

Applications of Smalltalk/V to Digital Image Processing



A thesis submitted in fulfilment of the requirements for the award of the degree of

MASTER OF SCIENCE

from

Victoria University of Technology

by

Fei Liu, BE

Department of Computer and Mathematical Sciences

February 1994

FTS THESIS
006.42 LIU
30001004589695

Liu, Fei
Applications of Smalltalk/V
to digital image processing

Abstract

The author has demonstrated Smalltalk as a medium for explorations in image processing by creating within it an expandable environment for image processing. He has produced two versions: one for DOS, and one for Windows which is hereafter referred to as *ImageLab*. The Windows version, ImageLab, may be used in two ways: as a stand alone by a non-programmer for whom the existing functionality is adequate; or within the Smalltalk environment by a programmer who might wish to expand the functionality.

In creating ImageLab the author recognised that:

- the BitBlt operation was eminently suitable for implementing basic morphological operations;
- and that Smalltalk was eminently suitable for implementing other morphological operations as algebraic expressions in three basic operations (using Huang's BIA).

He has given examples to show ImageLab 'in action'. In particular, he has applied his environment to making a contribution to the detection and counting of clusters of points (relevant to the detection and counting of clusters of microcalcifications revealed in mammograms of patients with early signs of breast cancer).

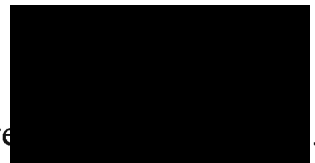
He has also demonstrated the suitability of Smalltalk for the use of quad-tree techniques in image processing. In particular he has indicated how a recently published adaptive technique can be applied to quad-trees.

Declaration

The candidate hereby declare that the work in this thesis, presented for the award of the Master of Applied Science and submitted in the Department of Computer and Mathematical Sciences, Victoria University of Technology:-

- is that of the candidate alone and has not been submitted previously, in whole or in part, in respect of any other academic award and has not been published in any form by other person except where due reference is given, and
- has been carried out during the period from January 1992 to February 1994 under the supervision of Dr. Don Watson and Mr. Tom Peachey.

Signature

A solid black rectangular box redacting the signature of the candidate.

Fei Liu

Acknowledgments

The author of this thesis wishes to acknowledge the following organisations or people for directly or indirectly providing assistance and guidance while carrying out this research and writing this thesis.

Dr. Don Watson and *Mr. Tom Peachey*, the academic supervisors, for their patient, encouraging supervision, invaluable guidance and assistance, enlightening advice and suggestions, and constructive criticisms during the research and preparation of the thesis.

Mr. Robert Hinterting, for Smalltalk and debugging of my programming. Thanks are extended to *Associate-Professor Charles Osborne*, *Mr. Alasdair McAndrew*, *Mr. Martin Schweitzer*, *Dr. Hao Shi* for their constructive suggestions, and the people who discussed the topic with me through internet.

Mr. P. Rajendran and Mr. Damon Burgess, technical officers, for their significant help in providing machines, manuals and other facilities.

The Australian International Development Assistance Bureau, for providing the scholarship covering tuition fees and health insurance cover.

Since the English is the author's second language, a lot of people helped him when he wrote the thesis. The author would like to give his thanks to *Dr. Don Watson*, *Mr. Tom Peachey*, Mrs. Betty Watson and Mr. Mehmet Tat.

Finally, the author would like to express his special gratitude to his wife Cao-wei Xie and his lovely daughter Xing-Xing for their continuous support, encouragement, understanding, patience and love. Grateful acknowledgment is extended to the author's father Yu-cai Liu and mother Jian-xi Luo for their encouragement and understanding.

Table of contents

Abstract		i
Declaration		ii
Acknowledgments		iii
Table of contents		iv
List of figures		viii
List of tables		xi
Chapter One	Introduction	1
1.1	The problem	1
1.2	The scope of the thesis	2
Chapter Two	Image-processing languages - a brief review	5
2.1	The computer and digital image processing	5
2.2	The computer languages used in image processing	7
Chapter Three	Object-oriented techniques and Smalltalk	9
3.1	The history of object-oriented technology	10
3.2	The history of object-oriented programming languages	10
3.3	Constituents of an OOL	11
3.3.1	Programming with objects	11
3.3.2	Computation by message-passing	11
3.3.3	Abstraction to classes and subclasses	12
3.3.4	Programming in the presence of inheritance	12
3.4	The benefits of introducing OOL to image processing	13
3.4.1	Modelling an image as an object	14
3.4.2	Image class	15

3.4.3	Quadtree and multi-resolution	17
3.5	The history of Smalltalk	19
3.6	Smalltalk versus other OOLs	20
3.7	The development environment	21
3.7.1	Inspecting an instance of Form	22
3.7.2	Browsing a disk file containing a Form	23
3.7.3	Constructing windows	24
3.8	Bit-block transfer (BitBlt)	24
3.8.1	The operations	24
3.8.2	Advantages of Smalltalk in image processing	25
Chapter Four	Mathematical morphology and image algebra	26
4.1	The overview of morphology	26
4.1.1	Image and image transformation	27
4.1.2	Mathematical morphology for binary images	28
4.2	History of image algebra	30
4.3	Huang's image algebra	31
4.3.1	Two principles and the basic elements	31
4.3.2	Three basic operations	32
4.3.3	Other operations	34
4.4	Applications of image algebra in Smalltalk	41
4.4.1	Applications: filters	41
4.4.2	Application: shape recognition (template matching)	43
4.4.3	Application: edge detection	44
4.4.4	Application: convex hull	44
Chapter Five	Development of image-algebra applications in Smalltalk	47
5.1	Overview of image-processing applications	48
5.2	An image-processing system in Smalltalk/V286	50
5.2.1	The user interface of the application	50
5.2.2	The class ImageDataBaseInspector	54
5.2.3	The class ImageProcessor	56
5.2.4	Adding methods to a system class	57
5.2.5	Limitations in Smalltalk/V286	59
5.3	ImageLab in Smalltalk/VWin	60
5.3.1	The Smalltalk/VWin environment	60
5.3.2	The classes GraphLab and ImageLab	62

5.3.3	Multi-Document Interface	65
5.3.4	Tool bar and status bar	68
5.3.5	On-line help system	69
5.4	Customised design	72
Chapter Six	Exploring gray-scale images	75
6.1	Ways of viewing gray-scale erosion and dilation	75
6.2	Pseudo gray-scale morphological operations	79
6.3	Three-level gray-scale images	82
Chapter Seven	An application of ImageLab	85
7.1	Labelling connected components in binary image	85
7.1.1	The classical method	86
7.1.2	The morphological method	90
7.1.3	Application of the new labelling method	94
Chapter Eight	Non-morphological image processing methods in Smalltalk	96
8.1	Quad-tree in Smalltalk	96
8.1.1	Quad-tree fundamentals	96
8.1.2	Quadcode in Smalltalk	97
8.2	Adaptive quadtree: A new method for image coding	101
Chapter Nine	Conclusion and further work	102
9.1	Conclusions	102
9.2	Further work	102
9.2.1	Further work on gray-scale images	103
9.2.3	Other Smalltalk platforms	103
9.2.3	Object-oriented image database	103
9.2.4	An image processing language	103
References		105
Appendices		A1
Appendix A	Disk 1 (The programs in VWin)	A1

Appendix B	Disk 2 (Sample images and programs in V286)	A1
Appendix C	Classes and methods of Small-Image Database	A2
Appendix D	Classes and methods of Image processor	A6
Appendix E	Classes and methods of Image(V286)	A14
Appendix F	Classes and methods of ImageLab	A19
F.1	Methods in class Bitmap	A19
F.2	Methods in class ImageLab	A31
F.3	Methods in class ILTextWindow	A49
F.4	Methods in class ImageWindow	A50
F.5	Methods in class Matrix	A51

List of figures

Figure 2.1	Fundamental steps in digital image processing	6
Figure 3.1	Illustration of the relationship between class and objects	12
Figure 3.2	An image object	15
Figure 3.3	Class Image and its hierarchy	16
Figure 3.4	Partitioned image	17
Figure 3.5	Corresponding quadtree of partitioned image	18
Figure 3.6	A typical Smalltalk environment(VWin)	21
Figure 3.7	An instance of class Form	22
Figure 3.8	Bit array of form circular disk	22
Figure 3.9	The form store format in Smalltalk/V286	23
Figure 4.1	Illustration of dilation	28
Figure 4.2	The three basic operations	33
Figure 4.3	Sample images for the difference operation	35
Figure 4.4	Sample images for the intersection operation	35
Figure 4.5	Sample images for the erosion operation	36
Figure 4.6	Sample images for the symmetric difference operation	37
Figure 4.7	Sample images for the open operation	38
Figure 4.8	Sample images for the close operation	38
Figure 4.9	Sample images for the hit or miss transform operation	39
Figure 4.10	Sample images for the thinning operation	40
Figure 4.11	Sample images for the thickening operation	41
Figure 4.12	Sample images showing noise removal	42

Figure 4.13	Sample images for removing high frequencies	42
Figure 4.14	Sample images for band pass filter	43
Figure 4.15	Sample images for shape recognition	44
Figure 4.16	Sample image of edge detection	44
Figure 4.17	Sample image of convex hull	46
Figure 4.18	Sample images of shape analysis by using convex hull	46
Figure 5.1	The run-time structure	50
Figure 5.2	The user interface of the Small-Image Database	51
Figure 5.3	The window of image processor environment	52
Figure 5.4	The structure of class ImageDatabaseInspector	55
Figure 5.5	Communication via a global variable Q	57
Figure 5.6	Class hierarchy structure	61
Figure 5.7	A typical ImageLab working environment	68
Figure 5.8	Tool pane of ImageLab	69
Figure 5.9	Status pane of ImageLab	69
Figure 5.10	Work flow diagram	71
Figure 5.11	Contents of ImageLab's on-line help system	72
Figure 6.1	OR table	81
Figure 6.2	MAX table	81
Figure 6.3	The original image (Lena)	82
Figure 6.4	The edge detected image	83
Figure 7.1	J shapes template for blob colouring	87
Figure 7.2	L shapes template for blob colouring	89
Figure 7.3	Original image for D-transform	91
Figure 7.4	D-transformed image	91
Figure 7.5	Foreground reference image	92
Figure 7.6	Background reference image	92
Figure 7.7	Original image	92
Figure 7.8	Transformed image	92

Figure 7.9	The original image X	93
Figure 7.10	The noise need to be removed	93
Figure 7.11	High frequency noise is removed	93
Figure 7.12	The blobs of interest are extracted	93
Figure 7.13	Selected blobs transformed to single points	93
Figure 7.14	The band filter image pair	94
Figure 7.15	Original image with clusters	95
Figure 7.16	Structuring element	95
Figure 7.17	Result after closing operation	95
Figure 7.18	Selected blobs transformed to single points	95
Figure 8.1	Representation of a binary image by a region quadtree	97
Figure 8.2	The original image	101
Figure 8.3	The quadtree	101
Figure 8.4	The adaptive quadtree	101

List of tables

Table 3.1	An image class	16
Table 3.2	A subclass of image class	16
Table 3.3	The form store format in Smalltalk/VWin	23
Table 3.4	BitmapFileInfoHeader format	23
Table 3.5	Bit Block operations	25
Table 4.1	Five basic elements	32
Table 5.1	A survey of current image-processing software	48

Chapter 1

Introduction

1.1 The problem

One of the approaches traditionally used for the building of image-processing systems makes use of subroutine libraries. Such an approach, while allowing some ease of development through standard pieces of software in the context of widely diffused languages, such as Fortran or C[CART89], has significant limitations. Extensibility is restricted: the *collage* of routines with other pieces of software can produce inconsistencies due to the overlaying of global control variables and the possible mismatch of the routine interfaces with the problem to be solved. Speed is affected by the difficulties of supporting integration for specific architectures.

Recent developments in the field of programming languages, specifically object-oriented programming, offer a highly modular structure of programs, good reusability of software modules with ease of reconfiguration and a single data model between the main memory and the database. Moreover, object-oriented languages allow the extension of the basic data types to data types related to the specific application. For example in the development of an image-processing system, an object-oriented language will facilitate the creation of an image data-type. In this way, a highly specific development environment can be created, based on *standard platforms*, without the need to define specific languages.

Smalltalk is a pure object-oriented programming language that provides great flexibility for scientific experiment and application development [LALO90a]. The programming development environment offers full object-oriented characteristics that allows programmers to enjoy the benefits of object-oriented programming such as *inheritance*, *encapsulation*, *polymorphism* and *classification*.

This work develops an image processing system within the Smalltalk environment. Since our aim is to explore the use of object-oriented programming in image processing, the system does not offer all the facilities of a commercial imaging system. In the main we have concentrated on those techniques based on *mathematical morphology*.

Mathematical morphology modifies an image by applying various binary operations to the image and a smaller image, called the *structuring element*. The first systematic treatment of mathematical morphology was the two-volume work by J. Serra [SERR82a]. However, a visit to Serra's work could well be prefaced by the reading of two relevant chapters of a recent book by R.M. Haralick and L. Shapiro[HARA92].

In the course of developing his Digital Optical Cellular Image Processor (DOCIP), K-S Huang [HUAN89] devised a Binary-Image Algebra (BIA) which allowed him to express morphological operations in terms of *three fundamental operations*. These three operations are implemented in Smalltalk as basic methods in this work.

Within Smalltalk the key underlying process in the author's work has been the *bit-block transfer (BitBlt)*. BitBlt appears to the programmer as a parallel operation on a rectangular array of bits(or bytes) and the operations of BitBlt are admirably suitable for morphological operations.

The use that one can make of image-processing software is limited by the functionality provided by the designer. There is a place for software that is expandable in the hands of a user. The present work creates within the Smalltalk environment a sub-environment for image processing. The environment may be used as a stand alone program, in which case no knowledge of Smalltalk is required. Alternatively, it may be run from inside the Smalltalk environment using only a cursory knowledge of Smalltalk. Within that environment the kernel operations may be readily expanded by a Smallalk programmer to perform other processes required by the user.

1.2 The scope of the thesis

The thesis is divided into nine chapters. Chapter Two contains a brief overview of the application of computers to image processing.

In Chapter Three, we introduce the object-oriented concepts, and the benefits which an object-oriented language can offer to image processing. Also in this chapter, we demonstrate how to model an image as an object. Beginning with a brief history of Smalltalk followed by a comparison of Smalltalk with other object-oriented languages, we highlight two features of Smalltalk particularly pertinent to image processing - the graphics classes and the bit-block-transfer operation.

Chapter Four presents the theoretical background of this project. We begin with a brief introduction to the concept of mathematical morphology and its history. We then turn to Huang's image algebra [HUAN89] which expresses each morphological operation in terms of three fundamental operations. For each operation we state Huang's expression, give the Smalltalk method implementing it, and use Smalltalk to generate the images illustrating the operation.

The core of Chapter Five is a description of the image-processing environment which we have developed in Smalltalk. This chapter begins with a brief overview of existing image-processing packages available either commercially or from the public domain, and running under the most common operating systems (DOS, Windows, Macintosh and UNIX). We then introduce our own image-processing environment in two versions: (1) the DOS version and (2) the Windows version - *ImageLab*.

Chapter Six gives an exploration on gray-scale images. First we introduce a different way of viewing the gray-scale dilation, then we present the pseudo gray-scale morphological operation.

The purpose of Chapter Seven is to show *ImageLab in action* by describing how the author used *ImageLab* to make a contribution to the research on the counting of 'blobs' in a binary image. Chapter Eight complements the preceding chapter by giving two examples of the implementation of non-morphological operations. In the first we create a class *QuadCode* which offers an introduction to multi-resolution techniques, and propose the adaptive quad-tree. In the second we explore the fast Fourier transform (FFT).

The final Chapter of the thesis contains the concluding remarks and some suggestions for future work.

In this thesis ordinary text appears in Times New Roman. However, we use Courier New for Smalltalk scripts and methods. We use Arial whenever Smalltalk

words (instance variable, method names, class names, etc) are embedded in ordinary text.

Smalltalk/V286, Smalltalk/VWin are registered trademarks of Digitalk Inc.. Microsoft and MS-DOS are registered trademarks, and Windows is trademarks of Microsoft Corporation. Smalltalk-80 is a trademark of Xerox Corporation.

Chapter 2

Image-processing languages - a brief review

Introduction

This chapter gives an overview of digital image-processing. We begin with a brief history of the impact of computers on image processing, and display the structure of a modern digital image-processing system. We then survey the computer languages, including object-oriented languages, that are used in digital image-processing.

2.1 The computer and digital image processing

Digital image-processing is used for two different purposes:

- enhancing particular aspects of an image; and
- preparing images for the measurement of the features and structures present.

In the early days, that is, in the 60s and 70s, digital image-processing was done on large computers, such as IBM main frames, DEC 11, and Cyber 170 etc.[PRES83]. In the late 1970s and early 1980s, image-processing languages tended to be specific to special-purpose and high speed image-processing hardware. Examples are MORPHAL generated at the Centre de Morphologie Mathematique for the Leitz TAS cellular logic machine, PPL (Pattern Processing Language) coded at the University of Linkoping for use with their PICAP hardware[PRES81]. Another development in image processing in the 1970s was the introduction of several specialized image-processing systems which, initially, were programmed entirely from a control console using specialized keysets[PATO79].

At that time (from late 60's to early 80's), there were three prominent scientific applications of digital image processing: (1) remote sensing, (2) medical imaging and (3) particle physics. Images in such applications are large, require high resolution, and need much computation time. The cost was such that only government and military agencies, in the main, could sponsor the work.

Today, we are in the middle of a revolution sparked by the rapid progress in video and computer technology. Personal computers and workstations have become powerful enough to process image data. They have also become cheap enough to be widely used. Consequently, image processing is turning from a specialized science in areas such as astronomy, remote sensing, electrical engineering, and computer science into a standard scientific tool. Applications of image processing are now found in virtually all natural sciences[JAIN89] and are serving many industrial purposes (eg. robot vision).

A general digital image processing system involves five sub-systems which are [GONZ92]:-

1. image acquisition,
2. image storage,
3. image processing,
4. image communication, and
5. image output.

The fundamental steps in digital image processing are shown in the following diagram:-

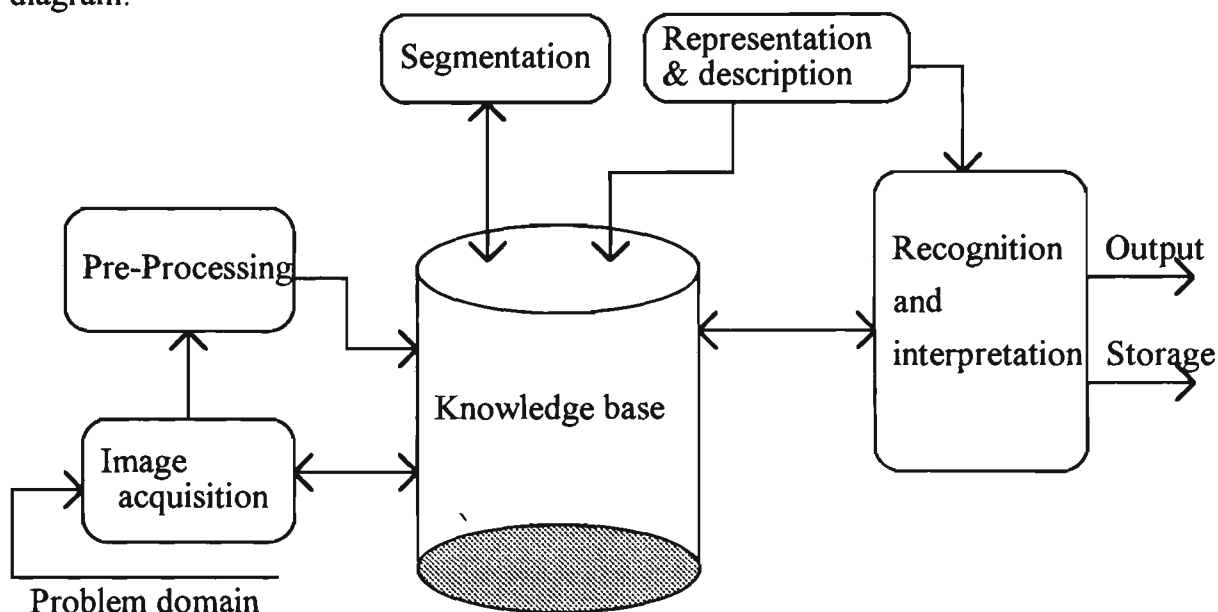


Fig. 2.1 Fundamental steps in digital image processing*

*Redrawn from [JAIN89]

Digital images can be acquired by video cameras, scanners and other digitisers. Output devices include monitors, film and printers. Storage of images entails the need for high capacity devices such as optical disks, magnetic tapes and magnetic hard disks. Software for general-purpose digital image-processing tends to be concentrated in the areas of pre-processing, segmentation, representation and description, recognition and interpretation [RUSS92].

2.2 The computer languages used in image processing

The origins of image-processing languages in the United States lie in two projects: (1) the DoD (ARPA)-sponsored ILLIAC project at the University of Illinois[McCO63] from which evolved the PAX language, later more fully developed and documented at the University of Maryland and (2) the NASA-sponsored IPL (Image Processing Laboratory) at JPL (Jet Propulsion Laboratory) which produced the language VICAR (Video Image Communication and Retrieval) reviewed by Castleman[CAST79]. These projects were implemented in the early to mid-1960s. PAX was written as a collection of FORTRAN subroutines to run on the Univac 1108, while VICAR, also in FORTRAN, was written for the IBM 360. PAX is a general purpose image-processing language while VICAR is more mission oriented being intended for use with the Ranger, Surveyor, Mariner, Viking and Voyager space exploration projects[PRES81].

It is not surprising that, until the early 1980s, FORTRAN was the language most widely used for digital image processing[PRES83]; it was widely available on many platforms and could satisfy the heavy computational demands of image processing.

However, since even the best FORTRAN compilers can not always generate assembly language optimised for rapid manipulation of arrays of numbers, many image-analysis systems are partly coded directly in assembly language, thus placing a burden on the image-processing programmer especially one whose training is not in computer science.

The growing population of small computers and the expanding use of the UNIX operating system has led to C becoming the most popular language for both system software and digital image-processing software[PIPE85].

Today, many widely used image-processing packages are written in C[LIND91]. Khoros, a complete image-processing package, running on the Sun-Sparc workstation, which has been placed in the public domain, is completely written in C. This package is widely used by scientists around the world, and there is a very active news group on Internet. Much commercial image-processing software is also written in C; examples are GlobLab Image from Data Translation, and OPTIMAS from BioScan Incorporated.

Other popular computer languages are also employed in image processing. Pascal, a typical procedural computer language, was used to create NIH Image 1.52, image processing software for the Macintosh. Prolog was used by Vision Dynamics in creating the VSP software[BATC91].

Image-processing languages have also emerged as extensions of existing computer languages. An example is PICL (PIctorial C Language) which is an extension of PCL (Pyramid C language) and is oriented to image analysis because it supports pictorial data-types[GESU91].

To combine the advantages of mathematical rigour and completeness with the benefits of the object-oriented paradigm, Roberto Cecchini et al developed IL (Image Language) by extending a host language called Common Lisp Object System (CLOS)[CECC91]. Lambert used object-oriented programming in image-processing algorithms; he proposed a model for use of the object-oriented programming (C++) for digital image-processing software design and maintenance[LAMB93].

Chapter 3

Object-oriented techniques and Smalltalk

Introduction

This chapter has two objectives: to introduce object-oriented technology and its relevance to image processing; and to show why we have chosen Smalltalk as our medium for applying object-oriented techniques to image processing.

To attain the first objective we begin with a brief review of the history of object-oriented technology, and object-oriented languages (OOLs). We then examine object-oriented languages in more detail. We shall discuss the main constituents of an OOL. These will include *objects*, *messages*, *methods*, *classes* and *inheritance*. We shall then be ready to see the benefits which an OOL can offer to image processing. We shall see how an image may be treated as an *object*, how objects with identical behaviour and structure may be gathered in to a *class*, and how *inheritance* enables the commonality of similar classes to be elevated to a superclass. As a further example, we shall see how readily a data-structure such as a *quad-code*, which is part of the image-processing toolbox, may be accommodated within the Smalltalk system.

The work of this thesis conjoins two activities - image processing and object-oriented programming. Of the various object-oriented languages the one that emerged as most suitable for our task was Smalltalk, the subject of the second objective of this chapter. After a brief note on the history of Smalltalk and comparison with other OOLs we turn to a description of the Smalltalk environment.

Smalltalk is both a language and an environment. We shall examine both aspects in this chapter. For the moment we remark that amongst the object-oriented languages

Smalltalk is the purest, and amongst programming environments, Smalltalk's is the most creative and productive.

Finally we highlight two features of Smalltalk particularly pertinent to image processing - the graphics classes and the bit-block transfer operation.

3.1 The history of object-oriented technology

The first person to formally identify the importance of composing a system in levels of abstraction was Dijkstra [DIJK76]. Parnas later introduced the concept of *information hiding* [PARN72] which is central to the nature of an object. The greatest influence upon object-oriented development derives from a small number of programming languages which will be discussed in the next section. In two decades, the object-oriented technology has become mature and has divided into several branches such as, object-oriented language (OOL), object-oriented programming (OOP), object-oriented analysis (OOA) and object-oriented design (OOD)

3.2 The history of object-oriented programming languages

Several programming languages have contributed to the evolution of today's object-oriented programming languages (OOL). In the 1950s, LISP, a language for list processing, introduced the concept of dynamic binding. SIMULA 67, developed in the 1960s as a language for programming simulations, introduced the class as a language mechanism for encapsulating data, and inheritance as a mechanism for elevating the commonality of two classes into a superclass. Data *abstraction*, in the form of abstract data types, was introduced in the 1970s, first in academic languages such as CLU, developed at the Massachusetts Institute of Technology, and later in more commercially popular languages such as Ada and Modula-2. With the beginning of the 1980s came the real dawn of the object-oriented programming era. Smalltalk-80 was introduced commercially in 1983. Other object-oriented programming languages, such as C++, Objective-C, Eiffel, the Common Lisp Object System, and Actor became commercially available [COX91].

The object-oriented languages which emerged during the last decade fall into two camps. One camp contains the pure object-oriented languages where almost

everything is an object. This group includes Smalltalk, Actor, and Eiffel. The other camp includes the hybrid languages such as C++, Objective-C, The Common Lisp Object System (CLOS) and the various object-oriented Pascals [WINB90].

In general the pure object-oriented languages emphasise exploration and rapid prototyping, while hybrid languages emphasise runtime speed and ease of incorporating object-oriented extensions for the programmer with an orientation towards procedural languages. The more mature OOLs, such as Smalltalk, also offer robust class libraries and rich sets of development tools. These capabilities are gradually being incorporated into the hybrid languages.

3.3 Constituents of an OOL

An object-oriented programming language has four basic mechanisms. They are:-

- Objects,
- Messages and methods,
- Classes, subclasses,
- Inheritance.

3.3.1 Programming with objects

A traditional program consists of procedures and data. An object-oriented program consists only of objects that encapsulate both procedures and data. An object orientation views such a system as a collection of objects, where each object models an entity or event in an application problem and where all objects work together to achieve the goal and task of the overall system. The central software concept is "object". An object captures the identity, structure and behaviour of the application entity that it represents.

3.3.2 Computation by message-passing

Objects have the ability to act. Action occurs when an object receives a *message*, that is, a request asking the object to behave in some way. When object-oriented programs are executed, objects are receiving, interpreting and responding to message from objects.

Methods are procedures which reside in an object and determine how the object acts when it receives a message. The instance variables store information or data local to the object. Methods execute in response to messages and manipulate the values of instance variables. Methods may also send messages to other objects requesting action or information.

3.3.3 Abstraction to classes and subclasses

A *class* is a description of a set of nearly identical objects. It consists of methods and data that summarise common characteristics of a set of objects. The ability to abstract common methods and data descriptions from a set of objects and store them in a class is central to the power of object-orientation. An object is an instance of a class. A class can also summarise common elements for a set of subclasses. By using subclasses, object-oriented programmers describe applications as collections of general, or abstract, modules. Common methods and data are elevated as high as possible so that they are accessible to all relevant subclasses. The relationship between a class and its instances is similar to the relationship between a factory and its products:-

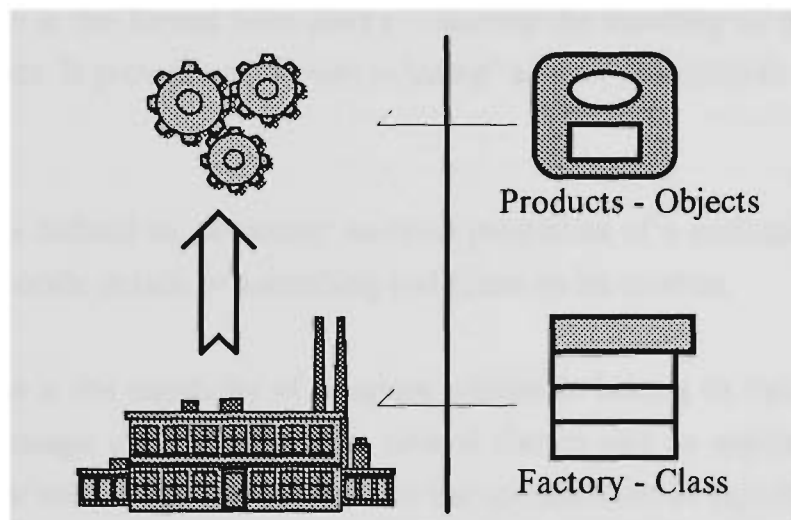


Fig. 3.1 Illustration of the relationship between class and objects

3.3.4 Programming in the presence of inheritance

The most important feature supported by object-oriented programming languages is *inheritance*: the ability to derive new classes from existing ones. Inheritance allows a programmer to use and extend large amounts of existing program code. Inheritance is

the object-oriented concept that contributes most to the increase in productivity which flows from the use of an object-oriented programming language.

Classes and their subclasses form class hierarchies, which capture the "is a" relationship among them. An instance of a derived class is also an instance of all its superclasses. While object-oriented languages all implement inheritance, they vary in how they treat multiple inheritance, and how they handle the access and redefinition of features in a subclass that are defined by a superclass. For example, some languages provide multiple inheritance, which means that classes can have more than one superclass. These languages provide more generality, solving, for example, problems like expressing what a toy house is. Is it a kind of toy, or is it a kind of house? With multiple inheritance, it can be both [COX91].

3.4 The benefits of introducing OOL to image processing

Four key concepts that summarise the advantages of the object-oriented approach are *encapsulation*, *abstraction*, *polymorphism* and *persistence* [MEYE88].

Encapsulation is the formal term used to describe the bundling of data and methods inside an object. It provides *information hiding** as well as access to selected features of an object.

Abstraction is defined as extracting essential properties of a concept. It allows us to neglect the specific details of something and focus on its essence.

Polymorphism is the capability of program entities to belong to more than one type. The same message may be defined in several classes and an argument passed by a message is not restricted in type. It allows the specification of algorithms at higher or more abstract levels.

Persistence refers to the permanence of an object, that is, the amount of time for which it is allocated space and remains accessible in the computer memory.

* The principle that users of a software component (such as a class) need to know the essential details of how to initialize and access the component, and do not need to know the details of the implementation. By reducing the degree of interconnectedness between separate elements of a software system, the principle of information hiding helps in the development of reliable software [BUDD91].

Before talking about the utility of image processing in an OOL, we need to examine the general benefits of an OOP language. An object-oriented programming language offers a major opportunity for improving software productivity [EGE92]. A programming language that supports the object-oriented paradigm benefits the software developer by providing a natural way to model complex, real-world phenomena, this resulting in faster and easier coding. The overall approach of reducing code by using inheritance to program the differences is one of the key tactics of object-oriented programming and a unique capability of object-oriented languages. Pre-defined class libraries, a component of the mature object-oriented languages which results in a reduction of design time and coding time, enhance the benefits of using object-oriented languages.

Encapsulation is one of the most beneficial concepts in the context of object-oriented programming. It combines data structures and functionality into objects. It also hides internal aspects of objects from its users and declares which features of an object are accessible[YOUR91a].

Another very important characteristic and benefit of object-oriented programming is that the interpretation of a message is in the hands of the receivers. Operations exhibiting this property are said to be *polymorphic*. Messages can be thought of as late-bound procedure calls, where the actual method of procedure to be invoked is not determined until the message is actually sent to a specific receiver. The programmer does not have to memorise a unique vocabulary for each class used in building his applications[YOUR91b].

3.4.1 Modelling an image as an object

Like other objects in the real world, images can be modelled as objects. Images have common characteristics which can be used in the description of the image. For example:-

- Size -- width and height.
- Colour -- mono or colour.
- Resolution -- bits per pixel.
- Offset -- positions mapping to screen or storage media.
- Data set -- set of pixel that describe an image(or bits).

Operations frequently performed on images are:-

- Copy -- copy to another device.
- Cut -- cut partial image.
- Paste -- paste another image to the image.
- Reflect -- reflect the original image about a line.
- Complement -- forming the photographic negative of the image.

In OOP this set of operations is called the *behaviour* of the image objects. Other examples of operations, combine two images, are the *union* of image and logical operations such as the logical *and* of pixel values.

The data and the behaviour can be encapsulated into an object. A set of image objects may be represented as a pack of templates:-

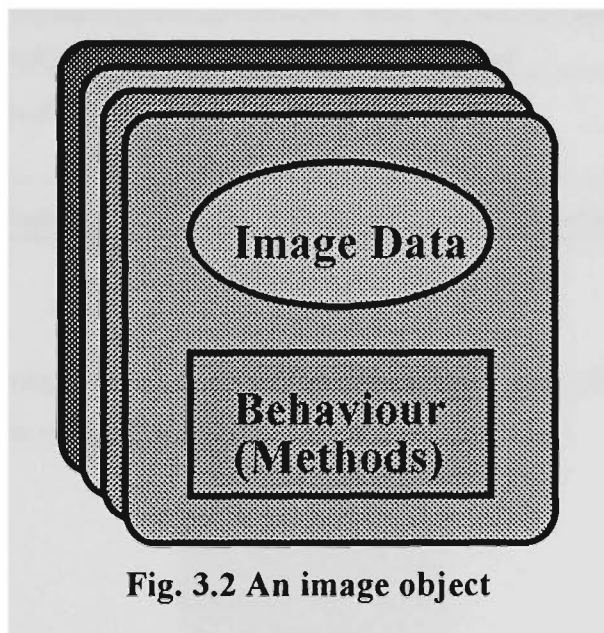


Fig. 3.2 An image object

3.4.2 Image class

After we have modelled images as objects, we can abstract the identical structure and behaviour of some into a class, and then elevate the commonality of similar classes into a superclass. The commonality of all images might be captured by a class Image:-

Class name	Image
Instance Variable	Size, name, bits...
Class Method*	fromFile:aFileName.
Instance Method	Copy, cut, paste...

Table 3.1 An image class

We can then introduce subclasses of Image to differentiate among binary, grey-scale and colour images. Consider, for example, a class BinaryImage as subclass of the class Image. Taking advantage of inheritance, we do not need to re-code the copy, cut and paste methods. The instance variables and methods of the class Image will be inherited by the class BinaryImage, and methods may be re-defined if necessary. Additional instance variables and methods which might be needed by the class BinaryImage may be added to it. In the following table, the parentheses enclose what is inherited from the superclass:-

Sub-Class name	BinaryImage
Instance Variable	(Size, name, bits...) offset
Class Method	(fromFile:aFileName...), fromScreen...
Instance Method	(Copy, cut, paste...), union, reflect ...

Table 3.2 A subclass of image class

In the same way we may define classes GrayScaleImage and ColorImage as subclasses of the class Image, thus yielding the following class hierarchy:-

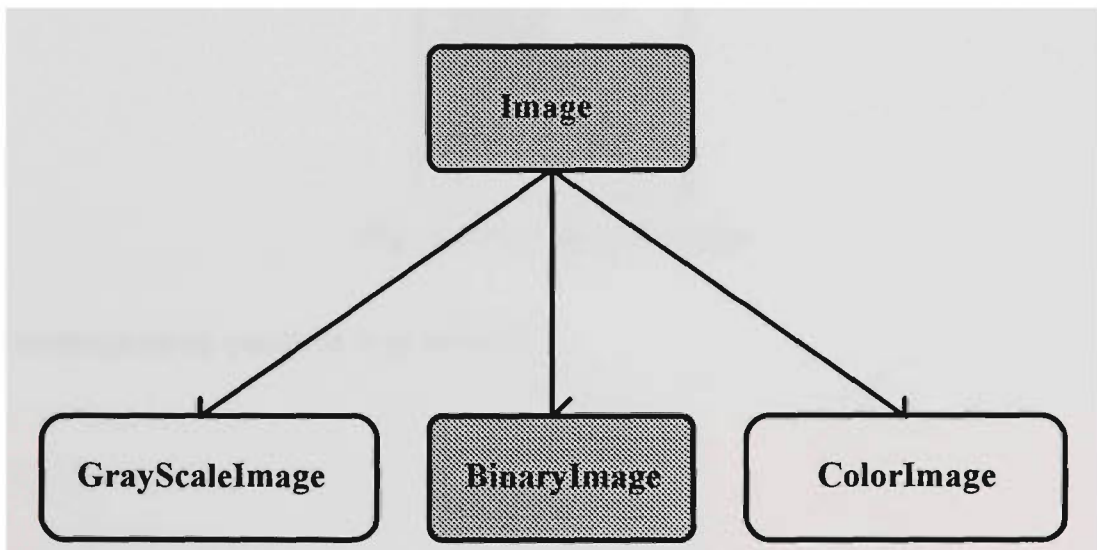


Fig. 3.3 Class Image and its hierarchy

* Class methods respond to message sent to class objects, rather than to instances of the class. They are often used for creating initialized instances of a class.

An object-oriented programming language supports polymorphism which allows us to pass the same message to objects produced by different classes. When an object receives a message the method for the message will be found in the class to which the receiver belongs. For example, the unary message selector **union** might be found in the class `GrayScaleImage` and in the class `BinaryImage`, but the methods would be different. When the message **union** is passed to an object which is an instance of class `BinaryImage`, the method **union** in class `BinaryImage` will be executed.

3.4.3 Quadtree and multi-resolution

A quadtree is a tree data-structure (a tree in which each node has exactly four descendants) which represents an image [HARA93]. Each node of the quadtree represents a square subset of the image's spatial domain. The root node of the quadtree represents the spatial domain of the entire image. If all pixels of the square represented by a node have the same value, then that node becomes a leaf in the tree. If the image being represented is a binary image, then the corresponding quadtree is called binary quadtree. If the image represented is a gray-scale image, then the corresponding quadtree is called a grey-scale quadtree.

For example, suppose we have the following image:-

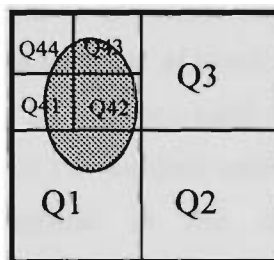


Fig. 3.4 Partitioned image

The corresponding quadtree is as follows:-

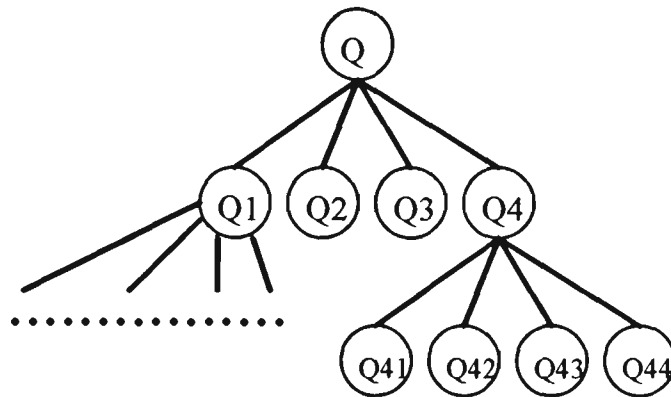


Fig. 3.5 Corresponding quadtree of above partitioned image

The root of the tree corresponds to the entire image, and each set of four child nodes of a parent node corresponds to a subdivision of the square represented by the parent node. In this case it is only the quadrants represented by Q1 and Q4 which have undergone further subdivision since all the pixels in Q2 and Q3 have the same value.

Since we model the image as an object and abstract it into a class and its subclasses, it is easy to use the data structure to encode and decode an image. All we need to do is to introduce a few instance methods into the class `BinaryImage` (to encode a binary image). If the method offers scope for generalisation we can elevate it to the class `Image`. We need to introduce instance methods as follows:-

- Introduce in the class `Image` an instance method to return a nominated quadrant of an `Image` - **quadrant: anInt**
- Introduce an instance method in the class `Image` to return a 'sub-quadrant' or quadcell by a `Quadtree` - **quadcellAt: aQuadcode**
- Introduce in the class `Image` an instance method to determine whether an image is white (pixel values equal to 1 if we deal with a binary image) - **isWhite**.
- Introduce an instance method in the class `Image` which returns an `OrderedCollection` of instances of `Quadcode`, indicating the quadrants which are white - **quadsWhite**

Two more instance methods are introduced into the class `Image`, **quadRecursiveWhite** and **quadRecursiveWhite:qC qCdClln:qCdColln**, which specify all quadrants and sub-quadrants.

We have written code to reconstruct an image from its quadcode. As with the encoding, we simply introduce pertinent methods into the appropriate classes. The details of implementing quadtree in an object-oriented language will be given in

Chapter Eight which discusses non-morphological image-processing methods in Smalltalk.

The class `Quadtree` exemplifies the use of object-oriented techniques in image processing. The first benefit is that we only need to program the difference, since the class `Quadtree` is a subclass of `OrderedCollection`. This not only increases our programming productivity but also reduces the number of mistakes in coding the program thus reducing the time spent in debugging. Another obvious advantage is the reuseability of the code. Some of the methods for a binary image will be applicable to a gray-scale image or a colour image without modification because we defined `GrayScaleImage` and `ColorImage` as subclasses of the class `Image`. More advantages will be discussed in later chapters.

3.5 The history of Smalltalk

In the early 1970s, Alan Kay and Adele Goldberg developed the *Smalltalk* system at the Software Concepts Group of the Xerox Palo Alto Research Centre. While not the first object-oriented programming language, it was Smalltalk which lead the way into the object-oriented era [WINB90].

The first publicly available version of Smalltalk was released in 1983 as Smalltalk-80. It was a result of evolution from the early versions Smalltalk-72, Smalltalk-74, Smalltalk-76 and Smalltalk-78 respectively. Smalltalk-80 was initially available only on powerful graphical workstations. Because Smalltalk is embedded in a complete interactive programming environment it requires significant memory, computing, and graphics capabilities. However these requirements become less onerous as the cost of memory, storage and speed falls. The current versions of Smalltalk, Object-works for Smalltalk-80 and Smalltalk/V, are available for a range of computers, from small personal computers up to the most advanced graphical workstations [LALO90a].

There are also two versions of Smalltalk which reside in the public domain: Little Smalltalk which implements the language only and provides a text-based user interface; and GNU Smalltalk, which is based on the X11 window system.

In this project, we used the dialect of Smalltalk which is marketed by Digital Inc. -- Smalltalk/V 286 (a DOS version) and Smalltalk/VWin 2.0(a WINDOWS version).

In this chapter, VWin refers to Smalltalk/VWin and V286 refers to Smalltalk/V286. Smalltalk/V is available on various personal platforms such as PCs and Macs.

3.6 Smalltalk versus other OOLs

Smalltalk is the purest of the object-oriented languages. In Smalltalk every object is an instance of a class; in contrast, C++ grafts class objects onto the non-class objects of C. In Smalltalk a variable is a pointer to an object, the assignment operator re-directs a pointer, and one explicitly makes a copy of an object when necessary; in contrast, in C++, making copies is implicit and one must explicitly introduce pointers in order to avoid copying. In Smalltalk it is implicit that all objects are created at runtime, and a garbage collector gathers any object which is no longer referenced by another object; in contrast, in C++, one must explicitly use `new` to create an object at runtime, and use `delete` to reclaim its space. In Smalltalk it is implicit that the method invoked by a message is determined by the class (rather than a superclass) of the receiver; in contrast, in C++, one must explicitly use `virtual` to accomplish the same effect. Smalltalk has always been equipped with a vast library of classes; in contrast, although class libraries have begun to appear with various C++ compilers, they have yet to achieve the commonality that exists amongst the dialects of Smalltalk.

The cost of simplicity and purity can be loss of power. Smalltalk achieves encapsulation by hiding all instance-variables and exposing all instance-methods. C++ grants the programmer the discretion to classify members as private or public (or protected).

However all such comparisons as above must be relative to the context in which the language is to be used. For example, if the context is the teaching of object-oriented programming then Smalltalk has much to commend it, even though a novice, especially if the person is an experienced programmer of conventional languages, might find it hard to master the Smalltalk approach at the beginning. The biggest step is to write the first Smalltalk program. After that, programming productivity will rapidly increase.

Smalltalk excels in the context of development of a fully-functioning prototype of an application. Its wealth of predefined classes allows rapid creation of new applications. Execution speed, however, is a negative factor, due to Smalltalk's dynamic nature .

Smalltalk also excels in the context of exploring concepts from some domain, such as image processing.

3.7 The development environment

Smalltalk provides the richest and most mature programming environment. Smalltalk programming is characterised by a total integration of tools. Editors, file managers, compilers, debuggers, and print utilities are all included within the Smalltalk environment. All those tools are available at all times. The Smalltalk programmer carries on a series of activities or conversations with individual tools. These activities can be interleaved. Activities or conversations can be interrupted and resumed at any time without loss of context or information. Switching from one activity or conversation to another is as simple as clicking a mouse button. The following is a typical screen display in the Smalltalk development environment:-

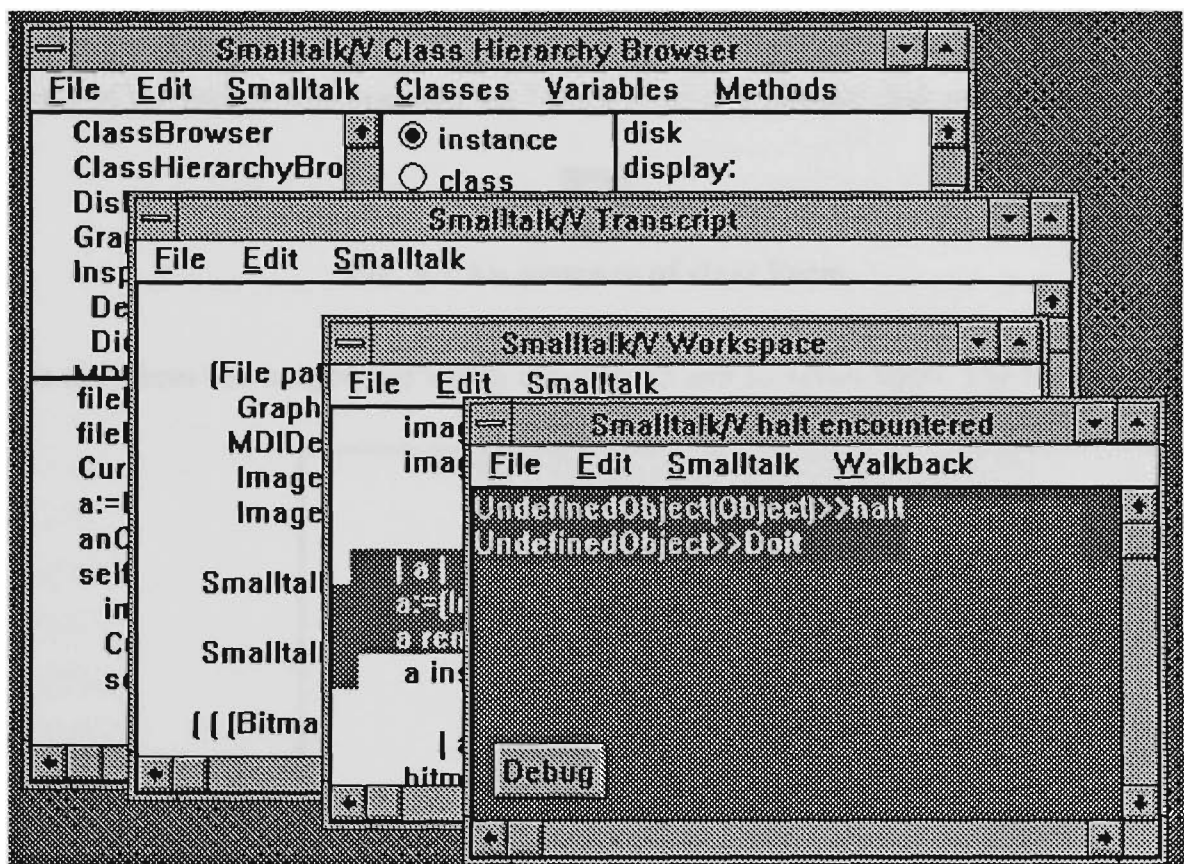


Fig. 3.6 An typical Smalltalk environment(VWin)

Smalltalk applications are developed by piece-by-piece addition to the Smalltalk system -programming by extension. The system contains an extensive on-line library of classes. Moreover, the source is written almost entirely in Smalltalk and can be viewed and modified by the programmer. Programming is totally interactive. New or

modified code can be recompiled and tested in a matter of seconds. Sequences of such modifications result in working prototypes and eventually elaborate designs that can be polished and turned into finished applications. This style of program development could be described as programming by iterative enhancement.

3.7.1 Inspecting an instance of Form

The two fundamental classes for creating and manipulating graphical images in Smalltalk-80 are the classes `Form` and `BitBlt`. Forms are used to represent images, while instances of class `BitBlt` represent operations on forms. The corresponding classes in `VWin` are `Bitmap` and `GraphicsTool` respectively.

Let us examine an instance of the class `Form`. The data of an instance of `Form` comprises a height, a width, an offset, and a bit-array that stores the image. The offset of a form is the amount by which the form should be offset when it is displayed or when its position is tested. Every form has an assumed origin at the top left-hand corner of the image. For example, we have a form of a circular disk as follow:-



Fig. 3.7 An instance of class Form

This disk form has **height** and **width** equal to 32 and an **offset** 0@0. The bit array is:-

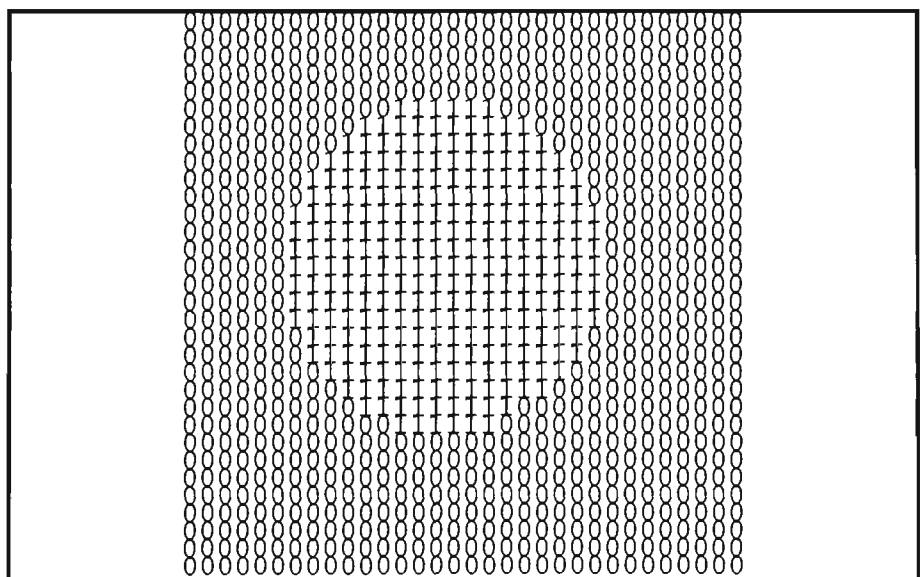


Fig. 3.8 Bit array of form circular disk

In the above figure, "1" represents a white pixel in the circle form and "0" stands for a black pixel in the form.

3.7.2 Browsing a disk file containing a Form

Different platforms have different ways of storing forms. In Smalltalk/V 286 we shall define our own file format as:-

Image type(1)	space	Image name(8)	space	Width(2) Height(2)	Bit Array(n)
------------------	-------	------------------	-------	-----------------------	-----------------

Fig. 3.9 The form store format in Smalltalk/V286*

In Smalltalk/VWin, forms (images) are stored in windows bitmap format. A bitmap file stored by Smalltalk/VWin will have following structure:-

Bitmap File Header
Bitmap Information Header
Bitmap Byte Array

Table 3.3 The form store format in Smalltalk/VWin

The BitmapFileHeader in the above table is the first fourteen bytes of the bitmap file. The following table illustrates the BitmapFileHeader structure*:

Field Type	Argument Type	Description	Restrictions
WORD	bfType	Type of file	Must be BM
DWORD	bfSize	Size of file	Specified in DWORDs
WORD	bfReserved	RESERVED	Must be set to 0
WORD	bfReserved	RESERVED	Must be set to 0
DWORD	bfOffBits	Offset to beginning of bitmap	Specified in bytes

Table 3.4 BitmapFileInfoHeader Structure Format

* Numbers shown in parentheses are bytes occupied in the format

* Source: Microsoft Windows 3.0 SDK Programmer's Reference, Page 7-10

3.7.3 Constructing windows

We can easily manipulate images with the help of Smalltalk's graphical programming environment. To construct a graphics window to display images, we need to invoke the `GraphicPane` class in V286 or the `GraphPane` class in VWin, to obtain an instance of those classes.

VWin graphics windows are built on a graphics interface language called Graphics Device Interchange (GDI) in Microsoft Windows. In run time, VWin calls the GDI library functions to implement Smalltalk graphical operations.

3.8 Bit-block transfer (BitBlt)

The class `BitBlt` is the heart of Smalltalk's graphics system. It performs all bitmap manipulations. Smalltalk-74 was the first Smalltalk to use `BitBlt` as its main operation for bitmap graphics. The specification for `BitBlt` arose out of earlier experience with Turtle graphics, text display and other screen operations such as scrolling and menu overlays. The specification of `BitBlt` has been used by others under the name `RasterOp`. These operations are implemented directly as machine-coded primitives to improve performance [KRAS83].

Recent work by Simon Lau [LAU93] demonstrated the performance gains in using `BitBlt`. He produced various implementations of Conway's game of Life in V286. The best time he could get without `BitBlt` was 3 minutes. He then used the `BitBlt` algorithm from Adele Goldberg's book "Smalltalk-80 The language" [GOLD89], and reduced the time to 7 seconds - reduction by a factor of about 25.

3.8.1 The operations

In Smalltalk, the `BitBlt` operation is implemented by sending the message `copyBits` with different rules. The `BitBlt` operation rules in Smalltalk/V286 are:

3	Form over	destination becomes source
7	Form orRule	source OR into destination
1	Form andRule	source AND into destination
1	Form under	source AND into destination
4	Form erase	if source is 1 then destination becomes 0
6	Form reverse	source XOR into destination
17	Form orThru	first erase without specifying mask form, then OR with mask form specified

Table 3.5 Bit Block operations*

VWin uses Window's 256 BitBlt operations which include the seven operations in V286. In Smalltalk/V286 syntax, the copyBits message is executed like this:-

```
dForm copy: (sPoint extent:sExtent)
  from: sForm
  to: dPoint
  rule: anInt
```

3.8.2 Advantages of Smalltalk in image processing

An important class of image processing algorithms is based upon the theory of mathematical morphology, described in detail in the next chapter. Morphological operations form logical combinations of the values at sets of adjacent pixels. The BitBlt operations are admirably suited to morphological operations since they appear to the programmer as parallel operations on a rectangular array of bits (or bytes). In the image processing area, one approach is to use image algebra to express image operations.

We shall find that Huang's image algebra [HUAN89] provides a decomposition of general operations, including low-level image processing operations, into three fundamental operations. This decomposition is inherently parallel and provides a direct mapping to the machine architecture, such as the *Bit Block Transfer* in Smalltalk.

* Re-write from Smalltalk/V286 User Manual

Chapter 4

Mathematical morphology and image algebra

Introduction

The word *morphology* commonly denotes a branch of biology that deals with the form and structure of animals and plants. We use the same word here in the context of *mathematical morphology* as a tool for extracting image components that are useful in the representation and description of region shape, such as boundaries, skeletons, and the convex hull.

This chapter presents the theoretical background of this project. We begin with a brief introduction to the concept of mathematical morphology and its history. We then turn to Huang's image algebra which expresses each morphological operation in terms of three fundamental operations. For each operation we state Huang's expression, give the Smalltalk method for implementing it, and use Smalltalk to generate the images illustrating the operation. Next, we give examples of the use of these operations in image processing: construction of filters, shape recognition and forming the convex hull of a set.

4.1 The overview of morphology

Mathematical morphology is that part of image processing which is concerned with image filtering and analysis by *structuring elements*. It grew out of the early work of H. Minkowski and H. Hadwiger [MINK03] on geometric measure theory and integral geometry, and entered the modern era through the work of G. Matheron and J. Serra [SERR82a] of the Ecole des Mines in Fontainebleu, France. Matheron and Serra not

only formulated the modern concepts of morphological image-transformations but also designed and built the Texture Analyser System. Since those early days, morphological operations and techniques have been applied to vision problems at all levels.

Mathematical morphology is a form of mathematics used for analysing and describing shapes. It treats images as set of points in space (rather than as arrays of numbers or as connected blobs). Because it treats images as sets, the operations for combining two images are set operations, rather than arithmetic ones.

This approach is both non-linear and irreversible -- for most morphological operations there is no inverse operation to undo their effect. Each operation thus loses some of the information that was there before. It is the art of the user to manage the loss of information, so that he can extract the message that he seeks and progressively eliminate irrelevant detail.

4.1.1 Image and image transformation

We are concerned with images defined at points (x, y) in the plane. We denote the universal set of all image points by W . Normally W will be some rectangular window. An image is defined by a function g on W such that $g(x, y)$ is the value of the image at (x, y) . This will be a real value in the case of a gray-scale image or a 3-component vector for a colour image.

A *binary image* is a special case of gray-scale image for which the set of image values is composed of two elements: "1" representing white, a foreground point or image point, and "0" representing black, a background point. The set of foreground points in such an image is interpreted as the object or shape depicted by the image. So a binary image may be considered as a set, the set of points in W with value 1.

An *image transformation* is a mapping T from each image function g to a new image function T_g , converting each image to a new image. Mathematical morphology is concerned with a special class of such functions. This is most easily explained for binary images.

4.1.2 Mathematical morphology for binary images

For binary images, mathematical morphology treats images as sets of points. So the image transformations are considered as set operations. Serra [SERR82a] restricts mathematical morphology on binary images to those set operations that satisfy four principles. The first is that the transformation must be independent of a translation of the sets, that is, the transform of the translation of a set must be the translation of the transform of that set. The second is that the transformation must be independent of a scaling of the set. The final two principles are rather technical, concerned with the localisation of the information used to compute the result and a requirement that arbitrarily small changes in a set cannot produce large changes in its transform.

In practice, mathematical morphology is mainly concerned with transformations based upon two fundamental ones: *dilation* and *erosion*. We now describe these. At this stage we restrict the underlying image space to the discrete lattice $J \times J$, the set of points with integer coordinates, although mathematical morphology also applies to images defined on a continuous domain. Now (x, y) is considered to define a pixel and $g(x, y)$ the image value there.

Consider an image A (considered as a subset of the pixels in W). We perform a dilation to obtain a new set B in the following way. Each pixel in W will be in B if it, or any of its "neighbours" are in A . There is complete freedom in deciding which pixels are considered neighbours so long as there is consistency. For example, Figure 4.1(a) shows one possible selection of neighbours. The central pixel, containing a circle, is interpreted as the pixel under consideration and the pixels denoted by crosses are the neighbours. This particular dilation will convert set A , shown in Fig. 4.1(b), to set B in Fig. 4.1(c).

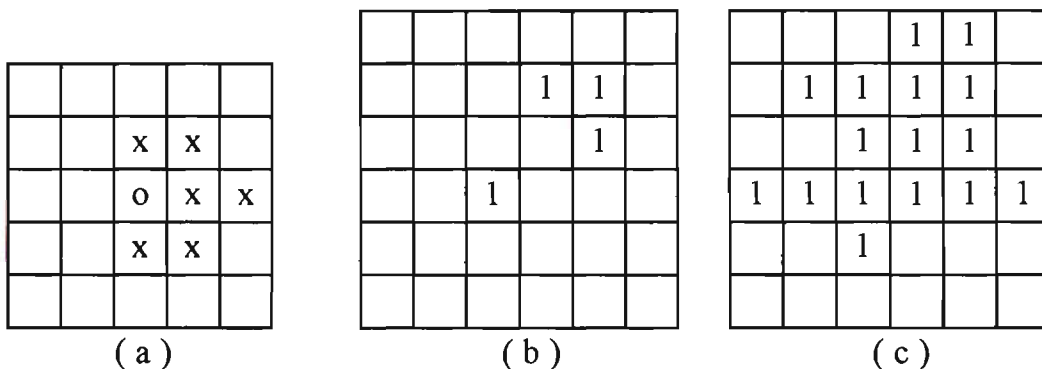


Fig. 4.1 Illustration of dilation

Mathematically, the most convenient way to describe this process is to consider the pixels marked o or x in (a) as a set or image itself, called the *structuring element*. We denote it by R here. Then the result of the dilation is written $A \oplus R$. For the structuring element the origin is taken at the o pixel. Then in set notation,

$$A \oplus R = \{(x, y) \mid (x + u, y + v) \in A \text{ for any } (u, v) \in R\}$$

For later generalisation to gray-scale images, an alternative formula is

$$(A \oplus R)(x, y) = \bigvee_{(u,v) \in R} g(x+u, y+v)$$

where \bigvee denotes the logical OR of the Boolean g values.

An erosion C of a set A is defined similarly to a dilation, except that pixel (x, y) will be in C if (x, y) and all of its neighbours are in A . For a structuring element R the result is written $A \ominus R$. So

$$A \ominus R = \{(x, y) \mid (x + u, y + v) \in A \text{ for all } (u, v) \in R\}$$

or equivalently,

$$(A \ominus R)(x, y) = \bigwedge_{(u,v) \in R} g(x+u, y+v)$$

Where \bigwedge denotes the logical AND.

We may consider the structuring element as a mask. The origin of the mask is moved to each pixel in turn and that pixel is considered part of the dilation of A if the structuring element touches A . It is considered part of the erosion of A if the structuring element lies entirely inside A . In the example shown in Fig. 4.1, the structuring element can nowhere fit inside A , so the erosion of A in that case is the null set.

The complement of a set A (denoted by \overline{A}) is the set of points in the universal set W but not in A . Clearly dilating a set has the effect of eroding its complement. More precisely

$$A \ominus R = \overline{\overline{A} \oplus \check{R}}$$

Here the notation \check{R} signifies the reflection of the set R ,

$$\check{R} = \{(x, y) \mid (-x, -y) \in R\}$$

More generally, a morphological operation is some logical combination of the Boolean values $g(x+u, y+v)$ where (u, v) ranges over the structuring element. In general we drop the requirement that the origin is in the structuring element. Different structuring elements address different concerns. For example, a 3-by-3 square structuring element can be used to eliminate a round corner in a shape, and a structuring element in the form of a thin line can be used in the processing of striated textures.

4.2 History of image algebra

Serra and Sternberg were the first to unify morphological concepts and methods into a coherent algebraic theory specifically designed for image processing and image analysis. Sternberg was also the first to use the term *image algebra* [STER80]. More recently, P. Maragos introduced a new image algebra unifying a large class of linear and non-linear systems under the theory of mathematical morphology [MARA85].

G.X. Ritter et al introduced Air Force Armament Laboratory (AFATL) Standard Image Algebra [RITT90]. This algebra provides a common mathematical environment for image-processing algorithm development and methodologies for algorithm optimisation, comparison and performance evaluation. In addition, the image algebra provides a powerful algebraic language for image processing which, if properly embedded into high level programming, will greatly increase a programmer's productivity as programming tasks are greatly simplified due to replacement of large blocks of code by short algebraic statements.

Another algebraic structure with the same goals as the AFATAL image algebra is the algebra described by Giardina and Dougherty [DOUG87]. It also defines a set of basic or primitive image operations.

In the course of developing his Digital Optical Cellular Image Processor (DOCIP), K.S Huang [HUAN89] devised a Binary-Image Algebra(BIA) which allowed him to express morphological operations in terms of three fundamental operations. A basic purpose of BIA is the development of a programming language for the specific parallel architecture of DOCIP.

4.3 Huang's image algebra

Huang's image algebra provides a decomposition of general operations, including low-level image processing operations, into three fundamental operations. The parallelism of these fundamental operations makes them suitable for expression in terms of the bit block-transfer operations in Smalltalk.

4.3.1 Two principles and the basic elements

Huang proved his two fundamental principles that basically define the BIA. They are:-

Principle 1.

Fundamental Principle of Image Transformations

Any binary image morphology transformation T can be implemented by using an appropriate reference image R and the three fundamental operations (details in next section):-

1. Complement of an image.
2. Union of two images.
3. Dilation of an image using a reference image.

Principle 2.

Fundamental Principle of Reference Images R

Any reference image R can be generated from elementary images $(I, A, A^{-1}, B, B^{-1})$ by using the three fundamental operations.

The five elementary images are constant images. Each elementary image has only one image point. They are:-

$I = \{(0,0)\}$	consisting of an image point at the origin
$A = \{(1,0)\}$	consisting of an image point right of the origin
$A^{-1} = \{(-1,0)\}$	consisting of an image point left of the origin
$B = \{(0,1)\}$	consisting of an image point above of the origin
$B^{-1} = \{(0,-1)\}$	consisting of an image point below of the origin

Table 4.1 Five basic elements

For example, we can define a 4-connected reference image thus:-

$$N_4 \equiv I \cup A \cup A^{-1} \cup B \cup B^{-1}$$

Huang defined 4-connected and 8-connected as follows:-

Two image points (x,y) and (i,j) of an image are 4-connected \leftrightarrow there exists a sequence of image points $(x,y) = (x_0,y_0), (x_1,y_1), \dots, (x_m,y_m) = (i,j)$, where (x_k,y_k) is a 4-neighbour of (x_{k-1},y_{k-1}) and $(x_k,y_k) \in X, 1 \leq k \leq m$.

Two image points (x,y) and (i,j) of an image are 8-connected \leftrightarrow there exists a sequence of image points $(x,y) = (x_0,y_0), (x_1,y_1), \dots, (x_m,y_m) = (i,j)$, where (x_k,y_k) is a 8-neighbour of (x_{k-1},y_{k-1}) and $(x_k,y_k) \in X, 1 \leq k \leq m$.

Other operations such as *opening, closing, thinning and thickening* can be expressed in terms of three basic operations to be defined in the next section.

4.3.2 Three basic operations

Huang's binary-image algebra (BIA) expresses general operations on binary-images in terms of three fundamental operations:-

1. Complement of an image X .

$$\bar{X} = \{(x,y) | (x,y) \in W \wedge (x,y) \notin X\}$$

2. Union of two images X and R .

$$X \cup R = \{(x,y) | (x,y) \in X \vee (x,y) \in R\}$$

3. Dilation of an image X with a reference image R .

$$X \oplus R = \{(x1+x2, y1+y2) \in W | (x1, y1) \in X, (x2, y2) \in R\}$$

Following are some examples of the three basic operations:-

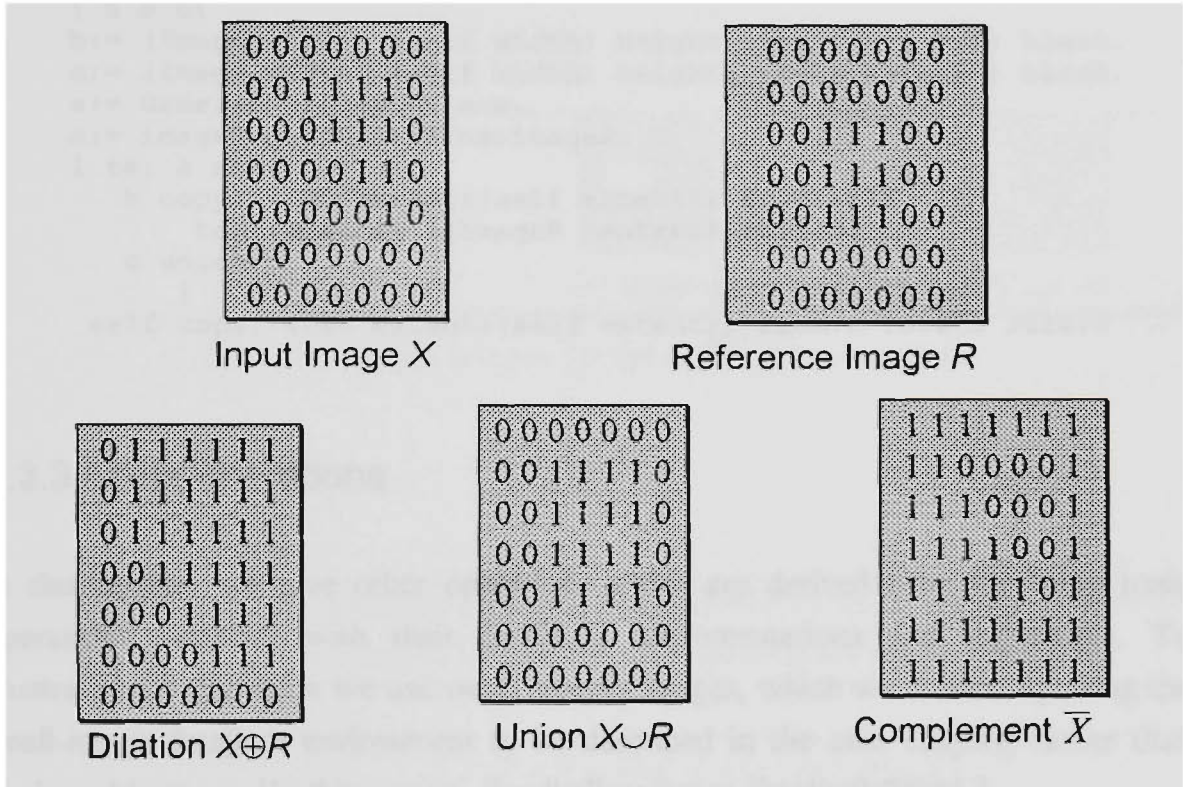


Fig. 4.2 The three basic operations

In the following operations and in section 4.3.3, each operation is implemented by an instance method of the class Image. In conformance with Smalltalk convention, we use the comment to make it clear whether we are changing the receiver and returning it, or whether we are returning a new instance of Image. In Smalltalk the default return is self, that is, the receiver. In the following methods, we are changing the receiver.

We may implement the fundamental operations in Smalltalk/V286 by means of the following instance methods:-

```

complement
"Answer the complement of an image. "
^self reverse
    
```


union:imageR

"Answer an image containing the image of the union of imageR and the receiver imageX. "

```
self copy: (0@0 extent:(imageR extent)) from:imageR to:0@0 rule:7 .
```

dilationBy:imageR

"Dilation of an image (imageX) by a reference image (imageR) "

```
| a b c|
b:= (Image width: (self width) height: (self height)) black.
c:= (Image width: (self width) height: (self height)) black.
a:= OrderedCollection new.
a:= imageR getPointsFrom:imageR.
1 to: a size do:[:i|
  b copy: (0@0 extent:(self extent)) from:self
    to:((a at:i)- (imageR centre)) rule:3.
  c union:b
] .
self copy: (0@0 extent:(self extent)) from:c to:0@0 rule:3
```

4.3.3 Other operations

In this section, we give other operations which are derived from the three basic operations, together with their Smalltalk implementations and test-scripts. To illustrate each operation we use *small sample* images, which we created by using the small-image database environment to be described in the next chapter, rather than *real-world* images. [In this section, Smalltalk refers to Smalltalk/V286.]

Difference

The difference is one of the standard operations. If R is a *template* or ideal image and X is the actual image then X/R shows defects in X by showing the extra pixels that X contains. Huang's expression for the difference of an image X by an image R is given in his Formula 4.1:-

$$X / R = \{(x, y) | (x, y) \in X \wedge (x, y) \notin R\}$$

We may implement the operation in Smalltalk by means of the following instance method:-

difference:imageR

"Answer an image that contains the difference between original image (imageX) and reference image (imageR) "

```
((self complement) union: imageR ) complement
```

We may test the method in the following way using global variables X, Y and Z:-

```
X:= ImageDataBase idbFromDisk:'image.dbs'.
Y:=X at:'differenceX'.
Y displayAt: 0@20.
R:=X at:'differenceR'.
R displayAt: 50@20.
Y difference: R.
Y displayAt: 100@20
```

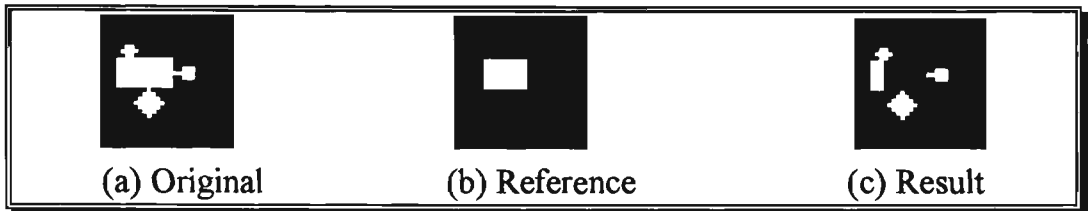


Fig. 4.3 Sample images for the difference operation

Intersection

The *intersection* operation is the parallel form of Boolean *AND*. Huang's expression for the intersection of an image X by an image R is given in his Formula 4.2 :-

$$X \cap R = \{(x,y) \in X \wedge (x,y) \in R = \overline{\overline{X} \cup \overline{R}}\}$$

We may implement the operation in Smalltalk by means of the following instance method:-

```
intersection:imageR
```

```
"Answer an image that contains the intersection of original image (imageX) and reference image (imageR)"
```

```
((self complement) union:(imageR complement)) complement
```

Although we do not give the script, this, and following operations may be tested in the same way as the preceding operation.

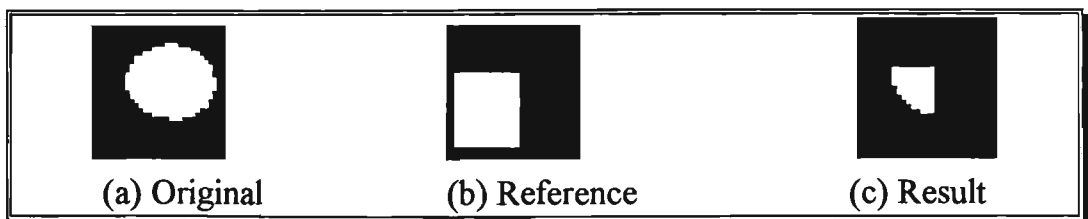


Fig. 4.4 Sample images for the intersection operation

Erosion

Erosion is the morphological dual to dilation. It is the morphological transformation which combines two sets using the vector subtraction of set elements. In general, the erosion of an image X by a reference image R can be used to remove pixels around the boundary of regions and so decrease their size, increase the size of holes, eliminate regions and break bridges in X . Huang's expression for the erosion of an image X by an image R is given in his Formula 4.3 :-

$$X \ominus R = \overline{\overline{X} \oplus \overline{R}}$$

We may implement the operation in Smalltalk by means of the following instance method:-

erosionBy:imageR

"Answer an image that containing the erosion of original image (imageX) by reference image (imageR) "

```
((self complement) dilationBy:(imageR reflect)) complement
```

We may test erosionBy: in the same way as before.

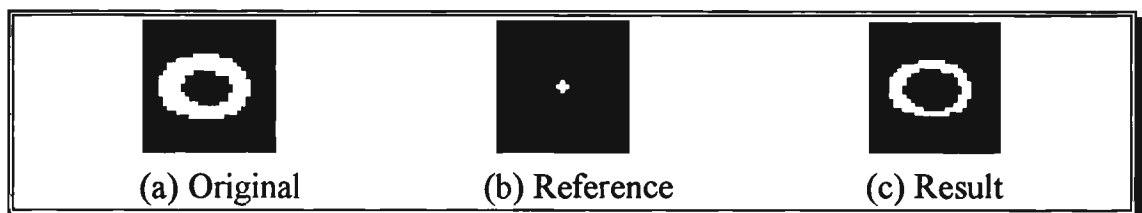


Fig. 4.5 Sample images for the erosion operation

Symmetric Difference

The *symmetric difference* operation is the parallel form of the Boolean Exclusive-OR. It is a commutative operation and its inverse operation is itself. If R is a template then $X \Delta R$ indicates all the discrepancies between X and that template and so detects defects in X . Huang's expression for the symmetric difference of an image X by an image R is given in his Formula 4.4 :-

$$X \Delta R = (X / R) \cup (R / X) = \overline{\overline{X} \cup R} \cup \overline{\overline{R} \cup X}$$

We may implement the operation in Smalltalk by means of the following instance method:-

symmetricDiff:imageR

"Answer an image which is the symmetric difference between original image (imageX) and reference image (imageR)"

```
| a b |
a:= (Image width: (self width) height: (self height)) black.
a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
a difference:imageR.
b:= (Image width: (self width) height: (self height)) black.
b copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
imageR difference:b.
a union:imageR.
self copy: (0@0 extent:(self extent)) from:a to:0@0 rule:3
```

We may test symmetricDiff: in the same way as before.

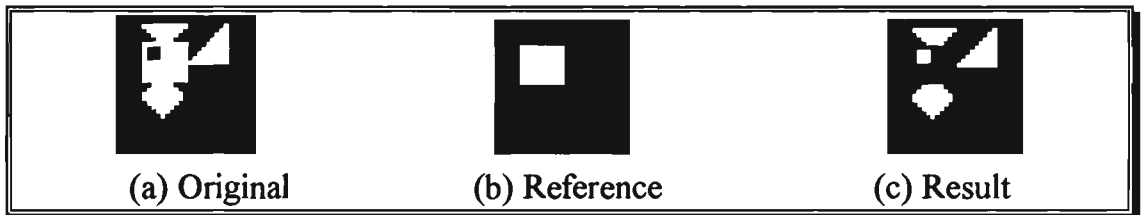


Fig. 4.6 Sample images for the symmetric difference operation

Opening

Opening of images X and R is simply the erosion of X by R , followed by a dilation of the result by R . We can interpret opening as moving the structuring element around inside the foreground of the image. Those parts that the structuring elements can reach are preserved. Opening generally smooths the contours of an image, breaks narrow isthmuses and eliminates thin protrusions. Huang's expression for the opening of an image X by an image R is given in his Formula 4.5 :-

$$X \circ R = (X \ominus R) \oplus R = \overline{\overline{X} \ominus \overline{R}} \oplus R$$

We may implement the operation in Smalltalk by means of the following instance method:-

openingBy:imageR

"The opening operation is an erosion followed by a dilation with the same reference imageR."

```
(self erosionBy:imageR) dilationBy:imageR
```

We may test `openingBy:` in the same way as before.

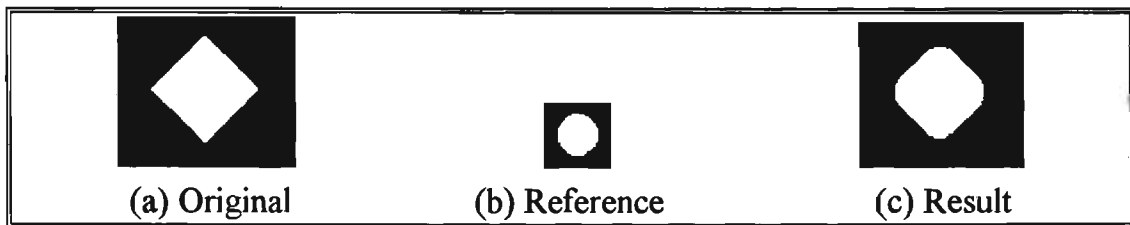


Fig. 4.7 Sample images for the open operation

Closing

The *closing* of images X and R is simply the dilation of X by R , followed by an erosion of the result by R . We can interpret closing as moving the reference image around in the background of the image. Those parts of the image that are not reached by the reference image constitute the closing. Closing tends to smooth sections of contours but, as opposed to opening, it generally fuses narrow breaks and long thin gulfs, eliminates small holes and fills gaps in the contour. Huang's expression for the closing of an image X by an image R is given in his Formula 4.6 :-

$$X \bullet R = (X \oplus R) \ominus R = \overline{\overline{(X \oplus R)} \ominus R}$$

We may implement the operation in Smalltalk by means of the following instance method:-

`closingBy:imageR`

"The closing operation is an dilation followed by an erosion with the same reference imageR."

```
(self dilationBy:imageR) erosionBy:imageR
```

We may test `closingBy:` in the same way as before.

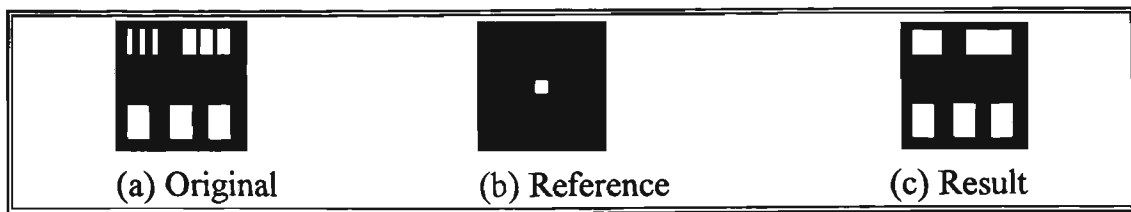


Fig. 4.8 Sample images for close operation

Hit or Miss Transform

The hit or miss transform uses a pair of reference images $R=(R1,R2)$. The transform is used to detect a given shape in the connected components of the foreground of an image. The reference images constitute a template for the shape sought; $R1$ specifies its foreground and $R2$ its background. Huang's expression for the hit or miss transform of an image X by an image pair $(R1,R2)$ is given in his Formula 4.7:-

$$X \ominus R = (X \ominus R1) \cap (\bar{X} \ominus R2) = \overline{(\bar{X} \oplus \bar{R}1) \cup (X \oplus \bar{R}2)}$$

We may implement the operation in Smalltalk by means of the following instance method:-

```
hitMissTransBy:imageR1 and:imageR2
"Instance method of Hit-or-miss transform."
```

```
| a b |
a:= (Image width: (self width) height: (self height)) black.
a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
a erosionBy:imageR1.
b:= (Image width: (self width) height: (self height)) black.
b copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
(b complement) erosionBy:imageR2.
a intersection:b.
self copy: (0@0 extent:(self extent)) from:a to:0@0 rule:3
```

In the following test, the pyramids in the original image are recognised, and represented by a dot in the resulting image.

```
X:= ImageDataBase idbFromDisk:'image.dbs'.
Y:=X at:'hitMissTrans'.
Y displayAt: 0@20.
R:=X at:'triangle'.
R displayAt: 50@20.
X:=X at:'cap'.
X displayAt: 100@20.
Y hitMissTransBy: R and: X .
Y displayAt: 150@20
```

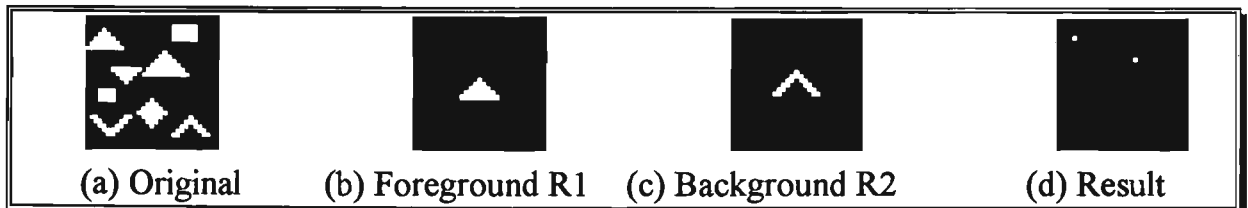


Fig. 4.9 Sample images for the hit or miss transform operation

Thinning

The *thinning* operation decreases the size by removing the central points of the regions which match the reference image pair $R = (R1, R2)$. Huang's expression for the thinning of an image X by an image pair $(R1, R2)$ is given in his Formula 4.8 :-

$$X \odot R = X / (X \ominus R) = \overline{\overline{X} \cup (\overline{X} \oplus \check{R}1) \cup (X \oplus \check{R}2)}$$

We may implement the operation in Smalltalk by means of the following instance method:-

thinningBy:imageR1 and:imageR2

"The instance method thinning."

```
| a |
a:= (Image width: (self width) height: (self height)) black.
a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
a hitMissTransBy:imageR1 and:imageR2.
self difference: a
```

We may test thinningBy: in the same way as we test hitTransBy: and: method.

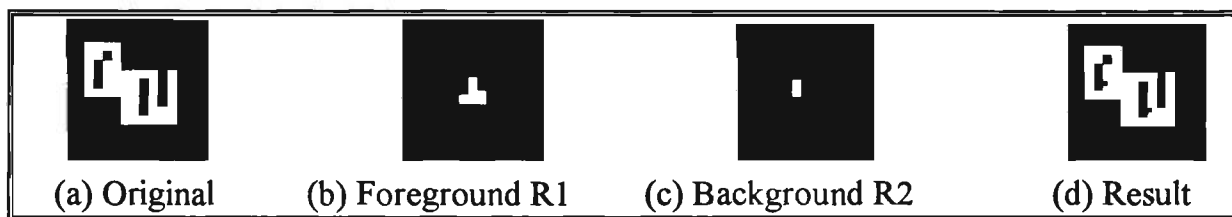


Fig. 4.10 Sample images for the thinning operation

Thickening

The thickening operation increases the size by filling the image points where the regions match the reference image pair $R = (R1, R2)$. Huang's expression for the thickening of an image X by an image pair $(R1, R2)$ is given in his Formula 4.9 :-

$$X \odot R = X \cup (X \ominus R) = X \cup (\overline{\overline{X} \oplus \check{R}1} \cup (X \oplus \check{R}2))$$

We may implement the operation in Smalltalk by means of the following instance method:-

```
thickeningBy:imageR1 and:imageR2
```

```
"Thickening instance method."
```

```
| a |  
a:= (Image width: (self width) height: (self height)) black.  
a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.  
a hitMissTransBy:imageR1 and:imageR2.  
self union: a
```

We may test thickeningBy: in the same way as we test hitTransBy: and: method.

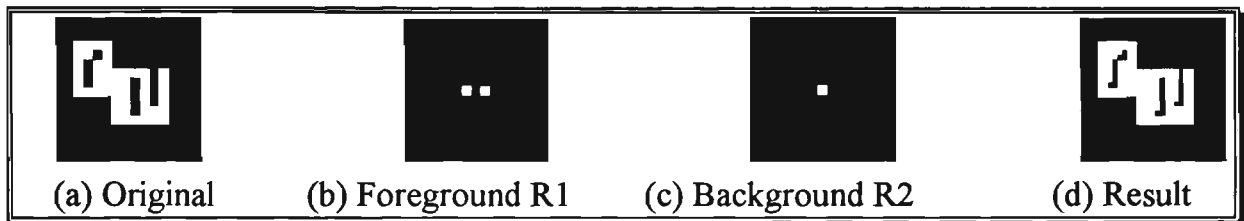


Fig. 4.11 Sample images for the thickening operation

4.4 Applications of image algebra in Smalltalk

With the preceding discussion as background, we can now give some practical applications of image algebra and their Smalltalk implementation. When dealing with binary images, the principal application of image algebra is extracting components that are useful in the representation and description of shape. In particular, we consider image algebra algorithms for finding boundaries, connected components and the convex hull. We also present some examples of image filters that are often used in conjunction with the above methods as pre- or post-processing steps. For convenience, we assign X , Y , R as globe variables in Smalltalk.

4.4.1 Application: filters

One kind of morphological low pass filter, to remove high frequencies in the foreground of an image, can be achieved by opening:-

```
X:= ImageDataBase idbFromDisk:'image.dbs'.  
Y:=X at:'noise'.  
Y displayAt: 0@20.  
R:=X at:'openOrcloseR'.  
R displayAt: 50@20.  
Y openingBy: R .  
Y displayAt: 100@20
```


The little white spots are considered as *noise* that we intend to remove. We use a 3-by-3 square reference image as the filter. We can get a *clean* image as follows:-

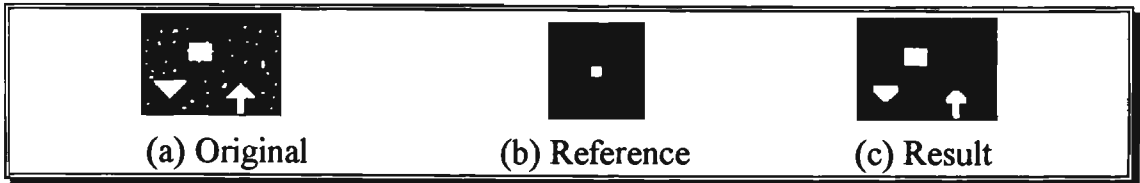


Fig. 4.12 Sample images showing noise removal

A second kind of morphological low pass filter can be achieved by closing:-

```
X:= ImageDataBase idbFromDisk:'image.dbs'.
Y:=X at:'openOrcloseX'.
Y displayAt: 0@20.
R:=X at:'openOrcloseR'.
R displayAt: 50@20.
Y closingBy: R .
Y displayAt: 100@20
```

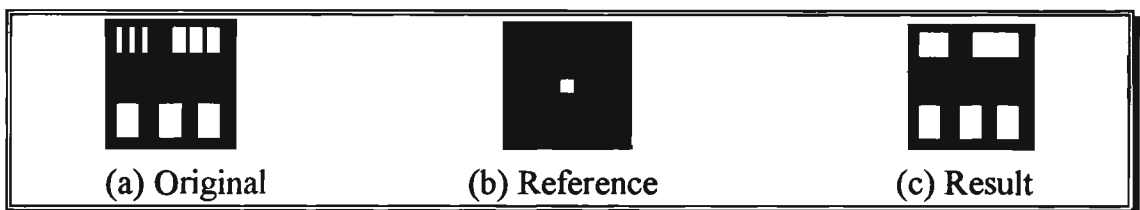


Fig. 4.13 Sample images for remove high frequencies

A morphological band-pass filter which removes low frequencies and high frequencies in the foreground of an image X can be achieved by the difference between its opening with a smaller reference image R , and its opening with a larger reference image Q , where R is a subset of Q :-

$$Y = (X \circ R) / (X \circ Q)$$

```
X:= ImageDataBase idbFromDisk:'image.dbs'.
Y:=X at:'openOrcloseX'.
Y displayAt: 0@20.
R:=X at:'openOrcloseR'.
R displayAt: 50@20.
Y openingBy: R .
X:= ImageDataBase idbFromDisk:'image.dbs'.
R:=X at:'openOrcloseX'.
Q:=X at:'openOrcloseQ'.
Q displayAt: 100@20.
R openingBy: Q.
Y difference: R.
```

Y displayAt:150@20

Fig. 4.14 shows removal of the high-frequency and low-frequency components from an image, leaving only the mid-frequency components

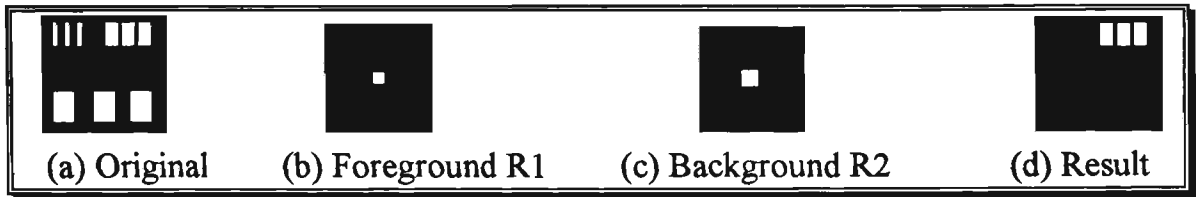


Fig. 4.14 Sample images for band pass filter

4.4.2 Application: shape recognition (template matching)

In one form of shape recognition we may use the hit or miss transform to recognise locations of foreground points given by $R1$, and locations of background points given by $R2$:-

```
X:= ImageDataBase idbFromDisk:'image.dbs'.
Y:=X at:'hitMissTrans1'.
Y displayAt: 0@20.
R:=X at:'triangle'.
R displayAt: 50@20.
X:=X at:'cap'.
X displayAt: 100@20.
Y hitMissTransBy: R and: X .
Y displayAt: 150@20
```

In Fig. 4.15 two up-triangles have been recognised and represented by two dots.

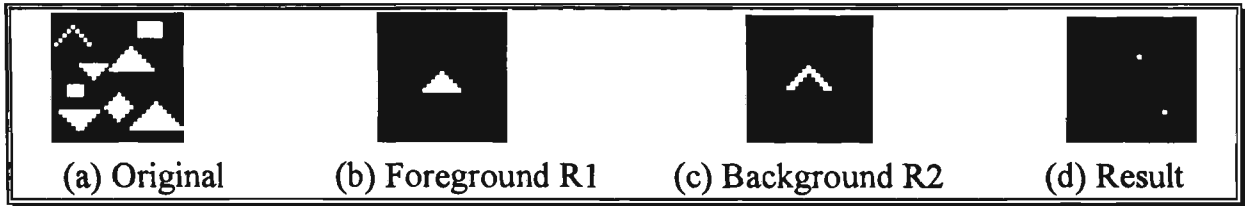


Fig. 4.15 Sample images for shape recognition

4.4.3 Application: edge detection

In a binary image, the edge is the boundary between white (foreground) area and black(background) areas. The detection of an 8-connected edge of an image X can be achieved by:-

$$X/(X \ominus N_8) = \overline{X} \cup (\overline{X} \oplus N_8)$$

The edge is detected by finding the difference between the original image and the original image eroded by a 3-by-3 8-connected reference image (N_8). An 8-connected edge is the boundary of two regions which have 8-connectivity (defined in 4.3.1 cited from Huang's definition). Here is an example of edge detection, the edge has been detected by using a 3-by-3 8-connected reference image (N_8):-

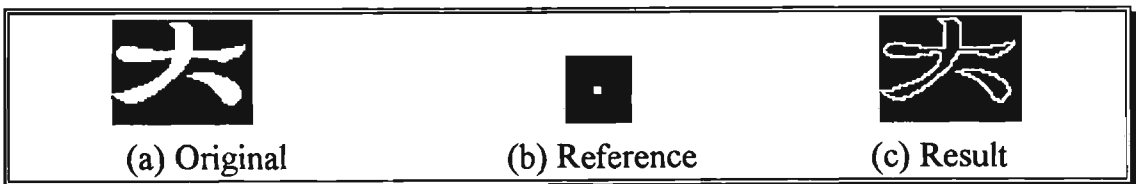


Fig. 4.16 Sample images of edge detection

Similarly, using N_4 instead of N_8 would give the 4-connected edge.

4.4.4 Application: convex hull

Azriel Rosenfeld and Avinash C. Kak defined *convex* and *convex hull* in their book *Digital Picture Processing*[ROR82]:-

In the real plane, S is called convex if any straight line meets S at most once, i.e., in only one run of points. Evidently, a convex set must be connected and can

have no holes; and an arc can be convex only if it is a strait-line segment. It is easily seen that each of the following properties is equivalent to convexity:

- For any points P,Q of S, the strait-line segment from P to Q lies in S.
- For any points P=(x,y), Q=(u,v) of S, the midpoint((x+u)/2, (y+v)/2) of P and Q lies in S.

In the real plane, there is a smallest convex set S_H containing any given set S (*Proof:* Readily, any intersection of convex sets is convex; in particular, the intersection of all the convex sets containing S is convex.) S_H is called Convex hull of S.

If there is only one object in the image X, or several objects separated by distances greater than their own diameters, then the convex hull is the intersection of projections:-

$$C(X) = \bigcap_{i=1}^4 (X \oplus \Theta_i^k)$$

where Θ_i , $i = 1.2.3.4$ are projections of horizontal, vertical, left-diagonal and right-diagonal. The superscript k should be greater than the longest radius of objects in X. For example, Θ_1^4 is the set $\{ (1, 0), (2, 0), (3, 0), (4, 0) \}$ and the horizontal projection of X is $X \oplus \Theta_1^4$. We might say that X has been smeared 4 pixels to the right and 4 pixels to the left. The following instance method may be introduced into the class Image in Smalltalk/V to find the convex hull of an image:-

convexHull:anInteger

```

    "Private -- Answer the convex hull of an image."

    | a b c d |
    a := Image width: self width
        height: self height
        planes: self planes
        bitCount: self bitCount .
    self displayAt:0@0 with: a pen.
    b := Image width: self width          "Create image working
space."
        height: self height
        planes: self planes
        bitCount: self bitCount .
    self displayAt:0@0 with: b pen.
    c := Image width: self width
        height: self height
        planes: self planes
        bitCount: self bitCount .
    self displayAt:0@0 with: c pen.
    d := Image width: self width
        height: self height
        planes: self planes
        bitCount: self bitCount .
    self displayAt:0@0 with: d pen.

```

```

    a dilationBy:( self builtH:anInteger).      "Project          to
horizontal                                     horizontal
direction."
    b dilationBy:( self builtV:anInteger).      "Project to vertical
                                                direction."
    c dilationBy:( self builtLD:anInteger).     "Project to left-
                                                diagonal
direction."
    d dilationBy:( self builtRD:anInteger).     "Project to right-
                                                diagonal
direction."
    ((a intersection:b) intersection:c) intersection:d.
    a displayAt:0@0 with: self pen.
    a release.
    b release.
    c release.
    d release.
    ^self

```

An example appears in Fig. 4.17:-

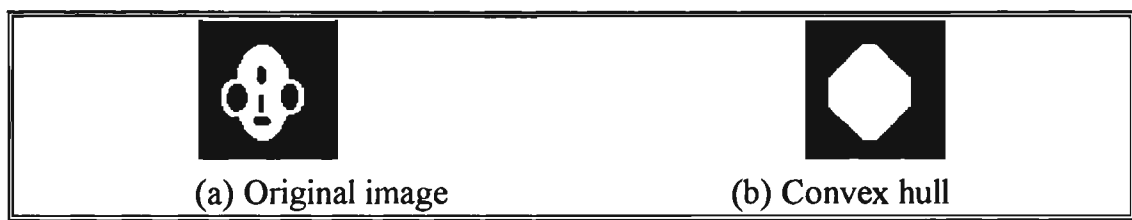


Fig. 4.17 Sample images of convex hull

Comparing an image X with its convex hull is a useful technique to analyse shape. The difference of the convex hull and the image [ROR82] X indicates how many concavities the image X has and what their individual shapes and sizes are.

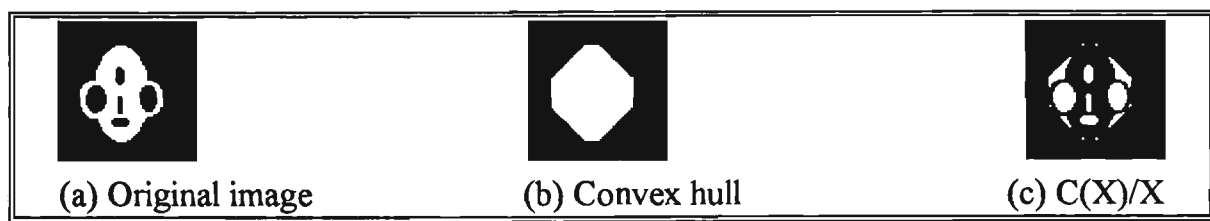


Fig. 4.18 Sample images of shape analysis by using convex hull

Chapter 5

Development of image-algebra applications in Smalltalk

Introduction

The core of this chapter is a description of the image processing environment which we have developed in Smalltalk.

The chapter begins with a brief overview of existing image processing packages available either commercially or from the public domain, and running under the most common operating systems (DOS, Windows, Macintosh and UNIX). We then introduce our own image processing environment in two versions:

- The DOS version (Smalltalk/V286) has two parts: the *Small-Image Database* and *Image Processor* ;
- The Windows version (Smalltalk/VWin), which we call *ImageLab*, includes not only binary morphological operations but also some image transforms and a simple graphics editor. To use the windows facility, we have built a user interface which includes a *tool bar*, a *status bar*, *on-line help* and a *Multi-Document Interface(MDI)*.

The chapter closes with some remarks on customising either version to one's own requirements.

5.1 Overview of image-processing applications

We begin with a brief overview of existing image processing packages available either commercially or from the public domain, and running under the most common operating systems (DOS or Windows on PC, Macintosh and UNIX). This information emerged from a survey conducted by the author over internet.

Software	Running Platform(s)	Written Language	User interface	Available from (Author/Supplier)
GlobeLab	MS-Windows	C	GUI(Windows)	Data Translation Inc.
Khoros	X-windows	C	GUI(X-windows)	ftp pprg.eece.unm.edu in pub/khoros/*
NIH Image	MAC	Think Pascal	GUI(MAC)	Wayne Rasband(author)
Morph 4.1	SUN-openWin	C	Command Line	R A. Peters II(author) rap2@vuse.vanderbilt.edu
Xlips V1.0	X-windows	C	GUI(X-windows)	L. Rosenthaler (author) rosenthaler@urz.unibas.ch
SCIDESK	MS-DOS	C, C++	Dos Menu-drive	JASP Research dsimkins@cscns.com
XITE	X-windows	C	Shell commands	ftp ftp.ifi.uio.no
Wang Image Cabinet	Muti-plateform	C	GUI(Windows)	Wang Coorporation
Image Version Library	SUN-OS	C++, interface C, Fortran	Command line	Silicon Graphics Inc.
Xcalibur	X-windows	C, PixScript	GUI(X-windows)	BDS Systems (703)4377651
Utah Raster Toolkit	X-windows	C	GUI(X-windows)	ftp cs.utah.edu in pub/urt-*
OPTMAS	MS-Windows	MS-C	GUI(Windows)	BioScan Incorporated

Table 5.1 A survey of current image-processing software

Khoros is a complete image-processing package, running on the Sun-Sparc workstation. It is an integrated software development environment for information processing and visualisation. *Khoros* features include a visual programming language, the ability to add new application packages to the system, an interactive user interface editor, interactive image display programs, surface visualisation, an extensive library

of image processing, numerical analysis and signal processing routines, and 2D or 3D plotting packages.

Globe-Lab Image is a commercial product which provides a standard Windows interface to a wide range of image-processing and image-analysis operations including object counting, measurement calibrated to user-defined real-world coordinates, Fourier analysis with editing in the frequency domain, morphological filtering, image enhancement etc.

NIH Image is a small but complete image processing for the Macintosh which is available in the public domain with its PASCAL source code. It supports many standard image processing functions, including contrast enhancement, density profiling, smoothing, sharpening, edge detection, median filtering, and spatial convolution with user defined kernels with size up to 63 by 63 pixels.

Wang's *Image Cabinet* is an image-database interface which emphasises image storage and retrieval from an existing database rather than image processing. It is available on many platforms.

Morph 4.0 is comprehensive in its coverage of 2-dimensional and 3-dimensional morphological operations with images. It is in the public domain, replete with the source-code, in C, for hundreds of sub-routines.

XLIPS VI.0 is an X-based image processing environment which is designed to help scientists to develop and implement their own routines. It is not a complete system; it is a framework furnished with low-level tools to facilitate the development of application-specific routines. There are three parts:

- display of multiple images using operations such as zoom, paste and contrast adjustment;
- a library of subroutines dealing with input, output, networking and command line parsing, etc;
- a set of routines which provide some image processing functions(eg. convolution, median filter, etc) which are programmed using the above mentioned library.

5.2 An image-processing system in Smalltalk/V286

Smalltalk/V286 is an MS-DOS version of a dialect of Smalltalk first marketed by Digitalk Inc in 1988. It offers an entry to Smalltalk via lower-performance machines such as the IBM-PC, PS/2 with an 80286 or 80386 processor. Smalltalk/V286 enabled the author to adapt quickly to the object-oriented paradigm and to begin creating, almost immediately, an image processing environment.

There are two essentially separate parts of the environment. One is an image-browser which we call the *Small-Image Database* which can store and retrieve binary images and colour images, the other we call the *Image Processor* which can perform binary morphological operations. Both share an image-dictionary file. The following figure gives the run-time structure:-

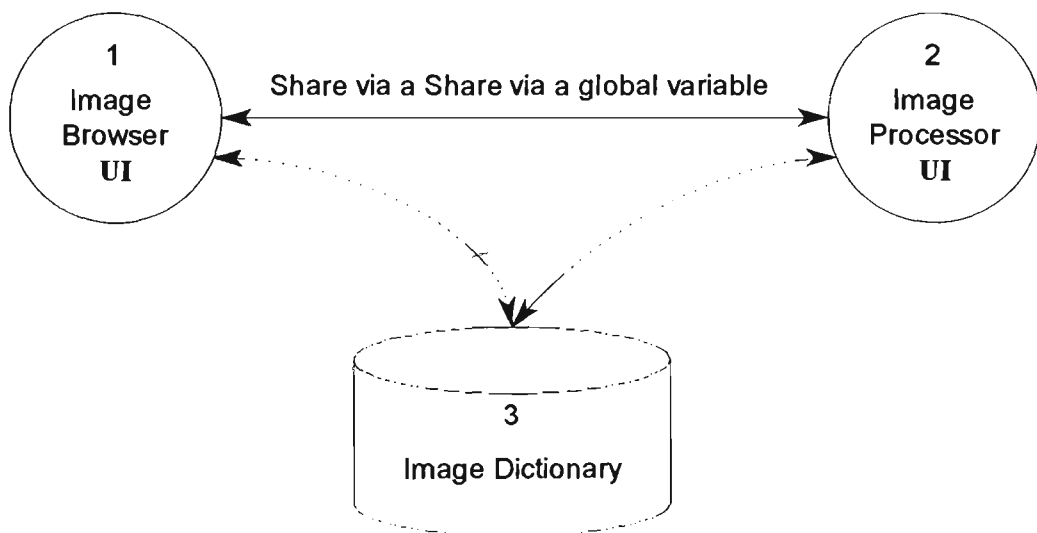


Fig. 5.1 The run-time structure*

5.2.1 The user interface of the application

There are two windows that permit the user to interact with computer. One is the interface to the Small-Image Database which allows the user to browse images. The other is the interface to the Image Processor.

* In the figure, UI stand for User Interface

The Small-Image Database allows the user to browse images stored against dictionary keys listed in the left pane, and to add and remove images from the dictionary. A selected image will be displayed in the right pane(Fig. 5.2).

The interface to the Image Processor presents the user with three upper panes above a fourth pane (the result pane). The first of the three panes holds the source image which can be loaded from the Small-Image Database, or from disk, or from the result plane. The remaining two upper panes each hold a reference image which may be loaded from either the disk or the Small-Image Database. A menu offers the user a choice of binary morphological operations to be applied to the upper images; the result of the operation appears in the lower pane(Fig 5.3).

The following figure is a sketch of the user interface of the Small-Image Database:-

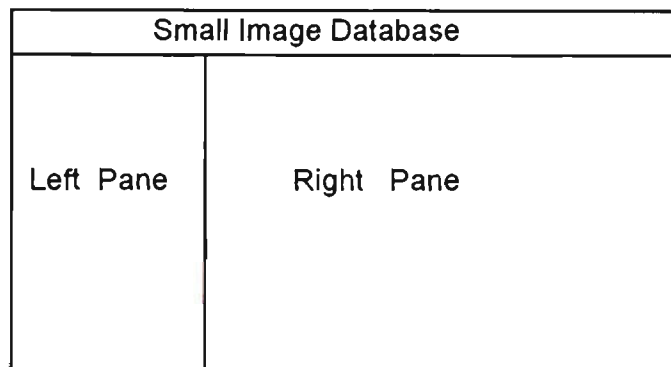


Fig. 5.2 The user interface of the Small-Image Database

Following Smalltalk we shall refer to the left pane as the *list pane* and to the right pane as the *instance pane*.

To build the Small-Image Database user interface in Smalltalk/V286, we inherit the V286's window user-interface system. We couple an instance of class GraphPane, which will display images, with an instance of class ListPane, which will list keys to the images. These two panes are organised by a top pane which is an instance of the class IdbTopPane.

The menu for the left-pane offers the following options:-

add from disk	Add a new image from a disk file .
bit editor	Enter bit editor environment .
free drawing	Enter free hand drawing environment.
inspect	Inspect the structure of the selected image.
remove	Remove the selected image.
save IDB	Save whole Small-Image Database as a disk file.

The menu for the right-pane offers the following options:-

clear	Clear the displayed image in the pane .
print	Print the selected image on a pin printer.
save as	Save the selected image as a disk file.

The user interface of the Image Processor has five components -- the top-pane containing the title, left pane, middle pane, right pane and bottom pane -- as shown in Fig 5.3 :-

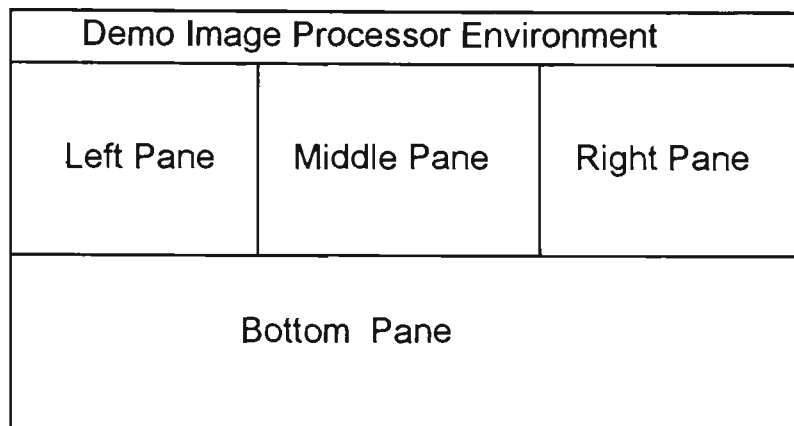


Fig. 5.3 The window of image processor environment

The left pane displays a selected original image X. The middle pane and the right pane display reference images that we refer to as R1 and R2 respectively. The bottom pane displays Z, the image resulting from a chosen operation on the images in the previous panes.

The menu for the left upper pane is:-

get it	Get X from a selected image in Small-Image Database.
load IDB	Load the Small-Image Database from disk and open the data base environment.
from result	Get the image from the result image. Then we can perform a sequence of morphology operations.
from disk	Get the image from disk file.

The menu for the middle upper pane is:-

get it	Get R1 from selected image in Small-Image Database.
from result	Get the image from the result image.
from disk	Get the reference image from disk file.

The menu for the right upper pane is:-

get it	Get R2 from selected image in Small-Image Database.
from result	Get the image from the result image.
from disk	Get the reference image from disk file.

The menu for the lower pane is:-

complement	Reverse X.
union	Form the union of the two images X and R1.
dilation	Dilate X by R1.
erosion	Erode X by R1.
difference	Get the difference between X and R1
reflect	Reflect X in the origin.
closing	Close X with R1.
opening	Open X with R1.
intersection	Form the intersection of X and R1.
symmetricDiff	Form the symmetric difference of X and R1.
hitMissTrans	Form the hit-and-miss transform of X with the image pair (R1, R2).
thicken	Thicken X with the reference image pair (R1, R2).
thin	Thinning X with the reference image pair (R1, R2).
print	Print the result-image on a pin printer.
save as	Save the result-image as a disk file.

The following method attaches the interface to the Small-Image Database:-

openOn: anObject

"Open an Inspector on anObject. Define the pane sizes and behaviour, and schedule the window."

```

object := anObject.
instPane := GraphPane new
    menu: #idbInsPaneMenu;
    model: self;
    name: #instance;;
    framingRatio: (1/3 @ 0 extent: 2/3 @ 1).

```

```

anIdbTopPane := IdbTopPane new.
anIdbTopPane
  label: 'Small-Image Database';
  model: anIdbTopPane dispatcher;
  menu: #workSpaceMenu;
  minimumSize: 80@80;
  yourself.
anIdbTopPane addSubpane:
  (ListPane new
    menu: #idbListMenu;
    model: self;
    name: #instVarList;
    change: #selectInstance;;
    returnIndex: true;
    framingRatio: ( 0@0 extent: 1/3 @ 1)).
anIdbTopPane addSubpane: instPane.
CursorManager normal change.
self setInstList.
anIdbTopPane dispatcher open scheduleWindow!

```

5.2.2 The class ImageDataBaseInspector

The class ImageDataBaseInspector inherits the properties of class DictionaryInspector. It is the basic class for the Small-Image Database (image dictionary). The class DictionaryInspector provides the basic mechanism for browsing the instance variables, the class variables and the instance methods of an object. To enable an Inspector to browse images contained in a dictionary, we need to create a class ImageDataBaseInspector as a sub-class of DictionaryInspector.

In Fig. 5.4, the left block is an illustration of the class ImageDataBaseInspector. The right block shows how a Small-Image Database has been created by sending the message:-

```
ImageDataBaseInspector new.
```

The following diagram shows the structure and functionality of the inspector:-

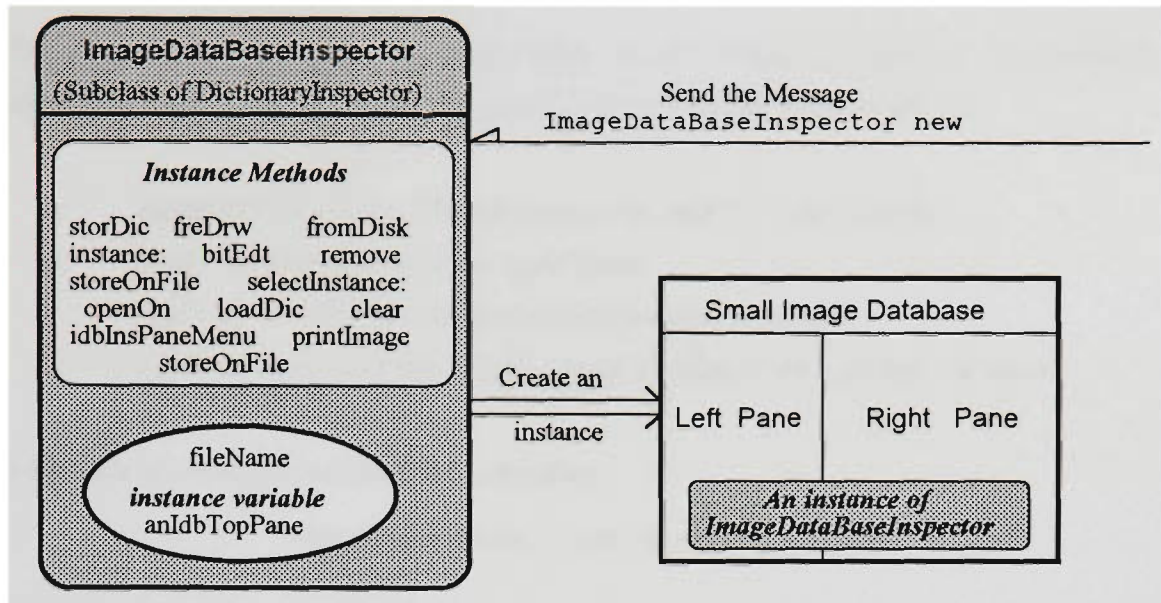


Fig. 5.4 The structure of class ImageDataBaseInspector

The instance methods of the ImageDataBaseInspector:-

bitEdt	Edit the image by using a bit editor.
clear	Clear the instance pane, that is, the image display pane.
freDrw	Draw the graph by using a free hand drawing tool.
fromDisk	Retrieve image from disk file, insert into data base.
idbInsPaneMenu	Pop up a menu on the right pane.
idbListMenu	Answer the list pane menu.
instance:anImage	Display the selected image on the instance pane.
loadDic	Retrieve whole image dictionary from disk.
openOn:anObject	Open an inspector window on an object, define the pane sizes and behaviour and schedule the window.
printImage	Print the selected image.
remove	Remove the selected key and value from the dictionary.
selectInstance:anInteger	Select the instance at index position anInteger in the list.
storDic	Store the image dictionary as a disk file.
storeOnFile	Store the selected image as a disk file.

5.2.3 The class ImageProcessor

The class ImageProcessor, the main class in the image processing environment, is created as a sub-class of Object. The tasks of an ImageProcessor are to:-

- create the window with one-top pane and four sub-panes;
- build the menu system for each pane;
- execute the relevant method of each menu selector;
- communicate with the Small-image Database via a global variable.

We create an ImageProcessor by executing

```
ImageProcessor new openOn:anObject
```

anObject is an image dictionary which is shared with a Small Image Database environment object.

The menu system is provided by the system's Menu class. The class Menu defines the protocol for an application to present a menu of items to the user, allows the selection of an item, and then takes some action based on the selection. There are many menus in the Smalltalk/V286 environment such as menu bars or pop-up menus which are hidden behind windows, panes and the system screen. In this version of the image-processor environment, the author uses pop-up menus.

In the class ImageProcessor, the three upper panes -- upper left pane, upper middle pane and upper right pane -- use the same instance method to communicate, via a global variable, with the class SmallImageDatabase. The global variable contains the index of the selected image in the SmallImageDatabase.

getIt

```
"To get an image from the Small Image Data Base."
```

```
instIndex:=Q.                "Globe variable (Q) for exchange data"
leftPane hasCursor ifTrue:[
indexSet at:1 put: instIndex.
^(self changed: #instanceLft:)].
middlePane hasCursor ifTrue:[
indexSet at:2 put: instIndex.
^(self changed: #instanceMid:)].
rightPane hasCursor ifTrue:[
indexSet at:3 put: instIndex.
^( self changed: #instanceRit:)]
```

The above code illustrates the use of a global variable in the class ImageProcessor, the value of the global variable Q being assigned to an instance variable instIndex. The

following code shows the setting of the value of the global variable Q in the class ImageDataBaselnspector:-

```
instance: anImage
"private:- to set the current instance index to a globe variable
for exchange data with other class."

Q:=instIndex.    "Global variable (Q) for exchange data"
instIndex isNil
ifTrue: [^(instPane clear)].
instPane clear.
^(object at: (instList at: instIndex) key)
    displayAt: instPane frame origin
    clippingBox: ((instPane frame origin)
    extent:( instPane frame extent))!
```

Figure 5.5 details the communication between the two classes via the global variable.

5.2.4 Adding methods to a system class

In object-oriented modelling, we usually identify an object, model its behaviour as methods, and represent its data as instance variables. We gather objects with identical behaviour and structure into a class. If several classes have sufficient commonality in behaviour and structure, we extract the commonality by making the classes subclasses of a superclass.

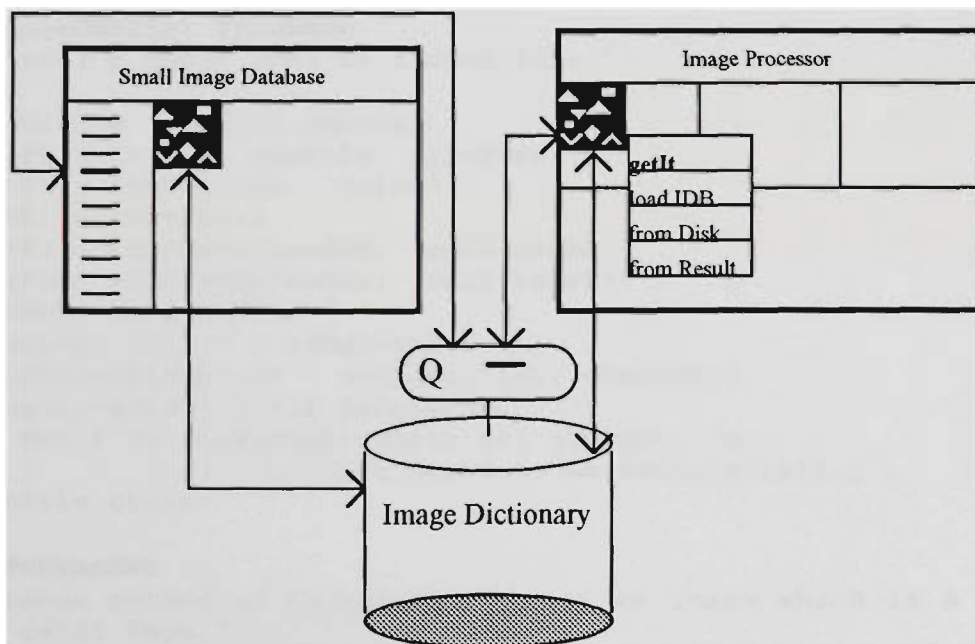


Fig. 5.5 Communication via a global variable Q

Often, however, we can adapt an existing class to accommodate our application objects. We can do this either directly by enhancing the class with additional methods

or indirectly by subclassing the class and adding variables and methods as required. As an example of the former, consider the instance-method `clear` is a method which we introduced into the system-class `GraphPane`:-

```
clear
  "Clears the instance pane "

  ^(Image new width:self frame
           width height:self frame height; white)
    displayAt: self frame origin.
```

As further examples we have the following instance methods which we introduced into the system-class `ColorForm` for storing and retrieving an instance of `ColorForm` :-

```
storeAsBkAndWt: fileName
"Stores an Colorform as a black and white form."

| anImage length aBitmap aWidth|
aWidth:=self width.
((aWidth\\16)=0 ) ifFalse:[
aWidth:=aWidth+(16 - (aWidth\\16)) deepCopy].
length:=aWidth*(self height)/8.
aBitmap:= Bitmap new:length.
1 to: length do:[:i|aBitmap at:i put:((( (bits at: 1 ) at: We )
                                     bitAnd: ((bits at: 2) at: i))
                                     bitAnd: ((bits at: 3) at: i))
                                     bitAnd: ((bits at: 4) at:
i))].
anImage:= Image new width:self width height:self height.
anImage bitmap: aBitmap.
anImage storeOnFile:fileName
```

```
storeColorOnFile: fileName
"Stroes a color form as a disk file."

| outFile length aWidth|
outFile := Disk newFile: fileName.
outFile nextPutAll: 'color'.
outFile nextPut:$ .
outFile nextTwoBytesPut: (self width) .
outFile nextTwoBytesPut: (self height) .
aWidth:=self width.
((aWidth\\16)=0 ) ifFalse:[
aWidth:=aWidth+(16 - (aWidth\\16)) deepCopy].
length:=aWidth*(self height)/8.
1 to: 4 do:[:aBitmap| (bits at: aBitmap) do:
[:ea | outFile nextPut: ea asCharacter].] .
outFile close.
```

```
changeToBkAndWt
"Instance method of ColorForm. Answer an image which is a derived
from a color form."
```

```
| anImage length aBitmap anArray |
length:=(self width)*(self height) deepCopy.
length inspect.
((length\\16)=0 ) ifFalse:[
length:=length+(16 - (length\\16)) deepCopy].
anArray:= self getBits:length.
```

```

aBitmap:= Bitmap new:length.
1 to: length do:[:i|aBitmap at:i put:(((( (anArray at: 1 ) at:
We )
                                bitOr: ((anArray at: 2) at: i))
                                bitOr: ((anArray at: 3) at: i))
                                bitOr: ((anArray at: 4) at:
i))].
anImage:= Image new width:self width height:self height.
anImage bitmap: aBitmap.
^anImage

```

colorFromFile: fileName

"Retrieve a color form from a disk file."

```

|inFile aBitmap1 aBitmap2 aBitmap3 aBitmap4 anArray w h anImage
wTemp head length|
inFile := Disk file: fileName.
inFile reset.
head:= inFile nextWord.
(head='color') ifFalse:[^nil].
inFile next.
h := inFile next asciiValue.
wTemp := inFile next asciiValue.
wTemp := (wTemp bitShift: 8) bitAnd:65280.
w := wTemp bitOr: h.
h := inFile next asciiValue.
wTemp := inFile next asciiValue.
wTemp := (wTemp bitShift: 8) bitAnd:65280 deepCopy.
h := wTemp bitOr: h.
anImage := (ColorForm new width: w height: h).
((w\\16)=0 ) ifFalse:[
w:=w+(16 - (w\\16)) deepCopy].
length:=w*h/8 deepCopy.
aBitmap1:= Bitmap new:length.
aBitmap2:= Bitmap new:length.
aBitmap3:= Bitmap new:length.
aBitmap4:= Bitmap new:length.
(1 to:length) do:[:i| aBitmap1 at: We put:(inFile next
asciiValue)].
(1 to:length) do:[:i| aBitmap2 at: We put:(inFile next
asciiValue)].
(1 to:length) do:[:i| aBitmap3 at: We put:(inFile next
asciiValue)].
(1 to:length) do:[:i| aBitmap4 at: We put:(inFile next
asciiValue)].
anArray:= Array with:aBitmap1
with:aBitmap2
with:aBitmap3
with:aBitmap4.
anImage bitmap:anArray.
^anImage

```

5.2.5 Limitations in Smalltalk/V286

Limitations in Smalltalk/V286 now began to loom as obstacles to further development of the image-processing environment. First, the programmer can not make a stand-alone version of his/her product, thus impeding the sharing of it with others. Second,

to process an image larger than 640k, the programmer must use an array of arrays rather than a single array. To avoid these limitations, we turn to another Smalltalk dialect -- Smalltalk/VWin.

5.3 ImageLab in Smalltalk/VWin

Microsoft Windows offers a graphical user-interface. Smalltalk/VWin 2.0 is an object-oriented programming environment running under Windows 3.x. To access the windows facility, the programmer can make an API call through the class KernelDLL. ImageLab is a user-friendly image processing environment developed in Smalltalk/VWin 2.0.

5.3.1 The Smalltalk/VWin environment

Object-oriented modelling involves recognising objects, defining their behaviour and delineating their interrelationships. The Smalltalk/VWin environment supports an incremental and evolutionary approach to such modelling.

A typical Smalltalk/VWin development environment includes a transcript window, one or more workspaces, a class-hierarchy browser and an online debugger. The transcript window is used by the system for various messages and can be used by the programmer to display output and code diagnostics. The multi-paned class hierarchy browser window shows the class inheritance within Smalltalk/VWin, and the structure and behaviour of each class; it also allows editing of the variables and methods associated with a class. The bulk of application development takes place within this window.

The online debugger has four sub-panes, it gives an expanded view of the Walkback window (a window pops up automatically when errors are detected) and provides a high-level debugging aid to help the programmer correct programming errors. Fig. 3.6 shows a typical VWin programming environment.

To apply VWin to image processing, we need to add some classes to the class hierarchy. The parts of the class hierarchy relevant to image processing are shown in Fig. 5.6.

Development of a typical Smalltalk/V application divides into six steps:-

- state the problem;
- draft the user interfaces that appear to the user;
- describe the objects;
- identify the classes;
- list the object interfaces;
- implement the methods.

We applied the language to develop an environment for processing images. Because it is image data which is to be manipulated in our application we define a class Image to store, in each of its instances, the structure and behaviour of an image. To create the graphical user interface we define the class ImageLab as a subclass of GraphLab which is, in turn, a subclass of ViewManager.

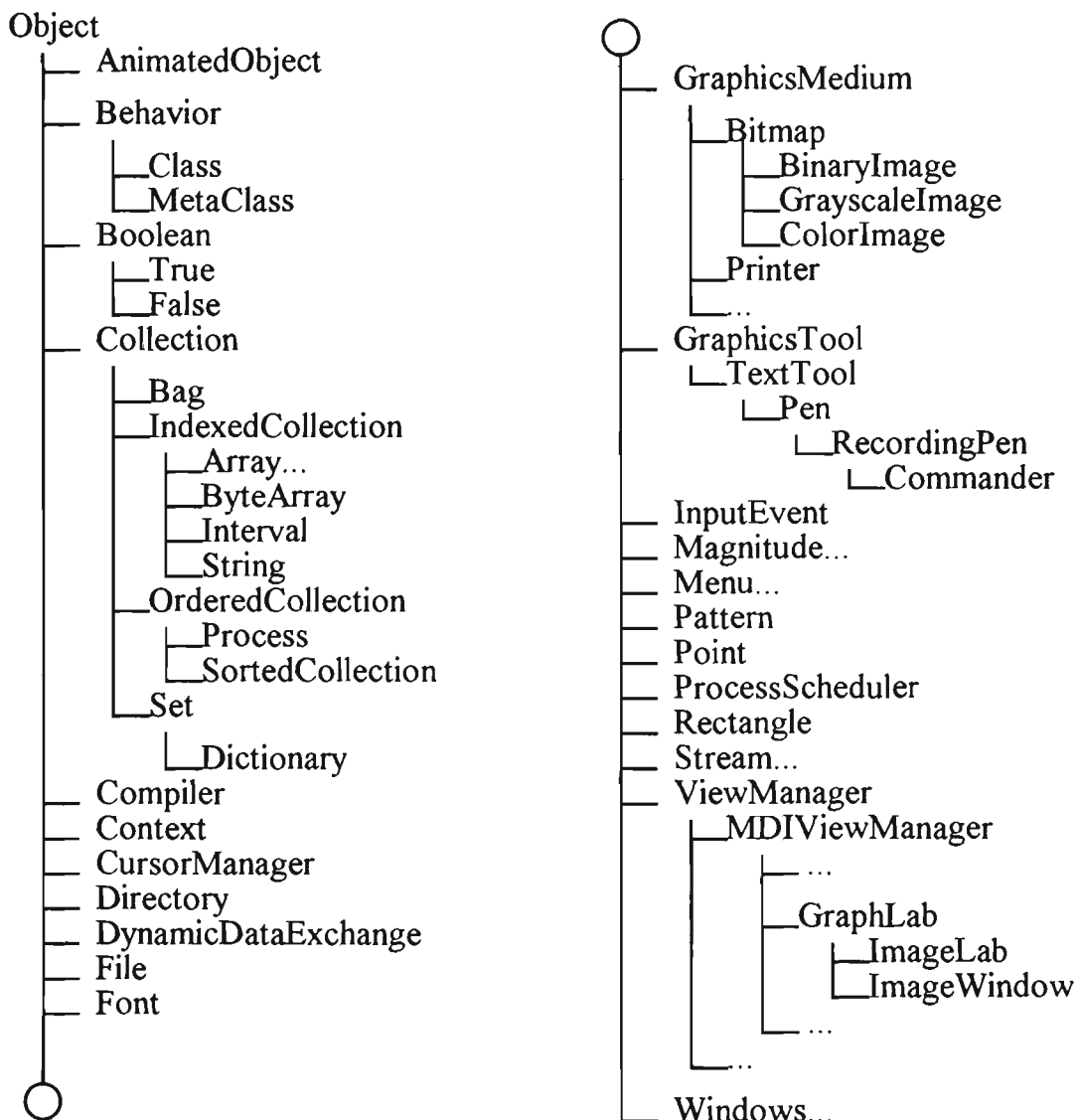


Fig. 5.6 Class hierarchy structure

5.3.2 The classes GraphLab and ImageLab

With the Microsoft Windows 3.x user interface, Smalltalk/V is able to take direct and powerful advantage of this graphical interface capability, allowing the user to program a Windows application from within an entirely object-oriented framework.

Of all the objects which populate the Windows world, the most important are windows. Windows are the *access points* between users and applications. Although it is the application which creates a window for user interaction, its behaviour and control is a cooperative effort between the application and the host system.

Much of the generic behaviour of a window -- its position on our display, its scroll bar action, resizing etc. -- is automatically handled by the host Window manager. Our main concern as a developer is in making a generic window appear for some special purpose by providing the information and interaction unique to the application.

In Smalltalk/VWin, an application with a graphical user interface typically involves three kinds of windows classes:-

- a subclass of *ViewManager* (such as *GraphLab*), which handles the interaction between panes, and the interaction between the user and the underlying model;
- *TopPane*, which is the top level window containing all the subpanes;
- a subclass of *SubPane*, which displays output and processes user input.

As already mentioned, the classes *GraphLab* and *ImageLab* are descendants of the class *ViewManager*. The class *ViewManager* has five major functions:-

- to remember the current state;
- to create panes;
- to provide the contents of panes;
- to carry out communication and synchronisation;
- to define menus for panes.

The classes *GraphLab* and *ImageLab* automatically inherit the above five functions. Additionally, the class *GraphLab* has the basic graphics editor functions such as drawing a line or circle, cut and paste etc.. The class *ImageLab*, a subclass of *GraphLab*, contains the morphological operations.

The class GraphLab has the following data structure:-

```
ViewManager subclass: #GraphLab
  instanceVariableNames:
    'pen state start previous penSize penColor pane '
  classVariableNames: ''
  poolDictionaries:
    'WinConstants ColorConstants '
```

The instance variable pen stores the graphics tool of current graphics pane, state records the pen state, start contains the current pen start position and pane points to current opened graphics pane.

The class ImageLab has the following data structure:-

```
GraphLab subclass: #ImageLab
  instanceVariableNames:
    'image seImage fileName anOc '
  classVariableNames:
    'MapDictionary '
  poolDictionaries:
    'ColorConstants WinConstants VirtualKeyConstants '
```

The instance variable image stores the current working image, selimage contains the current reference image(structure element), fileName holds the file name of the current working image and anOc holds the white pixel co-ordinates of the current reference image. The class variable MapDictionary contains the online help file address.

The instance methods that support ImageLab and GraphLab classes can be classified into three kinds:-

- methods for building the environment,
- methods to perform morphological operations,
- methods for graphics editing,

Methods for building the environment:-

open	Open an image-processing window
openTextWindow	Open a TextWindow application as an MDI document
aboveV	Open the dialogue box about Image Lab
activeImageProPane	Answer current active ImageProPane
childActivate:	Update the StatusPane, current working pane and its pen and show the label of the active MDI document.
childClose:	Update the StatusPane. If there is no more document, erase the StatusBox #status
close:	Close the receiver

Methods for creating the menu system:-

anaMenu	Answer a menu with a list of image-analysis operations
fileMenu	Answer the receiver's first menu title
mdiMenu:	Create the menuBar for the frame
morphMenu	Answer a menu with a list of morphology operations
optionsMenu	Answer a menu with a list of miscellaneous operations
otherMenu	Answer a menu with a list of other morphology operations
paletteMenu	Answer a menu with a list of palette changing operations

Methods to handle the status pane and the tool pane:-

toggleMenu: item:	Toggle the selected menu item
toggleStatusPane	Show/Hide the StatusPane
toggleToolPane	Show/Hide the ToolPane
toolPane:	Set the ToolPane contents
timer:	Update the time in the StatusPane

Methods to perform morphological operations:-

closing	Perform close operation of the loaded image with the selected structure element
complement	Answer the complement of the original image
convexHull	Answer the convex hull of the image
damond	Answer the diamond transform of the image
differ	Answer the difference of the two images
dilation	Dilate the loaded image with the selected structure element
edgeTrace	Trace the edge of the loaded image with the selected structure element
erosion	Erode the loaded image with the selected structure element
intersec	Answer the intersection of two images
opening	Perform open operation of the loaded image with the selected structure element
symeDif	Answer the symmetric difference of two images

The methods for graphics editing in class GraphLab are inherited by both class ImageLab and class ImageWindow. They are:-

display:	display current image on a user selected place
copyGraph	copy a portion of the receiver's contents to the clipboard and current image working buffer
pasteGraph	paste the graphics from the clipboard to the receiver
circle:	draw a circle
clear	erase current working pane to white
curve:	draw a curve
bitEdit	prompt the user for a rectangle and open a BitEditor on the bitmap associated with the rectangle
changeSize	change the pen size
ellipse:	draw a ellipse
freeDraw:	change to free hand drawing mode
line:	draw a strait line

The source code for these methods is given in the Appendix.

5.3.3 Multi-Document Interface

The multiple-document interface (MDI) is a user-interface standard for presenting and manipulating multiple documents within a single application. An MDI application has one main window, within which the user can open and work with several documents. Each document appears in its own separate child window within the main application window.

The main window of an MDI application is similar to that of most Windows applications. In an MDI application, the main window is called the "frame window". The frame window differs from a normal main window in that its client area is filled by a special child window called the "client window". Because Windows maintains the MDI client window and controls the MDI interface, the application needs to store very little information about the MDI user interface.

Smalltalk/VWin 2.0 provides the MDI classes as an extra feature. We can install it if we need. New classes that have been introduced are:

- MDIChild
- MDIClient
- MDIFrame
- MDIMenu
- MDITranscript
- MDISystem
- MDIViewManager

MDIChild class is responsible for creating MDI documents and manage its subpanes. It responds to the system MDI messages and allows applications to set event hooks.

MDIFrame class is responsible for creating MDI frame window and managing its subpanes. It responds to the system MDI messages and allows applications to set event hooks. Typical subpanes of an MDI frame window are MDIClient, StatusPane and ToolPane. The MDIFrame class creates the MDIClient subpane.

The owner of an MDIFrame class should be an MDIViewManager class. An MDIViewManager subclass is primarily responsible for provide the basic data structure and methods of multi-document interface.

By sub-classing the ImageLab under MDIViewManager, the open instance method creates the structure of ImageLab's MDI environment. The method for open in the class ImageLab is:-

```

open
    "Open an image Processing window."

    | hm |

self
    addView: (self frame:(
        MDIFrame new
            owner:self;
            labelWithoutPrefix: 'Image Lab';
            icon: (Icon fromModule: self resourceDLLFile
id:'face');
                when: #mdiMenuBuilt perform: #mdiMenu;;
                when: #childClose perform: #childClose;;
                when: #toggleKey perform: #toggleKey;;
                when: #close perform: #close;;
                when: #validated perform: #startTimer;;
                when: #timer perform: #timer;;
                when: #childActivate perform: #childActivate;;
                when: #menuBuilt perform: #menu:)).

self
    addSubpane:(
        ToolPane new
            owner: self;

```

```

        height: 27;
        when: #getContents perform:#toolPane:).
self
    addSubpane:(
        StatusPane new
            owner:self;
            when:#getContents perform: #statusPane:).

self class buildMapDictionary.

hm:=HelpManager
    for: self mainView
    title: 'ImageLab Help'
    file: 'c:\windows\help_il.hlp'
    dialogs: nil.

hm map: MapDictionary.

self openWindow.
self menuWindow
    removeMenu: (self menuWindow menuTitled: '&Color')

```

The above code creates the MDIFrame. To open a MDIChild window we need a separate class called ImageWindow. This class, parallel with class ImageLab, is also a sub class of GraphLab.

```

open
    "Open an MDI Child."

|child|
self addView: (child :=
    MDIChild new
        when: #activate perform: #activate;;
        style: WsMaximizebox |
            WsMinimizebox |
            WsThickframe |
            WsSystemmenu |
            WsCaption |
            WsOverlapped |
            WsClipsiblings |
            WsClipchildren;
        frame:self frame;
        owner:self;
        icon:(Icon fromModule:self resourceDLLFile
            id:'IMAGE_WORKSPACE');
        label:'Image Workspace';
        yourself).

child addSubpane:(
    pane := ImageProPane new
        owner: self;
        when: #getMenu perform: #modeMenu;;
        when: #getContents perform: #initPen;;
        when: #button1Down perform: #mouseDown;;
        yourself).
child openWindow

```

The following picture(Figure 5.7) is a sample of a typical ImageLab's multi-document interface.

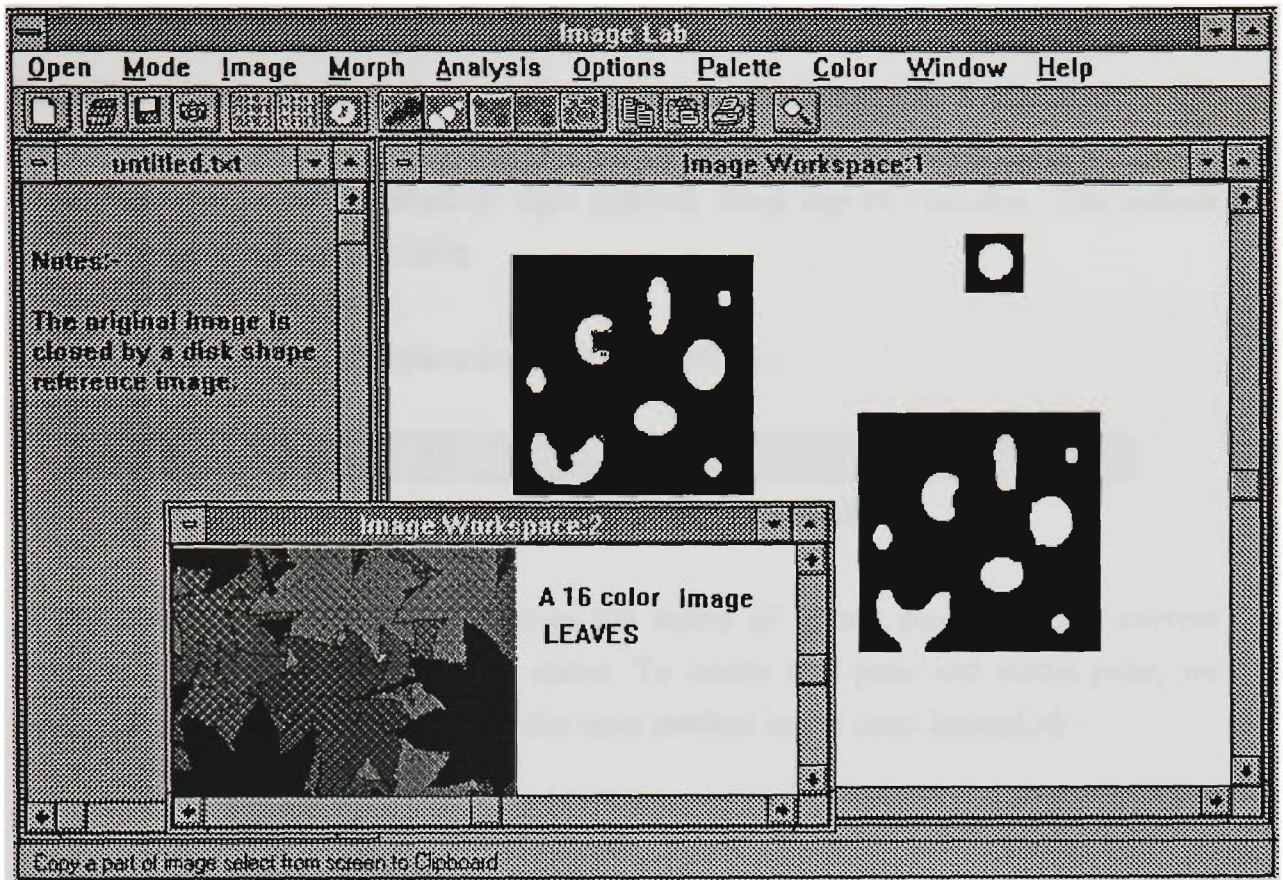


Fig. 5.7 A typical ImageLab working environment

5.3.4 Tool bar and status bar

The two major contributors to the graphic user interface are Tool Bar and Status Bar. They give the user a convenient way of accessing the system's property and display information reflecting the states of the application.

Class ToolPane provides the necessary methods for creating and using a ToolPane. ToolPane is a subclass of GroupPane and therefore inherits all the behaviour of SubPane. One creates a ToolPane in an application's open method just as one creates any subpane.

A ToolPane window uses an auxiliary class called Tool. One loads a ToolPane by sending the message contents: to it, together with an argument referencing a collection of instances of Tool.

Figure 5.8 is the tool pane inside the ImageLab:-



Fig. 5.8 Tool pane of ImageLab

StatusPane uses an auxiliary class called StatusBox. StatusPane handles a collection of StatusBox objects. A StatusBox object appears as a small box with a 3D effect.

StatusPane can be left justified or right justified, fixed size or resizable. The default style is left justified and resizable.

Figure 5.9 shows the status pane inside the ImageLab:-



Fig. 5.9 Status pane of ImageLab

In the ImageLab's status pane, it gives the status of cursor position, time, current working status and other keyboard status. To create tool pane and status pane, we need add following code included in the open method in the class ImageLab:-

```
self
  addSubpane:(
    ToolPane new
      owner: self;
      height: 27;
      when: #getContents perform:#toolPane:).
self
  addSubpane:(
    StatusPane new
      owner:self;
      when:#getContents perform: #statusPane:).
```

The detailed code to create tool pane and status pane are given in the attached Appendices (program code list).

5.3.5 On-line help system

The ImageLab environment has an on-line help system. The help system provides users with online information about an application. Basically, there are three main steps for developing an on-line help system:-

1. Write the help documents and save it as a Rich Text Format (a .rtf file).
2. Compile it by using a Help compiler either from Windows SDK or from Borland C++; it can be run as a stand alone file under MS-Windows environment.

3. Build it into the application.

The flow-diagram, Fig. 5.10 from *Tools -- Microsoft Windows Development Kit*, shows the general flow of work in the conception and development of the Help system.

There two steps to including Help in ImageLab. The first is to use any compatible text editor, to compose the help text into the help_il.rtf file. The second step is to create an instance of HelpManager in the open: method of the class ImageLab by inserting the following code before the message self openWindow.

```
hm :=HelpManager
    for: self mainView
        title: 'ImageLab'
        file: 'c:\windows\help_il.hlp'.
hm mapDictionary: aMapDictionary
```

The #for:title:file: message needs to be sent once for each view, passing that view as the parameter of *for:*. While each view of the ViewManager has a different HelpManager associated with it, the help file associated with each of them is the same.

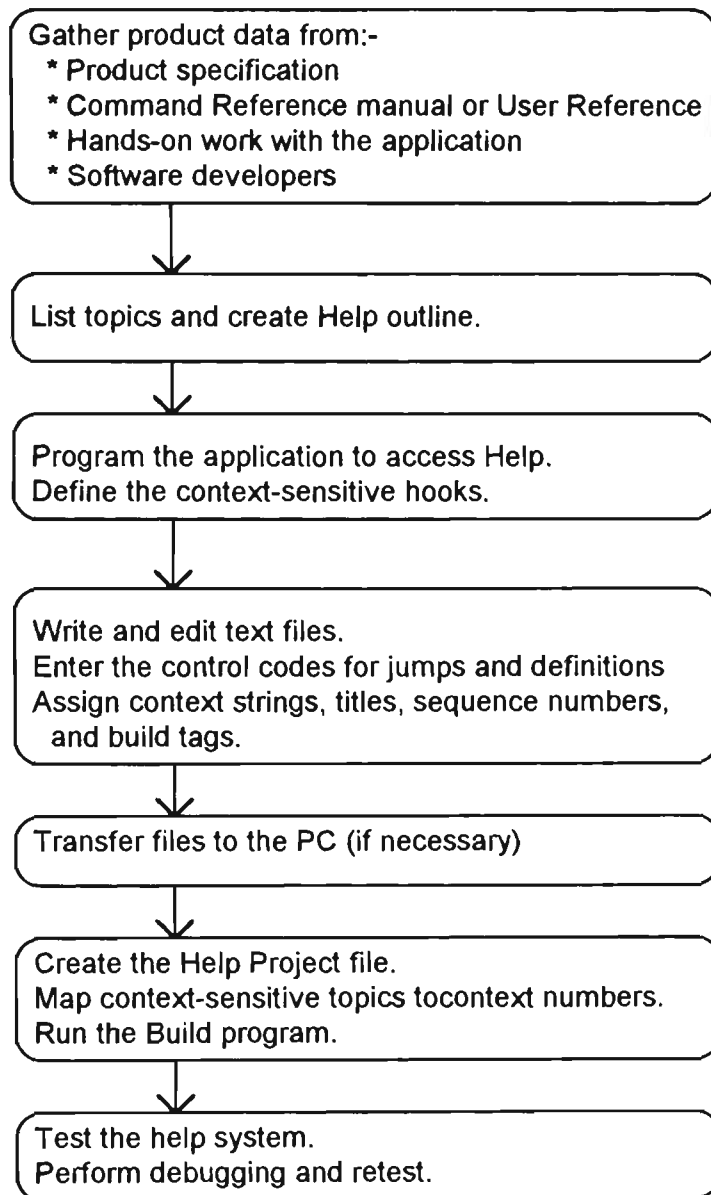


Fig. 5.10 Work flow diagram

Figure 5.11 shows the appearance of the on-line help system:-

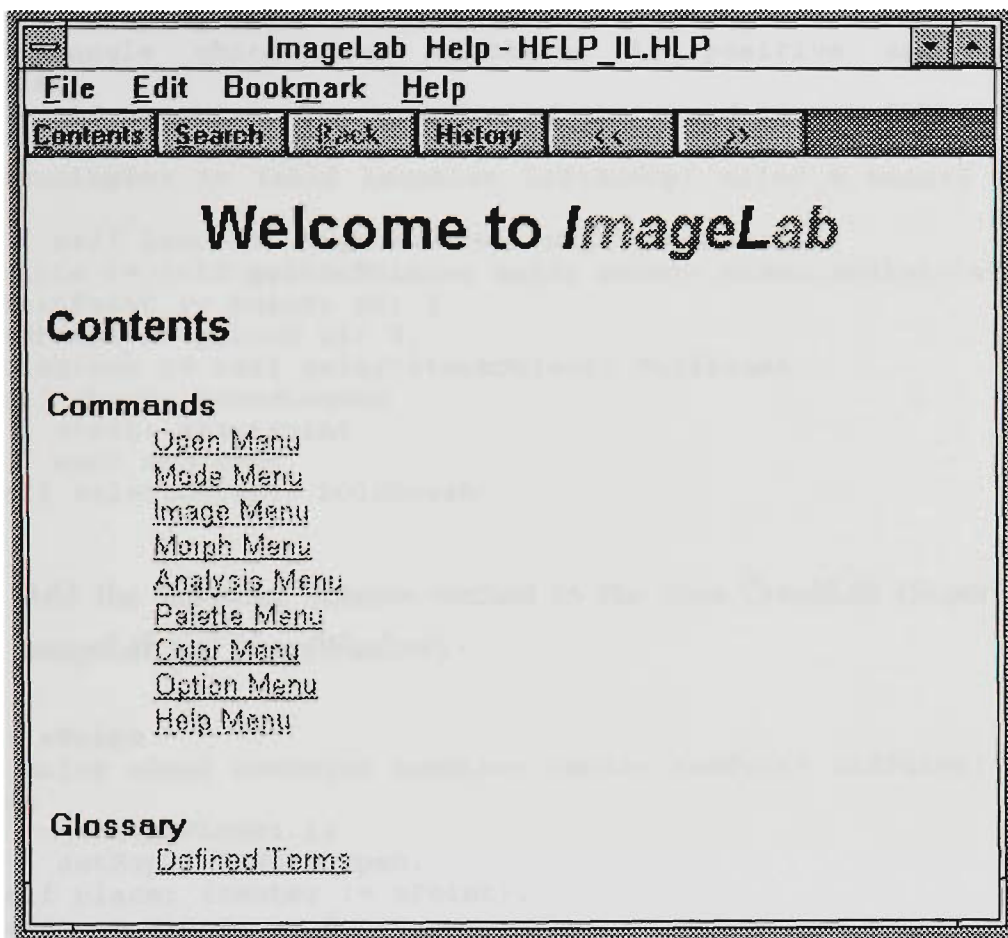


Fig. 5.11 Contents of ImageLab's on-line help system

5.4 Customised design

ImageLab, although less extensive than other image processing software, offers the advantage of being readily expandable. The extensibility flows not only from its object-orientation but also from its being written in Smalltalk. We can introduce a new function (such as a transform) into the environment simply by adding an instance method into the related class.

For example, suppose we wish to add a function to draw a *chord* line into the graphics editor inside the ImageLab application. We only need to perform following steps:-

1. Add the chord method into class Pen (subclass of GraphicsTool), this method is provided within the Smalltalk/VWin 2.0

```
chord: major minor: minor angles: aPoint  
    "Draw a chord whose major axis is major and minor
```

```

axis is minor. aPoint x
is the starting angle of the arc in degrees. aPoint y is
the
sweeping angle between the starting angle and ending
angle which goes clockwise if positive and counter-
clockwise
if negative. The interior of the chord is not filled."
| startPoint endPoint boundingBox hOldBrush points |
boundingBox := (self location leftAndUp: major @ minor) corner:
(
self location rightAndDown: major @ minor).
points := self getArcPoints: major minor: minor angles: aPoint.
startPoint := points at: 1.
endPoint := points at: 2.
hOldBrush := self selectStockObject: NullBrush.
self chord: boundingBox
start: startPoint
end: endPoint.
self selectObject: hOldBrush

```

2. Add the following instance method to the class GraphLab (Super class of ImageLab and ImageWindow):-

```

chord: aPoint
| major minor newMajor newMinor center newPoint oldPoint|
pen
setLineWidth: 1;
setRop2: R2Notxorpen.
self place: (center := aPoint).
major := minor := 0.
Notifier consumeInputUntil: [: event |
newPoint := self mouseLocation.
newMajor := (newPoint x - center x) abs.
newMinor := (newPoint y - center y) abs.
newPoint = oldPoint
ifFalse: [
pen
place: center;
chord: major minor: minor angles: aPoint;
chord: newMajor minor: newMinor angles: aPoint.
major := (newPoint x - center x) abs.
minor := (newPoint y - center y) abs.
oldPoint := newPoint].
event selector = #button1Up:}.
self reset.
pen chord: major minor: minor angles: aPoint.
pane backup pen chord: major minor: minor angles: aPoint

```

3. Introduce **chord** as a label in the draw menu and insert the symbol **chord:** into the list of selectors:-

```

modeMenu: aPane
"Answer a menu with a list of drawing modes."
aPane setMenu: (
(Menu
labels: ('draw\line\rectangle\circle\ellipse\chord
\curve\fill\text\erase')withCrs
lines: #())

```



```
        selectors: #(freeDraw: line: rectangle: circle: ellipse:
chord: curve: fillAt: text: erase:))
        selector: #drawingMode;;
        owner: self;
        title: '&Mode';
        checkItem: #freeDraw;;
        yourself)
```

To customise ImageLab one can begin by using ImageLab itself to experiment with various sequences of morphological operations, and then modify ImageLab to include the resulting new compound operation as a meun-item.

Chapter 6

Exploring gray-scale images

Introduction

Morphological concepts can be extended to gray-scale images. The binary morphological operations of dilation, erosion, opening, and closing are all naturally extended to gray-scale imagery by the use of a minimum or maximum operation. Nakagawa and Rosenfeld [NAKA78] first discussed the use of neighbourhood min and max operators. Peleg and Rosenfeld [PELE81] use gray-scale morphology to generalise the medial axis transform to gray-scale imaging. Werman and Peleg [WERM85] use gray-scale morphology for texture feature extraction.

The extension often leads to theoretical issues and to implementation complexities. When applied to a binary image, dilation and erosion operations cause an image to increase or decrease in spatial extent, respectively. Also, it is assumed that the objects and background are both relatively spatially smooth. Under these conditions, it is reasonable to ask: Why not just threshold the image and perform binary image morphology? The reason for not taking this approach is that the thresholding operation often introduces significant error in segmenting objects from the background. This is especially true when the gray-scale image contains shading caused by nonuniform scene illumination.

6.1 Ways of viewing gray-scale erosion and dilation

For morphological dilation, gray-scale dilation of f by b (f is the original image, b is an SE, which may be a binary image or a gray-scale image), denoted $f \oplus b$, is defined as [GONZ92]

$$(f \oplus b)(s, t) = \max\{f(s-x, t-y) + b(x, y) \mid (s-x, t-y) \in D_f; (x, y) \in D_b\}$$

where D_f and D_b are the domains of f and b , respectively. Similar to its binary operation counterpart, b is the structuring element of the morphological process but note that b is now a function rather than a set.

The condition that the displacement parameters $(s-x)$ and $(t-y)$ have to be contained in the domain of f is analogous to the condition in the binary definition of dilation, where the two sets had to overlap by at least one element.

To compute the defined gray-scale operation, we first translate the image f by $(-x, -y)$ for each (x, y) in the domain of b . We then add the value $g(x, y)$ to the translated image. Finally for the set of images obtained, create an image that has the maximum value at each pixel.

These three steps are shown in the following example [DOUGH92]. Consider the image:-

$$f = \begin{pmatrix} * & 0 & 2 & 2 & 2 & 1 \\ * & 1 & 2 & 6 & 2 & 1 \\ * & 0 & 6 & 7 & 2 & 1 \\ * & 1 & 1 & 6 & 1 & * \\ * & 1 & 0 & 2 & 2 & 1 \\ * & * & * & * & * & * \end{pmatrix}$$

and the structuring element:-

$$g = \begin{pmatrix} 0 & 3 \\ 3 & 4 \end{pmatrix}$$

In the matrices, each entry is the gray value of a pixel in the image, and the star (*) represents the non-interest pixel. After we have performed the first two steps, we get four transformed images. The following shows the formation of the maximum of four images and the resulting dilation:

$$f(x-0, y-1) + 0 = \begin{pmatrix} * & 0 & 2 & 2 & 2 & 1 \\ * & 1 & 2 & 6 & 2 & 1 \\ * & 0 & 6 & 7 & 2 & 1 \\ * & 1 & 1 & 6 & 1 & * \\ * & 1 & 1 & 6 & 1 & * \\ * & 1 & 0 & 2 & 2 & 1 \\ * & * & * & * & * & * \\ * & * & * & * & * & * \end{pmatrix}$$

$$f(x-0, y-0) + 3 = \begin{pmatrix} * & 3 & 5 & 5 & 5 & 4 \\ * & 4 & 5 & 9 & 5 & 4 \\ * & 3 & 9 & 10 & 5 & 4 \\ * & 4 & 4 & 9 & 4 & * \\ * & 4 & 3 & 5 & 5 & 4 \\ * & * & * & * & * & * \end{pmatrix}$$

$$f(x-1, y-1) + 3 = \begin{pmatrix} * & * & 3 & 5 & 5 & 5 & 4 \\ * & * & 4 & 5 & 9 & 5 & 4 \\ * & * & 3 & 9 & 10 & 5 & 4 \\ * & * & 4 & 4 & 9 & 4 & * \\ * & * & 4 & 3 & 5 & 5 & 4 \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \end{pmatrix}$$

$$f(x-1, y-0) + 4 = \begin{pmatrix} * & * & 4 & 6 & 6 & 6 & 5 \\ * & * & 5 & 6 & 10 & 6 & 5 \\ * & * & 4 & 10 & 11 & 6 & 5 \\ * & * & 5 & 5 & 10 & 5 & * \\ * & * & 5 & 4 & 6 & 6 & 5 \\ * & * & * & * & * & * & * \end{pmatrix}$$

$$\text{DILATE}(f, g) = \begin{pmatrix} * & 0 & 3 & 5 & 5 & 5 & 4 \\ * & 3 & 5 & 6 & 9 & 6 & 5 \\ * & 4 & 6 & 9 & 10 & 6 & 5 \\ * & 3 & 9 & 10 & 11 & 6 & 5 \\ * & 4 & 5 & 9 & 10 & 5 & 4 \\ * & 4 & 5 & 5 & 6 & 6 & 5 \\ * & * & * & * & * & * & * \end{pmatrix}$$

Haralick defined and computed gray-scale morphology in different ways [HARA92], all equivalent to the earlier maximum formula.

Basically, he introduced the concepts of the surface of a set and the related concept of the umbra of a surface. Suppose a set A in Euclidean N -space is given. We adopt the convention that the first $(N - 1)$ coordinates of the N -tuples of A constitute the spatial domain of A , and the N_{th} coordinate is for the surface. For ordinary gray-scale imagery, $N = 3$. The *top* or *top surface* of A is onto its first $(N - 1)$ coordinates. For

each $(N - 1)$ -tuple x , the top surface of A at x is the highest value y such that the N -tuple $(x, y) \in A$. If the underlying space is Euclidean, we can express this relationship by using the concept of supremum. If the space is discrete, we use the more familiar concept of maximum. Since we have suppressed the underlying space in what follows, we use maximum throughout.

Let $A \subseteq E^N$, and $F = \{x \in E^{N-1} \mid \text{for some } y \in E, (x, y) \in A\}$. The *top* or top surface of A , denoted by $T[A]: F \rightarrow E$, is defined by

$$T[A](x) = \max \{y \mid (x, y) \in A\}$$

A set $A \subseteq E^{N-1} \times E$ is an *umbra* if and only if $(x, y) \in A$ implies that $(x, z) \in A$ for every $z \leq y$.

For any function f defined on some subset F of Euclidean $(N - 1)$ -space, the umbra of f is a set consisting of the surface f and everything below the surface.

let $F \subseteq E^{N-1}$ and $f: F \rightarrow E$. The *umbra* of f , denoted by $U[f]$, $U[f] \subseteq F \times E$, is defined by:-

$$U[f] = \{(x, y) \in F \times E \mid y \leq f(x)\}$$

Haralick defined the gray-scale dilation of two functions as the surface of the dilation of their umbra:-

$$f \oplus k = T\{U[f] \oplus U[k]\}$$

The above refers to a one-dimensional image. It also works on the two-dimensional image by substituting the function f and k into two-dimensional functions.

Haralick also gives a reasonable way to compute it in hardware. The following theorem establishes that gray-scale dilation can be accomplished by taking the maximum of a set of sums. Hence gray-scale dilation has the same complexity as convolution. However, instead of doing the summation of products as in convolution, we perform a maximum of sums.

Let $f: F \rightarrow E$. Then $f \oplus k: F \oplus K \rightarrow E$ can be computed by using:-

$$(f \oplus k)(x) = \max \{ f(x-z) + k(z) \mid z \in K, x-z \in F \}$$

Gray-scale erosion is defined as [GONZ92]

$$(f \ominus b)(s, t) = \min \{ f(s+x, t+y) - b(x, y) \mid (s+x, t+y) \in D_f; (x, y) \in D_b \}.$$

In binary morphology, dilation and erosion are duals with respect to complementation followed by reflection. So we can compute erosion as follow:

$$f \oplus g = -[(-f) \ominus (-g^{\wedge})]$$

Here, $-f$ is the complement of f and g^{\wedge} is the reflection of g .

The expressions for opening and closing of gray-scale images have the same form as their binary counterparts [HARA87].

6.2 Pseudo gray-scale morphological operations

We tried using bit-block transfer to implement gray-scale morphological operations, hoping to treat gray-scale operation in the same way as the binary image. We use the word *pseudo* to distinguish our operations on gray-scale images from *functional* gray-scale morphology operations defined above.

Our pseudo gray-scale morphological operations are based on Smalltalk V/Win version 2.0 which uses the Windows GDI library. From the definition of gray-scale morphological operations, the result of a gray-scale dilation will be the **maximum** of Minkowski [MINK03] addition with the given structure element (gray-scale or binary, usually flap-top).

In implementing morphological operations on binary images in Smalltalk/V286 we were able to express the three basic operations in terms of logical operations on instances of the class `Image`. However, one of the basic operations for gray-scale images is `max` which is not directly available in the class `Bitmap` in Smalltalk/VWin. Exploration of ways of expressing `max` indirectly in terms of the available operations on instances of `Bitmap` did not prove fruitful. However, if we restrict the gray-scale levels to $2^n - 1$ we find that the operations `OR` and `max` are equivalent.

Consider a 5-level image, with levels selected from (0, 1, 3, 7, 15) that is, 2^n-1 .

Now, we can have a **OR** table:-

	0	1	3	7	15
0	0	1	3	7	15
1	1	1	3	7	15
3	3	3	3	7	15
7	7	7	7	7	15
15	15	15	15	15	15

From the table, we can see that **OR** is equivalent to the operation **maximum**. But for the 16 level image or 256 level image, the **OR** does not correspond with **maximum**. For example, 12 **OR** 6 is 15 not 12(the maximum of 12 and 6). Since the operation **OR** adds more 1's into the result, the result will not be less than the maximum of two numbers. For verification, execute the following Smalltalk code:-

```
| aP bM bA w h numBytes |
    bM:=OrderedCollection new.
    w :=256.
    h := 256.
    numBytes := w*h.
    bA := ByteArray new: numBytes.
        (1 to: w) do: [:i |
            (1 to: h) do: [:j| " Create an OR table "
                aP:=((j- 1) bitOr:(i - 1)).
                bA at:((i- 1)*256 + j) put:aP.
            (aP < (j- 1)) ifTrue:[ bM add:aP ].
            (aP < (i- 1)) ifTrue:[ bM add:aP ].
        ] ].
    bM inspect
```

The result will be an empty ordered collection, indicating that in no case does OR yield a result which is less than either of its operand.

We can set the colour table to give 256 gray levels with the following code:-

```
| colorTab |
colorTab := ByteArray new: 1024.
(0 to: 255) do: [:i |
    (1 to: 4) do: [:j |
        colorTab at: 4*i + j put: i
    ] ].
(0 to: 255) do: [:i | colorTab at: (4*i + 4) put: 0].
```

The Window's 8 bit colour fields are:-

RED	GREEN	BLUE	FLAG
0 - 255	0 - 255	0 - 255	0

The above script replaces the colour table of a bitmap with 256 shades of gray. For any shade each field is given the same value between 0 and 255..

The following is the spectrum of an 256-level image **OR** table, and we can compare it with the maximum table:-

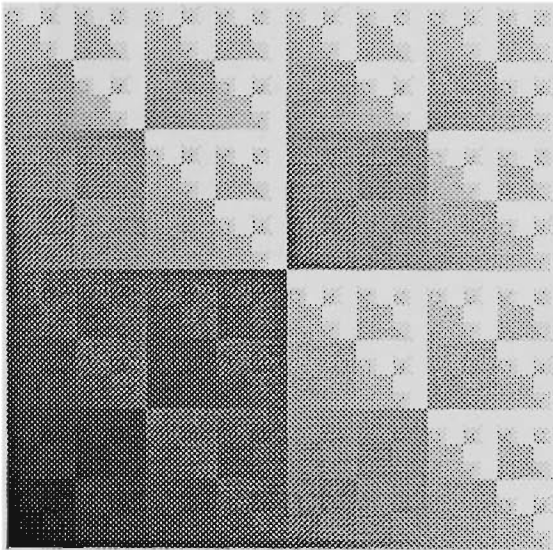


Fig. 6.1 OR table

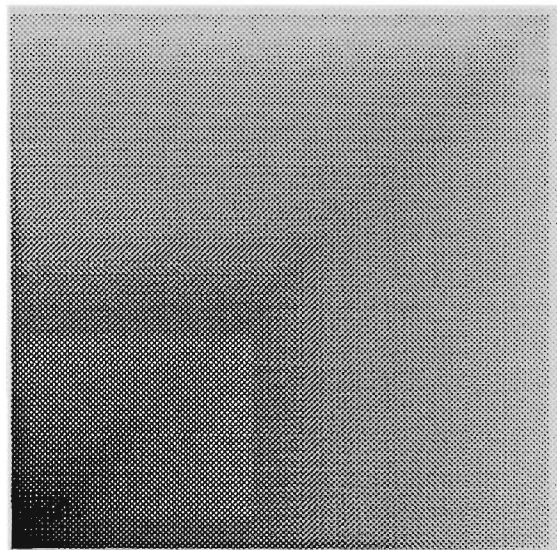


Fig. 6.2 MAX table

To compare the two tables, consider a point moving vertically from the bottom of one to its top. In the MAX table there is a uniform increase in brightness (that is, whiteness); for the corresponding vertical line in the OR table, the point is always at least as bright as the corresponding point in the OR table but can decrease as well as increase.

It would seem then, that if we wish to extend our work into gray-scale images, exploiting BitBlit within Smalltalk/VWin, then we must accept either the loss of information in reducing the number of levels to 5 or the loss of monotonicity in replacing MAX by OR.

The figures are examples of using pseudo gray-scale morphology to perform an edge detection on the test image LENA(512 by 512):-



Fig. 6.3 The original image

6.3 Three level gray-scale images

A binary image has pixels of two shades, black and white. We could use the term ternary image to denote a three-level monochrome image which has pixels of three shades, say, black, dark-red and red. Ultimately we wish to study monochrome

images with 16 (or 64, or 256) shades, and to generalise the morphological operations which we have been applying to binary images.

Ternary images offer a useful stepping stone in this direction. However, we may find that, like binary images, ternary images have an intrinsic role to play in image processing.



Fig. 6.4 The edge detected image

We begin with ternary images using the shades `ClrBlack`, `ClrDarkred` and `ClrRed` (the standard colors named in the global Dictionary `ColorConstants` of `Smalltalk/VWin`), and explore the copy operations (in the class `GraphicsTool`) using various logical rules. For example, construct the AND table:-

```
| bM colNum baseMap bigMap srcRect arr |  
Transcript pane cancel.  
arr := Array with: ClrBlack with: ClrDarkred with: ClrRed.  
bM := Bitmap screenWidth: 32 height:32.  
baseMap := Bitmap screenWidth: 96 height: 32.
```

```

(0 to: 2) do: [ :i |
    bM pen fill: (arr at: i + 1).
    bM displayAt: 32*i@0 with: baseMap pen. ].
baseMap displayAt: 50@0 with: Display pen.
srcRect := 0@0 extent: 32@32.
bigMap := Bitmap screenWidth: 96 height: 32.
(0 to: 2) do: [ :j |
    bM pen fill: (arr at: j + 1).
    bM displayAt: 0@(j*32 + 50) with: Display pen.
    bigMap pen copyBitmap: baseMap from: (baseMap boundingBox)
at: 0@0 rule: Srccopy.
    (0 to: 2) do: [ :i |
        bigMap pen copyBitmap: bM from: srcRect at: 32*i@0
rule: Srcand. ].
        "AND"
        bigMap displayAt: 50@(j*32 + 50) with: Display pen.
    ] .
bM release.
bigMap release.
baseMap release

```

We can construct the OR table by using Srcpaint rule instead of Srcand rule on above code.

We also experimented with other operations (they are listed on the global Dictionary WinConstants) by using above code:-

```

Srccopy, Srcerase, Srcinvert, Notscrcopy, Notsrcerase,
Dstinvert, Mergepaint.

```

From the RGB values of the shades, we find that AND corresponds to MIN and that OR corresponds to MAX (prelude to gray-scale morphology).

We also can use other triplets of colours and find out the equivalence of MAX/MIN with AND/OR:-

```

arr := Array with: ClrBlack with: ClrDarkred with: ClrRed.
arr := Array with: ClrBlack with: ClrDarkgreen with: ClrGreen.
arr := Array with: ClrBlack with: ClrDarkblue with: ClrBlue.
arr := Array with: ClrBlack with: ClrBrown with: ClrYellow.
arr := Array with: ClrBlack with: ClrDarkpink with: ClrPink.
arr := Array with: ClrBlack with: ClrDarkcyan with: ClrCyan.
arr := Array with: ClrBlack with: ClrDarkgray with: ClrWhite.
arr := Array with: ClrBlack with: ClrPalegray with: ClrWhite.

```

Chapter 7

An application of ImageLab

Introduction

The purpose of this chapter is to show ImageLab *in action* by describing how the author used ImageLab to explore the counting of 'blobs' in a binary image.

We begin with a description of the blob-counting problem; we then discuss a classical algorithm as a preface to presenting an alternative method for counting blobs. Finally, we show how the method might be applied to the detection of clusters of microcalcifications in mammograms.

7.1 Labelling connected components in binary image

Separation of objects from their background is a major problem in pattern recognition and image analysis [ROSE76]. In the case of grey scale pictures, it is commonly referred to as segmentation. In the simpler case of binary pictures, it is known as detection of connected components.

Detection of connected components in binary pictures is an indispensable step in such applications as automatic visual inspection, optical character recognition, extraction of karyotypes from photomicrographs of mitotic cells, robot vision, facsimile coding systems, etc. In some industrial applications we need to count 'blobs', as in tablet packaging or cluster counting in digitised mammograms.

Blob counting algorithms vary according to the input images. The more complicated the input image, the longer and more complex is the algorithm. In this chapter, we

introduce an alternative method for counting blobs which uses morphological operations. This removes some restrictions on the input image and also reduces the counting time.

7.1.1 The classical method

Here we demonstrate how do we implement a modified classical blobs counting algorithm in Smalltalk. The basic idea of the algorithm is to walk over the image from left to right and from top to bottom. For each pixel, we determine if it is the same as the one above or the one to the left. If it is the same as one of those then mark it as the same blob. If its upper neighbour is different from its left neighbour, then we change its left neighbour into the same as its upper neighbour.

The original algorithm is described in Dana H. Ballard & Christopher M. Brown's book *Computer Vision* p151[BALL82]; they use the term *blob colouring*; here is the pseudo code of their algorithm:-

Algorithm: Blob Colouring

Let the initial colour, $k=1$. Scan the image from left to right and top to bottom.

```
If  $f(X_c)=0$  then continue
else
begin
```

```
if ( $f(X_u)=1$  and  $f(X_L)=0$ )
then colour ( $X_c$ ) := colour( $X_u$ )
```

```
if ( $f(X_L)=1$  and  $f(X_u)=1$ )
then begin
    colour ( $X_c$ ) := colour( $X_L$ )
```

```
comment: colour ( $X_L$ ) is equivalent to colour ( $X_u$ )
```

```
end
```

```
comment: two colors are equivalent.
```

```
if ( $f(X_L)=0$  and  $f(X_u)=0$ )
then colour ( $X_L$ ) :=  $k$ ,  $k:=k+1$ 
```

```
comment: new colour
```

```
end
```

To perform the above algorithm, we need the following template:-

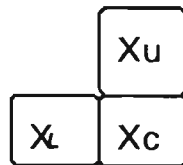


Fig. 7.1 J shaped template for blob colouring

To implement the above algorithm in Smalltalk, we first define a class Matrix as a subclass of Array. In defining the matrix we could make it a subclass of Array, or make it a subclass of Object with an instance variable holding an array. After choosing the former we found that Duško Savic [SAVI90] and Carleton University Smalltalk group [DIGI88] had donelike wise. Next, we represent an image numerically, with 1s and 0s, as a matrix. Application of the the labelling algorithm replaces the 1s by characters, with all entries in a connected component receiving the same character.

The data structure of class Matrix is as follow:-

```
Array variableSubclass: #Matrix
  instanceVariableNames:
    'rows columns pivotValue '
  classVariableNames: ''
  poolDictionaries:
    'CharacterConstants '
```

The class Matrix currently has twenty four instance methods and two class methods. The instance methods which we used for connected component labelling include:-

at: i at: j	Access matrix element in (i,j)
at: i at: j put: k	Assignment for matrices
columns	Answer the number of columns in matrix
columns: anInt	Force a new number of columns
eNbr: j at: i	Answer the east neighbour of element (i,j)
eNbr: j at: i put: k	Replace the east neighbour of element (i,j) with k
loc: i with: j	Locate the (i,j) matrix element in the matrix
nNbr: j at: i	Answer the north neighbour of element (i,j)
nNbr: j at: i put: k	Replace the north neighbour of element (i,j) with k
nUp: j at: i	Fill the north neighbours with element at (j,i) untill a zero value is met
rows	Answer the number of rows in matrix

rows: anInt	Force new number of rows
sNbr:j at:i	Answer the south neighbour of element (i,j)
sNbr:j at:i put:k	Replace the south neighbour of element (i,j) with k
wBack:j at:i	Fill the west neighbours with element at (j,i) until a zero value is met
westNorth:j at:i	Fill the west neighbours with north neighbour's value until a zero is met
wNbr:j at:i	Answer the west neighbour of element (i,j)
wNbr:j at:i put:k	Replace the west neighbour of element (i,j) with k

This algorithm can not handle U-shape or W-shape blobs. It will count a U-shape blob as two blobs and a W-shape blob as three blobs. We improved the algorithm by adding *back-track*. The improvement that we made involves backtracking at each point by using a L-shaped trace template:-

Improved Algorithm: Connected Components Labelling

Let the initial colour, $k=1$. Scan the image from left to right and top to bottom.

```
If  $f(X_c)=0$  then continue
else
begin
```

```
  if ( $f(X_u)=1$  and  $f(X_L)=0$ )
  then begin
    colour ( $X_c$ ) := colour( $X_u$ )
    while colour( $N(X_{L-1})$ ) != 0 do
      colour( $N(X_{L-1})$ ) := colour( $X_u$ )
    end while
  end
```

```
  if ( $f(X_L)=1$  and  $f(X_u)=1$ )
  then begin
    colour ( $X_c$ ) := colour( $X_L$ )
    colour ( $X_L$ ) is equivalent to colour ( $X_u$ )
    begin
      while colour( $X_{L-1}$ ) != 0 do
        [colour( $X_{L-1}$ ) := colour( $X_u$ )
          while colour( $N(X_{L-1})$ ) != 0 do
            colour( $N(X_{L-1})$ ) := colour( $X_u$ )
```

comment: $N(X_{L-1})$ is north neighbour of X_{L-1} .

```
          end while]
    end while
  end
end
```

comment: two colors are equivalent, set three colour same as original north neighbour colour and fill all non-zero left

value same as original north neighbour. Then the U-shaped and W-shaped blob can be counted as one blob.

```
if (f(XL)=0 and f(Xu)=0)
then colour (XL):=k, k:=k+1
```

```
comment: new colour
end
```

For the backtrack, we use the following template at each point:-

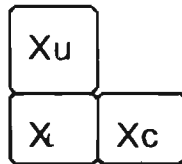


Fig. 7.2 L shaped template for blob colouring

Armed with the class Matrix, we may implement the above algorithm by using following Smalltalk code:-

```
label: aColor
  "This is the method for labeling specified color blobs."

  |label n row column|
  n:=1.
  row:=self height.
  column:=self width.
  label:=Matrix new:row with:column.

  "Obtain a Matrix object to mapping image"

  0 to: (row- 1) do:[:j|
    0 to: (column- 1) do:[:i|

      i:=i+1. j:=j+1.

      ((self getPixel:((i- 1)@(j- 1)))= aColor) ifTrue:[

        (((label wNbr:j at:i)=0)and:[(label nNbr:j at:i)=0])
ifTrue:[
          label at:j at:i put:n.
          n:=n+1. "New color" ].

        (((label wNbr:j at:i)~=0)and:[(label nNbr:j at:i)=0])
ifTrue:[
          label at:j at:i put:(label wNbr:j at:i).

          "Has left neighbour, but no upper neighbour"

        (((label wNbr:j at:i)=0)and:[(label nNbr:j at:i)~=0])
ifTrue:[
          label at:j at:i put:(label nNbr:j at:i).
          label nUp:j at:i ].

          "Has upper neighbour, but no left neighbour"

        (((label wNbr:j at:i)~=0)and:[(label nNbr:j at:i)~=0])
ifTrue:[
          label at:j at:i put:(label nNbr:j at:i).
```



```

label westNorth:j at:i].

"Has both upper and left neighbour, set three color
same as upper neighbour color and fill all non-zero
left value same as upper neighbour."

] ifFalse:[
label at:j at:i put:0.           ].
i:=i- 1. j:=j- 1.
]
^label

```

To locate the blobs in the image, we send the message `locate` to the matrix representing the image. The instance method `locate` of class `Matrix` returns an ordered collection which contains the numbers of blobs and the positions of the first upper left point in the related blob. The return value is follow:-

count	value1	point1	value	point	...	valueLas	pointLast
			2	2		t	

Here `count` is the number of counted blobs, `valueN` is the labelled number and `pointN` is the first upper left point in the related blob. The code of `locate` as follow:-

locate

"Locate the position of same value non-zero group in a matrix.(first from left-top, related to left-top corner."

```

| n e count|
n:= OrderedCollection new.
count:=0.
1 to: (self rows) do:[:i|
  1 to: (self columns) do:[:j|
    e:=self at:i at:j .
    (e~=0) ifTrue:[
      (n includes:e) ifFalse:[
        count:=count+1.
        n add:e.
        n add:(j@i)]
    ]].
n addFirst:count.
^n

```

7.1.2 The morphological method

The algorithms we discussed above are based on pixel-level operations. They are time consuming if we just need to count the number of blobs in the image. Mathematical morphology is a form of mathematics for analysing and describing shapes. It treats images as sets of points in space (rather than as arrays of numbers or as connected

blobs). Because it treats images as sets, the operations for combining two images are set operations, rather than arithmetic ones.

The basic idea of using mathematical morphology to count blobs is to transform the blobs into dots, i.e. each blob is represented by a single point. Then, after the transformation, the numbers of white pixels in the entire image equals the number of blobs in the original. Gasperi[GASP86] employed a D-type skeleton to perform the transformation. His method repeatedly applies the morphological thinning operation to the original image by using a set of templates called D-type skeleton templates, until the blobs become single points. There are two shortcomings in his algorithm. On the one hand, all his operations are based on a hexagonal co-ordinate system and he used special hardware because he wants to achieve the designed speed. On the other hand, there is a restriction on the input image - the region must be continuous without holes.

Here we introduce *another* operation to transform the blobs to a single points. Because it gives blobs a diamond-like shape we call it the diamond transform. It will eliminate the holes inside the blobs and fill the cavities of the U-shape or W-shape blobs. The diamond transform is expressed by the following formula:-

$$D(X) = \bigcap_{i=1}^2 (X \oplus \Theta_i^k)$$

where Θ_i , ($i=1,2$) are the projections of the left-diagonal and the right-diagonal. The upper script k in the formula is greater than the size of the smallest blobs. The following example shows the effect of a diamond-shape transform on an image:-



Fig. 7.3 Original image

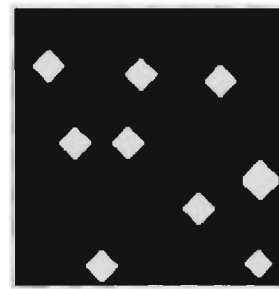


Fig. 7.4 D-transformed image

In the above example, we can see that the blobs acquire a diamond-like shape, and that the holes and cavities in the blobs have been removed.

After applying the diamond transform, we can employ another morphological operation, the *hit-or-miss transform*, to convert the diamonds into single points. The hit-or-miss transform of an image pair $R=(R1,R2)$ is used to match the shape (or template) defined by the reference image pair R where $R1$ defines the foreground of the shape and $R2$ defines the background of the shape. It is a basic tool for shape detection. It can be expressed by the following formula:-

$$X \ominus R = (X \ominus R1) \cap (\bar{X} \ominus R2) = \overline{(\bar{X} \oplus \bar{R}1) \cup (X \oplus \bar{R}2)}$$

The operation \oplus (dilation) is one of the fundamental morphological operations. For diamond-like blobs we can use a triangle as the foreground reference image and an inverted v (^) as the background image. The template pair is as follows:-

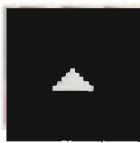


Fig. 7.5 Foreground reference image



Fig. 7.6 Background reference image

The following figure shows the transformation of the original image to single points.

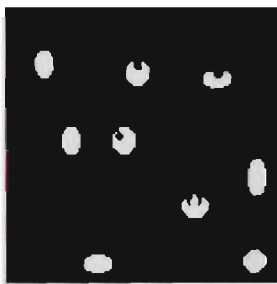


Fig. 7.7 Original image



Fig. 7.8 Transformed image

The only restriction on the input image is that the blobs inside the input image are of similar size or classified into several size groups. If the blobs in the input image differ in size we may employ a *band filter* to extract the blobs of the size of interest. The following example shows how we use a band-filter to extract blobs of a required size. Let X denote the original image, $(R1, R2)$ denote the band filter image pair, we have:-

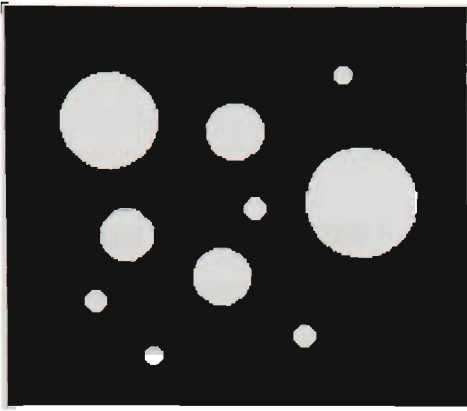


Fig. 7.9 The original image X

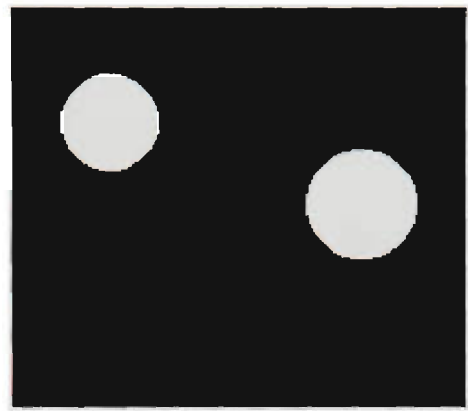


Fig. 7.10 Extract the noise need to be removed
(Result of $X \circ R1$)

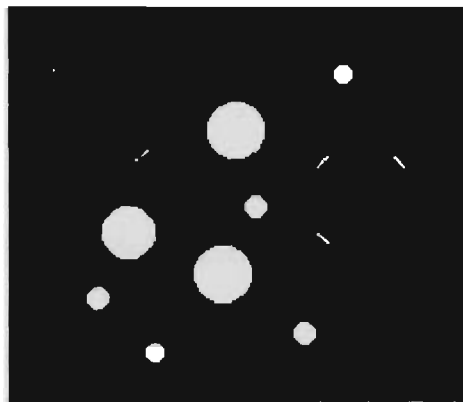


Fig. 7.11 High frequency noise is removed (Result of $X \cap (\overline{X \circ R1})$)

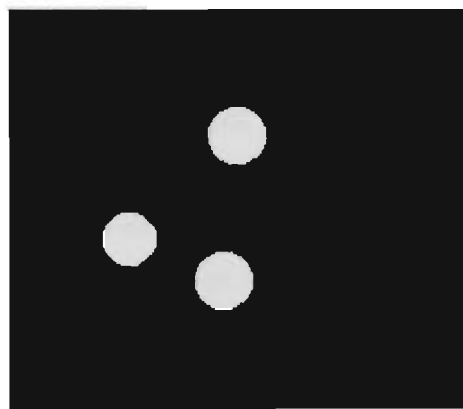


Fig. 7.12 The blobs of interest are extracted (Result of $(X \cap (\overline{X \circ R1})) \bullet R2$)



Fig. 7.13 Selected blobs transformed to single points



Fig. 7.14 The band filter image pair (R_1 , R_2)

We tested this image (209 by 182) on a 486/33 DX with 8 meg RAM under Smalltalk /V Win environment. The result shows that it is about 2.5 times faster than the classical algorithm in the same environment. The above operations also can be performed inside the ImageLab we described in Chapter Five.

The above work, although still in progress, suffices to indicate a way in which one might use ImageLab. Further work needs to be done on the choice of a template which avoids the merging of blobs, and a rigorous assessment of the speed of the algorithm is needed.

7.1.3 Application of the new labelling method

Blob counting has many industrial applications. One typical use is in counting the pharmaceutical tablets in a tablets packaging line [GASP86]. Another application is counting clusters in a digitised X-ray mammogram. Here we give a detailed discussion of this application.

We intend to use this technique to detect the microcalcifications in mammograms. Breast cancer is a leading cause of death in women. The early diagnosis and treatment significantly improves the chances of survival for patients with breast cancer. Because microcalcification in the breast is the main symptom of breast cancer, better detection of clusters of microcalcification in mammograms will lead to further improvements in the early detection of breast cancer [SICK82].

H.P. Chan et al investigated the feasibility of an automated computer method for the detection of clusters of microcalcification in digital mammograms [CHAN87]. F.F. Yin et al used the bilateral subtraction technique for the detection of potential masses in digital mammograms [YIN91]. Y.Z. Wu used artificial neural networks to distinguish actual "true" clusters from normal parenchyma patterns and also to distinguish actual clusters from false-positive clusters as reported by a computerised scheme for the detection of microcalcifications in digital mammograms [WUYZ92].

The steps of technique described in Wu's paper [WUYZ92] are:-

1. Digitise the mammogram
2. If simulating, superpose the clusters of calcifications.
3. Enhance the signals; and suppress the signal.
4. Form the difference in (2) and (3)
5. Use a threshold to create a binary image.
6. Extract the signal and locate the clusters

We have concentrated on step six. After we have created a binary image, the morphological process of closing is employed to transform the clusters into blobs. Then, we can locate the clusters and count the numbers of clusters by using the method developed above.

The mathematical expression of the morphological *closing* operation is:-

$$X \bullet R = (X \oplus R) \ominus R = \overline{(\overline{X \oplus R}) \oplus \overline{R}}$$

where \oplus is dilation of image by a structure-element (filter) and \ominus is the erosion by the same structure element.

We use the following image to simulate the found clusters in a digitised mammogram.



Fig. 7.15 Original image with clusters



Fig 7.16 Structure Element



Fig. 7.17 Result after closing operation

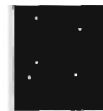


Fig. 7.18 Selected blobs transformed to single points

These operations can be performed either by using the image processing environment (ImageLab) which we described in Chapter Five or by implementing the following code inside the development environment:-

```
| anOc1 anOc2 bitmap image result|
result :=1.
bitmap:=Bitmap fromFile:'c:\vwin\bmp/blob4.bmp'.
image:=Image fromBitmap:bitmap.
image displayAt:0@0 with: Display pen.
image label:5.
image displayAt:0@260 with: Display pen.
```

CHAPTER 8

Non-morphological image processing methods in Smalltalk

Introduction

One could say that the general context of the present thesis is image-processing in an object-oriented environment, and that the specific focus has been the development of ImageLab (Chapter Five) and its application to the particular problem of counting blobs (Chapter Seven). Looking back, we see that there were two key steps:-

1. We were able to use the class BitBlit to implement the three fundamental morphological processes.
2. Smalltalk facilitated our implementing other morphological operations as algebraic expressions involving the basic operations.

This chapter complements the preceding chapter by giving examples of the implementation of non-morphological operations. In the first we create a class QuadCode which offers an introduction to multi-resolution techniques. In the second we offer an introduction to adaptive quadtree coding.

8.1 Quad-tree in Smalltalk

In Chapter Three, we demonstrated an advantage of an object-oriented language through a quad-tree example. We now give more details on how to implement a quad-tree in Smalltalk.

8.1.1 Quad-tree fundamentals

The quadcode is a base-4 code representing the quadrants of a binary image. First, the whole image is decomposed into four equal-sized quadrants. If a quadrant is not

included entirely in the object or in the background, it is again subdivided into four sub-quadrants; otherwise the decomposition ends. [JÄHN91].

The recursive decomposition can be represented in a tree. At the top of the tree, known as the root, the decomposition starts. The root corresponds to the entire binary image. It is connected via four edges to four child-nodes which represent from left to right the NW, NE, SE, and SW quadrants. If a quadrant needs no further subdivision, it is represented by a terminal or leaf node in the tree. It is called black when the quadrant belongs to an object and white otherwise, and is indicated as such by a filled and open square, respectively. Non-leaf nodes require further subdivision and are said to be gray and are shown as open rectangles. (Fig. 8.1)

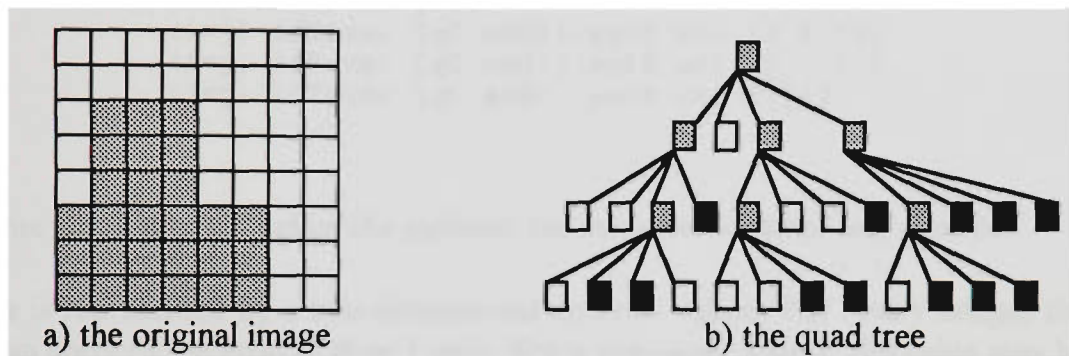


Fig. 8.1 Representation of a binary image by a region quadtree

Quadtrees can be encoded, for example, by a depth-first travel of the tree starting at the root. A quadtree is a compact representation of a binary image if it contains many leaf nodes at high levels. However, in the worst case, for example a regular checkerboard pattern, all leaf nodes are at the lowest level. The quad tree then contains as many as pixels and requires much more bytes of storage space than the direct representation of the binary image as a matrix [SAME90].

8.1.2 Quadcode in Smalltalk

We now implement quadcode in Smalltalk/V286. To start, we define the class Quadcode as a subclass of OrderedCollection. First, we introduce the instance method:-

```

asPoint
    "Returns coords of quad-cell determined by self"

    | i j n |
    i :=0.
    j :=0.
    n := self size.
    (1 to: n) do: [:k|
        i := (self at: k)//2 * (2 raisedTo: (n - k)) + i.
        j := (self at: k)//2 * (2 raisedTo: (n - k)) + j ].
    ^i@j

```


Next, we introduce instance methods for returning the quadcode of a neighbouring quad-cell. Each quadcode has north, south, west and east neighbours. The methods are as follows:-

```
eastNbr
  " Returns the east quad-cell of self, false if none"

  | j qC |
  j := 0.
  (1 to: self size) do: [ :i |
    (((self at: i) = 0) or: [(self at: i)=2]) ifTrue: [j:=i]].
  (j = 0) ifTrue: [^false]
  ifFalse:[qC := Quadcode new.
    ( 1 to: self size) do:[:i|
      (i>j) ifTrue: [qC add:((self at: i) + 2)].
      (i=j) ifTrue: [qC add:((self at: i) - 2)].
      (i<j) ifTrue: [qC add: (self at: i)]]].
  ^qC
```

In the same way we could define the methods for northNbr, southNbr and westNbr.

An image is represented by a two-dimensional array of values. For binary images the value of an entry in the array is 0 or 1 only. For a gray-scale image, the value may be an integer in the range 0 - 15 for a VGA display, or in the range 0 - 255 for a super VGA display.

We have introduced the class Quadcode in connection with the nesting of quadrants in an arbitrary 2^n - by - 2^n array. We shall now apply Quadcode to encoding and decoding a binary image, that is, to an array with each element either 0 or 1.

Let us **encode** a binary image first. To encode a binary image into a quadcode, we need to introduce into the class Form an instance method to return a nominated quadrant of a Form :-

```
quadrantAt: anInt
  "anInt = 0, 1, 2, 3"
  | w qF orgn cnr y |
  y := 0.
  w := self width/2.
  qF := Form width: w height:w.
  (anInt > 1) ifTrue:[ y:=w].
  orgn := (anInt\\2 * w)@y.
  qF copy: (orgn extent: w@w) from: self to: 0@0 rule:3.
  ^qF.
```

Now introduce an instance method in the class Form to return a 'sub-quadrant' or quadcell defined by aQuadcode:-

```
quadCellAt: aQuadcode
  "Returns a Form for the relevant quadcell"
  | qF |
  qF := self quadrantAt: (aQuadcode at: i).
  (aQuadcode size = 1) ifTrue:[^qF].
  (2 to: aQuadcode size) do:
```

```

[:i |qF := qF quadrantAt: (Quadcode at: i)].
^qF

```

Then, we need an instance method in the class Form to determine whether an image is white(pixel values equal to 1):-

```

iswhite
self bitmap detect: [ :ea | ea ~=255] ifNone: [^true].
^false

```

In our case, 'isWhite' works for a 4-by-4 Form because the padding to 16-by-4 is via white pixels. Hence all this work is in terms of 'white'.

In the class Form, we need to introduce another instance method which returns an OrderedCollection of instances of Quadcode, indicating the quadrants which are white:-

```

quadsWhite
| qCdColln qC |
qCdColln := OrderedCollection new.
qC := quadcode new.
qC add: 0.
( 0 to: 3) do: [ :i |
    qC removeLast.
    qC add: i.
    (self quadrantAt: i) isWhite ifTrue: [
        qCdColln add: qC deepCopy
    ]
].
^qCdColln

```

The following two instance methods, to be introduced into the class Form, return an OrderedCollection of instances of Quadcode specifying all quadrants and sub-quadrants which are white:-

```

quadRecursiveWhite
" This method initiates the recursion"
| qCdColln qC quad |
qCdColln := QuadcodeCollection new.
qC := Quadcode new.
qC add: 0.
(0 to: 3) do: [ :i |
    qC removeLast.
    qC add:i.
    quad := self quadrantAt:i.
    quad isWhite
        ifTrue: [
            qCdColln add:qC deepCopy
        ]
        ifFalse:[
            quad quadRecursiveWhite: qC qCdColln: qCdColln
        ]
].
^qCdColln

```

quadRecursiveWhite: qC qCdColln: qCdColln

```

" The recursion occurs in this method"
| qCdColln qC quad |

qC add: 0.
(0 to: 3) do: [:i |
    qC removeLast.
    qC add: i.
    quad := self quadrantAt:i.
    quad isWhite
    ifTrue:[
        qCdColln add: qC deepCopy
    ]
    ifFalse:
    [
        (quad width>1)
        ifTrue:
            [quad quadRecursiveWhite: qC qCdColln: qCdColln]
        ifFalse:[]
    ] ].
qC removeLast.
^qCdColln

```

The above codes are intended to encode a binary image into quadcode. Now we want to reconstruct an image from its quadcode. You might have noticed in the instance method of class Form quadRecursiveWhite, that we invoked a class called QuadcodeCollection which is a subclass of class OrderedCollection. The instance of class QuadcodeCollection contains the quadcodes of encoded image. In order to decode the quadcode, we need to introduce an instance method asForm into the class QuadcodeCollection:-

asForm

```

" Returns the Form determined by a QuadcodeCollection"
|length w z qCdSize orgn aFm quad qW tP |
length :=0.
self do : [:ea | (ea size > length) ifTrue: [length := ea
size]].
w := 2 raisedToInteger: length.
aFm := (Form width: w height: w) reverse.
z := self deepCopy.
z do: [:ea |
    qCdSize := ea size.
    qW := length - qCdSize.
    tP := 2 raisedToInteger: qW.
    qW timesRepeat: [ea add:0].
    orgn := ea asPoint.
    quad := Form width: tP height: tP.
    aFm copy: (0@0 extent: tP@tP) from:quad to: orgn rule:3
].
^aFm

```

8.2 Adaptive quadtree: A new method for image coding

In the preceding section, we discussed the quadcode and its Smalltalk implementation. This conventional scheme is based on the regular decomposition of space that is recursively divided into four quadrants (quadtrees) which are square regions of the same size. It is a non-adaptive, in the sense that its rule of decomposition is fixed and not dependent on the content of the image.

Aurélio J. C. Campilho recently proposed a new method for image representation by means of an adaptive tree [CAMP93]. His adaptive scheme recursively decomposes the image into two rectangular regions, according to a joint uniformity measure of the regions. He studied two adaptive methods, one simply directionally adaptive and the other more general. The first method bisects the image recursively, the choice between a horizontal bisection and a vertical bisection being determined by the content of the image. In the second method the division is not restricted to a bisection; the line of division is dependent upon the content of the image.

Our purpose in this section is show how readily Smalltalk allows us to explore an extension of Campilho's adaptive technique. We shall construct a quadtree which recursively divides a rectangular regions into four subregions where the positions of the two lines of division are dependent upon the image content. We shall call the intersection of the two division lines the 'hub' of the division. We redraw figure 8.1 here and compare it with its adaptive tree.

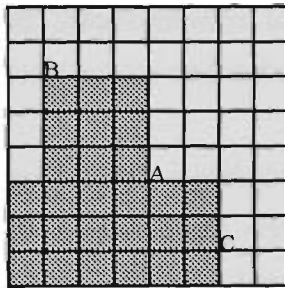


Fig. 8.2 The original image

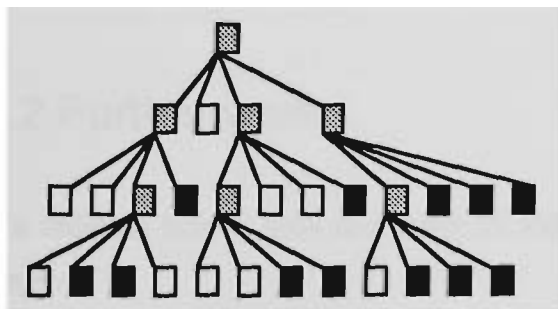


Fig. 8.3 The quadtree

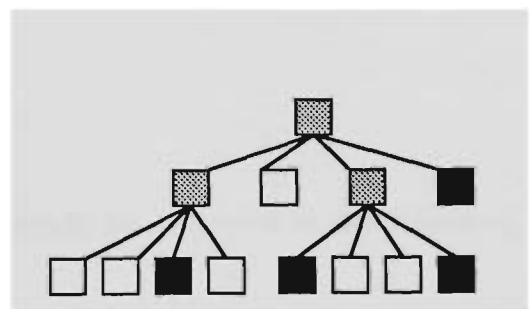


Fig. 8.4 The adaptive quadtree

The quadtree has four levels but the adaptive tree only has three levels and has fewer leaves. The only extra work is we need to record the 'hub' points (A, B, C in the original image Fig. 8.2) which are the place where we made the vertical and horizontal division.

We borrow the partitioning criteria from Campilho's paper [CAMP93]. The decision to make a division is based on a partition which evaluates the uniformity between regions.

Chapter 9

Conclusion and further work

9.1 Conclusions

The chief concern of this thesis has been the creation of ImageLab which is intended to give a researcher ready access to an object-oriented language. We have seen several reasons for embedding ImageLab in Smalltalk. Smalltalk is uncluttered by features which, although offering advantages in other endeavours, can only place obstacles in the way of exploration. Smalltalk is an environment as well as a language, offering a class library, browsers, inspectors and debuggers.

Our interest in morphological operations highlighted two special reasons for the choice of Smalltalk. With the BitBlit class we can readily represent images and operate on them; and the Smalltalk language facilitates the algebraic expression of complex image operations in terms of simpler ones.

9.2 Further work

We indicate below how the work of this thesis might be extended in the following ways:-

- introducing true gray-scale image-processing;
- transporting the environment from the PC to other platforms such as UNIX;
- transporting the environment from Smalltalk/V to ObjectWorks (formerly ST-80);
- expanding the object-oriented image database;
- introducing an image-processing language.

9.2.1 Further work on gray-scale images

In chapter seven, we proposed pseudo gray-scale morphological operations. Further work could address the following questions:-

- (1) Where might we find the pseudo operation useful?
- (2) To what extent can we exploit BitBlit in introducing 16-level and 256-level gray-scale operations?

9.2.2 Other Smalltalk platforms

The most efficient way to port ImageLab to other platforms (UNIX, Mac) would be via a conversion from Smalltalk/VWin to ObjectWorks (or VisualWorks, the version of ObjectWorks carrying code-generating class for creating interfaces). Code or an image created in ObjectWorks (VisualWorks) on, say, a PC will run under ObjectWorks (VisualWorks) on other platforms.

In the same way, the present work could be translated into Envy/Smalltalk

9.2.3 Object-oriented image database

ImageLab could be coupled with an object-oriented database, the best candidate appearing to be Gemstone [STE188]. Shamim Ahmed [AHME92] has reviewed commercial object-oriented databases, and has rated them according to several attributes. He examined five products: Orion/Tasca, Gemstone, Ontos, ObjectStore and Versant. Orion/Tasca tended to lead in all attributes. Gemstone compared favourably with the others (where comparisons were relevant). Gemstone was the only product which could be accessed from both Smalltalk and C++.

9.2.4 An image processing language

This present work could be overlaid with an image processing language. To do this, we need to define a syntax for the image-processing language. We have two ways to extend the current work towards the image processing languages:-

- build a functional language so that a user can program by inputting a sequence of mathematical morphology functions;
- build a graphical language so that users can use it to do image processing in visual programming style

References

- [AHME92] **S. Ahmed, et al.** "Object-Oriented database management systems for engineering: A comparison", *JOOP*, pp. 27-43 June, 1992
- [BALL82] **D. H. Ballard, C. M. Brown** *Computer Vision*, Prentice-Hall, 1982
- [BATC91] **B. Batchelor** *Intelligent Image Processing in Prolog*, Springer-Verlag, 1991
- [BEIL89] **R. Beilinson, A. Ginige, et al.** "An Object-Oriented Approach to Feature Extraction", in *Image Processing and the Impact of New Technologies, Proce. of IREE Australia*, pp. 119-121, Dec. 18-20, 1989
- [BUDD91] **T. Budd** *An Introduction to Object-Oriented Programming*, Addison Wesley, 1991
- [CAMP93] **A.J.C. Campilho** "Adaptive-tree: A new method for image representation", in *DICTA-93, Conference Proceeding*, pp. 706-713, Sydney, Dec., 1993
- [CART89] **M. Carter, et al.** "The Design and implementation of a Portable Image Processing Library (IPAL) in Fortran and C", in *3rd IEE International Conference on Image Processing and its Applications*, Warwick, 1989
- [CAST79] **K.R. Castleman** *Digital Image Processing*, Prentice-Hall, Englewood, pp. 401-411, 1979

- [CHAN87] **H.P. Chan, et al.** "Image feature analysis and computer-aided diagnosis in digital radiography. 1. Automated detection of microcalcifications in mammography", in *Medical Physics*(14), pp. 538-548, 1987
- [COX91] **B.J. Cox, A. Nonobilski** *Object-oriented programming: an evolutionary approach*, Addison-Wesley, 1991
- [DIGI88] *Goodies #3, Carleton Projects Application Pack for Smalltalk/V*, Digitalk Inc., 1988
- [DIJK76] **E.W. Dijkstra**, *A Discipline of Programming*, Prentice-Hall, 1976
- [DOUG87] **E.R. Dougherty, C.R. Giardina** "Image Algebra -- Induced operators and induced subalgebras", in *Proceedings, SPIE Conf. -- Visual Commutation and Image Processing II, Cambridge, MA*, pp. 270-275, Oct., 1987
- [DOUG92] **E.R. Dougherty** *An Introduction to Morphological Image Processing*, SPIE, 1992
- [DOUG93] **E.R. Dougherty** *Mathematical Morphology In Image Processing*, Marcel Dekker Inc., 1993
- [EGE92] **R.K. Ege** *Programming in an Object-Oriented Environment*, Academic Press, 1992
- [GASP86] **M.L. Gasperi** "Introduction to Morphological Image Processing", in *Conference Proceedings Vision 86*, pp. 5.63-5.84, 1986
- [GOLD89] **A. Goldberg, R. David** *Smalltalk-80 The Language*, Reading Mass: Addison-Wesley, 1989
- [GONZ92] **R.C. Gonzalez, R.E. Woods** *Digital Image Processing*, Addison Wesley, 1992

- [HARA87] **R.M. Haralick, et al.** "Image Analysis Using Mathematical Morphology", in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-9, No.4, pp. 532-550, July, 1987
- [HARA92] **R.M. Haralick, L.G. Shapiro** *Computer And Robot Vision*, Vol. 1 Addison Wesley, 1992
- [HARA93] **R.M. Haralick, L.G. Shapiro** *Computer And Robot Vision*, Vol. 2 Addison Wesley, 1993
- [HUAN89] **K.S. Huang** *A Digital Optical Cellular Image Processor*, World Scientific, 1989.
- [JAIN89] **A.K. Jain** *Fundamentals of Digital Image Processing*, Prentice Hall, 1989
- [KRAS83] **G. Krasner** *Smalltalk-80 Bits of History, words of advance*, Addison Wesley, 1983
- [LALO90a] **W.R. LaLonde, J.R. Pugh** *Inside Smalltalk*, Vol. 1 Prentice Hall, 1990
- [LALO90b] **W.R. LaLonde, J.R. Pugh** *Inside Smalltalk*, Vol. 2 Prentice Hall, 1990
- [LAMB93] **A. Lambert** "Using Object-Oriented Programming in Image Processing", in *DICTA-93, Conference Proceeding*, pp. 220-227, Sydney, Dec, 1993
- [LAU93] **S. Lau** "Performance comparison: The Object-Oriented Languages", *Minor Thesis*, CAMS, Victoria Uni. of Tech., 1993
- [LIND91] **C.A. Lindley** *Practical Image Processing in C: acquisition, manipulation, storage*, New York: Wiley, 1991

- [MARA85] **P. Maragos** *A Unified Theory of Translation-Invariant Systems with Applications to Morphological Analysis and Coding of Images*, PhD thesis, Georgia Inst. Tech, Atlanta, 1985
- [McCO63] **B.H. McCormick** "The Illinois pattern recognition computer-ILLIAC III", in *IEEE Trans. Electron, Comput*, pp. 791-813, EC-12, 1963
- [MEYE88] **B. Meyer** *Object-oriented Software Construction*, Prentice Hall, 1988
- [MINK03] **H. Minkowski** "Volumen und oberflache", in *Math. Ann.*, pp. 447-495, Vol. 57, 1903
- [NAKA78] **Y. Nakagawa, A. Rosenfeld** "A Note on the use of Local min and max operation in Digital Picture Processing", in *IEEE Transactions on Systems, MAN, AND Cybernetics*, Vol. SMC-8, No.8, Aug., 1978
- [PARN72] **D.L. Parnas** "On the Criteria to Be Used in Decomposing Systems into Modules", in *Communications of the ACM*, 15(12), pp. 1059-1062, 1972
- [PATO79] **K.A. Paton** *Clinical Research Centre, Harrow*, Personal Communication, 1979
- [PELE81] **S. Peleg, A. Rosenfeld** "A min max medial axis transformation", in *IEEE Trans. Pattern Anal. Machine Intell.*, VOL. PAMI-3, pp. 206-210, 1981
- [PIPE85] **J. Piper, D. Rutowitz** "Data Structure for Image Processing in a C Language and Unix Environment", in *Pattern Recognition letters*, vol. 3, pp. 119, 1985
- [PRES81] **K. Preston Jr.** "Image Processing Software-A Survey", *Progress in Pattern Recognition*, Nothe-Holland Publishing, pp. 123-147, 1981
- [PRES83] **K. Preston Jr.** "Progress in Image Processing", in *Computing Structures for Image Processing*, Academic Press, pp., 195-211, 1983

- [RITT90] **G.X. Ritter, et al.** "Image Algebra: an overview", in *Computer Vision, Graphics, and Image Processing*, 49, pp. 297-331, 1990
- [ROSE76] **A. Rosenfeld** "Adjacency in digital pictures", in *Information and Control*, vol. 26, pp. 24-33, 1976
- [ROSE82] **A. Rosenfeld, A.C. Kak** *Digital Picture Processing*, Vol. 2 Academic Press, 1982
- [RUSS92] **J.C. Russ** *The Image Processing Handbook*, CRC Press, 1992
- [SAVI90] **D. Savic** *Object Oriented Programming with Smalltalk/V*, Ellis Horwood, 1990
- [SERR82a] **J. Serra** *Image Analysis and Mathematical Morphology*, Vol. 1 Academic Press, 1982.
- [SERR82b] **J. Serra** *Image Analysis and Mathematical Morphology*, Vol. 2 Academic Press, 1982.
- [STEI88] **J. Stein** "Object-oriented programming and databases", in *Dr. Dobb's Journal*, pp. 18-34 Mar., 1988
- [STER80] **S.R. Sternberg** "Language and architecture for parallel image processing", in *Proceedings, Conf. Pattern Recognition in Practice*, Amsterdam, May, 1980
- [WERM85] **M. Werman, S. Peleg** "Min-max operators in texture analysis", in *IEEE Trans. Pattern Anal. Machine Intell.*, VOL. PAMI-7, pp. 730-733, 1985
- [WINB90] **A.L. Winblad, et al.** *Object-Oriented Software*, Addison Wesley, 1990
- [WUYZ92] **Y.Z. Wu, et al.** "Computerised detection of clustered microcalcification in digital mammograms: Applications of artificial neural networks", in *Medical Physics*, pp. 555-560 18(5), May/June, 1992

- [YIN91] F.F. Yin, et al. "Computerised detection of masses in digital mammograms: Analysis of bilateral subtraction images", in *Medical Physics*, pp. 955-963 18(5), Sep./Oct., 1991
- [YOUR91a] P. Coad, E. Yourdon *Object-Oriented Analysis*, Yourdon Press, 1991
- [YOUR91b] P. Coad, E. Yourdon *Object-Oriented Design*, Yourdon Press, 1991

Appendices

Appendix A. Disk 1, Programs in VWin

There are two disks accompanied with this thesis. The first disk(Disk1) has ImageLab programs written in VWin. The second disk(Disk2) contains the Small-image database programs written in V286 and also some sample images. In each disk there is a **read.me** file which explains to the user how to install the programs and sample images. The subdirectories of the two disks contain a **readme.1st** file, these files explain to the user how to implement the programs and some brief instructions for the use of the programs.

Disk1 has three subdirectories. The reader is recommended to go first to *vwin_run* which contains a compressed file **vwin_run.exe**. Copy into a new directory on the hard drive. Then type:-

```
vwin_run[↵]
```

then open windows and install (**v.exe**) in the usual way. Finally double-click the icon.

If the reader wishes to study the VWin development he/she should go to the subdirectories *vwin_img* and *vwin_cls*. The first contains a compressed image and **change.log**. The second contains compressed ***.cls** files which can be filed into an existing image.

Appendix B. Disk 2, Sample images and programs in V286

The subdirectory *bin* contains sample binary images in Microsoft Windows Bitmap format. The subdirectory *gray* contains gray-scale sample images also in Bitmap format. The subdirectory *img* contains sample images in the format we defined in chapter three for Small-image database.

If the reader wishes to study the V286 development he/she should go to the subdirectories *v286_img* and *v286_cls*. The first contains a compressed image and **change.log**. The second contains compressed *.cls files which can be filed into an existing image.

Appendix C. Classes and methods of Small-image Database

```
DictionaryInspector subclass: #ImageDataBaseInspector
  instanceVariableNames:
    'anIdbTopPane fileName '
  classVariableNames: ''
  poolDictionaries: '' !

!ImageDataBaseInspector class methods ! !

!ImageDataBaseInspector methods !

bitEdt

  | aPrompter anImage width height key size index|

  (instIndex = nil ) ifTrue:[
    aPrompter:=Prompter prompt:'Graph object name?'
                        default: ''.
    fileName:= aPrompter.
    fileName isEmpty ifTrue:[^nil].
    width:=(Prompter prompt:'Width = ?'
                        default: '47') asInteger.
    (width = 0) ifTrue:[^nil].
    height:=(Prompter prompt:'Height = ?'
                        default: '47') asInteger.
    anImage:=Image new width:width height:height .
    IdBtEdt new openOn: anImage .
    ^self    ]
    ifFalse:[
  aPrompter:=Prompter prompt:'Graph object name?'
                        default: (instList at: instIndex) key].
  fileName:= aPrompter.
  (fileName = nil) ifTrue:[^nil].
  fileName isEmpty ifTrue:[
  width:=(Prompter prompt:'Width = ?'
                        default: '47') asInteger.
  (width = 0) ifTrue:[^nil].
  height:=(Prompter prompt:'Height = ?'
                        default: '47') asInteger.
  anImage:=Image new width:width height:height .
  IdBtEdt new openOn: anImage ].
  anImage:=object at: (instList at: instIndex) key.
  IdBtEdt new openOn: anImage .!

clear

  " To clear the instance pane"

  ^(Image new width:instPane frame
```

```
width height:instPane frame height; white)
  displayAt: instPane frame origin.!
```

freDraw

```
IdbFreeDraw new!
```

fromDisk

```
" To get image from disk file, insert into the
  data base"
```

```
| anImage aPrompter key size index |

aPrompter:=Prompter prompt:'file name?'
  default: fileName.
fileName:= aPrompter.
(fileName=nil ) ifTrue:[^nil].
fileName isEmpty ifTrue:[^nil].
CursorManager execute change.
anImage:=Image idbFromFile:fileName.
CursorManager normal change.

key := Prompter
  prompt: 'new key expression'
  default: String new.
key isNil
  ifTrue: [^self].
(object includesKey: key)
  ifTrue: [
    Menu message: 'key already in dictionary'.
    ^self].
object at: key put: anImage.
instList add:
  (Association key: key value: key ).
size := instList size.
index := 1.
[index > size
  or: [(instList at: index) key = key]]
  whileFalse: [index := index + 1].
instIndex := index.
self
  changed: #instVarList
  with: #restoreSelected:
  with: instIndex;
  changed: #instance:!
```

idbInsPaneMenu

```
" Answer instance pane menu"

| menu |
menu := Menu
  labels: 'clear\print\save as' withCrs
  lines: #(1)
  selectors: #{clear printImage storeOnFile}.
^menu!
```

idbListMenu

```
"Private - Answer the small image data base inspector list
pane menu."
```



```

^Menu
  labels:'add      from      disk\bit      editor\free      drawing\
inspect\remove \save IDB' withCrs
  lines: #(1 3 5)
  selectors: #( fromDisk bitEdt freDrw inspectSelection remove
storDic )!

```

instance: anImage

```

Q:=instIndex.
instIndex isNil
ifTrue: [^(instPane clear)].
instPane clear.
^(object at: (instList at: instIndex) key)
  displayAt: instPane frame origin
  clippingBox: ((instPane frame origin)
  extent:( instPane frame extent))!

```

loadDic

"To load all the images in data base onto disk"

```

| aPrompter fileName |
aPrompter:=Prompter prompt:'file name?'
  default:'image.dic'.
fileName:= aPrompter.
(fileName=nil) ifTrue:[^nil].
(fileName size=0) ifTrue:[^nil].
CursorManager execute change.
object:=ImageDataBase idbFromDisk:fileName deepCopy.
CursorManager normal change .
anIdbTopPane dispatcher close scheduleWindow.
^self openOn:object!

```

openOn: anObject

"Open an inspector window on anObject. Define the pane sizes and behavior, and shedule the window."

```

object := anObject.
instPane := GraphPane new
  menu: #idbInsPaneMenu;
  model: self;
  name: #instance;;
  framingRatio: (1/3 @ 0
  extent: 2/3 @ 1).
anIdbTopPane := IdbTopPane new.
anIdbTopPane
  label: ' Small Image Data Base ';
  model: anIdbTopPane dispatcher;
  menu: #workSpaceMenu;
  minimumSize: 80@80;
  yourself.
anIdbTopPane addSubpane:
  (ListPane new
    menu: #idbListMenu;
    model: self;
    name: #instVarList;
    change: #selectInstance;;
    returnIndex: true;
    framingRatio: (
      0@0 extent: 1/3 @ 1)).

```

```

anIdbTopPane addSubpane: instPane.
CursorManager normal change.
self setInstList.
anIdbTopPane dispatcher open scheduleWindow!

```

printImage

```

    "To print out the selected image on pin-printer"

^(object at: (instList at: instIndex) key)

    outputToPrinterUpright!

```

remove

```

    "Private - Remove the selected
    key from the dictionary."
| assoc |
instIndex isNil
    ifFalse: [
        assoc := instList at: instIndex.
        instList remove: assoc.
        object removeKey: assoc key.
        instIndex := nil.
        self
            changed: #instVarList with: #restore;
            changed: #instance:!]

```

selectInstance: anInteger

```

    "Private - Select the instance variable at index position
    anInteger in the list."

| lastIndex |
lastIndex := instIndex.
instIndex := anInteger.
self changed: #instance:.
instIndex = lastIndex
    ifTrue: [self inspectSelection!]

```

setInstList

```

    "Private - Compute instList, an OrderedCollection of key
    strings for the list pane."
| aSet |
aSet := Set new: object size.
object keysDo: [:aKey |
    aSet add:
        (Association key: aKey value: aKey )].
instIndex := nil.
(instList := SortedCollection new)
    sortBlock: [:a :b| a value < b value];
    addAll: aSet!

```

storDic

```

    "To store all the image data base on disk"

| aPrompter |
aPrompter:=Prompter prompt:'file name?'
                    default: fileName .

```

```

fileName:= aPrompter.
(fileName=nil) ifTrue:[^nil].
(fileName size=0) ifTrue:[^nil].
CursorManager execute change.
object idbStoreOnDisk:fileName.
CursorManager normal change!

```

storeOnFile

" To save selected image on a disk file."

```

| aPrompter anImage |
instIndex isNil
    ifTrue: [^self].
aPrompter:=Prompter prompt:'file name?'
                    default:fileName .
fileName:= aPrompter.
(fileName=nil) ifTrue:[^nil].
(fileName size=0) ifTrue:[^nil].
anImage:= (object at: (instList at: instIndex) key).
(anImage class name)='Image' ifTrue:[
    CursorManager execute change.
    anImage storeOnFile:fileName.
    CursorManager normal change.]
                    ifFalse:[
    CursorManager execute change.
    anImage storeColorOnFile:fileName.
    CursorManager normal change.]] !

```

Appendix D. Classes and methods of Image processor

```

Object subclass: #ImageProcessor
instanceVariableNames:
    'object leftPane middlePane rightPane bottomPane fileName
instList instIndex indexSet resultImage resultImageA resultImageB
resultImageC '
classVariableNames: ''
poolDictionaries: '' !

```

!ImageProcessor class methods ! !

!ImageProcessor methods !

bottomPaneMenu

```

^Menu
labels:
'complement\union\dilation\reflect\erosion\difference\closing\openin
g\intersection\symmetricDiff\hitMissTrans\thicken\thin\print\save
as' withCrs
lines: #(1 5 9 13)
selectors: #(complement union dilation reflect erosion difference
closing opening intersection symmetricDiff hitMissTrans thicken thin
printResult storeOnFile)!

```

closing

| a b |

```

bottomPane clear.
a:=resultImageA deepCopy.
b:=resultImageB deepCopy.
CursorManager execute change.
a closingBy: b.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
  clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!

```

complement

| a |

```

bottomPane clear.
a:=resultImageA deepCopy.
CursorManager execute change.
a complement.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
  clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!

```

difference

| a b |

```

bottomPane clear.
a:=resultImageA deepCopy.
b:=resultImageB deepCopy.
CursorManager execute change.
a difference: b.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
  clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!

```

dilation

| a b |

```

bottomPane clear.
a:=resultImageA deepCopy.
b:=resultImageB deepCopy.
CursorManager execute change.
a dilationBy: b.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
  clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!

```

erosion

```
| a b |
```

```
bottomPane clear.
a:=resultImageA deepCopy.
b:=resultImageB deepCopy.
CursorManager execute change.
a erosionBy: b.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
  clippingBox: ((bottomPane frame origin)
  extent:( bottomPane frame extent))!
```

fromDisk

" To get image from disk file, insert into the data base"

```
| aPrompter key size index |
```

```
aPrompter:=Prompter prompt:'file name?'
  default: fileName.
fileName:= aPrompter.
(fileName=nil ) ifTrue:[^nil].
fileName isEmpty ifTrue:[^nil].
CursorManager execute change.
resultImage:=Image idbFromFile:fileName.
CursorManager normal change.
```

```
leftPane hasCursor ifTrue:[
leftPane clear.
resultImageA:=resultImage deepCopy.
^resultImageA
  displayAt: leftPane frame origin
  clippingBox: ((leftPane frame origin)
  extent:( leftPane frame extent))].
middlePane hasCursor ifTrue:[
middlePane clear.
resultImageB:=resultImage deepCopy.
^resultImageB
  displayAt: middlePane frame origin
  clippingBox: ((middlePane frame origin)
  extent:( middlePane frame extent))].
rightPane hasCursor ifTrue:[
rightPane clear.
resultImageC:=resultImage deepCopy.
^resultImageC
  displayAt: rightPane frame origin
  clippingBox: ((rightPane frame origin)
  extent:( rightPane frame extent))]!
```

fromResult

"To get the image from the result image for another operation."

```
leftPane hasCursor ifTrue:[
resultImageA:=resultImage deepCopy.
leftPane clear.
^resultImageA
  displayAt: leftPane frame origin
  clippingBox: ((leftPane frame origin)
```

```

    extent:( leftPane frame extent))].
    midlePane hasCursor ifTrue:[
    resultImageB:=resultImage deepCopy.
    midlePane clear.
^resultImageB
    displayAt: midlePane frame origin
    clippingBox: ((midlePane frame origin)
    extent:( midlePane frame extent))].
    rightPane hasCursor ifTrue:[
    resultImageC:=resultImage deepCopy.
    rightPane clear.
^resultImageC
    displayAt: rightPane frame origin
    clippingBox: ((rightPane frame origin)
    extent:( rightPane frame extent))!

```

getIt

"To get the image from the Small Image Data Base."

```

    instIndex:=Q.
    leftPane hasCursor ifTrue:[
    indexSet at:1 put: instIndex.
    ^ (self changed: #instanceLft:)].
    midlePane hasCursor ifTrue:[
    indexSet at:2 put: instIndex.
    ^ (self changed: #instanceMid:)].
    rightPane hasCursor ifTrue:[
    indexSet at:3 put: instIndex.
    ^ ( self changed: #instanceRit:)]!

```

hitMissTrans

```

    | a b c |

    bottomPane clear.
    a:=resultImageA deepCopy.
    b:=resultImageB deepCopy.
    c:=resultImageC deepCopy.
    CursorManager execute change.
    a hitMissTransBy: b and: c.
    CursorManager normal change.
    resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
    clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!

```

instanceBtm: anImage

```

    instIndex isNil
    ifTrue: [^( bottomPane clear)].
    bottomPane clear.
^resultImage
    displayAt: bottomPane frame origin
    clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!

```

instanceLft: anImage

```

    instIndex isNil

```

```

    ifTrue: [^leftPane clear].
    leftPane clear.
    resultImageA:=(object at: (instList at: instIndex) key)
deepCopy.
    ^resultImageA
                                displayAt: leftPane frame origin
                                clippingBox: ((leftPane frame origin)
                                extent:( leftPane frame extent))!

```

instanceMid: anImage

```

    instIndex isNil
    ifTrue: [^midlePane clear].
    midlePane clear.
    resultImageB:=(object at: (instList at: instIndex) key)
deepCopy.
    ^resultImageB
                                displayAt: midlePane frame origin
                                clippingBox: ((midlePane frame origin)
                                extent:( midlePane frame extent))!

```

instanceRit: anImage

```

    instIndex isNil
    ifTrue: [^(rightPane clear)].
    rightPane clear.
    resultImageC:=(object at: (instList at: instIndex) key)
deepCopy.
    ^resultImageC
                                displayAt: rightPane frame origin
                                clippingBox: ((rightPane frame origin)
                                extent:( rightPane frame extent))!

```

intersection

| a b |

```

bottomPane clear.
a:=resultImageA deepCopy.
b:=resultImageB deepCopy.
CursorManager execute change.
a intersection: b.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
    clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!

```

leftPaneMenu

```

^Menu
    labels: 'get it\load IDB\from result\from disk' withCrs
    lines: #(0)
    selectors: #( getIt openIdb fromResult fromDisk)!

```

openIdb

```

(ImageDataBaseInspector allInstances size)=0
    ifFalse:[^nil].

```

object open!

opening

```
| a b |
```

```
bottomPane clear.
a:=resultImageA deepCopy.
b:=resultImageB deepCopy.
CursorManager execute change.
a openingBy: b.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
  clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!
```

openOn:anObject

"To open an image processing environment"

```
| anIdbTopPane |
object:=anObject.
indexSet:=Array new:4.
anIdbTopPane := IdbTopPane new.
anIdbTopPane
  label: 'Demo Image Processing';
  model: anIdbTopPane dispatcher;
  menu: #workspaceMenu;
  minimumSize: 80@80;
  yourself.
```

```
anIdbTopPane addSubpane:
  (leftPane:=GraphPane new
  menu: #leftPaneMenu;
  model: self;
  name: #instanceLft;;
  framingRatio: (0 @ 0
    extent: 3/7 @ (1/2))).
```

```
anIdbTopPane addSubpane:
  (middlePane:=GraphPane new
  menu: #rightPaneMenu;
  model: self;
  name: #instanceMid;;
  framingRatio: (3/7 @ 0
    extent: 2/7 @ (1/2))).
```

```
anIdbTopPane addSubpane:
  (rightPane:=GraphPane new
  menu: #rightPaneMenu;
  model: self;
  name: #instanceRit;;
  framingRatio: (5/7 @ 0
    extent: 2/7 @ (1/2))).
```

```
anIdbTopPane addSubpane:
  (bottomPane:=GraphPane new
  menu: #bottomPaneMenu;
  model: self;
  name: #instanceBtm;;
  framingRatio: (0 @ (1/2))
```



```
extent: 1 @ (1/2))).
```

```
CursorManager normal change.
self setInstList.
anIdbTopPane dispatcher open scheduleWindow!
```

printResult

```
"To print out the result image on pin-printer"
^resultImage outputToPrinterUpright!
```

reflect

```
| a |

bottomPane clear.
a:=resultImageA deepCopy.
CursorManager execute change.
a reflect.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
  clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))!
```

rightPaneMenu

```
^Menu
  labels: 'get it\from disk\from result' withCrs
  lines: #(0)
  selectors: #( getIt fromDisk fromResult)!
```

setInstList

```
"Private - Compute instList, an
  OrderedCollection of key strings
  for the list pane."
| aSet |
aSet := Set new: object size.
object keysDo: [:aKey |
  aSet add:
    (Association key: aKey value: aKey printString)].
instIndex := nil.
(instList := SortedCollection new)
  sortBlock: [:a :b| a value < b value];
  addAll: aSet!
```

storeOnFile

```
" To save selected image on a disk file."

| aPrompter |
instIndex isNil
  ifTrue: [^self].
aPrompter:=Prompter prompt:'file name?'
  default:fileName .
fileName:= aPrompter.
(fileName=nil) ifTrue:[^nil].
(fileName size=0) ifTrue:[^nil].
```

```
(resultImage class name)='Image' ifTrue:[
  CursorManager execute change.
  resultImage storeOnFile:fileName.
  CursorManager normal change.]
  ifFalse:[
  CursorManager execute change.
  resultImage storeColorOnFile:fileName.
  CursorManager normal change.]!
```

symmetricDiff

```
| a b |

bottomPane clear.
a:=resultImageA deepCopy.
b:=resultImageB deepCopy.
CursorManager execute change.
a symmetricDiff: b.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
  clippingBox: ((bottomPane frame origin)
  extent:( bottomPane frame extent))!
```

thicken

```
| a b c |

bottomPane clear.
a:=resultImageA deepCopy.
b:=resultImageB deepCopy.
c:=resultImageC deepCopy.
CursorManager execute change.
a thickeningBy: b and: c.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
  clippingBox: ((bottomPane frame origin)
  extent:( bottomPane frame extent))!
```

thin

```
| a b c |

bottomPane clear.
a:=resultImageA deepCopy.
b:=resultImageB deepCopy.
c:=resultImageC deepCopy.
CursorManager execute change.
a thinningBy: b and: c.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
  clippingBox: ((bottomPane frame origin)
  extent:( bottomPane frame extent))!
```

union

```
| a b |
```

```

bottomPane clear.
a:=resultImageA deepCopy.
b:=resultImageB deepCopy.
CursorManager execute change.
a union: b.
CursorManager normal change.
resultImage:=a deepCopy.
^resultImage displayAt: bottomPane frame origin
  clippingBox: ((bottomPane frame origin)
    extent:( bottomPane frame extent))! !

```

Appendix E. Classes and methods of Image(V286)

Form subclass: **#Image**

```

instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: '' !

```

!Image class methods !

fromFile: fileName

```

| inFile aBitmap w h anImage wTemp|
anImage:=Image new.
inFile := Disk file: fileName.
inFile reset.

1 to: 18 do[:i| inFile next].

h := inFile next asciiValue.
wTemp := inFile next asciiValue.
wTemp := (wTemp bitShift: 8) bitAnd:65280.
w := wTemp bitOr: h.
inFile next.
inFile next.
h := inFile next asciiValue.
wTemp := inFile next asciiValue.
wTemp := (wTemp bitShift: 8) bitAnd:65280 deepCopy.
h := wTemp bitOr: h.
anImage := (anImage width: w height: h) .
((w\\16)=0 ) ifFalse:[
w:=w+(16 - (w\\16)) deepCopy].

1 to:36 do[:i| inFile next].

aBitmap := Bitmap new: w*h/8.
(1 to: w*h/8) do:
  [ :i | aBitmap at: i put: inFile next asciiValue].
anImage bitmap: aBitmap .
^anImage!

```

idbFromFile: fileName

"To retrieve the stroed image from disk-- both color image and black and white image."

```

| inFile aBitmap aBitmap1 aBitmap2 aBitmap3 aBitmap4
  anArray w h anImage wTemp head length|
inFile := Disk file: fileName.
inFile reset.

```

```

head:= inFile nextWord.
(head='color') iffFalse:[ "black an white"
  inFile reset.
  h := inFile next asciiValue.
  wTemp := inFile next asciiValue.
  wTemp := (wTemp bitShift: 8) bitAnd:65280.
  w := wTemp bitOr: h.
  h := inFile next asciiValue.
  wTemp := inFile next asciiValue.
  wTemp := (wTemp bitShift: 8) bitAnd:65280 deepCopy.
  h := wTemp bitOr: h.
  anImage := Image new width: w height: h .
  ((w\\16)=0 ) iffFalse:[
  w:=w+(16 - (w\\16)) deepCopy].
  aBitmap := Bitmap new: w*h/8.
  (1 to: w*h/8) do:
    [ :i | aBitmap at: i put: inFile next asciiValue].
  anImage bitmap: aBitmap .
  ^anImage
    ].

```

```

inFile next.
h := inFile next asciiValue.
wTemp := inFile next asciiValue.
wTemp := (wTemp bitShift: 8) bitAnd:65280.
w := wTemp bitOr: h.
h := inFile next asciiValue.
wTemp := inFile next asciiValue.
wTemp := (wTemp bitShift: 8) bitAnd:65280 deepCopy.
h := wTemp bitOr: h.
anImage := (ColorForm new width: w height: h).
((w\\16)=0 ) iffFalse:[
w:=w+(16 - (w\\16)) deepCopy].
length:=w*h/8 deepCopy.
aBitmap1:= Bitmap new:length.
aBitmap2:= Bitmap new:length.
aBitmap3:= Bitmap new:length.
aBitmap4:= Bitmap new:length.
(1 to: length) do:[ :i | aBitmap1 at: i put: (inFile next
asciiValue)].
(1 to: length) do:[ :i | aBitmap2 at: i put: (inFile next
asciiValue)].
(1 to: length) do:[ :i | aBitmap3 at: i put: (inFile next
asciiValue)].
(1 to: length) do:[ :i | aBitmap4 at: i put: (inFile next
asciiValue)].
anArray:= Array with:aBitmap1
                with:aBitmap2
                with:aBitmap3
                with:aBitmap4.
anImage bitmap:anArray.
^anImage! !

```

!Image methods !

center

"Answer a Point, the center of the receiver."

```
^((self width- 1)@(self height- 1)//2)!
```

closingBy:imageR

"The opening operation is an dilation followed by a erosion with the same reference imageR ."

```
(self dilationBy:imageR) erosionBy:imageR!
```

complement

"Answer the complement of an image."

```
^self reverse!
```

difference:imageR

"Answer an image that containing the difference between original image (imageX) and reference image (imageR)"

```
((self complement) union: imageR ) complement!
```

dilationBy:imageR

"Dilation an image (imageX) by a reference image (imageR)"

```
| a b c |
b:= (Image width: (self width) height: (self height)) black.
c:= (Image width: (self width) height: (self height)) black.
a:= OrderedCollection new.
a:= imageR getPointsFrom:imageR.
1 to: a size do:[:i|
    b copy: (0@0 extent:(self extent))
      from: self
      to: ((a at:i)- (imageR center))
      rule: 3.
    c union:b
  ] .
self copy: (0@0 extent:(self extent)) from:c to:0@0 rule:3!
```

erosionBy:imageR

"Answer an image that containing the erosion of original image (imageX) by reference image (imageR)"

```
((self complement) dilationBy:(imageR reflect)) complement!
```

getPointsFrom:imageR

"Method to get the position of each white pixel (foreground) of the reference image, answer a ordered collection containing white points."

```
| a |
a:=OrderedCollection new.
0 to: (imageR height- 1) do:[:j|
    0 to: (imageR width- 1) do:[:i|
        ((imageR at:(i@j))=1) ifTrue:[a add:i@j]]] .
^a!
```

hitMissTransBy:imageR1 and:imageR2

"The hit or miss transform of an image pair R=(R1,R2) is used to match the shape (or template) defined by the reference image pair R where R1 defines the for ground of the shape and R2 defines the background of the shape."

```
| a b |
a:= (Image width: (self width) height: (self height)) black.
a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
```

```

a erosionBy:imageR1.
b:= (Image width: (self width) height: (self height)) black.
b copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
(b complement) erosionBy:imageR2.
a intersection:b.
self copy: (0@0 extent:(self extent)) from:a to:0@0 rule:3!

```

intersection:imageR

"Answer an image that containing the intersection of original image (imageX) and reference image (imageR)"

```
((self complement) union:(imageR complement)) complement!
```

label

" This is the method for counting and labeling blobs. Based on Algorithm 5.1: Blob Coloring. Dana H.Ballard & Christopher M.Brown Computer Vision, 621.380 414 BAL pp.151 The author made some improvement"

```

|label n row column|
n:=1.
row:=self height.
column:=self width.
label:=Matrix new:row with:column.
0 to: (row- 1) do[:j|
    0 to: (column- 1) do[:i|

        i:=i+1. j:=j+1.

        ((self at:((i- 1)@(j- 1)))=1) ifTrue:[

            (((label wNbr:j at:i)=0)and:[(label nNbr:j at:i)=0])
ifTrue:[
                label at:j at:i put:n.
                n:=n+1.
                "New color"
            ].

            (((label wNbr:j at:i)~=0)and:[(label nNbr:j at:i)=0])
ifTrue:[
                label at:j at:i put:(label wNbr:j at:i).
                "Has left neighbour, but no upper neighbour"
            ].

            (((label wNbr:j at:i)=0)and:[(label nNbr:j at:i)~=0])
ifTrue:[
                label at:j at:i put:(label nNbr:j at:i)
            ].
                "Has upper neighbour, but no left neighbour"
            ].

            (((label wNbr:j at:i)~=0)and:[(label nNbr:j at:i)~=0])
ifTrue:[
                label at:j at:i put:(label nNbr:j at:i).
                label westNorth:j at:i
            ].

            "Has both upper and left neighbour, set three
color
            same as upper neighbour color and fill all
non-zero
            left value same as upper neighbour."
                ] ifFalse:[
                    label at:j at:i put:0.
                ].

```

```

        i:=i- 1. j:=j- 1.
        ]
    ]

```

^label!

openingBy:imageR

"The opening operation is an erosion followed by a dilation with the same reference imageR ."

```
(self erosionBy:imageR) dilationBy:imageR!
```

reflect

"Answer an image which containing the Reflected Reference Image ."

```

| a b |
a:= OrderedCollection new.
b:= OrderedCollection new.
a:= self getPointsFrom: self.
1 to: a size do: [:i| b add:(self center- (a at:i)+self center ).
                    self at:(a at:i) put:0].
1 to: b size do:[:i| self at:(b at:i) put: 1] .!

```

storeOnFile: fileName

```

| outFile |
outFile := Disk newFile: fileName.
"Delete existing contents"
outFile nextTwoBytesPut: (self width) .
outFile nextTwoBytesPut: (self height) .
self bitmap do:
    [:ea | outFile nextPut: ea asCharacter].
outFile close.!

```

symmetricDiff:imageR

"Answer an image that containing the symmetric difference between original image (imageX) and reference image (imageR)"

```

| a b |
a:= (Image width: (self width) height: (self height)) black.
a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
a difference:imageR.
b:= (Image width: (self width) height: (self height)) black.
b copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
imageR difference:b.
a union:imageR.
self copy: (0@0 extent:(self extent)) from:a to:0@0 rule:3!

```

thickeningBy:imageR1 and:imageR2

"The thinning operation is extensive and increases the size by filling the image points where the regions match the reference image pair R = (R1,R2)."

```

| a |
a:= (Image width: (self width) height: (self height)) black.

```

```
a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
a hitMissTransBy:imageR1 and:imageR2.
self union: a!
```

thinningBy:imageR1 and:imageR2

"The thinning operation is antiextensive and decreases the size by removing the central points of the regions which match the reference image pair $R = (R1, R2)$."

```
| a |
a:= (Image width: (self width) height: (self height)) black.
a copy: (0@0 extent:(self extent)) from:self to:0@0 rule:3.
a hitMissTransBy:imageR1 and:imageR2.
self difference: a!
```

union: imageR

"Answer an image containing the image of the union of imageR and the receiver imageX. Usage:-- imageX union: imageR"

```
self copy: (0@0 extent:(imageR extent)) from:imageR to:0@0
rule:7 .! !
```

Appendix F. Classes and methods of ImageLab

F.1 Methods in class Bitmap

```
GraphicsMedium subclass: #Bitmap
instanceVariableNames:
'bitmapHandle bitmapInfo archive '
classVariableNames:
'AuxillaryDC OldMemoryContext MemoryContext DefaultBitmap
BitmapHandleTable '
poolDictionaries:
'WinConstants ' !
```

!Bitmap class methods !

No class methods has been add in in this project.

!Bitmap methods !

center

"Prvite: --- Answer a Point, the center of the receiver."

```
^((self width)@(self height)//2)!
```

complement

" To reverse the receiver image."

```
self pen copyBitmap: self
from: (0 @ 0 extent: ((self width) @ (self height)))
at: 0 @ 0
rule:Dstinvert.
^self!
```


difference:imageR

" Answer an image that containing the difference between original image and referrence image."

^(((self complement) union:imageR) complement)!

extractColor:aColor1 by:bitmapR rColor:aColor2

" Extract a color by a reference bitmap (bitmapR) on a specified color."

| a b c |

```

a := OrderedCollection new.
a:= self getPoints: aColor1.           " Original image "
b:= bitmapR getPoints: aColor2.       " Refference image "
a isEmpty ifTrue:[ ^ self].
b isEmpty ifTrue:[ ^ self].
c := bitmapR center.
1 to: a size do[:i|
    1 to: b size do[:j|
        self setPixelAt: ((a at: i) + ((b at: j)- c))
            withColor: aColor1
    ]
].
^self!
```

getPixel:aPoint

" Answer a color value at point (x ,y)"

```

^GraphicsTool rgbToPalette:
    (GDILibrary getPixel: self pen handle
        x: aPoint x
        y: aPoint y )!
```

getPoints:aColor

" Get the points of aColor from a bitmap. Answer an ordered collection which contained the co-ordinate of this points."

```

| result center|

center:=self center.
result := OrderedCollection new.
0 to: ((self height) - 1) do[:h|
    0 to:((self width) - 1) do[:w|
        (self getPixel:(w@h)) = aColor ifTrue:[
            result add:((w@h)- center)]
    ]
].
^result!
```

isBitmap

"Answer true if receiver is an instance of class Bitmap or one of its subclasses, else answer false."

^true!

label1: aColor

"This is the method for labeling specified color blobs(the V/win version. Based on Algorithm 5.1: Blob Coloring. Dana H.Ballard & Christopher M.Brown Computer Vision, 621.380 414 BAL pp.151. The Author made some improvement."

```

|label n row column|
n:=1.
row:=self height.
column:=self width.
label:=Matrix new:row with:column.
0 to: (row- 1) do:[:j|
    0 to: (column- 1) do:[:i|

        i:=i+1. j:=j+1.

        ((self getPixel:((i- 1)@(j- 1)))= aColor) ifTrue:[

            ((label wNbr:j at:i)=0)and:[(label nNbr:j at:i)=0]]
ifTrue:[
            label at:j at:i put:n.
            n:=n+1.
            "New color"
        ].

        ((label wNbr:j at:i)~=0)and:[(label nNbr:j at:i)=0]]
ifTrue:[
            label at:j at:i put:(label wNbr:j at:i).
            "Has left neighbour, but no upper neighbour"
        ].

        ((label wNbr:j at:i)=0)and:[(label nNbr:j at:i)~=0]]
ifTrue:[
            label at:j at:i put:(label nNbr:j at:i).
            label nUp:j at:i
            ].
            "Has upper neighbour, but no left neighbour"
        ].

        ((label wNbr:j at:i)~=0)and:[(label nNbr:j at:i)~=0]]
ifTrue:[
            label at:j at:i put:(label nNbr:j at:i).
            label westNorth:j at:i
            ].

            "Has both upper and left neighbour, set three
color
            same as upper neighbour color and fill all
non-zero
            left value same as upper neighbour."
            ] ifFalse:[
            label at:j at:i put:0.
            ].

            i:=i- 1. j:=j- 1.

            ]
        ].

    ]
^label!

```

outputToFile: aFileName

"Output the receiver in DIB format to aFileName."

```

| file fileHeader colors bitmapHeader bits |
bits := self getDIBits.
bits isNil ifTrue: [^self].
fileHeader := WinBitmapFileHeader new.
fileHeader bfType: 'BM';
    bfSize: 14 + bitmapInfo contents size + bits size;
    reserved1: 0;

```

```

    reserved2: 0;
    offBits: 14 + bitmapInfo contents size.
    file := (File pathName: aFileName) asByteFileStream.
    file nextPutAll: fileHeader contents.
    bitmapInfo contents do: [: aByte |
        file nextPut: aByte asCharacter].
    bits do: [: aByte |
        file nextPut: aByte asCharacter].
    file close!

```

outputToPrinter

"Print the receiver on the system printer."

```

| printer |
CursorManager execute change.
printer := Printer new.
printer startPrintJob.
printer pen copyBitmap: self
    from: self boundingBox
    to: (self boundingBox scaleBy: (
        (printer pen width // Display width) min: (
            printer pen height // Display height))).
printer endPrintJob.
CursorManager normal change!

```

reflectReal

" Answer a bitmap which containing the reflected reference bitmap."

```

| bits rBits size |

bits := self getDIBits.
size := bits size.
rBits := ByteArray new:( size ).
0 to: (size- 1) do:[each|
    rBits at:(each+1) put:(( bits at:( size - each)) reverse) ].
self setDIBits: rBits.
^self!

```

release

"Delete the receiver from the device context. Freeing up all system storage associated with the receiver."

```

self handle isNil ifTrue: [^self].
deviceContext = MemoryContext
    ifTrue: [self deselect]
    ifFalse: [graphicsTool deleteDC].
GDILibrary deleteObject: bitmapHandle.
BitmapHandleTable removeKey: bitmapHandle ifAbsent: [ ].
graphicsTool := nil.
bitmapHandle := nil!

```

setPixelAt:aPoint withColor:aColor

" Set a color value at point (x ,y) with color aColor"

```

^GDILibrary setPixel: self pen handle
    x: aPoint x
    y: aPoint y
    color: aColor!

```

symmetricDiff:imageR

" Answer an image that containing the symmetric difference between original image and referrence image."

```

| a b c|

c := Bitmap width: self width
      height: self height
      planes: self planes
      bitCount: self bitCount .

imageR displayAt:0@0 with:c pen.

a := Bitmap width: self width
      height: self height
      planes: self planes
      bitCount: self bitCount .
b := Bitmap width: self width
      height: self height
      planes: self planes
      bitCount: self bitCount .
a pen copyBitmap: self
@ (self height))
      from: (0 @ 0 extent: ((self width)
      at: 0 @ 0
      rule:Srcrcopy.
a difference:c.
b pen copyBitmap: self
@ (self height))
      from: (0 @ 0 extent: ((self width)
      at: 0 @ 0
      rule:Srcrcopy.
c difference:b.
a union:c.
self pen copyBitmap: a
@ (a height))
      from: (0 @ 0 extent: ((a width)
      at: 0 @ 0
      rule:Srcrcopy.
a release.
b release.
c release.
^self!
```

union: aBitmap

" To union two bitmaps the receiver and aBitmap. Use raster operation Srcpaint."

```

self pen copyBitmap: aBitmap
@ (self height))
      from: (0 @ 0 extent: ((self width)
      at: 0 @ 0
      rule:Srcpaint.
^self! !
```

```

Bitmap subclass: #Image
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries:
'WinConstants ' !
```

!Image class methods !**fromBitmap:aBitmap**

" Answer an Image that is copied from a bitmap."

```

| b |
b := Image width: aBitmap width
      height: aBitmap height
      planes: aBitmap planes
      bitCount: aBitmap bitCount .
aBitmap displayAt: 0@0 with:b pen.
^b! !

```

!Image methods !**built4N**

" To built the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. A cross, 4-connected neighbour."

```

| anOc |
anOc:=OrderedCollection new.
anOc add:0@0.
anOc add:0@1.
anOc add:1@0.
anOc add:0@-1.
anOc add:-1@0.
^anOc!

```

built8N

" To built the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. A 3 by 3 square, 8-connected neighbour."

```

| anOc |
anOc:=self built4N.
anOc add:1@1.
anOc add:-1@1.
anOc add:1@-1.
anOc add:-1@-1.
^anOc!

```

builtCap

" To built the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. A cross, 4-connected neighbour."

```

| anOc |
anOc:=OrderedCollection new.
anOc add:-3@1.
anOc add:-1@-1.
anOc add:-2@0.
anOc add:0@-2.
anOc add:3@1.
anOc add:2@0.
anOc add:1@-1.
^anOc!

```

builtE:anInteger

" To built the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. To EAST direction."

```
| anOc |
anOc:=OrderedCollection new.
0 to: anInteger do[:i|
    anOc add:( i@0 )].
^anOc!
```

builtH:anInteger

" To built the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. Horizontal direction."

```
| anOc |
anOc:=OrderedCollection new.
(anInteger negated) to: anInteger do[:i|
    anOc add:( i@0 )].
^anOc!
```

builtLD:anInteger

" To built the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. Left Diagonal direction."

```
| anOc |
anOc:=OrderedCollection new.
(anInteger negated) to: anInteger do[:i|
    anOc add:( i@i )].
^anOc!
```

builtN:anInteger

" To built the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. To NORTH direction."

```
| anOc |
anOc:=OrderedCollection new.
0 to: anInteger do[:i|
    anOc add:( 0@(i negated))].
^anOc!
```

builtNE:anInteger

" To built the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. To NORTH EAST direction."

```
| anOc |
anOc:=OrderedCollection new.
0 to: anInteger do[:i|
    anOc add:( i@(i negated))].
^anOc!
```

builtNW:anInteger

" To build the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. To NORTH WEST direction."

```
| anOc |
anOc:=OrderedCollection new.
0 to: anInteger do[:i|
    anOc add:( (i negated)@(i negated))].
^anOc!
```

builtRD:anInteger

" To build the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. Right Diagonal direction."

```
| anOc |
anOc:=OrderedCollection new.
(anInteger negated) to: anInteger do[:i|
    anOc add:( (i negated)@i )].
^anOc!
```

builtS:anInteger

" To build the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. To SOUTH direction."

```
| anOc |
anOc:=OrderedCollection new.
0 to: anInteger do[:i|
    anOc add:( 0@i)].
^anOc!
```

builtSW:anInteger

" To build the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. To SOUTH WEST direction."

```
| anOc |
anOc:=OrderedCollection new.
0 to: anInteger do[:i|
    anOc add:( (i negated)@i)].
^anOc!
```

builtTriangle

" To build the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element."

```
| anOc |
anOc:=self built4N.
anOc add:1@1.
anOc add:-1@1.
anOc add:-2@1.
anOc add:2@1.
^anOc!
```

builtV:anInteger

" To built the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. Vertical direction."

```
| anOc |
anOc:=OrderedCollection new.
(anInteger negated) to: anInteger do[:i|
    anOc add:( 0@i )].
^anOc!
```

builtW:anInteger

" To built the structure element, anser an OrderedCollection that contains the coordinate sets of 1's in a structure element. To WEST direction."

```
| anOc |
anOc:=OrderedCollection new.
0 to: anInteger do[:i|
    anOc add:( (i negated)@0 )].
^anOc!
```

closingBy:anOc

" Answer an image that containing the opening between original image and referrence image."

```
^((self dilationBy: anOc) erosionBy: anOc)!
```

convexHull:anInteger

" Answer the convex hull of an image."

```
| a b c d |
a := Image width: self width
    height: self height
    planes: self planes
    bitCount: self bitCount .
self displayAt:0@0 with: a pen.
b := Image width: self width
    height: self height
    planes: self planes
    bitCount: self bitCount .
self displayAt:0@0 with: b pen.
c := Image width: self width
    height: self height
    planes: self planes
    bitCount: self bitCount .
self displayAt:0@0 with: c pen.
d := Image width: self width
    height: self height
    planes: self planes
    bitCount: self bitCount .
self displayAt:0@0 with: d pen.
a dilationBy:( self builtH:anInteger).
b dilationBy:( self builtV:anInteger).
c dilationBy:( self builtLD:anInteger).
d dilationBy:( self builtRD:anInteger).
a intersection:b.
a intersection:c.
a intersection:d.
a displayAt:0@0 with: self pen.
a release.
```



```
b release.
c release.
d release.
^self!
```

damond:anInteger

```
" Change the object in an image into a damond shape."

| c d |
c := Image width: self width
      height: self height
      planes: self planes
      bitCount: self bitCount .
self displayAt:0@0 with: c pen.
d := Image width: self width
      height: self height
      planes: self planes
      bitCount: self bitCount .
self displayAt:0@0 with: d pen.
c dilationBy:( self builtLD:anInteger).
d dilationBy:( self builtRD:anInteger).
c intersection:d.
c displayAt:0@0 with: self pen.
c release.
d release.
^self!
```

dilationBy:anOc

" Dilation an bitmap (bitmapX) by a set of points in an OrderedCollection."

```
| b e |
b := Image width: self width
      height: self height
      planes: self planes
      bitCount: self bitCount .
self displayAt: 0@0 with:b pen.
e := self extent.
1 to: anOc size do[:i|
  b pen copyBitmap: self
    from: (0@0 extent: e )
    at: ((anOc at:i))
    rule: Srcpaint.
] .
b displayAt: 0@0 with:self pen.
b release.
^self!
```

edgeTraceBy:anOc

" Answer an image that containing the edge traced image."

```
| temp |

temp:= Image width: self width
      height: self height
      planes: self planes
      bitCount: self bitCount.
self displayAt:0@0 with:temp pen.
self difference:(temp erosionBy:anOc).
temp release.
^self!
```

erosionBy:anOc

" Answer an image that containing the erosion between original image and referrence image."

```
^(((self complement) dilationBy:(self ocReflect:anOc)
complement)!
```

gradientBy:anOc

" Answer an image that containing the gradient image."

```
| templ |
```

```
templ:= Image width: self width
           height: self height
           planes: self planes
           bitCount: self bitCount.
self displayAt:0@0 with:templ pen.
templ erosionBy:anOc.
self dilationBy:anOc.
self difference:templ.
templ release.
^self!
```

hmtBy:anOc1 and:anOc2

"The hit or miss transform(V/win version) of an image pair R=(R1,R2) is used to match the shape (or template) defined by the reference image pair R where R1 defines the for ground of the shape and R2 defines the background of the shape."

```
| b |
b := Image width: self width
           height: self height
           planes: self planes
           bitCount: self bitCount .
self displayAt:0@0 with: b pen.
b erosionBy:anOc1.
(self complement) erosionBy:anOc2.
           self dilationBy:(self builtSW:2).
self intersection:b.
b release.
^self!
```

intersection:imageR

" Answer an image that containing the intersection between original image and referrence image."

```
self pen copyBitmap: imageR
           from: (0 @ 0 extent: ((self width)
@ (self height)))
           at: 0 @ 0
           rule:Srcand.
^self!
```

label:anInteger

"Another method to label the connected components. Given an NXN image X which consists of k connected components(each with size

larger than M X M , M is larger than the bottom of the triangle which is the foreground pattn), label each connected component by a single image point (the upper left image point of its damond)"

```
self damond:anInteger.
self dilationBy:(self builtW:1).
self hmtBy:(self builtTriangle) and:(self builtCap).!
```

ocReflect:anOc

"Answer the reflect of a set of points in an ordered collection.

$$R = \{(-x, -y) \mid (x,y) \text{ belong to } R\}."$$

```
| ocl |
ocl:=OrderedCollection new.
1 to: anOc size do[:i| ocl add:((anOc at:i) negated)].
^ocl!
```

openingBy:anOc

" Answer an image that containing the opening between original image and referrence image."

```
^((self erosionBy: anOc) dilationBy: anOc)!
```

rt:anInteger

" Answer the convex hull of an image."

```
| a b c d |
a := Image width: self width
           height: self height
           planes: self planes
           bitCount: self bitCount .
self displayAt:0@0 with: a pen.
b := Image width: self width
           height: self height
           planes: self planes
           bitCount: self bitCount .
self displayAt:0@0 with: b pen.
a dilationBy:( self builtH:anInteger).
b dilationBy:( self builtV:anInteger).
a intersection:b.
a displayAt:0@0 with: self pen.
a release.
b release.
^self!
```

thickeningBy:imageR1 and:imageR2

"The thicking operation is extensive and increases the size by filling the image points where the regions match the reference image pair R = (R1,R2)."

```
| a |
a := Image width: self width
           height: self height
           planes: self planes
           bitCount: self bitCount .
self displayAt:0@0 with: a pen.
a hmtBy:imageR1 and:imageR2.
self union: a.
a release!
```

thinningBy:imageR1 and:imageR2

"The thinning operation is antiextensive and decreases the size by removing the central points of the regions which match the reference image pair R = (R1,R2)."

```

| a |
a := Image width: self width
           height: self height
           planes: self planes
           bitCount: self bitCount .
self displayAt:0@0 with: a pen.
a hmtBy:imageR1 and:imageR2.
self difference: a.
a release! !

```

F.2 Methods in Class ImageLab

GraphLab subclass: #ImageLab

```

instanceVariableNames:
  'image selImage fileName anOc '
classVariableNames: ''
poolDictionaries:
  'ColorConstants WinConstants VirtualKeyConstants ' !

```

!ImageLab class methods !

mdiMenu

" Answer the standard MDI Window menu "

```

^MDIMenu new
  appendItem: '&New Text Window' selector:#createDocuments
  accelKey:$n accelBits: AfControl;
  appendItem: '&Cascade Shift+F5' selector: #mdiCascade
  accelKey: VkF5 accelBits: AfVirtualkey|AfShift ;
  appendItem: '&Tile Shift+F4' selector: #mdiTile accelKey: VkF4
  accelBits: AfVirtualkey|AfShift ;
  appendItem: 'Arrange &Icons' selector: #mdiArrange;
  appendItem: 'Close &All' selector: #mdiCloseAll;
  title: '&Window'!! !

```

!ImageLab methods !

aboutV

" Open the dialog box about Image Lab."

```

AboutImageLabDialog new open!

```

activeImageProPane

" Answer current active ImageProPane."

```

|a |

a:=self mainView mdiChildren.
1 to: (a size) do:[:i|
  (a at:i) isActive ifTrue:[

```

```

        ( (a at:i) children at:1) isImageProPane ifTrue:[
            ^((a at:i) children at:1)]].
    MessageBox message:'Current window is not an Image
Workspace!!'.
    ^nil!

```

anaMenu

"Answer a menu with a list of image analysis operations."

```

^ (MDIMenu
  labels: '&Convex Hull\&Label\&Hit Miss Trans.' withCrs
  lines: #(1 )
  selectors: #(convexHull label hmt )
  owner: self;
  title: '&Analysis';
  yourself!

```

blobSize

"Define the blob size."

```

| rect |
rect := Display rectangleFromUser.
rect isNil ifTrue: [rect := 0 @ 0 extent: 4 @ 4].
^( rect width max: rect height)!

```

childActivate:aPane

" Update the StatusPane, current working pane and its pen. Shows the label of the active MDI document."

```

|mdiActive|
(mdiActive := self frame mdiGetActive) notNil ifTrue:[
  (self statusPane statusBoxAt: #status) contents: mdiActive
label].
pane:=mdiActive children at:1 .
pen:=pane pen!

```

childClose:aPane

"Update the StatusPane. If there is no more document, erase the StatusBox #status"

```

self frame mdiGetActive isNil ifTrue: [
  (self statusPane statusBoxAt: #status) contents: ''].!

```

close:aPane

" Close the receiver. "

```

|answer|
Smalltalk isRunTime
  ifTrue:[
    (MessageBox confirm:'Are you sure you want to
exit?')
      ifTrue:[self close.
        (Bitmap allInstances ) do:[:ea|ea
become: String new].
          ^Smalltalk exit]
      ifFalse:[^self]]
  ifFalse:[
    (Bitmap allInstances ) do:[:ea|ea
become: String new].
    ^self close]!

```

closing

```
" Opening the loaded image with selected structure element."

self imageCheck isNil ifTrue:[^nil].
CursorManager execute change.
image closingBy:anOc.
CursorManager normal change.
self display:image!
```

complement

```
" Answer the complement of original image."
```

```
image isNil ifTrue:[MessageBox message:'Open an image or
select from screen!!'.
                                ^nil].
image complement.
self display:image!
```

convexHull

```
" Answer the convex hull of image."
```

```
| blobSize |
blobSize:=self blobSize.
self imageCheck isNil ifTrue:[^nil].
CursorManager execute change.
image convexHull:blobSize.
CursorManager normal change.
self display:image!
```

copyGraph

```
"Copy a portion of the receiver's contents to the
clipboard."
```

```
| bitmap |
bitmap := Bitmap fromUser.
bitmap isNil ifTrue:[^nil].
image:= Image fromBitmap:bitmap.
Clipboard setBitmap: image.!
```

createDocuments

```
" Create the MDI documents"
```

```
|buffer winaddress pathName |
buffer:=String new: 160.
winaddress:=WinAddress copyToNonSmalltalkMemory: buffer.
KernelLibrary getWindowsDirectory:winaddress asParameter
length:buffer size.
pathName:=String fromAddress:winaddress.
winaddress unlockAndFree.

ILTextWindow new
    frame:self frame;
    label:'untitled.txt';
    icon:(Icon fromModule:self resourceDLLFile
id:'TextWindow');
    openOn:''.!
```

cross

"Private -- Answer a 3X3 cross bitmap."

```
| aByteArray index |
CursorManager execute change.
index:=1.
aByteArray:=ByteArray new: 124 .
```

```
 #(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 1 0 0 0 3 128 0 0 1 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
```

do:[:i|

```
  aByteArray at:index put:i.
  index:=index+1].
```

```
seImage:=Image new width:31
                                     height:31
                                     bits:aByteArray.
```

```
anOc:=seImage built4N.
CursorManager normal change.
self display:seImage.
^seImage!
```

differ

" Answer the difference of two images."

```
self imageCheck isNil ifTrue:[^nil].
image difference:seImage.
self display:image.!
```

dilation

" Dilate the loaded image with selected structure element."

```
| answer |
self imageCheck isNil ifTrue:[^nil].
answer := Prompter prompt: 'How many times?' default:'1'.
(answer = nil) ifTrue:[ answer:='1'].
CursorManager execute change.
1 to: (answer asInteger) do:[:i|
image dilationBy:anOc].
CursorManager normal change.
self display:image.!
```

disk

" Answer a 6X6 disk bitmap."

```
| aByteArray index |
CursorManager execute change.
index:=1.
aByteArray:=ByteArray new: 128 .
```

```
 #(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   7 224 0 0 31 240
   0 0 63 252 0 0 127 254 0 0 255 255 0 0 255 255 0 1 255
  255 128 1 255
   255 128 1 255 255 128 1 255 255 128 1 255 255 128 1 255
  255 128 1 255
```


fileMenu

```
" Answer the receiver's fileMenu"
```

```
  ^Menu new
```

```
    appendItem: 'Ope&n an image workspace' selector: #imagePane;
    appendItem: 'Open a text &workspace' selector:
#createDocuments;
    appendItem: 'Open a text &file' selector: #openTextWindow;
    appendSeparator ;
    appendItem: '&About Image Lab' selector: #aboutV ;
    appendItem: '&Exit' selector: #exit ;
    title: '&Open'.!
```

gradient

```
" Gradient the loaded image with selected structure
element."
```

```
self imageCheck isNil ifTrue:[^nil].
CursorManager execute change.
image gradientBy:anOc.
CursorManager normal change.
self display:image.!
```

grayScalePal

```
"Change the system palette in 256 gray scale for the pseudo
gray scale operation."
```

```
  | bM handle hdc hPalette hPrevious bA logPalette colorTab |
```

```
colorTab := ByteArray new: 1024.
```

```
(0 to: 255) do: [ :i |
  (1 to: 4) do: [ :j |
    colorTab at: 4*i + j put: i
  ] ].
```

```
(0 to: 255) do: [ :i | colorTab at: (4*i + 4) put: 0].
```

```
hdc := UserLibrary getDC: nil.
```

```
GDIlibrary setSystemPaletteUse: hdc wUsage:SyspalNostatic.
GDIlibrary unrealizeObject: hdc .
```

```
logPalette := WinStructure new: (256 * 4 + 4).
```

```
logPalette uShortAtOffset: 0 put: 16r300;
```

```
uShortAtOffset: 2 put: (1 bitShift: 8).
```

```
logPalette contents
```

```
  replaceFrom: 5
```

```
            to: logPalette size
```

```
            with: colorTab.
```

```
bA:=GDIlibrary createPalette: logPalette asParameter.
```

```
hPrevious := UserLibrary selectPalette: hdc
```

```
with:
```

```
bA
```

```
forceBackground: true.
```

```
UserLibrary realizePalette: hdc.
```

```
UserLibrary selectPalette: hdc
```

```
with: hPrevious
```

```
forceBackground: false.
```

```
GDIlibrary deleteObject: bA .
UserLibrary releaseDC: nil with: hdc.!
```

gsInvert

" Change the system palette in invert form of 256 gray scale for the pseudo gray scale operation."

```
| bM handle  hdc hPalette hPrevious bA logPalette colorTab |
```

```
colorTab := ByteArray new: 1024.
(0 to: 255) do: [ :i |
    (1 to: 4) do: [ :j |
        colorTab at: 4*i + j put: (255- i )
    ] ].
(0 to: 255) do: [ :i | colorTab at: (4*i + 4) put: 0].
```

```
hdc := UserLibrary getDC: nil.
```

```
GDIlibrary setSystemPaletteUse: hdc wUsage:SyspalNostatic.
GDIlibrary unrealizeObject: hdc .
```

```
logPalette := WinStructure new: (256 * 4 + 4).
logPalette uShortAtOffset: 0 put: 16r300;
uShortAtOffset: 2 put: (1 bitShift: 8).
logPalette contents
    replaceFrom: 5
```

```
                to: logPalette size
                with: colorTab.
```

```
bA:=GDIlibrary createPalette: logPalette asParameter.
```

```
hPrevious := UserLibrary selectPalette: hdc
                                                    with:
```

```
bA
                                                    forceBackground: true.
```

```
UserLibrary realizePalette: hdc.
```

```
UserLibrary selectPalette: hdc
                                                    with: hPrevious
                forceBackground: false.
```

```
GDIlibrary deleteObject: bA .
UserLibrary releaseDC: nil with: hdc.!
```

hmt

" Using the Hit or Missing transform to locate the objects defined by its foreground and background in an image."

```
| anOc1 anOc2 a |
fileName:= (FileDialog new openImage:'Open Forground Image')
file.
```

```
fileName isNil ifTrue:[^nil].
CursorManager execute change.
a:=Bitmap fromFile:fileName.
anOc1:=a getPoints:ClrWhite.
CursorManager normal change.
self display:a.
```

```
fileName:= (FileDialog new openImage:'Open Background Image')
file.
fileName isNil ifTrue:[^nil].
CursorManager execute change.
```

```

a:=Bitmap fromFile:fileName.
anOc2:=a getPoints:ClrWhite.
self display:a.
    image hmtBy:anOc1 and:anOc2.
    CursorManager normal change.
    self display:image!

```

```

image
    " Answer current working image."
    ^image!

```

imageCheck

```

    "answer nil if no image or reference image loaded."
    image isNil ifTrue: [MessageBox message:'Please load an
image or copy from screen!!'.
                                ^nil].
    seImage isNil ifTrue:[ MessageBox message:'Please select a
SE!!'.
                                ^nil].
    ^self!

```

imageMenu

```

    "Answer a menu with a list of image I/O operations."
    ^(MDIMenu
        labels:      '&Open...\S&ave\Save      se&lect\&Print\Print
Par&tial\&View' withCrs
        lines:      #(1 3 5)
        selectors:  #(loadImage save saveSelect print printSelected
viewImage))
        owner: self;
        title: '&Image';
        yourself!

```

imagePane

```

    " Open an MDI Child "
    ImageWindow new
        frame:self frame;
        icon:(Icon      fromModule:self      resourceDLLFile
id:'WORKSPACE');
        open.!

```

initWindowSize

```

    " Initial window size as whole screen size."
    ^Display extent!

```

intersec

```

    " Answer the intersection of two images."
    | imageR aPoint |
    self imageCheck isNil ifTrue:[^nil].
    image intersection:seImage.
    self display:image.!

```

label

```
" Labeling the connected components of image."

| blobSize |
blobSize:=self blobSize.
self imageCheck isNil ifTrue:[^nil].
CursorManager execute change.
image label:blobSize.
CursorManager normal change.
self display:image!
```

loadImage

```
"Load the image from the disk to the receiver."

| aPoint bitmap|

fileName:= (FileDialog new openImage:'Open Image') file.
fileName isNil ifTrue:[^nil].
image notNil ifTrue:[image release].
CursorManager execute change.
bitmap:=Bitmap fromFile:fileName.
CursorManager normal change.
image:=Image fromBitmap:bitmap.
self display:image.
bitmap release!
```

loadRefImage

```
"Load the image from the disk to the receiver as reference
image."

| aPoint points|

fileName:= (FileDialog new openImage:'Open Reference Image')
file.
fileName isNil ifTrue:[^nil].
CursorManager execute change.
seImage:=Bitmap fromFile:fileName.
anOc:=seImage getPoints:ClrWhite.
CursorManager normal change.
self display:seImage.
^anOc!
```

maxTable

```
" Answer a 256 gray maxium table in current palette."

| aPoint bM bA w h numBytes colTab|

CursorManager execute change.
w :=256.
h := 256.
numBytes := w*h.
bA := ByteArray new: numBytes.
(1 to: 256) do: [:i |
(1 to: 256) do: [:j|
bA at:((i- 1)*256 + j) put:((j- 1) max:(i -
1))]].

image isNil ifTrue:[
MessageBox message:'Please load an image!!'.
```

```

                                ^nil      ]
                                ifFalse:[
                                colTab:= image bitmapInfo colorTable].
                                bM:=Bitmap new initPenWidth: w
                                height: h
                                planes: 1
                                bitCount:      image
bitCount.
    bM bitmapInfo colorTable: colTab.
    bM createBitmap: bA .
    CursorManager normal change.
    self display:bM.
    bM release.!

```

```

mdiMenu: anMDIFrame
    " Create the menuBar for the frame"

    anMDIFrame mdiMenuWindow addMenu:self imageMenu owner:self.
    anMDIFrame mdiMenuWindow addMenu:self morphMenu owner:self.
    anMDIFrame mdiMenuWindow addMenu:self anaMenu owner:self.
    anMDIFrame mdiMenuWindow addMenu:self optionsMenu owner:self.
    anMDIFrame mdiMenuWindow addMenu:self paletteMenu owner:self.
    anMDIFrame mdiMenuWindow addMenu: (
        MDIMenu    colorMenu:    self    selector:    #colorSelected:)
owner:self.
    anMDIFrame mdiMenuWindow addMenu:self class mdiMenu owner:self.!

```

```

menu: anMDIFrame
    " Build the file menu"

    anMDIFrame menuWindow addMenu: self fileMenu owner:self.!

```

```

morphMenu
    "Answer a menu with a list of morphology operations."

    | aMenu |

    aMenu:= (MDIMenu
        labels:    '&Cross    3X3\&Square    3X3\&Disk    \From
F&ile...\&From &User' withCrS
        lines:    #(2)
        selectors:    #(cross    square    disk    loadRefImage
seFromUser))
        title:'&Select SE';
        owner:self.
    ^(MDIMenu
        labels:    '&View SE\&Dilation\&Erosion\&Opening\&Closing\Edge
&Trace\&Gradient' withCrS
        lines:    #(1 4 6)
        selectors:    #(viewSE    dilation    erosion    opening    closing
edgeTrace gradient))
        owner: self;
        title: '&Morph';
        appendSubMenu: (self otherMenu);
        appendSubMenu: aMenu;
        yourself!

```

```

open
    "Open a Image Processing window."

    self

```

```

        addView: (self frame:(
            MDIFrame new
                owner:self;
                labelWithoutPrefix: 'Image Lab';
                icon: (Icon fromModule: self resourceDLLFile
id:'face'));
                when: #mdiMenuBuilt perform: #mdiMenu;;
                when: #childClose perform: #childClose;;
                when: #toggleKey perform:#toggleKey;;
                when: #close perform: #close;;
                when: #validated perform: #startTimer;;
                when: #timer perform:#timer;;
                when: #childActivate perform:#childActivate;;
                when: #menuBuilt perform: #menu:
    )).
    self
        addSubpane:(
            ToolPane new
                owner: self;
                height: 27;
                when: #getContents perform:#toolPane:).
    self
        addSubpane:(
            StatusPane new
                owner:self;
                when:#getContents perform: #statusPane:).
    self openWindow.
    self menuWindow
        removeMenu: (self menuWindow menuTitled: '&Color')!

```

opening

```

    " Opening the loaded image with selected structure"

    self imageCheck isNil ifTrue:[^nil].
    CursorManager execute change.
    image openingBy:anOc.
    CursorManager normal change.
    self display:image.!

```

openTextWindow

" Open a TexWindow application as an MDI document"

```

|file|
fileName:= (FileDialog new openFile) file.
fileName isNil ifTrue:[^nil].
file := File pathName:fileName.
(self statusPane statusBoxAt: #status) contents: fileName .
ILTextWindow new
    frame:self frame;
    icon:(Icon fromModule:self resourceDLLFile
id:'TextWindow');
    openOnFile: file.
file close.!

```

optionsMenu

"Answer a menu with a list of miscellaneous operations."

```

^(MDIMenu
    labels: 'Clear\Copy\Paste\Fonts...\Pen size\BitEdit\Tool
Bar\Status Bar' withCrs
    lines: #(4)

```

```

selectors: #(clear copyGraph pasteGraph changeFont
changeSize bitEdit toggleToolPane toggleStatusPane))
owner: self;
title: '&Options';
checkItem: #toggleToolPane;
checkItem: #toggleStatusPane;
yourself!

```

orTable

" Answer a 256 gray or table in current palette."

```

| bM bA w h numBytes colTab|
CursorManager execute change.
w :=256.
h := 256.
numBytes := w*h.
bA := ByteArray new: numBytes.
    (1 to: 256) do: [:i |
        (1 to: 256) do: [:j |
            bA at:((i- 1)*256 + j) put:((j- 1) bitOr:(i
- 1))]].

image isNil ifTrue:[
    MessageBox message:'Please load an image!!'.
                    ^nil ]
    ifFalse:[
        colTab:= image bitmapInfo colorTable].
bM:=Bitmap new initPenWidth: w
                                height: h
                                planes: 1
                                bitCount:      image
bitCount.
bM bitmapInfo colorTable: colTab.
bM createBitmap: bA .
CursorManager normal change.
self display:bM.
bM release.!

```

otherMenu

"Answer a menu with a list of other morphology operations."

```

^ (MDIMenu

```

```

labels:'Complement\Reflect\Union...\Difference...\Intersection...
\Symetric Difference...\Thinning...\Thickening...' withCrs
    lines: #(2 5)
    selectors: #(complement reflectReal union differ
intersec symeDif thin thick))
    title:'&Others';
    owner:self.!

```

paletteMenu

"Answer a menu with a list of palette changing operations."

```

^ (MDIMenu
    labels: 'System\Gray Scale(256)\GSInvert\Spectrum\Or
Table\Max Table' withCrs
    lines: #(1 3)
    selectors: #(sysPal grayScalePal gsInvert spectrum orTable
maxTable))
    owner: self;

```

```

title: '&Palette';
yourself!

```

pane

```

" Answer current image pane."

^pane!

```

pasteGraph

```

" Paste the graphics from the clipboard to the receiver."

```

```

| bitmap |
(bitmap := Clipboard getBitmap) isNil
  ifTrue: [^self].
self display:bitmap.!

```

print

```

" Output image to print. Output loaded image or it operat
result."

```

```

image isNil ifTrue:[Prompter prompt:'No image loaded'
default:'Chose Print select'.
^nil].
image outputToPrinter!

```

printSelected

```

" Print selected area from screen."

```

```

| temp |
temp:= Bitmap fromUser.
temp isNil ifTrue:[^nil].
temp outputToPrinter .
temp release!

```

reflectReal

```

" Answer the reflection of original image."

```

```

image isNil ifTrue:[MessageBox message:'Open an image or
select from screen!!!!'.
^nil].
image reflectReal.
self display:image.!

```

resourceDLLFile

```

" Answer the DLL filename for resources, tool bar bitmaps
and icons."

```

```

^'imagelab.dll'!

```

save

```

" Output image to file. Output loaded image or it operat
result."

```

```

image isNil ifTrue:[Prompter prompt:'No image loaded'

```



```
default:'Chose Save select'.
        ^nil].
        fileName:= (FileDialog new saveImage:'Save Image'
        fileName:
fileName) file.
        fileName isNil ifTrue: [^nil].
        CursorManager normal change.
        image outputToFile:fileName.
        CursorManager normal change!
```

saveSelect

" Save select area from screen."

```
| image1 |
image1:= Bitmap fromUser.
image1 isNil ifTrue:[^nil].
fileName:= (FileDialog new saveTitle:'Save Image'
        fileName:
fileName) file.
        fileName isNil ifTrue: [^nil].
        image1 outputToFile:fileName.
        image1 release!
```

seFromUser

"Copy a portion of the receiver's contents as structure element."

```
| bitmap |
bitmap := Bitmap fromUser.
seImage:=bitmap.
CursorManager execute change.
anOc:= seImage getPoints:ClrWhite.
CursorManager normal change.!
```

spectrum

" Answer a 256 gray shade spectrum in current palette."

```
| bM bA w h numBytes colTab|
w :=256.
h := 32.
numBytes := w*h.
bA := ByteArray new: numBytes.
        (1 to: numBytes) do: [ :i | bA at: i put: (i- 1)\256].
image isNil ifTrue:[
        MessageBox message:'Please load an image!!'.
        ^nil ]
        ifFalse:[
        colTab:= image bitmapInfo colorTable].
bM:=Bitmap new initPenWidth: w
        height: h
        planes: 1
        bitCount:          image
bitCount.
        bM bitmapInfo colorTable: colTab.
        bM createBitmap: bA .

        self display:bM.
        bM release.!
```



```

        add: ( StatusBox new
            space: aStatusPane font width;
            width: (aStatusPane font stringWidth: 'OVR');
            contents: 'OVR';
            name: #ovr);
    yourself.

```

```

aStatusPane contents: statusBoxes.
(self statusPane statusBoxAt: #status) contents: 'Ready'.!

```

statusPaneHelp:aKey

" Answer the hint text used by the StatusPane for the help support."

```

^HelpImageLab at:aKey ifAbsent:[^super statusPaneHelp:aKey].!

```

symeDif

" Answer the Symetric Difference of two images."

```

| imageR aPoint |
self imageCheck isNil ifTrue:[^nil].
image symmetricDiff:selImage.
self display:image.!

```

sysPal

" Change system pallete into system palette."

```

| hdc |

hdc := UserLibrary getDC: nil.

GDIlibrary setSystemPaletteUse: hdc wUsage:SyspalStatic.
GDIlibrary unrealizeObject: hdc .

UserLibrary postMessage:16rFFFF
                    msg: WmSyscolorchange
                    wparam: 16rFFFF
                    lparam: 1.
UserLibrary releaseDC: nil with: hdc.!

```

thick

```

MessageBox message:'Sorry, not available in this version!!!!'.
^self!

```

thin

```

MessageBox message:'Sorry, not available in this version!!!!'.
^self!

```

timer:aTopPane

" Update the time in the StatusPane."

```

(aTopPane statusPane statusBoxAt: #time) contents:Time now.
(aTopPane statusPane statusBoxAt: #posi)
    contents:CursorManager cursorPosition.!

```

toggleKey:aPane

" Update the StatusPane to display the state of the toggle keys. The normal modes, such as Insert or non-Caps-lock mode, are indicated in the status bar by the absence of the indicator for the opposite mode. This is compliant with the Microsoft user Interface Style Guide for Windows 3.1."

```
(aPane statusPane statusBoxAt: #caps) show: (Notifier
isKeyToggled: VkCapital).
```

```
(aPane statusPane statusBoxAt: #num) show: (Notifier
isKeyToggled: VkNumlock).
```

```
(aPane statusPane statusBoxAt: #ovr) show: (Notifier
isKeyToggled: VkInsert).!
```

toggleMenu: menuName item: itemName

" Toggle the selected menu item."

```
| theMenu aBoolean|
theMenu := self frame menuWindow menuTitled: menuName.
(aBoolean:=theMenu isChecked:itemName)
  ifTrue: [self frame uncheckItem:itemName
forAllMDIChildMenus:menuName]
  ifFalse: [self frame checkItem:itemName
forAllMDIChildMenus:menuName].
```

^aBoolean!

toggleStatusPane

" Show/Hide the StatusPane."

```
self toggleMenu: '&Options' item:#toggleStatusPane.
self statusPane show.
self mdiArrange.!
```

toggleToolPane

" Show/Hide the ToolPane."

```
self toggleMenu: '&Options' item:#toggleToolPane.
self toolPane show.
self mdiArrange.!
```

toolPane:aPane

" Set the toolPane contents."

```
|aToolCollection aTool|
```

```
aToolCollection:=OrderedCollection new.
```

```
aTool:=Tool fromModule:self resourceDLLFile id:'create'.
aTool selector:#imagePane; owner:self; space:6.
aToolCollection add:aTool.
```

```
aTool:=Tool fromModule:self resourceDLLFile id:'filein'.
aTool selector:#loadImage; owner:self; space:6.
aToolCollection add:aTool.
```

```

aTool:=Tool fromModule:self resourceDLLFile id:'save'.
aTool selector:#save; owner:self.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'saveimage'.
aTool selector:#saveSelect; owner:self.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'cross_3X3'.
aTool selector:#cross;owner:self; space:6.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'squre_3X3'.
aTool selector:#square;owner:self.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'circle'.
aTool selector:#disk;owner:self.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'dilation'.
aTool selector:#dilation; owner:self; space:6.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'erosion'.
aTool selector:#erosion; owner:self.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'opening'.
aTool selector:#opening; owner:self.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'closing'.
aTool selector:#closing; owner:self.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'edge_trace'.
aTool selector:#edgeTrace; owner:self.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'copy'.
aTool selector:#copyGraph; owner:self; space:6.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'paste'.
aTool selector:#pasteGraph; owner:self.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'print'.
aTool selector:#print; owner:self.
aToolCollection add:aTool.

aTool:=Tool fromModule:self resourceDLLFile id:'inspectit'.
aTool selector:#bitEdit;owner:self;space:11.
aToolCollection add:aTool.

" aTool:=Tool fromModule:self resourceDLLFile id:'cut'.
aTool selector:#erase;owner:self.
aToolCollection add:aTool.      "

aPane contents: aToolCollection.

```

union

" Answer the union of two images"

```
| imageR aPoint |
self imageCheck isNil ifTrue:[^nil].
image union:seImage.
self display:image.!
```

viewImage

"View current image."

```
image isNil ifTrue:[MessageBox message:'No image to view!!'.
                    ^nil].
self display:image.!
```

viewSE

" View current structure element."

```
seImage isNil ifTrue:[MessageBox message:'No SE is
selected!!'.
                    ^nil].
self display:seImage.!!
```

F.3 Methods in Class ILTextWindow

```
TextWindow subclass: #ILTextWindow
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries:
'ColorConstants ' !
```

!ILTextWindow class methods ! !

!ILTextWindow methods !

openOn: aString

```
"Open the receiver with aString as its initial contents."
file := aString.
self addSubpane:
(ILTextPane new
owner: self;
backColor:ClrCyan;
when: #getContents perform: #stringContents;;
when: #save perform: #saveString;;
framingBlock: [ :box | box]).
self icon: (Icon fromModule: Icon defaultDLLFileName
id:'TextWindow').

self openWindow.
partial := false.
self menuWindow
removeMenu: (self menuWindow menuTitled: '&Image');
removeMenu: (self menuWindow menuTitled: '&Morph');
removeMenu: (self menuWindow menuTitled: '&Options');
removeMenu: (self menuWindow menuTitled: '&Palette');
removeMenu: (self menuWindow menuTitled: '&Color')!
```

openOnFile: aFile

"Open the receiver with aFile as its contents."

```

file := aFile.
partial := file size > 10000.
self addSubpane:
    (ILTextPane new
        owner: self;
        backColor:ClrCyan;
        when: #getContents perform: #readFile;;
        when: #save perform: #saveFile;;
        framingBlock: [ :box | box]).
self label: file pathName.
self icon: (Icon fromModule: Icon defaultDLLFileName
id:'TextWindow').
self openWindow.
partial ifTrue: [
    self menuWindow addMenu: (Menu new
        owner: self;
        title: '&Partial File';
        appendItem: '&Read Entire File' selector:
#readEntireFile). ] .
self menuWindow
    removeMenu: (self menuWindow menuTitled: '&Image');
    removeMenu: (self menuWindow menuTitled: '&Morph');
    removeMenu: (self menuWindow menuTitled: '&Options');
    removeMenu: (self menuWindow menuTitled: '&Palette');
    removeMenu: (self menuWindow menuTitled: '&Color')
"    (self menuWindow menuTitled: '&File')
        disableItem: #accept].    "!
```

readEntireFile

```

"Private - If a partial file was read, read it all now."
partial := false.
self changed: #readFile:.
self menuWindow removeMenu:
    (self menuWindow menuTitled: '&Partial File')! !
```

F.4 Methods in Class ImageWindow

```

GraphLab subclass: #ImageWindow
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries:
        'WinConstants' !
```

!ImageWindow class methods ! !

!ImageWindow methods !

drawingMode: aSymbol

"Set the state to aSymbol and have the menu reflect it."

```

(self frame menuWindow menus at: 2)
    uncheckItem: state.
(self frame menuWindow menus at: 2)
    checkItem: aSymbol.
```

```
state := aSymbol!
```

open

```
" Open an MDI Child "
```

```
|child|
self addView: (child :=
    MDIChild new
        when: #activate perform: #activate;;
        style: WsMaximizebox |
                WsMinimizebox |
                WsThickframe |
                WsSystemmenu |
                WsCaption |
                WsOverlapped |
                WsClipsiblings |
                WsClipchildren;
        frame:self frame;
        owner:self;
        icon:(Icon          fromModule:self          resourceDLLFile
id:'IMAGE_WORKSPACE');
        label:'Image Workspace';
        yourself).
```

```
child addSubpane:(
    pane := ImageProPane new
        owner: self;
        when: #getMenu perform: #modeMenu;;
        when: #getContents perform: #initPen;;
        when: #button1Down perform: #mouseDown;;
        yourself).
child openWindow!
```

pane

```
" Answer current working image pane."
```

```
^pane!
```

```
resourceDLLFile
```

```
"Private - Answer the DLL filename for resources, tool bar
bitmaps and icons."
```

```
^'imagelab.dll'! !
```

F.5 Methods in Class Matrix

```
Array variableSubclass: #Matrix
```

```
instanceVariableNames:
    'rows columns pivotValue '
classVariableNames: ''
poolDictionaries:
    'CharacterConstants ' !
```

```
!Matrix class methods !
```

```
new: row with: column
```

```
| answer |
answer:=super new: row * column.
```



```

    answer rows: row.
    answer columns: column.
    ^answer!

```

```

rows: row columns: column
    | answer |
    answer:=super new: row * column.
    answer rows: row.
    answer columns: column.
    ^answer! !

```

!Matrix methods !

*** aMat**

"Classical matrix multiplication."

```

    | result v1 v2 td |
    result := self class new: self rows with: self columns.
    1 to: self rows do:[:i|
        1 to: aMat columns do: [:j |
            td:=0.
            1 to: self columns do:[ :k |
                td:= td + ((self at: i at: k) *(aMat at: k at:j)). ].
            result at: i at: j put: td. ].].
    ^result!

```

at: i at: j

"Access matrix element in (i,j)."

```

    |value |
    (i <0 or: [i > self rows]) ifTrue: [ ^ nil ].
    (j <0 or: [j > self columns]) ifTrue: [ ^nil ].
    value:= self loc: i with: j.
    ^(self at: value)!

```

at: i at: j put: k

"Assignment for matrices."

```

    | value |
    value:= self loc:i with:j.
    self at: value put: k.
    ^ self!

```

columns

"Answer the number of columns in matrix. "

^columns!

columns: anlnt

"Force new number of columns."

columns:= anlnt.!

eNbr:j at:i

"Answer the east neighbour of element (i,j)"

```

    ((i+1) > (self columns)) ifTrue:[^0].
    ^(self at:j at:(i+1))!

```

eNbr:j at:i put:k

"Replace the east neighbour of element (i,j) with k"

```
((i+1) > (self columns)) ifTrue:[^0].
self at:j at:(i+1) put:k.
^self!
```

loc:i with:j

" Locate the (i,j) matrix element in Matrix "
 ^ ((i- 1) * (self columns) + j)!"

locate

"Locate the position of same value non-zero group in a matrix.(first from left-top, related to left-top corner) Answer an ordered collection Format:-(value1, point1, value2, point2, ... valueX, pointX, count)"

```
| n e count|
n:= OrderedCollection new.
count:=0.
1 to: (self rows) do:[:i|
  1 to: (self columns) do:[:j|
    e:=self at:i at:j .
    (e~=0) ifTrue:[
      (n includes:e) ifFalse:[
        count:=count+1.
        n add:e.
        n add:(j@i)]
    ]].
n add:count.
^n!
```

nNbr:j at:i

"Answer the north neighbour of element (i,j)"

```
(j- 1) = 0 ifTrue:[^0].
^(self at:(j- 1) at:i)!
```

nNbr:j at:i put:k

"Replace the north neighbour of element (i,j) with k"

```
(j- 1) = 0 ifTrue:[^0].
self at:(j- 1) at:i put:k.
^self!
```

nUp:j at:i

"Fill the north neighbours with element at (j,i) untill zero value is met."

```
| k m |
k:= self at:j at:i.
m:=j.
[(self nNbr:m at:i)~=0] whileTrue:[
  self nNbr:m at:i put:k.
  m:=m- 1.
].
^self!
```

```
rows
    "Answer the number of rows in matrix."
    ^rows!
```

```
rows: anlnt
    "Force new number of rows."
    rows:= anlnt.!
```

```
show
    |sum|
    sum := 0.
    1 to: self rows do: [ :i |
        1 to: self columns do: [ :j |
            Transcript show:
                (self at: i at: j) printString, ' '.
            Transcript show: Lf printString.
        ].
    ]!
```

```
sNbr:j at:i
    "Answer the south neighbour of element (i,j)"
    ((j+1) >(self rows)) ifTrue:[^0].
    ^(self at:(j+1) at:i)!
```

```
sNbr:j at:i put:k
    "Replace the south neighbour of element (i,j) with k"
    ((j+1) >(self rows)) ifTrue:[^0].
    self at:(j+1) at:i put:k.
    ^self!
```

```
wBack:j at:i
    "Fill the west neighbours with element at (j,i) untill zero value
    is met."
    | k m |
    k:= self at:j at:i.
    m:=i.
    [(self wNbr:j at:m)~=0] whileTrue:[
        self wNbr:j at:m put:k.
        m:=m- 1.
    ].
    ^self!
```

```
westNorth:j at:i
    "Fill the west neighbours and north neighbour's value untill zero
    value is met."
    | k m |
    k:= self nNbr:j at:i.
    m:=i.
    [(self wNbr:j at:m)~=0] whileTrue:[
        self wNbr:j at:m put:k.
        self nUp:j at:m.
    ]
```

```

        m:=m- 1.                ].
    ^self!

```

```

wNbr:j at:i
    "Answer the west neighbour of element (i,j)"

    (i- 1) = 0 ifTrue:[^0].
    ^(self at:j at:(i- 1))!

```

```

wNbr:j at:i put:k
    "Replace the west neighbour of element (i,j) with k"

    (i- 1) = 0 ifTrue:[^0].
    self at:j at:(i- 1) put:k.
    ^self! !

```

```

AboutDialog subclass: #AboutImageLabDialog
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: '' !

```

!AboutImageLabDialog class methods ! !

!AboutImageLabDialog methods !

```

open
    "Open a dialog box telling about ImageLab."

    | lineHeight |
    self
        labelWithoutPrefix: 'About Image Lab'.

    lineHeight := 8.
    self addSubpane:
        (StaticText new
            centered;
            contents: 'Image Lab Release 1.00';
            framingBlock: [:box |
                (box leftTop down: lineHeight)
                extentFromLeftTop: box width @ lineHeight ] ).

    self addSubpane:
        (StaticText new centered;
            contents: 'By Fei Liu. May 1993';
            framingBlock: [:box | (box leftTop down: lineHeight *
3)
                extentFromLeftTop: box width @ lineHeight ] ).

    self addSubpane:
        (StaticText new centered;
            contents: 'All rights reserved';
            framingBlock: [:box | (box leftTop down: lineHeight * 4)
                extentFromLeftTop: box width @ lineHeight ] ).

    self addSubpane:
        (StaticText new centered;
            contents: 'PO Box 14428 MMC';
            framingBlock: [:box | (box leftTop down: lineHeight * 6)
                extentFromLeftTop: box width @ lineHeight ] ).

```

```

self addSubpane:
  (StaticText new centered;
   contents: 'Melbourne Victoria 3000';
   framingBlock: [:box | (box leftTop down: lineHeight * 7)
    extentFromLeftTop: box width @ lineHeight ] ).

self addSubpane:
  (StaticText new centered;
   contents: '(61) (03) 688-4854';
   framingBlock: [:box | (box leftTop down: lineHeight * 8)
    extentFromLeftTop: box width @ lineHeight ] ).

self addSubpane:
  (StaticText new centered;
   contents: 'Image Lab is a research product';
   framingBlock: [:box | (box leftTop down: lineHeight *
12)
    extentFromLeftTop: box width @ lineHeight ] ).

self addSubpane:
  (StaticText new centered;
   contents: 'of Fei Liu at VUT';
   framingBlock: [:box | (box leftTop down: lineHeight *
13)
    extentFromLeftTop: box width @ lineHeight ] ).

self addSubpane:
  (Button new defaultPushButton;
   idOK;
   contents: 'OK';
   when: #clicked perform: #ok;;
   framingBlock: [:box | (box leftTop rightAndDown:
    (box width - 35) // 2 @ (lineHeight * 19 //
2))
    extentFromLeftTop: 35 @ (lineHeight * 2)] ).

self openWindow! !

```