2-D Digital Filter Analysis and Application With DEDIP

by
William Alfred Lyle Keddy
B.Eng., University of Victoria, 1988.

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER of APPLIED SCIENCE

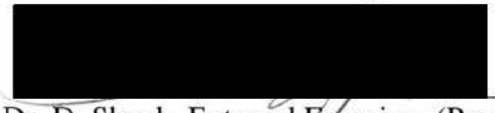
in the Department of
Electrical and Computing Engineering

We accept this thesis as conforming
to the required standard

Dr. P. Agathoklis, Supervisor (Dept. of Electrical and Computer Engineering)

Dr. K. Li, Departmental Member (Dept. of Electrical and Computer Engineering)

Dr. C. Bradley, Outside Member (Dept. of Mechanical Engineering)



Dr. D. Shpak, External Examiner (Royal Roads Military College)

Supervisor: Dr. P. Agathoklis

ABSTRACT

Two-dimensional (2-D) digital signal processing is applied in many areas including digital image processing, medicine, geography, geology and robotics. In many of these applications 2-D linear, shift invariant (LSI) digital filtering is used and it is desirable to be able to analyze all types of 2-D, LSI filters (finite-duration impulse response (FIR), infinite-duration impulse response (IIR) and separable versions of both) and evaluate their performance by applying them to real images. Though many digital image processing (DIP) packages are available, they lack the capability for 2-D filtering beyond simple convolution. The work presented here describes an environment for developing DIP algorithms with particular emphasis on the 2-D LSI digital filtering applications.

DEDIP, a Developmental Environment for Digital Image Processing, has been developed using the traditional two level programming approach. The low level is a C-language software library of DIP and system management functions. The high level is a set of digital image processing and file management tools operating under UNIX. Through representation-dependent system management functions, the data and file structures are hidden. These functions perform access and modification, file input and output, and dynamic memory allocation on an image which is treated as a single data object. This hides the details of the implementation from the programmer and forms the base of a hierarchical library. The library's top level of DIP functions, built using the system management functions, are data and file structure independent which enhances the portability, reusability and reliability of DEDIP.

The DIP algorithms available in DEDIP can be grouped by the operation types of point, intensity and filter. Point operations such as negation and addition of two images are performed on individual pixels. Intensity operations include histogram equalization and linear contrast stretching that modify the histogram of an image. Filter operations include the application of FIR, IIR, separable FIR and separable IIR filter coefficient masks to images as well as the analysis of filter mask characteristics which include the impulse, step, magnitude and phase responses, and the group delays.

The capabilities of DEDIP for the implementation of 2-D LSI digital filters are demonstrated through examples of FIR lowpass, IIR lowpass and FIR fan filters, first directional-derivative edge-detection operators and the Laplacian of a Gaussian second-derivative edge-detection operator. In each example the particular filter coefficient mask is analyzed, the filtering operation on a test image is performed and the filtering result is discussed.

Examiners

Dr. P. Agathoklis, Supervisor (Dept. of Electrical and Computer Engineering)

Dr. K. Li, Departmental Member (Dept. of Electrical and Computer Engineering)

Dr. C. Bradley, Outside Member (Dept. of Mechanical Engineering)

Dr. D. Shpak, External Examiner (Royal Roads Military College)

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank Dr. Agathoklis for his patience and support, especially during the writing and editing of this thesis; my family for their gentle encouragement; and the members of the Department of Electrical and Computer Engineering for their support and indulgence with the debugging of DEDIP and the writing of this thesis.

Trademark Acknowledgements

Dedication

Dedicated to the memory of Sharon Clarke,
a good friend and supporter.

List of Acronyms

1-D	One Dimensional
2-D	Two Dimensional
CAD	Computer Aided Design
CAT	Computer Aided Tomography
CLI	Command Line Interpreter
DEDIP	Developmental Environment for Digital Image Processing
DIP	Digital Image Processing
DSP	Digital Signal Processing
FIR	Finite-duration Impulse Response
GUI	Graphical User Interface
IIR	Infinite-duration Impulse Response
IO	Input and Output
LSI	Linear, Shift Invariant
SAR	Synthetic Aperture Radar
SFIR	Separable Finite-duration Impulse Response
SIIR	Separable Infinite-duration Impulse Response
VLSI	Very Large Scale Integrated

Chapter 1

Introduction

1.1 Introduction

Digital image processing (DIP) is a rapidly expanding field with applications found in many areas. In manufacturing, computer vision techniques are used to analyze images to obtain information such as visual flaws, visual measurements and features for object identification. This information is then used for purposes such as quality control, character recognition, part location, target identification and robotic navigation [1-3]. In diagnostic medicine, Computer-Aided Tomography (CAT), digital X-ray and ultrasound are familiar image processing applications [4-7]. Less familiar areas found in medical laboratories and research use image processing and pattern recognition to match and count blood cells from micrographs and for protein molecule identification from two-dimensional (2-D) electrophoresis gels [8-10]. In geography, earth mapping of mineral, crop and land resources through satellite imagery techniques such as Synthetic Aperture Radar (SAR) construct very detailed information on the geographic distribution of resources from their optical and electro-magnetic properties [11-14]. Recent interest has been in global environmental changes such as climatic variations and ozone depletion. In geology, seismic exploration uses acoustical techniques to develop pseudo cross-sectional views of the earth's geologic layers [15].

DIP research areas include digital filter design and implementation, visualization, image compression and seismic signal processing. In the performance of this research, tools for the development and evaluation of algorithms are required. The Developmental Environment for Digital Image Processing (DEDIP) was developed to provide a basic environment for DIP algorithm development. In section 1.2, this environment and some of the design criteria for DEDIP are presented. In section 1.3, 2-D digital filters and the

contribution of DEDIP to their application and analysis are introduced. An overview of this thesis is presented in section 1.4.

1.2 DIP Algorithm Development Environments

DIP algorithm development environments are generally composed of tools that provide the ability to manipulate images through predefined operations. Many such tools are available with capabilities that range from very detailed analysis of algorithms and their implementations to basic image manipulation.

Digital image-processing algorithms can be developed using Computer Aided Design (CAD) systems [16-20]. Under a CAD system, DIP algorithm implementations can be evaluated in various combinations of hardware and software. Generally, CAD systems provide a framework that combines a database of algorithms and implementation parameters with a graphical environment to manipulate the database entities. Graphical block diagrams of DIP algorithms can be edited within a CAD framework. From this block diagram the algorithm can be evaluated through simulation. Using the block diagram as an algorithm specification, an implementation in software such as computer generated-programs, or in hardware, such as Digital Signal Processing (DSP) processors or Very Large Scale Integrated (VLSI) chips, can be realized. The framework provides a common environment for analysis, simulation and implementation.

The main disadvantage of CAD systems is that their cost is quite large in software as well as in computer system requirements. A second disadvantage is that digital image processing through a simulation can be time-consuming as the algorithm is being evaluated as well as the implementation. This can be minimized by configuring different levels of analysis detail. Currently, CAD systems are not directly capable of DIP algorithm development since the development of a DIP algorithm database would be required. CAD systems are better suited to designing hardware implementations of DIP algorithms.

The lack of DIP algorithm databases can be easily overcome by incrementally adding algorithms as research progresses but the good overall benefit provided by a CAD system is offset by its cost. Many DIP software packages are available for general purpose computers that provide different levels of capability at a reasonable cost [9, 10, 21-34]. These applications have been developed for the narrow class of algorithms based on pixel manipulation which includes some digital image and graphical processing. As these

packages do not have the very general capabilities of a CAD system, they are much smaller, cost less and are easier to use. For these reasons DIP applications and packages are currently better suited for DIP algorithm development.

1.2.1 Application User Interfaces

The user interface plays an important role in the usability of a development environment. Application interfaces can generally be divided into two groups: menu-driven and interpreted [23]. The menu-driven interface provides a fixed number of options for the user to select from and is very useful to novice users because all available operations are displayed. Menu-driven interfaces can be graphical or textual. Some well known menu-driven, Graphical User Interfaces (GUIs) are X-Windows™, MicroSoft Windows™ and Apple Macintosh™. Gopher is an example of a textual menu-driven interface [35].

Menu-driven interfaces are the easiest to learn and use mechanism for a user to interface with a computer. One of the problems with menu-driven interfaces is that the number of displayed options is limited. Techniques used to increase the number of options are pulldown and popup menus and forms but many traversals through menu hierarchies may lead to frustration. As the human-computer relationship evolves, better metaphors for the interface will be found and users will become more familiar with common metaphors.

Though menu-driven interfaces provide the easiest to learn interface, more experienced users generally prefer an interpreted environment. An interpreted environment allows for a large number of options, avoiding the nested menus, at the cost of the user having to remember them as they are not displayed. Experienced users, through their experience, have learned the commands they most commonly use and prefer to skip the menu display step by going directly to the selection operation by entering the command.

An interpreted environment can be implemented as a custom environment such as Matlab [36] or under the operating system's Command Line Interpreter (CLI). The CLI approach has been called a traditional, two-level approach to programming where the compiled libraries contain low-level programming and the programs are high-level programming [21]. The low and high levels refer to the level of programming abstraction. The lower the level the greater the detail required in the program implementation.

This traditional approach using the CLI has the benefit over a custom interpreted environment in that the operating system provides basic mechanisms for handling errors, input and output (IO), file system structure, dynamical linking of new operations and hardware interfaces through device drivers. Also, software development tools such as compilers, debuggers and profilers are readily available. A custom environment, whether interpreted or menu-driven, would be required to supply these tools or mechanisms to access the system operations. Though a custom environment would be required to supply these mechanisms, it has the advantage in that better mechanisms which are more relevant to an environment's purpose could be included.

1.2.2 Implementation Efficiency

The design trade-offs for evaluating DIP algorithm implementations are test image size, programming ease and execution time for the implementation under evaluation. The efficiency of implementation is based on the hardware capabilities available and the tuning of software to take the greatest advantage of them. Considering a general purpose computer, the most efficient use of hardware is an optimized and compiled program. A DIP algorithm researcher must balance these trade-offs to maximize research efficiency.

The easiest to use development environment is an interpreter which allows pixel manipulation through basic high-level operations such as addition, subtraction, multiplication, division, etc. It provides an easy to use interface for creating DIP algorithm implementations but at the expense of high execution time. Each basic operation has the overhead of being interpreted before it is executed. With small images an interpreter is fine for evaluating an algorithm and provides a quick implementation time. As the size of the test image gets larger, the overhead time contributed by interpreting adds considerable delay between evaluation steps. Compiling the algorithm implementation makes its execution faster but the overall algorithm implementation time is increased by the compilation time.

In general using an interpreted language is good for small test images and for the testing of simple algorithms. Once these simple algorithms are developed they can be combined and compiled for faster execution. From the researcher's point of view, the extra overhead of compiling may not be as efficient and an interpreted environment may be better suited to algorithm development. The researcher must determine if an interpreted or a compiled environment is more efficient depending on the computational complexity and the test images used.

As DIP is very data intensive, DEDIP was developed based on a compiled environment to be used with real images of arbitrary size. With larger images obtained from frame grabbers and scanners, the compilation time is recovered through much faster execution.

1.2.3 DEDIP

DEDIP has been developed using the traditional, two-level approach to software development. It is composed of a function library and a set of programs. This allows a DIP researcher to combine existing programs to process digital image data files from the CLI or add new functions and programs. The library and programs have been developed using structured programming techniques of modularity and information hiding to create a flexible, extensible, easy to use, and efficient environment. One method of applying modularity is functional abstraction where a function is defined by its input and output parameters and its operation but not its implementation. Through information hiding, the programmer does not have direct access to the data structure [37-39]. Access to data structure fields is through an interface function. This allows file and data structures to be modified at a later date with minimal or no effect to existing programs while freeing the DIP developer from the details of the data structure implementation.

These software design methods used in the development of DEDIP minimize the effect of changes on existing functions and programs and allow modular testing and validation of DIP algorithms.

1.3 Digital Filters

Digital filters are a common method of processing 2-D digital images which are composed of signal and noise components. The signal component is the desired information which is composed of edges, textures and constant contrast regions. Noise component is any undesired features or effects in images. The purpose of filtering is to enhance the signal component and attenuate the noise component.

A digital image is a 2-D discrete signal which can be represented by a frequency spectrum. Linear digital filtering achieves its purpose of enhancing the desired information through the manipulation, modification and reshaping of the frequency spectrum of

a digital image [40]. To generalize, edges and sharp transitions in the intensity level of an image contribute heavily to the high frequency components of the image's frequency spectrum. Conversely, the constant intensity regions with slight variations in the intensity levels corresponds to the lower frequency components.

Noise introduced in digital images by imaging transducers and transmission channels tends to be spatially independent [40]. The energy content of the noise is often concentrated at higher frequencies than the energy content of the image. Attenuating the high frequency components by lowpass filtering can therefore improve the image's appearance.

DEDIP has basic linear filter algorithms in its library and programs. These algorithms are software implementations of Finite-duration Impulse-Response (FIR) and Infinite-duration Impulse-Response (IIR) filtering and separable versions of both. The ability to analyze the filters using the frequency response, group delay, impulse response and step response has also been implemented.

1.4 Organization

This thesis presents the DEDIP system and its 2-D digital filtering aspects. DEDIP has the ability to analyze filter coefficients and to apply them to real images which can be used in the course of research in 2-D digital filter realization.

An overview of the DEDIP system is presented in chapter 2. Structured design techniques and how they are used in DEDIP are briefly discussed. The description of the DEDIP system structure, data access techniques, functional library (low level), and programs (high level) follows. X-Windows programs for displaying images and for filter coefficient analysis are presented.

In chapter 3, the theoretical background of 2-D digital filters through the description of basic 2-D signals and linear, shift-invariant systems properties is discussed. The convolution summation, transfer function and difference equations which are used to characterize 2-D FIR and 2-D IIR filters are described. Frequency response, group delay, impulse response and step response are also briefly described, as well as their contribution to understanding a filter's characteristics.

Examples of the DEDIP tools described in chapter 2 and 3 are presented in chapter 4. The first example has a FIR and an IIR lowpass filter applied to a noisy image which shows the possibilities given by DEDIP for evaluating the performance of 2-D filters on real images. The fan filter's directional capabilities are demonstrated and analyzed in the second example. The third and fourth examples demonstrate and analyze edge-detection operators. The third example considers the simple difference, Prewitt, Sobel and Rosenfeld-Thurston first-derivative edge-detection operators. These operators, implemented as correlation templates, are analyzed and applied to a test image. The Laplacian of a Gaussian operator is a second-derivative edge-detection operator which is implemented as a separable filter.

The contribution of this thesis and the filtering and filter analysis capabilities of DEDIP are summarized in chapter 5. Future directions in the development and use of DEDIP are also discussed.

Chapter 2

A Developmental Environment for Digital Image Processing: DEDIP

2.1 Introduction

Digital Image Processing (DIP) involving real images is very data intensive and it is generally more efficient to test algorithms under a compiled development environment rather than an interpreted one. In addition, advanced programming techniques can be used to further enhance the efficiency of compiled implementations. These techniques, although easily understood by skilled programmers, can be very confusing for occasional or novice users. In an effort to reduce the complexity associated with these advanced programming techniques, modern programming methods can be used to hide the details of these techniques. DEDIP was developed using such methods as modularity, structured programming and information hiding to provide an environment for DIP algorithm development.

DEDIP is composed of DIP function libraries and a set of DIP programs [21]. This allows a DIP researcher to apply existing programs to process digital image data files from the CLI or to add new library functions and programs. The DIP programs are based on DIP library functions. The DIP library functions were developed using structured programming methods and built with modules developed using information hiding to hide the detail of the data structure implementation and access.

An overview of the DEDIP system is presented in this chapter. The methods of modularity, structured programming and information hiding are briefly discussed in section 2.2. In section 2.3, the modular structure of DEDIP is presented. The low-level implementation-dependent system-management functions, the imaging-processing func-

tions and the DEDIP programs are briefly presented. X-Windows applications for displaying images and for analysis of 2-D digital filter coefficients are described in section 2.4. The significant features of DEDIP are summarized in section 2.5.

2.2 Modularity and Structured Programming

The understanding of complex systems is facilitated by using abstraction. Abstraction is the process of decomposing what is being studied into its major elements while ignoring the details [37, 38]. Modularity is a form of abstraction. A program can be decomposed into subprogram components called modules. The development of a program can be simplified by designing, implementing and validating the modules independently. Understanding is improved by only having to understand the module being developed; not the details of the complete program.

Generally a module is a collection of functions and data with a common goal [37]. A module has functions and data accessible within its scope and global functions providing external access to the module. In the following subsections, top-down design, bottom-up design and information-hiding methods will be discussed. The top-down design method uses a narrow definition of a module as being a single function with data within the scope of the function. The bottom-up design and information hiding methods use the more general definition of a module as having related functions and data within the same module.

In structured programming, constructs are defined and rules for applying them to produce blocks of code that are easier to understand are given [37-39, 41]. Thus novice and occasional users can review the code block and the logic used. The three constructs of sequence, selection and repetition provide the basic structure elements and are used in such a way that each construct has one entry and one exit point.

Applying structured programming constructs and rules to a program allows it to be viewed as a collection of modules each with one starting and one ending point. A very clear path of control flow can be followed into and out of a module, making each module easier to understand.

2.2.1 Top-Down Design

A systematic method for translating the abstract description of the problem into a program structure is required. One method, known as top-down design or functional decom-

position, is a good example of software abstraction and the ease of understanding that results [37, 38].

The top-down method first constructs an abstract description of the overall problem. Next, this description is decomposed into its major components. Each major component is defined as a function. The decomposition process continues until a function of appropriate detail is reached. The detail will be such that the function will be easily understood in terms of the programming language primitives or the operating system's library functions.

Different metrics are used to define "appropriate detail." One is an arbitrary number of lines of code while another is to use quantitative measures of software complexity such as cyclomatic complexity measure of McCabe [37]. Through abstraction, the detail of a function can be reduced by defining it in terms of other functions. With the mechanism and operation of the sub-functions known, the function being studied can be understood in terms of the sub-functions.

Top-down design realizes the structured programming rules of single entry and exit points inherently through the function-calling mechanism.

2.2.2 Bottom-Up Design

Bottom-up design is based on the observation that some lower-level functions perform the same or similar purpose [38]. This alternative design method starts by considering the low-level primitives and building software layers to provide common interface functions for access by higher-level modules. An example of a bottom-up design is a link-list manipulation library. The link-list data structure and basic functions that operate on it are defined within the module. The common interface functions can then be used in a top-down program design as programming primitives. Bottom-up design is generally used in the development of libraries with interface functions accessing implementation dependent features.

The bottom-up design method is very effective in isolating hardware and software implementation detail through an interface library. With an interface library, execution of the operation requires knowledge of the interface function's calling parameters and its functionality but not its implementation details. The implementation can be in software, or special-purpose hardware for better performance, but the result of executing the function is the same in either case. Through a standard interface mechanism, independence

between the algorithm software and its special purpose implementation requirements can be made.

2.2.3 Information Hiding

Information hiding is the basis for a very powerful software development technique [37, 38]. Data abstraction groups data fields into a single data structure and functional abstraction describes the mechanism and the functionality of an operation without implementation details. Information hiding combines these two methods. For a data structure, information hiding encapsulates it and functions for its access and modification in the same module [38]. The encapsulated data can not be directly accessed. Only through the associated functions can the data be accessed or modified.

Programs developed using information hiding can be reused through many different data representations and hardware systems. It is a simple matter to change the data representation and functional implementation to a new format while maintaining the same higher level functional interface mechanism. Therefore, the reusability, portability, maintainability and performance of the higher level functions and the program are improved [37, 38].

An example of the usefulness of information hiding in image processing is through the use of different data structures. The most common and general purpose representation of image data for a rectangular two-dimensional array of intensity levels is called an iconic image. Alternative image data structures that use linked lists of pixel sub-groups have been investigated [32] and shown to increase processing speed and reduced data storage requirements. By grouping the data into a data structure that represents the image and by restricting data access to functions associated with the data structure, information hiding can be used to provide a common access mechanism for obtaining information about the image. The actual data structure is never accessed directly by the programmer. Higher level modules that use this common access mechanism to implement algorithms can use the same implementation for different image representations.

2.2.4 Structured Programming in DEDIP

DEDIP was developed using a meet in the middle program design method [38]. Data structures using information hiding and system management functions were developed using the bottom-up design method. This provided a basic library from which a basic set of DIP application tools were developed using top-down design. The DIP tools were

designed to parallel the DIP library functions with similar parameter lists and functionality. Multiple DIP tools can be applied to an image as a sequence for quick evaluation of an algorithmic combination. Later, the combination could easily be converted to a single application using the mirroring of the DIP tools and library functions. The DEDIP system is described in more detail in the following section.

2.3 DEDIP System

The DEDIP system uses the UNIX CLI at the high level and the C programming language at the lower level. This combination was not so much selected as it was the available development environment. Even so this combination provides a good general development environment.

The UNIX operating system provides mechanisms for error handling, file and device IO, virtual memory, and adding new programs as well as a rich set of software development tools [42,43]. Uniform file handling provides a common interface mechanism to files, devices and interprocess communications where each is considered a data stream of sequential information. Virtual memory is another UNIX feature that allows storage space on a hard disk drive to be used as an extension to the main computer memory. The virtual memory allows images much larger than the main memory to be processed at a much lower cost than having the equivalent amount of main memory. These features make UNIX a good choice for a general purpose DIP development environment.

A UNIX CLI is ***csh*** (C-shell) which provides the user with mechanisms to sequence, start, stop and suspend a program's execution. The ***csh*** redirection operators are the basis for two important concepts of the UNIX filter and stream redirection. UNIX utility programs (called tools) are designed as simple transformations. The UNIX filter and stream redirection concepts can be used to sequence simple DIP tools to perform a more complicated algorithm. If the sequence performs adequately it can be translated to a compiled tool for faster, more efficient execution.

DEDIP uses the mechanisms provided by the C programming language for defining functions and data structures, creating new data types and manipulating memory to implement new data types representing a digital image and for defining functions to operate on them. The access to information with the new data types is restricted through function-like mechanisms resulting in data representation-independent algorithm implementations.

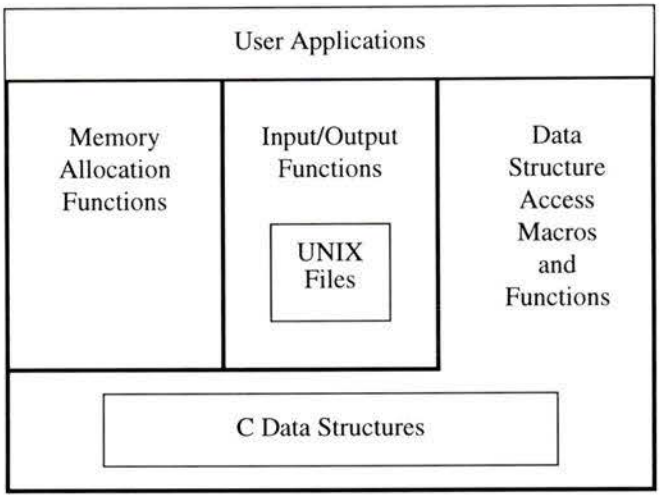


Figure 2.1 DEDIP system management functional hierarchy.

The UNIX operating system and C programming language provide very useful features and mechanisms required for a general DIP algorithm development environment. DEDIP uses these features to develop an environment based on a hierarchical, modular design structure as shown in figure 2.1. The applications are DIP programs and functions which are developed using the system management function modules of data structure access, file input and output (IO) and memory allocation.

In the following sections the system management functions, the DIP functions and the DIP tools are described.

2.3.1 DEDIP System Management Functions

The system management functions which form the base of the library hierarchy are divided into three modules shown in figure 2.1. The access to data structure fields is restricted to being through special macros resulting in data structure independence. The macros are used for maximum speed through direct substitution eliminating the function call overhead. The access mechanisms not only ensure data structure independence but they also shield the user from data representation details.

Similarly, file structure access is through the IO module functions which provide access to image data stored in files as a single data object. These IO functions transfer image data between a file and an encapsulated data structure. This allows the data file structure to be independent of its use and hidden by the data structure access functions.

The file format is a custom one developed for DEDIP. This format can be changed with no effect on the DIP functions. The IO modules would require modification for a new format but the algorithms would be unaffected. The specific file format is unimportant. Also, with the UNIX's uniform file handling, the "file" could be a file, an interprocess stream, a frame buffer hardware or some other special purpose hardware. The higher levels of functional abstraction are independent of the data structure demonstrating DEDIP's changeability, reusability, maintainability, portability and ease of use.

Memory management functions simplify the allocation of memory arrays. They are used by the IO functions and use the data structure access macros to maintain data structure independence. System management functions provide a standard interface mechanisms that form the basis of an extensible system.

2.3.1.1 Data Representation and Access

DEDIP uses information hiding to define and access aggregate data types of IMAGE, REAL_2D and COMPLEX_2D which represent 2-D 8-bit unsigned integer arrays, 2-D floating point arrays and 2-D complex floating point arrays, respectively. These aggregate data types are C structures that are an abstract representation of the image or array and include pointers to dynamically allocated memory for the image or array elements. Macros have been defined to access these new data types. These macros add a level of data abstraction that have two main advantages: data representation independence and ease of user interface.

The data structure fields can be accessed only within the scope of its access macros. Data representation independence allows the actual representation of the data to be changed with little or no effect on the higher level library functions or the tools. In the worst case, the function library and tools would need recompiling. In this situation, the source code of the functions and tools is reused easily after major changes to the data representation. This makes programming more manageable by hiding the complexity of C data structure access.

The use of pointers and dynamic memory allocation can be very complex and confusing. The data access macros release the occasional user from this encumbrance by hiding the complexity of the data structure, indirection and pointer arithmetic.

Possible problems exist with the use of macros. Type checking is performed after the macro substitutions which can result in errors caused by type mismatches and incor-

rect operation where the automatic type conversion does not perform correctly. Inline functions with more rigorous type checking would be a better mechanism but are currently compiler dependent.

The access macros are divided into four groups: public header access, private header access, getting array element values and setting array element values. The public header access group allows access to general information about the data structures. Some basic access operations are shown in table 2.1 where the type OBJECT denotes any one of the three abstract types. With changes to a data structure, the public data access macros maintain their functionality and data integrity.

WidthOf()	return the width of the OBJECT
HeightOf()	return the height of the OBJECT
DataTypeOf()	return the OBJECT type of IMAGE, REAL_2D or COMPLEX_2D
DepthOf()	return the depth of an image or precision of floating point array element in bits

Table 2.1 Public header access macros.

The private header access macros are primarily used to set header parameters. They are considered volatile because data structure changes may not maintain the functionality and data integrity of these macros. The private header access macros are used for implementation-dependent data structure access such as obtaining pointers to data fields or setting private structure fields with information about the structure, which should not normally be under user control. The meaning of a returned pointer can change with different implementations so caution must be observed when private header access macros are used.

The get and set array access macros are used to manipulate the array elements. The get access macros are listed in table 2.2. There are corresponding set access macros where Get is substituted with Set and the operation sets the element value.

GetMappedPixel()	get an element value from an IMAGE array
GetReal2dEntry()	get an element value from an REAL_2D array
GetComplex2dEntry()	get a complex element value from a COMPLEX_2D array

Table 2.2 Get array access macros.

GetMagnitudeComplex2dEntry()	get the magnitude of a complex element value from a COMPLEX_2D array
GetPhaseComplex2dEntry()	get the phase of a complex element value from a COMPLEX_2D array
GetRealComplex2dEntry()	get the real part of a complex element value from a COMPLEX_2D array
GetImaginaryComplex2dEntry()	get the imaginary part of a complex element value from a COMPLEX_2D array

Table 2.2 (Continued) Get array access macros.

2.3.1.2 File Input and Output

The file IO functions perform reading and writing of complete data structures. The header and data information IO are handled by low level functions not available to the user. There are five user IO functions for each array type of IMAGE, REAL_2D and COMPLEX_2D shown in table 2.3.

DReadImage() DReadReal2d() DReadComplex2d()	read an OBJECT from a FILE pointer
DWriteImage() DWriteReal2d() DWriteComplex2d()	write an OBJECT to a FILE pointer
DLoadImage() DLoadReal2d() DLoadComplex2d()	read an OBJECT from a string file name
DSaveImage() DSaveReal2d() DSaveComplex2d()	write OBJECT to a string file name in binary format
DPrintImage() DPrintReal2d() DPrintComplex2d()	write OBJECT to a string file name in ASCII format

Table 2.3 Input/Output functions.

The *read* and *write* functions operate on a file referenced by a C language FILE pointer. The *load*, *save* and *print* functions reference the file by the name of the file. The input functions *read* and *load* also can obtain data from files compressed by the UNIX ***compress*** program which reduces file size using lossless Lempel-Ziv data compression. The purpose of these functions is to bundle the IO of images and arrays into one step operations on a single data object.

2.3.1.3 Memory Allocation

The memory management functions continue the single data object concept and introduce dynamic memory control to DEDIP. A complete data structure with an array of the specified width and height can be allocated with a single function call. Table 2.4 lists the memory management functions that allocate and deallocate (free) memory.

DAllocImage() DAllocReal2d() DAllocComplex2d()	allocate memory for an OBJECT
DFreeImage() DFreeReal2d() DFreeComplex2d()	free memory allocated for an OBJECT

Table 2.4 Memory allocation functions.

2.3.2 DEDIP Digital Image Processing Functions

The DIP functions are divided into five categories based on processing classifications of point, intensity, filtering, utility and data conversion operations.

Point operations are performed on a single pixel or entry where no dependency exists with neighboring values. Table 2.5 lists the point operations.

DAndImage()	logically AND two images
DOrImage()	logically OR two images
DXorImage()	logically XOR two images
DAddImage()	ADD two images with overflow and underflow truncation
DNegateImage()	Negate the pixels of an images
DScaleReal2d()	Scale and Offset the entries of a REAL_2D array

Table 2.5 Basic image point operations.

Intensity operations could be considered a form of point operation. But, they are often based on the image's probability density function (histogram). Intensity operations modify an image through adjustment of its intensity levels. Table 2.6 lists the intensity operations.

DLinearContrastStretch()	linearly stretch the pixel intensity within specified limits
DThresholdImage()	threshold a greyscale image into a monochrome image

Table 2.6 Intensity operations.

DHistogramEqualization()	make the cumulative density function of an image histogram approximate unity

Table 2.6 (Continued) Intensity operations.

Filter operations are neighborhood spatial algorithms. Each pixel or array entry in the result depends on a neighborhood in the input array. Four filtering algorithms of finite-duration impulse response, infinite-duration impulse response, separable finite-duration impulse response and separable infinite-duration impulse response have been implemented. Special subtypes of the REAL_2D data type have been added for four filter coefficients array types shown in table 2.7.

FIR	finite-duration impulse response filter mask
IIR	infinite-duration impulse response filter mask
SFIR	separable finite-duration impulse response filter mask
SIIR	separable infinite-duration impulse response filter mask

Table 2.7 Filter mask types.

The weighted averaging of an image with the weights provided in a REAL_2D array has also been implemented. The array is assumed to have odd dimensions for an operation about the referenced point. This allows uniform neighborhood processing about the reference point.

The frequency response, group delay, impulse and step responses for all filter mask types can be performed. Table 2.8 lists filtering and analysis functions.

DWeightedAverageReal2d()	perform spatial averaging.
DBoundaryCondition()	convert an image to a REAL_2D array and add boundary conditions for filtering
DFilterReal2d()	perform FIR, IIR, SFIR or SIIR filtering on a REAL_2D array
DFrequencyResponse()	calculate frequency response of filter mask
DGroupDelay()	calculate the group delay of a filter mask
DImpulseResponse()	calculate the impulse response of a filter mask
DStepResponse()	calculate the step response of a filter mask

Table 2.8 Filter Operations and analysis.

Utility functions providing general operations such as partitioning and magnifying an image are listed in table 2.9

DInterpolateImage()	magnify an image to a specified height and width using interpolation
DMagnifyImage()	magnify an image by a scale factor and aspect ratio using interpolation
DPixelReplicate()	magnify an image by pixel replication
DPartitionImage()	partition (crop) an image into a smaller image
DPartitionReal2d()	partition an array into smaller array
DSubSampleImage()	reduce the image by averaging of four pixels
DCopyImage()	copy an image (or part of) onto another image
DRotateImage()	rotate an image clockwise by 90 degrees

Table 2.9 Utility functions.

Conversion functions provide the translation from one type of array to another. They have been provided primarily for file type conversion between the program tools.

2.3.3 DEDIP Digital Image Processing Tools

The DEDIP tools are DIP programs that use the UNIX filter concept. The filter transformations are basic DIP operations that mirror the functions described in the DEDIP library. When a program is executed under the ***csh*** CLI, default file streams of *stdin* for input, *stdout* for output and *stderr* for errors are opened.

If command line execution flags that specify the image or mask files to be used are not found then the input is looked for on *stdin* file stream. Other flagged parameters have internal default values.

The tool command line options have the same parameters as the functions. The exception to this are the output and verbose flags which are used to specify the output file name and to display extra comments on *stderr* during execution, respectively. Normally, the tools do not display any information during execution.

All tools display a usage message following incorrect command line options. The usage message for the histogram equalization tool is:

```
DEDIP: DHistogramEqualization [options]
        -i input   : input image file
        -o outfile : output image file
        -v         : verbose
```

The implementation of tools should be performed using a consistent four step process. The tool should only 1) process the command line flags, 2) read the required input

files, 3) call the DEDIP function and 4) store the output. The function called in step 3) should have the same name and functionality as the tool for the purpose of mirroring at the two interface levels. The development of new tools is simplified by copying an existing tool and slightly modifying it to reflect the command line flags, inputs and outputs. The algorithm development should be performed in the mirrored function. This technique simplifies the development overhead for a tool as well as develops a new function that can later be placed in the library.

2.4 X-Window Applications

The DEDIP X-Windows applications consists of two tools for interactively displaying images and filter analysis. DXDisplay is an image display tool which can also perform some basic interactive operations. DXFilterAnalysis is an interactive tool for obtaining the frequency, group delay, impulse and step responses for a 2-D filter mask.

X-Windows is a networked, graphical user interface working as part of a networked computing environment. Two basic computer environment operations are starting applications and accessing data files. In the following, some mechanisms for these operations, their problems and how the DEDIP X-Windows applications have worked around them are discussed.

Application Execution

With textual interfaces, applications are started by specifying the name of an executable file on the command line. The X-Windows system can be used as an extension to a textual interface terminal, with the ability of having multiple terminals on the same display. Each terminal window can be accessing a different directory and running different programs on different computers. When the X-Windows GUI is used in this way, applications can be started in the traditional method of textual interfaces.

More user friendly interfaces can be developed using GUI techniques of popup menus, menu bars and form windows [44, 45]. A service program running under X-Windows, called a window manager, is used to control the size and position of windows on a screen background called the root window. The window manager has features that allow the mouse button actions in the root window to be defined by the user. Through the use of popup menus, predefined commands can be executed by the window manager to provide a simple, easy to implement method of starting applications.

Default File IO

Using a window manager to start application execution has some side effects with respect to default file IO of UNIX. Under UNIX, *stdin*, *stdout* and *stderr* file IO streams are defined when any program is executed. Under a window manager, applications are still executed under UNIX but *stdin*, *stdout* and *stderr* have different meanings. When using a textual interface, such as the **csh** shell, the default file IO can be used normally. When an X application is executed from a window manager the *stdin* has effectively no meaning. The application must supply the input from graphical mechanisms such as text entry forms, buttons, selection lists, etc.

Another problem not encountered with a textual interface is input context or focus. With the ability to have many applications on a display, which application is receiving the input from the keyboard? In general the window manager uses one of two methods controlled by the mouse. With the "point and click" method the mouse is placed on the application and a mouse button is clicked to give the application focus for keyboard input. The second method sets the focus by the positioning the mouse on the application or in a particular region of the application being displayed.

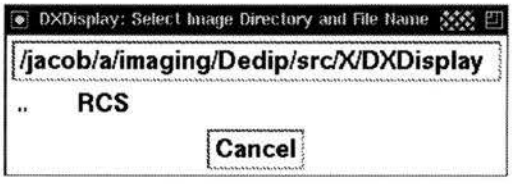
Messages printed on *stdout* and *stderr* are displayed in the window manager's console, logged to a file or just discarded. A popup dialog window must be used as an equivalent to *stdout* and *stderr* to display messages from the application. The application programs must control the file IO.

File Locating

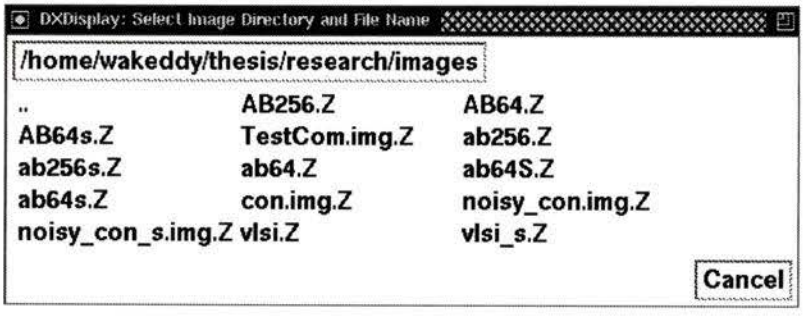
Locating data files under a window manager is also a problem. The CLI environment has a current directory associated with it which can be change with the "cd" (change directory) command. Using multiple terminal windows, each window may have its own current directory. Usually, the directory that the window manager is started in is the current directory for applications started by it.

There are different solutions to this problem. The solution used by the DEDIP X-Windows applications uses a filtered, selectable list of available files. Figure 2.2-a shows files of type IMAGE found in the directory "/jacob/a/imaging/Dedip/src/X/DXDisplay". Clicking the mouse on a file's name will select the image file and continue execution of the application.

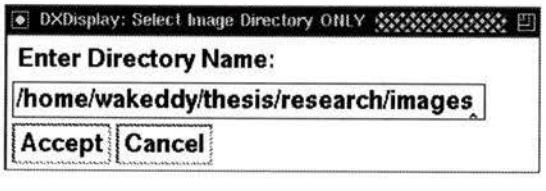
The names with the suffix "/" indicates a directory and "../" indicates the parent directory. Selecting a directory will change the current directory to the one selected and



(a)



(b)



(c)

Figure 2.2 DXDisplay: file selection.
(a) Initial directory popup window. (b) Selection from current directory window.
(c) Keyboard directory path entry window.

displays a list of the IMAGE type files and directories in the new current directory as displayed in figure 2.2-b.

An alternative directory selection method is direct text entry. The top line displaying the directory listed can be clicked on by the mouse to popup a direct entry text window shown in figure 2.2-c that can change the current directory. Accepting the text entry changes the current directory and lists the IMAGE type files and directories from the new directory.

Mixed Textual and Graphics Interfaces

With the restrictions on the default IO, UNIX pipe-line sequencing techniques used by DEDIP are not available under a window manager. An alternate to using a full GUI would be to use a mix of textual and graphical user interfaces. The multiple text windows available under X-Windows can be used to edit, compile, execute and sequence programs using the *csh* pipeline techniques with image display and surface plotting under the GUI. The GUI can facilitate image display and surface plotting.

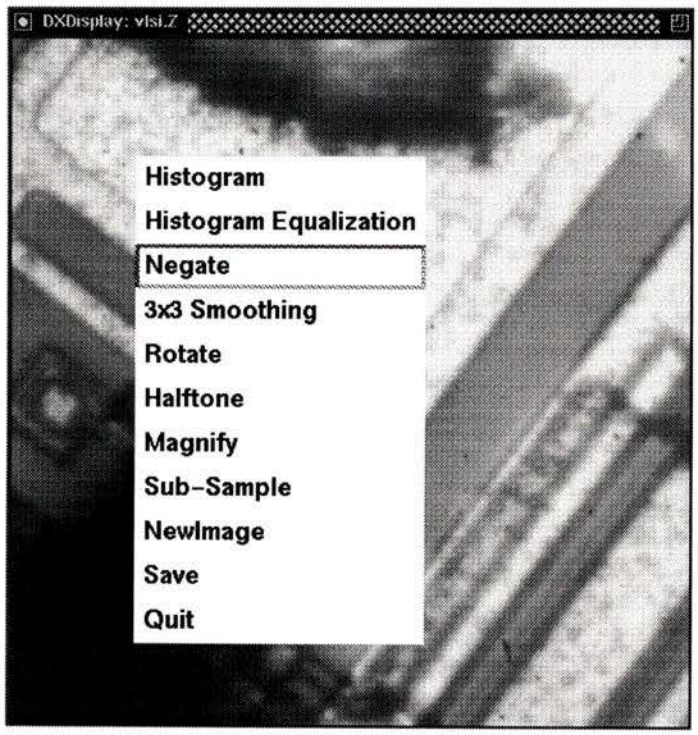
2.4.1 DXDisplay

DXDisplay has command line options used to specify an input image file. If the specified file is not found or no input file option given, the method described in "File Locating" on page 21 and shown in figure 2.2 will be used to get the input image file name.

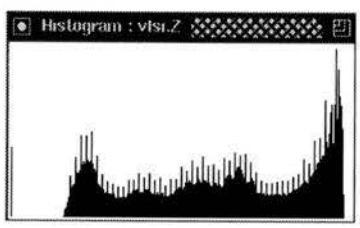
DXDisplay can perform a few basic, interactive operations such as: compute and display the image's histogram, histogram equalization, negation, 3x3 smoothing (averaging), rotation, halftone, magnification by pixel replication, 2x2 sub-sample, redisplay the original image and save the currently displayed image. These operations are very simple and minor image enhancement operations that can be completed fairly quickly for qualitative, interactive evaluation.

The operations menu shown in figure 2.3-a is selected by pressing the left mouse button anywhere on the displayed image. The mouse can be dragged down the menu to select the desired operation being highlighted. Figure 2.3-b shows the histogram window displayed by the histogram operation. This window remains open and is updated for any changes to the image due to another operation. The menu operations are performed on the currently displayed image.

Interactive partitioning can be performed by pressing the middle mouse button and dragging a rubberband box that defines the region to be partitioned. The partition window is popped up to display the offset and size as the rubberband box changes. When the



(a)



(b)

Figure 2.3 DXDisplay: image window.
(a) Image display window with popup menu. (b) Histogram display window.

button is released the box size is defined. The partitioning is performed after selecting the OK button. Figure 2.4-a shows the partition window and the box after the button is released. Figure 2.4-b shows the result of the partitioning.

2.4.2 DXFilterAnalysis

DXFilterAnalysis provides a GUI to DEDIP filter analysis functions. It uses a form window to interactively set analysis parameters and control execution and termination of each of the analysis functions. The DEDIP analysis functions are DFrequencyResponse, DGroupDelay, DImpulseResponse and DStepResponse.

DXFilterAnalysis uses the directory search described above to find the filter mask for analysis. Only REAL_2D DEDIP data files representing filter or cross-correlation masks will appear in the directory window. When the first input mask is found, the parameter form window shown in figure 2.5-a is displayed. The parameter fields shown as grey boxes are editable windows.

DFrequencyResponse function is initiated by selecting the Frequency Response option and calculates the magnitude and phase responses in the ω_1 and ω_2 directions between the specified frequency limits.

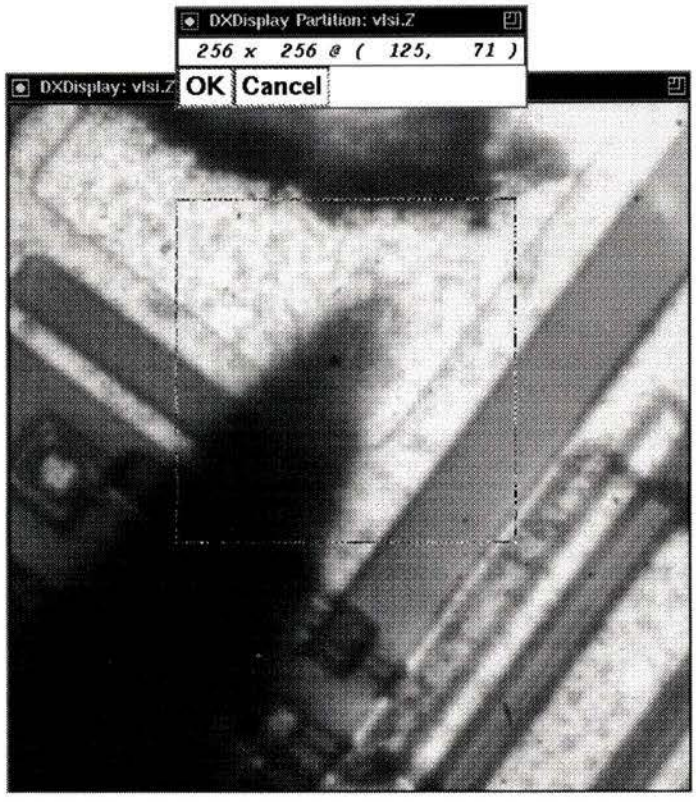
DGroupDelay function is initiated by selecting the Group Delay option and calculates the group delays in the ω_1 and ω_2 directions between the specified frequency limits.

DImpulseResponse function is initiated by selecting the Impulse Response option and calculates the impulse response for the number of samples specified by the plot grid size.

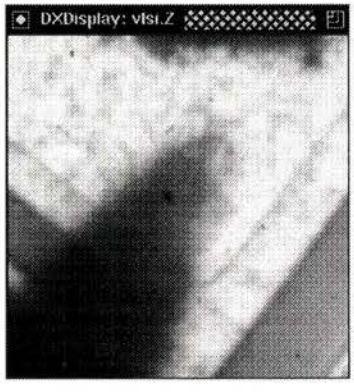
DStepResponse function is initiated by selecting the Step Response option and calculates the step response for the number of samples specified by the plot grid size.

The "**X**" to the right of each function selection button is a termination button. It is insensitive until the function is selected. Once selected the function selection button becomes insensitive and the "**X**" becomes active. The active "**X**" is used for early termination of the execution of the selected analysis. Multiple analyses can be run and controlled simultaneously.

The responses are plotted directly to the X-Windows display using DXPlot3D as shown in figure 2.5-b. DXPlot3D plots a DEDIP REAL_2D file format as a surface with hidden line removal based on the program plot3 [46]. DXPlot3D allows interactive viewpoint changes to be observed.

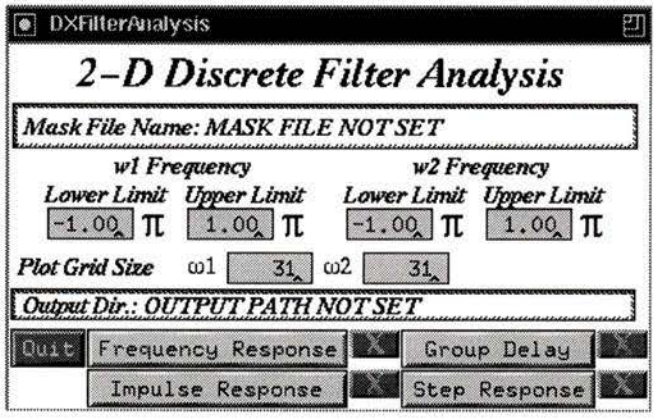


(a)

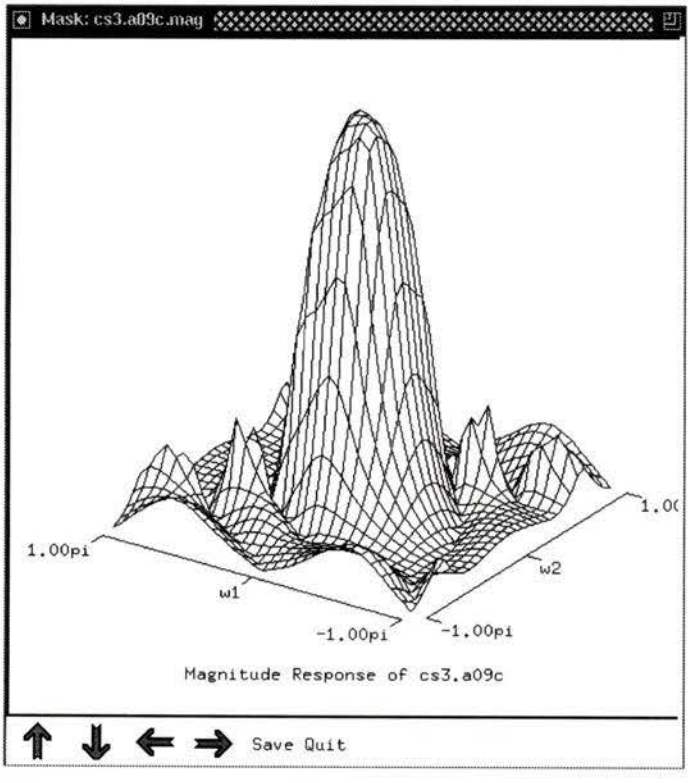


(b)

Figure 2.4 DXDisplay: partitioning displayed image.
(a) Interactive image partition window and popup feedback window.
(b) Partition result window.



(a)



(b)

Figure 2.5 DXFilterAnalysis: windows.
(a) Menu window. (b) 3-D plot window.

2.5 Summary

The Developmental Environment for Digital Image Processing (DEDIP) system is a flexible, extensible and easy to use programming environment for the development of DIP software. It has been designed using structured design principles with two levels of user interface. The low level is a C-language software library of digital image processing and system management functions. The high level is a set of digital image processing and file management tools operating under UNIX.

Modular design and data encapsulation methods are powerful structured design techniques. The application of these techniques is used to enhance the portability, reusability and reliability of the DEDIP system through representation-dependent system-management functions that hide the data and file structure representations. The hierarchical design has DIP functions built on these system-management functions for representation-independent algorithm implementation.

DIP algorithm development can proceed from either user interface level. At the high level, tools are used to operate on data files. Algorithm control is through intermediate files or the concatenation of multiple tools to form a desired algorithmic sequence. The low level uses the C programming language to implement a modular and hierarchical library composed of data and file structure dependent system management functions and data representation independent DIP operation functions. The syntax for the two levels are different. To ease transition of new algorithms from the high to the low levels, the tools and functions are mirrored. For an operation, the name and functionality are the same for both levels and the tool's command line flags are the same as the function's parameter list.

X-Windows applications have been developed for DEDIP at the high level. The X-Windows system is used to implement image display and surface plotting features. A tool for displaying images, DXDisplay, and a tool for analyzing 2-D filter masks, DXFilter-Analysis, are the currently available DEDIP X-Windows applications tools.

With the data representation independence and hierarchical library design the DEDIP system and the DEDIP X-Windows applications constitute an extensible software library and easy-to-use environment to develop tools, and to test and evaluate digital image-processing algorithms.

Chapter 3

2-D Digital Filters

3.1 Introduction

2-D digital filtering is used in many 2-D signal processing applications. 2-D discrete system theory including 2-D digital filters has evolved from 1-D discrete system theory [47]. Discrete systems have proved to be a very powerful and flexible tool for signal processing. This chapter presents an overview of 2-D discrete systems, 2-D digital filtering [40, 48-50], and their implementation in DEDIP.

In signal processing, a signal is generally considered to be composed of two components: information and noise. Filtering is applied to a signal to attenuate the noise component, thereby enhancing the information component. There are various kinds of filters that are generally grouped into categories of linear and non-linear filters. There are also various kinds of noise which degrade 2-D signals differently [51] and require different types of filters to improve the information component of the signal. The linear filter theory is better understood and several design methods for linear filters exist. The 2-D filtering options available in DEDIP are linear, shift invariant (LSI) filters which perform well for the attenuation of additive noise.

The theoretical background of 2-D LSI digital filters is briefly discussed in section 3.2. Fundamental properties of 2-D discrete systems are defined and used to present the 2-D convolution summation which is the basis for understanding LSI filters.

There are two types of 2-D LSI digital filters, nonrecursive and recursive. Special cases of nonrecursive and recursive 2-D digital filters that are separable can be used to realize a computationally more efficient implementation. Each of these four characterizations of 2-D digital filters are described in section 3.3. Each implementation's transfer

function and difference equation are defined and special features are discussed, followed by the description of the DEDIP 2-D digital filter interface and implementation.

In section 3.4, techniques for analyzing 2-D digital filters in the spatial and frequency domains are described. The DEDIP programs and functions that implement these filter analysis methods are described.

3.2 Background Theory

Digital image processing is performed by a 2-D discrete system where a 2-D discrete input signal, $x(n_1, n_2)$, is mapped to a 2-D discrete output signal, $y(n_1, n_2)$, by the system. Equation 3.1 shows this relationship in mathematical notation where the operator $T[\bullet]$ represents the mapping rules.

$$y(n_1, n_2) = T[x(n_1, n_2)] \tag{3.1}$$

This notation is applicable to linear and nonlinear systems. A system's properties categorizes it as linear or nonlinear. This section defines some important properties for 2-D signals and LSI systems.

3.2.1 2-D Discrete Signal Properties

A 2-D discrete signal, x, is a function of two independent integer variables, n_1 and n_2, as defined in equation 3.2

$$x = \{x(n_1, n_2) \,|\, -\infty < n_1 < \infty, -\infty < n_2 < \infty\} \tag{3.2}$$

where the notation $x(n_1,n_2)$ represents the value of x at the point (n_1,n_2). The independent integer variables, n_1 and n_2, are spatial indices to the function x.

For example, a seismic signal is composed of spatially discrete acoustic transponders that sample an acoustic signal. The n_1 index represents the transponder location and the n_2 index is the time domain sample number. The discrete signal is represented by $x(n_1,n_2T)$ where T is the sampling period for acoustic signal having the units of seconds.

3.2.1.1 Regions of Support

The finite-extent 2-D discrete signal is an important 2-D signal class. A finite-extent 2-D signal has a finite region outside of which all samples are zero. This is called the region of support. The basic definition of 2-D discrete signal in equation 3.2 extends to infinity. In practice, many 2-D discrete signals have a finite region of support. For example, an image

has a fixed number of pixels in the horizontal and vertical dimensions; the region not included in the defined image is outside the region of support and assumed to be zero.

A common practice is to describe the region of support based on the graph quadrant. First quadrant support is the most common found in signals and systems and is defined in equation 3.3. The next most common is the four quadrant support defined in equation 3.4. Half-plane support is also found with right half-plane support defined in equation 3.5.

$$x = \{x(n_1, n_2) \mid 0 \le n_1 < N_1;\ 0 \le n_2 < N_2\} \tag{3.3}$$

$$x = \{x(n_1, n_2) \mid -N_{1a} \le n_1 < N_{1b};\ -N_{2a} \le n_2 < N_{2b}\} \tag{3.4}$$

$$x = \{x(n_1, n_2) \mid 0 \le n_1 < N_1;\ -N_{2a} \le n_2 < N_{2b}\} \tag{3.5}$$

3.2.1.2 Elementary 2-D Discrete Signals

The 2-D discrete impulse signal is defined in equation 3.6. This elementary signal is quite important for derivation of the convolution summation which is the basis of linear system design. The impulse response is also used in spatial-domain analysis which will be discussed later.

$$\delta(n_1, n_2) = \begin{cases} 1 & \text{for } n_1 = n_2 = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

The unit step, defined in equation 3.7, has first quadrant support and is also used in spatial-domain analysis.

$$u(n_1, n_2) = \begin{cases} 1 & \text{for } n_1 \ge 0 \text{ and } n_2 \ge 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

An interesting property of both elementary signals is that they are separable as shown in equations 3.8 and 3.9.

$$\delta(n_1, n_2) = \delta_1(n_1)\,\delta_1(n_2) \tag{3.8}$$

where

$$\delta_1(n) = \begin{cases} 1 & \text{for } n = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$u(n_1, n_2) = u_1(n_1)\, u_1(n_2) \tag{3.9}$$

where

$$u_1(n) = \begin{cases} 1 & \text{for } n \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

3.2.2 2-D Discrete Systems

In studying complex systems, it is often necessary to restrict them to a special class having certain fundamental properties that allow the complexity to be reduced. The properties of linearity, shift invariance, causality and stability are fundamental properties that are used to limit the class of systems under consideration. A system with these properties is mathematically tractable while still exhibiting some interesting behavior. These properties are defined and their use in the structuring of systems is discussed through the definition of the convolution summation.

3.2.2.1 Linearity

A 2-D discrete system is linear if it satisfies the two conditions of homogeneity and additivity. A discrete system, $L[\bullet]$, is linear if for given two sequences, $x_1(n_1, n_2)$ and $x_2(n_1, n_2)$, and if

$$y_1(n_1, n_2) = L[x_1(n_1, n_2)]; \qquad y_2(n_1, n_2) = L[x_2(n_1, n_2)]$$

then

$$ay_1(n_1, n_2) + by_2(n_1, n_2) = L[ax_1(n_1, n_2) + bx_2(n_1, n_2)] \tag{3.10}$$

for all possible values of a and b, otherwise it is nonlinear.

3.2.2.2 Shift Invariance

A 2-D discrete system is shift invariant if for a given shift in the input sequence, $x(n_1, n_2)$, there is a corresponding shift in the output sequence, $y(n_1, n_2)$. Given a system

$$y(n_1, n_2) = T[x(n_1, n_2)]$$

then $T[\bullet]$ is shift invariant if and only if

$$y(n_1 - k, n_2 - l) = T[x(n_1 - k, n_2 - l)]. \tag{3.11}$$

A discrete system is shift dependent if the condition of equation 3.11 is not satisfied.

3.2.2.3 Causality

Causality evolved from 1-D time domain systems emulating real, physical systems where the output cannot be dependent on a future input. Causality of a 2-D discrete system exists if the output, $y(n_1, n_2)$, with $n_1 \leq k_1$ and $n_2 \leq k_2$ is independent of the input, $x(n_1, n_2)$, for values $n_1 > k_1$ and $n_2 > k_2$ for all possible values of k_1 and k_2. In 2-D discrete systems, the causality constraint is often less of a concern with the independent variables usually being spatial and all the input samples being available beforehand.

3.2.2.4 2-D Convolution Summation

A 2-D discrete signal can be decomposed into a sum of weighted and shifted 2-D impulses as shown in equation 3.12. Using the right hand side of equation 3.12 as the input signal to a linear system, $L[\bullet]$, the output, $y(n_1,n_2)$, is shown in equation 3.13. Because a linear system satisfies the additivity condition, the response to an input signal composed of a weighted sum of equation 3.12 is equal to the sum of the response to the individual weighted input signals. The result of applying the additivity condition to equation 3.13 is shown in equation 3.14.

$$x(n_1, n_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2)\, \delta(n_1 - k_1, n_2 - k_2) \tag{3.12}$$

$$y(n_1, n_2) = L\left[\sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2)\, \delta(n_1 - k_1, n_2 - k_2) \right] \tag{3.13}$$

$$\begin{aligned} y(n_1, n_2) &= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2)\, L[\delta(n_1 - k_1, n_2 - k_2)] \\ &= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x(k_1, k_2)\, h_{k_1 k_2}(n_1, n_2) \end{aligned} \tag{3.14}$$

where equation 3.15

$$h_{k_1 k_2}(n_1, n_2) \equiv L[\delta(n_1 - k_1, n_2 - k_2)] \tag{3.15}$$

defines the spatially varying impulse response of a 2-D discrete linear system.

Further restricting a 2-D discrete linear system to being shift invariant, the spatially-varying impulse response becomes a shifted version of the spatially-invariant impulse

response, $h(n_1,n_2)$, shown in equation 3.16

$$h_{k_1 k_2}(n_1, n_2) = h(n_1 - k_1, n_2 - k_2) \tag{3.16}$$

where

$$h(n_1, n_2) \equiv h_{00}(n_1, n_2)$$

is the spatially-invariant impulse response. Substituting equation 3.16 into equation 3.14 yields the convolution summation of equation 3.17.

$$y(n_1, n_2) = \sum_{k_1 = -\infty}^{\infty} \sum_{k_2 = -\infty}^{\infty} x(k_1, k_2)\, h(n_1 - k_1, n_2 - k_2) \tag{3.17}$$

With the variable substitution of $n_1 - k_1 = l_1$ and $n_2 - k_2 = l_2$, the convolution summation can be rearranged to equation 3.18.

$$y(n_1, n_2) = \sum_{l_1 = -\infty}^{\infty} \sum_{l_2 = -\infty}^{\infty} h(l_1, l_2)\, x(n_1 - l_1, n_2 - l_2) \tag{3.18}$$

3.2.2.5 Stability

Bounded-input, bounded-output (BIBO) stability is one type of stability criterion commonly used to characterize the stability of a system. A system, $T[\bullet]$, is BIBO stable if and only if for every bounded sequence $x(n_1, n_2)$ such that $|x(n_1, n_2)| \leq B$ the response of the system is also bounded.

The necessary and sufficient condition for BIBO stability of a 2-D LSI discrete system is that the impulse response is absolutely summable as in equation 3.19.

$$\sum_{k_1 = -\infty}^{\infty} \sum_{k_2 = -\infty}^{\infty} \left| h(k_1, k_2) \right| = S < \infty \tag{3.19}$$

3.2.2.6 Frequency-Domain Characterization

The 2-D discrete transfer function, $H(z_1, z_2)$, is defined by the ratio of the z-transform of the output, $Y(z_1, z_2)$, to the z-transform of the input, $X(z_1, z_2)$, as shown in equation 3.20.

$$H(z_1, z_2) = \frac{Y(z_1, z_2)}{X(z_1, z_2)} = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} h(k_1, k_2) z_1^{-k_1} z_2^{-k_2} \tag{3.20}$$

Therefore, $H(z_1, z_2)$ is the z-transform of the spatially-invariant impulse response.

3.2.2.7 2-D Separable Discrete Systems

The special case of a 2-D discrete system that can be factored into its two 1-D components, as shown in equation 3.21,

$$H(z_1, z_2) = H_1(z_1) H_2(z_2) \tag{3.21}$$

is a separable system. $H(z_1,z_2)$ is the 2-D transfer function and $H_1(z_1)$ and $H_2(z_2)$ are 1-D transfer functions in the z_1 and z_2 directions, respectively. A separable digital filter can be realized with a substantial reduction in computational requirements. The separability property will be discussed further in "Separable 2-D FIR Digital Filters" on page 39 and "Separable 2-D IIR Digital Filters" on page 40.

3.3 2-D LSI Digital Filter Characterization

The two types of 2-D LSI digital filters are nonrecursive and recursive. They can be realized based on the finite-order difference equation for a 2-D discrete LSI system shown in equation 3.22.

$$\sum_{l_1=0}^{L_1} \sum_{l_2=0}^{L_2} b(l_1, l_2) y(n_1 - l_1, n_2 - l_2) = \sum_{k_1=0}^{K_1} \sum_{k_2=0}^{K_2} a(k_1, k_2) x(n_1 - k_1, n_2 - k_2) \tag{3.22}$$

where the $a(k_1, k_2)$ are the coefficients applied to the input data and the $b(l_1, l_2)$ are the coefficients applied to the output data. Because the limits of the summation are finite, the output, $y(n_1, n_2)$, can be computed from the input and output sequences with a finite amount of computation. The extents of both coefficients masks, $a(k_1, k_2)$ and $b(l_1, l_2)$, are finite. The order of a difference equation is described by the region of support for the output coefficients, $b(l_1, l_2)$. The region of support can have a wide variety of shapes. Using the constraint of first quadrant support for $a(k_1, k_2)$ and $b(l_1, l_2)$, the shape can often be assumed to be rectangular with an order of $L_1 \times L_2$ [48].

The LSI nonrecursive and recursive digital filters are also known as finite-duration impulse-response (FIR) and infinite-duration impulse-response (IIR) filters, respectively.

3.3.1 2-D FIR Digital Filters

The 2-D, LSI nonrecursive digital filter response at $y(n_1, n_2)$ can be expressed as a weighted sum of the shifted inputs $x(n_1\text{-}k_1, n_2\text{-}k_2)$, for $0 \le k_1 \le K_1$ and $0 \le k_2 \le K_2$. Because the extent of the impulse response is finite, this filter is also known as a finite-duration impulse-response (FIR) filter. Equation 3.23 shows FIR difference equation.

$$y(n_1, n_2) = \sum_{k_1=0}^{K_1} \sum_{k_2=0}^{K_2} h(k_1, k_2)\, x(n_1 - k_1, n_2 - k_2) \tag{3.23}$$

where the $h(k_1, k_2)$ are the filter coefficients, and, assuming first quadrant support, $\{h(k_1, k_2) | \ 0 \le k_1 \le K_1, 0 \le k_2 \le K_2\}$. $x(n_1, n_2)$ is the input signal and $y(n_1, n_2)$ is the output signal. The order of a first-quadrant FIR filter is defined to be the rectangular region of support, $K_1 \times K_2$. The FIR filter is a 0×0 order finite-order difference equation. The transfer function for an FIR filter is shown equation 3.24.

$$H(z_1, z_2) = \sum_{k_1=0}^{K_1} \sum_{k_2=0}^{K_2} h(k_1, k_2)\, z_1^{-k_1} z_2^{-k_2} \tag{3.24}$$

The FIR filter has some very useful properties. It is always stable because for finite $h(k_1, k_2)$ the impulse response is absolutely summable. This meets the necessary and sufficient condition for BIBO stability for an LSI discrete system.

The second property is that linear phase-response FIR filters are easily designed. Linear phase-response filters maintain a constant group delay.

A third property is that FIR filters can be implemented using Fast Fourier Transform (FFT) techniques. For low order filters, these are not as computationally efficient as a convolution summation but for higher order filters, improved computational efficiency that is effectively independent with respect to the filter order can be realized with FFT techniques. A Problem with the FFT implementation is the large on-line storage requirement. Block FFT methods of overlap-add and overlap-save can be used to reduce this requirement [48, 49].

3.3.1.1 FIR Implementation Considerations

The initial or boundary conditions must be considered at the boundaries of an image. Generally, they are assumed to be zero since we have a finite extent for the image. This need not be the situation in all cases. But in any case, the LSI properties must be maintained. The linearity property may be violated if the response to an all zero input is not an all zero output.

The region of support for a filter also determines which boundary conditions are required. For a four quadrant 2-D digital filter, boundary conditions are required on all four boundaries of the images region of support. For a first quadrant 2-D digital filter, boundary conditions are only required on two boundaries of the image.

The computational requirements for the FIR filter is $(K_1+1)(K_2+1)$ multiplications and $(K_1+1)(K_2+1)$ -1 additions for each pixel. The result, $y(n_1, n_2)$, has the dimensions of $(N_1 + K_1 - 1)\mathbf{x}(N_2 + K_2 - 1)$.

3.3.2 2-D IIR Digital Filters

The 2-D LSI recursive digital filter response, $y(n_1, n_2)$, can be expressed as a weighted sum of shifted inputs, $x(n_1-k_1, n_2-k_2)$, for $0 \le k_1 \le K_1$ and $0 \le k_2 \le K_2$, and the outputs, $y(n_1-l_1, n_2-l_2)$, for $0 \le l_1 \le L_1$, $0 \le l_2 \le L_2$ and $l_1=l_2\neq 0$. The finite-order difference equation of equation 3.22 is rewritten in equation 3.25

$$y(n_1, n_2) = \frac{1}{b(0,0)} \sum_{k_1=0}^{K_1} \sum_{k_2=0}^{K_2} a(k_1, k_2)\, x(n_1 - k_1, n_2 - k_2)$$

$$- \frac{1}{b(0,0)} \sum_{\substack{l_1=0 \\ (l_1 = l_2) \neq 0}}^{L_1} \sum_{l_2=0}^{L_2} b(l_1, l_2)\, y(n_1 - l_1, n_2 - l_2) \qquad (3.25)$$

where the $a(k_1, k_2)$ are the input filter coefficients, the $b(l_1, l_2)$ are the output filter coefficients, $x(n_1, n_2)$ is the input signal and $y(n_1, n_2)$ is the response. $L_1 \times L_2$ is the order of the filter. When the filter is represented in the form of the convolution summation, the impulse response has infinite extent. This filter is also known as an Infinite-duration Impulse-Response (IIR) filter.

The transfer function for an IIR filter is a rational function as shown in equation 3.26.

$$H(z_1, z_2) = \frac{\sum_{k_1=0}^{K_{A1}} \sum_{k_2=0}^{K_{A2}} a(k_1, k_2) z_1^{-k_1} z_2^{-k_2}}{\sum_{l_1=0}^{L_{B1}} \sum_{l_2=0}^{L_{B2}} b(l_1, l_2) z_1^{-l_1} z_2^{-l_2}} \tag{3.26}$$

The main feature of the 2-D IIR digital filter is its computational efficiency compared to that of the 2-D FIR digital filter for filters satisfying the same frequency domain specifications with high selectivity. The selection of filter mask coefficients can be such that the order of an IIR filter is less than a FIR filter with equivalent magnitude response thereby reducing the computational requirements.

3.3.2.1 IIR Implementation Considerations

The IIR filter uses previous output samples to calculate the current output sample. This feedback connection can lead to an unstable system. An IIR filter must be checked for stability before it can be used.

The design of an IIR filter is more difficult than that of a FIR filter. The use of a special transformation on 1-D IIR filters is one method used to design 2-D IIR filters. A second method uses optimization to approximate a desired magnitude response. Designing zero or linear phase IIR filters is more difficult [40].

Another problem with an IIR digital filter is recursive computability. The shape of the region of support for output coefficients and the order that the calculations are performed may not be realizable. The work presented here is based on first-quadrant support IIR filters which are quarter plane support filters and are recursively computable.

The computational requirements are $(K_{A1}+1)(K_{A2}+1) + (L_{B1}+1)(L_{B2}+1)$-1 multiplications and $(K_{A1}+1)(K_{A2}+1) + (L_{B1}+1)(L_{B2}+1)$-2 additions for each pixel. $(N_1 + K_{1max} - 1)\mathbf{x}(N_2+K_{2max} - 1)$ are the dimensions of the IIR filter result where K_{1max} = maximum (K_{A1}, L_{B1}) and K_{2max} = maximum (K_{A2}, L_{B2}).

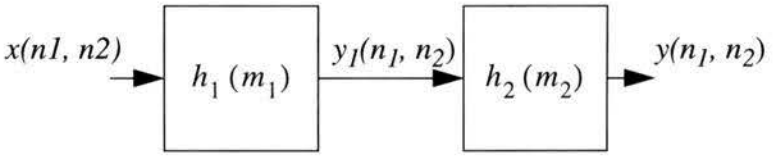


Figure 3.1 SFIR block diagram: separable 1-D FIR filters.

3.3.3 Separable 2-D FIR Digital Filters

Separable FIR (SFIR) filters are special cases of the FIR filter that can often realize a more efficient implementation. Equation 3.27 separates the 2-D FIR filter into two 1-D FIR filters. Figure 3.1 shows the flow graph for its realization.

$$H(z_1, z_2) = H_1(z_1) H_2(z_2) \rightarrow h(k_1, k_2) = h_1(k_1) \cdot h_2(k_2) \tag{3.27}$$

Equation 3.28 and 3.29 show the convolution summation for the second and first blocks in figure 3.1, respectively. Equation 3.29 performs the FIR filtering in the k_1 direction while equation 3.28 performs the filtering in the k_2 direction. The transfer function for the separable FIR filter is shown in equation 3.30.

$$\begin{aligned} y(n_1, n_2) &= \sum_{k_2=0}^{K_2} h_2(k_2) \sum_{k_1=0}^{K_1} h_1(k_1)\, x(n_1 - k_1, n_2 - k_2) \\ &= \sum_{k_2=0}^{K_2} h_2(k_2)\, y_1(n_1, n_2 - k_2) \end{aligned} \tag{3.28}$$

where

$$y_1(n_1, n_2) = \sum_{k_1=0}^{K_1} h_1(k_1)\, x(n_1 - k_1, n_2) \tag{3.29}$$

$$H(z_1, z_2) = \left[\sum_{k_1=0}^{K_1} h_1(k_1)\, z_1^{-k_1} \right] \left[\sum_{k_2=0}^{K_2} h_2(k_2)\, z_2^{-k_2} \right] \tag{3.30}$$

3.3.3.1 SFIR Implementation Considerations

Similarly to the FIR filter, the SFIR filter is also always stable. The initial conditions can be applied to the input or separated over each stage as required by the filter mask orientation.

The computational requirements for the SFIR filter are $(K_1+1)+(K_2+1)$ multiplications and K_1+K_2 additions for each pixel. The SFIR filter result has the dimensions of $(N_1+K_1-1)\mathbf{x}(N_2+K_2-1)$. The required storage for $y_1(n_1,n_2)$ is $(N_1+K_1-1)\ \mathbf{x}\ (K_2-1)+1$ samples.

3.3.4 Separable 2-D IIR Digital Filters

As with the 2-D FIR digital filter, the 2-D IIR digital filter also has a special case that is separable and can be realized more efficiently. Equation 3.31 separates the recursive filter into two cascaded 1-D recursive filters. Figure 3.2 shows the flow graph for the implementation.

$$H(z_1, z_2) = H_1(z_1)\,H_2(z_2) \rightarrow \left[\frac{a_1(k_1)}{b_1(l_1)}\right]\left[\frac{a_2(k_2)}{b_2(l_2)}\right] \tag{3.31}$$

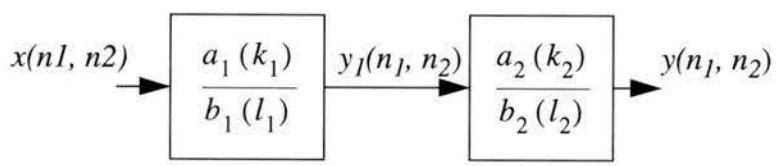


Figure 3.2 SIIR block diagram: separable recursive 1-D filters.

Equation 3.32 and 3.33 show the difference equations for the first and second blocks in figure 3.2, respectively. The difference equation in equation 3.32 performs a recursive filtering operation in the k_1 direction while equation 3.33 performs the filtering in the k_2 direction. The transfer function for the SIIR filter is shown in equation 3.34.

$$y_1(n_1, n_2) = \frac{1}{b_1(0)} \sum_{k_1=0}^{K_{A1}} a_1(k_1)\, x(n_1 - k_1, n_2)$$

$$-\frac{1}{b_1(0)} \sum_{l_1=1}^{L_{B1}} b_1(l_1)\, y_1(n_1 - l_1, n_2) \tag{3.32}$$

$$y(n_1, n_2) = \frac{1}{b_2(0)} \sum_{k_2=0}^{K_{A2}} a_2(k_2)\, y_1(n_1, n_2 - k_2)$$

$$-\frac{1}{b_2(0)} \sum_{l_2=1}^{L_{B2}} b_2(l_2)\, y(n_1, n_2 - l_2) \tag{3.33}$$

$$H(z_1, z_2) = \left[\frac{\sum_{k_1=0}^{K_{A1}} a_1(k_1) z_1^{-k_1}}{\sum_{l_1=0}^{L_{B1}} b_1(l_1) z_1^{-l_1}} \right] \left[\frac{\sum_{k_2=0}^{K_{A2}} a_2(k_2) z_2^{-k_2}}{\sum_{l_2=0}^{L_{B2}} b_2(l_2) z_2^{-l_2}} \right] \tag{3.34}$$

3.3.4.1 SIIR Implementation Considerations

As with the IIR filter the SIIR filter has potential stability problems which must be checked before implementation. The initial conditions can also be placed at the input signal or separated over each stage. However, it is much easier to check the stability of the two 1-D filters, $H_1(z_1)$ and $H_2(z_2)$, than it is to check general 2-D stability [49].

The computational requirements for the SFIR filter are $K_{A1}+K_{A2}+L_{B1}+L_{B2}+4$ multiplications and $K_{A1}+K_{A2}+L_{B1}+L_{B2}+2$ additions for each pixel. The SFIR filter result has the dimensions of $(N_1+K_{1max}-1)$**x**$(N_2+K_{2max}-1)$ where K_{1max} = maximum (K_{A1}, L_{B1}) and K_{2max} = maximum (K_{A2}, L_{B2}). The storage required for $y_1(n_1,n_2)$ is $(N_1+K_{1max}-1)$ **x** $(K_{2max}-1)+1$ samples.

3.3.5 DEDIP 2-D Filter Implementation and Interface

In many applications, 2-D filters are designed based on frequency-domain specifications. To evaluate the performance of the filter designed it is desirable to observe the output of the filter using test images which can be real images or special artificial images as input. DEDIP gives the possibility of carrying out such a performance evaluation very easily and efficiently. The filter implementations can be tested using the DEDIP tools DFilterImage, DFilterReal2d, DWeightedAverageImage and DWeightedAverageReal2d. DFilterImage and DFilterReal2d perform first quadrant filtering on IMAGE and REAL_2D file

type, respectively. The filter masks can be FIR, IIR, SFIR or SIIR types. DWeightedAverageImage and DWeightedAverageReal2d perform weighted neighborhood averaging on IMAGE and REAL_2D file types, respectively. An overview of the filtering capabilities are discussed in chapter 2.

DFilterImage and DFilterReal2d accept FIR, IIR, SFIR and SIIR filter mask types. Initial conditions are applied to the top and left edges of the image before filtering. The result of the filtering is a REAL_2D array file which is truncated so that the output dimension is the same as for the input array. The UNIX manual pages for DEDIP filtering operations, DedipFilters(1) for programs and DedipFilters(3) for library functions, describe the implementations in more detail [29]. The capabilities for testing filter implementations using these programs will be demonstrated in the next chapter.

3.4 2-D LSI Filter Analysis

A 2-D LSI digital filter is given as a set of coefficients for the finite-order difference equation. The analysis of a filter coefficient mask reveals spatial- and frequency-domain characteristics. The following subsections describe briefly the main filter analysis methods implemented in DEDIP.

3.4.1 Spatial-Domain Analysis

The main spatial-domain analysis operations are the impulse and the step response which represent the response of the filter to the elementary signals defined in equations 3.6 and 3.7, respectively.

3.4.1.1 Impulse Response

The response of a LSI system to the impulse signal is its shift-invariant impulse response, $h(k_1, k_2)$. The impulse response of a FIR filter is the filter coefficients themselves. This can be seen by comparing the definition of the convolution summation to the FIR difference equation. By applying an impulse to an IIR filter the spatially-invariant impulse response can be obtained. From its definition the impulse response of an IIR filter has infinite extent. For an IIR filter to be BIBO stable its impulse response must be absolutely summable.

3.4.1.2 Step Response

In 1-D systems, the step response is used to determine the transient response time of a system which is the period the filter takes to stabilize to a steady state. The 2-D step response provides similar information for 2-D systems.

3.4.2 Frequency-Domain Analysis

A linear filter is concerned with the manipulation of the frequency spectrum of a signal. The convolution summation is the basis for LSI digital systems and its transfer function defines the mapping from the input to the output in the frequency domain. Frequency-domain analysis analyzes the frequency-domain properties of a LSI filter mask.

3.4.2.1 Frequency Response

The frequency response of a filter is its response to a complex sinusoidal stimulus. The complex response in equation 3.35 is analyzed in terms of its magnitude and phase components of equations 3.36 and 3.37, respectively.

$$H(\omega_1, \omega_2) = H(z_1, z_2)\Big|_{z_1 = e^{j\omega_1}, z_2 = e^{j\omega_2}}$$

$$= M(\omega_1, \omega_2)\, e^{j\theta(\omega_1, \omega_2)} \tag{3.35}$$

$$M(\omega_1, \omega_2) = \left|H(\omega_1, \omega_2)\right| \tag{3.36}$$

$$\theta(\omega_1, \omega_2) = \arg H(\omega_1, \omega_2) \tag{3.37}$$

The frequency response is an indication of how the frequency components of the input image will be modified by the filter.

3.4.2.2 Group Delay

The group delays, τ_{g1} and τ_{g2}, defined in equation 3.38 are the partial derivatives of the phase component in the ω_1 and ω_2 directions, respectively.

$$\tau_{gi}(\omega_1, \omega_2) = -\frac{\partial}{\partial \omega_i}(arg\,[H(\omega_1, \omega_2)]) \tag{3.38}$$

A constant group delay indicates a linear phase filter response. Research [48, 49] has indicated that the phase component carries significant information about a 2-D signal

and changes in the phase lead to visible distortion. A linear phase filter is designed to minimize the distortion due to difference in the group delays at various frequencies in the passband.

3.4.3 DEDIP 2-D Filter Analysis Interface

The 2-D filter analysis discussed in this section can be carried out using DImpulseResponse, DStepResponse, DFrequencyResponse and DGroupDelay which are DEDIP programs and library functions whose name indicates the analysis operation. See chapter 2 for an overview of filter analysis functions. The analysis programs take a 2-D FIR or IIR filter mask as the input. The DImpulseResponse and DStepResponse operations each produces output as a REAL_2D array whose dimensions are specified by parameter options. The output from DFrequencyResponse and DGroupDelay operations is a COMPLEX_2D array whose dimensions are specified by parameter options as well the frequency limits of the analysis can also be specified by options. The UNIX manual pages for DEDIP filter analysis operations, DedipFilterAnalysis(1) for programs and DedipFilterAnalysis(3) for library functions, describe the implementations in more detail [29]. The filter analysis capabilities of DEDIP will be demonstrated in the next chapter.

Chapter 4

Filter Examples: Application and Analysis

4.1 Introduction

An important aspect of DEDIP is the analysis of LSI 2-D digital filter coefficient masks and their application to images. The background theory for LSI 2-D digital filters is discussed in chapter 3. In this chapter, four examples will demonstrate the analysis and application of LSI 2-D digital filter coefficient masks to images.

The first example demonstrates FIR and IIR implementations of filters with low-pass magnitude responses. The FIR and IIR implementations are analyzed and then applied to a test image. The two implementations' performance is discussed briefly.

The second example is a fan filter which has the property of highlighting directionally-oriented edges. It is designed from frequency-domain specifications that pass selected portions of the input image spectrum which correspond to intensity transitions with a particular spatial orientation. Images with rectangular features that have simple spectra are used to demonstrate the directional selectivity. The fan filter coefficient mask, implemented as an FIR filter, is analyzed and applied to test images.

The last two examples demonstrate directional first-derivative and rotationally-invariant second-derivative edge-detection operators. Edge detection is an important step in obtaining features for higher-level image processing such as image analysis and object recognition. In these examples it is shown how this can be done using DEDIP.

The third example discusses directional first-derivative operators and their evolution through the simple difference, Prewitt, Sobel and Rosenfeld-Thurston operators. The X-

and *Y*-oriented directional-derivative operators of these four operators are analyzed and applied in subsequent sections. The resulting directional-derivative images are combined to create gradient images. The gradient images are thresholded at a value that defines a certain gradient steepness as an edge. The resulting binary images typically have edges that are multiple pixels wide. During the thresholding stage, the edges are also thinned to reduce the edge to a single pixel width for use in feature extraction algorithms.

The Laplacian of a Gaussian (LoG), a rotationally-invariant second-derivative operator, is described in the fourth example [52, 53]. Through a special signal flow decomposition of the LoG filter, a modified separable filter is developed. The separable coefficients reduce the number of additions and multiplications required to perform the same filtering operation. The performance of the LoG filter for different values of the Gaussian parameter, σ, which leads to edge detection at various spatial resolutions is considered. As with the directional-derivative edge operators, non-linear post processing is required to produce an edge map with single pixel wide edges. The edge thinning is accomplished by using the fact that the zero-crossings of the second derivative coincide with the midpoints of edges.

4.1.1 Test Images and Analysis Tools

Some mathematical notation and tools used in the analysis and presentation of the filtering examples are described.

4.1.1.1 Coordinate System

Computer graphical displays often use an inverted Y axis where the upper left of the display is the (0,0) position. The X axis is increasing from left to right and the Y axis increases from top to bottom. The images presented here also use this convention.

In the 2-D filter theory discussion of chapter 3, the image region of support is given by the general notation $\{I(n_1,n_2) \mid 0 \le n_1 \le N_1, 0 \le n_2 \le N_2\}$. This chapter presents examples using spatial notation. An image is defined as $\{I(x,y) \mid 0 \le x \le N_1, 0 \le y \le N_2\}$. A FIR filter mask is defined by $\{h(k_1,k_2) \mid 0 \le k_1 \le K_1, 0 \le k_2 \le K_2\}$. An IIR filter input coefficient mask is defined by $\{a(k_1,k_2) \mid 0 \le k_1 \le K_{A1}, 0 \le k_2 \le K_{A2}\}$ and its output coefficient mask is defined by $\{b(l_1,l_2) \mid 0 \le l_1 \le K_{B1}, 0 \le l_2 \le K_{B2}\}$.

4.1.1.2 Fast Fourier Transform Display

The Fast Fourier Transform (FFT) is used to calculate the discrete Fourier transform, $F(\kappa_1,\kappa_2)$, of images to obtain information about the distribution of complex frequency components. The magnitude of the FFT as defined in equation 4.1 is also called the Fourier spectrum where $Re(\kappa_1,\kappa_2)$ is the real part of $F(\kappa_1,\kappa_2)$ and $Im(\kappa_1,\kappa_2)$ is the imaginary part.

$$\left|F(\kappa_1, \kappa_2)\right| = \sqrt{(Re(\kappa_1, \kappa_2))^2 + (Im(\kappa_1, \kappa_2))^2} \tag{4.1}$$

The coordinate system of the FFT is $\{F(\kappa_1,\kappa_2) \mid 0 \le \kappa_1 \le N_1\text{-}1,\ 0 \le \kappa_2 \le N_2\text{-}1\}$. The spectral distribution of interest is at the origin. Because the FFT is periodic, dividing the FFT image into four quadrants and shifting the quadrants results in a FFT with support $\{F(\kappa_1,\kappa_2) \mid -N_1/2 \le \kappa_1 \le N_1/2,\ -N_2/2 \le \kappa_2 \le N_2/2\}$ [50]. The Fourier spectrum images shown here have been shifted and the origin of the κ_1–κ_2 plane is at the center of the image. The horizontal axis is the κ_1 axis which increases from left to right and the κ_2 axis is the vertical axis which increases from top to bottom.

In most cases, image spectra decrease quite quickly as the frequency increases. As a result of this high-frequency components are not clearly visible when spectra are displayed as images. A technique to improve the display of spectral images is to take the logarithm of the spectrum plus one as shown in equation 4.2 [50]

$$LF(\kappa_1, \kappa_2) = \log(k\left|F(\kappa_1, \kappa_2)\right| + 1) \tag{4.2}$$

where k is a scale factor used to control the contrast in the image. Zero values are preserved as $log(1)=0$. The term "log spectrum" will be used to describe this image.

4.1.1.3 Histogram Equalization

A technique to improve the display quality of a spatial-domain image is histogram equalization [50]. The histogram equalization redistributes the pixel intensity based on the probability density function also known as a histogram. Where as the log spectrum method applies the logarithm function to each value independently of other values, histogram equalization modifies a value based on the distribution of all other values. This introduces distortion in the separation of the intensity levels but enhances subjective image quality. Histogram equalized images can improve the viewing of qualitative information within an image. Because it is based on the distribution of pixel intensities within an image, histogram equalization is not accurate for comparison between images.

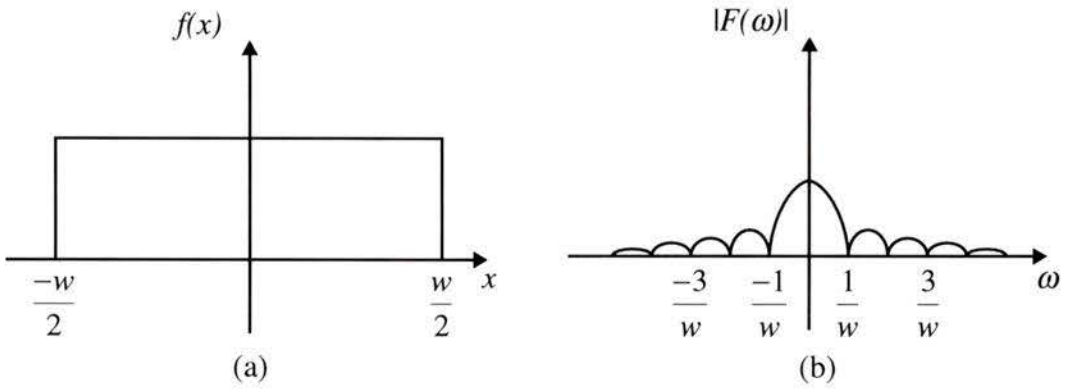


Figure 4.1 Pulse function.
(a) Pulse function in spatial domain. (b) Fourier spectrum of the pulse function.

4.1.1.4 2-D Rectangular Feature Analysis

In this section, the images to be used for the filter examples will be analyzed in the frequency domain. The continuous 1-D Fourier transform can be used as a basis for understanding the spectrum of a 2-D discrete image with a rectangular feature. Figure 4.1-a shows a pulse function whose form is similar to the cross-section of a rectangular feature along a major axis through the origin. Figure 4.1-b shows its Fourier spectrum [54]. The width of the pulse is inversely proportional to the spacing of the zero values in the spectrum.

Figure 4.2-a shows a horizontal rectangular feature, figure 4.2-b is the spectrum for the horizontal feature and figure 4.2-c is its log spectrum. Comparison of the horizontal cross-section of the feature in figure 4.2-a to its vertical cross-section shows that it is the wider pulse. From the inverse proportionality of the pulse function's spectrum, the spectrum of the horizontal cross-section should have closer zero separations compared to the vertical cross-section. Observations of figure 4.2-c support this hypothesis.

Figures 4.2-d, 4.2-e and 4.2-f show the rectangular feature rotated through 45^{o}, its spectrum and log spectrum, respectively. The corresponding 45^{o} rotation of the spectrum can clearly be seen in figure 4.2-f. Figures 4.2-g, 4.2-h and 4.2-i show a vertical rectangular feature, its spectrum and the log spectrum, respectively. Rotation in the spatial domain is shown to have a corresponding rotation in the frequency domain.

The benefit of using log spectrum can be clearly seen in comparing figures 4.2-b, 4.2-e and 4.2-h to figures 4.2-c, 4.2-f and 4.2-i, respectively. The first group shows a small amount of information concerning the higher-frequency spectra. The log spectrum

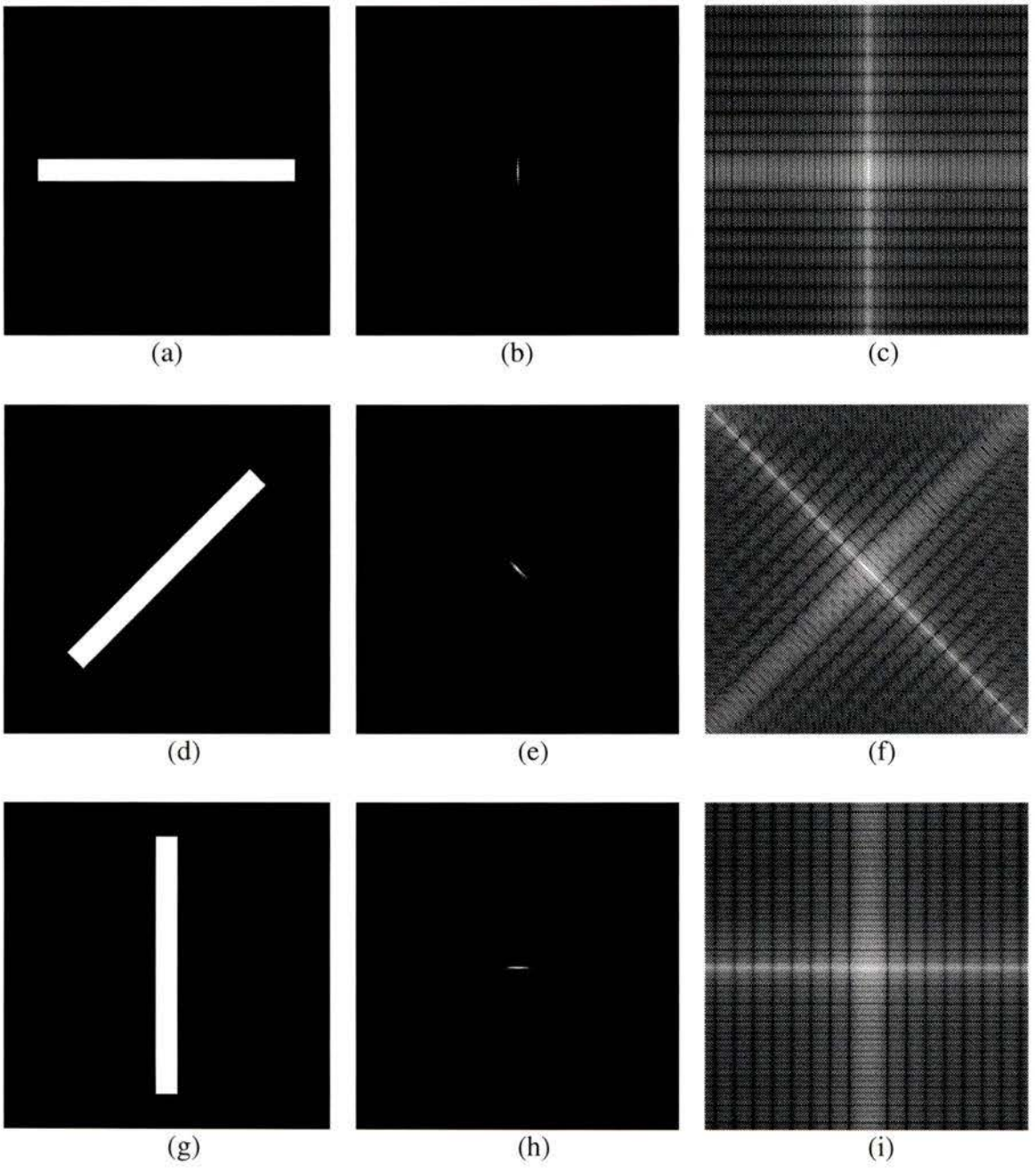


Figure 4.2 Relationship of rectangular image features with their spectra.
(a) Horizontal rectangular feature.(b) Spectrum of (a). (c) Log spectrum of (a).
(d) 45^{o} rectangular feature. (e) Spectrum of (d). (f) Log spectrum of (d).
(g) Vertical rectangular feature. (h) Spectrum of (g). (i) Log spectrum of (g).

images in figures 4.2-c, 4.2-f and 4.2-i convey more information about the overall spectral distribution.

From these observations, the spatial orientations of the features can be used to describe the frequency orientation of edge components. The direction of an edge gradient is oriented in the direction of the intensity change that defines the edge. The gradient direction in the spatial domain has the same orientation as in the frequency domain. The gradient of the long edge of the horizontal rectangular feature has a vertical orientation. The corresponding log spectrum image has a larger spacing between the zero (black) frequency components on the line κ_2=0 (vertical axis) than the line κ_1=0 (horizontal axis). That corresponds to the hypothesized orientation based on the Fourier transform model. A similar hypothesis can also be confirmed with the 45^o and vertical features.

4.1.1.5 Abingdon Cross

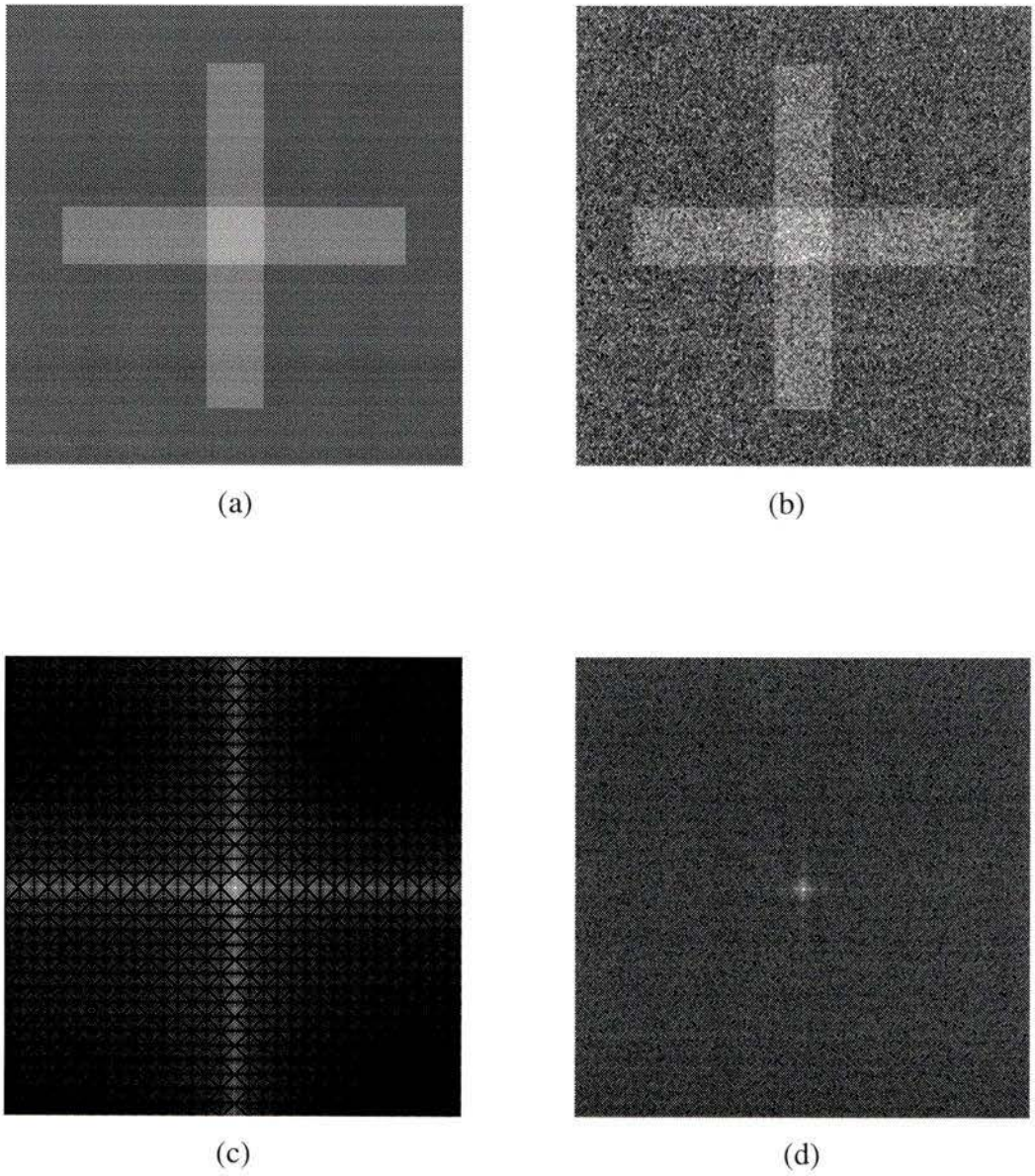
The Abingdon cross used for the lowpass filter test image is similar to that described in [55, 56] but scaled in half to a 256x256x8 image. The background intensity level is 128 and two orthogonal rectangles of medium gray have an intensity level of 160 superimposed on the background. The square in the middle has an intensity level of 192. The cross has Gaussian noise with zero mean and standard deviation of 32 added to each pixel. Figure 4.3-a shows the Abingdon cross without noise and its log spectrum in figure 4.3-c. Most of the spectrum's energy is centered about the origin of the κ_1–κ_2 plane. The Abingdon cross is shown in figure 4.3-b and its log spectrum in figure 4.3-d. Again the energy patterns at the origin are observed as shown in figure 4.3-d but at the higher frequencies a greater noise component is observed.

4.1.1.6 VLSI Image

Figure 4.4 is a VLSI probe station image used as the test image for the edge detection examples.

4.1.1.7 Other programs

In addition to the DEDIP programs, other programs were used in processing the figures and images for presentation here. Gnuplot [57] was used for the surface graphs. PBMplus [33] and ghostview [58] were used to generate the encapsulated PostScript™ image files shown here.



(a) (b)

(c) (d)

Figure 4.3 Abingdon cross image.
(a) Abingdon cross without noise. (b) Abingdon cross.
(c) Log spectrum of (a). (d) Log spectrum of (b).

Figure 4.4 VLSI probe station image.

4.2 Example 1: Lowpass Filtering

Noise is introduced in digital images by imaging transducers and transmission channels. This noise tends to be spatially independent [40]. The noise energy content is often concentrated at higher frequencies than those of the image. Lowpass filtering (LPF) attenuates high frequency components thereby reducing the noise content. In this example, lowpass filtering is performed using both FIR and IIR coefficient masks. The FIR and IIR filter coefficient masks have approximately the same magnitude responses.

4.2.1 FIR Analysis

The coefficient filter mask listed in figure 4.5 is for a 6x6 order FIR filter. The coefficients are organized with the K_1 dimension being the horizontal axis. The *h(0,0)* coefficient is the first coefficient; *h(1,0)* is the next coefficient. *h(0,1)* is the first coefficient in the second row. For this first quadrant filter, figure 4.6-a shows the impulse response as a surface plot. The impulse response for a FIR filter is the same as the filter mask coefficients.

The step response is shown in figure 4.6-b. This response indicates the spatial delay following zero initial conditions which are not part of the usable response. In this case there is a six pixel delay which can be seen in the result. Comparing the top and left edges

```
DEDIP 2.0
Type : FIR Filter Mask
Author : Al Keddy
Date : Thu Jun 9 10:55:27 1994
Comment : ( 0 line(s) ):
History : ( 0 line(s) ):
Format : ascii
Word Representation : Big Endian
Height : 7
Width : 7
Depth : 32
 -0.000000   0.000000   0.000000   0.0179876  0.000000   0.000000   -0.000000
  0.000000   0.0225188  0.0391886  0.0458943  0.0391886  0.0225188  0.000000
  0.000000   0.0391886  0.0612736  0.0700362  0.0612736  0.0391886  0.000000
  0.0179876  0.0458943  0.0700362  0.0795775  0.0700362  0.0458943  0.0179876
  0.000000   0.0391886  0.0612736  0.0700362  0.0612736  0.0391886  0.000000
  0.000000   0.0225188  0.0391886  0.0458943  0.0391886  0.0225188  0.000000
 -0.000000   0.000000   0.000000   0.0179876  0.000000   0.000000   -0.000000
```

Figure 4.5 LPF FIR filter coefficients.

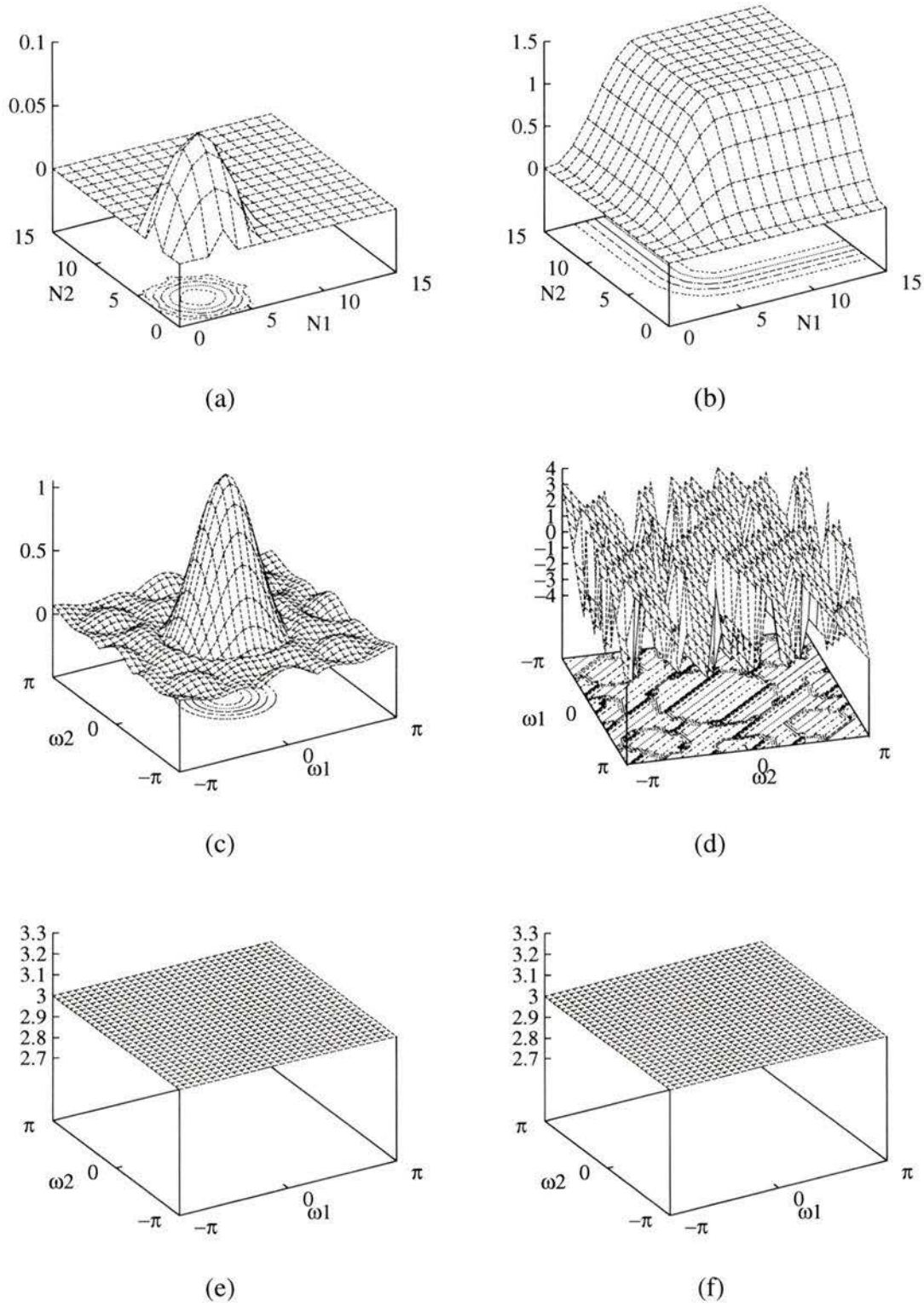


Figure 4.6 LPF FIR filter analysis.
(a) Impulse response. (b) Step response. (c) Magnitude response.
(d) Phase Response. (e) Group delay in ω_1. (f) Group delay in ω_2.

of figure 4.3-b and 4.7-a, a small black border on the top and left edges of figure 4.7-a can be observed. This is the response to the zero (black) initial conditions.

Figure 4.6-c shows the magnitude response. From the magnitude response the filter can be seen to be a lowpass filter. A lowpass filter passes frequency components near the origin of the ω_1–ω_2 frequency plane. The stopband is the region outside the passband with the near zero values where the frequency components are attenuated. This filter attenuates the higher frequencies having the effect of removing sharp transitions and high-frequency noise in an image. In figure 4.6-c of the FIR example, the stopband has non-zero ripples but because this is small compared to the passband magnitude and its effect can be considered negligible.

The phase response of figure 4.6-d is linear though this is hard to see clearly due to the phase wrap around. The group delays, the partial derivative of the phase response in ω_1 and ω_2, provide a clearer method of observing the linear phase of this filter. The ω_1 and ω_2 group delays are shown in figures 4.6-e and 4.6-f, respectively. It can be clearly seen from these two surface plots that the derivative is a non-zero constant for all frequencies. This indicates a constant slope in the phase response which is a linear phase response.

4.2.2 FIR Application

The FIR filter mask of figure 4.5 was applied to the image of figure 4.3-b using the DFilterImage program. The result is shown in figure 4.7-a. The log spectrum with *k=20* of the filtering result is shown in figure 4.7-b. The intensity levels at and near the origin are high (white) indicating a large low frequency component while the levels are near zero (black) elsewhere. Figure 4.7-c shows the magnitude response as an image. The passband is the white region and the stopband is the black region. The log spectrum with *k=20* of figure 4.7-c is shown in figure 4.7-d. In figure 4.7-b the pattern of the magnitude response shown as an image in figure 4.7-d can be observed.

4.2.3 IIR Analysis

The coefficient filter mask shown in figure 4.8 is for a 3x3 order first-quadrant IIR filter. The coefficients are organized similarly to the FIR. The first block is the numerator and the second is the denominator. The *a(0,0)* coefficient is the first coefficient, the *a(1,0)* is the next coefficient, etc. The first coefficient of the second block is *b(0,0)*.

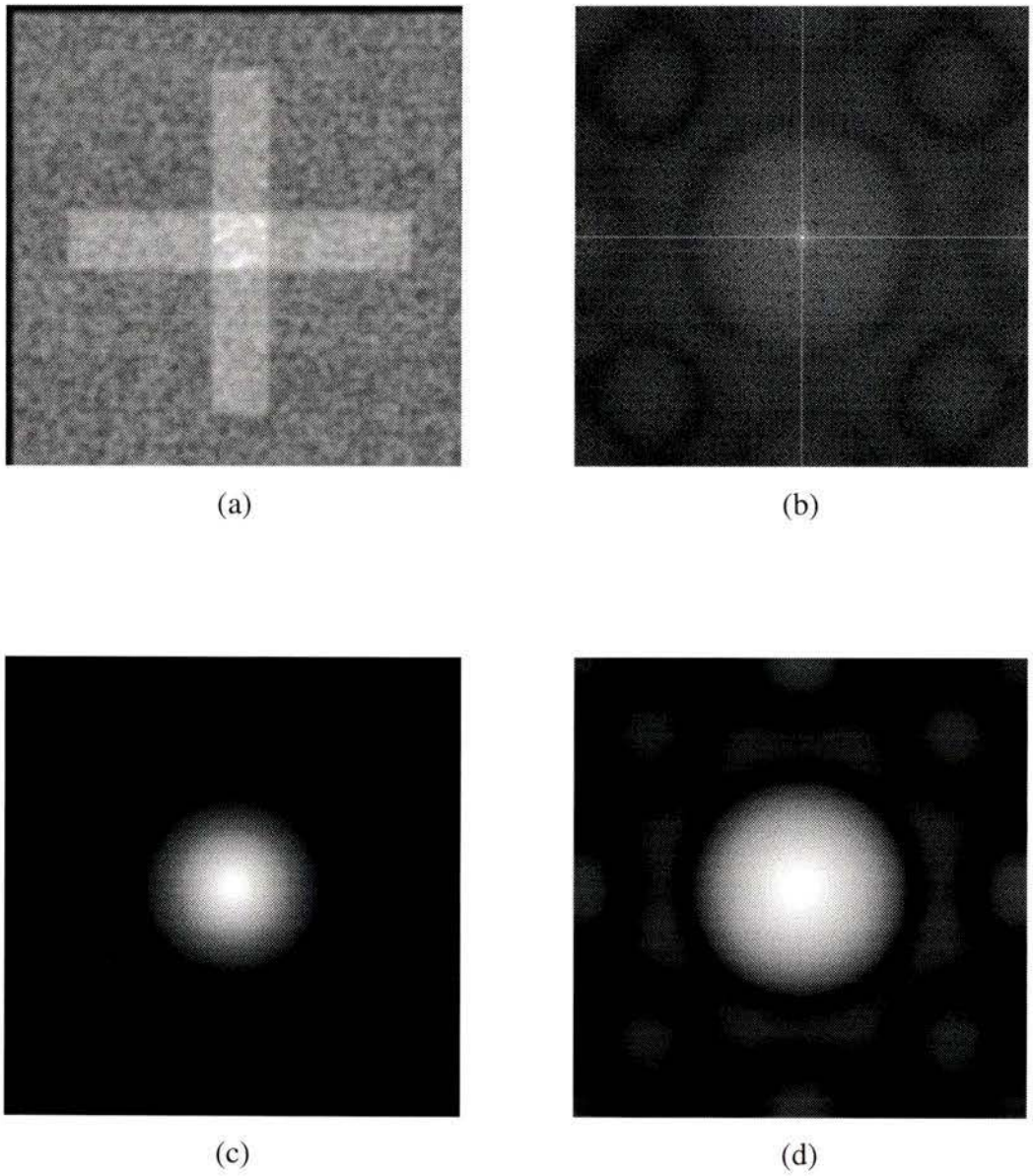


Figure 4.7 LPF FIR filter application.
(a) After filtering. (b) Log spectrum of (a) with *k=20*.
(c) LPF FIR filter magnitude response image. (d) Log spectrum of (c) with *k=20*.

```
DEDIP 2.0
Type : IIR Filter MASK
Author : Al Keddy
Date : Sun Apr 22 19:39:26 1990
Comment : ( 0 line(s) ):
History : ( 0 line(s) ):
Format : ascii
Word Representation : Big Endian
Height : 4
Width : 4
Depth : 32
Denominator :
Height : 4
Width : 4
  0.00587798 0.0199599  0.0352237  -0.00398064
  0.0292648  0.0186813  0.0167116   0.0120553
  0.0171139  0.028525   0.0404975  -0.00680827
  0.0520454  0.0131197  0.00206904 -0.011026

  1.000000  -0.425888   0.0618899   0.00627417
 -0.132873  -0.826031   0.324159   -0.0144896
  0.0706494 -0.030721   0.349173   -0.062828
 -0.0221387  0.140586  -0.131101   -0.0358049
```

Figure 4.8 LPF IIR filter coefficients.

Figure 4.9-a shows the impulse response as a surface plot. The impulse response for a IIR filter has extra ripples beyond the main impulse response. This filter is known to be stable and the summation to infinity of the absolute values of the impulse response will sum to a constant that is less than infinity.

The step response is shown in figure 4.9-b. This response indicates the spatial delay before initial conditions is not part of the response. In this case there is a nine pixel delay. Also note that there is an overshoot before steady state response is achieved.

Figure 4.9-c shows the magnitude response. From the magnitude response the filter can be seen to be a lowpass filter, passing frequencies about the origin of the ω_1-ω_2 frequency plane. The stopband attenuates the higher frequencies which has the effect of removing sharp transitions in an image. In figure 4.9-c the stopband has ripple but as with the FIR example it is small compared to the passband magnitude and its effect can be considered minimal.

The phase response of figure 4.9-d is nearly linear within the passband region. The ω_1 and ω_2 group delays of figures 4.9-e and 4.9-f, respectively, also clearly show linear phase in the region of the passband. The phase distortion in the stopband can be over-

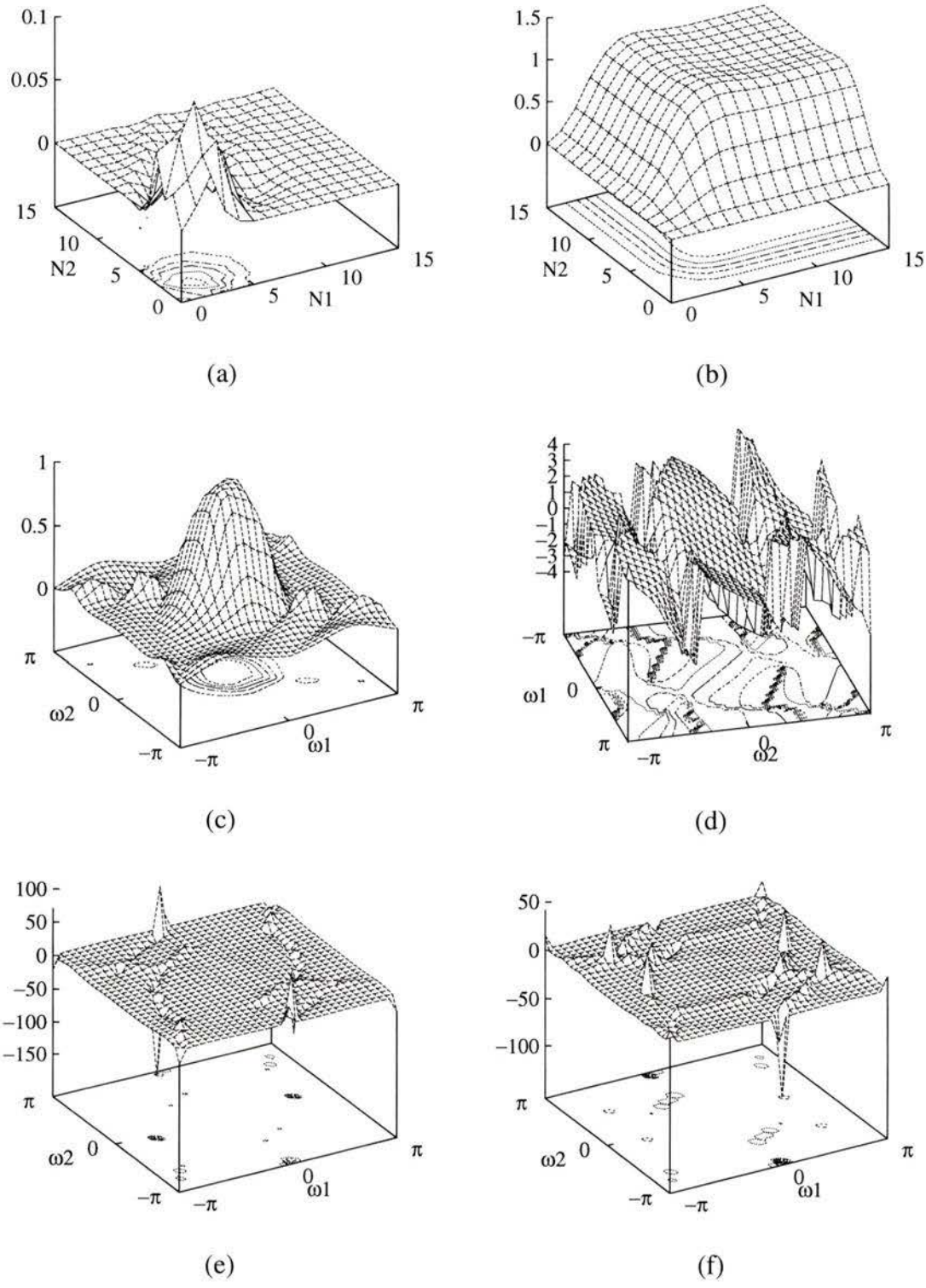


Figure 4.9 LPF IIR filter analysis.
(a) Impulse response. (b) Step response. (c) Magnitude response.
(d) Phase Response. (e) Group delay in ω_1. (f) Group delay in ω_2.

looked because the frequency components in the stopband are attenuated and have minimal contribution to the output image.

4.2.4 IIR Application

The IIR filter mask that is listed in figure 4.8 was applied to the image of figure 4.3-b using the DFilterImage program. The result is shown in figure 4.10-a. The smoothing effect of a LPF is observed. The log spectrum with $k=20$ of the filtering result is shown in figure 4.10-b. Figure 4.10-c shows the magnitude response of the IIR filter as an image and its log spectrum with $k=20$ is shown in figure 4.10-d. In figure 4.10-b the pattern of the magnitude response shown as an image in figure 4.10-d can be observed.

4.2.5 FIR and IIR Filter Performance

Though the two filters did not have exactly the same magnitude responses, they were similar and produced similar results. The processing similarities are observable in the log spectrum with $k=20$ of the filtering results of figure 4.7-b and figure 4.10-b which both pass low frequency components and attenuate the high frequency components. The FIR filters are simpler to design and do not have stability problems. IIR filters are often more difficult to design and may suffer from finite word length effects. However, IIR filters tend to be of lower order than comparable FIR filters when filters with high selectivity are required. For such a high selectivity filter the direct implementation of the IIR filter requires less computation than the comparable FIR filter.

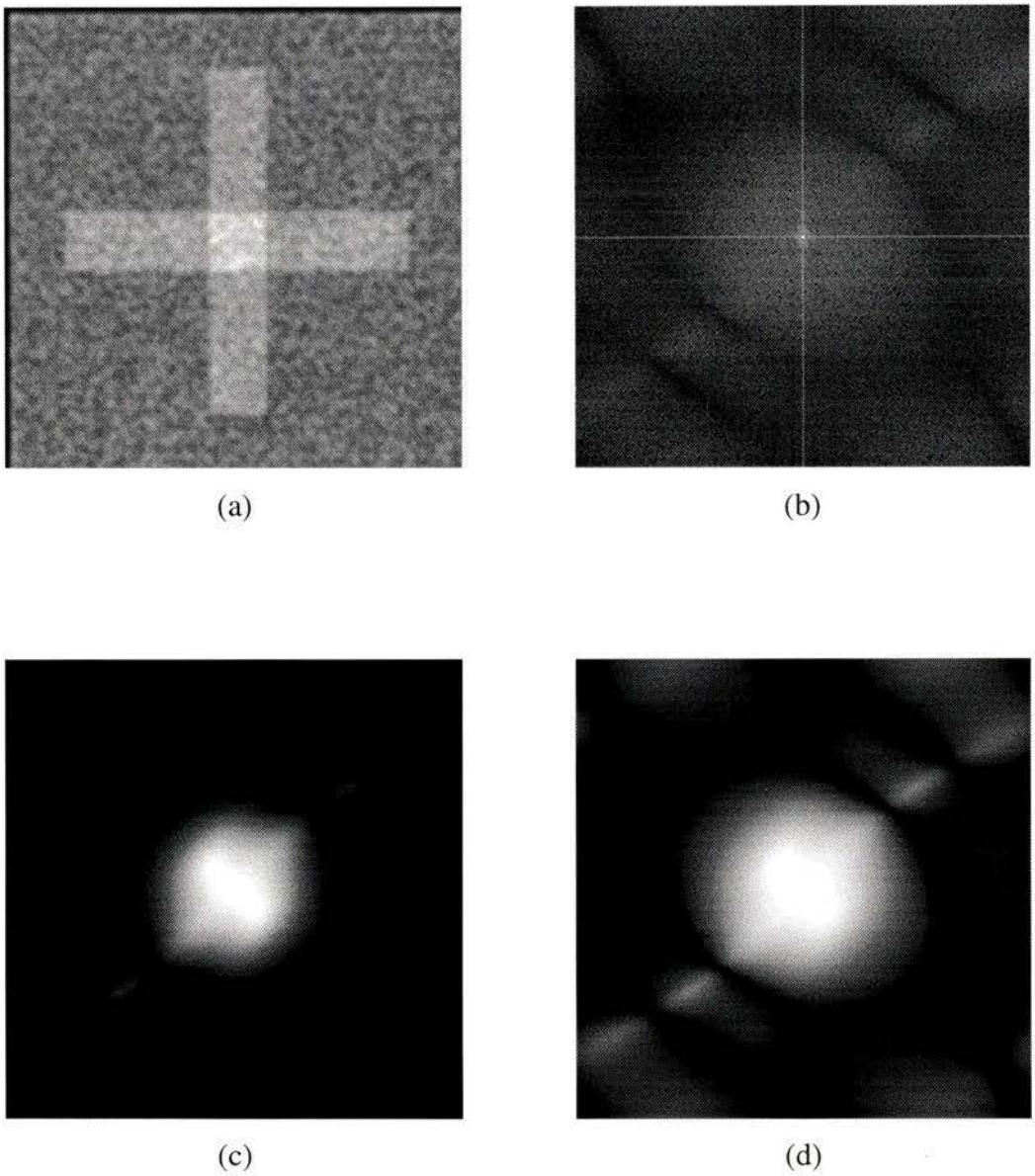


Figure 4.10 LPF IIR filter application.
(a) After filtering. (b) Log spectrum of (a) with *k=20*.
(c) LPF IIR filter magnitude response image. (d) Log spectrum of (c) with *k=20*.

4.3 Example 2: Fan Filter

The fan filter is used to highlight edges with a particular orientation in an image.

An interesting use of a fan filter is in seismic image understanding [40]. A seismic waveforms are composed of a reflection component due to acoustic reflection from substrate formations and a surface wave called a ground roll. The ground roll component is a form of noise in that it is not part of the desired signal. Analysis of the ground roll shows that its spectrum falls on a line in the ω_1–ω_2 plane and in practice the line has an angle in the 3^o to 5^o range. The fan filter with a stopband that covers the region of the spectrum including the ground roll component can be used to remove this noise.

Another possibility is to use a fan filter to extract features with a particular orientation from an image. This is done in this example using the images with horizontal and vertical rectangular features of figures 4.2-a and 4.2-g. The following section analyzes the filter in the spatial and frequency domains. The filter is applied to images with horizontal and vertical rectangular features and the results are discussed.

4.3.1 Fan Filter Analysis

The coefficient filter mask listed in figure 4.11 is for a 20x20 order first-quadrant FIR filter. The coefficients' organization was described in the LPF FIR analysis section. Figure 4.12-a shows the impulse response as a surface plot.

The step response is shown in figure 4.12-b. This response indicates the spatial shift of features between the input and output images.

Figure 4.12-c shows the magnitude response. The fan filter is so named because the magnitude spectrum resembles a fan with the ω_1–ω_2 origin being the center of rotation. Alternately the fan can be described as three lines, two symmetric about the third with all three passing through the origin. Figure 4.12-c shows that the passband is centered about the ω_2 axis.

The phase response of figure 4.12-d is linear. The ω_1 and ω_2 group delays of figures 4.12-e and 4.12-f, respectively, are constant corresponding to a linear phase.

4.3.2 Fan Filter Application

The FIR filter mask of figure 4.11 was applied to the horizontal feature image of figure 4.2-a using the DFilterImage program. The result is shown in figure 4.13-a. The log

DEDIP 2.0
Type: FIR Filter MASK
Author :
Date :
Comment : (0 line(s)):
History : (0 line(s)):
Format : ascii
Word Representation : Big Endian
Height : 21
Width : 21
Depth : 32

-3.1638550e-06 -1.1173925e-05 -1.6617200e-05 -1.0835392e-05 -8.1696408e-07 -6.0052812e-06
-3.1471531e-05 -5.3824165e-05 -4.6696937e-05 -1.6859030e-05 0.0000000e+00 -1.6859030e-05
-4.6696937e-05 -5.3824165e-05 -3.1471531e-05 -6.0052812e-06 -8.1696408e-07 -1.0835392e-05
-1.6617200e-05 -1.1173925e-05 -3.1638550e-06
-7.7420570e-06 -1.5648143e-06 -4.2368596e-06 -3.5692892e-05 -7.7926236e-05 -8.3463277e-05
-3.9505876e-05 -7.9115778e-07 -2.8727804e-05 -1.0629168e-04 -1.4853416e-04 -1.0629168e-04
-2.8727804e-05 -7.9115778e-07 -3.9505876e-05 -8.3463277e-05 -7.7926236e-05 -3.5692892e-05
-4.2368596e-06 -1.5648143e-06 -7.7420570e-06
-1.6358645e-05 -5.6754391e-05 -8.3150947e-05 -5.3548296e-05 -3.9960970e-06 -2.9129599e-05
-1.5165033e-04 -2.5806150e-04 -2.2310381e-04 -8.0379690e-05 0.0000000e+00 -8.0379690e-05
-2.2310381e-04 -2.5806150e-04 -1.5165033e-04 -2.9129599e-05 -3.9960970e-06 -5.3548296e-05
-8.3150947e-05 -5.6754391e-05 -1.6358645e-05
-3.2853737e-05 -6.4562632e-06 -1.7082659e-05 -1.4122070e-04 -3.0362068e-04 -3.2120893e-04
-1.5057552e-04 -2.9936188e-06 -1.0815187e-04 -3.9896158e-04 -5.5696556e-04 -3.9896158e-04
-1.0815187e-04 -2.9936188e-06 -1.5057552e-04 -3.2120893e-04 -3.0362068e-04 -1.4122070e-04
-1.7082659e-05 -6.4562632e-06 -3.2853737e-05
-6.7182619e-05 -2.2182628e-04 -3.1255361e-04 -1.9513391e-04 -1.4207663e-05 -1.0157606e-04
-5.2094092e-04 -8.7664781e-04 -7.5207526e-04 -2.6973353e-04 0.0000000e+00 -2.6973353e-04
-7.5207526e-04 -8.7664781e-04 -5.2094092e-04 -1.0157606e-04 -1.4207663e-05 -1.9513391e-04
-3.1255361e-04 -2.2182628e-04 -6.7182619e-05
-1.5588548e-04 -2.7545609e-05 -6.7450825e-05 -5.2616123e-04 -1.0823961e-03 -1.1072394e-03
-5.0607273e-04 -9.8771262e-06 -3.5236740e-04 -1.2903486e-03 -1.7970416e-03 -1.2903486e-03
-3.5236740e-04 -9.8771262e-06 -5.0607273e-04 -1.1072394e-03 -1.0823961e-03 -5.2616123e-04
-6.7450825e-05 -2.7545609e-05 -1.5588548e-04
-6.2855693e-04 -1.4060777e-03 -1.5933910e-03 -8.6493154e-04 -5.7206343e-05 -3.8202079e-04
-1.8658050e-03 -3.0336217e-03 -2.5436393e-03 -9.0038931e-04 0.0000000e+00 -9.0038931e-04
-2.5436393e-03 -3.0336217e-03 -1.8658050e-03 -3.8202079e-04 -5.7206343e-05 -8.6493154e-04
-1.5933910e-03 -1.4060777e-03 -6.2855693e-04
8.2400712e-04 6.0661595e-04 -1.3444512e-03 -4.4931880e-03 -6.5422682e-03 -5.5121819e-03
-2.2316251e-03 -4.0270954e-05 -1.3674390e-03 -4.8721604e-03 -6.7259575e-03 -4.8721604e-03
-1.3674390e-03 -4.0270954e-05 -2.2316251e-03 -5.5121819e-03 -6.5422682e-03 -4.4931880e-03
-1.3444512e-03 6.0661595e-04 8.2400712e-04
3.5473939e-04 1.7914995e-03 4.2040586e-03 5.3615220e-03 2.0078499e-03 -6.5456616e-03
-1.5943757e-02 -1.9365952e-02 -1.3955573e-02 -4.5777083e-03 0.0000000e+00 -4.5777083e-03
-1.3955573e-02 -1.9365952e-02 -1.5943757e-02 -6.5456616e-03 2.0078499e-03 5.3615220e-03
4.2040586e-03 1.7914995e-03 3.5473939e-04
2.7412430e-04 7.2223641e-05 2.6256248e-04 3.1003621e-03 1.0140804e-02 1.8273382e-02
1.8845531e-02 3.5730856e-03 -2.6275626e-02 -5.6761108e-02 -6.9707957e-02 -5.6761108e-02
-2.6275626e-02 3.5730856e-03 1.8845531e-02 1.8273382e-02 1.0140804e-02 3.1003621e-03
2.6256248e-04 7.2223641e-05 2.7412430e-04

Figure 4.11 Fan filter coefficients.
$h(k_1,k_2) = \{h(k_1,k_2) \mid 0 \le k_1 \le 20, 0 \le k_2 \le 9 \}$.

2.7412430e-04 7.2223641e-05 2.6256248e-04 3.1003621e-03 1.0140804e-02 1.8273382e-02
1.8845531e-02 3.5730856e-03 -2.6275626e-02 -5.6761108e-02 -6.9707957e-02 -5.6761108e-02
-2.6275626e-02 3.5730856e-03 1.8845531e-02 1.8273382e-02 1.0140804e-02 3.1003621e-03
2.6256248e-04 7.2223641e-05 2.7412430e-04
2.5614148e-04 1.1464452e-03 2.2076822e-03 1.9177018e-03 2.0013781e-04 2.1483318e-03
1.7791698e-02 5.4567785e-02 1.0717941e-01 1.5535186e-01 1.7500000e-01 1.5535186e-01
1.0717941e-01 5.4567785e-02 1.7791698e-02 2.1483318e-03 2.0013781e-04 1.9177018e-03
2.2076822e-03 1.1464452e-03 2.5614148e-04
2.7412430e-04 7.2223641e-05 2.6256248e-04 3.1003621e-03 1.0140804e-02 1.8273382e-02
1.8845531e-02 3.5730856e-03 -2.6275626e-02 -5.6761108e-02 -6.9707957e-02 -5.6761108e-02
-2.6275626e-02 3.5730856e-03 1.8845531e-02 1.8273382e-02 1.0140804e-02 3.1003621e-03
2.6256248e-04 7.2223641e-05 2.7412430e-04
3.5473939e-04 1.7914995e-03 4.2040586e-03 5.3615220e-03 2.0078499e-03 -6.5456616e-03
-1.5943757e-02 -1.9365952e-02 -1.3955573e-02 -4.5777083e-03 0.0000000e+00 -4.5777083e-03
-1.3955573e-02 -1.9365952e-02 -1.5943757e-02 -6.5456616e-03 2.0078499e-03 5.3615220e-03
4.2040586e-03 1.7914995e-03 3.5473939e-04
8.2400712e-04 6.0661595e-04 -1.3444512e-03 -4.4931880e-03 -6.5422682e-03 -5.5121819e-03
-2.2316251e-03 -4.0270954e-05 -1.3674390e-03 -4.8721604e-03 -6.7259575e-03 -4.8721604e-03
-1.3674390e-03 -4.0270954e-05 -2.2316251e-03 -5.5121819e-03 -6.5422682e-03 -4.4931880e-03
-1.3444512e-03 6.0661595e-04 8.2400712e-04
-6.2855693e-04 -1.4060777e-03 -1.5933910e-03 -8.6493154e-04 -5.7206343e-05 -3.8202079e-04
-1.8658050e-03 -3.0336217e-03 -2.5436393e-03 -9.0038931e-04 0.0000000e+00 -9.0038931e-04
-2.5436393e-03 -3.0336217e-03 -1.8658050e-03 -3.8202079e-04 -5.7206343e-05 -8.6493154e-04
-1.5933910e-03 -1.4060777e-03 -6.2855693e-04
-1.5588548e-04 -2.7545609e-05 -6.7450825e-05 -5.2616123e-04 -1.0823961e-03 -1.1072394e-03
-5.0607273e-04 -9.8771262e-06 -3.5236740e-04 -1.2903486e-03 -1.7970416e-03 -1.2903486e-03
-3.5236740e-04 -9.8771262e-06 -5.0607273e-04 -1.1072394e-03 -1.0823961e-03 -5.2616123e-04
-6.7450825e-05 -2.7545609e-05 -1.5588548e-04
-6.7182619e-05 -2.2182628e-04 -3.1255361e-04 -1.9513391e-04 -1.4207663e-05 -1.0157606e-04
-5.2094092e-04 -8.7664781e-04 -7.5207526e-04 -2.6973353e-04 0.0000000e+00 -2.6973353e-04
-7.5207526e-04 -8.7664781e-04 -5.2094092e-04 -1.0157606e-04 -1.4207663e-05 -1.9513391e-04
-3.1255361e-04 -2.2182628e-04 -6.7182619e-05
-3.2853737e-05 -6.4562632e-06 -1.7082659e-05 -1.4122070e-04 -3.0362068e-04 -3.2120893e-04
-1.5057552e-04 -2.9936188e-06 -1.0815187e-04 -3.9896158e-04 -5.5696556e-04 -3.9896158e-04
-1.0815187e-04 -2.9936188e-06 -1.5057552e-04 -3.2120893e-04 -3.0362068e-04 -1.4122070e-04
-1.7082659e-05 -6.4562632e-06 -3.2853737e-05
-1.6358645e-05 -5.6754391e-05 -8.3150947e-05 -5.3548296e-05 -3.9960970e-06 -2.9129599e-05
-1.5165033e-04 -2.5806150e-04 -2.2310381e-04 -8.0379690e-05 0.0000000e+00 -8.0379690e-05
-2.2310381e-04 -2.5806150e-04 -1.5165033e-04 -2.9129599e-05 -3.9960970e-06 -5.3548296e-05
-8.3150947e-05 -5.6754391e-05 -1.6358645e-05
-7.7420570e-06 -1.5648143e-06 -4.2368596e-06 -3.5692892e-05 -7.7926236e-05 -8.3463277e-05
-3.9505876e-05 -7.9115778e-07 -2.8727804e-05 -1.0629168e-04 -1.4853416e-04 -1.0629168e-04
-2.8727804e-05 -7.9115778e-07 -3.9505876e-05 -8.3463277e-05 -7.7926236e-05 -3.5692892e-05
-4.2368596e-06 -1.5648143e-06 -7.7420570e-06
-3.1638550e-06 -1.1173925e-05 -1.6617200e-05 -1.0835392e-05 -8.1696408e-07 -6.0052812e-06
-3.1471531e-05 -5.3824165e-05 -4.6696937e-05 -1.6859030e-05 0.0000000e+00 -1.6859030e-05
-4.6696937e-05 -5.3824165e-05 -3.1471531e-05 -6.0052812e-06 -8.1696408e-07 -1.0835392e-05
-1.6617200e-05 -1.1173925e-05 -3.1638550e-06

Figure 4.11, continued. Fan filter coefficients.
$h(k_1,k_2) = \{h(k_1,k_2) \mid 0 \le k_1 \le 20,\ 10 \le k_2 \le 20\ \}$.

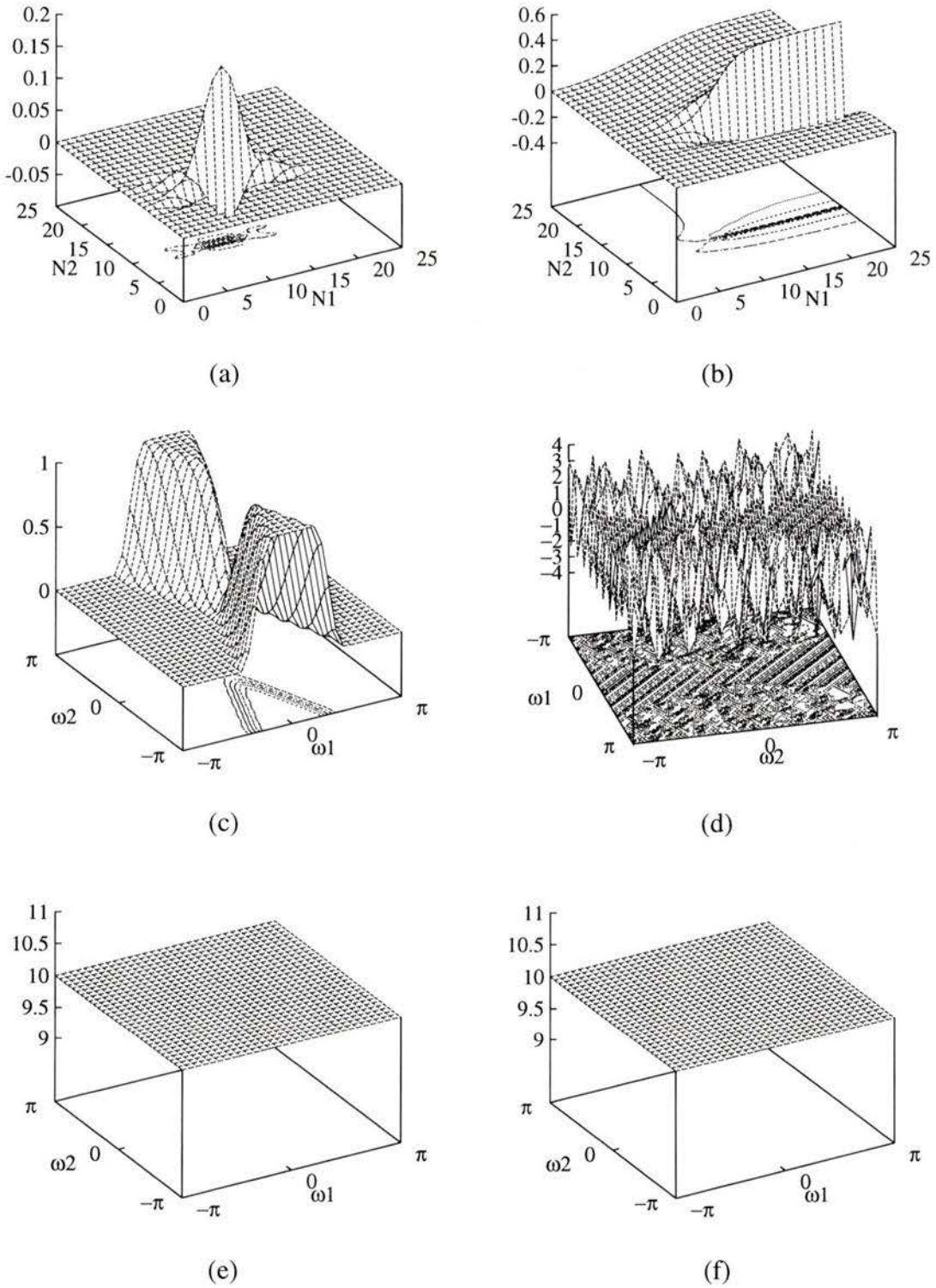


Figure 4.12 FIR Fan filter analysis.
(a) Impulse response. (b) Step response. (c) Magnitude response.
(d) Phase response. (e) Group delay in ω_1. (f) Group delay in ω_2.

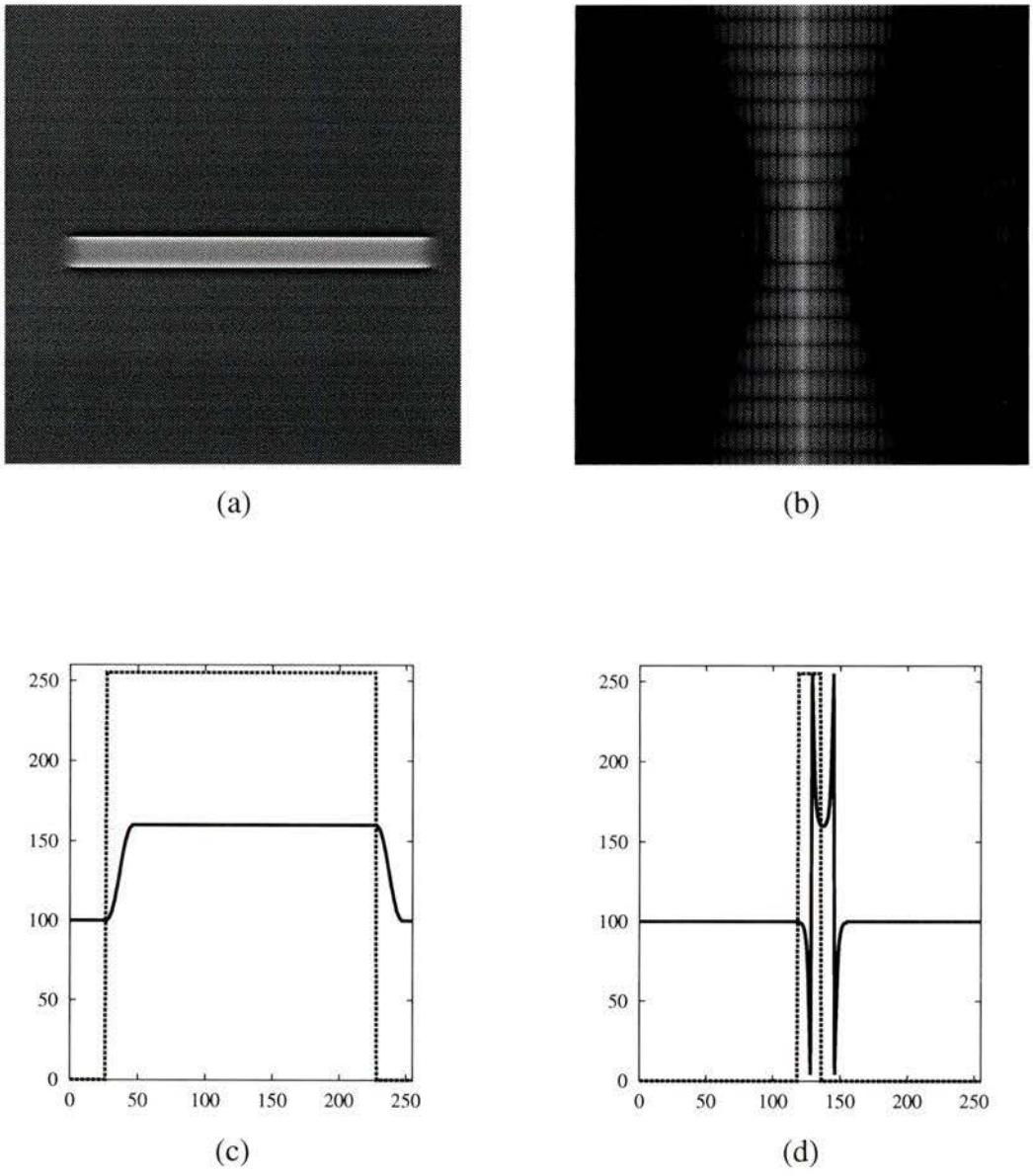


Figure 4.13 FIR fan filter application to horizontal feature image.
(a) Filtered horizontal feature image. (b) Log spectrum of (a).
(c) Horizontal cross section. (d) Vertical cross section.

spectrum of the filtering result is shown in figure 4.13-b. The log spectrum of figure 4.2-c is observed in figure 4.13-b except with darker regions on the left and right sides of the spectral image. These regions are the stopband regions where the frequency components are attenuated.

From the log spectrum of figure 4.13-b it would be expected that the vertical edge whose gradient is aligned with the horizontal (ω_1) axis would be attenuated. Figure 4.13-a shows the horizontal edges highlighted and the vertical edges attenuated which confirms the expectation from observing the log spectrum. The horizontal and vertical cross-sections through the center of the features of figures 4.2-a and 4.13-a are shown in figures 4.13-c and 4.13-d, respectively. A cross-section corresponding to figure 4.2-a is shown as a dotted line and from figure 4.13-a as a solid line. The delay shown in the step response can be clearly seen in the cross-sections. The attenuation of the vertical edge is observed in figure 4.13-c. Figure 4.13-d shows the highlighting of the horizontal edge.

The FIR filter mask of figure 4.11 was also applied to the vertical feature image of figure 4.2-g using the DFilterImage program. The result is shown in figure 4.14-a. The log spectrum of the filtering result is shown in figure 4.14-b. The spectrum of figure 4.2-i is observed in figure 4.14-b except for the dark stopband regions.

Again the horizontal edges are highlighted and the vertical edges are attenuated which is confirmed in figure 4.14-a. The horizontal and vertical cross-sections through the center of the features of figures 4.2-g and 4.14-a are shown in figures 4.14-c and 4.14-d, respectively. A figure 4.2-g cross-section is shown as the dotted line and a figure 4.14-a cross-section as the solid line. The delay shown in the step response can be observed in the cross-sections. The attenuation of the long vertical edge is observed in figure 4.14-c. Figure 4.14-d shows the highlighting of the horizontal edge.

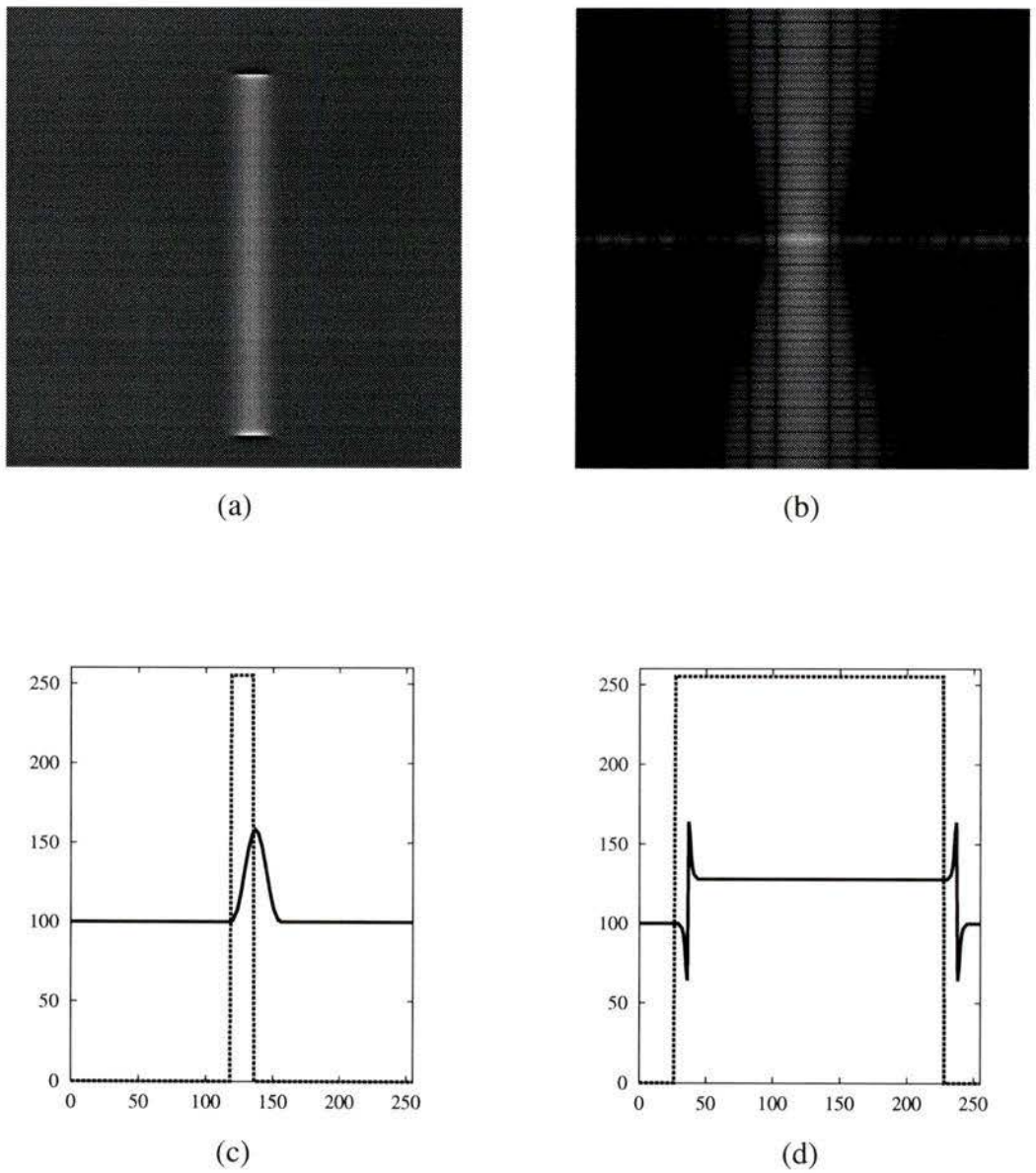


Figure 4.14 FIR fan filter application to vertical feature image.
(a) Filtered vertical feature image. (b) Log spectrum of (a).
(c) Horizontal cross section. (d) Vertical cross section.

4.4 Example 3: Edge-Detection Operators

In image analysis, one method used to segment an image into a set of features is edge detection. An edge is defined as the boundary of a feature and can be characterized by a sharp intensity transition in an image.

The problem of identifying edges in an image has been approached as a spatial-domain problem. Maxima and minima in the gradient of an image correspond to changes in intensity. An edge map which is a binary image of edges is obtained from the gradient by thresholding and thinning. The threshold is a setting of steepness of the intensity changes (slope of the edges) above which is considered an edge. Thinning is necessary because an edge such as a ramp edge can have a constant slope over multiple pixels producing a wide edge after thresholding. Feature extraction algorithms require edge maps with single pixel wide edges.

The computation of the image gradient is performed by computing the directionally-oriented first derivatives in the X and Y directions. For ease of understanding, correlation templates are used to describe the filtering. The correlation operation shown in equation 4.3 applies the template, $d(x,y)$, to the image, $I(x,y)$, to produce the result, $R(x,y)$.

$$R(x, y) = \sum_{(k_1, k_2) \in K} \sum d(k_1, k_2)\, I(x + k_1, y + k_2) \tag{4.3}$$

where $K = \{(k_1,k_2) |\ K_{1min} \le k_1 \le K_{1max};\ K_{2min} \le k_2 \le K_{2max}\}$. In the following discussion, the notation of equation 4.4 will be used as a shorthand notation for the correlation template.

$$\boldsymbol{d} = \begin{bmatrix} d(k_{1min}, k_{2min}) & \cdots & d(k_{1max}, k_{2min}) \\ \vdots & & \vdots \\ d(k_{1min}, k_{2max}) & \cdots & d(k_{1max}, k_{2max}) \end{bmatrix} \tag{4.4}$$

Correlation is quite similar to FIR filtering. The correlation template can be implemented using as an FIR filter by reversing the order of the coefficients in both k_1 and k_2 directions. For the correlation template in equation 4.5, the equivalent FIR filter coefficient mask is shown in equation 4.6.

$$\text{correlation template} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, K = \{ (k_1, k_2) \mid 0 \le k_1 \le 2, 0 \le k_2 \le 2 \} \tag{4.5}$$

$$\text{FIR filter mask} = \begin{bmatrix} i & h & g \\ f & e & d \\ c & b & a \end{bmatrix}, K = \{ (k_1, k_2) \mid 0 \le k_1 \le 2, 0 \le k_2 \le 2 \} \tag{4.6}$$

The analysis was performed by converting the correlation templates into 2-D FIR filter masks and analyzing them as FIR filters.

The gradient approximation image is computed from two directional-derivative filtering results. Two methods for this computation are presented and discussed. From the gradient approximation image an edge map image is computed by thresholding and thinning.

4.4.1 First Directional-Derivative Operators

The gradient of a continuous 2-D function, $f(\tilde{x}, \tilde{y})$ is defined in terms of the directionally-oriented derivatives in equation 4.7 [59,50]

$$\nabla f(\tilde{x}, \tilde{y}) = \begin{bmatrix} \dfrac{\partial f(\tilde{x}, \tilde{y})}{\partial \tilde{x}} \\ \dfrac{\partial f(\tilde{x}, \tilde{y})}{\partial \tilde{y}} \end{bmatrix} \tag{4.7}$$

where $\tilde{x}$ and $\tilde{y}$ denote continuous variables.

The two-point approximations used to estimate the directionally-oriented derivatives are shown in equations 4.8 and 4.9.

$$\frac{\partial f(\tilde{x}, \tilde{y})}{\partial \tilde{x}} \approx \frac{f(\tilde{x} + \Delta\tilde{x}, \tilde{y}) - f(\tilde{x}, \tilde{y})}{\Delta\tilde{x}} \tag{4.8}$$

$$\frac{\partial f(\tilde{x}, \tilde{y})}{\partial \tilde{y}} \approx \frac{f(\tilde{x}, \tilde{y} + \Delta\tilde{y}) - f(\tilde{x}, \tilde{y})}{\Delta\tilde{y}} \tag{4.9}$$

With digital images $\Delta\tilde{x} = \Delta\tilde{y} = 1$ and the two-point approximations to the directionally-oriented derivatives of equations 4.8 and 4.9 become the correlation operations D_x and D_y. Equations 4.10 and 4.11 are the image operation equations that correspond to

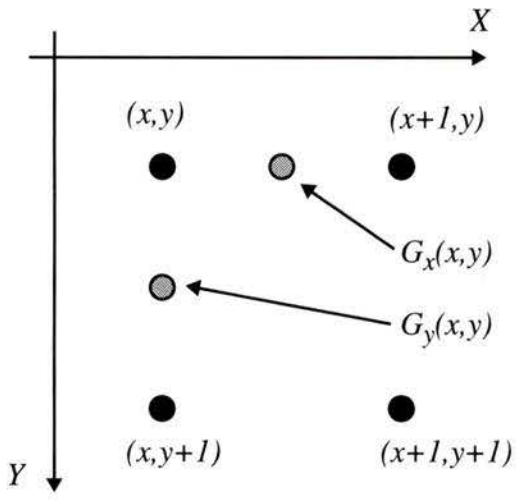


Figure 4.15 Position of two-point derivative approximations.

the operations D_x and D_y, respectively. $G_x(x,y)$ and $G_y(x,y)$ are images resulting from the application the directional-derivative operations D_x and D_y in the X and Y directions, respectively.

$$G_x(x, y) = D_x[I(x, y)] = I(x+1, y) - I(x, y) \tag{4.10}$$

$$G_y(x, y) = D_y[I(x, y)] = I(x, y+1) - I(x, y) \tag{4.11}$$

The cross-correlation templates $\boldsymbol{d_x}$ and $\boldsymbol{d_y}$ for equations 4.10 and 4.11 are shown in equations 4.12 and 4.13, respectively.

$$\boldsymbol{d_x} = \begin{bmatrix} -1 & 1 \end{bmatrix}, K = \{ (k_1, k_2) \,|\, k_1 = \{0, 1\}, k_2 = \{0\} \} \tag{4.12}$$

$$\boldsymbol{d_y} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, K = \{ (k_1, k_2) \,|\, k_1 = \{0\}, k_2 = \{0, 1\} \} \tag{4.13}$$

Equations 4.10 and 4.11 best approximate the derivatives at the midpoint of their respective intervals. The problem is the approximations, $G_x(x,y)$ and $G_y(x,y)$, at a given point (x,y) are at different locations than the input point $I(x,y)$ as shown in figure 4.15.

For discrete derivative approximations, the central difference approximation of equations 4.14 and 4.15 positions the directional derivatives at the proper location.

$$\frac{\partial f(\tilde{x}, \tilde{y})}{\partial \tilde{x}} \approx \frac{f(\tilde{x} + \Delta\tilde{x}, \tilde{y}) - f(\tilde{x} - \Delta\tilde{x}, \tilde{y})}{2(\Delta\tilde{x})} \tag{4.14}$$

$$\frac{\partial f(\tilde{x}, \tilde{y})}{\partial \tilde{y}} \approx \frac{f(\tilde{x}, \tilde{y} + \Delta\tilde{y}) - f(\tilde{x}, \tilde{y} - \Delta\tilde{y})}{2(\Delta\tilde{y})} \tag{4.15}$$

With $\Delta\tilde{x} = \Delta\tilde{y} = 1$, the central difference approximations of equations 4.14 and 4.15 become the correlation operations D_{cx} and D_{cy}. Equations 4.16 and 4.17 are the image operation equations that correspond to operations D_{cx} and D_{cy}, respectively. The scale factor of 1/2 is omitted during the filtering stage. It is later included in the selection of the threshold value.

$$D_{cx}[I(x, y)] = I(x+1, y) - I(x-1, y) \tag{4.16}$$

$$D_{cy}[I(x, y)] = I(x, y+1) - I(x, y-1) \tag{4.17}$$

The correlation masks for D_{cx} and D_{cy} operations are shown as $\boldsymbol{d_{cx}}$ and $\boldsymbol{d_{cy}}$ in equations 4.18 and 4.19, respectively.

$$\boldsymbol{d}_{cx} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, K = \{ (k_1, k_2) \,|\, k_1 = \{-1, 0, 1\}, k_2 = \{0\} \} \tag{4.18}$$

$$\boldsymbol{d}_{cy} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, K = \{ (k_1, k_2) \,|\, k_1 = \{1\}, k_2 = \{-1, 0, 1\} \} \tag{4.19}$$

The directional derivatives described so far tend to be sensitive to noise in the direction orthogonal to the operator. A method to reduce the effect of noise is to combine the derivative operators, $\boldsymbol{d_{cx}}$ and $\boldsymbol{d_{cy}}$, with a component, $\boldsymbol{d_{ave}}$, which averages orthogonally to the operator and thus reduces the effect of noise. This leads to a collection of 3x3 operators which consists of a profile (the directional-derivative operator) component and a projection (the component that averages) component [59]. These edge operators are usually represented as the correlation templates $\boldsymbol{d_x}$ and $\boldsymbol{d_y}$, shown in equations 4.20 and 4.21, respectively.

$$\boldsymbol{d}_x = \boldsymbol{d}_{ave} \cdot \boldsymbol{d}_{cx} \tag{4.20}$$

$$\boldsymbol{d}_y = \boldsymbol{d}_{cy} \cdot \boldsymbol{d}'_{ave} \tag{4.21}$$

The Prewitt and Sobel operators are the two most common 3x3 directional derivation operators. Both operators use the $\boldsymbol{d_{cx}}$ and $\boldsymbol{d_{cy}}$ correlation templates of equations 4.18 and 4.19 to obtain the directional-derivative images $G_x(x,y)$ and $G_y(x,y)$, respectively. But, the projection components are different.

The projection component, $\boldsymbol{d}_{ave}$, of the Prewitt operators is simple three pixel averaging shown in equation 4.22.

$$\boldsymbol{d}_{ave} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, K = \{ (k_1, k_2) \,|\, k_1 = \{0\}, k_2 = \{-1, 0, 1\} \} \tag{4.22}$$

Equation 4.23 shows how the $\boldsymbol{d}_{ave}$ and $\boldsymbol{d}_{cx}$ combine to form the *X*-oriented 3x3 Prewitt correlation template that enhances vertical edges aligned on the zeros [59, 60]. It takes the *X*-direction gradient at the center of the template. Equation 4.24 shows the *Y*-oriented 3x3 Prewitt correlation template that enhances horizontal edges aligned with the zeros of the template. It takes the *Y*-direction gradient at the center of the template.

$$\boldsymbol{d}_x = \boldsymbol{d}_{ave} \cdot \boldsymbol{d}_{cx} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \tag{4.23}$$

$$K = \{ (k_1, k_2) \,|\, k_1 = \{-1, 0, 1\}, k_2 = \{-1, 0, 1\} \}$$

$$\boldsymbol{d}_y = \boldsymbol{d}_{cy} \cdot \boldsymbol{d}'_{ave} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \tag{4.24}$$

$$K = \{ (k_1, k_2) \,|\, k_1 = \{-1, 0, 1\}, k_2 = \{-1, 0, 1\} \}$$

The Sobel operators also use three pixel averaging but with extra weight on the center pixel with $\boldsymbol{d}_{ave}$ shown in equation 4.25.

$$\boldsymbol{d}_{ave} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}, K = \{ (k_1, k_2) \,|\, k_1 = \{0\}, k_2 = \{-1, 0, 1\} \} \tag{4.25}$$

Equation 4.26 shows how the $\boldsymbol{d}_{ave}$ and $\boldsymbol{d}_{cx}$ combine to form the *X*-oriented 3x3 Sobel correlation template that filters for vertical edges aligned on the zeros of the template [50, 59, 60]. It takes the *X*-direction gradient at the center of the template. Equation 4.27 shows the *Y*-oriented 3x3 Sobel correlation template that filters for horizontal edges aligned with the correlation template's zeros. It takes the *Y*-direction gradient at the center of the template.

$$\boldsymbol{d}_x = \boldsymbol{d}_{ave} \cdot \boldsymbol{d}_{cx} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{4.26}$$

$$K = \{ (k_1, k_2) \,|\, k_1 = \{-1, 0, 1\}, k_2 = \{-1, 0, 1\} \}$$

$$\boldsymbol{d}_y = \boldsymbol{d}_{cy} \cdot \boldsymbol{d}'_{ave} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{4.27}$$

$$K = \{ (k_1, k_2) \,|\, k_1 = \{-1, 0, 1\}, k_2 = \{-1, 0, 1\} \}$$

A set of operators attributed to Rosenfeld and Thurston [60] use simple three pixel averaging as did the Prewitt operators but added averaging to the profile component which averages the gradient. Equation 4.28 shows how the $\boldsymbol{d}_{ave}$ and $\boldsymbol{d}_{cx}$ combine to form the *X*-oriented 5x3 Rosenfeld-Thurston correlation template that filters for vertical edges aligned with its zeros. It takes the *X*-direction gradient at the center of the template. Equation 4.29 shows the *Y*-oriented 3x5 Rosenfeld-Thurston correlation template that filters for horizontal edges aligned with the zeros. It takes the *Y*-direction gradient at the center of the template.

$$\boldsymbol{d}_x = \boldsymbol{d}_{ave} \cdot \boldsymbol{d}_{cx} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & -1 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 & 1 \\ -1 & -1 & 0 & 1 & 1 \end{bmatrix} \tag{4.28}$$

$$K = \{ (k_1, k_2) \,|\, k_1 = \{-2, -1, 0, 1, 2\}, k_2 = \{-1, 0, 1\}$$

$$\boldsymbol{d}_y = \boldsymbol{d}_{cy} \cdot \boldsymbol{d}'_{ave} = \begin{bmatrix} -1 \\ -1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{4.29}$$

$$K = \{ (k_1, k_2) \,|\, k_1 = \{-1, 0, 1\}, k_2 = \{-2, -1, 0, 1, 2\}$$

4.4.2 Directional-Derivative Operator Analysis

The magnitude of the frequency responses for the *X* -direction simple difference, Prewitt, Sobel and Rosenfeld-Thurston correlation operators are shown in figure 4.16. The corre-

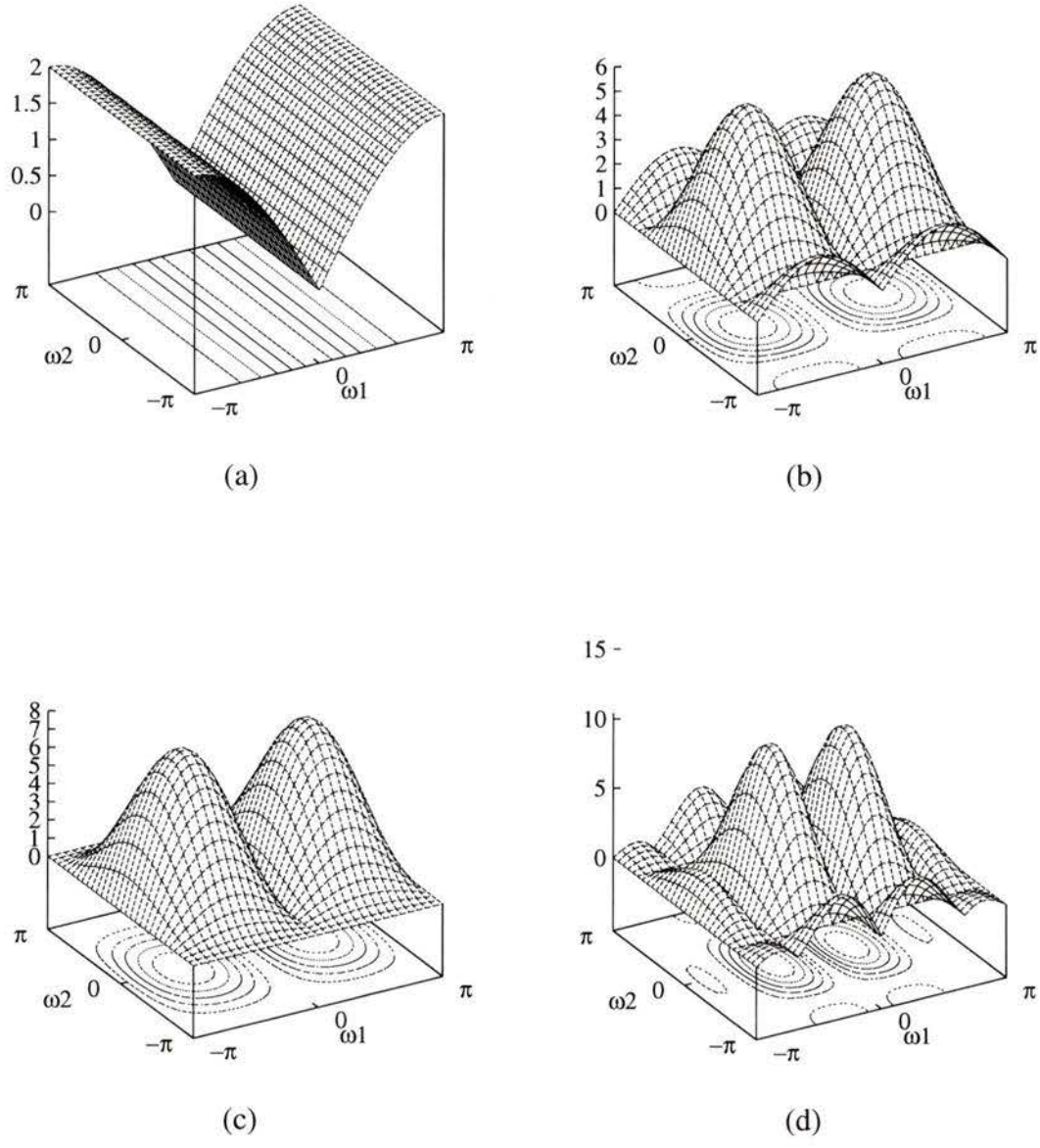


Figure 4.16 Magnitude responses for $\boldsymbol{d}_x$ directional derivative operators.
(a) Simple operator. (b) Prewitt operator.
(c) Sobel operator. (d) Rosenfeld-Thurston operator.

sponding *Y*-direction magnitude responses are the same as the *X*-direction magnitude responses except they are rotated 90^0 on the ω_1–ω_2 plane. Figure 4.16-a is the magnitude response for equation 4.12. This response indicates a highpass filter in the ω_1 direction which in the spatial domain corresponds to the orientation of the *X* axis and the profile component of the operator. The projection component in the *Y* direction is an allpass filter in the ω_2 direction. This filter has poor noise performance in the Y direction due to the allpass projection component. This can result in a discontinuous edge where a constant edge should be.

The magnitude response for the Prewitt $\boldsymbol{d_x}$ operator is shown in figure 4.16-b. The profile component, in the ω_1 direction, is a bandpass filter. This provides better noise performance. The projection component is also a bandpass filter but the stopband has a large ripple at the $\pm\pi$ positions on the ω_2 axis. This allows high frequency noise to be passed in the direction of the projection component.

The magnitude response for the Sobel $\boldsymbol{d_x}$ operator is shown in figure 4.16-c. The profile component, in the ω_1 direction, is a bandpass filter with the same response as the Prewitt operators. Again this provides good noise performance in the profile component direction. The projection component is also a bandpass filter with no ripples in the stopband at the $\pm\pi$ positions on the ω_2 axis. This provides better noise performance in the projection component direction.

The magnitude response for the Rosenfeld-Thurston $\boldsymbol{d_x}$ operator is shown in figure 4.16-d. This is a bandpass filter in both the profile and projection directions. It has the ripple like the Prewitt operator in the projection component as well as in profile component direction.

The step responses for these four operators are shown in figures 4.17-a to 4.17-d. These responses indicate the coverage of the correlation operators. The width of the step response is the minimum feature that will be enhanced by the operator. Figure 4.17-a shows the simple difference operator has a width of two pixels. The Prewitt and Sobel $\boldsymbol{d_x}$ operators have a three pixel width shown in figures 4.17-b and 4.17-c, respectively. The Rosenfeld-Thurston $\boldsymbol{d_x}$ operator has a five-pixel width shown in figure 4.17-d. The effect of the width is exhibited in the blurring of the edge profiles. Comparison of the filtering results of figures 4.18-a to 4.18-d shows an increasing blurring of the vertical edges from the simple difference to the Rosenfeld-Thurston operators.

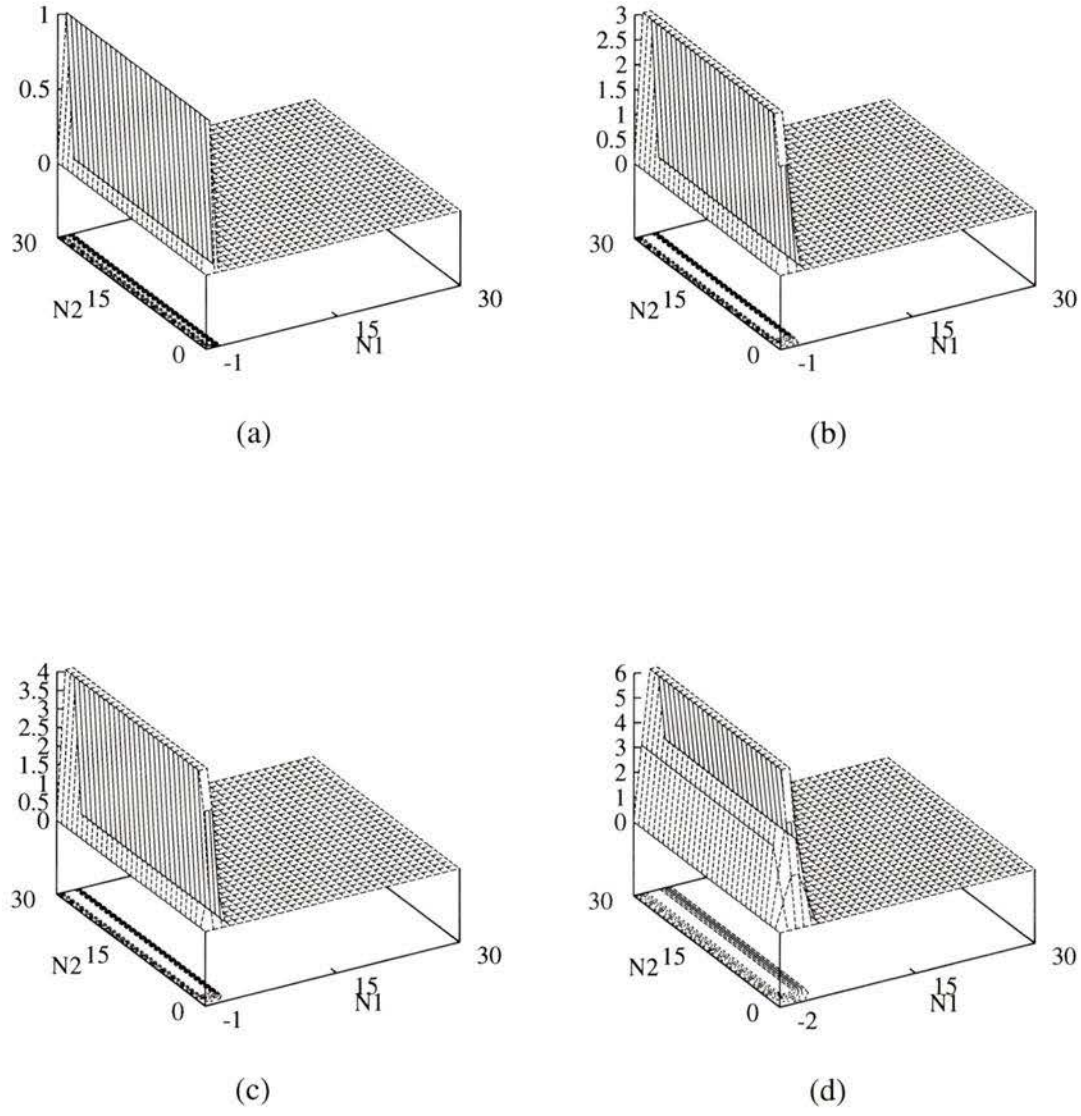


Figure 4.17 Step responses for $\boldsymbol{d}_x$ directional derivative operators.
(a) Simple operator. (b) Prewitt operator.
(c) Sobel operator. (d) Rosenfeld-Thurston operator.

4.4.3 Directional-Derivative Operator Application

The directional-derivative operators, $\boldsymbol{d}_x$ and $\boldsymbol{d}_y$, for simple difference, Prewitt, Sobel and Rosenfeld-Thurston operators were applied to the VLSI probe station image of figure 4.4 using the DWeightedAverageImage program. This program implements the cross-correlation operation. It assumes that the cross-correlation mask has odd dimensions so that the output is placed at the center of the operator. This is a form of four quadrant filtering.

Figures 4.18-a to 4.18-d show the filtering results, $G_x(x,y)$, for the simple difference, Prewitt, Sobel and Rosenfeld-Thurston X directional derivative operators, respectively. The corresponding Y directional derivatives, $G_y(x,y)$, are shown in figure 4.19. The images in figures 4.18 and 4.19 have been histogram equalized for printing clarity. The dark areas indicate a negative slope and the light areas indicate a positive slope in the images of figure 4.18. Taking an arbitrary row of pixels through the probe shows a dark area at the left of the probe. This negative slope represents the intensity transition from white to black along a corresponding row of pixels in figure 4.4 while moving from left to right.

4.4.3.1 Gradient Image Computation

The directional-derivative operators are used to obtain directionally-oriented edge information. It is also desired to obtain directionally invariant edge information. This is achieved by computing an approximation to the gradient based on $G_x(x,y)$ and $G_y(x,y)$.

One method of computing the gradient is to take the magnitude (MaG) of $G_x(x,y)$ and $G_y(x,y)$ as in equation 4.30 [59, 50].

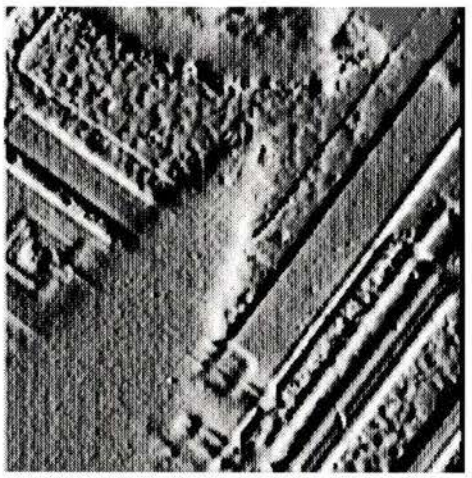
$$\text{MaG}\,[G_x(x,y), G_y(x,y)] = \sqrt{G_x^2(x,y) + G_y^2(x,y)} \tag{4.30}$$

The problem with this method is that a bias of $\sqrt{2}$ is introduced to edges at 45^o.

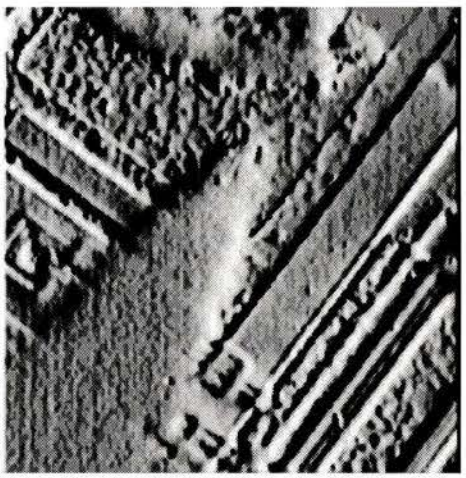
An alternate method that is computationally simpler and does not introduce a bias is the Maximum of the Absolute (MoA) values of the $G_x(x,y)$ and $G_y(x,y)$ shown in equation 4.31 [59].

$$\text{MoA}[G_x(x,y), G_y(x,y)] = \text{maximum}\,(|G_x(x,y)|, |G_y(x,y)|) \tag{4.31}$$

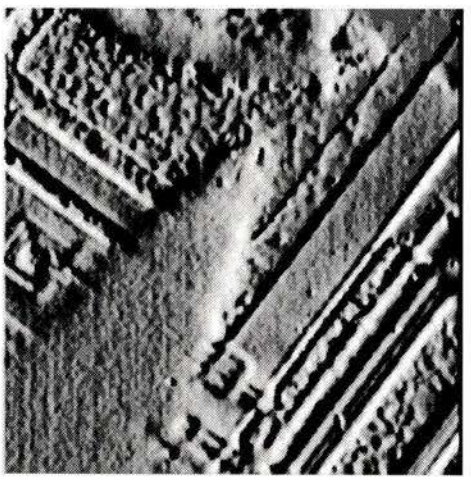
The X and Y directional-derivative filtering results, $G_x(x,y)$ and $G_y(x,y)$, post-filtered by MaG and MoA methods to obtain gradient images are shown in figures 4.20 and 4.21, respectively. These images have been histogram equalized and negated to display



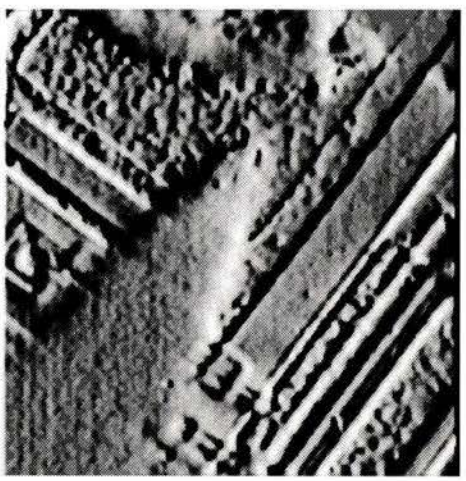
(a)



(b)

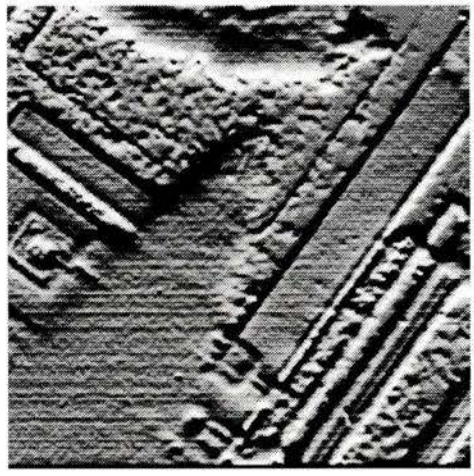


(c)

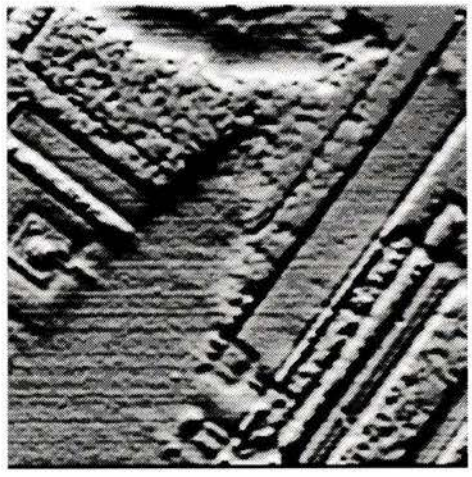


(d)

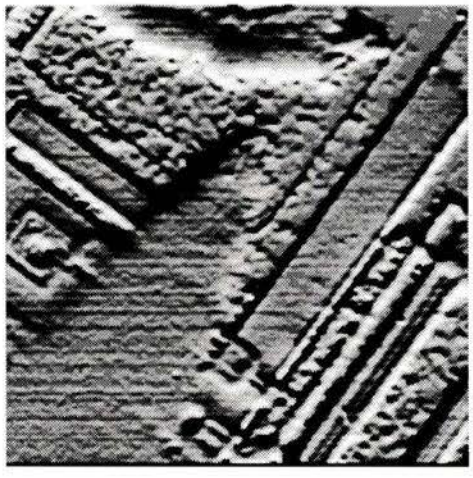
Figure 4.18 Filtering results: $G_x(x,y)$.
(a) Simple operator. (b) Prewitt operator.
(c) Sobel operator. (d) Rosenfeld-Thurston operator.



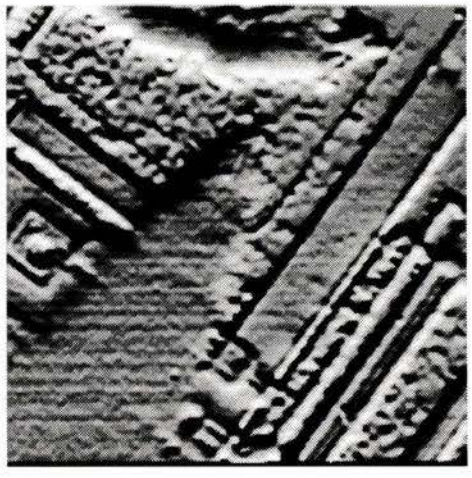
(a)



(b)

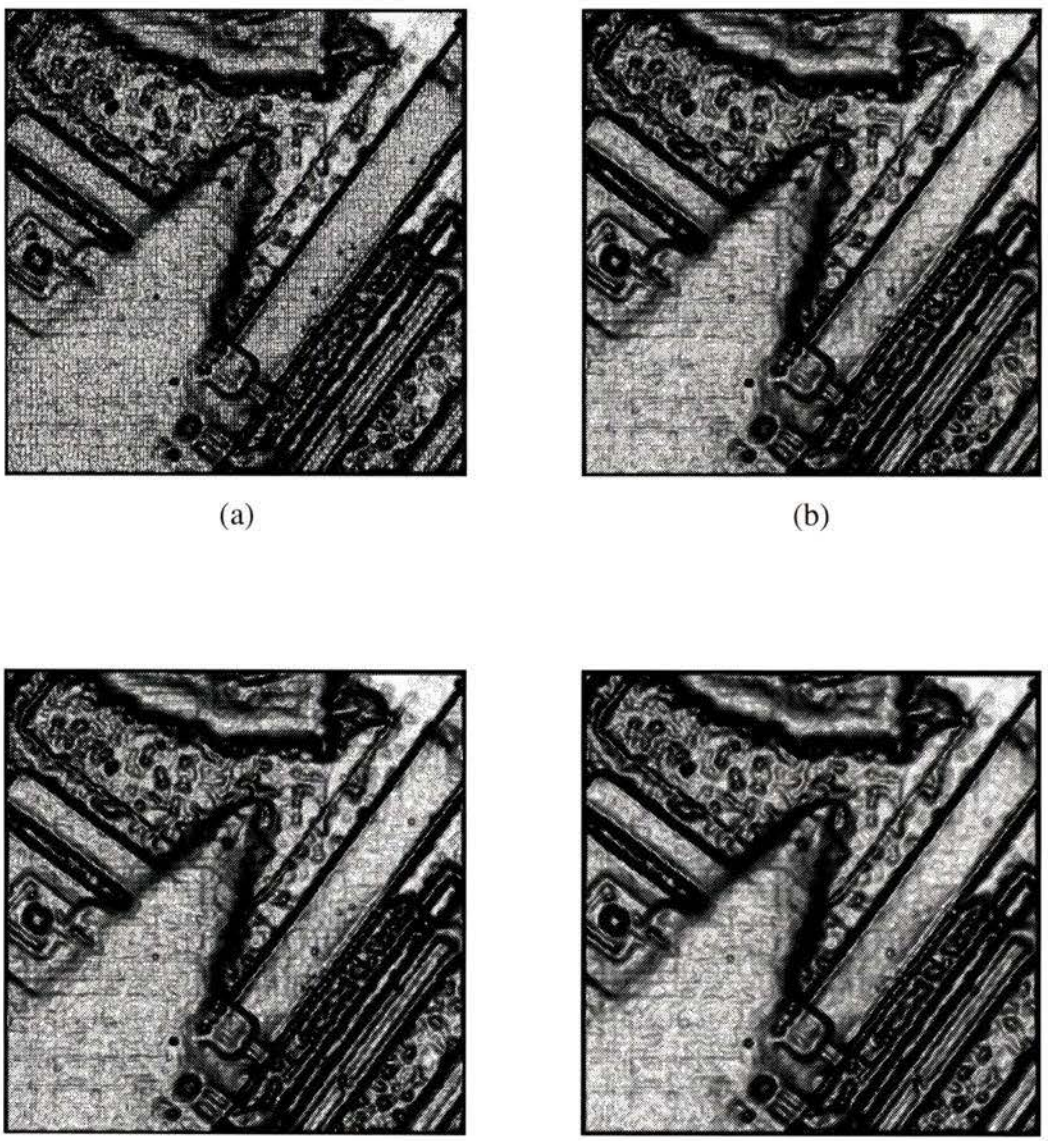


(c)



(d)

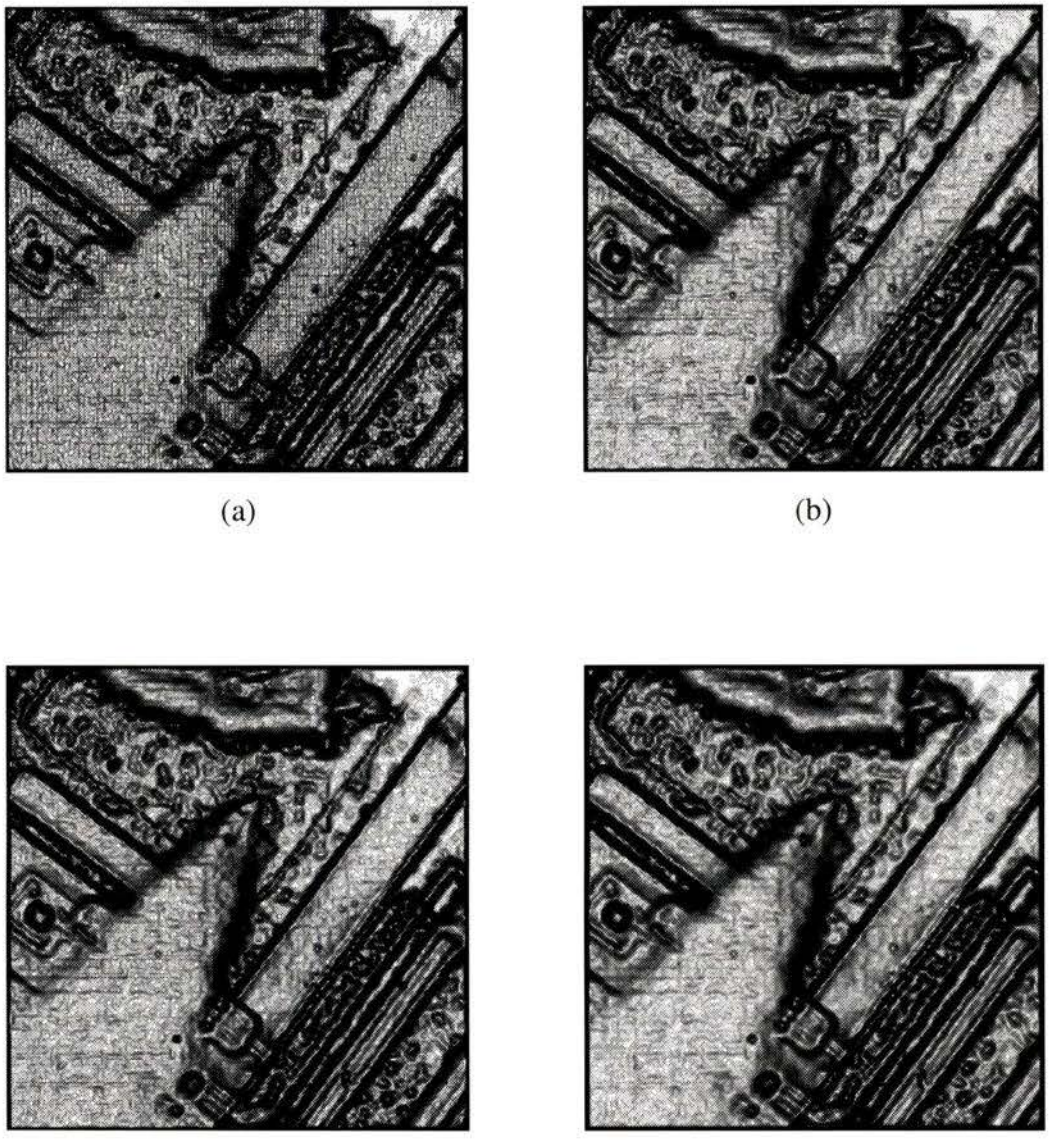
Figure 4.19 Filtering results: $G_y(x,y)$.
(a) Simple operator. (b) Prewitt operator.
(c) Sobel operator. (d) Rosenfeld-Thurston operator.



(a) (b)

(c) (d)

Figure 4.20 Magnitude of $G_x(x,y)$ and $G_y(x,y)$.
(a) Simple operator. (b) Prewitt operator.
(c) Sobel operator. (d) Rosenfeld-Thurston operator.



(a) (b)

(c) (d)

Figure 4.21 Maximum of Absolute of $G_x(x,y)$ and $G_y(x,y)$.
(a) Simple operator. (b) Prewitt operator.
(c) Sobel operator. (d) Rosenfeld-Thurston operator.

the gradient clearly. The darker intensity levels correspond to steeper gradients and the lighter levels indicate a slight gradient.

4.4.3.2 Edge Map Computation

MaG and MoA methods are used to compute gradient images. For feature extraction algorithms an edge map of distinct edges is required. To obtain an edge map the gradient image is thresholded at a value that defines a certain gradient steepness as an edge. The resulting binary image can have edges that are multiple pixels wide which are not distinct. The edges in the thresholded edge map image are then thinned to single pixel width. The thresholding and thinning are performed by the same program which was developed based on the DEDIP libraries and the algorithms described in [61, 62].

Figures 4.22-a to 4.22-d show the thresholded and thinned MaG computed gradient images for the simple difference, Prewitt, Sobel and Rosenfeld-Thurston operators, respectively. The threshold values are 8, 30, 48 and 60 for the respective operators. The selection of the threshold values can vary the quality and quantity of the feature edges. Figure 4.23 shows the corresponding thresholded and thinned edge maps for the MoA computed gradient images. The images in figures 4.22 and 4.23 have been negated to show edges as black lines.

Comparing the different operators in figures 4.22-a to 4.22-d shows many missed feature edges in the simple difference operator of figures 4.22-a while the Rosenfeld-Thurston operators produce a fairly detailed edge map from the MaG gradient images. The variation can be attributed to the operators, the threshold value and the bias introduced by the MaG computations.

The MoA edge maps of figures 4.23-a to 4.23-d have variations between the operators similar to that of the MaG edge maps. Graphical comparison of the MaG and MoA operators are less clear but the MoA edge map method has the advantages of unbiased edge gradients and ease of computation.

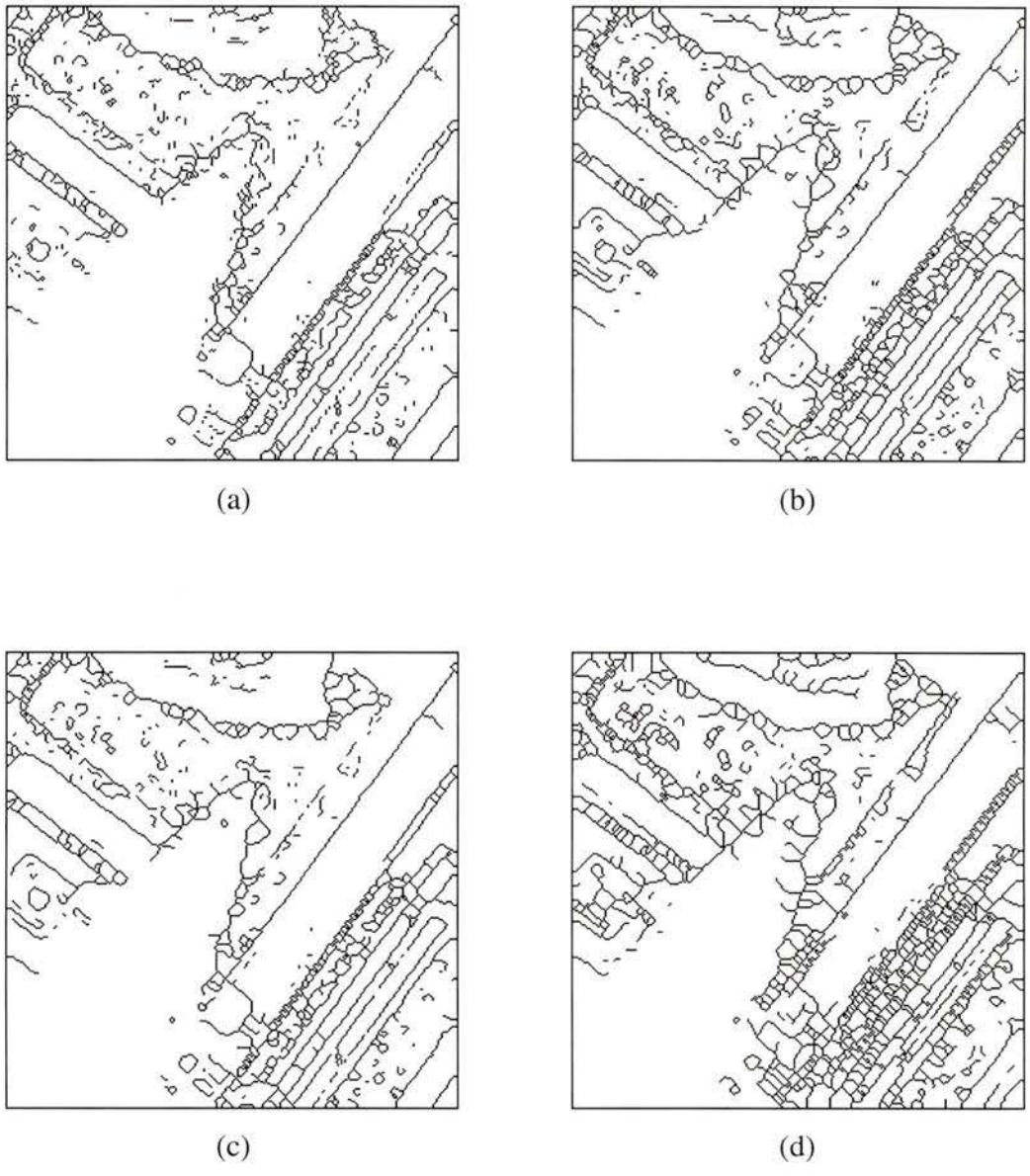


Figure 4.22 Thresholded and thinned MaG images.
(a) Simple operator. (b) Prewitt operator.
(c) Sobel operator. (d) Rosenfeld-Thurston operator.

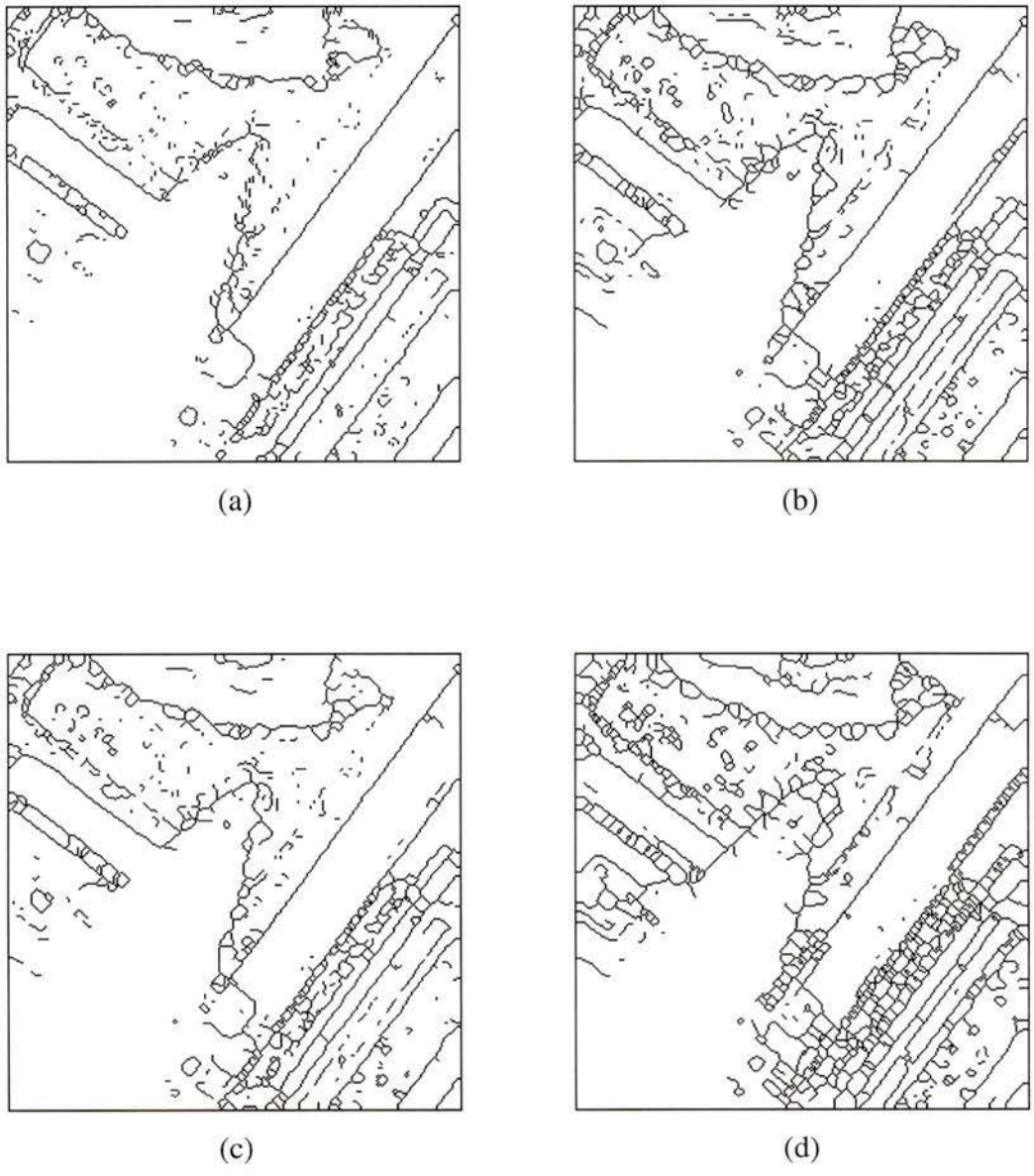


Figure 4.23 Thresholded and thinned MoA images.
(a) Simple operator. (b) Prewitt operator.
(c) Sobel operator. (d) Rosenfeld-Thurston operator.

4.5 Example 4: Laplacian of a Gaussian: Separable Implementation

One limitation of the first derivatives is their sensitivity to 2-D ramp edges where the edge's gradient is spread over many pixels. The first derivative operation returns the slope which must be thresholded to determine if the slope is large enough to be considered to be an edge. After thresholding, a ramp edge will produce a multiple pixel wide binary edge. Thinning is required to produce a single pixel wide edge whose position is dependent on the method for thinning of the binary edge, not the location of the edge.

The second derivative produces zero crossings that correspond to the edges. The location of the zero crossing of a ramp edge closely approximates its midpoint. In the calculation of the zero crossings, sub-pixel accuracy can be realized [62, 63].

The Laplacian of a Gaussian (LoG) operator is a popular rotationally-invariant second-derivative operator. The LoG operator combines a Gaussian smoothing filter and the Laplacian operation to perform the second derivative operation. It is considered to be a close parallel to biological vision systems [52, 53].

4.5.1 Laplacian of a Gaussian Operator

The description of the Laplacian of a Gaussian (LoG) operation uses continuous, 2-D functions. The LoG filtering operation is described in two steps which will be rearranged into a single, continuous filtering kernel that can be sampled for filtering digital images.

In the first step, a continuous image, $f(\tilde{x}, \tilde{y})$ is convolved with a 2-D Gaussian kernel, G_{σ}, defined in equation 4.32 [52].

$$G_{\sigma}(\tilde{x}, \tilde{y}) = \sigma^2 exp\left(-\frac{\tilde{x}^2 + \tilde{y}^2}{2\sigma^2}\right) \tag{4.32}$$

The second step is the Laplacian operation shown in equation 4.33 which is applied to the result of the Gaussian filtering.

$$\nabla^2(\bullet) = \frac{\partial^2}{\partial x^2}(\bullet) + \frac{\partial^2}{\partial y^2}(\bullet) \tag{4.33}$$

The Laplacian of a Gaussian filtering is shown in equation 4.34 where ** denotes 2-D convolution.

$$\nabla^2 [f(\tilde{x}, \tilde{y}) ** G_\sigma(\tilde{x}, \tilde{y})] =$$

$$\frac{\partial^2}{\partial \tilde{x}^2}(f(\tilde{x}, \tilde{y}) ** G_\sigma(\tilde{x}, \tilde{y})) + \frac{\partial^2}{\partial \tilde{y}^2}(f(\tilde{x}, \tilde{y}) ** G_\sigma(\tilde{x}, \tilde{y})) \tag{4.34}$$

The 2-D convolution and Laplacian operations are linear and can have their order of operations changed as in equation 4.35. The LoG operation can be considered as the convolution of an image with the Laplacian of a Gaussian which can be performed in one filtering step.

$$\begin{aligned}\nabla^2 [f(\tilde{x}, \tilde{y}) ** G_\sigma(\tilde{x}, \tilde{y})] &= f(\tilde{x}, \tilde{y}) ** \frac{\partial^2}{\partial \tilde{x}^2} G_\sigma(\tilde{x}, \tilde{y}) + f(\tilde{x}, \tilde{y}) ** \frac{\partial^2}{\partial \tilde{y}^2} G_\sigma(\tilde{x}, \tilde{y}) \\ &= f(\tilde{x}, \tilde{y}) ** \left(\nabla^2 G_\sigma(\tilde{x}, \tilde{y})\right)\end{aligned} \tag{4.35}$$

Equations 4.36 and 4.37 show the second partial derivatives of the Gaussian in the X and Y directions, respectively.

$$\frac{\partial^2}{\partial \tilde{x}^2} G_\sigma(\tilde{x}, \tilde{y}) = \left(\frac{\tilde{x}^2}{\sigma^2} - 1\right) exp\left(-\frac{\tilde{x}^2 + \tilde{y}^2}{2\sigma^2}\right) \tag{4.36}$$

$$\frac{\partial^2}{\partial \tilde{y}^2} G_\sigma(\tilde{x}, \tilde{y}) = \left(\frac{\tilde{y}^2}{\sigma^2} - 1\right) exp\left(-\frac{\tilde{x}^2 + \tilde{y}^2}{2\sigma^2}\right) \tag{4.37}$$

Equation 4.38 defines the Laplacian of a Gaussian kernel [52]. An arbitrary scale factor can be applied to the LoG kernel [59]. In some papers a negative scale factor has been used. The important information in LoG filtering is the zero crossings on which the scaling has no effected.

$$\begin{aligned}\nabla^2 G_\sigma(\tilde{x}, \tilde{y}) &= \frac{\partial^2}{\partial x^2} G_\sigma(\tilde{x}, \tilde{y}) + \frac{\partial^2}{\partial y^2} G_\sigma(\tilde{x}, \tilde{y}) \\ &= \left(\frac{\tilde{x}^2 + \tilde{y}^2}{\sigma^2} - 2\right) exp\left(-\frac{\tilde{x}^2 + \tilde{y}^2}{2\sigma^2}\right)\end{aligned} \tag{4.38}$$

A sampled version of the LoG operation is denoted by $\nabla^2 G_\sigma(x, y)$ where x and y are integer variables. This kernel could be used for FIR filtering except that it has an infi-

nite extent. The extent can be limited by truncating the kernel to a certain size because the significant extent of the kernel is bounded. 99% of the area under a Gaussian curve is within $\pm 3\sigma$ of the origin. The distance between zero crossings of the LoG kernel is $w = 2\sqrt{2}\sigma$ [52, 59]. As a general rule, the LoG operator dimensions should be 3 to 4 w to provide adequate trade-offs between computational requirements and maximum Gaussian support.

The LoG kernel has special properties that allows it to be realized as a modified separable filter [59, 62]. The LoG mask can be split into the summation of two separable filters of equation 4.39.

$$\begin{aligned} \nabla^2 G_\sigma(x, y) &= \left(\frac{x^2}{\sigma^2} - 1\right) exp\left(-\frac{x^2}{2\sigma^2}\right) \quad exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &\quad + \left(\frac{y^2}{\sigma^2} - 1\right) exp\left(-\frac{y^2}{2\sigma^2}\right) \quad exp\left(-\frac{x^2}{2\sigma^2}\right) \\ &= \mathrm{LoG}_x + \mathrm{LoG}_y \end{aligned} \tag{4.39}$$

The block diagram for the modified separable LoG filter implementation is shown in figure 4.24.

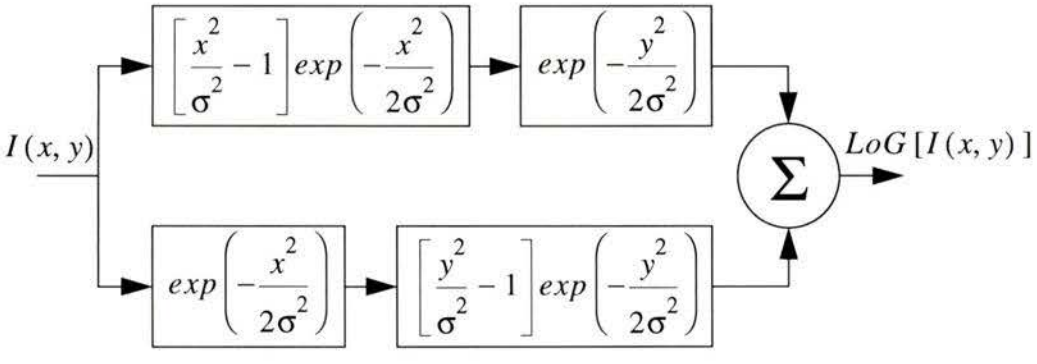


Figure 4.24 Signal flow graph for separable LoG filter.

4.5.2 Scale-Space Representation

Pyramidal or multi-scale image analysis involves the computation of a sequence of images of various spatial resolutions or scales [65, 59]. A fine scale shows considerable detail which in many cases is extraneous while a coarse scale causes some features to be lost or distorted. A scale-space representation can be found by convolving an input image with a filter mask which depends on a scale parameter. The LoG kernel has the constant, σ, which is used as a scale parameter. The scale-space representations from fine to coarse scale is computed by convolving the input image with a LoG kernel having increasing values for the space constant, σ. The finest possible scale is limited by the input image.

Consistency of response to features between scales is important. Features should not appear and disappear randomly in the scale-space sequence. The LoG kernel satisfies this condition. As the scale-space sequence varies from coarse to fine scale, zero crossings in a LoG kernel response never disappear but new ones can appear [59]. The zero crossings obtained from a LoG scale-space sequence can be used to identify major feature boundaries at a coarse scale and then obtain the most accurate position of the zero crossing at the fine scale.

4.5.3 Laplacian of a Gaussian Analysis

Figures 4.25-a and 4.25-b show the LoG_x and LoG_y coefficient masks for σ=1.061. The full FIR filter has the dimension 9x9 which would have resulted in 81 multiplications and 80 additions per pixel. The modified separable implementation resulted in 36 multiplications and 33 additions per pixel.

Figures 4.26-a and 4.26-b show the LoG_x and LoG_y coefficient masks for σ=2.121. The full FIR filter has the dimension 17x17 which would have resulted in 289 multiplications and 288 additions per pixel. The modified separable implementation resulted in 68 multiplications and 65 additions per pixel.

Figures 4.27-a and 4.27-b show the LoG_x and LoG_y coefficient masks for σ=3.182. The full FIR filter has the dimension 25x25 which would have resulted in 625 multiplications and 624 additions per pixel. The modified separable implementation resulted in 100 multiplications and 97 additions per pixel.

Figures 4.28 and 4.29 show the LoG_x and LoG_y coefficient masks for σ=4.243. The full FIR filter has the dimension 33x33 which would have resulted in 1089 multiplica-

```
DEDIP 2.0
Type : Separable FIR Filter Mask
Author : Al Keddy
Date : Tue Aug 14 15:27:40 1990
Comment : ( 0 line(s) ):
History : ( 0 line(s) ):
Format : ascii
Word Representation : Big Endian
Height : 9
Width : 9
Depth : 64
 0.009590360321027  0.113963908594821  0.383931556157861 -0.063326309518092
-0.888889169073428 -0.063326309518092  0.383931556157861  0.113963908594821
 0.009590360321027

 0.000815986006057  0.018315615795834  0.169013220696257  0.641180298605545
 1.000000000000000  0.641180298605545  0.169013220696257  0.018315615795834
 0.000815986006057
```

(a)

```
DEDIP 2.0
Type : Separable FIR Filter Mask
Author : Al Keddy
Date : Tue Aug 14 15:27:40 1990
Comment : ( 0 line(s) ):
History : ( 0 line(s) ):
Format : ascii
Word Representation : Big Endian
Height : 9
Width : 9
Depth : 64
 0.000815986006057  0.018315615795834  0.169013220696257  0.641180298605545
 1.000000000000000  0.641180298605545  0.169013220696257  0.018315615795834
 0.000815986006057

 0.009590360321027  0.113963908594821  0.383931556157861 -0.063326309518092
-0.888889169073428 -0.063326309518092  0.383931556157861  0.113963908594821
 0.009590360321027
```

(b)

Figure 4.25 LoG operator coefficients for σ=1.061.
(a) LoG_x separable operator. (b) LoG_y separable operator.

DEDIP 2.0
Type : Separable FIR Filter Mask
Author : Al Keddy
Date : Tue Aug 14 16:34:12 1990
Comment : (0 line(s)):
History : (0 line(s)):
Format : ascii
Word Representation : Big Endian
Height : 17
Width : 17
Depth : 64

0.002397590080257	0.009493849580040	0.028490977148705	0.062944124353538
0.095982889039465	0.081751038464049	-0.015831577379523	-0.154663615027680
-0.222222292268357	-0.154663615027680	-0.015831577379523	0.081751038464049
0.095982889039465	0.062944124353538	0.028490977148705	0.009493849580040
0.002397590080257			

0.000815986006057	0.004320232060007	0.018315615795834	0.062176469581825
0.169013220696257	0.367879325213062	0.641180298605545	0.894839285474353
1.000000000000000	0.894839285474353	0.641180298605545	0.367879325213062
0.169013220696257	0.062176469581825	0.018315615795834	0.004320232060007
0.000815986006057			

(a)

DEDIP 2.0
Type : Separable FIR Filter Mask
Author : Al Keddy
Date : Tue Aug 14 16:34:12 1990
Comment : (0 line(s)):
History : (0 line(s)):
Format : ascii
Word Representation : Big Endian
Height : 17
Width : 17
Depth : 64

0.000815986006057	0.004320232060007	0.018315615795834	0.062176469581825
0.169013220696257	0.367879325213062	0.641180298605545	0.894839285474353
1.000000000000000	0.894839285474353	0.641180298605545	0.367879325213062
0.169013220696257	0.062176469581825	0.018315615795834	0.004320232060007
0.000815986006057			

0.002397590080257	0.009493849580040	0.028490977148705	0.062944124353538
0.095982889039465	0.081751038464049	-0.015831577379523	-0.154663615027680
-0.222222292268357	-0.154663615027680	-0.015831577379523	0.081751038464049
0.095982889039465	0.062944124353538	0.028490977148705	0.009493849580040
0.002397590080257			

(b)

Figure 4.26 LoG operator coefficients for σ=2.121.
(a) LoG_x separable operator. (b) LoG_y separable operator.

```
DEDIP 2.0
Type : Separable FIR Filter Mask
Author : Al Keddy
Date : Tue Aug 14 16:40:26 1990
Comment : ( 0 line(s) ):
History : ( 0 line(s) ):
Format : ascii
Word Representation : Big Endian
Height : 25
Width : 25
Depth : 64
 0.001065595189179  0.002747889977804  0.006283248738253  0.012662654748506
 0.022284968543231  0.033728334651138  0.042659063756359  0.042218337618120
 0.026005909498858 -0.007036252688317 -0.049037632008330 -0.084722021717624
-0.098765470630655 -0.084722021717624 -0.049037632008330 -0.007036252688317
 0.026005909498858  0.042218337618120  0.042659063756359  0.033728334651138
 0.022284968543231  0.012662654748506  0.006283248738253  0.002747889977804
 0.001065595189179

 0.000815985571283  0.002540711870722  0.007166961229771  0.018315610306442
 0.042404742983545  0.088943492159671  0.169013198182865  0.290960318723642
 0.453788628636543  0.641180277253452  0.820754745047804  0.951816766085440
 1.000000000000000  0.951816766085440  0.820754745047804  0.641180277253452
 0.453788628636543  0.290960318723642  0.169013198182865  0.088943492159671
 0.042404742983545  0.018315610306442  0.007166961229771  0.002540711870722
 0.000815985571283
```

(a)

```
DEDIP 2.0
Type : Separable FIR Filter Mask
Author : Al Keddy
Date : Tue Aug 14 16:40:26 1990
Comment : ( 0 line(s) ):
History : ( 0 line(s) ):
Format : ascii
Word Representation : Big Endian
Height : 25
Width : 25
Depth : 64
 0.000815985571283  0.002540711870722  0.007166961229771  0.018315610306442
 0.042404742983545  0.088943492159671  0.169013198182865  0.290960318723642
 0.453788628636543  0.641180277253452  0.820754745047804  0.951816766085440
 1.000000000000000  0.951816766085440  0.820754745047804  0.641180277253452
 0.453788628636543  0.290960318723642  0.169013198182865  0.088943492159671
 0.042404742983545  0.018315610306442  0.007166961229771  0.002540711870722
 0.000815985571283

 0.001065595189179  0.002747889977804  0.006283248738253  0.012662654748506
 0.022284968543231  0.033728334651138  0.042659063756359  0.042218337618120
 0.026005909498858 -0.007036252688317 -0.049037632008330 -0.084722021717624
-0.098765470630655 -0.084722021717624 -0.049037632008330 -0.007036252688317
 0.026005909498858  0.042218337618120  0.042659063756359  0.033728334651138
 0.022284968543231  0.012662654748506  0.006283248738253  0.002747889977804
 0.001065595189179
```

(b)

Figure 4.27 LoG operator coefficients for σ=3.182.
(a) LoG_x separable operator. (b) LoG_y separable operator.

```
DEDIP 2.0
Type : Separable FIR Filter Mask
Author : Al Keddy
Date : Tue Aug 14 16:43:06 1990
Comment : ( 0 line(s) ):
History : ( 0 line(s) ):
Format : ascii
Word Representation : Big Endian
Height : 33
Width : 33
Depth : 64
 0.000599397520064  0.001233344079722  0.002373462395010  0.004262459511624
 0.007122744287176  0.011030115077639  0.015736031088385  0.020494293895517
 0.023995722259866  0.024529788618534  0.020437759616012  0.010788477752599
-0.003957894344881 -0.021633353380565 -0.038665903756920 -0.051031731084459
-0.055555573067089 -0.051031731084459 -0.038665903756920 -0.021633353380565
-0.003957894344881  0.010788477752599  0.020437759616012  0.024529788618534
 0.023995722259866  0.020494293895517  0.015736031088385  0.011030115077639
 0.007122744287176  0.004262459511624  0.002373462395010  0.001233344079722
 0.000599397520064

 0.000815986006057  0.001930450333145  0.004320232060007  0.009145933504955
 0.018315615795834  0.034696648886849  0.062176469581825  0.105399149810957
 0.169013220696257  0.256375646692885  0.367879325213062  0.499351679294092
 0.641180298605545  0.778800721700425  0.894839285474353  0.972604468600450
 1.000000000000000  0.972604468600450  0.894839285474353  0.778800721700425
 0.641180298605545  0.499351679294092  0.367879325213062  0.256375646692885
 0.169013220696257  0.105399149810957  0.062176469581825  0.034696648886849
 0.018315615795834  0.009145933504955  0.004320232060007  0.001930450333145
 0.000815986006057
```

Figure 4.28 LoG_x operator coefficients for σ=4.243.

```
DEDIP 2.0
Type : Separable FIR Filter Mask
Author : Al Keddy
Date : Tue Aug 14 16:43:06 1990
Comment : ( 0 line(s) ):
History : ( 0 line(s) ):
Format : ascii
Word Representation : Big Endian
Height : 33
Width : 33
Depth : 64
 0.000815986006057  0.001930450333145  0.004320232060007  0.009145933504955
 0.018315615795834  0.034696648886849  0.062176469581825  0.105399149810957
 0.169013220696257  0.256375646692885  0.367879325213062  0.499351679294092
 0.641180298605545  0.778800721700425  0.894839285474353  0.972604468600450
 1.000000000000000  0.972604468600450  0.894839285474353  0.778800721700425
 0.641180298605545  0.499351679294092  0.367879325213062  0.256375646692885
 0.169013220696257  0.105399149810957  0.062176469581825  0.034696648886849
 0.018315615795834  0.009145933504955  0.004320232060007  0.001930450333145
 0.000815986006057

 0.000599397520064  0.001233344079722  0.002373462395010  0.004262459511624
 0.007122744287176  0.011030115077639  0.015736031088385  0.020494293895517
 0.023995722259866  0.024529788618534  0.020437759616012  0.010788477752599
-0.003957894344881 -0.021633353380565 -0.038665903756920 -0.051031731084459
-0.055555573067089 -0.051031731084459 -0.038665903756920 -0.021633353380565
-0.003957894344881  0.010788477752599  0.020437759616012  0.024529788618534
 0.023995722259866  0.020494293895517  0.015736031088385  0.011030115077639
 0.007122744287176  0.004262459511624  0.002373462395010  0.001233344079722
 0.000599397520064
```

Figure 4.29 LoG_y operator coefficients for σ=4.243.

tions and 1088 additions per pixel. The modified separable implementation resulted in 132 multiplications and 129 additions per pixel.

The impulse responses of the four LoG filter masks with the Gaussian space constants of σ=1.061, 2.121, 3.182 and 4.243 are shown in figure 4.30. The impulse responses show the shape of the Laplacian of a Gaussian function. As the scale changes from fine to coarse the extent of the LoG kernel impulse response increases.

The magnitude responses of the four LoG filter masks with the Gaussian parameter σ=1.061, 2.121, 3.182 and 4.243 are shown in figure 4.31. There are some interesting points to note from the magnitude responses. The LoG filter is a circularly-symmetric bandpass filter with a response independent of the orientation of the edge: it is a rotationally-invariant derivative operator. Comparing the differences as the scale-space resolution decreases (the Gaussian space constant increases) it is observed that the fine scale of σ=1.061 allows more high frequency components to pass through the filter than the coarse scale of σ=4.243. This increased attenuation of the high frequency components will decrease sharp intensity changes and blur the detail for a coarse view of an image's features.

Comparing the general shape of the scale-space representation for the impulse and magnitude responses an inverse relationship with respect to the width is observed. At the fine scale the impulse response has a small width and the magnitude response has a wide width. The correlation operation uses the information from a few pixels and keeps considerable feature detail. The magnitude response shows that the filter allows considerable high frequency components to pass which leads sharp intensity changes. These sharp intensity changes form the feature details.

At the coarse scale the impulse response is wide and the magnitude response is narrow. The correlation operation uses the information from many pixels and reduces feature detail considerably. The magnitude response shows the filtering passes very little high frequency components with considerable blurring. The feature detail is reduced and small-scale discontinuities disappear.

The step responses of the four LoG filter masks with the Gaussian space constants of σ=1.061, 2.121, 3.182 and 4.243 are shown in figure 4.32. As was discussed with the first directional-derivative operators, the delay observed in the step response is reflected in wide, blurred edges. The fine scale step response has a very small delay and has a very

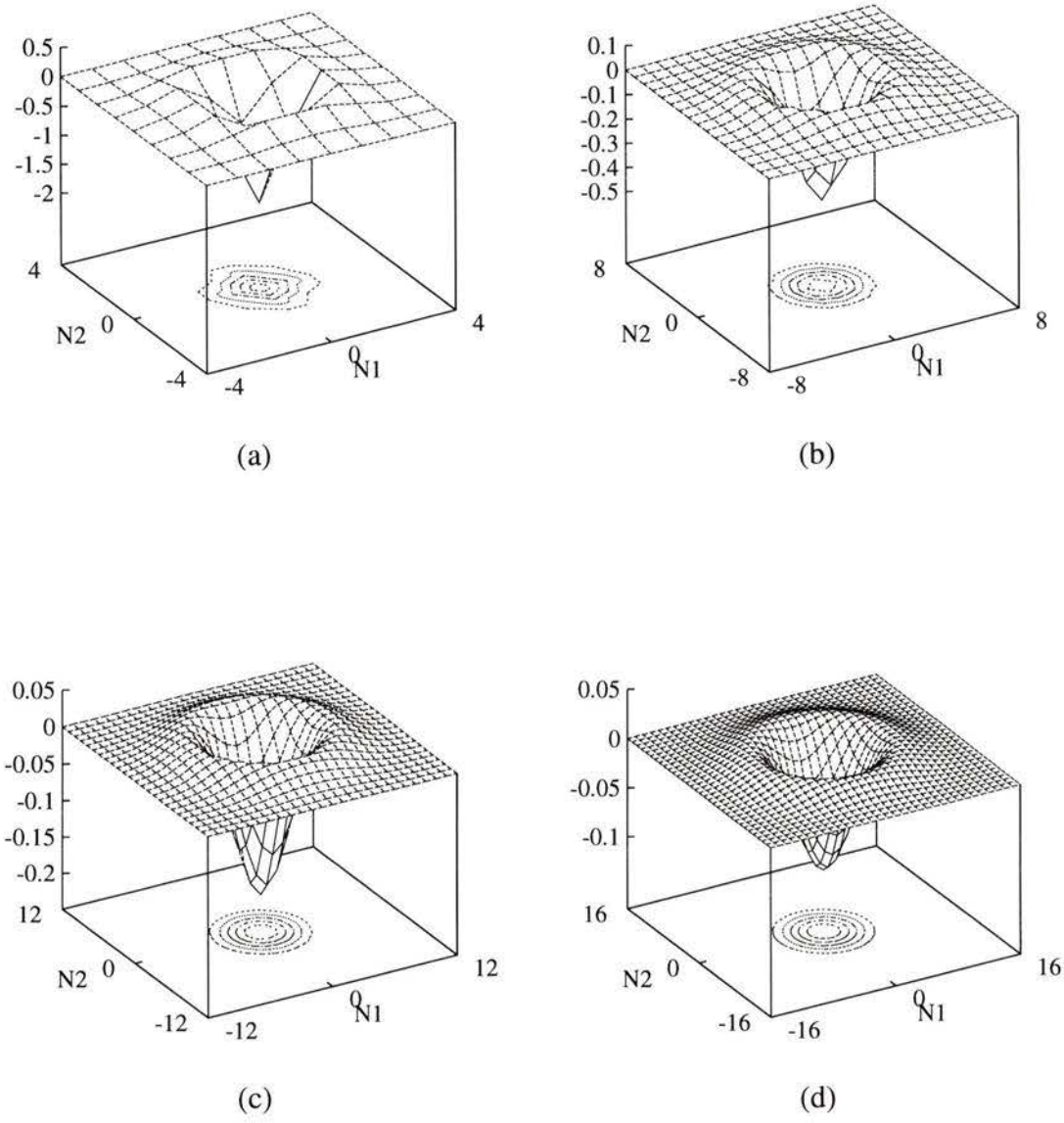


Figure 4.30 Laplacian of a Gaussian operator: impulse response.
(a) σ=1.061. (b) σ=2.121. (c) σ=3.182. (d) σ=4.243.

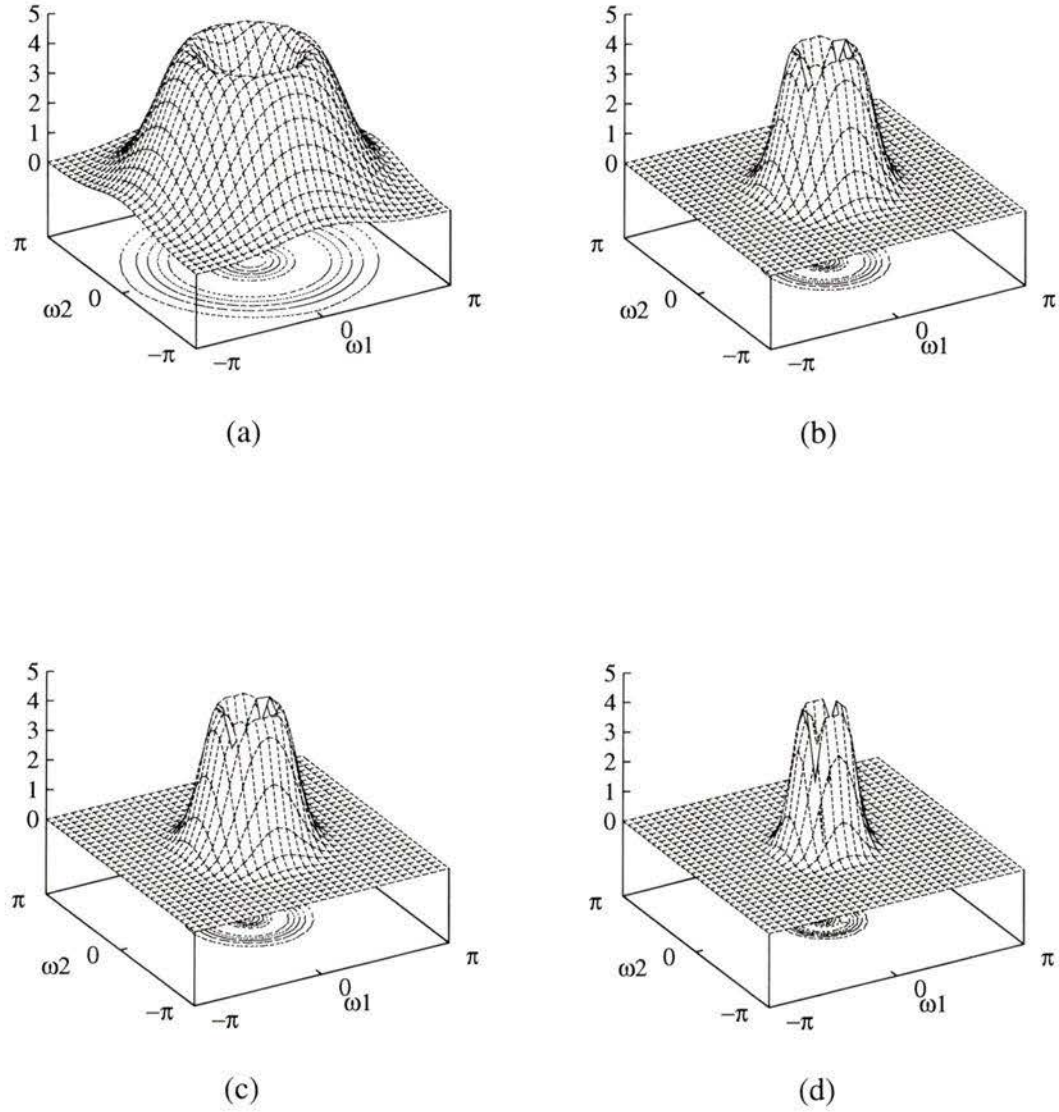


Figure 4.31 Laplacian of a Gaussian operator: magnitude response.
(a) σ=1.061. (b) σ=2.121. (c) σ=3.182. (d) σ=4.243.

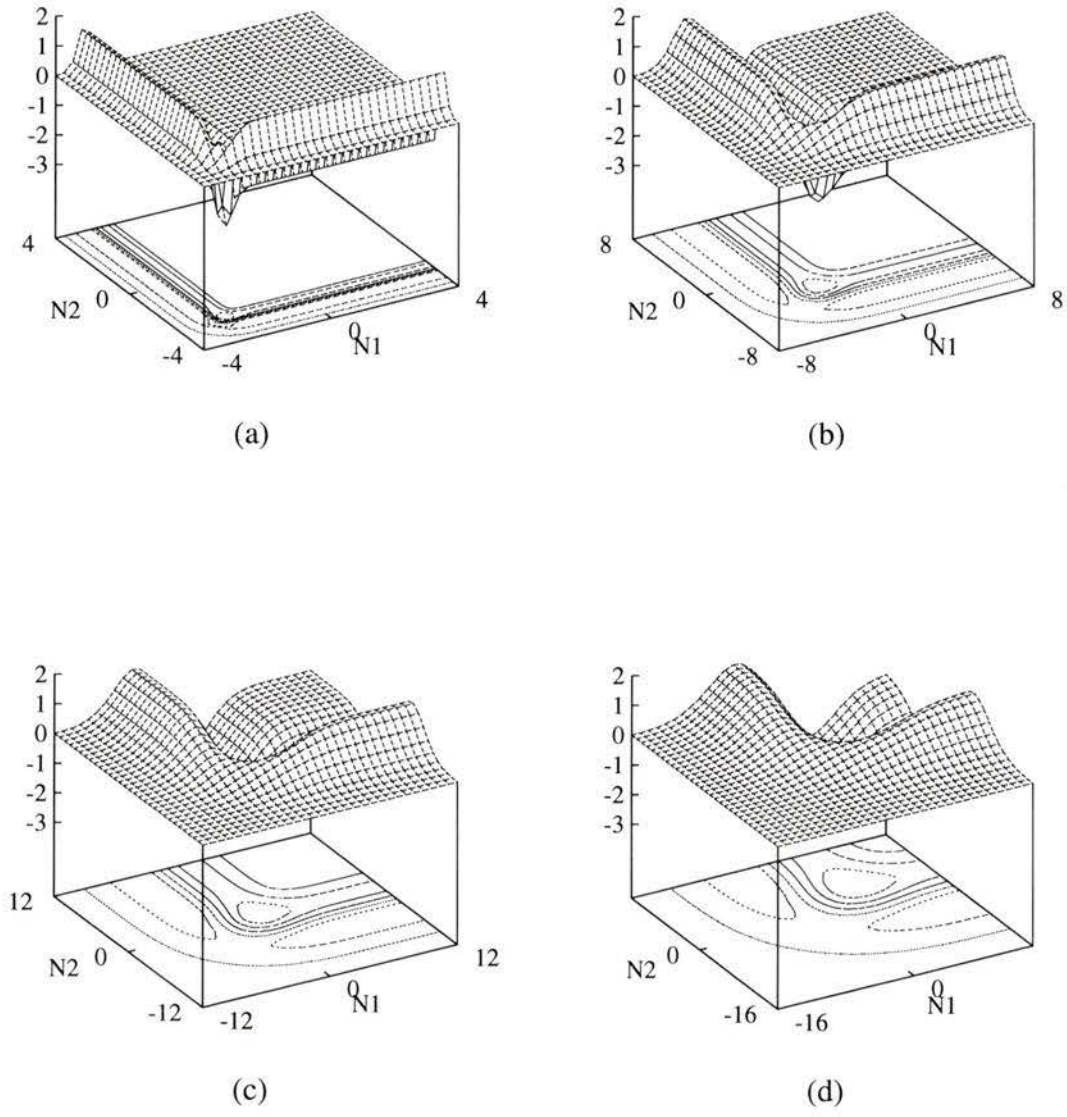
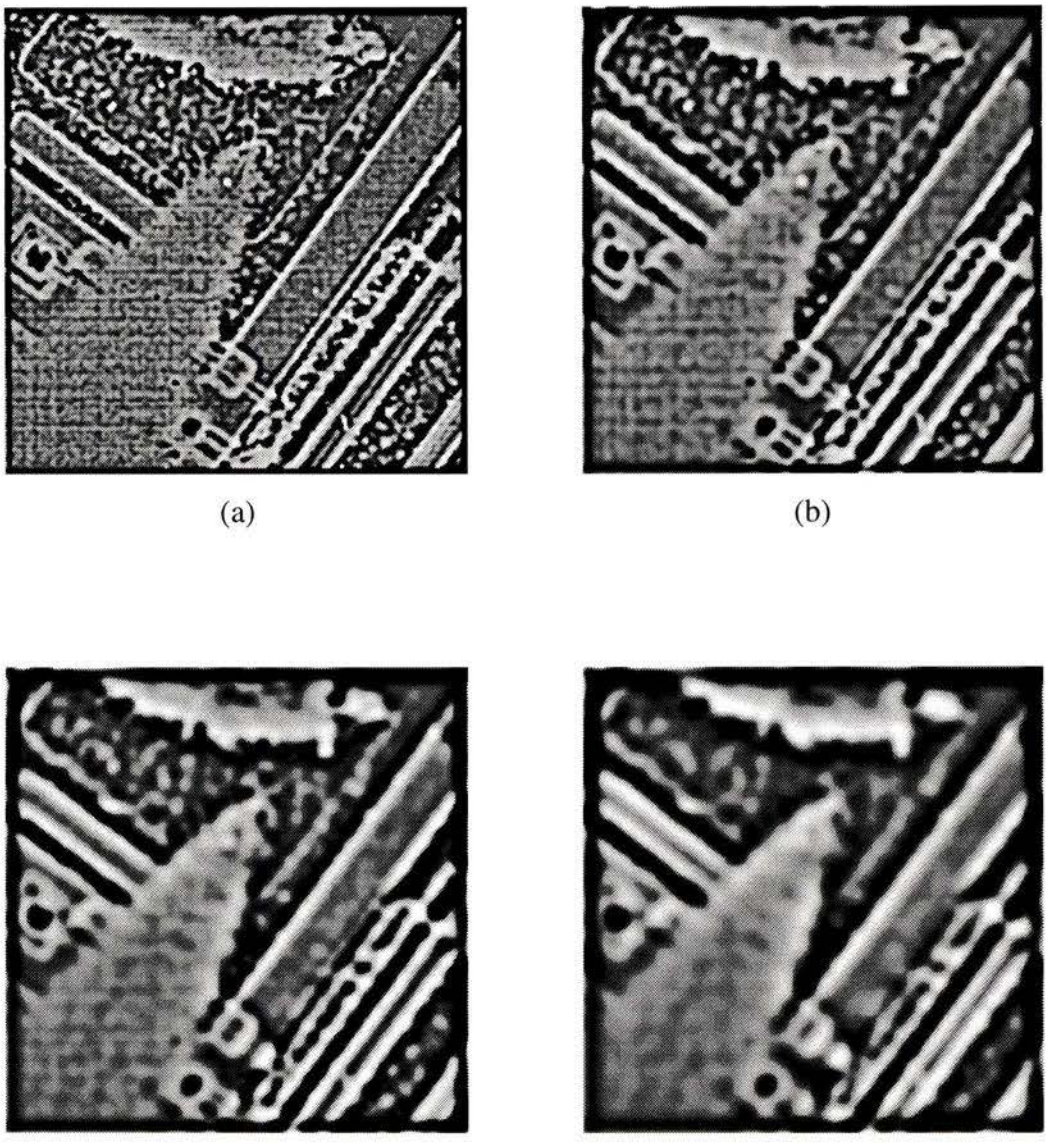


Figure 4.32 Laplacian of a Gaussian operator: step response.
(a) σ=1.061. (b) σ=2.121. (c) σ=3.182. (d) σ=4.243.

sharp response to edges. The coarse scale step response is large and has very wide, blurred edges.

4.5.4 Laplacian of a Gaussian Application

Figure 4.33 shows the results of filtering the test image of figure 4.4 with the four LoG filters with the Gaussian space constants of σ=1.061, 2.121, 3.182 and 4.243. The blurring of the features as the space constant increases expected from the analysis can be observed. Figure 4.34 shows the zero crossings of figure 4.33. The increase in feature detail of scale-space sequence from the coarse scale resolution of figure 4.34-d to the fine scale resolution of figure 4.34-a is clearly observable.



(a) (b)

(c) (d)

Figure 4.33 Laplacian of a Gaussian operator: filter result.
(a) σ=1.061. (b) σ=2.121. (c) σ=3.182. (d) σ=4.243.

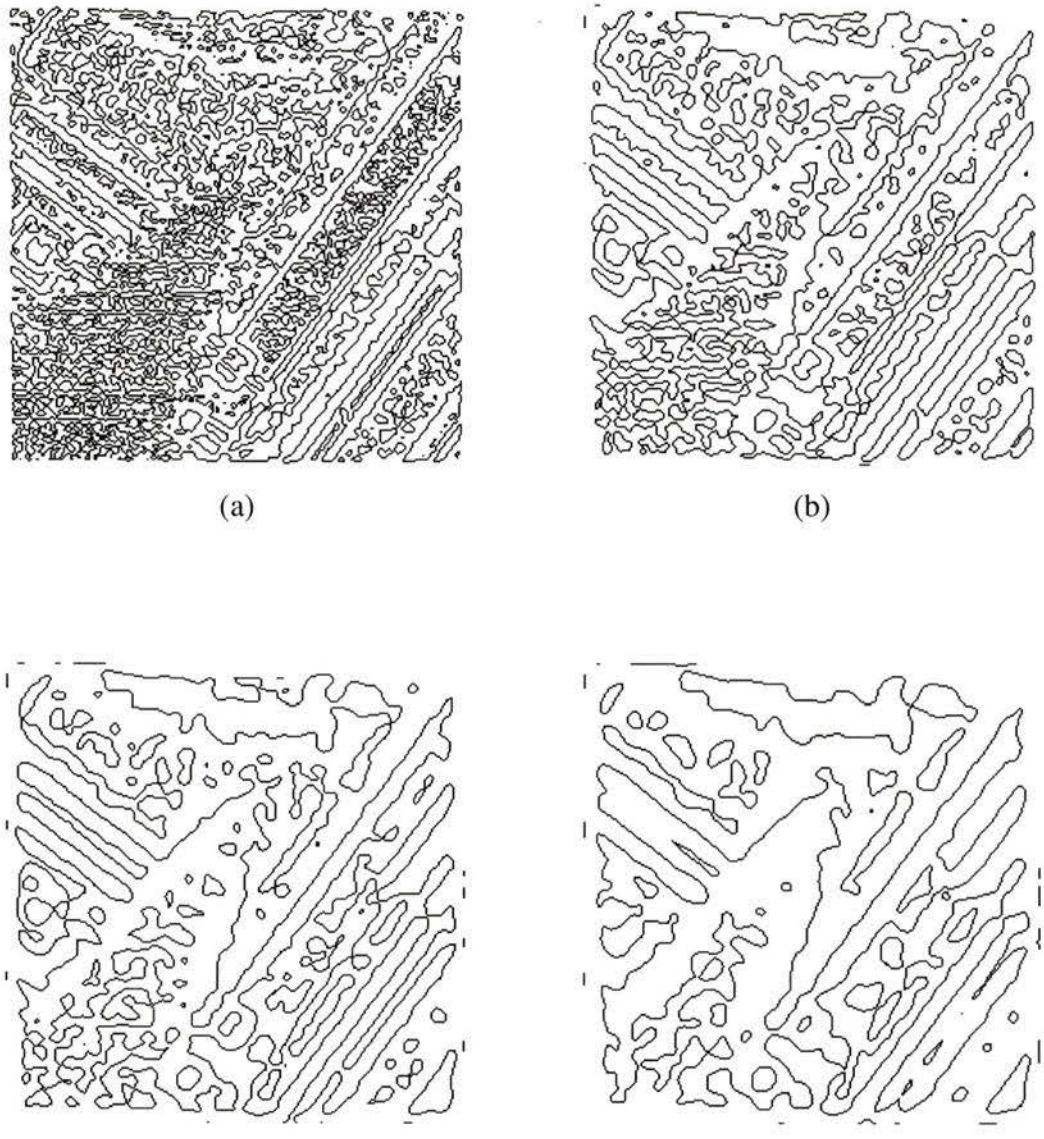


Figure 4.34 Laplacian of a Gaussian operator: zero-crossings.
(a) σ=1.061. (b) σ=2.121. (c) σ=3.182. (d) σ=4.243.

Chapter 5

Summary

5.1 Summary

A software package DEDIP has been developed to provide an efficient environment for DIP algorithm development. A particular focus of DEDIP is the implementation of 2-D, LSI filters for DIP applications. Although many general image-manipulation environments are available, none of the ones surveyed provide a general application of 2-D LSI filtering beyond simple convolution.

5.1.1 DEDIP

DEDIP is a two-level programming environment which is composed of a function library and programs. A DIP researcher can combine existing programs to process digital image data files from the CLI or create new programs using the function libraries. The filtering examples demonstrated the CLI level programs.

The library and programs were developed using structured programming techniques. The modularity and information hiding simplified development to small, easy-to-understand modules. The encapsulation of the data structures and functions made the programming of algorithms much easier. The focus was on the development of the algorithms, not the auxiliary functions such as the IO functions. Though the library level was not demonstrated here, the programs that implemented the MaG, MoA, thinning and zero-crossing algorithms for the examples were developed using the library level.

The system presented in this thesis provides the ability to analyze and apply 2-D linear, shift invariant digital filter coefficient masks. The filtering aspects of DEDIP were demonstrated using FIR, IIR and SFIR filter implementations.

The first example showed the ability to analyze and apply lowpass filters implemented as FIR and IIR filters. The two implementations' performance was discussed. DEDIP does not consider the problem of designing 2-D digital filters but the body of knowledge for designing 2-D digital filters is richer for FIR filter design.

The second example demonstrated the fan filter. It is a frequency domain specification of a filter that highlights edges with a particular orientation.

The third and fourth examples described and demonstrated edge-detection filtering designed and specified in the spatial domain.

The third example described the historical development of common directional first spatial derivative cross-correlation masks. This example also showed the result of different stages of nonlinear filtering used to produce edge maps.

The fourth example discussed the implementation of the rotationally-invariant second spatial derivative operator, Laplacian of a Gaussian (LoG). It is implemented as a modified separable FIR filter. This implementation improved the computational efficiency of the LoG processing. To complete the edge map processing, the results of nonlinear filtering to obtain an edge map from the zero crossings in the LoG filtering results is shown.

The examples demonstrated many properties of 2-D linear, shift invariant (LSI) digital filters. DEDIP contributes the method and mechanisms for a general interface of programs and C functions that can analyze and apply 2-D LSI digital coefficient masks to images, as demonstrated in the examples.

5.1.2 Other Research Projects

DEDIP has already been used in several digital image processing research projects and has proved to be useful for algorithm development. The following briefly describes these projects and the contribution of DEDIP to each project.

Mr. A. El Maleh used DEDIP in research of 1-D vector quantization image coding techniques [66]. Mr. El Maleh wrote programs using the DEDIP libraries for the evaluation of image compression algorithms using 2-D and 1-D vector quantization. His research also investigated different modifications to the 1-D vector quantization to evaluate the perceived image quality improvements.

Another area of research that used DEDIP was in reconstruction of 3-dimensional

representations from 2-dimensional slices [67]. In the course of her research, Ms. J. Li used the LoG filtering demonstrated in example four of chapter 4 to obtain edge contours that represented feature boundaries in the slice. The feature boundaries from multiple slices were then converted to a 3-D triangular objects that were rendered to visualize the original object.

Motion estimation research using block matching algorithms is being performed using DEDIP by Mr. J. Chana [68]. Mr. Chana has used DEDIP to implement and compare the performance of a number of efficient block matching algorithms. Programs written using DEDIP are also being used to manipulate frames in a raw video sequences and the DXDisplay program is used to evaluate the subjective quality of predicted frames.

5.2 Future Directions

DEDIP's future development goes in two different directions. The first direction is the further development of DIP tools through the implementation of new algorithms such as 2-D filter design. The second one is improving the software implementation by using new features of recently-developed compiler languages.

The DEDIP system was developed using the Kernigan and Richie (K&R) version of the C language [41]. A problem with K&R C is weak function parameter type checking and no type checking on macros. ANSI C [69] and C++ [70] provide strong function parameter type checking through function prototypes. C++ also provides inline functions which allow fast execution without function call overhead as do macros but also provides function parameter type checking.

Future development of DEDIP using C++ is under consideration. In addition to the type-checking advantages, C++ classes provide an interesting method of encapsulating functions and data as well as controlling access to different parts of a class. C++ can also be used to overload existing operators such as *, /, + and -. A common example is a user defined complex data type including the basic arithmetic operators.

Bibliography

[1] J.L.C. Sanz. Special issue on industrial machine vision and computer vision technology - part I. *IEEE Transactions on Pattern Analysis and Machine Intellegence*, vol. PAMI-10, Jan. 1988.

[2] J.L.C. Sanz. Special issue on industrial machine vision and computer vision technology - part II. *IEEE Transactions on Pattern Analysis and Machine Intellegence*, vol. PAMI-10, May. 1988.

[3] J.J. Le Moigne. Domain-dependent reasoning for visual navigation of roadways. *IEEE Transactions on Robotics and Automation*, vol. RA-4, no. 4, pp. 419–427, Aug. 1988.

[4] C. Nimrod. Ultrasound and high risk pregnancies. *Dimensions in health service*, vol. 68, no. 2, pp. 13–15, Mar. 1991.

[5] D.M. Hynes. Digital videofluorography: a new direction in diagnostic imaging. *Dimensions in health service*, vol. 68, no. 2, pp. 227-31, Mar. 1991.

[6] L. Reese. Magnetic resonance imaging in Canada. *Dimensions in health service*, vol. 67, no. 2, pp. 10–14, Mar. 1990.

[7] D. Vellet. Magnetic resonance imaging: in need of assessment.. *Dimensions in health service*, vol. 68, no. 3, pp. 24-26, Mar. 1991.

[8] S.R. Sternberg. Biomedical image processing. *Computer*, vol. 16, no. 1, pp. 36–47, Jan. 1983.

[9] M.S. Landy, Y. Cohen, and G. Sperling. HIPS: A UNIX-based image processing system. *Computer Vision, Graphics and Image Processing*, vol. 25, pp. 331–347, 1984.

[10] M.S. Landy, Y. Cohen, and G. Sperling. HIPS: image processing under UNIX: software and applications. *Behavior Research Methods, Instruments and Computers*, vol. 16, no. 2, pp. 199–216, 1984.

[11] N. Denyer, J. Princz, T. Feehan, and P. Ketchum. ERS-1 points the way for RADARSAT. *GEOS*, vol. 19, no. 2, pp. 23–29, Spring 1990.

[12] M. Manore. Remote sensing and GIS - together at last! *GEOS*, vol. 19, no. 2, pp. 17–22, Spring 1990.

[13] R.J. Brown, W.G. Best, and G.K. Walker. Satellites monitor global vegetation condition. *GEOS*, vol. 19, no. 2, pp. 12–16, Spring 1990.

[14] R. Ryerson and J. Cihlar. Remote sensing techniques monitor global change. *GEOS*, vol. 19, no. 2, pp. 1–6, Spring 1990.

[15] G.P. Watson, A.N. Rencz, and G.F. Bonham-Carter. Computers assist prospecting. *GEOS*, vol. 18, no. 1, pp. 8–15, Winter 1989.

[16] Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134. *Cadence 4.2 Design Software and Online Manuals*. 1992.

[17] Cadence Design Systems, Inc. *Cadence Edge manuals*, 1988,1989.

[18] Cadence Design Systems, Inc. *Verilog Logic Analysis Family*. 1991.

[19] Mentor Graphics Corp., 8005 S.W. Broeckman Rd., Willsonville, Or. 97070. *Mentor Graphics Software Manuals*. 1993.

[20] P.J. Ashenden. *The VHDL Cookbook*. Dept. of Computer Science, University of Adelaide, South Australia, 1990.

[21] P.J. Brumfit. Environments for image processing algorithm development. *Image and Vision Computing*, vol. 2, no. 4, pp. 198–203, 1984.

[22] M.J.B. Duff and S. Levaldi. *Languages and Architectures for Image Processing*. Academic Press, New York, 1981.

[23] W. Frei. Digital image processing software. *Proceedings of the Society of Photo-optical Instrumentation Engineers*, vol. 528, pp. 88–94, 1985.

[24] Graftek, 9, route de Verriere, 92360, Meudon-La-Foret, FRANCE. *Ultimage User Manual*, 1988. In Canada and United States: GTFS Inc., 2455 Bennett Valley Rd. #100C, Santa Rosa, CA 95404.

[25] W. Rasband. *Image Version 1.30 Manual*. National Institutes of Health, Aug. 1990.

[26] R. Morris. Image processing on the macintosh. *Computer*, vol. 23, no. 8, pp. 103–107, Aug. 1990.

[27] D.G. Bailey and R.M. Hodgson. VIPS - a digital image processing algorithm develpment environment. *Image and Vision Computing*, vol. 6, no. 3, pp. 176–184, 1988.

[28] P. Agathoklis and W. A. Keddy. DEDIP: a developmental environment for digital image processing. *COMCON88 Advances in communications and control systems*, pp. 199–216, Oct 19-21 1988.

[29] W. A. Keddy and P. Agathoklis. DEDIP: a developmental environment for digital image processing. Technical Report Report VMC-1/1991, Victoria Micronet Center, 1991.

[30] W. A. Keddy and P. Agathoklis. DEDIP: a user-friendly environment for digital image processing algorithm development. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 733–736, 1991.

[31] J.L. Mannos. Powerful hardware/software architecture for a minicomputer-based interactive image processing system. *Proceedings of the Society of Photo-optical Instrumentation Engineers*, vol. 301, pp. 129–134, 1981.

[32] J. Piper and D. Rutovitz. Data structures for image processing in a C language and UNIX environment. *Pattern Recognition Letters*, vol. 3, no. 2, pp. 119–129, 1985.

[33] J. Poskanzer. *Extended portable bitmap toolkit*. Carnegie Mellon University, Dec. 1991.

[34] M.L. Mauldin. *The fuzzy pixmap package*. Carnegie Mellon University, Jun. 1990.

[35] *University of Minnesota's Internet gopher distribution for UNIX machines: gopher-1.12*. University of Minnesota, 1993.

[36] The MathWorks, Inc., 24 Prime Park Way, Natick, Ma. 01760. *Matlab User's Guide*. Aug. 1992.

[37] D. Bell, I. Morrey and J. Pugh. *Software engineering, a programming approach*. Prentice-Hall International (UK) Ltd., London, 1986.

[38] A. Von Mayrhauser. *Software engineering: methods and management*. Academic Press, Inc., San Deigo, 1990.

[39] A.P. Sage and J.D. Palmer. *Software systems engineering*. John Wiley and Sons, Inc., New York, 1990.

[40] W.-S. Lu and A. Antoniou. *Two-dimensional digital filters*. Marcel Dekker, Inc., New York, 1992.

[41] B.W. Kernighan and D.M. Ritchie. *The C programming language*. Prentice-Hall Inc., Englewood, NJ., 1978.

[42] R.N. Horspool. *C Programming in the Berkeley UNIX Environment*. Prentice-Hall Canada Inc., Scarborough, On., 1986.

[43] M.R.M. Dunsmuir and G.J. Davies. *Programming the UNIX System*. MacMillan Publishers, London, 1985.

[44] R.B. Coates and I. Vlaeminke. *Man-computer interfaces*. Blackwell Scientific Publications, Oxford, 1987.

[45] Apple Computers, Inc. *Macintosh human interface guidelines*. Addison-Wesley Publishing Co., Reading, Ma., 1992.

[46] D. Shpak. *Plot3*. University of Victoria. Feb.1988.

[47] A. Antoniou. *Digital filters: analysis and design*. McGraw-Hill, Inc., New York, 1979.

[48] D.E. Dudgeon and R.M. Mersereau. *Multidimensional digital signal processing*. Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1984.

[49] J.S. Lim. *Two-dimensional signal and image processing*. Prentice-Hall Inc., Englewood, NJ., 1990.

[50] R.C. Gonzalez and P.A. Wintz. *Digital Image Processing*. Addison-Wesley Publishing Company, Inc., Reading, Ma., 1987.

[51] I.Pitas and A.N. Venetsanopoulos. A new filter structure for the implementation of certain classes of image processing operations. *IEEE Transactions on Circuits and Systems*, vol. CAS-35, Jun. 1988.

[52] E.C. Hildreth. The detection of intensity changes by computer and biological vision systems. *Computer Vision, Graphics and Image Processing*, vol. 22, no. 1, pp. 1-27, Apr. 1983.

[53] R.M. Haralick. Digital step edges from zero crossing of the second directional derivative. *IEEE Transactions on Pattern Analysis and Machine Intellegence*, vol. PAMI-6, no. 1, pp. 58-68, Jan. 1984.

[54] S. Haykin. *Communications Systems*. John Wiley and Sons, Inc., New York, 1978.

[55] K. Preston, jr. The abingdon cross benchmark survey. *Computer*, vol. 22, no. 7, pp. 9-18, Jul. 1989.

[56] K. Preston, jr. Benchmark results:the abingdon cross. *Evaluation of Multicomputers for Image Processing*. L. Uhr, K, Preston, jr., S. Levialdi and M.J.B. Duff, Ed. Academic Press, Inc., Orlando, Fl., 1986.

[57] T. Williams et al. *Gnuplot: An Interactive Plotting Program, version 3.5*. 1988-1993.

[58] Aladdin Enterprises. *Ghostview, version 2.6.1*. 1989, 1992, 1993.

[59] R.J. Schalkoff. *Digital Image Processing and Computer Vision*. John Wiley and Sons, Inc., New York, 1989.

[60] B. Chanda, B.B. Chaudhuri and D.D. Majumder. A differentiation / enhancement edge detector and its properties. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, no. 1, pp. 162-168, Jan./Feb. 1985.

[61] N.J. Naccache and R. Shinghal. An investigation into the skeletonization approach of Hilditch. *Pattern Recognition*. vol. 12, no. 4, pp. 279-284, 1984.

[62] S.M. Ali and R.E. Burge. A new algorithm for extracting the interior of bounded regions based on chain coding. *Computer Vision, Graphics and Image Processing*, vol. 43, pp. 256-264, 1988.

[63] I. Overington and P. Greenway. Practical first-difference edge detection with subpixel accuracy. *Image and Vision Computing*, vol. 5, no. 3, pp. 217-224, Aug. 1987.

[64] A. Heurtas and G. Medioni. Detection of intensity changes with subpixel accuracy using Laplacian-Gaussian masks. *IEEE Transactions on Pattern Analysis and Machine Intellegence*, vol. PAMI-8, no. 5, pp. 651-664, May 1986.

[65] Y. Lu and R.C. Jain. Behaviour of Edges in Scale Space. *IEEE Transactions on Pattern Analysis and Machine Intellegence*, vol. PAMI-11, no. 4, pp. 337-356, Apr. 1989.

[66] A.H. El-Maleh. *Image Compression using One-Dimensional Vector Quantization*. University of Victoria, 1991.

[67] J.-P. Li. *Personal conversation*. University of Victoria, Jul. 1994.

[68] J. Chana. *Personal conversation*. University of Victoria, Jul. 1994.

[69] N. Barkakati. *Object-Oriented Programming in C++*. SAMS, Carmel, IN., 1991.

[70] J.O. Coplien. Advanced C++ programming styles and idioms. Addison-Wesley Publishing Company, Inc., Reading, Ma., 1992

VITA

Surname: Keddy Given Names: William Alfred Lyle

Place of Birth: Victoria, B.C. Date of Birth: December 2, 1954

Educational Institutions Attended:

University of Victoria 1983-1994

Degrees Awarded:

Bachelor of Engineering, University of Victoria, 1988.

Honours and Awards:

B.C. Post-Secondary Scholarship	1988
NSERC Summer Research Grant	1986
NSERC Summer Research Grant	1985
IBEW Founders' Scholarship	1985-1988

Publications:

1. W. A. Keddy and P. Agathoklis. DEDIP: a user-friendly environment for digital image processing algorithm development. *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 733–736, 1991.
2. W. A. Keddy and P. Agathoklis. DEDIP: a developmental environment for digital image processing. Technical Report VMC-1/1991, Victoria Micronet Center, 1991.
3. N.J. Dimopoulos, D. Radvan and W.A. Keddy, Learning in asymptotically behaving neural networks, *International Joint Conference on Neural Networks*, San Diego, CA, pp III-233- III-238, 1990.
4. P. Agathoklis and W. A. Keddy. DEDIP: a developmental environment for digital image processing. *COMCON88 Advances in communications and control systems*, pp. 199–216, Oct 19-21 1988.

Partial Copyright License

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis: **2-D Digital Filter Analysis and Application with DEDIP**

Author [redacted]

(Signature)

William Alfred Lyle Keddy

(Name in Block Letters)

Aug. 31, 1994

(Date)