ComponentWorks™

Getting Results with ComponentWorks™ IMAQ Vision

NATIONAL INSTRUMENTS™

June 1998 Edition
Part Number 321883A-01

Internet Support
E-mail: `support@natinst.com`
FTP Site: `ftp.natinst.com`
Web Address: `http://www.natinst.com`

Bulletin Board Support
BBS United States: 512 794 5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support
512 418 1111

Telephone Support (USA)
Tel: 512 795 8248
Fax: 512 794 5678

International Offices
Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 288 3336,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 6120092, Italy 02 413091,
Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
United Kingdom 01635 523545

National Instruments Corporate Headquarters
6504 Bridge Point Parkway Austin, Texas 78730-5039 USA Tel: 512 794 0100

© Copyright 1998 National Instruments Corporation. All rights reserved.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

ComponentWorks™, IMAQ™, and NI-IMAQ™ are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

About This Manual

Organization of This Manual xix
Part I, Building ComponentWorks IMAQ Vision Applications xix
Part II, Using the ComponentWorks IMAQ Vision Controls xx
Part III, Introduction to Vision xxi
Appendices, Glossary, and Index xxii
Conventions Used in This Manual xxii
Related Documentation xxiii
Customer Communication xxiii

Chapter 1
Introduction to ComponentWorks IMAQ Vision

What Is ComponentWorks IMAQ Vision? 1-1
System Requirements 1-2
Installing ComponentWorks IMAQ Vision 1-2
Installing From Floppy Disks 1-3
Installed Files 1-3
About the ComponentWorks Controls 1-4
Properties, Methods, and Events 1-4
Object Hierarchy 1-5
Collection Objects 1-6
Setting the Properties of an ActiveX Control 1-7
Using Property Pages 1-7
Changing Properties Programmatically 1-9
Item Method 1-10
Working with Control Methods 1-10
Developing Event Handler Routines 1-11
Learning the Properties, Methods, and Events 1-11

Chapter 2
Getting Started with ComponentWorks

Install and Configure Driver Software 2-1
Explore the ComponentWorks IMAQ Vision Documentation 2-2
Getting Results with ComponentWorks IMAQ Vision Manual 2-2
ComponentWorks Online Reference 2-3
Accessing the Online Reference 2-3
Finding Specific Information 2-3
Become Familiar with the Examples Structure 2-4

Develop Your Application 2-4
Seek Information from Additional Sources 2-6

Part I
Building ComponentWorks IMAQ Vision Applications

Chapter 3
Building ComponentWorks IMAQ Vision Applications with Visual Basic

Developing Visual Basic Applications 3-1
Loading ComponentWorks IMAQ Vision Controls into the Toolbox 3-2
Building the User Interface Using ComponentWorks 3-2
Using Property Pages 3-3
Using Your Program to Edit Properties 3-4
Working with Control Methods 3-5
Developing Control Event Routines 3-5
Using the Object Browser to Build Code in Visual Basic 3-6
Pasting Code into Your Program 3-8
Adding Code Using Visual Basic Code Completion 3-9
Creating Standalone Objects 3-10

Chapter 4
Building ComponentWorks IMAQ Vision Applications with Visual C++

Developing Visual C++ Applications 4-1
Creating Your Application 4-2
Adding ComponentWorks Controls to the Visual C++ Controls Toolbar 4-3
Building the User Interface Using ComponentWorks 4-4
Programming with the ComponentWorks Controls 4-5
Using Properties 4-6
Using Methods 4-8
Using Events 4-9
Creating Standalone Objects 4-10

Chapter 5
Building ComponentWorks IMAQ Vision Applications with Delphi

Running Delphi Examples 5-1
Developing Delphi Applications 5-1
Loading ComponentWorks Controls into the Component Palette 5-2
Building the User Interface 5-4
Placing Controls 5-4
Using Property Pages 5-4

Programming with ComponentWorks 5-5
Using Your Program to Edit Properties 5-6
Using Methods 5-6
Using Events 5-7
Creating Standalone Objects 5-8

Part II
Using the ComponentWorks IMAQ Vision Controls

Chapter 6
Using the Viewer and Hardware Controls

Image Acquisition Configuration 6-1
What Are the Viewer and Hardware Controls? 6-1
Object Hierarchy 6-2
Viewer Control—IMAQ User Interface Control 6-2
Viewer Object 6-3
Regions Collection 6-4
Region Object 6-4
Palette Object 6-4
Viewer Events 6-4
IMAQ Control—IMAQ Hardware Interface 6-5
IMAQ Object 6-6
Image Object 6-6
IMAQ Methods and Events 6-8
Asynchronous Acquisition 6-8
Synchronous Acquisition 6-8
Error Handling 6-9
ExceptionOnError and ErrorEventMask 6-9
Tutorial: Using the Viewer and IMAQ Controls 6-10
Part 1: Synchronous, Single-Image Acquisition and Display 6-10
Designing the Form 6-10
Setting the IMAQ Properties 6-11
Developing the Code 6-11
Testing Your Program 6-12
Part 2: Asynchronous, Continuous Single-Image Acquisition and Display 6-12
Designing the Form 6-13
Setting the IMAQ Properties 6-13
Developing the Code 6-13
Testing Your Program 6-13

Chapter 7
Using the Vision Control

What is the Vision Control? 7-1
Vision Functions 7-2
Tutorial: Using Simple Image Processing Functions 7-7
Part 1: Reading an Image From a File and Thresholding 7-7
Designing the Form 7-7
Developing the Code 7-8
Testing Your Program 7-9
Part 2: Particle Analysis 7-10
Designing the Form 7-10
Developing the Code 7-10
Testing Your Program 7-11
Part 3: Acquisition and Image Processing 7-11
Designing the Form 7-12
Setting the IMAQ Properties 7-13
Developing the Code 7-13
Testing Your Program 7-14

Chapter 8
Building Advanced IMAQ Vision Applications

Finding Features on a Printed Circuit Board 8-1
Manipulating Regions of Interest through the User Interface 8-2
AutoDelete, Active, and Visible Properties 8-3
Finding Features and Displaying Results 8-4
Floppy Disk Inspection 8-5
Manipulating Regions of Interest Programmatically 8-7
Edge Detection and Shape Matching 8-7
Report Objects 8-8
Adding Testing and Debugging to your Application 8-9
Error Checking 8-9
Exceptions 8-10
Return Codes 8-11
Error and Warning Events 8-12
Debugging 8-13
Debug Print 8-13
Breakpoint 8-13
Watch Window 8-14
Single Step, Step Into, and Step Over 8-14

Part III
Introduction to Vision

Chapter 9
Algorithms, Principles of Image Files, and Data Structures

Introduction to Digital Images 9-1
Properties of a Digitized Image 9-1
Image Resolution 9-2
Image Definition 9-2
Number of Planes 9-2
Image Types and Formats 9-3
Gray-Level Images 9-3
Color Images 9-3
Complex Images 9-3
Image Files 9-5
Processing Color Images 9-5
Image Pixel Frame 9-6
Rectangular Frame 9-7
Hexagonal Frame 9-8

Chapter 10
Tools and Utilities

Palettes 10-1
B&W (Gray) Palette 10-2
Temperature Palette 10-3
Rainbow Palette 10-3
Gradient Palette 10-3
Binary Palette 10-4
Image Histogram 10-4
Definition 10-4
Linear Histogram 10-5
Cumulative Histogram 10-6
Interpretation 10-6
Histogram of Color Images 10-6
Histogram Scale 10-7
Line Profile 10-8
3D View 10-8

Chapter 11
Lookup Transformations

About Lookup Table Transformations 11-1
Example 11-2
Predefined Lookup Tables 11-3
Equalize 11-4
Example 1 11-4
Example 2 11-5
Reverse 11-5
Example 11-6
Logarithmic and Inverse Gamma Correction 11-6
Exponential and Gamma Correction 11-8

Chapter 12
Operators

Concepts and Mathematics 12-1
Arithmetic Operators 12-2
Logic Operators 12-2
Truth Tables 12-4
Example 1 12-5
Example 2 12-6

Chapter 13
Spatial Filtering

Concept and Mathematics 13-1
Spatial Filter Classification Summary 13-3
Linear Filters or Convolution Filters 13-3
Gradient Filter 13-4
Example 13-4
Kernel Definition 13-5
Filter Axis and Direction 13-6
Edge Extraction and Edge Highlighting 13-7
Edge Thickness 13-9
Predefined Gradient Kernels 13-10
Laplacian Filters 13-12
Example 13-12
Kernel Definition 13-13
Contour Extraction and Highlighting 13-14
Contour Thickness 13-15
Predefined Laplacian Kernels 13-16

Smoothing Filter 13-17
Example 13-17
Kernel Definition 13-18
Predefined Smoothing Kernels 13-19
Gaussian Filters 13-20
Example 13-20
Kernel Definition 13-21
Predefined Gaussian Kernels 13-21
Nonlinear Filters 13-22
Nonlinear Prewitt Filter 13-22
Nonlinear Sobel Filter 13-23
Example 13-24
Nonlinear Gradient Filter 13-25
Roberts Filter 13-25
Differentiation Filter 13-25
Sigma Filter 13-26
Lowpass Filter 13-26
Median Filter 13-27
Nth Order Filter 13-27
Examples 13-28

Chapter 14
Frequency Filtering

Introduction to Frequency Filters 14-1
Lowpass FFT Filters 14-2
Highpass FFT Filters 14-2
Mask FFT Filters 14-2
Definition 14-3
FFT Display 14-4
Standard Representation 14-5
Optical Representation 14-6
Frequency Filters 14-7
Lowpass Frequency Filters 14-7
Lowpass Attenuation 14-7
Lowpass Truncation 14-8
Highpass Frequency Filters 14-9
Highpass Attenuation 14-10
Highpass Truncation 14-10

Chapter 15
Morphology Analysis

Thresholding 15-1
Example 15-2
Thresholding a Color Image 15-3
Automatic Threshold 15-3
Clustering 15-3
Entropy 15-5
Metric 15-5
Moments 15-5
Interclass Variance 15-6
Structuring Element 15-6
Primary Binary Morphology Functions 15-7
Erosion Function 15-8
Concept and Mathematics 15-8
Dilation Function 15-8
Concept and Mathematics 15-8
Erosion and Dilation Examples 15-9
Opening Function 15-10
Closing Function 15-11
Opening and Closing Examples 15-11
External Edge Function 15-12
Internal Edge Function 15-12
External and Internal Edge Example 15-12
Hit-Miss Function 15-13
Concept and Mathematics 15-13
Example 1 15-13
Example 2 15-14
Thinning Function 15-15
Examples 15-16
Thickening Function 15-17
Examples 15-18
Proper-Opening Function 15-19
Proper-Closing Function 15-20
Auto-Median Function 15-20
Advanced Binary Morphology Functions 15-21
Border Function 15-21
Hole Filling Function 15-21
Labeling Function 15-21
Lowpass Filters 15-22
Highpass Filters 15-22
Lowpass and Highpass Example 15-23

Separation Function 15-24
Skeleton Functions 15-24
L-Skeleton Function 15-25
M-Skeleton Function 15-25
Skiz Function 15-26
Segmentation Function 15-26
Comparisons Between Segmentation and Skiz Functions 15-27
Distance Function 15-27
Danielsson Function 15-28
Example 15-28
Circle Function 15-29
Example 15-29
Convex Function 15-30
Example 15-30
Gray-Level Morphology 15-31
Erosion Function 15-31
Concept and Mathematics 15-31
Dilation Function 15-31
Concept and Mathematics 15-31
Erosion and Dilation Examples 15-32
Opening Function 15-33
Closing Function 15-33
Opening and Closing Examples 15-33
Proper-Opening Function 15-34
Proper-Closing Function 15-35
Auto-Median Function 15-36

Chapter 16
Quantitative Analysis

Spatial Calibration 16-1
Intensity Calibration 16-2
Definition of a Digital Object 16-2
Intensity Threshold 16-2
Connectivity 16-3
Connectivity-8 16-3
Connectivity-4 16-4
Area Threshold 16-4
Object Measurements 16-5
Areas 16-5
Particle Number 16-5
Number of Pixels 16-5
Particle Area 16-5

Scanned Area 16-6
Ratio 16-6
Number of Holes 16-6
Holes' Area 16-6
Total Area 16-6
Lengths 16-7
Particle Perimeter 16-7
Holes' Perimeter 16-7
Breadth 16-7
Height 16-8
Coordinates 16-8
Center of Mass X and Center of Mass Y 16-8
Min(X, Y) and Max(X, Y) 16-9
Max Chord X and Max Chord Y 16-9
Chords and Axes 16-9
Max Chord Length 16-10
Mean Chord X 16-10
Mean Chord Y 16-10
Max Intercept 16-10
Mean Intercept Perpendicular 16-10
Particle Orientation 16-10
Shape Equivalence 16-11
Equivalent Ellipse Minor Axis 16-12
Ellipse Major Axis 16-12
Ellipse Minor Axis 16-13
Ellipse Ratio 16-13
Rectangle Big Side 16-13
Rectangle Small Side 16-13
Rectangle Ratio 16-14
Shape Features 16-14
Moments of Inertia Ixx, Iyy, Ixy 16-14
Elongation Factor 16-14
Compactness Factor 16-15
Heywood Circularity Factor 16-15
Hydraulic Radius 16-15
Waddel Disk Diameter 16-16
Densitometry 16-18
Diverse Measurements 16-18

Appendices, Glossary, and Index

Appendix A
Common Questions

Appendix B
Error Codes

Appendix C
Distribution and Redistributable Files

Appendix D
Customer Communication

Glossary

Index

Figures and Tables

Figures

Figure 1-1. IMAQ Control Object Hierarchy 1-6
Figure 1-2. Visual Basic Default Property Sheets 1-8
Figure 1-3. ComponentWorks Custom Property Pages 1-8

Figure 3-1. Visual Basic Property Pages 3-3
Figure 3-2. ComponentWorks Custom Property Pages 3-4
Figure 3-3. Selecting Events in the Code Window 3-6
Figure 3-4. Viewing CWIMAQ in the Object Browser 3-7
Figure 3-5. Viewing CWIMAQ in the Object Browser 3-8
Figure 3-6. Visual Basic 5 Code Completion 3-9

Figure 4-1. New Dialog Box 4-2
Figure 4-2. MFC AppWizard—Step 1 4-3
Figure 4-3. CWIMAQ Control Property Sheets 4-5
Figure 4-4. MFC ClassWizard—Member Variable Tab 4-6
Figure 4-5. Viewing Property Functions and Methods in the Workspace Window 4-7
Figure 4-6. Event Handler 4-10

Figure 5-1. Delphi Import ActiveX Control Dialog Box 5-2
Figure 5-2. ComponentWorks Controls on a Delphi Form 5-4
Figure 5-3. Delphi Object Inspector 5-5
Figure 5-4. ComponentWorks IMAQ Control Property Pages 5-5
Figure 5-5. Delphi Object Inspector Events Tab 5-8

Figure 6-1. Viewer Control Object Hierarchy 6-3
Figure 6-2. IMAQ Control Object Hierarchy 6-6
Figure 6-3. Viewer Control and IMAQ Control Can Share an Image Object 6-7
Figure 6-4. Simple IMAQ Example Form 6-11
Figure 6-5. Testing the Simple IMAQ Example 6-12

Figure 7-1. IMAQ File Example 7-8
Figure 7-2. Testing the IMAQ File Example 7-9
Figure 7-3. Testing the IMAQ File Example After Adding Particle Analysis 7-11
Figure 7-4. Image Acquisition Threshold Example 7-12
Figure 7-5. Testing the Image Acquisition Threshold Example 7-14

Figure 8-1. Feature Find Application 8-2
Figure 8-2. Floppy Disk Inspection 8-6
Figure 8-3. Visual Basic Error Message 8-10
Figure 8-4. Error Message Box 8-12

Figure 9-1. Rectangular Frame 9-7
Figure 9-2. Hexagonal Frame 9-8

Figure 10-1. Linear Vertical Scale 10-5
Figure 10-2. Linear Cumulative Scale 10-6
Figure 10-3. Linear Vertical Scale 10-7
Figure 10-4. Logarithmic Vertical Scale 10-7

Figure 15-1. Rectangular Frame, Neighborhood 3 × 3 15-7
Figure 15-2. Hexagonal Frame, Neighborhood 5 × 3 15-7

Tables

Table 2-1. Chapters about Specific Programming Environments 2-5

Table 7-1. IMAQ Vision Functions 7-2

Table 9-1. Bytes Per Pixel 9-4

Table 13-1. Prewitt Filters 13-10
Table 13-2. Sobel Filters 13-11

Table 13-3. Gradient 5 × 5 13-12
Table 13-4. Gradient 7 × 7 13-12
Table 13-5. Laplacian 3 × 3 13-16
Table 13-6. Laplacian 5 × 5 13-16
Table 13-7. Laplacian 7 × 7 13-17
Table 13-8. Smoothing 3 × 3 13-19
Table 13-9. Smoothing 5 × 5 13-19
Table 13-10. Smoothing 7 × 7 13-20
Table 13-11. Gaussian 3 × 3 13-21
Table 13-12. Gaussian 5 × 5 13-21
Table 13-13. Gaussian 7 × 7 13-22

Table B-1. ComponentWorks Errors B-1
Table B-2. ComponentWorks IMAQ Errors B-1
Table B-3. IMAQ Errors B-3
Table B-4. Vision Errors B-5

About This Manual

The *Getting Results with ComponentWorks IMAQ Vision* manual contains the information you need to get started with the ComponentWorks IMAQ controls. ComponentWorks adds the instrumentation-specific tools for acquiring, analyzing, and displaying data in Visual Basic, Visual C++, Delphi, and other ActiveX control environments.

This manual contains step-by-step instructions for building applications with ComponentWorks IMAQ Vision. You can modify these sample applications to suit your needs. This manual does not show you how to use every control or solve every possible programming problem. Use the online reference for further, function-specific information.

To use this manual, you already should be familiar with one of the supported programming environments and Windows 95/98 or Windows NT.

Organization of This Manual

The *Getting Results with ComponentWorks IMAQ Vision* manual is organized as follows.

- Chapter 1, *Introduction to ComponentWorks IMAQ Vision*, contains an overview of ComponentWorks IMAQ Vision, lists the ComponentWorks IMAQ Vision system requirements, describes how to install the software, and presents basic information about ComponentWorks ActiveX controls.
- Chapter 2, *Getting Started with ComponentWorks*, describes approaches to help you get started using ComponentWorks IMAQ Vision, depending on your application needs, your experience using ActiveX controls in your particular programming environment, and your specific goals in using ComponentWorks.

Part I, Building ComponentWorks IMAQ Vision Applications

This section describes how to use ActiveX controls in the most commonly used programming environments—Visual Basic, Visual C++, and Borland Delphi.

Part I, *Building ComponentWorks IMAQ Vision Applications*, contains the following chapters.

- Chapter 3, *Building ComponentWorks IMAQ Vision Applications with Visual Basic*, describes how you can use the ComponentWorks controls with Visual Basic 5; insert the controls into the Visual Basic environment, set their properties, and use their methods and events; and perform these operations using ActiveX controls in general. This chapter also outlines Visual Basic features that simplify working with ActiveX controls.
- Chapter 4, *Building ComponentWorks IMAQ Vision Applications with Visual C++*, describes how you can use ComponentWorks controls with Visual C++, explains how to insert the controls into the Visual C++ environment and create the necessary wrapper classes, shows you how to create an application compatible with the ComponentWorks controls using the Microsoft Foundation Classes Application Wizard (MFC AppWizard) and how to build your program using the ClassWizard with the controls, and discusses how to perform these operations using ActiveX controls in general.
- Chapter 5, *Building ComponentWorks IMAQ Vision Applications with Delphi*, describes how you can use ComponentWorks controls with Delphi; insert the controls into the Delphi environment, set their properties, and use their methods and events; and perform these operations using ActiveX controls. This chapter also outlines Delphi features that simplify working with ActiveX controls.

Part II, Using the ComponentWorks IMAQ Vision Controls

This section describes the basic operation of the ComponentWorks IMAQ Vision controls. These chapters contain overviews of the controls, describing their most commonly used properties, methods, and events. The descriptions also include short code segments to illustrate programmatic control and tutorials that lead you through building an application with the controls.

Part II, *Using the ComponentWorks IMAQ Vision Controls*, contains the following chapters.

- Chapter 6, *Using the Viewer and Hardware Controls*, describes how you can use the IMAQ Viewer and Hardware controls to acquire and display images; explains individual controls and their most commonly used properties, methods, and events; and includes a tutorial with step-by-step instructions for using the controls.

- Chapter 7, *Using the Vision Control*, describes how you can use the ComponentWorks IMAQ Vision control and its functions. You can use the Vision control alone or with other controls to perform image analysis, manipulation, and processing. Vision functions include operations such as filtering, morphology, arithmetic, and measurement.
- Chapter 8, *Building Advanced IMAQ Vision Applications*, discusses how you can build applications using more advanced features of ComponentWorks IMAQ Vision, including advanced image acquisition techniques, image processing, and advanced user interface options. It also explains error tracking, error checking, and debugging techniques.

Part III, Introduction to Vision

This section presents the basics of computer-based vision applications.

Part III, *Introduction to Vision*, contains the following chapters.

- Chapter 9, *Algorithms, Principles of Image Files, and Data Structures*, describes the algorithms and principles of image files and data structures.
- Chapter 10, *Tools and Utilities*, describes the tools and utilities used in IMAQ Vision.
- Chapter 11, *Lookup Transformations*, provides an overview of lookup table transformations.
- Chapter 12, *Operators*, describes the arithmetic and logic operators used in IMAQ Vision.
- Chapter 13, *Spatial Filtering*, provides an overview of the spatial filters, including linear and nonlinear filters, used in IMAQ Vision.
- Chapter 14, *Frequency Filtering*, describes the frequency filters used in IMAQ Vision.
- Chapter 15, *Morphology Analysis*, provides an overview of morphology image analysis.
- Chapter 16, *Quantitative Analysis*, provides an overview of quantitative image analysis. The *quantitative analysis* of an image consists of obtaining *densitometry* and object measurements. Before starting this analysis, it is necessary to calibrate the image spatial dimensions and intensity scale to obtain measurements expressed in real units.

Appendices, Glossary, and Index

- Appendix A, *Common Questions*, contains answers to frequently asked questions.
- Appendix B, *Error Codes*, lists the error codes returned by ComponentWorks, the ComponentWorks IMAQ controls, IMAQ hardware, and Vision.
- Appendix C, *Distribution and Redistributable Files*, contains information about ComponentWorks IMAQ Vision redistributable files and distributing applications that use ComponentWorks controls.
- Appendix D, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including acronyms, abbreviations, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

<>

Angle brackets enclose the name of a key on the keyboard—for example, <Shift>.

-

A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys—for example, <Control-Alt-Delete>.

»

The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options»Substitute Fonts** directs you to pull down the **File** menu, select the **Page Setup** item, select **Options**, and finally select the **Substitute Fonts** options from the last dialog box.

☞

This icon to the left of bold italicized text denotes a note, which alerts you to important information.s

bold

Bold text denotes the names of menus, menu items, parameters, and dialog box options.

bold italic

Bold italic text denotes a note.

<Control>	Key names are capitalized.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept.
`monospace`	Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subroutines, device names, functions, operations, properties and methods, filenames and extensions, and for statements and comments taken from programs.
paths	Paths in this manual are denoted using backslashes (\) to separate drive names, directories, folders, and files.

Related Documentation

The following documents contain information you might find useful as you read this manual:

- ComponentWorks IMAQ Vision online reference (available by selecting **Programs»National Instruments ComponentWorks» IMAQ Vision»ComponentWorks IMAQ Reference** from the Windows **Start** menu)
- *Getting Results with ComponentWorks* (If you have one of the ComponentWorks development systems.)
- ComponentWorks online reference (If you have ComponentWorks installed on your computer, you can access the online reference by selecting **Start»Programs»National Instruments ComponentWorks»ComponentWorks Reference**)

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix D, *Customer Communication*, at the end of this manual.

1

Introduction to ComponentWorks IMAQ Vision

This chapter contains an overview of ComponentWorks IMAQ Vision, lists the ComponentWorks IMAQ Vision system requirements, describes how to install the software, and presents basic information about ComponentWorks ActiveX controls.

What Is ComponentWorks IMAQ Vision?

ComponentWorks IMAQ Vision is a collection of ActiveX controls for developing machine-vision applications within any compatible ActiveX control container. ActiveX controls also are known as OLE (Object Linking and Embedding) controls, and the two terms can be used interchangeably in this context. Use the online reference for specific information about the properties, methods, and events of the individual ActiveX controls. You can access this information by selecting **Programs»National Instruments ComponentWorks»IMAQ Vision»ComponentWorks IMAQ Reference** from the Windows **Start** menu.

With ComponentWorks IMAQ Vision, you can acquire images from image acquisition cards supported by the NI-IMAQ driver, display them in your application, perform interactive viewer operations, and analyze your images to extract information. The ComponentWorks IMAQ Vision package contains the following components:

- IMAQ Hardware Control—ActiveX control for acquiring images from devices supported by the NI-IMAQ driver.
- Viewer Control—ActiveX control for displaying images in your application. With this control, you can interactively select a region of interest, zoom and pan an image, and apply different color palettes.
- Vision Control—ActiveX control for analyzing and processing images. Functions include caliper tools, pattern matching, histogram, blobs analysis, and more.

Note ***If you received the IMAQ Hardware control as part of NI-IMAQ and have not purchased ComponentWorks IMAQ Vision, the Viewer and Vision controls are in evaluation mode.***

The ComponentWorks IMAQ Vision ActiveX controls are designed for use in Visual Basic, a premier ActiveX control container application. Some ComponentWorks features and utilities have been incorporated with the Visual Basic user in mind. However, you can use ActiveX controls in other applications that support them, including Visual C++ and Delphi.

System Requirements

To use the ComponentWorks IMAQ Vision ActiveX controls, your computer must meet the following minimum requirements:

- Personal computer using at least a 33 MHz 80486 or higher microprocessor (National Instruments recommends a 90 MHz Pentium or higher microprocessor)
- Microsoft Windows 95/98 or Windows NT version 4.0
- VGA resolution (or higher) video adapter
- 32-bit ActiveX control container such as Visual Basic 4.0 or greater, Visual C++ 4.*x* or greater, or Delphi
- NI-IMAQ 1.5 or greater for Windows 95/98 or Windows NT (if you are using the IMAQ Hardware control)
- Minimum of 16 MB of memory
- Minimum of 10 MB of free hard disk space
- Microsoft-compatible mouse

Installing ComponentWorks IMAQ Vision

The ComponentWorks IMAQ Vision setup program installs the controls through a process that lasts approximately five minutes. If you do not have a compatible version of the NI-IMAQ driver software, you can install it from the ComponentWorks IMAQ Vision CD.

Note ***To install ComponentWorks IMAQ Vision on a Windows NT system, you must be logged in with Administrator privileges.***

1. Make sure that your computer and monitor are turned on and that you have installed Windows 95/98 or Windows NT.

2. Insert the ComponentWorks IMAQ Vision CD in the CD drive of your computer. From the CD startup screen, click on **Install ComponentWorks IMAQ Vision**. If the CD startup screen does not appear, use the Windows Explorer to run the `SETUP.EXE` program in the `\Setup` directory on the CD.
3. Follow the instructions on the screen. The installer provides different options for setting the directory in which ComponentWorks IMAQ Vision is installed and choosing examples for different programming environments. Use the default settings if you are unsure about which settings to choose. If necessary, you can run the installer at a later time to install additional components.

Installing From Floppy Disks

If your computer does not have a CD drive, you can copy the files to floppy disks and install the controls from those disks, as described by the following steps.

1. On another computer with a CD drive and disk drive, copy the files in the individual subdirectories of the `\Setup\disks` directory on the CD onto individual 3.5-inch floppy disks. The floppy disks should not contain any directories and should be labeled `disk1`, `disk2`, and so on, following the name of the source directories.
2. On the target computer, insert the floppy labeled `disk1` and run the `setup.exe` program from the floppy.
3. Follow the on-screen instructions to complete the installation.

Installed Files

The ComponentWorks IMAQ Vision setup program installs the following groups of files on your hard disk.

- ActiveX controls, documentation, and other associated files

  Directory: `\Windows\System\`

  Files: `cwimaq.ocx`, `cwimaq.dep`, `cwimaq.hlp`, `cwimaq.cnt`
- Example programs and applications

  Directory: `\ComponentWorks\Samples\...`
- Tutorials

  Directory: `\ComponentWorks\Tutorials-IMAQ\...`
- Images

  Directory: `\ComponentWorks\Images`

- Miscellaneous files

  Directory: `\ComponentWorks\`

Note ***You select the location of the*** `\ComponentWorks\...` ***directory during installation.***

About the ComponentWorks Controls

This section presents background information about the ComponentWorks ActiveX controls. Make sure you understand these concepts before continuing. You also should refer to your programming environment documentation for more information about using ActiveX controls in that environment.

Properties, Methods, and Events

ActiveX controls consist of three different parts—properties, methods, and events—used to implement and program the controls.

Properties are the attributes of a control. These attributes describe the current state of the control and affect the display and behavior of the control. The values of the properties are stored in variables that are part of the control.

Methods are functions defined as part of the control. Methods are called with respect to a particular control and usually have some effect on the control itself. The operation of most methods is affected by the current property values of the control.

Events are notifications generated by a control in response to some particular occurrence. Events are passed to the control container application to execute a particular subroutine in the program (event handler).

For example, the ComponentWorks IMAQ Hardware control has several properties that determine how images are acquired. To configure your image acquisition, set properties such as `Interface`, `Channel`, and `ColorMode`.

The IMAQ Hardware control has high-level methods, or functions, that you can invoke to perform specific operations. For example, use the `Start` method to start acquiring images.

The IMAQ Hardware control generates events when particular operations occur. For example, when an image is acquired, the IMAQ control passes an event to your program so you process the image in your application.

Note ***Use the online reference for specific information about the properties, methods, and events of the ActiveX controls. You can access the online reference by selecting* Programs»National Instruments ComponentWorks»IMAQ Vision»ComponentWorks IMAQ Reference *from the Windows* Start *menu.***

Object Hierarchy

The three parts of an ActiveX control—properties, methods, and events—are stored in a *software object*. Because some ActiveX controls are very complex and contain many properties, ActiveX controls are often subdivided into different software objects, the sum of which make up the ActiveX control. Each individual object in a control contains specific parts (properties) and functionality (methods and events) of the ActiveX control. The relationships among different objects of a control are maintained in an object hierarchy. At the top of the hierarchy is the actual control itself.

This top-level object contains its own properties, methods, and events. Some of the top-level object properties are actually references to other objects that define specific parts of the control. Objects below the top-level have their own methods and properties, and their properties can be references to other objects. The number of objects in a hierarchy is not limited.

Another advantage of subdividing controls is the re-use of different objects between different controls. One object might be used at different places in the same object hierarchy or in several different object hierarchies. For example, both the Hardware and Viewer controls use the Image object.

Figure 1-1 shows the object hierarchy of the ComponentWorks IMAQ control. The IMAQ object contains some of its own properties, such as `Name` and `Interface`. It also contains the `Images` property, which is a separate object. The Images object contains individual Image objects, each describing one image stored by the IMAQ control. Each Image object has properties, such as `Height` and `Width`, while the Images collection object has a property `Count`. The Images collection object is a special type of object referred to as a collection, which is described in the *Collection Objects* section.

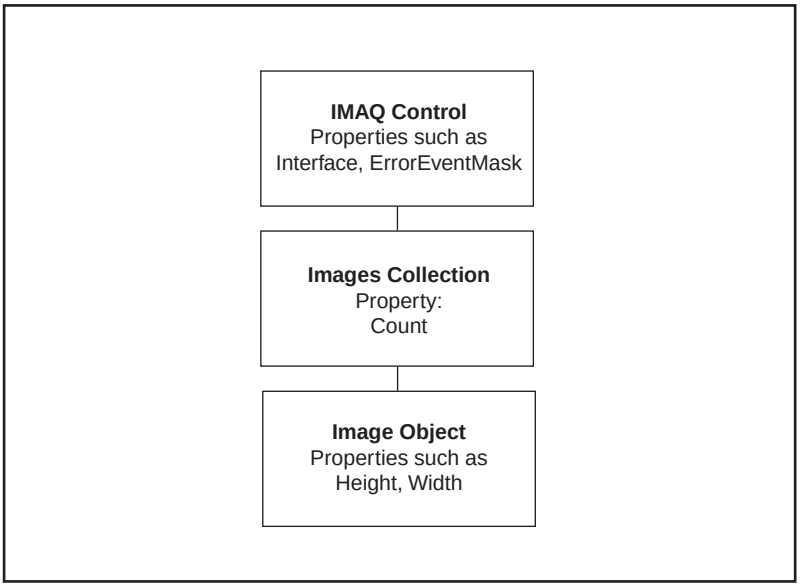


Figure 1-1. IMAQ Control Object Hierarchy

Collection Objects

One object can contain several objects of the same type. For example, the IMAQ object uses several Image objects, each representing one image acquired by the control. The number of objects in the group of objects might not be defined and might change while the program is running (that is, you can add or remove images as part of your program). To handle these groups of objects more easily, an object called a *collection* is created.

A collection is an object that contains or stores a varying number of objects of the same type. You can consider a collection as an array of objects. The name of a collection object is usually the plural of the name of the object type contained within the collection. For example, a collection of Image objects is referred to as Images. In the ComponentWorks software, the terms *object* and *collection* are rarely used, only the type names Image and Images are listed.

Each collection object contains an `Item` method that you can use to access any particular object stored in the collection. Refer to *Changing Properties Programmatically* later in this chapter for information about the `Item` method and accessing particular objects stored in the collection.

Setting the Properties of an ActiveX Control

You can set the properties of an ActiveX control from its property pages or from within your program.

Using Property Pages

Property pages are common throughout the Windows 95/98 and Windows NT interface. When you want to change the appearance or options of a particular object, right click on the object and select **Properties**. A property page or tabbed dialog box appears with a variety of properties that you can set for that particular object. You customize ActiveX controls in exactly the same way. Once you place the control on a form in your programming environment, right click on the control and select **Properties** to customize the appearance and operation of the control.

Use the property pages to set the property values for each ActiveX control while you are creating your application. The property values you select at this point represent the state of the control at the beginning of your application. You can change the property values from within your program as shown in the next section, *Changing Properties Programmatically*.

In some programming environments (such as Visual Basic and Delphi), you have two different property pages. The property page common to the programming environment is called the *default property sheet*; it contains the most basic properties of a control.

Your programming environment assigns default values for some of the basic properties, such as the control name and the tab order. You must edit these properties through the default property sheet.

Figure 1-2 shows the Visual Basic default property sheet for the IMAQ control.

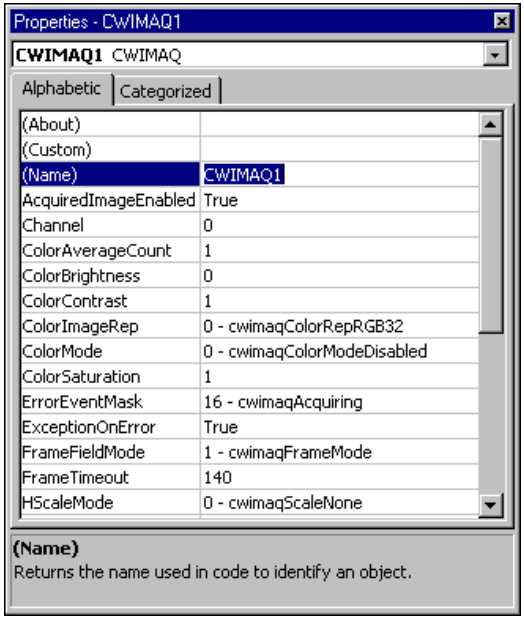


Figure 1-2. Visual Basic Default Property Sheets

The second property sheet is called the *custom property page*. The layout and functionality of the custom property pages vary for different controls. Figure 1-3 shows the custom property page for the IMAQ control.

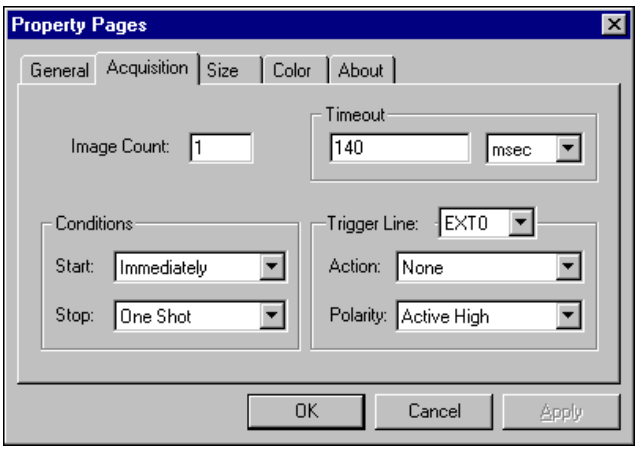


Figure 1-3. ComponentWorks Custom Property Pages

Changing Properties Programmatically

You also can set or read the properties of your controls programmatically. For example, if you want to change the color mode of an image acquisition during program execution, change the `ColorMode` property of the IMAQ control.

Note ***The exact syntax for reading and writing property values depends on the programming language. Refer to the appropriate*** *Building ComponentWorks IMAQ Vision Applications* ***chapter for information about using ComponentWorks in your programming environment. Code examples are written in Visual Basic syntax, which is similar to most implementations.***

Each control you create in your program has a name (like a variable name) which you use to reference the control in your program. You can set the value of a property on a top-level object with the following syntax.

```
name.property = new_value
```

For example, you can change the `ColorMode` property of an IMAQ control acquiring RGB images using the following line of code, where `CWIMAQ1` is the default name of the IMAQ control.

```
CWIMAQ1.ColorMode = cwimaqColorModeRGB
```

`cwimaqColorModeRGB` is a constant defined by the IMAQ control.

To access properties of sub-objects referenced by the top-level object, use the control name, followed by the name of the sub-object and the property name. For example, consider the following code for the Viewer control.

```
CWIMAQViewer1.Palette.Type = cwimaqPaletteGrayScale
```

In the above code, `Palette` is a property of the Viewer control and refers to a Palette object. `Type` is one of several Palette properties.

You can retrieve the value of control properties from your program in the same way. For example, you can print the value of the IMAQ `ColorMode` property.

```
Print CWIMAQ1.ColorMode
```

You can display the palette used by the Viewer control in a Visual Basic text box with the following code.

```
Text1.Text = CWIMAQViewer1.Palette.Type
```

Item Method

To access an object or its properties in a collection, use the `Item` method on the collection object. For example, set the border width of the second image of an IMAQ control with the following code.

```
CWIMAQ1.Images.Item(2).BorderWidth = 5
```

The term `CWIMAQ1.Images.Item(2)` refers to the second Image object in the Images collection of the IMAQ object. The parameter of the `Item` method is an integer representing the (one-based) index of the object in the collection.

Because the `Item` method is the most commonly used method on a collection, it is referred to as the *default method*. Therefore, some programming environments do not require you to specify the `.Item` method. For example, in Visual Basic

```
CWIMAQ1.Images(2).BorderWidth = 5
```

is programmatically equivalent to

```
CWIMAQ1.Images.Item(2).BorderWidth = 5
```

Working with Control Methods

ActiveX controls and objects have their own methods, or functions, that you can call from your program. Methods can have parameters that are passed to the method and return values that pass information back to your program.

For example, the `ExportStyle` method of the IMAQ control has a required parameter—the file used to save the state of the control—that you must include when you call the method.

```
CWIMAQ1.ExportStyle "c:\ContinuousIMAQ.cwx"
```

Methods can have required and optional parameters in some programming environments, such as Visual Basic. You can omit these optional parameters if you want to use their default values. Other programming environments require all parameters to be passed explicitly.

Depending on your programming environment, parameters might be enclosed in parentheses. If the function or method is not assigned a return variable, Visual Basic does not use parentheses to pass parameters. The `Copy` method in the Vision control has the following form when used with the return variable `status`.

```
status = CWIMAQVision1.Copy(SourceImage, DestImage)
```

Without the return value the syntax is

```
CWIMAQVision1.Copy SourceImage, DestImage
```

Developing Event Handler Routines

After configuring your controls on a form, you can create event handler routines in your program to respond to events generated by the controls. For example, the IMAQ control has an `AcquiredImage` event that *fires* (occurs) when the acquired image is ready to be processed based on the acquisition options you have configured in the control property pages.

You can configure the control to acquire single images or to acquire images continuously. Each time an image is acquired, the image buffer is ready and the `AcquiredImage` event is fired. In your `AcquiredImage` event routine, you can write code to analyze the image data, store it to disk, and display it.

To develop the event routine code, most programming environments generate a skeleton function to handle each event. For information about generating these function skeletons, refer to Chapter 3, *Building ComponentWorks IMAQ Vision Applications with Visual Basic*, Chapter 4, *Building ComponentWorks IMAQ Vision Applications with Visual C++*, and Chapter 5, *Building ComponentWorks IMAQ Vision Applications with Delphi*. For example, the Visual Basic environment generates the following function skeleton into which you insert the functions to call when the `AcquiredImage` event occurs.

```
Private Sub CWIMAQ1_AcquiredImage(ImageIndex As Variant)

End Sub
```

In most cases, the event also returns some data to the event handler that can be used in your event handler routine, such as `ImageIndex` in the previous example.

Learning the Properties, Methods, and Events

The ComponentWorks IMAQ online reference contains detailed information about each control and its associated properties, methods, and events. You can open the online reference from within most programming environments by clicking on the **Help** button in the custom property pages, or you can open it from the Windows **Start** menu by selecting **Programs»National Instruments ComponentWorks»IMAQ Vision»ComponentWorks IMAQ Reference**.

Some programming environments have built-in mechanisms for detailing the available properties, methods, and events for a particular control and sometimes include automatic links to the help file.

2

Getting Started with ComponentWorks

This chapter describes approaches to help you get started using ComponentWorks IMAQ Vision, depending on your application needs, your experience using ActiveX controls in your particular programming environment, and your specific goals in using ComponentWorks.

Install and Configure Driver Software

If you will be using the IMAQ Hardware control to acquire images from your IMAQ cards, you must install and configure the NI-IMAQ driver software before using this control.

The NI-IMAQ driver software performs the low-level calls to your hardware. It is configured using a separate configuration utility provided with the driver software. The configuration utility also provides parameters or values you need to use in your controls.

Install the most current version of the NI-IMAQ driver. The ComponentWorks controls might require features provided only in the newest version of the driver. You can install a version of NI-IMAQ that is compatible with the ComponentWorks IMAQ Vision controls from the CD (`\Drivers` directory). You also can download the most current version of the driver from the National Instruments Web or FTP site.

To run the installation and configuration programs, follow the directions provided with the driver. The driver includes a readme file or printed documentation that provides the latest information as well as operating system details.

Explore the ComponentWorks IMAQ Vision Documentation

The printed and online manuals contain the information necessary to learn and use the ComponentWorks IMAQ Vision controls to their full capabilities. The manuals are divided into different sections. Each section addresses a specific step on the learning curve.

Use the *Getting Results with ComponentWorks IMAQ Vision* manual to learn how to develop simple applications with the machine vision controls. The manual contains information you can use in specific circumstances, such as debugging particular problems.

After you understand the operation and organization of the controls, use the ComponentWorks IMAQ Vision online reference to obtain information about specific features of each control.

Getting Results with ComponentWorks IMAQ Vision Manual

The *Getting Results with ComponentWorks IMAQ Vision* manual contains three different parts.

Part I, *Building ComponentWorks IMAQ Vision Applications*—These chapters describe how to use ActiveX controls in the most commonly used programming environments—Visual Basic, Visual C++, and Borland Delphi.

If you are familiar with using ActiveX controls in these environments, you should not need to read these chapters. If you are using the controls in another environment, consult your programming environment documentation for information about using ActiveX controls. You can check the ComponentWorks Support Web site for information about additional environments.

Part II, *Using the ComponentWorks IMAQ Vision Controls*—These chapters describe the basic operation of the ComponentWorks IMAQ Vision controls. Each chapter contains an overview of a control, describing its most commonly used properties, methods, and events. The description also includes short code segments to illustrate programmatic control and tutorials that lead you through building an application with the control.

Part III, *Introduction to Vision*—These chapters present the basics of computer-based vision applications.

ComponentWorks Online Reference

The ComponentWorks IMAQ Vision online reference includes complete reference information for all controls—all properties, methods, and events for every control—as well as the text from *Getting Results with ComponentWorks IMAQ Vision*.

To use the online reference efficiently, you should understand the material presented in the *Getting Results with ComponentWorks IMAQ Vision* manual about using ComponentWorks ActiveX controls.

After going through this manual and its tutorials, use the online reference as your main source of information. Refer to it when you need specific information about a particular feature in ComponentWorks.

Accessing the Online Reference

You can open the online reference from the Windows **Start** menu (**Programs»National Instruments ComponentWorks»IMAQ Vision»ComponentWorks IMAQ Reference**). The reference opens to the main contents page. From the contents page, you can browse the contents of the online reference or search for a particular topic.

Most programming environments support some type of automatic link to the online reference (help) file from within their environment, often the <F1> key. Try selecting the control on a form or placing the cursor in code specific to a control and pressing <F1> to evoke the online reference.

In most environments, the property pages for the ComponentWorks controls include a **Help** button that provides information about the property pages.

Finding Specific Information

To find information about a particular control or feature of a control, select the **Index** tab under the **Help Topics** page. Enter the name of the control, property, method, or event. Control names always begin with `CW` (for example, `CWIMAQ`). Property, method, and event names are identical to those used in the code (for example, `Interface`, `Start`, `Images`).

One group of objects that frequently generates questions are the Collection objects. Search the online reference for `Collections` and the `Item` method for more information. You also can find information about collection objects in the *Collection Objects* section of Chapter 1, *Introduction to ComponentWorks IMAQ Vision*.

Become Familiar with the Examples Structure

The examples installed with ComponentWorks IMAQ Vision show you how to use the machine vision controls in applications. You can use these examples as a reference to become more familiar with the use of the controls, or you can build your application by expanding one of the examples.

When you install ComponentWorks IMAQ vision, you can install examples for selected programming environments. The examples are located in the `\ComponentWorks\samples` directory, organized by programming environment (`\Visual Basic`, `\Visual C++`, and so on), and grouped in the IMAQ folder under each language. Within these directories, the examples are further subdivided by functionality.

The online reference includes a searchable list of all the examples included with ComponentWorks IMAQ Vision. Select **Examples** to see the list of examples.

Develop Your Application

Depending on your experience with your programming environment, ActiveX controls, and ComponentWorks, you can get started using ComponentWorks IMAQ Vision in some of the following ways.

Are you new to your particular programming environment?

Spend some time using and programming in your development environment. Check the documentation that accompanies your programming environment for getting started information or tutorials, especially tutorials that describe using ActiveX controls in the environment. If you have specific questions, search the online documentation of your development environment. After becoming familiar with the programming environment, continue with the following steps.

Are you new to using ActiveX controls or do you need to learn how to use ActiveX controls in a specific programming environment?

Make sure you have read and understand the information about ActiveX controls in Chapter 1, *Introduction to ComponentWorks IMAQ Vision*, and the appropriate chapter about your specific programming environment. Refer to Table 2-1 to find out which chapter you should read for your specific programming environment.

If you use Borland C++ Builder, most of Chapter 5, *Building ComponentWorks IMAQ Vision Applications with Delphi*, pertains to you. If you use another programming environment, see the ComponentWorks Support Web site (`www.natinst.com/support`) for current information about particular environments.

Table 2-1. Chapters about Specific Programming Environments

Environment	Read This Chapter
Microsoft Visual Basic	Chapter 3, *Building ComponentWorks IMAQ Vision Applications with Visual Basic*
Microsoft Visual C++	Chapter 4, *Building ComponentWorks IMAQ Vision Applications with Visual C++*
Borland Delphi	Chapter 5, *Building ComponentWorks IMAQ Vision Applications with Delphi*

Regardless of the programming environment you use, consult its documentation for information about using ActiveX controls. After becoming familiar with using ActiveX controls in your environment, continue with the following steps.

Are you familiar with ActiveX controls but need to learn ComponentWorks controls, hierarchies, and features?

If you are familiar with using ActiveX controls, including collection objects and the `Item` method, read the chapters pertaining to the controls you want to use. Part II, *Using the ComponentWorks IMAQ Vision Controls*, provides basic information about each of the IMAQ vision controls and describes their most commonly used properties, methods, and events. The chapters also offer tutorials to help you become more familiar with using the controls. Solutions to each tutorial are installed with your software (`\ComponentWorks\Tutorials-IMAQ`).

After becoming familiar with the information in these chapters, try building applications with the ComponentWorks controls. You can find detailed information about all properties, methods, and events for every control in the online reference.

Do you want to develop applications quickly or modify existing examples?

If you are familiar with using ActiveX controls, including collections and the `Item` method, and have some experience using ComponentWorks or

other National Instruments products, you can get started more quickly by looking at the examples.

Most examples demonstrate how to perform operations with a particular control. Generally, the examples avoid presenting complex operations on more than one control. To become familiar with a control, look at the example for that control. Then, you can combine different programming concepts from the different controls in your application.

The examples include comments to provide more information about the steps performed in the example. The examples avoid performing complex programming tasks specific to one programming environment; instead, they focus on showing you how to perform operations using the ComponentWorks controls. When developing applications with ActiveX controls, you do a considerable amount of programming by setting properties in the property pages. Check the value of the control properties in the examples because the values greatly affect the operation of the example program. In some cases, the actual source code used by an example might not differ from other examples; however, the values of the properties change the example significantly.

Seek Information from Additional Sources

After working with the ComponentWorks controls, you might need to consult other sources if you have questions. The following sources can provide you with more specific information.

- *Getting Results with ComponentWorks IMAQ Vision* Appendices—The appendices include common questions and error descriptions for the IMAQ controls.
- ComponentWorks IMAQ Vision Online Reference—The online reference includes the complete reference documentation and text of this manual. If you cannot find a particular topic in the index, choose the **Find** tab in the **Help Topics** page and search the complete text of the online reference.
- ComponentWorks Support Web Site—The ComponentWorks Support Web site, as part of the National Instruments Support Web site (`www.natinst.com/support`), contains support information, updated continually. You can find application and support notes and information about using ComponentWorks in additional programming environments. The Web site also contains the KnowledeBase, a searchable database containing thousands of entries answering common questions related to the use of ComponentWorks and other National Instruments products.

Part I

Building ComponentWorks IMAQ Vision Applications

This section describes how to use ActiveX controls in the most commonly used programming environments—Visual Basic, Visual C++, and Borland Delphi.

If you are familiar with using ActiveX controls in these environments, you should not need to read these chapters. If you are using the controls in another environment, consult your programming environment documentation for information about using ActiveX controls. You can check the ComponentWorks Support Web site for information about additional environments.

Part I, *Building ComponentWorks IMAQ Vision Applications*, contains the following chapters.

- Chapter 3, *Building ComponentWorks IMAQ Vision Applications with Visual Basic*, describes how you can use the ComponentWorks controls with Visual Basic 5; insert the controls into the Visual Basic environment, set their properties, and use their methods and events; and perform these operations using ActiveX controls in general. This chapter also outlines Visual Basic features that simplify working with ActiveX controls.
- Chapter 4, *Building ComponentWorks IMAQ Vision Applications with Visual C++*, describes how you can use ComponentWorks controls with Visual C++, explains how to insert the controls into the Visual C++ environment and create the necessary wrapper classes, shows you how to create an application compatible with the ComponentWorks controls using the Microsoft Foundation Classes Application Wizard (MFC AppWizard) and how to build your

program using the ClassWizard with the controls, and discusses how to perform these operations using ActiveX controls in general.

- Chapter 5, *Building ComponentWorks IMAQ Vision Applications with Delphi*, describes how you can use ComponentWorks controls with Delphi; insert the controls into the Delphi environment, set their properties, and use their methods and events; and perform these operations using ActiveX controls. This chapter also outlines Delphi features that simplify working with ActiveX controls.

3

Building ComponentWorks IMAQ Vision Applications with Visual Basic

This chapter describes how you can use the ComponentWorks controls with Visual Basic 5; insert the controls into the Visual Basic environment, set their properties, and use their methods and events; and perform these operations using ActiveX controls in general. This chapter also outlines Visual Basic features that simplify working with ActiveX controls.

Note ***The descriptions and figures in this chapter apply specifically to the Visual Basic 5 environment.***

Developing Visual Basic Applications

The following procedure explains how you can start developing Visual Basic applications with ComponentWorks.

1. Select the type of application you want to build. Initially select a Standard EXE for your application type.
2. Design the form. A *form* is a window or area on the screen on which you place controls and indicators to create the user interface for your program. The toolbox in Visual Basic contains all of the controls available for developing the form.
3. After placing each control on the form, configure the properties of the control using the default and custom property pages.

   Each control on the form has associated code (event handler routines) in your Visual Basic program that automatically executes when the user operates that control.
4. To create this code, double click on the control while editing your application and the Visual Basic code editor opens to a default event handler routine.

Loading ComponentWorks IMAQ Vision Controls into the Toolbox

Before building an application using ComponentWorks controls, you must add them to the Visual Basic toolbox. Use the following procedure to add ComponentWorks controls to the project toolbox.

1. In a new Visual Basic project, right click on the toolbox and select **Components**.
2. Place a checkmark in the box next to **National Instruments CW IMAQ**.

   If the ComponentWorks controls are not in the list, select the control files from the `\Windows\System(32)` directory by pressing the **Browse** button.

If you need to use the ComponentWorks controls in several projects, create a new default project in Visual Basic 5 to include the IMAQ Vision controls and serve as a template.

1. Create a new Standard EXE application in the Visual Basic environment.
2. Add the ComponentWorks IMAQ controls to the project toolbox as described in the preceding procedure.
3. Save the form and project in the `\Template\Projects` directory under your Visual Basic directory.
4. Give the form and project a descriptive name, such as `CWForm` and `CWProject`.

After creating this default project, you have a new option, `CWProject`, that includes the ComponentWorks IMAQ controls in the **New Project** dialog by default.

Building the User Interface Using ComponentWorks

After you add the ComponentWorks IMAQ controls to the Visual Basic toolbox, use them to create the front panel of your application. To place the controls on the form, select the corresponding icon in the toolbox and click and drag the mouse on the form. This step creates the corresponding control. After you create controls, move and size them by using the mouse. To move a control, click and hold the mouse on the control and drag the control to the desired location. To resize a control, select the control and place the mouse pointer on one of the hot spots on the border of the control. Drag the border to the desired size. Notice that you cannot resize the IMAQ Hardware or Vision icons after placing them on the form. They also are not visible during run time.

Once ActiveX controls are placed on the form, you can edit their properties using their property sheets. You can also edit the properties from within the Visual Basic program at run time.

Using Property Pages

After placing a control on a Visual Basic form, configure the control by setting its properties in the Visual Basic property pages (see Figure 3-1) and ComponentWorks custom control property pages (see Figure 3-2). Visual Basic assigns some default properties, such as the control name and the tab order. When you create the control, you can edit these stock properties in the Visual Basic default property sheet. To access this sheet, select a control and select **Properties Window** from the **View** menu, or press <F4>. To edit a property, highlight the property value on the right side of the property sheet and type in the new value or select it from a pull down menu. The most important property in the default property sheet is `Name`, which is used to reference the control in the program.

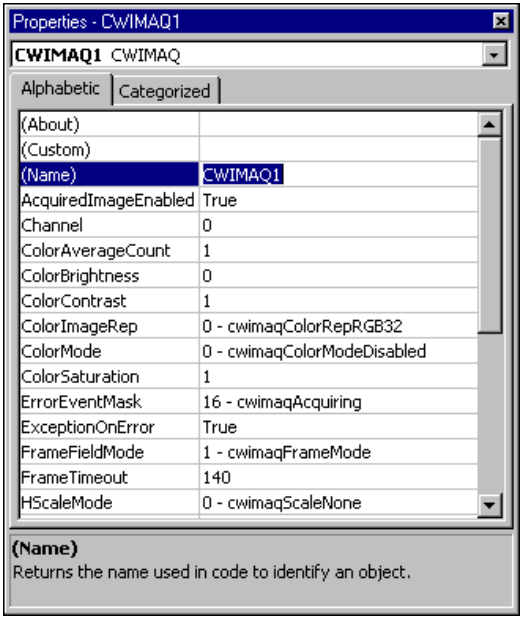


Figure 3-1. Visual Basic Property Pages

Edit all other properties of an ActiveX control in the custom property sheets. To open the custom property sheets, right click on the control on the form and select **Properties** or select the controls and press <Shift-F4>.

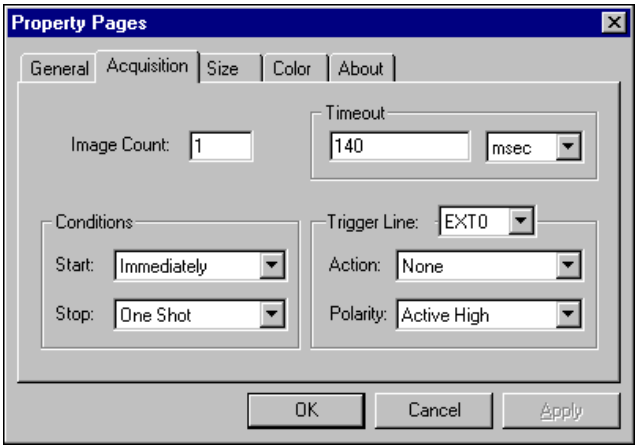


Figure 3-2. ComponentWorks Custom Property Pages

Using Your Program to Edit Properties

You can set and read the properties of your controls programmatically in Visual Basic. Use the name of the control with the name of the property as you would with any other variable in Visual Basic. The syntax for setting a property in Visual Basic is `name.property` = `new value`.

For example, you can change the `ColorMode` property of an IMAQ control acquiring RGB images using the following line of code, where `CWIMAQ1` is the default name of the IMAQ control.

```
CWIMAQ1.ColorMode = cwimaqColorModeRGB
```

`cwimaqColorModeRGB` is a constant defined by the IMAQ control.

To access properties of sub-objects referenced by the top-level object, use the control name, followed by the name of the sub-object and the property name. For example, consider the following code for the Viewer control.

```
CWIMAQViewer1.Palette.Type = cwimaqPaletteGrayScale
```

In the above code, `Palette` is a property of the Viewer control and refers to a Palette object. `Type` is one of several Palette properties.

You can retrieve the value of control properties from your program in the same way. For example, you can print the value of the IMAQ `ColorMode` property.

```
Print CWIMAQ1.ColorMode
```

You can display the palette used by the Viewer control in a Visual Basic text box with the following code.

```
Text1.Text = CWIMAQViewer1.Palette.Type
```

Working with Control Methods

Calling the methods of an ActiveX control in Visual Basic is similar to working with the control properties. To call a method, add the name of the method after the name of the control (and sub-object if applicable). For example, you can call the `Start` method on the IMAQ Hardware control.

```
CWIMAQ1.Start
```

Methods can have parameters that you pass to the method and return values that pass information back to your program. For example, the `ExportStyle` method of the IMAQ control has a required parameter—the file used to save the state of the control—that you must include when you call the method.

```
CWIMAQ1.ExportStyle "c:\ContinuousIMAQ.cwx"
```

In Visual Basic if you call a method without assigning a return variable, any parameters passed to the method are listed after the method name, separated by commas without parentheses.

```
CWIMAQVision1.Copy SourceImage, DestImage
```

If you assign the return value of a method to a return variable, enclose the parameters in parentheses.

```
status = CWIMAQVision1.Copy (SourceImage, DestImage)
```

You can use the Visual Basic Object Browser to add method calls to your program.

Developing Control Event Routines

After you configure your controls in the forms editor, write Visual Basic code to respond to events on the controls. The controls generate these events in response to user interactions with the controls or in response to some other occurrence in the control. To develop the event handler routine code for an ActiveX control in Visual Basic, double click on the control to open the code editor, which automatically generates a default event handler routine for the control. The event handler routine skeleton includes the control name, the default event, and any parameters that are passed to the event handler routine. The following code is an example of

the event routine generated for the IMAQ Hardware control. This event routine (`AcquiredImage`) is called when a new image is acquired.

```
Private Sub CWIMAQ1_AcquiredImage(ImageIndex As Variant)
End Sub
```

To generate an event handler for a different event of the same control, double click the control to generate the default handler, and select the desired event from the right pull-down menu in the code window, as shown in the following illustration.

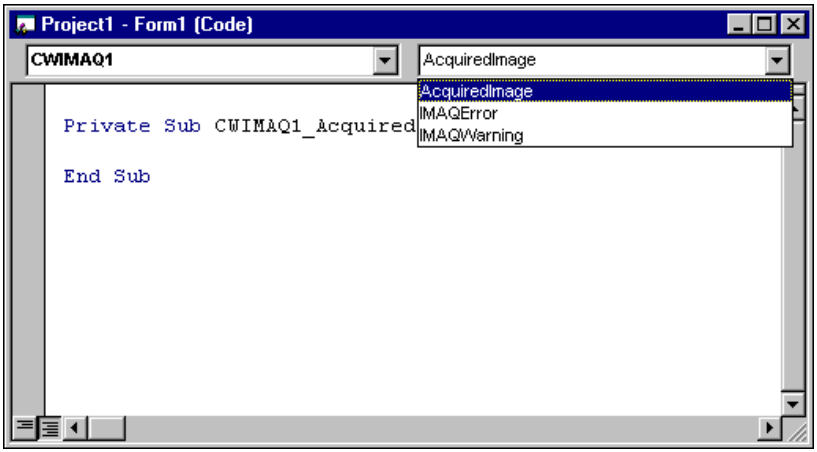


Figure 3-3. Selecting Events in the Code Window

Use the left pull-down menu in the code window to change to another control without going back to the form window.

Using the Object Browser to Build Code in Visual Basic

Visual Basic includes a tool called the Object Browser that you can use to work with ActiveX controls while creating your program. The Object Browser displays a detailed list of the available properties, methods, and events for a particular control. It presents a three-step hierarchical view of controls or libraries and their properties, methods, functions, and events. To open the Object Browser, select **Object Browser** from the **View** menu, or press <F2>.

In the Object Browser, use the top left pull-down menu to select a particular ActiveX control file. You can select any currently loaded control or driver. The Classes list on the left side of the Object Browser displays a list of

controls, objects, and function classes available in the selected control file or driver.

Figure 3-4 shows the ComponentWorks IMAQ control file selected in the Object Browser. The Classes list shows all the IMAQ controls and associated object types. Each time you select an item from the Classes list in the Object Browser, the Members list on the right side displays the properties, methods, and events for the selected object or class.

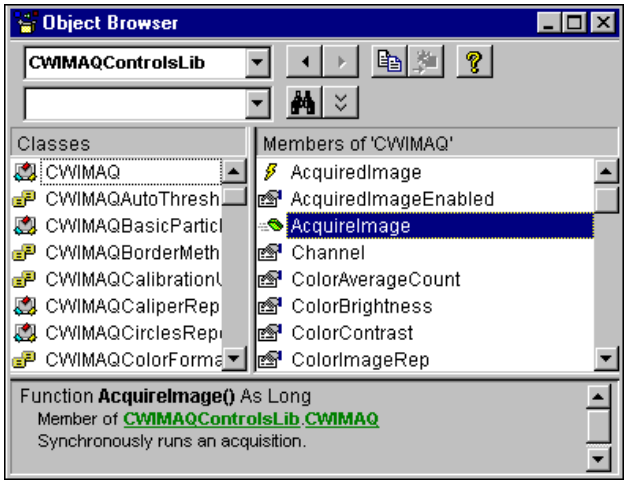


Figure 3-4. Viewing CWIMAQ in the Object Browser

When you select an item in the Members list, the prototype and description of the selected property, method, or function are displayed at the bottom of the Object Browser dialog box. In Figure 3-4, the CWIMAQ control is selected from the Classes list. For this control, the `AcquireImage` method is selected and the prototype and description of the method appear in the dialog box. The prototype of a method or function lists all parameters, required and optional.

When you select a property of a control or object in the Members list which is an object in itself, the description of the property includes a reference to the object type of the property. For example, Figure 3-5 shows the CWIMAQ control selected in the Classes list and its `Images` property selected in the Members list.

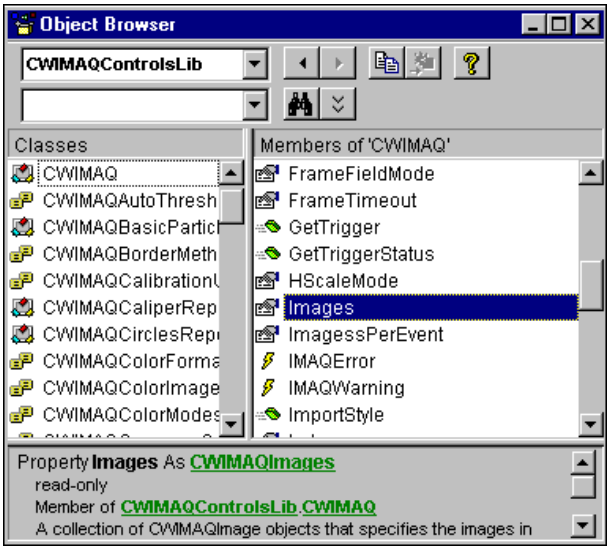


Figure 3-5. Viewing CWIMAQ in the Object Browser

The Images object on the CWIMAQ control is a separate object, so the description at the bottom of the dialog window lists the Images property as CWIMAQImages. CWIMAQImages is the type name of the Images collection object, and you can select CWIMAQImages in the Classes list to see its properties and methods. Move from one level of the object hierarchy to the next level using the Object Browser to explore the structure of different controls.

The question mark (**?**) button at the top of the Object Browser opens the help file to a description of the currently selected item. To find more information about the CWIMAQ control, select the control in the window and press the **?** button.

Pasting Code into Your Program

If you open the Object Browser from the Visual Basic code editor, you can copy the name or prototype of a selected property, method, or function to the clipboard and then paste it into your program. To perform this task, select the desired Member item in the Object Browser. Press the **Copy to Clipboard** button at the top of the Object Browser or highlight the prototype at the bottom and press <Ctrl-C> to copy it to the clipboard. Paste it into your code window by selecting **Paste** from the **Edit** menu or pressing <Ctrl-V>.

Use this method repeatedly to build a more complex reference to a property of a lower-level object in the object hierarchy. For example, you can create a reference to

```
CWIMAQ1.Images.Item(1).Type
```

by typing in the name of the control (`CWIMAQ1`) and then using the Object Browser to add each section of the property reference.

Adding Code Using Visual Basic Code Completion

Visual Basic 5 supports automatic code completion in the code editor. As you enter the name of a control, the code editor prompts you with the names of all appropriate properties and methods. Try placing a control on the form and then entering its name in the code editor. After typing the name, add a period as the delimiter to the property or method of the control. As soon as you type the period, Visual Basic drops down a menu of available properties and methods, as shown in Figure 3-6.

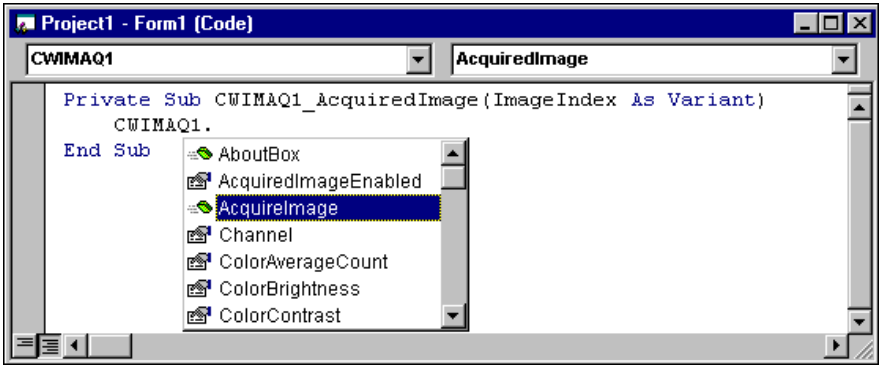


Figure 3-6. Visual Basic 5 Code Completion

You can select from the list or properties and events by scrolling through the list and selecting one or by typing in the first few letters of the desired item. Once you have selected the correct item, type the next logical character such as a period, space, equal sign, or carriage return to enter the selected item in your code and continue editing the code.

Creating Standalone Objects

Standalone objects are ActiveX objects created at run time, so you do not place them on a form. They are used as parameters to many IMAQ Vision functions. ComponentWorks uses two kinds of standalone objects: Images and Reports. Standalone Image objects are identical to the Image objects found on the CWIMAQ and CWIMAQViewer controls.

Standalone Image objects can store temporary results of image processing functions that you do not want to display. Standalone Report objects are arrays of data used as inputs and outputs to IMAQ Vision functions.

You can create standalone objects in Visual Basic using `Create` methods on the CWIMAQVision control, such as `CreateCWIMAQImage`, `CreateCWIMAQCaliperReport`, and `CreateCWIMAQShapeReport`.

```
' Create a standalone object
Dim Image as New CWIMAQImage
```

Standalone objects are automatically deallocated when the variable goes out of scope.

4

Building ComponentWorks IMAQ Vision Applications with Visual C++

This chapter describes how you can use ComponentWorks controls with Visual C++, explains how to insert the controls into the Visual C++ environment and create the necessary wrapper classes, shows you how to create an application compatible with the ComponentWorks controls using the Microsoft Foundation Classes Application Wizard (MFC AppWizard) and how to build your program using the ClassWizard with the controls, and how to perform these operations using ActiveX controls in general.

Note ***The descriptions and figures in this chapter apply specifically to the Visual C++ 5 environment.***

Developing Visual C++ Applications

The following procedure explains how you can start developing Visual C++ applications with ComponentWorks.

1. Create a new workspace or project in Visual C++.
2. To create a project compatible with the ComponentWorks ActiveX controls, use the Visual C++ MFC AppWizard to create a skeleton project and program.
3. After building the skeleton project, add the ComponentWorks controls to the controls toolbar. From the toolbar, you can add the controls to the application itself.
4. After adding a control to your application, configure its properties using its property pages.
5. While developing your program code, use the control properties and methods and create event handlers to process different events generated by the control.

   Create the necessary code for these different operations using the ClassWizard in the Visual C++ environment.

Creating Your Application

When developing new applications, use the MFC AppWizard to create new project workspace so the project is compatible with ActiveX controls. The MFC AppWizard creates the project skeleton and adds the necessary code that enables you to add ActiveX controls to your program.

1. Create a new project by selecting **New** from the **File** menu. The **New** dialog box opens (see Figure 4-1).

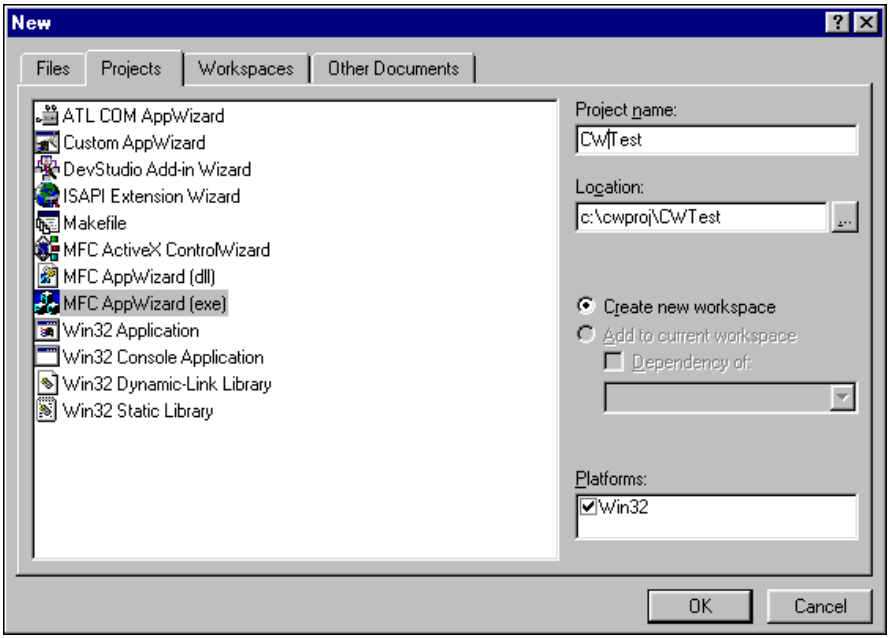


Figure 4-1. New Dialog Box

2. On the **Projects** tab, select the MFC AppWizard (exe) and enter the project name and the directory.
3. Click on **OK** to setup your project.

   Complete the next series of dialog windows in which the MFC AppWizard prompts you for different project options. If you are a new Visual C++ or the MFC AppWizard user, accept the default options unless otherwise stated in this documentation.
4. In the first step, select the type of application you want to build. For this example, select a dialog-based application, as shown in Figure 4-2, to make it easier to become familiar with the ComponentWorks controls.

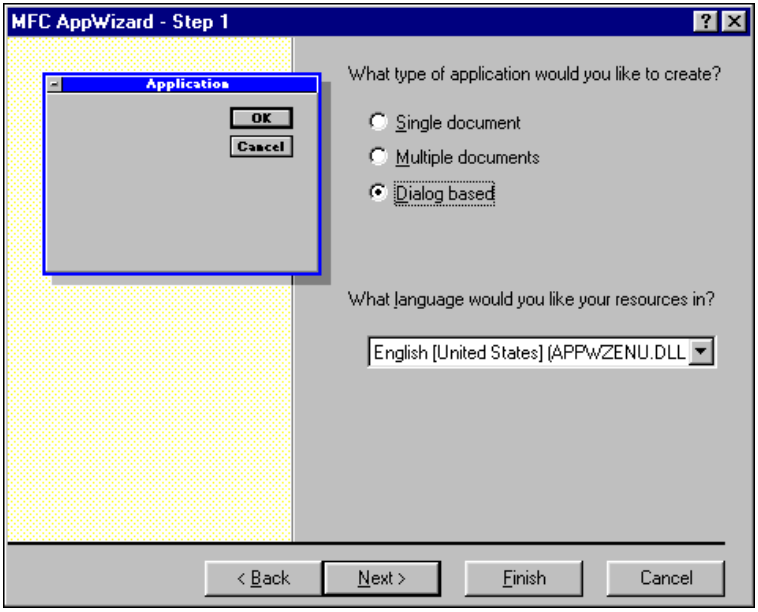


Figure 4-2. MFC AppWizard—Step 1

5. Click on the **Next>** button to continue.
6. Enable ActiveX controls support. If you have selected a Dialog based application, step two of the MFC AppWizard enables **ActiveX Controls** support by default.
7. Continue selecting desired options through the remainder of the MFC AppWizard. When you finish the MFC AppWizard, it builds a project and program skeleton according to the options you specified. The skeleton includes several classes, resources, and files, all of which can be accessed from the Visual C++ development environment.
8. Use the Workspace window, which you can select from the **View** menu, to see the different components in your project.

Adding ComponentWorks Controls to the Visual C++ Controls Toolbar

Before building an application using the ComponentWorks IMAQ controls, you must load the controls into the Controls toolbar in Visual C++ from the Component Gallery in the Visual C++ environment. When you load the controls using the Component Gallery, a set of C++ wrapper classes is generated automatically in your project. You must have wrapper classes to work with the ComponentWorks controls.

The Controls toolbar is visible in the Visual C++ environment only when the Visual C++ dialog editor is active. Use the following procedure to open the dialog editor.

1. Open the Workspace window by selecting **Workspace** from the **View** menu.
2. Select the Resource View (second tab along the bottom of the Workspace window).
3. Expand the resource tree and double click on one of the Dialog entries.
4. If necessary, right click on any existing toolbar and enable the Controls option.

By adding controls to your project, you create the necessary wrapper classes for the control in your project and add the control to the toolbox. Use the following procedure to add new controls to the toolbar.

1. Select **Project»Add To Project»Components and Controls** and, in the following dialog, double click on Registered ActiveX Controls.
2. Select one of the ComponentWorks IMAQ controls and click the **Insert** button.
3. Click on **OK** in the following dialog windows. Repeat Steps 1 through 3 to add other controls.
4. When you have inserted the controls, click **Close** in the Components and Controls Gallery.

Building the User Interface Using ComponentWorks

After adding the controls to the Controls toolbar, use the controls in the design of the application user interface. Place the controls on the dialog form using the dialog editor. You can size and move individual controls in the form to customize the interface. Use the custom property sheets to configure control representation on the user interface and control behavior at run time.

To add ComponentWorks controls to the form, open the dialog editor by selecting the dialog form from the Resource View of the Workspace window. If the Controls toolbar is not displayed in the dialog editor, open it by right clicking on any existing toolbar and enabling the Controls option.

To place a ComponentWorks control on the dialog form, select the desired control in the Controls toolbar and click and drag the mouse on the form to create the control. After placing the controls, move and resize them on the form as needed.

After you add a ComponentWorks control to a dialog form, configure the default properties of the control by right clicking the control and selecting **Properties** to display its custom property sheets. Figure 4-3 shows the CWIMAQ control property pages.

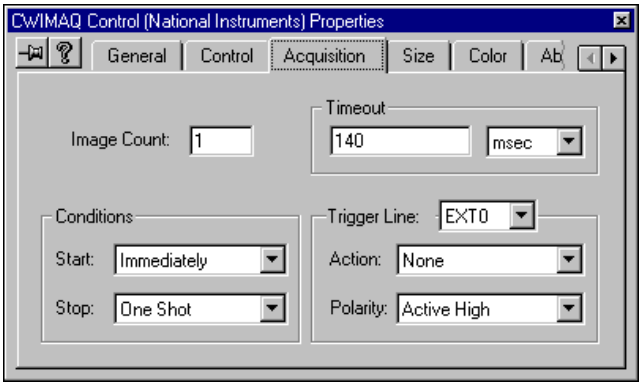


Figure 4-3. CWIMAQ Control Property Sheets

So you can see immediately how different properties affect the control, a separate window displays a sample copy of the control that reflects the property changes as you make them in the property sheets.

Programming with the ComponentWorks Controls

To program with ComponentWorks controls, use the properties, methods, and events of the controls as defined by the wrapper classes in Visual C++.

Before you can use the properties or methods of a control in your Visual C++ program, assign a member variable name to the control. This member variable becomes a variable of the application dialog class in your project.

To create a member variable for a control on the dialog form, right click on the control and select **ClassWizard**. In the **MFC Class Wizard** window, activate the **Member Variables** tab, as shown in Figure 4-4.

Select the new control in the Control IDs field and press the **Add Variable** button. In the dialog window that appears, complete the member variable name and press **OK**. Most member variable names start with `m_`, and you should adhere to this convention. After you create the member variable, use it to access a control from your source code. Figure 4-4 shows the MFC Class Wizard after member variables have been added for each of the IMAQ controls.

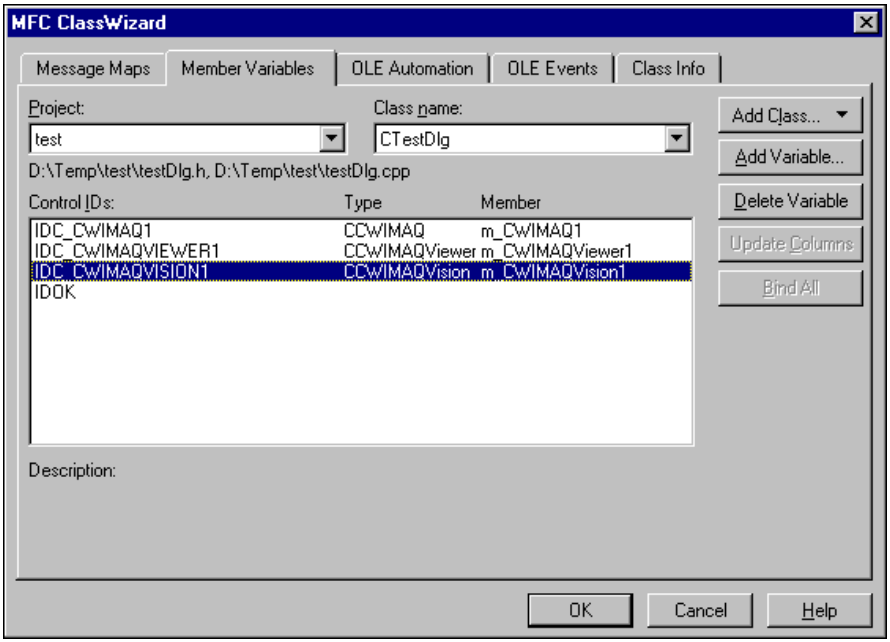


Figure 4-4. MFC ClassWizard—Member Variable Tab

Using Properties

Unlike Visual Basic, you do not read or set the properties of ComponentWorks controls directly in Visual C++. Instead, the wrapper class of each control contains functions to read and write the value of each property. These functions are named starting with either `Get` or `Set` followed by the name of the property. For example, to set the `BorderWidth` property of a viewer, use the `SetBorderWidth` function of the wrapper class for the Viewer control. In the source code, the function call is preceded by the member variable name of the control to which it applies.

```
m_CWIMAQViewer1.SetBorderWidth(5);
```

Some values passed to properties need to be of variant type. Convert the value passed to the property to a variant using `COleVariant()`. For example, set the `ZoomScale` property of a Viewer control.

```
m_CWIMAQViewer1.SetZoomScale(COleVariant(1.0));
```

You can view the names of all the property functions (and other functions) for a given control in the ClassView of the Workspace window. In the Workspace window, select **ClassView** and then the control for which you

want to view property functions and methods. Figure 4-5 shows the functions for the IMAQ Viewer object as listed in the Workspace. These are created automatically when you add a control to the Controls toolbar in you project.

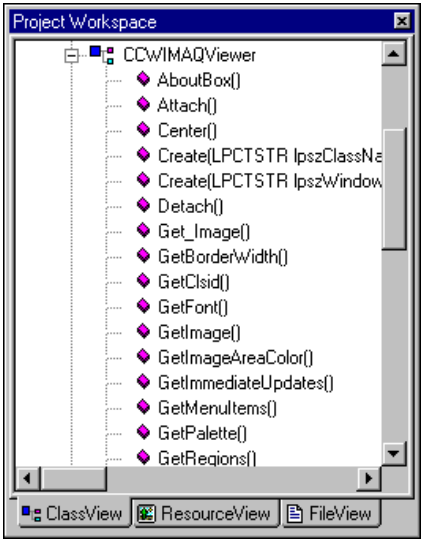


Figure 4-5. Viewing Property Functions and Methods in the Workspace Window

If you need to access a property of a control which is in itself another object, use the appropriate property function to return the sub-object of the control. Make a call to access the property of the sub-object. Include the header file in your program for any objects used. For example, use the following code to set the border width of the image contained in the Viewer control.

```
#include "_cwimaqimage.h"
C_CWIMAQImage Image;
Image = m_CWIMAQViewer1.GetImage();
Image.SetBorderWidth(5);
```

You can chain this operation into one function call without having to declare another variable.

```
#include "_cwimaqimage.h"
m_CWIMAQViewer1.GetImage().SetBorderWidth(5);
```

If you need to access an object in a collection property, use the `Item` method with the index of the object. Remember to include the header file for the collection object. For example, to set the border width of the first image on an IMAQ control, use the following code.

```
#include "_cwimaqimage.h"
#include "cwimaqimages.h"
m_CWIMAQ1.GetImages().Item(COleVariant(1.0)).
    SetBorderWidth(5);
```

Using Methods

Use the control wrapper classes to extract all methods of the control. To call a method, append the method name to the member variable name and pass the appropriate parameters. If the method does not require parameters, use a pair of empty parentheses.

```
m_CWIMAQ1.Start();
```

Most methods take some parameters as variants. You must convert any such parameter to a variant if you have not already done so. You can convert most scalar values to variants with `COleVariant()`. For example, the `Add` method of the Vision control requires a scalar value as variant.

```
#include "_cwimaqimage.h"
C_CWIMAQImage Image;
Image = m_CWIMAQViewer1.GetImage();
m_CWIMAQVision1.Add(Image, COleVariant(128.0), Image);
```

Note ***Consult Visual C++ documentation for more information about variant data types.***

If you need to call a method on a sub-object of a control, follow the conventions outlined in the *Using Properties* section earlier in this chapter. For example, to call `GetRegions` on one particular region of the viewer, use the following line of code.

```
#include "cwimaqregions.h"
#include "cwimaqregion.h"
m_CWIMAQViewer1.GetRegions().Item(COleVariant(1.0)).
    Move(10,-10);
```

Using Events

After placing a control on your form, you can start defining event handler functions for the control in your code. Events generate automatically at run time when different controls respond to conditions, such as a user clicking on the viewer on the form or the image acquisition process acquiring an image.

Use the following procedure to create an event handler.

1. Right click on a control and select **ClassWizard**.
2. Select the **Message Maps** tab and the desired control in the Object IDs field. The Messages field displays the available events for the selected control. (See Figure 4-6).
3. Select the event and press the **Add Function** button to add the event handler to your code.
4. To switch directly to the source code for the event handler, click on the **Edit Code** button. The cursor appears in the event handler, and you can add the functions to call when the event occurs. You can use the **Edit Code** button at any time by opening the class wizard and selecting the event for the specific control.

The following figure is an example of an event handler generated for the `AcquiredImage` event of a knob.

```
#include "_cwimaqimage.h"
void CTestDlg::OnAcquiredImageCwimaq1(VARIANT FAR*
    ImageIndex)
{
    // Create a new image object.
    C_CWIMAQImage Image1 =
       m_CWIMAQVision1.CreateCWIMAQImage();
    //Copy the image from the IMAQ control to a
    // separate image object.
    m_CWIMAQVision1.Copy
       (m_CWIMAQ1.GetImages().Item(*ImageIndex),
       Image1);
}
```

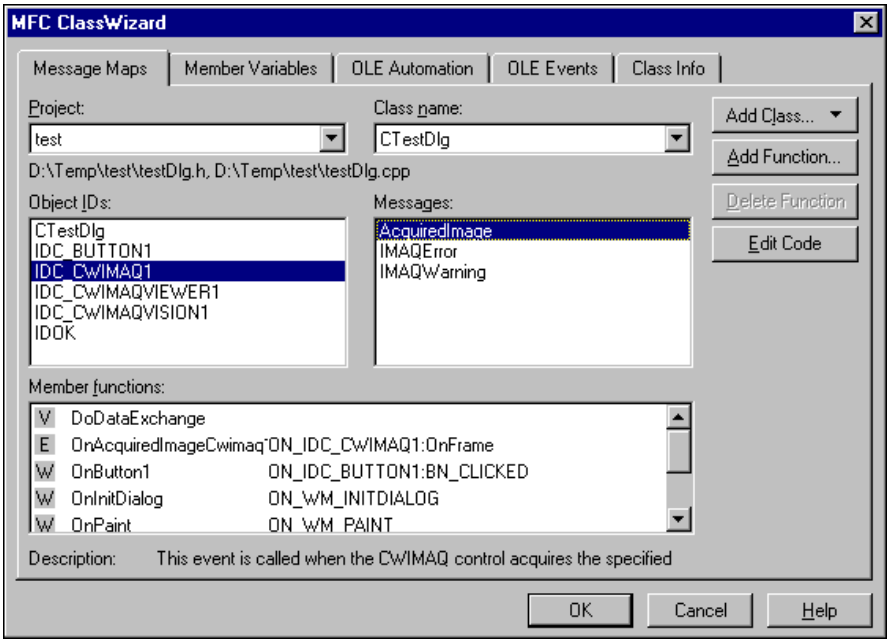


Figure 4-6. Event Handler

Creating Standalone Objects

Standalone objects are ActiveX objects created at run time, so you do not place them on a form. They are used as parameters to many IMAQ Vision functions. ComponentWorks uses two kinds of standalone objects: Images and Reports. Standalone Image objects are identical to the Image objects found on the CWIMAQ and CWIMAQViewer controls.

Standalone Image objects can store temporary results of image processing functions that you do not want to display. Standalone Report objects are arrays of data used as inputs and outputs to IMAQ Vision functions.

You can create standalone objects in Visual C++ using `Create` methods on the CWIMAQVision control, such as `CreateCWIMAQImage`, `CreateCWIMAQCaliperReport`, and `CreateCWIMAQShapeReport`.

```
// Create a standalone image
C_CWIMAQImage Image =
    m_CWIMAQVision1.CreateCWIMAQImage();
```

Standalone objects are automatically deallocated when the variable goes out of scope.

5

Building ComponentWorks IMAQ Vision Applications with Delphi

This chapter describes how you can use ComponentWorks controls with Delphi; insert the controls into the Delphi environment, set their properties, and use their methods and events; and perform these operations using ActiveX controls. This chapter also outlines Delphi features that simplify working with ActiveX controls.

Note ***The descriptions and figures in this chapter apply specifically to the Delphi 3 environment. If you have the original release of Delphi 3, you might experience significant problems with ActiveX controls, but Borland offers a newer version of Delphi that corrects most of these problems. Before using ComponentWorks with Delphi 3, contact Borland to receive the Delphi 3 patch or a newer version.***

Running Delphi Examples

To run the Delphi examples installed with ComponentWorks, you need to import the controls into the Delphi environment. See the section on *Loading ComponentWorks Controls into the Component Palette* for more information about loading the controls.

Developing Delphi Applications

You start developing applications in Delphi using a form. A *form* is a window or area on the screen on which you can place controls and indicators to create the user interface for your programs. The Component palette in Delphi contains all of the controls available for building applications. After placing each control on the form, configure the properties of the control with the default and custom property pages. Each control you place on a form has associated code (event handler routines) in the Delphi program that automatically executes when the user operates the control or the control generates an event.

Loading ComponentWorks Controls into the Component Palette

Before you can use the ComponentWorks controls in your Delphi applications, you must add them to the Component palette in the Delphi environment. You need to add the controls to the palette only once because the controls remain in the Component palette until you explicitly remove them. When you add controls to the palette, you create Pascal import units (header files) that declare the properties, methods, and events of a control. When you use a control on a form, a reference to the corresponding import unit is automatically added to the program.

Note ***Before adding a new control to the Component palette, make sure to save all your work in Delphi, including files and projects. After loading the controls, Delphi closes any open projects and files to complete the loading process.***

Use the following procedure to add ActiveX controls to the Component palette.

1. Select **Import ActiveX Control** from the **Component** menu in the Delphi environment. The Import ActiveX Control window displays a list of currently registered controls.

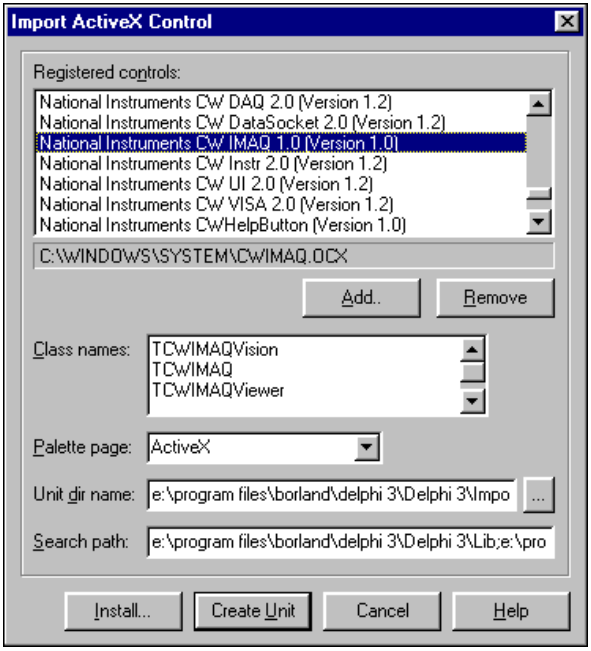


Figure 5-1. Delphi Import ActiveX Control Dialog Box

2. Select **National Instruments CW IMAQ** to add the IMAQ controls to the Component palette.
3. After selecting the control group, click **Install**.

   Delphi generates a Pascal import unit file for the selected `.OCX` file, which is stored in the Delphi `\Imports` directory. If you have installed the same `.OCX` file previously, Delphi prompts you to overwrite the existing import unit file.
4. In the **Install** dialog box, click on **OK** to add the controls to the Delphi user's components package.
5. In the following dialog, click on **Yes** to rebuild the user's components package with the added controls. Another dialog box acknowledges the changes you have made to the user's components package, and the package editor displays the components currently installed.

   At this point, you can add additional ActiveX controls with the following procedure.

   a. Click on the **Add** button.
   b. Select the **Import ActiveX** tab.
   c. Select the ActiveX control you want to add.
   d. Click on **OK**.
   e. After adding the ActiveX controls, compile the user's components package.

If your control does not appear in the list of registered controls, click the **Add** button. To register a control with the operating system and add it to the list of registered controls, browse to and select the OCX file that contains the control. Most OCX files reside in the `\Windows\System(32)` directory.

New controls are added to the **ActiveX** tab in the Components palette. You can rearrange the controls or add a new tab to the Components palette by right clicking on the palette and selecting **Properties**.

Building the User Interface

After you add the ComponentWorks controls to the Component palette, use them to create the user interface. Open a new project, and place different controls on the form. After placing the controls on the form, configure their default property values through the stock and custom property sheets.

Placing Controls

To place a control on the form, select the control from the Component palette and click and drag the mouse on the form. Use the mouse to move and resize controls to customize the interface, as in Figure 5-2. After you place the controls, you can change their default property values by using the default property sheet (Object Inspector) and custom property sheets.

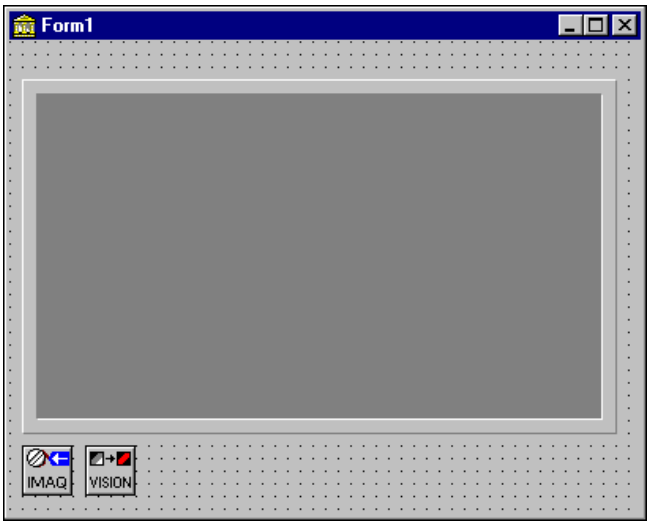


Figure 5-2. ComponentWorks Controls on a Delphi Form

Using Property Pages

Set property values such as `Name` in the Object Inspector of Delphi. To open the Object Inspector, select **Object Inspector** from the **View** menu or press <F11>. Under the **Properties** tab of the Object Inspector, you can set different properties of the selected control.

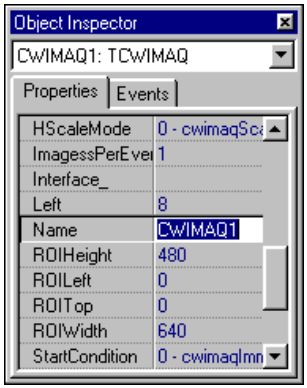


Figure 5-3. Delphi Object Inspector

To open the custom property pages of a control, double click on the control or right click on the control and select **Properties**. You can edit most control properties from the custom property pages. The following figure shows the ComponentWorks IMAQ Hardware control property page.

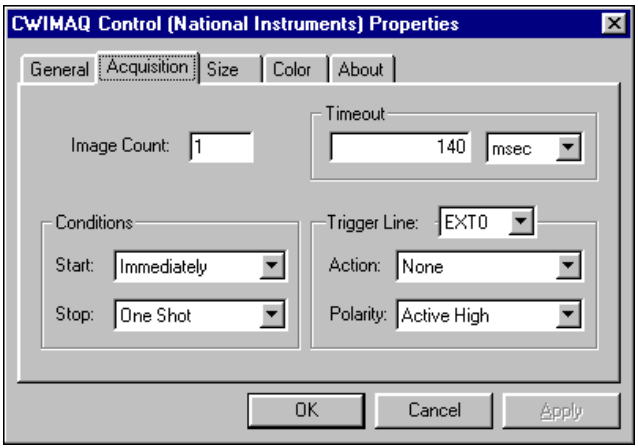


Figure 5-4. ComponentWorks IMAQ Control Property Pages

Programming with ComponentWorks

The code for each form in Delphi is listed in the Associated Unit (code) window. You can toggle between the form and Associated Unit window by pressing <F12>. After placing controls on the form, use their methods in your code and create event handler routines to process events generated by the controls at run time.

Using Your Program to Edit Properties

You can set or read control properties programmatically by referencing the name of the control with the name of the property, as you would any variable name in Delphi. The name of the control is set in the Object Inspector.

The syntax for setting the `Value` property in Delphi is

```
name.property := new_value;
```

For example, you can change the `ColorMode` property of an IMAQ control acquiring RGB images using the following line of code, where `CWIMAQ1` is the default name of the IMAQ control and `cwimaqColorModeRGB` is a constant defined by the IMAQ control.

```
CWIMAQ1.ColorMode := cwimaqColorModeRGB;
```

A property can be an object itself that has its own properties. To set properties in this case, combine the name of the control, sub-object, and property. For example, consider the following code for the IMAQ Viewer control. `Palette` is both a property of the Viewer control and an object itself. `Type` is a property of the `Palette` object. As an object of the Viewer control, `Palette` itself has several additional properties.

```
CWIMAQViewer1.Palette.Type := cwimaqPaletteGrayScale;
```

You can retrieve the value of a control property from your program in the same way. For example, you can assign the width of an image to a text box on the user interface.

```
Edit1.Text := CWIMAQViewer1.Image.Width;
```

To use the properties or methods of an object in a collection, use the `Item` method to extract the object from the collection. Once you extract the object, use its properties and methods as you usually would.

```
CWIMAQ1.Images.Item(1).Type := cwimaqImageTypeRGB32;
```

Using Methods

Each control has defined methods that you can use in your program. To call a method in your program, use the control name followed by the method name.

```
CWIMAQ1.Start;
```

Some methods require parameters, as does the following method.

```
CWIMAQVision1.Copy (SourceImage, DestImage);
```

In most cases, parameters passed to a method are of type variant. Simple scalar values can be automatically converted to variants and, therefore, might be passed to methods. Arrays, however, must be explicitly declared as variant arrays.

```
procedure TForm1.Button1Click(Sender: TObject);
var
   image1: CWIMAQImage;
   dataIn, dataOut: OleVariant;
   i,j : integer;
begin
   image1 := CWIMAQVision1.CreateCWIMAQImage();
   dataIn := VarArrayCreate([0, 100, 0, 100],
     varDouble);
   for i := 0 to 99 do
      for j := 0 to 99 do
         dataIn[i,j] := i+j;

   image1.ArrayToImage(dataIn);
   CWIMAQViewer1.Attach (image1);
   dataOut := image1.ImageToArray(0,0,100,100);
end;
```

Using Events

Use event handler routines in your source code to respond to and process events generated by the different ComponentWorks IMAQ controls. Events are generated by user interaction with an object in response to internal conditions (for example, completed acquisition or an error). You can create a skeleton for an event handler routine using the Object Inspector in the Delphi environment.

To open the Object Inspector, press <F11> or select **Object Inspector** from the **View** menu. In the Object Inspector, select the **Events** tab. This tab, as shown in the following figure, lists all the events for the selected control. To create a skeleton function in your code window, double click on the empty field next to the event name. Delphi generates the event handler routine in the code window using the default name for the event handler.

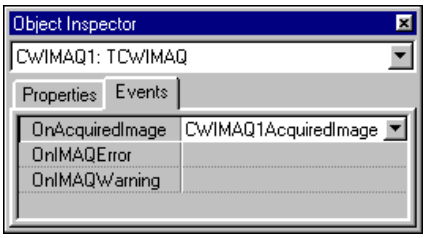


Figure 5-5. Delphi Object Inspector Events Tab

To specify your own event handler name, click in the empty field in the Object Inspector next to the event, and enter the function name. After the event handler function is created, insert the code in the event handler. The following is an example of the event handler function for the `AcquiredImage` event of the IMAQ control.

```
procedure TForm1.CWIMAQ1AcquiredImage(Sender: TObject;
  var ImageIndex: OleVariant);
begin

end;
```

Creating Standalone Objects

Standalone objects are ActiveX objects created at run time, so you do not place them on a form. They are used as parameters to many IMAQ Vision functions. ComponentWorks uses two kinds of standalone objects: Images and Reports. Standalone Image objects are identical to the Image objects found on the CWIMAQ and CWIMAQViewer controls.

Standalone Image objects can store temporary results of image processing functions that you do not want to display. Standalone Report objects are arrays of data used as inputs and outputs to IMAQ Vision functions.

You can create standalone objects in Delphi using `Create` methods on the CWIMAQVision control, such as `CreateCWIMAQImage`, `CreateCWIMAQCaliperReport`, and `CreateCWIMAQShapeReport`.

```
// Create a standalone image.
procedure TForm1.Button1Click(Sender:TObject);
var
   Image:CWIMAQImage;
begin
   Image := CWIMAQVision1.CreateCWIMAQImage;
end;
```

Part II

Using the ComponentWorks IMAQ Vision Controls

This section describes the basic operation of the ComponentWorks IMAQ Vision controls. These chapters contain overviews of the controls, describing their most commonly used properties, methods, and events. The descriptions also include short code segments to illustrate programmatic control and tutorials that lead you through building an application with the controls.

Part II, *Using the ComponentWorks IMAQ Vision Controls*, contains the following chapters.

- Chapter 6, *Using the Viewer and Hardware Controls*, describes how you can use the IMAQ Viewer and Hardware controls to acquire and display images; explains individual controls and their most commonly used properties, methods, and events; and includes a tutorial with step-by-step instructions for using the controls.
- Chapter 7, *Using the Vision Control*, describes how you can use the ComponentWorks IMAQ Vision control and its functions. You can use the Vision control alone or with other controls to perform image analysis, manipulation, and processing. Vision functions include operations such as filtering, morphology, arithmetic, and measurement.
- Chapter 8, *Building Advanced IMAQ Vision Applications*, discusses how you can build applications using more advanced features of ComponentWorks IMAQ Vision, including advanced image acquisition techniques, image processing, and advanced user interface options. It also explains error tracking, error checking, and debugging techniques.

6

Using the Viewer and Hardware Controls

This chapter describes how you can use the IMAQ Viewer and Hardware controls to acquire and display images; explains the individual controls and their most commonly used properties, methods, and events; and includes a tutorial with step-by-step instructions for using the controls.

You can find additional information about the Viewer and Hardware controls and their properties, methods, and events in the ComponentWorks IMAQ online reference, available by selecting **Programs»National Instruments ComponentWorks»IMAQ Vision»ComponentWorks IMAQ Reference** from the Windows **Start** menu.

Image Acquisition Configuration

Before using your National Instruments image acquisition (IMAQ) hardware with the ComponentWorks IMAQ controls, configure your IMAQ device using the IMAQ Configuration Utility. Make sure you follow the directions in the IMAQ Configuration Utility online documentation to properly configure the hardware. Use the configuration utility to test the hardware and perform simple image acquisition operations. Once configured, the image acquisition device is assigned an interface name that you can use to reference it in applications. This is the name you select in the IMAQ property pages.

What Are the Viewer and Hardware Controls?

Use the Viewer and Hardware controls to display and acquire images. The Viewer and Hardware controls are named CWIMAQViewer and CWIMAQ, respectively.

You can set most properties for these controls through property pages as you design your program. To better understand the potential and versatility of these controls, try experimenting with the control properties on the property pages.

In certain cases, you might need to change the value of one or more properties in your program code. Throughout this chapter, examples demonstrate how to change values programmatically.

☞ **Note** ***Although the code and examples in the tutorials use Visual Basic syntax, you can apply the concepts and implement the steps in any programming environment. Remember to adjust all code to your specific programming language.***

Object Hierarchy

The ComponentWorks Viewer and Hardware controls are made up of a hierarchy of simple objects. Understanding the relationship among the objects in a control is the key to properly programming with the control. Dividing a control into individual objects makes it easier to work with because each individual component has fewer parts. Furthermore, several objects in the object hierarchy are reused throughout different controls.

Viewer Control—IMAQ User Interface Control

Use the Viewer control to display and manipulate images. This control displays static or dynamic (live) images and supports graphical region of interest tools.

☞ **Note** ***If you received the IMAQ Hardware control as part of NI-IMAQ and have not purchased ComponentWorks IMAQ Vision, the Viewer control is in evaluation mode.***

Like other objects, the Viewer control is built from a hierarchy of objects, as illustrated in Figure 6-1. The objects in the Viewer control represent the different parts displayed on the physical representation of the viewer, including the Viewer object, Regions collection and Region object, Palette object, and Image object.

- Viewer object—An object that contains the basic properties of the control, such as `BorderWidth` and `ZoomScale`.
- Regions collection and Region object—Objects used to control the selection of regions of interest on the image. Regions are used to construct a mask image that various image processing functions can use.
- Palette object—An object that controls the mapping of bitmap pixel values to colors viewed on screen. Use palettes to view image data in different ways. You can use any of the predefined palettes or define your own.

- Image object—The object that holds the image data the viewer is displaying.

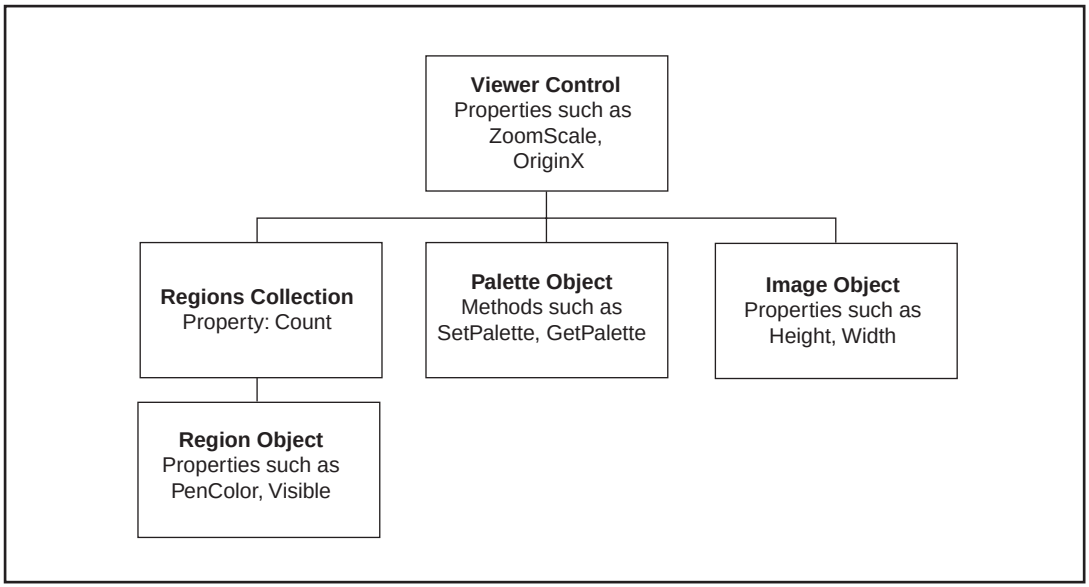


Figure 6-1. Viewer Control Object Hierarchy

Viewer Object

The Viewer object has several simple properties, such as `Name` and `BorderWidth`, that you can set in the property pages during design. The properties `OriginX`, `OriginY`, and `ZoomScale` are other important properties that affect how the Viewer object displays an image.

Several methods in the Viewer object are called directly on the Viewer control: `Attach`, `Detach`, and `Center`.

- `Attach` method—Replaces the Image object with an image that you pass as a parameter to the `Attach` method, causing the Viewer to display the new image and automatically update it as the image changes.
- `Detach` method—Breaks the connection between a Viewer object and an Image object. The image is no longer displayed.
- `Center` method—Centers the Viewer object around a point specified by a parameter to the `Center` method.

Regions Collection

The Regions collection is a standard collection containing Region objects. The collection contains one property, `Count`, which returns the number of Region objects in the collection.

```
NumRegions = CWIMAQViewer1.Regions.Count
```

Use the `Add`, `Remove`, and `RemoveAll` methods to programmatically change the number of regions on the Viewer object. The `Remove` method requires the index of the plot to be removed.

```
CWIMAQViewer1.Regions.Add
CWIMAQViewer1.Regions.Remove 3
```

Use the `Item` method of the Regions collection to access any particular Region object in the collection.

```
Dim Region1 as CWIMAQRegion
Set Region1 = CWIMAQViewer1.Regions.Item(1)
```

Region Object

The Region object represents an individual region on the Viewer object. The Region object contains a number of different properties that determine the display of the region, including `PenColor`, `Shape`, `XData`, and `YData`. You can set these properties programmatically and modify them with the `SetRegion` method.

```
CWIMAQViewer1.Regions.Item(1).PenColor = vbBlue
CWIMAQViewer1.Regions.Item(1).Shape = cwimaqRegionRect
```

Palette Object

Use the Palette object methods and property to specify user-defined or predefined palettes. The `GetPalette` and `SetPalette` methods each take three arrays as arguments to represent the red, green, and blue color plane values either being read or set. The `Type` property sets the Palette object to contain one of the predefined palettes. To set a predefined palette at run time, right click on the Viewer object and select a palette on the Palettes submenu.

Viewer Events

To enable applications to react to user interactions with a Viewer object, the Viewer control generates a number of different events. There are eight key events: `PaletteChanged`, `ToolChanged`, `RegionsChanged`,

`ViewerZoomed`, `ViewerPanned`, `ImageMouseMove`, `ImageMouseDown`, and `ImageMouseUp`.

- `PaletteChanged`, `ToolChanged`, `ViewerZoomed`, and `ViewerPanned`—These events are generated every time the palette, region tool, zoom scale, or origin is changed from the program or through the pop-up menu.
- `RegionsChanged`—This event is generated every time a region is added or removed through the user interface. It returns the new count of the number of Region objects in the Regions collection.
- `ImageMouseMove`, `ImageMouseDown`, and `ImageMouseUp`—These events are similar to the standard mouse events. Rather than returning a point in terms of screen coordinates, these events return a point in terms of the coordinates of the currently displayed image.

IMAQ Control—IMAQ Hardware Interface

Use the IMAQ Hardware control to acquire images from your IMAQ hardware, including the capture of single or multiple images in continuous or single-shot mode. You can configure the IMAQ control for many different modes, including start triggers, skip counts, and frame or field mode. After the properties are set, the application can perform acquisitions using method calls.

The object hierarchy of the IMAQ control separates the functionality of the control into individual objects, as shown in Figure 6-2.

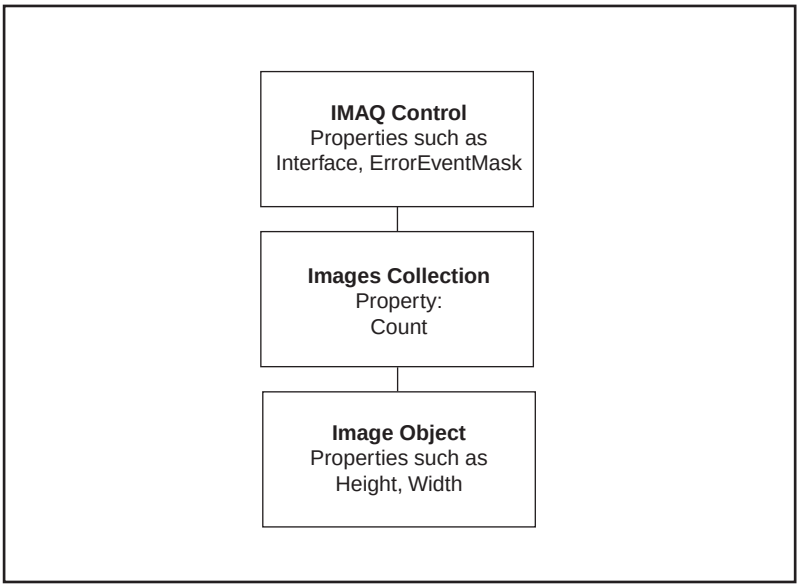


Figure 6-2. IMAQ Control Object Hierarchy

The Images collection and Image objects represent the acquired images. When you acquire multiple images, additional Image objects are added to the Images collection.

IMAQ Object

The IMAQ object has an `Interface` property that selects the hardware device used by the control. You can set this property on the General property page for each control, or you can set it programmatically.

```
CWIMAQ1.Interface = "img0"
```

Additional properties include `StopCondition` and `StartCondition`, which allow you to configure how the acquisition will operate. For example, the acquisition might run continuously or start on a trigger.

```
CWIMAQ1.StartCondition = cwimaqHardwareTrigger
```

Image Object

The Image object is the object that encapsulates image data. The Image object contains properties for manipulating image attributes and provides a mechanism for transferring images between the Hardware object and the Viewer object.

The Hardware control contains an Images collection, which can store multiple Image objects. An Image object contains an image that the Hardware control has acquired. The Image object in the Viewer control contains the data the Viewer object displays.

Notice that an Image object can be shared between an IMAQ Hardware control and a Viewer control, as depicted in Figure 6-3. When a new image is acquired by the IMAQ Hardware object, the Viewer object automatically displays that image.

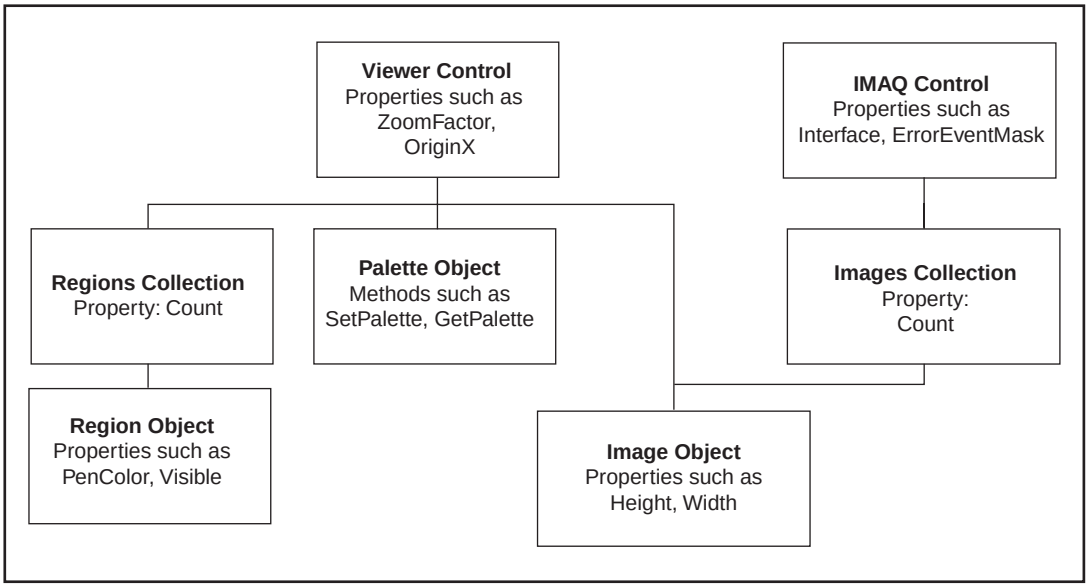


Figure 6-3. Viewer Control and IMAQ Control Can Share an Image Object

To share and display an Image object, use the `Attach` method on the Viewer object, as in the following code.

```
CWIMAQViewer1.Attach CWIMAQ1.Images.Item(1)
```

You can create other Image objects independently for image processing functions.

```
Dim DestImage As CWIMAQImage
Set DestImage = CWIMAQVision1.CreateCWIMAQImage

CWIMAQViewer1.Attach CWIMAQ1.Images.Item(1)
CWIMAQVision1.Threshold CWIMAQ1.Images.Item(1),
   DestImage, 50, 100, True, 255
CWIMAQViewer2.Attach DestImage
```

IMAQ Methods and Events

The IMAQ object has a number of methods for running asynchronous and synchronous image acquisition processes.

Asynchronous Acquisition

The methods to perform an asynchronous acquisition include `Start`, `Stop`, and `Reset`. These methods do not require any parameters. Use the `Start` method to begin the acquisition. Use the `Stop` method only during a continuous acquisition to stop such an operation. Use the `Reset` method to unconfigure the IMAQ control and free resources reserved during configuration.

```
Private Sub StartAcquisition_Click()
   CWIMAQ1.Start
End Sub
```

```
Private Sub StopAcquisition_Click()
   CWIMAQ1.Stop
End Sub
```

```
Private Sub ResetAcquisition_Click()
   CWIMAQ1.Reset
End Sub
```

When a new image is acquired, the IMAQ control generates an `AcquiredImage` event. The image index is returned to the event handler. In the event handler, use the image index to access the most recent image in multiple image acquisitions.

```
Private Sub CWIMAQ1_AcquiredImage(ImageIndex as Variant)
   Dim h as Variant
   CWIMAQVision1.Histogram CWIMAQ1.Images.
     Item(ImageIndex), h
End Sub
```

Synchronous Acquisition

Use the IMAQ control to perform synchronous acquisition with the `AcquireImage` method.

```
Private Sub RunAcquisition_Click()
   Dim h as Variant
   CWIMAQ1.AcquireImage
   CWIMAQVision1.Histogram CWIMAQ1.Images.Item(1), h
End Sub
```

Error Handling

Use the `IMAQError` and `IMAQWarning` events for error handling.

```
Private Sub CWIMAQ1_IMAQError(ByVal StatusCode as Long,
  ByVal ContextID As Long, ByVal ContextDescription
  As String)
   MsgBox "IMAQ Error: " + CStr(StatusCode)
End Sub
```

ExceptionOnError and ErrorEventMask

The CWIMAQ control handles error checking in a number of different ways. By default, an exception is generated when an error occurs and is handled by the programming environment. You can disable exception generation using the `ExceptionOnError` property of the IMAQ control. If exceptions are disabled, each call to a control method returns an error code. If the code is equal to zero, the method completed normally. If the value is non-zero, either a warning or error occurred and the application should handle the condition.

The IMAQ controls can generate error and warning events in response to error conditions. Each event calls a corresponding event handler to process the error information. Use the `ErrorEventMask` property on the IMAQ control to limit the error and warning event generation to specific operations (contexts) of the IMAQ control. By default, the IMAQ control generates an error event only during `cwimaqAcquiring` contexts, indicating that the IMAQ control is asynchronously acquiring images and returning them to the application. Contexts such as `cwimaqStarting` or `cwimaqConfiguring` do not generate error events by default. You can select which contexts generate error events by adding the values of the `CWIMAQErrorContexts` constants and assigning the sum to the `ErrorEventMask` property.

```
CWIMAQ1.ErrorEventMask =
  cwimaqAcquiring + cwimaqStopping  + cwimaqConfiguring
```

For more information about error handling, refer to *Error Handling* in Chapter 8, *Building Advanced IMAQ Vision Applications*.

Tutorial: Using the Viewer and IMAQ Controls

This tutorial shows you how to integrate the Viewer and IMAQ controls to complete a simple acquisition. This tutorial is divided into two parts:

- Synchronous, single-image acquisition and display
- Asynchronous, continuous single-image acquisition and display

Although the code and examples in the tutorial use Visual Basic syntax, you can apply the concepts and implement the steps in any programming environment. Remember to adjust all code to your specific programming language.

Part 1: Synchronous, Single-Image Acquisition and Display

In the first part of this tutorial, you use an IMAQ object to acquire an image and then you display that image in a Viewer object.

Designing the Form

1. Open a new project and form. If you are working in Visual C++, select a dialog-based application and name your project `SimpleIMAQExample`.
2. Load the ComponentWorks IMAQ controls into your programming environment.

3. Place a CWIMAQViewer control on the form. Keep its default name, `CWIMAQViewer1`.

4. Place a CWIMAQ control on the form. You configure its properties in the next section, *Setting the IMAQ Properties*.

5. Place a CommandButton control on the form. Change its Caption property to `Acquire`.

Your form should look similar to Figure 6-4.

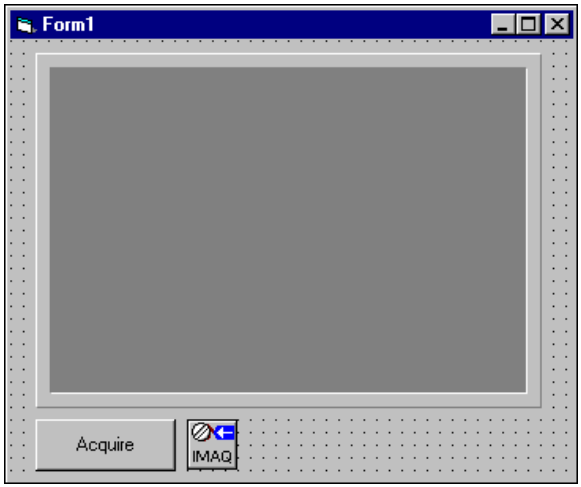


Figure 6-4. Simple IMAQ Example Form

Setting the IMAQ Properties

You normally configure the default property values of the different controls before you develop your program code. When using the IMAQ control, you will set most properties, if not all, during design and will not change them during program execution. Use this program to start and stop the acquisition process only. If necessary, you can edit the properties of the IMAQ control at run time.

1. To open the custom property pages for the IMAQ control placed on the form, right click on the control and select **Properties**.
2. On the General page, select your image acquisition device from the **Interface** combobox.
3. Close the IMAQ property pages.

Developing the Code

Develop the code so that data is acquired and displayed when you press the **Acquire** button.

1. Generate the event handler routine for the `Click` event of the **Acquire** button.
2. In the event handler, attach CWIMAQViewer1 to the first image object of the CWIMAQ control and acquire the image using the CWIMAQ control. The viewer automatically updates and displays the changed image.

```
Private Sub Acquire_Click()
    ' Attach the image to the viewer.
    CWIMAQViewer1.Attach CWIMAQ1.Images.Item(1)
    ' Acquire the image.
    CWIMAQ1.AcquireImage
End Sub
```

Testing Your Program

Run and test your program. Click on the **Acquire** button to perform the acquisition. Although the image displayed in your viewer depends on the object your camera is viewing, your application should look similar to the following figure.

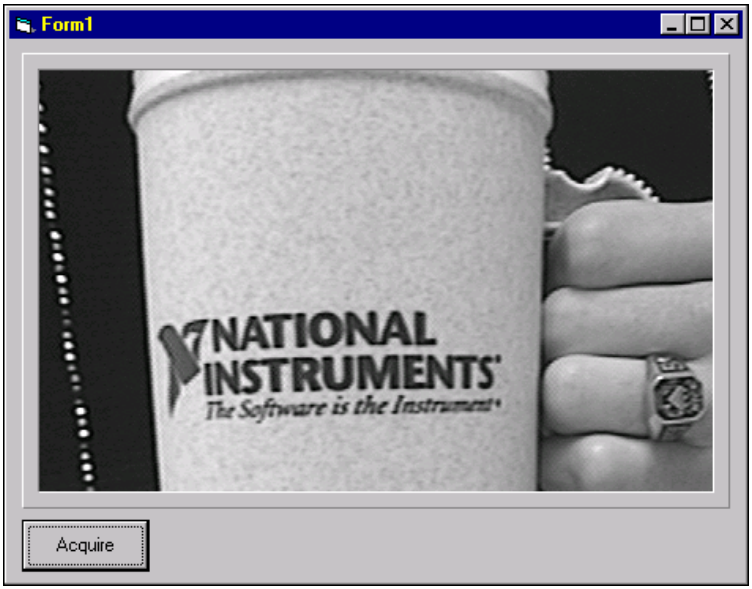


Figure 6-5. Testing the Simple IMAQ Example

Part 2: Asynchronous, Continuous Single-Image Acquisition and Display

In the second part of this tutorial, you use an IMAQ object to asynchronously and continuously acquire an image and then display the continuous acquisition with a Viewer object.

Designing the Form

Continue using the form you designed in the first part of this tutorial, `SimpleIMAQExample`.

Setting the IMAQ Properties

1. To open the custom property pages for the IMAQ control placed on the form, right click on the control and select **Properties**.
2. On the Acquisition page, select **Continuous** for the **Stop** Condition so the acquisition runs continuously.
3. Close the IMAQ property pages.

Developing the Code

Develop the code so that data is acquired and displayed when you press the **Acquire** button.

1. Open the event handler from the first part of this example to edit the code.
2. Change `CWIMAQ1.AcquireImage` to `CWIMAQ1.Start` to run the acquisition asynchronously.

```
Private Sub Acquire_Click()
    ' Attach the image to the viewer.
    CWIMAQViewer1.Attach CWIMAQ1.Images.Item(1)
    ' Start the acquisition.
    CWIMAQ1.Start
End Sub
```

Testing Your Program

Run and test your program. Click on the **Acquire** button to perform the acquisition and continuously capture images from the hardware.

7

Using the Vision Control

This chapter describes how you can use the ComponentWorks IMAQ Vision control and its functions. You can use the Vision control alone or with other controls to perform image analysis, manipulation, and processing. Vision functions include operations such as filtering, morphology, arithmetic, and measurement.

☞ **Note** ***If you received the IMAQ Hardware control as part of NI-IMAQ and have not purchased ComponentWorks IMAQ Vision, the Vision control is in evaluation mode.***

What is the Vision Control?

The Vision control includes different image processing functions. Each function is a method of the Vision control. You can pass parameters to a function as you would with any other function. Most functions take Image objects as inputs, outputs, or both.

Operations can be performed in-place when the same Image object is used as both an input and an output. The following line of code adds `image1` to `image2` and stores the result in `image1`.

```
CWIMAQVision1.Add image1, image2, image1
```

The Vision control handles error checking in two different ways. By default, an exception is generated when an error occurs in and is handled by the programming environment. You can disable exception generation using the `ExceptionOnError` property of the Vision control. If exceptions are disabled, each call to a method returns an error code. If the code is equal to zero, the method completed normally. If the value is nonzero, an error occurred and the application should handle the condition.

Vision Functions

Table 7-1 lists all the IMAQ Vision functions, grouped by category.

Table 7-1. IMAQ Vision Functions

Category	Function
Files	GetFileInfo
	ReadFile
	WriteFile
Tools	BorderOperation
	ClipboardToImage
	ConvertByLookup
	Copy
	DrawLine
	DrawOval
	DrawRect
	DrawText
	Expand
	Extract
	GetCalibration
	GetLine
	GetPixelValue
	GetRowColumn
	ImageToClipboard
	ImageToImage
	MagicWand
	Resample
	SetCalibration
	SetLine

Table 7-1. IMAQ Vision Functions (Continued)

Category	Function
Tools (*continued*)	SetPixelValue
	SetRowColumn
	Shift16To8
Arithmetic Operators	Add
	Divide
	Modulo
	MulDiv
	Multiply
	Subtract
Logic Operators	And
	Compare
	LogDiff
	Mask
	Or
	Xor
Processing	AutoBThreshold
	AutoMThreshold
	Equalize
	Label
	MathLookup
	MultiThreshold
	Threshold
	UserLookup
Filters	Convolute
	Correlate
	GrayEdge

Table 7-1. IMAQ Vision Functions (Continued)

Category	Function
Filters (*continued*)	LowPass
	NthOrder
Morphology	Circles
	Convex
	Danielsson
	Distance
	FillHole
	GrayMorphology
	Morphology
	RejectBorder
	RemoveParticle
	Segmentation
	Separation
	Skeleton
Analysis	BasicParticle
	Centroid
	Histogram
	LineProfile
	Particle
	ParticleCoefficients
	ParticleDiscrimination
	Quantify
Geometry	Rotate
	Shift
	Symmetry
	View3D

Table 7-1. IMAQ Vision Functions (Continued)

Category	Function
Complex	CxAdd
	CxAttenuate
	CxConjugate
	CxDivide
	CxFlipFrequency
	CxMultiply
	CxSubtract
	CxTruncate
	ExtractComplexPlane
	FFT
	InverseFFT
	ReplaceComplexPlane
Color	ColorEqualize
	ColorHistogram
	ColorThreshold
	ColorUserLookup
	ColorValueConversion
	ColorValueToInteger
	ExtractColorPlanes
	GetColorLine
	GetColorPixelValue
	IntegerToColorValue
	ReplaceColorPlane
	SetColorLine
	SetColorPixelValue

Table 7-1. IMAQ Vision Functions (Continued)

Category	Function
Object Creation	CreateCWIMAQBasicParticleReport
	CreateCWIMAQCaliperReport
	CreateCWIMAQCirclesReport
	CreateCWIMAQDiscriminationData
	CreateCWIMAQEdgeReport
	CreateCWIMAQFullParticleReport
	CreateCWIMAQHistogramReport
	CreateCWIMAQImage
	CreateCWIMAQKernal
	CreateCWIMAQProfileReport
	CreateCWIMAQQuantifyReport
	CreateCWIMAQShapeReport
	CreateCWIMAQStructuringElement
	CreateCWIMAQThresholdData
Caliper	Caliper
	FindEdges
	GetAngles
	Interpolate1D
	LineGauge
	PointDistances
Search	ShapeMatch
Alignment	CoordinateReference

Tutorial: Using Simple Image Processing Functions

This tutorial shows you how to use Vision functions and integrate the Vision control with the IMAQ and Viewer controls. This tutorial is divided into three parts:

- Reading an image from a file and thresholding
- Particle analysis
- Acquisition and image processing

Although the code and examples in the tutorial use Visual Basic syntax, you can apply the concepts and implement the steps in any programming environment. Remember to adjust all code to your specific programming language.

Part 1: Reading an Image From a File and Thresholding

In the first part of this tutorial, you use a Vision object to read an image from a file and then display the thresholded image in a Viewer object.

Designing the Form

1. Open a new project and form. If you are working in Visual C++, select a dialog-based application and name your project `IMAQFileExample`.
2. Load the ComponentWorks IMAQ controls into your programming environment.

3. Place a CWIMAQVision control on the form. Keep its default name, `CWIMAQVision1`.

4. Place a CWIMAQViewer control on the form. Keep its default name, `CWIMAQViewer1`.

5. Place a CommandButton control on the form. Change its Caption property to `Read`.

Your form should look similar to Figure 7-1.

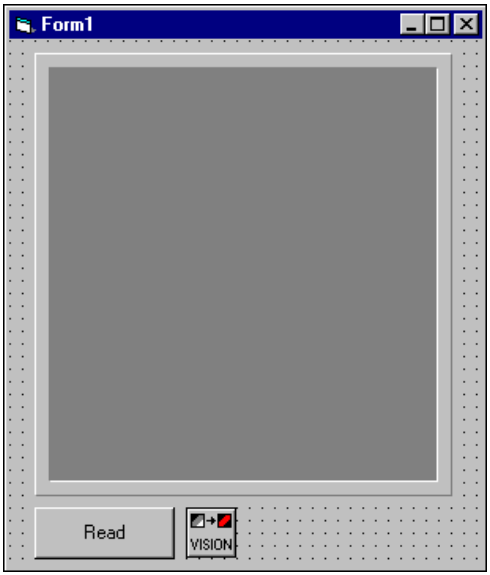


Figure 7-1. IMAQ File Example

Developing the Code

Develop the code so that an image is read, thresholded, and displayed when you press the **Read** button.

1. Double click on the **Read** button to define an event handler routine to be called when the **Read** button is pressed.
2. In the event handler, declare a CWIMAQImage object. This is the image into which the file data will be loaded. Containers such as Visual Basic and Borland Delphi automatically create and allocate objects.

Note ***Environments such as Visual C++ require you to use the*** `CreateCWIMAQImage` ***method to create a new CWIMAQImage object.***

3. Use the `ReadFile` method to read the contents of the `iron.bmp` file. The image file is located in the `ComponentWorks\Images` directory. You might need to modify the path to match the actual location of the `iron.bmp` file on your hard drive.
4. Threshold the image into `CWIMAQViewer1.Image`. Notice how the thresholded image is automatically displayed in the Viewer when the image contents change.

Your `Read_Click()` routine should be similar to the following code. You can experiment with different threshold values and other files.

```
Private Sub Read_Click()
    Dim Image1 as New CWIMAQImage
    Dim ColorTable
    ' Read the file.
    CWIMAQVison1.ReadFile Image1, "c:\iron.bmp",
      ColorTable
    ' Threshold the image into CWIMAQViewer.
    CWIMAQVision1.Threshold Image1, CWIMAQViewer1.Image,
      128, 255, TRUE, 255
End Sub
```

Testing Your Program

Run and test your program. Click on the **Read** button to read and process the file. Your application should look similar to Figure 7-2.

Figure 7-2. Testing the IMAQ File Example

Part 2: Particle Analysis

In the second part of this tutorial, you modify the first procedure to perform particle analysis. This part of the tutorial illustrates a more complex image processing function and demonstrates the use of Report objects.

Designing the Form

1. Continue using the form you designed in Part 1, `IMAQFileExample`.
2. Place a TextBox control on the form. Name it `TotalArea`.

Developing the Code

Develop the code so that the thresholded image is processed and the total area of the particles is displayed in the text box.

1. Double click on the **Read** button to define an event handler routine to be called when the **Read** button is pressed.
2. In the event handler, declare a CWIMAQBasicParticleReport object. This object will contain the result of the `BasicParticle` processing function.

   Containers such as Visual Basic and Borland Delphi automatically create and allocate objects.

Note ***To create a new object in environments such as Visual C++, you must use the*** `CreateCWIMAQBasicParticleReport` ***method.***

3. Use the `BasicParticle` function to perform the particle analysis. Notice how `CWIMAQViewer1.Image` is being used as a source image for an image processing function.
4. Calculate the total area of all particles by looping through the `CWIMAQBasicParticleReport`.

Your `Read_Click()` routine should be similar to the following code.

```
Private Sub Read_Click()
    Dim Image1 as New CWIMAQImage
    Dim Report1 as New CWIMAQBasicParticleReport
    Dim ColorTable
    Dim i, area as Integer
    area = 0
    ' Read the file.
    CWIMAQVison1.ReadFile Image1, "c:\iron.bmp",
      ColorTable
    ' Threshold the image into CWIMAQViewer1.
```

```
    CWIMAQVision1.Threshold Image1, CWIMAQViewer1.Image,
      128, 255, TRUE, 255
    ' Perform particle analysis.
    CWIMAQVision1.BasicParticle CWIMAQViewer1.Image,
      Report1
    ' Calculate the total area of all particles.
    For i=1 to Report1.Count
       area = area + Report1.Area(i)
    Next i
    TotalArea.Text = area
End Sub
```

Testing Your Program

Run and test your program. Click on the **Read** button to read and process the file. Your application should look similar to Figure 7-3.

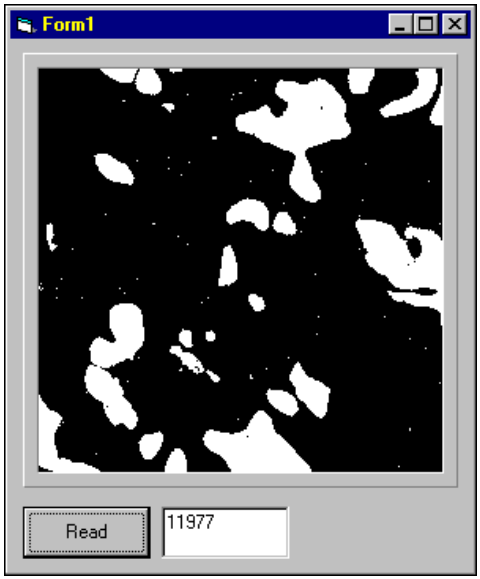


Figure 7-3. Testing the IMAQ File Example After Adding Particle Analysis

Part 3: Acquisition and Image Processing

In the third part of this tutorial, you acquire an image using the IMAQ control and perform a threshold operation on it. This procedure is similar to the first procedure, except it acquires an image from hardware rather than reading an image from a file.

Designing the Form

1. Open a new project and form. If you are working in Visual C++, select a dialog-based application and name your project `IMAQAcqThreshExample`.
2. Load the ComponentWorks IMAQ controls into your programming environment.

3. Place a CWIMAQVision control on the form. Keep its default name, `CWIMAQVision1`.

4. Place two CWIMAQViewer controls on the form. Keep their default names, `CWIMAQViewer1` and `CWIMAQViewer2`.

5. Place a CWIMAQ control on the form. Keep its default name, `CWIMAQ1`.

6. Place a CommandButton control on the form. Change its Caption property to `Read`.

Your form should look similar to Figure 7-4.

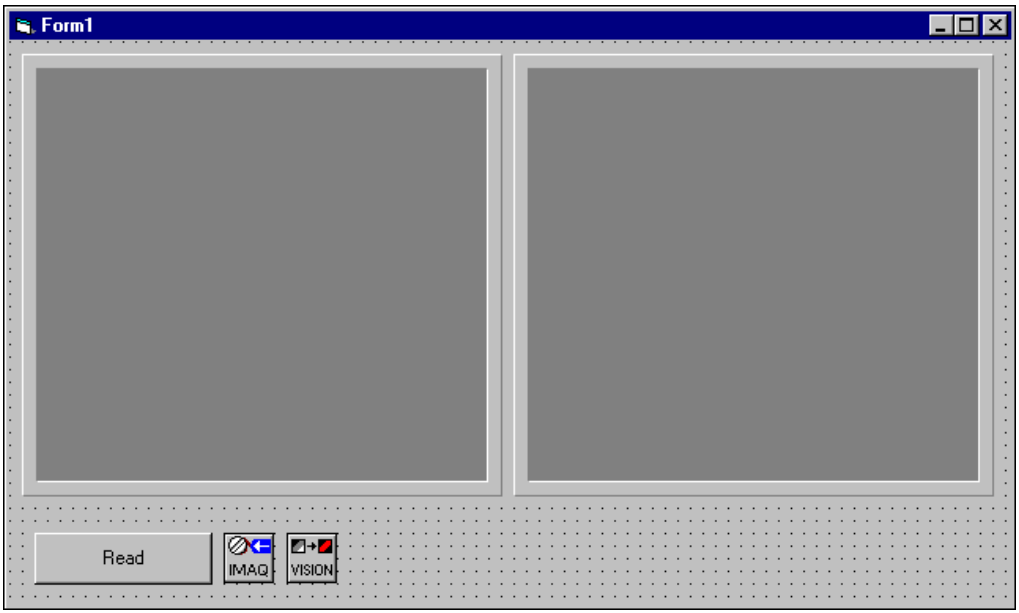


Figure 7-4. Image Acquisition Threshold Example

Setting the IMAQ Properties

1. To open the custom property pages for the IMAQ control placed on the form, right click on the control and select **Properties**.
2. On the General page, select your image acquisition device from the **Interface** combobox.
3. Close the IMAQ property pages.

Developing the Code

Develop the code so that the CWIMAQ control acquires an image and the image is thresholded and displayed in a CWIMAQViewer object.

1. Double click on the **Read** button to define an event handler routine to be called when the **Read** button is pressed.
2. Acquire an image using the `AcquireImage` method on the CWIMAQ control.
3. Threshold the image using the CWIMAQVision control and place the result in the image that belongs to the CWIMAQViewer control.

Your `Read_Click()` routine should be similar to the following code.

```
Private Sub Read_Click()
    ' Attach the image to the viewer.
    CWIMAQViewer1.Attach CWIMAQ1.Images(1)
    ' Acquire the image.
    CWIMAQ1.AcquireImage
    ' Threshold the image.
    CWIMAQVision1.Threshold CWIMAQViewer1.Image,
      CWIMAQViewer2.Image, 128, 255, TRUE, 255
End Sub
```

Testing Your Program

Run and test your program. Click on the **Read** button to read and process the file. Your application should look similar to Figure 7-5.

Figure 7-5. Testing the Image Acquisition Threshold Example

8

Building Advanced IMAQ Vision Applications

This chapter discusses how you can build applications using more advanced features of ComponentWorks IMAQ Vision, including advanced image acquisition techniques, image processing, and advanced user interface options. It also explains error tracking, error checking, and debugging techniques.

The applications presented in this chapter illustrate advanced IMAQ Vision functions, such as shape matching and edge detection, and use advanced user interface control features, such as viewer regions of interest.

You can customize these examples to implement the advanced features in your own applications. They are located in the `\ComponentWorks\Tutorials-IMAQ` directory.

Finding Features on a Printed Circuit Board

The Feature Find application uses the ComponentWorks IMAQ Vision user interface and image processing controls to build an application that finds arbitrary features on a printed circuit board. Load the sample program from `\ComponentWorks\Tutorials-IMAQ` into your development environment to follow the discussion. Figure 8-1 shows the application at run time.

Depending on the state of the region selection radio buttons, the main viewer operates in two different modes. The first mode, **Template Image**, enables selection of a template image that contains the feature for which to be searched. The second mode, **Search Regions**, selects one or more rectangular regions where the application will search for the feature. The **Tolerance** input controls the precision level of the search, from locating an exact mach to finding all similar objects. **Tolerance** ranges in value from 0 to 1, where 1 specifies that only exact matches should be found. As **Tolerance** decreases, the matching criteria is reduced and more similar matches are found.

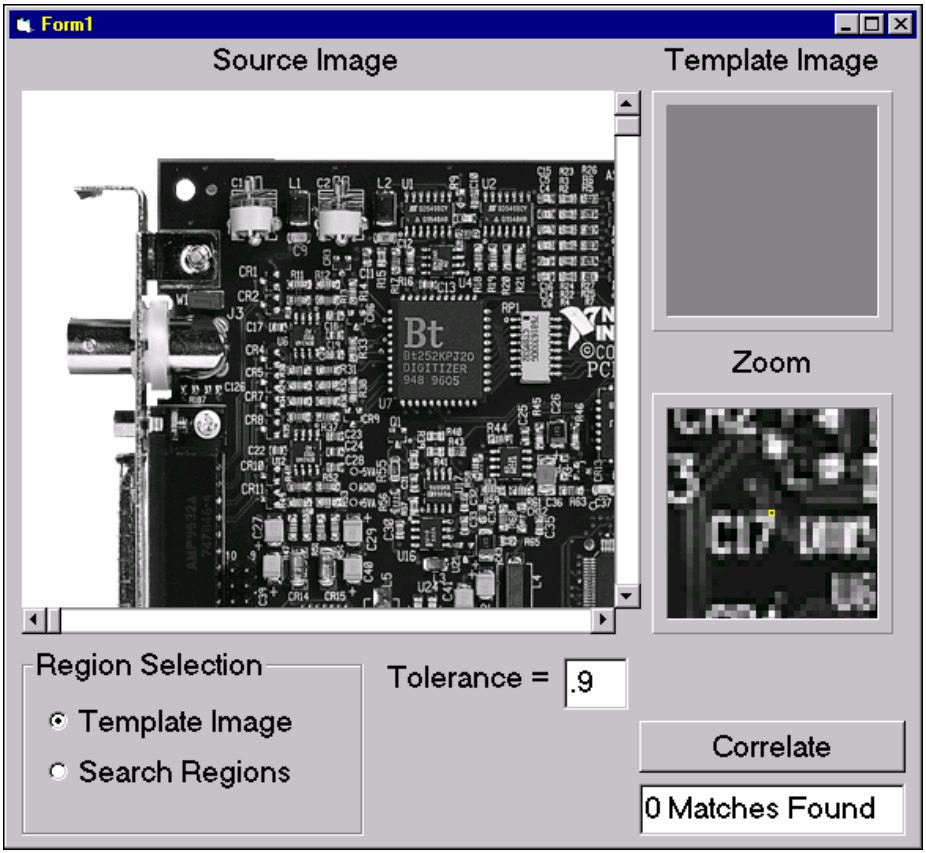


Figure 8-1. Feature Find Application

Manipulating Regions of Interest through the User Interface

Each of the radio button states allows the user to draw rectangular regions of interest on the main viewer control. As regions are manipulated through the user interface, the `RegionsChanged` event is generated. This event returns the new number of regions in the collection.

```
Private Sub CWIMAQViewer1_RegionsChanged(Count As
    Variant)
    Dim top as Long, left as Long, width as Long,
      height as Long
    CWIMAQViewer1.Regions(Count).GetBoundingRect left,
      top, width, height

    If (Option1.Value = True) Then
       'Selecting the template
```

```
        CWIMAQVision1.Extract board,
          CWIMAQViewer2.Image, left, top, width, height
        CWIMAQViewer1.Regions.Remove Count
    Else
        'Selecting Search Regions
        CWIMAQViewer1.Regions(Count).PenColor = vbGreen
        CWIMAQViewer1.Regions(Count).Active = False
    End If
```

In Template Image mode, you can select the template image by clicking and dragging a rectangle. You can extract the template image with the `Extract` method on the CWIMAQVision control. Finally, you can remove the region that was just added with the `Remove` method on the Regions collection. To ensure that only one template image is selected at a time, remove previous template images.

With the Search Regions mode, you can select one or more regions in which to look for the template image. To select multiple regions, press and hold the <Shift> key as you draw a new rectangular selection. By default, the viewer deletes all existing regions when a new region is specified through the user interface.

AutoDelete, Active, and Visible Properties

The `AutoDelete`, `Active`, and `Visible` properties specify the behavior of the regions of interest. `AutoDelete` is a property of the Viewer control, while `Active` and `Visible` are properties of the Region object.

When a new region is specified through the user interface, the viewer removes all existing regions. To override this default for a single instance, press the <Shift> key as you begin a new region. To completely disable this feature, set the `AutoDelete` property of the Viewer control to `False`.

```
Private Sub Option1_Click()
    CWIMAQViewer1.AutoDelete = False
End Sub

Private Sub Option2_Click()
    CWIMAQViewer1.AutoDelete = True
End Sub
```

In this application, `AutoDelete` is set to `False` when specifying the template image so that any existing search regions are not deleted. When selecting search regions, `AutoDelete` is enabled.

The `Active` property determines if the Region object will be used on certain methods on the Regions Collection. It is `True` by default.

The `RegionsToMask` function produces a mask image that is composed of those individual Region objects that have their `Active` properties set to `True`. In this example, the `Active` properties of all search regions are set to `False` as they are added. These regions are not part of the mask image that is generated to highlight located features.

```
CWIMAQViewer1.Regions.RegionsToMask mask
```

The `Visible` property determines if an individual region is drawn on the Viewer control, which is useful for defining temporary regions or intermediate results. By default, the `Visible` property is `True`.

Finding Features and Displaying Results

When you press the **Correlate** button, code is executed. This code finds the selected template image in the selected regions using the following steps.

First, the bounding rectangle for the region is computed and passed to the `Correlate` function, along with the source image, template image, and destination image. The `Correlate` function searches the source image for features that match the template images and assigns increasing pixel values to areas of greatest correlation in the destination image. Because only the area specified by the bounding rectangle is considered, the `Correlate` function is faster than searching the entire source image.

```
CWIMAQViewer1.Regions(i).GetBoundingRect left, top,
  width, height
```

```
CWIMAQVision1.Correlate board, CWIMAQViewer2.Image,
  dest, left, top, width, height
```

Next, the destination image is thresholded to extract the areas with the closest matches. The level of thresholding is based on the `Tolerance` parameter. The result of the threshold operation is a binary image with particles at areas of high correlation.

```
CWIMAQVision1.Threshold dest, dest, Text1.Text * 255, 255
```

The `Particle` function generates a report of all the particles in the binary image. Using this report, the `ParticleCoefficents` function calculates the x and y coordinates of the center of mass for each particle.

```
CWIMAQVision1.Particle dest, rep
```

```
CWIMAQVision1.ParticleCoefficients dest,
  Array(cwimaqParticleCenterMassX,
  cwimaqParticleCenterMassY), rep, coeffs
```

Once the x and y coordinates of the center of mass for each particle are known, a highlighted region is drawn around each area. The `DrawRect` function draws the outline, and the `UserLookup` function highlights the area. Because `UserLookup` requires a mask image, a temporary region is programmatically created from which the mask image is generated.

```
CWIMAQVision1.DrawRect CWIMAQViewer1.Image,
CWIMAQViewer1.Image, x, y, CWIMAQViewer2.Image.width,
CWIMAQViewer2.Image.height, cwimaqDrawModeFrame, 255

Set r = CWIMAQViewer1.Regions.Add

r.Visible = False

r.SetRegion cwimaqRegionRect, Array(x, x +

CWIMAQViewer2.Image.width - 1), Array(y, y +
  CWIMAQViewer2.Image.height - 1)

CWIMAQViewer1.Regions.RegionsToMask mask

CWIMAQVision1.UserLookup CWIMAQViewer1.Image,
  CWIMAQViewer1.Image, table, mask

CWIMAQViewer1.Regions.Remove
  CWIMAQViewer1.Regions.Count
```

Floppy Disk Inspection

The Floppy Disk Inspection application uses the ComponentWorks IMAQ Vision user interface and image processing controls to build an application that examines images of diskettes and determines if a feature has been printed correctly. Load the sample program from `\ComponentWorks \Tutorials-IMAQ` into your development environment to follow the discussion. Figure 8-2 shows the application at run time.

Use the Test Configuration window to setup and configure the test. The Unit Under Test window displays the ongoing testing and running statistics, such as the number of units tested and the current test rate.

The test is configured by sequentially clicking the four buttons in the Test Configuration window.

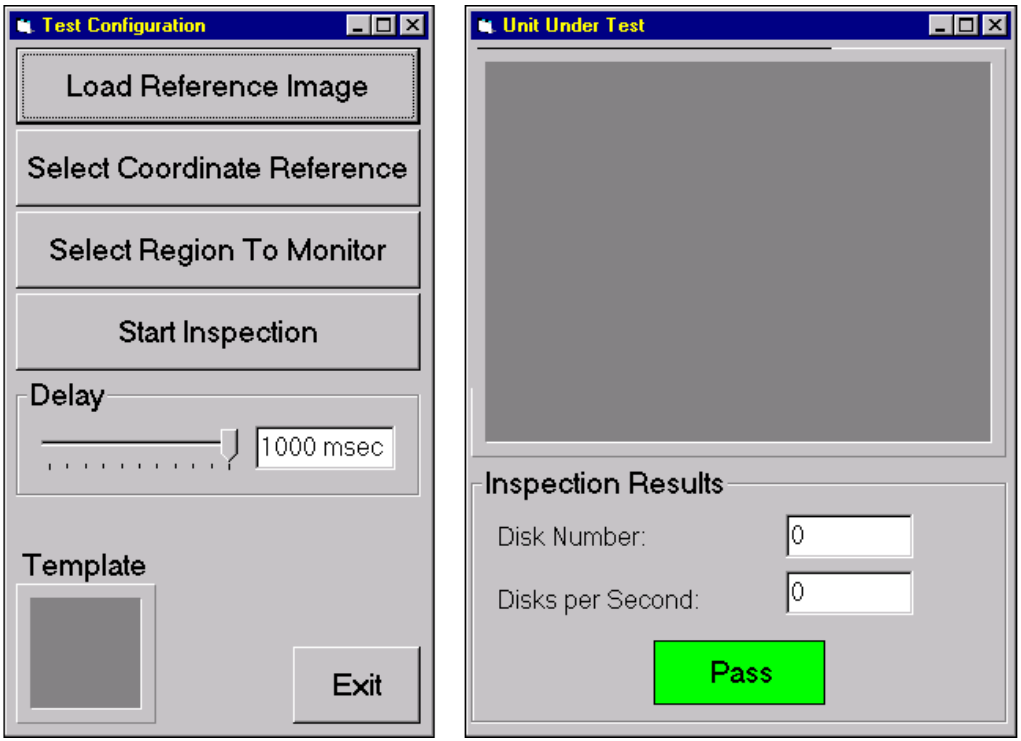


Figure 8-2. Floppy Disk Inspection

First, press **Load Reference Image**. An image of a diskette is loaded into the viewer window. You can use this image to select a coordinate reference system and a search region.

Next, press **Select Coordinate Reference**. Three line regions of interest are programmatically added, and an edge detection is performed along each region. These three points define a coordinate reference system that defines the initial position of the search region.

Now, press **Select Region To Monitor**. A rectangular region is programmatically added around the HD text, which is the printed feature in which this test is interested.

Finally, press **Start Inspection**. You can control the rate of inspection with the **Delay** slider control. As new disk images are loaded, edge detections are performed along the same three line regions of interest. These three points define a new coordinate reference system for the rotated and translated disk image. Using the new and old reference systems, the search

region is rotated and translated to match the motion of the disk. The HD on the new disk is inspected for defects.

Manipulating Regions of Interest Programmatically

The Disk Inspection application uses several regions of interest to perform functions such as edge detection and search location specification. In this application, regions are specified programmatically rather than through the user interface as in the Feature Find application.

When the **Select Coordinate Reference** button is pressed, three line regions of interest are defined.

```
Set Line = diskviewer.Regions.Add
Line.PenColor = vbYellow
Line.SetRegion cwimaqRegionLine, Array(40, 90),
  Array(130, 130)
MakeCoordinateReference xdata(0), ydata(0), x0, y0
Line.Active = False
```

The `SetRegion` method programmatically defines regions of interest. This method takes a shape parameter, an array of X coordinates, and an array of Y coordinates as inputs. This line is defined as starting at point 40,130 and ending at 90,130, defining a horizontal line 50 pixels long.

Edge Detection and Shape Matching

Edge detection is performed using the `FindEdges` function.

```
Dim EdgeReport As New CWIMAQEdgeReport
Dim i

CWIMAQVision1.FindEdges diskviewer.Image,
  XProfilePoints, YProfilePoints, 40, 4, 2,
  cwimaqInterpolateBiLinear, cwimaqSubPixelNone,
  EdgeReport, x, y

x = x(0)

y = y(0)

CWIMAQVision1.DrawOval diskviewer.Image,
  diskviewer.Image, x - 4, y - 4, 8, 8,
  cwimaqDrawModePaint, 255
```

The `FindEdges` function produces an array of x and y coordinates. The pairs of x and y values specify locations of edges that have been found. Because the regions in this example were set up and tested to produce only

one edge, only the first element of each array is used. The `DrawOval` method draws a circle with a diameter of 8 pixels centered over the edge coordinate.

Shape matching is performed using the `ShapeMatch` function.

```
Set r = diskviewer.Regions(4)
r.Visible = False
diskviewer.Regions(5).Visible = False
diskviewer.Regions(5).SetRegion r.Shape, r.xdata,
  r.ydata, r.External
diskviewer.Regions.TransformRegions originx, originy,
  degrees, newx, newy, newdegrees
diskviewer.Regions(5).Visible = True

diskviewer.Regions(5).GetBoundingRect, left, top,
  width, height
CWIMAQVision1.Extract diskviewer.Image, search, left,
  top, width, height
CWIMAQVision1.Threshold search, search, 150, 255
CWIMAQVision1.ShapeMatch search, template, search, rep,
  , 0.05
```

In the above code fragment, the original search region is copied into a new Region object. This new region is translated and rotated to the new coordinate system using the `TransformRegions` method. The bounding rectangle for the new region is computed, and the image contained in the rectangle is extracted to produce a small source image for shape match to operate on, reducing processing time.

The source image is thresholded and passed into the `ShapeMatch` function, along with the template image. `ShapeMatch` operates on binary images, and it is insensitive to rotation.

Report Objects

Many IMAQ functions, such as `FindEdges` and `ShapeMatch`, require Report objects. There are several different types of Report objects, and their use depends on which IMAQ Vision function is invoked. You can consider all reports as arrays of individual items. Each item contains data that is read or written by IMAQ Vision functions. Reports are 1-based (the first index is 1, not 0).

For example, the CWIMAQEdgeReport object has several properties: `Contrast`, `Count`, `Polarity`, `Position`, and `Score`. The `Count`

property is common to all Report objects. `Count` is always readable and can be writable if the user can resize the report. Because the other properties are specific to edge detection functions, they are set by the `FindEdges` function. You can read the values of individual properties as shown in the following code.

```
Dim EdgeReport As New CWIMAQEdgeReport

CWIMAQVision1.FindEdges diskviewer.Image
   XProfilePoints, YProfilePoints, 40, 4, 2,
   cwimaqInterpolateBiLinear, cwimaqSubPixelNone,
   EdgeReport, x, y

' Get the polarity of the first edge.
Text1.Text = EdgeReport.Polarity(1)
```

You can create reports in Visual Basic using the `New` operator. To create reports in environments such as Visual C++ or Delphi, use the `Create` functions on the CWIMAQVision object, as described in the *Creating Standalone Objects* sections of Chapter 4, *Building ComponentWorks IMAQ Vision Applications with Visual C++*, and Chapter 5, *Building ComponentWorks IMAQ Vision Applications with Delphi*. You can find additional information about each report type in the online reference.

Adding Testing and Debugging to your Application

Although they vary depending on the programming environment, debugging tools normally include features such as breakpoints, step-run modes, and watch windows.

Error Checking

ComponentWorks controls can report error information to you and to the application in a number of different ways:

- Return an error code from a function or method call
- Generate an error or warning event
- Throw an exception handled by your programming environment

The type of error reporting depends on the type of application and the preference of the programmer.

By default, all the ComponentWorks IMAQ Vision controls generate exceptions when errors occur, rather than returning error codes from the methods. However, the CWIMAQ and CWIMAQVision controls have a property, `ExceptionOnError`, that you can set to `False` if you want

methods to return error codes instead of generating exceptions. Error events are generated by the CWIMAQ control if an error occurs during specific contexts of an acquisition process. The contexts for which error events are generated are set in the `ErrorEventMask` property of the IMAQ control.

Exceptions

Exceptions are error messages returned directly to your programming environment. Usually, exceptions are processed by displaying a default error message. The error message allows you to end your application or to enter debug mode and perform certain debugging functions. Part of the exception returned is an error number and error description, displayed as part of the error message. For example the CWIMAQ control may return the following exception to Visual Basic:

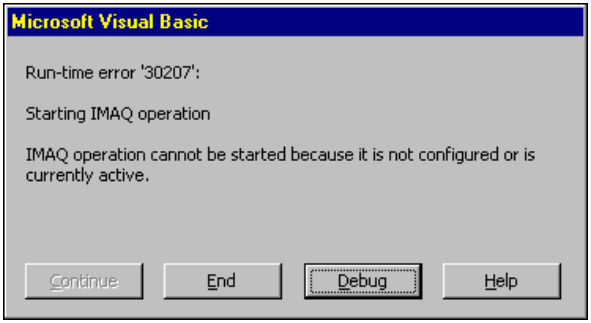


Figure 8-3. Visual Basic Error Message

Depending on your programming environment, you might be able to insert code that can catch exceptions being sent to your application and handle them in another manner. In Visual Basic, you can do this by using the `On Error` statement.

- `On Error Resume Next` disables automatically generated error messages. The program continues running at the next line. To handle an error in this mode, you should check and process the information in the `Err` object in your code.

```
Private Sub Acquire_Click()
On Error Resume Next
    CWIMAQ1.AcquireImage
    If Err.Number <> 0 Then MsgBox "Acquire: " +
     CStr(Err.Number)
End Sub
```

- `On Error GoTo` disables automatically generated error messages and causes program execution to continue at a specified location in the subroutine. You can define one error handler in your subroutine.

```
Private Sub Acquire_Click()
On Error GoTo ErrorHandler
    CWAI1.AcquireImage
Exit Sub
ErrorHandler:
    MsgBox "IMAQ Error: " + CStr(Err.Number)
    Resume Next
End Sub
```

If you are not using Visual Basic, consult the documentation for your programming environment for information about handling exceptions.

Return Codes

If the `ExceptionOnError` property is set to `False`, the CWIMAQ and CWIMAQVision control methods return a status code to indicate whether an operation completed successfully. If the return value is something other than zero, it indicates a warning or error. A positive return value indicates a warning, signifying that a problem occurred in the operation, but that you should be able to continue with your application. A negative value indicates an error—a critical problem that has occurred in the operation—and that all other functions or methods dependent on the failed operation also will fail.

To retrieve the return code from a method call, assign the value of the function or method to a long integer variable and check the value of the variable after calling the function or method. For example, the following code checks the return code of the CWIMAQ control `Start` method.

```
lerr = CWIMAQ1.Start
```

```
If lerr <> 0 Then MsgBox "Error at IMAQ Start: " +
CStr(lerr)
```

In Visual Basic, you can use the **MsgBox** popup window to display error information. Normally, you can write one error handler for your application instead of duplicating it for every call to a function or method. For example, the following code creates a `LogError` subroutine to use with the `Start` method and later functions or methods.

```
Private Sub LogError(code As Long)
    If code <> 0 Then
        MsgBox "IMAQ Error: " + CStr(code)
    End If
End Sub
```

To use the `LogError` subroutine, call `LogError` before every function or method call. The return code is passed to `LogError` and processed.

```
LogError CWIMAQ1.Start
```

Error and Warning Events

The IMAQ control also includes its own error and warning events – IMAQError and IMAQWarning. Although you normally use return codes for error checking of the methods used on the IMAQ control, you cannot use return codes for error checking in asynchronous operations such as a continuous acquisition because no methods are called after the first `Start` method. In this case, the CWIMAQ control generates its own error event if an error or warning occurs during an ongoing acquisition. You can develop an event handler to process these error and warning events. The following code shows the skeleton event functions for the CWIMAQ control.

```
CWIMAQ1_IMAQError (ByVal StatusCode As Long, ByVal
  ContextID As Long, ByVal ContextDescription As String)
CWIMAQ1_IMAQWarning (ByVal StatusCode As Long, ByVal
  ContextID As Long, ByVal ContextDescription As String)
```

The `StatusCode` variable that is passed to the event handler contains the value of the error or warning condition. The `ContextID` contains a value describing the operation where the error or warning occurred, and the `ContextDescription` contains a string describing the operation where the error or warning occurred.

The following code shows an example of how to use the `CWIMAQ_IMAQError` event in a Visual Basic application.

```
Private Sub CWIMAQ1_IMAQError(ByVal StatusCode As Long,
  ByVal ContextID As Long, ByVal ContextDescription As
  String)
   MsgBox ContextDescription + ": CStr(StatusCode)
End Sub
```

This code produces the following error message box.

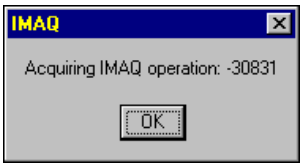


Figure 8-4. Error Message Box

By default, only asynchronous operations call error and warning events. You can set the `ErrorEventMask` property of the CWIMAQ control to specify the operations for which the error and warning events are called.

Debugging

This section outlines a number of general debugging methods that you might use in your application development. If you experience some unexpected behavior in your program, use these methods to locate and correct the problem in your application.

Debug Print

One of the most common debugging methods is to print out or display important variables throughout the program execution. You can monitor critical values and determine when your program varies from the expected progress. Some programming environments have dedicated debugging windows that are used to display such information without disturbing the rest of the user interface. For example, you can use the `Debug.Print` command in Visual Basic to print information directly to the debug window.

```
Debug.Print CWIMAQ1.Interface
```

Breakpoint

Most development environments include breakpoint options so you can suspend program execution at a specific point in your code. Breakpoints are placed on a specific line of executable code in the program to pause program execution.

Stopping at a breakpoint confirms that your application ran to the line of code containing the breakpoint. If you unsure whether a specific section of code is being called, place a breakpoint in the routine to find out. Once you have stopped at a specific section of your code, you can use other tools, such as a watch window or debug window, to analyze or even edit variables.

In some environments, breakpoints might include conditions so program execution halts if certain other conditions are met. These conditions usually check program variables for specific values. Once you have completed the work at the breakpoint, you can continue running your program, either in the normal run mode or in some type of single-step mode.

Watch Window

Use a watch window to display the value of a variable during program execution. You can use it to edit the value of a variable while the program is paused. In some cases, you can display expressions, which are values calculated dynamically from one or more program variables.

Single Step, Step Into, and Step Over

Use *single stepping* to execute a program one line at a time. This way, you can check variables and the output from your program during execution. Single stepping is commonly used after a breakpoint to slowly step through a questionable section of code.

If you use *step into*, the program executes any code available for subroutines or function calls and steps through it one line at a time. Use this mode if you want to check the code for each function called. The *step over* mode assumes that you do not want to go into the code for functions being called and runs them as one step.

In some cases, you might want to test a limited number of iterations of a loop but then run the rest of the iterations without stopping again. For this type of debugging, several environments include an option *step to cursor* or *run to cursor* options. Under this option, you can place your cursor at a specific point in the code, such as the first line after a loop, and run the program to that point.

Part III

Introduction to Vision

This section presents the basics of computer-based vision applications.

Part III, *Introduction to Vision*, contains the following chapters.

- Chapter 9, *Algorithms, Principles of Image Files, and Data Structures*, describes the algorithms and principles of image files and data structures.
- Chapter 10, *Tools and Utilities*, describes the tools and utilities used in IMAQ Vision.
- Chapter 11, *Lookup Transformations*, provides an overview of lookup table transformations.
- Chapter 12, *Operators*, describes the arithmetic and logic operators used in IMAQ Vision.
- Chapter 13, *Spatial Filtering*, provides an overview of the spatial filters, including linear and nonlinear filters, used in IMAQ Vision.
- Chapter 14, *Frequency Filtering*, describes the frequency filters used in IMAQ Vision.
- Chapter 15, *Morphology Analysis*, provides an overview of morphology image analysis.
- Chapter 16, *Quantitative Analysis*, provides an overview of quantitative image analysis. The *quantitative analysis* of an image consists of obtaining *densitometry* and object measurements. Before starting this analysis, it is necessary to calibrate the image spatial dimensions and intensity scale to obtain measurements expressed in real units.

9

Algorithms, Principles of Image Files, and Data Structures

This chapter describes the algorithms and principles of image files and data structures.

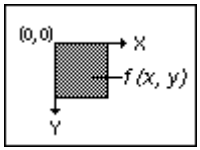
Introduction to Digital Images

An *image* is a function of the light intensity

$$f(x, y)$$

where f is the brightness of the point (x, y), and x and y represent the spatial coordinates of a *picture element* (abbreviated *pixel*).

By default the spatial reference of the pixel with the coordinates (0, 0) is located at the upper-left corner of the image.



In *digital image processing*, an acquisition device converts an image into a discrete number of pixels. This device assigns a numeric location and *gray-level* value which specifies the brightness of pixels.

Properties of a Digitized Image

A digitized image has three basic properties: *image resolution*, *image definition*, and *number of planes*.

Image Resolution

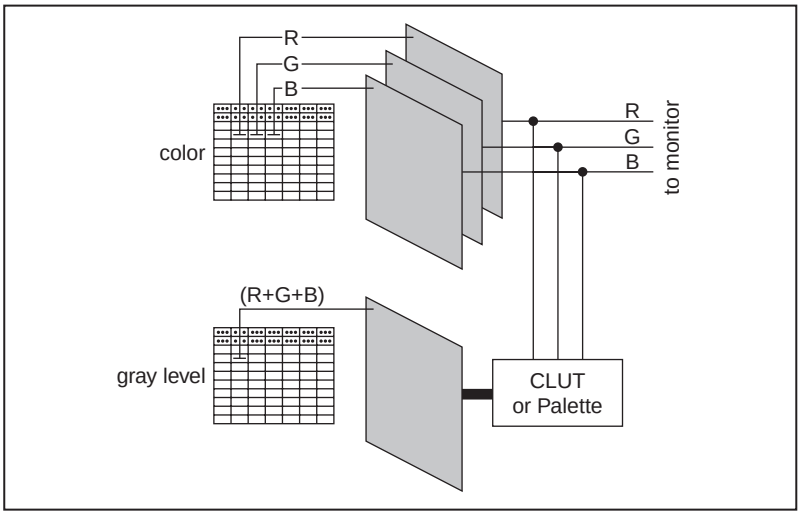
The *spatial resolution* of an image is its number of rows and columns of pixels. An image composed of *m* rows and *n* columns has a resolution of *mn*. This image has *n* pixels along its horizontal axis and *m* pixels along its vertical axis.

Image Definition

The definition of an image, also called *pixel depth*, indicates the number of colors or shades that you can see in the image. Pixel depth is the number of bits used to code the intensity of a pixel. For a given definition of *n*, a pixel can take 2^n different values. For example, if *n* equals 8-bits, a pixel can take 256 different values ranging from 0 to 255. If *n* equals 16 bits, a pixel can take 65,536 different values ranging from 0 to 65,535 or –32,768 to 32,767.

Number of Planes

The number of planes in an image is the number of arrays of pixels that compose the image. A gray-level or pseudo-color image is composed of one plane, while a true-color image is composed of three planes (one for the red component, one for the blue, and one for the green), as shown in the following figure.



In gray-level images, the red, green, and blue intensities (*RGB*) of a pixel combine to produce a single value. This single value is converted back to an RGB intensity when displayed on a monitor. This conversion is performed by a *color lookup table* (CLUT) transformation.

In three-plane or true color images, the red, green, and blue intensities of a pixel are coded into three different values. The image is the combination of three arrays of pixels corresponding to the red, green, and blue components.

Image Types and Formats

The IMAQ Vision libraries can manipulate three types of images: *gray-level*, *color*, and *complex* images.

Gray-Level Images

Gray-level images are composed of a single plane of pixels. Standard gray-level formats are 8-bit *PICT* (Macintosh only), *BMP* (PC only), *TIFF*, *RASTR*, and *AIPD*. Standard 16-bit gray-level formats are TIFF and AIPD. AIPD is an internal file format that offers the advantage of storing the spatial calibration of an image. Gray-level images that use other formats and have a pixel depth of 8-bit, 16-bit or 32-bit can be imported into the IMAQ Vision libraries.

Color Images

Color images are composed of three planes of pixels in which each pixel has a red, green, and blue intensity, each coded on 8-bit planes. Color images coded using the *RGB-chunky* standard contain an extra 8-bit plane, called the *alpha channel*. These images have a definition of 32-bit or 4 × 8-bit. Standard color formats are PICT, BMP, TIFF and AIPD.

Complex Images

Complex images are composed of complex data in which pixel values have a real part and an imaginary part. Such images are derived from the *Fast Fourier Transform* of gray-level images. Four representations of a complex image can be given: the real part, imaginary part, magnitude, and phase.

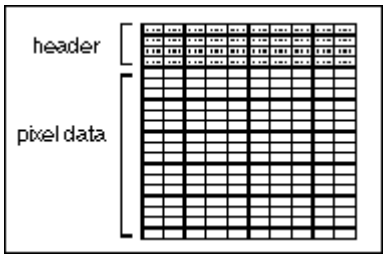
The following table shows how many bytes are used per pixel in gray-level, color, and complex images. For an identical spatial resolution, a color image occupies four times the memory space used by an 8-bit gray-level image and a complex image occupies eight times this amount.

Table 9-1. Bytes Per Pixel

Image Type	Number of Bytes Per Pixel Data			
8-bit (Unsigned) Integer Gray-Level (1 byte or 8-bit)	8-bit for the gray-level intensity			
16-bit (Signed) Integer Gray-Level (2 bytes or 16-bit)	16-bit for the gray-level intensity			
32-bit Floating-Point Gray-Level (4 bytes or 32-bit)	32-bit floating for the gray-level intensity			
Color (3 bytes or 24-bit)	8-bit for the alpha value (not used)	8-bit for the red intensity	8-bit for the green intensity	8-bit for the blue intensity
Complex (8 bytes or 64-bit)	32-bit floating for the real part		32-bit floating for the imaginary part	

Image Files

An *image file* is composed of a header followed by pixel values. Depending on the file format, the header contains information such as the image horizontal and vertical resolution, its pixel definition, the physical calibration, and the original palette.



Processing Color Images

Most image-processing and analysis functions apply to 8-bit images. However, you also can process color images by manipulating their color components individually.

You can break down a color image into various sets of primary components such as RGB (red, green, and blue), *HSL* (hue, saturation, and lightness), or *HSV* (hue, saturation, and value). Each component becomes an 8-bit image and can be processed as any gray-level image.

You can reassemble a color image later from a set of three 8-bit images taking the place of its RGB, HSL, or HSV components.

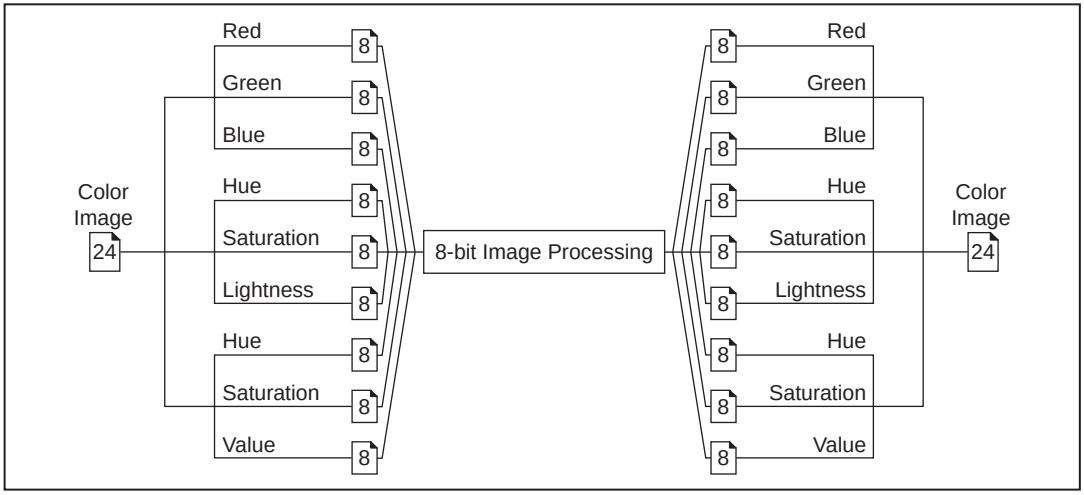
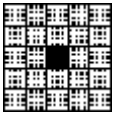
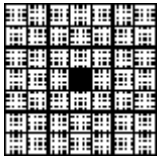
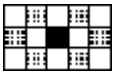
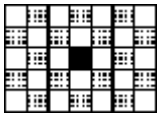
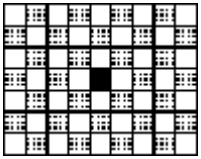


Image Pixel Frame

As introduced earlier, a digital image is a two-dimensional array of pixel values. Using this definition, you might assume that pixels are arranged in a regular rectangular frame. However from an image processing point of view you can consider another grid arrangement, such as a hexagonal pixel frame which offers the advantage that the six neighbors of a pixel are equidistant.

The pixels in an image are arranged in a rectangular grid. However, some image processing algorithms can reproduce a hexagonal neighborhood using the representations illustrated in the following table. The pixels considered as neighbors of the given pixel (shown in solid) are indicated by the shaded pattern.

Pixel Frame	Neighborhood Size		
Rectangular	3 × 3	5 × 5 	7 × 7 
Hexagonal	5 × 3 	7 × 5 	9 × 7 

Rectangular Frame

Each pixel is surrounded by eight neighbors.

$\sqrt{2}d$	d	$\sqrt{2}d$
d		d
$\sqrt{2}d$	d	$\sqrt{2}d$

Figure 9-1. Rectangular Frame

If d is the distance from the vertical and horizontal neighbors to the central pixel, then the diagonal neighbors are at a distance of $\sqrt{2}d$ from the central pixel.

Hexagonal Frame

Each pixel is surrounded by six neighbors. Each neighbor is found at an equal distance *d* from the central pixel.

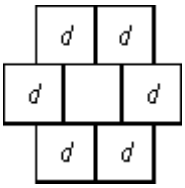


Figure 9-2. Hexagonal Frame

This notion of pixel frame is important for a category of image processing functions called *neighborhood operations*. These functions alter the value of pixels depending on the intensity values of their neighbors. They include *spatial filters*, which alter the intensity of a pixel with respect to variations in intensities of neighboring pixels, and *morphological transformations*, which extract and alter the structure of objects in an image.

10

Tools and Utilities

This chapter describes the tools and utilities used in IMAQ Vision.

Palettes

At the time an image is displayed on the screen, the value of each pixel is converted into a red, green, and blue intensity which produces a color. This conversion is defined in a table called color lookup table (CLUT). For 8-bit images, it associates a color to each gray-level value and produces a gradation of colors, called a *palette*.

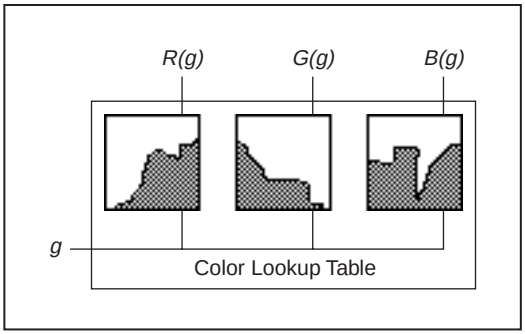
With palettes, you can produce different visual representations of an image without altering the pixel data. Palettes can generate effects such as a photonegative display or color-coded displays. In the latter case, palettes are useful for detailing particular image constituents in which the total number of colors are limited.

Displaying images in different palettes helps emphasize regions with particular intensities, identify smooth or abrupt gray-level variations, and convey details that might be lost in a gray-scale image.

In the case of 8-bit resolution, pixels can take 2^8 or 256 values ranging from 0 to 255. A black and white palette associates different shades of gray to each value so as to produce a linear and continuous gradation of gray, from black to white. At this point, the palette can be set up to assign the color black to the value 0 and white to 255, or vice versa. Other palettes can reflect linear or nonlinear gradations going from red to blue, light brown to dark brown, and so forth.

The gray-level value of a pixel acts as an address that is indexed into three tables, with three values corresponding to a red, green, and blue (RGB) intensity. This set of three conversion tables defines a palette in which

varying amounts of red, green, and blue are mixed to produce a color representation of the value range [0, 255].



Five pseudo-color palettes are predefined in the programs and libraries. Each palette emphasizes different shades of gray. However, they all use the following conventions:

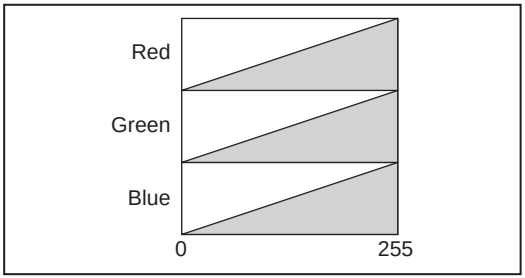
- Gray level 0 is assigned to black.
- Gray level 255 is assigned to white.

Because of these conventions, you can associate bright areas to the presence of pixels with high gray-level values, and dark areas to the presence of pixels with low gray-level values.

The following sections introduce the five predefined palettes. The graphs in each section represent the three RGB lookup tables used by each palette. The horizontal axes of the graphs represent the input gray-level range [0, 255], while the vertical axes give the RGB intensities assigned to a given gray-level value.

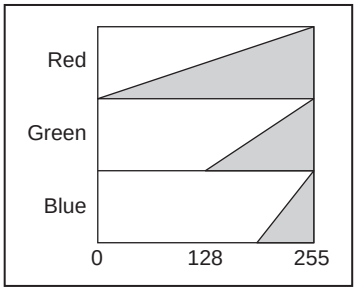
B&W (Gray) Palette

This palette has a gradual gray-level variation from black to white. Each value is assigned to an equal amount of the RGB intensities.



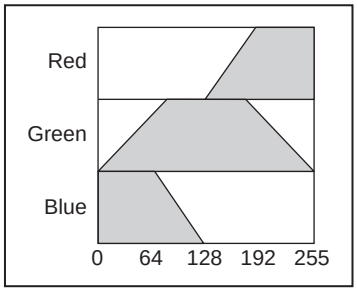
Temperature Palette

This palette has a gradation from light brown to dark brown. 0 is black and 255 is white.



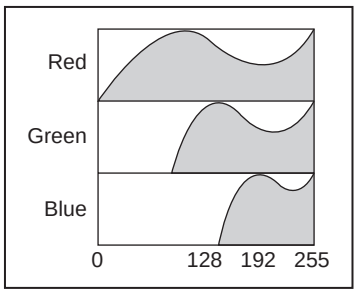
Rainbow Palette

This palette has a gradation from blue to red with a prominent range of greens in the middle value range. 0 is black and 255 is white.



Gradient Palette

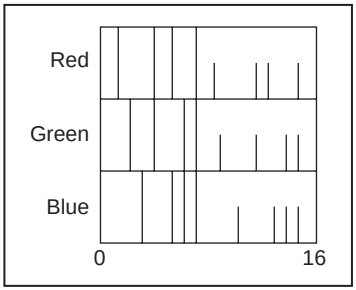
This palette has a gradation from red to white with a prominent range of light blue in the upper value range. 0 is black and 255 is white.



Binary Palette

This palette has 16 cycles of 16 different colors, *where g* is the gray-level value and

$g = 0$ corresponds to $R = 0$, $G = 0$, $B = 0$, which appears black;
$g = 1$ corresponds to $R = 1$, $G = 0$, $B = 0$,which appears red;
$g = 2$ corresponds to $R = 0$, $G = 1$, $B = 0$ which appears green;
and so forth.



This periodic palette is appropriate for the display of binary and labeled images.

Image Histogram

The *histogram* of an image indicates the quantitative distribution of pixels per gray-level value. It provides a general description of the appearance of an image and helps identify various components such as the background, objects, and noise.

Definition

The histogram is the function H defined on the gray-scale range [0, …, k, …, 255] such that the number of pixels equal to the gray-level value k is

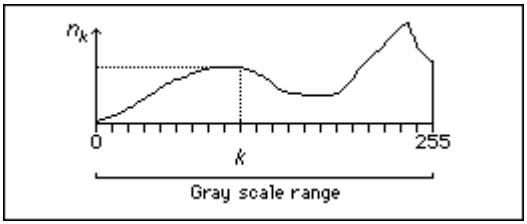
$$H(k) = n_k$$

where k is the gray-level value,

n_k is the number of pixels in an image with a gray-level value equal to k,

and $\sum n_k = n$ is the total number of pixels in an image.

The following histogram plot reveals which gray levels occur frequently and which occur rarely.



Two types of histograms can be plotted per image: the linear and cumulative histograms.

In both cases, the horizontal axis represents the gray-level range from 0 to 255. For a gray-level value k, the vertical axis of the linear histogram indicates the number of pixels n_k set to the value k, and the vertical axis of the cumulative histogram indicates the percentage of pixels set to a value less than or equal to k.

Linear Histogram

The *density function* is

$$H_{Linear}(k) = n_k$$

where $H_{Linear}(k)$ is the number of pixels equal to k.

The *probability function* is

$$P_{Linear}(k) = n_k / n$$

where $P_{Linear}(k)$ is the probability that a pixel is equal to k.

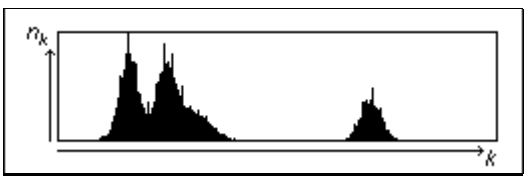


Figure 10-1. Linear Vertical Scale

Cumulative Histogram

The distribution function is

$$H_{Cumul}(k) = \sum_{0}^{k} n_k$$

where $H_{Cumul}(k)$ is the number of pixels that are less than or equal to k.

The probability function is

$$P_{Cumul}(k) = \sum_{0}^{k} \frac{n_k}{n}$$

where $P_{Cumul}(k)$ is the probability that a pixel is less than or equal to k.

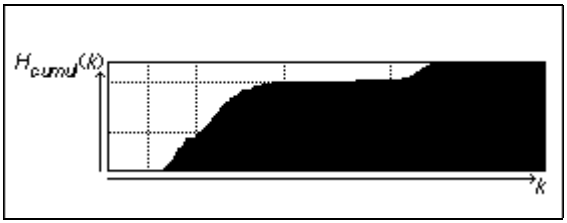


Figure 10-2. Linear Cumulative Scale

Interpretation

The gray-level intervals with a concentrated set of pixels reveal the presence of significant components in the image and their respective intensity ranges.

In the previous example, the linear histogram reveals that the image is composed of three major elements. The cumulative histogram shows that the two left-most peaks compose approximately 80 percent of the image, while the remaining 20 percent corresponds to the third peak.

Histogram of Color Images

The histogram of a color image is expressed as a series of three tables corresponding to the histograms of the three primary components (R, G, and B; H, S, and L; or H, S, and V).

Histogram Scale

The vertical axis of a histogram plot can be shown in a linear or logarithmic scale. A logarithmic scale lets you visualize gray-level values used by small numbers of pixels. These values might appear unused when the histogram is displayed in a linear scale.

In the case of a logarithmic scale, the vertical axis of the histogram gives the logarithm of the number of pixels per gray-level value. The use of minor gray-level values is made more prominent at the expense of the dominant gray-level values.

The following two figures illustrate the difference between the display of the histogram of the same image in a linear and logarithmic scale. In this particular image, three pixels are equal to 0. This information is unobservable in the linear representation of the histogram but evident in the logarithmic representation.

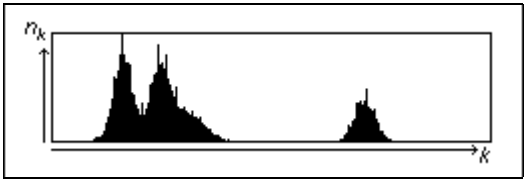


Figure 10-3. Linear Vertical Scale

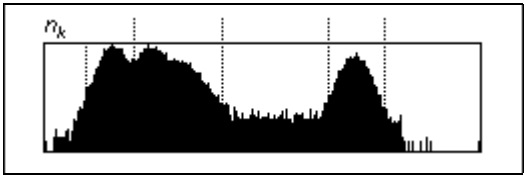
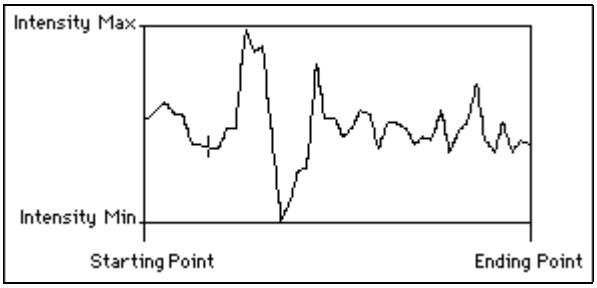


Figure 10-4. Logarithmic Vertical Scale

Line Profile

A *line profile* plots the variations of intensity along a line. This utility is helpful for examining boundaries between components, quantifying the magnitude of intensity variations, and detecting the presence of repetitive patterns. The following figure illustrates a line profile.

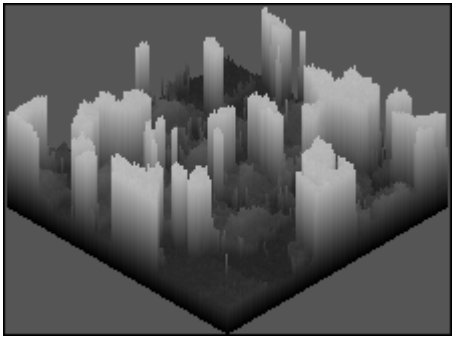


The peaks and valleys reveal increases and decreases of the light intensity along the line selected in the image. Their width and magnitude are proportional to the size and intensity of their related regions.

For example, a bright object with uniform intensity appears in the plot as a plateau. The higher the contrast between an object and its surrounding background, the steeper the slopes of the plateau. Noisy pixels, on the other hand, produce a series of narrow peaks.

3D View

The *3D view* illustrated in the following graphic displays a three-dimensional perspective of the light intensity in an image. It gives a relief map of the image in which high-intensity values are associated to summits and low-intensity values are associated to valleys.



11

Lookup Transformations

This chapter provides an overview of lookup table transformations.

About Lookup Table Transformations

The *lookup table* (LUT) transformations are basic image-processing functions that you can use to improve the contrast and brightness of an image by modifying the intensity dynamic of regions with poor contrast. The LUT transformations can highlight details in areas containing significant information, at the expense of other areas. These functions include *histogram equalization*, *histogram inversion*, *Gamma corrections*, *Inverse Gamma corrections*, *logarithmic corrections*, and *exponential corrections*.

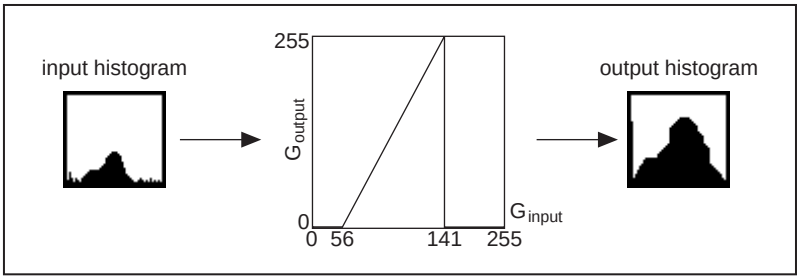
An LUT transformation converts input gray-level values (those from the source image) into other gray-level values (in the transformed image). The transfer function has an intended effect on the brightness and contrast of the image.

Each input gray-level value is given a new value such that

$$\textit{output value} = F(\textit{input value}),$$

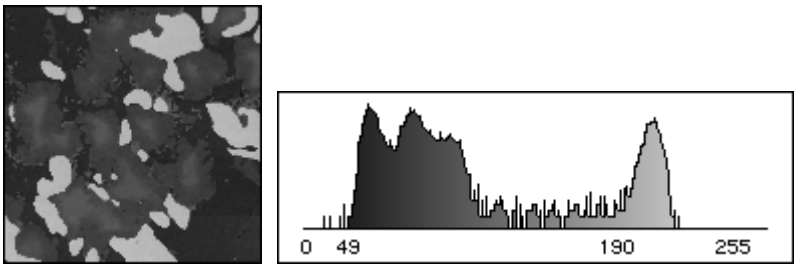
where F is a linear or nonlinear, continuous or discontinuous transfer function defined over the interval [0, max].

In the case of an 8-bit resolution, an LUT is a table of 256 elements. Each element of the array represents an input gray-level value. Its content indicates the output value.



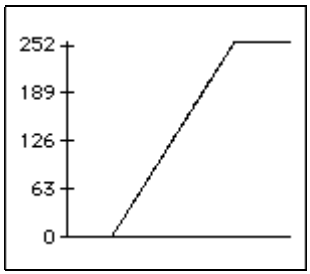
Example

In this example, the following source image is used. In the histogram of the source image, the gray-level intervals [0, 49] and [191, 255] do not contain significant information.

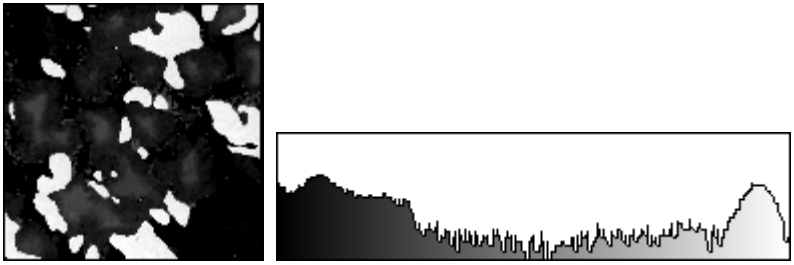


Using the following LUT transformation, any pixel with a value less than 49 is set to 0, and any pixel with a value greater than 191 is set to 255. The interval [50, 190] expands to [1, 255], increasing the intensity dynamic of the regions with a concentration of pixels in the gray-level range [50, 190].

If G_{input} is between [0, 49],
then $F(G_{input}) = 0$,
If G_{input} is between [191, 255],
then $F(G_{input}) = 255$,
else $F(G_{input}) = 1.8 \times G_{input} - 91$.



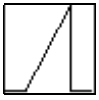
The LUT transform produces the following image. The histogram of the new image only contains the two peaks of the interval [50, 190].



Predefined Lookup Tables

Eight predefined LUTs are available in IMAQ Vision: Reverse, Equalize, Logarithmic, Power 1/Y, Square Root, Exponential, Power Y, and Square.

The following table shows the transfer function for each LUT and describes its effect on an image displayed in a palette that associates dark colors to low intensity values and bright colors to high intensity values (such as the B&W or Gray palette).

LUT	Transfer Function	Shading Correction
Equalize		Increases the intensity dynamic by evenly distributing a given gray-level interval [min, max] over the full gray scale [0, 255]. Min and max default values are 0 and 255 for an 8-bit image.
Reverse		Reverses the pixel values, producing a photometric negative of the image.
Logarithmic Power 1/Y Square Root		Increases the brightness and contrast in dark regions. Decrease the contrast in bright regions.
Exponential Power Y Square		Decreases the brightness and increases the contrast in bright regions. Decreases the contrast in the dark regions.

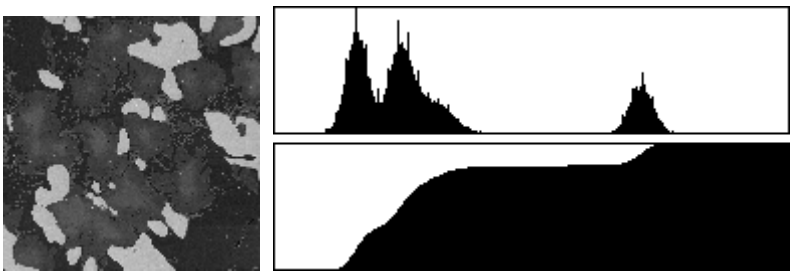
Equalize

The *Equalize function* alters the gray-level value of pixels so they become distributed evenly in the defined gray-scale range (0 to 255 for an 8-bit image). The function associates an equal amount of pixels per constant gray-level intervals and takes full advantage of the available shades of gray. Use this transformation to increase the contrast of images in which gray-level intervals are not used.

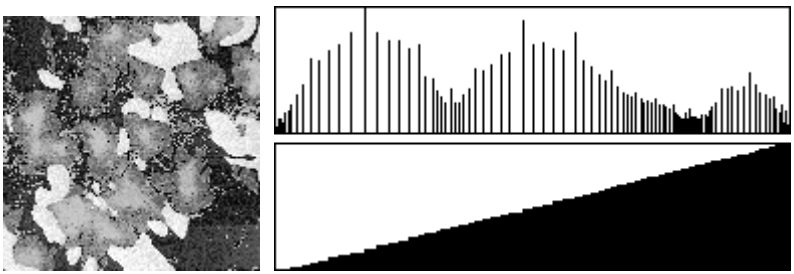
The equalization can be limited to a gray-level interval, also called the equalization range. In this case, the function evenly distributes the pixels belonging to the equalization range over the full interval (0 to 255 for an 8-bit image) and the other pixels are set to 0. The image produced reveals details in the regions that have an intensity in the equalization range; other areas are cleared.

Example 1

This example shows how an equalization of the interval [0, 255] can spread the information contained in the three original peaks over larger intervals. The transformed image reveals more details about each component in the original image. The following graphics show the original image and histograms.



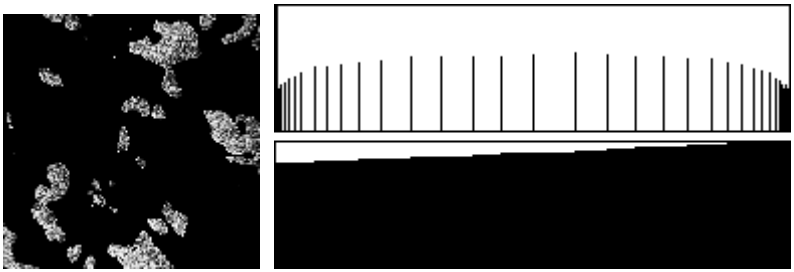
An equalization from [0, 255] to [0, 255] produces the following image and histograms.



Note ***The cumulative histogram of an image after a histogram equalization always has a linear profile, as seen in the preceding example.***

Example 2

This example shows how an equalization of the interval [166, 200] can spread the information contained in the original third peak (ranging from 166 to 200) to the interval [1, 255]. The transformed image reveals details about the component with the original intensity range [166, 200] while all other components are set to black. An equalization from [166, 200] to [0, 255] produces the following image and histograms.



Reverse

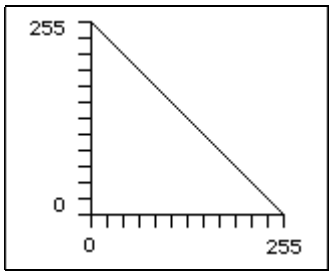
The Reverse function displays the photometric negative of an image.

G_{output} = $Maximum - G_{input}$

For an 8-bit image, *Maximum* = 255. Therefore,

G_{output} = $255 - G_{input}$

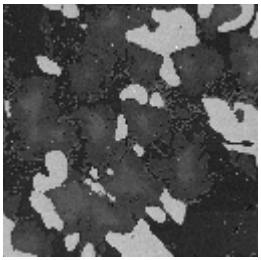
0	corresponds to	255
1	corresponds to	254
2	corresponds to	253
...		
128	corresponds to	128
...		
253	corresponds to	2
254	corresponds to	1
255	corresponds to	0



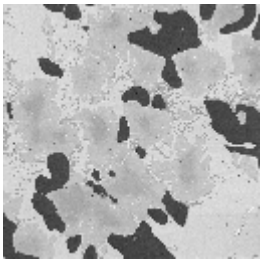
The histogram of a reversed image is equal to the histogram of the original image after a vertical symmetry centered on the gray-level value 128 (when processing an 8-bit image).

Example

This example uses the following original image and histogram.



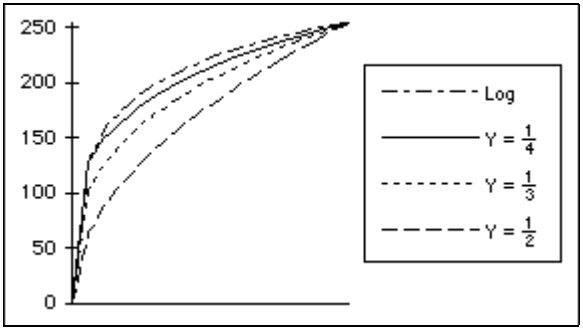
A Reverse transformation produces the following histogram and image.



Logarithmic and Inverse Gamma Correction

The *logarithmic and inverse gamma corrections* expand low gray-level ranges while compressing high gray-level ranges. When using the B&W (or Gray) palette, these transformations increase the overall brightness of an image and increase the contrast in dark areas at the expense of the contrast in bright areas.

The following graphs show how the transformations behave. The horizontal axis represents the input gray-level range and the vertical axis represents the output gray-level range. Each input gray-level value is plotted vertically, and its point of intersection with the lookup curve is plotted horizontally to give an output value.

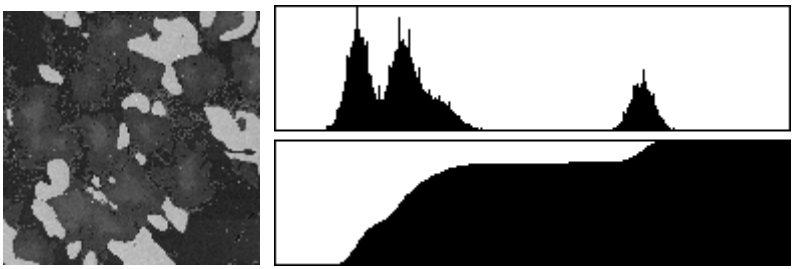


The *Logarithmic*, *Square Root*, and *Power 1/Y* functions expand intervals containing low gray-level values while compressing intervals containing high gray-level values.

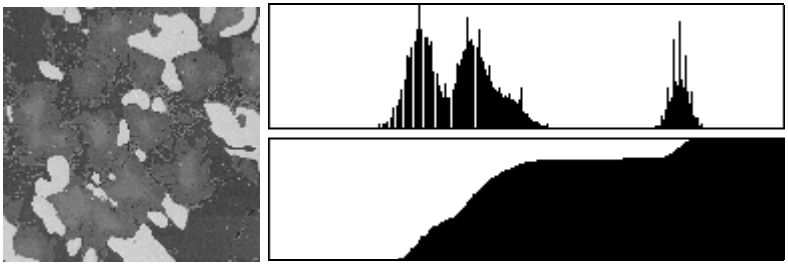
The higher the gamma coefficient *Y*, the stronger the intensity correction. The Logarithmic correction has a stronger effect than the Power 1/Y function.

The following series of illustrations presents the linear and cumulative histograms of an image after various LUT transformations. The more the histogram is compressed on the right, the brighter the image.

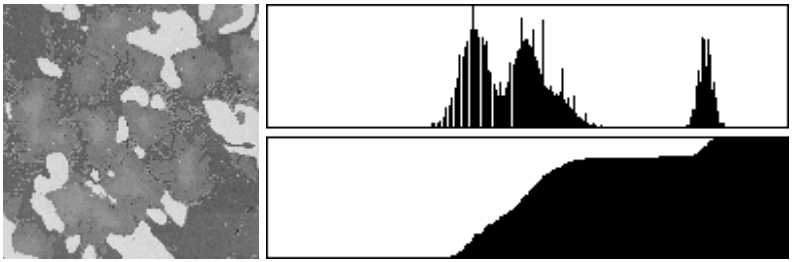
The following graphic shows the original image and histograms.



A Power 1/Y transformation (where $Y = 1.5$) produces the following image and histograms.



A Square Root or Power 1/Y transformation (where $Y = 2$) produces the following image and histograms.

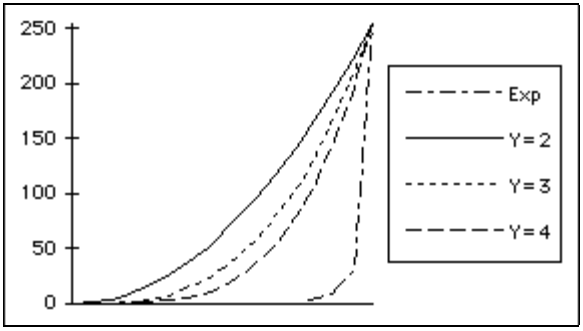


A Logarithm transformation produces the following image and histograms.

Exponential and Gamma Correction

The *exponential and gamma corrections* expand high gray-level ranges while compressing low gray-level ranges. When using the B&W (or Gray) palette, these transformations decrease the overall brightness of an image and increase the contrast in bright areas at the expense of the contrast in dark areas.

The following graphs show how the transformations behave. The horizontal axis represents the input gray-level range and the vertical axis represents the output gray-level range. Each input gray-level value is plotted vertically, and its point of intersection with the lookup curve then is plotted horizontally to give an output value.

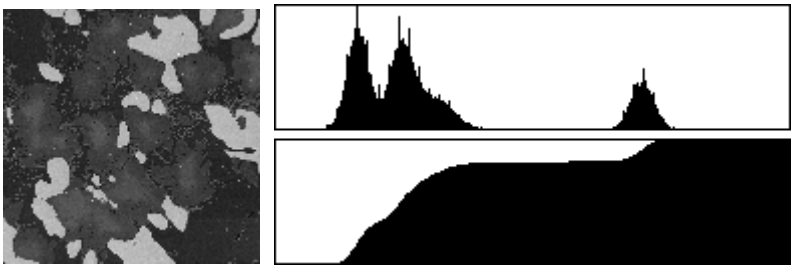


The *Exponential*, *Square*, and *Power Y* functions expand intervals containing high gray-level values while compressing intervals containing low gray-level values.

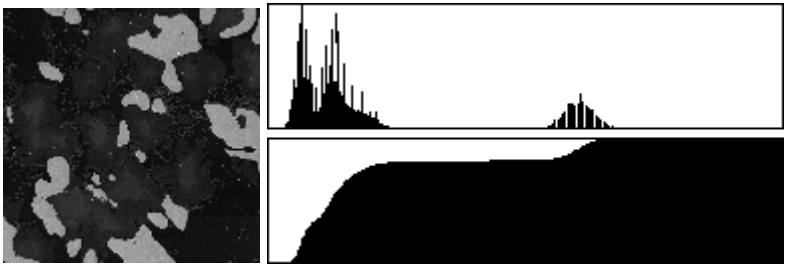
The higher the gamma coefficient *Y*, the stronger the intensity correction. The Exponential correction has a stronger effect than the Power Y function.

The following series of illustrations presents the linear and cumulative histograms of an image after various LUT transformations. The more the histogram is compressed on the left, the darker the image.

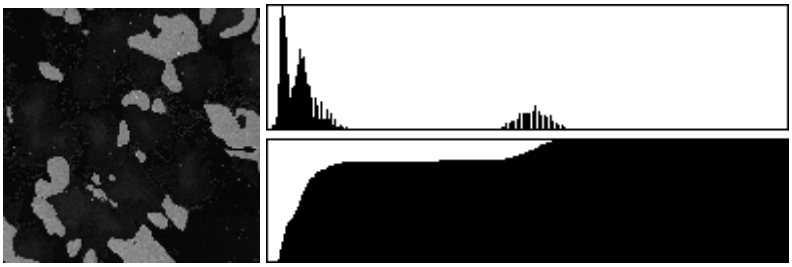
The following graphic shows the original image and histograms.



A Power Y transformation (where $Y = 1.5$) produces the following image and histograms.



A Square or Power Y transformation (where $Y = 2$) produces the following image and histograms.



An Exponential transformation produces the following image and histograms.

12

Operators

This chapter describes the arithmetic and logic operators used in IMAQ Vision.

Concepts and Mathematics

Arithmetic and logic *operators* mask, combine, and compare images. Common applications of these operators include time-lapse comparisons, identification of the union or intersection between images, and comparisons between several images and a model. Operators also can be used to *threshold* or *mask* images and to alter contrast and brightness.

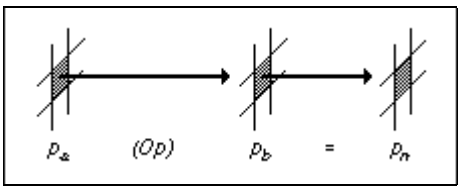
An arithmetic or logic operation between images is a pixel-by-pixel transformation. It produces an image in which each pixel derives from the values of pixels with the same coordinates in other images.

If *A* is an image with a resolution *XY*, *B* is an image with a resolution *XY*, and *Op* is the operator,

then the image *N* resulting from the combination of *A* and *B* through the operator *Op* is such that each pixel *P* of *N* is assigned the value

$$p_n = (p_a)(Op)(p_b),$$

where p_a is the value of pixel *P* in image *A*, and p_b is the value of pixel *P* in image *B*.



Arithmetic Operators

In the case of images with 8-bit resolution, the following equations describe the usage of the *arithmetic operators*:

Operator	Equation
Multiply	$p_n = \min(p_a \times p_b, 255)$
Divide	$p_n = \max(p_a/p_b, 0)$
Add	$p_n = \min(p_a + p_b, 255)$
Subtract	$p_n = \max(p_a - p_b, 0)$
Remainder	$p_n = p_a \bmod p_b$

If the resulting pixel value p_n is negative, it is set to 0. If it is greater than 255, it is set to 255.

Logic Operators

Logic operators are bit-wise operators. They manipulate gray-level values coded on one byte at the bit level. The *truth tables* for logic operators are presented in the *Truth Tables* section.

Operator	Equation
AND	$p_n = p_a$ AND p_b
NAND	$p_n = p_a$ NAND p_b
OR	$p_n = p_a$ OR p_b
NOR	$p_n = p_a$ NOR p_b
XOR	$p_n = p_a$ XOR p_b
Difference	$p_n = p_a$ AND (NOT p_b)
Mask	*if* $p_b = 0$, *then* $p_n = 0$, *else* $p_n = p_a$

Operator	Equation
Mean	p_n = mean[p_a, p_b]
Max	p_n = max[p_a, p_b]
Min	p_n = min[p_a, p_b]

In the case of images with 8-bit resolution, logic operators mainly are designed to combine gray-level images with mask images composed of pixels equal to 0 or 255 (in binary format 0 is represented as 00000000 and 255 is represented as 11111111).

The following table illustrates how logic operations can be used to extract or remove information in an image.

For a given p_a	***If p_b = 255, then***	***If p_b = 0, then***
(AND)	p_a AND 255 = p_a	p_a AND 0 = 0
(NAND)	p_a NAND 255 = NOT p_a	p_a NAND 0 = 255
(OR)	p_a OR 255 = 255	p_a OR 0 = p_a
(NOR)	p_a NOR 255 = 0	p_a NOR 0 = NOT p_a
(XOR)	p_a XOR 255 = NOT p_a	p_a XOR 0 = p_a
(Logic Difference)	p_a – NOT 255 = p_a	p_a – NOT 0 = 0

Truth Tables

The following truth tables describe the rules used by the logic operators. The top row and left column give the values of input bits. The cells in the table give the output value for a given set of two input bits.

AND

	***b* = 0**	***b* = 1**
***a* = 0**	0	0
***a* = 1**	0	1

NAND

	***b* = 0**	***b* = 1**
***a* = 0**	1	1
***a* = 1**	1	0

OR

	***b* = 0**	***b* = 1**
***a* = 0**	0	1
***a* = 1**	1	1

NOR

	***b* = 0**	***b* = 1**
***a* = 0**	1	0
***a* = 1**	0	0

XOR

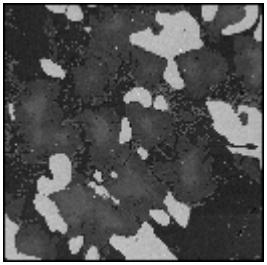
	***b* = 0**	***b* = 1**
***a* = 0**	0	1
***a* = 1**	1	0

NOT

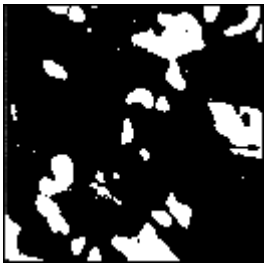
	NOT *a*
***a* = 0**	1
***a* = 1**	0

Example 1

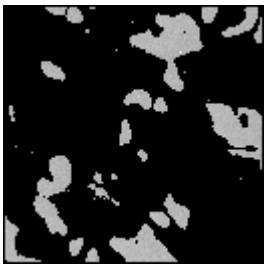
The following series of graphics illustrates images in which regions of interest have been isolated in a binary format, retouched with morphological manipulations, and finally multiplied by 255. The following gray-level source image is used for this example.



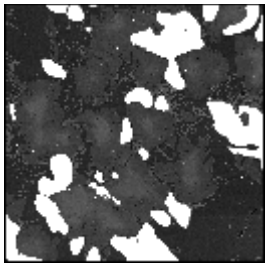
The following *mask image* results.



The operation (*source image* AND *mask image*) has the effect of restoring the original intensity of the object regions in the mask.



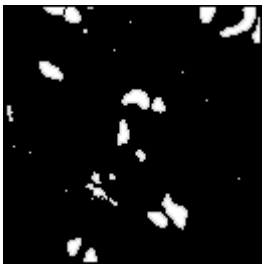
The operation (*source image* OR *mask image*) has the effect of restoring the original intensity of the background region in the mask.



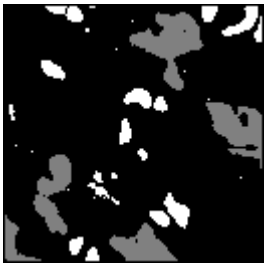
Example 2

An image reveals two groups of objects that require different processing results in two binary images. Multiplying each binary image by a constant and applying an OR operation produces an image that shows their union, as illustrated in the following series of graphics. The following image illustrates *Object Group #1* × 128.

The following image illustrates *Object Group #2* × 255.



Object Group #1 OR *Object Group #2* produces a union, as shown in the following image.



13

Spatial Filtering

This chapter provides an overview of the spatial filters, including linear and nonlinear filters, used in IMAQ Vision.

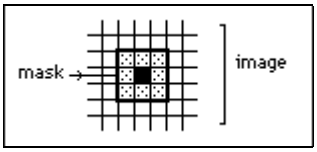
Concept and Mathematics

Spatial filters alter pixel values with respect to variations in light intensity in their neighborhood. The neighborhood of a pixel is defined by the size of a matrix, or mask, centered on the pixel itself. These filters can be sensitive to the presence or absence of light intensity variations. Spatial filters can serve a variety of purposes, such as the detection of edges along a specific direction, the contouring of patterns, noise reduction, and detail outlining or smoothing.

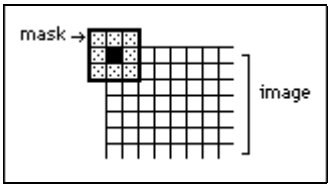
Spatial filters can be divided into two categories:

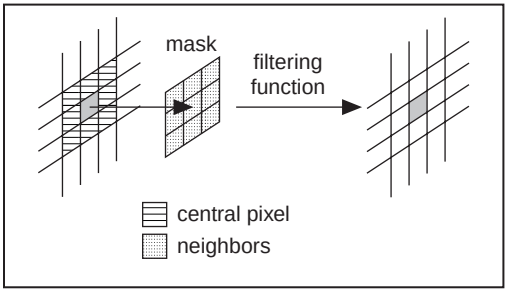
- *Highpass filters* emphasize significant variations of the light intensity usually found at the boundary of objects.
- *Lowpass filters* attenuate variations of the light intensity. They have the tendency to smooth images by eliminating details and blurring edges.

In the case of a 3 × 3 matrix as illustrated in the following illustration, the value of the central pixel (shown in solid) derives from the values of its eight surrounding neighbors (shown in shaded pattern).



A 5 × 5 matrix specifies 24 neighbors, a 7 × 7 matrix specifies 48 neighbors, and so forth.





If $P_{(i,j)}$ represents the intensity of the pixel P with the coordinates (i, j), the pixels surrounding $P_{(i,j)}$ can be indexed as follows (in the case of a 3 × 3 matrix):

$P_{(i-1,j-1)}$	$P_{(i,j-1)}$	$P_{(i+1,j-1)}$
$P_{(i-1,j)}$	$P_{(i,j)}$	$P_{(i+1,j)}$
$P_{(i-1,j+1)}$	$P_{(i,j+1)}$	$P_{(i+1,j+1)}$

A *linear filter* assigns to $P_{(i,j)}$ a value that is a linear combination of its surrounding values. For example:

$$P_{(i,j)} = (P_{(i,j-1)} + P_{(i-1,j)} + 2P_{(i,j)} + P_{(i+1,j)} + P_{(i,j+1)}).$$

A *nonlinear filter* assigns to $P_{(i,j)}$ a value that is not a linear combination of the surrounding values. For example:

$$P_{(i,j)} = \max(P_{(i-1,j-1)}, P_{(i+1,j-1)}, P_{(i-1,j+1)}, P_{(i+1,j+1)}).$$

Spatial Filter Classification Summary

The following table describes the classification of spatial filters

.

	Highpass Filters	Lowpass Filters
Linear Filters	Gradient, Laplacian	Smoothing, Gaussian
Nonlinear Filters	Gradient, Roberts, Sobel, Prewitt, Differentiation, Sigma	Median, Nth Order, Lowpass

Linear Filters or Convolution Filters

A *convolution* is a mathematical function that replaces each pixel by a weighted sum of its neighbors. The matrix defining the neighborhood of the pixel also specifies the weight assigned to each neighbor. This matrix is called the *convolution kernel*.

For each pixel $P_{(i, j)}$ in an image (where i and j represent the coordinates of the pixel), the convolution kernel is centered on $P_{(i, j)}$. Each pixel masked by the kernel is multiplied by the coefficient placed on top of it. $P_{(i, j)}$ becomes the sum of these products.

In the case of a 3 × 3 neighborhood, the pixels surrounding $P_{(i, j)}$ and the coefficients of the kernel, K, can be indexed as follows:

$P_{(i-1, j-1)}$	$P_{(i, j-1)}$	$P_{(i+1, j-1)}$
$P_{(i-1, j)}$	$P_{(i, j)}$	$P_{(i+1, j)}$
$P_{(i-1, j+1)}$	$P_{(i, j+1)}$	$P_{(i+1, j+1)}$

$K_{(i-1, j-1)}$	$K_{(i, j-1)}$	$K_{(i+1, j-1)}$
$K_{(i-1, j)}$	$K_{(i, j)}$	$K_{(i+1, j)}$
$K_{(i-1, j+1)}$	$K_{(i, j+1)}$	$K_{(i+1, j+1)}$

The pixel $P_{(i, j)}$ is given the value $(1/N)\Sigma K_{(a, b)}P_{(a, b)}$, with a ranging from $(i - 1)$ to $(i + 1)$, and b ranging from $(j - 1)$ to $(j + 1)$. N is the *normalization factor*, equal to $\Sigma K_{(a, b)}$ or 1, whichever is greater.

Finally, if the new value $P_{(i, j)}$ is negative, it is set to 0. If the new value $P_{(i, j)}$ is greater than 255, it is set to 255 (in the case of 8-bit resolution).

The greater the absolute value of a coefficient $K_{(a, b)}$, the more the pixel $P_{(a, b)}$ contributes to the new value of $P_{(i, j)}$. If a coefficient $K_{(a, b)}$ is null, the neighbor $P_{(a, b)}$ does not contribute to the new value of $P_{(i, j)}$ (notice that $P_{(a, b)}$ might be $P_{(i, j)}$ itself).

If the convolution kernel is

$$\begin{matrix} 0 & 0 & 0 \\ -2 & 1 & 2 \\ 0 & 0 & 0 \end{matrix}$$

then

$$P_{(i,j)} = (-2P_{(i-1,j)} + P_{(i,j)} + 2P_{(i+1,j)}).$$

If the convolution kernel is

$$\begin{matrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{matrix}$$

then

$$P_{(i,j)} = (P_{(i,j-1)} + P_{(i-1,j)} + P_{(i+1,j)} + P_{(i,j+1)}).$$

If the kernel contains both negative and positive coefficients, the transfer function is equivalent to a weighted differentiation, and produces a sharpening or highpass filter. Typical highpass filters include gradient and Laplacian filters.

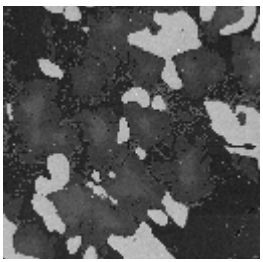
If all coefficients in the kernel are positive, the transfer function is equivalent to a weighted summation and produces a smoothing or lowpass filter. Typical lowpass filters include smoothing and Gaussian filters.

Gradient Filter

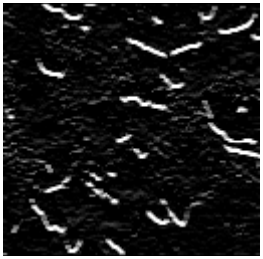
A *gradient filter* highlights the variations of light intensity along a specific direction, which has the effect of outlining edges and revealing texture.

Example

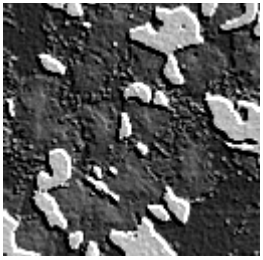
This example uses the following source image.



A gradient filter extracts horizontal edges to produce the following image.



A gradient filter highlights diagonal edges to produce the following image.



Kernel Definition

A *gradient convolution filter* is a first order derivative and its kernel uses the following model:

$$\begin{matrix} a & -b & c \\ b & x & -d \\ c & d & -a \end{matrix}$$

where a, b, and c are integers and $x = 0$ or 1.

This kernel has an axis of symmetry that runs between the positive and negative coefficients of the kernel and through the central element. This axis of symmetry gives the orientation of the edges to outline.

Filter Axis and Direction

The *axis of symmetry* of the gradient kernel gives the orientation of the edges to outline. For example:

where $a = 0$, $b = -1$, $c = -1$, $d = -1$, and $x = 0$, the kernel is the following:

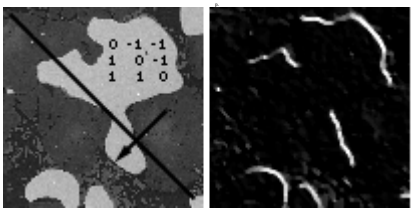
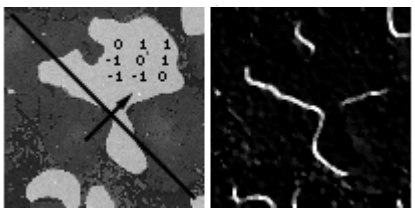
$$\begin{matrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{matrix}$$

The axis of symmetry is at 135 degrees.

For a given direction, you can design a gradient filter to highlight or darken the edges along that direction. The filter actually is sensitive to the variations of intensity perpendicular to the axis of symmetry of its kernel. Given the direction *D* going from the negative coefficients of the kernel towards the positive coefficients, the filter highlights the pixels where the light intensity increases along the direction *D*, and darkens the pixels where the light intensity decreases.

Examples

The following two kernels emphasize edges oriented at 135 degrees.

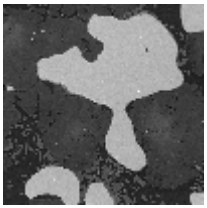
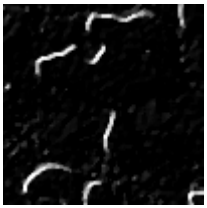
Gradient #1	Gradient #2
0 –1 –1 1 0 –1 1 1 0	0 1 1 –1 0 1 –1 –1 0
Gradient #1 highlights pixels where the light intensity increases along the direction going from northeast to southwest. It darkens pixels where the light intensity decreases along that same direction. This processing outlines the northeast front edges of bright regions such as the ones in the illustration.	Gradient #2 highlights pixels where the light intensity increases along the direction going from southwest to northeast. It darkens pixels where the light intensity decreases along that same direction. This processing outlines the southwest front edges of bright regions such as the ones in the illustration.
	

Note ***Applying Gradient #1 to an image gives the same results as applying Gradient #2 to its photometric negative, because reversing the lookup table of an image converts bright regions into dark regions and vice versa.***

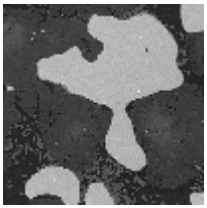
Edge Extraction and Edge Highlighting

The gradient filter has two effects, depending on whether the central coefficient x is equal to 1 or 0:

- If the central coefficient is null ($x = 0$), the gradient filter highlights the pixels where variations of light intensity occur along a direction specified by the configuration of the coefficients a, b, c, and d. The transformed image contains black-white borders at the original edges and the shades of the overall patterns are darkened.

Source Image	Gradient #1	Filtered Image
	$\begin{matrix} -1 & -1 & 0 \\ -1 & \mathbf{0} & 1 \\ 0 & 1 & 1 \end{matrix}$	

- If the central coefficient is equal to 1 ($x = 1$), the gradient filter detects the same variations as mentioned above, but superimposes them over the source image. The transformed image looks like the source image with edges highlighted. You can use this type of kernel for grain extraction and perception of texture.

Source Image	Gradient #2	Filtered Image
	$\begin{matrix} -1 & -1 & 0 \\ -1 & \mathbf{1} & 1 \\ 0 & 1 & 1 \end{matrix}$	

Notice that the kernel Gradient #2 can be decomposed as follows:

$$\begin{matrix} -1 & -1 & 0 \\ -1 & \mathbf{1} & 1 \\ 0 & 1 & 1 \end{matrix} \quad = \quad \begin{matrix} -1 & -1 & 0 \\ -1 & \mathbf{0} & 1 \\ 0 & 1 & 1 \end{matrix} \quad + \quad \begin{matrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{matrix}$$

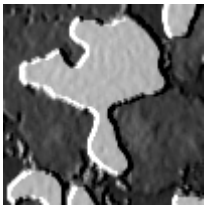
Note ***The convolution filter using the second kernel on the right side of the equation reproduces the source image. All neighboring pixels are multiplied by 0 and the central pixel remains equal to itself: ($P_{(i,j)} = 1 \times P_{(i,j)}$).***

This equation indicates that Gradient #2 adds the edges extracted by the Gradient #1 to the source image.

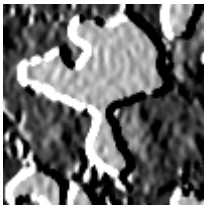
Gradient #2 = Gradient #1 + Source Image

Edge Thickness

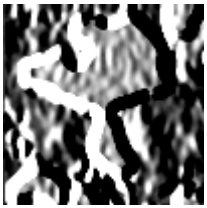
The larger the kernel, the larger the edges. The following image illustrates gradient west–east 3 × 3.



The following image illustrates gradient west–east 5 × 5.



Finally, the following image illustrates gradient west–east 7 × 7.



Predefined Gradient Kernels

The tables in this section list the predefined gradient kernels.

Prewitt Filters

The *Prewitt filters* have the following kernels. The notations West (W), South (S), East (E), and North (N) indicate which edges of bright regions they outline.

Table 13-1. Prewitt Filters

W/Edge	W/Image	SW/Edge	SW/Image
–1 0 1 –1 0 1 –1 0 1	–1 0 1 –1 1 1 –1 0 1	0 1 1 –1 0 1 –1 –1 0	0 1 1 –1 1 1 –1 –1 0
S/Edge	**S/Image**	**SE/Edge**	**SE/Image**
1 1 1 0 0 0 –1 –1 –1	1 1 1 0 1 0 –1 –1 –1	1 1 0 1 0 –1 0 –1 –1	1 1 0 1 1 –1 0 –1 –1
E/Edge	**E/Image**	**NE/Edge**	**NE/Image**
1 0 –1 1 0 –1 1 0 –1	1 0 –1 1 1 –1 1 0 –1	0 –1 –1 1 0 –1 1 1 0	0 –1 –1 1 1 –1 1 1 0
N/Edge	**N/Image**	**NW/Edge**	**NW/Image**
–1 –1 –1 0 0 0 1 1 1	–1 –1 –1 0 1 0 1 1 1	–1 –1 0 –1 0 1 0 1 1	–1 –1 0 –1 1 1 0 1 1

Sobel Filters

The *Sobel filters* are very similar to the Prewitt filters except that they highlight light intensity variations along a particular axis that is assigned a stronger weight. The Sobel filters have the following kernels. The notations West (W), South (S), East (E), and North (N) indicate which edges of bright regions they outline.

Table 13-2. Sobel Filters

W/Edge	W/Image	SW/Edge	SW/Image
–1 0 1	–1 0 1	0 1 2	0 1 2
–2 0 2	–2 1 2	–1 0 1	–1 1 1
–1 0 1	–1 0 1	–2 –1 0	–2 –1 0
S/Edge	**S/Image**	**SE/Edge**	**SE/Image**
1 2 1	1 2 1	2 1 0	2 1 0
0 0 0	0 1 0	1 0 –1	1 1 –1
–1 –2 –1	–1 –2 –1	0 –1 –2	0 –1 –2
E/Edge	**E/Image**	**NE/Edge**	**NE/Image**
1 0 –1	1 0 –1	0 –1 –2	0 –1 –2
2 0 –2	2 1 –2	1 0 –1	1 1 –1
1 0 –1	1 0 –1	2 1 0	2 1 0
N/Edge	**N/Image**	**NW/Edge**	**NW/Image**
–1 –2 –1	–1 –2 –1	–2 –1 0	–2 –1 0
0 0 0	0 1 0	–1 0 1	–1 1 1
1 2 1	1 2 1	0 1 2	0 1 2

The following tables list the predefined gradient 5 × 5 and 7 × 7 kernels.

Table 13-3. Gradient 5 × 5

W/Edge					W/Image				
0	–1	0	1	0	0	–1	0	1	0
–1	–2	0	2	1	–1	–2	0	2	1
–1	–2	0	2	1	–1	–2	1	2	1
–1	–2	0	2	1	–1	–2	0	2	1
0	–1	0	1	0	0	–1	0	1	0

Table 13-4. Gradient 7 × 7

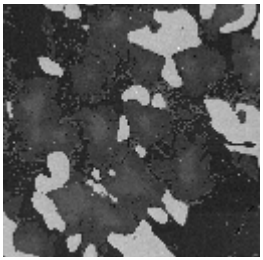
W/Edge							W/Image						
0	–1	–1	0	1	1	0	0	–1	–1	0	1	1	0
–1	–2	–2	0	2	2	1	–1	–2	–2	0	2	2	1
–1	–2	–3	0	3	2	1	–1	–2	–3	0	3	2	1
–1	–2	–3	0	3	2	1	–1	–2	–3	1	3	2	1
–1	–2	–3	0	3	2	1	–1	–2	–3	0	3	2	1
–1	–2	–3	0	3	2	1	–1	–2	–3	0	3	2	1
0	–1	–1	0	1	1	0	0	–1	–1	0	1	1	0

Laplacian Filters

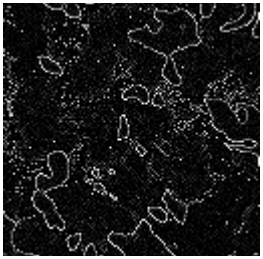
A *Laplacian filter* highlights the variation of the light intensity surrounding a pixel. The filter extracts the contour of objects and outlines details. Unlike the gradient filter, it is omni-directional.

Example

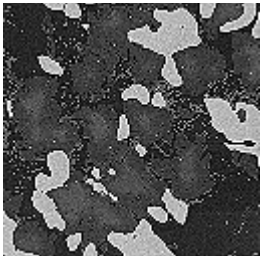
This example uses the following source image.



A Laplacian filter extracts contours to produce the following image.



A Laplacian filter highlights contours to produce the following image.



Kernel Definition

The *Laplacian convolution filter* is a second order derivative and its kernel uses the following model:

$$\begin{matrix} a & d & c \\ b & x & b \\ c & d & a \end{matrix}$$

where a, b, c, and d are integers.

The Laplacian filter has two different effects, depending on whether the central coefficient x is equal to or greater than the sum of the absolute values of the outer coefficients:

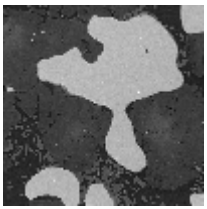
$$x \geq 2(|a| + |b| + |c| + |d|) .$$

Contour Extraction and Highlighting

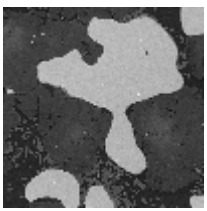
If the central coefficient is equal to this sum $(x = 2(|a| + |b| + |c| + |d|))$, the Laplacian filter extracts the pixels where significant variations of light intensity are found. The presence of sharp edges, boundaries between objects, modification in the texture of a background, noise, and other effects can cause these variations. The transformed image contains white contours on a black background.

Examples

Notice the following source image, Laplacian kernel, and filtered image.

Source Image	Laplacian #1	Filtered Image
	–1 –1 –1 –1 **8** –1 –1 –1 –1	

If the central coefficient is greater than the sum of the outer coefficients ($x > 2(a + b + c + d)$), the Laplacian filter detects the same variations as mentioned above, but superimposes them over the source image. The transformed image looks like the source image, with all significant variations of the light intensity highlighted.

Source Image	Laplacian #2	Filtered Image
	–1 –1 –1 –1 **9** –1 –1 –1 –1	

Notice that the Laplacian #2 kernel can be decomposed as follows:

$$\begin{matrix} -1 & -1 & -1 \\ -1 & \mathbf{9} & -1 \\ -1 & 1 & -1 \end{matrix} \quad = \quad \begin{matrix} -1 & -1 & -1 \\ -1 & \mathbf{8} & -1 \\ -1 & -1 & -1 \end{matrix} \quad + \quad \begin{matrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{matrix}$$

Note ***The convolution filter using the second kernel on the right side of the equation reproduces the source image. All neighboring pixels are multiplied by 0 and the central pixel remains equal to itself: ($P_{(i, j)} = 1 \times P_{(i, j)}$).***

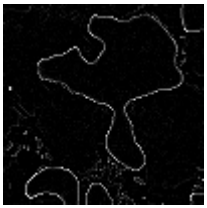
This equation indicates that the Laplacian #2 kernel adds the contours extracted by the Laplacian #1 kernel to the source image.

Laplacian #2 = Laplacian #1 + Source Image

For example, if the central coefficient of Laplacian #2 kernel is 10, the Laplacian filter adds the contours extracted by Laplacian #1 kernel to the source image times 2, and so forth. A greater central coefficient corresponds to less-prominent contours and details highlighted by the filter.

Contour Thickness

Larger kernels correspond to larger contours. The following image is a Laplacian 3 × 3.



The following image is a Laplacian 5 × 5.

The following image is a Laplacian 7 × 7.

Predefined Laplacian Kernels

The following tables list the predefined Laplacian kernels.

Table 13-5. Laplacian 3 × 3

Contour 4	**+ Image × 1**	**+ Image × 2**
0 –1 0	0 –1 0	0 –1 0
–1 4 –1	–1 5 –1	–1 6 –1
0 –1 0	0 –1 0	0 –1 0
Contour 8	**+ Image × 1**	**+ Image × 2**
–1 –1 –1	–1 –1 –1	–1 –1 –1
–1 8 –1	–1 9 –1	–1 10 –1
–1 –1 –1	–1 –1 –1	–1 –1 –1
Contour 12	**+ Image × 1**	
–1 –2 –1	–1 –2 –1	
–2 12 –2	–2 13 –2	
–1 –2 –1	–1 –2 –1	

Table 13-6. Laplacian 5 × 5

Contour 24	**+ Image × 1**
–1 –1 –1 –1 –1	–1 –1 –1 –1 –1
–1 –1 –1 –1 –1	–1 –1 –1 –1 –1
–1 –1 24 –1 –1	–1 –1 25 –1 –1
–1 –1 –1 –1 –1	–1 –1 –1 –1 –1
–1 –1 –1 –1 –1	–1 –1 –1 –1 –1

Table 13-7. Laplacian 7 × 7

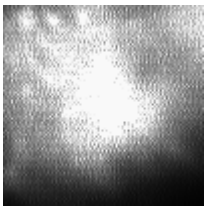
Contour 48							+ Image × 1						
–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1
–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1
–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1
–1	–1	–1	48	–1	–1	–1	–1	–1	–1	49	–1	–1	–1
–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1
–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1
–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1	–1

Smoothing Filter

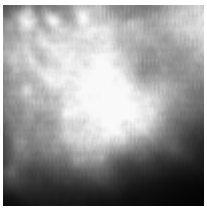
A *smoothing filter* attenuates the variations of light intensity in the neighborhood of a pixel. It smoothes the overall shape of objects, blurs edges, and removes details.

Example

This example uses the following source image.



A smoothing filter produces the following image.



Kernel Definition

A *smoothing convolution filter* is an averaging filter and its kernel uses the following model:

$$\begin{matrix} a & d & c \\ b & x & b \\ c & d & a \end{matrix}$$

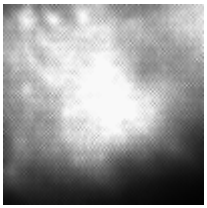
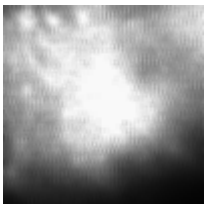
where a, b, c, and d are integers and $x = 0$ or 1.

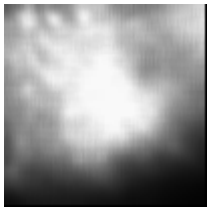
Because all the coefficients in a smoothing kernel are positive, each central pixel becomes a weighted average of its neighbors. The stronger the weight of a neighboring pixel, the more influence it has on the new value of the central pixel.

For a given set of coefficients (a, b, c, d), a smoothing kernel with a central coefficient equal to 0 ($x = 0$) has a stronger blurring effect than a smoothing kernel with a central coefficient equal to 1 ($x = 1$).

Examples

Notice the following smoothing kernels and filtered images. A larger kernel size corresponds to a stronger smoothing effect.

Kernel #1	**Filtered Image**
$\begin{matrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{matrix}$	
Kernel #2	**Filtered Image**
$\begin{matrix} 2 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 2 \end{matrix}$	

Kernel #3	Filtered Image
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
Kernel #4	**Filtered Image**
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	

Predefined Smoothing Kernels

The following tables list the predefined smoothing kernels.

Table 13-8. Smoothing 3 × 3

0	1	0	0	1	0	0	2	0	0	4	0
1	0	1	1	1	1	2	1	2	4	1	4
0	1	0	0	1	0	0	2	0	0	4	0
1	1	1	1	1	1	2	2	2	4	4	4
1	0	1	1	1	1	2	1	2	4	1	4
1	1	1	1	1	1	2	2	2	4	4	4

Table 13-9. Smoothing 5 × 5

1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

Table 13-10. Smoothing 7 × 7

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

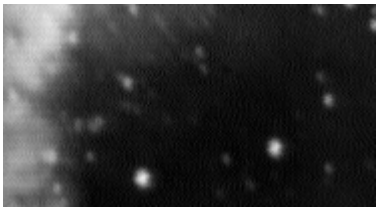
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

Gaussian Filters

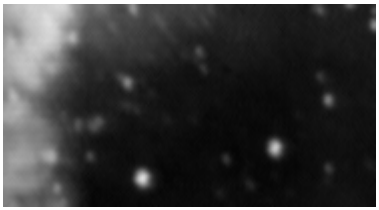
A *Gaussian filter* attenuates the variations of light intensity in the neighborhood of a pixel. It smoothes the overall shape of objects and attenuates details. It is similar to a smoothing filter, but its blurring effect is more subdued.

Example

This example uses the following source image.



A Gaussian filter produces the following image.



Kernel Definition

A *Gaussian convolution filter* is an averaging filter and its kernel uses the following model:

$$\begin{matrix} a & d & c \\ b & x & b \\ c & d & a \end{matrix}$$

where a, b, c, and d are integers and $x > 1$.

Since all the coefficients in a Gaussian kernel are positive, each pixel becomes a weighted average of its neighbors. The stronger the weight of a neighboring pixel, the more influence it has on the new value of the central pixel.

Unlike a smoothing kernel, the central coefficient of a Gaussian filter is greater than 1. Therefore the original value of a pixel is multiplied by a weight greater than the weight of any of its neighbors. As a result, a greater central coefficient corresponds to a more subtle smoothing effect. A larger kernel size corresponds to a stronger smoothing effect.

Predefined Gaussian Kernels

The following tables list the predefined Gaussian kernels.

Table 13-11. Gaussian 3 × 3

0	1	0
1	2	1
0	1	0

0	1	0
1	4	1
0	1	0

1	1	1
1	2	1
1	1	1

1	1	1
1	4	1
1	1	1

1	2	1
2	4	2
1	2	1

1	4	1
4	16	4
1	4	1

Table 13-12. Gaussian 5 × 5

1	2	4	2	1
2	4	8	4	2
4	8	16	8	4
2	4	8	4	2
1	2	4	2	1

Table 13-13. Gaussian 7 × 7

1	1	2	2	2	1	1
1	2	2	4	2	2	1
2	2	4	8	4	2	2
2	4	8	16	8	4	2
2	2	4	8	4	2	2
1	2	2	4	2	2	1
1	1	2	2	2	1	1

Nonlinear Filters

A *nonlinear filter* replaces each pixel value with a nonlinear function of its surrounding pixels. Like the convolution filters, the nonlinear filters operate on a neighborhood. The following notations describe the behavior of the nonlinear spatial filters.

If $P_{(i, j)}$ represents the intensity of the pixel P with the coordinates (i, j), the pixels surrounding $P_{(i, j)}$ can be indexed as follows (in the case of a 3 × 3 matrix):

$P_{(i-1, j-1)}$	$P_{(i, j-1)}$	$P_{(i+1, j-1)}$
$P_{(i-1, j)}$	$P_{(i, j)}$	$P_{(i+1, j)}$
$P_{(i-1, j+1)}$	$P_{(i, j+1)}$	$P_{(i+1, j+1)}$

In the case of a 5 × 5 neighborhood, the i and j indexes vary from –2 to 2, and so forth. The series of pixels including $P_{(i, j)}$ and its surrounding pixels is annotated as $P_{(n, m)}$.

Nonlinear Prewitt Filter

The *nonlinear Prewitt filter* is a highpass filter that extracts the outer contours of objects. It highlights significant variations of the light intensity along the vertical and horizontal axes.

Each pixel is assigned the maximum value of its horizontal and vertical gradient obtained with the following Prewitt convolution kernels:

Kernel #1			**Kernel #2**		
–1	0	1	–1	–1	–1
–1	0	1	0	0	0
–1	0	1	1	1	1

$$P_{(i,j)} = \max[|P_{(i+1,j-1)} - P_{(i-1,j-1)} + P_{(i+1,j)} - P_{(i-1,j)} + P_{(i+1,j+1)} - P_{(i-1,j+1)}|, \\ |P_{(i-1,j+1)} - P_{(i-1,j-1)} + P_{(i,j+1)} - P_{(i,j-1)} + P_{(i+1,j+1)} - P_{(i+1,j-1)}|]$$

Nonlinear Sobel Filter

The *nonlinear Sobel filter* is a highpass filter that extracts the outer contours of objects. It highlights significant variations of the light intensity along the vertical and horizontal axes.

Each pixel is assigned the maximum value of its horizontal and vertical gradient obtained with the following Sobel convolution kernels:

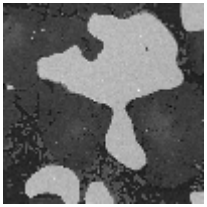
Kernel #1			**Kernel #2**		
–1	0	1	–1	–2	–1
–2	0	2	0	0	0
–1	0	1	1	2	1

As opposed to the Prewitt filter, the Sobel filter assigns a higher weight to the horizontal and vertical neighbors of the central pixel:

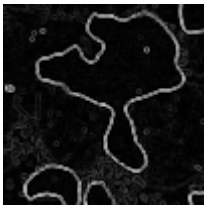
$$P_{(i,j)} = \max[|P_{(i+1,j-1)} - P_{(i-1,j-1)} + \mathbf{2}P_{(i+1,j)} - \mathbf{2}P_{(i-1,j)} + P_{(i+1,j+1)} - P_{(i-1,j+1)}|, \\ |P_{(i-1,j+1)} - P_{(i-1,j-1)} + \mathbf{2}P_{(i,j+1)} - \mathbf{2}P_{(i,j-1)} + P_{(i+1,j+1)} - P_{(i+1,j-1)}|]$$

Example

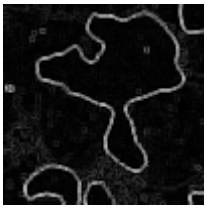
This example uses the following source image.



A nonlinear Prewitt filter produces the following image.



A nonlinear Sobel filter produces the following image.



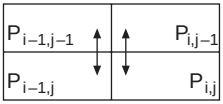
Both filters outline the contours of the objects. Because of the different convolution kernels they combine, the nonlinear Prewitt has the tendency to outline curved contours while the nonlinear Sobel extracts square contours. This difference is noticeable when observing the outlines of isolated pixels.

Nonlinear Gradient Filter

The *nonlinear gradient filter* outlines contours where an intensity variation occurs along the vertical axis.

The new value of a pixel becomes the maximum absolute value between its deviation from the upper neighbor and the deviation of its two left neighbors.

$$P_{(i,j)} = \max[|P_{(i,j-1)} - P_{(i,j)}|, |P_{(i-1,j-1)} - P_{(i-1,j)}|]$$

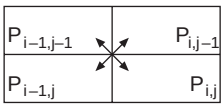


Roberts Filter

The *Roberts filter* outlines the contours that highlight pixels where an intensity variation occurs along the diagonal axes.

The new value of a pixel becomes the maximum absolute value between the deviation of its upper-left neighbor and the deviation of its two other neighbors.

$$P_{(i,j)} = \max[|P_{(i-1,j-1)} - P_{(i,j)}|, |P_{(i,j-1)} - P_{(i-1,j)}|]$$

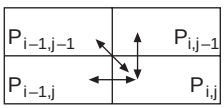


Differentiation Filter

The *differentiation filter* produces continuous contours by highlighting each pixel where an intensity variation occurs between itself and its three upper-left neighbors.

The new value of a pixel becomes the absolute value of its maximum deviation from its upper-left neighbors.

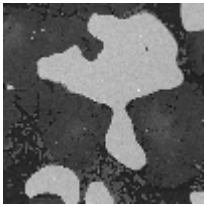
$$P_{(i,j)} = \max[|P_{(i-1,j)} - P_{(i,j)}|, |P_{(i-1,j-1)} - P_{(i,j)}|, |P_{(i,j-1)} - P_{(i,j)}|]$$



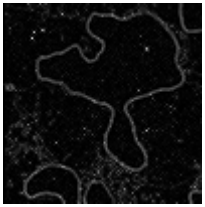
Sigma Filter

The *Sigma filter* is a highpass filter. It outlines contours and details by setting pixels to the mean value found in their neighborhood, if their deviation from this value is not significant.

Given M, the mean value of $P_{(i,j)}$ and its neighbors and S, their standard deviation, each pixel $P_{(i,j)}$ is set to the mean value M if it falls inside the range $[M - S, M + S]$.



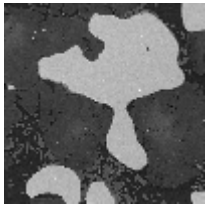
$$\begin{aligned} \textit{If}\ & P_{(i,j)} - M > S, \\ \textit{then}\ & P_{(i,j)} = P_{(i,j)}, \\ \textit{else}\ & P_{(i,j)} = M. \end{aligned}$$



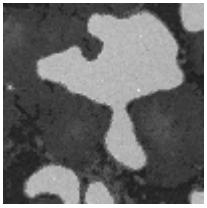
Lowpass Filter

The *lowpass filter* reduces details and blurs edges by setting pixels to the mean value found in their neighborhood, if their deviation from this value is large.

Given M, the mean value of $P_{(i,j)}$ and its neighbors and S, their standard deviation, each pixel $P_{(i,j)}$ is set to the mean value M if it falls outside the range $[M - S, M + S]$.



$$\begin{aligned} \textit{If}\ & P_{(i,j)} - M < S, \\ \textit{then}\ & P_{(i,j)} = P_{(i,j)}, \\ \textit{else}\ & P_{(i,j)} = M. \end{aligned}$$



Median Filter

The *median filter* is a lowpass filter. It assigns to each pixel the median value of its neighborhood, effectively removing isolated pixels and reducing details. However, the median filter does not blur the contour of objects.

$$P_{(i,\,j)} = \text{median value of the series } [P_{(n,\,m)}].$$

Nth Order Filter

The *Nth order filter* is an extension of the median filter. It assigns to each pixel the *N*th value of its neighborhood (when sorted in increasing order). The value *N* specifies the order of the filter, which you can use to moderate the effect of the filter on the overall light intensity of the image. A lower order corresponds to a darker transformed image; a higher order corresponds to a brighter transformed image.

Each pixel is assigned the *N*th value of its neighborhood, *N* being specified by the user.

$$P_{(i,\,j)} = N\text{th value in the series } [P_{(n,\,m)}],$$

where the $P_{(n,\,m)}$ are sorted in increasing order.

The following example uses a 3 × 3 neighborhood:

$P_{(i-1,\,j-1)}$	$P_{(i,\,j-1)}$	$P_{(i+1,\,j-1)}$
$P_{(i-1,\,j)}$	$P_{(i,\,j)}$	$P_{(i+1,\,j)}$
$P_{(i-1,\,j+1)}$	$P_{(i,\,j+1)}$	$P_{(i+1,\,j+1)}$

=

13	10	9
12	4	8
5	5	6

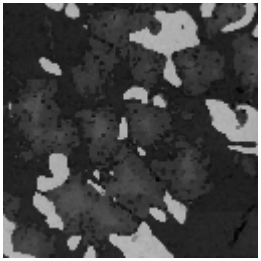
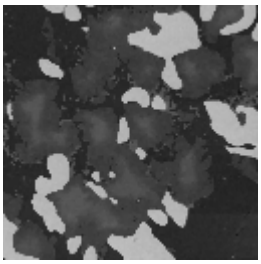
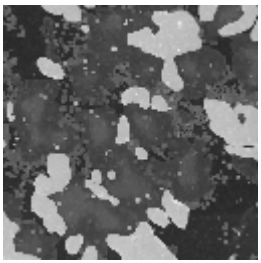
The following table shows the new output value of the central pixel for each *N*th order value:

*N*th Order	0	1	2	3	4	5	6	7	8
New Pixel Value	4	5	5	6	8	9	10	12	13

Note that for a given filter size *f*, the *N*th order can rank from 0 to $f^2 - 1$. For example, in the case of a filter size 3, the *N*th order ranges from 0 to 8 ($3^2 - 1$).

Examples

To see the effect of the Nth order filter, notice the example of an image with bright objects and a dark background. When viewing this image with the B&W (or Gray) palette, the objects have higher gray-level values than the background.

For a Given Filter Size $f \times f$	Example of a Filter Size 3 × 3	
• If $N < (f^2 - 1)/2$, the Nth order filter has the tendency to erode bright regions (or dilate dark regions). • If $N = 0$, each pixel is replaced by its local minimum.	Order 0 (smoothes image, erodes bright objects)	
• If $N = (f^2 - 1)/2$, each pixel is replaced by its local median value. Dark pixels isolated in objects are removed, as well as bright pixels isolated in the background. The overall area of the background and object regions does not change.	Order 4 (equivalent to a median filter)	
• If $N > (f^2 - 1)/2$, the Nth order filter has the tendency to dilate bright regions (or erode dark regions). • If $N = f^2 - 1$, each pixel is replaced by its local maximum.	Order 8 (smoothes image, dilates bright objects)	

14

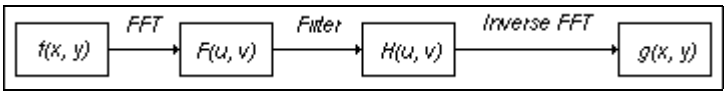
Frequency Filtering

This chapter describes the frequency filters used in IMAQ Vision.

Introduction to Frequency Filters

Frequency filters alter pixel values with respect to the periodicity and spatial distribution of the variations in light intensity in the image. Highpass frequency filters help isolate abruptly varying patterns which correspond to sharp edges, details, and noise. Lowpass frequency filters help emphasize gradually varying patterns such as objects and the background. Frequency filters do not apply directly to a spatial image, but to its frequency representation. The latter is obtained through a function called the *Fast Fourier Transform* (FFT). It reveals information about the periodicity and dispersion of the patterns found in the source image.

The spatial frequencies seen in an FFT image can be filtered and the Inverse FFT then restores a spatial representation of the filtered FFT image.

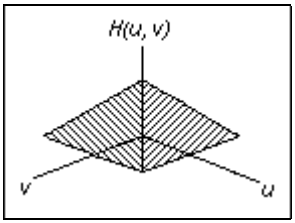


In an image, details and sharp edges are associated to high spatial frequencies because they introduce significant gray-level variations over short distances. Gradually varying patterns are associated to low spatial frequencies.

For example, an image can have extraneous noise such as periodic stripes introduced during the digitization process. In the frequency domain, the periodic pattern is reduced to a limited set of high spatial frequencies. Truncating these particular frequencies and converting the filtered FFT image back to the spatial domain produces a new image in which the grid pattern has disappeared, yet the overall features remain.

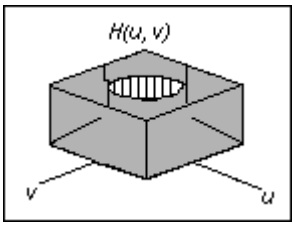
Lowpass FFT Filters

A *lowpass FFT filter* attenuates or removes high frequencies present in the FFT plane. It has the effect of suppressing information related to rapid variations of light intensities in the spatial image. In this case, the Inverse FFT command produces an image in which noise, details, texture, and sharp edges are smoothed.



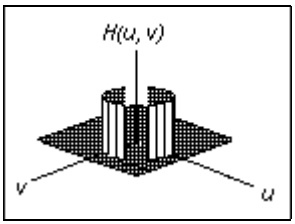
Highpass FFT Filters

A *highpass FFT filter* attenuates or removes low frequencies present in the FFT plane. It has the effect of suppressing information related to slow variations of light intensities in the spatial image. In this case, the Inverse FFT command produces an image in which overall patterns are attenuated and details are emphasized.



Mask FFT Filters

A *mask filter* removes frequencies contained in a mask specified by the user. Depending on the mask definition, this filter may behave as a lowpass, bandpass, highpass, or any type of selective filter.



Definition

The spatial frequencies of an image are calculated by a function called the *Fourier Transform*. It is defined in the continuous domain as:

$$F(u, v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)e^{-j2\pi(xu + yv)}dxdy$$

where $f(x, y)$ is the light intensity of the point (x, y), and (u, v) are the horizontal and vertical spatial frequencies. The Fourier Transform assigns a complex number to each set (u, v).

Inversely, a Fast Fourier Transform $F(u, v)$ can be transformed into a spatial image $f(x, y)$ using the following formula:

$$f(x, y) = \sum_{u=0}^{N-1}\sum_{v=0}^{M-1} F(u, v)e^{j2\pi\left(\frac{ux}{N} + \frac{vy}{M}\right)}$$

In the discrete domain, the Fourier Transform is calculated with an efficient algorithm called the Fast Fourier Transform (FFT). This algorithm requires that the resolution of the image be $2^n \times 2^m$. Notice that the values n and m can be different, which indicates that the image does not have to be square.

$$F(u, v) = \frac{1}{NM}\sum_{x=0}^{N-1}\sum_{y=0}^{M-1} f(x, y)e^{-j2\pi\left(\frac{ux}{N} + \frac{vy}{M}\right)}$$

where NM is the resolution of the spatial image $f(x, y)$.

Because $e^{-j2\pi ux} = \cos 2\pi ux - j\sin 2\pi ux$, $F(u, v)$ is composed of an infinite sum of sine and cosine terms. Each pair (u, v) determines the frequency of its corresponding sine and cosine pair. For a given set (u, v), note that all values $f(x, y)$ contribute to $F(u, v)$. Because of this complexity, the FFT calculation is time consuming.

The relation between the sampling increments in the spatial domain $(\Delta x, \Delta y)$ and the frequency domain $(\Delta u, \Delta v)$ is:

$$\Delta u = \frac{1}{N \times \Delta x} \qquad \Delta v = \frac{1}{M \times \Delta y}$$

The FFT of an image, $F(u, v)$, is a two dimensional array of complex numbers, or a complex image. It represents the frequencies of occurrence of light intensity variations in the spatial domain. The low frequencies (u, v) correspond to smooth and gradual intensity variations found in the overall patterns of the source image. The high frequencies (u, v) correspond to abrupt and short intensity variations found at the edges of objects, around noisy pixels, and around details.

FFT Display

An FFT image can be visualized using any of its four complex components: real part, imaginary part, magnitude, and phase. The relation between these components is expressed by

$$F(u, v) = R(u, v) + jI(u, v),$$

where $R(u, v)$ is the real part and $I(u, v)$ is the imaginary part, and

$$F(u, v) = |F(u, v)| \times e^{j\varphi\langle u, v\rangle},$$

where $|F(u, v)|$ is the magnitude and $\varphi(u, v)$ is the phase.

The magnitude of $F(u, v)$ is also called the *Fourier spectrum* and is equal to

$$|F(u, v)| = \sqrt{R(u, v)^2 + I(u, v)^2}$$

The Fourier spectrum to the power of two is known as the power spectrum or spectral density.

The phase $\varphi(u, v)$ is also called the phase angle and is equal to

$$\varphi(u, v) = \operatorname{atan}\left[\frac{I(u, v)}{R(u, v)}\right].$$

Given an image with a resolution *NM* and given Δ x and Δ y the spatial step increments, the FFT of the source image has the same resolution *NM* and its frequency step increments Δ u and Δ v, which are defined in the following equations:

$$\Delta u = \frac{1}{N \times \Delta x} \qquad \Delta v = \frac{1}{M \times \Delta y}.$$

The FFT of an image has the following two properties:

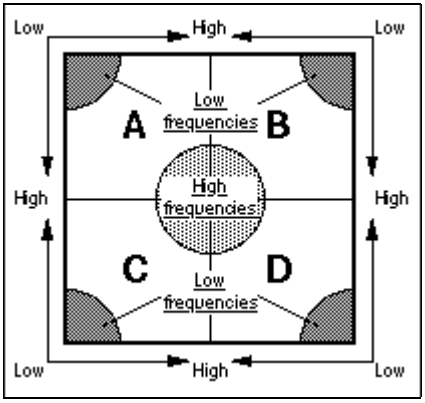
- It is periodic: $F(u, v) = F(u + N, v + M)$
- It is conjugate-symmetric: $F(u, v) = F^*(-u, -v)$

These properties result in two possible representations of the Fast Fourier Transform of an image: the *standard representation* and the *optical representation*.

Standard Representation

High frequencies are grouped at the center while low frequencies are located at the edges. The constant term, or null frequency is in the upper-left corner of the image. The frequency range is

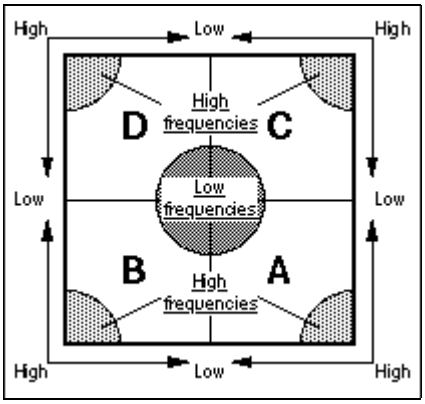
$$0, N\Delta u] \times [0, M\Delta v.$$



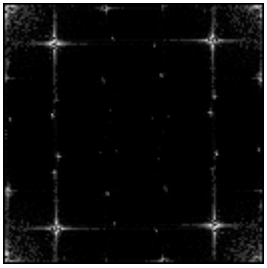
Optical Representation

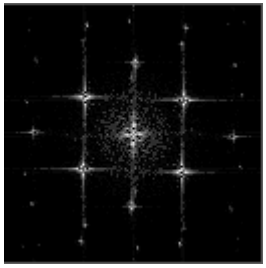
Low frequencies are grouped at the center while high frequencies are located at the edges. The constant term, or null frequency, is at the center of the image. The frequency range is

$$\left[-\frac{N}{2}\Delta u, \frac{N}{2}\Delta u\right] \times \left[-\frac{M}{2}\Delta v, \frac{M}{2}\Delta v\right].$$



You can switch from the standard representation to the optical representation by permuting the *A*, *B*, *C*, and *D* quarters.





Intensities in the FFT image are proportional to the amplitude of the displayed component.

Frequency Filters

This section describes the frequency filters in detail and includes information on lowpass and highpass attenuation and truncation.

Lowpass Frequency Filters

A *lowpass frequency filter* attenuates or removes high frequencies present in the FFT plane. This filter suppresses information related to rapid variations of light intensities in the spatial image. In this case, the Inverse FFT command produces an image in which noise, details, texture, and sharp edges are smoothed.

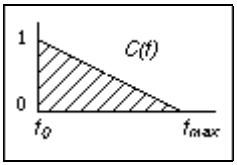
A lowpass frequency filter removes or attenuates spatial frequencies located outside a frequency range centered on the fundamental (or null) frequency.

Lowpass Attenuation

Lowpass attenuation applies a linear attenuation to the full frequency range, decreasing from f_0 to f_{max}. This is done by multiplying each frequency by a coefficient C which is a function of its deviation from the fundamental and maximum frequencies.

$$C(f) = \frac{f_{max} - f}{f_{max} - f_0},$$

where $C(f_0) = 1$ and $C(f_{max}) = 0$.



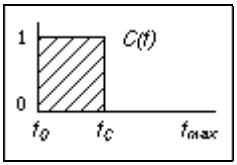
Lowpass Truncation

Lowpass truncation removes a frequency f if it falls outside the truncation range $[f_0, f_c]$. This is done by multiplying each frequency f by a coefficient C equal to 0 or 1, depending on whether the frequency f is greater than the truncation frequency f_c.

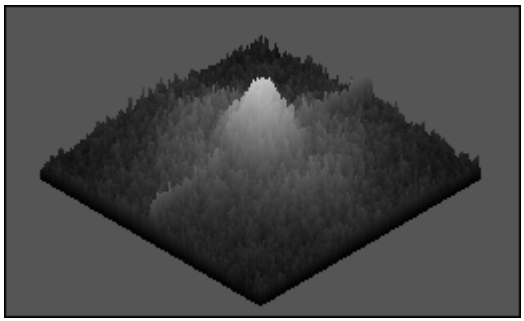
If $f > f_c$,

then $C(f) = 0$

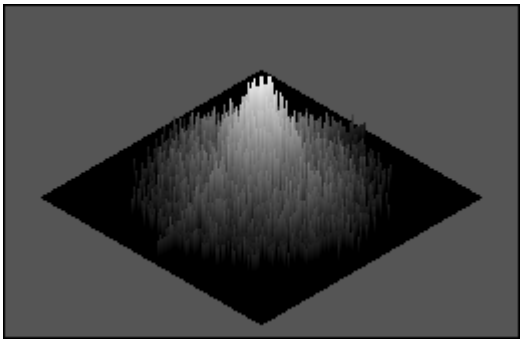
else $C(f) = 1$.



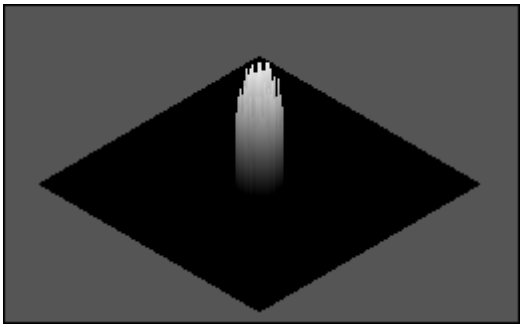
The following series of graphics illustrates the behavior of each type of filter. They give the 3D-view profile of the magnitude of the FFT. This example uses the following original FFT.



After lowpass attenuation, the magnitude of the central peak has been attenuated, and variations at the edges almost have disappeared.



After lowpass truncation with $f_c = f_0 + 20\%(f_{max} - f_0)$, spatial frequencies outside the truncation range [f_0, f_c] are removed. The part of the central peak that remains is identical to the one in the original FFT plane.



Highpass Frequency Filters

A *highpass frequency filter* attenuates or removes low frequencies present in the FFT plane. It has the effect of suppressing information related to slow variations of light intensities in the spatial image. In this case, the inverse FFT produces an image in which overall patterns are attenuated and details are emphasized.

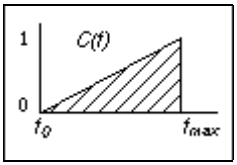
A highpass frequency filter removes or attenuates spatial frequencies located inside a frequency range centered on the fundamental (or null) frequency.

Highpass Attenuation

Highpass attenuation applies a linear attenuation to the full frequency range, decreasing from f_{max} to f_0. This is done by multiplying each frequency by a coefficient C which is a function of its deviation from the fundamental and maximum frequencies.

$$C(f) = \frac{f - f_0}{f_{max} - f_0},$$

where $C(f_0) = 1$ and $C(f_{max}) = 0$.



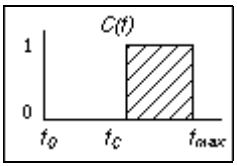
Highpass Truncation

Highpass truncation removes a frequency f if it falls inside the truncation range $[f_0, f_c]$. This is done by multiplying each frequency f by a coefficient C equal to 1 or 0, depending on whether the frequency f is greater than the truncation frequency f_c.

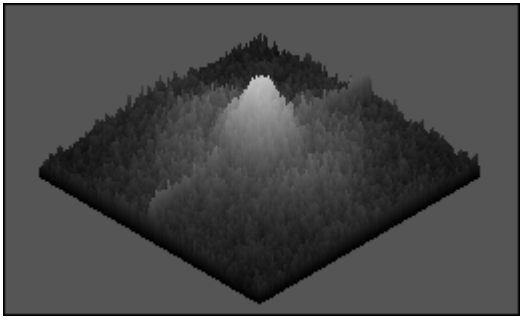
If $f < f_c$,

then $C(f) = 0$

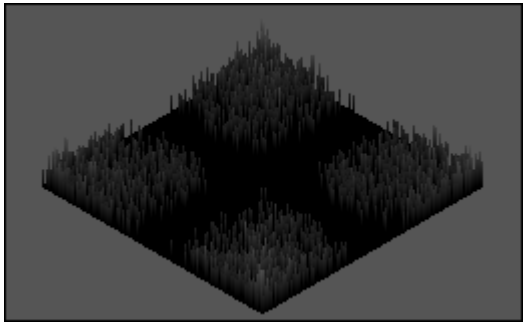
else $C(f) = 1$.



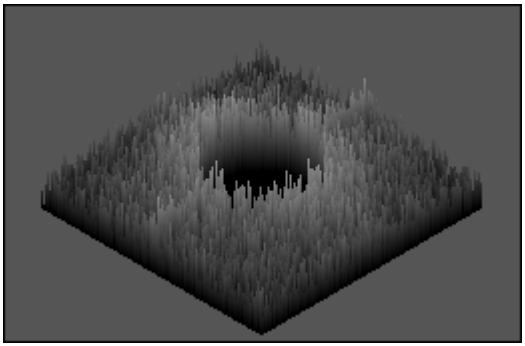
The following series of graphics illustrates the behavior of each type of filter. They give the 3D-view profile of the magnitude of the FFT. This example uses the following original FFT image.



After highpass attenuation, the central peak has been removed and variations present at the edges remain.



After highpass truncation with $f_c = f_0 + 20\%(f_{max} - f_0)$, spatial frequencies inside the truncation range $[f_0, f_c]$ are set to 0. The remaining frequencies are identical to the ones in the original
FFT plane.



15

Morphology Analysis

This chapter provides an overview of morphology image analysis.

Morphological transformations extract and alter the structure of objects in an image. You can use these transformations to prepare objects for quantitative analysis, observe the geometry of regions, extract the simplest forms for modeling and identification purposes, and so forth.

The morphological transformations can be used for expanding or reducing objects, filling holes, closing inclusions, smoothing borders, removing dendrites, and more. They can be divided into two categories:

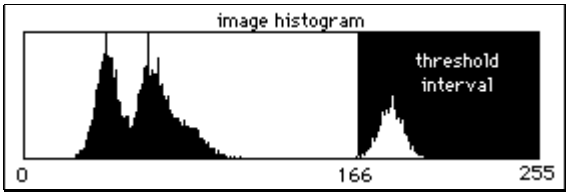
- *Gray-level morphology* functions, which apply to gray-level images
- *Binary Morphology* functions, which apply to binary images

A *binary image* is an image that has been segmented into an object region (pixels equal to 1) and a background region (pixels equal to 0). Such an image is generated by the *thresholding* process.

Thresholding

Thresholding consists of segmenting an image into two regions: an object region and a background region. This is performed by setting to 1 all pixels that belong to a gray-level interval, called the threshold interval. All other pixels in the image are set to 0.

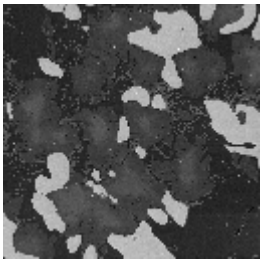
You can use this operation to extract areas that correspond to significant structures in an image and to focus the analysis on these areas.



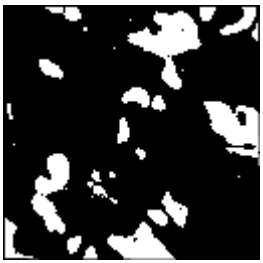
Pixels outside the threshold interval are set to 0 and considered as part of the background area. Pixels inside the threshold interval are set to 1 and considered as part of an object area.

Example

This example uses the following source image.



Highlighting the pixels that belong to the threshold interval [166, 255] (the brightest areas) produces the following image.



A critical and frequent problem in segmenting an image into an object and a background region occurs when the boundaries are not sharply demarcated. In such a case, the choice of a correct threshold becomes subjective. Therefore, it is highly recommended that images be enhanced prior to thresholding, so as to outline where the correct borders lie. Observing the intensity profile of a line crossing a boundary area can also be helpful in selecting a correct threshold value. Finally, keep in mind that morphological transformations can help you retouch the shape of binary objects and therefore correct unsatisfactory selections that occurred during the thresholding.

Thresholding a Color Image

To threshold a color image, three threshold intervals need to be specified, one for each color component. The final binary image is the intersection of the three binary images obtained by thresholding each color component separately.

Automatic Threshold

A number of different automatic thresholding techniques are available, including clustering, entropy, metric, moments, and interclass variance. In contrast to manual thresholding, these methods do not require that the user set the minimal and maximal light intensities. These techniques are well suited for conditions in which the light intensity varies.

Depending on your source image, it is sometimes useful to invert (reverse) the original gray scale image before applying an automatic threshold function (for example, moments and entropy). This is especially true for cases in which the region you want to threshold is black and the background you want to eliminate is red (when viewing with a binary palette).

Clustering is the only multi-class thresholding method available. Clustering operates on multiple classes so you can create tertiary or even higher level images. The other four methods (entropy, metric, moments, and interclass variance) are reserved for strictly binary thresholding techniques. The choice of which algorithm to apply depends on the type of image to threshold.

Clustering

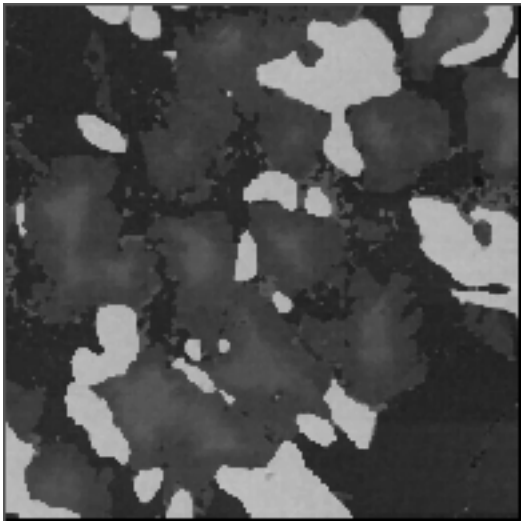
In this rapid technique, the image is randomly sorted within a discrete number of classes corresponding to the number of phases perceived in an image. The gray values are determined and a *barycenter* is determined for each class. This process is repeated until a value is obtained that represents the center of mass for each phase or class.

Example

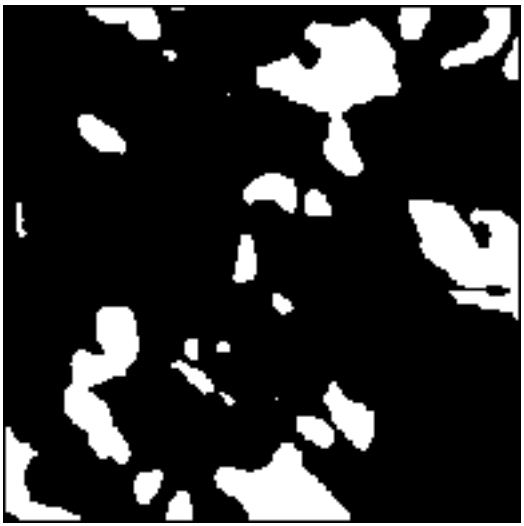
The automatic thresholding method most frequently used is clustering, also known as multi-class thresholding.

This example uses a clustering technique in two and three phases on an image. Note that the results from this function are generally independent of the lighting conditions as well as the histogram values from the image.

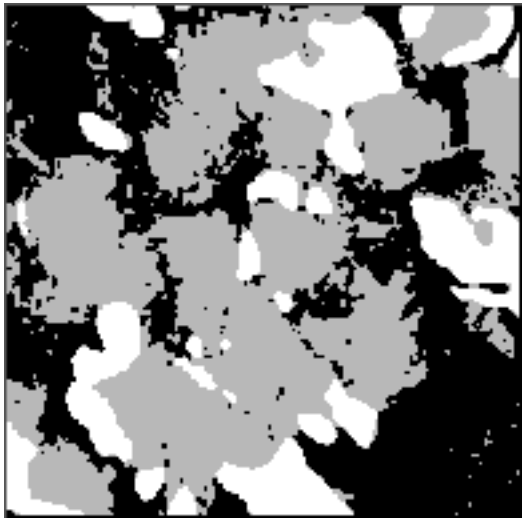
This example uses the following original image.



Clustering in two phases produces the following image.



Clustering in three phases produces the following image.



Entropy

Based on a classical image analysis technique, this method is best suited for detecting objects that are present in minuscule proportions on the image. For example, this function would be suitable for default detection.

Metric

Use this technique in situations similar to interclass variance. For each threshold, a value is calculated that is determined by the surfaces representing the initial gray scale. The optimal threshold corresponds to the smallest value.

Moments

This technique is suited for images that have poor contrast (an overexposed image is better processed than an underexposed image). The moments method is based on the hypothesis that the observed image is a blurred version of the theoretically binary original. The blurring that is produced from the acquisition process (electronic noise or slight defocalization) is treated as if the statistical moments (average and variance) were the same for both the blurred image and the original image. This function recalculates a theoretical binary image.

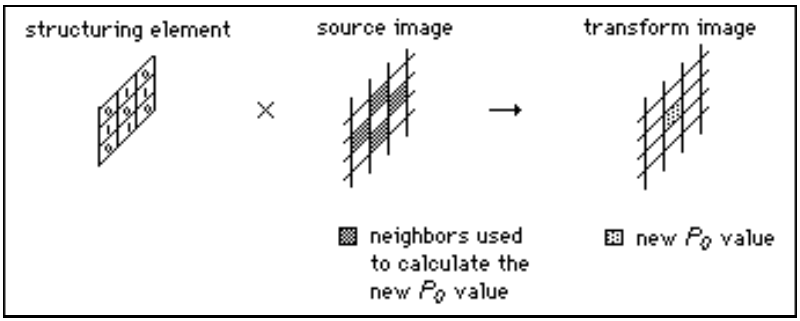
Interclass Variance

Interclass variance is a classical statistic technique used in discriminating factorial analysis. This method is well-suited for images in which classes are not too disproportionate. For satisfactory results, the smallest class must be at least five percent of the largest one. Note that this method has the tendency to underestimate the class of the smallest standard deviation if the two classes have a significant variation.

Structuring Element

A *structuring element* is a binary mask used by most morphological transformations. You can use a structuring element to weigh the effect of these functions on the shape and the boundary of objects.

A morphological transformation using a structuring element alters a pixel P_0 so that it becomes a function of its neighboring pixels. These neighboring pixels are masked by 1 when the structuring element is centered on P_0. A neighbor masked by 0 simply is discarded by the function.



The structuring element is a binary mask (composed of 1 and 0 values). It is used to determine which neighbors of a pixel contribute to its new value. A structuring element can be defined in the case of a rectangular or hexagonal pixel frame, as shown in the following examples.

The following graphic illustrates a morphological transformation using a structuring element. This example uses a 3 × 3 image which has a rectangular frame.

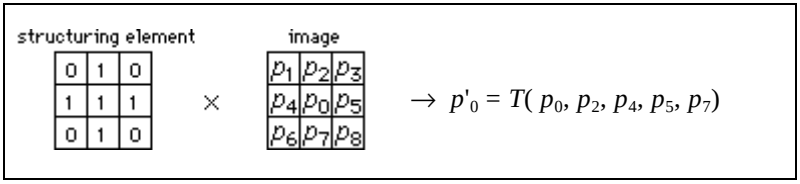


Figure 15-1. Rectangular Frame, Neighborhood 3 × 3

The next graphic illustrates a morphological transformation using a structuring element for an image that has a hexagonal frame. This example uses a 5 × 3 image.

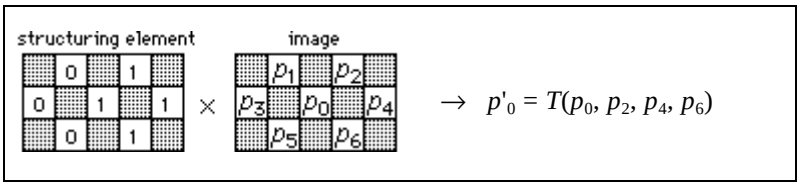


Figure 15-2. Hexagonal Frame, Neighborhood 5 × 3

The default configuration of the structuring element is a 3 × 3 matrix with each coefficient set to 1:

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Primary Binary Morphology Functions

The *primary morphology* functions apply to binary images in which objects have been set to 1 and the background is equal to 0. They include three fundamental binary processing functions: erosion, dilation, and hit-miss. The other transformations derive from combinations of these three functions.

The primary morphology transformations are described in detail in this section of the manual. They include: erosion, dilation, opening, closing, inner gradient, outer gradient, hit-miss, thinning, thickening, proper-opening, proper-closing, and auto-median.

Note ***In the following descriptions, the term pixel denotes a pixel equal to 1 and the term object denotes a group of pixels equal to 1.***

Erosion Function

An *erosion* eliminates pixels isolated in the background and erodes the contour of objects with respect to the template defined by the structuring element.

Concept and Mathematics

For a given pixel P_0, the structuring element is centered on P_0. The pixels masked by a coefficient of the structuring element equal to 1 are then referred as P_i. In the example of a structuring element 3 × 3, the P_i can range from P_0 itself to P_8.

1. If the value of one pixel P_i is equal to 0, *then* P_0 is set to 0, *else* P_0 is set to 1.
2. If AND(P_i) = 1, *then* P_0 = 1, *else* P_0 = 0.

Dilation Function

A *dilation* has the reverse effect of an erosion because dilating objects is equivalent to eroding the background. This function eliminates tiny holes isolated in objects and expands the contour of the objects with respect to the template defined by the structuring element.

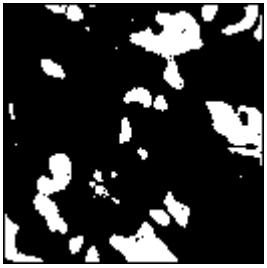
Concept and Mathematics

For a given pixel P_0, the structuring element is centered on P_0. The pixels masked by a coefficient of the structuring element equal to 1 then are referred to as P_i. In the example of a structuring element 3 × 3, the P_i can range from P_0 itself to P_8.

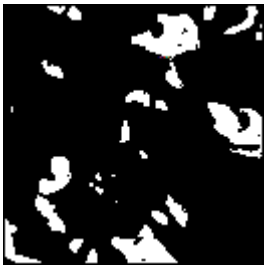
1. If the value of one pixel P_i is equal to 1, *then* P_0 is set to 1, *else* P_0 is set to 0.
2. If OR(P_i) = 1, *then* P_0 = 1, *else* P_0 = 0.

Erosion and Dilation Examples

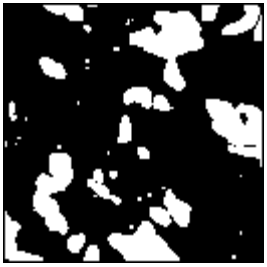
This example uses the following binary source image.



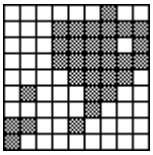
The erosion function produces the following image.



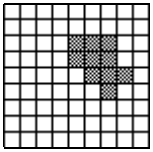
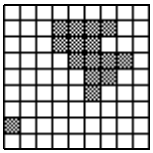
The dilation function produces the following image.

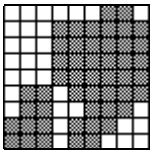
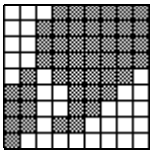


The next example uses the following source image. Gray cells indicate pixels equal to 1.



The following tables show how the structuring element can be used to control the effect of an erosion or a dilation. The larger the structuring element, the more templates can be edited and the more selective the effect.

Structuring Element	After Erosion	Description
		A pixel is cleared if it is equal to 1 and does not have its three upper-left neighbors equal to 1. The erosion truncates the upper-left borders of the objects.
		A pixel is cleared if it is equal to 1 and does not have its lower and right neighbors equal to 1. The erosion truncates the bottom and right borders of the objects, but retains the corners.

Structuring Element	After Dilation	Description
		A pixel is set to 1 if it is equal to 1 or if it has one of its three upper-left neighbors equal to 1. The dilation expands the lower-right borders of the objects.
		A pixel is set to 1 if it is equal to 1 or if it has its lower or right neighbor equal to 1. The dilation expands the upper and left borders of the objects.

Opening Function

The *opening function* is an erosion followed by a dilation. This function removes small objects and smoothes boundaries. If *I* is an image,

$$opening(I) = dilation(erosion(I)).$$

This operation does not alter the area significantly and shape of objects because erosion and dilation are dual transformations. Borders removed by the erosion are restored by the dilation. However, small objects that vanish during the erosion do not reappear after the dilation.

Closing Function

The *closing function* is a dilation followed by an erosion. It fills tiny holes and smoothes boundaries. If I is an image,

$$closing(I) = erosion(dilation(I)).$$

This operation does not alter significantly the area and shape of objects because dilation and erosion are morphological complements. Borders expanded by the dilation function are reduced by the erosion function. However, tiny holes filled during the dilation do not reappear after the erosion.

Opening and Closing Examples

The following series of graphics illustrate examples of openings and closings.

Original Image

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Structuring Element

After Opening

After Closing

$$\begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$$

Structuring Element

After Opening

$$\begin{matrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{matrix}$$

Structuring Element

After Closing

External Edge Function

The *external edge* subtracts the source image from the dilated image of the source image. The remaining pixels correspond to the pixels added by the dilation. If I is an image,

$$\textit{external edge}(I) = \textit{dilation}(I) - I = \mathrm{XOR}(I, \textit{dilation}(I)).$$

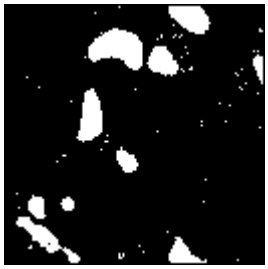
Internal Edge Function

The *internal edge* subtracts the eroded image from its source image. The remaining pixels correspond to the pixels eliminated by the erosion. If I is an image,

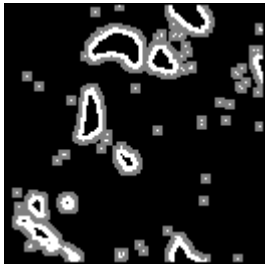
$$\textit{internal edge}(I) = I - \textit{erosion}(I) = \mathrm{XOR}(I, \textit{dilation}(I)).$$

External and Internal Edge Example

This example uses the following binary source image.



Extraction using a 5 × 5 structuring element produces the following image. The superimposition of the internal edge is in white and the external edge is in gray.



The thickness of the extracted contours depends on the size of the structuring element.

Hit-Miss Function

You can use the *hit-miss function* to locate particular configurations of pixels. It extracts each pixel of an image that is placed in a neighborhood matching exactly the template defined by the structuring element. Depending on the configuration of the structuring element, the hit-miss function can be used to locate single isolated pixels, cross-shape or longitudinal patterns, right angles along the edges of objects, and other user-specified shapes. The larger the size of the structuring element, the more specific the researched template can be.

Concept and Mathematics

For a given pixel P_0, the structuring element is centered on P_0. The pixels masked by the structuring element are then referred as P_i. In the example of a structuring element 3 × 3, the P_i range from P_0 to P_8.

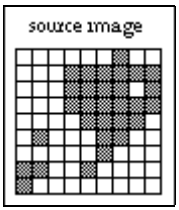
If the value of each pixel P_i is equal to the coefficient of the structuring element placed on top of it, *then* the pixel P_0 is set to 1, *else* the pixel P_0 is set to 0.

In other words, *if* the pixels P_i define the exact same template as the structuring element, *then* P_0 is set to 1, *else* P_0 is set to 0.

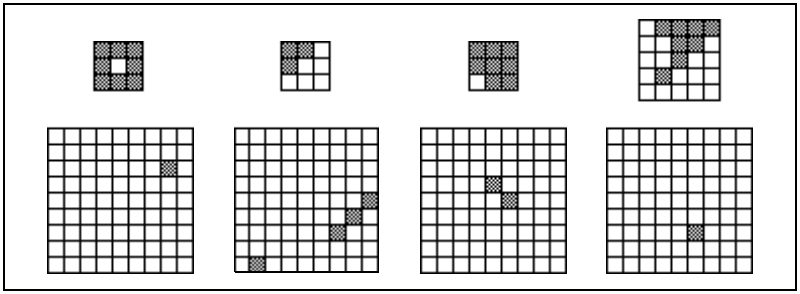
A hit-miss function using a structuring element with a central coefficient equal to 0 changes all pixels set to 1 in the source image to the value 0.

Example 1

This example uses the following source image.

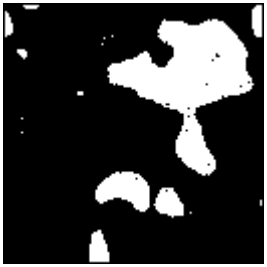


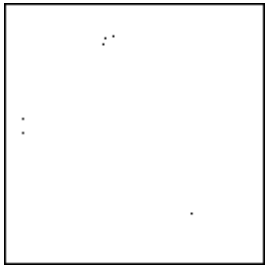
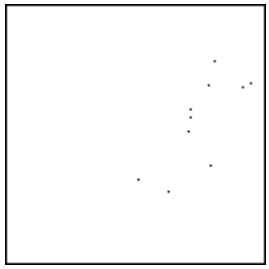
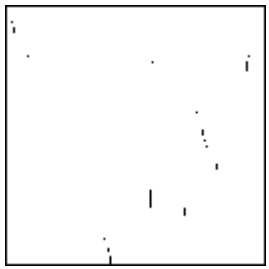
The following series of graphics shows the results of three hit-miss functions applied to the same source image. Each hit-miss function uses a different structuring element (specified above each transformed image). Gray cells indicate pixels equal to 1.



Example 2

This example uses the following binary source image. Given this binary image, the hit-miss function can be used to locate pixels surrounded by various patterns specified via the structuring element.



Use the hit-miss function to locate pixels isolated in a background. The structuring element presented on the right extracts all pixels equal to 1 that are surrounded by at least two layers of pixels equal to 0.	0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0 0	
Use the hit-miss function to locate single pixel holes in objects. The structuring element presented on the right extracts all pixels equal to 0 that are surrounded by at least one layer of pixels equal to 1.	1 1 1 1 **0** 1 1 1 1	
Use the hit-miss function to locate pixels along a vertical left edge. The structuring element presented on the right extracts pixels surrounded by at least one layer of pixels equal to 1 to the left and pixels equal to 0 to the right.	1 1 0 1 **1** 0 1 1 0	

Thinning Function

The *thinning function* eliminates pixels that are located in a neighborhood that matches a template specified by the structuring element. Depending on the configuration of the structuring element, thinning can be used to remove single pixels isolated in the background and right angles along the edges of objects. The larger the size of the structuring element, the more specific the template can be.

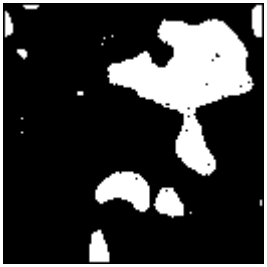
The thinning function extracts the intersection between a source image and its transformed image after a hit-miss function. In binary terms, the operation subtracts its hit-miss transformation from a source image. If I is an image,

$$thinning(I) = I - hit\text{-}miss(I) = \mathrm{XOR}\ (I, hit\text{-}miss(I)).$$

This operation is useless when the central coefficient of the structuring element is equal to 0. In such cases, the hit-miss function can only change the value of certain pixels in the background from 0 to 1. The subtraction of the thinning function then resets these pixels back to 0 anyway.

Examples

This example uses the following binary source image.

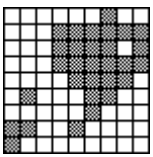


This example uses the thinning function and the following structuring element:

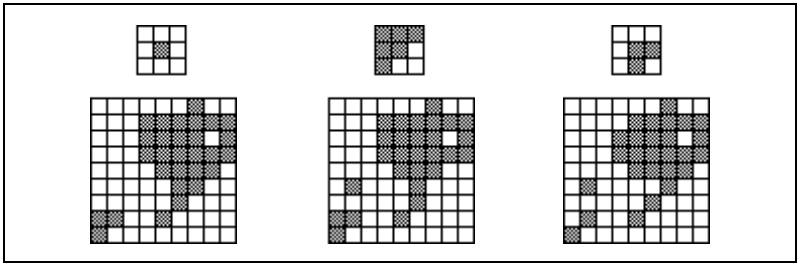
$$\begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix}$$

Thinning produces the following image. Single pixels in the background of this image have been removed.

The next example uses the following source image.



The following series of graphics shows the results of three thinnings applied to the source image. Each thinning uses a different structuring element (specified above each transformed image). Gray cells indicate pixels equal to 1.



Thickening Function

The *thickening function* adds to an image those pixels located in a neighborhood that matches a template specified by the structuring element. Depending on the configuration of the structuring element, thickening can be used to fill holes, smooth right angles along the edges of objects, and so forth. The larger the size of the structuring element, the more specific the template can be.

The thickening function extracts the union between a source image and its transformed image after a hit-miss function that uses the structuring element specified for the thickening. In binary terms, the operation adds a hit-miss transformation to a source image. If I is an image,

$$thickening(I) = I + hit\text{-}miss(I) = \text{OR}\ (I, hit\text{-}miss(I)).$$

This operation is useless when the central coefficient of the structuring element is equal to 1. In such case, the hit-miss function only can turn certain pixels of the objects from 1 to 0. The addition of the thickening function resets these pixels to 1 anyway.

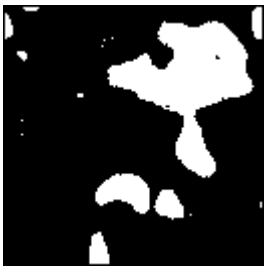
Examples

This example uses the following binary source image.

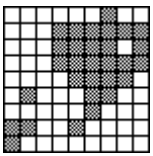
Thickening using the structuring element

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{matrix}$$

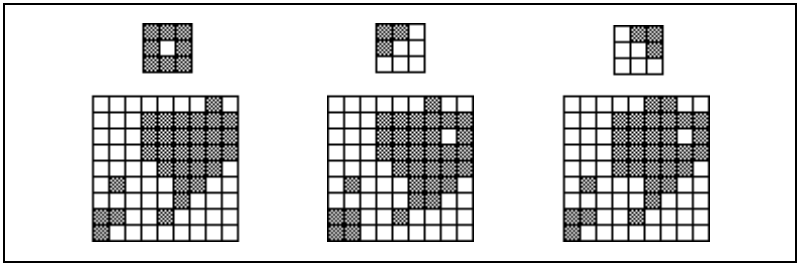
produces the following image. Single pixel holes are filled.



The next example uses the following source image.



The following series of graphics shows the results of three thickenings applied to the source image. Each thickening uses a different structuring element (specified on top of each transformed image). Gray cells indicate pixels equal to 1.



Proper-Opening Function

The *proper-opening function* is a finite and dual combination of openings and closings. It removes small particles and smoothes the contour of objects with respect to the template defined by the structuring element.

If I is the source image, the proper-opening extracts the intersection between the source image I and its transformed image obtained after an opening, followed by a closing, and followed by another opening.

$$\textit{proper-opening}(I) = \mathrm{AND}(I, OCO(I)), \text{ or}$$
$$\textit{proper-opening}(I) = \mathrm{AND}(I, DEEDDE(I)),$$

where I is the source image,

E is an erosion,

D is a dilation,

O is an opening,

C is a closing,

$F(I)$ is the image obtained after applying the function F to the image I, and

$GF(I)$ is the image obtained after applying the function F to the image I followed by the function G to the image I.

Proper-Closing Function

The *proper-closing function* is a finite and dual combination of closings and openings. It fills tiny holes and smoothes the inner contour of objects with respect to the template defined by the structuring element.

If *I* is the source image, the proper-closing extracts the union of the source image *I* and its transformed image obtained after a closing, followed by an opening, and followed by another closing.

$$\textit{proper-closing}(I) = \mathrm{OR}(I, COC(I)), \text{ or}$$
$$\textit{proper-closing}(I) = \mathrm{OR}(I, EDDEED(I)),$$

where *I* is the source image,

E is an erosion,

D is a dilation,

O is an opening,

C is a closing,

F(*I*) is the image obtained after applying the function *F* to the image *I*, and

GF(*I*) is the image obtained after applying the function *F* to the image *I* followed by the function *G* to the image *I*.

Auto-Median Function

The *auto-median function* uses dual combinations of openings and closings. It generates simpler objects that have fewer details.

If *I* is the source image, the auto-median function extracts the intersection between the proper-opening and proper-closing of the source image *I*.

$$\textit{auto-median}(I) = \mathrm{AND}(OCO(I), COC(I)), \text{ or}$$
$$\textit{auto-median}(I) = \mathrm{AND}(DEEDDE(I), EDDEED(I)),$$

where *I* is the source image,

E is an erosion,

D is a dilation,

O is an opening,

C is a closing,

F(*I*) is the image obtained after applying the function *F* to the image *I*, and

GF(*I*) is the image obtained after applying the function *F* to the image *I* followed by the function *G* to the image *I*.

Advanced Binary Morphology Functions

The advanced morphology functions are conditional combinations of fundamental transformations such as the binary erosion and dilation. They apply to binary images in which a threshold of 1 has been applied to objects and the background is equal to 0. The advanced binary morphology functions include the border, hole filling, labeling, lowpass filters, highpass filters, separation, skeleton, segmentation, distance, Danielsson, circle, and convex functions.

Note ***In this section of the manual, the term pixel denotes a pixel equal to 1 and the term object denotes a group of pixels equal to 1.***

Border Function

The *border function* removes objects that touch the border of the image. These objects may have been truncated during the digitization of the image, and their elimination might be useful to avoid erroneous particle measurements and statistics.

Hole Filling Function

The *hole filling function* fills the holes within objects.

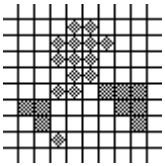
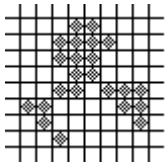
Labeling Function

The *labeling function* assigns a different gray-level value to each object. The image produced is not a binary image, but a labeled image using a number of gray-level values equal to the number of objects in the image plus the gray level 0 used in the background area.

The labeling function can identify objects using connectivity-4 or connectivity-8 criteria.

Lowpass Filters

The *lowpass filter* removes small objects with respect to their width (specified by a parameter called *filter size*).

	Connectivity-4	Connectivity-8
Definition Two pixels are considered as part of the same object if they are horizontally or vertically adjacent.	The pixels are considered as part of two different objects if they are diagonally adjacent.	The pixels are considered as part of the same object if they are horizontally, vertically, or diagonally adjacent.
Illustration For a same pixel pattern, different sets of objects can be identified.		
Example		

For a given filter size N, the lowpass filter eliminates objects with a width less than or equal to $(N - 1)$ pixels. These objects are those that would disappear after $(N - 1)/2$ erosions.

Highpass Filters

The *highpass filter* removes large objects with respect to their width (specified by a parameter called filter size).

For a given filter size N, the highpass filter eliminates objects with a width greater than or equal to N pixels. These objects are those which would not disappear after $(N/2 + 1)$ erosions.

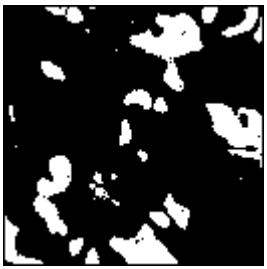
Both the highpass and lowpass morphological filters use erosions to determine if an object is to be removed. Therefore, they cannot discriminate objects with a width of $2k$ pixels from objects with a width of $2k - 1$ pixels. For example, one erosion eliminates both objects that are 2-pixels and 1-pixel wide.

The precision of the filters then depends on the parity of the filter size *N*.

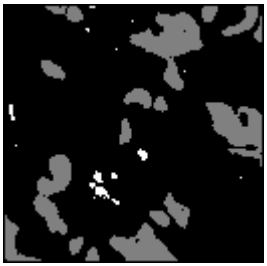
	Highpass Filter	Lowpass Filter
***If N* is an even number (*N* = 2*k*)**	• removes objects with a width greater than or equal to 2*k* • uses *k* – 1 erosions	• removes objects with a width less than or equal to 2*k* – 2 • uses *k* – 1 erosions
***If N* is an odd number (*N* = 2*k* + 1)**	• removes objects with a width greater than or equal to 2*k* + 1 • uses *k* erosions	• removes objects with a width less than or equal to 2*k* • uses *k* erosions

Lowpass and Highpass Example

This example uses the following binary source image.



For a given filter size, a highpass filter produces the following image. Gray objects and white objects are filtered out by a lowpass and highpass filter, respectively.



Separation Function

The *separation function* breaks narrow isthmuses and separates objects that touch each other with respect to an user-specified filter size.

For example, after thresholding an image, two gray-level objects overlapping one another might appear as a single binary object. A narrowing can be observed where the original objects intersected each other. *If* the narrowing has a width of M pixels, a separation using a filter size of $(M + 1)$ breaks it and restore the two original objects. This applies at the same time to all objects that contain a narrowing shorter than N pixels.

For a given filter size N, the separation function segments objects having a narrowing shorter than or equal to $(N – 1)$ pixels. These objects are those that are divided into two parts after $(N – 1)/2$ erosions.

This operation uses erosions, labeling, and conditional dilations.

The above definition is true when N is an odd number. It needs to be modified slightly when N is an even number. This modification is due to the use of erosions to determine if a narrowing has to be broken or kept. The function cannot discriminate a narrowing with a width of $2k$ pixels from a narrowing with a width of $(2k – 1)$ pixels. For example, one erosion breaks both a narrowing that is two pixels wide and a narrowing that is one pixel wide.

The precision of the separation is then limited to the elimination of constrictions having a width lesser than an even number of pixels:

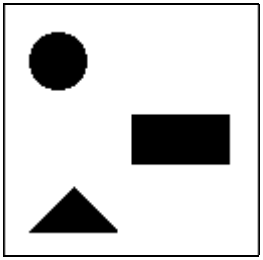
- If N is an even number $(2k)$, the separation breaks a narrowing with a width smaller than or equal to $(2k – 2)$ pixels. It uses $(k – 1)$ erosions.
- If N is an odd number $(2k + 1)$, the separation breaks a narrowing with a width smaller than or equal to $2k$. It uses k erosions.

Skeleton Functions

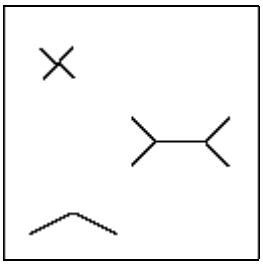
A *skeleton function* applies a succession of thinnings until the width of each object becomes equal to one pixel. The skeleton functions are both time- and memory-consuming. They are based on conditional applications of thinnings and openings using various configurations of structuring elements.

L-Skeleton Function

The *L-skeleton function* indicates the L-shaped structuring element skeleton function. For example, notice the following original image.

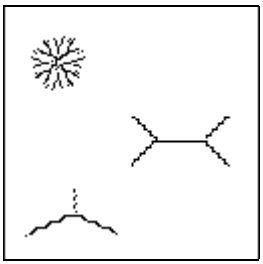


The L-skeleton function produces the following rectangle pixel frame image.



M-Skeleton Function

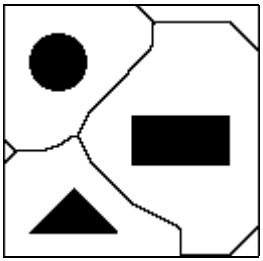
The *M-skeleton* (M-shaped structuring element) function extracts a skeleton with more dendrites or branches. Using the same original image as in the previous example, the M-skeleton function produces the following image.



Skiz Function

The *skiz* (skeleton of influence zones) function behaves like an L-skeleton applied to the background regions, instead of the object regions. It produces median lines that are at an equal distance from the objects.

Using the same source image as in the previous example, the skiz function produces the following image (shown after superimposition on top of the source image).

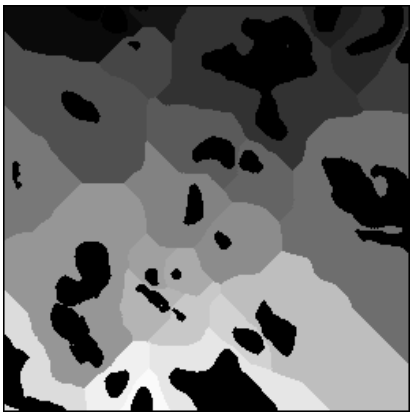


Segmentation Function

The *segmentation function* is only applied to labeled images. It partitions an image into segments, each centered on an object, such that they do not overlap each other or leave empty zones. This result is obtained by dilating objects until they touch one another.

Note ***The segmentation function is time-consuming. It is recommended that you reduce the image to its minimum significant size before selecting this function.***

In the following image, binary objects (shown in black) are superimposed on top of the segments (shown in gray shades).



When applied to an image with binary objects, the transformed image turns entirely red because it is entirely composed of pixels set to 1.

Comparisons Between Segmentation and Skiz Functions

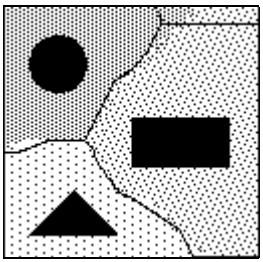
The segmentation function extracts segments that each contain one object and represent the area in which this object can be moved without intercepting another object (assuming that all objects move at the same speed).

The edges of these segments give a representation of the external skeletons of the objects. As opposed to the skiz function, segmentation does not involve median distances.

Segments are obtained by successive dilations of objects until they touch each other and cover the entire image. The final image contains as many segments as there were objects in the original image. On the other hand, if you consider the inside of closed skiz lines as segments, you might produce more segments than objects originally present in the image. Notice the upper-right region in the following example.

The following image shows:

- Original objects in black
- Segments in dotted patterns
- Skiz lines



Distance Function

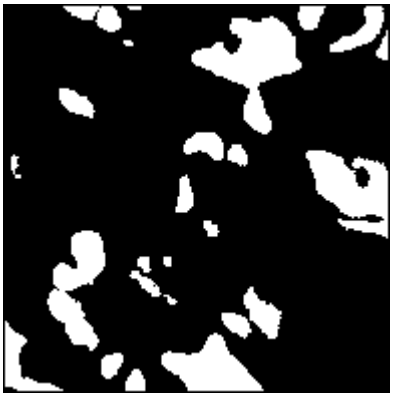
The *distance function* assigns to each pixel a gray-level value equal to the shortest distance to the border of the object. That distance may be equal to the distance to the outer border of the object or to a hole within the object.

Danielsson Function

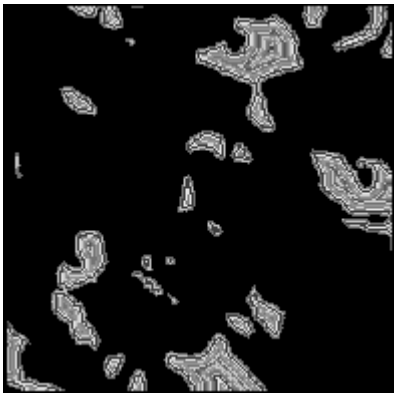
The *Danielsson function* also creates a distance map, but is a more accurate algorithm than the classical distance function. Use the Danielsson function instead of the distance function when possible.

Example

This example uses the following source threshold image.



The image is sequentially processed with a lowpass filter, hole filling, and the Danielsson function. The Danielsson function produces the following distance map image.



It is useful to view this final image with a binary palette. In this case, each level corresponds to a different color. The user easily can determine the relation of a set of pixels to the border of an object. The first layer (the layer

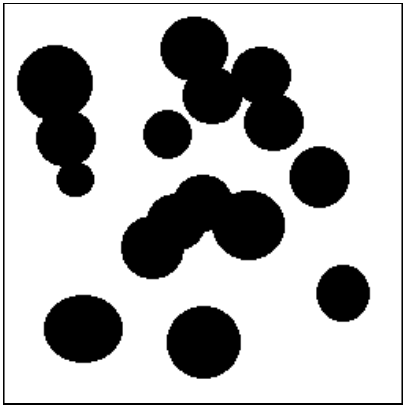
that forms the border) is colored red. The second layer (the layer closest to the border) is green, the third layer is blue, and so forth.

Circle Function

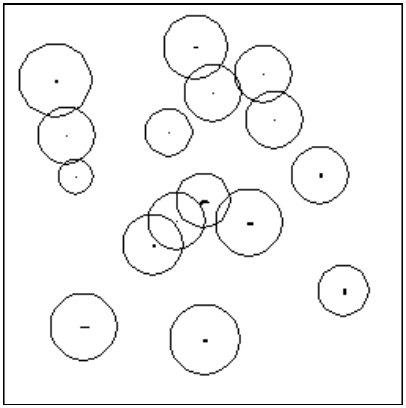
The *circle* function enables the user to separate overlapping circular objects. The circle function uses the Danielsson coefficient to reconstitute the form of an object, provided that the objects are essentially circular. The objects are treated as a set of overlapping discs that is then separated into separate discs. Therefore, it is possible to trace circles corresponding to each object.

Example

This example uses the following source image.



The circle function produces the following processed image.



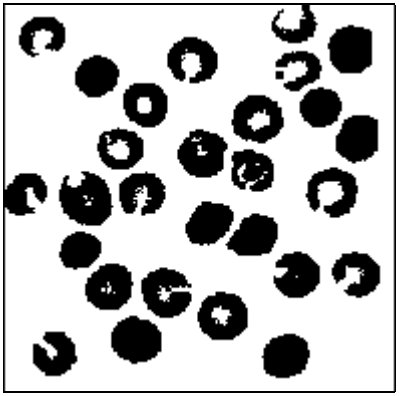
Convex Function

The *convex function* is useful for closing particles so that measurements can be made on the particle, even though the contour of the object is discontinuous. This command is usually needed in cases in which the sample object is cut because of the acquisition process.

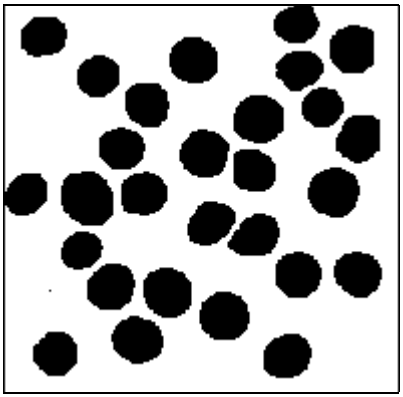
The convex function calculates a convex envelope around the perimeter of each object, effectively closing the object. The image to be treated must be both binary and labeled.

Example

This example uses the following original binary labeled image.



The convex function produces the following image.



Gray-Level Morphology

The gray-level morphology functions apply to gray-level images. You can use these functions to alter the shape of regions by expanding bright areas at the expense of dark areas and vice-versa. These functions smooth gradually varying patterns and increase the contrast in boundary areas. The gray-level morphology functions include the erosion, dilation, opening, closing, proper-opening, proper-closing, and auto-median functions. These functions derive from the combination of gray-level erosions and dilations that use the structuring element.

Erosion Function

A gray-level *erosion* reduces the brightness of pixels that are surrounded by neighbors with a lower intensity. The concept of neighborhood is determined by the template of the structuring element.

Concept and Mathematics

Each pixel P_0 in an image becomes equal to the minimum value of its neighbors. For a given pixel P_0, the structuring element is centered on P_0. The pixels masked by a coefficient of the structuring element equal to 1 are then referred as P_i. In the example of a 3 × 3 structuring element, P_i can range from P_0 to P_8.

$$P_0 = \min(P_i).$$

Note ***A gray-level erosion using a structuring element f × f with all its coefficients set to 1 is equivalent to an Nth order filter with a filter size f × f and the value N equal to 0 (refer to the nonlinear spatial filters).***

Dilation Function

The *gray-level dilation* has the same effect as the gray-level erosion, because dilating bright regions is equivalent to eroding dark regions. This function increases the brightness of each pixel that is surrounded by neighbors with a higher intensity. The concept of neighborhood is determined by the structuring element.

Concept and Mathematics

Each pixel P_0 in an image becomes equal to the maximum value of its neighbors. For a given pixel P_0, the structuring element is centered on P_0. The pixels masked by a coefficient of the structuring element equal to 1 are

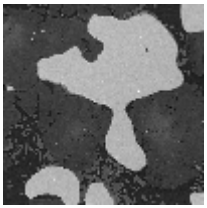
then referred as P_i. In the example of a structuring element 3 × 3, P_i can range from P_0 to P_8.

$$P_0 = \max(P_i).$$

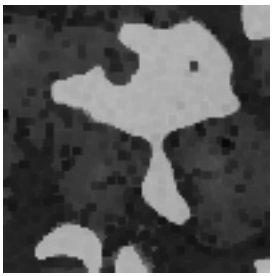
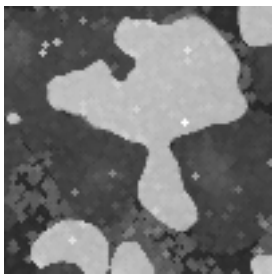
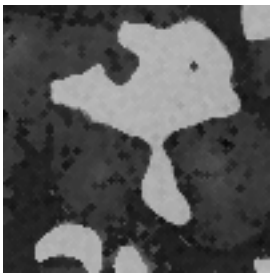
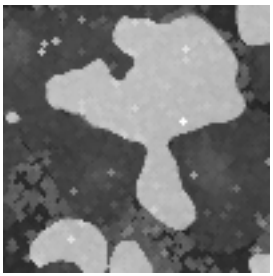
Note ***A gray-level dilation using a structuring element f × f with all its coefficients set to 1 is equivalent to an Nth order filter with a filter size f × f and the value N equal to f × f – 1 (refer to the nonlinear spatial filters).***

Erosion and Dilation Examples

This example uses the following source image.



The following table provides example structuring elements, and the corresponding eroded and dilated images.

Structuring Element	Erosion	Dilation
1 1 1 1 1 1 1 1 1		
0 1 0 1 1 1 0 1 0		

Opening Function

The gray-level *opening function* consists of a gray-level erosion followed by a gray-level dilation. It removes bright spots isolated in dark regions and smoothes boundaries. The effects of the function are moderated by the configuration of the structuring element.

$$opening(I) = dilation(erosion\ (I)).$$

This operation does not alter significantly the area and shape of objects because erosion and dilation are morphological opposites. Bright borders reduced by the erosion are restored by the dilation. However, small bright objects that vanish during the erosion do not reappear after the dilation.

Closing Function

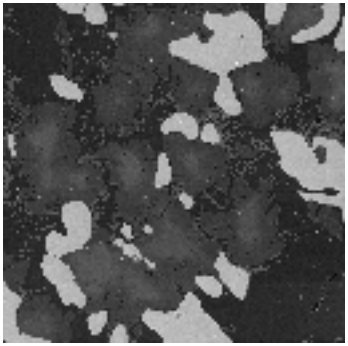
The gray-level *closing function* consists of a gray-level dilation followed by a gray-level erosion. It removes dark spots isolated in bright regions and smoothes boundaries. The effects of the function are moderated by the configuration of the structuring element.

$$closing(I) = erosion(dilation\ (I)).$$

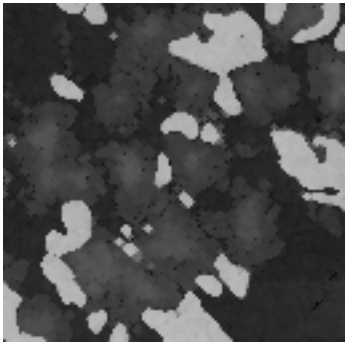
This operation does not alter significantly the area and shape of objects because dilation and erosion are morphological opposites. Bright borders expanded by the dilation are reduced by the erosion. However, small dark objects that vanish during the dilation do not reappear after the erosion.

Opening and Closing Examples

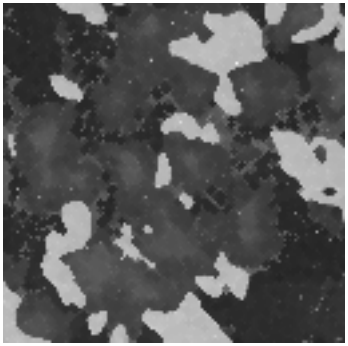
This example uses the following source image.



The opening function produces the following image.



Consecutive applications of an opening or closing command always give the same results. A closing function produces the following image.



Proper-Opening Function

The gray-level *proper-opening* is a finite and dual combination of openings and closings. It removes bright pixels isolated in dark regions and smoothes the boundaries of bright regions. The effects of the function are moderated by the configuration of the structuring element.

If I is the source image, the proper-opening extracts the minimum value of each pixel between the source image I and its transformed image obtained after an opening, followed by a closing, and followed by another opening.

$$\textit{proper-opening}(I) = \min(I, OCO\ (I)), \text{ or}$$
$$\textit{proper-opening}(I) = \min(I, DEEDDE(I)),$$

where *I* is the source image,

E is an erosion,

D is a dilation,

O is an opening,

C is a closing,

F(I) is the image obtained after applying the function *F* to the image *I*, and

GF(I) is the image obtained after applying the function *F* to the image *I* followed by the function *G* to the image *I*.

Proper-Closing Function

The *proper-closing* is a finite and dual combination of closings and openings. It removes dark pixels isolated in bright regions and smoothes the boundaries of dark regions. The effects of the function are moderated by the configuration of the structuring element.

If *I* is the source image, the proper-closing extracts the maximum value of each pixel between the source image *I* and its transformed image obtained after a closing, followed by an opening, and followed by another closing.

$$\textit{proper-closing}(I) = \max(I, COC(I)), \text{ or}$$
$$\textit{proper-closing}(I) = \max(I, EDDEED(I)),$$

where *I* is the source image,

E is an erosion,

D is a dilation,

O is an opening,

C is a closing,

F(*I*) is the image obtained after applying the function *F* to the image *I*, and

GF(*I*) is the image obtained after applying the function *F* to the image *I* followed by the function *G* to the image *I*.

Auto-Median Function

The *auto-median function* uses dual combinations of openings and closings. It generates simpler objects that have fewer details.

If I is the source image, the auto-median extracts the minimum value of each pixel between the two images obtained by applying a proper-opening and a proper-closing of the source image I.

$$\textit{auto-median}(I) = \min(OCO(I), COC(I)), \text{ or}$$
$$\textit{auto-median}(I) = \min(DEEDDE(I), EDDEED(I)),$$

where I is the source image,

E is an erosion,

D is a dilation,

O is an opening,

C is a closing,

$F(I)$ is the image obtained after applying the function F to the image I, and

$GF(I)$ is the image obtained after applying the function F to the image I followed by the function G to the image I.

16

Quantitative Analysis

This chapter provides an overview of quantitative image analysis. The *quantitative analysis* of an image consists of obtaining *densitometry* and object measurements. Before starting this analysis, it is necessary to calibrate the image spatial dimensions and intensity scale to obtain measurements expressed in real units.

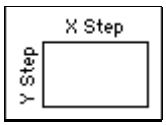
Spatial Calibration

Spatial calibration consists of correlating the area of a pixel with physical dimensions. The latter can be defined by three parameters:

X Step, **Y Step**, and **Unit**.

X Step and **Y Step** are the horizontal and vertical lengths of a pixel. **Unit** is the selected unit of distance.

The area of a pixel is then equal to $(X\ Step \times Y\ Step)Unit^2$.



If a pixel represents a square area, then

$$X\ Step = Y\ Step = Sampling\ Step.$$

The spatial calibration of an image can be performed using two methods:

- *Pixel calibration*, or editing the dimensions of a single pixel
- *Distance calibration*, or editing a the length of a line selected in the image

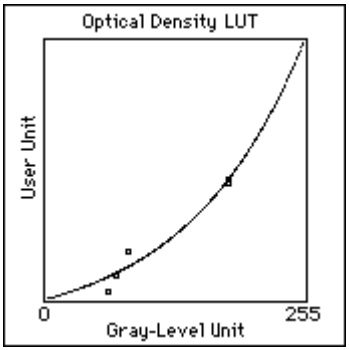
Intensity Calibration

Intensity calibration consists of correlating the gray-scale values to user-defined quantities such as optical densities or concentrations.

The intensity calibration of an image is performed in two steps:

- Selection of sample points in an image and calibration of their gray-level value
- Selection of a curve-fitting algorithm to calibrate the entire gray scale

The following example uses an 8-bit image, or 256 gray levels.



Definition of a Digital Object

In digital images, objects can be defined by three criteria: *intensity threshold*, *connectivity*, and *area threshold*.

Intensity Threshold

Objects are characterized by an *intensity range*. They are composed of pixels with gray-level values belonging to a given threshold interval (overall luminosity or gray shade). Then other pixels are considered part of the background.

The *threshold interval* is defined by the two parameters [**Lower Threshold**, **Upper Threshold**]. In the case of binary objects the threshold interval is [1, 1].

Connectivity

Once the pixels belonging to a specified intensity threshold are identified, they are grouped into objects. This process introduces the notion of adjacent pixels or connectivity.

In a rectangular pixel frame, each pixel P_0 has eight neighbors, as shown in the following graphic. From a mathematical point of view, the pixels P_1, P_3, P_5, and P_7 are closer to P_0 than the pixels P_2, P_4, P_6, and P_8.

$$\begin{bmatrix} P_8 & P_1 & P_2 \\ P_7 & P_0 & P_3 \\ P_6 & P_5 & P_4 \end{bmatrix}$$

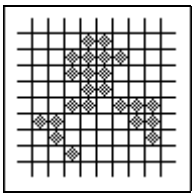
If D is the distance from P_0 to P_1, then the distances between P_0 and its eight neighbors can range from D to $\sqrt{2}D$, as shown in the following graphic.

$$\begin{bmatrix} \sqrt{2}D & D & \sqrt{2}D \\ D & 0 & D \\ \sqrt{2}D & D & \sqrt{2}D \end{bmatrix}$$

Connectivity-8

A pixel belongs to an object if it is at a distance D or $\sqrt{2}\,D$ from another pixel in the object.

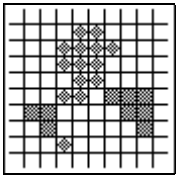
Two pixels are considered as part of a same object if they are horizontally, vertically, or diagonally adjacent. In the following image, the object count equals 1.



Connectivity-4

A pixel belongs to an object if it is at a distance *D* from another pixel in the object.

Two pixels are considered as part of a same object if they are horizontally or vertically adjacent. They are considered as part of two different objects if they are diagonally adjacent. In the following image, the object count equals 4.



Area Threshold

Finally a size criteria can be specified to detect only objects falling in a given area range.

The area threshold is defined by the two parameters [**Minimum Area**, **Maximum Area**].

Examples

In the following example, 1 pixel = 1 square inch.

Objects to Detect	Lower Threshold	Upper Threshold	Minimum Area	Maximum Area
Black objects (gray level 0) as small as 1 sq-µin.	0	0	1	65536
White objects (gray level 255) bigger than 500 sq-µin.	255	255	500	65536
Labeled objects placed in a black background and ranging from 200 to 1000 sq-µin.	1	255	200	1000
Light-gray objects belonging to the gray-level range [190, 200] and smaller than 3000 sq-µin.	190	200	1	3000

Note ***The most straightforward way to isolate objects is to use the threshold function and convert them to binary objects. This method offers the advantage of clearly showing the objects while the threshold interval remains constant and equal to [1, 1].***

Object Measurements

A digital object can be characterized by a set of morphological and intensity parameters described in the *Areas*, *Lengths*, *Coordinates*, *Chords and Axes*, *Shape Equivalence*, *Shape Features*, *Densitometry*, and *Diverse Measurements* sections.

Areas

This section describes the following area parameters:

- **Number of pixels**—Area in number of pixels
- **Particle area**—Area expressed in real units (based on image spatial calibration)
- **Scanned area**—Area of the entire image expressed in real units
- **Ratio**—Ratio of the object area to the entire image area
- **Number of holes**—Number of holes within the object
- **Holes' area**—Total area of the holes
- **Total area**—Area of the object including its holes' area (equals **Particle Area** + **Holes' Area**)

Particle Number

Identification number assigned to an object. Particles are numbered starting from 1 in increasing order from the upper-left corner of the image to the lower-right corner.

Number of Pixels

Number of pixels in an object. This value gives the area of an object, without holes, in pixel units.

Particle Area

Area of an object expressed in real units. This value is equal to **Number of pixels** when the spatial calibration is such that one pixel represents one square unit.

Scanned Area

Area of the entire image expressed in real units. This value is equal to the product (**Resolution X** × **X-Step**)(**Resolution Y** × **Y-Step**).

Ratio

The percentage of the image occupied by all objects.

$$\text{Ratio} = \frac{\text{particle area}}{\text{scanned area}}$$

Number of Holes

Number of holes inside an object. The software detects holes inside an object as small as 1 pixel.

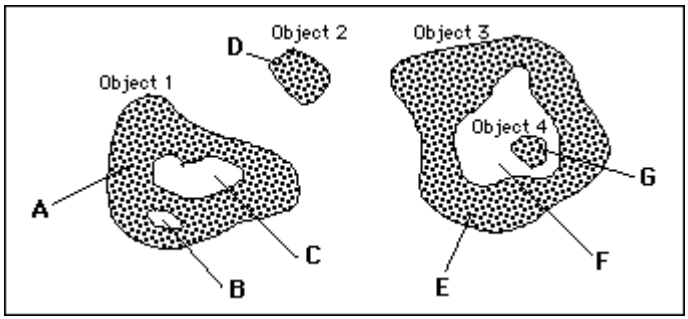
Holes' Area

Total area of the holes within an object.

Total Area

Area of an object including the area of its holes. This value is equal to (**Particle Area** + **Holes' Area**).

☞ **Note** ***An object located inside a hole of a bigger object is identified as a separate object. The area of a hole that contains an object includes the area covered by the object.***



Object #	Particle Area	Holes' Area	Total Area
Object 1	A	B + C	A + B + C
Object 2	D	0	D
Object 3	E	F + G	E + F + G
Object 4	G	0	G

Lengths

This section describes the following length parameters:

- **Particle perimeter**—Length of the outer contour.
- **Holes' perimeter**—Sum of the perimeters of the holes within the object
- **Width**—Distance between the left-most and right-most pixels in the object
- **Height**—Distance between the upper-most and lower-most pixels in the object

Particle Perimeter

Length of the outer contour of an object.

Holes' Perimeter

Sum of the perimeters of the holes within an object.

Note ***Holes' measurements can turn into valuable data when studying constituents A and B such that B is occluded in A. If the image can be processed so that the B regions appear as holes in A regions after a threshold, the ratio (Holes' Area ÷ Particle Total Area) gives the percentage of B in A. Holes' perimeter gives the length of the boundary between A and B.***

Breadth

Distance between the left-most and right-most pixels in an object, or $\max(X_i) - \min(X_i)$. It is also equal to the horizontal side of the smallest horizontal rectangle containing the object, or the difference $\max X - \min X$.

Height

Distance between the upper-most and lower-most pixels in an object, or $\max(Y_i) - \min(Y_i)$. It is also equal to the vertical side of the smallest horizontal rectangle containing the object, or the difference $\max Y - \min Y$.

Coordinates

Coordinates are expressed with respect to an origin (0, 0), located at the upper-left corner of the image. This section describes the following coordinate parameters:

- **Center of Mass (X, Y)**—Coordinates of the center of gravity
- **Min X, Min Y**—Upper-left corner of the smallest horizontal rectangle containing the object
- **Max X, Max Y**—Lower-right corner of the smallest horizontal rectangle containing the object
- **Max chord X and Y**—Left-most point along the longest horizontal chord

Center of Mass X and Center of Mass Y

Coordinates of the center of gravity of an object. The center of gravity of an object composed of N pixels P_i is defined as the point G such that

$$\overline{OG} = \frac{1}{N}\sum_{i=1}^{i=N}\overline{OP_i}, \text{ and}$$

$$\text{the center of mass } X_G = \frac{1}{N}\sum_{i=1}^{i=N} X_i.$$

X_G gives the average location of the central points of horizontal segments in an object.

$$\text{The center of mass } Y_G = \frac{1}{N}\sum_{i=1}^{i=N} Y_i.$$

Y_G gives the average location of the central points of horizontal segments in an object.

Note ***G* can be located outside an object if the latter has a convex shape.**

Min(X, Y) and Max(X, Y)

Coordinates of the upper-left and lower-right corners of the smallest horizontal rectangle containing an object.

The origin (0, 0) has two pixels that have the coordinates (min$X$, min$Y$) and (max$X$, max$Y$) such that

$$\min X = \min(X_i)$$

$$\min Y = \min(Y_i)$$

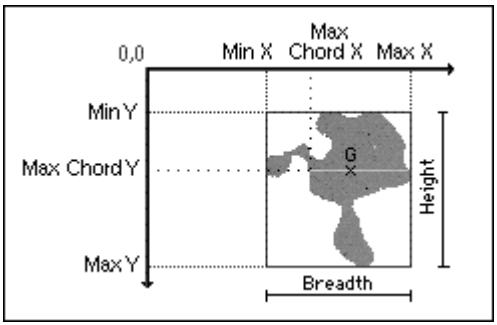
$$\max X = \max(X_i)$$

$$\max Y = \max(Y_i)$$

where X_i and Y_i are the coordinates of the pixels P_i in an object.

Max Chord X and Max Chord Y

Coordinates of the left-most pixel along the longest horizontal chord in an object.



Chords and Axes

This section describes the following chord and axis parameters:

- **Max chord length**—Length of the longest horizontal chord
- **Mean chord X**—Mean length of horizontal segments
- **Mean chord Y**—Mean length of vertical segments
- **Max intercept**—Length of the longest segment (in all possible directions)

- **Mean intercept perpendicular**—Mean length of the segments perpendicular to the max intercept
- **Particle orientation**—Orientation in degree with respect to the horizontal axis

Max Chord Length

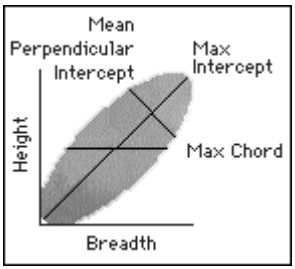
Length of the longest horizontal chord in an object.

Mean Chord X

Mean length of horizontal segments in an object.

Mean Chord Y

Mean length of vertical segments in an object.



Max Intercept

Length of the longest segment in an object (in all possible directions of projection).

Mean Intercept Perpendicular

Mean length of the segments in an object perpendicular to the max intercept.

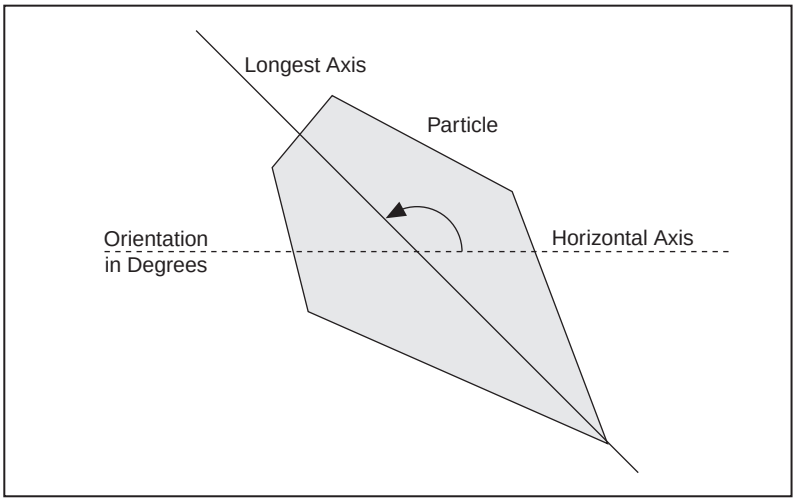
$$\text{Mean intercept perpendicular} = \frac{\text{particle area}}{\text{max intercept}}$$

Particle Orientation

The angle of the longest axis with respect to the horizontal axis. The value can be between 0° and 180°.

Notice that this value does not give information regarding the symmetry of the particle.

Therefore, an angle of 190° is considered the same as 10°.



Shape Equivalence

This section describes the following shape-equivalence parameters:

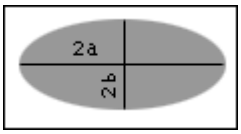
- **Equivalent ellipse minor axis**—Minor axis of the ellipse that has the same area as the object and a major axis equal to half its max intercept
- **Ellipse major axis**—Major axis of the ellipse that has the same area and same perimeter as the object
- **Ellipse minor axis**—Minor axis of the ellipse that has the same area and same perimeter as the object
- **Ellipse Ratio**—Ratio of the major axis of the equivalent ellipse to its minor axis
- **Rectangle big side**—Big side of the rectangle that has the same area and same perimeter as the object
- **Rectangle small side**—Small side of the rectangle that has the same area and same perimeter as the object
- **Rectangle ratio**—Ratio of the big side of the equivalent rectangle to its small side

Equivalent Ellipse Minor Axis

The *equivalent ellipse minor axis* is the minor axis of the ellipse that has the same area as the object and a major axis equal to half the max intercept of the object.

This definition gives the following set of equations:

$$\text{particle area} = \pi ab, \text{ and}$$
$$\text{max intercept} = 2a.$$



The equivalent ellipse minor axis is defined as

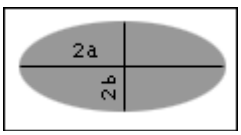
$$2b = \frac{4 \times \text{particle area}}{\pi \times \text{max intercept}}.$$

Ellipse Major Axis

The *ellipse major axis* is the total length of the major axis of the ellipse that has the same area and same perimeter as an object. This length is equal to $2a$.

This definition gives the following set of equations:

$$Area = \pi ab$$
$$Perimeter = \pi\sqrt{2(a^2 + b^2)}$$



This set of equations can be expressed so that the sum $a + b$ and the product ab become functions of the parameters **Particle Area** and **Particle Perimeter**. a and b then become the two solutions of the polynomial equation $X^2 - (a + b)X + ab = 0$.

Notice that for a given area and perimeter, only one solution (a, b) exists.

Ellipse Minor Axis

The *ellipse minor axis* is the total length of the minor axis of the ellipse that has the same area and same perimeter as an object. This length is equal to $2b$.

Ellipse Ratio

The *ellipse ratio* is the ratio of the major axis of the equivalent ellipse to its minor axis.

It is defined as $\frac{\text{ellipse major axis}}{\text{ellipse minor axis}} = \frac{a}{b}$.

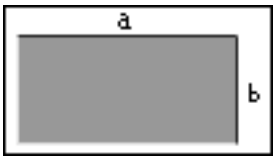
The more elongated the equivalent ellipse, the higher the ellipse ratio. The closer the equivalent ellipse is to a circle, the closer to 1 the ellipse ratio.

Rectangle Big Side

Rectangle big side is the length of the big side (a) of the rectangle that has the same area and same perimeter as an object.

This definition gives the following set of equations:

$$Area = ab$$
$$Perimeter = 2(a + b)$$



This set of equations can be expressed so that the sum $a + b$ and the product ab become functions of the parameters **Particle Area** and **Particle Perimeter**, a and b then become the two solutions of the polynomial equation $X^2 - (a + b)X + ab = 0$.

Notice that for a given area and perimeter, only one solution (a, b) exists.

Rectangle Small Side

Rectangle small side is the length of the small side of the rectangle that has the same area and same perimeter as an object. This length is equal to b.

Rectangle Ratio

Rectangle ratio is the ratio of the big side of the equivalent rectangle to its small side.

It is defined as $\frac{\text{rectangle big side}}{\text{rectangle small side}} = \frac{a}{b}$.

The more elongated the equivalent rectangle, the higher the **Rectangle ratio**.

The closer the equivalent rectangle is to a square, the closer to 1 the **Rectangle ratio**.

Shape Features

This section describes the following shape-feature parameters:

- **Moments of Inertia**—Moments of Inertia I_{xx}, I_{yy}, I_{xy} with respect to the center of gravity
- **Elongation factor**—Ratio of the longest segment within the object to the mean length of the perpendicular segments
- **Compactness factor**—Ratio of the object area to the area of the smallest rectangle containing the object
- **Heywood Circularity factor**—Ratio of the object perimeter to the perimeter of the circle with the same area
- **Hydraulic Radius**—Ratio of the object area to its perimeter
- **Waddel Disk Diameter**—Diameter of the disk with the same area as the object

Moments of Inertia I_{xx}, I_{yy}, I_{xy}

The *moments of inertia* give a representation of the distribution of the pixels in an object with respect to its center of gravity.

Elongation Factor

The elongation factor is the ratio of the longest segment within an object to the mean length of the perpendicular segments. It is defined as

$$\frac{\text{max intercept}}{\text{mean perpendicular intercept}}.$$

The more elongated the shape of an object, the higher its elongation factor.

Compactness Factor

The *compactness factor* is the ratio of an object area to the area of the smallest rectangle containing the object. It is defined as

$$\frac{\text{particle area}}{breadth \times width}.$$

The compactness factor belongs to the interval [0, 1]. The closer the shape of an object is to a rectangle, the closer to 1 the compactness factor.

Heywood Circularity Factor

The *Heywood circularity factor* is the ratio of an object perimeter to the perimeter of the circle with the same area. It is defined as

$$\frac{\text{particle perimeter}}{\text{perimeter of circle with same area as particle}} = \frac{\text{particle perimeter}}{2\sqrt{\pi \times \text{particle area}}}.$$

The closer the shape of an object is to a disk, the closer the Heywood circularity factor to 1.

Hydraulic Radius

The hydraulic radius is the ratio of an object area to its perimeter. It is defined as

$$\frac{\text{particle area}}{\text{particle perimeter}}.$$

If a particle is a disk with a radius *R*, *then* its hydraulic radius is equal to

$$\frac{\pi R^2}{2\pi R} = \frac{R}{2}.$$

The hydraulic radius is equal to half the radius *R* of the circle such that

$$\frac{\text{circle area}}{\text{circle perimeter}} = \frac{\text{particle area}}{\text{particle perimeter}}.$$

Waddel Disk Diameter

Diameter of the disk with the same area as the particle. It is defined as

$$\frac{2\sqrt{\text{particle area}}}{\sqrt{\pi}}.$$

The following tables list the definition of the primary measurements and the measurements that are derived from them.

Definitions of Primary Measurements

A	Area
p	Perimeter
Left	Left-most point
Top	Top-most point
Right	Right-most point
Bottom	Bottom-most point
P_x	Projection x
P_y	Projection y

Derived Measurements

Symbol	Derived Measurement	Primary Measurement
l	**Width**	Right – Left
h	**Height**	Bottom – Top
d	**Diagonal**	$\sqrt{l^2 + h^2}$
M_x	**Center of Mass X**	$(\Sigma x)/A$
M_y	**Center of Mass Y**	$(\Sigma y)/A$
I_{xx}	**Inertia XX**	$(\Sigma x^2) - A \times M_{x^2}$
I_{yy}	**Inertia YY**	$(\Sigma y^2) - A \times M_{y^2}$
I_{xy}	**Inertia XY**	$(\Sigma xy) - A \times M_x \times M_y$

Symbol	Derived Measurement	Primary Measurement
C_x	**Mean Chord X**	A/P_y
C_y	**Mean Chord X**	A/P_x
S_{max}	**Max Intercept**	$(C_{max}/h)^2 \times \max(h, l) + d(1 - (C_{max}/l)^2)$
C	**Mean Perpendicular Intercept**	A/S_{max}
A_{2b}	**Equivalent Ellipse Minor Axis**	$4 \times A / (\pi S_{max})$
$d°$	**Orientation**	If $I_{xx} = I_{yy}$, then $d° = 45$, else $d° = \dfrac{90}{\operatorname{atan}(2 \times I_{XY} \div (I_{XX} - I_{YY}))}$ If $I_{xx} \geq I_{yy}$ and $I_{xy} \geq 0$, then $d° = 180 - d°$ If $I_{xx} \geq I_{yy}$ and $I_{xy} < 0$, then $d° = -d°$ If $I_{xx} < I_{yy}$, then $d° = 90 - d°$ If $d° < 0$, then $d° = 0°$
E_{2a}	**Ellipse major axis (2*a*)**	$E_{2a} = \sqrt{\frac{p^2}{2\pi^2} + \frac{2\pi}{A}} + \sqrt{\frac{p^2}{2\pi^2} - \frac{2\pi}{A}}$
E_{2b}	**Ellipse minor axis (2*b*)**	$2b = \sqrt{\frac{p^2}{2\pi^2} + \frac{2\pi}{A}} - \sqrt{\frac{p^2}{2\pi^2} - \frac{2\pi}{A}}$
E_{ab}	**Ellipse Ratio**	E_{2a} / E_{2b}
R_c	**Rectangle big side**	$\frac{1}{4}(p + t')$ where $t' = \sqrt{p^2 - 16A}$
r_c	**Rectangle small side**	$\frac{1}{4}(p - t')$ where $t' = \sqrt{p^2 - 16A}$
R_{Rr}	**Rectangle Ratio**	R_c/r_c
F_e	**Elongation factor**	$S_{max}/C\pi$
F_c	**Compactness factor**	$A/(h \times l)$
F_H	**Heywood Circularity factor**	$\frac{p}{2\sqrt{\pi A}}$

Symbol	Derived Measurement	Primary Measurement
F_t	**Type factor**	$\frac{A^2}{4\pi\sqrt{I_{XX} \times I_{YY}}}$
R_h	**Hydraulic Radius**	A/p
R_d	**Waddel Disk Diameter**	$2\sqrt{\frac{A}{\pi}}$

Densitometry

IMAQ Vision contains the following densitometry parameters:

- **Minimum Gray Value**—Minimum intensity value in gray-level units
- **Maximum Gray Value**—Maximum intensity value in gray-level units
- **Sum Gray Value**—Sum of the intensities in the object expressed in gray-level units
- **Mean Gray Value**—Mean intensity value in the object expressed in gray-level units
- **Standard deviation**—Standard deviation of the intensity values
- **Minimum User Value**—Minimum intensity value in user units
- **Maximum User Value**—Maximum intensity value in user units
- **Sum User Value**—Sum of the intensities in the object expressed in user units
- **Mean User Value** —Mean intensity value in the object expressed in user units
- **Standard deviation (Unit)**—Standard deviation of the intensity values in user units

Diverse Measurements

These primary coefficients are used in the computation of measurements such as moments of inertia and center of gravity. IMAQ Vision contains the following diverse-measurement parameters

- **SumX**—Sum of the *x* coordinates of each pixel in a particle
- **SumY**—Sum of the *y* coordinates of each pixel in a particle
- **SumXX, SumYY, SumXY**—Sum of *x* coordinates squared, sum of *y* coordinates squared, and sum of *x*y coordinates for each pixel in a particle

- **Corrected Projection X**—Sum of the horizontal segments that do not superimpose any other horizontal segment
- **Corrected Projection Y**—Sum of the vertical segments that do not superimpose any other horizontal segment

A

Common Questions

This appendix contains answers to frequently asked questions.

How do I use the `AutoDelete` property on the CWIMAQViewer control?

When set to `TRUE`, the `AutoDelete` property removes all existing Region objects in the Regions collection when a new region is added with the mouse. To prevent existing Region objects from being removed, press the <Shift> key as you add the region or set `AutoDelete` to `FALSE`.

How do I use the `Active` property on the CWIMAQRegion object?

Region objects that have their `Active` property set to `TRUE` are used by methods on the Regions collection. For example, the `RegionsToMask` method creates a mask image including all regions with `Active` set to `TRUE`.

How can I access individual elements of an IMAQ Vision report?

Pass the index of the element that you want to get or set to the property of the report. Reports are indexed starting at 1. For example, the following code returns the polarity of the first edge.

```
polarity = CWIMAQEdgeReport.Polarity(1)
```

How do I size the CWIMAQViewer control to match the size of the image that it is viewing?

To match the size of the viewer to the image, set the `Height` and `Width` properties of the Viewer object. If you use the `BorderWidth` property, add that value to the `Height` and `Width` properties. For example, if the `BorderWidth` is 10, add 10 pixels to the `Width` and `Height` properties.

Why do I get a type error when using a color or complex CWIMAQVision method?

Some IMAQ Vision functions only operate on specific image types. You can change the image type by setting the `Type` property on the Image object.

The user response of my application seems to decrease when a CWIMAQViewer control is updating rapidly. How can I correct this?

Set `ImmediateUpdates` on the Viewer to `FALSE` to prevent the Viewer from redrawing on every image update. When `ImmediateUpdates` is set to `FALSE`, the application redraws only when instructed by the operating system.

How can I stop drawing a polygon or broken line region in the CWIMAQViewer control?

Hold down the <Ctrl> key when selecting the last point of the region. Polygon regions automatically close themselves by connecting the first and last points with a line segment.

B

Error Codes

This appendix lists the error codes returned by ComponentWorks, the ComponentWorks IMAQ controls, IMAQ hardware, and Vision.

Table B-1. ComponentWorks Errors

Error Code	Description
–30000	Unexpected error.
–30002	You have passed an invalid value for one of the parameters to the function, method, or property.
–30003	You have passed an invalid type into a parameter of a Variant type.
–30004	Divide by zero error.
–30005	Result of a calculation is an imaginary number.
–30006	Overflow error.
–30007	Out of memory.
–30008	You have called a function or method requiring a ComponentWorks product for which you do not have a license. For example, you might be using a method that is not supported in the base or standard Analysis package. To upgrade your product, contact National Instruments.

Table B-2. ComponentWorks IMAQ Errors

Error Code	Description
–30200	Corrupt camera file detected.
–30201	Change requires reconfigure to take effect.
–30202	Interface still locked.
–30203	Unstable blanking reference.

Table B-2. ComponentWorks IMAQ Errors (Continued)

Error Code	Description
–30204	Bad quality colorburst.
–30205	IMAQ operation not configured.
–30206	IMAQ operation not active.
–30207	IMAQ operation cannot be started because it is not configured or is currently active.
–30208	Triggered asynchronous acquisitions are not supported with StillColor enabled.
–30209	Triggered asynchronous acquisitions require NI-IMAQ version 2.0 or greater.
–30210	Object has been locked by an ongoing acquisition and cannot be modified.
–30211	IMAQ Operation Active.
–30212	Maximum number of RTSI lines already in use.
–30213	Cannot load IMAQ Vision DLL.
–30214	Cannot load IMAQ DLL. Make sure the IMAQ driver is installed correctly.
–30215	Specified function is not located in the driver.
–30216	Maximum number of devices in use exceeded.
–30217	Specified device is already in use by another CWIMAQ control.
–30218	Currently selected color mode is not supported by the specified device.
–30219	Invalid ColorMode for specified device.
–30220	Invalid FrameFieldMode for specified device.
–30221	Acquisition timed out.

Table B-3. IMAQ Errors

Error Code	Description
–30801	Function not implemented.
–30802	Too many interfaces open.
–30803	Not enough memory to perform the operation.
–30804	Operating system error occurred.
–30805	Invalid parameter #1.
–30806	Invalid parameter #2.
–30807	Invalid parameter #3.
–30808	Invalid parameter #4.
–30809	Invalid parameter #5.
–30810	Invalid parameter #6.
–30811	Invalid parameter #7.
–30812	Too many buffers already allocated.
–30813	DLL internal error—bad logic state.
–30814	Buffer size is too small for minimum acquisition frame.
–30815	Exhausted buffer ids.
–30816	Not enough physical memory.
–30817	Error releasing the image buffer.
–30818	Bad buffer pointer in list.
–30819	Buffer list is not locked.
–30820	No camera defined for this channel.
–30821	Bad interface.
–30822	Rowbytes is less than region of interest.
–30823	ROI width is greater than rowbytes.
–30824	No interface or bad camera file.
–30825	Hardware limitation.

Table B-3. IMAQ Errors (Continued)

Error Code	Description
–30826	Invalid action—no buffers configured for session.
–30827	Buffer list does not contain a valid final command.
–30828	Buffer list contains an invalid command.
–30829	A buffer list frame buffer address is null.
–30830	No acquisition in progress.
–30831	Cannot lock on video source.
–30832	Bad DMA transfer—use reset.
–30833	Unable to perform request—acquisition in progress.
–30834	Wait timed out—acquisition not complete.
–30835	No buffers available—too early in acquisition.
–30836	Zero buffer size—no bytes filled.
–30837	Bad parameter to low level—check attributes and high level arguments.
–30838	Rigger loopback problem—cannot drive with action enabled.
–30839	No interface found.
–30840	Unable to load DLL.
–30841	Unable to find API function in DLL.
–30842	Unable to allocate system resources.
–30843	No trigger action—acquisition will time out.
–30844	FIFO overflow caused acquisition to halt.
–30845	Memory lock error—cannot perform acquisition.
–30846	Interface locked.
–30847	No external pixel clock.
–30848	Field scaling mode not supported.
–30849	Channel not set to 1 when using StillColor RGB acquisition.

Table B-3. IMAQ Errors (Continued)

Error Code	Description
–30850	Error during small buffer allocation.
–30851	Error during large buffer allocation.
–30852	Bad camera type—camera needs to be of type NTSC or PAL.
–30853	Camera not supported—must be an 8-bit camera.
–30854	Bad camera parameter in configuration file.
–30855	PAL key detection error.

Table B-4. Vision Errors

Error Code	Description
–31000	Demo version timeout.
–31001	System error.
–31002	Memory full.
–31003	Memory error.
–31004	Bad image reference.
–31005	Image reference not specified.
–31006	Bad image type.
–31007	Image(s) type incompatible.
–31008	Image(s) size incompatible.
–31009	Bad border size.
–31010	Image type not supported.
–31011	Image types must be the same.
–31012	Unable to open driver.
–31013	IO error.
–31014	Board not found.
–31015	IO timeout.

Table B-4. Vision Errors (Continued)

Error Code	Description
–31016	String cannot be NULL.
–31017	Bad file header.
–31018	Bad file type.
–31019	Color table error.
–31020	Invalid parameter.
–31021	File already open for writing.
–31022	File not found.
–31023	Too many files open.
–31024	Unspecified I/O error.
–31025	Access denied.
–31026	File type not supported.
–31027	Unable to get info.
–31028	Unable to read data.
–31029	Unable to write data.
–31030	End of file occurred.
–31031	Bad file format.
–31032	Bad file operation (info, read, write).
–31033	File data type not supported.
–31034	GUI initialization error.
–31035	GUI unable to create window.
–31036	GUI bad window id.
–31037	Image window does not exist.
–31038	Bad window attribute.
–31039	Bad number of classes.
–31040	Bad particle.
–31041	Bad number of measure.

Table B-4. Vision Errors (Continued)

Error Code	Description
–31042	Kernel internal error.
–31043	DLL not found.
–31044	DLL access error.
–31045	Bad ROI descriptor.
–31046	Bad ROI global rectangle.
–31047	8-bit image expected.
–31048	16-bit image expected.
–31049	Float image expected.
–31050	Complex image expected.
–31051	RGB image expected.
–31052	Bad complex plane.

C

Distribution and Redistributable Files

This chapter contains information about ComponentWorks IMAQ Vision redistributable files and distributing applications that use ComponentWorks controls.

Files

The files in the `\Setup\redist` directory of the ComponentWorks IMAQ CD are necessary for distributing applications and programs that use ComponentWorks controls. You need to distribute only those files needed by the controls you are using in your application.

Distribution

When installing an application using ComponentWorks IMAQ controls on another computer, you also must install the necessary control files and supporting libraries on the target machine. In addition to installing all necessary OCX files on a target computer, you must register each of these files with the operating system. This allows your application to find the correct OCX file and create the controls.

If your application performs any I/O operations requiring separate driver software, such as image acquisition, you must install and configure the driver software and corresponding hardware on the target computer. For more information, consult the hardware documentation for the specific driver used.

When distributing applications with the ComponentWorks IMAQ controls, do not violate the license agreement (section 5) provided with the software. If you have any questions about the licensing conditions, contact National Instruments.

Automatic Installers

Many programming environments include some form of a setup or distribution kit tool. This tool automatically creates an installer for your application so that you can easily install it on another computer. To function successfully, this tool must recognize which control files and supporting libraries are required by your application and include these in the installer it creates. The resulting installer also must register the controls on the target machine.

Some of these tools, such as the Visual Basic 5 Setup Wizard, use dependency files to determine which libraries are required by an OCX file. The ComponentWorks IMAQ OCX file includes a corresponding dependency file located in the `\Windows\System` directory (`\Windows\System32` for WindowsNT) after you install the ComponentWorks IMAQ software.

Some setup tools might not automatically recognize which files are required by an application but provide an option to add additional files to the installer. In this case, verify that all necessary OCX files (corresponding to the controls used in your application) as well as all the DLL and TLB files from the `\redist` directory are included. You also should verify that the resulting installer does not copy older versions of a file over a newer version on the target machine.

If your programming environment does not provide a tool or wizard for building an installer, you may use third-party tools, such as InstallShield. Some programming environments provide simplified or trial versions of third-party installer creation tools on their installation CDs.

Manual Installation

If your programming environment does not include a setup or distribution kit tool, you must build your own installer and perform the installation task manually. To install your application on another computer, follow these steps:

1. Copy the application executable to the target machine.
2. Copy the ComponentWorks IMAQ OCX file to the System directory (`\Windows\System` for Windows 95 or `\Windows\System32` for WindowsNT) on the target machine.

3. Copy all DLL and TLB files in the `\redist` directory to the System directory on the target machine.
4. Copy any other DLLs and support files required by your application to the System directory on the target machine.

Some of these files might already be installed on the target machine. If the file on the target machine has an earlier version number than the file in the `\redist` directory, copy the newer file to the target machine.

After copying the files to the target machine, you must register all OCX files with the operating system. To register an OCX file, you need a utility such as `REGSVR32.EXE`. You must copy this utility to the target machine to register the OCX files, but you can delete it after completing the installation. Use this utility to register each OCX file with the operating system, as in the following example.

```
regsvr32 c:\windows\system\cwimaq.ocx
```

ComponentWorks IMAQ Evaluation

Once the ComponentWorks IMAQ OCX file is installed and registered on a target computer, your application can create the controls as necessary. You or your customer also can use the same OCX file in any compatible development environment as an evaluation version of the controls.
If desired, you may distribute the ComponentWorks IMAQ reference files (from the `\redist` directory) with your application, which provide complete documentation of the ComponentWorks IMAQ controls when used in evaluation mode.

If you would like to use the ComponentWorks IMAQ controls as a development tool on this target machine, you must purchase another ComponentWorks IMAQ development system. Contact National Instruments to purchase additional copies of the ComponentWorks IMAQ software.

Run-Time Licenses

For each copy of your ComponentWorks IMAQ-based application that you distribute, you must have a valid run-time license. A limited number of run-time licenses are provided with the ComponentWorks IMAQ development systems. National Instruments driver software might also provide you with ComponentWorks IMAQ run-time licenses. You can purchase additional ComponentWorks IMAQ run-time licenses from

National Instruments. Consult the license agreement (section 5) provided with the software for more detailed information. If you have any questions about the licensing conditions, contact National Instruments.

Troubleshooting

Try the following suggestions if you encounter problems after installing your application on another computer.

The application is not able to find an OCX file or is not able to create a control.

- The control file or one of its supporting libraries is not copied on the computer. Verify that the correct OCX files and all their supporting libraries are copied on the machine. If one control was built using another, you might need multiple OCX files for one control.
- The control is not properly registered on the computer. Make sure you run the registration utility and that it registers the control.

Controls in the application run in evaluation (demo) mode.

- The application does not contain the correct run-time license. When developing your application, verify that the controls are running in a fully licensed mode. Although most programming environments include a run-time license for the controls in the executable, some do not.

  If you are developing an application in Visual C++ using SDI (single document interface) or MDI (multiple document interface), you must include the run-time license in the program code for each control you create. Consult the ComponentWorks documentation, National Instruments Knowledgebase (`www.natinst.com/support`) or technical support if you are not familiar with this operation.

D

Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call 512 795 6990. You can access these services at:

United States: 512 794 5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at 512 418 1111.

E-Mail Support (Currently USA Only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

Country	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Brazil	011 288 3336	011 288 8528
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 6120092	03 6120095
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___) _____Phone (___) _____

Computer brand_____ Model _____Processor_____

Operating system (include version number) _____

Clock speed _____MHz RAM _____MB Display adapter _____

Mouse ___yes ___no Other adapters installed_____

Hard disk capacity _____MB Brand_____

Instruments used _____

_____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

_____

_____

_____

_____

List any error messages: _____

_____

_____

The following steps reproduce the problem: _____

_____

_____

_____

_____

ComponentWorks IMAQ Vision Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

IMAQ hardware _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

ComponentWorks and NI-IMAQ version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *Getting Results with ComponentWorks™ IMAQ Vision*

Edition Date: June 1998

Part Number: 321883A-01

Please comment on the completeness, clarity, and organization of the manual.

_____

_____

_____

_____

_____

_____

_____

If you find errors in the manual, please record the page numbers and describe the errors.

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

E-Mail Address _____

Phone (___) _____ Fax (___) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, Texas 78730-5039

Fax to: Technical Publications
National Instruments Corporation
512 794 5678

Glossary

Prefix	Meanings	Value
p-	pico	10^{-12}
n-	nano-	10^{-9}
µ-	micro-	10^{-6}
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}
G-	giga-	10^{9}
t-	tera-	10^{12}

Numbers

1D — One-dimensional.

2D — Two-dimensional.

3D — Three-dimensional.

3D view — Displays the light intensity of an image in a three-dimensional coordinate system, where the spatial coordinates of the image form two dimensions and the light intensity forms the third dimension.

A

ActiveX — Set of Microsoft technologies for reusable software components. Formerly called *OLE*.

ActiveX control — Standard software tool that adds additional functionality to any compatible ActiveX container. The DAQ, UI, and analysis tools in ComponentWorks are all ActiveX controls. An ActiveX control has properties, methods, objects, and events.

AIPD	The National Instruments internal image format used for saving calibration information associated with an image and for saving complex images.
area threshold	Detects objects based on their size, which can fall within a user-specified range.
arithmetic operators	The image operations multiply, divide, add, subtract, and remainder.
asynchronous	Property of a function or operation that begins an operation and returns control to the program prior to the completion or termination of the operation.
auto-median function	A function that uses dual combinations of opening and closing operations to smooth the boundaries of objects.

B

binary image	An image containing objects usually represented with a pixel intensity of 1 (or 255) and the background of 0.
binary morphology	Functions that perform morphological operations on a binary image.
BMP	Image format commonly used for 8-bit images on PCs.
border function	Removes objects (or particles) in a binary image that touch the image border.

C

caliper	Finds edge pairs along a specified path in the image. This function performs an edge extraction and then finds edge pairs based on specified criteria such as the distance between the leading and trailing edges, edge contrasts, and so forth.
circle function	Detects circular objects in a binary image.
closing	A dilation followed by an erosion. A closing fills small holes in objects and smooths the boundaries of objects.

collection — Control property and object that contains a number of objects of the same type, such as pointers, axes, and plots. The type name of the collection is the plural of the type name of the object in the collection. For example, a collection of CWAxis objects is called CWAxes. To reference an object in the collection, you must specify the object as part of the collection, usually by index. For example, `CWGraph.Axes.Item(2)` is the second axis in the CWAxes collection of a graph.

color images — Images containing color information, usually encoded in the RGB form.

color lookup table — Table for converting the value of a pixel in an image into a red, green, and blue (RGB) intensity.

complex images — Save information obtained from the FFT of an image. The complex numbers which compose the FFT plane are encoded in 64-bit floating-point values: 32 bits for the real part and 32 bits for the imaginary part.

connectivity — Defines which of the surrounding pixels of a given pixel constitute its neighborhood.

connectivity-4 — Only pixels adjacent in the horizontal and vertical directions are considered neighbors.

connectivity-8 — All adjacent pixels are considered as neighbors.

control — Standard software tool that adds additional functionality to a compatible environment. ComponentWorks is a collection of ActiveX controls that have properties, methods, objects, and events.

convex function — Computes the convex regions of objects in a binary image.

convolution — *See* linear filter.

convolution kernel — Simple 3 × 3, 5 × 5, or 7 × 7 matrices (or templates) used to represent the filter in the filtering process. The contents of these kernels are a discrete two-dimensional representation of the impulse response of the filter that they represent.

D

Danielsson function — Similar to the distance functions, but with more accurate results.

density function — For each gray level in a linear histogram, it gives the number of pixels in the image that have the same gray level.

device	Plug-in data acquisition board that can contain multiple channels and conversion devices.
differentiation filter	Extracts the contours (edge detection) in gray level.
digital image	An image $f(x, y)$ that has been converted into a discrete number of pixels. Both spatial coordinates and brightness are specified.
dilation	Increases the size of an object along its boundary and removes tiny holes in the object.
distance calibration	Determination of the physical dimensions of a pixel by defining the physical dimensions of a line in the image.
distance function	Assigns to each pixel in an object a gray-level value equal to its shortest Euclidean distance from the border of the object.
driver	Software that controls a specific hardware device, such as a data acquisition board.

E

edge	Defined by a sharp change (transition) in the pixel intensities in an image or along an array of pixels.
edge contrast	The difference between the average pixel intensity before and the average pixel intensity after the edge.
edge hysteresis	The difference in threshold level between a rising and a falling edge.
edge steepness	The number of pixels that corresponds to the slope or transition area of an edge.
Equalize function	*See* histogram equalization.
erosion	Reduces the size of an object along its boundary and eliminates isolated points in the image.
event	Object-generated response to some action or change in state, such as a mouse click or x number of points being acquired. The event calls an event handler (callback function), which processes the event. Events are defined as part of an OLE control object.
event handler	*See* callback (function) *and* event.

exception	Error message generated by a control and sent directly to the application or programming environment containing the control.
exponential and gamma corrections	Expand the high gray-level information in an image while suppressing low gray-level information.
Exponential function	Decreases the brightness and increases the contrast in bright regions of an image, and decreases contrast in dark regions.

F

Fast Fourier Transform	A method used to compute the Fourier transform of an image.
FFT	Fast Fourier Transform.
form	Window or area on the screen on which you place controls and indicators to create the user interface for your program.
Fourier spectrum	The magnitude information of the Fourier transform of an image.
Fourier Transform	Transforms an image from the spatial domain to the frequency domain.
frequency filters	Counterparts of spatial filters in the frequency domain. For images, frequency information is in the form of spatial frequency.
FTP	File Transfer Protocol. Protocol based on TCP/IP to exchange files between computers.

G

Gaussian filter	A filter similar to the smoothing filter, but using a Gaussian kernel in the filter operation. The blurring in a Gaussian filter is more gentle than a smoothing filter.
gradient convolution filter	*See* gradient filter.
gradient filter	Extracts the contours (edge detection) in gray-level values. Gradient filters include the Prewitt and Sobel filters.
gray level	The brightness of a point (pixel) in an image.

gray-level dilation	Increases the brightness of pixels in an image that are surrounded by other pixels with a higher intensity.
gray-level erosion	Reduces the brightness of pixels in an image that are surrounded by other pixels with a lower intensity.
gray-level images	Images with monochrome information.
gray-level morphology	Functions that perform morphological operations on a gray-level image.

H

highpass attenuation	Inverse of lowpass attenuation.
highpass FFT filter	Removes or attenuates low frequencies present in the FFT domain of an image.
highpass filter	Emphasizes the intensity variations in an image, detects edges (or object boundaries), and enhances fine details in an image.
highpass frequency filter	Attenuates or removes (truncates) low frequencies present in the frequency domain of the image. A highpass frequency filter suppresses information related to slow variations of light intensities in the spatial image.
highpass truncation	Inverse of lowpass truncations.
histogram	Indicates the quantitative distribution of the pixels of an image per gray-level value.
histogram equalization	Transforms the gray-level values of the pixels of an image to occupy the entire range (0 to 255 in an 8-bit image) of the histogram, increasing the contrast of the image.
hit-miss function	Locates objects in the image similar to the pattern defined in the structuring element.
hole filling function	Fills all holes in objects that are present in a binary image.
HSL	Color encoding scheme in Hue, Saturation, and Lightness.
HSV	Color encoding scheme in Hue, Saturation, and Value.

I

image	A two-dimensional light intensity function $f(x, y)$, where, x and y denote spatial coordinates and the value f at any point (x, y) is proportional to the brightness at that point.
image file	A file containing image information and data.
image processing	Encompasses various processes and analysis functions which you can apply to an image.
image visualization	The presentation (display) of an image (image data) to the user.
inner gradient	Finds the inner boundary of objects.
inspection functions	Detects specific features in an image. The features detected include edges, peaks, and rotational shifts.
intensity calibration	Assigning user-defined quantities such as optical densities or concentrations to the gray-level values in an image.
intensity range	Defines the range of gray-level values in an object of an image.
intensity threshold	Characterizes an object based on the range of gray-level values in the object. If the intensity range of the object falls within the user specified range, it is considered an object; otherwise it is considered part of the background.
interpolation	Is the technique used to find values in between known values when resampling an image or array of pixels.

L

labeling	The process by which each object in a binary image is assigned a unique value. This process is useful for identifying the number of objects in the image and giving each object a unique identity.
Laplacian filter	Extracts the contours of objects in the image by highlighting the variation of light intensity surrounding a pixel.
line gauge	Measures the distance between selected edges with high-precision subpixel accuracy along a line in an image. For example, this function can be to measure distances between points and edges and vice versa. This function also can step and repeat its measurements across the image.

line profile	Represents the gray-level distribution along a line of pixels in an image.
linear filter	A special algorithm that calculates the value of a pixel based on its own pixel value as well as the pixel values of its neighbors. The sum of this calculation is divided by the sum of the elements in the matrix to obtain a new pixel value.
logarithmic and inverse gamma corrections	Expand low gray-level information in an image while compressing information from the high gray-level ranges.
Logarithmic function	Increases the brightness and contrast in dark regions of an image, and decreases the contrast in bright regions of the image.
Logic operators	The image operations AND, NAND, OR, XOR, NOR, difference, mask, mean, max, and min.
lookup table	Table containing values used to transform the gray-level values of an image. For each gray-level value in the image, the corresponding new value is obtained from the lookup table.
lowpass attenuation	Applies a linear attenuation to the frequencies in an image, with no attenuation at the lowest frequency and full attenuation at the highest frequency.
lowpass FFT filter	Removes or attenuates high frequencies present in the FFT domain of an image.
lowpass filter	Attenuates intensity variations in an image. You can use these filters to smooth an image by eliminating fine details and blurring edges.
lowpass frequency filter	Attenuates high frequencies present in the frequency domain of the image. A lowpass frequency filter suppresses information related to fast variations of light intensities in the spatial image.
lowpass truncation	Removes all frequency information above a certain frequency.
L-skeleton function	Uses an L-shaped structuring element in the Skeleton function.

M

mask	Isolates parts of an image for further processing.
mask filter	Removes frequencies contained in a mask (range) specified by the user.

mask image — An image containing a value of 1 and values of 0. Pixels in the source image with a corresponding mask image value of 1 are processed, while the others are left unchanged.

mechanical action — Specifies how a zone is activated. In the **Switch** mode, the first click on a zone turns the zone to TRUE and a second click turns it to FALSE. In the **Latch** mode, a click causes the zone to be temporarily TRUE.

median filter — A low pass filter that assigns to each pixel the median value of its neighbors. This filter effectively removes isolated pixels without blurring the contours of objects.

method — Function that performs a specific action on or with an object. The operation of the method often depends on the values of the object properties.

mile — An Imperial unit of length equal to 5,280 feet or 1,609.344 meters. Also known as a statute mile to discern from a nautical mile. *See also* nautical mile.

morphological transformations — Extract and alter the structure of objects in an image. You can use these transformations for expanding (dilating) or reducing (eroding) objects, filling holes, closing inclusions, or smoothing borders. They mainly are used to delineate objects and prepare them for quantitative inspection analysis.

M-skeleton — Uses an M-shaped structuring element in the skeleton function.

N

nautical mile — International unit of length used for sea and air navigation equal to 6,076.115 feet or 1,852 meters. *See also* mile.

neighborhood operations — Operations on a point in an image that take into consideration the values of the pixels neighboring that point.

nonlinear filter — Replaces each pixel value with a nonlinear function of its surrounding pixels.

nonlinear gradient filter — A highpass edge-extraction filter that favors vertical edges.

nonlinear Prewitt filter — A highpass edge-extraction filter that favors horizontal and vertical edges in an image.

nonlinear Sobel filter — A highpass edge-extraction filter that favors horizontal and vertical edges in an image.

Nth order filter — Filters an image using a nonlinear filter. This filter orders (or classifies) the pixel values surrounding the pixel being processed. The pixel being processed is set to the *N*th pixel value, where *N* is the order of the filter.

O

object — Software tool for accomplishing tasks in different programming environments. An object can have properties, methods, and events. You change an object's state by changing the values of its properties. An object's behavior consists of the operations (methods) that can be performed on it and the accompanying state changes.

See property, method, event.

Object Browser — Dialog window that displays the available properties and methods for the controls that are loaded. The object browser shows the hierarchy within a group of objects. To activate the object browser in Visual Basic, press <F2>.

OCX — OLE Control eXtension. Another name for ActiveX controls, reflected by the .OCX file extension of ActiveX control files.

OLE — Object Linking and Embedding. *See* ActiveX.

OLE control — *See* ActiveX control.

opening — An erosion followed by a dilation. An opening removes small objects and smoothes boundaries of objects in the image.

operators — Allow masking, combination, and comparison of images. You can use arithmetic and logic operators in IMAQ Vision.

optical representation — Contains the low-frequency information at the center and the high-frequency information at the corners of an FFT-transformed image.

outer gradient — Finds the outer boundary of objects.

P

palette — The gradation of colors used to display an image on screen, usually defined by a color lookup table.

PICT — Image format commonly used for 8-bit images on Macintosh and Power Macintosh platforms.

picture element — An element of a digital image.

pixel — Picture element.

pixel calibration — Directly calibrating the physical dimensions of a pixel in an image.

pixel depth — The number of bits used to represent the gray level of a pixel.

Power 1/Y function — Similar to a logarithmic function but with a weaker effect.

Power Y function — *See* exponential function.

Prewitt filter — Extracts the contours (edge detection) in gray-level values using a 3 × 3 filter kernel.

probability function — Defines the probability that a pixel in an image has a certain gray-level value.

proper-closing — A finite combination of successive closing and opening operations that you can use to fill small holes and smooth the boundaries of objects.

proper-opening — A finite combination of successive opening and closing operations that you can use to remove small particles and smooth the boundaries of objects.

property — Attribute that controls the appearance or behavior of an object. The property can be a specific value or another object with its own properties and methods. For example, a value property is the color (property) of a plot (object), while an object property is a specific Y axis (property) on a graph (object). The Y axis itself is another object with properties, such as minimum and maximum values.

Q

quantitative analysis — Obtaining various measurements of objects in an image.

R

reference	Link to an external code source in Visual Basic. References are anything that add additional code to your program, such as OLE controls, DLLs, objects, and type libraries. You can add references by selecting the **Tools»References…** menu.
region of interest	An area of the image that is graphically selected from a window displaying the image. This area can be used focus further processing.
Reverse function	Inverts the pixel values in an image, producing a photometric negative of the image.
RGB	Color image encoding using red, green, and blue colors.
RGB chunky	Color encoding scheme using red, green and blue (RGB) color information where each pixel in the color image is encoded using 32 bits: 8 bits for red, 8 bits for green, 8 bits for blue, and 8 bits for the alpha value (unused).
Roberts filter	Extracts the contours (edge detection) in gray level, favoring diagonal edges.
ROI	Region of interest.
rotational shift	The amount by which one image is rotated with respect to a reference image. This rotation is computed with respect to the center of the image.

S

segmentation function	Fully partitions a labeled binary image into non-overlapping segments, with each segment containing a unique object.
separation function	Separates objects that touch each other by narrow isthmuses.
shape matching	Finds objects in an image whose shape matches the shape of the object specified by a template. The matching process is invariant to rotation and can be set to be invariant to the scale of the objects.
Sigma filter	A highpass filter that outlines edges.
skeleton function	Applies a succession of thinning operations to an object until its width becomes one pixel.

skiz	Obtains lines in an image that separate each object from the others and are equidistant from the objects that they separate.
smoothing filter	Blurs an image by attenuating variations of light intensity in the neighborhood of a pixel.
Sobel filter	Extracts the contours (edge detection) in gray-level values using a 3 × 3 filter kernel.
spatial calibration	Assigning physical dimensions to the area of a pixel in an image.
spatial filters	Alter the intensity of a pixel with respect to variations in intensities of its neighboring pixels. You can use these filters for edge detection, image enhancement, noise reduction, smoothing, and so forth.
spatial resolution	The number of pixels in an image, in terms of the number of rows and columns in the image.
Square function	*See* exponential function.
Square Root function	*See* logarithmic function.
standard representation	Contains the low-frequency information at the corners and high-frequency information at the center of an FFT-transformed image.
start condition	Condition on a data acquisition process that determines when the actual acquisition starts. The condition can be a software trigger, an analog hardware trigger, or a digital hardware trigger.
statute mile	*See* mile.
stop condition	Condition on a data acquisition process that determines when the acquisition stops. The condition can be none (the acquisition stops when all points have been acquired), continuous (the acquisition runs continuously), software analog trigger, hardware analog trigger, or hardware digital trigger.
structuring element	A binary mask used in most morphological operations. A structuring element is used to determine which neighboring pixels contribute in the operation.
sub-pixel analysis	Used to find the location of the edge coordinates in terms of fractions of a pixel.

synchronous — Property or operation that begins an operation and returns control to the program only when the operation is complete.

syntax — Set of rules to which statements must conform in a particular programming language.

T

thickening — Alters the shape of objects by adding parts to the object that match the pattern specified in the structuring element.

thinning — Alters the shape of objects by eliminating parts of the object that match the pattern specified in the structuring element.

threshold — Separates objects from the background by assigning all pixels with intensities within a specified range to the object and the rest of the pixels to the background. In the resulting binary image, objects are represented with a pixel intensity of 255 and the background is set to 0.

threshold interval — Two parameters, the lower threshold gray-level value and the upper threshold gray-level value.

TIFF — Image format commonly used for encoding 8-bit and 16-bit images and color images on both Macintosh and PC platforms.

truth table — A table associated with a logic operator which describes the rules used for that operation.

Z

zones — Areas in a displayed image that respond to user clicks.

Index

Numbers

3D view, 10-8

A

acquisition and image processing (tutorial), 7-11 to 7-14
Active property
    common questions, A-1
    Feature Find application, 8-3 to 8-4
ActiveX controls. *See* controls.
Add operator (table), 12-2
advanced applications. *See* application development.
advanced binary morphology analysis functions. *See* binary morphology analysis functions.
alignment functions, IMAQ Vision (table), 7-6
alpha channel, of color images, 9-3
analysis functions, IMAQ Vision (table), 7-4
AND operator (table), 12-2
application development
    Delphi, 5-1 to 5-8
        building user interface, 5-4 to 5-6
        creating standalone objects, 5-8
        editing properties programmatically, 5-6
        events, 5-7 to 5-8
        loading ComponentWorks controls into palette, 5-2 to 5-3
        methods, 5-6 to 5-7
        programming with ComponentWorks, 5-5 to 5-8
        running examples, 5-1
    Feature Find application, 8-1 to 8-5
        AutoDelete, Active, and Visible properties, 8-3 to 8-4
        finding features and displaying results, 8-4 to 8-5
        manipulating regions of interest, 8-2 to 8-3
Floppy Disk Inspection application, 8-5 to 8-8
    edge detection and shape matching, 8-7 to 8-8
    manipulating regions of interest, 8-7
getting started, 2-4 to 2-6
Report objects, 8-8 to 8-9
testing and debugging, 8-9 to 8-14
    debugging, 8-13 to 8-14
    error and warning events, 8-12 to 8-13
    error checking, 8-9 to 8-10
    exceptions, 8-10 to 8-11
    return codes, 8-11 to 8-12
Visual Basic, 3-1 to 3-10
    automatic code completion, 3-9
    building user interface, 3-2 to 3-5
    creating standalone objects, 3-10
    developing event routines, 3-5 to 3-6
    loading ComponentWorks IMAQ Vision controls into toolbox, 3-2
    pasting code into programs, 3-8 to 3-9
    procedure overview, 3-1
    using Object Browser, 3-6 to 3-8
    working with methods, 3-5
Visual C++, 4-1 to 4-10
    adding ComponentWorks IMAQ Vision controls to toolbar, 4-3 to 4-4
    building user interface, 4-4 to 4-5
    creating standalone objects, 4-10
    events, 4-9 to 4-10
    methods, 4-8
    MFC AppWizard, 4-2 to 4-3
    procedure overview, 4-1

programming with ComponentWorks controls, 4-5 to 4-6
properties, 4-6 to 4-8
Application Wizard, MFC, 4-1, 4-2
area of digital objects, 16-5 to 16-7
holes' area, 16-6
number of holes, 16-6
number of pixels, 16-5
particle area, 16-5
particle number, 16-5
ratio, 16-6
scanned area, 16-6
total area, 16-6 to 16-7
area threshold, 16-4 to 16-5
arithmetic operator functions, IMAQ Vision (table), 7-3
arithmetic operators
list of operators (table), 12-2
overview, 12-1
asynchronous acquisition
IMAQ Hardware control, 6-8
tutorial, 6-12 to 6-13
attenuation
highpass FFT filters, 14-10
lowpass FFT filters, 14-7
AutoDelete property
common questions, A-1
Feature Find application, 8-3 to 8-4
automatic code completion, in Visual Basic, 3-9
auto-median function
binary morphology analysis, 15-20 to 15-21
gray-level morphology analysis, 15-36
axes. *See* chords and axes.
axis of symmetry, gradient filter, 13-6

B

binary morphology analysis functions
advanced, 15-21 to 15-36
border, 15-21
circle, 15-29
comparisons between segmentation and skiz functions, 15-27
convex, 15-30
Danielsson, 15-28 to 15-29
distance, 15-27
highpass filters, 15-22 to 15-23
hole filling, 15-21
labeling, 15-21
lowpass and highpass filter example, 15-23
lowpass filters, 15-22
segmentation, 15-26 to 15-27
separation, 15-24
skeleton, 15-24 to 15-26
primary, 15-7 to 15-21
auto-median, 15-20 to 15-21
closing, 15-11
dilation, 15-8
erosion, 15-8
erosion and dilation examples, 15-9 to 15-10
external and internal edge examples, 15-12
external edge, 15-12
hit-miss, 15-13 to 15-15
internal edge, 15-12
opening, 15-10 to 15-11
opening and closing examples, 15-11
proper-closing, 15-20
proper-opening, 15-19
thickening, 15-17 to 15-19
thinning, 15-15 to 15-17

binary palette, 10-3
border function, binary morphology analysis, 15-21
breadth parameter, 16-7
breakpoints, 8-13
broken line region, stopping drawing of, A-2
bulletin board support, D-1
B&W (gray) palette, 10-2

C

C++. *See* Visual C++.
calibration
    intensity, 16-2
    spatial, 16-1
caliper functions, IMAQ Vision (table), 7-6
center of mass X and center of mass Y, 16-8 to 16-9
chords and axes, 16-9 to 16-11
    max chord length, 16-10
    max intercept, 16-10
    mean chord X, 16-10
    mean chord Y, 16-10
    mean intercept perpendicular, 16-10
    particle orientation, 16-10 to 16-11
circle function, binary morphology analysis, 15-29
closing function
    binary morphology analysis
        description, 15-11
        examples, 15-11
    gray-level morphology analysis
        description, 15-33
        examples, 15-33 to 15-34
clustering technique, in automatic thresholding, 15-3 to 15-5
code completion, automatic, in Visual Basic, 3-9
collection objects, 1-6
color functions, IMAQ Vision (table), 7-5
color images
    histogram, 10-6
    overview, 9-3
    processing, 9-5 to 9-6
    standard formats, 9-3
    thresholding, 15-3
color lookup table (CLUT) transformation, 9-2. *See also* lookup table (LUT) transformations.
common questions, A-1 to A-2
compactness factor parameter, 16-15
complex functions, IMAQ Vision (table), 7-5
complex images, 9-3
ComponentWorks IMAQ Vision
    components, 1-1
    evaluation mode (note), 1-2
    examples structure, 2-4
    exploring documentation, 2-2
    getting started, 2-1 to 2-6
    installing
        NI-IMAQ driver software, 2-1
        procedure, 1-2 to 1-4
    online reference, 2-3
    overview, 1-1 to 1-2
    sources for additional information, 2-6
    system requirements, 1-2
ComponentWorks Support Web Site, 2-6
configuring image acquisition, 6-1
connectivity of digital objects, 16-3 to 16-4
    connectivity-4, 16-4
    connectivity-8, 16-3
contour extraction and highlighting, 13-14 to 13-15
contour thickness, 13-15 to 13-16
controls, 1-4 to 1-6. *See also* events; methods; properties.
    collection objects, 1-6
    IMAQ Hardware control, 6-5 to 6-9

loading into programming environments
Delphi, 5-2 to 5-3
Visual Basic, 3-2
Visual C++, 4-3 to 4-4
object hierarchy, 1-5 to 1-6
properties, methods, and events, 1-4 to 1-5
Viewer control, 6-2 to 6-5
Vision control, 7-1 to 7-14
convex function, 15-30
convolution, definition, 13-3
convolution filters. *See* linear or convolution filters.
convolution kernel, 13-3
coordinates of objects, 16-8 to 16-9
center of mass X and center of mass Y, 16-8 to 16-9
max chord X and max chord Y, 16-9
min(X, Y) and max(X, Y), 16-9
cumulative histogram, 10-6
custom property page
definition, 1-8
example (figure), 1-8
customer communication, *xxiii*, D-1 to D-2
CWIMAQ control. *See* IMAQ Hardware control.
CWIMAQViewer control. *See* Viewer control.

D

Danielsson function, 15-28 to 15-29
debug print commands, 8-13
debugging applications, 8-13 to 8-14
Delphi, 5-1 to 5-8
building user interface, 5-4 to 5-6
creating standalone objects, 5-8
editing properties programmatically, 5-6
events, 5-7 to 5-8
loading ComponentWorks controls into palette, 5-2 to 5-3
methods, 5-6 to 5-7
newest version of Delphi required (note), 5-1
programming with ComponentWorks, 5-5 to 5-8
running examples, 5-1
densitometry parameters, 16-18
developing applications. *See* application development.
Difference operator (table), 12-2
differentiation filter, 13-25
digital image processing, 9-1
digital images. *See also* color images; images.
image definition, 9-2
image resolution, 9-2
number of planes, 9-2 to 9-3
overview, 9-1
properties of digitized images, 9-1 to 9-3
digital objects
criteria for defining, 16-2 to 16-5
area threshold, 16-4 to 16-5
connectivity, 16-3 to 16-4
intensity threshold, 16-2
measurements, 16-5 to 16-19
areas, 16-5 to 16-7
chords and axes, 16-9 to 16-11
coordinates, 16-8 to 16-9
densitometry, 16-18
diverse measurements, 16-18 to 16-19
lengths, 16-7 to 16-8
shape equivalence, 16-11 to 16-14
shape features, 16-14 to 16-18
dilation function
binary morphology analysis
concept and mathematics, 15-8
examples, 15-9 to 15-10
gray-level morphology analysis
concept and mathematics, 15-31 to 15-32
examples, 15-32
direction, gradient filter, 13-6 to 13-7

distance function, binary morphology analysis, 15-27
distribution and redistribution files, C-1 to C-4. *See also* standalone objects, creating.
ComponentWorks IMAQ evaluation, C-3
distribution procedure, C-1 to C-3
automatic installers, C-2
manual installation, C-2 to C-3
files required, C-1
running on target computer, C-3
run-time licenses, C-3 to C-4
troubleshooting, C-4
diverse measurements, 16-18 to 16-19
Divide operator (table), 12-2
documentation. *See also* online reference.
conventions used in manual, *xxii-xxiii*
exploring, 2-2
organization of manual, *xix-xxii*
related documentation, *xxiii*

E

edge detection (example), 8-7 to 8-8
edge extraction and edge highlighting, 13-7 to 13-8
edge thickness, 13-9
electronic support services, D-1 to D-2
ellipse major axis parameter, 16-12
ellipse minor axis parameter, 16-13
ellipse ratio parameter, 16-13
elongation factor parameter, 16-14
e-mail support, D-2
entropy technique, in automatic thresholding, 15-5
Equalize function
description, 11-4
examples, 11-4 to 11-5
transfer function (table), 11-3
equivalent ellipse minor axis parameter, 16-12
erosion function
binary morphology analysis
concept and mathematics, 15-8
examples, 15-9 to 15-10
gray-level morphology analysis
concept and mathematics, 15-31
examples, 15-32
error and warning events, adding to applications, 8-12 to 8-13
error checking, adding to applications, 8-9 to 8-10
error codes
ComponentWorks errors (table), B-1
ComponentWorks IMAQ errors (table), B-1 to B-2
IMAQ errors (table), B-3 to B-5
Vision errors (table), B-5 to B-7
error handling, IMAQ Hardware control, 6-9
ErrorEventMask property, 6-9
event handler routines, developing
Delphi, 5-7 to 5-8
overview, 1-11
Visual Basic applications, 3-5 to 3-6
Visual C++ applications, 4-9 to 4-10
events
definition, 1-4
learning about, 1-11 to 1-12
Viewer events, 6-4 to 6-5
examples
becoming familiar with, 2-4
location of examples, 2-4
ExceptionOnError property, 6-9
exceptions, adding to applications, 8-10 to 8-11
exponential and gamma corrections
lookup table transformations, 11-8 to 11-10
transfer function (table), 11-3

Exponential function
example, 11-10
lookup transformations, 11-9
transfer functions (table), 11-3
external edge function, binary morphology analysis
description, 15-12
examples, 15-12

F

Fast Fourier Transform (FFT). *See also* FFT display.
definition, 14-3 to 14-4
of gray-level images, 9-3
obtaining frequency representation, 14-1
fax and telephone support numbers, D-2
Fax-on-Demand support, D-2
Feature Find application, 8-1 to 8-5
AutoDelete, Active, and Visible properties, 8-3 to 8-4
finding features and displaying results, 8-4 to 8-5
manipulating regions of interest, 8-2 to 8-3
FFT display, 14-4 to 14-6
optical representation, 14-6
standard representation, 14-5
files functions, IMAQ Vision (table), 7-2
files installed on hard disk, 1-3 to 1-4
filters. *See* frequency filters; spatial filters.
filters functions, IMAQ Vision (table), 7-3 to 7-4
finding features on printed circuit board. *See* Feature Find application.
Floppy Disk Inspection application, 8-5 to 8-8
edge detection and shape matching, 8-7 to 8-8
manipulating regions of interest, 8-7
frame. *See* image pixel frame.
frequency filters, 14-1 to 14-12
definition, 14-3 to 14-4
FFT display, 14-4 to 14-6
optical representation, 14-6
standard representation, 14-5
highpass FFT filters, 14-9 to 14-12
attenuation, 14-10
examples, 14-11 to 14-12
overview, 14-2
truncation, 14-10
lowpass FFT filters, 14-7 to 14-9
attenuation, 14-7
examples, 14-8 to 14-9
overview, 14-2
truncation, 14-8
mask FFT filters, 14-2
overview, 14-1 to 14-2
FTP support, D-1

G

Gaussian filters, 13-20 to 13-22
example, 13-20
kernel definition, 13-21
predefined kernels, 13-21 to 13-22
geometry functions, IMAQ Vision (table), 7-4
gradient filters
linear, 13-4 to 13-12
definition, 13-4
edge extraction and edge highlighting, 13-7 to 13-8
edge thickness, 13-9
example, 13-4 to 13-5
filter axis and direction, 13-6 to 13-7
kernel definition, 13-5
predefined gradient kernels, 13-10 to 13-12
Prewitt filters, 13-10
Sobel filters, 13-11
nonlinear, 13-25
gradient palette, 10-3
gray-level images, 9-3

gray-level morphology analysis functions, 15-31 to 15-36
auto-median, 15-36
closing, 15-33
dilation, 15-31 to 15-32
erosion, 15-31
erosion and dilation examples, 15-32
opening, 15-33
opening and closing examples, 15-33 to 15-34
proper-closing, 15-35
proper-opening, 15-34 to 15-35
gray-level value, 9-1

H

Hardware control. *See* IMAQ Hardware control.
height parameter, 16-8
help. *See* online reference.
hexagonal image pixel frame, 9-8
Heywood circularity factor, 16-15
highpass FFT filters, 14-9 to 14-12
attenuation, 14-10
examples, 14-11 to 14-12
overview, 14-2
truncation, 14-10
highpass filters
binary morphology analysis
description, 15-22 to 15-23
example, 15-23
classes (table), 13-3
definition, 13-1
histogram. *See* image histogram.
hit-miss function, binary morphology analysis, 15-13 to 15-15
concept and mathematics, 15-13
examples, 15-13 to 15-15
hole filling function, 15-21
holes' area parameter, 16-6
holes' perimeter parameter, 16-7
hydraulic radius parameter, 16-15

I

image acquisition configuration, 6-1
image definition, 9-2
image files, 9-5
image histogram
3D view, 10-8
cumulative histogram, 10-6
definition, 10-4 to 10-5
histogram of color images, 10-6
interpretation, 10-6
line profile, 10-8
linear histogram, 10-5
scale of histogram, 10-7
Image object
IMAQ Hardware control, 6-6 to 6-7
setting type property, A-1
shared between IMAQ Hardware control and Viewer control (figure), 6-7
image pixel frame, 9-6 to 9-8
arrangement (figure), 9-7
hexagonal frame, 9-8
rectangular frame, 9-7
image processing
acquisition and image processing (tutorial), 7-11 to 7-14
color images, 9-5 to 9-6
image resolution, 9-2
images. *See also* color images; digital images.
definition, 9-1
types and formats, 9-3 to 9-4
bytes per pixel (table), 9-4
color images, 9-3
complex images, 9-3
gray-level images, 9-3

IMAQ Hardware control, 6-5 to 6-9
asynchronous acquisition, 6-8
error handling, 6-9
ExceptionOnError and ErrorEventMask properties, 6-9
Image object, 6-6 to 6-7
IMAQ object, 6-6
methods and events, 6-8 to 6-9
object hierarchy, 6-2
example (figure), 6-6
overview, 6-1 to 6-2
synchronous acquisition, 6-8
tutorial
asynchronous, single-image acquisition and display, 6-12 to 6-13
synchronous, single-image acquisition and display, 6-10 to 6-12
IMAQ object
methods and events, 6-8 to 6-9
overview, 6-6
IMAQ User Interface control. *See* Viewer control.
installation, 1-2 to 1-4
Administrator privileges required (note), 1-2
files installed on hard disk, 1-3 to 1-4
from floppy disks, 1-3
NI-IMAQ driver software, 2-1 to 2-2
intensity calibration, 16-2
intensity threshold, 16-2
interclass variance technique, in automatic thresholding, 15-6
interface. *See* user interface, building.
internal edge function, binary morphology analysis
description, 15-12
examples, 15-12
Item method, setting properties, 1-10

L

labeling function, 15-21
Laplacian filters, 13-12 to 13-17
contour extraction and highlighting, 13-14 to 13-15
contour thickness, 13-15 to 13-16
example, 13-12 to 13-13
kernel definition, 13-13
predefined kernels, 13-16 to 13-17
length of digital objects, 16-7 to 16-8
breadth, 16-7
height, 16-8
holes' perimeter, 16-7
particle perimeter, 16-7
line profile, 10-8
linear histogram, 10-5
linear or convolution filters, 13-3 to 13-22
classes (table), 13-3
Gaussian filters, 13-20 to 13-22
example, 13-20
kernel definition, 13-21
predefined kernels, 13-21 to 13-22
gradient filters, 13-4 to 13-12
edge extraction and edge highlighting, 13-7 to 13-8
edge thickness, 13-9
example, 13-4 to 13-5
filter axis and direction, 13-6 to 13-7
kernel definition, 13-5
predefined gradient kernels, 13-10 to 13-12
Prewitt filters, 13-10
Sobel filters, 13-11
Laplacian filters, 13-12 to 13-17
contour extraction and highlighting, 13-14 to 13-15
contour thickness, 13-15 to 13-16
example, 13-12 to 13-13
kernel definition, 13-13
predefined kernels, 13-16 to 13-17

mathematical concepts, 13-2
overview, 13-3 to 13-4
smoothing filters, 13-17 to 13-20
example, 13-17
kernel definition, 13-18 to 13-19
predefined kernels, 13-19 to 13-20
logarithmic and inverse gamma corrections, 11-6 to 11-8
examples, 11-7 to 11-8
Logarithmic, Square Root, and Power 1/Y functions, 11-7
transfer function (table), 11-3
Logarithmic function
example, 11-8
lookup transformations, 11-7
transfer functions (table), 11-3
logic operator functions, IMAQ Vision (table), 7-3
logic operators, 12-2 to 12-7
examples, 12-5 to 12-7
extracting and removing information (example), 12-3
list of operators (table), 12-2 to 12-3
truth tables, 12-4
lookup table (LUT) transformations, 11-1 to 11-10
color lookup table transformation, 9-2
definition, 11-1
example, 11-2 to 11-3
predefined lookup tables
Equalize function, 11-4 to 11-5
exponential and gamma corrections, 11-8 to 11-10
logarithmic and inverse gamma corrections, 11-6 to 11-8
Reverse function, 11-5 to 11-6
summary (table), 11-3
purpose and use, 11-1 to 11-2
lowpass FFT filters, 14-7 to 14-9
attenuation, 14-7
examples, 14-8 to 14-9
overview, 14-2
truncation, 14-8
lowpass filters
binary morphology analysis
description, 15-22
example, 15-23
classes (table), 13-3
definition, 13-1
nonlinear, 13-26
L-skeleton function, 15-25
LUT. *See* lookup table (LUT) transformations.

M

manual. *See* documentation.
mask FFT filters, 14-2
Mask operator (table), 12-2
max chord length parameter, 16-10
max chord X and max chord Y parameter, 16-9
max intercept parameter, 16-10
Max operator (table), 12-3
mean chord X parameter, 16-10
mean chord Y parameter, 16-10
mean intercept perpendicular parameter, 16-10
Mean operator (table), 12-3
median filter, 13-27
methods
definition, 1-4
learning about, 1-11 to 1-12
setting properties, 1-10 to 1-11
Viewer object, 6-3
working with control methods
Delphi, 5-6 to 5-7
overview, 1-10 to 1-11
Visual Basic, 3-5
Visual C++, 4-8
metric technique, in automatic thresholding, 15-5
Microsoft Foundation Classes Application (MFC) Wizard, 4-1, 4-2
Min operator (table), 12-3

min(X, Y) and max(X, Y) parameter, 16-9
moments of inertia I_{xx}, I_{yy}, I_{xy}, 16-14
moments technique, in automatic thresholding, 15-5
morphological transformations
categories, 15-1
definition, 15-1
morphology analysis, 15-1 to 15-36
advanced binary functions, 15-21 to 15-36
border, 15-21
circle, 15-29
comparisons between segmentation and skiz functions, 15-27
convex, 15-30
Danielsson, 15-28 to 15-29
distance, 15-27
highpass filters, 15-22 to 15-23
hole filling, 15-21
labeling, 15-21
lowpass and highpass filter example, 15-23
lowpass filters, 15-22
segmentation, 15-26 to 15-27
separation, 15-24
skeleton, 15-24 to 15-26
gray-level functions, 15-31 to 15-36
auto-median, 15-36
closing, 15-33
dilation, 15-31 to 15-32
erosion, 15-31
erosion and dilation examples, 15-32
opening, 15-33
opening and closing examples, 15-33 to 15-34
proper-closing, 15-35
proper-opening, 15-34 to 15-35
overview, 15-1
primary binary functions, 15-7 to 15-21
auto-median, 15-20 to 15-21
closing, 15-11
dilation, 15-8
erosion, 15-8
erosion and dilation examples, 15-9 to 15-10
external and internal edge examples, 15-12
external edge, 15-12
hit-miss, 15-13 to 15-15
internal edge, 15-12
opening, 15-10 to 15-11
opening and closing examples, 15-11
proper-closing, 15-20
proper-opening, 15-19
thickening, 15-17 to 15-19
thinning, 15-15 to 15-17
structuring element, 15-6 to 15-7
thresholding, 15-1 to 15-6
automatic, 15-3 to 15-6
clustering, 15-3 to 15-5
color image, 15-3
entropy, 15-5
example, 15-2
interclass variance, 15-6
metric, 15-5
moments, 15-5
morphology functions, IMAQ Vision (table), 7-4
M-skeleton function, 15-25
Multiply operator (table), 12-2

N

NAND operator (table), 12-2
NI-IMAQ driver software, installing, 2-1 to 2-2
nonlinear filters, 13-22 to 13-28
classes (table), 13-3
differentiation filter, 13-25
gradient filter, 13-25
lowpass filter, 13-26
mathematical concepts, 13-2

median filter, 13-27
Nth order filter, 13-27 to 13-28
Prewitt filter, 13-22 to 13-23
Roberts filter, 13-25
Sigma filter, 13-26
Sobel, 13-23 to 13-24
NOR operator (table), 12-2
Nth order filter
description, 13-27
examples, 13-28
number of holes parameter, 16-6
number of pixels parameter, 16-5

O

Object Browser, in Visual Basic, 3-6 to 3-8
object creation functions, IMAQ Vision (table), 7-6
object hierarchy
example (figure), 1-6
IMAQ Hardware control, 6-2
overview, 1-5
Viewer control, 6-2
objects. *See* digital objects.
OLE controls. *See* controls.
online reference
accessing, 1-1, 1-11, 2-3
ComponentWorks Support Web Site, 2-6
finding specific information, 2-3
learning about properties, methods, and events, 1-11 to 1-12
overview, 2-3
searching complete text, 2-6
opening function
binary morphology analysis
description, 15-10 to 15-11
opening and closing examples, 15-11
gray-level morphology analysis
description, 15-33
examples, 15-33 to 15-34
operators, 12-1 to 12-7
arithmetic operators (table), 12-2
concepts and mathematics, 12-1
logic operators, 12-2 to 12-7
examples, 12-5 to 12-7
extracting and removing information (example), 12-3
list of operators (table), 12-2 to 12-3
truth tables, 12-4
optical representation, FFT display, 14-6
OR operator (table), 12-2

P

Palette object
properties and methods, 6-4
Viewer control, 6-2
palettes, 10-1 to 10-4
binary palette, 10-3
B&W (gray) palette, 10-2
definition, 10-1
gradient palette, 10-3
purpose and use, 10-1 to 10-2
rainbow palette, 10-3
temperature palette, 10-3
particle analysis (tutorial), 7-10 to 7-11
particle area parameter, 16-5
particle number parameter, 16-5
particle orientation parameter, 16-10 to 16-11
particle perimeter parameter, 16-7
pasting code, in Visual Basic, 3-8 to 3-9
picture element, 9-1
pixel depth, 9-2
pixels, 9-1. *See also* image pixel frame.
planes, number of, in digital images, 9-2 to 9-3
polygon region, stopping drawing of, A-2
Power 1/Y function
examples, 11-8
lookup transformations, 11-7
transfer functions (table), 11-3

Power Y function
example, 11-10
lookup transformations, 11-9
transfer functions (table), 11-3
Prewitt filters
nonlinear, 13-22 to 13-23
predefined gradient filter, 13-10
primary binary morphology analysis functions. *See* binary morphology analysis functions.
processing functions, IMAQ Vision (table), 7-3
proper-closing function
binary morphology analysis, 15-20
gray-level morphology analysis, 15-35
proper-opening function
binary morphology analysis, 15-19
gray-level morphology analysis, 15-34 to 15-35
properties
definition, 1-4
editing programmatically
Delphi, 5-6
overview, 1-9
Visual Basic, 3-4 to 3-5
learning about, 1-11 to 1-12
setting, 1-7 to 1-11
developing event handler routines, 1-11
Item method, 1-10
using property pages, 1-7 to 1-9
working with methods, 1-10 to 1-11
using in programming environments
Delphi, 5-4 to 5-5
Visual Basic, 3-3 to 3-5
Visual C++, 4-6 to 4-8
property pages
custom property page
definition, 1-8
example (figure), 1-8
setting properties for controls, 1-7 to 1-8
Visual Basic default property sheets (figure), 1-8

Q

quantitative analysis, 16-1 to 16-19
definition of digital objects, 16-2 to 16-5
area threshold, 16-4 to 16-5
connectivity, 16-3 to 16-4
intensity threshold, 16-2
intensity calibration, 16-2
object measurements, 16-5 to 16-19
areas, 16-5 to 16-7
chords and axes, 16-9 to 16-11
coordinates, 16-8 to 16-9
densitometry, 16-18
diverse measurements, 16-18 to 16-19
lengths, 16-7 to 16-8
shape equivalence, 16-11 to 16-14
shape features, 16-14 to 16-18
spatial calibration, 16-1
questions about ComponentWorks, A-1 to A-2

R

rainbow palette, 10-3
ratio area parameter, 16-6
reading image from file and thresholding (tutorial), 7-7 to 7-9
rectangle big side parameter, 16-13
rectangle ratio parameter, 16-14
rectangle small side parameter, 16-13
rectangular image pixel frame, 9-7
redistribution files. *See* distribution and redistribution files.
Regions collection
properties and methods, 6-4
in Viewer control, 6-2
Regions object, 6-2

regions of interest, manipulating
Feature Find application, 8-2 to 8-3
programmatically (example), 8-7
Remainder operator (table), 12-2
Report objects, 8-8 to 8-9
requirements for getting started, 1-2
return codes, adding to applications, 8-11 to 8-12
Reverse function, 11-5 to 11-6
display of photometric negative of image (figure), 11-5
example, 11-6
transfer function (table), 11-3
RGB intensities, 9-2
RGB-chunky standard, 9-3
Roberts filter, 13-25
run-time licenses, C-3 to C-4

S

scale of histogram, 10-7
scanned area parameter, 16-6
search function, Vision control (table), 7-6
segmentation function
compared with skiz function, 15-27
description, 15-26 to 15-27
separation function, binary morphology analysis, 15-24
shape equivalence, 16-11 to 16-14
ellipse major axis, 16-12
ellipse minor axis, 16-13
ellipse ratio, 16-13
equivalent ellipse minor axis, 16-12
rectangle big side, 16-13
rectangle ratio, 16-14
rectangle small side, 16-13
shape features, 16-14 to 16-18
compactness factor, 16-15
elongation factor, 16-14
Heywood circularity factor, 16-15
hydraulic radius, 16-15
moments of inertia I_{xx}, I_{yy}, I_{xy}, 16-14
Waddel disk diameter, 16-16 to 16-18
derived measurements (table), 16-16 to 16-18
primary measurements, 16-16
shape matching (example), 8-7 to 8-8
Sigma filter, 13-26
single stepping, for debugging applications, 8-14
skeleton functions, 15-24 to 15-26
comparison between segmentation and skiz functions, 15-27
L-skeleton, 15-25
M-skeleton, 15-25
skiz, 15-26
skiz function
compared with segmentation function, 15-27
description, 15-26
smoothing filters, 13-17 to 13-20
example, 13-17
kernel definition, 13-18 to 13-19
predefined kernels, 13-19 to 13-20
Sobel filters
nonlinear, 13-23 to 13-24
predefined gradient filters, 13-11
software object, 1-5
spatial calibration, 16-1
spatial filters, 13-1 to 13-28
categories, 13-1
classification summary (table), 13-3
concepts and mathematics, 13-1 to 13-3
definition, 13-1
linear or convolution filters, 13-3 to 13-22
Gaussian filters, 13-20 to 13-22
gradient filter, 13-4 to 13-12
Laplacian filters, 13-12 to 13-17
smoothing filters, 13-17 to 13-20
nonlinear filters, 13-22 to 13-28
differentiation filter, 13-25
gradient filter, 13-25

lowpass filter, 13-26
median filter, 13-27
Nth order filter, 13-27 to 13-28
Prewitt filter, 13-22 to 13-23
Roberts filter, 13-25
Sigma filter, 13-26
Sobel, 13-23 to 13-24
spatial resolution, 9-2
Square function
example, 11-10
lookup transformations, 11-9
transfer functions (table), 11-3
Square Root function
example, 11-8
lookup transformations, 11-7
transfer functions (table), 11-3
standalone objects, creating. *See also* distribution and redistribution files.
Delphi applications, 5-8
Visual Basic applications, 3-10
Visual C++ applications, 4-10
standard representation, FFT display, 14-5
step into, 8-14
step over, 8-14
structuring element, in morphology analysis, 15-6 to 15-7
Subtract operator (table), 12-2
synchronous acquisition
IMAQ Hardware control, 6-8
tutorial, 6-10 to 6-12
system requirements, 1-2

T

technical support, D-1 to D-2
telephone and fax support numbers, D-2
temperature palette, 10-3
testing and debugging applications, 8-13 to 8-14
thickening function, binary morphology analysis, 15-17 to 15-19
description, 15-17
examples, 15-18 to 15-19
thinning function, binary morphology analysis, 15-15 to 15-17
description, 15-15 to 15-16
examples, 15-16 to 15-17
three dimensional (3D) view, 10-8
threshold
area threshold, 16-4 to 16-5
intensity threshold, 16-2
thresholding, 15-1 to 15-6
automatic, 15-3 to 15-6
clustering, 15-3 to 15-5
entropy, 15-5
interclass variance, 15-6
metric, 15-5
moments, 15-5
color image, 15-3
example, 15-2
overview, 15-1 to 15-2
reading an image from file and thresholding (tutorial), 7-7 to 7-9
tools and utilities. *See* image histogram; palettes.
tools functions, IMAQ Vision (table), 7-2 to 7-3
total area parameter, 16-6 to 16-7
transformation functions. *See* morphology analysis.
troubleshooting distribution and redistribution files, C-4
truncation
highpass FFT filters, 14-10
lowpass FFT filters, 14-8
truth tables, 12-4

U

user interface, building
- Delphi applications, 5-4 to 5-5
  - placing controls, 5-4
  - using property pages, 5-4 to 5-5
- Visual Basic applications, 3-2 to 3-5
  - editing properties programmatically, 3-4 to 3-5
  - using property pages, 3-3 to 3-4
- Visual C++ applications, 4-4 to 4-5

V

Viewer control, 6-2 to 6-5
- common questions, A-1
- events, 6-4 to 6-5
- object hierarchy, 6-2
  - example (figure), 6-3
- objects in Viewer control, 6-2
- overview, 6-1 to 6-2
- Palette object, 6-4
- Regions collection, 6-4
- setting redraw rate, A-2
- sizing, A-1
- tutorial
  - asynchronous, single-image acquisition and display, 6-12 to 6-13
  - synchronous, single-image acquisition and display, 6-10 to 6-12

Viewer object
- definition, 6-2
- methods, 6-3
- properties, 6-3

Visible property, Feature Find application, 8-3 to 8-4

Vision control, 7-1 to 7-14
- function summary (table), 7-2 to 7-6
  - alignment, 7-6
  - analysis, 7-4
  - arithmetic operators, 7-3
  - caliper, 7-6
  - color, 7-5
  - complex, 7-5
  - files, 7-2
  - filters, 7-3 to 7-4
  - geometry, 7-4
  - logic operators, 7-3
  - morphology, 7-4
  - object creation, 7-6
  - processing, 7-3
  - search, 7-6
  - tools, 7-2 to 7-3
- overview, 7-1
- tutorial, 7-7 to 7-14
  - acquisition and image processing, 7-11 to 7-14
  - particle analysis, 7-10 to 7-11
  - reading image from file and thresholding, 7-7 to 7-9

Visual Basic, 3-1 to 3-10
- automatic code completion, 3-9
- building user interface, 3-2 to 3-5
  - editing properties programmatically, 3-4 to 3-5
  - using property pages, 3-3 to 3-4
- creating standalone objects, 3-10
- default property sheets (figure), 1-8
- developing event routines, 3-5 to 3-6
- loading ComponentWorks IMAQ Vision controls into toolbox, 3-2
- pasting code into programs, 3-8 to 3-9
- procedure overview, 3-1
- using Object Browser, 3-6 to 3-8
- working with methods, 3-5

Visual C++, 4-1 to 4-10
adding ComponentWorks IMAQ Vision controls to toolbar, 4-3 to 4-4
building user interface, 4-4 to 4-5
creating standalone objects, 4-10
events, 4-9 to 4-10
methods, 4-8
MFC AppWizard, 4-2 to 4-3
procedure overview, 4-1
programming with ComponentWorks controls, 4-5 to 4-6
properties, 4-6 to 4-8

W

Waddel disk diameter, 16-16 to 16-18
derived measurements (table), 16-16 to 16-18
primary measurements, 16-16
warning events, adding to applications, 8-12 to 8-13
watch windows, 8-14
Web site for ComponentWorks support, 2-6

X

XOR operator (table), 12-2