Orientational Filters For Real–Time Computer Vision Problems

A Thesis
Presented to
The Academic Faculty

by

Toshiro Kubota

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in Electrical and Computer Engineering

Georgia Institute of Technology
December 1995

Table of Contents

Appendix

List of Tables

List of Figures

Summary

Orientational filters have been used to solve many computer vision problems. However, their use in real–time applications has been limited due to high computational load of the filters. This thesis investigates the computational scheme of the filters and proposes an efficient and inexpensive way of computing and implementing them. The proposed scheme is called *Separable Approximation*. Using the scheme, non–separable orientational filters are decomposed into a sum of separable filters. The scheme can be implemented efficiently using a filter bank structure. Two algorithms are developed for decomposing filters into a separable form. They are *Orthogonal Sequence Decomposition* and *Singular Value/Orthogonal Sequence Decomposition* (SV/OSD). The thesis shows that SV/OSD achieves the best performance/implementation trade–off.

The second part of the thesis is concerned with multi–resolution image decomposition. An efficient decomposition method is developed, which is a combination of SV/OSD and a functional approximation using a basic spline. The thesis shows that the method can perform the decomposition with small amount of error, and the amount of computation and hardware required for the decomposition is much less than the direct implementation.

The third part of the thesis is concerned with a steerable system where the orientation of the filter can be dynamically changed in real–time. The system can be constructed effectively using SV/OSD and *Fourier Series Approximation*.

Finally the thesis proposes a VLSI architecture for an efficient orientational filter system with multi–resolution decomposition capability and steerability.

CHAPTER 1

The Problem

In the last two decades, an immense amount of research has been conducted on how to construct a machine system which is capable of seeing and understanding a visual scene as well as humans. This problem is referred to as the computer vision problem. The problem has two major sub–problems; (1) to identify some particular object in a scene, or (2) to understand and describe all the objects in a scene. Computer vision problems are usually grouped with the most computationally intensive problems; global weather modeling, fluid turbulence and molecular dynamics[14]. The computational requirement for a vision system can be estimated by assuming that 1024x1024 pixels will be processed at a rate of 30 frames/sec. This system will need to process 30 million data elements (pixels) per second. If one thousand operations have to be performed on each pixel to fulfill the goal of understanding the scene, this will require 30 billion operations per second. For most systems, the number of operations per pixel will be higher by a factor of 10 to 100 leading to an estimated requirement of computational power between 100 and 1,000 billion operations per second[14]. To build such a system will require parallel processing and special hardware.

Another difficulty in computer vision is that the problem is not well–posed from a computational perspective. For example, a system must detect a chair from image sequences which represent dynamic views inside a room. A simple pattern matching algorithm which compares an image with a small template representing the chair does not work, since the chair can be any size and at any orientation relative to the image frame. The chair can also be partially blocked by other objects (occlusion), and obscured by lighting and signal noise. Thus input images representing the scene have to be processed in such a way that the result produces a data representation of each object which is minimally dependent on an affine transformation, partial occlusion and noise. This representation is often expressed as a collection of characteristic called s"*features*". Edges[43][58], corners[61] and texture[19][37] have been suggested as good features for image analysis.

A reasonable approach for attacking the problem is to apply the human visual processing mechanism to computer vision algorithms.[57][88][89][93][98] This interaction of neuro–physiological research and computer engineering not only produces better vision algorithms, but also helps understand the mechanism of the human visual system. However, only a small portion of human visual processing is understood. Thus, no consensus has been established on how to solve vision problems.

In order to make the vision problem tractable, the problem is divided into three levels of processing: low, intermediate, and high.[14][62][73][81] Low level processing takes pixel data as inputs and extracts primitive features such as edges, texture, depth map, and optical flow. This level of processing is mostly regular and data–independent, requiring numeric operations on huge amounts of pixel data. The intermediate level takes the primitive features generated from low level processing and extracts more meaningful features such as surfaces and contours. This level of processing is data dependent and irregular. The computations are often both symbolic and numeric. High level processing interacts with the database of objects to determine types of objects in the image. This level of processing is highly data–dependent and very diverse. The focus of this research is on the low level part of computer vision processing. As stated above, the processing at this level requires very large amounts of numeric computation in a regular and data–independent form. The input data rate can easily exceed 30 million pixels/sec. Because of the high input rate and the quantity of operations involved, a hardware based algorithm is needed.

In low–level vision processing, two types of information have to be extracted from images simultaneously. One is a feature type which can be characterized well in the frequency domain. The second is the location of the feature in the spatial domain. Taking a Fourier transform of the image is not acceptable since it loses all the spatial information and the location of features cannot be determined. In order to extract the information from two domains, spatial–frequency analysis needs to be performed on images. Also, in order to extract features of various orientations, the spatial–frequency analysis must have directional selectivity. In this thesis, an operator which performs spatial–frequency analysis with directional selectivity is called an *orientational filter*. The operator can be performed in either the spatial domain or the frequency domain. Many such operators appear in computer vision research. Some examples are the windowed Fourier transform, Gabor filters and Gaussian derivatives. Figure 1–1 illustrates the operation of orientational filters. The simple test image has two features: a bar oriented at approximately 45 degrees and another bar oriented at approximately 135 degrees. The test image is processed using two orientational filters; the real part of Gabor filter oriented at 45 degrees and 135 degrees respectively. As can be seen, the two filters could extract the edges of the bars and their location precisely.

According to neuro–physiological research, the very early stage of processing in the human visual system performs a localized frequency analysis with directional selectivity[2][28][32][39][51][56][63][95]. (Appendix A gives a more detailed

description of the human visual system.) The research suggests that the spatial–frequency analysis is an essential part of the early stage processing, and the later stage processing is often performed on top of the decomposition[32][39][51][57][92][95]. It has been demonstrated by many researchers in computer science that orientational filters are indeed useful for many image analysis tasks including texture analysis[9][88][102][44], texture segmentation[41][52][68], edge detection[17][53], contour following[98][78], shape analysis[42], stereo analysis[100] and motion detection[23][8]. (Section 2.3 describes the use of orientational filters in image analysis applications.) For these reasons, this research assumes that orientational filters can be the core of the low–level part of a computer vision system if efficient algorithms and implementation schemes to perform the decomposition can be found.

As seen in Figure 1–1, many filters tuned to different orientations would be needed to detect features with various orientations. In order to detect features with various sizes, filters with different sizes are needed. Thus, a computer vision system must contain many filters tuned to different frequency regions in order to be flexible enough to solve different problems. Therefore it is important to investigate an inexpensive implementation scheme of each filter because the implementation cost of the system depends heavily on the implementation cost of each filter. In order to minimize cost without losing flexibility, it is essential to be able to tune the filter to a particular frequency response dynamically from frame to frame, or even within a frame. The system can then be adapted to a particular problem and to changes of input images.

1.1.Objectives of The Research

1.1.1.Primary Objective

The primary objective of this research is to develop an efficient algorithm and implementation scheme for orientational filters suitable for real–time computer vision applications. Real–time applications require processing of many pixels (high throughput) and producing outputs in a reasonable time (low latency) using a system which is inexpensive to implement[46]. In order to attack the problem in a constructive manner, these requirements are formulated in such a way that they are independent of technology and applications. These requirements are called *implementation criteria* in this thesis.

The following assumptions are made:

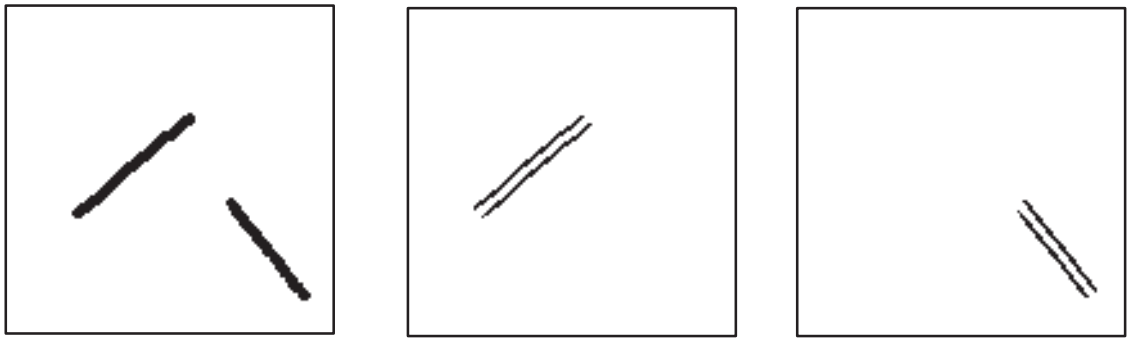
1) Input images consists of NxN pixels, and they are processed in a scan line order.
2) The number of filters in the system is F_N.
3) For simplicity, all the filters have the same size, MxM where $M << N$.
4) The critical time, t_m, is the longest operation time between one multiply–accumulate operation and one memory write/read operation. These two operations govern the speed of a real–time filter system in current VLSI technology.

Based on these assumption, the implementation criteria can be defined as the following.

1. High throughput

The system must be capable of handling an input rate of $O(1/t_m)$ and producing outputs at the same rate. With current VLSI technology, a multiply–accumulate operation on 16–bit integers can be easily done in 30 nsec, and fast RAM can provide an access time in under 30 nsec. This implies a possible throughput of about 33 million pixels/sec in current technology.

2. Low latency



(a) Test Image (b) Output from Orientational Filter I (c) Output from Orientational Filter II

Figure 1–1: Simple Demonstration of Orientational Filtering

Latency is defined as the time delay between the first input available and the output produced by this input. The requirement on system latency is to be bounded by $O(t_m NM)$. This is the best that can be done when the input stream is arriving in a scan line order, since the system requires at least M rows of data in order to perform a filter operation of size MxM.

3. Small computational complexity

Computational complexity is defined as the order of the total amount of computation required to complete the filtering operation for the *whole* image. The requirement is that the computational complexity of the system to be less than $O(F_N N^2 M^2)$. A direct implementation of the filters has a complexity of $O(F_N N^2 M^2)$. This research explores a computational scheme that is better than direct implementation. Small computational complexity implies that real–time performance can be achieved with less hardware.

4. Small storage

The storage is the amount of memory in bytes required to implement the filters. A system which requires small storage is less expensive in terms of chip count, board size, and cost than a system which requires larger storage. Although commercial RAMs have become more dense and less expensive, smaller size memory chips are faster than larger size memory chips with the same technology. Thus by keeping the storage requirements smaller, a faster system can be designed with the same implementation cost. Storage requirement can easily grow large if every filter in a system requires its own storage since the number of filters in a system can be fairly large (6 different orientations, 7 different radial frequencies and 1 quadrature pair amounts to 6 x 7 x 2 = 84 filters). Also storage requirement of an algorithm to perform *one* orientational filter tend to measure the latency of the algorithm. For example, suppose Algorithm A requires $O(N$x$N)$ bytes of memory to implement an orientational filter whereas Algorithm B requires $O(N)$ bytes of memory. It implies that Algorithm A needs to wait for $O(N$x$N)$ data before producing an output whereas Algorithm B needs to wait for only $O(N)$ data. Thus the latency of Algorithm A is at least $O(N$x$N)$ and the latency of Algorithm B is at least $O(N)$. Note that at least $O(N)$ words of storage are necessary when the inputs are coming in raster order since at least M rows of data are needed to perform MxM filtering operations.

From the above arguments, the storage requirement of the system is $O(N)$, and is not dependent on F_N.

The first objective is to design an algorithm and implementation scheme which satisfies the four implementation criteria.

1.1.2.Second Objective – Multi–Resolution Decomposition

The second objective is to develop an efficient multi–resolution decomposition algorithm of orientational filters without violating the implementation criteria. Multi–resolution decomposition (MRD) is a technique to produce a hierarchical image representation suitable for many image processing/analysis algorithms[3][13][55][76][98]. The first level of decomposition is the set of filtering operations using orientational filters. The second level of decomposition is the set of filtering operations using the orientational filters dilated by 2 in both directions.

For decimated MRD, the outputs of the filtering operation in the second level are decimated typically by 2x2. Subsequent levels of decomposition are done in a similar way with the dilation factor of the filter and the decimation factor of outputs increased by 2x2 as the decomposition level increases. The structure of the decimated MRD is shown in Figure 5–1. Thus the decomposition produces a pyramid like data representation with the first level of the decomposition being the bottom of the pyramid. Algorithms utilizing the decomposition first examine the top level of the pyramid which is small in size and coarse in resolution. Then the algorithms examine lower levels of the pyramid for more detailed analysis. This type of processing is called *coarse–to–fine* processing.

Assume a L–level decomposition is to be performed. At the k^{th} level, the size of filters is $2^{k-1}M$x$2^{k-1}M$ due to the dilation of the filters, and the size of the output images is $2^{1-k}N$x$2^{1-k}N$ due to the decimation of output images. Thus it appears that the computational complexity of the decomposition at the k^{th} level is $O(F_N N^2 M^2)$ and the computational complexity of the whole decomposition is $O(LF_N N^2 M^2)$. However, when the filters possess certain properties, the k^{th} level decomposition can be obtained from the result of the $k–1^{th}$ level decomposition. With this recursive algorithm, the computational complexity of the whole decomposition reduces to $O(F_N N^2 M^2)$.

For undecimated MRD, the outputs of the filtering operation at any level are not decimated. Thus, the size of the output is the same as the input at any level. The amount of computation increases exponentially due to the dilation of the filters. At the L^{th} level, the amount of computation is $O(4^{L-1} F_N N^2 M^2)$.

The second and third implementation criteria have to be modified for MRD since MRD involves more computation than single level orientational filtering.

2 Low latency

Define the latency for the MRD as the time delay between when the first input available and the output at the L^{th} level produced by the input. Then, the smallest latency possible is $2^{L-1} t_m NM$ since the filter size increases to $2^{L-1}M$ at the L^{th} level due to dilation. Thus, the requirement in this paper is that the latency be bounded by $O(2^L t_m NM)$.

3 Small computational complexity

The computational complexity for MRD is defined as the order of the total amount of computation required to complete the *whole* decomposition. The complexity of implementing MRD directly on the spatial domain is $O(LF_N N^2 M^2)$ for decimated MRD, and $O(F_N 4^L N^2 M^2)$ for undecimated MRD. The third criterion for MRD is that the computational complexity be less than the complexity of direct implementation.

1.1.3.Third Objective – Dynamic Tuning (Steerability)

The third objective is to investigate the feasibility of a system which can tune filters dynamically in real–time without violating the implementation criteria used for the primary objective. In this thesis, only directional tuning will be considered since radial tuning can be implemented through multi–resolution decomposition. In some literature, directional tuning is called *steerability*[31][66]. The ability to dynamically tune the filters enhances its potential in computer vision research.

Some vision applications require capability of tuning the direction of filters from frame to frame, or even pixel to pixel. It requires too much time for real–time applications if the host must reload a new set of coefficients in each filter every time the direction tuning has to be modified. One approach to achieve steerability without violating the implementation criteria is to compute the output of a filter at an arbitrary direction through a weighted linear sum of a set of basis filter outputs. The basis filters are independent of the orientation of the filter, and only the weights are orientation dependent. Thus, the orientation tuning can be done by modifying the weights rather than the filter coefficients. Equation (1.1) shows how the orientation tuning can be achieved.

$$h^{\theta}(x,y) = \sum_{i=1}^{Q} q_i(\theta)\, G_i(x,y) \quad , \tag{1.1}$$

where Q is the approximation order, $G_i(x,y)$ is a basis filter and $q_i(\theta)$ is a weight.

An extension to the steerable system is a steerable MRD system. The modified implementation criteria used in the second objective should be used here.

1.1.4.Fourth Objective – VLSI Implementation

The fourth and final objective is to design a system satisfying the implementation criteria, MRD capability, and steerability using VLSI. The design implements the algorithm obtained through the first, second, and third objectives of this research.

Another issue considered in the design is scalability of the system. Four types of scalability are considered,

1. Input image scalability

The system can accommodate any input image size by only changing the size of the input buffer which is implemented with discrete memory chips.

2. Filter number scalability

The design is modular so that the number of filters can be increased by adding extra filter components to the system. The amount of hardware increases only linearly as the number of filters increases.

3. Filter size scalability

The size of filters can be increased by adding extra filter components to the system. No new design is necessary for a different filter size. The amount of hardware increases only linearly as the filter size increases.

4. Approximation order scalability

The order of approximation can be increased by adding extra filter components to the system. No new design is necessary for a different approximation order. The amount of hardware increases linearly as the order of approximation increases.

The numbers of VLSI chips needed to construct various filter systems will be evaluated based on VLSI design techniques.

1.2.Organization of the thesis

The rest of the thesis is organized in the following way. Chapter 2 provides background information on various orientational filters and implementation schemes. These filtering operations are compared based on the implementation criteria. The chapter also reviews how orientational filters are applied to image analysis. Chapter 3 develops the separable approximation algorithms. They are *Singular Value Decomposition* (SVD), *Orthogonal Sequence Decomposition* (OSD) and *Singular Value/ Orthogonal Sequence Decomposition* (SV/OSD). They are compared in terms of approximation performance and implementation cost. In Chapter 4, an extensive performance evaluation of SV/OSD is presented to substantiate its ability to meet the implementation criteria, and shows how the approximation can be incorporated into low level processing for computer vision. In Chapter 5, a scheme to achieve the second objective of this research, an efficient MRD scheme, is presented. Chapter 6 shows an extensive performance evaluation of the MRD scheme. Chapter 7 introduces an algorithm to achieve the third objective

of this research, a real–time steerable system. Chapter 8 shows performance evaluation results of the steerable filter scheme. Chapter 9 studies detailed VLSI architecture and VLSI design for the system based on the results of Chapter 3, 5, and 7. Finally Chapter 10 summarizes the research with conclusions and future research.

CHAPTER 2

Filter Investigation

The purpose of this chapter is to provide background material relevant to the development of algorithms and implementation schemes in later chapters. This chapter gives a detailed review of various orientational filters and implementation schemes followed by a review of applications of orientational filters in computer vision problems.

2.1.Orientational Filters

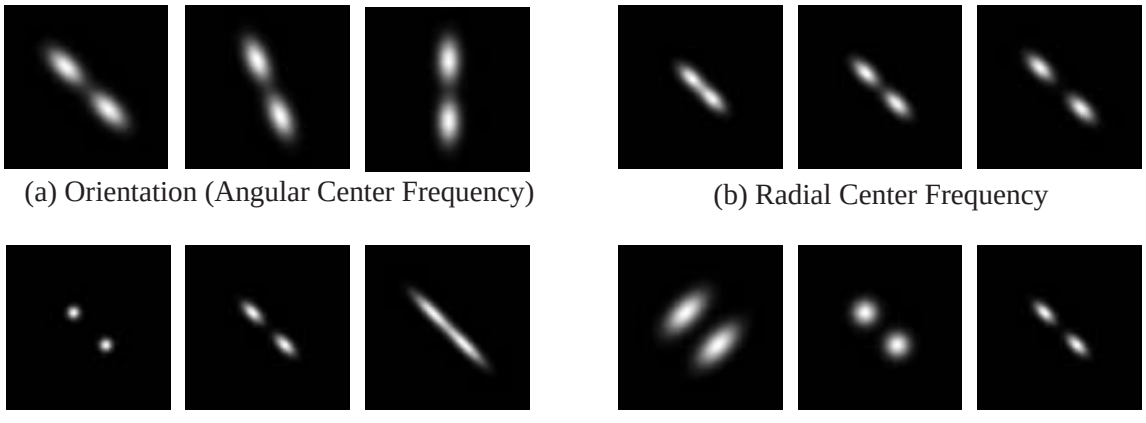
Orientational filters (directional filters) are a class of filters which have a narrow angular bandwidth in the frequency domain and are used for spatial–frequency analysis. Their applicability has been demonstrated in texture analysis[9][88][102][44], texture segmentation[41][52][68], edge detection[17][53], contour following[98][78], shape analysis[42], stereo analysis[100], image coding[22][4][45], video coding[96], image restoration[97], image enhancement[64] and motion detection[23][8]. Their advantages in computer vision problems are; 1) the ability to be tuned to a certain radial frequency and orientation, and perform as a spatially localized frequency analyzer, 2) the ability to extract many image features easily, and 3) the resemblance to certain functions in the human visual system[51][39].

In computer vision problems, objects of interest are often any size and orientation relative to the image frame, and their size and orientation can change from frame to frame due to the movement of the camera or the movement of the objects. For this reason orientational filters have to be tuned to a particular angular/radial frequency and a spatial/frequency resolution so that the analysis can be performed on multiple objects, and follow the dynamic motion of the objects.

Filters in two dimensions have several methods of tuning as shown in Figure 2–1. The representations in Figure 2–1 are all in the frequency domain. Which set of filters performs best depends on the application and objects of interest. For a computer vision system to be used in various applications, the orientational filters need to be tunable in the four aspects shown in Figure 2–1 so that they can be matched to the applications and situations.

A shift between images is caused by either a small time difference or a different viewing point. Applications such as motion analysis and stereo analysis need to find a match between certain features in multiple images and compute the distance between the matched features. However if the transform does not preserve translation, the distance cannot be computed properly. Since motion analysis and stereo analysis form a core in image understanding problems, preserving image translation is essential. Denote $\mathcal{F}$ as an orientational filter operation and *I[m,n]* as an input image. Then $\mathcal{F}$ is shift invariant if

$$\mathcal{F}\, I[m,n] \;=\; J[m,n]\;, \tag{2.1}$$



(a) Orientation (Angular Center Frequency)

(b) Radial Center Frequency

(c) Radial Frequency Bandwidth

(d) Angular Frequency Bandwidth

Figure 2–1: Examples of Filter Tuning

implies

$$\mathcal{F}\, I[m + \Delta_1, n + \Delta_2] = J[m + \Delta_1, n + \Delta_2] \quad (\forall\ \Delta_1, \Delta_2 \in \textit{integer}). \tag{2.2}$$

The next two lemmas describe shift invariance of two operators; convolution and decimation. They are used to investigate shift invariance of orientational filters in this section.

Lemma 1: A convolution operation is shift invariant.

Proof: Denote a filter kernel as *h[m,n]*, the image the filter operates on as *f[m,n]* and the output of the convolution as *y[m,n]*. Denote their Fourier transforms as $H(\omega_1, \omega_2)$, $F(\omega_1, \omega_2)$ and $Y(\omega_1, \omega_2)$ respectively. Then the convolution of a shifted image is

$$f[m + \Delta_1, n + \Delta_2] * h[m, n] \rightarrow F(\omega_1, \omega_2)H(\omega_1, \omega_2)e^{j\Delta_1\omega_1 + j\Delta_2\omega_2} = Y(\omega_1, \omega_2)e^{j\Delta_1\omega_1 + j\Delta_2\omega_2} \rightarrow y[m + \Delta_1, n + \Delta_2]. \quad (2.3)$$

Thus a convolution operation is shift invariant.

Lemma 2: A decimation operation is not shift invariant.

Proof: Denote the image to be decimated as *f[m,n]* and the output of the decimation as *y[m,n]*. After decimation by factors of μ_1 and μ_2 along each dimension, the image becomes

$$y[m, n] = f[\mu_1 m, \mu_2 n] \quad . \quad (2.4)$$

For a shifted input $\tilde{f}[m, n] = f[m + \Delta_1, n + \Delta_2]$, the corresponding output $\tilde{y}[m, n]$ is

$$\tilde{y}[m, n] = f[\mu_1 m + \Delta_1, \mu_2 n + \Delta_2] \neq y[m + \Delta_1, n + \Delta_2] \quad . \quad (2.5)$$

Thus a decimation operator is not shift invariant.

The following sub–sections describe several orientational filters and evaluate their applicability to computer vision problems based on their frequency response tunability and shift invariance.

2.1.1.Gabor Functions

Gabor functions $g_b(x,y)$ are exponentially modulated Gaussian functions and defined as

$$g_b(x, y) = g_a(\tilde{x}, \tilde{y})\, e^{j\alpha \tilde{x}} \quad , \quad (2.6)$$

where $g_a(x, y) = e^{-x^2/\sigma_x^2 - y^2/\sigma_y^2}$ is a 2D Gaussian filter, $\tilde{x} = Ux + Vy$, $\tilde{y} = -Vx + Uy$, $U = \cos\theta$, $V = \sin\theta$, θ is the angle of orientation, α determines the oscillation frequency, and σ_x and σ_y are standard deviations of $g_a(x,y)$. This class of filters is the most popular of all orientational filters used in computer vision research[9][22][41][52][68][88]. The frequency response of this class of filters is

$$G_b(u, v) = e^{-\left(\sigma_x^2 (u - \alpha U)^2 + \sigma_y^2 (v - \alpha V)^2\right)/4} \quad . \quad (2.7)$$

Thus, $G_b(u,v)$ is a bandpass Gaussian function centered at $(\alpha U, \alpha V)$ with a minor axis oriented at an angle $\theta = tan^{-1}(V/U)$ from the u axis and aspect ratio σ_x/σ_y.

It is very important in computer vision applications to know precise space (time) information and frequency information of input images at the same point in time. The uncertainty principle states that no function can be both time–limited and band–limited at the same time, and as the resolution in either domain increases, the resolution in the other domain decreases. Figure 2–2 illustrates this principle. The spatial resolution of the analysis can be increased by using smaller filters. However, a smaller filter stretches over a larger region in the frequency domain, thus has less resolution in the frequency domain. On the other hand, the frequency resolution of the analysis can be increased by using a larger filter at the expense of losing resolution in the spatial domain.

Define an effective duration of a filter function *f(x)* as the square root of the second moment of its energy distribution;

$$\Delta x = \sqrt{\frac{\int_{-\infty}^{\infty} f(x)f^*(x)x^2 dx}{E}} \quad (2.8)$$

where $f^*(x)$ is the conjugate of *f(x)*, and *E* is the total energy of *f(x)* defined as

$$E = \int_{-\infty}^{\infty} f(x)f^*(x)dx \, . \quad (2.9)$$

Similarly, define an effective bandwidth of *f(x)* in terms of its Fourier transform *F(ω)*;

$$\Delta\omega = \sqrt{\frac{\int_{-\infty}^{\infty} F(\omega)F^*(\omega)\omega^2 d\omega}{E}}. \tag{2.10}$$

Then the uncertainty principle specifies a lower bound on the possible values of their product;

$$\Delta x\,\Delta\omega \geq \frac{1}{4\pi}. \tag{2.11}$$

Gabor found the general class of functions which achieves the lower bound of (2.11). Namely,

$$g_b(x) = e^{-x^2/\sigma^2}e^{i\alpha t}. \tag{2.12}$$

In [24], the uncertainty principle is extended to 2D signals. The energy distribution of a 2D function *f(x,y)* has an effective width Δx and effective height Δy. They are defined by the square root of the second moment around the x and y axes, respectively;

$$\Delta x = \sqrt{\frac{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)f(x,y)^* x^2 dxdy}{E}}, \tag{2.13}$$

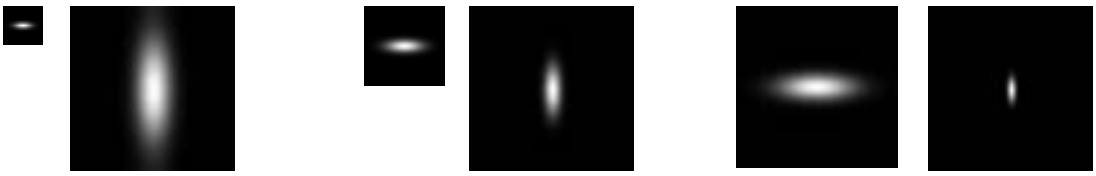
$$\Delta y = \sqrt{\frac{\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)f(x,y)^* y^2 dxdy}{E}} \tag{2.14}$$

where *E* is the total energy of *f(x,y)*. In the frequency domain, an effective bandwidth of *F(u,v)* along both the *u* and v axes are defined similarly and denoted as Δu and Δv. Then the 2D uncertainty principle is written as

$$\Delta x\,\Delta y\,\Delta u\,\Delta v \geq \frac{1}{16\pi^2}. \tag{2.15}$$

In [24], it is shown that the 2D Gabor function achieves the lower bound of this 2D uncertainty principle. This makes the 2D Gabor function very attractive in many computer vision applications. Also neuro–physiological studies indicate that the 2D Gabor function resembles the response of simple cells in the human visual cortex[56][2]. This similarity of processing to the human vision system is another reason for the 2D Gabor function's popularity. The mechanism of the human visual system based on the pioneer work of Hubel and Wiesel[39] is discussed in Appendix A.

Gabor filters have four parameters, $\theta, \sigma_x, \sigma_y$ and α. The frequency response of the filters can be tuned easily by changing these parameters. The orientation of the filters can be modified by changing θ. The radial center frequency can be increased



(a) Filter I (15x15) (b) Filter II (31x31) (c) Filter III (63x63)

Gaussian filters of different sizes are shown in both the spatial domain (left) and the frequency domain (right).

Figure 2–2: Spatial/Frequency Resolution of Orientational Filters

by increasing the value of α. The radial frequency bandwidth can be increased by decreasing σ_x while holding the value of σ_y. The angular frequency bandwidth can be increased by increasing σ_y while holding the value of σ_x.

Since the Gabor filters involve only a convolution, they are shift invariant (Lemma 1).

2.1.2.Gaussian Derivatives

This class of filters is constructed by taking k[th] derivatives of the Gaussian, $g_a(x,y)$, along a direction $\boldsymbol{n}$,

$$d^k g_a(x,y) \;=\; \frac{d^k\, g_a(x,y)}{d\boldsymbol{n}^k} \quad . \tag{2.16}$$

The first derivative and second derivative are used most frequently in image analysis applications. The appearance of this class of filters dates back to Marr's theory of vision[57].

Variations of Gaussian derivatives are Difference of Gaussian (DOG) and Difference of Offset Gaussian (DOOG)[101]. DOG is defined as

$$G_{dog}(x,y) \;=\; \sum_i c_i\, g_a\,(x,y;\sigma_{\xi},\sigma_{yi}) \tag{2.17}$$

and DOOG is defined as

$$G_{doog}(x,y) \;=\; \sum_i c_i\, g_a\,(x+\delta x_i,\; y+\delta y_i : \sigma_{\xi},\sigma_{yi}) \quad . \tag{2.18}$$

One of the important DOG filters is Wilson's DOG[99] implemented in the Georgia Tech Vision (GTV) model[25] and defined as

$$f(x,y) \;=\; \frac{A}{2\pi} e^{-y^2/4\pi s_y}\left\{ e^{-x^2/4\pi s_1} - Be^{-x^2/4\pi s_2} + Ce^{-x^2/4\pi s_3}\right\} \quad . \tag{2.19}$$

An advantage of using these variations instead of derivatives of the Gaussian is that various filters can be obtained with relatively few convolution operations. Laplacian of Gaussian can be approximated by a subtraction of two concentric circular Gaussian filters with different variances (σ_x, σ_y),

$$g_a(x,y;\sigma_1) - g_a(x,y;\sigma_2) \qquad \sigma_2/\sigma_1 \approx 1.5\ . \tag{2.20}$$

The second derivative of the directional Gaussian along the x axis can be approximated by a weighted addition of three directional Gaussian filters with small offsets,

$$-\, g_a(x,y+\delta y;\sigma_x,\sigma_y) + 2g_a(x,y;\sigma_x,\sigma_y) - g_a(x,y-\delta y;\sigma_x,\sigma_y) \qquad \delta y \approx \sigma_y\ . \tag{2.21}$$

A similar approximation is used along the y axis,

$$-\, g_a(x+\delta x,y;\sigma_x,\sigma_{y)} + 2g_a(x,y;\sigma_x,\sigma_y) - g_a(x-\delta x,y;\sigma_x,\sigma_y) \qquad \delta x \approx \sigma_x\ . \tag{2.22}$$

Gaussian derivatives of the form (2.17) have three parameters, θ, σ_x and σ_y. The frequency response of the filters can be tuned by changing these parameters. The orientation of the filters can be changed by changing θ. The radial frequency bandwidth can be increased by decreasing σ_x while holding the value of σ_y. The angular frequency bandwidth can be increased by increasing σ_y while holding the value of σ_x. However, there is no parameter to change the radial center frequency.

Since the filter involves only a convolution, it is shift invariant (Lemma 1).

2.1.3.Prolate Spheroidal Functions

The uncertainty principle in the form of (2.11) based on the definition of the effective duration (2.8) and the effective bandwidth (2.10), gives a somewhat unclear picture of what is happening. An alternative is to consider the proportion of energy e_1 inside a finite time duration *[–T/2, T/2]* and another energy proportion, e_2, inside a finite bandwidth $[-\omega_c, \omega_c]$, i.e.

$$\frac{\displaystyle\int_{-T/2}^{T/2} f(t)f^*(t)dt}{E} \;=\; e_1^2 \quad \text{and} \tag{2.23}$$

$$\frac{\displaystyle\int_{-\omega_c}^{\omega_c} F(\omega)F^*(\omega)d\omega}{E} \;=\; e_2^2\ , \tag{2.24}$$

where E represents the total energy of $f(t)$ in the interval $[-\infty, \infty]$. One of the questions concerning the uncertainty principle is that which time limited function $f(t)$ has the most energy concentrated inside $[-\omega_c, \omega_c]$.

It is well known that the answer to the above question is the eigenfunction $T_{p0}(t; \omega_c)$ corresponding to the largest eigenvalue λ_0 of the equation[83],

$$B_{-\omega_c,\omega_c} D_{-T/2,T/2} T_p(t) = \lambda T_p(t) \tag{2.25}$$

where D is a time–limiting operator defined by

$$D_{-T/2,T/2} = \begin{cases} 1 & (-T/2 \le t \le T/2) \\ 0 & otherwise \end{cases}, \tag{2.26}$$

and B is a band–limiting operator defined by

$$B_{-\omega_c,\omega_c} f(t) = F^{-1} D_{-\omega_c,\omega_c} F f(t) \tag{2.27}$$

with F being the Fourier transform operator. In the time domain, (2.25) can be rewritten as

$$\frac{1}{2\pi} \int_{-\omega_c}^{\omega_c} d\omega\, e^{i\omega t} \int_{-T/2}^{T/2} d\tau\, T_P(\tau)\, e^{-i\omega\tau} = \int_{-T/2}^{T/2} \frac{\sin \omega_c(t-\tau)}{\pi(t-\tau)}\, T_p(\tau)\, d\tau = \lambda\, T_p(t)\,. \tag{2.28}$$

Eigenfunctions of the operator BD are called *prolate spheroidal wave functions* (*PSWFs*) and have the following properties[50][83].

1. $\{T_{pi}(t; T,\omega_c)\}$ are bandlimited in $[-\omega_c, \omega_c]$ and span all band–limited function in $[-\omega_c, \omega_c]$.
2. In the interval $[-\infty, \infty]$, $\{T_{pi}(t; T,\omega_c)\}$ are orthonormal to each other,

$$\int_{-\infty}^{\infty} T_{pi}(t)\, T_{pj}(t) dt = \delta_{i,j}\,. \tag{2.29}$$

3. In the interval $[-T/2, T/2]$, $\{T_{pi}(t; T,\omega_c)\}$ are orthogonal to each other,

$$\int_{-T/2}^{T/2} T_{pi}(t)\, T_{pj}(t) dt = \lambda_i\, \delta_{i,j}\,. \tag{2.30}$$

In the discrete domain, the equivalent of a PSWF is called a *prolate spheroidal sequence* (*PSS*) and is defined as the solution to the linear equation[85].

$$\sum_{m=0}^{N-1} \frac{\sin(n-m)\pi\epsilon_P}{(n-m)\pi}\, T_{Pi}[m] = \lambda_i\, T_{pi}[n]\,,\ n=0, 1, 2, \ldots N-1 \tag{2.31}$$

where N is the sample size and $\epsilon_P = \omega_c/\pi$. Prolate Spheroidal Sequences, $T_{Pi}[n; \omega_c]$, possess the following properties[85].

1. They satisfy a discrete orthonormality condition assuming the PSSs are normalized ($\sum_{n=0}^{N-1} T_{Pi}^2 = 1$).

$$\sum_{n=0}^{N-1} T_{Pi}[n] T_{Pj}[n] = \delta_{i,j}\quad. \tag{2.32}$$

2. For even i, $T_{Pi}[n]$ is even symmetric and for odd i, $T_{Pi}[n]$ is odd symmetric.

$$T_{Pi}[n] = (-1)^k T_{Pi}[N-1-n]\quad. \tag{2.33}$$

3. Eigenvalues λ_i represent the energy ratio

$$\lambda_i = \sum_{k=0}^{M-1} |\hat{T}_{Pi}[k]|^2 / \sum_{k=0}^{N-1} |\hat{T}_{Pi}[k]|^2\,, \tag{2.34}$$

where $\hat{T}_{Pi}[k]$ is the DFT of $T_{Pi}[k]$, and $M = \epsilon_P N$.

To save computation time, the *NxN* matrix in (2.31) can be tridiagonalized and PSS can be derived from the linear equation,

$$\boldsymbol{M}_{PS} T_{Pi} = \lambda_i T_{Pi} \ , \tag{2.35}$$

where $\boldsymbol{M_{PS}}$ is the tridiagonal matrix given by

$$\boldsymbol{M_{PS}}[n,m] = \begin{cases} \frac{1}{2}m(N-m) & n = m-1 \\ (\frac{N-1}{2} - m)^2 \cos 2\pi\omega_c & n = m \\ \frac{1}{2}(m+1)(N-1-m) & n = m+1 \\ 0 & |n-m| > 1 \end{cases} \ . \tag{2.36}$$

There are many ways to extend the theory of PSWF to two dimensions[84]. In image processing applications, two useful forms are Cartesian separable and Polar separable forms.

For the Cartesian separable case, a 2D PSWF is a Cartesian product of two 1D PSWFs. Thus,

$$T_p(x,y;T_x,T_y,\omega_x,\omega_y) = T_p(x;T_x,\omega_x)T_p(y;T_y,\omega_y) \ . \tag{2.37}$$

For the Polar separable case, the band–limiting and time–limiting operators limit signals within a circle *S* in the spatial domain and a circle *R* in the frequency domain, respectively, with the centers located at the origin. Thus,

$$D_s f(x,y) = \begin{cases} f(x,y) & (x^2+y^2 \le s^2) \\ 0 & otherwise \end{cases} \text{, and} \tag{2.38}$$

$$B_r f(x,y) = F^{-1} D_r F f(x,y) \ . \tag{2.39}$$

For this case, the PSWFs are eigenfunctions of the operator $B_r D_s$.

$$B_r D_s f(x,y) = \frac{1}{4\pi^2} \int_R d\omega_x d\omega_y e^{j\omega_x x + j\omega_y y} \int_S d\eta d\zeta \, f(\eta,\zeta) \, e^{-j\omega_x \eta - j\omega_y \zeta}$$

$$= \frac{r}{2\pi} \int_S \frac{J_1(r\sqrt{(x-\eta)^2+(y-\zeta)^2})}{\sqrt{(x-\eta)^2+(y-\zeta)^2}} f(\eta,\zeta) \, d\eta d\zeta \ , \tag{2.40}$$

where $J_1(\,)$ is the Bessel functions of the first kind of order 1[26].

The frequency response of the filters can be tuned by re–defining the bandlimited operator *B*. This is done by defining *B* such that it passes only a region where the filters frequency response is located. This method of tuning filters, however, involves finding eigenfunctions of (2.25) which is often computationally intensive.

Since the filter involves only a convolution, it is shift invariant (Lemma 1).

2.1.4.Multirate Filter Banks and Wavelets

Croisier, Esteban and Galand first observed the fact that a signal could be split into multiple channels using non–ideal filters, sub–sampled and reconstructed without aliasing[16]. This coding scheme is called *multirate filter banks.* The structure using *P* bands of multirate filter banks is shown in Figure 2–3. $H_k(\omega)$ is an analysis (decomposition) filter and $G_k(\omega)$ is a synthesis (reconstruction) filter. Later Smith and Barnwell demonstrated that tree–structured subband coders based on 2–channel multirate filter banks can reconstruct, not only alias free, but the exact replica of the input (named a *perfect reconstruction filter bank*)[87]. P. P. Vaidyanathan extended the perfect reconstruction filter bank design to the case of *M* channels, with arbitrary *M*[94], and J. Kovacevic extended it to arbitrary rational sampling rates[45].

The effectiveness of multirate filter banks was first demonstrated for speech coding. Later the scheme was extended to multidimensional signals for image and video coding. In multidimensional cases, sampling (down sampling and up–sampling) plays an important role. A straightforward and popular way is rectangular sampling. Also hexagonal sampling is often used in coding because 1) the spectra of the signal and its repeated occurrences do not overlap, and 2) the human eye is less sensitive to resolution along the diagonal, hence it is more reasonable for the low pass filter to have a diagonal cutoff. Finer separability can be accomplished by using more general non–separable sampling.

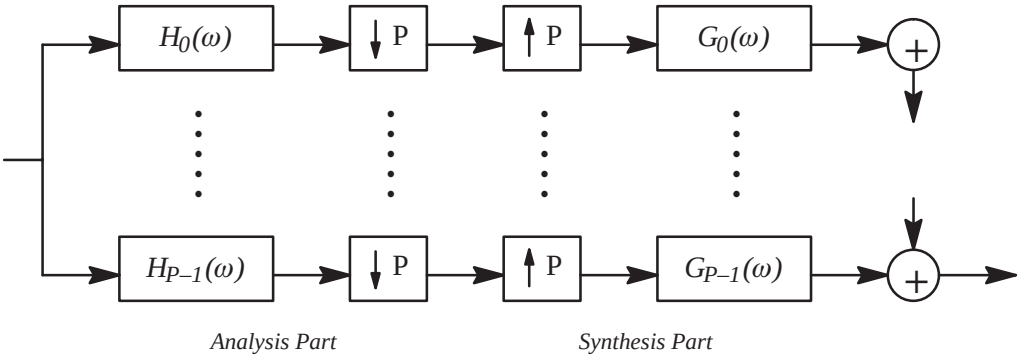


Figure 2–3:Structure of a Multirate Filter Bank

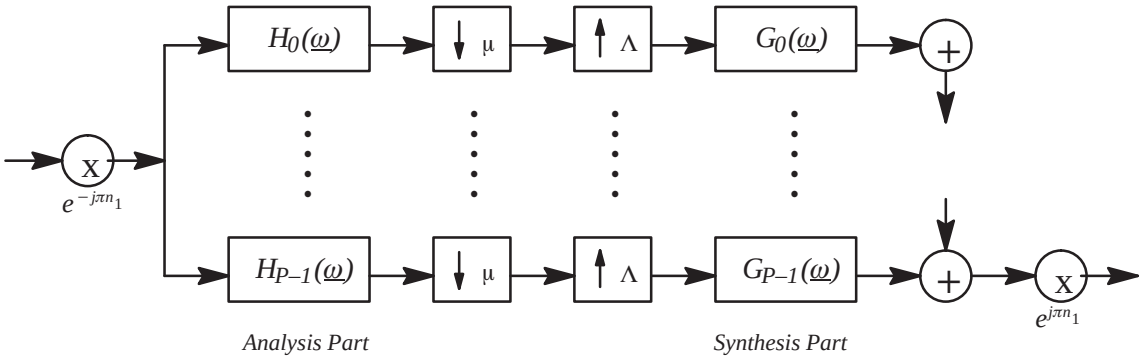
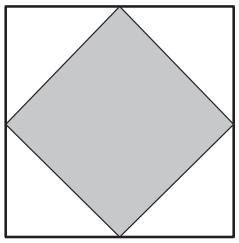


Figure 2–4: First or Second Stage of a Directional Filter Bank

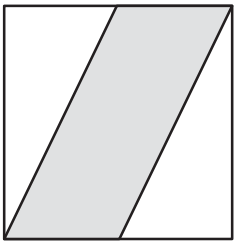
R. H. Bamberger [4] and J. Kovacevic [45] developed directional filter banks independently based on non–separable sampling. In [4], 2D signals are decomposed into $P=2^n$ directional banks using modulation, resampling, and two types of filters: a diamond shaped filter and a parallelogram shaped filter. The structure of the decomposition/reconstruction system is shown in Figure 2–4 and an 8–band directional filter bank is described here as an example. The frequency response of the diamond shaped filter and the parallelogram shaped filter are shown in Figure 2–5(a) and (b). The frequency partition pattern for an 8–band directional decomposition is shown in Figure 2–5 (c). FIrst an image is modulated by π in either ω_x or ω_y in the frequency domain. The result of modulation in the ω_x direction is shown in Figure 2–5(d). The frequency components 1–4 and 5–8 are separated by the diamond shaped filter. The output of the filters are downsampled by μ_1, and the results after the down sampling are shown in Figure 2–5(e). Another modulation is applied after the down sampling and the diamond shaped filters are applied to separate the frequency components 1–2, 3–4, 5–6 and 7–8. The result of the 4–band partition after the down sampling is shown in Figure 2–5(f). As can be seen in Figure 2–5(f), each frequency component can be extracted by applying the parallelogram shaped filter or its rotated/reflected version. Each output is downsampled by μ_2 in order to maintain the sampling size. Other 2^n decomposition systems can be constructed in a similar way. In [4] it is shown that the decomposition/reconstruction system can be either alias free or provide an exact reconstruction. The 2D filters can be implemented efficiently using 1D polyphase filters.

Advantages of using filter banks are computational efficiency and simple implementation. The primary application is image compression coding. Much research has been focused on the reconstruction part of the system. In image analysis ap-



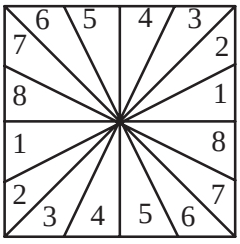
$$\mu_1 = \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$$

(a) Diamond Shaped Filter and the Associated Sampling Matrix

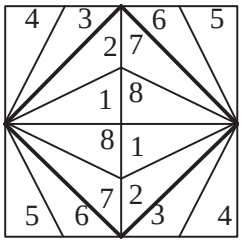


$$\mu_2 = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}$$

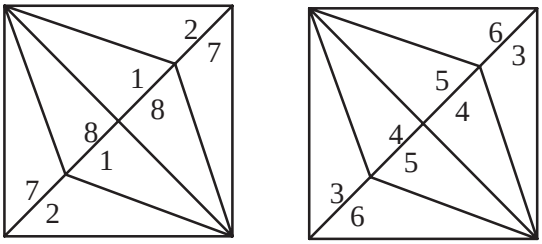
(b) Parallelogram Shaped Filter and the Associated Sampling Matrix



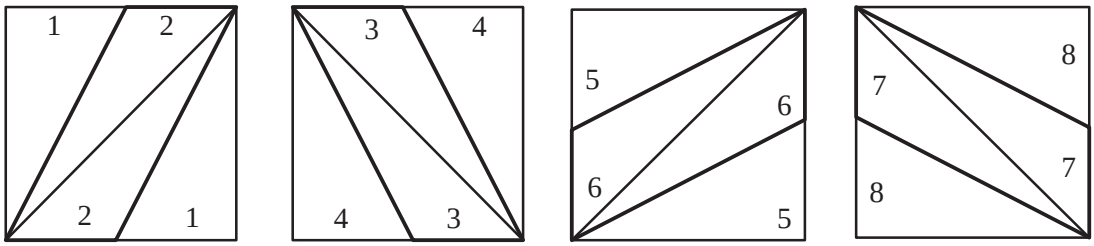
(c) Frequency Partition



(d) First Modulation



(e) 2–Band Partition (After Being Filtered by a Diamond Shaped Filter and Downsampled by μ_I)



(f) 4–Band Partition

Figure 2–5: 8–Band Directional Decomposition by Bamburger [4]

plications, reconstruction of images is usually unnecessary, and the aliasing introduced after decimation causes the transformation to be shift variant. Moreover, it is difficult to tune the orientation and bandwidth of the filter dynamically by using a multirate filter bank structure.

Independent of the development of filter banks, the theory of wavelets first appeared in the mid 80's for the purpose of non–stationary signal analysis. *Wavelets* are a set of functions obtained by dilation and translation of a single function which is often called the *mother wavelet*. The analysis is performed by taking a projection of an input signal over each wavelet (*wavelet transform)***.** The wavelet transform of a function *f(t)* is defined as

$$Wf(a,b) \quad = \quad a^{\frac{-1}{2}} \int f(t)\psi\Big(\frac{t}{a} - b\Big)dt \,, \tag{2.41}$$

where $\psi(t)$ is a mother wavelet, and *a* and *b* are dilation and translation parameters respectively. By changing the dilation parameter, one can perform the analysis at different scales, and by changing the translation parameter, one can perform the analysis at different time points. Hence the wavelet transform is called *time–scale analysis* which is related to *time–frequency analysis*. As the scale parameter increases, the wavelets expand and the frequency responses shift toward a lower frequency. As the scale parameter decreases, the wavelets contract and the frequency responses shift toward a higher frequency. The time–scale behavior of the wavelet is shown in Figure 2–6. The advantage of analyzing signals using wavelets is based on the trade–off in time/frequency resolution. The resolution in the time–frequency domain is constrained by the uncertainty principle. The wavelet transform achieves a high frequency resolution in the low frequency region at the expense of time resolution, and achieves a high time resolution in the high frequency region at the expense of frequency resolution. This trade–off has a good fit for many applications including image analysis. Most natural images consist of high frequency components with short duration such as edges and low frequency components with long duration such as texture surfaces. Image analysis applications require precise locations of the high frequency components, but do not require precise frequency information of those components. On the other hand, the applications require good frequency information on the low frequency components (texture surface), but location information is not required to be precise.

The dilation and the translation parameters can be either continuous or discrete leading to different analysis schemes similar to Fourier analysis. Both parameters can vary continuously leading to the continuous wavelet transform (CWT). They can be discrete, and the set of basis wavelets construct an orthonormal set leading to the wavelet series (WS). The wavelet itself can be discrete leading to the discrete wavelet transform (DWT). It has been observed that the DWT is a particular type of multi–rate filter bank.

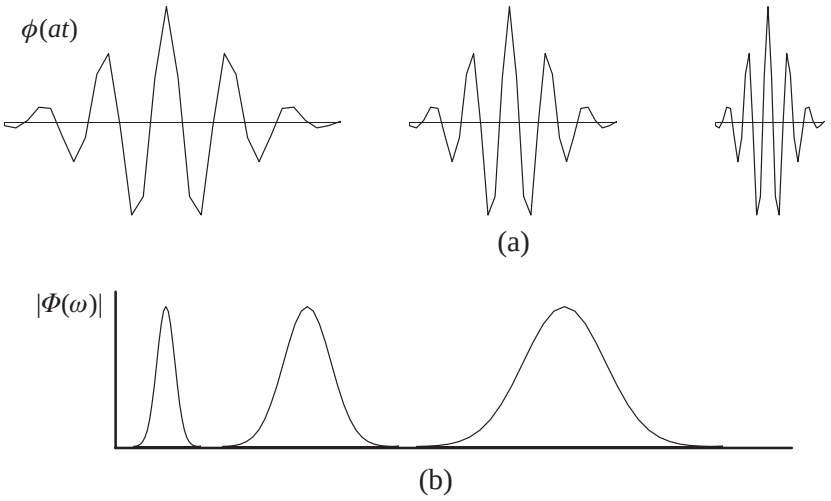


Figure 2–6: Characteristic of the Wavelet Transform Basis

(a) Wavelets scaled in time. Note that the window size varies as the scaling changes. They contract as the scaling gets larger and expand as it gets smaller. (b) The corresponding Fourier transform of (a). They contract as they move to a lower frequency and expand as they move to a higher frequency.

The wavelet transform is discussed in detail in Chapter 5 in conjunction with multi–resolution image decomposition.

2.1.5.Frequency Domain Filters

In this class of orientational filters, the filtering is done in the frequency domain. Hence it involves Fourier transforms of input images. After the filtering, the output is transformed back to the spatial domain using the inverse Fourier transform for further precessing. The advantage of this scheme is that orientation and bandwidth tuning are trivial and it is often more efficient to implement a complicated filter in the frequency domain using the FFT and the inverse FFT than by a direct implementation in the spatial domain. However, this increases latency and the amount of hardware because of the FFT and the inverse FFT.

One disadvantage of using the Fourier Transform in image analysis is that there is no joint spatial/frequency information after the Fourier Transform. This is the reason that the image has to be transformed back to the spatial domain for further analysis. This problem can be somewhat alleviated by using the Windowed Fourier Transform or Short–time Fourier Transform. It is important to choose the right window size and window functions for the application. If the window size is too small, the transform loses resolution in the frequency domain, and if it is too big, it loses the resolution in spatial domain.

The cortical transform [98] uses the global Fourier Transform. The transform is constructed from two types of filters: a mesa filter and a bisection filter. The mesa filter $\tilde{m}_0(u,v)$ is a low–pass filter in the radial frequency with unit gain within the pass–band and Gaussian fall–off beyond some corner frequency, f_c, at which the gain falls to 1/2. It is created by convolving a cylinder of radius f_c centered at the origin, with a Gaussian. Thus,

$$\tilde{m}_0(u,v) \;=\; \left(\frac{r}{f_c}\right)^2 e^{-\pi\left(\frac{r\gamma}{f_c}\right)^2} * \Pi\left(\frac{r}{2f_c}\right) \quad , \tag{2.42}$$

where u and v are two coordinates of the frequency space, $r = \sqrt{u^2 + v^2}$, $\Pi(r)$ is a unit disc with unit height centered at the origin, and the parameter γ controls the sharpness of fall–off at f_c. The larger the value of γ, the sharper the fall–off becomes. In order to construct a multi–resolution pyramid, multiple mesa filters with different cut–off frequencies are required. They can be obtained by scaling the original mesa filter by a factor of 2, 4, 8, etc. The scaled mesa filter with the scaling factor 2^k is defined as

$$\tilde{m}_k(u,v) \;=\; \tilde{m}_0(2^k u, 2^k v) \quad . \tag{2.43}$$

By subtracting $\tilde{m}_{k+1}(u,v)$ from $\tilde{m}_k(u,v)$ for each k, a disc shaped filter is obtained, which is used to decompose the frequency space into different radial frequency regions.

The bisection filter bisects the frequency space in half. The filter which bisects along the horizontal frequency axis is

$$\tilde{b}_0(v) \;=\; U(v) * \eta\, e^{-\pi\eta^2 v^2} \;=\; \int_{-\infty}^{v} \eta e^{-\pi\eta^2 r^2}\, dr \quad , \tag{2.44}$$

where $U(v)$ is the step function and the parameter η controls the sharpness of fall off along the bisection line. The bisection filter oriented at an angle θ is

$$\tilde{b}_\theta(\tilde{v}) \;=\; U(\tilde{v}) * \eta\, e^{-\pi\eta^2 \tilde{v}^2} \quad , \tag{2.45}$$

where $\tilde{v} = u\cos\theta - v\sin\theta$.

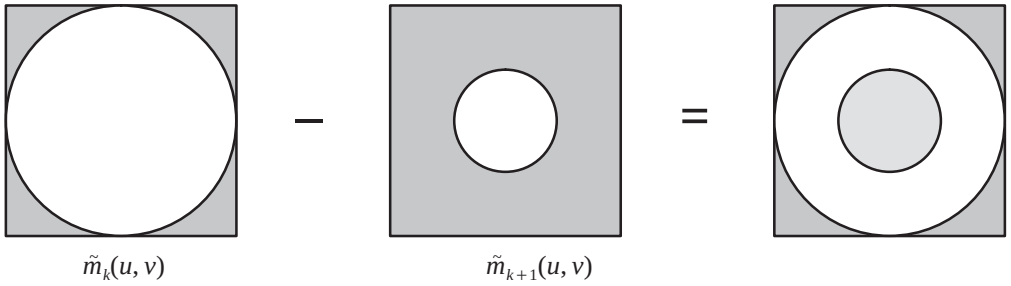
By using the mesa and bisectional filters in a repeated way, the frequency plane can be split into directional and radial components. This procedure is shown in Figure 2–7.

The operator in [34] uses the Windowed Fourier Transform. It is designed such that it is general purpose and can perform a number of useful operations in parallel. The operator uses 2D Gaussian for the window function. First it calculates the Windowed Fourier Transform in position (x_0, y_0), at the single radial frequency (r), and several different directions (θ_n) using

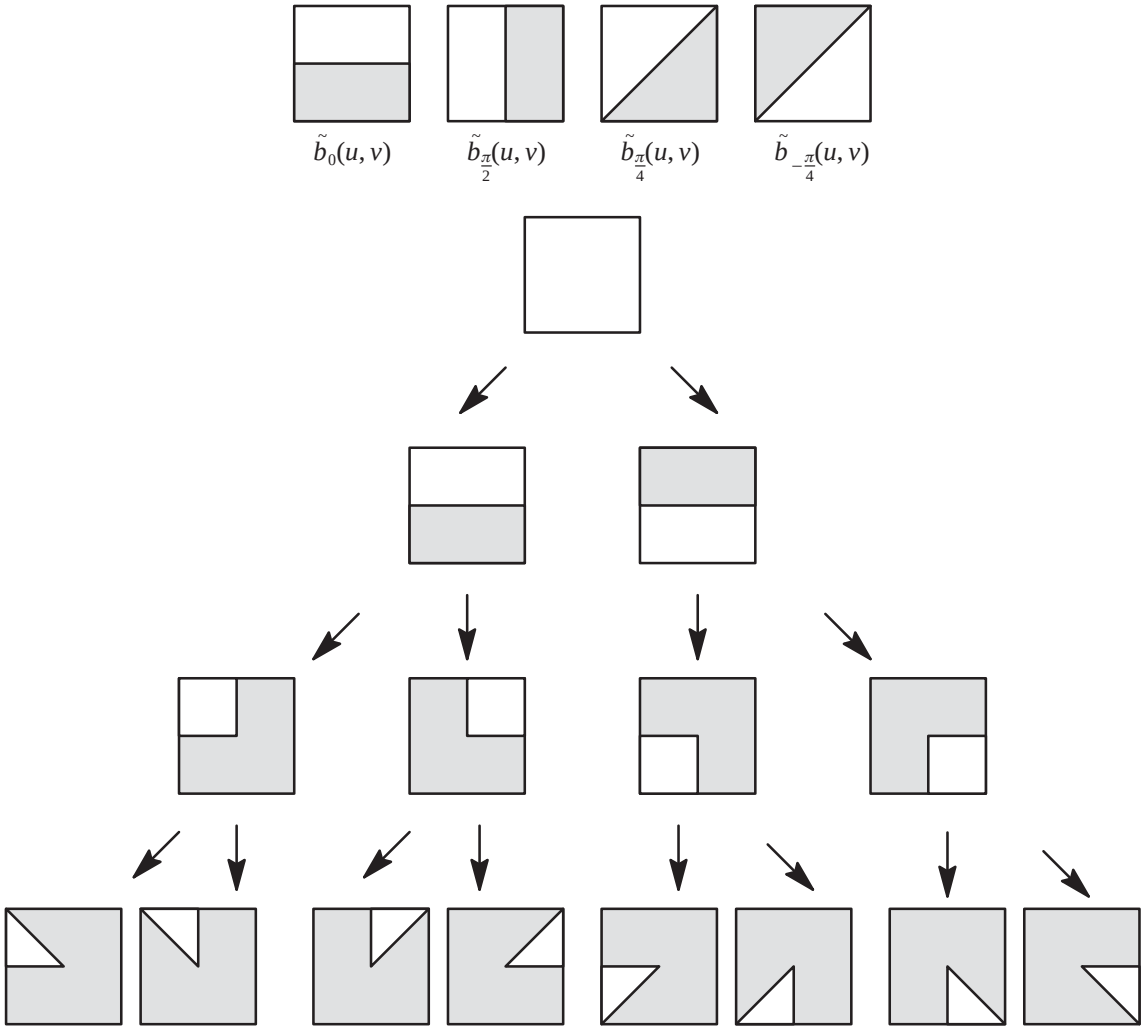
$$F(r,\theta_n) \;=\; F(u_n, v_n) \;=\; \int_{-\infty}^{\infty}\int f(x,y)\, g(x - x_0, y - y_0) e^{-j2\pi(u_n x + v_n y)} dx dy \quad , \tag{2.46}$$

where $f(x,y)$ is the input image, $u_n = r\cos\theta_n$, $v_n = r\sin\theta_n$, and $g_a(x,y)$ is the 2D Gaussian. Next, the maximum value of $|F(r,\theta_n)|$ as a function of θ_n is obtained from

$$A(r,x,y) \;=\; \max_{\theta_n} |F(r,\theta_n)|_{x_0,y_0} \quad . \tag{2.47}$$



(a) Decomposition in the Radial Direction



(b) Decomposition in the Angular Direction

Figure 2–7:Hierarchical Process of the Cortical Transformation

The operator returns a complex function $\hat{f}(x, y)$ defined by

$$\hat{f}(x, y) = A(r, x, y)e^{j2\theta_{\max}} . \quad (2.48)$$

For frequency domain filters, tuning the frequency response is easy. The filters are not shift invariant unless processing on the frequency domain is applied to only the amplitude part of the Fourier Transform and not applied to the phase.

2.1.6.Comments

Gabor filters have the optimal spatial–frequency localization in the sense of the uncertainty principle, and tuning the frequency response is quite straightforward. However, in general, Gabor filters are non–separable and computationally intensive. Thus, they have rarely been used in real–time applications.

Gaussian derivatives have good spatial–frequency localization. It is easy to tune the orientation to an arbitrary direction. However, it is difficult to tune the radial frequency of the filter. Although computational complexity is similar to that of the Gabor filters, when multiple filters are to be implemented, this complexity can be reduced by using DOG and DOOG.

Prolate spheroidal functions have an excellent spatial–frequency localization. It is difficult to update the frequency/orientation tuning, since it involves solving a large linear system of equations for eigenvectors. The filters are generally non–separable, and computationally intensive.

Orientational filters implemented using a filter bank structure can be designed to have an efficient computational structure and inexpensive implementation. However, it is difficult to tune the frequency responses and this type of filter is not shift invariant due to the decimation process introduced when decomposition of the directional components is done.

Frequency domain filters can be designed for good spatial–frequency localization, and it is easy to tune the frequency responses. The computational complexity is insensitive to the filter size and smaller than the direct computation of non–separable filters with a large mask size. However, they are expensive to implement since both the FFT and the inverse FFT are involved. They also introduce a large latency.

Table 2–1 summarizes the orientational filters introduced in this section. Table 2–1 (a) summarizes the applicability of each type of filter to computer vision problems based on their frequency response tunability and shift invariance. Table 2–1 (b) shows the implementation characteristics associated with each type of filter. Gabor filters, Gaussian derivatives and PSSs have high computational complexity since they are non–separable in general. Filter banks have a small complexity since in most cases they are separable, and can be implemented by using poly–phase structures and tree–structures. Frequency domain filters have smaller complexity than non–separable filters when the size of the filters is large. Latencies are small in all cases except the frequency domain filter since it involves a FFT of an input image and an inverse FFT of the output. The last column in Table 2–1 (b) measures how computational complexity increases as the filter size increases. If a filter size is denoted as MxM, then the complexity is a factor of M^2 for non–separable filters (Gabor, Gaussian Derivatives and PSS), M for separable filters (Filter banks) and insensitive for frequency domain filters.

Overall, Gabor filters have good performance characteristics for computer vision problems, but they are computationally expensive.

Table 2–1: Comparison of Orientational Filters

(a) Filtering Performance Characteristics

	Radial Frequency Response Tunability	Angular Frequency Response Tunability	Shift Invariance
Gabor	easy	easy	invariant
Gaussian Derivatives	difficult	easy	invariant
PSF	difficult	difficult	invariant
Filter Banks	difficult	difficult	often variant
Frequency Domain	easy	easy	often variant

(b) Filtering Implementation Characteristics

	Computational Complexity	Latency	Hardware	Effect of Filter Size on Computation
Gabor	high	small	fairly expensive	high
Gaussian Derivatives	high	small	fairly expensive	high

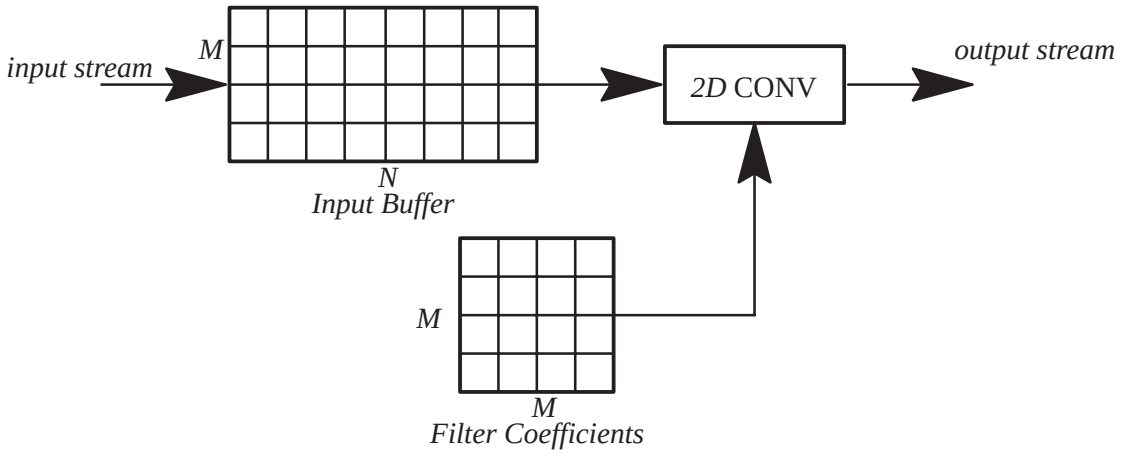


Figure 2–8: Direct Filtering

PSS	high	small	fairly expensive	high
Filter Banks	low	small	inexpensive	low
Frequency Domain	fairly high	large	expensive	none

2.2.Computational Schemes

In this section, different implementation schemes are described, and comparisons are made in terms of the implementation criteria; throughput, latency, computational complexity and storage. Certain other hardware requirements are also examined. The following assumptions are made; input images are processed in a scan line order, the image size is *NxN*, the filter size is *MxM*, the number of filters to be implemented is F_N, the frame rate is *R frames/sec*, and values of both the images and the filters are real. The input rate is (RN^2) *pixels/sec*.

2.2.1.Direct Methods

In this scheme, a 2D filter mask is directly applied to an input image. Computation of each output pixel involves computing *MxM* 2D convolutions. Figure 2–8 depicts this scheme. The number of multiplications required to process one input frame is M^2N^2, and the number of additions is $(M^2-1)N^2$. Thus the computational complexity is $O(M^2N^2)$. Using a pipeline structure the throughput is $O(1/t_m)$. The latency is $O(NM)$ since a convolution requires only *M* rows of pixel values, and the amount of storage required is $O(NM)$.

For a system with multiple filters, the input buffer can be shared among the filters. Hence the storage requirement is independent of F_N and is $O(NM)$.

2.2.2.FFT Methods

In this scheme, both an input image and a filter are transformed into the frequency domain (The frequency response of the filter is pre–computed.), the input image and the filter are multiplied in the frequency domain, and the inverse FFT is applied to the result of multiplication. The size of the FFT has to be at least $(N+M-1)x(N+M-1)=\tilde{N}^2$, where $\tilde{N} = N + M - 1$, in order not to introduce any aliasing. Assume a 2x2 vector radix FFT is used. Then a FFT of size $\tilde{N}x\tilde{N}$ requires $\frac{3}{2}\tilde{N}^2 \log \tilde{N}$ complex multiplications and $4\tilde{N}^2 \log \tilde{N}$ complex additions[26]. Since values of both the input image and the filter are assumed to be real, only a half plane of the Fourier Transform needs to be computed, reducing the complexity by half. One complex multiplication is equal to 4 real multiplications and 2 real additions, and one complex addition is equal to 2 real additions. The transformation based on the FFT requires a forward FFT, an inverse FFT, and a multiplication of an input image and the filter in the frequency domain. Considering these operations, the total number of multiplications for this method is $\tilde{N}^2(6 \log \tilde{N} + 2)$ and the total number of additions is $\tilde{N}^2(11 \log \tilde{N} + 1)$. The FFT method is insensitive to the mask size of the filter. Therefore as the filter becomes larger, the FFT method has an edge over the direct method. The computational complexity is $O(\tilde{N}^2 \log \tilde{N})$. The latency is $O(\tilde{N}^2 \log \tilde{N})$ using serial processing, and the storage requirement is $O(\tilde{N}^2)$.

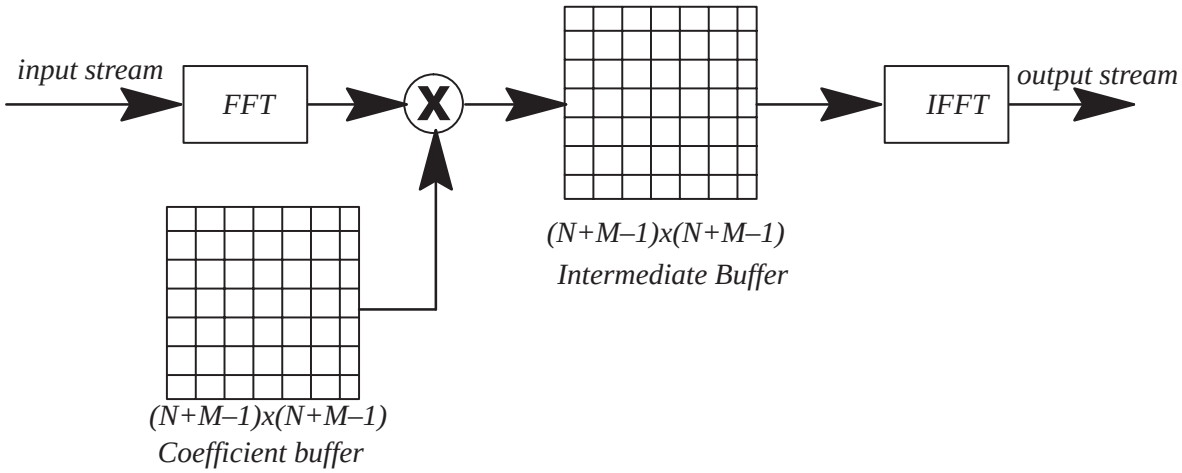


Figure 2–9: FFT Method

When F_N filters are to be implemented, the amount of storage increases by F_N. Thus the storage requirement increases to $O(F_N\tilde{N}^2)$.

To find ways of implementing a real–time transformation system using the FFT, it is necessary to examine real–time FFT systems. Active research has been speeding up FFT operations using parallel processing and special purpose hardware[70][79][6]. In [70], a parallel 2D FFT architecture is presented. The system computes a *NxN* 2D FFT in $O(\sqrt{N}\,N\log\sqrt{N})$ which is a speed–up of $O(\sqrt{N})$ over the conventional row–column approach. It performs the transformation by decomposing *NxN* kernel into two smaller N_sxN_s kernels where $N_s = \sqrt{N}$. Define the 2D discrete Fourier Transform (*DFT*) of *x[n1, n2]* $(0 \le n_1, n_2 < N)$ as

$$X[k_1,k_2] \;=\; \sum_{n_1}^{N_s-1}\sum_{n_2}^{N_s-1} x[n_1,n_2]\; W_N^{\,k_1n_1+k_2n_2} \quad , \tag{2.49}$$

where $0 \le k_1, k_2 < N$ and $W_N = e^{-i2\pi/N}$. The regular row–column FFTs take a total of *2N* 1D FFTs, *N* times for the row direction and *N* times for the column direction. The 2D DFT can also be performed by decomposing the *NxN* kernel into two smaller N_sxN_s kernels. Then the formula of the 2D DFT becomes

$$X(r_1,s_1;r_2,s_2) \;=\; \sum_{m_1=0}^{N_s-1}\sum_{m_2=0}^{N_s-1}\sum_{l_1=0}^{N_s-1}\sum_{l_2=0}^{N_s-1} x(l_1,m_1;l_2,m_2)W_N^{(N_sl_1+m_1)(N_sr_1+s_1)+(N_sl_2+m_2)(N_sr_2+s_2)} \tag{2.50}$$

with the substitution of variables, $n_i = N_sl_i + m_i,\ k_i = N_sr_i + s_i,\ i = 1,2$ and $0 \le l_i, m_i, r_i, s_i < N_s$. The equation (2.50) can be rewritten as

$$X(r_i,s_1;r_2,s_2) \;=\; \sum_{m_{1=0}}^{N_s-1}\sum_{m_2=0}^{N_s-1} W_N^{(m_1s_1+m_2s_2)}\, W_N^{m_1r_1m_2r_2} \sum_{l_{1=0}}^{N_s-1}\sum_{l_2=0}^{N_s-1} x(l_1,m_1;l_2,m_2)W_N^{l_1s_1l_2s_2} \quad . \tag{2.51}$$

The second double summation yields a N_sxN_s 2D DFT of *x* addressed by l_1 and l_2. The outer double summation is another N_sxN_s 2D DFT addressed by m_1 and m_2, except for the twiddle factor, $W_M^{\,m_1r_1+m_2r_2}$. Thus by using equation (2.51), the 2D DFT can be computed by the system shown in Figure 2–10. The index mapping memory provides the parallel inputs to the 2D FFT blocks in proper order. The 2D parallel FFT computes a N_sxN_s FFT in row–column using a N_s–points FFT unit followed by N_s, N_s–point FFT units.

The scheme requires the following hardware (the quantity is indicated inside parentheses): *Ns*–bit shift registers($2N_s$), radix–4 butterfly units ($3N_s log_4 N_s$), memory ($2N^2$ words), interconnection networks of size *N* (4), and address generators (2). It is claimed that the system can perform a 256x256 FFT in 3.27 msec assuming a clock rate of 12.5 MHz. When multiple filters are to be implemented, this hardware has to be duplicated for each filter.

It is possible to build a real–time FFT system around a special purpose FFT chip[6]. The LH9124 DSP chip is a 24–bit fixed point processor with a maximum clock speed of 80 MHz. It performs not only the FFT but also cosine and sine transforms,

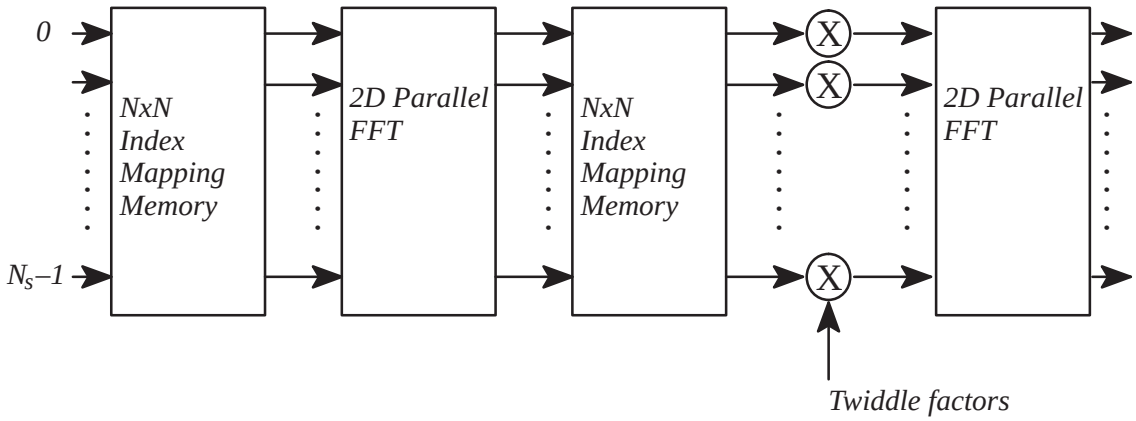


Figure 2–10: System Diagram of the FFT Using A Smaller Kernel

complex and real FIRs and convolution operations. Unique to this processor is the use of the *quasi radix–16* approach instead of the more common radix–2 or radix–4 approaches. The bottleneck of a high speed FFT processor is often found at its I/O; the processor's I/O cannot keep up with the speed of its internal computation. Hence it is desirable to implement the FFT using as high a radix as possible for butterfly operations so that data once brought on chip is processed many times before going out. However a radix beyond 2 or 4 for butterfly operations increases the hardware complexity tremendously. The chip compensates the trade–off by implementing a radix–16 butterfly operation using eight radix–4 butterfly units and on–chip memory.

With a single LH9124 operating at 80 MHz, a 1k complex FFT can be done in 80.7 μsec. Multiple chips can be cascaded to enhance the performance. With 3 stages of a cascaded configuration, a 1k complex FFT can be done in 25.6 nsec. If a row–column FFT is performed on 1k x 1k images with 3 stages cascade configuration, it will take 25.6 μsec x 1024 = 26.2 msec to finish the transform. It requires the following amount of hardware: LH9124 chips (3), LH9320 address generator chips (11), and 1k memory chips (22).

It is thus possible to implement a real–time orientational filter system using the real–time FFT system. However, the system becomes very complicated and requires an immense amount of hardware. It needs at least twice as much hardware as the FFT system described above, since filtering in the frequency domain involves both forward and inverse FFTs. For a system with multiple filters, independent hardware is needed for each filter.

2.2.3.Parallel 1D Convolution

This scheme computes the convolution for each row of an orientational filter, in parallel, and the result of each convolution is accumulated to generate the result of the 2D convolution (Figure 2–11). *M*, 1D convolvers compute the parallel portion of the 2D convolution. The processing required at the k^{th} 1D convolution unit when the center of the filter mask is located at $[n_1, n_2]$ is

$$p_k = \sum_{m=0}^{M-1} I\left[n_1 - \frac{M-1}{2} + m, n_2 - \frac{M-1}{2} + k\right] h[M-m-1, M-k-1] \quad . \tag{2.52}$$

Then the final result of the 2D convolution is

$$y[n_1, n_2] = \sum_{m=0}^{M-1} p_m \quad . \tag{2.53}$$

The idea of this parallel 1D convolution scheme is applied to Gabor functions in [71]. It also takes advantage of symmetric and anti–symmetric characteristics of the Gabor functions.

Figure 2–11 shows only *M* 1D convolvers, 1 Accumulator and 1 image plane. However, in order for the convolvers to operate in parallel, *M* independent memory banks are necessary to let each convolver access image data simultaneously. Also after each row, filter coefficients in each 1D convolver must be updated. The computational complexity of this scheme is $O(N^2M^2)$, and the latency is $O(NM)$. The throughput can be $O(1/t_m)$ by using a pipeline structure. It requires *NM* words of memory with *M* independent banks. For a multiple filter system, the memory banks can be shared, thus *M* banks of *N* memory words are enough for the system.

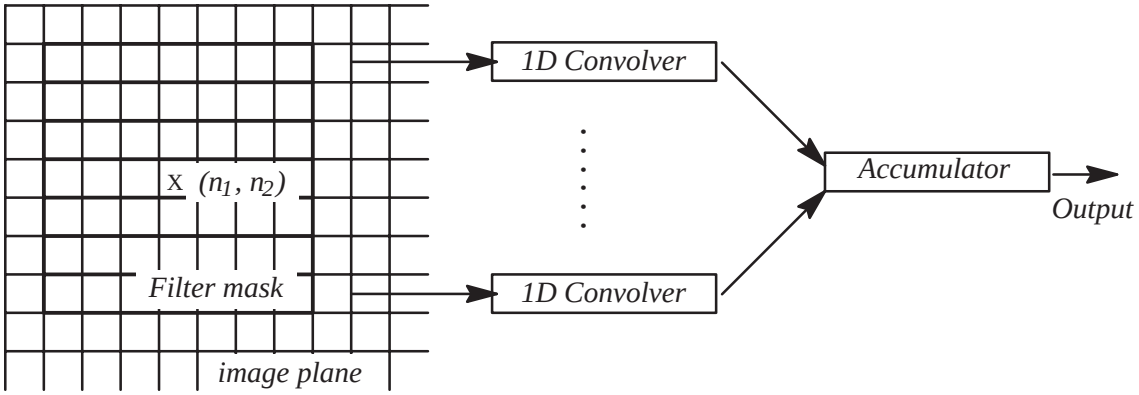
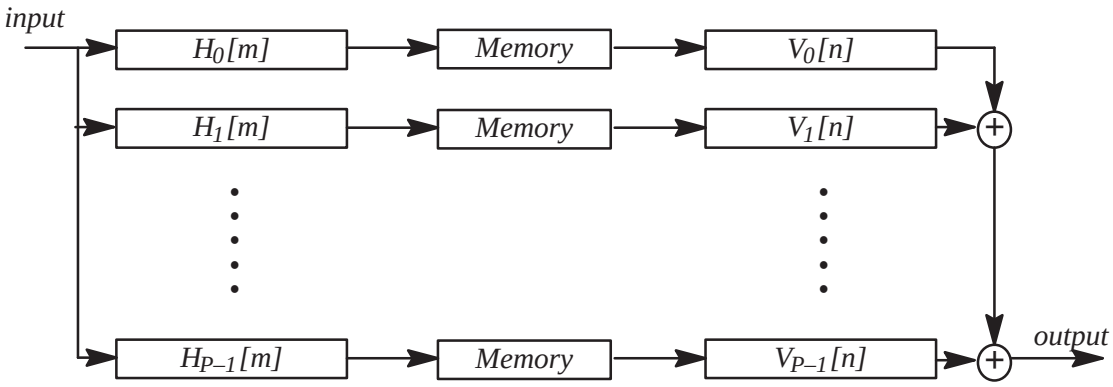


Figure 2–11: Structure of the Parallel 1D Convolution Scheme



$H_k[m]$ is a horizontal filter, and $V_k[n]$ is a vertical filter.

Figure 2–12: Structure of the Separable Approximation Scheme

2.2.4.Separable Approximation Method

The separable approximation method approximates a 2D filter by a linear sum of separable filters[91],

$$h(x,y) \approx \sum_{i=1}^{P} a_i(x)\, b_i(y)\ . \tag{2.54}$$

Its computational scheme is illustrated in Figure 2–12. If a 2D filter is separable, the convolution of the filter can be computed by first applying the horizontal filters to the data along the rows and applying the vertical filters along the columns to the horizontal filtering results. Hence the computational complexity of the filter changes from $O(N^2M^2)$ to $O(N^2MP)$. Computational savings over direct implementation and parallel convolution can be achieved when $2P < M$. In Chapter 3, it will be shown that more computational savings can be achieved using the separable approximation method when multiple filters are to be implemented. By using the separable approximation method, each pair of separable filters, $a_i(x)b_i(y)$ can be computed independently.

The biggest advantage of separable approximation in real–time computer vision applications lies in its intrinsic parallelism; each term in (2.54) can be computed in parallel. The latency is $O(NM)$, and the storage requirement is $O(NMP)$. The throughput can be $O(1/t_m)$ by using a pipeline structure. In Chapter 3, it will be shown that the amount of storage required can

be *O(NM)* no matter how many filters are to be implemented. Because of the regular filter bank structure, implementation of this scheme is very simple.

2.2.5.Comments

The direct method can be implemented to achieve a small latency and a high throughput using parallel processing. The storage requirement is small and does not grow as the number of filters increases. However, the computational complexity becomes high for large filters. This implies that a large amount of hardware is needed to achieve the small latency and the high throughput. The method satisfies the first, the second and the fourth criteria of the implementation criteria, but violates the third criterion.

Parallel 1D convolution can be viewed as a parallel implementation of the direct method. Thus the same arguments can be applied to this method as the direct method. It satisfies the first, the second and the fourth criteria of the implementation criteria, but violates the third criterion. Due to the high computational complexity, the amount of hardware is large. It requires MF_N 1D convolvers to implement F_N orientational filters.

The FFT based method can be implemented to achieve a high throughput by using real–time FFT modules. The computational complexity is smaller for large filters than the direct and the parallel 1D convolution methods. However, it has a large latency due to the FFT and inverse FFT operations involved, and the storage grows linearly as the number of filters increases. Thus the method satisfies the first and third requirements, but fails to satisfy the second and fourth requirements.

The separable approximation method can achieve a high throughput and a small latency. The computational complexity is smaller than the direct method provided that a good approximation can be achieved with a small number of filters. The simple filter bank structure of the method is very attractive for an inexpensive implementation. The storage requirement does not grow as the number of orientational filters in the system increases by employing a *parallel–pipelined implementation* described in Chapter 3. The method satisfies all the implementation criteria provided that a good approximation is achieved with the approximation order smaller than *M/2*. For a multiple filter system, this approximation accuracy and implementation advantage trade–off can be relaxed, and the method satisfies all the implementation criteria provided that a good approximation is achieved with the approximation satisfying $P(1 + F_N) < F_N M$. Chapter 3 gives a detailed explanation. Table 2–2 summarizes the filtering schemes described in this section.

In the next chapter, three different algorithms to derive the separable filters, $a_i(x)$ and $b_i(y)$,in (2.54), are described. The first two algorithms are the singular value decomposition method (*SVD)* originally described in [91] and the orthogonal decomposition method. Orthogonal decomposition has two variations; orthogonal function decomposition (*OFD*) and orthogonal sequence decomposition (*OSD*). OSD is a discrete version of OFD and more suitable for digital computation than OFD. The performance of OSD is equivalent to that of OFD, although it can perform the approximation much faster. SVD produces less error in the approximation than the OSD approximation with the same order. However, OSD has a simpler implementation when a system is comprised of multiple filters, and it is easier to update the OSD filter coefficients. The third algorithm is called SV/OSD and possesses the implementation advantages of OSD and the performance advantages of SVD.

Table 2–2: Comparison Summary of Computational Schemes

The comparison summarized in this table is based on the computation of a single orientational filter. The comparison based on the computation of multiple orientational filters differs, and will be discussed in the next chapter after the separable approximation technique is fully defined.

	Comp. Complexity	Latency	Storage	Remarks on Hardware
Direct	N^2M^2	NM	NM	
FFT	$(N+M)(N+M)\log N$	$(N+M)(N+M)\log N$	$(N+M)(N+M)$	*FFT* and *IFFT* modules
Parallel Conv	N^2M^2	NM	NM	M 1D convolvers
Separable Approx.	N^2MP	NM	NM	P 1D convolvers

Another advantage of separable approximation is that a multi–resolution image decomposition can be performed efficiently by using a tree–structured filter bank. The algorithm for multi–resolution decomposition is described in Chapter 5.

2.3.Applications in Computer Vision

This section reviews the use of various orientational filters in computer vision applications. The purpose is to show how useful and important orientational filters are in computer vision applications.

As stated in the previous chapter, orientational filters are useful for detecting features, mainly edges, and for analyzing spatial–frequency characteristic of images. This section is divided into two parts: orientational filters as feature extractors (edge detectors) and as frequency analyzers.

2.3.1.Feature Extraction

David Marr and his colleagues were the first to approach computational vision problems based on the human visual system in a systematic way[57]. Their pioneering work in the late 70s opened new directions in computer vision research. Their motivation was to produce a primitive but rich description of the image to be used for image analysis. They suggested that intensity changes or edges are good features for image description, hence a good edge detector is essential for image analysis[58].

Various edge operators appeared in the literature during the 70s, such as the Sobel operator, Robert operator and Prewitt operator[40]. They are small in size (normally 3x3), and most of them are separable. Their performances are acceptable for simple images such as binary images and simple applications. However, these operators are prone to noise and respond well only to sharp edges. Also their orientational selectivities are quite limited. As images and applications become more complicated, pre–processing for noise suppression and post–processing for orientational analysis, edge connection, and spurious edge elimination have to be combined to fulfil the goal, and the computational advantage of these operators is lost.

The contributions of Marr's work in edge detection algorithms can be summarized in two points. First he suggested to perform edge detection at multiple different resolutions, since intensity changes can occur at different scales. Gaussian filters of different sizes are used to smooth the image in different resolutions. The Gaussian is used because of its good spatial–frequency localization. The second point is the use of a zero–crossing of a second derivative operator rather than a peak of a first derivative operator. Both are sensitive to intensity changes, but finding a zero crossing is computationally easier than finding a peak. The result is an edge detector called a Laplacian of Gaussian operator. It is expressed as

$$\nabla^2 G(r) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{r^2}{2\sigma^2}\right] e^{-r^2/2\sigma^2} \quad . \tag{2.55}$$

The function looks like a Mexican hat and resembles the response of ganglion–cells (Appendix A). As can be seen from (2.55), the Laplacian of Gaussian is not directionally sensitive. It was designed this way because of computational and implementational advantages.

As Marr himself pointed out, the Laplacian of Gaussian cannot locate edges properly if intensity varies non–linearly around the edges, and some experiments show that directional selectivity is an important property for a good edge detector[36][30]. There are two approaches for introducing directional selectivity to edge operators. One is a post–processing approach and the other is a multiple filter approach. In the post–processing approach, edge direction and edge strength are estimated from outputs of two directional filters h_1 and h_2, normally tuned to horizontal and vertical directions respectively. The edge strength at a location *(x,y)* of an image *I* is estimated by

$$S(x,y) = \sqrt{\Delta_1(x,y)^2 + \Delta_2(x,y)^2} \ , \tag{2.56}$$

and the edge direction is estimated by

$$\theta_d(x,y) = \tan^{-1}\left\{\frac{\Delta_1(x,y)}{\Delta_2(x,y)}\right\} \ , \tag{2.57}$$

where $\Delta_1 = I * h_1$ and $\Delta_2 = I * h_2$ with * being a convolution operator. An advantage of the post–processing approach is simple implementation; only two orientational filters are needed. Disadvantages are that the edge direction computed by post–processing is not precise, especially around a texture, multiple edges, and corners.

In the multiple filter approach, more than two orientational filters are provided, and each orientational filter detects edges of a particular direction and its vicinity. Advantages of this approach are that it is more robust and flexible than the post–processing approach, and it can be used for region analysis such as texture analysis. A disadvantage is that it is more expensive to implement.

Based on three criteria, Canny derived a 1D optimal edge detector for step edges with the presence of white noise[17]. The following is the criteria.

- The produced outputs have low probability of failing to mark real edges, and a low probability of marking spurious edges.
- The points marked as edges should be as close to the true edge as possible.
- Only one response should be marked for a single edge.

These criteria were mathematically formulated and the optimal step edge detector was generated numerically. It turned out that the numerical result is very close to the first derivative of the Gaussian. For 2D cases, his edge detector is a concentric first derivative of the Gaussian written as:

$$E_{canny}(x,y) = \tilde{\boldsymbol{n}} \cdot \nabla g_a(x,y) \tag{2.58}$$

where $\tilde{\boldsymbol{n}}$ is an estimated edge orientation obtained by

$$\tilde{\boldsymbol{n}} = \frac{\nabla(g_a * I)}{|\nabla(g_a * I)|}, \tag{2.59}$$

where I is the input image. The edge location is estimated as a local maximum in the direction of $\tilde{\boldsymbol{n}}$, and the edge strength is obtained by $|\nabla(g_a * I)|$.

He also discussed the importance of having edge detectors with different widths (multi–resolution edge detectors) for different sizes of edges, and having orientational tuning of the operator for improving both detection and localization of edges.

Natural scenes are composed of not only step edges but typically a combination of steps, peak and roof edges. In [65], Perona and Malik concluded that a linear filter cannot localize these composite edges properly, but they can be localized by quadratic filtering using a linear filter *h[n]* and its Hilbert transformed pair $\tilde{h}[n]$. The edge detection looks for local maxima in $[(h * I)^2 + (\tilde{h} * I)^2]$.

2.3.2.Frequency Analyzer

Another strength of orientational filters in image analysis applications is their region analysis capability that is very important in analyzing textured regions. Texture is a very important cue in image analysis[3], especially for natural images which are comprised of a rich set of textures each corresponding to a different object or part of an object. Although there is no rigid definition for the notation 'texture', it is apt to say that texture is "something composed of closely interwoven elements"[3]. Examples of natural textures are shown in Figure 2–14 along with their frequency spectra. These textures are extracted from the Brodatz's photo album[11]. As can be seen, they have their energy concentrated in small regions scattered over the frequency domain, and they can be characterized well by observing the locations of the energy concentrations in the domain. Hence it is natural to utilize frequency domain analysis in order to distinguish different textures.

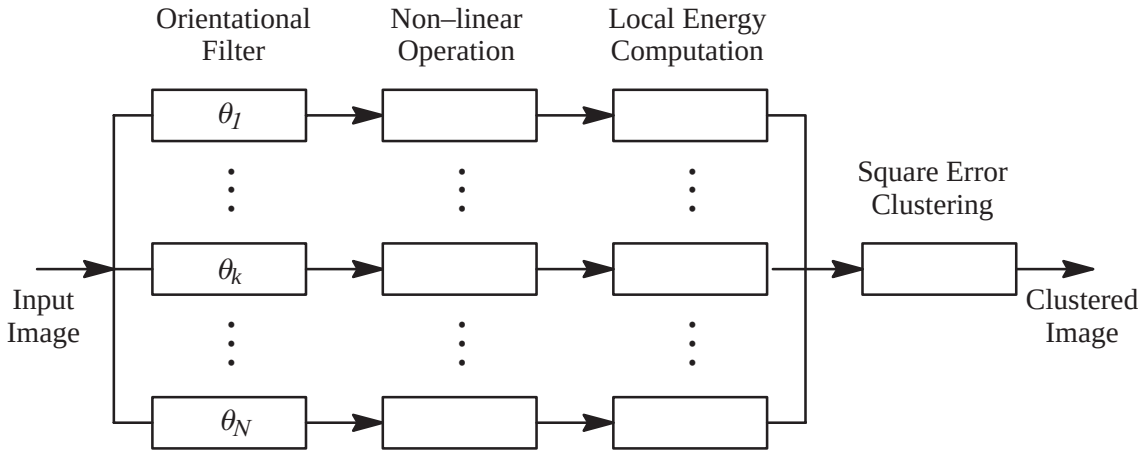
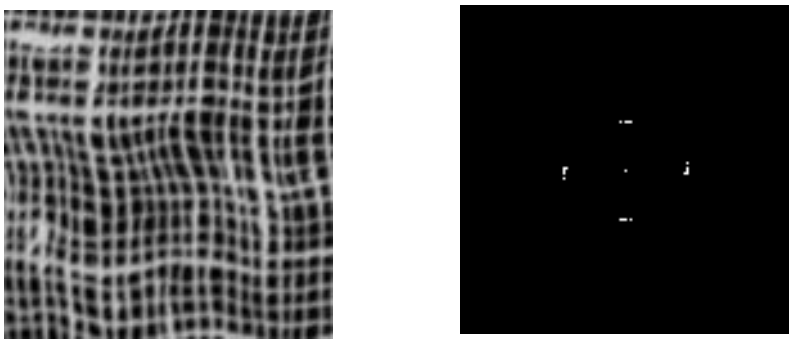
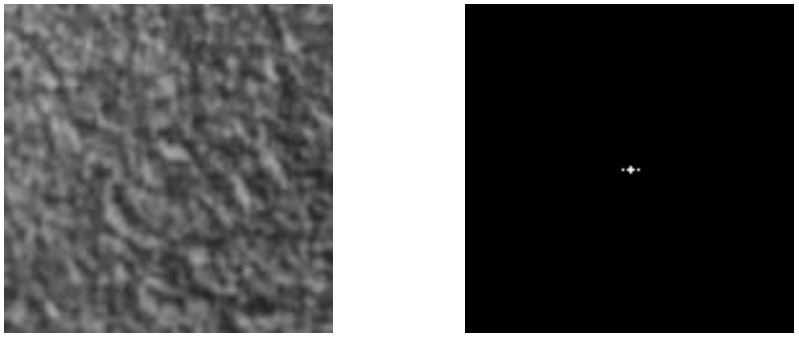


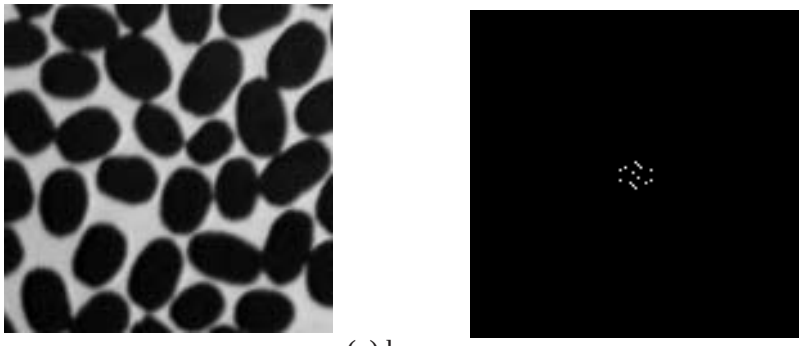
Figure 2–13:Multi–channel Texture Segmentation System Using Orientational Filters



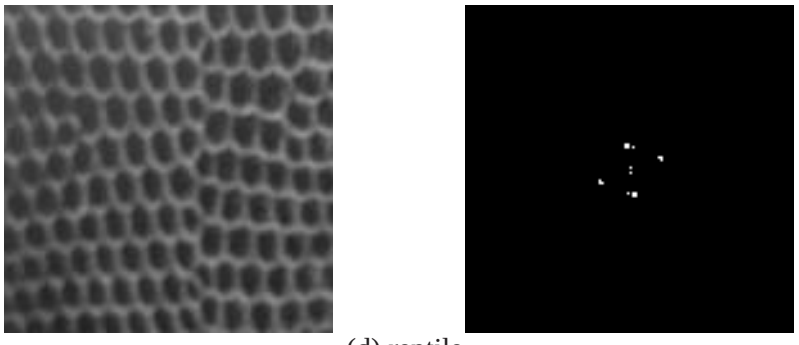
(a) burlap



(b) cork



(c) beans



(d) reptile

Figure 2–14: Samples of Natural Textures and Their Fourier Transforms (Magnitude)

As stated in Chapter 1, an orientational filter useful for image analysis should have good spatial–frequency localization. Many texture segmentation/analysis methods based on orientational filters have been proposed [7][10][15][18][12][29][35][41][54][72][77]. They all consists of multiple orientational filters tuned to different angular and radial frequency ranges followed by some non–linear transforms forming multiple parallel channels. Outputs of the channels are combined at the last stage for robust unsupervised texture segmentation. An example of such a system proposed in [41] is shown in Figure 2–13. Many non–linear operations have been proposed in order to improve the segmentation performance. Since most clustering algorithms are designed to distinguish between classes with non–overlapping centroids, the purpose of these operators is to convert variance disparities into mean value differences. Typical operators are $f(x) = |x|$, $\tanh(x)$, x^2 and $\log|x|$.

2.3.3.Comments

In summary, orientational filters can be used as a feature extractor and region analyzer. Hence, they are useful as a low–level operator for image analysis.

CHAPTER 3

Separable Approximation

Assume that the discrete orientational filters to be approximated originate from continuous orientational functions, and the filters are derived from sampling the continuous functions using a rectangular sampling grid. Throughout this chapter, *h(x,y)* denotes the orientational function. It is sampled at $\{(x_m, y_n)\}$ to form a discrete orientational filter *h[m,n]*. The size of the discrete filter is $M \times M$ as before.

In the separable approximation method, an orientational filter *h(x, y)* is approximated by a linear sum of separable filters as

$$h(x,y) \approx \sum_{i=1}^{P} a_i(x)\, b_i(y)\,. \tag{3.1}$$

In this form, the orientational filter can be implemented by *P* banks of separable filters operating in parallel followed by accumulators (Figure 2–12). As described in 2.2.4, this scheme enables fast operation of the filter and reduces the hardware complexity significantly. The remaining question is whether the scheme provides a sufficiently accurate approximation. Three approximation methods are introduced in this section. They are (1)Singular Value Decomposition (*SVD*), (2)Orthogonal Sequence Decomposition (*OSD*) and (3)Singular Value/ Orthogonal Sequence Decomposition (*SV/OSD*).

3.1.Singular Value Decomposition

3.1.1.Algorithm

The idea of using SVD for decomposing a non–separable 2D filter into a separable form first appeared in[91]. The SVD method is a discrete approximation that is performed on a discrete filter. The approximation is applied directly to *h[m,n]*. Consider *h[m,n]* as a $M_x \times M_y$ matrix $\boldsymbol{A}$, where M_x is the number of columns, M_y is the number of rows and $M_y \geq M_x$. If $M_x > M_y$, $\boldsymbol{A}$ represents the transposition of *h[m,n]*. Then the matrix $\boldsymbol{A}$ can be decomposed into the form

$$\boldsymbol{A} = \boldsymbol{U}\,\boldsymbol{W}\,\boldsymbol{V}^T\,. \tag{3.2}$$

$\boldsymbol{U}$ is a $M_x \times M_y$ column–orthonormal matrix, and each column vector is an eigenvector of $\boldsymbol{A}\boldsymbol{A}^T$. $\boldsymbol{V}$ is a $M_y \times M_y$ column–orthonormal matrix, and each column vector is an eigenvector of $\boldsymbol{A}^T\boldsymbol{A}$. $\boldsymbol{A}\boldsymbol{A}^T$ and $\boldsymbol{A}^T\boldsymbol{A}$ are non–negative, symmetric and have the identical set of eigenvalues, $\{\lambda_i\}$. $\boldsymbol{W}$ is a $M_y \times M_y$ diagonal matrix with positive or zero elements. Each non–zero diagonal element w_i in $\boldsymbol{W}$ is a square–root of the eigenvalue, λ_i. This decomposition is called Singular Value Decomposition (*SVD*). Equation (3.2) can be also written as

$$\boldsymbol{A} = \sum_{i=1}^{M_y} w_{ii}\, \overline{u}_i \times \overline{v}_i\,, \tag{3.3}$$

where $\overline{u}_i$ and $\overline{v}_i$ are i^{th} column vector of $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively, and $\times$ represents an outer product operator. Suppose $\boldsymbol{W}$ is arranged so that $w_{ii} \geq w_{jj}$ if $i < j$, and corresponding rows in $\boldsymbol{U}$ and $\boldsymbol{V}$ are shuffled so that equation (3.3) still holds. Then matrix $\boldsymbol{A}$ can be approximated by $\tilde{\boldsymbol{A}}$, where

$$\tilde{\boldsymbol{A}} \approx \sum_{i=1}^{P} w_{ii}\, \overline{u}_i \times \overline{v}_i\,. \tag{3.4}$$

This is a P^{th} order separable approximation of $\boldsymbol{A}$, where the u_is are horizontal filters and the v_is are vertical filters, or vice versa. The weight w_{ii} can be incorporated into either u_i or v_i, or can be split between u_i and v_i.

3.1.2.Properties

The coefficients $\{w_{ii}\}$ are non–negative, since they are square roots of the eigenvalues of the matrix $\boldsymbol{AA^T}$. If the rank of $\boldsymbol{A}$ is r_A, then there are at most $r_{\boldsymbol{A}} \leq M_x$ nonzero eigenvalues. Given $\{\overline{u_i}\}$, $\{\overline{v_i}\}$ can be computed by

$$\overline{v_i} = \frac{1}{w_{ii}} \boldsymbol{A}\, \overline{u_i} \quad . \tag{3.5}$$

The approximation (3.4) is the best least squares rank–P approximation of $\boldsymbol{A}$ if the elements w_{ii} are in decreasing order of magnitude. The least square error of the approximation (3.4) is

$$\epsilon^2_{SVD} = \sum_{i=1}^{M} \sum_{j=1}^{N} |A_{i,j} - \tilde{A}_{i,j}|^2 \tag{3.6}$$

where $\tilde{\boldsymbol{A}}$ is the result of approximation (3.4). Equation (3.6) reduces to

$$\epsilon^2_{SVD} = \sum_{i=P+1}^{r_{\boldsymbol{A}}} w_{ii} \quad . \tag{3.7}$$

Hence by increasing the order of the approximation, P, up to r_{A}, the algorithm is guaranteed to converge to the original filter to be approximated.

Given the precision of the approximation required for a certain application, it is easy to determine the order of approximation necessary to achieve the requirement by observing the eigenvalues $\{\lambda_i\}$. For example, if the application requires the approximation error to be less than ϵ_t, then the approximation order should be the smallest P which satisfies

$$\epsilon_t^2 = \sum_{i=P+1}^{r_{\boldsymbol{A}}} w_{ii} \quad . \tag{3.8}$$

3.2.Orthogonal Sequence Decomposition

The orthogonal decomposition method approximates a continuous orientational function by orthogonal projection. The approximation is performed at discrete points to form a discrete orientational filter.

As stated above, the SVD method is optimal in a least squares sense. Hence other approximation methods cannot perform better than the SVD in the same sense. However, the method requires distinct pairs of horizontal and vertical filters for each orientational filter to be approximated. Thus the total computation is approximately $2F_N N^2 PM$. With the orthogonal decomposition method described here, the number of filters is less than the SVD method for the same order of approximation. This is because one set of filters, either the horizontal or the vertical direction, is merely a set of orthogonal sequences used for the decomposition, and are independent of the functions to be approximated. By performing *orthogonal filters* prior to the other set of filters (*projection filters*), the output of the orthogonal filters can be shared among all other orientational filters in the system. Figure 3–1 compares systems with multiple orientational filters implemented by the SVD method and the OSD method. The total computation for OSD is approximately,

$$C_{OSD} = N^2MP + F_N N^2 MP = N^2 MP(F_N + 1) \quad . \tag{3.9}$$

Therefore it is possible for OSD to achieve a better approximation than SVD with the same amount of computation. Figure 3–2 illustrates this claim. It plots

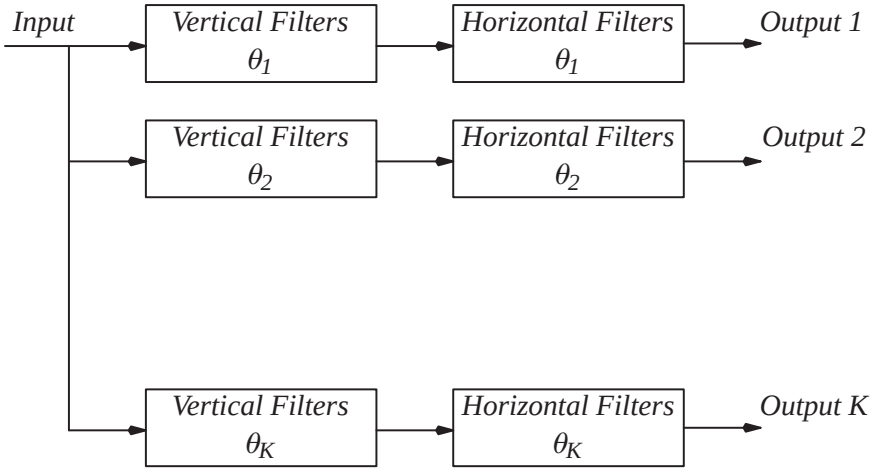
$$\gamma = \frac{2F_N N^2 PM}{N^2 PM(1 + F_N)} = \frac{2F_N}{1 + F_N} \tag{3.10}$$

which is the ratio of computation between SVD and OSD at the same approximation order. At F_N=10, γ is equal to 1.82. Suppose a sufficient approximation of F_N orientational filters can be achieved with the approximation order at P_S using SVD. OSD cannot achieve the same performance as SVD with the approximation order at P_S, however, if OSD can achieve the same performance with the approximation order at $P_O < 1.82P_S$, OSD achieves the same performance with less computation than SVD.

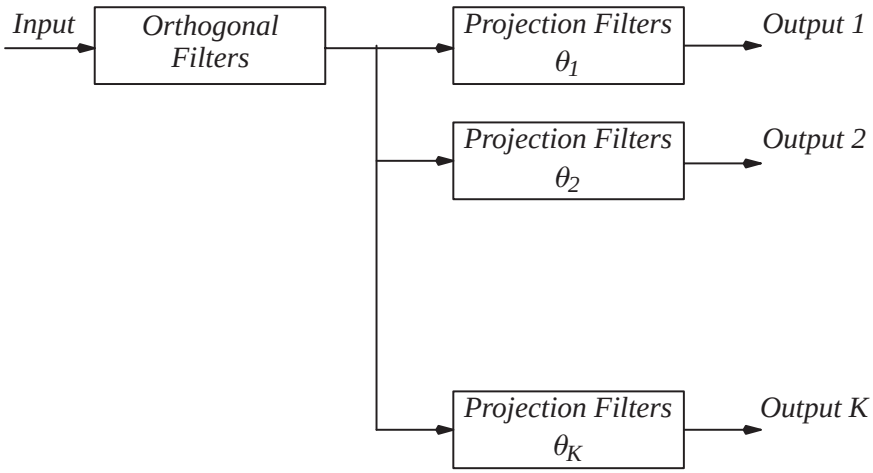
Therefore it is possible that even though OSD cannot outpeform SVD with the same approximation order, it can outperform SVD with the same amount of computation and implementation cost.

3.2.1.Algorithm

This section describes the orthogonal decomposition algorithm. First, Theorem 1 provides a proof that orthogonal decomposition can be used for separable approximation.



(a) Implementation based on the SVD Method



(b) Implementation based on the OSD Method

Figure 3–1: System Implementation of Multiple Orientational Filters

Definition 1: A set of functions $\{\phi_i(x)\}_{j\geq 0}$ is orthogonal over the interval *[a,b]* with respect to a weighting function *w(x)* if it satisfies the orthogonality condition:

$$\int_a^b w(x)\phi_i(x)\phi_j^*(x)dx \quad = \quad C_i\,\delta_{i,j} \quad , \tag{3.11}$$

where $\phi^*(x)$ is the conjugate of $\phi(x)$, C_i is a constant and δ_{ij} is the Kronecker delta function.

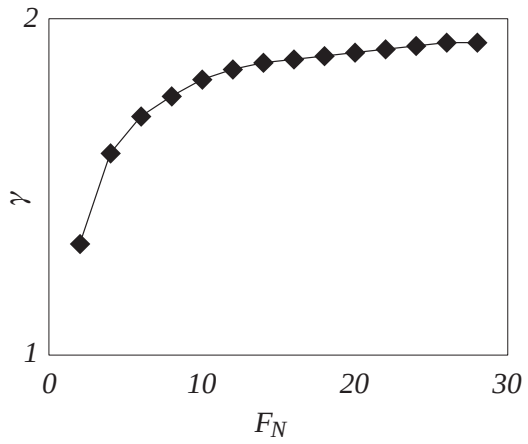


Figure 3–2: Computation Ratio of SVD vs OSD

Definition 2: A 2D function *f(x,y)* belongs to $S_1[a_1, b_1]xS_2[a_2, b_2]$ where S_1 and S_2 are 1D linear sub–spaces, if $f(x,\tilde{y}) \in S_1,\ \forall\, \tilde{y} \in [a_2, b_2]$ and $f(\tilde{x},y) \in S_2,\ \forall\, \tilde{x} \in [a_1, b_1]$.

Theorem 1: Let $\{\phi_i(x)\}_{i\geq 0}$ be the set of orthogonal functions over *[a,b]*, and assume $\{\phi_i(x)\}_{i\geq 0}$ forms a basis of a linear sub–space $S_1[a,b]$. Then any set of functions $\{f(x,y)\} \in (S_1xS_2)$ can be decomposed into a separable form

$$f(x,y) = \sum_i^{\infty} \psi_i(y)\phi_i(x) \text{ , where} \tag{3.12}$$

$$\psi_i(y) = \frac{1}{C_i}\int_{a_1}^{b_1} w(x)f(x,y)\phi_1(x)dx \tag{3.13}$$

and $\psi_i(x) \in S_2$.

Proof: Since $\{\phi_i(x)\}_{i\geq 0}$ spans S_1, at any line $y = \tilde{y}$, *f(x, ỹ)* can be represented by $\{\phi_i(x)\}_{i\geq 0}$ as

$$f(x,\tilde{y}) = \sum_i^{\infty} \psi_i(\tilde{y})\,\phi_i(x) \quad . \tag{3.14}$$

Due to orthonormality of $\{\phi_i(x)\}_{i\geq 0}$,

$$\int_{a_1}^{b_1} w(x)\phi_i(x)f(x,\tilde{y})dx = \int_{a_1}^{b_1} w(x)\phi_i(x)\sum_j \psi_j(\tilde{y})\phi_j(x)dx = \sum_j \psi_j(\tilde{y})\int_{a_1}^{b_1} w(x)\phi_i(x)\phi_j(x)dx = C_i\psi_i(\tilde{y})\,. \tag{3.15}$$

Hence (3.13) holds. Now, at any line $x = \tilde{x},\ f(\tilde{x},y)$ can be written as

$$f(\tilde{x},y) = \sum_i \psi_i(y)\phi_i(\tilde{x}) = \sum_i \beta_i\,\psi_i(y) \in S_2 \tag{3.16}$$

where $\beta_i = \phi_i(\tilde{x})$. Therefor $\psi_i(y) \in S_2$. □

Therefore, any continuous function $h(x,y) \in S[a_-,a_+]xS[b_-,b_+]$ can be decomposed by

$$h(x,y) = \sum_i^{\infty} \alpha_i(y)T_i(x) \text{ , with} \tag{3.17}$$

$$\alpha_i(y) = \frac{1}{C_i} \int_{a_-}^{a_+} w(x)\, h(x,y)\, T_i(x) dx \,, \tag{3.18}$$

and $\left\{T_j(x)\right\}_{j\geq 0}$ is an orthogonal basis of a linear sub–space $S[a_-, a_+]$.

If $h(x,y)$ is not in SxS, then the decomposition is not exact. However $a_i(y)$ obtained by (3.18) minimizes the distance, $\| h(x,y) - \sum_i^{\infty} a_i(y) T_i(x) \|^2$.

In order to approximate $h[m,n]$, $\{T_i(x)\}$ and $\{\alpha_i(y)\}$ are sampled at $\{x_m\}$ and $\{y_n\}$ respectively. Also in reality, the summation has to be truncated at some point. A Pth order approximation selects the first P set of functions. This decomposition is called *Orthogonal Function Decomposition* (OFD) and is expressed as

$$h[m,n] \approx \sum_i^P \alpha_i(y_n) T_i(x_m) \quad . \tag{3.19}$$

The support ranges of $\left\{T_j(x)\right\}_{j\geq 0}$ and $h(x,y)$ are not necessarily the same. It is more likely that they are different. If the support range of $h[x,y]$ is $[b^x_-, b^x_+]X[b^y_-, b^y_+]$, the range of $\left\{T_j(x)\right\}_{j\geq 0}$ needs to be adjusted by a change of variable

$$\hat{x} = \frac{(a_+ - a_-)\, x - (a_+ b^x_- - a_- b^x_+)}{(b^x_+ - b^x_-)} \quad . \tag{3.20}$$

It is often time consuming to evaluate the integral of (3.18). Instead of using orthogonal functions $\{T_i(x)\}$, discrete orthogonal sequences associated with $\{T_i(x)\}$ can be utilized. The orthogonal functions $\{T_i(x)\}$ are sampled at a set of discrete points $\{x_k\}$ $(0 \leq k < M_P)$ to form a set of discrete sequences $\left\{T_i(x_k)\right\}_{0\leq i,k<M_P}$ which satisfy the following discrete orthogonality condition,

$$\sum_{k=1}^{M_P} T_i(x_k)\, T_j(x_k) = C_i\, \delta(i,j) \tag{3.21}$$

where M_P is the sampling size for the orthogonal sequences and C_i is a constant. Then the decomposition is similar to the continuous case,

$$h[m,n] \approx \sum_i^P \alpha_i[n]\, T_i(\tau_s m) \tag{3.22}$$

where τ_s is the sampling period of $h(x,y)$, and

$$\alpha_i[n] = \frac{1}{C_i} \sum_{k=1}^{M_P} h(x_k, \tau_s n)\, T_i(x_k) \,. \tag{3.23}$$

M_P is different from the sample size of the discrete filter $a_i[n]$ which is assumed to be M. Equation (3.23) can be viewed as an approximation to the integral of (3.18) with a reasonably smooth filter $h(x,y)$ as stated in the following lemma. This discrete version of the OFD is called *Orthogonal Sequence Decomposition* (OSD).

Lemma 3: Denote Δ_{max} as the largest sample step in $\{x_k\}$, i.e.

$$\Delta_{\max} = \max_{0\leq k<M_P} (x_{k+1} - x_k) \,. \tag{3.24}$$

If $\left|g' = \frac{d}{dx}\left\{h(x,y_n)T_i(x)\right\}\right| < \infty$ in $a_-<x<a_+$, then $a_i[n]$ in (3.23) converges to $a_i(y_n)$ as $\Delta_{max}->0$. Thus,

$$\lim_{\Delta_{\max}\to 0} a_i[n] = a_i(y_n) \,. \tag{3.25}$$

Proof:

$$a_i(y_n) = \frac{1}{C_i} \int_{a_-}^{a_+} h(x,y_n) T_i(x) dx = \frac{1}{C_i} \sum_{k=0}^{M_P} h(x_k, y_n) T_i(x_k) + M_P\, O(\Delta^2_{\max}\, g')$$

$$= \quad a_i[n] + O(\Delta_{\max} g') \quad . \tag{3.26}$$

Hence, (3.25) holds. □

Table 3–1 compares computation time and accuracy of the approximation using OFD and OSD with the approximation order *P=12*. Sine functions and sine sequences are used for orthogonal functions and orthogonal sequences, respectively. The integration of OFD is computed using Simpson's method, and the computation terminates when the result does not change more than 1.0^{-8}. M_P is set to *10M+1* for the OSD. The real part of Gabor functions with various parameters are approximated. The computation time is measured as the time required to compute *one* projection operation, (3.18) and (3.23), on a Sparc II workstation. The criteria of approximation performance are a normalized peak error and a normalized energy error. The normalized peak error is defined as $\max_{i,j}|I[i,j] - Ap[i,j]| / \max_{i,j}|I[i,j]|$ where I is the original filter to be approximated and Ap is the result of the approximation. The normalized energy error is defined as $\sum_{i,j}(I[i,j] - Ap[i,j])^2 / \sum_{i,j} I[i,j]^2$. As can be seen in Table 3–1, the OSD performs as well as the OFD, and has much less computation time.

Table 3–1: Comparison of Approximation Using OFD and OSD

- $\theta=\pi/4$, $\alpha=10.0$, $\sigma_y=0.6$, $\sigma_y/\sigma_x=1.5$, $M=13$

	Computation Time	Peak Error	Energy Error
OFD	1.3 sec	1.86 %	0.0245 %
OSD	0.033 sec	1.86 %	0.0245 %

- $\theta=\pi/4$, $\alpha=10.0$, $\sigma_y=0.4$, $\sigma_y/\sigma_x=2.0$, $M=19$

	Computation Time	Peak Error	Energy Error
OFD	1.8 sec	1.63 %	0.0607 %
OSD	0.045 sec	1.63 %	0.0607 %

- $\theta=\pi/4$, $\alpha=10.0$, $\sigma_y=0.45$, $\sigma_y/\sigma_x=3.0$, $M=23$

	Computation Time	Peak Error	Energy Error
OFD	2.2 sec	4.56 %	0.421 %
OSD	0.053 sec	4.56 %	0.421 %

3.3.SV/OSD

The comparison of SVD and OSD showed that SVD has an advantage in its approximation performance and a disadvantage in its implementation, while the reverse is true for OSD. Hence there are two approaches to improve the separable approximation method. The first one is to improve the implementation of SVD by adapting the implementation structure of OSD. The other is to improve the performance of OSD by obtaining the set of orthogonal sequences which provides the best fit to a given set of orientational filters to be approximated. It will be shown that these two approaches lead to the same algorithm.

3.3.1.Algorithm

A set of orientational filters $\{h_k(x,y)\}$ $(0 \le k < O_N)$ is to be approximated, where each filter is represented as an FIR filter of the size $M \times M$. Typically, $\{h_k(x,y)\}$ is generated from the same prototype filter with different orientations. The orientational filters are combined to form a $M \times MF_N$ matrix $\boldsymbol{A}$ as shown in 3–3. The matrix $\boldsymbol{A}$ is called an *approximation matrix*. A set of M adjacent columns from column kM $(0 \le k < F_N)$ through column $(k+1)M-1$ constitutes the orientational filter h_k. SVD is performed on the approximation matrix to produce r_A vectors of length M, r_A vectors of the length (MF_N) and the corresponding eigenvalues. The first set of vectors are denoted $\overline{u_i}$ $(0 \le i < r_A)$, the second set of vectors are denoted $\overline{v_i}$ $(0 \le i < r_A)$, and the square root of the eigenvalues are denoted as w_{ii}. Then h_k can be expressed by the separable form,

$$h_k[m,n] = \sum_{i}^{r_A} w_{ii}\, \overline{u}_m\, \overline{v}_{kM_y+n} \quad . \tag{3.27}$$

Assume the weights, w_{ii}, are sorted by magnitude in descending order, and the corresponding vectors, $\overline{u_i}$ and $\overline{v_i}$, are shuffled in correspondence with w_{ii} so that the decomposition (3.27) still holds. Then the P^{th} order separable approximation is

$$h_k[m,n] \approx \sum_{i}^{P} w_{ii}\, \overline{u}_m\, \overline{v}_{kM_y+n} \quad . \tag{3.28}$$

Since the $\overline{u_i}$ are orthogonal vectors, and are common to all the orientational filters to be approximated, it is an OSD, although the vectors are generated from SVD. Hence this separable approximation method accomplishes the advantages of both SVD and OSD; it has the good approximation performance of SVD and the simple implementation of OSD. The amount of computation required to implement F_N orientational filters is

$$C_{SV/OSD} = N^2MP + F_N N^2 MP = N^2 MP(F_N + 1) \quad . \tag{3.29}$$

The error ε_A introduced by the approximation is measured as a sum of approximation errors in a least squares sense,

$$\epsilon_A = \sum_{k=1}^{F_N} \sum_{m,n} \left\{ h_k[m,n] - \sum_{i=0}^{P} w_{ii} \overline{u}_m \overline{v}_{kM+n} \right\}^2 = \sum_{i=P+1}^{r_A} w_{ii}^2 \quad . \tag{3.30}$$

In this sense, $\overline{u_i}$ are the optimum orthogonal sequences for approximating $\{h_k\}$. Since this algorithm is a combination of SVD and OSD, it is denoted as SV/OSD.

3.4.Convergence of Approximation

In this section, the convergence of the separable approximation method is examined. The next lemma gives an upper bound of the energy error for SVD and SV/OSD.

Lemma 4: A normalized energy error for an approximation order P is at most $\frac{M-P}{M}$ for SVD and SV/OSD.

Proof: From (3.7), a normalized energy error for an approximation order P is

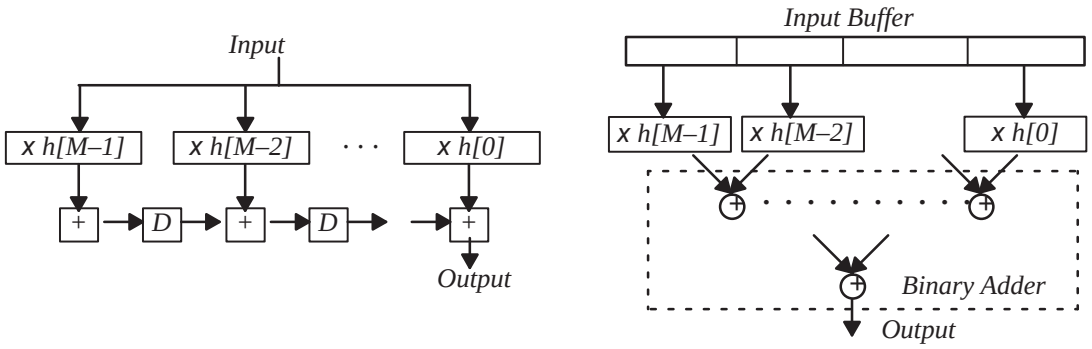
$$\epsilon_P^2 = \frac{\sum_{i=P+1}^{M} w_{ii}}{\sum_{i=1}^{M} w_{ii}} \quad . \tag{3.31}$$

By choosing the P largest eigen values (w_{ii}) for the approximation, the worst case is when all the eigenvalues have the same magnitude. In the case, the normalized energy error is

$$\bar{\epsilon}_P^2 = \frac{M-P}{M} \quad . \ \square \tag{3.32}$$

There is no guaranteed upper error bound for OSD. The approximation may not converge to the function being approximated if the function does not reside in the sub–space spanned by the set of orthogonal functions used for the approximation.

Figure 3–3: Construction of Matrix *A* with Multiple Orientational Filters



(a) Pipelined Filtering

(b) Parallel Filtering

Figure 3–4: 1D Spatial Filter Units

3.5.Implementation Scheme for SV/OSD

As described above, SV/OSD possesses good approximation performance (its performance will be demonstrated in Chapter 4.), good implementation characteristics, and good convergence behavior. Finally, this section addresses the implementation scheme tailored for SV/OSD so that all the implementation criteria are satisfied.

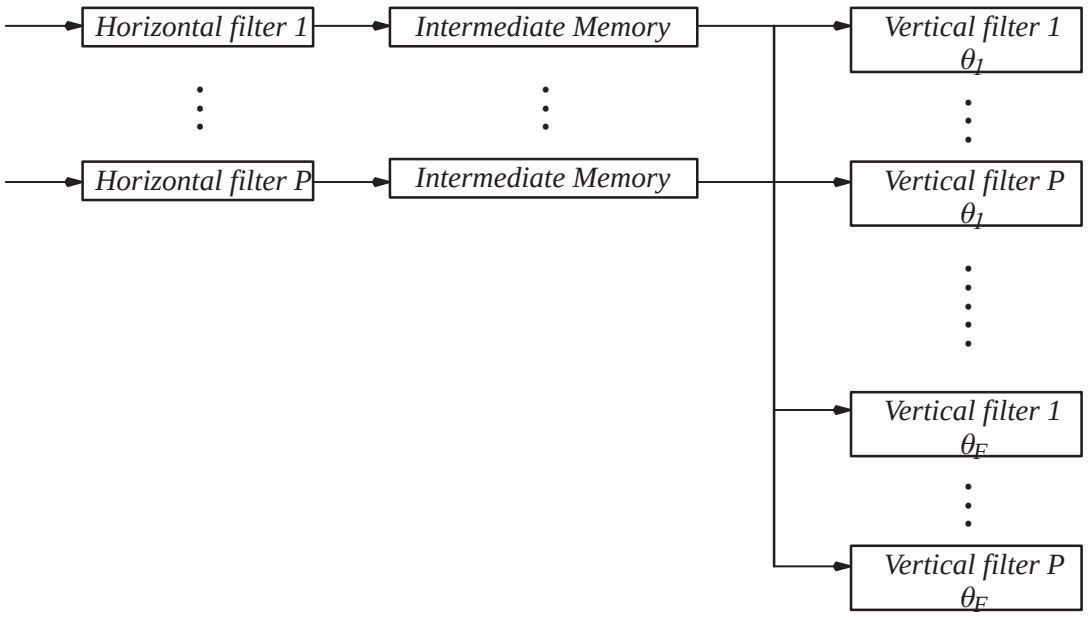
Before discussing the details of the design, basic 1D filter units are discussed. There are two possible schemes to implement 1D spatial filtering. They are called *pipelined filtering* and *parallel filtering*, and are depicted in Figure 3–4. Pipelined filtering takes one input at a time sequentially. The input is multiplied with all the filter coefficients at the same time, and each result of multiplication is accumulated at an accumulator attached to the multiplication unit. All the accumulators are connected together through delay units, and the output of the last accumulator in the chain is the output of the filter. The parallel filtering takes M inputs at a time, and each input is multiplied with a corresponding filter coefficient. The results of the multiplications are added through the binary tree adder to form the output. Both schemes require M multipliers and $M–1$ adders. Pipelined filtering is suitable for a sequential input stream, and parallel filtering is suitable for a parallel input stream. For SV/OSD, pipelined filtering is suitable for horizontal filtering since the input is coming sequentially in a raster order, and parallel filtering is suitable for vertical filtering since the input can be provided in parallel so that the latency of the system reduces to $O(NM)$. It is important to note that the parallel filtering requires an input buffer so that M parallel inputs can be provided to the filter unit simultaneously.

A point very influential to the structure of the filter system is the order of filtering. Separable filtering can be done in either the horizontal–vertical order or the vertical–horizontal order. With the horizontal–vertical order, the system requires an intermediate memory of size NM after each horizontal filter, thus requires a total of NMP words of memory. With the vertical–horizontal order, the intermediate memory can be shared among the vertical filters because the inputs to the vertical filters are identical. Thus the system requires a total of NM words of memory. Figure 3–5 depicts this difference. The amount of intermediate memory needed is NM no matter how many orientational filters are to be implemented.

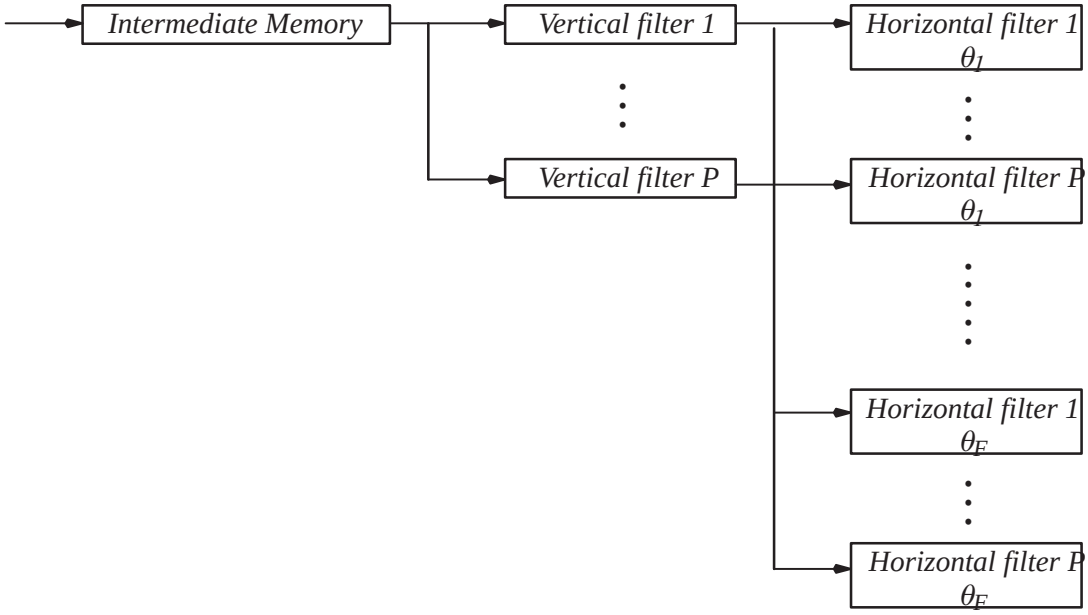
Note that the orthogonal filters are implemented as horizontal filters and the projection filters are implemented as vertical filters in the horizontal–vertical filtering scheme. On the other hand, the orthogonal filters are implemented as vertical filters and the projection filters are implemented as horizontal filters in vertical–horizontal filtering scheme. A care must be taken upon constructing an approximation matrix for target orientational filters. With the vertical–horizontal filtering scheme, each column in the matrix corresponds to a vertical slice of a filter, and with the horizontal–vertical filtering scheme, it corresponds to a horizontal slice of a filter. Figure 3–6 depicts these two ways of constructing the approximation matrix.

Because of the memory requirement advantage the vertical–horizontal filtering scheme possesses over the horizontal–vertical filtering scheme, it is assumed from now on that the vertical–horizontal filtering is employed to implement SV/OSD.

Due to Lemma 4, the approximation converges at least linearly to the original filter as the approximation order approaches M. Thus, even for the worst case, the computational complexity of the filters using the SV/OSD is only slightly more than the complexity of the direct method. In the worst case, the ratio of the computational complexity between the separable approximation and the direct method is

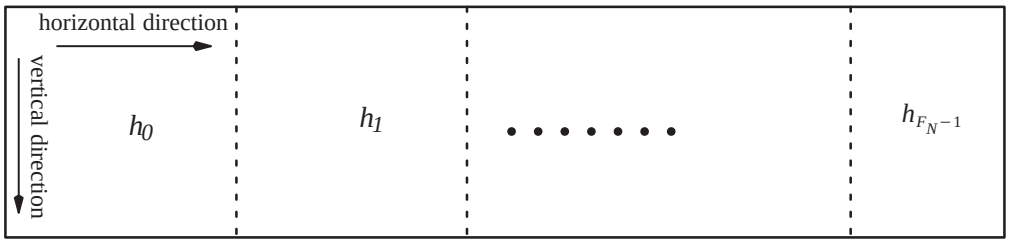


(a) Horizontal–Vertical Filtering

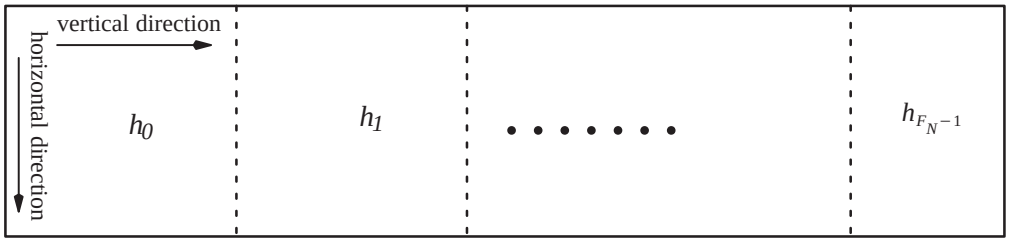


(b) Vertical–Horizontal Filtering

Figure 3–5: Two Implementation Schemes for SV/OSD



(a) Vertical–Horizontal Filtering



(b) Horizontal–Vertical Filtering

Figure 3–6: Construction of Approximation Matrix for Different Filtering Scheme

$$\frac{O(SV/OSD)}{O(Direct)} = \frac{M^2 + F_N M^2}{F_N M^2} = 1 + 1/F_N \quad , \tag{3.33}$$

and is close to 1 for large F_N. However, the convergence and speed of SV/OSD is much faster than linear convergence in most cases as demonstrated in Section 4.1, and sufficient approximation for computer vision applications can be achieved with a much smaller approximation order than M. Thus, SV/OSD satisfies all the implementation criteria. Table 3–2 compares the various schemes discussed in Section 2.2 when the comparison is based on the implementation of multiple filters. The computational advantage of SV/OSD increases as the number of filters in the system increases.

Table 3–2: Comparison Summary of Computational Schemes

	Comp. Complexity	Latency	Storage	Remarks on Hardware
Direct	$F_N N^2 M^2$	NM	NM	
FFT	$F_N(N+M)(N+M_)log N$	$(N+M)(N+M_)log N$	$F_N(N+M)(N+M)$	F_N *FFT* and F_N *IFFT* modules
Parallel Conv	$F_N N^2 M^2$	NM	NM	$F_N M$ 1D convolvers
Separable Approx	$N^2 MP(F_N+1)$	NM	NM	$(F_N+1)P$ 1D convolvers

CHAPTER 4

SV/OSD Performance Evaluation

In this section, the separable approximation method described previously is evaluated using SV/OSD. Gabor filters with various parameters are used for the evaluation. Gabor filters are often used in computer vision applications since they have an optimal spatial–frequency joint resolution and good smoothing effects. The real part of continuous Gabor filters is defined as

$$\mathrm{Re}\{g_b(x,y)\} = g_{br}(x,y) = g_a(\tilde{x},\tilde{y})\cos(\alpha(Ux + Vy)), \tag{4.1}$$

where $g_a(x,y) = e^{-x^2/2\sigma_x^2 - y^2/2\sigma_y^2}$, $\tilde{x} = Ux + Vy$, $\tilde{y} = -Vx + Uy$, $U = \cos\theta$, $V = \sin\theta$, θ is the angle of orientation, α determines the oscillation frequency, and σ_x and σ_y are standard deviations of $g_a(x,y)$. The imaginary part of continuous Gabor filters is defined as

$$\mathrm{Im}\{g_b(x,y)\} = g_{bi}(x,y) = g_a(\tilde{x},\tilde{y})\sin(\alpha(Ux + Vy)) \quad . \tag{4.2}$$

These continuous filters are sampled at a regular rectangular grid to form discrete counterparts.

The size of the filter is chosen such that very little aliasing is introduced after sampling. It is inevitable to have aliasing since Gabor filters are not bandlimited. However, the frequency response of the Gabor filter has an exponential decrease as the frequency moves away from the center frequency of the Gaussian at $(\alpha U, \alpha V)$, and the aliasing can be reduced to a negligible amount by proper sampling. First the sampling is done within the range *[–1<x,y<1]*. The parameters σ_x and σ_y are chosen so that a significant portion of the Gabor function stays in this range for any orientation. The sample size is determined from the prototype filter whose orientation (θ) is zero. The center frequency of the prototype filter lies on the *u* axis at $(\alpha, 0)$. The cut–off frequency of the prototype filter is set at the center frequency plus four times the standard deviation of the Gaussian which is $1/\sigma_x$. The sampling interval T_s is derived from

$$f_c = \frac{\pi}{T_s} = \alpha + \frac{4}{\sigma_x} \quad , \tag{4.3}$$

which implies a sampling size of

$$M = \frac{a_+ - a_-}{T_s} = \frac{2}{T_s} = \frac{2}{\pi}\left(\alpha + \frac{4}{\sigma_x}\right) \tag{4.4}$$

where a_+ and a_- are the sample range along the x axis, which are *1* and *–1* respectively. Note that the sampling size in both the vertical and horizontal directions are set to be equal and the sampling size of the filters is the same for all orientations. This is done for simplicity in performance evaluation. However in a real implementation, it is better to have smaller filters. The filter size can be reduced if the sampling size is determined for each direction independently and for each filter with a different orientation.

The approximation error for each case is measured in terms of a normalized energy error (*L2*). The normalized energy error is the *L*2 norm of the approximation difference normalized by the *L2* norm of the original filter. Thus,

$$L2 = \sum_{n,m} |I(n,m) - Ap(n,m)|^2 \; / \; \sum_{n,m} |I(n,m)|^2 \, . \tag{4.5}$$

4.1.Energy Error Analysis

First, the performance of SV/OSD is evaluated on Gabor filters with a fixed approximation order (*P*), various frequencies (α) and Gaussian aspect ratios (σ_x/σ_y). The result is shown in Figure 4–1 for the real part of Gabor filters and Figure 4–2 for the imaginary part. Each figure contains three 3D plots. The top one is for *P=10*, the middle one for *P=15*, and the bottom one for *P=20*. Each plot shows the maximum *L2* error among F_N filters which constitute an approximation matrix. The approxi-

mation matrix is constructed using 16 filters with the same α and σ_x/σ_y but different orientations which are uniformly distributed in $[0, \pi]$.

Second, the performance of SV/OSD is evaluated on both real and imaginary parts of Gabor filters with various frequencies (α) and Gaussian aspect ratios (σ_x/σ_y). The approximation order is adjusted so that the the ratio of approximation order and filter length (P/M) is fixed. This is to show the performance characteristic in terms of computational savings over the direct implementation. Figures 4–3 and 4–4 show the results of the evaluation for $P/M=0.3$, and Figures 4–5 and 4–6 show the results for $P/M=0.5$. Figures 4–3 and 4–5 are the results of approximating the real part of Gabor filters, whereas Figures 4–4 and 4–6 are the results of approximating the imaginary part of Gabor filters. An approximation matrix is generated by uniformly sampling the filter's orientation within $[0,\pi]$. The evaluation is done with $F_N=4$, 8, and 16. So when $F_N=4$, the approximation matrix consists of Gabor filters with $\theta=0, \pi/4, \pi/2,$ and $3\pi/4$. In each figure, the top chart is for $F_N=4$, the middle chart is for $F_N=8$, and the bottom chart is for $F_N=16$. A small table next to each chart shows the computational saving of using SV/OSD over the direct method. The computational saving is computed by the following formula,

$$\eta \;=\; 1 - \frac{PM + F_N PM}{F_N M^2} \;=\; 1 - \frac{P\,(1 + F_N)}{F_N M} \quad . \tag{4.6}$$

Third, the approximation order is increased gradually, and the convergence behavior of SV/OSD is observed using the real part of Gabor filters. Four filters are used for this test: $(\alpha, \sigma_x, \sigma_y)$ = (10, 0.2, 0.4), (20, 0.2, 0.4), (10, 0.15, 0.45), (20, 0.15, 0.45). The approximation matrix is formulated with 16 filters whose orientations are uniformly distributed in $[0, \pi]$. The maximum energy error among 16 filters is plotted in Figure 4–7. It exhibits much faster convergence than the linear convergence. The convergence speed of SV/OSD is compared with other convergence types, and the result is shown in Figure 4–8. The real part of Gabor filter with $(\alpha, \sigma_x, \sigma_y)$ = (20, 0.15, 0.45) is used for the comparison. As can be seen, the convergence speed of SV/OSD is much faster than any of polynomial convergences, and is very close to the exponential convergence.

Some remarks based on the evaluation results are the following.

1.. The approximation becomes more difficult as the frequency parameter α and the deviation ratio σ_y/σ_x increases.

2.. The approximation becomes more difficult as F_N increases.

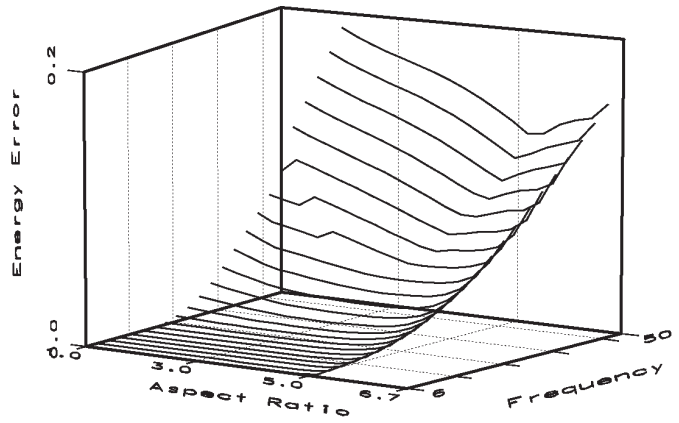
3.. SV/OSD exhibits much better convergence than linear convergence

4.. The approximation results are similar for the real part and the imaginary part.

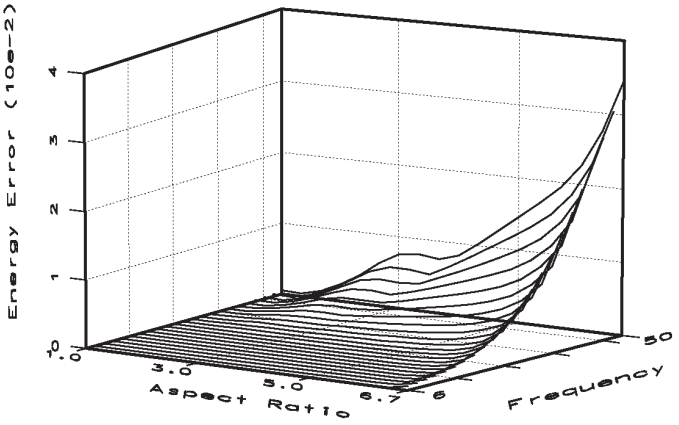
5.. It was observed that all the 1D filters are either symmetric or anti–symmetric since the Gabor filters are also either symmetric or anti–symmetric.

4.2.Frequency Distortion

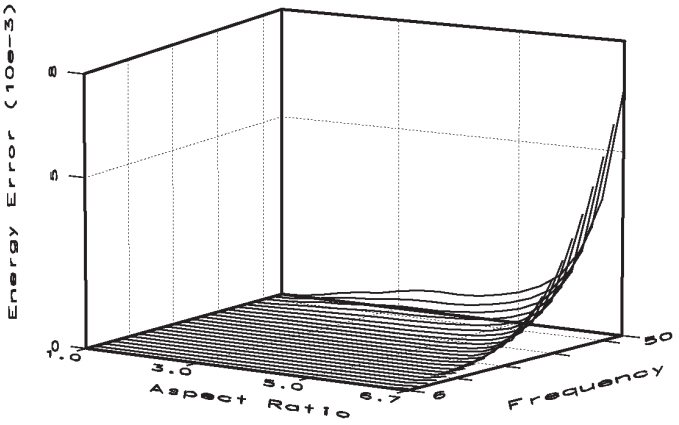
Second, performance is evaluated based on frequency response distortion introduced by the separable approximation method. Figure 4–9 shows the magnitude of the energy error in the frequency domain. The frequency distortion tends to match the frequency response of the filter at any orientation. The distortion energy is concentrated at the point where the energy content of the filter is significant. Thus, the effect of the distortion is insignificant in most cases compared to the effect of the filter. It can be also seen that the distortion energy is concentrated along the V axis. Distortion in the spatial domain is seen along the axis on which the projection filters are implemented, or perpendicular to the axis on which the orthogonal filters are implemented. This is due to the fact that the projection filters can be tailored to match each individual filter, whereas the orthogonal filters are tailored to match the whole set of filters. Thus, errors are dominant along the direction of the projection filters. This directionality of error appears in the distortion along the V axis in the frequency domain.



(a) Approximation Order = 10



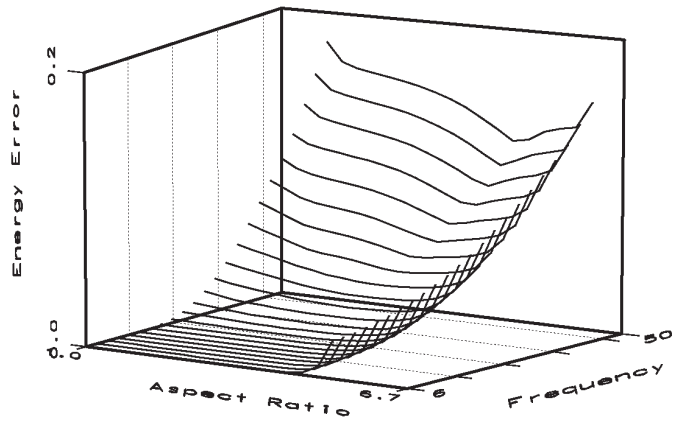
(b) Approximation Order = 15



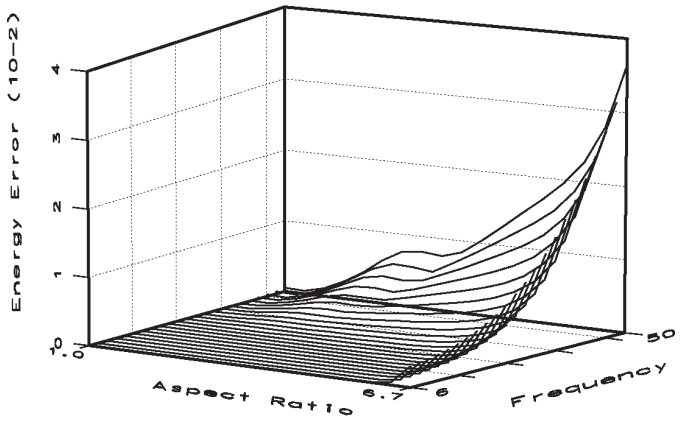
(c) Approximation Order = 20

Figure 4–1: SV/OSD Approximation Results (Test 1)

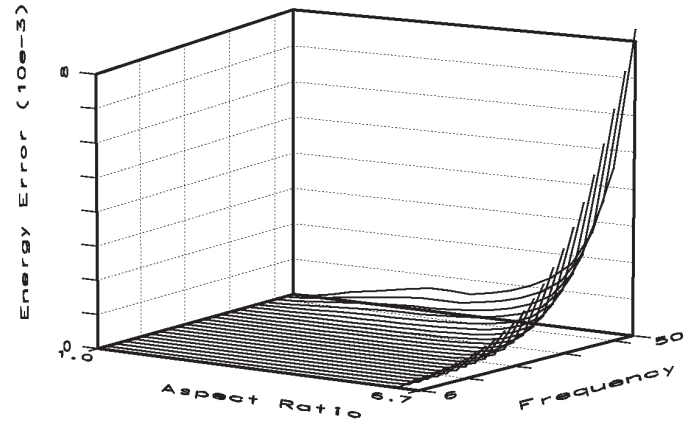
The real part of Gabor filters with various parameters are approximated using SV/OSD. F_N is set to 16, and the approximation order is fixed in each plot to either 15, 10, or 20.



(a) Approximation Order = 10



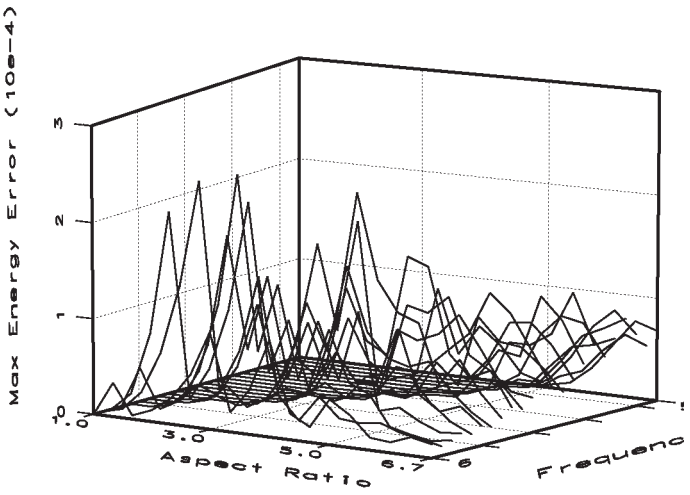
(b) Approximation Order = 15



(c) Approximation Order = 20

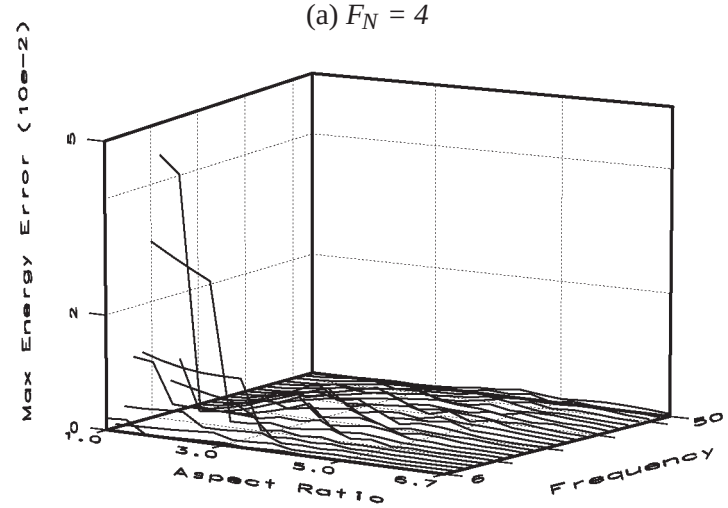
Figure 4–2: SV/OSD Approximation Results (Test 1)

The Imaginary part of Gabor filters with various parameters are approximated using SV/OSD. F_N is set to 16, and the approximation order is fixed in each plot to either 10, 15, or 20.



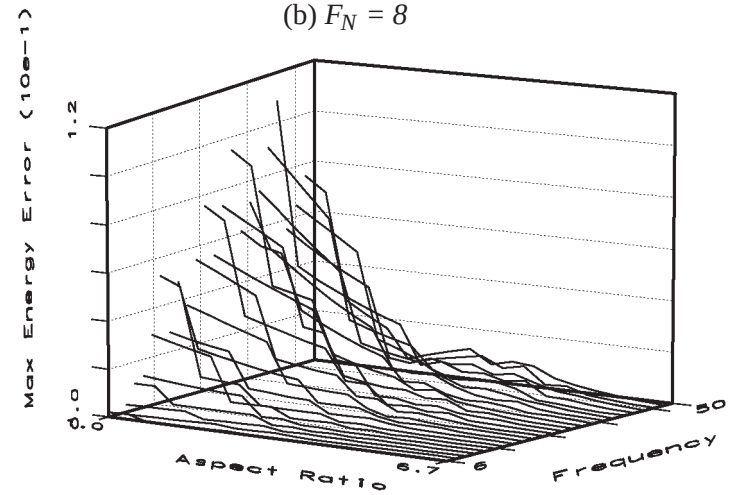
computational saving
62.5 %

(a) $F_N = 4$



computational saving
66.3 %

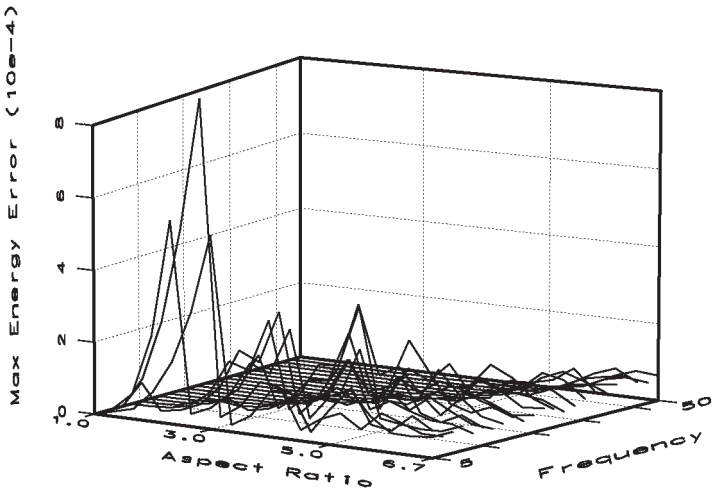
(b) $F_N = 8$



computational saving
68.1 %

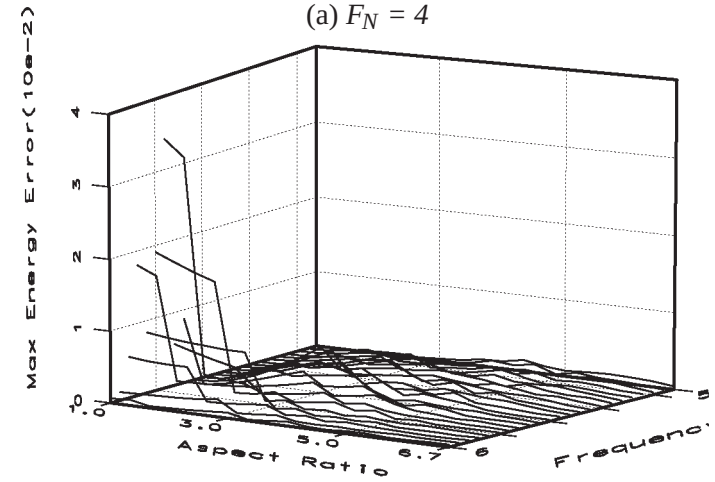
(c) $F_N = 16$

Figure 4–3: SV/OSD Approximation Result (Test 2)
The real part of Gabor filters with various parameters are approximated using SV/OSD. The approximation order is set to *round(0.3M).*



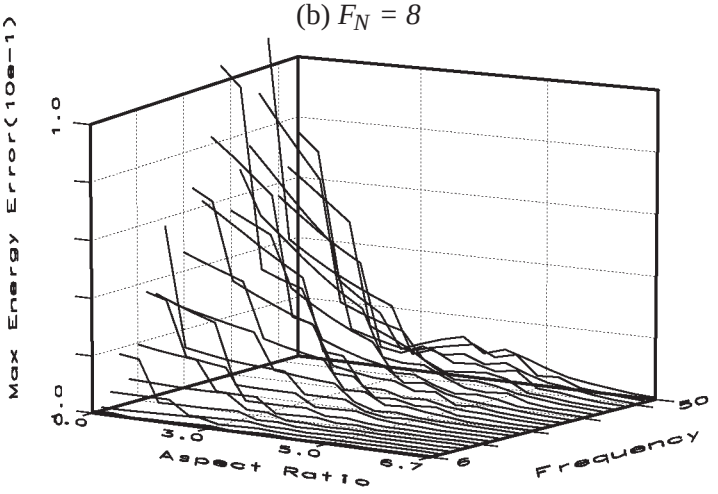
computational saving
37.5 %

(a) $F_N = 4$



computational saving
43.7 %

(b) $F_N = 8$

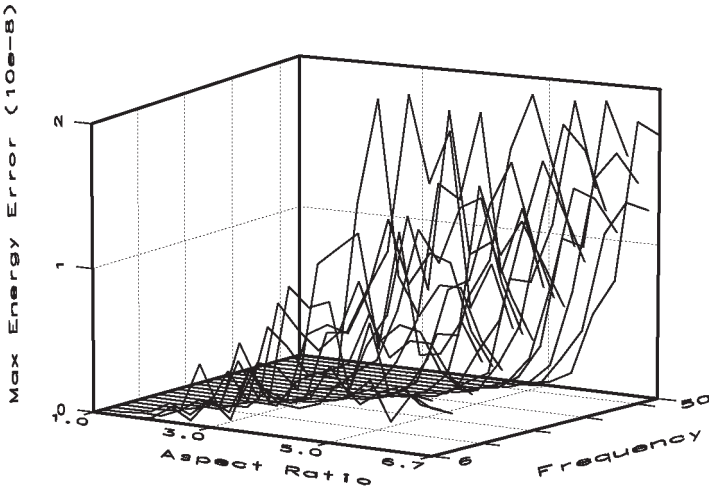


computational saving
46.8 %

(c) $F_N = 16$

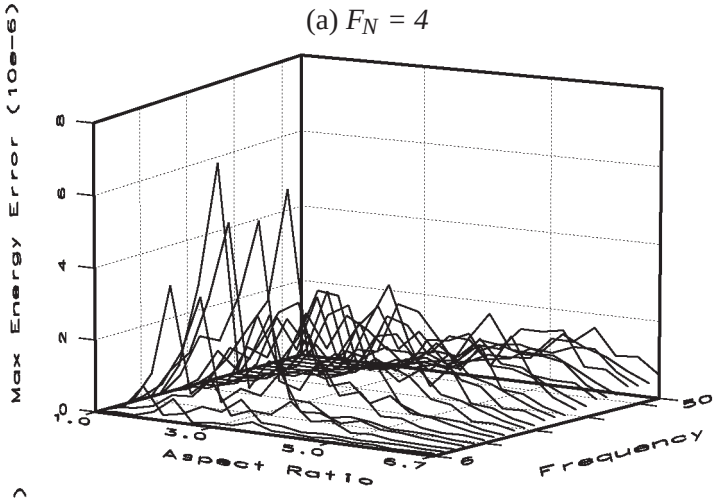
Figure 4–4: SV/OSD Approximation Result (Test 2)

The imaginary part of Gabor filters with various parameters are approximated using SV/OSD. The approximation order is set to *round(0.3M)*.



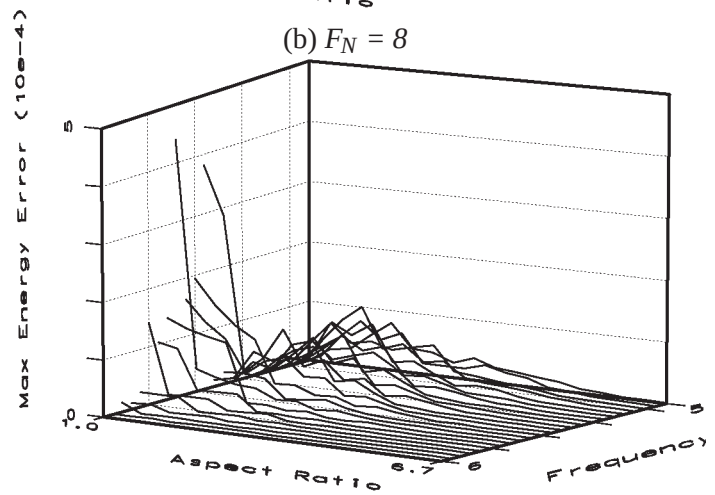
computational saving
37.5 %

(a) $F_N = 4$



computational saving
43.7 %

(b) $F_N = 8$

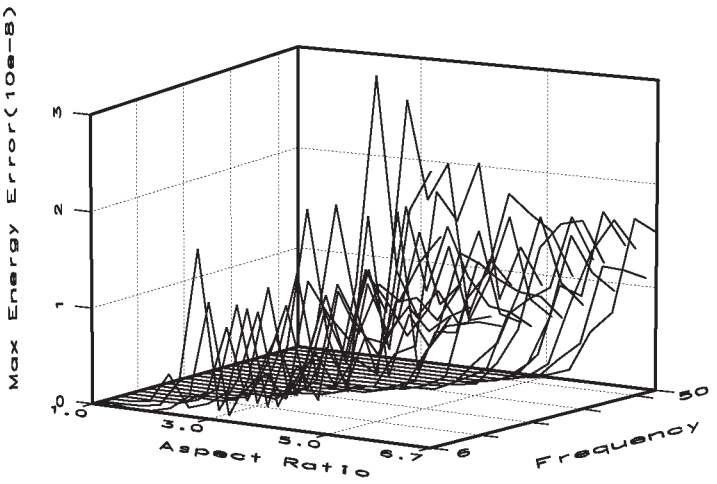


computational saving
46.8 %

(c) $F_N = 16$

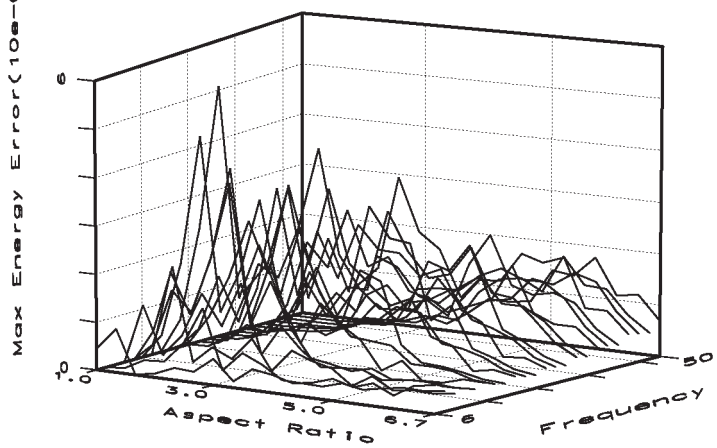
Figure 4–5: SV/OSD Approximation Result (Test 2)

The real part of Gabor filters with various parameters are approximated using SV/OSD. The approximation order is set to *round(0.5M).*



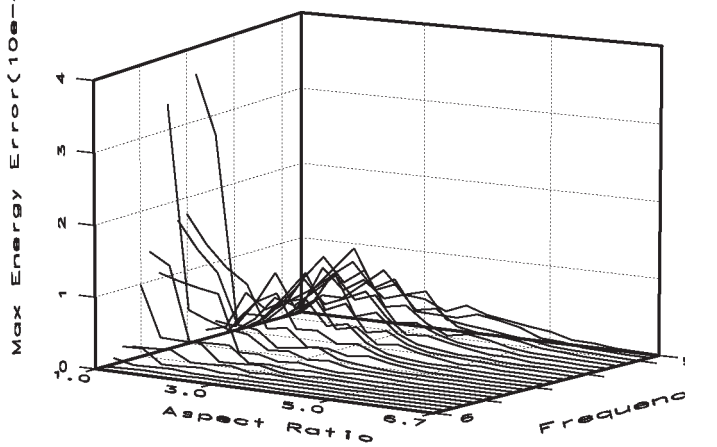
computational saving
37.5 %

(a) $F_N = 4$



computational saving
43.7 %

(b) $F_N = 8$



computational saving
46.8 %

(c) $F_N = 16$

Figure 4–6: SV/OSD Approximation Result (Test 2)

The imaginary part of Gabor filters with various parameters are approximated using SV/OSD. The approximation order is set to *round(0.5M)*.

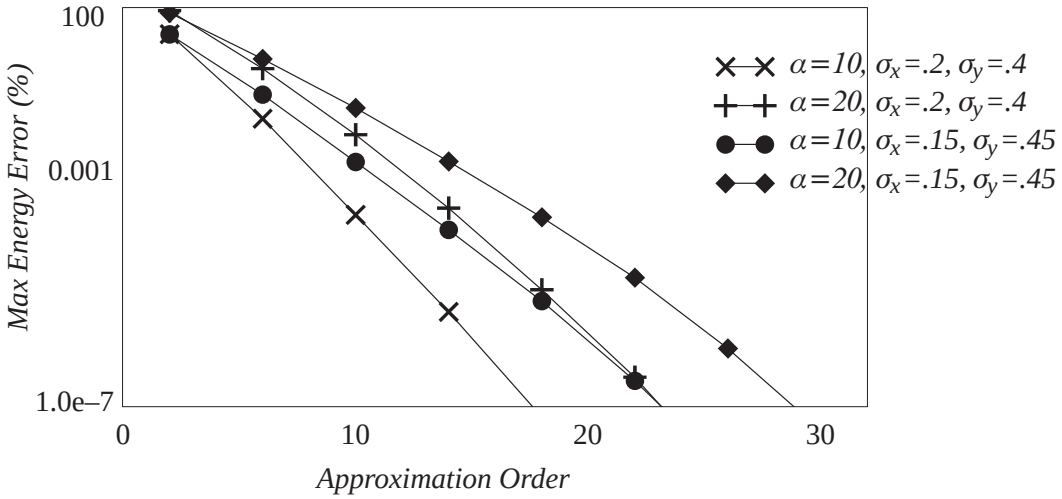


Figure 4–7: Separable Approximation Convergence Behavior I

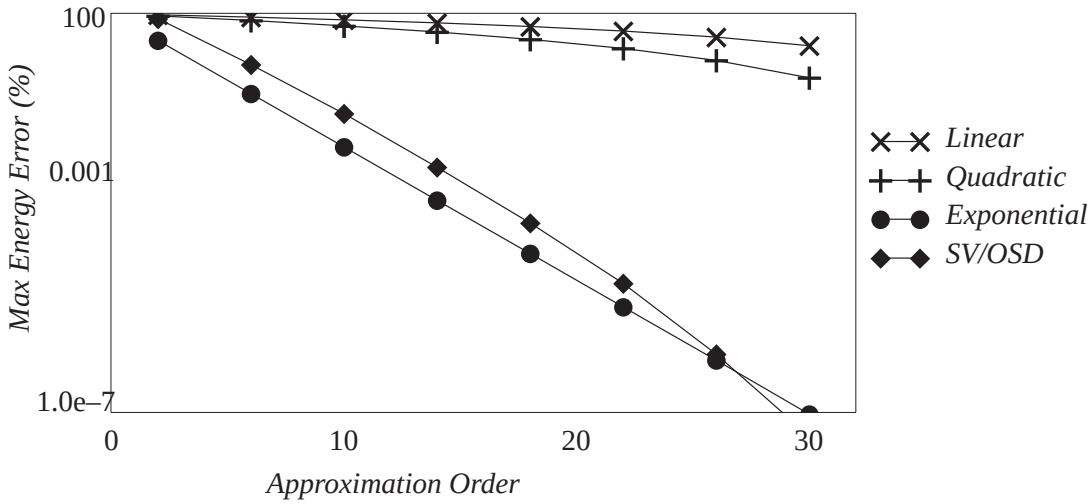


Figure 4–8: Separable Approximation Convergence Behavior II

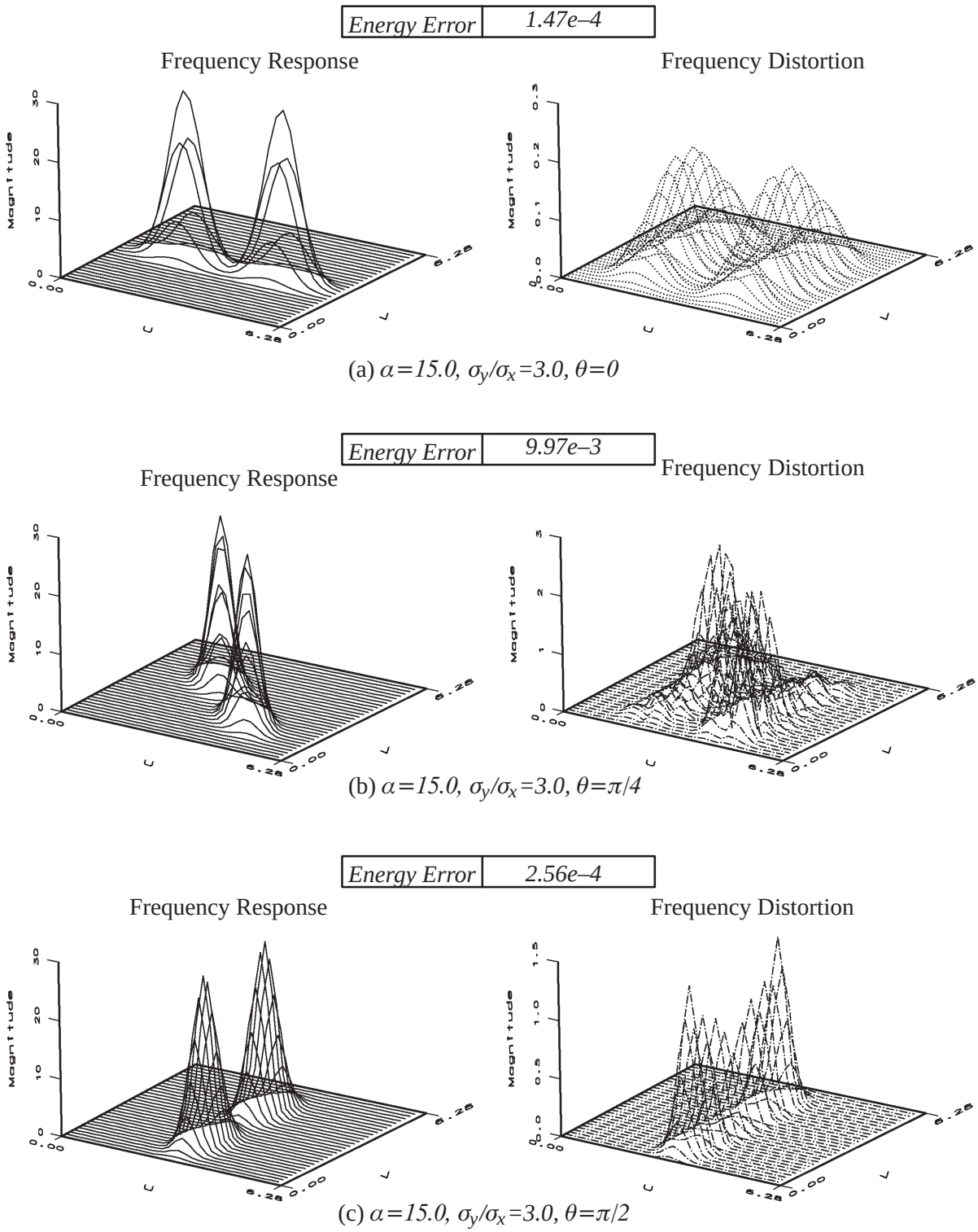


Figure 4–9: Frequency Distortion of Approximation Filters

The real part of Gabor filters are approximated using SV/OSD. The approximation order is 6, and the number of filters in the approximation matrix is 8.

4.3.Applicability to Computer Vision Algorithms

In this section, the performance of approximation filters are evaluated by applying the filters to vision algorithms. The two algorithms selected are edge detection and Jain and Farrokhnia's unsupervised texture segmentation[41].

4.3.1.Edge Detection

A simple edge detector is implemented for this purpose. It consists of orientational filters, zero crossing detectors and thresholding operations. The process of edge detection is shown in Fig. 4–10. For a performance evaluation, one bank of the process is applied to a simple test image shown in Fig. 4–11(a). The image consists of four step edges and Gaussian noise superimposed on the edge. The step edges start at the center of the image and extend to each corner of the image. The Gaussian noise has zero mean and its variance is set to 100, while the edge intensity is set to 100. Gabor filter g_b*:*$(\alpha, \sigma_x, \sigma_y, \pi)$ *= (15.0, 0.2, 0.4,* π*/4)* is used with a mask size of *27x27*, and the lower threshold value set adaptively to 1/2 of the maximum intensity in the intermediate image after the zero crossing detector. The zero crossing detector looks for zero crossings along both horizontal and vertical directions, and outputs the larger gradient at the zero crossing points. Thus, the zero crossing map Z_I of an image I can be expressed as

$$Z_I[m,n] = \begin{cases} \text{if } I[m,n]I[m-1,n] < 0 \ \textit{or} \ I[m,n]I[m,n-1] < 0 \ \textit{then} \\ \qquad \max(\, ABS(I[m,n] - I[m-1,n]),\ ABS(I[m,n] - I[m,n-1])\,) \\ \textit{else} \quad 0 \end{cases} . \tag{4.7}$$

Since the Gabor filter is oriented at $\pi/4$, the edge detector should extract only the edges oriented at $\pi/4$ (the diagonal edge from the left lower corner to the right upper corner) and ignore the other diagonal edge (the one from the left upper corner to the right lower corner). The performance is evaluated in terms of the number of pixels which are detected as a part of an edge but should not be detected as edge pixels, plus the number of pixels which are not detected as a part of an edge but should be detected as edge pixels. These pixels are called spurious edge pixels. Fig. 4–11(b) shows the correct edge map. The result of the edge detection using the Gabor filter is shown in Fig. 4–11(c). It only misses the point where all the four edges meet. Thus, the number of spurious edge pixels is 1.

The same edge detection is performed using approximation filters. SVD and SV/OSD are used to approximate the Gabor filter g_b*:*$(\alpha, \sigma_x, \sigma_y, \pi)$ *= (15.0, 0.2, 0.4,* π*/4)*. In the SV/OSD, the approximation matrix is formed through 8 filters which are a prototype filter g_b*:*$(\alpha, \sigma_x, \sigma_y)$ *= (15.0, 0.2, 0.4)* oriented at *0,* $\pi/8$*,* $\pi/4$*,* $3\pi/8$*,* $\pi/2$*,* $5\pi/8$*,* $3\pi/4$ and $7\pi/8$. The results of the evaluation are shown in Table Table 4–1. The right two columns show the errors of the approximations. The quantities shown in these two columns are normalized peak error (*PE*), normalized absolute sum error (*L1*) and normalized energy error (*L2*). *PE* and *L1* are defined as

$$PE = \max_{n,m} |I(n,m) - Ap(n,m)| / \max_{n,m} |I(n,m)| \text{ , and} \tag{4.8}$$

$$L1 = \sum_{n,m} |I(n,m) - Ap(n,m)| \ / \ \sum_{n,m} |I(n,m)| \quad , \tag{4.9}$$

where *I(n,m)* is the original filter and *Ap(n,m)* is the approximation filter. *L2* is defined in (4.5).

The number of spurious edge pixels is identical to the Gabor filter when $P \geq 4$ for SVD and $P \geq 5$ for SV/OSD. As the order of approximation decreases, the approximation filter loses its orientational selectivity and the filter cannot distinguish

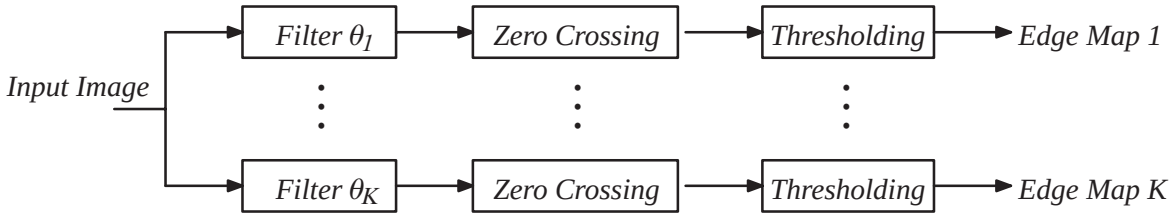


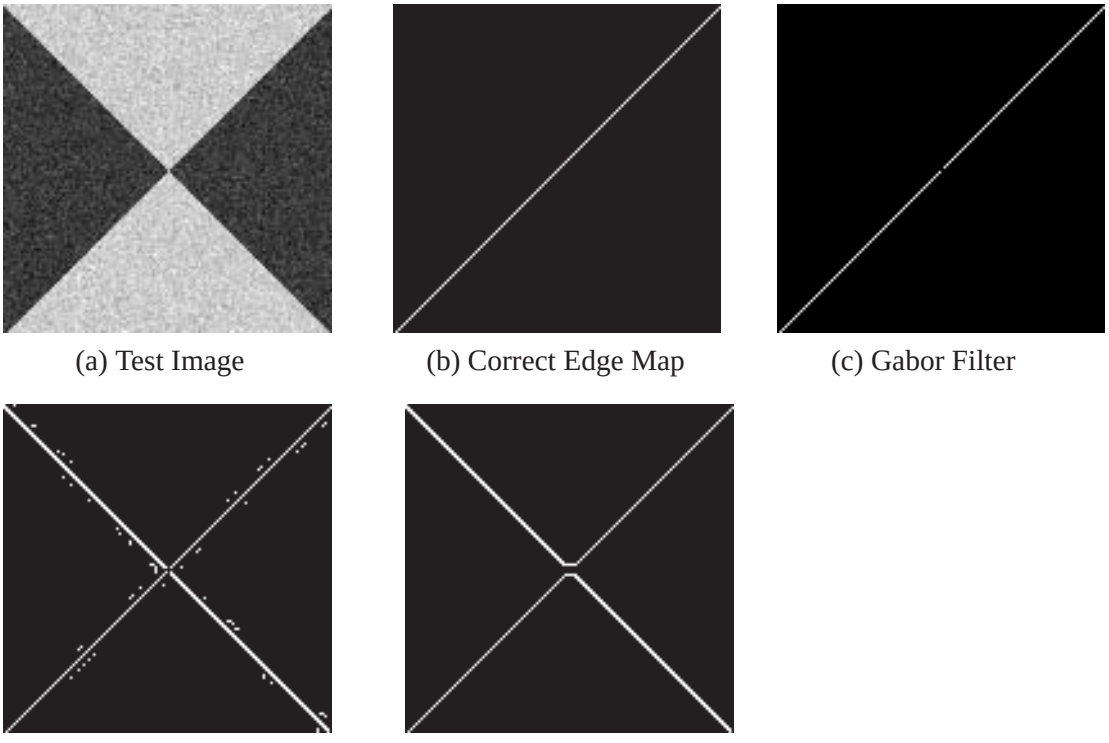
Figure 4–10: A Simple Edge Detector Using Orientational Filters

edges oriented in different directions. It also loses its smoothness as the approximation order decreases, and picks up the Gaussian noise edges. In terms of the approximation error, the edge detector shows no degradation until the peak error, the absolute sum error and the energy error reach 18%, 29% and 5.6% respectively. Fig. 4–11(d) and Fig. 4–11(e) show the edge maps resulting from these approximation filters.

Table 4–1: Edge Detection Results Using Approximated Filters

Approximation Order	# of Spurious Edge Pixels		Approximation Errors (*PE*, *L1*, *L2*)	
Gabor	1		0, 0, 0	
	SVD	SV/OSD	SVD	SV/OSD
P=8	1	1	0.631, 1.45, 6.55e–3	2.70, 7.29, 0.177
P=7	1	1	1.47, 2.90, 0.0290	3.91, 10.1, 0.330
P=6	1	1	2.95, 5.26, 0.107	8.45, 21.2, 1.66
P=5	1	1	6.27, 9.70, 0.451	12.1, 27.9, 30.8
P=4	1	6	9.23, 16.1, 1.29	18.6, 38.5, 6.25
P=3	37	8	23.2, 29.1, 5.56	26.5, 44.4, 13.6
P=2	150	14	28.1, 42.6, 11.6	41.3, 55.7, 22.8
P=1	309	256	74.3, 75.4, 54.8	73.6, 84.2, 64.8

4.3.2.Jain and Farrokhnia's Texture Segmentation



(a) Test Image (b) Correct Edge Map (c) Gabor Filter

(d) SVD: Order=1 (e) SV/OSD: Order=1

Figure 4–11: Test Image and the Segmentation Result

This unsupervised texture segmentation algorithm consists of four phases; orientational filters using Gabor filters, non–linear transform, local energy computation, and clustering[41]. The structure of the algorithm is shown in Figure 2–13. A hyperbolic tangent which resembles the sigmoidal activation function used in neural networks is used for the non–linear transform. It is described by

$$N_T(x,y) = \tanh(\alpha_n\, p(x,y)) = \frac{1 - e^{-2\alpha_n p}}{1 + e^{-2\alpha_n p}} \quad , \tag{4.10}$$

where p is the pixel value of a filtered image, and the parameter α_n is set to 0.25.

The local energy computation is an absolute sum within a local window, and is described as

$$E_T(x,y) = \sum_{(i,j)\in W_T} |N_T(x-i, y-j)| \quad , \tag{4.11}$$

where W_T is a local window.

The Isodata clustering algorithm [90] is used for the clustering.

Fig. 4–12(a) shows the test image. It consists of two synthetic textures; the background texture is a repetitive bar pattern oriented at *π/4*, and the foreground texture within a circle is the same pattern oriented at *3π/4*. Four orientational filters are used to process this image. They contain both real and imaginary parts of Gabor filters g_b*:(α, σ$_x$, σ$_y$, π) = (10.0, 0.2, 0.4, π/4)* and *(10.0, 0.2, 0.4, 3π/4)*. The window size for the local energy computation is 7x7. The segmentation result is shown in Fig. 4–12(b), and misclassified pixels are shown in white in Fig. 4–12(c). The number of misclassified pixels in Fig. 4–12(c) is 90.

The same segmentation is performed by replacing the Gabor filters with SV/OSD approximation filters. The performance is evaluated using the number of misclassified pixels. The results are shown in Table 4–2. For order 5 or greater, the results using approximation filters are identical to the result using Gabor filters. As the order of the approximation decreases below order 5, the number of misclassified pixels increases. Table Table 4–2 also shows approximation error for peak error (*PE*), absolute sum error (*L1*) and energy error (*L2)* used for the segmentation. Based on the approximation errors, texture segmentation produces the same result until the errors reach 4.8%, 10% and 0.4% respectively.

Table 4–2: Texture Segmentation Results Using Approximated Filters

Approximation Order	# of Mislabeled Pixels		Approximation Errors (*PE*, *L1*, *L2*)	
	SVD	SV/OSD	SVD	SV/OSD
P=8	90	90	0.189, 0.565, 6.55e–4	0.189, 0.565, 6.55e–4
P=7	90	90	1.03, 1.81, 9.44e–3	1.02, 1.81, 9.44e–3
P=6	90	90	1.05, 2.66, 0.0182	1.05, 2.66, 0.0182
P=5	90	90	4.74, 6.98, 0.221	5.16, 7.67, 0.221
P=4	91	92	4.82, 10.7, 0.424	4.82, 10.7, 0.424
P=3	111	116	23.8, 28.1, 4.01	23.2, 27.9, 4.01

(a) Test Image (b) Segmentation Result (c) Misclassified Pixels(90)

Figure 4–12: Test Image and the Segmentation Result

P=2	433	112	21.4, 38.3, 7.60	21.4, 38.3, 7.60
P=1	6716	6495	108, 79.6, 53.8	99.4, 79.8, 53.8

CHAPTER 5

Multi–resolution Decomposition

Multi–resolution image decomposition (MRD) produces multiple output images. Each output represents the contents of an input image over a certain frequency region. The output image corresponding to a lower frequency region has a lower resolution and can be decimated (decimated MRD). Hence, the multiple output images have a *pyramid* structure wherein the lowest frequency plane is the smallest and the size increases as the frequency band associated with the image increases. This multi–resolution image decomposition is suitable for an image analysis platform for the following reasons.

- The objects to be recognized often have very different sizes. Hence it is impossible to define the optimal resolution for all the objects.
- The objects can be recognized easily if the context of the image is known. For example, if a house is recognized first, then it is easy to find a window as a rectangle inside the house. But it is more difficult to recognize the window if the house is not recognized first. Using the multi–resolution technique, it is easy to first process the coarse image to understand the context of the original image and then move to finer images for further processing. (coarse–to–fine processing)
- Coarse–to–fine processing can speed up processing since the coarse information can be represented by fewer samples. The finer details require more samples, but the prior information derived from the context, constrains the region of observation. Moreover, if an object can be recognized from a coarse description, then processing finer details is not needed.
- Decomposing an image into different frequency bands is useful in analyzing the image. For example, an edge consists of higher frequencies while most texture information has its energy concentrated in narrow frequency bands.

The rest of this chapter discusses computation applied to MRD. The discussion is based on wavelets because the time–frequency characteristic of the wavelet transform is suitable for image analysis, and recent research on MRD has been discussed in the context of the wavelet transform[55][21][45]. More specifically, MRD is described in terms of Wavelet Series (WS) defined by

$$d^k_{m,n} = \sum_{i,j} f[i,j]\, 2^{-k+1}\psi(2^{-k+1}i - m, 2^{-k+1}j - n) = \sum_{i,j} f[i,j]\, \psi^{k-1}(i - 2^{k-1}m, j - 2^{k-1}n) \quad (5.1)$$

where $d^k_{m,n}$ is the k^{th} decomposition output at location (m,n), $f[i,j]$ is the input signal, $\psi(x)$ is the orientational filter, and $\psi^k(x) = 2^{-k/2}\psi(2^{-k}x)$. It is assumed that the sampling period of $f[m,n]$ is 1 in both directions for simplicity. For general sampling, $\psi^k(m,n)$ has to be replaced by $\psi^k(mT_x, nT_y)$ where T_x and T_y are sampling periods in the x direction and y directions respectively.

Note that Equation (5.1) describes decimated MRD. The formula for undecimated MRD is

$$d^k_{m,n} = \sum_{i,j} f[i,j]\, 2^{-k+1}\psi(2^{-k+1}i - 2^{-k+1}m, 2^{-k+1}j - 2^{-k+1}n) \quad . \quad (5.2)$$

The decimated MRD involves decimation by 2 in both directions. Thus, the size of the image decreases by 2x2 from one level of the decomposition to the next level. The first level decomposition requires N^2 convolutions with each convolution having a complexity $O(M^2)$. Hence the total complexity of the first level decomposition is $O(N^2M^2)$. Similarly the k^{th} level of decomposition requires $N^2/4^{k-1}$ convolutions. Each convolution at this level has a complexity $O(4^{k-1}M^2)$ due to the dilation of the filter. Hence the total complexity of the k^{th} level decomposition is $O(N^2M^2)$. Now if the decomposition is performed up to the L^{th} level, the complexity of the whole decomposition is $O(LN^2M^2)$. The process of decomposition is shown in Figure 5–1. It seems inevitable that the amount of computation will increase linearly as the level of decomposition increases. However, the discrete wavelet transform (DWT) performs MRD in a recursive fashion. The k^{th} level decomposition is performed on the $k–1^{\text{th}}$ decomposition using the same filter kernel. This recursive scheme is possible when the wavelets are orthogonal to each other. With DWT, the decomposition can be done in $O(N^2M^2)$ and is independent of L. Then at the k^{th} level of decomposition, the complexity is $O(N^2M^2/4^{k-1})$. Thus, the total amount of the computation is

$$\sum_{k=1}^{L} N^2M^2/4^{k-1} \le \frac{4}{3}N^2M^2 \ . \tag{5.3}$$

Recently, Shensa and Rioul developed independently an algorithm which approximates 1D continuous wavelets by their samples and an interpolation function in such a way that the DWT can be applied using the approximated wavelets even though the wavelets are not orthogonal[74][80]. This approximation algorithm will be called the *Wavelet Approximation* in this thesis. The Wavelet Approximation can be extended to 2D continuous wavelets by first decomposing the wavelets using Separable Approximation and applying Wavelet Approximation to each 1D filter separately. This new approximation algorithm is called the *Separable Wavelet approximation* (SWA)[47].

For undecimated MRD, the size of the image stays the same, whereas the size of the filter increases by 2x2. Thus the amount of computation increases exponentially as the decomposition level increases. For the level L MRD, the total computation is

$$\sum_{k=1}^{L} 4^{k-1}N^2M^2 \ = \ (4^L - 1)N^2M^2/3 \ . \tag{5.4}$$

Thus, the computational complexity of undecimated MRD is $O(4^LN^2M^2)$.

First, this chapter describes the Discrete Wavelet Transform (DWT) which computes a decimated 1D MRD efficiently. Second, it introduces the Wavelet Approximation. Third, it introduces SWA. Fourth, it discusses implementation issues for undecimated MRD, and suggests an efficient computation scheme based on SWA.

5.1.Discrete Wavelet Transform

Let V_0 define a vector space which includes all the functions of interest. Assume $\{\phi_s(x - i)\}_{n\in\mathbf{Z}}$ is an orthonormal basis of V_0, and satisfies a dilation relation,

$$\phi_s(x) \ = \ \sqrt{2} \sum_{i=-\infty}^{\infty} \tilde{h}[i]\, \phi_s(2x - i) \ . \tag{5.5}$$

Note that the coefficients $\tilde{h}[i]$ in the dilation equation are different from the orientational filters $h[m,n]$. Since $\{\phi_s(x - i)\}_{i\in\mathbf{Z}}$ is an orthonormal basis, any $f(x) \in V_0$ can be expressed as,

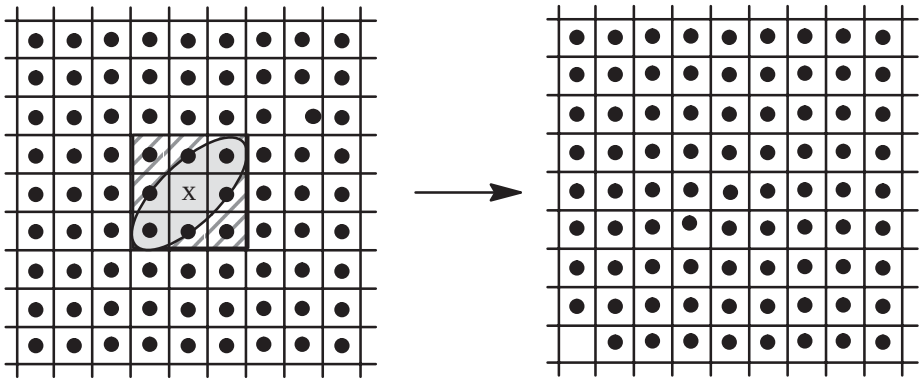
$$f(x) = \sum_{i=-\infty}^{\infty} c_i\, \phi_s(x - i) \quad \text{with} \tag{5.6}$$

$$c_m \ = \ \int f(x)\phi_s^*(x - m)dx \ . \tag{5.7}$$

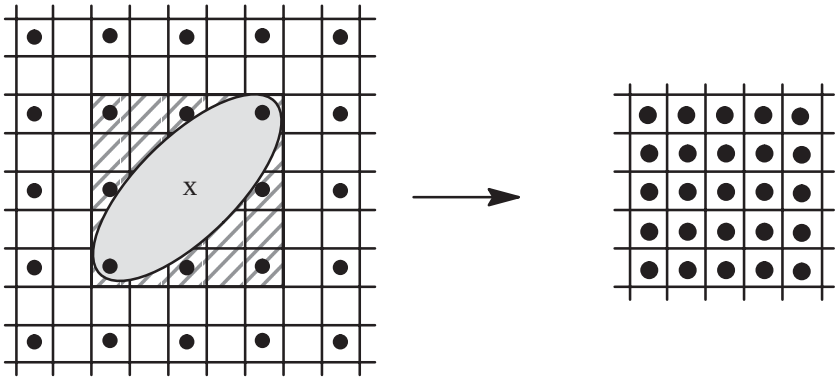
Next define the successive sub–spaces, V_1, V_2, ..., V_k, ... where V_k is a vector space spanned by an orthonormal set $\left\{\sqrt{2^{-k}}\phi_s(2^{-k}x - i)\right\}_{i\in\mathbf{Z}}$. In the literature, $\phi_s(x)$ is often called a *scaling function*[75]. Because of the dilation relation (5.5), the following relation holds;

$$V_0 \supset V_1 \supset V_2 \supset V_3.... \ . \tag{5.8}$$

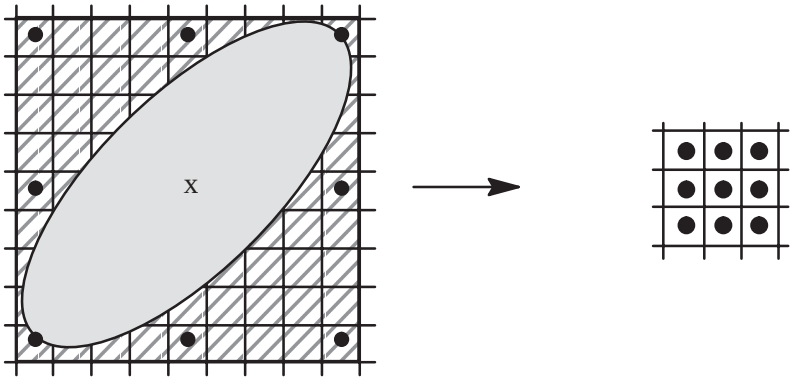
Define another set of vector spaces $\left\{W_k\right\}_{k\in\mathbf{Z}\ k\ge 0}$ as an orthogonal complement of V_k in V_{k-1}. This implies that



(a) First Level



(b) Second Level



(c) Third Level

Orientational Filter Mask

● Sample Point

Figure 5–1: Direct Computation of Decimated MRD

$$V_k \bigcap W_k = 0 \text{ and } V_k \bigcup W_k = V_{k-1} \ . \tag{5.9}$$

Then there exists an orthonormal basis $\left\{\sqrt{2^{-k}}\psi_a(2^{-k}x - n)\right\}_{n\in\mathbf{Z}}$ which spans W_k, where

$$\psi_a(x) = \sqrt{2}\sum_{i=-\infty}^{\infty}(-1)^{-i+1}\tilde{h}[-i+1]\phi_s(2x-i) = \sqrt{2}\sum_i \tilde{g}[i]\,\phi_s(2x-i) \ , \text{ and} \tag{5.10}$$

$$\tilde{g}[i] = (-1)^{-i+1}\tilde{h}[-i+1] \tag{5.11}$$

with $\tilde{h}[n]$ given by (5.5). The set $\left\{\sqrt{2^{-k}}\psi_a(2^{-k}x - n)\right\}_{n,k\in\mathbf{Z}}$ is called an *orthogonal wavelet*[21]. Due to (5.8) and (5.9), $\left\{W_k\right\}_{k\in\mathbf{Z}\ k\geq 0}$ are orthogonal to each other,

$$W_1 \perp W_2 \perp W_3 ... \perp W_k \perp ... \ . \tag{5.12}$$

V_0 can be decomposed into $\left\{W_l\right\}_{l\in\mathbf{Z}\ l\geq 1}$ where

$$V_0 = W_1 \bigcup W_2 \bigcup W_3 \bigcup ... \bigcup W_k \bigcup ... \ . \tag{5.13}$$

Thus, any function $f(x) \in V_0$ can be uniquely described by $\left\{\sqrt{2^{-k}}\psi_a(2^{-k}x - n)\right\}_{n,k\in\mathbf{Z},k\geq 1}$. The set of wavelet coefficients $\{d^k_m\}$ where

$$d^k_m = \int f(x)\sqrt{2^{-k}}\psi_a(m - 2^{-k}x)dx \tag{5.14}$$

is called a *wavelet series*[75]. Note that the wavelet series is critically sampled.

Although the decomposition can be done directly to each subspace W_k by (5.14), a more efficient way is to use recursion. The first decomposition divides V_o into two sub–spaces, V_1 and W_1. The second decomposition divides V_1 into two sub–spaces, V_2 and W_2. The kth decomposition divides V_{k-1} into V_k and W_k. If an input is given as a continuous function ($f(x) \in V_0$), it is mapped to a discrete sequence c^0_m by

$$c^0_m = \int f(x)\phi_s(x - m)dx \ . \tag{5.15}$$

If the input is given in discrete form (*f[m]*), it can be viewed as a set of projection coefficients associated with a function $f(x) \in V_0$ by (5.6), or *f[m]* is viewed as c^0_m. Now the DWT can be performed as follows. First the projection of $f(x) \in V_0$ onto V_1 and W_1 gives

$$c^1_m = \int f(x)\sqrt{2^{-1}}\phi_s(2^{-1}x - m)dx = \int f(x)\sum_i \tilde{h}[i]\phi_s(x - 2m - i)dx$$

$$= \sum_i \tilde{h}[i - 2m]\int f(x)\phi_s(x - i)dx = \sum_i \tilde{h}[i - 2m]\,c^0_i \ , \text{ and} \tag{5.16}$$

$$d^1_m = \int f(x)\sqrt{2^{-1}}\psi_a(2^{-1}x - m)dx = \int f(x)\sum_i \tilde{g}[i]\phi_s(x - 2m - i)dx$$

$$= \sum_i \tilde{g}[i - 2m]\int f(x)\phi_s(x - i)dx = \sum_i \tilde{g}[i - 2m]\,c^0_i \ . \tag{5.17}$$

The decomposition is merely a filtering of c^0 by the lowpass filter $\tilde{h}[i]$ for V_1 and by the high pass filter $\tilde{g}[i]$ for W_1. Similarly the k^{th} level decomposition gives

$$c^k_m = \int f(x)\sqrt{2^{-k}}\phi_s(2^{-k}x - m)dx = \int f(x)\sum_i \tilde{h}[i]\sqrt{2^{-k+1}}\phi_s(2^{-k+1}x - 2m - i)dx$$

$$= \sum_i \tilde{h}[i - 2m] \int f(x)\sqrt{2^{-k+1}}\phi_s(2^{-k+1}x - i)dx = \sum_i \tilde{h}[i - 2m]\, c_i^{k-1} \text{ , and} \tag{5.18}$$

$$d_m^k = \int f(x)\sqrt{2^{-k}}\psi_1(2^{-k}x - m)dx = \int f(x)\sum_i \tilde{g}[i]\sqrt{2^{-k+1}}\phi_s(2^{-k+1}x - 2m - i)dx$$

$$= \sum_i \tilde{g}[i - 2m] \int f(x)\sqrt{2^{-k+1}}\phi_s(2^{-k+1}x - i)dx = \sum_i \tilde{g}[i - 2m]\, c_i^{k-1} \text{ .} \tag{5.19}$$

Note that all the computation is done in the discrete domain. The above equations imply that the filter outputs are decimated by a factor of 2 at each decomposition due to the fact that the filters are applied at location *2m* rather than *m*. The structure of the DWT is shown in Figure 5–2. This is an iterative 2–channel filter bank structure with octave splitting done on only the low–pass portion. This is a critically sampled system (the sample size after decomposition is the same as the input size), and the original sequence can be perfectly reconstructed after the decomposition because of the orthogonality of the wavelets.

Due to the decimation by 2 at each stage of the decomposition, the computation complexity at the k^{th} level is 1/2 of the *k–1*$^{\text{th}}$ level decomposition. Thus, the total computation complexity of the L^{th} level multi–resolution decomposition is

$$\sum_{k=1}^{L} NM/2^k = NM\left(1 - \frac{1}{2^L}\right) < NM \text{ .} \tag{5.20}$$

5.2.Wavelet Approximation

This algorithm allows the wavelet series to be computed in a recursive fashion, the same way as the DWT even though the wavelets are not orthonormal to each other. The computational saving results from decomposing a wavelet filter into smaller filters for a cascaded implementation[26], and moving a decimation operation prior to the FIR operation.

A mother function of dyadic wavelets is denoted as $\psi_b(x)$. The continuous wavelets are approximated by

$$\psi_b(x) \approx \sum_i \tilde{g}[i]\,\phi_I(x - i) \text{ .} \tag{5.21}$$

The original form of this approximation uses $\phi_I(2x)$ rather than $\phi_I(x)$[74]. The modification of the approximation is to keep the first level decomposition shift invariant. Dilation on both sides of the equation results in

$$\psi_b(2^{-k}x) \approx \sum_i \tilde{g}[i]\,\phi_I(2^{-k}x - i) \text{ , or} \tag{5.22}$$

$$\psi_b^k(x) \approx \sum_i \tilde{g}[i]\phi_I^k(x - 2^k i) \quad \text{where} \tag{5.23}$$

$$\phi_I^k(x) = \sqrt{2^{-k}}\phi_I(2^{-k}x) \quad \text{and} \tag{5.24}$$

$$\psi_b^k(x) = \sqrt{2^{-k}}\psi_b(2^{-k}x) \text{ .} \tag{5.25}$$

Equation (5.23) shows that as the wavelet is dilated, the discrete filter $\tilde{g}[m]$ also expands with zeros being padded between each filter coefficient. This suggests that a dilation operation at each level of decomposition can be moved prior to the discrete filter $\tilde{g}[m]$ due to an operator identity

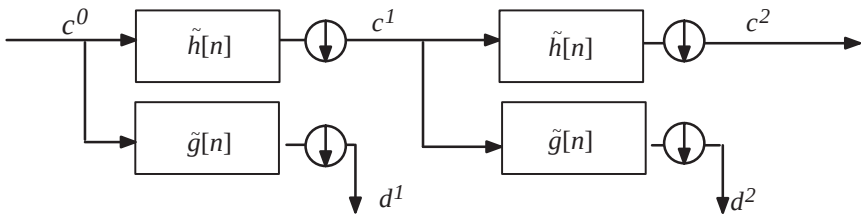


Figure 5–2: Computational Structure of DWT

$$D^K G(z^K) = G(z) D^K \tag{5.26}$$

where D^K is decimation by K, and $G(z)$ is a z–transformed representation of a discrete filter. It also shows that the wavelets $\psi_b^k(x)$ can be implemented in a cascaded fashion with two filters, $\tilde{g}[m]$ and $\phi_I^k(x)$.

The mother wavelet decays as $t \rightarrow \pm\ \infty$ and can be truncated at some time point in the above approximation. The sequence $\tilde{g}[m]$ is derived in such a way that the approximation (5.21) is exact at integer time points within the truncation points.

$\phi_I(x)$ is some smooth interpolation function, and satisfies the dyadic equation,

$$\phi_I(x) = \sqrt{2} \sum_i \tilde{h}[i]\, \phi_I(2x - i)\ , \text{ or} \tag{5.27}$$

$$\phi_I^k(x) = \sum_i \tilde{h}[i] \phi_I^{k-1}(x - 2^{k-1} i)\ . \tag{5.28}$$

Equation (5.27) implies that the interpolation operation involved in (5.21) can be implemented using the interpolation filter, $\phi_I(x)$ and a sequence of small discrete filters, $\tilde{h}[m]$.

Once the approximation of (5.21) is obtained, the wavelet decomposition for $k \geq 1$ can be recursively performed as,

$$\begin{aligned} d_m^k &= \sum_i f[i] \psi_b^{k-1}(i - 2^{k-1} m) \\ &= \sum_i f[i] \sqrt{2^{-k+1}}\, \psi_b(2^{-k+1} i - m) \approx \sum_i f[i] \sqrt{2^{-k+1}} \sum_j \tilde{g}[j] \phi_I(2^{-k+1} i - m - j) \\ &= \sum_j \tilde{g}[j - m] \sum_i f[i]\, \phi_I^{k-1}(i - 2^k j) = \sum_j \tilde{g}[j - m] c_j^{k-1}\ , \text{ where} \end{aligned} \tag{5.29}$$

$$\begin{aligned} c_m^k &= \sum_i f[i] \phi_I^k(i - 2^k m) = \sqrt{2^{-k}} \sum_i f[i] \phi_I(2^{-k} - m - i) = \sum_i f[i] \sqrt{2^{-k+1}} \sum_j \tilde{h}[j] \phi_I(2^{-k+1} i - 2m - j) \\ &= \sum_j \tilde{h}[j - 2m] \sum_i f[i]\, \phi_I^{k-1}(2^{k-1} j - i) = \sum_j \tilde{h}[j - 2m]\, c_j^{k-1}\ , \text{ with} \end{aligned} \tag{5.30}$$

$$c_m^0 = \sum_i f[i]\, \phi_I(i - m)\ . \tag{5.31}$$

After the prefiltering stage (Equation (5.31)), the subsequent decomposition is performed recursively on c^k in the same way as DWT. Note that the low pass sequence c^k is decimated by 2 at each decomposition, however, the high pass sequence d^k is not decimated. Thus the decomposition is not critically sampled.

A basic spline function is often used as the interpolation function, $\phi_I(x)$. With the basic spline of order k, the discrete filter $\tilde{h}[n]$ becomes

$$\tilde{h}[n] = 2^{-k-1/2} \binom{k+1}{n}\ . \tag{5.32}$$

Appendix B gives a derivation of the filter $\tilde{h}[m]$ when a basic spline function is employed as the interpolation function $\phi_I(x)$.

There are no constraints on the mother wavelet, $\psi_b(x)$. The algorithm works for any function as long as a sufficient approximation of the wavelet is done with (5.21). The computational structure of the wavelet approximation is shown in Figure 5–3.

5.3.Separable Wavelet Approximation

Most of the notations and approximations are 2D extensions of ones in the previous section. First a 2D mother wavelet $\psi_b(x,y)$ is decomposed into a separable form using SV/OSD,

$$\psi_b(x, y) = \sum_i^P a_i(x) b_i(y)\ . \tag{5.33}$$

Then each 1D function is approximated by the basic spline.

$$a_i(x) \approx \sum_j g_i^x[j]\,\phi_I(x-j) \quad \text{and} \tag{5.34}$$

$$b_i(y) \approx \sum_j g_i^y[j]\,\phi_I(y-j) \quad . \tag{5.35}$$

Following the development in Section 5.2, the decomposition can be done in a recursive fashion using discrete filters $\tilde{h}[m]$, $g_i^x[m]$ and $g_i^y[m]$.

$$d_{m,n}^k = \sum_{i_x,i_y} f[i_x,\ i_y]\,\psi_b^{k-1}(i_x - 2^{k-1}m,\ i_y - 2^{k-1}n) = \sum_{i_x,i_y} f[i_x,\ i_y]\,2^{-k+1}\psi_b(2^{-k+1}i_x - m,\ 2^{-k+1}i_y - n)$$

$$\approx \sum_{i_x,i_y} f[i_x,\ i_y]\,2^{-k+1} \sum_l a_l(2^{-k+1}\,i_x - m)b_l(2^{-k+1}\,i_y - n)$$

$$\approx \sum_{i_x,i_y} f[i_x,\ i_y]2^{-k+1} \sum_l \left\{ \sum_{j_x} g_l^x[j_x]\phi_I(2^{-k+1}i_x - m - j_x) \right\} \left\{ \sum_{j_y} g_l^y[j_y]\phi_I(2^{-k+1}i_y - n - j_y) \right\}$$

$$= \sum_l \sum_{j_x} g_l^x[j_x - m] \sum_{j_y} g_l^y[j_y - n]\,c_{j_x,j_y}^{k-1} \quad , \text{where} \tag{5.36}$$

$$c_{m,n}^k = \sum_{i_x,i_y} f[i_x,\ i_y]\,\phi_I^k(i_x - 2^k m)\phi_I^k(i_y - 2^k n) = \sum_{i_x,i_y} f[i_x,\ i_y]\,2^{-k}\phi_I(2^{-k}i_x - m)\phi_I(2^{-k}i_y - n)$$

$$= \sum_{i_x,i_y} f[i_x,\ i_y]2^{-k} \left\{ \sqrt{2} \sum_{j_x} \tilde{h}[j_x]\phi_I(2^{-k+1}i_x - 2m - j_x) \right\} \left\{ \sqrt{2} \sum_{j_y} \tilde{h}[j_y]\phi_I(2^{-k+1}i_y - 2n - j_y) \right\}$$

$$= \sum_{j_x} \tilde{h}[j_x - 2m] \sum_{j_y} \tilde{h}[j_y - 2n] \sum_{i_x,i_y} f[i_x,\ i_y]\phi_I^k(i_x - 2^{k-1}j_x)\phi_I^k(i_y - 2^{k-1}j_y)$$

$$= \sum_l \sum_{j_x} \tilde{h}[j_x - 2m] \sum_{j_y} \tilde{h}[j_y - 2n]c_{j_x,j_y}^{k-1} \quad , \text{with} \tag{5.37}$$

$$c_{m,n}^0 = \sum_{i_x,i_y} f[i_x,\ i_y]\,\phi_I(m - i_x)\,\phi_I(n - i_y) \quad . \tag{5.38}$$

The level k decomposition is performed on a decimated $k–1$ level decomposition, namely c^{k-1}.

Note that the low pass part of the decomposition (5.37) is merely a separable 2D filtering by the same filter $\tilde{h}[m]$ in both dimensions. The prefiltering part of the decomposition (5.38) is also a separable 2D filtering by the interpolation filter $\phi_I(m)$. Plots of $\phi_I(m)$ and $\tilde{h}[m]$ for various spline orders are shown in Figure 5–4. The high pass part of the decomposition ((5.36))

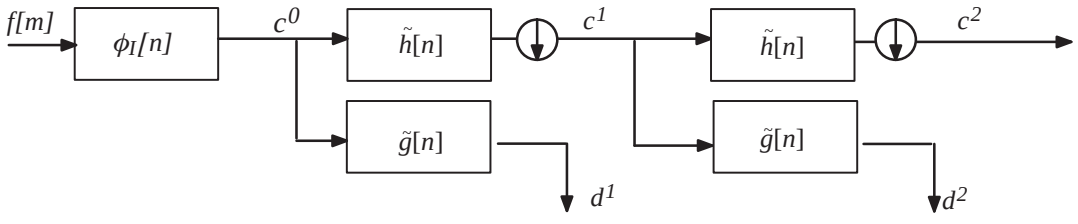
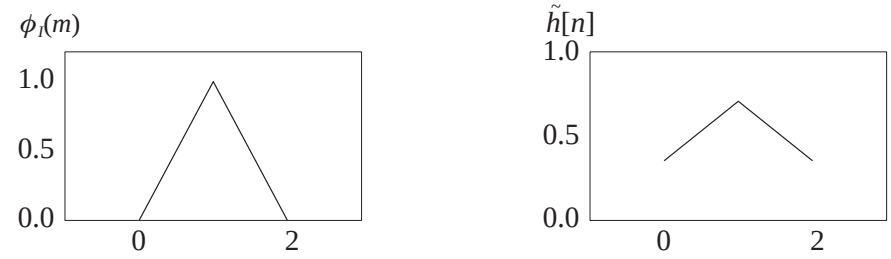
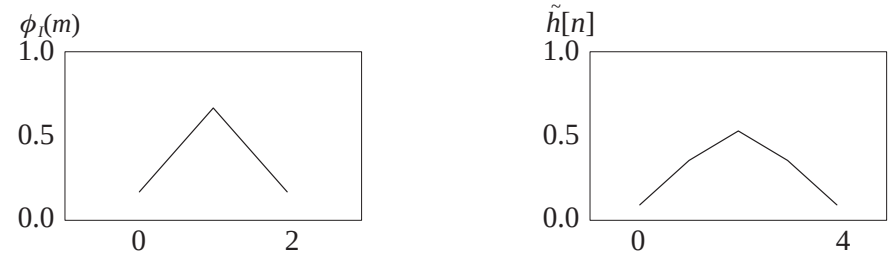


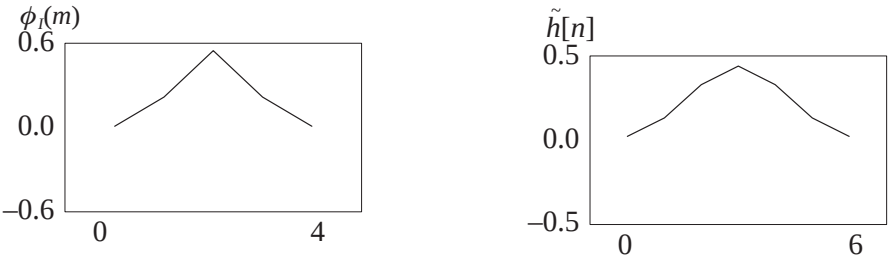
Figure 5–3: Computational Structure of the Wavelet Approximation



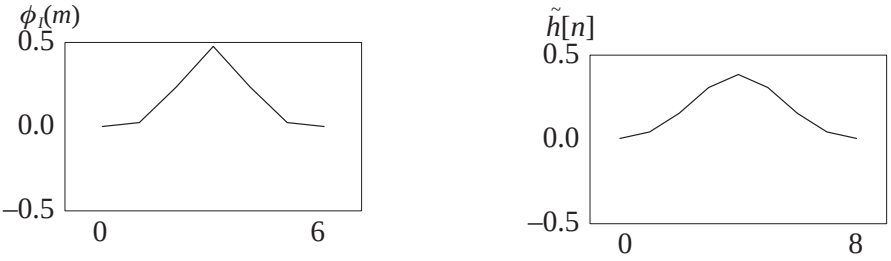
(a) Spline Order = 1



(b) Spline Order = 3



(c) Spline Order = 5



(d) Spline Order = 7

Figure 5–4: Low Pass Filters for SWA

uses a filter bank structure to implement the separable approximation. It contains a set of separable filters with $g_k^x[m]$ for the orthogonal direction and $g_k^y[m]$ for the projection direction. Figure 5–5 shows the structure of SWA. There can be multiple high–pass banks for a system with multiple orientational filters. In that case, the same pre–filter stage and the low–pass bank can be shared among the multiple high–pass filter banks. Also the set of orthogonal filters, $g_k^x[n]$, can be shared among the multiple orientational filters when SV/OSD is used for Separable Approximation. Figure 5–6 shows an example of multi–resolution image decomposition using SWA. Two gabor filters are involved in the decomposition. They are g_b:$(\alpha, \sigma_x, \sigma_y, \theta)$ = *(10.0, 0.15, 0.45, π/4)* and *(10.0, 0.15, 0.45, 3π/4)*. Only the real parts are used for the decomposition.

Benefits of SWA are examined in terms of the implementation criteria. Assume the size of $\tilde{h}[m]$ is M_h, and the size of the spline filter is M_I.

Since the decomposition can be implemented in a pipeline fashion as shown in Figure 5–5, and every filter is separable, the throughput can be as small as $1/t_m$. Thus, the SWA satisfies the first implementation criterion.

As soon as the pre–filtering stage starts generating outputs, the first stage decomposition can proceed. Also the k^{th} level of the decomposition can proceed as soon as the c^{k-1} is generated. Thus, the whole filter system operates in a pipeline fashion. The timing diagram of the pipeline operation is shown in Figure 5–7. The latency at the pre–filter is NM_It_m. The pixel input rate decreases by 1/4 from one level to the next level because of the decimation at each stage. This implies that the latency increases at each stage since it takes more time to collect necessary pixels. The input rate at the k^{th} level decomposition is $4^{t-1}t_m$. Also the size of the output image decreases by 2 in each dimension. Thus, the latency of the low pass filter bank at the k^{th} level decomposition is $(N/2^{k-1})M_h(4^{k-1}t_m) = 2^{k-1}NM_ht_m$, and the latency of the high bass filter bank at the L^{th} level is $2^{L-1}NM$. The total latency of the L^{th} level decomposition is approximately

$$NM_It_m + \sum_{k=1}^{L-1} 2^{k-1}NM_ht_m + 2^{L-1}NMt_m = Nt_m\{M_I + (2^{L-1} - 1)M_h + 2^{L-1}M\}$$

$$\approx 2^{L-1}N(M_h + M)t_m \tag{5.39}$$

with $M \approx M_h$. Note that M_I=5 and M_h=7 when the 5th order basic spline is used for the wavelet approximation. If the L^{th} level decomposition is done directly using a dilated filter whose size is $2^{L-1}M$x$2^{L-1}M$, the latency in this case is also $O(2^{L-1}t_m\,NM)$. Therefore, the SWA satisfies the second implementation criterion.

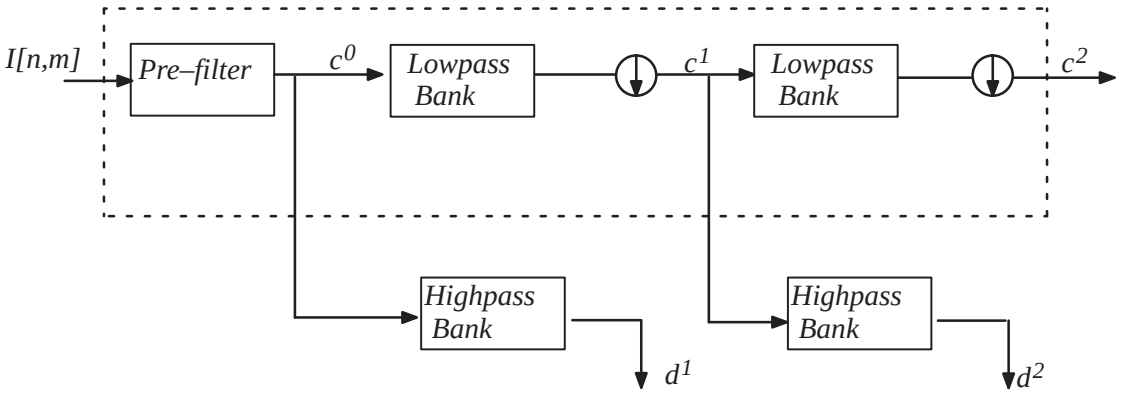
The direct implementation requires computation of $O(F_NLN^2M)$. When SWA is employed, the first level of decomposition requires $2N^2M_I$ multiply–accumulate operations (macs) for pre–filtering, $2N^2M_h$ macs for low–pass filtering, N^2PM macs for vertical filtering in the high–pass banks, and F_NN^2PM macs for horizontal filtering in the high–pass banks. The second level of the decomposition requires $N^2M_h/4$ macs for low–pass filtering, $PN^2M/4$ macs for horizontal filtering in the high–pass banks, and $F_NN^2PM/4$ macs for vertical filtering in the high–pass banks. Thus, the whole decomposition requires $2N^2M_I$ macs for pre–filtering, $4N^2M_h/3$ macs for low–pass filtering, $4PN^2M/3$ macs for vertical filtering in the high–pass banks, and $4F_NN^2PM/3$ macs for horizontal filtering in the high–pass banks. Horizontal filtering in the high–pass banks dominates the complexity of the computation. Therefore the amount of the computation for the decomposition is $4F_NN^2PM/3$. The computation increases only slightly from single–resolution to multi–resolution decomposition, and is much less than the direct method. The SWA satisfies the third implementation criterion.

The pre–filtering stage requires NM_I words of memory. At the k^{th} level of the decomposition, the size of the input image to the filter banks (i.e. c^{k-1}) is 2^{k-1}Nx$2^{k-1}N$ due to the decimation. Thus, at the k^{th} stage, the low–pass filter bank requires $NM_h/2^k$ words of memory and the high pass filter bank stage requires $NM/2^{k-1}$ words of memory. Both the low–pass and high–pass filter access the same part of the input image, thus the input buffer can be shared. Assume $M>M_h$. Then only $NM/2^{k-1}$ words of memory are required instead of $NM/2^{k-1} + NM_h/2^{k-1}$. The total storage required is

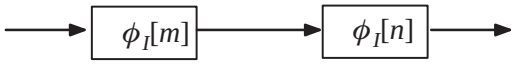
$$NM_I + \sum_{k=1}^{L} N(max(M, M_h))/2^{k-1} < N(M_I + 2max(M, M_h)) \,. \tag{5.40}$$

This quantity is independent of F_N and $O(N)$, and the approximation satisfies the fourth implementation criterion.

Another benefit of SWA is that a large part of the decomposition structure can be shared with other decomposition structures associated with different orientational filters. In Figure 5–5, the portion which can be shared with different orientational filters is enclosed in dashed boxes. Figure 5–6 shows an example image of the multi–resolution decomposition using SWA.

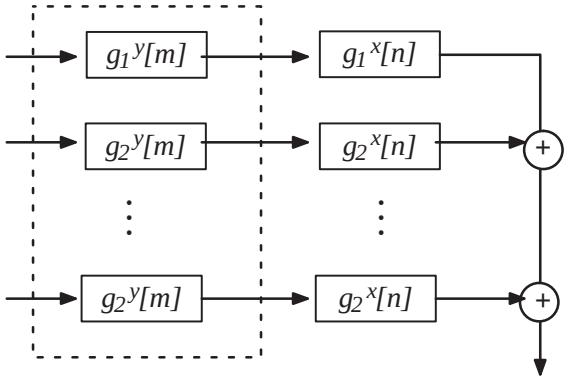


(a) Overall System



(b) Pre–filter Bank

(c) Lowpass Filter Bank



The portion enclosed by dashed boxes can be shared with other orientational filters

(d) Highpass Filter Bank

Figure 5–5: Computational Structure for 2D MRD Using SWA

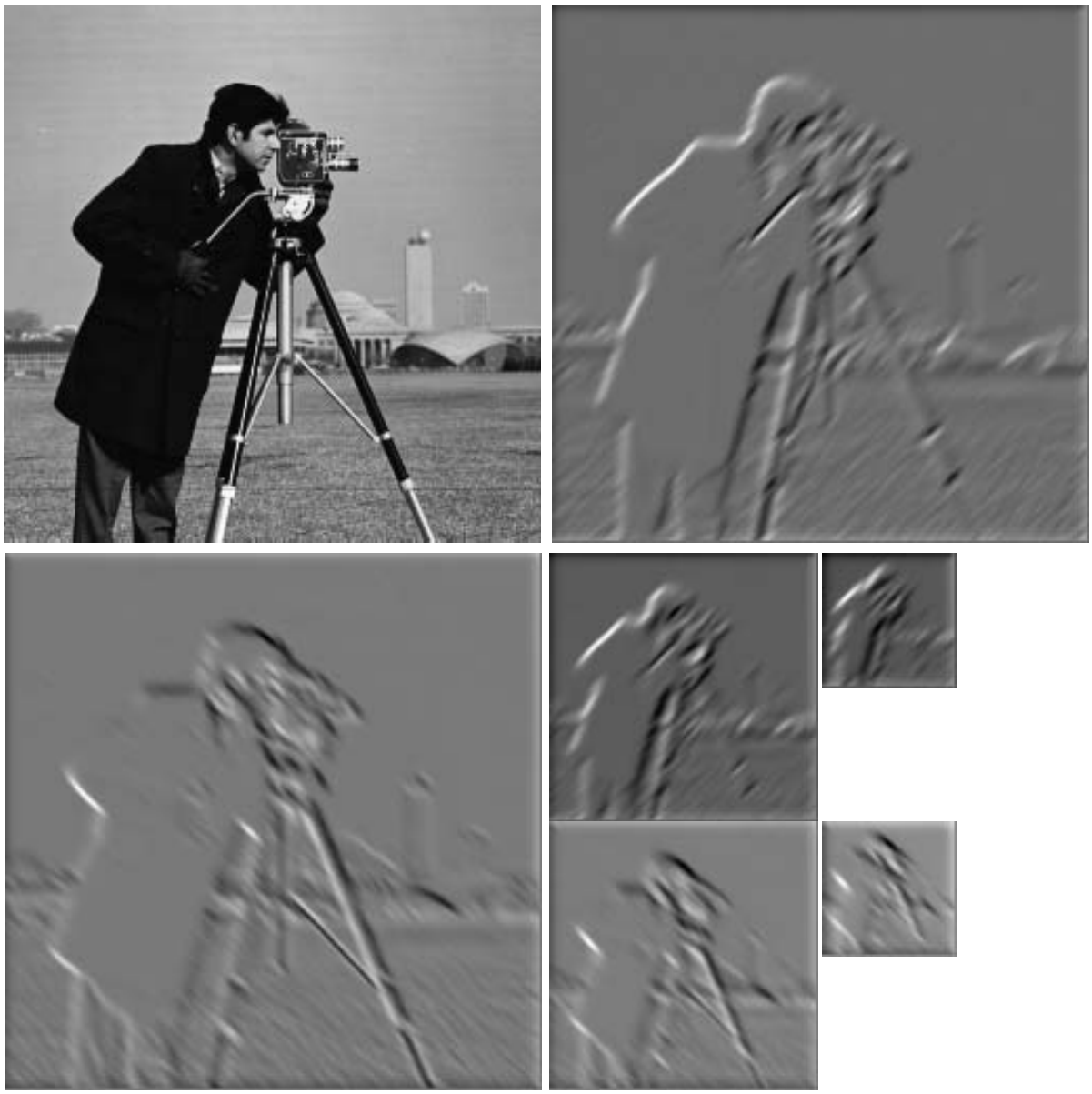


Figure 5–6: Example of Multi–resolution Decomposition Using SWA

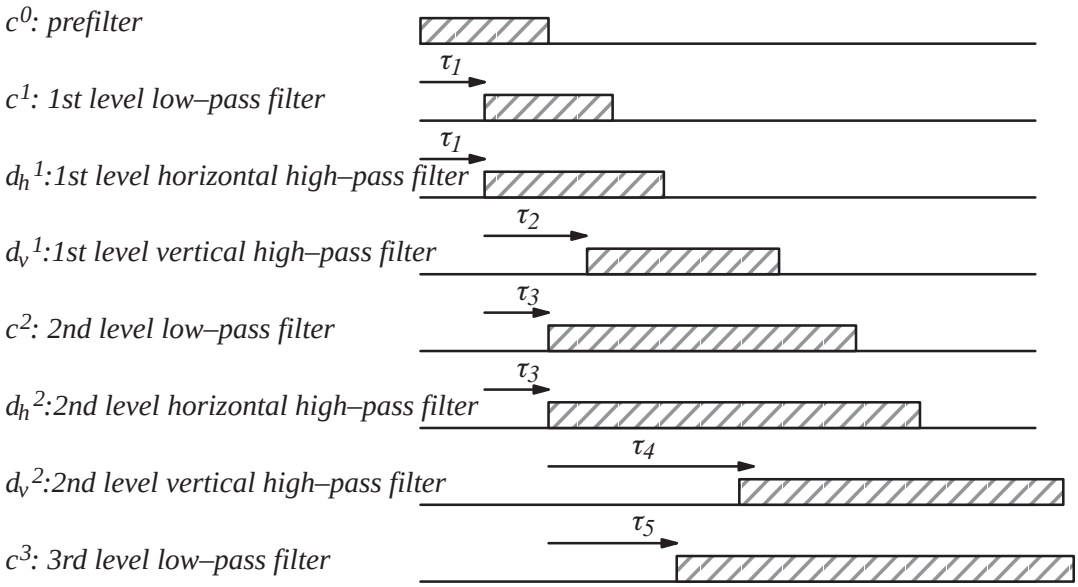
5.4.Undecimated Separable Wavelet Approximation

As described in Section 5.3, the decimation at each level of the SWA plays an essential role in efficient decomposition. However, in some applications, it is preferred to have the decomposition without decimation. This is common for image analysis applications since the decimation introduces aliasing and causes the transformation to be shift invariant. This section examines how to produce an undecimated MRD without losing computational and implementational advantages of the SWA.

In [80], Shensa described two schemes for undecimated DWT. The first scheme is equivalent to the Trous Algorithm introduced in [27] and [38] except the constraints imposed on filters in the Trous Algorithm are removed to be more general. In this algorithm, the wavelets are dilated consecutively at each decomposition level, and the input sequence is convolved with the dilated wavelet. The dilation operator is equivalent to inserting a zero at every sample. So the dilated sequence is twice as long as the sequence before dilation. This scheme is depicted in Figure 5–8. Since the dilation operator inserts zero, the number of multiplications for each convolution using the dilated filter is the same as that for using the non–dilated filter. If the non–decimated output $\tilde{d}^k$ is decimated by 2^k, the sequence becomes identical to the decimated output d^k.

The other scheme is to use duplicate hardware to compute a set of decimated DWTs. which are combined to form an undecimated DWT. The scheme is depicted in Figure 5–9. The idea is to use multiple DWT modules, where each module performs a DWT at different time points. This scheme increases the amount of hardware exponentially as the decomposition level increases. The utilization of the DWT modules becomes worse for later decomposition stages.

The idea of undecimated DWT schemes can be expanded for SWA. The 2D expansion of the second scheme (using duplicate DWT modules) is depicted in Figure 5–10. In the 2D case, shifting needs to be done in the horizontal direction only, the vertical direction only, and both the horizontal and the vertical directions. Thus, the scheme introduces 4 duplicate DWT mod-



	Description	Time
τ_1	Delay from c^0 to c^1 or $d_h{}^1$	$NM_I\ t_m$
τ_2	Delay from $d_h{}^1$ to $d_v{}^1$	$NM\ t_m$
τ_3	Delay from c^1 to c^2 or $d_h{}^2$	$NM_h\ t_m$
τ_4	Delay from $d_h{}^1$ to $d_v{}^2$	$2NM\ t_m$
τ_5	Delay from c^2 to c^3	$2NM_h\ t_m$

Figure 5–7: Pipeline Operation of the Separable Wavelet Approximation

ules at each parent DWT module as shown in Figure 5–10. An enormous amount of hardware is required for a system with a large decomposition level, and the utilization of the hardware becomes very poor.

Another scheme is developed based on Shensa's first scheme (Trous Algorithm). The Trous Algorithm can be adapted for 2D SV/OSD by dilating each 1D filter after the separable approximation. As discussed in Section 3.5, the horizontal filters can be efficiently implemented using pipelined filtering, and the vertical filters can be implemented effectively using parallel filtering. For horizontal filters, a suitable implementation of a dilated filter is to add a shift register with a programmable length between two adjacent adders. The number of shifts is the number of zeros introduced by the dilation and is equal to 2^{k-1} at the k^{th} level decomposition.

Assume VLSI chips are to be designed for the horizontal filtering. The design using shift registers as mentioned above has some disadvantages in terms of its scalability and the utilization of its silicon area. The number of shifts required in the

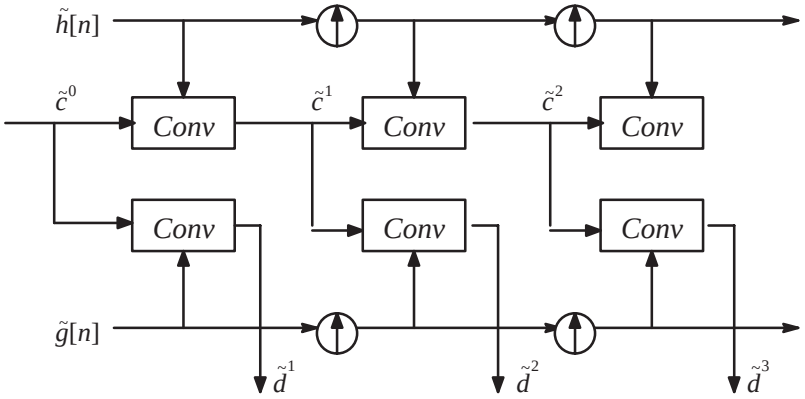


Figure 5–8: Structure of Trous Algorithm

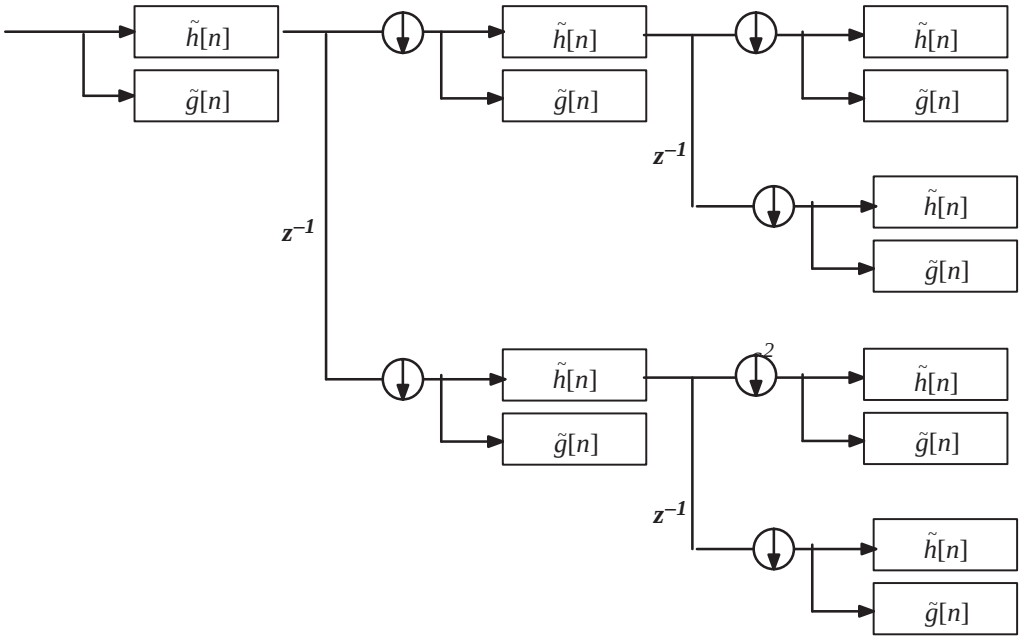


Figure 5–9: Structure of Undecimated DWT using Duplicate hardware

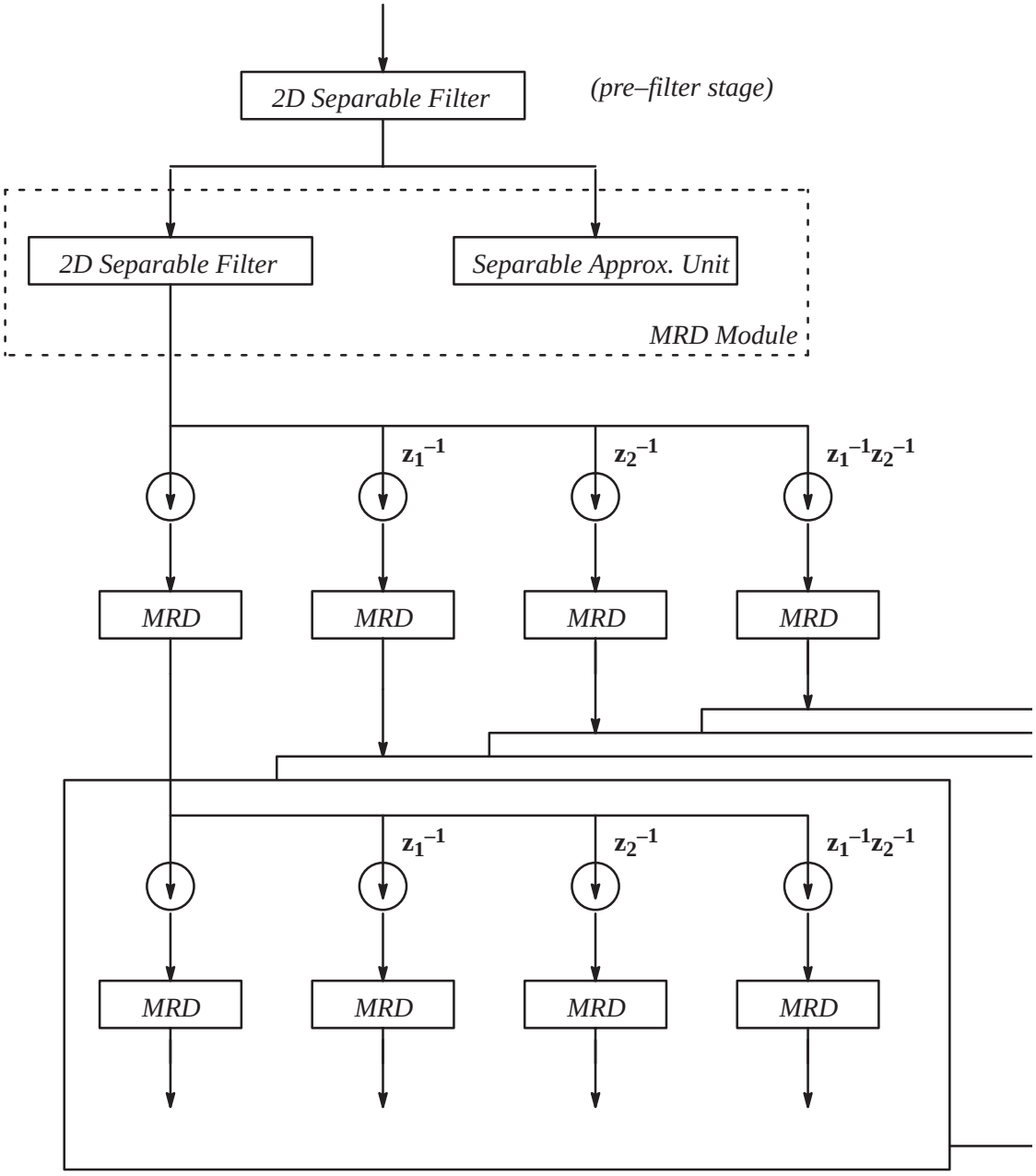


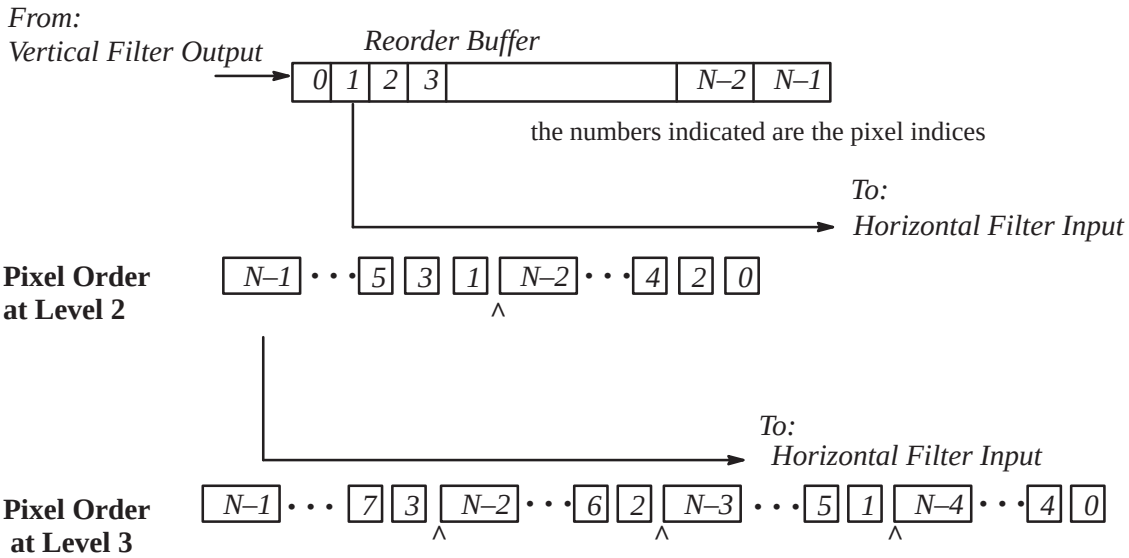
Figure 5–10: Undecimated SWA: Scheme I

shift registers increases exponentially as the decomposition level increases. At decomposition level seven, the number of shifts required is 2^{7-1}=64. If the number of multiplication units within the chip is 8, and the width of the shift registers is 16. The total number of bits to implement the shift registers is 64 x 8 x 16 = 8192. The number increases exponentially as the maximum decomposition level supported by the chip increases, and the design is not scalable. Also, all the shift registers are used only at the maximum decomposition level, and most bits are unused for lower decomposition levels.

An alternative to the above design is to use an external buffer (*Reorder Buffer*). The reorder buffer stores one row of data after vertical filtering, and feeds input to the horizontal filtering units so that the long shifting delay between two multiplication units can be deleted. The idea is related more closely to the duplicate DWT module than to the Trous Algorithm. The scheme uses one DWT module which is time–multiplexed to compute multiple DWTs at different time–points. Figure 5–11 illustrates the reordering scheme of the reorder buffer. A counter and a PAL programmed for each decomposition level is used to provide a proper address to the reorder buffer. This scheme is scalable and the utilization of the chip area for the horizontal filter unit is much better. The filter unit can contain more multipliers reducing the number of chips needed for implementing a horizontal filter. The scheme requires an extra PAL and counter. However, they can be shared among other horizontal filters at the same level.

For vertical filters, a suitable implementation of a dilated filter is to introduce $Mx2^{k-1}M$ multiplexers between the input buffer and the filter unit. The multiplexer selects the rows where non–zero filter coefficients are aligned. The structure of the vertical filter module is shown in Figure 5–12. These modifications to the pipelined and parallel filtering schemes are shown in Figure 5–13. The complete structure for an undecimated multi–resolution decomposition system using SWA is shown in Figure 5–14.

Using the scheme, the latency of the decomposition stays the same with the decimated SWA case, which is approximately $2^{L-1}N(M_h + M)t_m$, the throughput also stays $1/t_m$, the amount of computation is $2N^2M_I+2LN^2M_h+LN^2MP+LF_NN^2MP= 2N^2M_I + LN^2(2M_h+MP+F_NMP)$, and the amount of storage is $NM_I+LN\ max(M,M_h) + NL(F_NP+1)$. The number of reorder buffers in the system is $L(F_NP+1)$ and the number of reorder buffer address generators is L.



At ^ mark in the above pixel orders, the accumulation chain in the pipelined filter unit has to be broken, and a new accumulation has to start.

Figure 5–11: Reorder Buffer and Its Reordering Scheme

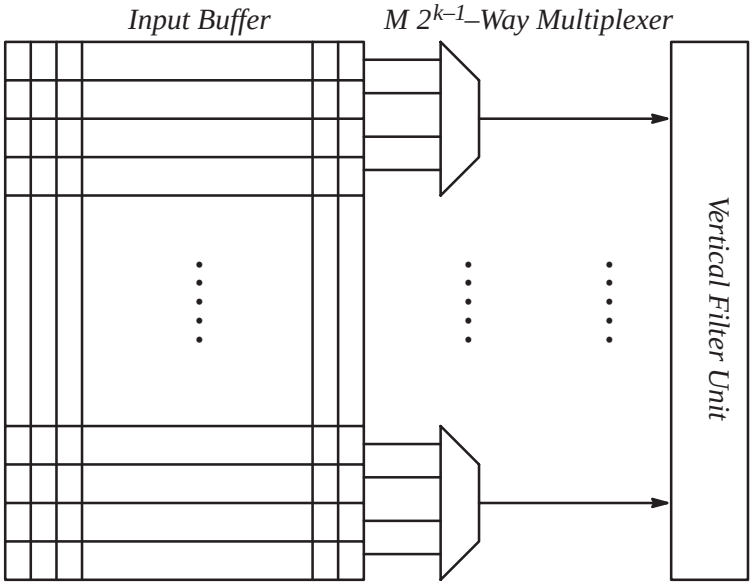


Figure 5–12: Structure of the Vertical Filter Module for the Trous Algorithm

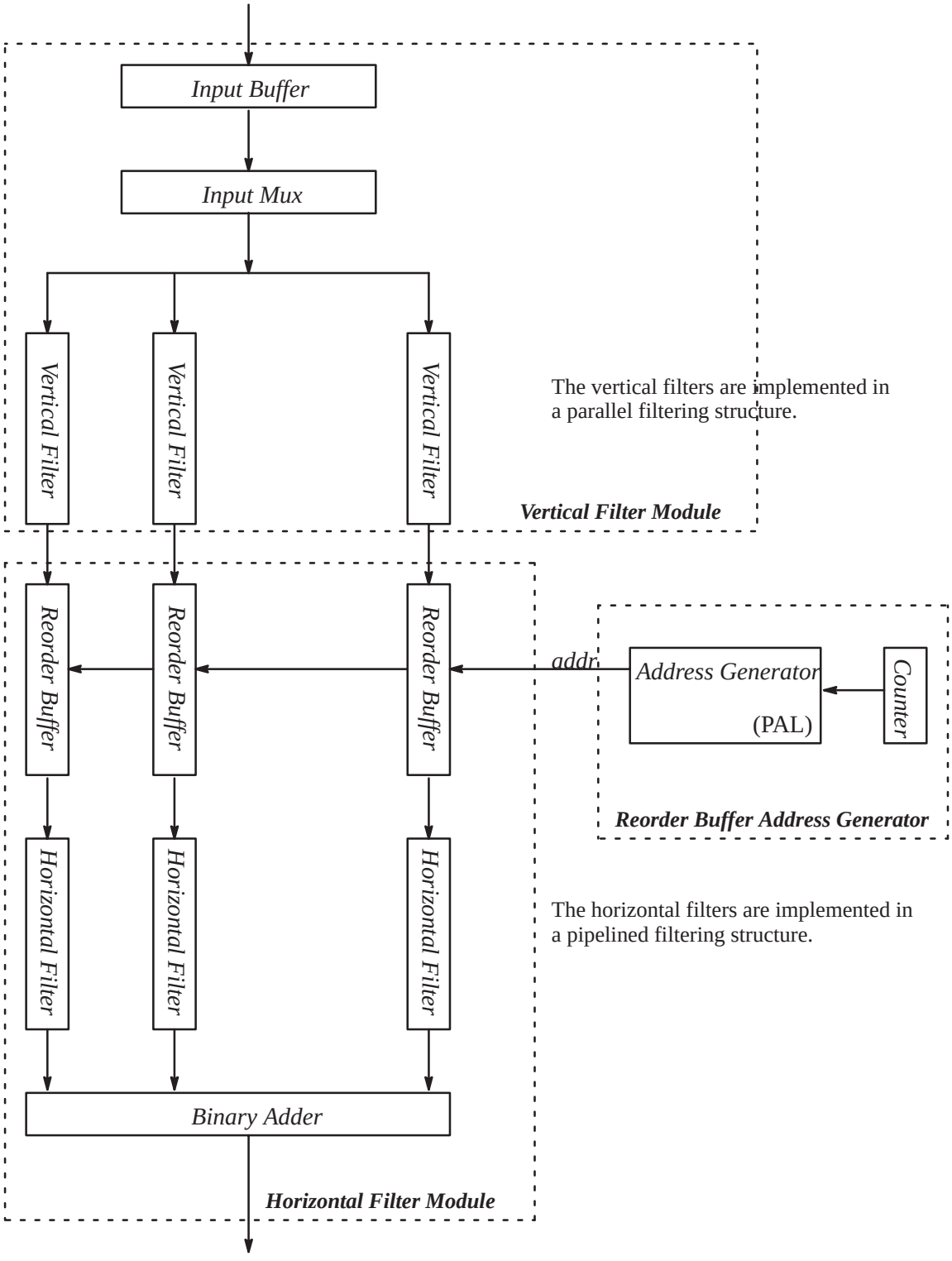
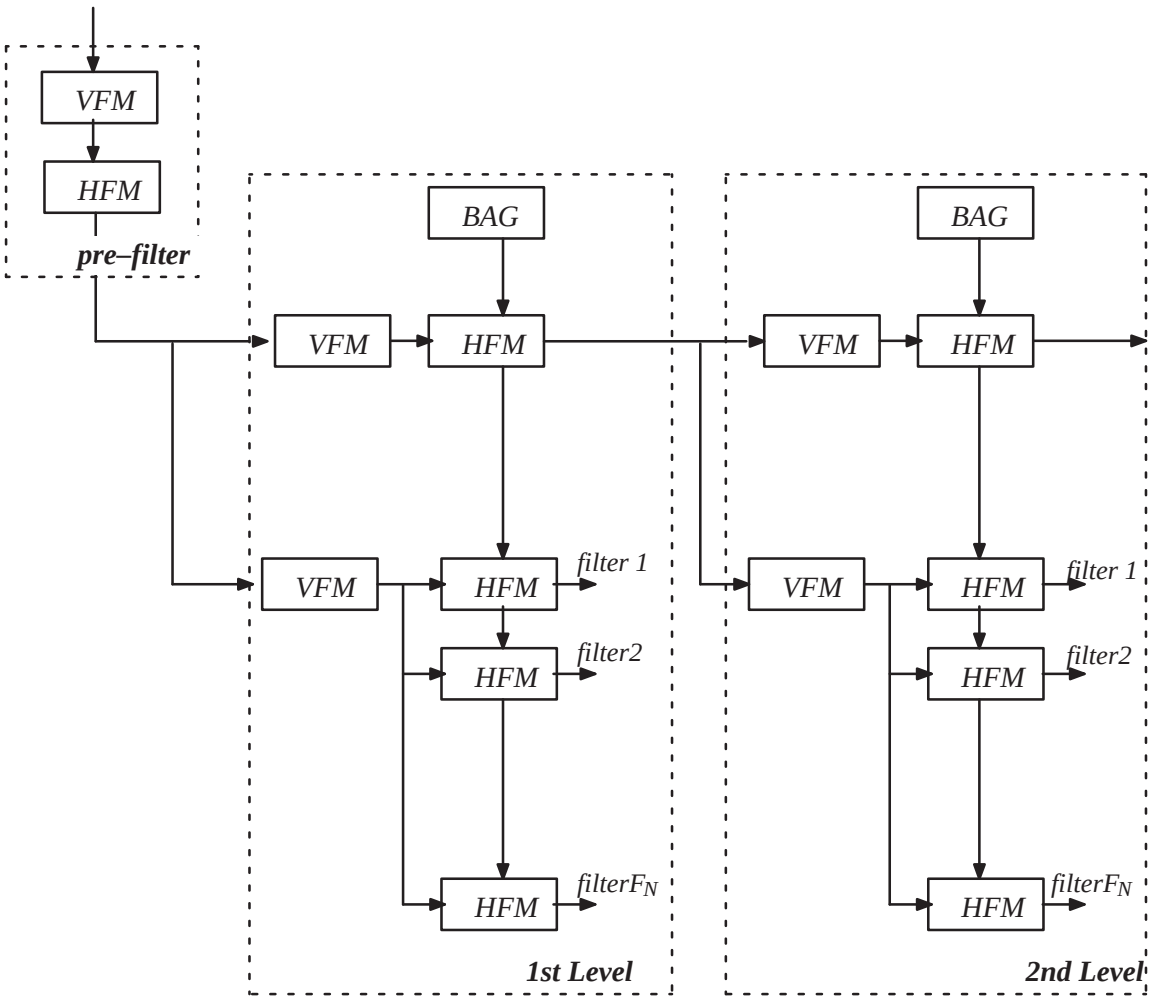


Figure 5–13: Undecimated SWA: Scheme II



BAG = Reorder Buffer Address Generator

VFM = Vertical Filter Module

HFM = Horizontal Filter Module

(See Figure 5–13)

Figure 5–14: The Structure of Undecimated MRD Using SWA

CHAPTER 6

MRD Performance Evaluation

This chapter examines the performance of SWA using energy and frequency distortion as measurement criteria. The chapter then examines the application of SWA in specific computer vision problems.

6.1.Energy Error Analysis

The behavior of the approximation error is studied in the spatial domain using Gabor filters with various parameter values. The real part and the imaginary part of the filters are investigated separately. The energy error (*L2* error) defined in (4.5) is used for measurement. In this case, *I[m,n]* is the original dilated filter obtained by sampling the Gabor function, and *Ap[m,n]* is obtained using the SWA. Thus, the filter is first separably approximated and each 1D filter is interpolated using a basic spline.

The results of the evaluation are shown in Figures 6–1 through 6–4. Figures 6–1 and 6–3 show the results of approximating the real part of Gabor filters with a separable approximation order of 5 and 10, respectively. Figures 6–2 and 6–4 show the results of approximating the imaginary part of Gabor filters with a separable approximation order of 5 and 10, respectively. In all cases, the spline order is set to 7. In each figure, the results are plotted separately for different dilation levels (Level 1 through 4), and the aspect ratio (σ_x/σ_y) and frequency (α) of the filter are varied within each plot.

The amount of error increases only slightly as the dilation level increases. The error characteristics are very similar between the real part and the imaginary part of the filter. The approximation becomes more difficult as the frequency increases. It also becomes more difficult as the aspect ratio increases/decreases away from 1.0.

6.2.Basic Spline Order

The effect of changing the order of the basic spline function is examined by approximating a Gabor function g_b*:*(α, σ_x, σ_y, θ *) = (10.0, 0.2, 0.4,* π*/4)* using SVD. The result is shown in Table 6–1. The *L2* error decreases as the spline order increases, and it stops decreasing after the order reaches 5. Note that the larger the order becomes, the larger the sizes of both the pre–filter $\phi[m, n]$ and low pass filter $\tilde{h}[n]$ become. Hence the spline function of order 5 achieves a good performance/implementation trade–off.

Table 6–1: Effect of Basic Spline Functions on Approximation Performance
The real part of Gabor filter g_b*:*(α, σ_x, σ_y, θ *) = (10.0, 0.2, 0.4,* π*/4)* is approximated using SV/OSD with O_N*=16*. The errors shown are L2 error at the fourth level of the multi–resolution decomposition.

Spline Order	1	3	5	7	9
L2 %	98.4	1.60	0.0934	0.0690	0.0648

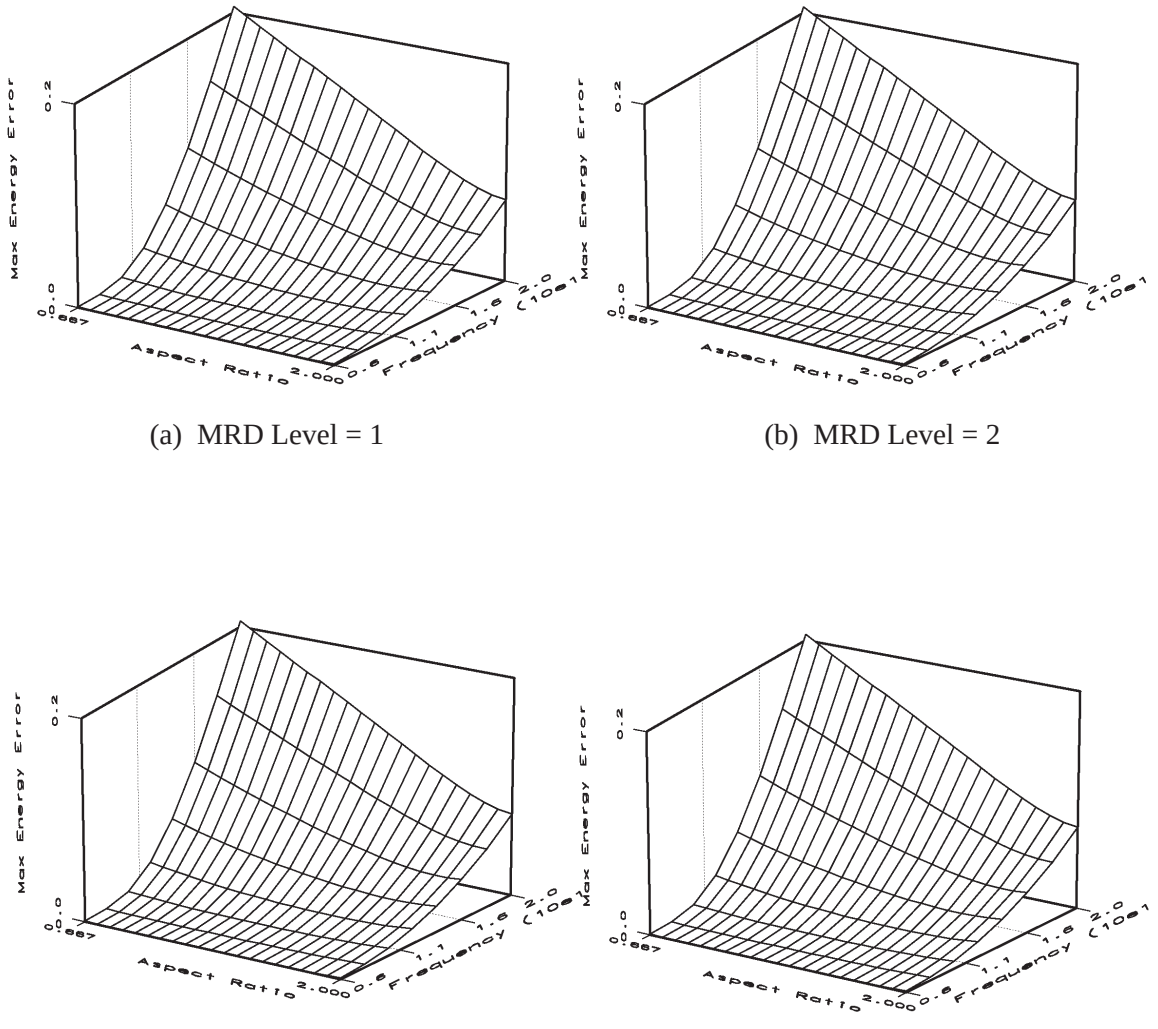
6.3.Frequency Distortion

Next, the error introduced by SWA is observed in the frequency domain. As an example, Figure 6–5 shows the frequency response of a dilated Gabor filter (σ_x=0.2, σ_y=0.4, and α=10.0) and the frequency distortion introduced by the approximation.

The frequency distortion tends to match the frequency response of the filter at any orientation as was the case for SV/OSD (see Section 4.2). The distortion energy is concentrated at the same points where the energy content of the filter is significant. Thus, the effect of the distortion is insignificant in most cases compared to the effect of the filter. The same can be said for Gabor filters with other sets of parameters.

6.4.Applicability to Computer Vision Problems

6.4.1.Edge Detection



(a) MRD Level = 1

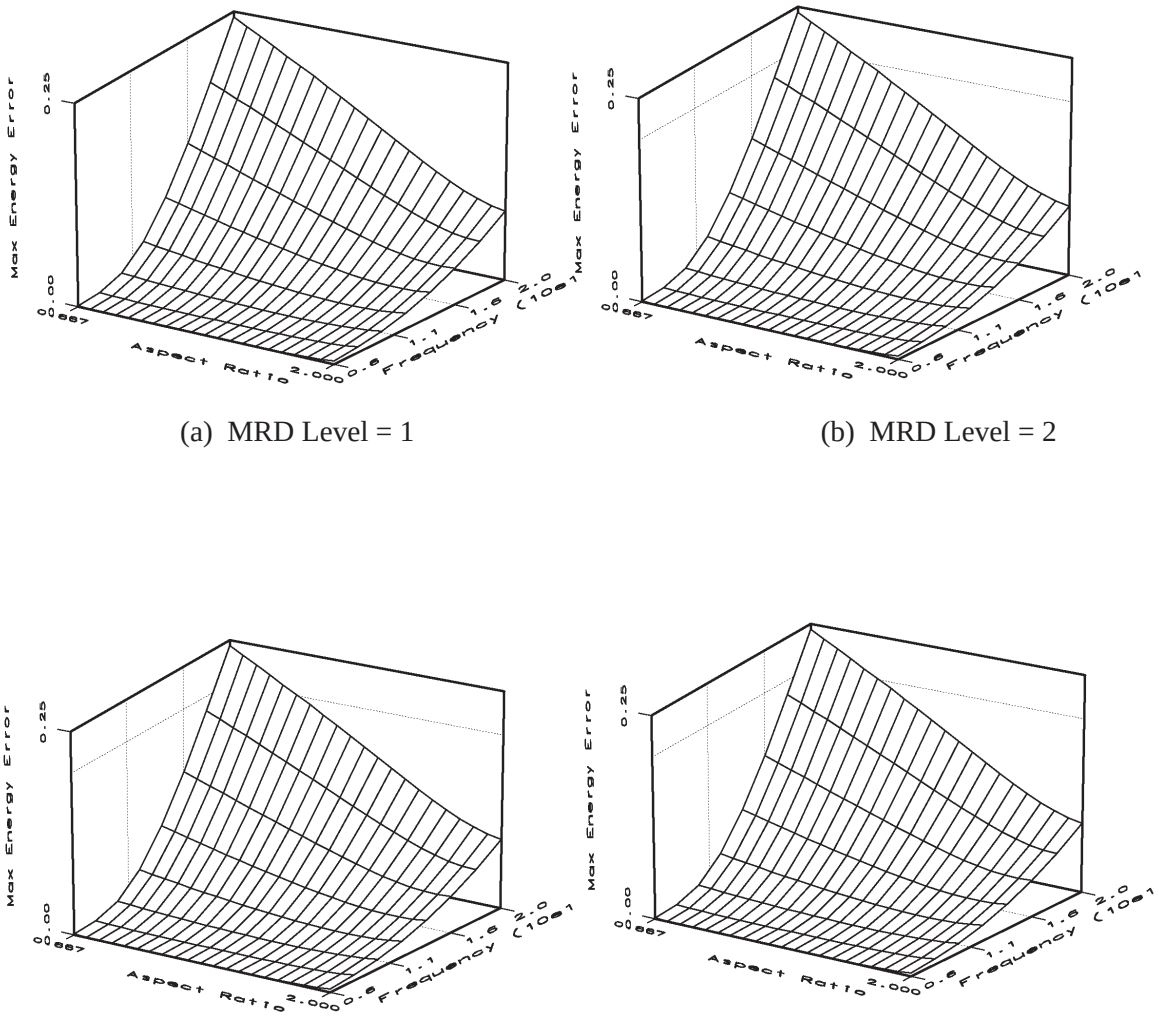
(b) MRD Level = 2

(c) MRD Level = 3

(d) MRD Level = 4

Figure 6–1: The Result of MRD Performance Evaluation

MRD using the real part of Gabor filters are approximated by SWA. The SV/OSD approximation order is set to 5, and the order of basic spline is set to 7 throughout the evaluation.



(a) MRD Level = 1

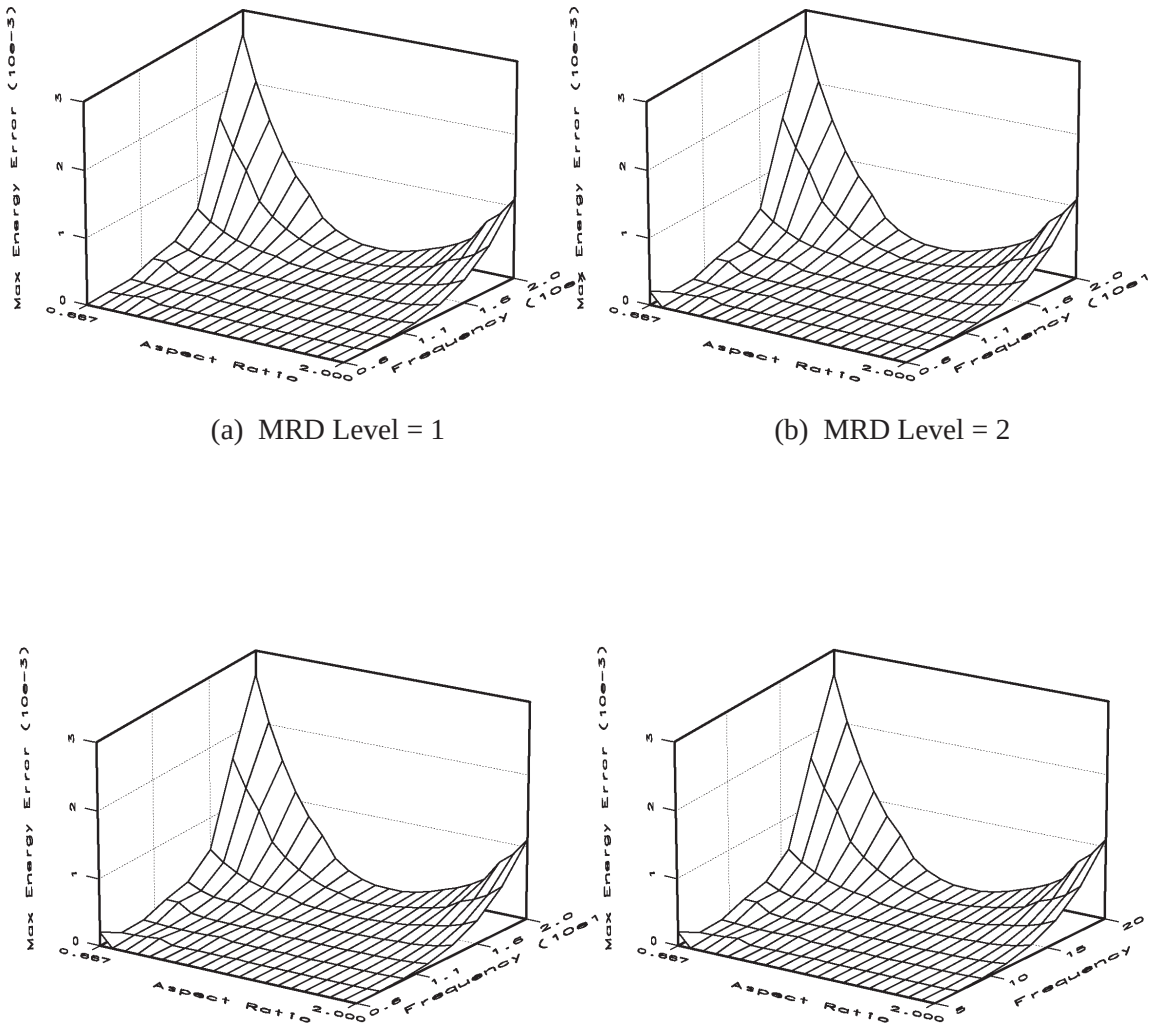
(b) MRD Level = 2

(c) MRD Level = 3

(d) MRD Level = 4

Figure 6–2: The Result of MRD Performance Evaluation

MRD using the imaginary part of Gabor filters are approximated by SWA. The SV/OSD approximation order is set to 5, and the order of basic spline is set to 7 throughout the evaluation.



(a) MRD Level = 1

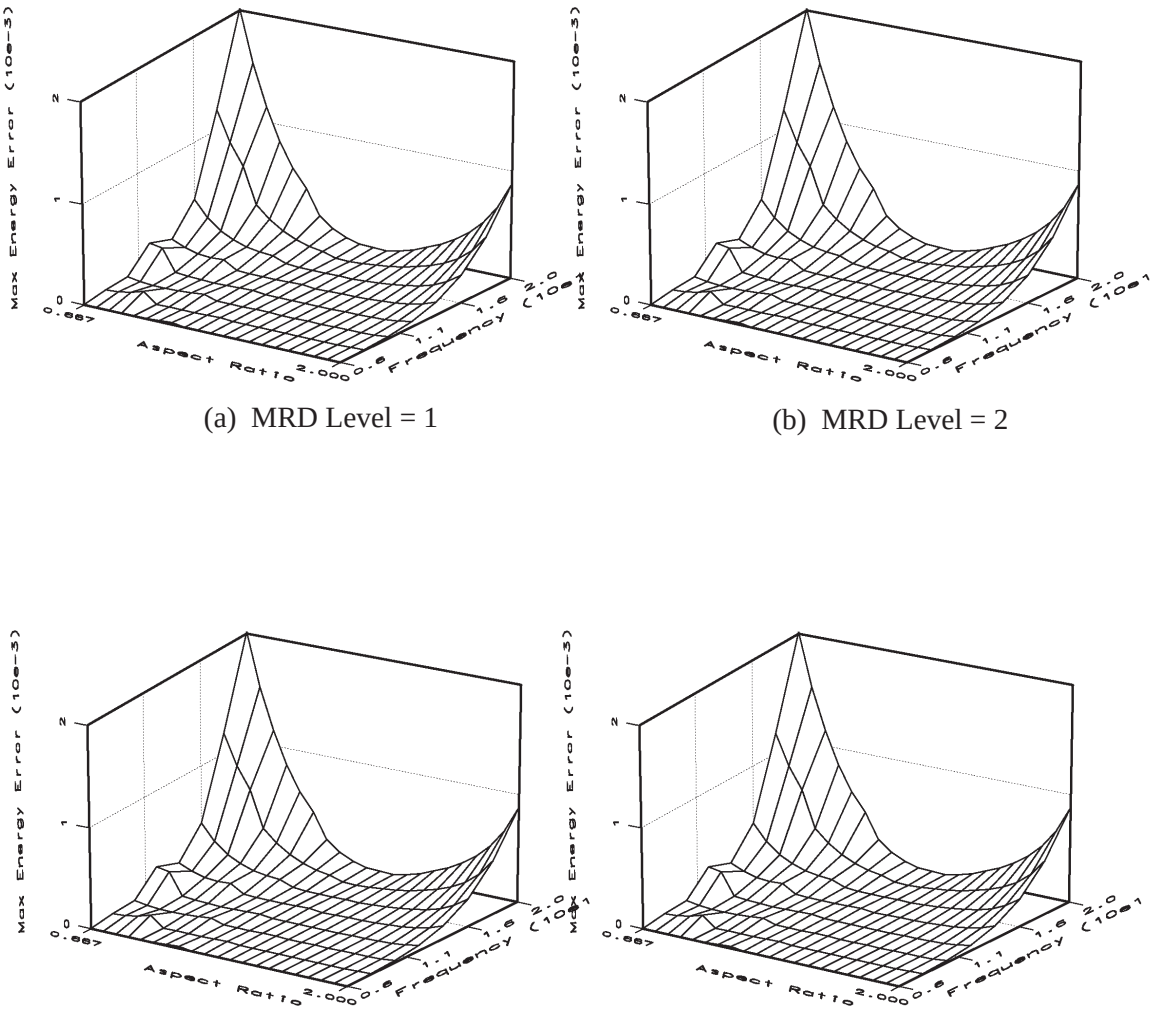
(b) MRD Level = 2

(c) MRD Level = 3

(d) MRD Level = 4

Figure 6–3: The Result of MRD Performance Evaluation

MRD using the real part of Gabor filters are approximated by SWA. The SV/OSD approximation order is set to 10, and the order of basic spline is set to 7 throughout the evaluation.



(a) MRD Level = 1

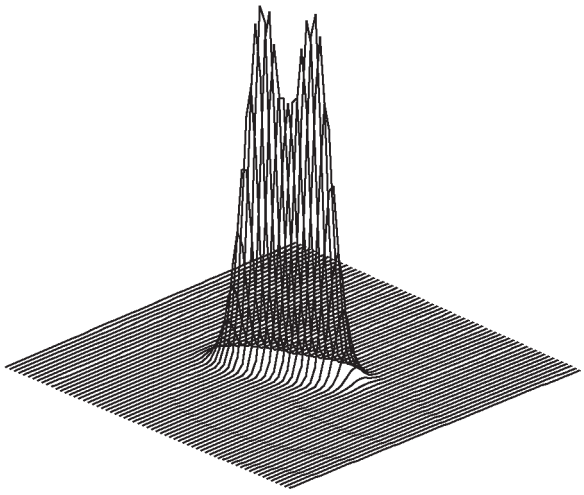
(b) MRD Level = 2

(c) MRD Level = 3

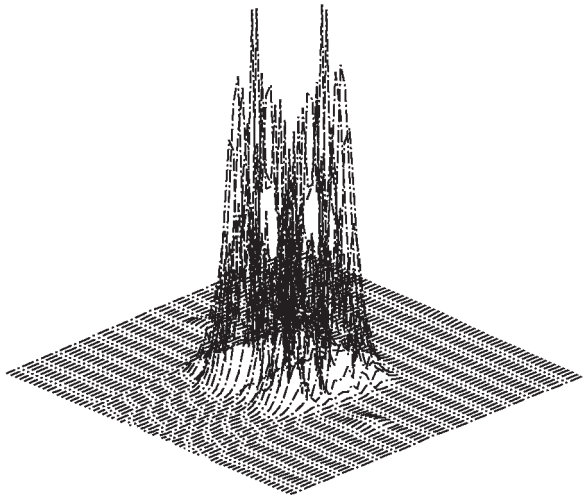
(d) MRD Level = 4

Figure 6–4: The Result of MRD Performance Evaluation

MRD using the imaginary part of Gabor filters are approximated by SWA. The SV/OSD approximation order is set to 10, and the order of basic spline is set to 7 throughout the evaluation.



(a) Frequency Response of the Original Filter
(Peak Value = 18.0)



(b) Frequency Distortion of the Approximated Filter
(Peak Value = 1.2)

Figure 6–5: Frequency Distortion Characteristic of the SWA

The real part of Gabor filter with α=10, σ_x=0.2 and σ_y=0.4 is used for this evaluation. The frequency distortion shown in (b) is for the dilated filter at the 1st level of the decomposition. The approximation order for SV/OSD is 5, and the order of basic spline is 7.

In this section, an edge detection algorithm is implemented on a decimated multi–resolution image pyramid. Performance is compared between the case when the algorithm is applied to the original multi–resolution pyramid and the case when it is applied to an approximated pyramid. The approximated pyramids are obtained by using the SWA.

The algorithm utilizes a decimated multi–resolution image structure. It performs edge detection recursively starting from the top level of the multi–resolution pyramid which is the smallest in size and coarsest in resolution. The algorithm searches for edges in the subsequent level only in the neighborhood where edges are detected in the previous level[3].

The edge operator is similar to the one introduced in Section 4.3.1 It uses the second derivative of the Gaussian, $d^2g/dx^2:(\sigma_x, \sigma_y, \theta) = (0.15, 0.45, \pi/4)$. The test image is also similar to the one used in Section 4.3.1 It consists of four step edges with Gaussian noise superimposed on the edges. The step edges start at the center of the image and extend all the way to each corner of the image. The Gaussian noise has zero mean and a variance of 100, while the edge intensity is set to 100. The size of the image is 256x256.

Without the multi–resolution image structure, the number of edge operations to detect all the edges is 256x256 = 65536. However, with the multi–resolution image structure, the number of edge operations is significantly reduced. The performance evaluation in this section shows that approximately 6000 edge operators are necessary to detect all the edges, approximately 10% of the operations necessary on a single layer image structure.

The performance is evaluated in terms of the number of spurious edge pixels. The orientational filter $d^2g/dx^2:(\sigma_x, \sigma_y, \theta) = (0.15, 0.45, \pi/4)$ is approximated using SV/OSD. The approximation matrix for the SV/OSD is obtained through a set of filters $\{d^2g/dx^2:(\sigma_x, \sigma_y, \theta) = (0.15, 0.45, \theta_k): \theta_k = \frac{\pi k}{O_N} \ (0 \le k < O_N)\}$. The evaluation is done for $O_N=1, 4, 8$ and 16. The results are shown in Table 6–2.

Table 6–2: Performance of Edge Detection on SWA

	# of False Edge Pixels			
d^2g/dx^2	1			
P	$O_N=1$	$O_N=4$	$O_N=8$	$O_N=16$
P=10	1	1	1	1
P=8	1	1	9	9
P=6	1	9	13	13
P=5	1	9	16	16
P=4	1	40	24	21
P=3	5	31	23	22
P=2	54	531	511	22
P=1	821	480	511	507

The results show that the edge detection algorithm is effectively implemented on an approximated multi–resolution image pyramid using SWA. The value, *P*, required for the separable approximation increases as O_N increases.

6.4.2.Texture Segmentation

In this section, unsupervised texture segmentation is implemented using the undecimated multi–resolution image pyramid. Performance is compared between the case when the algorithm is applied to the original multi–resolution pyramid and the case when it is applied to the approximated pyramids. The approximated pyramids are obtained by using the SWA.

The same segmentation algorithm described in 4.3.2 is used here. However, the input textures consist of natural textures taken from Brodatz's photo album [11] instead of synthetic ones, and the texture boundaries are more complicated hand–drawn ones. The templates used to create textured images are shown in Figure 6–6. The objective of the algorithm is to segment the textured image so that the result is as close to the template image as possible. In order to successfully segment the images, 7 levels of decomposition are required as stated in [41]. The number of orientations in each decomposition level is 4, and 4 Gabor filters oriented at $0, \pi/4, \pi/2$ and $3\pi/4$ are used. The Gaussian deviations of the filters are set to $\sigma_x=0.6$ and $\sigma_y=0.4$. In [41], filtering is implemented in the frequency domain. In order to incorporate the SWA into the segmentation algorithm with the least modification to the algorithm, the original filters used in the algorithm are transformed into the spatial domain, extracted at the center with a smaller window (9x9), and the extracted filters are approximated using the SWA. The approximated filters are transformed into the frequency domain, and they replace the original filters. During this process, errors other than the approximation errors are introduced due to the truncation of the filters. However, the effect of the extra errors appear to be

(a) Template I (b) Template II

Figure 6–6: Template Images Used for Creating Textured Images

insignificant. The results are shown in Figure 6–7. First two textured images were created using Template I, and the last two textured images were created using Template II. Results are very similar between Gabor filters and approximated filters in all test cases.

6.4.3.Georgia Tech Vision Model

The Georgia Tech Vision model (GTV)[25] simulates the human vision search and detection based on computational vision studies. The model includes multiple orientation filters in its *pattern perception units*. The filters are DOG (Difference of Gaussian)[99] and cortex filters introduced in [98]. In this section, SWA is applied to these orientational filters, and the performance is evaluated in terms of the normalized energy error.

The DOG in GTV is defined in the frequency domain as

$$F_{dog}(u,v) = A\pi s_y e^{-(\pi s_y v)^2}\left\{s_1 e^{-(\pi s_1 u)^2} - Bs_2 e^{-(\pi s_2 u)^2} + Cs_3 e^{-(\pi s_3 u)^2}\right\} \quad . \tag{6.1}$$

The inverse Fourier transform of F_{dog} is

$$f(x,y) = \frac{A}{2\pi} e^{-y^2/4\pi s_y}\left\{e^{-x^2/4\pi s_1} - Be^{-x^2/4\pi s_2} + Ce^{-x^2/4\pi s_3}\right\} \quad . \tag{6.2}$$

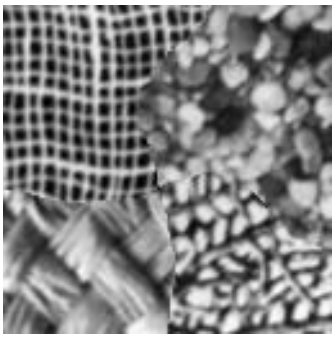
It has 7 parameters, A, B, C, s_1, s_2, s_3, and s_y. For the luminance channel in GTV, 24 DOG filters are implemented, which are divided into 6 radial frequencies (F_1, F_2, F_3, F_4, F_5, and F_6 as shown in Table 6–3) and 4 orientations ($0, \pi/4, \pi/2, 3\pi/4$). The amplitude parameter A can be assumed to be 1, and s_y is set to $.2s_1$ in the luminance channel of GTV. Table 6–3 shows the remaining five parameter values for DOG filters with different radial frequencies.

Now the DOG filters are approximated using SV/OSD. The filter is first derived in the frequency domain (128x128), and is inverse transformed to the spatial domain. The filter in the spatial domain is truncated by a square window so that all coefficients larger than 1% of the maximum coefficient are retained. Table 6–4 shows the result of the approximation.

Table 6–3 Basic Parameters of the Luminance Channel First Stage Filter in GTV

Parameter	F_1	F_2	F_3	F_4	F_5	F_6
B	0.267	0.333	0.894	0.894	1.266	1.266
C	0	0	0.333	0.333	0.500	0.500
s_1	0.198	0.098	0.084	0.059	0.038	0.019
s_2	0.593	0.294	0.189	0.132	0.060	0.030
s_3	–	–	0.253	0.177	0.076	0.038

Table 6–4 Approximation Result Using SV/OSD



Original Texture

Result using Gabor filters

Results using Approximated filters

(a) Texture I

The approximated filters are obtained with the Wavelet Approximation (spline order = 5) and Separable Approximation (order = 5). The result using Gabor filters has 1211 misclassified pixels whereas the result using the approximated filters has 1029 misclassified pixels.

Original Texture

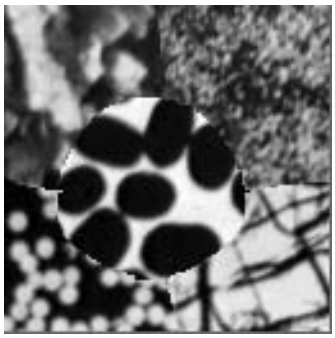
Result using Gabor filters

Results using Approximated filters

(b) Texture II

The approximated filters are obtained with the Wavelet Approximation (spline order = 5) and Separable Approximation (order = 5). The result using Gabor filters has 3315 misclassified pixels whereas the result using the approximated filters has 3390 misclassified pixels.

Figure 6–7: The result of Texture Segmentation



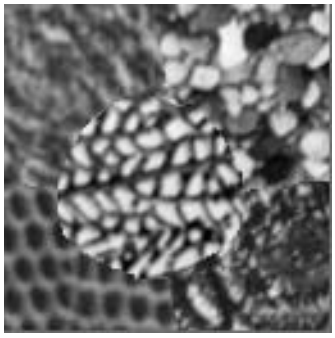
Original Texture

Result using Gabor filters

Results using Approximated filters

(c) Texture III

The approximated filters are obtained with the Wavelet Approximation (spline order = 5) and Separable Approximation (order = 5). The result using Gabor filters has 2538 misclassified pixels whereas the result using the approximated filters has 2494 misclassified pixels.



Original Texture

Result using Gabor filters

Results using Approximated filters

(d) Texture IV

The approximated filters are obtained with the Wavelet Approximation (spline order = 5) and Separable Approximation (order = 5). The result using Gabor filters has 1437 misclassified pixels whereas the result using the approximated filters has 1629 misclassified pixels.

Figure 6–7: The result of Texture Segmentation

24 Wilson DOG filters are approximated using SV/OSD. The table below shows the L2 error of each approximation. The approximation order is set to 6.

	F_1	F_2	F_3	F_4	F_5	F_6
Filter Size	128x128	123x123	128x128	103x103	51x51	25x25
$\theta=0$	2.04e–10	5.46e–7	5.46e–7	2.11e–5	6.02e–7	6.00e–7
$\theta=\pi/4$	6.33e–7	3.38e–4	3.38e–4	7.71e–4	3.20e–4	3.19e–4
$\theta=\pi/2$	1.55e–10	7.42e–7	7.42e–7	2.77e–5	8.94e–6	8.93e–6
$\theta=3\pi/4$	6.33e–7	3.38e–4	3.38e–4	7.71e–4	3.20e–4	3.19e–4

As can be seen in Table 6–4, the filter sizes are relatively large, and the amount of computation is still large even with SV/OSD. The computation can be significantly reduced by approximating these filters using SWA. When the above 24 filters are applied directly to an image, it requires $16740N^2$ macs. By using SWA, the amount of computation reduces to $3128N^2$ assuming a 7^{th} order basic spline is used, and the decomposition levels for F_1, through F_6 are 3, 3, 3, 2, 2, and 1 respectively. The result of the approximation using SWA is shown in Table 6–5. For comparison, if the above filters are implemented in the frequency domain and the input image size is 1024x1024, then the number of multiplications is $4384N^2$ using a row–column FFT. As discussed in Section 2.2.2, real–time filtering using the FFT requires an immense amount of hardware. Thus, even for large filters, implementation of SWA has advantages over the FFT in both the amount of computation and the amount of hardware.

Table 6–5 Approximation Result Using SWA

24 Wilson DOG filters are approximated using SWA. The table below shows L2 error of each approximation. The approximation order is set to 6 and the spline order is set to 7.

	F_1	F_2	F_3	F_4	F_5	F_6
Filter Size	16x16	16x16	16x16	27x27	14x14	13x13
dilation factor	3	3	3	2	2	1
$\theta=0$	2.41e–4	1.60e–5	4.79e–4	2.29e–5	4.54e–3	8.97e–3
$\theta=\pi/4$	8.97e–5	3.38e–4	7.94e–4	1.71e–3	4.54e–4	6.74e–4
$\theta=\pi/2$	2.41e–4	1.79e–5	5.10e–4	3.43e–5	4.58e–3	9.05e–3
$\theta=3\pi/4$	8.93e–5	3.38e–4	7.94e–4	1.71e–3	4.52e–4	6.74e–4

CHAPTER 7

Steerable System

7.1.Introduction

In many image analysis and image processing tasks, it is useful for the filter system to have a capability of changing its orientation dynamically under adaptive control. Such a filter system is called *steerable*[31], and those tasks which utilize the steerability include local orientation analysis, contour following, target tracking and image enhancement[31][67]. In [31], steerability is defined in the following manner.

Definition 3: Filter *h(x,y)* is steerable if a rotated copy of *h(x,y)* at an arbitrary orientation can be expressed by a finite linear sum of rotated copies of itself. Thus,

$$h_\theta(x,y) = \sum_{i=1}^{P} q_i(\theta)\, h_{\theta_i}(x,y) \quad . \tag{7.1}$$

Freeman and Adelson claimed that "all functions that can be expressed as a Fourier series in angle or in a polynomial expansion in *x* and *y* times a radially symmetric window function" are steerable.

Perona extended the idea of Freeman and Adelson, and provided an approximation of *f(x,y)* at an arbitrary orientation with a linear sum of basic filters. In [67], the definition of steerability is modified to a more general form.

Definition 4: Filter *h(x,y)* is steerable if a rotated copy of *h(x,y)* at an arbitrary orientation can be expressed by a finite linear sum of basis filters. Thus,

$$h^\theta(x,y) = \sum_{i=0}^{P} q_i(\theta)\, G_i(x,y) \quad . \tag{7.2}$$

In this thesis, the latter definition of steerability is adopted.

The next section briefly describes the work by Freeman and Adelson. The second section derives the approximation method proposed by Perona. Perona proposed a technique to closely approximate the separable system (7.2) using Singular Value Decomposition. SVD is used to derive a set of basis filters. This technique is referred as Deformable Kernel Approximation (DKA) here. The decomposition generates an infinite set of basis filters, and a sub–set of filters are chosen for the approximation. The basis filters of his technique are intrinsically complex even though the filter to be approximated is real. This thesis proposed to use Fourier series expansion instead of SVD. Then a steerable system of a real filter is approximated by a set of real filters instead of complex filters. The latter method is referred as Fourier Series Approximation (FSA). It can accomplish the same approximation performance as DKA, and it has more flexibility on selecting a set of basis filters for the approximation. Section 7.5 describes how to combine SV/OSD with FSA for a fast and inexpensive implementation, and Section 7.6 describes how to build a steerable MRD system using FSA and SWA. The chapter concludes with the performance evaluation of SV/OSD + FSA.

7.2.Freeman–Adelson's Steerable System

Freeman and Adelson appear to be the first to address the issues of steerable filters. Definition 3 is their definition of steerability. One example of steerable filters is the first derivative of a concentric Gaussian. The concentric Gaussian is written as

$$g_a(x,y) = e^{-(x^2+y^2)} \tag{7.3}$$

where its deviations are set to 1 for convenience. The first derivative of the Gaussian oriented at θ is

$$\frac{\delta}{\delta x} g_a^{\theta}(x,y) = \cos\theta \frac{\delta}{\delta x} g_a^{0}(x,y) + \sin\theta \frac{\delta}{\delta x} g_a^{\pi/2}(x,y) \quad . \tag{7.4}$$

Thus, the first Gaussian derivative is shiftable with $\theta_1 = 0$ and $\theta_2 = \pi/2$, and $q_1(\theta) = \cos\theta$ and $q_2(\theta) = \sin\theta$.

Freeman and Adelson provided two theorems. Each theorem describes a family of steerable functions. Their first theorem states that the functions are steerable if they can be expanded in a Fourier series in polar angle ϕ:

$$h(r,\phi) = \sum_{k=-N}^{N} G_k(r) e^{jk\phi} \quad , \tag{7.5}$$

where $r = \sqrt{x^2 + y^2}$ and $\phi = atan(y/x)$. The number of basis functions needed to steer this type of functions is the number of frequencies between $-N$ and N where the expansion has nonzero coefficients $a_k(r)$.

The other theorem states that the functions in the following form are steerable:

$$h(x,y) = W(r) P_N(x,y) \tag{7.6}$$

where $W(r)$ is an arbitrary concentric window and $P_N(x,y)$ is an N^{th} order polynomial in x and y whose coefficients can depend on r. At most $2N+1$ basis functions are enough to steer this type of function.

7.3.Deformable Kernel Approximation

In the deformable kernel decomposition, $h_\theta(x,y)$ is considered to be a kernel of a linear operator $\boldsymbol{L}$ mapping from (θ) to (x, y), written as

$$\boldsymbol{L}a(x,y) = b(\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_\theta(x,y) a(x,y) dx dy \; . \tag{7.7}$$

Similarly the adjoint of L denoted as L^* is

$$\boldsymbol{L}^* b(\theta) = a(x,y) = \int_{-\pi}^{\pi} h_\theta^*(x,y) b(\theta) d\theta \quad . \tag{7.8}$$

Now the SVD is equivalent to finding eigenfunctions $\tilde{a}_i(x,y)$ and $\tilde{b}_i(\theta)$ for L^*L and LL^* respectively. More specifically, the following equations have to be solved.

$$\boldsymbol{L}^*\boldsymbol{L}\tilde{a}_i(x,y) = \int_{-\pi}^{\pi} h_\theta^*(x,y) \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_\theta(\hat{x},\hat{y}) \tilde{a}_i(\hat{x},\hat{y}) d\hat{x} d\hat{y} d\theta = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \chi_\theta(x,\hat{x},y,\hat{y}) \tilde{a}_i(\hat{x},\hat{y}) d\hat{x} d\hat{y} = \lambda_i \tilde{a}_i(x,y) \tag{7.9}$$

where

$$\chi_\theta(x,\hat{x},y,\hat{y}) = \int_{-\pi}^{\pi} h_\theta^*(\hat{x},\hat{y}) h_\theta(x,y) d\theta \tag{7.10}$$

and

$$\boldsymbol{L}\boldsymbol{L}^* \tilde{b}_i(\theta) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_\theta(x,y) \int_{-\pi}^{\pi} h_{\hat{\theta}}^*(x,y) \tilde{b}_i(\hat{\theta}) d\hat{\theta} dx dy = \int_{-\pi}^{\pi} \kappa_{xy}(\theta,\hat{\theta}) \tilde{b}_i(\hat{\theta}) d\hat{\theta} = \lambda_i \tilde{b}_i(\theta) \tag{7.11}$$

where

$$\kappa_{xy}(\theta,\hat{\theta}) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h_{\hat{\theta}}^*(x,y) h_\theta(x,y) dx dy \quad . \tag{7.12}$$

Note that $\kappa_{xy}(\theta,\hat{\theta})$ is a function dependent on the difference $\theta - \hat{\theta}$. Thus, (7.11) becomes

$$\boldsymbol{LL}^{*}\tilde{b}_i(\theta) = \int_{-\pi}^{\pi} \kappa_{xy}(\theta - \hat{\theta})\tilde{b}_i(\hat{\theta})d\hat{\theta} = \lambda_i\tilde{b}_i(\theta) \ . \tag{7.13}$$

Taking the Fourier transform of (7.13) gives

$$K_{x,y}(\Theta)B(\Theta) = \lambda_i B(\Theta) \ , \tag{7.14}$$

where $K_{x,y}(\Theta)$ is the fourier transform of $\kappa_{x,y}(\theta)$. Non–trivial solutions of (7.14) are

$$\lambda_i = 2\pi\hat{h}_i \quad \text{and} \tag{7.15}$$

$$\mathrm{B}_i(\Theta) = 2\pi\delta(\Theta - i) \ , \text{or} \tag{7.16}$$

$$\tilde{b}_i(\theta) = e^{j\,i\theta} \quad \text{where,} \tag{7.17}$$

$\hat{h}_i$ is a Fourier series coefficient of a 2π periodic function $\kappa_{x,y}(\theta)$.

The solution for *a(x,y)* can be found by using the fact that $L^{*}\tilde{b}_i(\theta)$ is an eigenfunction of $L^{*}L$, since

$$L^{*}L(L^{*}\tilde{b}_i(\theta)) = L^{*}(LL^{*}\tilde{b}_i(\theta)) = L^{*}\lambda_i\tilde{b}_i(\theta) = \lambda_i(L^{*}\tilde{b}_i(\theta)) \ . \tag{7.18}$$

Thus,

$$\tilde{a}_i(x,y) = L^{*}b_i(\theta) = \int_{-\pi}^{\pi} h_{\theta}^{*}(x,y)e^{j\,i\,\theta}d\theta \ . \tag{7.19}$$

Q^{th} order DKA is accomplished by selecting $a_i(x,y)$ and $b_i(\theta)$ which corresponds to the set of Q largest $\hat{h}_i$. Thus the approximation formula is

$$h_{\theta}(x,y) \approx \sum_{i=0}^{Q} \hat{h}_i a_i(x,y)b_i(\theta) \tag{7.20}$$

where $\left\{\hat{h}_i\right\}$ is assumed to be sorted in a descending magnitude order.

7.4.Fourier Series Approximation

As seen in (7.17) and (7.19), the deformable kernel decomposition involves complex filters. This is true even when the filter to be approximated is real and does not have the imaginary part.

Since $\tilde{h}_{\theta}(x,y)$ is 2π periodic in θ, it can be decomposed using the Fourier series expansion as,

$$h_{\theta}(x,y) = \frac{1}{2}\alpha_0(x,y) + \sum_{n=1}^{\infty}\{\alpha_n(x,y)\cos(n\theta) + \beta_n(x,y)\sin(n\theta)\} \quad \text{where,} \tag{7.21}$$

$$\alpha_n(x,y) = \frac{1}{L}\int_{-\pi}^{\pi} h_{\theta}(x,y)\cos(n\theta)d\theta \quad \text{and} \tag{7.22}$$

$$\beta_n(x,y) = \frac{1}{L}\int_{-\pi}^{\pi} h_{\theta}(x,y)\sin(n\theta)d\theta \ . \tag{7.23}$$

Then a Q^{th} order approximation to the steerable system can be accomplished by selecting Q basis filters from $\alpha_i(x,y)$ and $\beta_i(x,y)$ with the Q largest energy values. The approximation formula is

$$h_{\theta}(x,y) \approx \sum_{i=1}^{Q} \gamma_i(x,y)q_i(\theta) \quad \text{where} \tag{7.24}$$

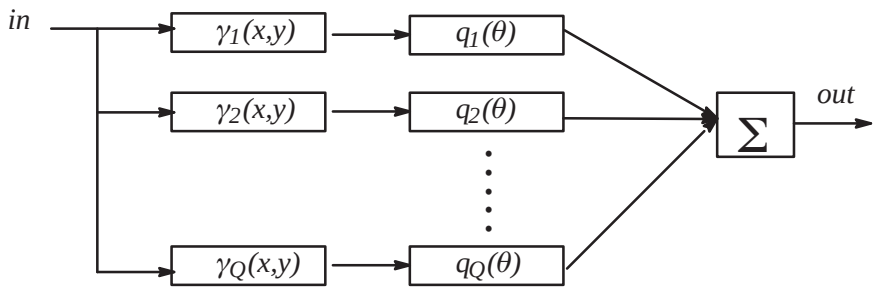


Figure 7–1: Structure of FSA

$$\{\gamma_i\} \subset \{\alpha_i, \beta_i\} \quad \text{and} \tag{7.25}$$

$$q_i(\theta) = \begin{cases} 1/2 & \text{if } \gamma_i = \alpha_0 \\ cos(n\theta) & \text{if } \gamma_i \in \{\alpha_i\},\ i \neq 0 \\ sin(n\theta) & \text{if } \gamma_i \in \{\beta_i\} \end{cases} . \tag{7.26}$$

Note that (7.19) is the complex Fourier series of $h_\theta{}^*(x,y)$ which is 2π periodical in θ. Thus,

$$a_i(x,y) = \alpha^*{}_i(x,y) + j\beta^*{}_i(x,y), \tag{7.27}$$

and FSA can produce the identical approximation result with DKA. The difference in the two approximation methods resides in their ways of selecting basis filters. In DKA, a complex filter $a_i(x,y)$ consists of two filters: a real part and an imaginary part, and these two filters are always selected as a pair for the approximation. In FSA, more flexibility is added in its filter selection, and $\alpha_i(x,y)$ and $\beta_i(x,y)$ which correspond respectively to the real and imaginary part of the complex filter $a_i(x,y)$ are separate entities in the filter selection process.

Since the decomposition process is equivalent to DKA and FSA is more flexible in its filter selection, FSA can achieve the same performance as DKA. The computational structure of FSA is shown in Figure 7–1.

7.5.FSA + SV/OSD

The basis filters $\gamma_i(x,y)$ in (7.24) are non–separable, in general, and the algorithm does not satisfy the implementation criteria. For the steerable system to satisfy the criteria, SV/OSD can be applied to $\{\gamma_i(x,y)\}$ for the separable approximation at the expense of additional error. The approximation matrix is formed by combining $\{\gamma_i(x,y)\}_{0 \le i < Q}$ for the Q^{th} order FSA. Using this combination of FSA and SV/OSD, denoted as FSA+SV/OSD, an orientational filter at an arbitrary orientation can be approximated as,

$$h_\theta(x,y) \approx \sum_{i=1}^{Q} q_i(\theta) \sum_{j=1}^{P} a_j(y) b_{ij}(x), \quad \text{where} \tag{7.28}$$

$$\gamma_i(x,y) \approx \sum_{j=1}^{P} a_j(y) b_{ij}(x) \ . \tag{7.29}$$

The computational structure of FSA + SV/OSD is shown in Figure 7–2. To implement a system with multiple orientations, the number of orientation control units (inside the dashed box in Figure 7–2) must equal the number of orientations are needed.

The throughput of the system can be $1/t_m$, the latency is $O(NM)$, the amount of computation is $N^2(MQP+F_N)$, and the storage requirement is NM.

7.6.FSA + SWA

The next extension for the steerable system is to add MRD capability. FSA and SWA can be combined in the following way. First, the filter of interest is decomposed with FSA. Second, each basis filter is decomposed into a separable form using

SV/OSD. Third, each separable pair is approximated using the Wavelet Approximation. The describes this approximation process is defined by

$$d^k_{m,n} = \sum_{c=1}^{Q} q_c(\theta) \sum_{l}^{P} \sum_{j_x} g^x_{cl}[j_x - m] \sum_{j_y} g^y_l[j_y - n]\, c^{k-1}_{j_x, j_y} \quad \text{, where} \tag{7.30}$$

$$c^k_{m,n} = \sum_{l} \sum_{j_x} \tilde{h}[j_x - 2m] \sum_{j_y} \tilde{h}[j_y - 2n] c^{k-1}_{j_x, j_y} \quad \text{, with} \tag{7.31}$$

$$c^0_{m,n} = \sum_{i_x, i_y} f[i_x,\ i_y]\, \phi_I(m - i_x)\, \phi_I(n - i_y) \quad . \tag{7.32}$$

Figure 7–3 shows the structure which corresponds to the approximation. The pre–filter and low–pass modules are the same as SWA. The high pass modules are divided into three parts: orthogonal filters, projection filters, and interpolation units. The orthogonal filters and projection filters approximate the FSA basis functions using SV/OSD. They are followed by the interpolation unit to steer the filter to a particular orientation. The orthogonal filters and projection filters can be shared among filters at different orientations. An independent interpolation unit is necessary for each orientation.

The throughput, latency, computational complexity, and storage requirement of decimated steerable MRD system using FSA+SWA can be derived in a similar way as decimated SWA as described in 5.3, and are $1/t_m$, $Nt_m(M_I+(2^{k-1}-1)M_h+2^{k-1}M) \approx 2^{L-1}N(M_h + M)t_m$,, $4N^2PQM/3$, and $N(M_I+max(M,M_h))$ respectively.

The idea of undecimated MRD discussed in 5.4 can be easily extended for steerability in a very similar way as the above decimated MRD scheme. Figure 7–4 shows the structure of the undecimated steerable MRD. The only difference between Figures 5–14 and 7–4 in terms of the hardware structure is the addition of the interpolation units which steer the filter. The lower portion of the MRD module in Figure 7–4 approximates a set of FSA basis functions, whereas the counterpart in Figure 5–14 approximates a set of orientational filters.

The throughput, latency, computational complexity, and storage requirement of an undecimated steerable MRD system using FSA+SWA can be derived in a similar way as undecimated SWA as described in 5.4, and are $1/t_m$, $2^{L-1}N(M_h + M)t_m$, $2N^2M_I+LN^2(2M_h+MP+MPQ)$, and $NM_I+LNmax(M,M_h)+ NL(PQ+1)$ respectively. It also requires $LQ(P+1)+1$ reorder buffer and L reorder buffer address generators.

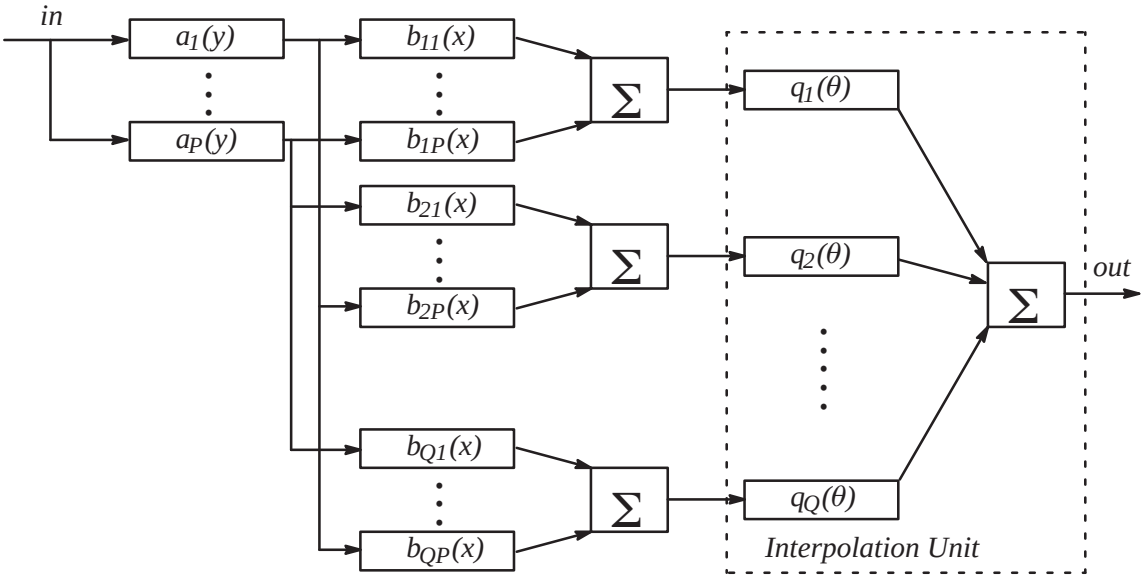
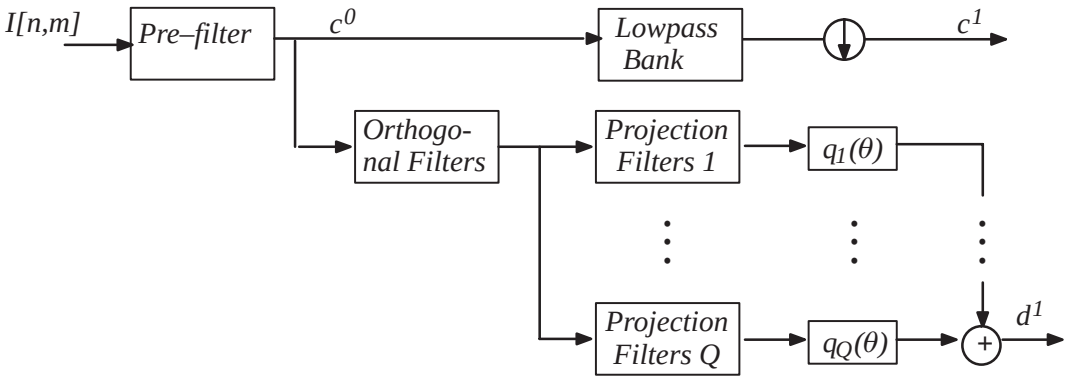
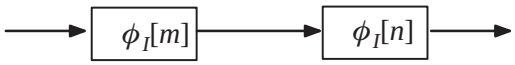


Figure 7–2: Structure of an SV/OSD Combined with FSA

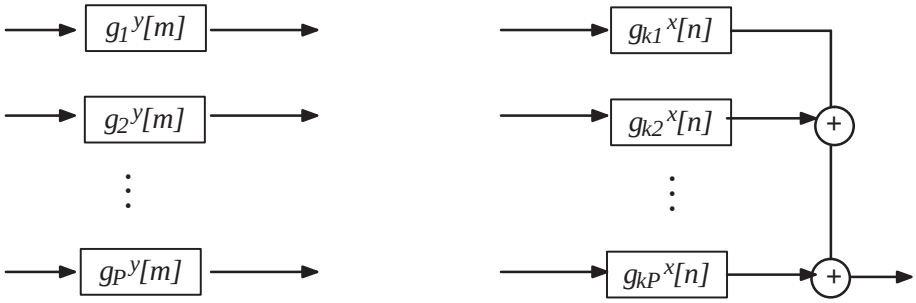


(a) Overall System



(b) Pre–filter Bank

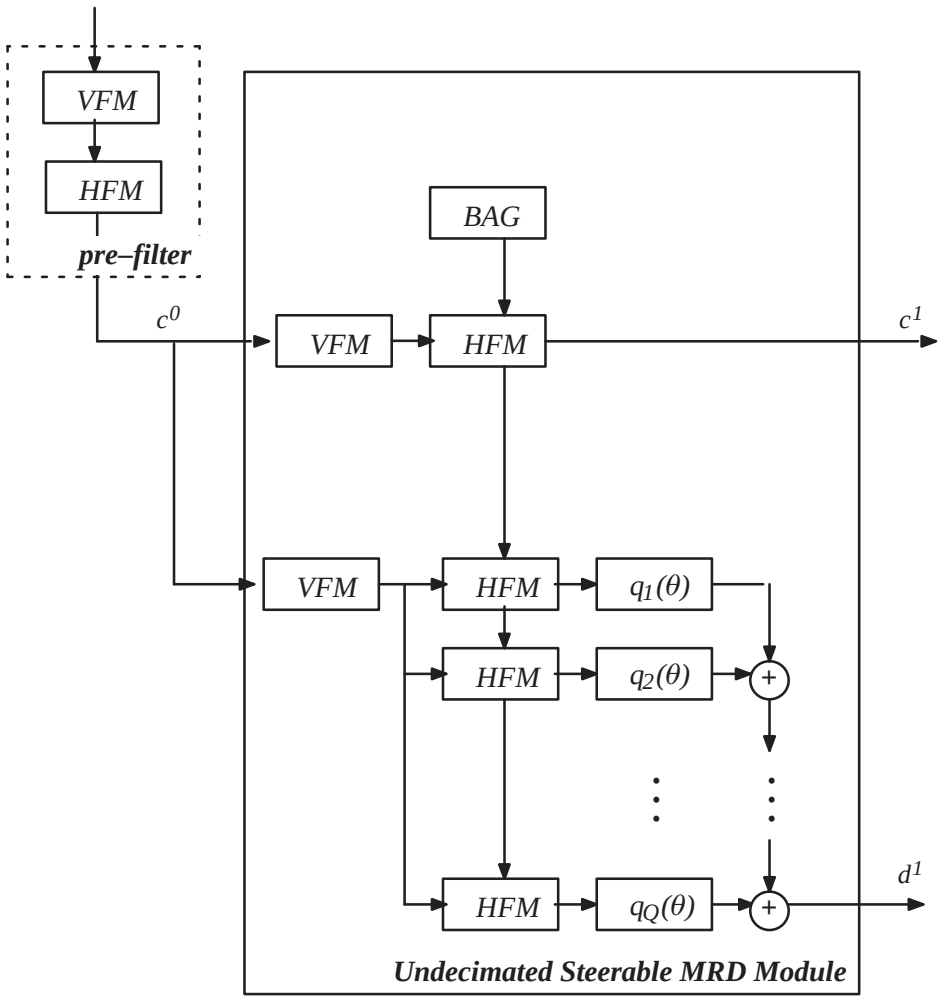
(c) Lowpass Filter Bank



(d) Orthogonal Filters

(e) Projection Filters *k*

Figure 7–3: Structure for FSA Combined with SWA



BAG = Reorder Buffer Address Generator

VFM = Vertical Filter Module

HFM = Horizontal Filter Module

(See Figure 5–13)

$q_k(\theta)$ *= Interpolation Unit*

The identical MRD Module can be cascaded for further MRD

A distinct set of interpolation units is needed for each orientation tuning.

Figure 7–4: Structure of Undecimated Steerable MRD

CHAPTER 8

FSA Performance Evaluation

8.1.Energy Error Analysis for FSA+SV/OSD

The behavior of the approximation error is studied in the spatial domain using Gabor filters with various parameter values. The real part and the imaginary part of the filters are investigated separately. The energy error (*L2* error) defined in (4.5) is used for the measurement. Four types of Gabor filters are used for this evaluation. Their parameter values are (α, σ_x, σ_y) = (10.0, 0.2, 0.4), (10.0, 0.15, 0.45), (20.0, 0.2, 0.4) and (20.0, 0.15, 0.45). The orientation of each filter is changed from 0 to π with 0.01 increments, and the energy error is computed at each orientation. The evaluation studies the approximation behavior by changing two approximation orders, FSA approximation order and SV/OSD approximation order. The results of the evaluation are shown in Figures 8–1 through 8–4. Each plot shows the maximum energy error for all the orientations of the filter tested.

Energy errors decreases almost exponentially as SV/OSD order decreases. In Figures 8–1 through 8–4, every plot shows that the maximum error does not decrease smoothly as the FSA order (noted as FA Order in the axis) increases, but rather decreases in steps. For example, in Figure 8–1(a), the maximum error does not change much from FSA order = 3 to FSA order = 4. It seems that the 4th basis filter for FSA does not do much to improve the approximation. However, the average error does decrease smoothly as the FSA order increases, and the 4th basis filter does contribute to the approximation performance.

Table 8–1 lists the results of the approximation at *(P,Q)=(5,5), (10,10), (15,15).* The approximation becomes more difficult as the frequency parameter α increases, and the aspect ratio σ_x/σ_y increases.

8.2.Frequency Distortion

The frequency domain error introduced by FSA is considered next. As an example, Figure 8–5 shows the frequency response of a Gabor filter (σ_x=0.2, σ_y=0.4, and α=10.0) at various orientations (θ=0, $\pi/4$, and $\pi/2$) and the frequency distortion introduced by the approximation. The peak of the distortion energy matches the peak in the frequency response, but the distortion is more widely spread than SV/OSD (Figure 4–9) and SWA (Figure 6–5). In all cases in Figure 8–5, the amount of energy content for the frequency response is normalized to 1.

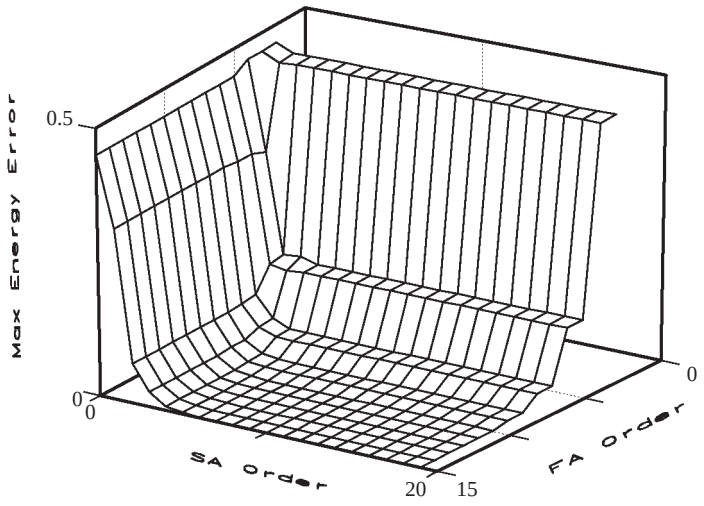
Table 8–1: Performance Evaluation: Numerical Results

This table shows the numerical values of the performance evaluation results shown in Figures 8–1 through 8–4 at *(P,Q)=(5,5), (10,10),* and *(15,15)*.

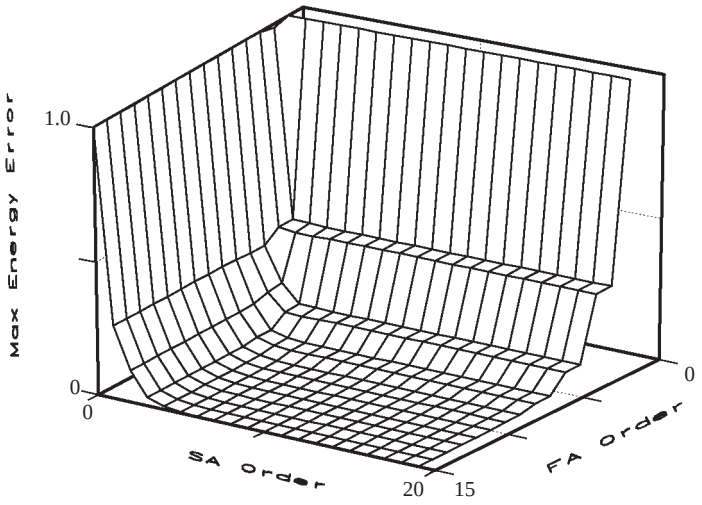
Gabor Parameters (α, σ_x, σ_y)		***P=5, Q=5***	***P=10, Q=10***	***P=15, Q=15***
(10.0, 0.2, 0.4)	Real	0.0299	0.00128	4.8e–5
	Imag	0.0750	7.86e–4	3.1e–5
(10.0, 0.15, 0.45)	Real	0.0587	0.00608	2.35e–4
	Imag	0.153	0.00680	7.55e–4
(20.0, 0.2, 0.4)	Real	0.203	0.320	0.0116
	Imag	0.300	0.193	0.00208
(20.0, 0.15, 0.45)	Real	0.260	0.0605	0.00550
	Imag	0.266	0.0432	0.00852

8.3.Energy Error Analysis for FSA+SWA

This section investigates the performance of FSA+SWA in terms of the normalized energy error using $g_b(\alpha, \sigma_x, \sigma_y)$ = (5.0, 0.25, 0.4). The filter is decomposed into a set of basis filters using FSA and the dilation of the basis filters are approximated



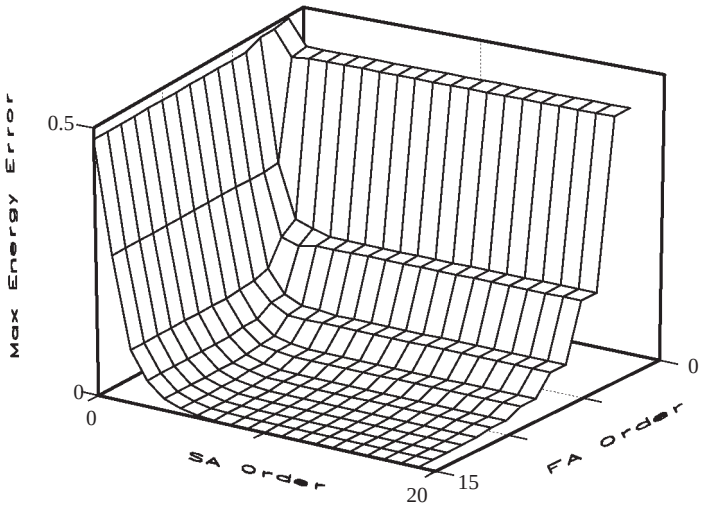
(a) Real Part



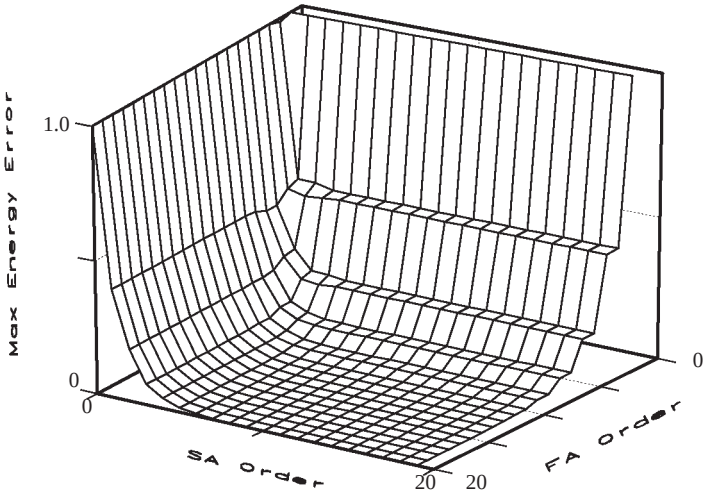
(b) Imaginary Part

Figure 8–1: Performance Evaluation Result for FSA (I)

Both the real and imaginary parts of Gabor filters g_b:$(\alpha, \sigma_x, \sigma_y)$ = $(10.0, 0.2, 0.4)$ with various orientations are approximated using FSA.

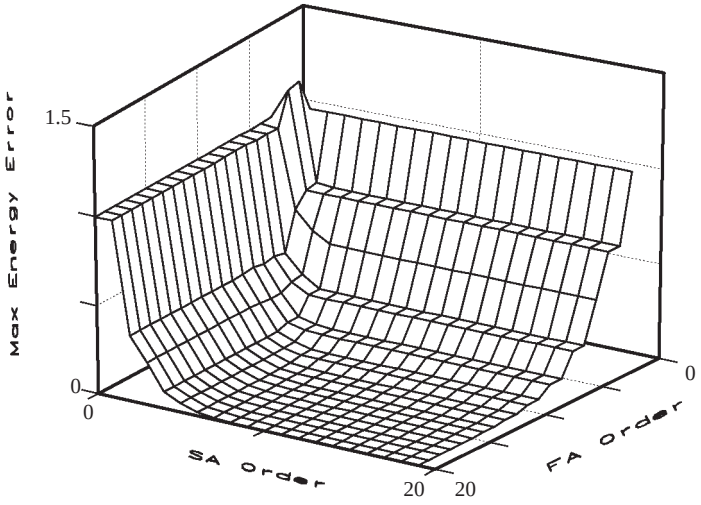


(a) Real Part

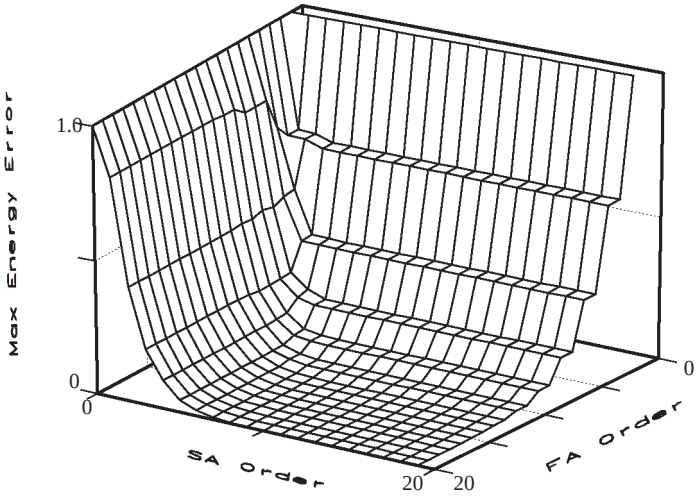


(b) Imaginary Part

Figure 8–2: Performance Evaluation Result for FSA (II)
Both the real and imaginary parts of Gabor filters g_b*:(α, σ_x, σ_y) = (10.0, 0.15, 0.45)* with various orientations are approximated using FSA.

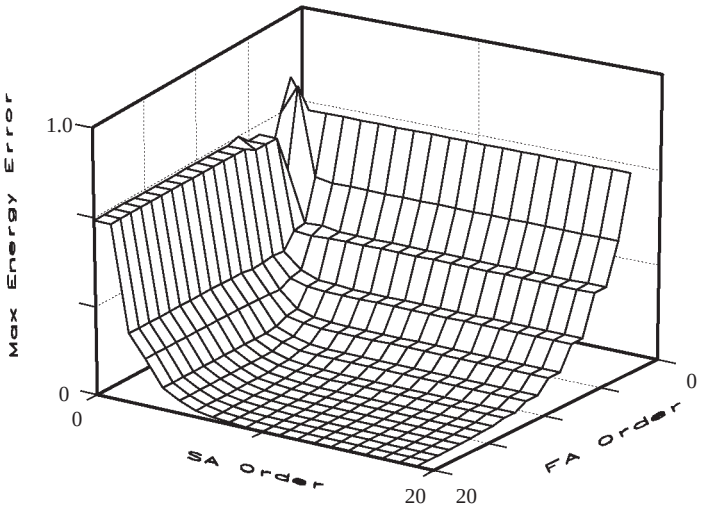


(a) Real Part

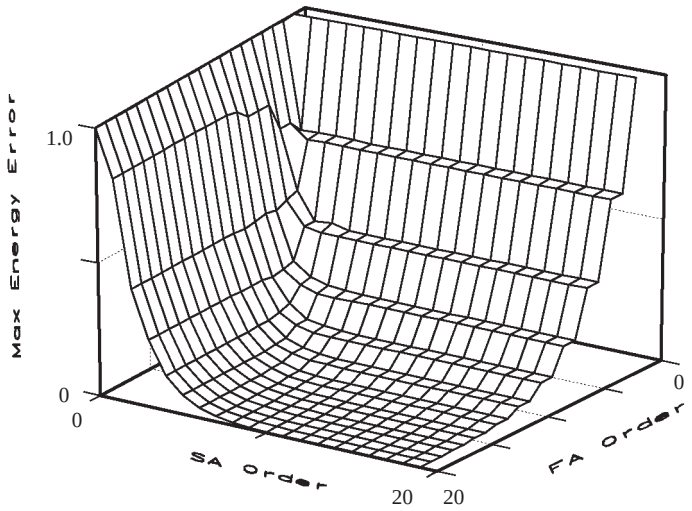


(b) Imaginary Part

Figure 8–3: Performance Evaluation Result for FSA (III)
Both the real and imaginary parts of Gabor filters $g_b:(\alpha, \sigma_x, \sigma_y) = (20.0, 0.2, 0.4)$ with various orientations are approximated using FSA.

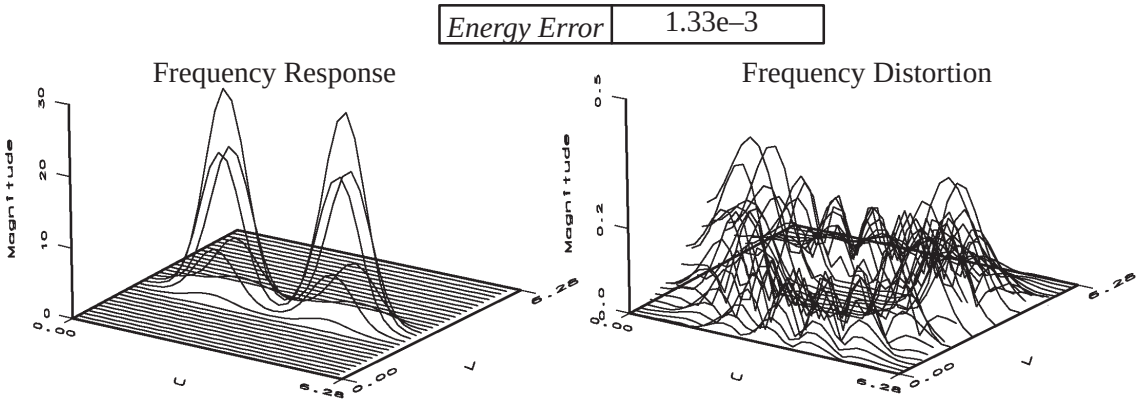


(a) Real Part

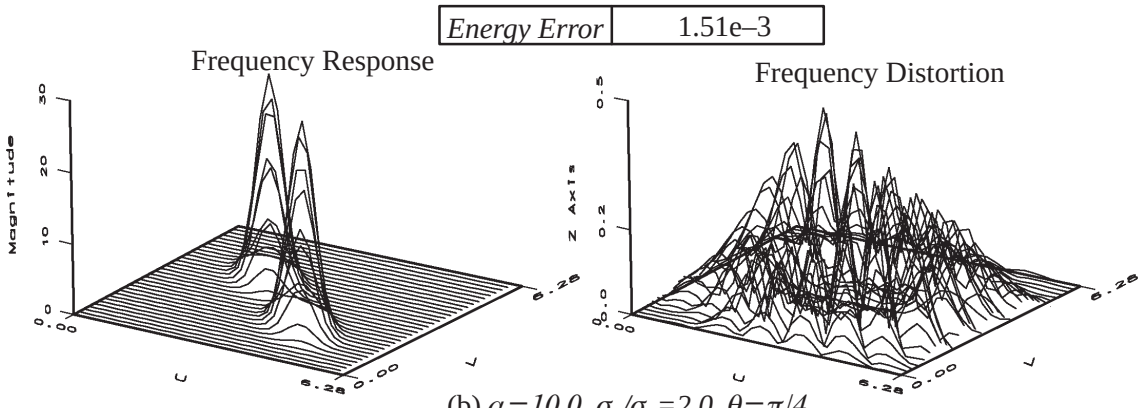


(b) Imaginary Part

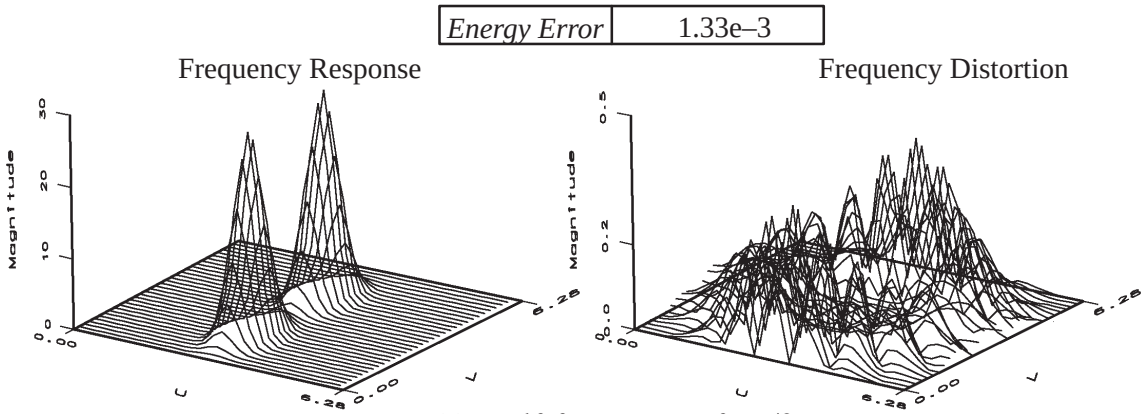
Figure 8–4: Performance Evaluation Result for FSA (IV)
Both the real and imaginary parts of Gabor filters g_b*:(α, σ_x, σ_y) = (20.0, 0.15, 0.45)* with various orientations are approximated using FSA.



(a) $\alpha=10.0,\ \sigma_y/\sigma_x=2.0,\ \theta=0$



(b) $\alpha=10.0,\ \sigma_y/\sigma_x=2.0,\ \theta=\pi/4$



(c) $\alpha=10.0,\ \sigma_y/\sigma_x=2.0,\ \theta=\pi/2$

Figure 8–5: Frequency Distortion of Approximation Filters

The real part of Gabor filters are approximated using FSA. The SV/OSD order is set to 8, and FSA order is set to 8.

with SWA. The SV/OSD order and the FSA order are incrementally changed from 3 to 11 and 3 to 15, respectively. The spline order is fixed at 7. Figures 8–6 and 8–7 show the maximum of the energy error among various orientations from 0 to π with 0.1 increment. The energy error decreases rapidly until the SV/OSD order (indicated by 'SA Order' in the plots) reaches 6 and the FSA order (indicated by 'FA Order') reaches 7. For these approximation orders, the maximum energy error is 1.87e–3 at all the decomposition levels (1 to 4) of the real part of the gabor filter, and 1.31e–3 at all the decomposition level of the imaginary part. The amount of energy error is visually constant as the decomposition level increases.

Table 8–2 lists the results of the approximation at *(P,Q)=(5,5),* and *(10,10).* Similar convergence behavior can be observed for other types of filters, however the knee of the plot may differ slightly.

Table 8–2: Performance Evaluation: Numerical Results

This table shows the numerical values of the performance evaluation results shown in Figures 8–1 through 8–4 at *(P,Q)=(5,5),* and *(10,10).*

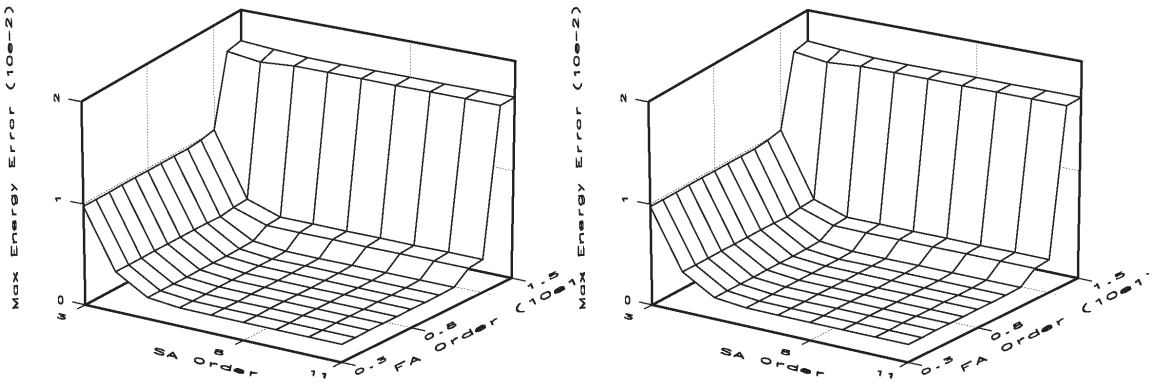
MRD Level		***P=5, Q=5***	***P=10, Q=10***
1	Real	0.00311	0.00179
	Imag	0.313	0.0198
4	Real	0.00311	0.00179
	Imag	0.0349	0.0215

8.4.Texture Segmentation with FSA+SWA

Jain's unsupervised texture segmentation is implemented using FSA+SWA. The approximation produces an undecimated steerable multi–resolution image pyramid using the schemes described in Section 7.6 and Section 5.4 Performance is compared between the case when the algorithm is applied to the original multi–resolution pyramid and the case when it is applied to the approximated pyramids.

The algorithm is the same as that described in Section 4.3.2 and the textured images are also the same as these used in Section 6.4.2 The number of decomposition levels is 7, and the number of orientations is 4. Gabor filters oriented at *0, $\pi/4$, $\pi/2$* and *$3\pi/4$* are used for the segmentation.

The results of the segmentation tests are shown in Figure 8–8. The results are all very similar between original pyramids and the approximated pyramids. The number of misclassified pixels differs (approximation – original) by –86, 171, –151, and 110.



(a) MRD Level=1

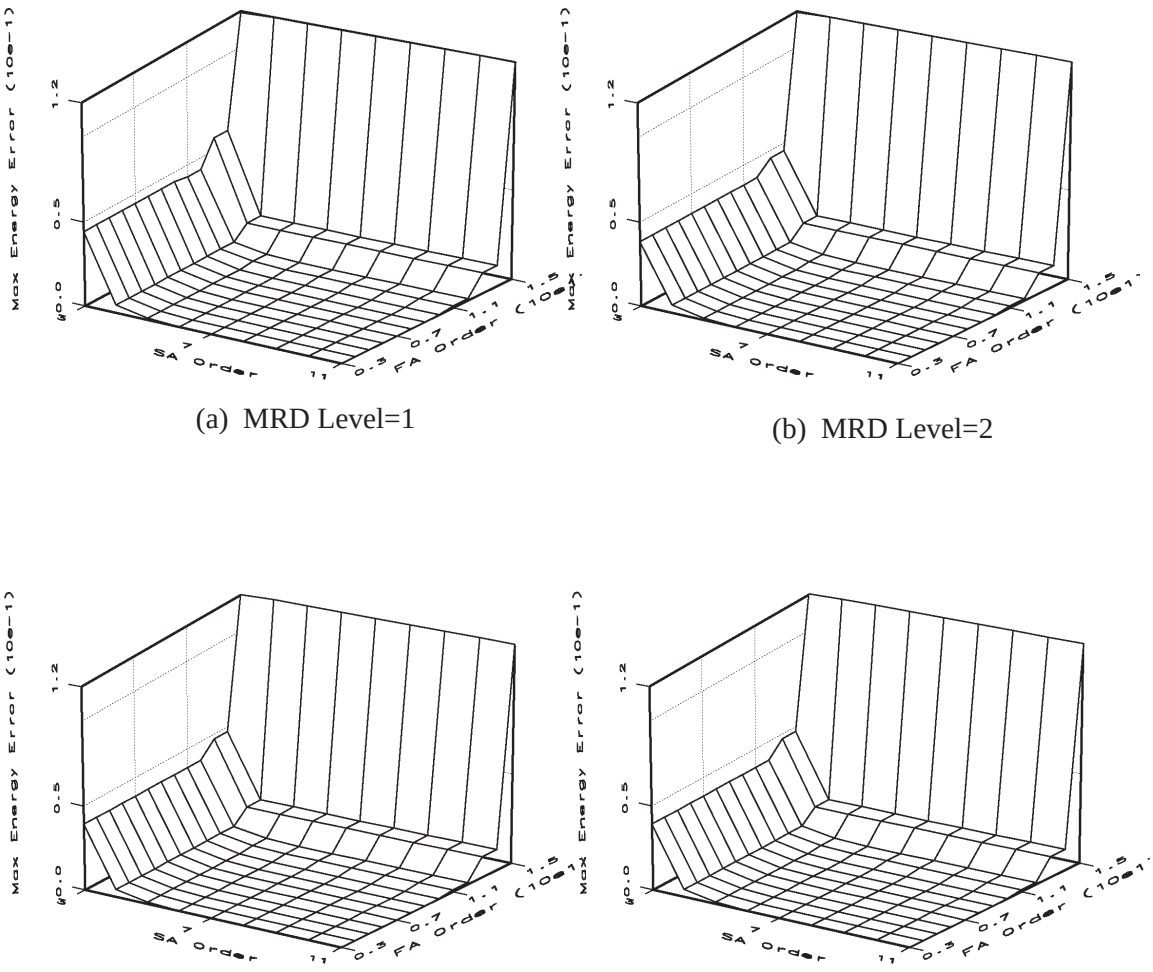
(b) MRD Level=2

(c) MRD Level=3

(d) MRD Level=4

Figure 8–6: Performance Evaluation Result for FSA+SWA (I)

The real part of Gabor filters $g_b(\alpha, \sigma_x, \sigma_y)=(5, 0.25, 0.4)$ is used for this evaluation. The filter size at the 0^{th} level is 11x11.



(a) MRD Level=1

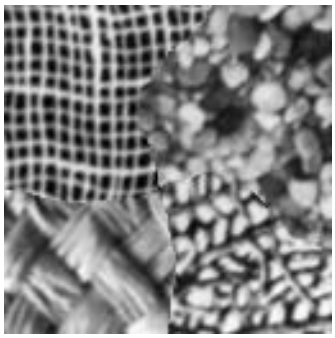
(b) MRD Level=2

(c) MRD Level=3

(d) MRD Level=4

Figure 8–7: Performance Evaluation Result for FSA+SWA (II)

The imaginary part of Gabor filters $g_b(\alpha, \sigma_x, \sigma_y)=(5, 0.25, 0.4)$ is used for this evaluation. The filter size at the 0^{th} level is 15x15.



Original Texture

Result using Gabor filters

Results using FSA, WA and SV/OSD

(a) Texture I

The approximated filters are obtained with a Wavelet Approximation (spline order = 5), Separable Approximation (order = 6) and Fourier Series Approximation (the order = 6). The result using Gabor filters has 1211 misclassified pixels whereas the result using the approximated filters has 1056 misclassified pixels.

Original Texture

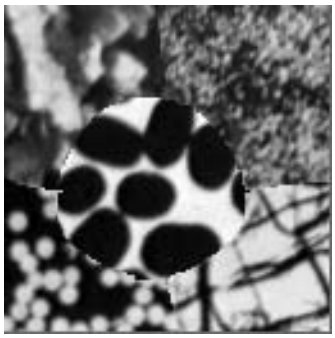
Result using Gabor filters

Results using FSA, WA and SV/OSD

(b) Texture II

The approximated filters are obtained with a Wavelet Approximation (spline order = 6), Separable Approximation (order = 6), and Fourier Series Approximation (the order = 6). The result using Gabor filters has 3315 misclassified pixels whereas the result using the approximated filters has 3425 misclassified pixels.

Figure 8–8: The result of Texture Segmentation



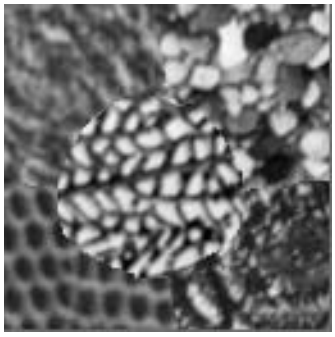
Original Texture

Result using Gabor filters

Results using Approximated filters

(c) Texture III

The approximated filters are obtained with a Wavelet Approximation (spline order = 5) and Separable Approximation (order = 5), and Fourier Series Approximation (the order = 5). The result using Gabor filters has 2435 misclassified pixels whereas the result using the approximated filters has 2349 misclassified pixels.



Original Texture

Result using Gabor filters

Results using Approximated filters

(d) Texture IV

The approximated filters are obtained with a Wavelet Approximation (spline order = 9) and Separable Approximation (order = 6), and Fourier Series Approximation (the order = 6). The result using Gabor filters has 1437 misclassified pixels whereas the result using the approximated filters has 1608 misclassified pixels.

Figure 8–8: The result of Texture Segmentation

CHAPTER 9

Hardware Design

This chapter discusses detailed hardware design of a set of VLSI chips which implements SV/OSD, SWA and FSA+SWA. The design has to satisfy the implementation criteria. Another design is *scalability.* Four types of scalability are considered: input image size, the number of orientational filters, filter size, and approximation order. A description of scalability is given in Chapter 1. First, the design for SV/OSD is considered. The main component of the design is the 2D separable filter. It is divided into a *vertical filter unit* and a *horizontal filter unit*. Second, the design for SV/OSD is adapted for SWA. Again, the 2D separable filter is the main functional component of the design. Third, the design for undecimated SWA will be discussed. Fourth, it will be shown that a modification on the design for SWA can implement FSA+SWA.

Due to a lack of VLSI design tools, the size of the design is estimated by counting the number of major components in the design. The types of components which influences the size of the chips are multiplier, adder, D flip–flop, global bus, memory, multiplexer, and register file. At each section, the number of each components is counted. Also the number of input/output ports external to the chip is counted so that an estimated number of I/O pads can be calculated. Only those carrying the filter inputs/outputs or the filter coefficients are counted since most I/O pads will be taken up by them, and obtaining a more accurate pin counts involves very detailed design with power dissipation analysis. This is beyond the scope of this thesis. The core of the design is the sequential/parallel multiply–accumulate operation. By placing D flip–flops at the output of every multiply–adder and binary adder, the chips can operate as fast as $1/t_m$ Hz assuming one multiply–accumulate operation takes more time than a register access, propagation in a PAL, multiplexing, and routing delay.

In the following designs, it is assumed that a memory chip allows simultaneous write and read operations in one clock cycle. This can be done by either using dual port RAM, using two separate memory chips for read and write, using a video RAM for random writes and sequential reads, or using fast RAM so that on one phase (high clock) the memory can be read and on the other phase (low clock) it can be written.

- Notation

The width of the input pixels to the system is denoted as B_P.

Many signal names in the design have numeric post–fixes, such as *hout1* and *vout2*. Within the text, the notation *<signal_prefix>$$* is used to refer to the set of signals with the same prefix but different numeric post–fixes. For example, *hout$$* implies *hout1*, *hout2*, *hout3*, and so on.

Signals internal to the VLSI chip are indicated with a single underline. All signals without an underline are external to the VLSI chip.

Figure 9–1 shows the symbols used in later figures to represent certain components in the designs.

9.1.SV/OSD

The implementation structure of SV/OSD is shown in Figure 3–5(b). The structure can be divided into three modules: input buffer, vertical filters and horizontal filters. The vertical filters and horizontal filters are to be designed in separate VLSI chips, since only one set of vertical filters is needed for the whole system (for single stage filtering), whereas each orientational filter needs its own set of horizontal filters.

9.1.1.Input Buffer

An intermediate memory module is used to provide M parallel inputs to the vertical filters. The memory module consists of M independent memory banks whose size is N words each. One memory module is needed for all the orientational filters. A separate intermediate memory module is needed at each level of MRD.

- Components Count

The memory module requires M memory chips of N words each.

9.1.2.Vertical Filter Chip (VFC)

As explained in Section 3.5, a parallel filtering scheme is suitable for vertical filtering. Each filter requires *M* multipliers and *M–1* adders as shown in Figure 3–4(b). Assume W_{FV} is the width of a filter implemented in a VFC, and N_{FV} is the number of filters implemented in the chip. The chip requires $B_P W_{FV}$ input ports where input pixels are received in parallel. It also requires $2B_P N_{FV}$ outputs ports where the output of N_{FV} vertical filters are sent to the horizontal filter chip. The pixel width is doubled after multiplication to preserve accuracy. Each filter in the chip requires a register file of $B_P W_{FV}$ bits where the filter coefficients are stored. The coefficients for each multiplication do not change while the input is coming from the same row, but change when the input moves to the next row since it is easier to update the coefficients at every new row than to change the parallel input pattern from the memory banks to the chip. The secondary register file is provided to enable a smooth transition from one row to the next row. Its contents are updated with a new set of filter coefficients while the filter coefficients are read from the primary register file. At the end of the row, the contents of the secondary register file are transferred to the primary register file, and filtering for the next row can proceed continuously without any interruption. Either the host or a local_controller which resides on the same board with the filter chip can be used to store a new set of coefficients to the secondary register file. The structure of the VFC is shown in Figure 9–2.

An extra adder at the end of each parallel filter is used to add the output of the filter (*sum$$*) and a partial result (*vpout$$*) which is provided externally. This adder is needed when the size of the filter is larger than W_{FV}.

- Component Count

Table 9–1 lists the major components in a VFC and their counts.

Table 9–1 Component Count of the Vertical Filter Chip

Component Type	Count	Size (bits)
Multiplier	$W_{FV}N_{FV}$	$B_P x B_P$
Adder	$W_{FV}N_{FV}$	$2B_P$
D flip–flop	$N_{FV}W_{FV}$	$2B_P$
Register File	$2W_{FV}$	B_P
Input Ports	$(W_{FV}+2N_{FV}+1)B_P$	1
Output Ports	$2N_{FV}B_P$	1
Global Bus	$W_{FV}+1$	B_P

9.1.3.Horizontal Filter Chip (HFC)

The horizontal filter unit employs a pipelined filtering scheme. Each filter requires *M* multipliers and *M–1* adders as shown in Figure 3–4(a). There is no need for a buffer between a vertical filter and horizontal filter pair. The output of the vertical filter can be directly fed to the HFC. Outputs of the horizontal filters are added together by a binary adder to form the linear sum of the separable approximation. The structure of the HFC is shown in Figure 9–3.

Assume W_{FH} is the width of a filter implemented in a HFC, and N_{FH} is the number of filters implemented in the chip. Note that the number of bits in the output of vertical filtering is $2B_P$ to preserve accuracy. The chip requires $2B_P N_{FH}$ input ports

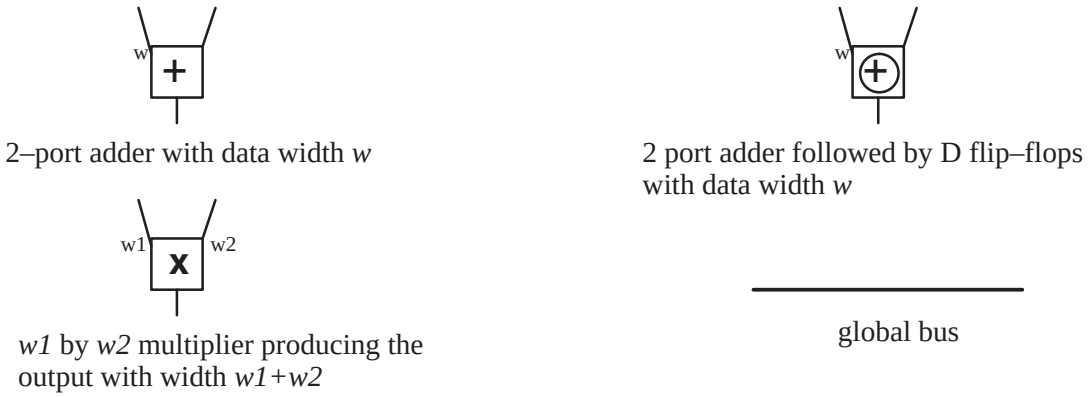


Figure 9–1: Hardware Symbol Representation

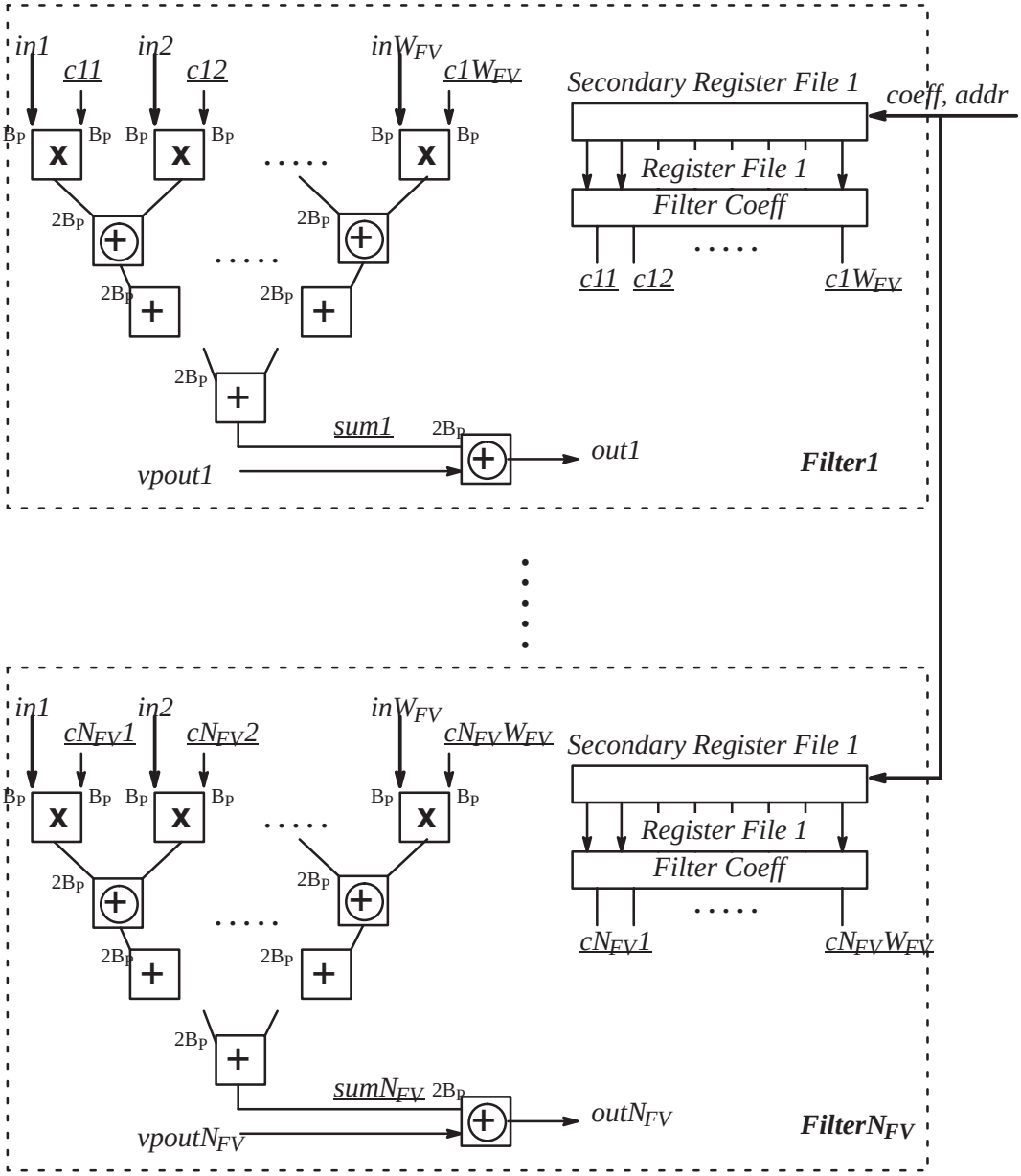


Figure 9–2: Structure of the Vertical Filter Chip

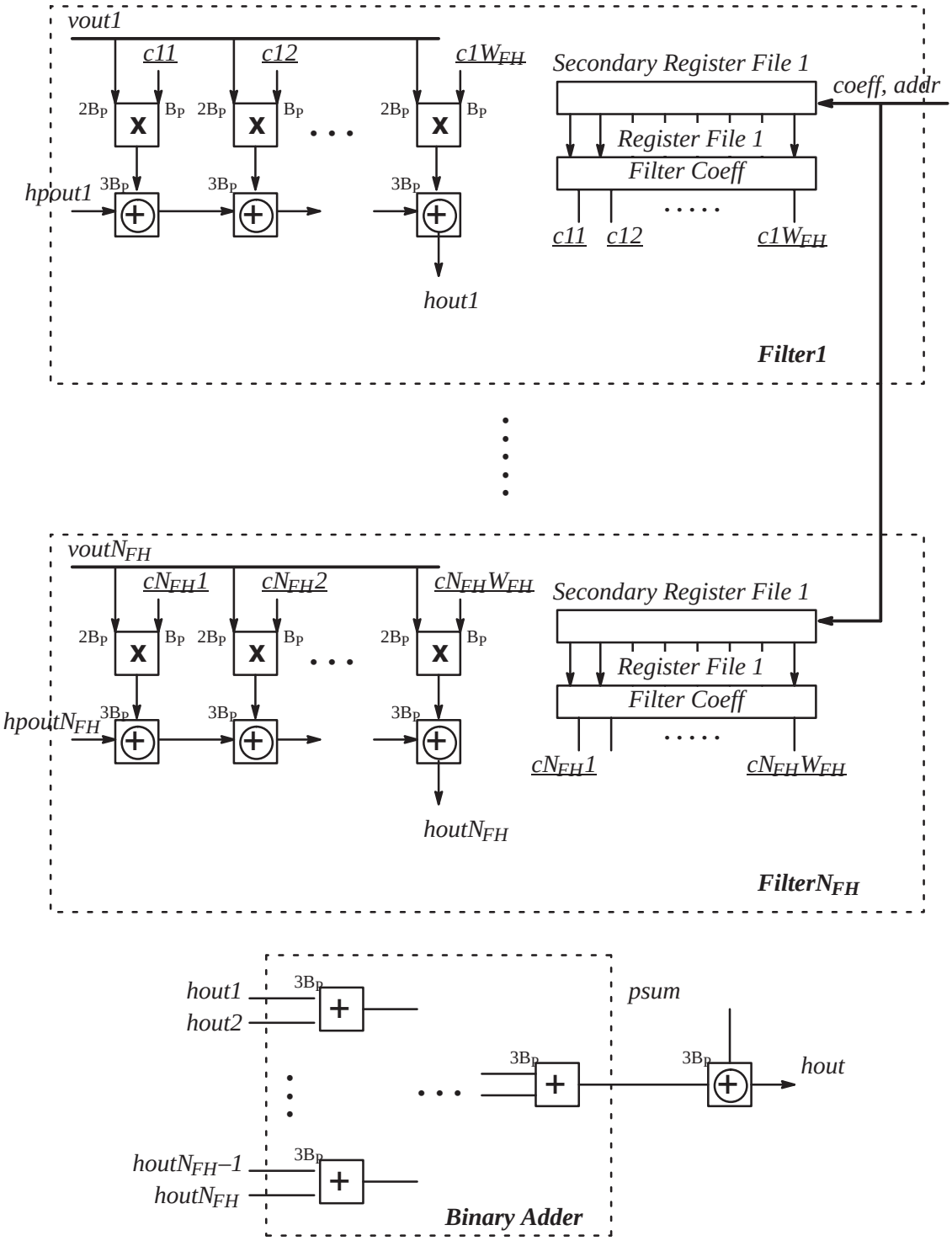


Figure 9–3: Structure of the Horizontal Filter Chip

where input pixels are received sequentially for N_{FH} filters. It also requires B_P input ports (*coeff*) for loading filter coefficients, and $3B_PN_{FH}$ input ports (*hpout$$*) for a partial result of the horizontal filtering, $3B_P$ input ports (*psum*) for a partial result of the linear sum for the separable approximation. The ports *hpout$$* are used when the filter size is larger than W_{FH}, and the ports *psum* is used when approximation order is larger than N_{FH} (See 9.1.4). It requires $3B_P$ output ports for the result from the binary adder, and $3N_{FH}B_P$ output ports for the partial results from each filter before the binary adder. The latter outputs are used together with *hpout$$* when the filter size is larger than W_{FH} (See 9.1.4). The secondary register files are provided for updating the filter coefficients without an interruption to the filtering. Unlike vertical filtering, the coefficients are fixed as long as the orientational filter being approximated is the same.

- Component Count

Table 9–2 lists the major components in a HFC and shows their sizes and counts.

Table 9–2 Component Count of the Horizontal Filter Chip

Component Type	Count	Size (bits)
Multiplier	$W_{FH}N_{FH}$	$2B_PxB_P$
Adder	$W_{FH}N_{FH}$	$3B_P$
D flip–flop	$N_{FH}W_{FH}+1$	$3B_P$
Register File	$2W_{FH}$	B_P
Input Ports	$(5N_{FH}+4)B_P$	1
Output Ports	$(3N_{FH}+3)B_P$	1
Global Bus	$N_{FH}+1$	$2B_P$

9.1.4.Architecture

Figure 9–4 shows the VLSI architecture of SV/OSD with F_N orientational filters being implemented. Each box corresponds to one VLSI chip. Every HFC obtains the same *vout* inputs from VFC, which may not be clear from the figure.

Only the input buffer module is affected by the input image size. Thus, the architecture satisfies the first scalability. The number of orientational filters implemented can be increased by adding more HFC module as shown in Figure 9–4. Thus, the second scalability is satisfied. When the filter size M is larger than W_{FV} and W_{FH}, additional VFCs and HFCs can be connected to as shown in Figure 9–5. Each VFC obtains inputs (*in$$*) from distinct memory banks. The outputs of the first VFC are connected to the input (*vpout$$*) of the next VFC. The last VFC of the sequence produces the results of the vertical filtering which are passed to HFCs in the same way as in Figure 9–4. Multiple HFC are connected in cascade. Every HFC obtains the same *vout$$* from the last VFC. Each HFC except the last one produces partial results from each filter (*hout$$*), which are connected to *hpout$$* of the next HFC in the chain. The last HFC produces the complete result of the separable approximation. The architecture satisfies the third scalability criterion. When the approximation order P is larger than N_{FV} and N_{FH}, additional VFCs

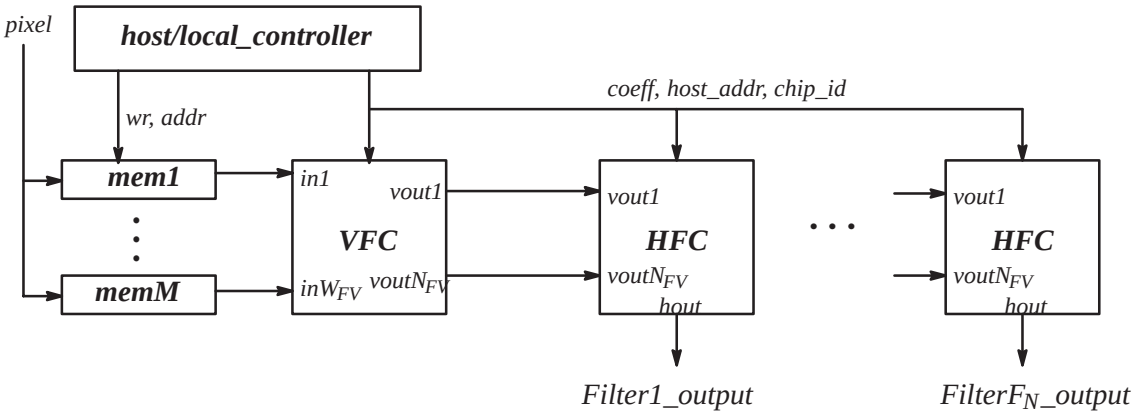
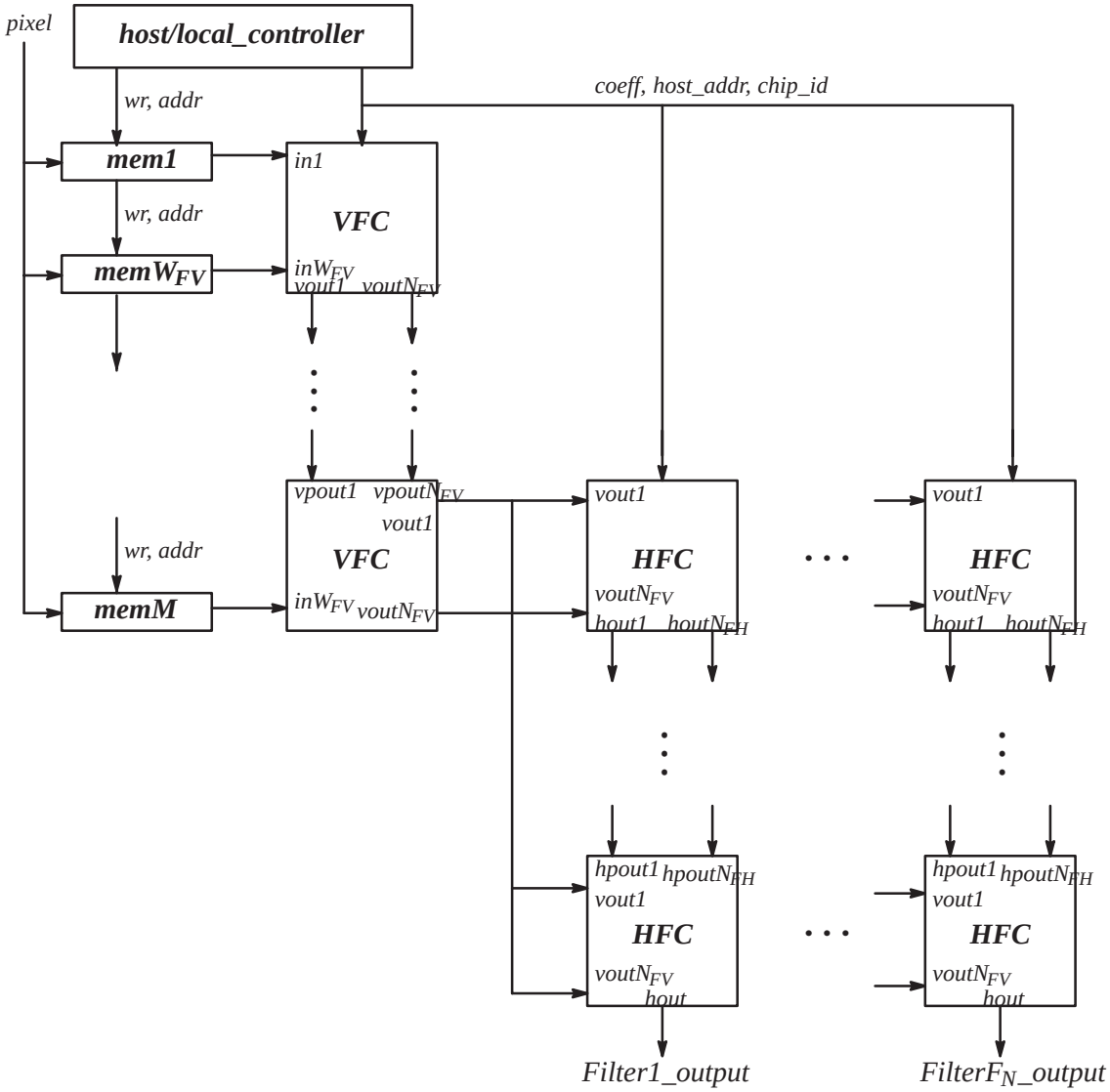


Figure 9–4: VLSI Architecture of SV/OSD



Note: *coeff, host_addr* and *chip_id* signals from the host are routed to every *VFC* and *HFC* although the above figure does not show the connections to some of the chips.

Figure 9–5: VLSI Architecture of SV/OSD with $M > W_{FV}$ and $M > W_{FH}$

and HFCs can be connected to existing VFCs and HFCs as shown in Figure 9–6. Every VFC receives the same set of inputs from the memory module. There is no interconnection among VFCs. The output of a VFC (*vout$$*) is connected to *vout$$* of the corresponding HFC. In the HFC chain, the output *hout* is connected to *psum* of the next HFC, if any. The last HFC produce the complete result of the separable approximation. The architecture satisfies the fourth scalability criterion.

9.2.SWA (Decimated MRD)

The structure of SWA is shown in Figure 5–5. It can be divided into three modules: pre–filter, low–pass filter, and high–pass filter modules. Both the pre–filter and low–pass filter are separable filters, and each of them can be implemented with a single VLSI chip. The high–pass filters are implemented using SV/OSD.

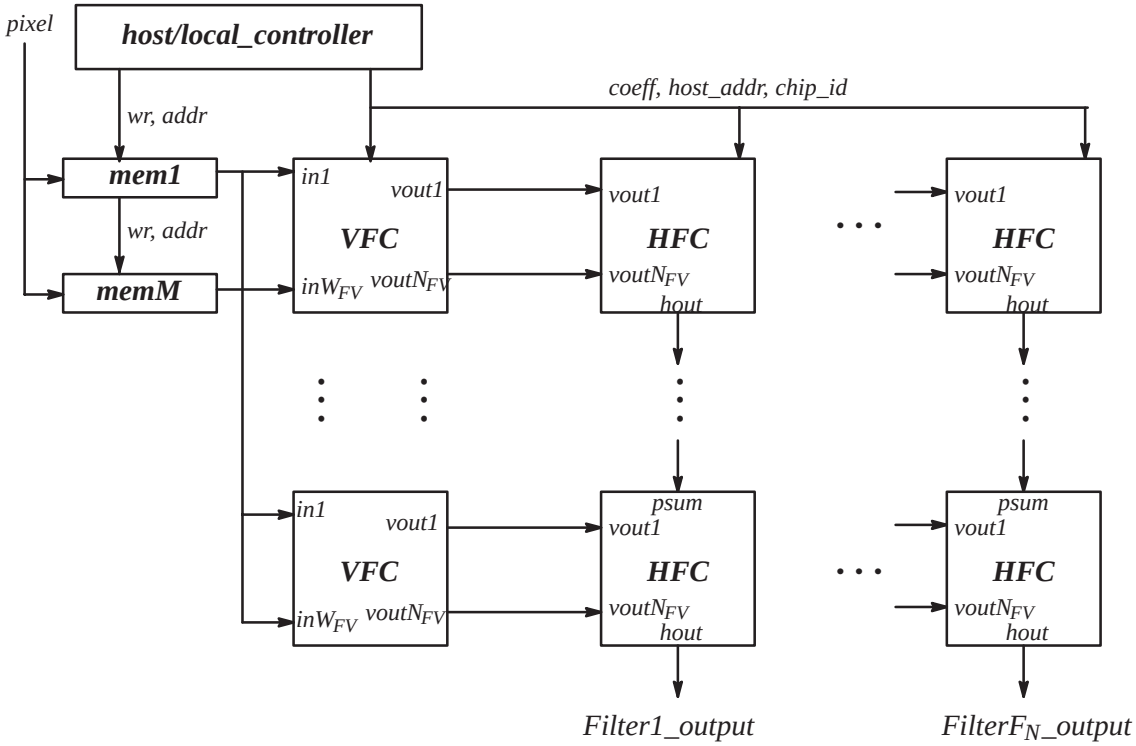
9.2.1.Pre–filter Unit

The pre–filter is a separable filter. Note that the size of the pre–filter is *k x k* when a k^{rh} order basic spline is used for the approximation, and the spline order is typically set to 7. Assume W_{FS} is the width/height of the filter implemented in the separable filter chip (SFC) shown in Figure 9–7. The chip contains $2W_{FS}$ multipliers and $2W_{FS}–2$ adders. The result of the vertical filter (*vout*) is an external output and the input to the horizontal filter (*muxout*) can be either *vout* or the internal signal, *hin*. This extra multiplexing is needed for undecimated MRD. For the decimated MRD, the select signal for the multiplexer (*decimate*) is 1, and the input to the horizontal filter is the immediate output from the vertical filter. For the undecimated MRD, the select signal for the multiplexer (*decimate*) is 0, and the input to the horizontal filter is the external input, *hin*, which is supplied from a reorder buffer. Section 9.3 describes the architecture of the undecimated MRD in detail.

It requires the following input ports: $B_P W_{FS}$ for parallel input pixels (*in$$*), $2B_P$ for *hin*, and B_P for filter coefficients (*coeff*). It requires the following output ports: $2B_P$ for *vout* and $3B_P$ for the filter output (*out*).

- Component Count

Table 9–3 lists the major components in a SFC and shows their sizes and counts.



Note: *coeff, host_addr* and *chip_id* signals from the host are routed to every *VFC* and *HFC* although the above figure does not show the connections to some of the chips.

Figure 9–6: VLSI Architecture of SV/OSD with $P > N_{FV}$ and $P > N_{FH}$

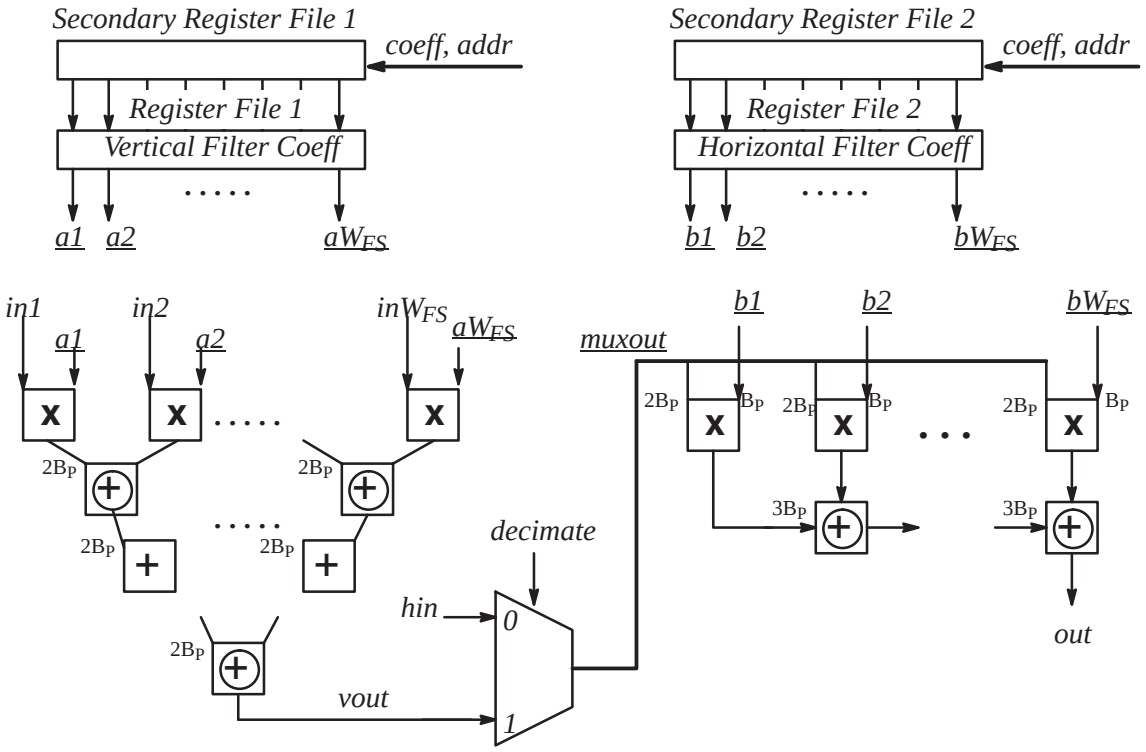


Figure 9–7: 2D Separable Filter Chip for Pre–filter and Low–pass Filter in SWA

9.2.2.Low–pass Filter Unit

The SFC shown in Figure 9–7 can be used for low–pass filtering. The size of the low–pass filter is *k+2 x k+2* for a k^{th} order basic spline. Thus, the size of the low–pass filter is a little larger than the pre–filter. A typical filter length for the low–pass filter is *9x9*.

9.2.3.High–pass Filter Unit

The high–pass filters are implemented with SV/OSD. The structure shown in Figure 9–4 can be used.

Table 9–3 Component Count of the Separable Filter Chip

Component Type	Count	Size (bits)
Multiplier I	W_{FS}	B_P x B_P
Multiplier II	W_{FS}	$2B_P$ x B_P
Adder I	$W_{FS}-1$	$2B_P$
Adder II	$W_{FS}-1$	$3B_P$
D flip–flop I	W_{FS}	$2B_P$
D flip–flop II	$W_{FS}-1$	$3B_P$
multiplexer	1	$2B_P$
Register File	$4W_{FS}$	B_P
Input Ports	$(W_{FS}+3)B_P$	1
Output Ports	$5B_P$	1
Global Bus	$3B_P$	1

9.2.4.Architecture

Figure 9–8 shows a VLSI architecture to implement SWA for decimated MRD.

The size of each memory bank at the k^{th} level of decomposition is $N/2^{k-1}$ due to a decimation of the output. The host/local_controller provides *write* signal to each memory bank. The decimation at the k^{th} stage can be done by providing the a *write* signal to memory banks so that only every other row of outputs are stored in which only every other column is stored. No decimation is done after the pre–filtering.

Only the memory chips have to be replaced when the input image size becomes large. Thus, the first scalability is satisfied. The number of orientational filters implemented can be increased by adding more HFCs in the high–pass filter module. Thus, the second scalability is satisfied. The sizes of the pre–filter and low–pass filter are independent of the size of the orientational filters and are dependent on the order of the basic spline used in the approximation. The size of the high–pass filters is the same as the size of the orientational filters. The third and fourth types of scalability can be accomplished by employing the scheme shown in Figure 9–5 and Figure 9–6 respectively for the high–pass filter module.

9.3.Undecimated SWA

9.3.1.Input Buffer

At the k^{th} level decomposition, the total amount of the input buffer increases due to dilation of filters to $2^{k-1}MN$ which is divided into M memory banks each having a size of $2^{k-1}N$. Consider each memory bank as a 2D memory array with the number of columns being N and the number of rows being 2^{k-1}. For the first N cycles, each memory bank outputs the data from the first row. For the second N cycles, it outputs the data from the second row. After $2^{k-1}N$ cycles, the memory access returns to the first row. This way of dividing the memory buffer removes the $M\ 2^{k-1}$ way multiplexers shown in Figure 5–12. The input pixel is written to the input buffer at a consecutive location in the first memory bank for $2^{k-1}N$ cycles, and the access moves to the next bank, and so on. After $2^{k-1}MN$ cycles, the access returns to the first memory bank. The host or the local_controller has to provide proper addresses to each memory bank.

9.3.2.Reorder Buffer

As described in Section 5.4, a reorder buffer stores one row of data after a vertical filtering. The reorder buffer and a reorder buffer address generator provides a sequence of data to horizontal filters. The reordering scheme is discussed in Section 5.4 and depicted in Figure 5–11.

9.3.3.Architecture

Figure 9–9 shows the VLSI architecture for undecimated SWA. The host/local_controller generates all the addresses to the reorder buffers as well as the input buffers instead of having a separate address generator as shown in Figure 5–13. The reorder buffers in the k^{th} level of the high–pass filters can be shared among the horizontal filters in the level. Note that in a

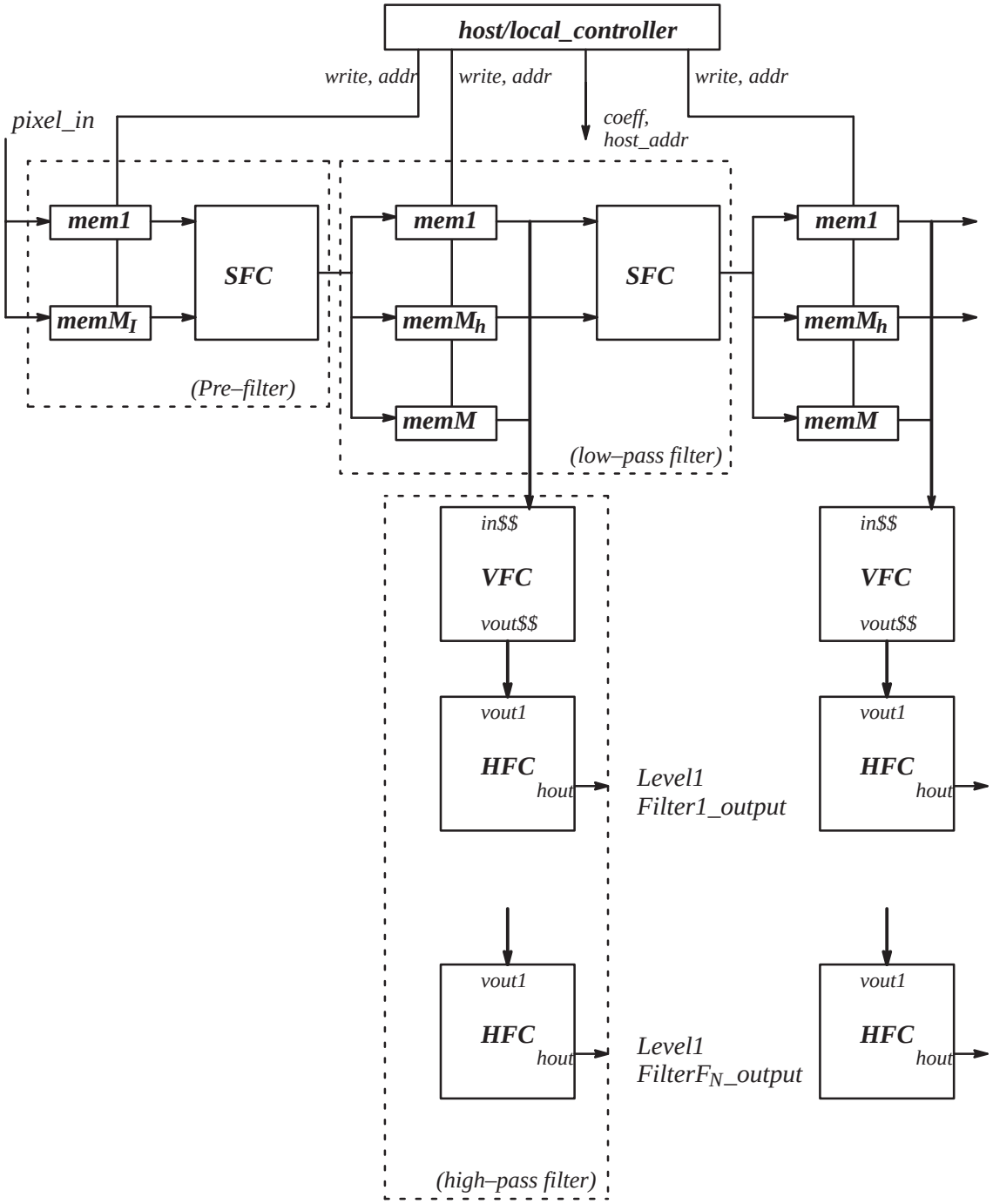


Figure 9–8: VLSI Architecture for Decimated SWA

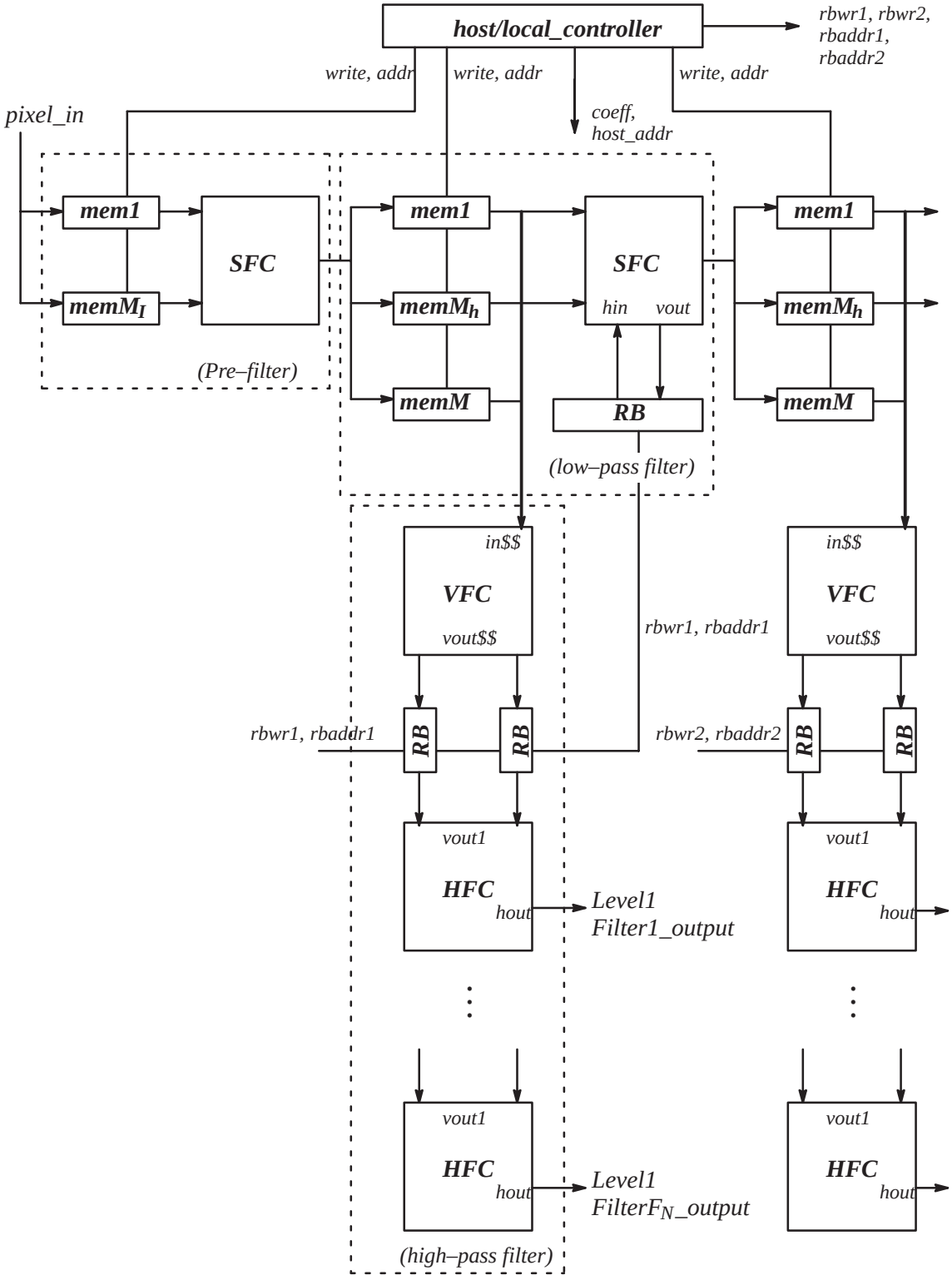


Figure 9–9: VLSI Architecture for Undecimated SWA

low–pass filter module, the SFC provides an input to the reorder buffer, and the output of the reorder buffer is fed back to the SFC for horizontal filtering. An input signal, *decimate*, at the SFC has to be 0 for this configuration. The architecture satisfies all the scalabilities.

9.4.FSA+SV/OSD

FSA+SV/OSD can be implemented with a modification to the architecture for SV/OSD shown in Figure 9–4. The interpolation module has to be designed and placed after the horizontal filters. The module interpolates the outputs from the basis filters to steer the filter. The module requires *Q* multipliers and *Q–1* adders assuming the order of FSA is *Q*.

9.4.1.Interpolation Unit

An interpolation unit takes an output of every basis filter as an input every cycle, and multiplies the output with an interpolation coefficient. The results of the multiplications are added together at a binary adder. Figure 7–2 depicts this computational scheme. It requires multiple interpolation units when the system implements multiple steerable filters.

It can be seen from Figure 9–2 that the VFC can be used to implement interpolation units. The parallel inputs (*in$$*) come from the set of basis filters, the register file contains the interpolation coefficients, and each unit in VFC is assigned to steer the orientational filter to a certain orientation.

9.4.2.Architecture

The VLSI architecture for FSA+SV/OSD is shown in Figure 9–10. The only difference from Figure 9–4 is the additional VFC after the set of HFCs. The first, third and fourth scalability can be accomplished in the same way as SV/OSD as discussed in Section 9.1.4. When the number of orientations the system must steer exceeds N_{FV}, multiple VFCs are needed after HFCs as shown in Figure 9–10. Thus the architecture satisfies the second scalability.

9.5.System Integration

Finally the VLSI architecture for FSA+SWA is considered. It achieves the implementation criteria, MRD capability, and steerability. The architecture is constructed by modifying the architecture for SWA. Both decimated and undecimated FSA+SWA are considered here.

9.5.1.Decimated FSA+SWA

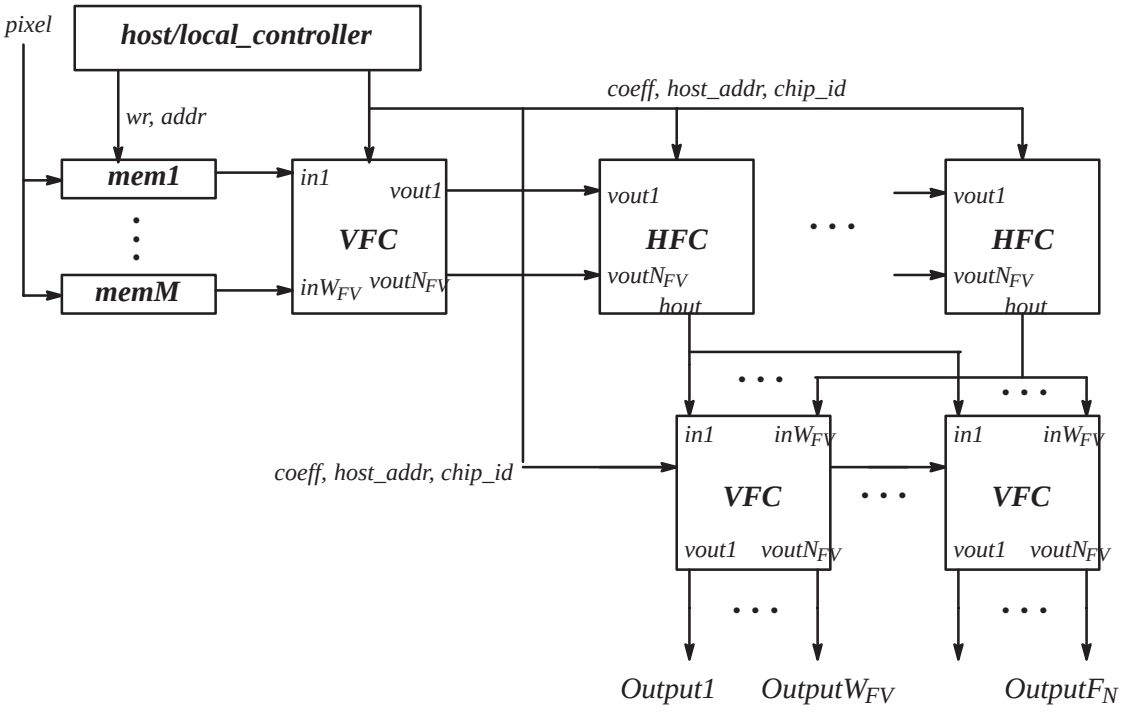


Figure 9–10: VLSI Architecture of FSA+SV/OSD

The architecture for decimated FSA+SWA is shown in Figure 9–11.

9.5.2.Undecimated FSA+SWA

The architecture for undecimated FSA+SWA is shown in Figure 9–12.

9.5.3.Chip Count Estimate

This section estimates the number of VLSI chips including memory to implement various systems. The major portion of filter chips (VFC, HFC and SFC) are occupied by multipliers and adders. One factor which limits the number of multipliers/adders in the filter chips is the number of input/output ports available in the VLSI package used. Assume that 232 input/output ports can be used for the filter inputs/outputs and the filter coefficient inputs. The chips needs power pads, a clock pad and input pads for some control signals such as host address, chip select, and input synchronization signals. However, the number of pads needed for the power, clock and control signals are small relative to the data I/O, and the total number of pads can be less than 300 if the number of data I/O pins is kept under 232. Also assume that a two–phase clocking scheme with non–overlapping clocks is used in the design. This scheme allows bi–directional pads to be output on one phase and input on the other. Thus, some inputs and outputs can share the same pads reducing the number of pads required. In VFC, $2N_{FV}B_P$ output ports for *vout$$* can share the same bi–directional pads with $2N_{FV}B_P$ input ports for *vpout$$* reducing the number of data pads required to $(W_{FV}+2N_{FV}+1)B_P$. In HFC, $3B_P$ output ports for *hout* can share the same bi–directional pads with $3B_P$ input ports for *psum* and $3N_{FH}B_P$ output ports for *hout$$* can share the same bi–directional pads with $3N_{FH}B_P$ input ports for *hpout$$* reducing the number of data pads required to $(5N_{FH}+4)B_P$. In SFC, $3B_P$ output ports for *out* can share the same bi–directional pads with $2B_P$ input ports for *hin* and B_P input ports for *coeff* reducing the number of data pads required to $(W_{FS}+3)B_P$.

Another factor which limits the number of multipliers/adders in the filter chips is the amount of silicon area available. According to [59], the size of an adder using the genesil silicon compiler with a 1.0 um CMOS process is 31.3x7.1 mil^2 and 55.9x7.5 mil^2 for a data width of 8 and 16 bits respectively. The size of a multiplier with a 0.8 um CMOS process is 24.1x18.4 mil^2 and 41.4x37.2 mil^2 for a data width of 8 and 16 bits respectively. The size of a D flip–flop with a 1.0 um CMOS process is 18.5x1.6 mil^2 and 30.9x1.6 mil^2 for a data width of 8 and 16 bits respectively. The size of a multiplexer with 1.0 um CMOS process is 17.2x2.8 mil^2 and 29.5x2.7 mil^2 for the data width 8 and 16 respectively. The size of a 3–port register file with a 1.0 um CMOS process is 18.5x11.1 mil^2 and 30.8x19.3 mil^2 for 8x8 bits and 16x16 bits respectively. A 3–port register allows simultaneous reads and writes, and can implement both the primary and secondary register file in one component. Although 0.5 um process is common in current technology, the above conservative figures are used here. Assume that the chip size can be as large as 400x400 mil^2 and 2/3 of the area can be allocated for adders, multipliers, D flip–flops, multiplexers, and 3–port register files. The rest of area is used for routing and miscellaneous logics. Also assume that the size of a multiplier increases linearly as the width of one of the operands increases, and assume that the size of an adder increases linearly as the width of operands increases. With these assumptions, estimated numbers of multipliers and adders can be obtained for VFC, HFC and SFC. Table 9–4 below shows parameters, W_{FV}, N_{FV}, W_{FH}, N_{FH}, and W_{FS} derived from the estimates.

Table 9–4 Estimated Values for Design Parameters (W_{FV}, N_{FV}, W_{FH}, N_{FH}, and W_{FS})

	$B_P=8$	$B_P=12$
W_{FV}	16	12
N_{FV}	6	3
W_{FH}	13	11
N_{FH}	5	3
W_{FS}	11	11

Using the parameter values listed in Table 9–4, the number of VLSI chips including discrete memory chips can be derived for various types of systems. The derivation is done with an input pixel size of 8 and 12, and the results are shown in Table 9–5.

Table 9–5 VLSI Chip Count for Various Types

- SV/OSD

  1. $F_N=6, P=5, M=11, N=1024$ (the total number of orientational filters = 6)

	$B_P=8$	$B_P=12$
VFC	1	2

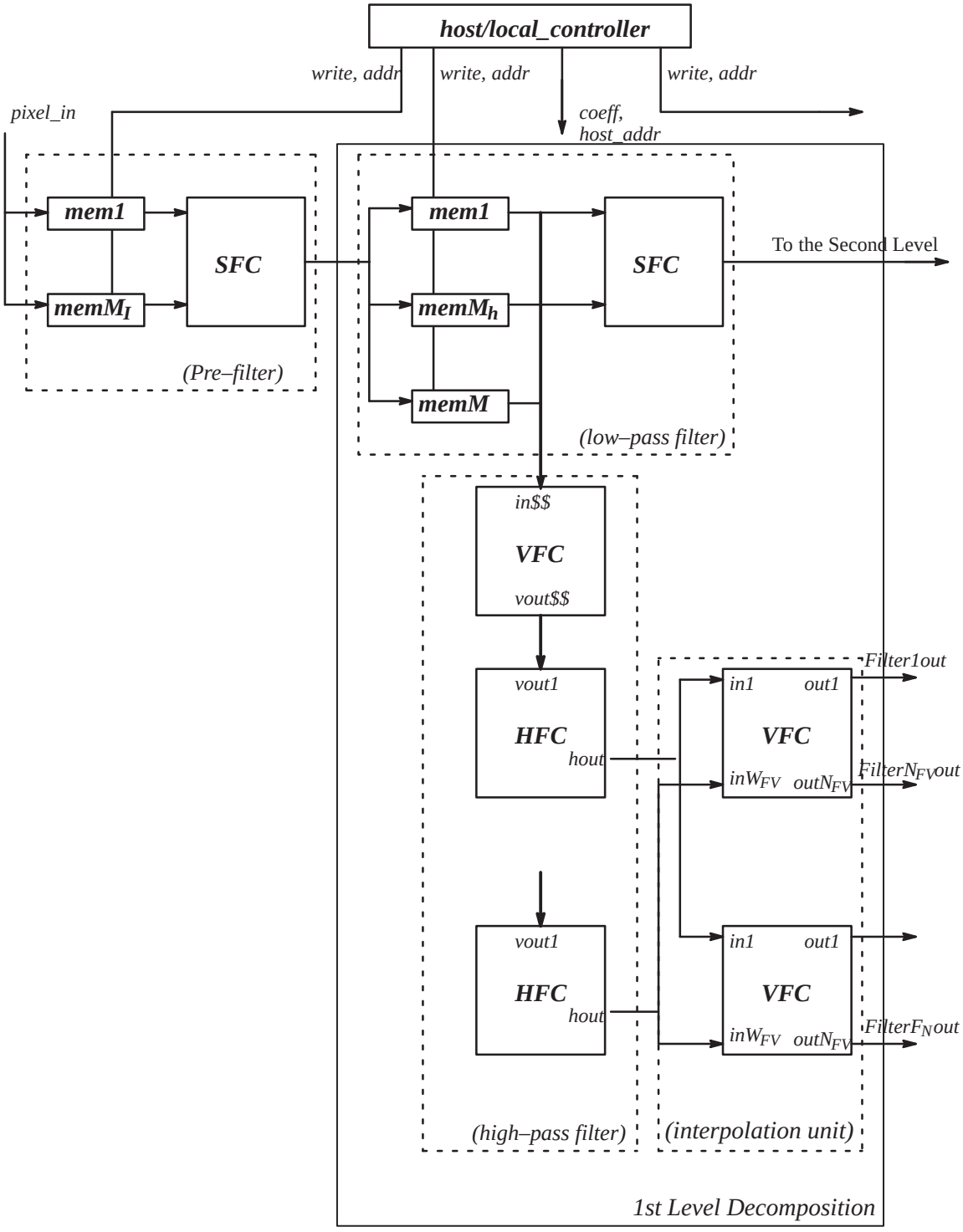
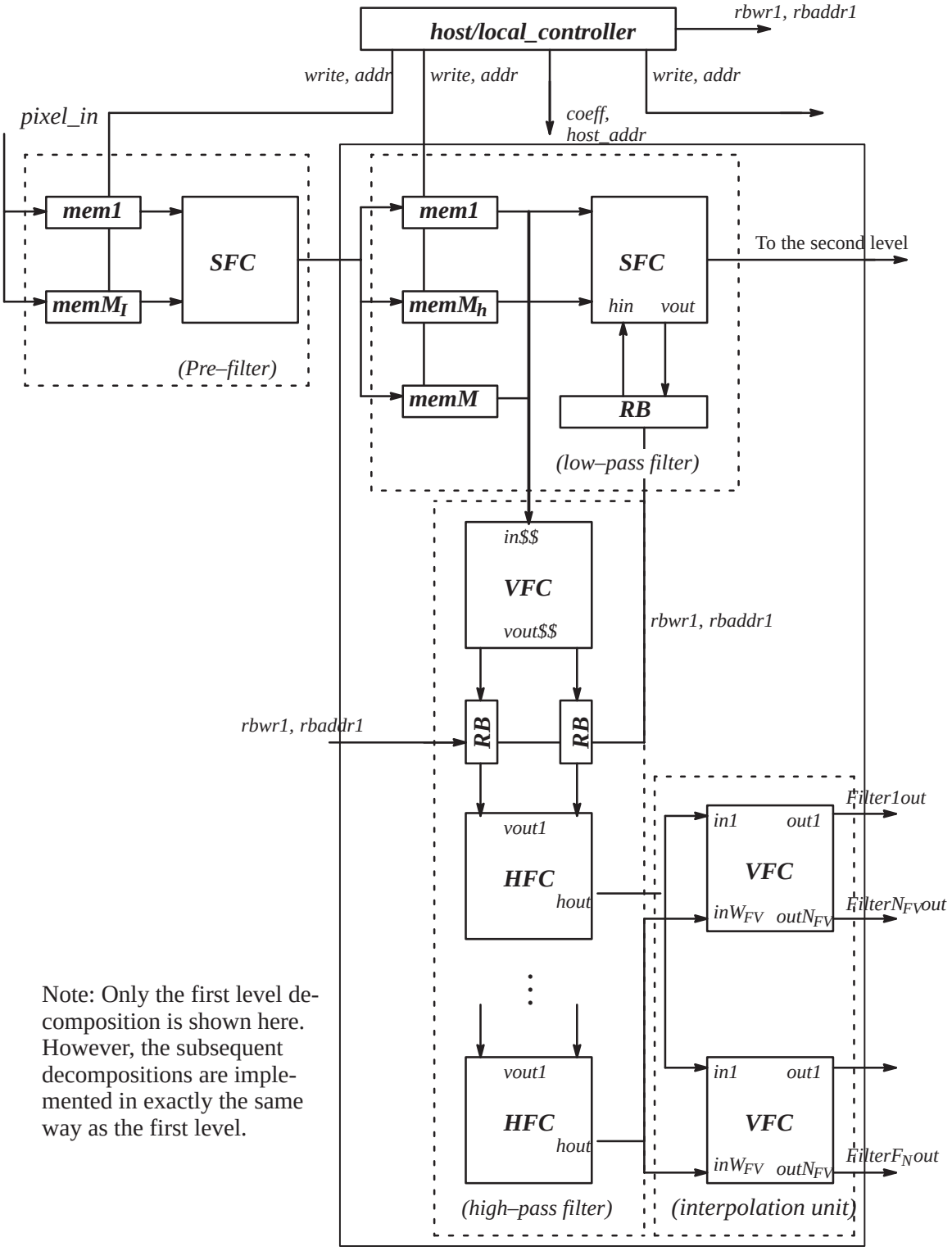


Figure 9–11: VLSI Architecture for Decimated FSA+SWA

Note: Only the first level decomposition is shown here. However, the subsequent decompositions are implemented in exactly the same way as the first level.



Note: Only the first level decomposition is shown here. However, the subsequent decompositions are implemented in exactly the same way as the first level.

Figure 9–12: VLSI Architecture for Undecimated FSA+SWA

HFC	6	12
SFC	0	0
Memory (1k x B_P)	11	11

2. $F_N=6, P=5, M=21, N=1024$ (the total number of orientational filters = 6)

	$B_P=8$	$B_P=12$
VFC	2	4
HFC	12	24
SFC	0	0
Memory (1k x B_P)	21	21

- Decimated SWA

1. $F_N=6, L=4, P=5, M=11, N=1024$ (the total number of orientational filters = 24)

	$B_P=8$	$B_P=12$
VFC	4	8
HFC	24	48
SFC	5	5
Memory (1k x B_P)	18	18
Memory (512 x B_P)	11	11
Memory (256 x B_P)	11	11
Memory (128 x B_P)	11	11

Table 9–5 VLSI Chip Count for Various Types

- Undecimated SWA

  1. $F_N=6, L=4, P=5, M=11, N=1024$ (the total number of orientational filters = 24)

	$B_P=8$	$B_P=12$
VFC	4	8
HFC	24	48
SFC	5	5
Memory(1k x B_P)	75	75

- Decimated FSA+SWA

  1. $F_N=6, L=4, P=8, Q=8, M=11, N=1024$ (the total number of orientational filters = 24)

	$B_P=8$	$B_P=12$
VFC	12	20
HFC	48	72
SFC	5	5
Memory (1k x B_P)	18	18
Memory (512 x B_P)	11	11
Memory (256 x B_P)	11	11
Memory (128 x B_P)	11	11

- Undecimated FSA+SWA

  1. $F_N=6, L=4, P=8, Q=8, M=11, N=1024$ (the total number of orientational filters = 24)

	$B_P=8$	$B_P=12$
VFC	12	20
HFC	48	72
SFC	5	5
Memory (1k x B_P)	75	75

9.5.4.Improvement Using Better Technology

The chip counts derived in Table 9–5 are based on conservative assumptions using old technology. Using current technology, the chip count can be significantly reduced.

The number of I/O pins tends to limit the number of multiplier and adders in a chip rather than the silicon area taken up by them. One way of reducing the limitation of I/O pins is to provide inputs in a half word at a time (it takes 2 cycles to load a full word.) reducing the number of I/O pins required for the data I/O by half at the expense of a using faster clock. The chip has to operate at a speed $2/t_m$ and faster RAMs are required. This I/O scheme together with a better CMOS process (0.5 um, for example) can double the number of multipliers and adders in a chip.

Another possibility is to implement input buffer on the same die with VFC and SFC, or to include the buffer in the same package with VFC and SFC as a multi–chip module. These schemes reduce the number of memory chips to almost zero. (External discrete memory is still needed for the reorder buffer.)

CHAPTER 10

Conclusion

10.1.Summary

This research has examined the role of orientational filters in computer vision applications, and devised an efficient and inexpensive implementation scheme to support many real–time computer vision applications. Some applications require multi–resolution capability, and some require steerability of the filters. The scheme proposed in this research centers around 'separable approximation' which approximates non–separable 2D filters using a set of separable filters.

10.2.Research Results

10.2.1.The Separable Approximation Method

It is possible to construct a system with real–time orientational filters using a FFT. However, a real–time FFT is very expensive in terms of the amount of hardware, and a typical vision system requires multiple orientational filters. Thus, the amount of hardware for a real–time vision system is enormous using the FFT.

It is also possible to construct a real–time system using separable approximation. The method approximates a non–separable filter by a linear sum of separable filters as described in Equation (3.1). Such a system has three advantages for VLSI implementation; (1) a regular filter bank structure with simple data flow, (2) small latency since there is no need for a Fourier transform, and (3) small memory requirements.

Singular Value Decomposition (SVD) achieves the best approximation performance in terms of energy error described in Equation (3.6). The approximation converges to the original filter as the approximation order approaches M, and guarantees a linear convergence.

Orthogonal Sequence Decomposition (OSD) does not achieve the same performance as SVD for the same approximation order. However, the amount of computation and the number of filters are less for OSD than SVD. Thus, OSD has an advantage over SVD as far as the implementation costs are concerned. Hence it is possible for OSD to achieve a better approximation than SVD using the same amount of computation or hardware.

SV/OSD is derived from both SVD and OSD so that it achieves the performance advantages of SVD and the implementation advantages of OSD. It produce a set of orthogonal filters which have better approximation performance than any other orthogonal filters in terms of the energy error defined in (3.30). The number of multiply–accumulate operations is $N^2MP(F_N+1)$ for SV/OSD, compared to $2N^2MPF_N$ for SVD. The throughput is $O(1/t_m)$, latency is $O(NM)$, and the amount of storage required is $O(NM)$ for SV/OSD.

10.2.2.Implementation Scheme for SV/OSD

Two schemes are suggested to perform a 1D filter operation; pipelined filtering and parallel filtering. For 2D separable filtering, the former is suitable for a filter whose direction aligns parallel to the input sequence, and the latter is suitable for a filter whose direction is perpendicular to the input sequence. Assuming the inputs are in a raster order, pipelined filtering is suitable for horizontal filtering, and parallel filtering is suitable for vertical filtering.

In order to make use of the implementation advantage of SV/OSD, the orthogonal filters have to be performed on an input image before the projection filters so that the outputs of the orthogonal filters can be shared among multiple sets of projection filters. In this case, the system requires $P(F_N+1)$ filters. If the filter order is reversed, the system requires $2PF_N$ filters. The amount of intermediate storage can be reduced to NM if the orthogonal filters are the vertical filters, and are performed before the projection filters which are horizontal filters. Figure 3–5 depicts the scheme.

10.2.3.Convergence of SV/OSD

SV/OSD guarantees a linear convergence of the approximation. The approximation converges to the original filter as the approximation order approaches M. For all filters tested, the speed of convergence was much faster than linear convergence, and was close to exponential.

10.2.4.Performance Evaluation for SV/OSD

SV/OSD's performance was evaluated by approximating various Gabor filters. The approximation becomes more difficult as the frequency parameter α and the deviation ratio σ_y/σ_x of the Gabor filters increase. It also becomes more difficult as F_N increases. The approximation results are similar for the real part and the imaginary part. Frequency domain error analysis showed the approximation errors are concentrated in region where the energy content of the filters is significant. Thus, the effects of the errors are small compared to the frequency responses of the filters.

10.2.5.Multi–resolution Decomposition (MRD)

Separable Wavelet Approximation (SWA) is the proposed method for an efficient decimated MRD scheme. It is a combination of separable approximation and wavelet approximation. Equations (5.36), (5.37), and (5.38) describe the process of the decomposition method. The amount of computation for an L level decomposition using the method is approximately $4F_N N^2 PM/3$ compared to $LF_N N^2 M^2$ using the direct method, the throughput is $1/t_m$, the latency is approximately $2^{L-1}N(M_h + M)t_m$, and the amount of storage required is less than $N(M_I + 2max(M, M_h))$.

The decomposition scheme of SWA employs a simple filter bank structure as shown in Figure 5–5. A large part of the decomposition structure can be shared with other decomposition structures associated with different orientational filters.

SWA can be modified to perform an undecimated MRD. The scheme employs an intermediate buffer called a reorder buffer, at the output end of each vertical filter, to provide a reordered input stream to the horizontal filter. The reorder buffer arranges the order of a pixel stream so that the horizontal filtering can be done using a pipelined filtering scheme. The size of each reorder buffer is N. The system requires $LF_N(P+1)+1$ reorder buffers and L reorder buffer address generators. The latency of the decomposition is the same as the decimated SWA case, which is approximately $2^{L-1}N(M_h + M)t_m$. The throughput is $1/t_m$, amount of computation is $2N^2M_I+2LN^2M_h+LN^2MP+LF_NN^2MP= 2N^2M_I + LN^2(2M_h+MP+F_NMP)$, and the amount of storage is $NM_I+LN\ max(M,M_h) + NL(F_NP+1)$.

SWA uses a basic spline function as an interpolation filter. The length of the pre–filter and the low pass filter in SWA depend on the order of the basic spline. If a k^{th} order basic spline is used, then the lengths of the pre–filter and the low pass filter are k and $k+2$ respectively. Based on the performance evaluation results, the basic spline of order 7 achieves a good performance/computation trade–off.

10.2.6.Performance Evaluation for SWA

SWA's performance was evaluated by approximating various Gabor filters at several dilation levels. The amount of error increases only slightly as the dilation level increases. The error characteristics are very similar for the real part and the imaginary part of the filter. The approximation becomes more difficult as the frequency parameter, α, increases. It also becomes more difficult as the aspect ratio σ_y/σ_x deviates from 1.0. Frequency domain error analysis showed the approximation errors are concentrated in the region where the energy content of the filters is significant. Thus, the effects of the errors are small compared to the frequency responses of the filters.

10.2.7.Applicability of SWA to Vision Algorithms

A multi–resolution edge detection algorithm was implemented using decimated SWA. The performance of the approximation filters was identical to Gabor filters for an approximation order larger than 8. An unsupervised texture segmentation method proposed by Jain and Farrokhnia was also implemented using undecimated SWA, and the performance was compared with the original method using Gabor filters. At an approximation order of 5, the approximation method produced very comparable results with the original method. SWA has been incorporated into the Georgia Tech Vision Model (GTV) for DOG filters in the pattern perception units, and its effectiveness has been demonstrated.

10.2.8.Steerable Filters

Fourier Series Approximation (FSA) can produce an identical approximation result as Deformable Kernel Approximation (DKA). FSA has more flexibility in the filter selection process, since the real and the imaginary parts are separate entities in the selection process, whereas they are treated as a pair in DKA. Thus, basic filters obtained by DKA are complex filters even though the filter to be steered is real.

SV/OSD can be combined with FSA (FSA + SV/OSD) for an efficient steerable filter implementation. Equations (7.28) and (7.29) describes the process. The throughput of the system can be $1/t_m$, the latency is $O(NM)$, the amount of computation is $N^2(MQP+F_N)$, and the storage requirement is NM.

FSA and SWA can be combined (decimated FSA+SWA) for an efficient decimated steerable MRD scheme. Equations (7.30) through (7.32) describes the decomposition process. The throughput of the system can be $1/t_m$, the latency is $2^{L-1}N(M_h + M)t_m$, the amount of computation is $4N^2PQM/3$, and the storage requirement is $N(M_I+ max(M,M_h))$.

FSA and SWA can be combined (undecimated FSA+SWA) for an efficient undecimated steerable MRD scheme. The throughput of the system can be $1/t_m$, the latency is $2^{L-1}N(M_h + M)t_m$, the amount of computation is $2N^2M_I + LN^2(2M_h + MP + MPQ)$, and the storage requirement is $NM_I + LNmax(M,M_h) + NL(PQ+1)$.

10.2.9.Performance Evaluation for FSA+SV/OSD and FSA+SWA

Various Gabor filters are used for the evaluation. The amount of error decreases rapidly as P and Q increase. The approximation becomes more difficult as the frequency parameter α increases, and the aspect ratio σ_x/σ_y increases.

10.2.10.VLSI Architecture for SV/OSD

Six VLSI architectures were defined. These architectures are proposed for (1) SV/OSD, (2) decimated MRD using SWA, (3) undecimated MRD using SWA, (4) FSA+SV/OSD, (5) a decimated steerable MRD using FSA+SWA, and (6) an undecimated steerable MRD using FSA+SWA. Each of these systems satisfies the four scalability requirements listed in Chapter 1.

10.3.Contributions

The contributions of this research are briefly described below. Each subsection is related to one of the research objectives described in Chapter 1.

10.3.1.Computation of Orientational Filters

Various filters and implementation schemes were investigated based on the implementation criteria which measures applicability of the filters to real–time image analysis applications. Based on the investigation, the thesis proposed to use the separable approximation method to implement orientational filters for real–time image analysis applications.

Three algorithms based on separable approximation have been investigated. They are Singular Value Decomposition (SVD), Orthogonal Sequence Decomposition (OSD), and Singular Value/Orthogonal Sequence Decomposition (SV/OSD). No prior research was found related to OSD and SV/OSD. SV/OSD achieves the best performance/implementation trade–off.

An efficient implementation scheme tailored for SV/OSD was developed. It implements vertical filters with parallel filtering and horizontal filters with pipelined filtering to reduce the latency and storage requirements.

Extensive performance evaluation has been done for SV/OSD to demonstrate the effectiveness of the approximation.

10.3.2.Multi–resolution Decomposition

SV/OSD and Wavelet Approximation have been combined to produce an efficient MRD scheme (SWA) with 2D orientational filters. An extension of SWA for undecimated MRD has also been developed.

Extensive performance evaluation of a MRD using SWA has been done. Undecimated SWA has been incorporated into the unsupervised texture segmentation of Jain and Farrokhnia. The result shows that the scheme achieves comparable performance with less computation and implementation cost. Undecimated SWA has been incorporated into the Georgia Tech Vision model.

10.3.3.Steerable System

An efficient steerable system based on SV/OSD and Fourier Series Approximation (FSA) was developed. Extensive performance evaluation of SV/OSD+FSA has been completed.

An efficient steerable MRD system based on SWA and FSA was developed. Extensive performance evaluation of SWA+FSA has been completed.

10.3.4.VLSI Architecture

A VLSI architecture disign for SV/OSD, SWA, undecimated SWA, FSA, SV/OSD+FSA, decimated SWA+FSA, and undecimated SWA+FSA was used to estimate implementation complexity. Based on this evaluation, each system satisfies four criteria: input image scalability, filter number scalability, filter size scalability, and approximation order scalability.

10.4.Research Direction

A next step in this research is to incorporate the approximation techniques in vision applications, and to evaluate the performance of the algorithms with different approximation orders. Such attempts were initiated in Chapter 7 for multi–resolution edge detection, unsupervised texture segmentation and pattern perception filters in GTV. There are many algorithms not limited to vision algorithms which use multiple non–separable filters and require real–time performance. Such algorithms can benefit greatly from the approximation methods proposed in this thesis.

Another important step is to implement the designs described in Chapter 9 on silicon, and construct a real system for real applications. The design for VFC, HFC, and SFC need to be expanded to provide more details The estimates in Chapter 9 are very conservative, and will improve significantly with current technology.

Appendix A.

Human Visual Systems

In this appendix, an overview of the human visual system (HVS) is given, and the processing of simple/complex cells in visual cortex is described. The purpose of this appendix is to show how orientational filtering is related to processing in the HVS. For this purpose, an emphasis is placed on computational schemes of the system.

A.1.Retina

The retina is a layer of cells located on the inner surface of the eyeball. It contains millions of light–sensitive receptors which spread over the whole retina. There are two types of receptors: rods and cones. Different characteristics of rods and cones result from different photosensitive substances called pigments. Rods are about 500 times more sensitive to light intensity than cones. Cones contain pigments sensitive to color. There are about 120 millions rods and 6 million cones in a human eye. The receptors are placed non–uniformly over the retina with more concentration around the center of the retina (fovea) and less concentration away from the center. Cones are concentrated around the fovea with a density of about 150,000 cones per square millimeter. Hence an image is sampled much more densely around the fovea, which enables a focused analysis of the image[32].

The sampling rate near the fovea is close to the highest frequency that the optics of the eye will pass. Thus, a human does not observe aliasing around his focused area. However, the drastic fall–off in cell density in the near periphery causes the retina to undersample, and high contrast, high frequency gratings can cause aliasing. The effects of aliasing are not prominent in our daily lives. Two reasons have been suggested[33]. First, irregular spacing of the cones causes the high spatial frequencies to be aliased back as widely scattered noise. Second, natural scenes do not have sufficient power in the high frequencies to produce aliasing.

Signals from receptors go through several cells before reaching the optic nerve. These cells are *bipolar cells*, *horizontal cells*, *amacrine cells* and *ganglion cells*. Bipolar cells combine responses of multiple receptors and pass them to a ganglion cell. Ganglion cells combine responses of multiple bipolar cells and pass these to an optic nerve. Horizontal cells transmit signals across the retina, enabling different receptors or bipolar cells to communicate with each other. Amacrine cells transmit signals across the retina, enabling different bipolar or ganglion cells to communicate with each other.

There are approximately 126 million receptors, and only 1 million optic nerves. Data size reduction is done along the pathways from receptors to bipolar cells and from bipolar cells to ganglion cells[51].

Each ganglion cell responds to a light stimulus falling within a limited retinal region, or *receptive field*, of the cell. There are two types of ganglion cells: on–center cells and off–center cells. The receptive field of an on–center cell has a center region where the rate of impulse discharge of the cell increases as illumination of light to the region increases (excitatory region), and has an annular surrounding region of lower sensitivity where the rate of impulse discharge decreases as illumination of light to the region increases (inhibitory region). Its spatial response (*receptive field profile*) has a concentric center–surround area.

On the other hand, the off–center cell has an inhibitory region at the center of its receptive field and excitatory region surrounding the center. Inhibition of both cells is provided mainly from amacrine cells. The response of on/off–center cells can be modeled by a difference of Gaussian[28],

$$R_{on}(r) \;=\; -\,R_{off}(r) \;=\; c_0 e^{-r^2/\sigma_0^2} \;-\; c_1 e^{-r^2/\sigma_1^2} \qquad \text{(A.1)}$$

with $c_0 > c_1$ and $\sigma_0 < \sigma_1$. This is also called Laplacian of Gaussian. These cells act as edge operators.

Most Ganglion cells are also classified into two groups: X cell and Y cell. X cells respond in a sustained fashion to a stimulus, producing a response as long as the stimulus lasts. Y cells respond in a transition fashion at the beginning and the end of the stimulus, producing a strong impulse. Because of this behavior, they are also called *sustained* and *transient* cells. X cells also exhibit linearity in their responses. When the cell's receptive field is illuminated alternately by a uniform field and by a sinusoidal grating of the same average luminance, the position of the grating can be found so that the response of the

cell does not change between the uniform and the sinusoidal stimuli. No such position can be found in Y cells. X cells are mainly found in the central retina, whereas the Y cells are mainly found in the periphery of the retina.

At the output of the ganglion cells, a signal becomes *all–or–none*, i.e. the signal is either on or off. The receptive fields of ganglion cells in general have larger sizes as they move from center to periphery.

A.2.Striate Cortex

Responses of the retinal ganglion cells are projected to the striate cortex through *lateral geniculate nuclei*. The striate cortex consists of a cluster of many constituent columns of cells. Each column is about 0.5 mm^2 on the cortical surface and about 3–4 mm deep. Columns are also divided into layers numbered from 1 to 6. Layer 4 is further divided into 4A, 4B, 4Cα and 4Cβ. Fibers from the lateral geniculate nuclei terminate at 4A, 4Cα and 4Cβ. Cells in the striate cortex are classified as *simple cells*, *complex cells*, and *hypercomplex cells*. These cells are all maximally excited by line stimuli oriented to certain directions, a property different from the ganglion cells which have no orientation selectivity. Interestingly, cells in the same column are tuned to the same orientation.

Simple cells have orientation preferences and respond best to lines of particular orientations. They have several alternating excitatory and inhibitory regions in their receptive fields. Another interesting property is that there are pairs of simple cells in approximate quadrature phase relation to each other, and these pairs locate side by side. The whole set of simple cells in the striate cortex covers various orientations and radial frequency regions, thus acting as a localized frequency analyzer. Simple cells receive signals mainly from X cells of retinal ganglion cells.

Because of their simple receptive field profiles, some mathematical models for the simple cells have been suggested and their roles in image analysis have been studied in conjunction with computer vision research. The most popular model for simple cells is Gabor functions[56]. Atick and Nedlich modeled the response of simple cells starting from ganglion afferents[2][95]. Ganglion cells are modeled by (A.1) followed by a half rectification. The linear responses of simple cells can be obtained from non–linear responses of ganglion cells only when both on and off cell of the same receptive field contribute with the same weight.

Simple cells of the same orientation preference but different receptive fields are located close together in the cortex. This region of the simple cells can be depicted as 'a slag of columns'. Simple cells in each column correspond to a particular receptive field. Each column is occupied by simple cells with the same orientation preference and the same receptive field but a different radial frequency preference. This structure is termed *hypercolumn*[39].

Complex cells do not have clear separation of excitatory and inhibitory regions as in simple cells. They respond to edges of a certain orientation, however the position of the edges do not affect their response as long as they are inside the receptive fields of the cells. If a line matched with the orientation preference of a particular complex cell moves across its receptive field, the cell responds vigorously as soon as the line appears on the receptive field, and keeps responding until the line leaves the receptive field. It has been suggested that complex cells are motion detectors of objects moving in a particular direction.

Other typical properties of complex cells are that their receptive fields are larger than simple cells. They often respond only to one direction of movements, i.e. a cell responds well to a movement from left to right, but not well to a movement from right to left. They receive signals mainly from Y cells rather than X cells and do not receive signals from simple cells.

Hypercomplex cells also do not have excitatory and inhibitory regions, and respond well to moving lines of certain directions. However, they are sensitive to stimuli length. They do not respond well to edges made too long at one end or both. This length selectivity suggests that hypercomplex cells act as corner detectors.

A.3.Other Cortical Area

Cells in the striate cortex are the source of two major cortical projection systems[92]. The projection into the temporal cortex is involved in object recognition. The projection into the parietal cortex is involved in spatial localization of objects and motion perception. The computational mechanisms in these cortical areas are largely unknown.

Communication takes place by short association "U" fibers in the underlying white matter. As the processing proceeds, larger portions of the visual field and communication between two hemispheres are involved, suggesting the development of more global perceptual mechanisms.

For object recognition, the processing has to satisfy the following three requirements[92].

- shape consistency – ability to perceive a specific shape with any orientations and sizes.
- color consistency – ability to perceive a color independently from the illuminating lights of a different spectral composition.

- visual attention – ability to filter out unimportant visual stimuli.

After the properties of an object is analyzed and coded, a matching needs to be made at database memory for recognition of the object.

A.4.Comments

Apparently, orientational selectivity is a major characteristic of an early stage of the human vision system, and the directionally sensitive processing is a major source of inputs to later stages of the system. As physiological research reveals both physical and computational structure of the human vision system, they can be incorporated into a computer vision system to give a more robust and flexible performance.

Some important points of the human visual system which have been adapted into computer vision algorithms are the following.

1. Receptive field profiles of simple cells are very similar to Gabor functions tuned to different radial frequencies and orientations.
2. Adjacent pairs of simple cells form a quadrature pair.
3. Ganglion cells can be coupled into an on/off–center cells pair with the same receptive field.

Some interesting points which are not fully known and have a potential impact on computer vision algorithms are

1. How reduction of data is done from the retinal level to the striate cortex.
2. Computational models of complex and hypercomplex cells.
3. Structure of higher cortical areas and how stimuli from the striate cortex are processed there.

Appendix B.

Derivation of Low–Pass Filter $\tilde{h}$

Denote the B–spline of order k as $\phi^k(x)$. $\phi^k(x)$ can be generated by taking an auto–correlation of $\phi^{k-1}(x)$ with $\phi^0(x)$. Namely,

$$\phi^k(x) = \phi^{k-1}(x) \hat{*} \phi^0(x) = \int \phi^{k-1}(y)\phi^0(y-x)dy \,, \tag{B.1}$$

where $\hat{*}$ denotes an auto–correlation operation.

Assume $\phi^{k-1}(x)$ satisfies a dilation relation,

$$\phi^{k-1}(x) = \sqrt{2}\sum_n h^{k-1}[n]\phi^{k-1}(2x-n) \quad . \tag{B.2}$$

Then,

$$\phi^k(x) = \phi^{k-1}(x) \hat{*} \phi^0(x) = \int \phi^{k-1}(y)\phi^0(y-x)dy$$

$$= \int \left\{ \sum_n h^{k-1}[n]\phi^{k-1}(2y-n) \right\} \left\{ \sum_m h^0[m]\phi^0(2y-2x-m) \right\} dy$$

$$= \sum_m \sum_n h^{k-1}[n]h^0[m] \int \phi^{k-1}(2y-n)\phi^0(2y-2x-m)dy = \sum_m \sum_n h^{k-1}[n]h^0[m] \int \phi^{k-1}(y)\phi^0(y-2x-m+n)\frac{dy}{2}$$

$$= \frac{1}{2}\sum_m h^{k-1}[m] \sum_n h^0[n]\phi^k(2x-m+n) \,. \tag{B.3}$$

Hence $\phi^k(x)$ also satisfies a dilation relation. The B–spline of order 0 is

$$\phi^0(x) = \begin{cases} 1 & 0 \le x \le 1 \\ 0 & otherwise \end{cases} \text{, and} \tag{B.4}$$

$$h^0[n] = \begin{cases} \frac{1}{\sqrt{2}} & n = 0, 1 \\ 0 & otherwise \end{cases} \,. \tag{B.5}$$

Using (B.5), (B.3) becomes

$$\phi^k(x) = 2^{-3/2}\sum_m \{ h^{k-1}[m] + h^{k-1}[m+1] \} \phi^k(2x-m) \quad . \tag{B.6}$$

Thus,

$$h^k[m] = 2^{-1}(h^{k-1}[m] + h^{k-1}[m+1]) \quad . \tag{B.7}$$

Using a combinatorial identity, $\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$, and $h^0[m] = 2^{-1/2}\binom{1}{m}$,

$$h^k[m] = 2^{-1}\left(h^{k-1}[m] + h^{k-1}[m-1] \right) = 2^{-k-1/2} \binom{k+1}{m} \quad . \tag{B.8}$$

References

[1]S. B. Abramson and F. S. Fay, "Application of Multi–resolution Spatial Filters to Long–Axis Tracking", *IEEE Transaction on Medical Imaging*, vol. 9, no. 2, pp. 151–8, June 1990.

[2]J. J. Atick and A. N. Redlich, "Mathematical Model of the Simple Cells in the Visual Cortex", *Biological Cybernetics*, vol. 63, no. 2, pp. 99–109, 1990.

[3]D. H. Ballard and C. M. Brown, "Computer Vision", *Prentice Hall*, 1982.

[4]R. H. Bamberger and M. J. T. Smith, "Efficient 2–D An Analysis/Synthesis Filter Banks For Directional Image Component Representation", *1990 IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 2009–12, 1990.

[5]R. H. Bamberger, "The Directional Filter Bank: A Multirate Filter Bank for the Directional Decomposition of Images", *Ph. D Thesis*, Georgia Institute of Technology, 1991.

[6]R. Bhatia, M. Furuta and J. Ponce, "A Quasi Radix–16 FFT VLSI Processor", *ICASSP 91*, vol. 2, pp. 1085–88, May 1991.

[7]J. Bigun and J. M. H. du Buf, "Geometric Image Primitives by Complex Moments in Gabor Space and the Application to Texture Segmentation", *Proc. 1992 IEEE Comp. Soc. Conf. on Comp. Vision and Pattern Recognition*, pp. 648–59, 1992.

[8]W. F. Bischof and V. D Lollo, "On the Half–cycle Displacement Limit of Sampled Directional Motion", *Vision Research*, vol. 31, no. 4, pp. 649–60, 1991.

[9]A. C. Bovik, M Clark, and W. S. Geisler, "Multichannel Texture Analysis Using Localized Spatial Filters", *IEEE Trans. on Pattern Anal. Machine Intell..*, vol. 12, No. 1, pp. 55–73, Jan. 1990.

[10]A. C. Bovik, "Analysis of Multichannel Narrow–band Filters for Image Texture Segmentation", *IEEE Trans. on Signal Processing*, vol. 39, no. 9, pp. 2025–43, Sept. 1991.

[11]P. Brodatz, "Textures – A Photographic Album for Artists and Designers", *New York: Dover*, 1966.

[12]J. M. H. du Buf, "Abstract Processes in texture Discrimination", *Spatial Vision*, vol. 6, no. 3, pp. 221–42, 1992.

[13]V. Cantoni and S. Leveialdi eds., "Pyramidal Systems for Computer Vision", *NATO ASI Series*, 1986.

[14]A. Choudhary and S. Ranka, "Parallel Processing for Computer Vision and Image Understanding", *Computer*, vol. 25, no. 2, pp. 7–10, Feb. 1992.

[15]M. Clark, A. C. Bovik and W. S. Geisler, "Texture Segmentation Using Gabor Modulation/Demodulation", *Pattern Recognition Letter*, vol. 6, no. 4, pp. 261–7, Sep. 1987.

[16]A. Croisier, D. Esteban and C. Galand, "Perfect Channel Splitting by Use of Interpolation/Decimation/Tree Decomposition Techniques", *Proceedings of Int. Conf. on Inform. Sciences and Systems*, pp. 443–446, August 1976.

[17]J. Canny, "A Computational Approach to Edge Detection", *IEEE Transaction on Pattern Recognition and Machine Intelligence*, vol. 8, pp. 679–98, November 1986.

[18]M. Clark and A. C. Bovik, "Experiments in Segmenting Texton Patterns Using Localized Spatial Filters", *Pattern Recognition*, vol. 22, no. 6, pp. 707–17, 1989.

[19]R. W. Conners and C. A. Harlow, "A Theoretical Comparison of Texture Algorithms", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 2, no. 3, pp. 204–222, May 1980.

[20]H. F. Davis, "Fourier Series and Orthogonal Functions", *Dover*, New York, 1963.

[21]I. Daubechies, "Orthonormal Bases of Compactly Supported Wavelets", *Communication on Pure and Applied Mathematics*, vol. 41, pp. 909–996, November 1988.

[22]J. G. Daugman, "Complete Discrete 2–D Gabor Transforms by Neural Networks for Image Analysis and Compression", *IEEE Transaction on Acoustics, Speech and Signal Processing*, vol. 36 No. 7, pp. 1169–79, July 1988.

[23]J. G. Daugman, "Networks for Image Analysis: Motion and Texture", *Proceedings of Int. Joint Conf. on Neural Networks*, vol.1 , pp. 189–93, 1989.

[24]J. G. Daugman, ”Uncertainty Relation for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two–dimensional Visual Cortical Filters”, *Journal of Opt. Soc. of Am.*, vol. 2, no. 7, pp. 1160–9, July 1985.

[25]T. J. Doll, S. W. McWhorter, D. E. Schmieder, and A. A. Wasilewski, ”Simulation of Selective Attention And Training Effects in Visual Search and Detection”, *Technical Report*, Georgia Tech Research Institute

[26]D. E. Dudgeon and R. M. Mersereau, ”Multi–dimensional Digital Signal Processing”, *Prentice–Hall*,

[27]P. Dutilleux, ”An Implementation of the ’algorithme a‘ trous’ to Compute the Wavelet Transform”, *Wavelets, Time–Frequency Methods and Phase Space*, Springer , pp. 298–304 , 1989.

[28]R. T. Eskew, Jr., ”White–Noise Analysis of Human Spatial Vision”, *Ph. D Thesis, Georgia Institute of Technology*, Sep. 1983.

[29]D. Dunn, W. Higgins and J. Wakeley, ”2D Analysis of Gabor–Filter Output Signatures for Texture Segmentation”, *ICASSP–92*, vol. 3, pp. 65–8, 1992.

[30]M. M. Fleck, ”Some Defects in Finite–Difference Edge Finders”, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 14, no. 3, pp. 337–345, March 1992.

[31]W. T. Freeman and E. H. Adelson, ”The Design and Use of Steerable Filters”, *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 13, no. 9, pp. 891–906, Sep. 1991.

[32]J. P. Frisby, ”Seeing: Illusion, Brain and Mind”, *Oxford University Press*, 1979.

[33]S. J. Galvin and D. R. Williams, ”No Aliasing at Edges in Normal Viewing”, *Vision Research*, vol. 32, no. 12, pp. 2251–2259, 1992.

[34]G. H. Granlund, ”In Search of General Picture Processing Operator”, *Computer Graphics And Image Processing*, vol. 8, pp. 155–173, 1978.

[35]H. Greenspan, R. Goodman and R. Chellappa, ”Texture Analysis via Unsupervised and Supervised Learning”, *Proc. Int. Joint Conf on Neural Networks*, vol. 1, pp. 639–44, 1991.

[36]R. M. Haralick, ”Digital Step Edges from Zero Crossing of Second Directional Derivatives”, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 6, no. 1, pp. 58–68, Jan. 1984.

[37]R. M. Haralick, ”Statistical and Structural Approaches to Texture”, *Proceedings of IEEE*, vol. 67, no. 5, pp. 786–804, May 1979.

[38]M. Holschneider, R. Kronland–Martinet, J. Morlet and Ph. Tchamitchian, ”A Real–Time Algorithm for Signal Analysis with the Help of the Wavelet Transform”, *Wavelets, Time–Frequency Methods and Phase Space*, Springer, pp. 286–301 , 1989.

[39]D. H. Hubel and T. N. Wiesel, ”Receptive Fields Binocular Interaction and Functional Architecture in the Cat’s Visual Cortex”, *Journal of Physiology*, vol. 160, pp. 105–154, 1962.

[40]A. K. Jain, ”Fundamentals of Digital Image Processing”, *Prentice Hall*, 1989.

[41]A. K. Jain and F. Farrokhnia, ”Unsupervised Texture Segmentation Using Gabor Filters”, *Pattern Recognition*, vol. 24, no. 12, pp. 1167–86, 1991.

[42]D. G. Jones and J. Malik, ”Determining Three–dimensional Shape from Orientation and Spatial Frequency Disparity”, *ECCV ’92. Second European Conference on Computer Vision Proceedings*, pp. 661–9, 1992.

[43]B. Julesz, ”Visual Pattern Discrimination”, *IRE Transaction of Information Theory*, vol. 8,, pp. 84–92, 1962.

[44]H. Knutsson, ”Filtering and Reconstruction in Image Processing”, *Ph.D dissertation*, Linkoping Univ., 1982.

[45]J. Kovacevic, ”Filter Banks and Wavelets: Extensions and Applications”, *Columbia University Center for Telecommunications Research Technical Report 257–91–38*, 1991.

[46]T. Kubota and C. O. Alford, ”Computation of Orientational FIlters for Real–Time Computer Vision Problems I: Implementation and Methodology”, *to appear in Real Time Imaging Journal*,

[47]T. Kubota and C. O. Alford, ”Computation of Orientational FIlters for Real–Time Computer Vision Problems II: Multi–resolution Image Decomposition”, *to appear in Real Time Imaging Journal*,

[48]T. Kubota, C. O. Alford and M. B. Woods, ”Design and Implementation of Orientational Filters Using Separable Approximation Methods”, *Proceedings of IEEE/SMC International Conference*, vol. 2, pp. 333–8, Oct. 1993.

[49]T. Kubota, C. O. Alford and M. B. Woods, ”Separable Approximation Methods for Orientational Filters and VLSI Implementations”, *Proceedings of International Conference on Optical 3D Measurement Techniques*, Oct. 1993.

[50]H. J. Landau and H. O. Pollak, ”Prolate Spheroidal Wavefunctions, Fourier Analysis and Uncertainty II”, *Bell System Technical Journal*, vol. 40, pp. 65–84, 1961.

[51]P. Lennie, ”Parallel Visual Pathways: A Review”, *Vision Research*, vol. 20, pp. 561–594, 1980.

[52]S. Lu, E. Hernandez and A. Clark, ”Texture Segmentation by Clustering of Gabor Feature Vectors”, *Proceedings of International Joint Conference on Neural Networks*, vol. 1, pp. 683–8 , July 1991.

[53]W. McIlhagga, ”A Model for Simple Cells as Optimal Edge Detectors”, *Biological Cybernetics*, vol. 66, no. 2, pp. 177–83, 1991.

[54]J. Malik and P. Perona, ”A Computational Model of Texture Segmentation”, *Journal of Optical Soc. of America*, vol. 7, no. 5, pp. 923–932, May 1990.

[55]S. Mallet, ”Theory for Multi–resolution Signal Decomposition: The Wavelet Representation”, *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 11, pp. 674–693, July 1989.

[56]S. Marcelja, ”Mathematical Description of The Responses of Simple Cortical Cells”, *J. Opt. Soc. America*, vol. 70, pp. 1297–1300, 1980.

[57]D. Marr, ”Vision: a computational Investigation into the Human Representation and Processing of Visual Information”, W. H. Freeman, 1982.

[58]D. Marr and E. Hildreth, ”Theory of Edge Detection”, *Proceedings of the Royal Society of London*, vol. 207, pp. 187–217, 1980.

[59]Mentor Graphics Corporation, ”Advanced Technology Library Reference Guide”, Oct 1991.

[60]A. W. Naylor and G. R. Sell, ”Linear Operator Theory in Engineering and Science”, *Springer–Verlag*, 1982.

[61]R. Nevatia and K. R. Babu, ”Linear Feature Extraction and Description”, *Computer Graphics and Image Processing*, vol. 13, pp. 257–269, 1980.

[62]G. Nudd, N. Francis, T. Atherton and D. Kerbyson, ”Hierarchical Multiple SIMD Architecture for Image Analysis”, *Machine Vision and Applications*, vol. 5, no. 2, pp. 85–103, 1992.

[63]A. J. Parker and M. J. Hawken, ”Two–dimensional Spatial Structure of Receptive Fields in Monkey Striate Cortex”, *Journal of Optical Society of America. A, Optics and Image Science*, vol. 5, no4, pp. 598–605, April 1988

[64]E. Peli, ”Adaptive Enhancement Based on a Visual Model”, *Opt. Eng., Bellingham*, vol. 26 , no. 7, pp. 655–60, July 1987.

[65]P. Perona and J. Malik, ”Detecting and Localizing Edges Composed of Steps, Peaks and Roofs”, *Proceedings of Third International Conference on Computer Vision*, pp. 52–57, 1990.

[66]P. Perona, ”Deformable Kernels for Early Vision”, *Proc. 1991 IEEE Computer Soc. Conf. on Computer Vision and Pattern Recognition*, pp. 222–7, 1991.

[67]P. Perona, "Finite Representation of Deformable Kernels", *Proc. of the SPIE – The Int. Soc. for Optical Engineering*, vol. 1571, pp. 8–17, 1991.

[68]A. Perry and D. G. Lowe, "Segmentation of Textured Images", *Proceedings CVPR 89*, pp. 319–25, June 1989.

[69]W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, "Numerical Recipes in C", *Cambridge University Press*, 1988.

[70]T. Pytosh and A. Magnani, "A New Parallel 2–D FFT Architecture", *ICASSP 1990*, vol. 2, pp. 905–8, April 1990.

[71]N. Ranganathan, R. Mehrotra and K. R. Namuduri, "An Architecture To Implement Multiresolution", *ICASSP 91*, vol. 2, pp. 1157–60, May 1991.

[72]T. R. Reed and H. Wechsler, "Segmentation of Textured Images and Gestalt Organization Using Spatial/Spatial–Frequency Representations", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 12, no. 1, pp. 1–12, Jan. 1990.

[73]A. Reeves, "Parallel Computer Architectures for Image Processing", *Computer Vision, Graphics and Image Processing*, vol. 25, pp. 68–88, 1984.

[74]O. Rioul and P. Duhamel, "Fast Algorithm for Discrete and Continuous Wavelet Transforms", *IEEE Trans. on Information Theory*, vol. 38, no. 2, pp. 569–586, March 1992.

[75]O. Rioul and M. Vetterli, "Wavelets and Signal Processing", *IEEE SP Magazine*, pp. 14–38, Oct. 1991.

[76]A. Rosenfeld ed., "Multiresolution Techniques in Computer Vision", *Springer–Verlag*, New York , 1984.

[77]B. S. Rubenstein and D. Sagi, "Spatial Variability as a Limiting Factor in Texture–Discrimination tasks: Implications for Performance Asymmetries", *Journal of the Opt. Soc. of America A (Opt. and Image Sci.)*, vol. 7, no. 9, pp. 1632–43, Sep. 1990.

[78]R. T. Shann and J. P. Oakley, "A Novel Approach to Boundary Finding", *Proceedings of the Fifth Alvey Vision Conference*, pp. 133–8, Sep. 1989.

[79]K. Sapiecha and R. Jarocki, "Modular Architecture for High Performance Implementation of the FFT Algorithm", *IEEE Transaction on Computers*, vol. 39 No. 12, pp. 1464–1468, Dec. 1990.

[80]M. J. Shensa, "The Discrete Wavelet Transform: Wedding the A Trous and Mallat Algorithms", *IEEE Transaction on Signal Processing*, vol. 40, no. 10, pp. 2464–2482, Oct. 1992.

[81]D. B. Shu, J. G. Nash and C. C. Weems, "A Multiple–level Heterogeneous Architecture for Image Understanding", *Proceedings of the Int. Conf. on Application Specific Array Processors*, pp. 615–27, 1990.

[82]E. P. Simoncelli, W. T. Freeman, E. H. Adelson and D. J. Heeger, "Shiftable Multiscale Transforms", *IEEE Trans. on Inf. Theory*, vol. 38, no 2, pp. 587–607, March 1992.

[83]D. Slepian and H. O. Pollak, "Prolate Spheroidal Wavefunctions, Fourier Analysis and Uncertainty I", *Bell System Technical Journal*, vol. 40, pp. 43–64, 1961.

[84]D. Slepian, "Prolate Spheroidal Wavefunctions, Fourier Analysis and Uncertainty IV: Extensions to Many Dimensions", *Bell System Technical Journal*, vol. 43, pp. 3009–3057, 1964.

[85]D. Slepian, "Prolate Spheroidal Wavefunctions, Fourier Analysis and Uncertainty V: The Discrete Case", *Bell System Technical Journal*, vol. 57, pp. 1317–1430, 1978.

[86]J. L. Shanks, "Recursion Filter for Digital Processing", *Geophysics*, vol. 32 no. 1, pp. 33–51 , Feb. 1967.

[87]M. J. T. Smith and T. P. Barnwell III, "Exact Reconstruction for Tree–structured Subband Coder", *IEEE Trans. Acoustic, Speech, and Signal Processings*, vol. 34, pp. 434–441, June 1986.

[88]T. N. Tan and A. G. Constantinides, "Texture Analysis Based on A Human Visual Model", *Proceedings ICASSP 90*, vol.4 , pp. 2137–40, April 1990.

[89]W. B. Thompson and S. T. Barnard, ”Lower–level Estimation and Interpretation of Visual Motion”, *Computer*, vol. 14, no. 8, pp. 57–69, August 1981.

[90]J. T. Tou and R. C. Gonzalez, ”Pattern Recognition Principles”, *Addison–Wesley*, 1981.

[91]S. Treitel and J. L. Shanks, ”The Design of Multistage Separable Planar FIlters”, *IEEE Trans. Geoscience Electronics*, vol. 9, no. 1, pp. 10–27, June, 1980.

[92]R. J. Tusa, ”Visual Cortex”, *American Journal of EEG Technology*, vol. 26, pp. 135–143, 1986.

[93]S. Ullman, ”Analysis of Visual Motion by Biological and Computer Systems”, *Computer*, vol. 14, no. 8, pp. 57–69 , August 1981.

[94]P. P. Vaidyanathan, ”Theory and Design of M–Channel Maximally Decimated Quadrature Mirror Filters with Arbitrary M, Having the Perfect–Reconstruction Property”, *IEEE Transaction on Acoustics, Speech, and Signal Processing*, vol. 35, pp. 476–492, April 1987.

[95]R. L. De Valois, ”Orientation and Spatial Frequency Selectivity: Properties and Modular Organization”, From ’Pigments to Perception’, Edited by A. Valberg and B. B. Lee, *Plenum Press*, 1991.

[96]Y. Wang and S. K. Mitra, ”Motion/Pattern Adaptive Interpolation of Interlaced Video Sequences”, *ICASSP 91*, vol. 4, pp. 2829–32, 1991.

[97]Y. Wang and S. K. Mitra, ”Image Representation Using Block Pattern Models and Its Image Processing Application”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no 4, pp. 321–56, April 1993.

[98]A. B. Watson, ”The Cortex Transform: Rapid Computation of Simulated Neural Images”, *Computer Vision, Graphics, and Image Processing*, vol. 39, pp. 311–327, 1987.

[99]H. R. Wilson, ”Model of Peripheral And Amblyopic Hyperacuity”, *Vision Research*, vol. 31, no. 6, pp. 967–82 , 1991.

[100]C. P. Yeh, ”Depth Perception Based on Fusion of Stereo Images”, *Proceedings of the SPIE, Imaging Technologies and Applications*, vol. 1778, pp. 221–5, 1992.

[101]R. A. Young, ”The Gaussian Derivative Model for Spatial Vision: I. Retinal Mechanisms”, *Spatial Vision*, vol. 2, no. 4, pp. 273–293 , 1987.

[102]R. C. Zhao, J. Kittler, J. Illingworth and I. Ng, ”A New 2D Quadrature Polar Separable Filter and Its Application to Texture Analysis”, *IEEE Symposium on Circuits and Systems*, vol. 2, pp. 1050–53, May, 1990.