[2*i][2*j] = in[i][j];
   splint(tab, in[i], yp[i], small, (float)(2*j-1), &out[2*i][2*j-1]);
   }
loop1ij(small, small*2){
```

```
   loop1k(small) tmp[k] = out[2*k][j];
   spline(tab, tmp, small, 1.0e30, 1.0e30, ptmp);
   splint(tab, tmp, ptmp, small, (float)(2*i-1), &out[2*i-1][j]);
   }
free_matrix(yp, 1, small, 1, small);
free_vector(tmp, 1, small);
free_vector(ptmp, 1, small);
free_vector(tab, 1, small);
}
void expand1(in, small, big, out)
   int small, big;
   float **in, **out;
{
int i,j,k,l, factor, index;
float **tmp;
if(small==big) loop1ij(small,small) out[i][j]=in[i][j];
else {
   tmp = matrix(1,big,1,big);
   factor = big/small;
   index = (int)(log((double)factor)/log(2.0));
   loop1ij(small,small) tmp[i][j] = in[i][j];
   loop1i(index){
      expand2(tmp, small, out);
      small *= 2;
      loop1kl(small, small) tmp[k][l] = out[k][l];
      }
   free_matrix(tmp, 1, big, 1, big);
   }
}
```

C.2.3 Listing of NRTUIL.C (See Appendix F.2)

C.2.4 Listing of SPLINE.C

```
void spline(x,y,n,yp1,ypn,y2)
float x[],y[],yp1,ypn,y2[];
int n;
{
int i,k;
float p,qn,sig,un,*u,*vector();
void free_vector();
u=vector(1,n-1);
if (yp1 > 0.99e30)
y2[1]=u[1]=0.0;
else {
y2[1] = -0.5;
u[1]=(3.0/(x[2]-x[1]))*((y[2]-y[1])/(x[2]-x[1])-yp1);
}
for (i=2;i<=n-1;i++) {
sig=(x[i]-x[i-1])/(x[i+1]-x[i-1]);
p=sig*y2[i-1]+2.0;
y2[i]=(sig-1.0)/p;
u[i]=(y[i+1]-y[i])/(x[i+1]-x[i]) - (y[i]-y[i-1])/(x[i]-x[i-1]);
u[i]=(6.0*u[i]/(x[i+1]-x[i-1])-sig*u[i-1])/p;
}
if (ypn > 0.99e30)
qn=un=0.0;
else {
```

```
qn=0.5;
un=(3.0/(x[n]-x[n-1]))*(ypn-(y[n]-y[n-1])/(x[n]-x[n-1]));
}
y2[n]=(un-qn*u[n-1])/(qn*y2[n-1]+1.0);
for (k=n-1;k>=1;k--)
y2[k]=y2[k]*y2[k+1]+u[k];
free_vector(u,1,n-1);
}
```

C.2.5 Listing of SPLINT.C

```
void splint(xa,ya,y2a,n,x,y)
float xa[],ya[],y2a[],x,*y;
int n;
{
int klo,khi,k;
float h,b,a;
void nrerror();
klo=1;
khi=n;
while (khi-klo > 1) {
k=(khi+klo) >> 1;
if (xa[k] > x) khi=k;
else klo=k;
}
h=xa[khi]-xa[klo];
if (h == 0.0) nrerror("Bad XA input to routine SPLINT");
a=(xa[khi]-x)/h;
b=(x-xa[klo])/h;
*y=a*ya[klo]+b*ya[khi]+((a*a*a-a)*y2a[klo]+(b*b*b-b)*y2a[khi])*(h*h)/6.0;
}
```

C.2.6 Listing of SPLIN2.C

```
void splin2(x1a,x2a,ya,y2a,m,n,x1,x2,y)
float x1a[],x2a[],**ya,**y2a,x1,x2,*y;
int m,n;
{
int j;
float *ytmp,*yytmp,*vector();
void spline(),splint(),free_vector();
ytmp=vector(1,n);
yytmp=vector(1,n);
for (j=1;j<=m;j++)
splint(x2a,ya[j],y2a[j],n,x2,&yytmp[j]);
spline(x1a,yytmp,m,1.0e30,1.0e30,ytmp);
splint(x1a,yytmp,ytmp,m,x1,y);
free_vector(yytmp,1,n);
free_vector(ytmp,1,n);
}
```

C.2.7 Listing of JSMACROS.H (See Appendix F.2)

C.2.8 Listing of STEWMATH.H (see Appendix F.2)

C.2.9 Listing of MAKEFILE

```
# Makefile routine for the Frame Builder by Laing and Smiley.
```

```
 DEFLAGS = -g
OBJS = fbuild.o futil.o nrutil.o splin2.o spline.o splint.o

fbuild: $(OBJS)
@echo "linking ..."
cc $(OBJS) -o fbuild $(DEFLAGS) -lm

fbuild.o: fbuild.c
cc -c $(DEFLAGS) fbuild.c

futil.o: futil.c
cc -c $(DEFLAGS) futil.c

nrutil.o: nrutil.c
cc -c $(DEFLAGS) nrutil.c

splin2.o: splin2.c
cc -c $(DEFLAGES) splin2.c

spline.o: spline.c
cc -c $(DEFLAGES) spline.c

splint.o: splint.c
cc -c $(DEFLAGES) splint.c
```

Appendix D. *Software for the Spatial-Temporal Model*

D.1 System Description

The following programs are used in the spatial-temporal analysis of Chapter VII:

1. *kangen* - A program used to generate the frames used in the analysis.
2. *wave2* - The Multiresolution Wavelet Decomposition and Reconstruction program for two dimensional images (See Appendix B.1).
3. *vbuild* - A program used to reduce the total number of time signals that are processes by the *wave1* program.
4. *strip1d* - A program that srips the one dimensional time signals designated by the *vbuild* program for processing by the *wave1* program.
5. *wave1* - The Multiresolution Wavelet Decomposition and Reconsruction program for one dimensional signals modified for use in the analysis of Chapter VII (See Appendix B.2).
6. *rbuild* - A program used to rebuild the frames based on the output of the *wave1* program.
7. *tblur* - A program used to animate the output of the *rbuild* program for demonstration.

The "kangen" program and the "tblur" program run on a Silicon Graphics workstation. The others run on any sytem with an ANSI C compiler. The "wave2" is described in Chapter IV and listed in Appendix B. Listed in this appendix are the "vbuild", "strip1d", "rbuild", and those modules of the "wave1" program that were modified from those in Appendix B for the spatial-temporal analysis.

The "kangen" program is used to create the files that contain the individual frames of the scene to be analyzed. To run this program, type the following on the command line:

```
command prompt: kangen
```

The parameters are hard coded and must be set before the program is compiled. Each output file from this program is then run through the "wave2" program separately to generate the desired level of resolution to be used in the analysis. Whatever level of resolution is used the output files must be expanded to the sample scale of the original "kangen" output before proceeding. The "expd" program may be used for this purpose (see Appendix F.2 for listing).

The output of the "wave2" program is used as input to the "vbuild", "strip1d", and "rbuild" programs. Two parameters are adjustable in these programs: 1. The number of frames in the scene to be analyzed, and 2. The number of vectors found by the "vbuild" program. Of these three, the "vbuid" program must be run first to generate the locations of the pixels that will be used as the one dimensional time signals. To run this program type on the command line the following:

```
command prompt:  vbuild <filename> <size>
```

The filename and its size are optional parameters. If they are not entered on the command line, "vbuild" will prompt the user for them. The filename is a common first part of all the frame files that are asctually read individually. For example, the filename "frame" would signal the "vbuild" to look for files named framex.asc where the x is the frame number starting at 1 and increasing by one up to the value of FRAMES set in the declaration section with the `#define` statement. The output of the "vbuild" program are the files strip.x and strip.y. These files are read in by the "strip1d", "rbuild", and the modified "wave1" programs.

Next, the strip1d program is run by typing the following command at the command prompt:

```
command prompt: strip1d <filename> <size>
```

Treatment of arguments is the same as for the "vbuild" program above. The output of the "strip1d" program is a file that contains the one dimensional time signals used as input

for the modified version of the "wavel" program. To run the "wavel" program, type the following on the command line:

```
command prompt: wavel <filename> <size>
```

Treatment of arguments is the same as for the "vbuild" program above. The output of the "wavel" program are files that contain the multiresolution approximations of the time signals. One of these files is used as input to the "rbuild" program specified by in the source code of the "rbuild" program. To run the "rbuild" program type the following on the command line:

```
command prompt: rbuild <filename> <size>
```

Treatment of the arguments is the same as for the "vbuild" program described above. The output of the "rbuild" program represents the 3D Multiresolution Wavelet Decomposition of the the original set of frames. The output are files that can be animated with the "tblur" program.

The "tblur" program generates an animation of successive frames of a scene. To run the program, simply type the following on the command line:

```
command prompt: tblur
```

All parameters indluding the name and size of the files that contain the frames are hard coded in the "tblur" program and must be properly set before compilation.

D.2 Spatial-Temporal Analysis Software

D.2.1 Listing of KANMOV.C

```
/*************************************************************************/
/***              BREATHING KANISZA TRIANGLE PROGRAM                   **/
/*************************************************************************/

/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  kanmov.c
    DESCRIPTION:  This program generates a "breathing" Kanisza Triangle
    illusion on the Silicon Graphics computers. There may be some code
    which is specific to the 4D series computers. To run the program just
    type <kanmov> at the command prompt. Pressing the down arrow will cause
    the motion to appear faster. Continued depressions will eventually slow
    the animation to a stop and then speed up again. Pressing the up arrow
```

```
    causes the opposite effect. Pressing the right arrow increases the minimum
    closure of the packmen up to the maximum angle defined in the code.
    Pressing the left arrow decreases the minimum
    closure of the packmen until it reaches 0 degrees.

    FILES READ: NONE

    FILES WRITTEN:  Frames may be output in a binary format by pressing
                    the spacebar at any time. The files are named "framex.bin".
                    where x stands for the number of frames saved up to that
                    point since the program was started.

    HEADERS USED:   <stdio.h> <device.h> <gl.h>

    CALLING PROGRAMS:   NONE

    PROGRAMS CALLED:    NONE

    AUTHOR: J. Stewart Laing (with help from John Brunderman and Greg Tarr)

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

#include <device.h>
#include <gl.h>
#include <stdio.h>

#define ANGLEINC 10
#define SETX     256
#define SETY     256
#define XORG     200
#define YORG     200

main()
{
   int             i, j, k=0, speed = 10, total=50, sign=1, base1=300;
   int             base2=1500, base3= -900, X=0, Y=1, maxangle=544;
   int             minangle=0;
   long            twin, radius, aradius, xsize, ysize, center[2];
   short           val;
   unsigned short  *image;
   long            key, t1[2], t2[2], t3[2], c1[2], c2[2], c3[2];
   FILE            *outfile;
   char            filename[64];

   image = (unsigned short *)calloc(SETX*SETY, sizeof(unsigned short));

   prefposition(XORG, XORG+SETX-1, YORG, YORG+SETY-1);

   twin = winopen("Kanisza");

   doublebuffer();
   gconfig();

   getsize(&xsize, &ysize);
   radius = .1*xsize;
   aradius = .31*xsize;
   center[X] = xsize/2.0;
   center[Y] = ysize/2.0 + .04*ysize;
   c1[X] = center[X] - .28*xsize;
   c1[Y] = center[Y] - .16*ysize;
   c2[X] = center[X] + .28*xsize;
   c2[Y] = center[Y] - .16*ysize;
   c3[X] = center[X];
   c3[Y] = center[Y] + .32*ysize;
   t1[X] = center[X] - .28*xsize;
   t1[Y] = center[Y] + .16*ysize;
   t2[X] = center[X] + .28*xsize;
   t2[Y] = center[Y] + .16*ysize;
   t3[X] = center[X];
   t3[Y] = center[Y] - .32*ysize;

   linewidth(5);
```

```
qdevice(UPARROWKEY);
qdevice(DOWNARROWKEY);
qdevice(LEFTARROWKEY);
qdevice(RIGHTARROWKEY);
qdevice(ESCKEY);
qdevice(SPACEKEY);
while (TRUE) {
   while (!qtest()) {
      total+=sign*speed;
      /* do drawing */
      color(WHITE);
      clear();
      color(BLACK);
      circfi((int)c1[X],(int)c1[Y],(int)radius);
      circfi((int)c2[X],(int)c2[Y],(int)radius);
      circfi((int)c3[X],(int)c3[Y],(int)radius);
      bgnclosedline();
      v2i(t1);
      v2i(t2);
      v2i(t3);
      endclosedline();
      color(WHITE);
      arcfi((int)c1[X],(int)c1[Y],(int)aradius, base1 - total,
         base1 + total);
      arcfi((int)c2[X],(int)c2[Y],(int)aradius, base2 - total,
         base2 + total);
      arcfi((int)c3[X],(int)c3[Y],(int)aradius, base3 - total,
         base3 + total);
      if((total > maxangle) || (total < minangle)) sign = -sign;
      swapbuffers();
   }
   /* get keyboard input */
   val = 1;
   while (val) {
      key = qread(&val);
      if((key!=UPARROWKEY) && (key!=DOWNARROWKEY) && (key!=ESCKEY)
      && (key != LEFTARROWKEY) && (key != RIGHTARROWKEY)
      && (key != SPACEKEY))
         val = 0;
   }
   /* act on keyboard input */
   switch (key) {
   case UPARROWKEY:
     speed++; /*speed += 17;*/
      break;
   case DOWNARROWKEY:
      speed--;
      break;
   case LEFTARROWKEY:
      minangle -= ANGLEINC;
      if (minangle < 0) minangle = 0;
      break;
   case RIGHTARROWKEY:
      minangle += ANGLEINC;
      if (minangle > maxangle) minangle = maxangle;
      break;
   case ESCKEY:
      gexit();
      break;
   case SPACEKEY:
      rectread(0, 0, SETX-1, SETY-1, image);
```

```
            k++;
            sprintf(filename, "frame%d.bin", k);
            outfile = fopen(filename, "w");
            if(outfile == NULL)
               printf("Error opening %s as inp - file", filename);
            else{
               for(i=0; i<SETX*SETY; i++) if(image[i]!=0) image[i]=255;
               fwrite(image, sizeof(unsigned short),(SETX*SETY), outfile);
               fclose(outfile);
            }
            break;
         } /* end switch */
      } /* end while(TRUE) */
   } /* THE END */
```

D.2.2 Listing of KANGEN.C

```
/*****************************************************************************/
/***              BREATHING KANISZA TRIANGLE PROGRAM                       **/
/*****************************************************************************/

/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  kanisza.c
    DESCRIPTION:  This program generates a "breathing" Kanisza Triangle
    illusion on the Silicon Graphics computers. There may be some code
    which is specific to the 4D series computers. To run the program just
    type <kanisza> at the command prompt. Pressing the down arrow will cause
    the motion to appear faster. Continued depressions will eventually slow
    the animation to a stop and then speed up again. Pressing the up arrow
    causes the opposite effect. Pressing the right arrow increases the minimum
    closure of the packmen up to the maximum angle defined in the code.
    Pressing the left arrow decreases the minimum
    closure of the packmen until it reaches 0 degrees.
    FILES READ: NONE
    FILES WRITTEN:  Frames may be output in a binary format by pressing
                    the spacebar at any time. The files are named "framex.bin".
                    where x stands for the number of frames saved up to that
                    point since the program was started.
    HEADERS USED:   <stdio.h> <device.h> <gl.h>
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:    NONE
    AUTHOR: J. Stewart Laing (with help from John Brunderman and Greg Tarr)
    HISTORY: Initial Version.
*/
/*****************************************************************************/

#include <device.h>
#include <gl.h>
#include <stdio.h>

#define ANGLEINC 10
#define SETX     256
#define SETY     256
#define XORG     200
#define YORG     200

main()
{
   int             i, j, k=0, speed = 10, total=50, sign=1, base1=300;
   int             base2 1500, base3= -900, X=0, Y=1, maxangle=544;
   int             minangle=0;
```

```
long            twin, radius, aradius, xsize, ysize, center[2];
short           val;
unsigned short  *image;
long            key, t1[2], t2[2], t3[2], c1[2], c2[2], c3[2];
FILE            *outfile;
char            filename[64];

image = (unsigned short *)calloc(SETX*SETY, sizeof(unsigned short));

prefposition(XORG, XORG+SETX-1, YORG, YORG+SETY-1);

twin = winopen("Kanisza");

doublebuffer();
gconfig();

getsize(&xsize, &ysize);
radius = .1*xsize;
aradius = .31*xsize;
center[X] = xsize/2.0;
center[Y] = ysize/2.0 + .04*ysize;
c1[X] = center[X] - .28*xsize;
c1[Y] = center[Y] - .16*ysize;
c2[X] = center[X] + .28*xsize;
c2[Y] = center[Y] - .16*ysize;
c3[X] = center[X];
c3[Y] = center[Y] + .32*ysize;
t1[X] = center[X] - .28*xsize;
t1[Y] = center[Y] + .16*ysize;
t2[X] = center[X] + .28*xsize;
t2[Y] = center[Y] + .16*ysize;
t3[X] = center[X];
t3[Y] = center[Y] - .32*ysize;

linewidth(5);

qdevice(UPARROWKEY);
qdevice(DOWNARROWKEY);
qdevice(LEFTARROWKEY);
qdevice(RIGHTARROWKEY);
qdevice(ESCKEY);
qdevice(SPACEKEY);

while (TRUE) {
   while (!qtest()) {
      total+=sign*speed;

      /* do drawing */

      color(WHITE);
      clear();
      color(BLACK);
      circfi((int)c1[X],(int)c1[Y],(int)radius);
      circfi((int)c2[X],(int)c2[Y],(int)radius);
      circfi((int)c3[X],(int)c3[Y],(int)radius);
      bgnclosedline();
      v2i(t1);
      v2i(t2);
      v2i(t3);
      endclosedline();

      color(WHITE);
      arcfi((int)c1[X],(int)c1[Y],(int)aradius, base1 - total,
         base1 + total);
      arcfi((int)c2[X],(int)c2[Y],(int)aradius, base2 - total,
         base2 + total);
      arcfi((int)c3[X],(int)c3[Y],(int)aradius, base3 - total,
         base3 + total);
      if((total > maxangle) || (total < minangle)) sign = -sign;
```

```
            swapbuffers();
            speed = 0;
        }
        /* get keyboard input */
        val = 1;
        while (val) {
            key = qread(&val);
            if((key!=UPARROWKEY) && (key!=DOWNARROWKEY) && (key!=ESCKEY)
            && (key != LEFTARROWKEY) && (key != RIGHTARROWKEY)
            && (key != SPACEKEY))
                val = 0;
        }
        /* act on keyboard input */
        switch (key) {
        case UPARROWKEY:
          /*speed++;*/ speed += 17;
            break;
        case DOWNARROWKEY:
            speed--;
            break;
        case LEFTARROWKEY:
            minangle -= ANGLEINC;
            if (minangle < 0) minangle = 0;
            break;
        case RIGHTARROWKEY:
            minangle += ANGLEINC;
            if (minangle > maxangle) minangle = maxangle;
            break;
        case ESCKEY:
            gexit();
            break;
        case SPACEKEY:
            rectread(0, 0, SETX-1, SETY-1, image);
            k++;
            sprintf(filename, "frame%d.bin", k);
            outfile = fopen(filename, "w");
            if(outfile == NULL)
                printf("Error opening %s as input file", filename);
            else{
                for(i=0; i<SETX*SETY; i++) if(image[i]!=0) image[i]=255;
                fwrite(image, sizeof(unsigned short),(SETX*SETY), outfile);
                fclose(outfile);
            }
            break;
      } /* end switch */
    } /* end while(TRUE) */
} /* THE END */
```

D.2.3 Listing of VBUILD.C

```
/*****************************************************************************/
/***                BREATHING KANISZA TRIANGLE PROGRAM                     **/
/*****************************************************************************/

/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  vbuild.c
    DESCRIPTION:  This program generates two files that
    represent the x and y coordinates of pixels in the input
    image that will be processes with the wavel program.
    FILES READ: Input file name given on command line or program
```

```
    prompt.
    FILES WRITTEN:  Two files named strip.x and strip.y.
    HEADERS USED:   <stdio.h> <math.h> "jsmacros.h"
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:    NONE
    AUTHOR: J. Stewart Laing
    HISTORY: Initial Version.
*/
/*****************************************************************************/

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

#define  FRAMES  32
#define  THRESH  128.0

int **imatrix();
void  free_imatrix();
float  *vector();
int   *ivector();
void  free_vector();
void  free_ivector();
float  **matrix();
void  free_matrix();

void main(argc, argv)
   int argc;
   char *argv[];
{
   int    i,j,k,l, *X, *Y, size, **temp;
   char   infilename[64], filename[64];
   FILE   *infile, *outfile;
   float    **in;

   /* load parameters */

   if(argc != 3 && argc != 1){
     printf("Usage: fbuild <filename> <N for NxN array of original image>\n");
     exit(0);
   }
   if(argc == 1){
      printf("\n\n\n  Input the size of the image (N for NxN array):");
      scanf("%d", &size);
      printf("  \n\n Input filename of image to be histogramed:>");
      fflush(stdout);
      scanf("%s", infilename);
   }
   else {
      sprintf(infilename, "%s", argv[1]);
      sscanf(argv[2], "%d", &size);
   }

   /* allocate memory */

   in = matrix(1, size, 1, size);
   temp = imatrix(1, size, 1, size);
   loop1ij(size, size) temp[i][j] = 0;
   X = ivector(1, size*size);
   Y = ivector(1, size*size);

   /* load approximations and accumulate high spots */

   loop1k(FRAMES){
      sprintf(filename, "%s%d.asc", infilename, k);
      OPEN_FILE(infile, filename, "The Vector Builder")
      loop1ij(size, size){
```

```
            fscanf(infile, "%f\n", &in[i][j]);
            if(in[i][j]< THRESH){
               if(temp[i][j]==0) temp[i][j]=1;
               if(temp[i][j]==1) temp[i][j]=1;
               if(temp[i][j]==2) temp[i][j]=3;
               if(temp[i][j]==3) temp[i][j]=3;
            }
            else{
               if(temp[i][j]==0) temp[i][j]=2;
               if(temp[i][j]==1) temp[i][j]=3;
               if(temp[i][j]==2) temp[i][j]=2;
               if(temp[i][j]==3) temp[i][j]=3;
            }
         } /* end ij loop */
         CLOSE_FILE(i, filename, "The Vector Builder", infile)
      }/* end k loop */
      /* build vectors */
      l=0;
      loop1ij(size, size){
         if(temp[i][j]==3){
            l++;
            X[l] = j;
            Y[l] = i;
         } /* end if */
      } /* end ij loop */
      /* output vectors */
      sprintf(filename, "strip.x");
      CREATE_FILE(outfile, filename, "The Vector Builder")
      loop1i(l) fprintf(outfile, "%d\n", X[i]);
      CLOSE_FILE(i, filename, "The Vector Builder", outfile)
      sprintf(filename, "strip.y");
      CREATE_FILE(outfile, filename, "The Vector Builder")
      loop1i(l) fprintf(outfile, "%d\n", Y[i]);
      CLOSE_FILE(i, filename, "The Vecter Builder", outfile)
      free_matrix(in, 1, size, 1, size);
      free_imatrix(temp, 1, size, 1, size);
   } /* THE END */
```

D.2.4 Listing of SPLIT1D.C

```
/*****************************************************************************/
/***              BREATHING KANISZA TRIANGLE PROGRAM                       **/
/*****************************************************************************/

/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  strip1d.c
    DESCRIPTION:  This program strips off the one dimensional
    time signals based on the coordinates provided by the input
    files strip.x and strip.y generated by the vbuild program.
    All signals are output in a signal file in which each signal
    is a row of a 2D matrix stored in that file.
    FILES READ: Input file name given on command line or program
    prompt. Files strip.x and strip.y are read in
    automatically.
    FILES WRITTEN:  One file with the suffix .tsig is written
    which holds each one dimensional time signal in a row.
    HEADERS USED:   <stdio.h> <math.h> "jsmacros.h"
```

```
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:    NONE
    AUTHOR: J. Stewart Laing
    HISTORY: Initial Version.
*/
/*****************************************************************************/

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

#define VECTORS  7189
#define FRAMES   32

int **imatrix();
void  free_imatrix();
float  *vector();
int   *ivector();
void  free_vector();
void  free_ivector();

void main(argc, argv)
   int  argc;
   char *argv[];
{
   int    i,j,k,l, *X, *Y, size;
   float  *in;
   char   infilename[64], filename[64];
   FILE   *infile, *outfile;

   /* load parameters */

   if(argc != 3 && argc != 1){
     printf("Usage: strip1d <filename> <N for NxN array of original image>\n");
     exit(0);
   }
   if(argc == 1){
      printf("\n\n\n  Input the size of the image (N for NxN array):");
      scanf("%d", &size);
      printf("  \n\n Input filename of image to be histogramed:>");
      fflush(stdout);
      scanf("%s", infilename);
   }
   else {
      sprintf(infilename, "%s", argv[1]);
      sscanf(argv[2], "%d", &size);
   }

   /* allocate memory */

   in = vector(1, size*size*FRAMES);
   X = ivector(1, VECTORS);
   Y = ivector(1, VECTORS);

   /* load strip locations */

   sprintf(filename, "strip.x");
   OPEN_FILE(infile, filename, "The Signal Stripper")
   loop1i(VECTORS) fscanf(infile, "%d\n", &X[i]);
   CLOSE_FILE(i, filename, "The Signal Stripper", infile)

   sprintf(filename, "strip.y");
   OPEN_FILE(infile, filename, "The Signal Stripper")
   loop1i(VECTORS) fscanf(infile, "%d\n", &Y[i]);
   CLOSE_FILE(i, filename, "The Signal Stripper", infile)

   /* load frames */

   loop1l(FRAMES){
```

```
      sprintf(filename, "%s%d.asc", infilename, l);
      OPEN_FILE(infile, filename, "The Signal Stripper")
      loop1i(size*size) fscanf(infile, "%f\n", &in[(l-1)*size*size+i]);
      CLOSE_FILE(i, filename, "The Signal Stripper", infile)
   } /* end l loop */
   /* begin stripping */
   sprintf(filename, "%s.tsig", infilename);
   CREATE_FILE(outfile, filename, "The Signal Stripper")
   loop1kl(VECTORS, FRAMES)
      fprintf(outfile, "%f\n", in[(l-1)*size*size+Y[k]*size+X[k]]);
   CLOSE_FILE(i,filename, "The Signal Stripper", outfile)
   /* free memory */
   free_vector(in, 1, size*size*FRAMES);
   free_ivector(X, 1, VECTORS);
   free_ivector(Y, 1, VECTORS);
} /* THE END */
```

D.2.5 Listing of Modified WAVE1 Modules

D.2.5.1 Listing of MAIN-WAVE1D.C

```
/*****************************************************************************/
/*****************************************************************************/
/***            WAVELET ANALYZER MAIN PROGRAM DRIVER                       **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  09 April 91, 18 June 91, 16 August 91, 5 Sept 91
    VERSION: 3.1
    NAME:  main-wave.c
    DESCRIPTION:  This program performs a multiresolution wavelet analysis
    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 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 to one
    dimensional signals.
    Version 3.1 modified the wave1 program to process the output
    of the split1d program which is one file whose contents is a
    2D in which each row is a 1D time signal. In this version
```

```
    all menus are bypassed and only the approximation
    coefficients are actually written out to files.
*/
/*****************************************************************************/
/*****************************************************************************/

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

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

#define  VECTORS  3490
#define  FRAMES   32

int_vector    loadsignal();
void         reconstruct();
void         decompose();
float         **matrix();
void         free_matrix();

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

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

   int          i,j,k,l, selection;
   float       **signal;
   char         filename[64];
   FILE         *infile;

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

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

   /* load time signal locations */

   signal = matrix(1, VECTORS, 1, FRAMES);

   /* load signal */

   sprintf(filename, "%s.tsig", argv[1]);
   OPEN_FILE(infile, filename, "wave1");
   loop1k1(VECTORS, FRAMES)
      fscanf(infile, "%f\n", &signal.vector[k][l]);
   CLOSE_FILE(i, filename, "wave1", infile)

   /* do decomposition */

   decompose(signal, filename);

   /* free memory */

   free_matrix(signal, 1, VECTORS, 1, FRAMES);
} /* THE END */
```

D.2.5.2 Listing of DECOMPOSE1D.C

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

```
/*****************************************************************/
/***            WAVELET DECOMPOSITION SUBROUTINE                 **/
/*****************************************************************/
/*****************************************************************/
/*  DATE:  19 June 91, 16 August 91. 5 Sept 91
    VERSION: 2.1
    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
    Recipies 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
    AUTHOR: Steve Smiley and J. Stewart Laing
    HISTORY: Initial Version.
    Version 2.0 nolonger uses the Fourier domain filtering. Now
    only spactial convolution is done. Also, this version was
    adapted from the two dimensional version 1.0.
    Version 2.1 is modified to work with version 3.1 of
    main-wave1.c. This version specifically processes multiple
    1D time signals as provided by the strip1d.c program.
*/
/*****************************************************************/
/*****************************************************************/

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

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

#define  VECTORS  3490
#define  FRAMES   32

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

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

void decompose(signal, infilename)
   float **signal;
   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          **temp;
FILE          *outfile1c, *outfile2c, *outfile3c, *outfile1d, *outfile2d;
FILE          *outfile3d;
char          filename[64], wave_code[64];

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

temp = matrix(1, VECTORS, 1, FRAMES);
c_coef.length = FRAMES;
c_coef.vector = vector(1, c_coef.length);
loop1i(c_coef.length) c_coef.vector[i] = 0.0;
d_coef.length = FRAMES;
d_coef.vector = vector(1, d_coef.length);
loop1i(d_coef.length) d_coef.vector[i] = 0.0;
h_of_n.vector = vector(1,FRAMES*2);
loop1i(FRAMES*2) h_of_n.vector[i] = 0.0;
g_of_n.vector = vector(1,FRAMES*2);
loop1i(FRAMES*2) g_of_n.vector[i] = 0.0;
h_of_nflipo.vector = vector(1,FRAMES*2);
loop1i(FRAMES*2) h_of_nflipo.vector[i] = 0.0;
g_of_nflipo.vector = vector(1,FRAMES*2);
loop1i(FRAMES*2) g_of_nflipo.vector[i] = 0.0;
phi.vector = vector(1,2*FRAMES);
loop1i(FRAMES*2) phi.vector[i] = 0.0;
phiflipo.vector = vector(1,2*FRAMES);
loop1i(FRAMES*2) phiflipo.vector[i] = 0.0;
phiflipc.vector = vector(1,2*FRAMES);
loop1i(FRAMES*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);
*/
wavelet_type = 3;
```

```
/* 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 filen . */
   /************************************ ****/
   if (wavelet_type == 3) sprintf(wave_code, "db3");
   if (wavelet_type == 4) sprintf(wave_code, "db4");
   if (wavelet_type == 5) sprintf(wave_code, "db5");
   if (wavelet_type == 6) sprintf(wave_code, "db6");
   if (wavelet_type == 7) sprintf(wave_code, "db7");
   if (wavelet_type == 8) sprintf(wave_code, "db8");
   if (wavelet_type == 9) sprintf(wave_code, "db9");
   if (wavelet_type == 10) sprintf(wave_code, "db10");
   if (wavelet_type == 11) sprintf(wave_code, "db11");
   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;
   /* open files */
   sprintf(filename, "%s.%d.c.%s", infilename, 1, wave_code);
   CREATE_FILE(outfile1c, filename, "The Wavelet Analyzer")
   sprintf(filename, "%s.%d.c.%s", infilename, 2, wave_code);
   CREATE_FILE(outfile2c, filename, "The Wavelet Analyzer")
   sprintf(filename, "%s.%d.c.%s", infilename, 3, wave_code);
   CREATE_FILE(outfile3c, filename, "The Wavelet Analyzer");
/*   sprintf(filename, "%s.%d.d.%s", infilename, 1,wave_code);
   CREATE_FILE(outfile1d, filename, "The Wavelet Analyzer")
   sprintf(filename, "%s.%d.d.%s", infilename, 2,wave_code);
   CREATE_FILE(outfile2d, filename, "The Wavelet Analyzer")
   sprintf(filename, "%s.%d.d.%s", infilename, 3,wave_code);
   CREATE_FILE(outfile3d, filename, "The Wavelet Analyzer")
*/
   /*****************************************************************/
   /* Call convolution routine and save the coefficient vectors for */
   /* each level of analysis.                                       */
   /*****************************************************************/
   loop1k(3){
      loop1j(VECTORS){
         convolve(temp[j], h_of_nflipopointer, g_of_nflipopointer,
                  c_coefpointer, d_coefpointer);
         loop1i(c_coef.length) temp[j][i] = c_coef.vector[i];
      }
         switch (k){
         case 1:
            loop1ij(VECTORS, c_coef.length)
               fprintf(outfile1c,"%f\n",temp[i][j]);
         case 2:
            loop1ij(VECTORS, c_coef.length)
               fprintf(outfile2c,"%f\n",temp[i][j]);
         case 3:
            loop1ij(VECTORS, c_coef.length)
```

```
            fprintf(outfile3c,"%f\n",temp[i][j]);
          }
        } /* end k loop */
   CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile1c)
   CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile2c)
   CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile3c)
/*   CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile1d)
   CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile2d)
   CLOSE_FILE(i, filename, "The Wavelet Analyzer", outfile3d)
*/
} /* end else */
/* free memory */
free_matrix(temp, 1, VECTORS, 1, FRAMES);
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,FRAMES*2);
free_vector(g_of_n.vector,1,FRAMES*2);
free_vector(phi.vector,1,FRAMES*2);
free_vector(phiflipo.vector,1,FRAMES*2);
free_vector(phiflipc.vector,1,FRAMES*2);
/* THE END */
}
```

D.2.6 Listing of RBUILD.C

```
/*****************************************************************/
/***                        REBUILD PROGRAM                    ***/
 /*****************************************************************/
/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  rbuild.c
    DESCRIPTION:  This program rebuilds the frames of the scene
    that is under analysis from the output of the wave1
    program.
    FILES READ: Input file name given on command line or program
    prompt. The files strip.x and strip.y generated by the
    vbuild program are the coordinates that tell the program
    where to place the 1D time signals back in the frames.
    FILES WRITTEN:  A new set of frames with suffix .comb.
    HEADERS USED:   <stdio.h> <math.h> "jsmacros.h"
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:    NONE
    AUTHOR: J. Stewart Laing
    HISTORY: Initial Version.
*/
/*****************************************************************/
#include <stdio.h>
#include "jsmacros.h"
#include <math.h>
#define VECTORS  3490
#define FRAMES   32
#define LEVEL    3
float **matrix();
void  free_matrix();
float  *vector();
int   *ivector();
void  free_vector();
```

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

void main(argc, argv)
   int  argc;
   char *argv[];
{
   int    i,j,k,l, *X, *Y, size, n;
   char   infilename[64], filename[64];
   FILE   *infile, *outfile;
   float  *in, **temp;
   void   expd();

   /* load parameters */

   if(argc != 3 && argc != 1){
     printf("Usage: fbuild <filename> <N for NxN array of original image>\n");
     exit(0);
   }
   if(argc == 1){
      printf("\n\n\n  Input the size of the image (N for NxN array):");
      scanf("%d", &size);
      printf("  \n\n Input filename of signals:");
      fflush(stdout);
      scanf("%s", infilename); printf("\n");
   }
   else {
      sprintf(infilename, "%s", argv[1]);
      sscanf(argv[2], "%d", &size);
   }

   /* allocate memory */

   X = ivector(1, VECTORS);
   Y = ivector(1, VECTORS);
   in = vector(1, size*size*FRAMES);
   temp = matrix(1, VECTORS, 1, FRAMES);

   /* load strip locations */

   sprintf(filename, "strip.x");
   OPEN_FILE(infile, filename, "The Signal Stripper")
   loop1i(VECTORS) fscanf(infile, "%d\n", &X[i]);
   CLOSE_FILE(i, filename, "The Signal Stripper", infile)

   sprintf(filename, "strip.y");
   OPEN_FILE(infile, filename, "The Signal Stripper")
   loop1i(VECTORS) fscanf(infile, "%d\n", &Y[i]);
   CLOSE_FILE(i, filename, "The Signal Stripper", infile)

   /* load frames */

   loop1l(FRAMES){
      sprintf(filename, "%s%d.asc", infilename, l);
      OPEN_FILE(infile, filename, "The Signal Rebuilder")
      loop1i(size*size) fscanf(infile, "%f\n", &in[(l-1)*size*size+i]);
      CLOSE_FILE(i, filename, "The Signal Rebuilder", infile)
   } /* end l loop */

   /* load signals */

      n = FRAMES/(int)pow(2.0, (double)LEVEL);
      sprintf(filename, "%s.tsig.%d.c.db2", infilename, LEVEL);
      OPEN_FILE(infile, filename, "The Signal Rebuilder")
      loop1k1(VECTORS, n)
         fscanf(infile, "%f\n", &temp[k][1]);
      CLOSE_FILE(i,filename, "The Signal Rebuilder",infile)
```

```
   /* rebuild frames */
   loop1i(LEVEL){
      expd(temp, VECTORS, n);
      n *= 2;
   }
   loop1k1(VECTORS, FRAMES)
      in[(1-1)*size*size+Y[k]*size+X[k]] = temp[k][1];
   /* output frames */
   loop1l(FRAMES){
      sprintf(filename, "%s%d.asc.comb.3.c.db2", infilename, 1);
      CREATE_FILE(outfile, filename, "The Signal Rebuilder")
      loop1i(size*size) fprintf(outfile, "%f\n", in[(1-1)*size*size+i]);
      CLOSE_FILE(i, filename, "The Signal Rebuilder", outfile)
   }
   free_ivector(X, 1, VECTORS);
   free_ivector(Y, 1, VECTORS);
   free_vector(in, 1, size*size*FRAMES);
   free_matrix(temp, 1, VECTORS, 1, FRAMES);
} /* THE END */
void expd(in, rows, cols)
   int rows, cols;
   float **in;
{
   int i,j;
   float *yp, *tmp, *tab;
   void spline(); splint();
   yp = vector(1, cols);
   tab = vector(1, cols);
   tmp = vector(1, 2*cols);
   loop1i(cols) tab[i] = (float)(2*i);
   loop1i(rows){
      spline(tab, in[i], cols, 1.0e30, 1.0e30, yp);
      loop1j(cols){
         tmp[2*j] = in[i][j];
         splint(tab, in[i], yp, cols, (float)(2*j-1), &tmp[2*j-1]);
      }
      loop1j(2*cols) in[i][j] = tmp[j];
   }
   free_vector(yp, 1, cols);
   free_vector(tab, 1, cols);
   free_vector(tmp, 1, 2*cols);
} /* THE END */
```

D.2.7 Listing of TBLUR.C

```
/*****************************************************************************/
/***            ANIMATION PROGRAM FOR A SET OF FRAMES IN A SCENE          ***/
/*****************************************************************************/
/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  tblur.c
    DESCRIPTION:  This program animates the frames of a scene
    on the Silicon Graphics computers. There may be some code
    which is specific to the 4D series computers. To run the program
    just type tblur at the command prompt. Pressing the escape
```

```
    key will halt the program.
    FILES READ: The frames of the scene to be animated.
    FILES WRITTEN: NONE
    HEADERS USED:   <stdio.h> <device.h> <gl.h>
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:    NONE
    AUTHOR: J. Stewart Laing (with help from John Brunderman and Greg Tarr)
    HISTORY: Initial Version.
*/
/*****************************************************************************/
#include <device.h>
#include <gl.h>
#include <stdio.h>

#define SETX     256
#define SETY     256
#define XORG     200
#define YORG     200
#define FRAMES   32

main()
{
   int    i,j,k,l;
   long   twin, key;
   unsigned short  *image, **buffer;
   short  val;
   char   filename[64];
   FILE   *infile;

   prefposition(XORG, XORG+SETX-1, YORG, YORG+SETY-1);
   twin = winopen("Kanisza");
   doublebuffer();
   gconfig();

   /* read frames into buffer */

   buffer = (Colorindex **)calloc(FRAMES, sizeof(Colorindex *));
   for(i=0; i<FRAMES; i++)
      buffer[i] = (Colorindex *)calloc(SETX*SETY, sizeof(Colorindex));

   for(i=1; i<=FRAMES; i++){
         sprintf(filename, "frame%d.bin.comb", i);
         infile = fopen(filename, "r");
         if(infile == NULL){
            printf("Error opening %s as input file\n", filename);
            gexit();
 }
         else{
            fread(buffer[i-1], sizeof(unsigned short),(SETX*SETY), infile);
            fclose(infile);
            for(j=0;j<SETX*SETY;j++) if(buffer[i-1][j]!=0) buffer[i-1][j]=7;
         }
   }

   /* annimate new frames */

   qdevice(ESCKEY);

   while (TRUE) {
      while (!qtest()) {

         for(i=0; i<FRAMES; i++){
            rectwrite(0, 0, SETX-1, SETY-1, buffer[i]);
            swapbuffers();
         }
         for(i=FRAMES-1; i>0; i--){
            rectwrite(0, 0, SETX-1, SETY-1, buffer[i]);
            swapbuffers();
         }
```

```
        }
        /* get keyboard input */
        val = 1;
        while (val) {
           key = qread(&val);
           if(key!=ESCKEY) val = 0;
        }
        /* act on keyboard input */
        switch (key) {
        case ESCKEY:
           gexit();
           break;
        }
     }
}
```

Appendix E. *Software for the Boundary Contour Model*

E.1 System Description

The following programs are used in the boundary contour analysis of Chapter VIII:

1. *wave2* - The Multiresolution Wavelet Decomposition and Reconstruction program for two dimensional images (See Appendix B.1).
2. *lenrow* - A program used to perform a lateral excitation along the rows of a two dimensional array of wavelet detail coefficients.
3. *lencol* - A program used to perform a lateral excitation along the columns of a two dimensional array of wavelet detail coefficients.

In the analysis of Chapter VIII, we use the *lenrow* program to spread horizontal energy along the rows of the d1 and d3 wavelet detail coefficients (see explanation in Chapter III and VI). To run this program, type the following on the command line:

```
command prompt: lenrow <filename> <size>
```

The arguements are optional. If not entered on the command line, the program will prompt the user for them interactivley.

Similar to the *lenrow* program the *lencol* program performs a lateral excitation along the columns intended for the d1 and d3 wavelet detain coefficients. To run this program, type the following at the command prompt:

```
command prompt: lencol <filename> <size>
```

The arguements are optional. If not entered on the command line, the program will prompt the user for them interactivley.

The output of both programs is a file the same size as the input file whose name is made up of the input file name with a `.len` suffix added. Data is stored in ASCII format as discussed previously.

E.2 Boundary Contour Model Analysis Software

E.2.1 Listing of LENROW.C

```
/*****************************************************************************/
/*****************************************************************************/
/***              LATERAL EXCITATION ALONG ROWS                              **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  27 Sept 91
    VERSION: 1.0
    NAME:  lenrow.c
    DESCRIPTION:  This program performs a lateral excitation
    along the rows a the input ASCII data file. The extent of
    the receptive field is determined by #define P. This program
    is intended to be part of a Boundary Contour Model devised
    for the author's masters thesis.
    FILES READ: wavelet detail images as genterated by the wave2
    program.
    FILES WRITTEN:  The output file has the name of the input
    file with the suffix .len added.
    HEADERS USED:   <stdio.h>, "jsmacros.h", "stewmath.h",
    <math.h>
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:    NONE
    AUTHOR: J. Stewart Laing
    HISTORY: Initial Version.
*/
/*****************************************************************************/
/*****************************************************************************/

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

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

#define     F     -1.0
/*#define     P     6*/
#define     POS   1
#define     NEG   0

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

float **matrix();
void  free_matrix();
int   **imatrix();
void  free_imatrix();

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

   int         i,j,k,p, extent, **flag, P;
   FILE        *infile, *outfile;
   float_array input, output;
   char        infilename[64], latfile[64];
   float       latfactor = 0.0, influence;
```

```
   /***********************/
   /* test parameters     */
   /***********************/
   if(argc != 4 && argc != 1){
      printf("Usage: threshold <filename> <size>\n");
      exit(0);
   }
   /*************************************/
   /*  prompt for parameters if not input */
   /*************************************/
   if(argc == 1){
      printf("\n\n\n  Input the size of the input N for NxN:");
      scanf("%d", &input.ROW);
      printf("  \n\n What is the input filename?:"); fflush(stdout);
      scanf("%s", infilename);
      printf("\n Input the lateral extent of receptive field:");fflush(stdout);
      scanf("%d", &P);
    }
   /*****************************************/
   /*  use parameters given on command line  */
   /*****************************************/
    else {
       sprintf(infilename, "%s", argv[1]);
       sscanf(argv[2], "%d", &input.ROW);
       sscanf(argv[3], "%d", &P);
    }
   input.COL = input.ROW;
   input.array = matrix(1, input.ROW+2*P, 1, input.COL+2*P);
   loop1ij(input.ROW+2*P,input.COL+2*P) input.array[i][j] = 0.0;
   output.ROW = input.ROW;
   output.COL = input.COL;
   output.array = matrix(1, output.ROW+2*P, 1, output.COL+2*P);
   loop1ij(output.ROW+2*P,output.COL+2*P) output.array[i][j] = 0.0;
   flag = imatrix(1, input.ROW, 1, input.COL);
   loop1ij(input.ROW, input.COL) flag[i][j] = 1;
   /************************************/
   /*  open input file and read in data */
   /************************************/
   printf("\nreading data file...\n"); fflush(stdout);
   OPEN_FILE (infile, infilename, "The latnet")
   loop1ij(input.ROW, input.COL)
      fscanf(infile, "%f\n", &input.array[i+P][j+P]);
   CLOSE_FILE(i, infilename, "The latnet", infile)
   /*********************************/
   /*  prompt user for latnet factor */
   /*********************************/
/*      printf(" \n\n Input latnet factor(float):");
      scanf("%f", &latfactor);
      printf("\n\n Input latnet extent(integer):");
      scanf("%d", &extent);
*/
   /*********************************************************/
   /*  This part actually performs the lateral excitation. */
   /*********************************************************/
   printf("\nperforming lateral excitation...\n"); fflush(stdout);
   for(i=1+P; i<=input.ROW+P; i++)
      for(j=1+P; j<=inp  COL+P; j++){
```

```
            output.array[i][j] = input.array[i][j] - input.array[i][j];
            if(output.array[i][j]==0.0) flag[i-P][j-P] = POS;
            else flag[i-P][j-P] = NEG;
            input.array[i][j] = (float)fabs((double)input.array[i][j]);
         }
      for(p= -P; p<= -1; p++)
         latfactor += (float)pow(2.0,(double)p);
      for(p=1; p<=P; p++)
         latfactor += (float)pow(2.0, -(double)p);
      for(i=1+P; i<=output.ROW+P; i++)
         for(j=1+P; j<=output.COL+P; j++){
            output.array[i][j] = latfactor * input.array[i][j];
            for(p= -P; p<= -1; p++)
               output.array[i][j]-=input.array[i][j+p]*(float)pow(2.0,(double)p);
            for(p=1; p<=P; p++)
               output.array[i][j]-=input.array[i][j+p]*(float)pow(2.0,-(double)p);
            output.array[i][j] *= F;
         }
      for(i=1+P; i<=output.ROW+P; i++)
         for(j=1+P; j<=output.COL+P; j++)
            if(flag[i-P][j-P]==NEG) output.array[i][j] = -output.array[i][j];
      /*********************************/
      /*  Create file and output data.*/
      /*********************************/
      printf("\nwriting output data file...\n"); fflush(stdout);
      sprintf(latfile, "%s.len", infilename);
      CREATE_FILE(outfile, latfile, "The Thresholder")
      loop1ij(output.ROW,output.COL)
         fprintf(outfile, "%f\n", output.array[i+P][j+P]);
      CLOSE_FILE(i, latfile, "The latnet",outfile)
      /*****************************************************/
      /*  Tell the user where the output file is located. */
      /*****************************************************/
      printf("\nCreated new file called: %s\n\n", latfile);
      free_matrix(input.array,1,input.ROW+2*P,1,input.COL+2*P);
      free_matrix(output.array,1,output.ROW+2*P,1,output.COL+2*P);
      free_imatrix(flag, 1, input.ROW, 1, input.COL);
} /* THE END */
```

E.2.2 Listing of LENCOL.C

```
/*************************************************************************/
/*************************************************************************/
/***              LATERAL EXCITATION ALONG COLUMNS                     **/
/*************************************************************************/
/*************************************************************************/
/*  DATE:  27 Sept 91
    VERSION: 1.0
    NAME:  lencol.c
    DESCRIPTION:  This program performs a lateral excitation
    along the cols a the input ASCII data file. The extent of
    the receptive field is determined by #define P. This program
    is intended to be part of a Boundary Contour Model devised
    for the author's masters thesis.
    FILES READ: wavelet detail images as genterated by the wave2
    program.
    FILES WRITTEN:  The output file has the name of the input
    file with the suffix .len added.
```

```
    HEADERS USED:   <stdio.h>, "jsmacros.h", "stewmath.h",
    <math.h>
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:    NONE
    AUTHOR: J. Stewart Laing
    HISTORY: Initial Version.
*/
/*****************************************************************************/
/*****************************************************************************/

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

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

#define     F     -1.0
/*#define     P      6*/
#define     POS   1
#define     NEG   0

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

float **matrix();
void  free_matrix();
int   **imatrix();
void  free_imatrix();

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

   int         i,j,k,p, extent, **flag, P;
   FILE        *infile, *outfile;
   float_array input, output;
   char        infilename[64], latfile[64];
   float       latfactor = 0.0, influence;

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

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

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

   if(argc == 1){
      printf("\n\n\n  Input the size of the input N for NxN:");
      scanf("%d", &input.ROW);
      printf("  \n\n What is the input filename?:"); fflush(stdout);
      scanf("%s", infilename);
      printf("\n Input the lateral extent of receptive field:");fflush(stdout);
      scanf("%d", &P);
    }

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

```
   /*  use parameters given on command line  */
   /****************************************/
    else {
       sprintf(infilename, "%s", argv[1]);
       sscanf(argv[2], "%d", &input.ROW);
       sscanf(argv[3], "%d", &P);
    }
   input.COL = input.ROW;
   input.array = matrix(1, input.ROW+2*P, 1, input.COL+2*P);
   loop1ij(input.ROW+2*P,input.COL+2*P) input.array[i][j] = 0.0;
   output.ROW = input.ROW;
   output.COL = input.COL;
   output.array = matrix(1, output.ROW+2*P, 1, output.COL+2*P);
   loop1ij(output.ROW+2*P,output.COL+2*P) output.array[i][j] = 0.0;
   flag = imatrix(1, input.ROW, 1, input.COL);
   loop1ij(input.ROW, input.COL) flag[i][j] = 1;
   /***********************************/
   /*  open input file and read in data */
   /***********************************/
   printf("\nreading data file...\n"); fflush(stdout);
   OPEN_FILE (infile, infilename, "The latnet")
   loop1ij(input.ROW, input.COL)
      fscanf(infile, "%f\n", &input.array[i+P][j+P]);
   CLOSE_FILE(i, infilename, "The latnet", infile)
   /*********************************/
   /*  prompt user for latnet factor */
   /*********************************/
/*      printf(" \n\n Input latnet factor(float):");
      scanf("%f", &latfactor);
      printf("\n\n Input latnet extent(integer):");
      scanf("%d", &extent);
*/
   /*************************************************************/
   /*  This part actually performs the lateral excitation. */
   /*************************************************************/
   printf("\nperforming lateral excitation...\n"); fflush(stdout);
   for(i=1+P; i<=input.ROW+P; i++)
      for(j=1+P; j<=input.COL+P; j++){
         output.array[i][j] = input.array[i][j] - input.array[i][j];
         if(output.array[i][j]==0.0) flag[i-P][j-P] = POS;
         else flag[i-P][j-P] = NEG;
         input.array[i][j] = (float)fabs((double)input.array[i][j]);
      }
   for(p= -P; p<= -1; p++)
      latfactor += (float)pow(2.0,(double)p);
   for(p=1; p<=P; p++)
      latfactor += (float)pow(2.0, -(double)p);
   for(i=1+P; i<=output.ROW+P; i++)
      for(j=1+P; j<=output.COL+P; j++){
         output.array[i][j] = latfactor * input.array[i][j];
         for(p= -P; p<= -1; p++)
            output.array[i][j]-=input.array[i+p][j]*(float)pow(2.0,(double)p);
         for(p=1; p<=P; p++)
            output.array[i][j]-=input.array[i+p][j]*(float)pow(2.0,-(double)p);
         output.array[i][j] *= F;
      }
   for(i=1+P; i<=output.ROW+P; i++)
      for(j=1+P; j<=output.COL+P; j++)
         if(flag[i-P][j-P]==NEG) output.array[i][j] = -output.array[i][j];
```

```
   /*******************************/
   /*  Create file and output data.*/
   /*******************************/
   printf("\nwriting output data file...\n"); fflush(stdout);
   sprintf(latfile, "%s.len", infilename);
   CREATE_FILE(outfile, latfile, "The Thresholder")
   loop1ij(output.ROW,output.COL)
      fprintf(outfile, "%f\n", output.array[i+P][j+P]);
   CLOSE_FILE(i, latfile, "The latnet",outfile)
   /*****************************************************/
   /*  Tell the user where the output file is located. */
   /*****************************************************/
   printf("\nCreated new file called: %s\n\n", latfile);
   free_matrix(input.array,1,input.ROW+2*P,1,input.COL+2*P);
   free_matrix(output.array,1,output.ROW+2*P,1,output.COL+2*P);
   free_imatrix(flag, 1, input.ROW, 1, input.COL);
} /* THE END */
```

Appendix F. *Software for Utilities*

F.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. `ascii2byte.c` - A program that converts an ascii data file to a binary data file. Data is read in as integer decimal with the fscanf function and converted to unsigned short written out with the fread function. The ascii file must be integers ranging from 0 to 1023 with each number on a separate row of the data file.
5. `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.
6. `byte2ascii.c` - A program that converts an input file from binary format in which an array of numbers is stored successively as unsigned short to ASCII format in which the output is made up of one number per line of decimal integers.
7. `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.

8. `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.

9. `expand.c` - A program that performs a square expansion on a square array of ASCII values. The input values can be integer or float and the output values are float. The expansion is via a "Bi-cubic Spline Interpolation" [35].

10. `matrixtoascii.c` - A program that converts a Khoros ASCII output to a file that has one integer per line [37]. This program strips off the matrix coordinates of the values.

11. `nrutil.c` - A set of utilities provided by *Numverical Recipies in C* used in this thesis mostly for dynamic memory allocation [35].

12. `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.

F.2 Spatial-Temporal Analysis Software

F.2.1 Listing of JSMACROS.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;
```

F.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 fskip_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
```

F.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;
}
```

F.2.4 Listing of Modified WAVE1 Modules

F.2.4.1 Listing of ASCII2BYTE.C

```
/*****************************************************************************/
/*****************************************************************************/
/***            ASCII to BYTE CONVERTER                                    **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  ascii2byte.c
    DESCRIPTION:  This routine converts an image from ascii
    in which each pixel's gray scale value (0 to 255) is stored
    on one row of the file to byte format in which the grey
    scale values are logically stored in consecutive bytes in
    the file.
    FILES READ: One file specified by the user.
    FILES WRITTEN:  One file specified by the user.
    HEADERS USED:   <stdio.h>,  "jsmacros.h"
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:   uses nrutil.c from Numerical Recipies
    AUTHOR: J. Stewart Laing
    HISTORY: Initial Version
*/
/*****************************************************************************/
/*****************************************************************************/
/*  DECLARATION SECTION  */
#include    <stdio.h>
#include    "jsmacros.h"
int         **imatrix();
void     free_imatrix();
```

```
/*  FUNCTION BODY       */
void main(argc, argv)
   int    argc;
   char   *argv[];
{
   /* initialize variables */
   int       i,j, rows, cols;
   FILE      *infile, *outfile;
   char      infilename[64], filename[64], outfilename[64];
   unsigned  amount;
   unsigned short *image;
   /*  parse command line  */
   switch (argc){
   case 1:
      printf("Input filename:");
      scanf("%s", infilename);
      printf("\nOutput filename:");
      scanf("%s", outfilename);
      printf("\n# of ROWS:");
      scanf("%d", &rows);
      printf("\n# of COLS:");
      scanf("%d", &cols);
      printf("\n");
      break;
   case 2:
      sprintf(infilename, "%s", argv[1]);
      sprintf(outfilename, "%s.bin", infilename);
      printf("\n# of ROWS:");
      scanf("%d", &rows);
      printf("\n# of COLS:");
      scanf("%d", &cols);
      printf("\n");
      break;
   case 3:
      sprintf(infilename, "%s", argv[1]);
      sprintf(outfilename, "%s", argv[2]);
      printf("\n# of ROWS:");
      scanf("%d", &rows);
      printf("\n# of COLS:");
      scanf("%d", &cols);
      printf("\n");
      break;
   case 5:
      sprintf(infilename, "%s", argv[1]);
      sprintf(outfilename, "%s", argv[2]);
      sscanf(argv[3], "%d", &rows);
      sscanf(argv[4], "%d", &cols);
      break;
   default:
      printf("Usage: ascii2byte [infilename] [outfilename]");
      printf(" [# of rows] [# of cols]\n");
      printf("Note: arguments are optional; but, position is");
      printf(" critical.\n");
      exit(0);
   }
   image = (unsigned short *)calloc(rows*cols, sizeof(unsigned short));
   /*  read ascii format  */
   /* printf("reading...\n"); fflush(stdout);  */
   OPEN_FILE (infile, infilename, "The ascii2byte Converter");
   for(i=0;i<rows*cols;i++) fscanf(infile, "%hu\n", &image[i]);
```

```
   CLOSE_FILE (i, infilename, "The ascii2byte Converter", infile)
   /*  write byte format   /
   /* printf("writing...\n"); fflush(stdout);  */
   CREATE_FILE(outfile, outfilename, "The ascii2byte Converter")
   amount = fwrite(image, sizeof(unsigned short), rows*cols, outfile);
   CLOSE_FILE(i, filename, "The ascii2byte Converter", outfile)
   /* free memory */
   free(image):
}/* THE END */
```

F.2.4.2 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 coefficent 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");
}
```

F.2.5 Listing of BYTE2ASCII.C

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

```
/*************************************************************************/
/***            BYTE to ASCII CONVERTER                                ***/
/*************************************************************************/
/*************************************************************************/
/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  byte2ascii.c
    DESCRIPTION:  This routine converts an image from byte
    format in which the gray scall values (0 to 255) are
    logically stored in consecutive bytes in the file to ascii
    format in which each pixel's grey scale value (0 to 255) is
    stored on one row of the file.
    FILES READ: One file specified by the user.
    FILES WRITTEN:  One file specified by the user.
    HEADERS USED:   <stdio.h>,  "jsmacros.h"
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:   uses nrutil.c from Numerical Recipies
    AUTHOR: J. Stewart Laing
    HISTORY: Initial Version
*/
/*************************************************************************/
/*************************************************************************/

/*  DECLARATION SECTION  */

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

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

/*  FUNCTION BODY       */

void main(argc, argv)
   int    argc;
   char   *argv[];
{
   /* initialize variables */
   int       i,j, rows, cols;
   FILE      *infile, *outfile;
   char      infilename[64], filename[64], outfilename[64];
   unsigned  amount;
   unsigned short *image;

   /*  parse command line  */
   switch (argc){
   case 1:
      printf("Input filename:");
      scanf("%s", infilename);
      printf("\nOutput filename:");
      scanf("%s", outfilename);
      printf("\n# of ROWS:");
      scanf("%d", &rows);
      printf("\n# of COLS:");
      scanf("%d", &cols);
      printf("\n");
      break;
   case 2:
      sprintf(infilename, "%s", argv[1]);
      sprintf(outfilename, "%s.asc", infilename);
      printf("\n# of ROWS:");
      scanf("%d", &rows);
      printf("\n# of COLS:");
```

```
      scanf("%d", &cols);
      printf("\n");
      break;
   case 3:
      sprintf(infilename, "%s", argv[1]);
      sprintf(outfilename, "%s", argv[2]);
      printf("\n# of ROWS:");
      scanf("%d", &rows);
      printf("\n# of COLS:");
      scanf("%d", &cols);
      printf("\n");
      break;
   case 5:
      sprintf(infilename, "%s", argv[1]);
      sprintf(outfilename, "%s", argv[2]);
      sscanf(argv[3], "%d", &rows);
      sscanf(argv[4], "%d", &cols);
      break;
   default:
      printf("Usage: byte2ascii [infilename] [outfilename]");
      printf(" [# of rows] [# of cols]\n");
      printf("Note: arguments are optional; but, position is");
      printf(" critical.\n");
      exit(0);
   }
   image = (unsigned short *)calloc(rows*cols, sizeof(unsigned short));
   /*  read byte format  */
   /* printf("reading...\n"); fflush(stdout);  */
   OPEN_FILE (infile, infilename, "The byte2ascii Converter");
   amount = fread(image, sizeof(unsigned short), rows*cols, infile);
   CLOSE_FILE (i, infilename, "The byte2ascii Converter", infile)
   /*  write ascii format  */
   /* printf("writing...\n"); fflush(stdout);  */
   CREATE_FILE(outfile, outfilename, "The byte2ascii Converter")
   loopi(rows*cols) fprintf(outfile, "%hu\n", image[i]);
   CLOSE_FILE(i, filename, "The ascii2byte Converter", outfile)
   /* free memory */
   free(image);
}/* THE END */
```

F.2.6 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 compilation. 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 .3965393195;
    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 -.0165745416;
    if(n == 10) return .0125509986;
    if(n == 11) return .0004295780;
    if(n == 12) return -.0018016407;
    if(n == 13) return .0003537138;
    else return 0.0;
    }
  if(N == 8) {
    if(n == 0) return .0544158422;
    if(n == 1) return .3128715909;
    if(n == 2) return .6756307363;
    if(n == 3) return .5853546837;
    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 .0018476469;
    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,1,((2*j)-i));
                }
  fprintf(outfile, "%.9f\n",1.414212562*temp_sum);
    }
CLOSE_FILE(i, filename, "The Daub routine", outfile);
printf("\n");
}
```

F.2.7 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     */
/*  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       */
/************************/

/*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); */
}
```

F.2.8 Listing of EXPAND.C

```
/*****************************************************************************/
/*****************************************************************************/
/***            ARRAY EXPADER PROGRAM                                      **/
/*****************************************************************************/
/*****************************************************************************/
/*  DATE:  3 Sept 91
    VERSION: 1.0
    NAME:  expand.c
    DESCRIPTION:  This program expands a square power of 2 size array
    by a factor specified interactively by the user. It uses a "bi-cubic
    spline interpolation routine from Numerical Recipies in C.
    FILES READ: One file specified by the user.
    FILES WRITTEN:  One file specified by the user.
    HEADERS USED:   <stdio.h>,  "jsmacros.h" , <math.h>
    CALLING PROGRAMS:   NONE
    PROGRAMS CALLED:   uses nrutil.c from Numerical Recipies
    AUTHOR: J. Stewart Laing
    HISTORY: Initial Version
*/
/*****************************************************************************/
/*****************************************************************************/

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

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

void main(argc, argv)
   int argc;
   char *argv[];
{
   /************************/
   /* initialize variables */
   /************************/
   int       i,j, factor;
   FILE      *infile, *outfile;
   float_array in, out;
   char      infilename[64], expandfile[64];
```

```
   void      expand0(), expand1(), expand2();
 /***********************/
/* test parameters     */
/***********************/
if(argc != 3 && argc != 1){
   printf("Usage: threshold <filename> <N for NxN>\n");
   exit(0);
}
/*************************************/
/*  prompt for parameters if not input */
/*************************************/
if(argc == 1){
   printf("\n\n\n  Input the size of the square in (ROW/COLUMN):");
   scanf("%d", &in.ROW);
   printf("  \n\n Input filename of in to be expanded:"); fflush(stdout);
   scanf("%s", infilename);
 }
/*****************************************/
/*  use parameters given on command line  */
/*****************************************/
 else {
    sprintf(infilename, "%s", argv[1]);
    sscanf(argv[2], "%d", &in.ROW);
 }
in.COL = in.ROW;
/************************************/
/*  prompt user for expansion factor */
/************************************/
 printf(" \n\n Input expansion factor:");
   scanf("%d", &factor);
/**********************************/
/*  create a matrix to hold the in */
/**********************************/
out.ROW = in.ROW*factor;
out.COL = out.ROW;
in.array = matrix(1, in.ROW, 1, in.COL);
out.array = matrix(1, out.ROW, 1, out.COL);
/************************/
/*  open input file     */
/************************/
OPEN_FILE (infile, infilename, "The Expander");
loop1ij(in.ROW, in.COL) fscanf(infile, "%f\n", &in.array[i][j]);
CLOSE_FILE(i, infilename, "The Expander", infile)
/*****************************/
/* call expansion routines  */
/*****************************/
expand1(in.array, in.ROW, out.ROW, out.array);
/************************************************/
/*  Create file and output the expanded array. */
/************************************************/
sprintf(expandfile, "%s.exp", infilename);
CREATE_FILE(outfile, expandfile, "The Expander");
loop1ij(out.ROW, out.COL) fprintf(outfile, "%f\n", out.array[i][j]);
CLOSE_FILE(i, expandfile, "The Expander", outfile)
/*************************************************/
/*  Tell the user where the output file is located. */
```

```
printf("\nI have expanded %s by a factor of %d and saved it in %s:\n\n",
                                                infilename, factor, expandfile);

/* THE END */
}

void expand0(in, small, big, out)
   int small, big;
   float **in, **out;
{
int i,j,k,l, factor;
float *tab, **yp;
void splin2(), spline();

tab = vector(1, small);
yp = matrix(1, small, 1, small);

factor = big/small;
loop1i(small) tab[i] = factor*i;
loop1i(small) spline(tab, in[i], small, 1.0e30, 1.0e30, yp[i]);
loop1ij(small,small)
   loopkl(factor, factor)
      splin2(tab,tab,in,yp,small,small,(float)(factor*i-k),
 (float)(factor*j-l),&out[factor*i-k][factor*j-l]);

free_vector(tab, 1, small);
free_matrix(yp, 1, small, 1, small);
}

void expand2(in, small, out)
   int small;
   float **in, **out;
{
int i,j,k,l, factor;
float **yp, *tmp, *ptmp, *tab;
void spline(), splint();

yp = matrix(1, small, 1, small);
tab = vector(1, small);
tmp = vector(1, small);
ptmp = vector(1, small);

loop1i(small) tab[i] = 2*i;
loop1i(small) spline(tab, in[i], small, 1.0e30, 1.0e30, yp[i]);
loop1ij(small, small){
   out[2*i][2*j] = in[i][j];
   splint(tab, in[i], yp[i], small, (float)(2*j-1), &out[2*i][2*j-1]);
   }
loop1ij(small, small*2){
   loop1k(small) tmp[k] = out[2*k][j];
   spline(tab, tmp, small, 1.0e30, 1.0e30, ptmp);
   splint(tab, tmp, ptmp, small, (float)(2*i-1), &out[2*i-1][j]);
   }

free_matrix(yp, 1, small, 1, small);
free_vector(tmp, 1, small);
free_vector(ptmp, 1, small);
free_vector(tab, 1, small);

}

void expand1(in, small, big, out)
   int small, big;
   float **in, **out;
{
int i,j,k,l, factor, index;
float **tmp;
```

```
tmp = matrix(1,big,1,big);
factor = big/small;
index = (int)(log((double)factor)/log(2.0));
loop1ij(small,small) tmp[i][j] = in[i][j];
loop1i(index){
   expand2(tmp, small, out);
   small *= 2;
   loop1kl(small, small) tmp[k][l] = out[k][l];
   }
free_matrix(tmp, 1, big, 1, big);
}
```

F.2.9 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, hold1, 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]);
        }
}
```

F.2.10 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));
}
```

F.2.11 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 */
}
```

Bibliography

1. Antonini, M. and others. "Image Coding Using Vector Quantization in the Wavelet Transform Domain." *Proceedings of IEEE International Conference in ASSP*. 2297-2300. 1990.
2. 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).
3. Cohen, A. and J. M. Schlenker. "Compactly Supported Bidimensional Wavelet Bases with Hexagonal Symmetry." AT&T Bell Laboratories, Preprint, 1991.
4. Cohen, I. "Time-Frequency Distributions - A Review," *Proceedings of the IEEE* (July 1989).
5. Combes, J. and others. *Time-Frequency Methods and Phase Space* (2 Edition), 21-37. Berlin: Springer-Verlag, 1989.
6. Daubechies, Ingrid. "Orthonormal Bases of Compactly Supported Wavelets," *Communications on Pure and Applied Mathematics*, *41*:909-996 (1988).
7. Daubechies, Ingrid. "Orthonormal Bases of Wavelets with Finite Support - Connection with Discrete Filters." AT&T Bell Laboratories, Preprint, 1990.
8. 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).
9. 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).
10. Fastman, Inc. *The Wavelet Handbook*. Technical Report, Defence Advanced Research Projects Agency, 1991 (AD-B151 677).
11. Gabor, D. "Theory of communication," *The Journal of the Institution of Electrical Engineers*, *93*:429-457 (1946).
12. Gabor, D. "New Possibilities in Speech Transmission," *The Journal of the Institution of Electrical Engineers*, *94*:369-390 (1947).
13. Ginsburg, A. "Specifying relevant spatial information for image evaluation and display design: An explanation of how we see certain objects," *Proceedings of the Society of Information Display*, *21*:219-227 (1987).
14. Ginsburg, Arthur. *Visual Information Processing Based on Spatial Filters Constrained by Biological Data*. Technical Report AMRL-TR-78-129, Volumes I and II, Aerospace Medical Research Laboratory, December 1978.

15. Gonzalez, Rafael C. and Paul Wintz. *Digital Image Processing* (2 Edition). Massachusetts: Addison-Wesley Publishing Company, Inc., 1987.

16. Grossberg, Stephen. *Mathematical Psychology and Psychophysiology.* Philadelphia: American Mathematical Society, 1980.

17. Grossberg, Stephen. *Neural Networks and Natural Intelligence.* Cambridge, Massachusetts: MIT Press, 1988.

18. Grossberg, Stephen and Ennio Mingolla. "Nerual Dynamics of Perceptual Grouping," *Perception and Psychophysics*, *38*(2):141–171 (1985).

19. Grossberg, Stephen and Ennio Mingolla. "The Role of Illusory Contours in Visual Segmentation." *The Perception of Illusory Contours* edited by Susan Petry and Glenn E. Meyer, chapter 12, 116–125, Springer-Verlag, 1987.

20. Guyton, Arthur. *Textbook of Medical Physiology.* Philadelphia: W. B. Saunders Company, 1976.

21. Hubel, D. and T. Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *Journal of Physiology*, *160*:106–154 (1962).

22. Hubel, David H. "The Visual Cortex of The Brain," *Scientific American*, *209*(5):54–62 (November 1963).

23. Hubel, David H. and Torsten N. Wiesel. "Brain Mechanisms of Vision," *Scientific American*, *241*(3):150–162 (September 1979).

24. Jones, Judson P. and Larry A. Palmer. "An Evaluation of the Two-Dimensional Gabor Filter Model of Simple Receptive Fields in Cat Striate Cortex," *Journal of Neurophysiology*, *58*(6):1233–1258 (December 1987).

25. Jr., Thomas G. Stockham. "Image Processing in the Context of a Visual Model," *Proceedings of the IEEE*, *60*(7):828–842 (July 1972).

26. Lewis, A. S., G. Knowles. "Video Compression using 3D Wavelet Transforms," *Electronic Letters*, *26*(6):396–398 (March 1990).

27. Mallat, Stephane G. "Multifrequency Channel Decompositions of Images And Wavelet Models," *ASSP*, *37*(12):2091–2109 (December 1989).

28. 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).

29. Mallat, Stephane G. "Zero-Crossings of a Wavelet Transform," *IEEE Transactions on Information Theory*, *37*(4):1019–1033 (July 1991).

30. Marr, David. *Vision.* New York: Freeman, 1982.

31. Norman, Ralph. *Contmporary Theory and Research in Visual Perception.* New York: Holt, Rinehart and Winston, Inc, 1968.

32. Oberndorf, Richard A. *Analysis of Visual Illusions Using Gabor Filters.* MS thesis, AFIT/GE/ENG/90D-47, Air Force Institute of Technology, 1990.

33. Ozawa, Kazumasa. "Simulation of the Optical Illusions Using a Spatial Filter," *Pattern Recognition*, *1*:237–242 (1978).

34. Parker, Donald E. and others. "Illusory Displacement of a Moving Trace with Respect to the Grid During Oscilloscope Motion," *Perception and Psychophysics*, *21*(5):439–444 (1977).

35. Press, William H. and others. *Numerical Recipes in C, The Art of Scientific Computing.* Cambridge University Press, 1988.

36. Ranganath, Surendra. "Image Filtering Using Multiresolution Representations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *13*(5):426–440 (May 1991).

37. Rasure, John and Danielle Argiro. *Khoros Users Manual.* Cambridge University Press, 1988.

38. Resnikoff, Howard L. *The Illusion of Reality.* New York: Springer-Verlag, 1989.

39. Ruch, T. C. and J. F. Fulton. *Medical Physiology and Biophysics.* Philadelphia: W. B. Saunders Company, 1960.

40. Shapiro, H. S. and others. "Uncertainty Principles for Basis in $\mathbf{L}^2(\mathbf{R})$." Prometheus Inc., Preprint, 1991.

41. Smiley, Steven E. *Image Segmentation Using Affine Wavelets.* MS thesis, AFIT/GE/ENG/91D-50, Air Force Institute of Technology, 1991.

42. Uz, K. Metin and others. "Interpolative Multiresolution Coding of Advanced Television with Compatible Subchannels," *IEEE Transactions on Circuits and Systems for Video Technology*, *1*(1):86–99 (March 1991).

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
ANALYSIS OF VISUAL ILLUSIONS USING MULTIRESOLUTION WAVELET DECOMPOSITION BASED MODELS

5. FUNDING NUMBERS

6. AUTHOR(S)
John S. Laing, 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/ENG/91D-34

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

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 word(s))

This thesis provides alternatives to the explanation that spatial filtering is responsible for the perception of illusory contours in the Kanisza Triangle illusion. Specifically, we use a Multiresolution Wavelet Decomposition to divide an image into spatial-frequency bands that are used as inputs to three biologically motivated models. The thesis includes a brief tutorial of Wavelet theory and an in-depth explanation of our implementation of recently published algorithms for Multiresolution Wavelet Analysis. The first model is based on the saccadic movements of the human eye. It demonstrates the importance of the high spatial-frequency content of an image in the formulation of the illusion. The second model is based on the serial architecture of the data transmission channel between the retina and the visual cortex of the brain. It demonstrates the importance of low temporal-frequency characteristics of the build-up of the visual world model. The third model considers only the high spatial-frequency content of the image. It consists of lateral excitation networks that serve to simulate the local high spatial-frequency energy interactions that contribute to illusory contours.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Wavelets, Multiresolution Analysis, Human Visual System, World Model	250
	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