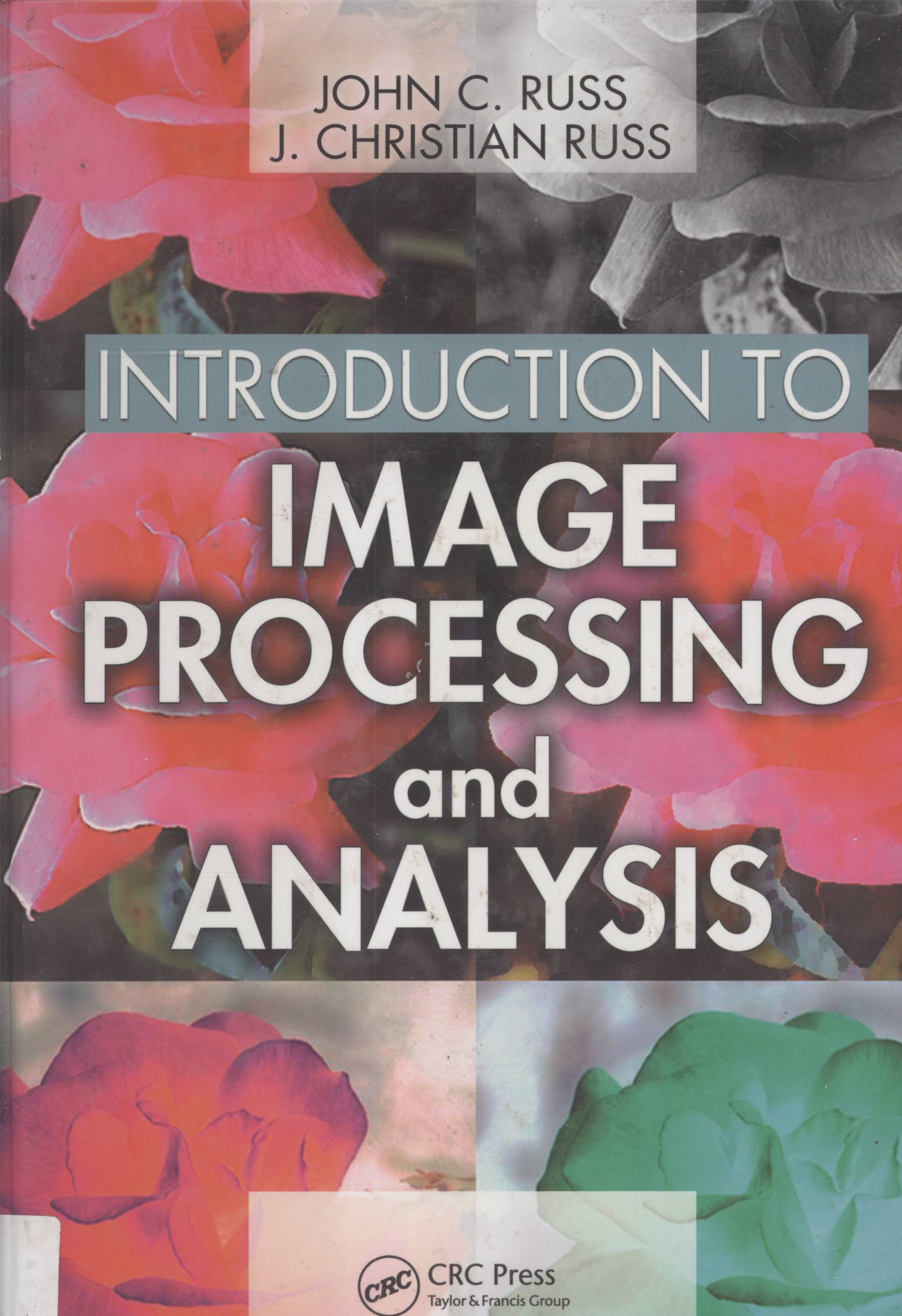
JOHN C. RUSS
J. CHRISTIAN RUSS

INTRODUCTION TO IMAGE PROCESSING and ANALYSIS



CRC Press
Taylor & Francis Group

TN911.73
R958

INTRODUCTION TO IMAGE PROCESSING and ANALYSIS

JOHN C. RUSS
J. CHRISTIAN RUSS

理工大学
图书馆藏书

E2009002796

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an informa business

附光盘壹张

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2008 by Taylor & Francis Group, LLC
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works
Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2 1

International Standard Book Number-13: 978-0-8493-7073-1 (Hardcover)

This book contains information obtained from authentic and highly regarded sources. Reprinted material is quoted with permission, and sources are indicated. A wide variety of references are listed. Reasonable efforts have been made to publish reliable data and information, but the author and the publisher cannot assume responsibility for the validity of all materials or for the consequences of their use.

Except as permitted under U.S. Copyright Law, no part of this book may be reprinted, reproduced, transmitted, or utilized in any form by any electronic, mechanical, or other means, now known or hereafter invented, including photocopying, microfilming, and recording, or in any information storage or retrieval system, without written permission from the publishers.

For permission to photocopy or use material electronically from this work, please access www.copyright.com (http://www.copyright.com/) or contact the Copyright Clearance Center, Inc. (CCC) 222 Rosewood Drive, Danvers, MA 01923, 978-750-8400. CCC is a not-for-profit organization that provides licenses and registration for a variety of users. For organizations that have been granted a photocopy license by the CCC, a separate system of payment has been arranged.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation without intent to infringe.

Library of Congress Cataloging-in-Publication Data

Russ, John C.
Introduction to image processing and analysis / John C. Russ and J. Christian Russ.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-8493-7073-1 (hardback : alk. paper) 1. Image processing. I. Russ, Christian. II. Title.

TA1637.R86 2008
621.36'7--dc22 2007016992

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

For Helen, Jenn, and Colette

Introduction

Assumptions

This text is intended to introduce students to the programming involved in image processing and analysis. The student is assumed to have some previous programming experience, and to be able to use a C compiler and programming environment. Either a Windows (Win 2K or later) or Macintosh (OS X) computer may be used. The concentration of topics here is on the processing and measurement of images, not on peripheral subjects such as the wide variety of file formats in which images are stored, nor the display and printing of images, nor the statistical techniques for interpreting measurement data. This text is not intended to be an encyclopedia of image processing (see the References for appropriate choices), but is instead focused on the implementation of many of the most widely used and most important image processing and analysis algorithms, which requires also an understanding of their results and purpose.

Unlike many texts that deal with image processing algorithms, this book does not concentrate on the mathematical underpinnings of the field (this is particularly true in the section on Fourier transforms). Rather, it introduces just enough of the math to explain the workings of the algorithms while emphasizing the practical reasons for the use of the methods, their effects on images, and their appropriate applications. Also, the intent in the chapters that follow is to combine image processing with image analysis. As indicated in Figure 0.1, ***image processing*** comprises a broad variety of methods that operate on images to produce another image. The changes that are introduced are generally intended to improve the visibility of features and detail, or to improve the images for printing or transmission, or to facilitate subsequent analysis. Chapters 1, 2 and 3 deal with the algorithms used in these processing steps. ***Image analysis*** is the process of obtaining numerical data from images. This is usually accomplished by a combination of measurement and processing operations, as described in Chapters 4 and 5. The data obtained by measurement may subsequently be analyzed statistically, or used to generate graphs or other visualizations.

The Program Environment

The selection of a host program for running the routines to be developed has an important consequence for the choice of a working environment to be used for writing them. We wanted to eliminate the need to create an entire working program that can acquire images from cameras or scanners, read various image file formats, and display and print images before any image processing and analysis routines could be introduced. Accordingly, the decision was taken to make use of a widely used program as a host. Adobe Photoshop® handles all of those tasks, presents a consistent user appearance on both Mac and Windows computers, and has a well-documented application program interface (API, the relevant

试读结束，需要全本PDF请购买 www.ertongbook.com

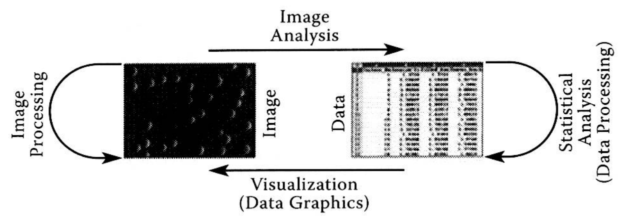


Image Analysis
Image Processing
Image
Data
Statistical Analysis (Data Processing)
Visualization (Data Graphics)

FIGURE 0.1
The relationships between image processing, image analysis, data processing, and data visualization.

portions of which are described in the Appendix) for "plug-ins" that can access the image data for processing and measurement.

The plug-ins themselves are dynamically linked libraries (DLLs) that can be separately compiled and placed in a folder where the program accesses them. Each individual plug-in can be dedicated to a specific function, which is listed as a separate menu item. This makes it possible for the student to easily and quickly compile and test routines, and to build a library of functions that can be applied in sequences and combinations to accomplish more complex tasks.

Furthermore, the API defined by Adobe engineers for Photoshop-compatible plug-ins has been adopted by many other programs, ranging from very inexpensive ones (e.g., Adobe Photoshop Elements®, Corel Paint Shop Pro®, Lemke Graphic Converter®) to other graphic arts packages (e.g., Deneba Canvas®, Corel Painter®) and even professional image analysis packages (e.g., Media Cybernetics Image-Pro Plus®). Several of these programs (including Photoshop) are available at reduced cost for educational and student use, or may be available on a campuswide license. The widespread use of Photoshop and the many compatible programs means that students gaining experience with programming image processing routines using this text will be directly able to use the same knowledge in many practical settings.

The use of the standard C programming language ensures a consistent style and knowledge base with other programming courses, and allows students to take advantage of the skills they have already acquired. Unlike image processing texts that use languages like Basic or Java, the use of C is consistent with professional applications, and it ensures clarity, simplicity, flexibility, and good performance. The use of C++ would have added unnecessary overhead and complexity. The interface code (often referred to as the "glue" that implements the API and binds the plug-ins to the host program) and the utilities provided on the companion CD handle the details of converting various image modes to a common one and provide the necessary supporting routines, which allows the student to concentrate on the central topics of image processing and analysis, rather than spending time on other programming tasks.

Specialized image processing texts that use Matlab®, Mathematica®, Mathcad®, and other mathematics and visualization packages make use of highly optimized and preprogrammed subroutines that can be invoked, typically by a complex command line syntax that does little or nothing to help the student understand the actual procedures or to create new ones. Indeed, there is no effort made toward optimization or efficiency in the code examples used in this text. Instead, relying on the fact that computers have become quite fast in recent years, the emphasis is entirely on clarity and consistency. Once the principles are understood and the basic routines are implemented, the student who is interested in optimization of either speed or memory usage (for example) is welcome to explore those possibilities.

The plug-in interface used for the examples and projects covered by this text can be used with all versions of Photoshop from 5.0 and later (the current version as of this writing is Photoshop CS3, aka Photoshop 10). Some of the routines that are programmed in the examples shown here duplicate basic image processing routines that are built into Photoshop (and the other programs named above). For example, just about every image processing program has a Gaussian smoothing function. However, it is still important for the student to understand how the function can be implemented as well as what effect it has on the image.

Other functions covered in this text extend standard routines in novel and very useful ways. An example is the commonly used median filter, and the conditional versions introduced in this text. Still other routines that are developed in the text go beyond the basics to provide advanced processing and measurement capabilities.

This text does not try to present an encyclopedic compendium of image processing algorithms, nor to overwhelm the reader with their mathematical complexities, notations, or derivations. The emphasis is on clear explanations of the most commonly used and most universally useful techniques, with examples. The examples include both code fragments and illustrations of results, as well as appropriate diagrams and necessary equations. The student who masters these basic tools will be able to build upon that knowledge base to handle other techniques he or she may encounter later on, in the extensive literature covering this dynamic and rapidly expanding field.

Image Values

Images may be acquired from many sources, including digital cameras and desktop scanners as well as scientific instruments, reconnaissance satellites and, of course, via the Internet. Each image is an array (generally a rectangular array) of ***pixels*** (short for picture elements), and each pixel contains information. The address of each pixel within the image is usually specified as an (x,y) pair, with x indicating the distance from the left edge and y indicating the distance down from the top, both values starting at zero and increasing to (width – 1) and (height – 1), respectively. Measuring y downward from the top rather than up from the bottom is a historical convention that arose because of the way that computer displays (and television sets) are scanned.

Note: Key words that are frequently encountered in image processing and analysis are highlighted in the text, as is ***pixels*** in the preceding paragraph. It is recommended that the student become familiar with them. The index also highlights the pages where these key words are defined.

In some cases, the information associated with each pixel may be a single value, representing the grayscale (monochrome) brightness of that point in a scene. The most common type of image encountered will contain color information, which the pixel may represent as a combination of red, green, and blue (RGB) values. Other combinations of values are possible and, indeed, very useful for processing as will be seen, but the RGB triplet is the most common format and is used in Photoshop and most image processing programs. It also corresponds to the colors used in cathode ray tube (CRT) and liquid crystal (LCD) displays used to present images to the viewer, and in most cameras or scanners that are used to acquire images.

Some image sources may provide more than three values at each pixel. For example, many satellite images consist of the visible red, green, and blue channels as well as several infrared values. Also, scientific instruments may have dozens or hundreds of channels, representing different signals or elemental composition, for instance. These are not dealt with explicitly in this text, but represent straightforward generalizations of three-channel RGB color images.

In many traditional image processing programs, the pixel values themselves are treated as integers that can range from 0 (black) to 255 (white). That artificial limitation was introduced historically because it corresponded to a single 8-bit byte of computer memory, was easily handled by integer math routines in slow CPUs, and was adequate to represent the dynamic range of signals from sources such as a video camera (which actually has fewer than 256 distinguishable levels). With the advent of digital cameras and scanners (not to mention scientific devices) with greater precision, and the appreciation that faster modern computers do not impose a significant penalty for using floating point arithmetic, it seems to us better to move beyond the "8-bit integer" limitation while still preserving a degree of consistency with the past.

Accordingly, the convention used for the image handling routines throughout this book is that the values can range from 0 (black) to 255 (maximum), but are not restricted to integer values. All values are inherently treated as floating point numbers. Furthermore, all images received from and returned to the host program are considered to be RGB color images. That means images which Photoshop considers to be 8 or 16 bits per channel, and either grayscale (monochrome) or RGB color, are all converted in the interface software to consist of three values per pixel, representing red, green, and blue, respectively, and with a floating point range from 0.0 to 255.0.

For an image that is actually monochrome, the red, green, and blue values will be identical. To confirm that this is true, simply examine the computer display with a magnifying glass. When a monochrome (grayscale) image is displayed, the signals sent to the red, green, and blue dots on the screen are equal in magnitude and visually blend to produce the impression of neutral grays. When red, green, and blue are all set to 255, the result is perceived as white. For an image with a precision greater than 8 bits, the pixel values in images will be represented as having fractional parts (e.g., 143.627).

Some of the plug-in routines will perform image processing by reading the original image from the host program, manipulating the pixel values, and writing the resulting image back to the host program. The conversion of values from whatever mode the image uses in the host program to the three-value floating point format, and back again, is handled by the interface software (which is fully documented in the Appendix).

As an example of a (trivial) program that reads, alters, and rewrites values in an image, Code Fragment 0.1 performs what most programs call "inverting" an image. This is not the inverse in the mathematical sense of 1.0 divided by the value, but rather replaces each value by 255 minus the value, and should more properly be called complementing or reversing the image values. For a monochrome (grayscale) image the result is like a photographic negative (Figure 0.2). For a color image, reds are replaced by cyans, blues by yellows and greens by magentas (Figure 0.3). The reasons for these complementary colors will become more apparent in the next chapter.

Code Fragment 0.1 – Complementing/reversing an image

```
// this struct is defined in the PhotoshopShell.h file
// it holds three floating point values for each pixel
typedef struct
{
   float red, green, blue;
}  RGBPixel;

// the following is the UserCode.c file for the plug-in
#include "PhotoshopShell.h"// include the interface 'glue' and utilities

ErrType  MainUserEntry()
{
   ErrType   result = noErr; // accumulate any errors that happen
   long      height, width;  // pixel dimensions of image
   long      x, y;           // loop counters to traverse image
   RGBPixel *line = NULL;    // memory buffer

   GetOriginalDimensions(&width, &height); // find out how big the image is
   // Allocate memory for one row of pixels
   line = CreateAPointer(width, sizeof(RGBPixel));
   //Image reversal example
   for (y = 0; y < height; y++)
   {  // test if user canceled (ESC/Cancel) & advance the progress bar
      if (DoProgressBarTestAbort(float)y/(float)height))// 0.0 .. 1.0
      {
         result = userCanceledErr;
         goto mainexit;
      }
      ReadOriginalLine(y, line);     // read a line of pixel values
      for (x = 0; x < width; x++)    // for each pixel on the line
      {  //reverse the values
         line[x].red =    255.0 - line[x].red;
         line[x].green =  255.0 - line[x].green;
         line[x].blue =   255.0 - line[x].blue;
      }//for x
      WriteResultLine(y, line);      //write changed line back
   }// for y

mainexit:                            // clean up everything we allocated
   if (line)  // test to see if the pointer is still null (not allocated)
      DisposeAPointer(line);
   return result;                    // pass any errors back
}// MainUserEntry
```

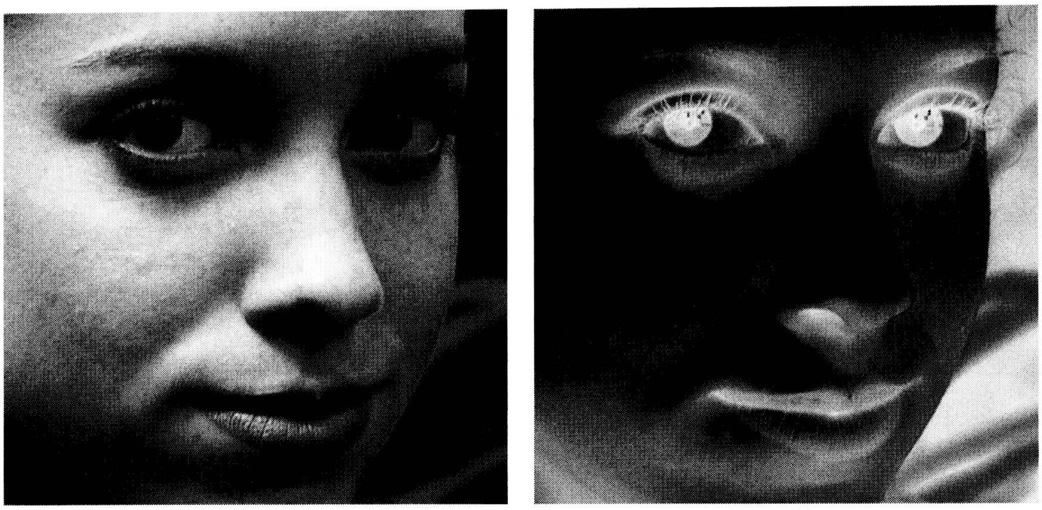


FIGURE 0.2
Reversing a monochrome image: (a) original [face.tif], (b) "inverse" or complement.

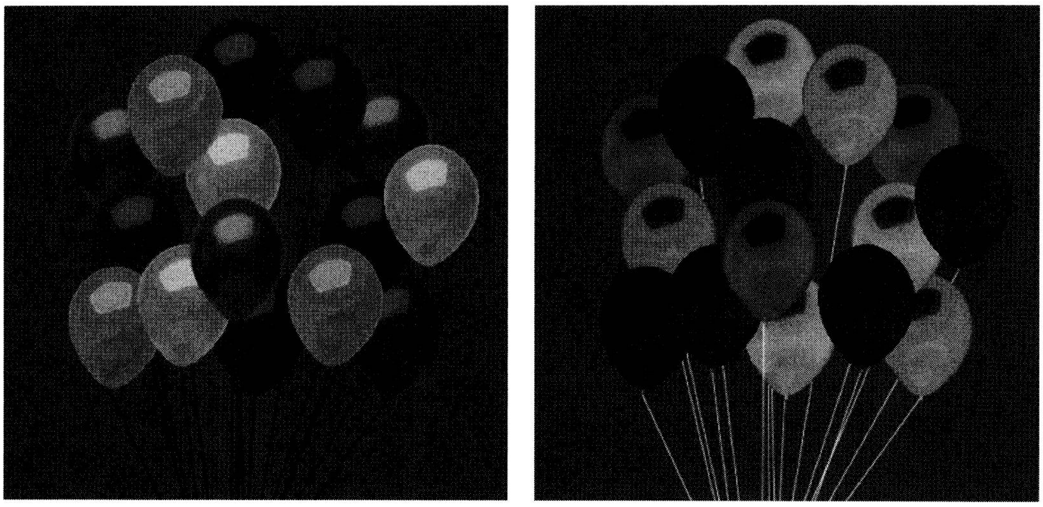


FIGURE 0.3
(See **Color insert following page 172.**) Reversing the values in a color image: (a) original [balloons.tif], (b) "inverse" or complement.

The code fragment shows the procedure for reading the image, one line of red-green-blue values at a time, followed by the actual processing of each pixel and, finally, rewriting each line to the host program. This line-by-line raster format corresponds to the way images are usually stored in memory, printed, displayed, and acquired. There are many reasons for this line-by-line format, some of them historical (the way televisions, cameras, and displays, and other hardware devices function) and some practical (the organization of computer memory and the efficiency that buffer memory in modern processors provides).

In the code fragment shown, a pointer is defined and memory allocated to hold one line of values (three floating point numbers for each pixel). Creating and disposing of pointers, and defining the other variables used in the code, is not shown explicitly in all of the code fragments in the text, but is assumed to be the responsibility of the student.

The interface convention used by Photoshop compatible plug-ins allows the original data to be read as many times as desired, but only to be rewritten once. (More specifically, you can write the values as many times as you want, but only the last time will matter, and reading will always obtain the original values, not reflecting any changes you have made.) Consequently, some procedures will need to keep a copy of the image data in memory to iterate upon it, and there are routines provided to create the necessary arrays in memory for that purpose. Those temporary image arrays may also contain three floating point values per pixel, but for some purposes it will be preferable to instead create arrays of a single brightness value per pixel, or a complex value with a real and imaginary part, according to the specific need.

Input and Output

Many image processing programs incorporate elaborate user interfaces with interactive dialogs that can be manipulated via mouse and keyboard to control parameters and choices used in the routines. They may also write files of data intended for use in spreadsheet and statistical analysis programs. Although very useful, these capabilities are secondary to the central interests of this text, which are the actual image processing and analysis.

For some purposes, changing the numerical or logical values of variables used in the various routines can be accomplished satisfactorily by recompiling the routines. A greater degree of flexibility is afforded by the ability to read numeric values from a simple text file, such as can be created with most word processors (including the editor used for programming). The support routines provided with this book (and documented in the Appendix) include the ability to open a text file, and to read one or more floating point numbers from it. Several of the examples and problems use this capability to allow making adjustments to internal variables as the routines are run.

There is a corresponding routine for output that will create and open a text file, and one to write a floating point number to it (followed by a line feed and carriage return). This can be used to create a text file containing data that can be opened in a spreadsheet, or most other data analysis and plotting programs. It is suggested that students use this basic capability along with a spreadsheet program such as Excel® to prepare graphs such as some of those shown in this text.

Compiling a Function

The following steps will allow you to create your own first image processing plug-in routine, by starting with an example project (the one that reverses or complements the image contrast, shown in the preceding code fragment) that is provided on the companion CD. The steps correspond to using Microsoft Visual Studio® on a Windows XP® or Windows 2K® computer, but can be easily adapted to other compilers or platforms. Additional

试读结束，需要全本PDF请购买 www.ertongbook.com

details and information can be found in the Appendix. More information, late additions, instructions for compiling plug-ins for the Macintosh, and other resources, can be found at the support Web site <www.intro2imaging.com>.

1. Duplicate the **`Example`** folder containing a minimal example project, and give the folder an appropriate name.
2. Click on the **`UserCode.sln`** file to open the project.
3. Edit the **`PiplData.h`** file. Remember to **`Save`** the result after editing. This file contains several items that control the appearance of the plug-in:
   a. The main menu category (**`R+R_Book`** by default) defines the name that will appear in the "Photoshop Filters" menu. A submenu with the individual filter plug-ins opens when this menu entry is selected. This name is a Pascal string that is always 32 characters long, so pad out a shorter entry with spaces as shown in the example and place the actual length of the string in hex at the beginning.
   b. The submenu name for the plug-in. The format for this string is identical to that for the category.
   c. A description for the "about" box. This is a C-string and may contain any printable characters, including the programmer's name. Inserting a '\n' in the string will start a new line.
   d. A unique signature for the plug-in that is used by Photoshop for actions and history. The format is four printable characters, at least one of which must be uppercase and one lowercase. The signature must be entered in both forward and reverse order, as shown in the example.
4. Edit the **`UserCode.c`** file to create the plug-in. The **`#include "PhotoshopShell.h"`** statement is necessary to bring in the various interface and support subroutine calls that are described later and illustrated throughout this text.
5. Compile the plug-in. Be sure that **`Release`** is selected in Visual Studio and select **`Build->Rebuild Solution`**. Any errors and warnings will be reported. A Warning from the linker that the filename ends in .8BF is expected and can be ignored. Remember to select **`File->Save All`** and **`File->Close Solution`** after successful compilation.
6. Copy the compiled plug-in (**`.8BF`** file) to the folder in which plug-ins are stored for access by the host program, and rename the plug-in from **`UserCode.8BF`** (for example, **`PlugInName.8BF`**). Your host program must be told where this folder is located (in Photoshop, select **`Edit->Preferences->Plug-Ins & Scratch Disks`**). Photoshop scans this folder when it is launched, and recognizes the **`.8BF`** files there to build its menu. You can replace an existing plug-in file with another one having the same name while Photoshop is running, but to add a new one or to change the menu entry, you must quit and relaunch the host program.

Problems

In each chapter, there are several sets of problems that relate to the procedures and examples shown. Within each set, there are some that can be implemented simply by putting together the example code fragments shown, and others that require a greater degree of effort on the part of the student. There is no single "correct" answer in terms of the exact code written, but the correctness of the result can be taken in many cases as evidence that the procedure has been properly implemented.

The text also describes and illustrates additional procedures that may be assigned as problems, depending on the skill level of the students, the time available and, of course, the whims and interests of the instructor.

Solutions to the problems marked with a (#) are provided both as source code and compiled plug-ins on the companion CD that is included with the book.

The images shown in the text are also provided on the CD. Each image is saved as a **.TIF** file, readable by all of the host programs listed above.

0.5.1#. Implement a program that reads the image from the host, complements the values, and writes it back.

0.5.2#. Implement a program that replaces the contents of an image with a horizontal (or vertical) linear ramp of grayscale values (red = green = blue). Optionally, modify this to generate a horizontal ramp in the red channel, a vertical ramp in the green channel, and a reversed vertical ramp in the blue channel, or other color combinations.

The Authors

John C. Russ received his B.S. and M.S. degrees in engineering and solid-state physics from California Institute of Technology and his Ph.D. in engineering from California Coast University. At the Homer Research Labs of Bethlehem Steel Corp., in the 1960s, development of new steel alloys, such as those used in the Trans-Alaska Pipeline, was strongly linked to the microstructure as revealed by light and electron microscopy, microprobe, and x-ray analysis. In 1968, Dr. Russ became director of the Applications Laboratories at Japan Electron Optics Laboratories (JEOL), introducing the scanning electron microscope. From there, it was a natural step to join in the formation of EDAX, which became the leading supplier of microanalysis instrumentation for use on SEMs and TEMs. As senior vice president, he was deeply involved in the development of these devices, the creation of software for qualitative and quantitative interpretation of the spectra, and the imaging of elemental distributions. After the sale of EDAX to Philips, Dr. Russ joined the faculty of North Carolina State University in 1978. He also participated in research at the Danish Technological Institute. After retirement from formal teaching duties at NCSU in 1996, he accepted a position as research director of Rank Taylor Hobson, a British manufacturer of precision instrumentation. He continues to be active as an adjunct professor at NCSU, as well as a consultant and author.

As a professor in the Materials Science and Engineering Department, Dr. Russ and his students have used a broad array of microscope technologies to study materials microstructures and surfaces. These have included conventional and confocal light microscopes, electron and ion microprobes, scanning and transmission electron microscopes, x-ray and neutron tomography, and a variety of scanned probe microscopes. The need to process these images to obtain quantitative structural information led to the development of computer control for instruments and computer processing for the data. Dr. Russ has become widely known as a leader in the development and use of these tools for image analysis. At NCSU his collaborations have extended far beyond the materials science field, including food science, archaeology, biology, veterinary medicine, textiles, and others. Beyond the campus, he has worked with a worldwide range of companies in fields such as pharmaceutical and energy applications, and has been retained as an expert witness in forensic cases, both civil and criminal.

Through academic courses and workshops, Dr. Russ has presented image analysis methods to more than 4,000 students. He has taught acclaimed hands-on workshops worldwide, from Australia to Slovenia, Japan to South Africa. His more than 300 publications, including more than a dozen books, have reached thousands more. These books include *Computer Assisted Microscopy, Practical Stereology* (with Robert Dehoff), *Fractal Surfaces, The Image Processing Handbook* (now in its fifth edition), *Forensic Uses of Digital Imaging,* and *Image Analysis of Food Microstructure.* On November 16, 2006, Dr. Russ received the 2006 Ernst Abbe Memorial Award of the New York Microscopical Society for achievements made in the field of microscopy.

J. Christian (Chris) Russ has been writing image processing and image analysis software since 1979. His undergraduate degree is in computer science from the University of

Michigan, and he subsequently attended graduate school in biomedical engineering at the University of Texas at Austin. Presently, he owns Reindeer Graphics, Inc., a supplier of image processing and related software. He also works as a senior scientist on forensic imaging for Ocean Systems, Inc.

Contents

试读结束，需要全本PDF请购买 www.ertongbook.com