

UNDERGRADUATE TOPICS
IN COMPUTER SCIENCE

Wilhelm Burger
Mark J. Burge

Principles of Digital Image Processing

Advanced Methods

Undergraduate Topics in Computer Science

For further volumes:
<http://www.springer.com/series/7592>

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

Wilhelm Burger • Mark J. Burge

Principles of Digital Image Processing

Advanced Methods

With 129 figures, 6 tables and 46 algorithms



Wilhelm Burger
School of Informatics/Communications/Media
Upper Austria University of Applied Sciences
Hagenberg, Austria

Mark J. Burge
MITRE
Washington, D.C., USA

Series editor

Ian Mackie, École Polytechnique, France and University of Sussex, UK

Advisory board

Samson Abramsky, University of Oxford, UK
Chris Hankin, Imperial College London, UK
Dexter Kozen, Cornell University, USA
Andrew Pitts, University of Cambridge, UK
Hanne Riis Nielson, Technical University of Denmark, Denmark
Steven Skiena, Stony Brook University, USA
Iain Stewart, University of Durham, UK
Karin Breitman, Catholic University of Rio de Janeiro, Brazil

Undergraduate Topics in Computer Science ISSN 1863-7310
ISBN 978-1-84882-918-3 ISBN 978-1-84882-919-0 (eBook)
DOI 10.1007/978-1-84882-919-0
Springer London Heidelberg New York Dordrecht

Library of Congress Control Number: 2013938415

© Springer-Verlag London 2013

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. Exempted from this legal reservation are brief excerpts in connection with reviews or scholarly analysis or material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work. Duplication of this publication or parts thereof is permitted only under the provisions of the Copyright Law of the Publisher's location, in its current version, and permission for use must always be obtained from Springer. Permissions for use may be obtained through RightsLink at the Copyright Clearance Center. Violations are liable to prosecution under the respective Copyright Law.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This is the 3rd volume of the authors' textbook series on ***Principles of Digital Image Processing*** that is predominantly aimed at undergraduate study and teaching:

- Volume 1: ***Fundamental Techniques***,
- Volume 2: ***Core Algorithms***,
- Volume 3: ***Advanced Methods*** (this volume).

While it builds on the previous two volumes and relies on their proven format, it contains all new material published by the authors for the first time. The topics covered in this volume are slightly more advanced and should thus be well suited for a follow-up undergraduate or Master-level course and as a solid reference for experienced practitioners in the field.

The topics of this volume range over a variety of image processing applications, with a general focus on “classic” techniques that are in wide use but are at the same time challenging to explore with the existing scientific literature. In choosing these topics, we have also considered input received from students, lecturers and practitioners over several years, for which we are very grateful. While it is almost unfeasible to cover all recent developments in the field, we focused on popular “workhorse” techniques that are available in many image processing systems but are often used without a thorough understanding of their inner workings. This particularly applies to the contents of the first five chapters on automatic thresholding, filters and edge detectors for color images, and edge-preserving smoothing. Also, an extensive part of the book is devoted to David Lowe’s popular *SIFT* method for invariant local feature detection, which has found its way into so many applications and has become a standard tool in the industry, despite (as the text probably shows) its inherent sophistication and complexity. An additional “bonus chapter” on Synthetic

Gradient Noise, which could not be included in the print version, is available for download from the book’s website.

As in the previous volumes, our main goal has been to provide *accurate*, *understandable*, and *complete* algorithmic descriptions that take the reader all the way from the initial idea through the formal description to a working implementation. This may make the text appear bloated or too mathematical in some places, but we expect that interested readers will appreciate the high level of detail and the decision not to omit the (sometimes essential) intermediate steps. Wherever reasonable, general prerequisites and more specific details are summarized in the Appendix, which should also serve as a quick reference that is supported by a carefully compiled Index. While space constraints did not permit the full source code to be included in print, complete (Java) implementations for each chapter are freely available on the book’s website (see below). Again we have tried to make this code maximally congruent with the notation used in the text, such that readers should be able to easily follow, execute, and extend the described steps.

Software

The implementations in this book series are all based on Java and ImageJ, a widely used programmer-extensible imaging system developed, maintained, and distributed by Wayne Rasband of the National Institutes of Health (NIH).¹ ImageJ is implemented completely in Java and therefore runs on all major platforms. It is widely used because its “plugin”-based architecture enables it to be easily extended. Although all examples run in ImageJ, they have been specifically designed to be easily ported to other environments and programming languages. We chose Java as an implementation language because it is elegant, portable, familiar to many computing students, and more efficient than commonly thought. Note, however, that we incorporate Java purely as an instructional vehicle because precise language semantics are needed eventually to achieve ultimate clarity. Since we stress the simplicity and readability of our programs, this should not be considered production-level but “instructional” software that naturally leaves vast room for improvement and performance optimization. Consequently, this book is not primarily on Java programming nor is it intended to serve as a reference manual for ImageJ.

Online resources

In support of this book series, the authors maintain a dedicated website that provides supplementary materials, including the complete Java source code,

¹ <http://rsb.info.nih.gov/ij/>

the test images used in the examples, and corrections. Readers are invited to visit this site at

www.imagingbook.com

It also makes available additional materials for educators, including a complete set of figures, tables, and mathematical elements shown in the text, in a format suitable for easy inclusion in presentations and course notes. Also, as a free add-on to this volume, readers may download a supplementary “bonus chapter” on synthetic noise generation. Any comments, questions, and corrections are welcome and should be addressed to

imagingbook@gmail.com

Acknowledgments

As with its predecessors, this volume would not have been possible without the understanding and steady support of our families. Thanks go to Wayne Rasband (NIH) for continuously improving ImageJ and for his outstanding service to the imaging community. We appreciate the contributions from the many careful readers who have contacted us to suggest new topics, recommend alternative solutions, or to suggest corrections. A special debt of gratitude is owed to Stefan Stavrev for his detailed, technical editing of this volume. Finally, we are grateful to Wayne Wheeler for initiating this book series and Simon Rees and his colleagues at Springer’s UK and New York offices for their professional support, for the high quality (full-color) print production and the enduring patience with the authors.

Hagenberg, Austria / Washington DC, USA
January 2013

Contents

Preface	v
1. Introduction	1
2. Automatic Thresholding	5
2.1 Global histogram-based thresholding	6
2.1.1 Statistical information from the histogram	8
2.1.2 Simple threshold selection	10
2.1.3 Iterative threshold selection (ISODATA algorithm)	11
2.1.4 Otsu's method	14
2.1.5 Maximum entropy thresholding	18
2.1.6 Minimum error thresholding	22
2.2 Local adaptive thresholding	30
2.2.1 Bernsen's method	30
2.2.2 Niblack's method	34
2.3 Java implementation	46
2.4 Summary and further reading	49
2.5 Exercises	49
3. Filters for Color Images	51
3.1 Linear filters	51
3.1.1 Using monochromatic linear filters on color images	52
3.1.2 Color space considerations	55
3.2 Non-linear color filters	66
3.2.1 Scalar median filter	66
3.2.2 Vector median filter	67

3.2.3	Sharpening vector median filter	69
3.3	Java implementation	76
3.4	Further reading	80
3.5	Exercises	80
4.	Edge Detection in Color Images	83
4.1	Monochromatic techniques	84
4.2	Edges in vector-valued images	88
4.2.1	Multi-dimensional gradients	88
4.2.2	The Jacobian matrix	93
4.2.3	Squared local contrast	94
4.2.4	Color edge magnitude	95
4.2.5	Color edge orientation	97
4.2.6	Grayscale gradients revisited	99
4.3	Canny edge operator	103
4.3.1	Canny edge detector for grayscale images	103
4.3.2	Canny edge detector for color images	105
4.4	Implementation	115
4.5	Other color edge operators	116
5.	Edge-Preserving Smoothing Filters	119
5.1	Kuwahara-type filters	120
5.1.1	Application to color images	123
5.2	Bilateral filter	126
5.2.1	Domain vs. range filters	128
5.2.2	Bilateral filter with Gaussian kernels	131
5.2.3	Application to color images	132
5.2.4	Separable implementation	136
5.2.5	Other implementations and improvements	141
5.3	Anisotropic diffusion filters	143
5.3.1	Homogeneous diffusion and the heat equation	144
5.3.2	Perona-Malik filter	146
5.3.3	Perona-Malik filter for color images	149
5.3.4	Geometry-preserving anisotropic diffusion	156
5.3.5	Tschumperlé-Deriche algorithm	157
5.4	Measuring image quality	161
5.5	Implementation	164
5.6	Exercises	165

6. Fourier Shape Descriptors	169
6.1 2D boundaries in the complex plane	169
6.1.1 Parameterized boundary curves	169
6.1.2 Discrete 2D boundaries	170
6.2 Discrete Fourier transform	171
6.2.1 Forward transform	173
6.2.2 Inverse Fourier transform (reconstruction)	173
6.2.3 Periodicity of the DFT spectrum	177
6.2.4 Truncating the DFT spectrum	177
6.3 Geometric interpretation of Fourier coefficients	179
6.3.1 Coefficient G_0 corresponds to the contour's centroid	180
6.3.2 Coefficient G_1 corresponds to a circle	181
6.3.3 Coefficient G_m corresponds to a circle with frequency m	182
6.3.4 Negative frequencies	183
6.3.5 Fourier descriptor pairs correspond to ellipses	183
6.3.6 Shape reconstruction from truncated Fourier descriptors	187
6.3.7 Fourier descriptors from arbitrary polygons	193
6.4 Effects of geometric transformations	195
6.4.1 Translation	197
6.4.2 Scale change	199
6.4.3 Shape rotation	199
6.4.4 Shifting the contour start position	200
6.4.5 Effects of phase removal	201
6.4.6 Direction of contour traversal	203
6.4.7 Reflection (symmetry)	203
6.5 Making Fourier descriptors invariant	203
6.5.1 Scale invariance	204
6.5.2 Start point invariance	205
6.5.3 Rotation invariance	208
6.5.4 Other approaches	209
6.6 Shape matching with Fourier descriptors	214
6.6.1 Magnitude-only matching	214
6.6.2 Complex (phase-preserving) matching	218
6.7 Java implementation	219
6.8 Summary and further reading	225
6.9 Exercises	225
7. SIFT—Scale-Invariant Local Features	229
7.1 Interest points at multiple scales	230
7.1.1 The Laplacian-of-Gaussian (LoG) filter	231
7.1.2 Gaussian scale space	237

7.1.3	LoG/DoG scale space	240
7.1.4	Hierarchical scale space	242
7.1.5	Scale space implementation in SIFT	248
7.2	Key point selection and refinement	252
7.2.1	Local extrema detection	255
7.2.2	Position refinement	257
7.2.3	Suppressing responses to edge-like structures	260
7.3	Creating Local Descriptors	263
7.3.1	Finding dominant orientations	263
7.3.2	Descriptor formation	267
7.4	SIFT algorithm summary	276
7.5	Matching SIFT Features	276
7.5.1	Feature distance and match quality	285
7.5.2	Examples	287
7.6	Efficient feature matching	289
7.7	SIFT implementation in Java	294
7.7.1	SIFT feature extraction	294
7.7.2	SIFT feature matching	295
7.8	Exercises	296

Appendix

A.	Mathematical Symbols and Notation	299
B.	Vector Algebra and Calculus	305
B.1	Vectors	305
B.1.1	Column and row vectors	306
B.1.2	Vector length	306
B.2	Matrix multiplication	306
B.2.1	Scalar multiplication	306
B.2.2	Product of two matrices	307
B.2.3	Matrix-vector products	307
B.3	Vector products	308
B.3.1	Dot product	308
B.3.2	Outer product	309
B.4	Eigenvectors and eigenvalues	309
B.4.1	Eigenvectors of a 2×2 matrix	310
B.5	Parabolic fitting	311
B.5.1	Fitting a parabolic function to three sample points	311
B.5.2	Parabolic interpolation	313
B.6	Vector fields	315

B.6.1	Jacobian matrix	315
B.6.2	Gradient	315
B.6.3	Maximum gradient direction	316
B.6.4	Divergence	317
B.6.5	Laplacian	317
B.6.6	The Hessian matrix	318
B.7	Operations on multi-variable, scalar functions (scalar fields)	319
B.7.1	Estimating the derivatives of a discrete function	319
B.7.2	Taylor series expansion of functions	319
B.7.3	Finding the continuous extremum of a multi-variable discrete function	323
C.	Statistical Prerequisites	329
C.1	Mean, variance and covariance	329
C.2	Covariance matrices	330
C.2.1	Example	331
C.3	The Gaussian distribution	332
C.3.1	Maximum likelihood	333
C.3.2	Gaussian mixtures	334
C.3.3	Creating Gaussian noise	335
C.4	Image quality measures	335
D.	Gaussian Filters	337
D.1	Cascading Gaussian filters	337
D.2	Effects of Gaussian filtering in the frequency domain	338
D.3	LoG-approximation by the difference of two Gaussians (DoG)	339
E.	Color Space Transformations	341
E.1	RGB/sRGB transformations	341
E.2	CIELAB/CIELUV transformations	342
E.2.1	CIELAB	343
E.2.2	CIELUV	344
Bibliography		347
Index		361

1

Introduction

This third volume in the authors' *Principles of Digital Image Processing* series presents a thoughtful selection of advanced topics. Unlike our first two volumes, this one delves deeply into a select set of advanced and largely independent topics. Each of these topics is presented as a separate module which can be understood independently of the other topics, making this volume ideal for readers who expect to work independently and are ready to be exposed to the full complexity (and corresponding level of detail) of advanced, real-world topics.

This volume seeks to bridge the gap often encountered by imaging engineers who seek to implement these advanced topics—inside you will find detailed, formally presented derivations supported by complete Java implementations. For the required foundations, readers are referred to the first two volumes of this book series [20, 21] or the “professional edition” by the same authors [19].

Point operations, automatic thresholding

Chapter 2 addresses *automatic thresholding*, that is, the problem of creating a faithful black-and-white (i.e., binary) representation of an image acquired under a broad range of illumination conditions. This is closely related to *histograms* and *point operations*, as covered in Chapters 3–4 of Volume 1 [20], and is also an important prerequisite for working with region-segmented binary images, as discussed in Chapter 2 of Volume 2 [21].

The first part of this chapter is devoted to global thresholding techniques that rely on the statistical information contained in the grayscale histogram to

calculate a single threshold value to be applied uniformly to all image pixels. The second part presents techniques that adapt the threshold to the local image data by adjusting to varying brightness and contrast caused by non-uniform lighting and/or material properties.

Filters and edge operators for color images

Chapters 3–4 address the issues related to building filters and edge detectors specifically for color images. Filters for color images are often implemented by simply applying monochromatic techniques, i. e., filters designed for scalar-valued images (see Ch. 5 of Vol. 1 [20]), separately to each of the color channels, not explicitly considering the vector-valued nature of color pixels. This is common practice with both linear and non-linear filters and although the results of these monochromatic techniques are often visually quite acceptable, a closer look reveals that the errors can be substantial. In particular, it is demonstrated in Chapter 3 that the colorimetric results of linear filtering depend crucially upon the choice of the working color space, a fact that is largely ignored in practice. The situation is similar for non-linear filters, such as the classic median filter, for which color versions are presented that make explicit use of the vector-valued data.

Similar to image filters, edge operators for color images are also often implemented from monochromatic components despite the fact that specific color edge detection methods have been around for a long time. Some of these classic techniques, typically rooted in the theory of discrete vector fields, are presented in Chapter 4, including Di Zenzo’s method and color versions of the popular Canny edge detector, which was not covered in the previous volumes.

Filters that eliminate noise by image smoothing while simultaneously preserving edge structures are the focus of Chapter 5. We start this chapter with a discussion of the classic techniques, in particular what we call *Kuwahara-type* filters and the *Bilateral* filter for both grayscale and color images. The second part of this chapter is dedicated to the powerful class of anisotropic diffusion filters, with special attention given to the techniques by *Perona/Malik* and *Tschumperlé/Deriche*, again considering both grayscale and color images.

Descriptors: contours and local keypoints

The final Chapters 6–7 deal with deriving invariant descriptions of image structures. Both chapters present classic techniques that are widely used and have been extensively covered in the literature, though not in this algorithmic form or at this level of detail.

Elegant contour-based shape descriptors based on *Fourier* transforms are the topic of Chapter 6. These *Fourier descriptors* are based on an intuitive

mathematical theory and are widely used for 2D shape representation and matching—mostly because of their (alleged) inherent invariance properties. In particular, this means that shapes can be represented and compared in the presence of translation, rotation, and scale changes. However, what looks promising in theory turns out to be a non-trivial task in practice, mainly because of noise and discretization effects. The key lesson of this chapter is that, in contrast to popular opinion, it takes quite some effort to build Fourier transform-based solutions that indeed afford invariant and unambiguous shape matching.

Chapter 7 gives an in-depth presentation of David Lowe’s *Scale-Invariant Local Feature Transform* (SIFT), used to localize and identify unique key points in sets of images in a scale and rotation-invariant fashion. It has become an almost universal tool in the image processing community and is the original source of many derivative techniques. Its common use tends to hide the fact that SIFT is an intricate, highly-tuned technique whose implementation is more complex than any of the algorithms presented in this book series so far. Consequently, this is an extensive chapter supplemented by a complete Java implementation that has been completely written from the ground up to be in sync with the mathematical/algorithmic description of the text. Besides a careful description of SIFT and an introduction to the crucial concept of scale space, this chapter also reveals a rich variety of smaller techniques that are interesting by themselves and useful in many other applications.

Bonus chapter: synthetic noise images

In addition to the topics described above, one additional chapter on the *synthesis of gradient noise images* was intended for this volume but could not be included in the print version because of space limitations. However, this “bonus chapter” is available in electronic form on the book’s website (see page vii). The topic of this chapter may appear a bit “exotic” in the sense that it does not deal with processing images or extracting useful information from images, but with *generating* new image content. Since the techniques described here were originally developed for texture synthesis in computer graphics (often referred to as *Perlin noise* [99,100]), they are typically not taught in image processing courses, although they fit well into this context. This is one of several interesting topics, where the computer graphics and image processing communities share similar interests and methods.

2

Automatic Thresholding

Although techniques based on binary image regions have been used for a very long time, they still play a major role in many practical image processing applications today because of their simplicity and efficiency. To obtain a binary image, the first and perhaps most critical step is to convert the initial grayscale (or color) image to a binary image, in most cases by performing some form of thresholding operation, as described in Volume 1, Section 4.1.4 [20].

Anyone who has ever tried to convert a scanned document image to a readable binary image has experienced how sensitively the result depends on the proper choice of the threshold value. This chapter deals with finding the best threshold automatically only from the information contained in the image, i. e., in an “unsupervised” fashion. This may be a single, “global” threshold that is applied to the whole image or different thresholds for different parts of the image. In the latter case we talk about “adaptive” thresholding, which is particularly useful when the image exhibits a varying background due to uneven lighting, exposure or viewing conditions.

Automatic thresholding is a traditional and still very active area of research that had its peak in the 1980s and 1990s. Numerous techniques have been developed for this task, ranging from simple ad-hoc solutions to complex algorithms with firm theoretical foundations, as documented in several reviews and evaluation studies [46, 96, 113, 118, 128]. Binarization of images is also considered a “segmentation” technique and thus often categorized under this term. In the following, we describe some representative and popular techniques in greater detail, starting in Section 2.1 with global thresholding methods and continuing with adaptive methods in Section 2.2.

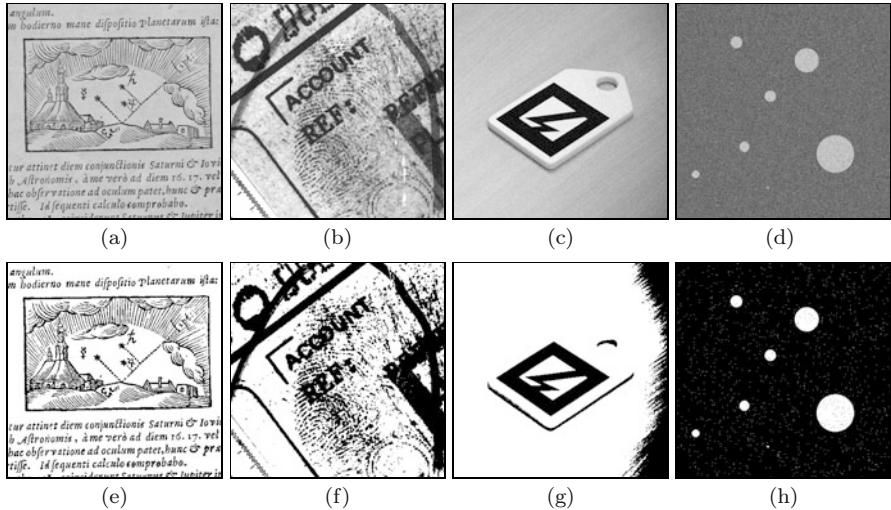


Figure 2.1 Test images used for subsequent thresholding experiments. Detail from a manuscript by Johannes Kepler (a), document with fingerprint (b), ARTToolkit marker (c), synthetic two-level Gaussian mixture image (d). Results of thresholding with the fixed threshold value $q = 128$ (e-h).

2.1 Global histogram-based thresholding

Given a grayscale image I , the task is to find a single “optimal” threshold value for binarizing this image. Applying a particular threshold q is equivalent to classifying each pixel as being either part of the *background* or the *foreground*. Thus the set of all image pixels is partitioned into two disjoint sets \mathcal{C}_0 and \mathcal{C}_1 , where \mathcal{C}_0 contains all elements with values in $[0, 1, \dots, q]$ and \mathcal{C}_1 collects the remaining elements with values in $[q+1, \dots, K-1]$, that is,

$$(u, v) \in \begin{cases} \mathcal{C}_0 & \text{if } I(u, v) \leq q \text{ (background),} \\ \mathcal{C}_1 & \text{if } I(u, v) > q \text{ (foreground).} \end{cases} \quad (2.1)$$

Note that the meaning of *background* and *foreground* may differ from one application to another. For example, the above scheme is quite natural for astronomical or thermal images, where the relevant “foreground” pixels are bright and the background is dark. Conversely, in document analysis, for example, the objects of interest are usually the *dark* letters or artwork printed on a bright background. This should not be confusing and of course one can always *invert* the image to adapt to the above scheme, so there is no loss of generality here. [Figure 2.1](#) shows several test images used in this chapter and the result of thresholding with a fixed threshold value. The synthetic image in [Fig. 2.1 \(d\)](#) is the mixture of two Gaussian random distributions $\mathcal{N}_0, \mathcal{N}_1$ for the background

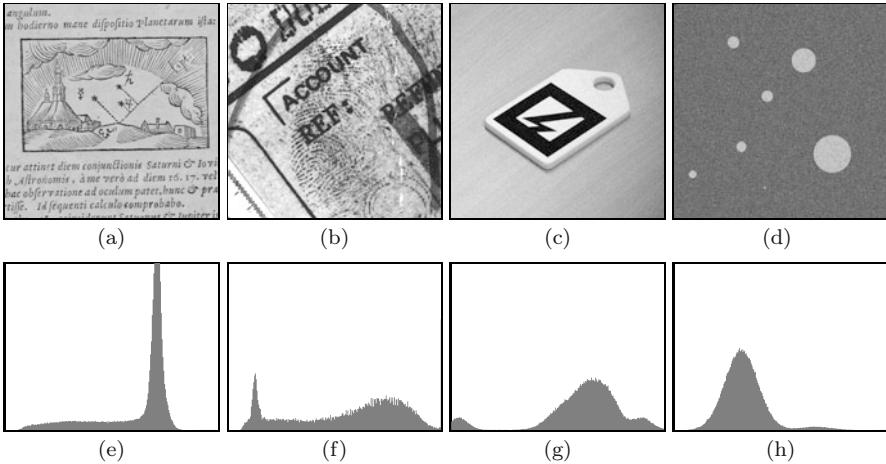


Figure 2.2 Test images (a–d) and their histograms (e–h). All histograms are normalized to constant area (not to maximum values, as usual), with intensity values ranging from 0 (left) to 255 (right). The synthetic image in (d) is the mixture of two Gaussian random distributions $\mathcal{N}_0, \mathcal{N}_1$ for the background and foreground, respectively, with $\mu_0 = 80$, $\mu_1 = 170$, $\sigma_0 = \sigma_1 = 20$. The two Gaussian distributions are clearly visible in the corresponding histogram (h).

and foreground, respectively, with $\mu_0 = 80$, $\mu_1 = 170$, $\sigma_0 = \sigma_1 = 20$. The corresponding histograms of the test images are shown in Fig. 2.2.

The key question is how to find a suitable (or even “optimal”) threshold value for binarizing the image. As the name implies, histogram-based methods calculate the threshold primarily from the information contained in the image’s histogram, without inspecting the actual image pixels. Other methods process individual pixels for finding the threshold and there are also hybrid methods that rely both on the histogram and the local image content. Histogram-based techniques are usually simple and efficient, because they operate on a small set of data (256 values in case of an 8-bit histogram); they can be grouped into two main categories: *shape-based* and *statistical* methods.

Shape-based methods analyze the structure of the histogram’s distribution, for example, by trying to locate peaks, valleys, and other “shape” features. Usually the histogram is first smoothed to eliminate narrow peaks and gaps. While shape-based methods were quite popular early on, they are usually not as robust as their statistical counterparts or at least do not seem to offer any distinct advantages. A classic representative of this category is the “triangle” (or “chord”) algorithm described in [150]. References to numerous other shape-based methods can be found in [118].

Statistical methods, as their name suggests, rely on statistical information derived from the image’s histogram (which of course is a statistic itself), such

as the mean, variance, or entropy. In Section 2.1.1, we discuss a few elementary parameters that can be obtained from the histogram, followed by a description of concrete algorithms that use this information. Again there is a vast number of similar methods and we have selected four representative algorithms to be described in more detail: iterative threshold selection by Ridler and Calvard [112], Otsu’s clustering method [95], the minimum error method by Kittler and Illingworth [57], and the maximum entropy thresholding method by Kapur, Sahoo, and Wong [64]. Before attending to these algorithms, let us review some elementary facts about the information that can be derived from an image’s histogram.

2.1.1 Statistical information from the histogram

Let $\mathbf{h}(g)$ denote the histogram of the grayscale image I with a total of N pixels and K possible intensity values $0 \leq g < K$ (for a basic introduction to histograms see Chapter 3 of Volume 1 [20]). The *mean* of all pixel values in I is defined as

$$\mu_I = \frac{1}{N} \cdot \sum_{u,v} I(u,v) = \frac{1}{N} \cdot \sum_{g=0}^{K-1} g \cdot \mathbf{h}(g), \quad (2.2)$$

and the overall *variance* of the image is

$$\sigma_I^2 = \frac{1}{N} \cdot \sum_{u,v} [I(u,v) - \mu_I]^2 = \frac{1}{N} \cdot \sum_{g=0}^{K-1} (g - \mu_I)^2 \cdot \mathbf{h}(g). \quad (2.3)$$

As we see, both the mean and the variance of the image can be computed conveniently from the histogram, without referring to the actual image pixels. Moreover, the mean and the variance can be computed simultaneously in a single iteration by making use of the fact that

$$\mu_I = \frac{1}{N} \cdot A \quad \text{and} \quad \sigma_I^2 = \frac{1}{N} \cdot \left(B - \frac{1}{N} \cdot A^2 \right), \quad (2.4)$$

$$\text{with} \quad A = \sum_{g=0}^{K-1} g \cdot \mathbf{h}(g), \quad B = \sum_{g=0}^{K-1} g^2 \cdot \mathbf{h}(g). \quad (2.5)$$

If we *threshold* the image at level q ($0 \leq q < K$), the set of pixels is partitioned into the disjoint subsets \mathcal{C}_0 , \mathcal{C}_1 , corresponding to the background and the foreground. The number of pixels assigned to each subset is

$$n_0(q) = |\mathcal{C}_0| = \sum_{g=0}^q \mathbf{h}(g) \quad \text{and} \quad n_1(q) = |\mathcal{C}_1| = \sum_{g=q+1}^{K-1} \mathbf{h}(g), \quad (2.6)$$

respectively. Also, since all pixels are assigned to either the background \mathcal{C}_0 or the foreground set \mathcal{C}_1 ,

$$n_0(q) + n_1(q) = |\mathcal{C}_0 \cup \mathcal{C}_1| = N. \quad (2.7)$$

For any threshold q , the *mean* values of the pixels in the corresponding partitions $\mathcal{C}_0, \mathcal{C}_1$ can be calculated from the histogram as

$$\mu_0(q) = \frac{1}{n_0(q)} \cdot \sum_{g=0}^q g \cdot h(g), \quad (2.8)$$

$$\mu_1(q) = \frac{1}{n_1(q)} \cdot \sum_{g=q+1}^{K-1} g \cdot h(g), \quad (2.9)$$

and they relate to the image's overall mean μ_I (Eqn. (2.2)) by¹

$$\mu_I = \frac{1}{N} [n_0(q) \cdot \mu_0(q) + n_1(q) \cdot \mu_1(q)] = \mu_0(K-1). \quad (2.10)$$

Similarly, the *variances* of the background and foreground partitions can be extracted from the histogram as

$$\sigma_0^2(q) = \frac{1}{n_0(q)} \cdot \sum_{g=0}^q (g - \mu_0(q))^2 \cdot h(g), \quad (2.11)$$

$$\sigma_1^2(q) = \frac{1}{n_1(q)} \cdot \sum_{g=q+1}^{K-1} (g - \mu_1(q))^2 \cdot h(g). \quad (2.12)$$

The overall variance σ_I^2 for the entire image is identical to the variance of the background for $q = K-1$,

$$\sigma_I^2 = \frac{1}{N} \cdot \sum_{g=0}^{K-1} (g - \mu_I)^2 \cdot h(g) = \sigma_0^2(K-1), \quad (2.13)$$

i. e., for all pixels being assigned to the background partition. Note that, unlike the simple relation of the means given in Eqn. (2.10),

$$\sigma_I^2 \neq \frac{1}{N} [n_0(q) \cdot \sigma_0^2(q) + n_1(q) \cdot \sigma_1^2(q)] \quad (2.14)$$

in general (see also Eqn. (2.24)).

We will use these basic relations in the discussion of histogram-based threshold selection algorithms in the following and add more specific ones as we go along.

¹ Note that $\mu_0(q), \mu_1(q)$ are functions and thus $\mu_0(K-1)$ in Eqn. (2.10) denotes the mean of partition \mathcal{C}_0 for the threshold $K-1$.

2.1.2 Simple threshold selection

Clearly, the choice of the threshold value should not be fixed but somehow based on the content of the image. In the simplest case, we could use the *mean* of all image pixels,

$$q = \text{mean}(I) = \mu_I, \quad (2.15)$$

as the threshold value q , or the *median*,

$$q = \text{median}(I), \quad (2.16)$$

or, alternatively, the average of the *minimum* and the *maximum* (mid-range value), i. e.,

$$q = \text{round}\left(\frac{\max(I) + \min(I)}{2}\right). \quad (2.17)$$

Like the image mean μ_I (see Eqn. (2.2)), all these quantities can be obtained directly from the histogram \mathbf{h} .

Thresholding at the median segments the image into approximately equal-sized background and foreground sets, i. e., $|\mathcal{C}_0| \approx |\mathcal{C}_1|$, which assumes that the “interesting” (foreground) pixels cover about half of the image. This may be appropriate for certain images, but completely wrong for others. For example, a scanned text image will typically contain a lot more white than black pixels, so using the median threshold would probably be unsatisfactory in this case. If the approximate fraction b ($0 < b < 1$) of expected background pixels is known in advance, the threshold could be set to that *quantile* instead. In this case, q is simply chosen as

$$q = \underset{j}{\operatorname{argmin}} \sum_{i=0}^j \mathbf{h}(i) \geq N \cdot b, \quad (2.18)$$

where N is the total number of pixels. This simple thresholding method is summarized in [Alg. 2.1](#).

For the *mid-range* technique (Eqn. (2.17)), the limiting intensity values $\min(I)$ and $\max(I)$ can be found by searching for the smallest and largest non-zero entries, respectively, in the histogram \mathbf{h} , that is,

$$\begin{aligned} \min(I) &= \underset{j}{\operatorname{argmin}}, \mathbf{h}(j) > 0, \\ \max(I) &= \underset{j}{\operatorname{argmax}}, \mathbf{h}(j) > 0. \end{aligned} \quad (2.19)$$

Applying the mid-range threshold (Eqn. (2.17)) segments the image at 50 % (or any other percentile) of the contrast range. In this case, nothing can be

Algorithm 2.1 Quantile thresholding. The optimal threshold value $q \in [0, K-2]$ is returned, or -1 if no valid threshold was found. Note the test in line 9 to check if the foreground is empty or not (the background is always non-empty by definition).

```

1: QUANTILETHRESHOLD( $\mathbf{h}, b$ )
   Input:  $\mathbf{h} : [0, K-1] \mapsto \mathbb{N}$ , a grayscale histogram.
           $b$ , the proportion of expected background pixels ( $0 < b < 1$ ).
          Returns the optimal threshold value or  $-1$  if no threshold is found.
2:  $K \leftarrow \text{Size}(\mathbf{h})$                                  $\triangleright$  number of intensity levels
3:  $N \leftarrow \sum_{i=0}^{K-1} \mathbf{h}(i)$                    $\triangleright$  number of image pixels
4:  $j \leftarrow 0$ 
5:  $c \leftarrow \mathbf{h}(0)$ 
6: while  $(j < K) \wedge (c < N \cdot b)$  do            $\triangleright$  quantile calc. (Eqn. (2.18))
7:    $j \leftarrow j + 1$ 
8:    $c \leftarrow c + \mathbf{h}(j)$ 
9: if  $c < N$  then                                $\triangleright$  foreground is non-empty
10:    $q \leftarrow j$ 
11: else                                      $\triangleright$  foreground is empty, all pixels are background
12:    $q \leftarrow -1$ 
13: return  $q$ .
```

said in general about the relative sizes of the resulting background and foreground partitions. Because a single extreme pixel value (outlier) may change the contrast range dramatically, this approach is not very robust.

In the pathological (but nevertheless possible) case that all pixels in the image have the same intensity g , all these methods will return the threshold $q = g$, which assigns all pixels to the background partition and leaves the foreground empty. Algorithms should try to detect this situation, because thresholding a uniform image obviously makes no sense.

Results obtained with these simple thresholding techniques are shown in Fig. 2.3. Despite the obvious limitations, even a simple automatic threshold selection (such as the quantile technique in Alg. 2.1) will typically yield more reliable results than the use of a fixed threshold.

2.1.3 Iterative threshold selection (ISODATA algorithm)

This classic iterative algorithm for finding an optimal threshold is attributed to Ridler and Calvard [112] and was related to ISODATA clustering by Velasco [137]. It is thus sometimes referred to as the “isodata” or “intermeans” algorithm. Like in many other global thresholding schemes it is assumed that

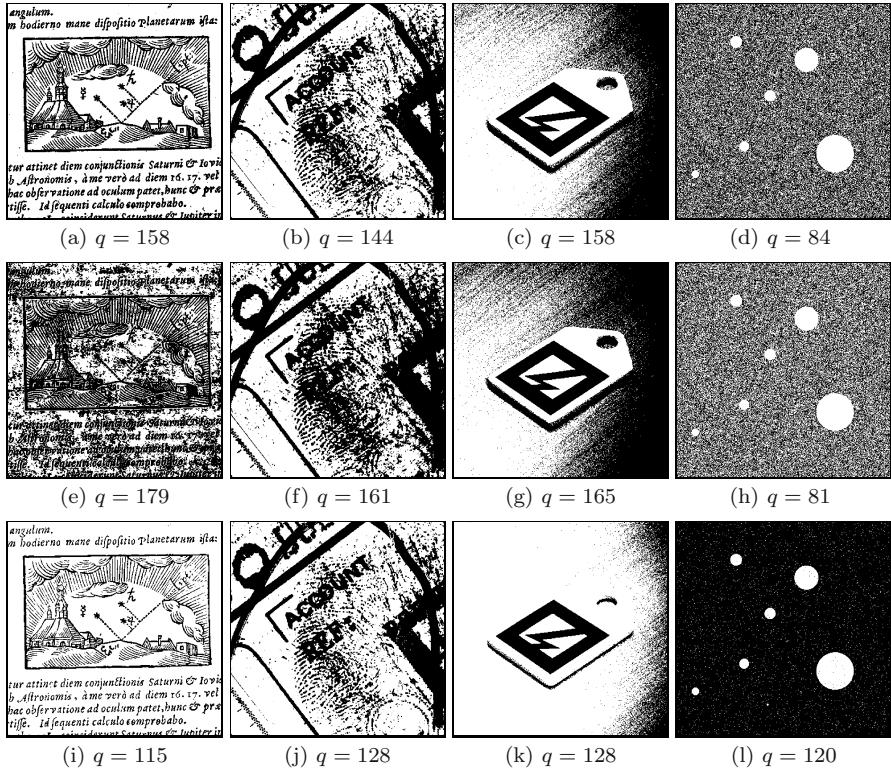


Figure 2.3 Results from various simple thresholding schemes. Mean (a–d), median (e–h), and mid-range (i–l) threshold, as specified in Eqns. (2.15–2.17).

the image’s histogram is a mixture of two separate distributions, one for the intensities of the background pixels and the other for the foreground pixels. In this case, the two distributions are assumed to be Gaussian with approximately identical spreads (variances).

The algorithm starts by making an initial guess for the threshold, for example, by taking the mean or the median of the whole image. This splits the set of pixels into a background and a foreground set, both of which should be non-empty. Next, the means of both sets are calculated and the threshold is repositioned to their average, i.e., centered between the two means. The means are then re-calculated for the resulting background and foreground sets, and so on, until the threshold does not change any longer. In practice, it takes only a few iterations for the threshold to converge.

This procedure is summarized in [Alg. 2.2](#). The initial threshold is set to the overall mean (line 3). For each threshold q , separate mean values μ_0, μ_1 are computed for the corresponding foreground and background partitions. The

Algorithm 2.2 “Isodata” threshold selection based on the iterative method by Ridler and Calvard [112].

```

1: ISODATATHRESHOLD( $\mathbf{h}$ )
   Input:  $\mathbf{h} : [0, K-1] \mapsto \mathbb{N}$ , a grayscale histogram.
          Returns the optimal threshold value or  $-1$  if no threshold is found.
2:    $K \leftarrow \text{Size}(\mathbf{h})$                                  $\triangleright$  number of intensity levels
3:    $q \leftarrow \text{Mean}(\mathbf{h}, 0, K-1)$                  $\triangleright$  set initial threshold to overall mean
4:   repeat
5:      $n_0 \leftarrow \text{Count}(\mathbf{h}, 0, q)$                    $\triangleright$  background population
6:      $n_1 \leftarrow \text{Count}(\mathbf{h}, q+1, K-1)$              $\triangleright$  foreground population
7:     if  $(n_0 = 0) \vee (n_1 = 0)$  then       $\triangleright$  backgrd. or foregrd. is empty
8:       return  $-1$ 
9:      $\mu_0 \leftarrow \text{Mean}(\mathbf{h}, 0, q)$                    $\triangleright$  background mean
10:     $\mu_1 \leftarrow \text{Mean}(\mathbf{h}, q+1, K-1)$              $\triangleright$  foreground mean
11:     $q' \leftarrow q$                                       $\triangleright$  keep previous threshold
12:     $q \leftarrow \left\lfloor \frac{\mu_0 + \mu_1}{2} \right\rfloor$   $\triangleright$  calculate the new threshold
13:   until  $q = q'$                                    $\triangleright$  terminate if no change
14:   return  $q$ 


---


15: Count( $\mathbf{h}, a, b$ ) :=  $\sum_{g=a}^b \mathbf{h}(g)$ 


---


16: Mean( $\mathbf{h}, a, b$ ) :=  $\left[ \sum_{g=a}^b g \cdot \mathbf{h}(g) \right] / \left[ \sum_{g=a}^b \mathbf{h}(g) \right]$ 

```

threshold is repeatedly set to the average of the two means until no more change occurs. The clause in line 7 tests if either the background or the foreground partition is empty, which will happen, for example, if the image contains only a single intensity value. In this case, no valid threshold exists and the procedure returns -1 .

The functions $\text{Count}(\mathbf{h}, a, b)$ and $\text{Mean}(\mathbf{h}, a, b)$ in lines 15–16 return the number of pixels and the mean, respectively, of the image pixels with intensity values in the range $[a, b]$. Both can be computed directly from the histogram \mathbf{h} without inspecting the image itself.

The performance of this algorithm can be easily improved by using tables $\mu_0(q), \mu_1(q)$ for the background and foreground means, respectively. The modified, table-based version of the iterative threshold selection procedure is shown in [Alg. 2.3](#). It requires two passes over the histogram to initialize the tables μ_0, μ_1 and only a small, constant number of computations for each iteration

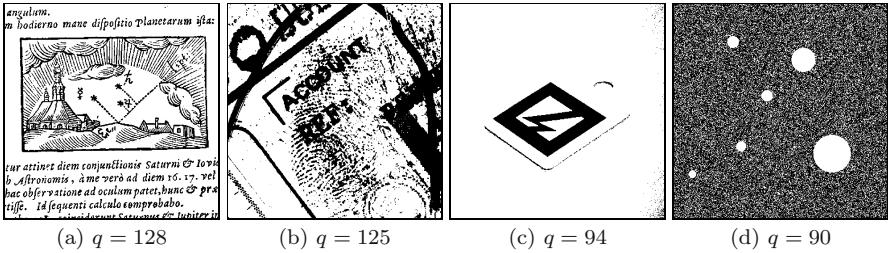


Figure 2.4 Thresholding with the isodata algorithm. Binarized images and the corresponding optimal threshold values (q).

in its main loop. Note that the image’s overall mean μ_I , used as the initial guess for the threshold q (Alg. 2.3, line 4), does not have to be calculated separately but can be obtained as $\mu_I = \mu_0(K-1)$ (threshold $q = K-1$ assigns all image pixels to the background). The time complexity of this algorithm is thus $\mathcal{O}(K)$, i.e., linear w.r.t. the size of the histogram. Figure 2.4 shows the results of thresholding with the isodata algorithm applied to the test images in Fig. 2.1.

2.1.4 Otsu’s method

The method by Otsu [74, 95] also assumes that the original image contains pixels from two classes, whose intensity distributions are unknown. The goal is to find a threshold q such that the resulting background and foreground distributions are maximally separated, which means that they are (a) each as narrow as possible (have minimal variances) and (b) their centers (means) are most distant from each other.

For a given threshold q , the variances of the corresponding background and foreground partitions can be calculated straight from the image’s histogram (see Eqn. (2.11)–(2.12)). The combined width of the two distributions is measured by the *within-class* variance

$$\begin{aligned}\sigma_w^2(q) &= P_0(q) \cdot \sigma_0^2(q) + P_1(q) \cdot \sigma_1^2(q) \\ &= \frac{1}{N} [n_0(q) \cdot \sigma_0^2(q) + n_1(q) \cdot \sigma_1^2(q)],\end{aligned}\tag{2.20}$$

where

$$\begin{aligned}P_0(q) &= \sum_{i=0}^q p(i) = \frac{1}{N} \cdot \sum_{i=0}^q h(i) = \frac{n_0(q)}{N}, \\ P_1(q) &= \sum_{i=q+1}^{K-1} p(i) = \frac{1}{N} \cdot \sum_{i=q+1}^{K-1} h(i) = \frac{n_1(q)}{N}\end{aligned}\tag{2.21}$$

Algorithm 2.3 Fast variant of “isodata” threshold selection using pre-calculated tables for the foreground and background means.

```

1:  FASTISODATATHRESHOLD( $\mathbf{h}$ )
   Input:  $\mathbf{h} : [0, K-1] \mapsto \mathbb{N}$ , a grayscale histogram.
   Returns the optimal threshold value or  $-1$  if no threshold is found.
2:   $K \leftarrow \text{Size}(\mathbf{h})$                                  $\triangleright$  number of intensity levels
3:   $\langle \mu_0, \mu_1, N \rangle \leftarrow \text{MAKEMEANTABLES}(\mathbf{h}, K)$ 
4:   $q \leftarrow \lfloor \mu_0(K-1) \rfloor$        $\triangleright$  take the overall mean  $\mu_I$  as initial threshold
5:  repeat
6:    if  $(\mu_0(q) < 0) \vee (\mu_1(q) < 0)$  then
7:      return  $-1$                                  $\triangleright$  background or foreground is empty
8:     $q' \leftarrow q$                                  $\triangleright$  keep previous threshold
9:     $q \leftarrow \left\lfloor \frac{\mu_0(q) + \mu_1(q)}{2} \right\rfloor$        $\triangleright$  calculate the new threshold
10:   until  $q = q'$                                  $\triangleright$  terminate if no change
11:   return  $q$ 

12: MAKEMEANTABLES( $\mathbf{h}, K$ )
13: Create maps  $\mu_0, \mu_1 : [0, K-1] \mapsto \mathbb{R}$ 
14:  $n_0 \leftarrow 0, s_0 \leftarrow 0$ 
15: for  $q \leftarrow 0, \dots, K-1$  do           $\triangleright$  tabulate background means  $\mu_0(q)$ 
16:    $n_0 \leftarrow n_0 + \mathbf{h}(q)$ 
17:    $s_0 \leftarrow s_0 + q \cdot \mathbf{h}(q)$ 
18:    $\mu_0(q) \leftarrow \begin{cases} s_0/n_0 & \text{if } n_0 > 0 \\ -1 & \text{otherwise} \end{cases}$ 
19:    $N \leftarrow n_0$ 
20:    $n_1 \leftarrow 0, s_1 \leftarrow 0$ 
21:    $\mu_1(K-1) \leftarrow 0$ 
22:   for  $q \leftarrow K-2, \dots, 0$  do           $\triangleright$  tabulate foreground means  $\mu_1(q)$ 
23:      $n_1 \leftarrow n_1 + \mathbf{h}(q+1)$ 
24:      $s_1 \leftarrow s_1 + (q+1) \cdot \mathbf{h}(q+1)$ 
25:      $\mu_1(q) \leftarrow \begin{cases} s_1/n_1 & \text{if } n_1 > 0 \\ -1 & \text{otherwise} \end{cases}$ 
26:   return  $\langle \mu_0, \mu_1, N \rangle$ 
```

are the class probabilities for $\mathcal{C}_0, \mathcal{C}_1$, respectively. Thus the within-class variance in Eqn. (2.20) is simply the sum of the individual variances weighted by the corresponding class probabilities or “populations”. Analogously, the *between-class* variance,

$$\sigma_b^2(q) = P_0(q) \cdot (\mu_0(q) - \mu_I)^2 + P_1(q) \cdot (\mu_1(q) - \mu_I)^2 \quad (2.22)$$

$$= \frac{1}{N} [n_0(q) \cdot (\mu_0(q) - \mu_I)^2 + n_1(q) \cdot (\mu_1(q) - \mu_I)^2], \quad (2.23)$$

measures the distances between the cluster means μ_0 , μ_1 and the overall mean μ_I . The total image variance σ_I^2 is the sum of the within-class variance and the between-class variance,

$$\sigma_I^2 = \sigma_w^2(q) + \sigma_b^2(q). \quad (2.24)$$

Since σ_I^2 is constant for a given image, the threshold q can be found by either *minimizing* the within-variance σ_w^2 or *maximizing* the between-variance σ_b^2 . The natural choice is to maximize σ_b^2 , because it only relies on first-order statistics (i.e., the within-class means μ_0, μ_1). Since the overall mean μ_I can be expressed as the weighted sum of the partition means μ_0 and μ_1 (Eqn. (2.10)), we can simplify Eqn. (2.23) to

$$\begin{aligned} \sigma_b^2(q) &= P_0(q) \cdot P_1(q) \cdot [\mu_0(q) - \mu_1(q)]^2 \\ &= \frac{1}{N^2} \cdot n_0(q) \cdot n_1(q) \cdot [\mu_0(q) - \mu_1(q)]^2. \end{aligned} \quad (2.25)$$

The optimal threshold is then found by *maximizing* the expression for the between-class variance in Eqn. (2.25) with respect to q , thereby *minimizing* the within-class variance in Eqn. (2.20).

Noting that $\sigma_b^2(q)$ only depends on the means (and *not* on the variances) of the two partitions for a given threshold q allows for a very efficient implementation, as outlined in [Alg. 2.4](#). The algorithm assumes a grayscale image with a total of N pixels and K intensity levels. As in [Alg. 2.3](#), precalculated tables $\mu_0(q), \mu_1(q)$ are used for the background and foreground means for all possible threshold values $q = 0, \dots, K - 1$. Initially (before entering the main **for**-loop in line 7) $q = -1$; at this point, the set of background pixels ($\leq q$) is empty and all pixels are classified as foreground ($n_0 = 0$ and $n_1 = N$). Each possible threshold value is examined inside the body of the **for**-loop.

As long as any one of the two classes is empty ($n_0(q) = 0$ or $n_1(q) = 0$),² the resulting between-class variance $\sigma_b^2(q)$ is zero. The threshold that yields the maximum between-class variance ($\sigma_{b\max}^2$) is returned, or -1 if no valid threshold could be found. This occurs when all image pixels have the same intensity, i.e., all pixels are in either the background or the foreground class.

Note that in line 11 of [Alg. 2.4](#), the factor $\frac{1}{N^2}$ is constant (independent of q) and can thus be ignored in the optimization. However, care must be taken at this point because the computation of σ_b^2 may produce intermediate values

² This is the case if the image contains no pixels with values $I(u, v) \leq q$ or $I(u, v) > q$, i.e., the histogram h is empty either below or above the index q .

Algorithm 2.4 Finding the optimal threshold using Otsu’s method [95]. Initially (outside the **for**-loop), the threshold q is assumed to be -1 , which corresponds to the background class being empty ($n_0 = 0$) and all pixels are assigned to the foreground class ($n_1 = N$). The **for**-loop (lines 7–14) examines each possible threshold $q = 0 \dots K-2$. The optimal threshold value is returned, or -1 if no valid threshold was found. The function `MAKEMEANTABLES()` is defined in [Alg. 2.3](#).

```

1: OTSUThreshold( $\mathbf{h}$ )
   Input:  $\mathbf{h} : [0, K-1] \mapsto \mathbb{N}$ , a grayscale histogram.
          Returns the optimal threshold value or  $-1$  if no threshold is found.
2:  $K \leftarrow \text{Size}(\mathbf{h})$                                  $\triangleright$  number of intensity levels
3:  $(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, N) \leftarrow \text{MAKEMEANTABLES}(\mathbf{h}, K)$        $\triangleright$  see Alg. 2.3
4:  $\sigma_{b\max}^2 \leftarrow 0$ 
5:  $q_{\max} \leftarrow -1$ 
6:  $n_0 \leftarrow 0$ 
7: for  $q \leftarrow 0, \dots, K-2$  do     $\triangleright$  examine all possible threshold values  $q$ 
8:    $n_0 \leftarrow n_0 + \mathbf{h}(q)$ 
9:    $n_1 \leftarrow N - n_0$ 
10:  if  $(n_0 > 0) \wedge (n_1 > 0)$  then
11:     $\sigma_b^2 \leftarrow \frac{1}{N^2} \cdot n_0 \cdot n_1 \cdot [\boldsymbol{\mu}_0(q) - \boldsymbol{\mu}_1(q)]^2$        $\triangleright$  see Eqn. (2.25)
12:    if  $\sigma_b^2 > \sigma_{b\max}^2$  then                                 $\triangleright$  maximize  $\sigma_b^2$ 
13:       $\sigma_{b\max}^2 \leftarrow \sigma_b^2$ 
14:       $q_{\max} \leftarrow q$ 
15: return  $q_{\max}$ .

```

that exceed the range of typical (32-bit) integer variables, even for medium-size images. Variables of type `long` should be used or the computation be performed with floating-point values.

The absolute “goodness” of the final thresholding by q_{\max} could be measured as the ratio

$$\eta = \frac{\sigma_b^2(q_{\max})}{\sigma_I^2} \in [0, 1], \quad (2.26)$$

which is invariant under linear changes in contrast and brightness [95]. Greater values of η indicate better thresholding.

Results of automatic threshold selection with Otsu’s method are shown in [Fig. 2.5](#), where q_{\max} denotes the optimal threshold and η is the corresponding “goodness” estimate, as defined in Eqn. (2.26). The graph underneath each image shows the original histogram (gray) overlaid with the variance within the background σ_0^2 (green), the variance within the foreground σ_1^2 (blue), and the between-class variance σ_b^2 (red) for varying threshold values q . The dashed

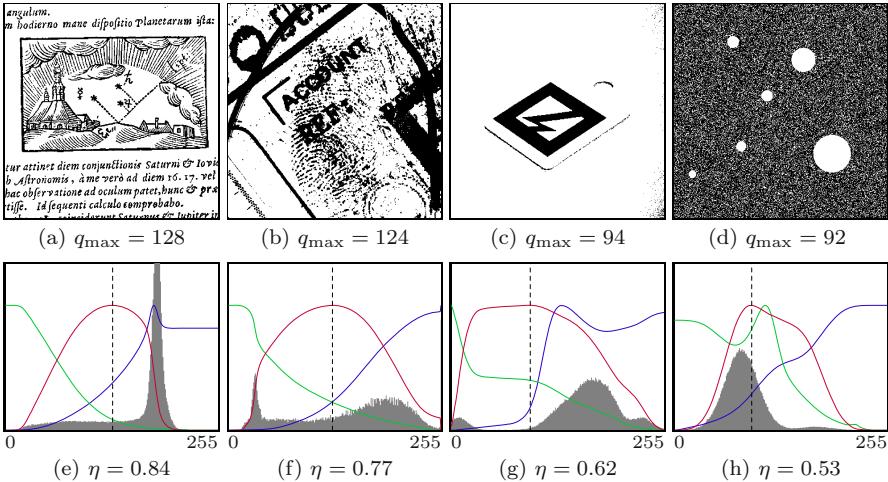


Figure 2.5 Results of thresholding with Otsu’s method. Calculated threshold values q and resulting binary images (a–d). Graphs in (e–h) show the corresponding within-background variance σ_0^2 (green), the within-foreground variance σ_1^2 (blue), and the between-class variance σ_b^2 (red), for varying threshold values $q = 0, \dots, 255$. The optimal threshold q_{\max} (dashed vertical line) is positioned at the maximum of σ_b^2 . The value η denotes the “goodness” estimate for the thresholding, as defined in Eqn. (2.26).

vertical line marks the position of the optimal threshold q_{\max} .

Due to the pre-calculation of the mean values, Otsu’s method requires only three passes over the histogram and is thus very fast ($\mathcal{O}(K)$), in contrast to opposite accounts in the literature. The method is frequently quoted and performs well in comparison to other approaches [118], despite its long history and its simplicity. In general, the results are very similar to the ones produced by the iterative threshold selection (“isodata”) algorithm described in Section 2.1.3.

2.1.5 Maximum entropy thresholding

Entropy is an important concept in information theory and particularly in data compression [51, 53]. It is a statistical measure that quantifies the average amount of information contained in the “messages” generated by a stochastic data source. For example, the N pixels in an image I can be interpreted as a message of N symbols, each taken independently from a finite alphabet of K (e.g., 256) different intensity values. Knowing the probability of each intensity value g to occur, entropy measures how likely it is to observe a particular image, or, in other words, how much we should be surprised to see such an image. Before going into further details, we briefly review the notion of probabilities in the context of images and histograms (see also Vol. 1, Sec. 4.6.1 [20]).

Modeling the image generation as a random process means to know the

probability of each intensity value g to occur, which we write as

$$p(g) = p(I(u, v) = g).$$

Since these probabilities are supposed to be known in advance, they are usually called *a priori* (or *prior*) probabilities. The vector of probabilities for the K different intensity values $g = 0, \dots, K-1$ (i.e., the alphabet of the data source) is

$$(p(0), p(1), \dots, p(K-1)),$$

which is called a *probability distribution* or *probability density function* (pdf). In practice, the *a priori* probabilities are usually *unknown*, but they can be estimated by observing how often the intensity values actually occur in one or more images, assuming that these are representative instances of the images. An estimate $\mathbf{p}(g)$ of the image's probability density function $p(g)$ is obtained by normalizing its histogram \mathbf{h} in the form

$$\mathbf{p}(g) = \frac{\mathbf{h}(g)}{N}, \quad (2.27)$$

for $0 \leq g < K$, such that $0 \leq \mathbf{p}(g) \leq 1$ and $\sum_{g=0}^{K-1} \mathbf{p}(g) = 1$. The corresponding *cumulative distribution function* (cdf) is

$$\mathbf{P}(g) = \sum_{i=0}^g \frac{\mathbf{h}(i)}{N} = \sum_{i=0}^g \mathbf{p}(i), \quad (2.28)$$

where $\mathbf{P}(0) = \mathbf{p}(0)$ and $\mathbf{P}(K-1) = 1$. This is simply the normalized *cumulative histogram*.³

Entropy

Given an estimate of its intensity probability distribution $\mathbf{p}(g)$, the *entropy* of an image is defined as⁴

$$\begin{aligned} H(I) &= \sum_{u,v} \mathbf{p}(I(u, v)) \cdot \log_b \left(\frac{1}{\mathbf{p}(I(u, v))} \right) \\ &= - \sum_{u,v} \mathbf{p}(g) \cdot \log_b (\mathbf{p}(g)), \end{aligned} \quad (2.29)$$

where $g = I(u, v)$ and $\log_b(x)$ denotes the logarithm of x to the base b . If $b = 2$, the entropy (or “information content”) is measured in *bits*, but proportional results are obtained with any other logarithm (such as \ln or \log_{10}). Note that

³ See also Vol. 1, Sec. 3.6 [20].

⁴ Note the subtle difference in notation for the cumulative histogram \mathbf{H} and the entropy H .

the value of $H()$ is always positive, because the probabilities $p()$ are in $[0, 1]$ and thus the terms $\log_b [p()]$ are negative or zero for any b .

Some other properties of the entropy are also quite intuitive. For example, if all probabilities $p(g)$ are zero except for one intensity g' , then the entropy $H(I)$ is zero, indicating that there is no uncertainty (or “surprise”) in the messages produced by the corresponding data source. The (rather boring) images generated by this source will contain nothing but pixels of intensity g' , since all other intensities are impossible to occur. Conversely, the entropy is a maximum if all K intensities have the same probability (uniform distribution),

$$p(g) = \frac{1}{K}, \quad \text{for } 0 \leq g < K, \quad (2.30)$$

and therefore in this case (from Eqn. (2.29)) the entropy is

$$H(I) = -\sum_{i=0}^{K-1} \frac{1}{K} \cdot \log\left(\frac{1}{K}\right) = \frac{1}{K} \cdot \sum_{i=0}^{K-1} \log(K) = \log(K). \quad (2.31)$$

Thus, the entropy of a discrete source with an alphabet of K different symbols is always in the range $[0, \log(K)]$.

Using image entropy for threshold selection

The use of image entropy as a criterion for threshold selection has a long tradition and numerous methods have been proposed. In the following, we describe the early but still popular technique by Kapur et al. [52, 64] as a representative example.

Given a particular threshold q (with $0 \leq q < K-1$), the estimated probability distributions for the resulting partitions \mathcal{C}_0 and \mathcal{C}_1 are

$$\begin{aligned} \mathcal{C}_0 : & \left(\frac{p(0)}{P_0(q)} \frac{p(1)}{P_0(q)} \cdots \frac{p(q)}{P_0(q)} \quad 0 \quad 0 \quad \dots \quad 0 \right), \\ \mathcal{C}_1 : & \left(\begin{array}{cccccc} 0 & 0 & \dots & 0 & \frac{p(q+1)}{P_1(q)} & \frac{p(q+2)}{P_1(q)} & \dots & \frac{p(K-1)}{P_1(q)} \end{array} \right) \end{aligned} \quad (2.32)$$

with

$$P_0(q) = \sum_{i=0}^q p(i) = P(q) \quad \text{and} \quad P_1(q) = \sum_{i=q+1}^{K-1} p(i) = 1 - P(q). \quad (2.33)$$

$P_0(q), P_1(q)$ are the cumulative probabilities (Eqn. (2.28)) for the pixels in the background and foreground partition, respectively. Note that $P_0(q)+P_1(q) = 1$. The entropies *within* each partition are defined as

$$H_0(q) = -\sum_{i=0}^q \frac{p(i)}{P_0(q)} \cdot \log\left(\frac{p(i)}{P_0(q)}\right), \quad (2.34)$$

$$H_1(q) = - \sum_{i=q+1}^{K-1} \frac{p(i)}{P_1(q)} \cdot \log\left(\frac{p(i)}{P_1(q)}\right), \quad (2.35)$$

and the *overall* entropy for the threshold q is

$$H_{01}(q) = H_0(q) + H_1(q). \quad (2.36)$$

This expression is to be maximized over q . To allow for an efficient computation, the expression for $H_0(q)$ in Eqn. (2.34) can be rearranged to

$$\begin{aligned} H_0(q) &= - \sum_{i=0}^q \frac{p(i)}{P_0(q)} \cdot [\log(p(i)) - \log(P_0(q))] \\ &= - \frac{1}{P_0(q)} \cdot \sum_{i=0}^q p(i) \cdot [\log(p(i)) - \log(P_0(q))] \\ &= - \frac{1}{P_0(q)} \cdot \underbrace{\sum_{i=0}^q p(i) \cdot \log(p(i))}_{S_0(q)} + \frac{1}{P_0(q)} \cdot \underbrace{\sum_{i=0}^q p(i) \cdot \log(P_0(q))}_{= P_0(q)} \quad (2.37) \\ &= - \frac{1}{P_0(q)} \cdot S_0(q) + \log(P_0(q)), \end{aligned}$$

and similarly $H_1(q)$ in Eqn. (2.35) becomes

$$\begin{aligned} H_1(q) &= - \sum_{i=q+1}^{K-1} \frac{p(i)}{P_1(q)} \cdot [\log(p(i)) - \log(P_1(q))] \\ &= - \frac{1}{P_1(q)} \cdot \sum_{i=q+1}^{K-1} p(i) \cdot [\log(p(i)) - \log(P_1(q))] \\ &= - \frac{1}{P_1(q)} \cdot \underbrace{\sum_{i=q+1}^{K-1} p(i) \cdot \log(p(i))}_{S_1(q)} + \frac{1}{P_1(q)} \cdot \underbrace{\sum_{i=q+1}^{K-1} p(i) \cdot \log(P_1(q))}_{= P_1(q)} \quad (2.38) \\ &= - \frac{1}{P_1(q)} \cdot S_1(q) + \log(P_1(q)) \\ &= - \frac{1}{1-P_0(q)} \cdot S_1(q) + \log(1-P_0(q)). \end{aligned}$$

Given the estimated probability distribution $p(i)$, the cumulative probability P_0 and the summation terms S_0 , S_1 (see Eqns. (2.37–2.38)) can be calculated from the recurrence relations

$$P_0(q) = \begin{cases} p(0) & \text{for } q = 0, \\ P_0(q-1) + p(q) & \text{for } 0 < q < K, \end{cases} \quad (2.39)$$

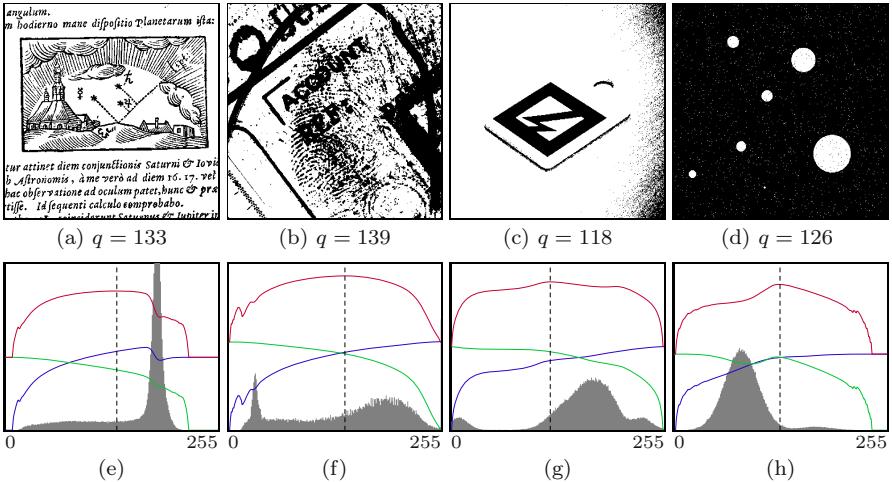


Figure 2.6 Maximum-entropy results. Calculated threshold values q and resulting binary images (a–d). Graphs in (e–h) show the *background* entropy $H_0(q)$ (green), *foreground* entropy $H_1(q)$ (blue) and *overall* entropy $H_{01}(q) = H_0(q) + H_1(q)$ (red), for varying threshold values q . The optimal threshold q_{\max} is found at the maximum of H_{01} (dashed vertical line).

$$S_0(q) = \begin{cases} p(0) \cdot \log(p(0)) & \text{for } q = 0, \\ S_0(q-1) + p(q) \cdot \log(p(q)) & \text{for } 0 < q < K, \end{cases} \quad (2.40)$$

$$S_1(q) = \begin{cases} 0 & \text{for } q = K-1, \\ S_1(q+1) + p(q+1) \cdot \log(p(q+1)) & \text{for } 0 \leq q < K-1. \end{cases} \quad (2.41)$$

The complete procedure is given in [Alg. 2.5](#), where the values $S_0(q), S_1(q)$ are obtained from precalculated tables $\mathbf{S}_0, \mathbf{S}_1$. The algorithm performs three passes over the histogram of length K (two for filling the tables $\mathbf{S}_0, \mathbf{S}_1$ and one in the main loop), so its time complexity is $\mathcal{O}(K)$, like the algorithms described before. Results obtained with this technique are shown in [Fig. 2.6](#).

The technique described in this section is simple and efficient because it again relies entirely on the image’s histogram. More advanced entropy-based thresholding techniques exist that, among other improvements, take into account the spatial structure of the original image. An extensive review of entropy-based methods can be found in [25].

2.1.6 Minimum error thresholding

The goal of minimum error thresholding is to optimally fit a combination (mixture) of Gaussian distributions to the image’s histogram. Before we proceed, we briefly look at some additional concepts from statistics. Note, however, that the following material is only intended as a superficial outline to explain the

Algorithm 2.5 Maximum entropy threshold selection after Kapur et al. [64].

```

1: MAXIMUMENTROPYTHRESHOLD( $h$ )
   Input:  $h : [0, K-1] \mapsto \mathbb{N}$ , a grayscale histogram.
   Returns the optimal threshold value or  $-1$  if no threshold is found.

2:  $K \leftarrow \text{Size}(h)$                                  $\triangleright$  number of intensity levels
3:  $p \leftarrow \text{Normalize}(h)$                           $\triangleright$  normalize histogram
4:  $(S_0, S_1) \leftarrow \text{MAKETABLES}(p, K)$ 
5:  $P_0 \leftarrow 0$                                       $\triangleright$  zero background pixels
6:  $q_{\max} \leftarrow -1$ 
7:  $H_{\max} \leftarrow -\infty$                              $\triangleright$  maximum joint entropy
8: for  $q \leftarrow 0, \dots, K-2$  do       $\triangleright$  examine all possible threshold values  $q$ 
9:    $P_0 \leftarrow P_0 + p(q)$ 
10:   $P_1 \leftarrow 1 - P_0$ 
11:   $H_0 \leftarrow \begin{cases} -\frac{1}{P_0} \cdot S_0(q) + \log(P_0) & \text{if } P_0 > 0 \\ 0 & \text{otherwise} \end{cases}$   $\triangleright$  background entropy
12:   $H_1 \leftarrow \begin{cases} -\frac{1}{P_1} \cdot S_1(q) + \log(P_1) & \text{if } P_1 > 0 \\ 0 & \text{otherwise} \end{cases}$   $\triangleright$  foreground entropy
13:   $H_{01} = H_0 + H_1$                                 $\triangleright$  overall entropy for  $q$ 
14:  if  $H_{01} > H_{\max}$  then                       $\triangleright$  maximize  $H_{01}(q)$ 
15:     $H_{\max} \leftarrow H_{01}$ 
16:     $q_{\max} \leftarrow q$ 
17: return  $q_{\max}$ 

18: MAKETABLES( $p, K$ )
19: Create maps  $S_0, S_1 : [0, K-1] \mapsto \mathbb{R}$ 
20:  $s_0 \leftarrow 0$                                       $\triangleright$  initialize table  $S_0$ 
21: for  $i \leftarrow 0, \dots, K-1$  do
22:   if  $p(i) > 0$  then
23:      $s_0 \leftarrow s_0 + p(i) \cdot \log(p(i))$ 
24:    $S_0(i) \leftarrow s_0$ 
25:    $s_1 \leftarrow 0$                                       $\triangleright$  initialize table  $S_1$ 
26:   for  $i \leftarrow K-1, \dots, 0$  do
27:      $S_1(i) \leftarrow s_1$ 
28:     if  $p(i) > 0$  then
29:        $s_1 \leftarrow s_1 + p(i) \cdot \log(p(i))$ 
30:   return  $(S_0, S_1)$ 

```

elementary concepts. For a solid grounding of these and related topics readers are referred to the excellent texts available on statistical pattern recognition, such as [13, 37].

Bayesian decision-making

The assumption is again that the image pixels originate from one of two classes, \mathcal{C}_0 and \mathcal{C}_1 , or background and foreground, respectively. Both classes generate random intensity values following unknown statistical distributions. Typically, these are modeled as Gaussian distributions with unknown parameters μ and σ^2 , as described below. The task is to decide for each pixel value x to which of the two classes it most likely belongs. Bayesian reasoning is a classic technique for making such decision in a probabilistic context.

The *probability*, that a certain intensity value x originates from a background pixel is denoted

$$p(x|\mathcal{C}_0).$$

This is called a “conditional probability”.⁵ It tells us how likely it is to see the gray value x when a pixel is a member of the background class \mathcal{C}_0 . Analogously, $p(x|\mathcal{C}_1)$ is the conditional probability of observing the value x when a pixel is known to be of the foreground class \mathcal{C}_1 . For the moment, let us assume that the conditional probability functions $p(x|\mathcal{C}_0)$ and $p(x|\mathcal{C}_1)$ are known.

Our problem is reverse though, namely to decide which class a pixel most likely belongs to, given that its intensity is x . This means that we are actually interested in the conditional probabilities

$$p(\mathcal{C}_0|x) \quad \text{and} \quad p(\mathcal{C}_1|x), \quad (2.42)$$

because if we knew these, we could simply select the class with the higher probability in the form

$$\text{assign } x \text{ to class } \begin{cases} \mathcal{C}_0 & \text{if } p(\mathcal{C}_0|x) > p(\mathcal{C}_1|x), \\ \mathcal{C}_1 & \text{otherwise.} \end{cases} \quad (2.43)$$

Bayes’ theorem provides a method for estimating these *posterior* probabilities, that is,

$$p(\mathcal{C}_j|x) = \frac{p(x|\mathcal{C}_j) \cdot p(\mathcal{C}_j)}{p(x)}, \quad (2.44)$$

⁵ In general, $p(A|B)$ denotes the (conditional) probability of observing the event A in a given situation B . It is usually read as “the probability of A , given B ”.

where $p(\mathcal{C}_j)$ is the (unknown) *prior* probability of class \mathcal{C}_j . In other words, $p(\mathcal{C}_0)$, $p(\mathcal{C}_1)$ are the “*a priori*” probabilities of any pixel belonging to the background or the foreground, respectively. Finally, $p(x)$ in Eqn. (2.44) is the overall probability of observing the intensity value x (also called “evidence”), which is typically estimated from its relative frequency in the image. Note that for a particular intensity x , the corresponding evidence $p(x)$ only *scales* the posterior probabilities and is thus not relevant for the classification itself. Consequently, we can reformulate the binary decision rule in Eqn. (2.43) to

$$\text{assign } x \text{ to class } \begin{cases} \mathcal{C}_0 & \text{if } p(x|\mathcal{C}_0) \cdot p(\mathcal{C}_0) > p(x|\mathcal{C}_1) \cdot p(\mathcal{C}_1), \\ \mathcal{C}_1 & \text{otherwise.} \end{cases} \quad (2.45)$$

This is called *Bayes’ decision rule*. It minimizes the probability of making a classification error if the involved probabilities are known and is also called the “minimum error” criterion.

If the probability distributions $p(x|\mathcal{C}_j)$ are modeled as *Gaussian* distributions⁶ $\mathcal{N}(x|\mu_j, \sigma_j^2)$, where μ_j, σ_j^2 denote the *mean* and *variance* of class \mathcal{C}_j , respectively, we can rewrite the scaled posterior probabilities as

$$p(x|\mathcal{C}_j) \cdot p(\mathcal{C}_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \cdot \exp\left(-\frac{(x-\mu_j)^2}{2\sigma_j^2}\right) \cdot p(\mathcal{C}_j). \quad (2.46)$$

As long as the ordering between the resulting class scores is maintained, these quantities can be scaled or transformed arbitrarily. In particular, it is common to use the *logarithm* of the above expression to avoid repeated multiplications of small numbers. For example, applying the natural logarithm⁷ to both sides of Eqn. (2.46) yields

$$\begin{aligned} \ln(p(x|\mathcal{C}_j) \cdot p(\mathcal{C}_j)) &= \ln(p(x|\mathcal{C}_j)) + \ln(p(\mathcal{C}_j)) \\ &= \ln\left(\frac{1}{\sqrt{2\pi\sigma_j^2}}\right) + \ln\left(\exp\left(-\frac{(x-\mu_j)^2}{2\sigma_j^2}\right)\right) + \ln(p(\mathcal{C}_j)) \\ &= -\frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(\sigma_j^2) - \frac{(x-\mu_j)^2}{2\sigma_j^2} + \ln(p(\mathcal{C}_j)) \\ &= -\frac{1}{2} \cdot \left[\ln(2\pi) + \frac{(x-\mu_j)^2}{\sigma_j^2} + \ln(\sigma_j^2) - 2\ln(p(\mathcal{C}_j)) \right]. \end{aligned} \quad (2.47)$$

Since $\ln(2\pi)$ in Eqn. (2.47) is constant, it can be ignored for the classification decision, as well as the factor $\frac{1}{2}$ at the front. Thus, to find the class j that

⁶ See also Appendix C.3.

⁷ Any logarithm could be used but the natural logarithm complements the exponential function of the Gaussian.

maximizes $p(x|\mathcal{C}_j) \cdot p(\mathcal{C}_j)$ for a given intensity value x , it is sufficient to *maximize* the quantity

$$-\left[\frac{(x - \mu_j)^2}{\sigma_j^2} + 2 \cdot [\ln(\sigma_j) - \ln(p(\mathcal{C}_j))] \right] \quad (2.48)$$

or, alternatively, to *minimize*

$$\varepsilon_j(x) = \frac{(x - \mu_j)^2}{\sigma_j^2} + 2 \cdot [\ln(\sigma_j) - \ln(p(\mathcal{C}_j))]. \quad (2.49)$$

The quantity $\varepsilon_j(x)$ can be viewed as a *measure of the potential error* involved in classifying the observed value x as being of class \mathcal{C}_j . To obtain the decision associated with the minimum risk, we can modify the binary decision rule in Eqn. (2.45) to

$$\text{assign } x \text{ to class } \begin{cases} \mathcal{C}_0 & \text{if } \varepsilon_0(x) \leq \varepsilon_1(x), \\ \mathcal{C}_1 & \text{otherwise.} \end{cases} \quad (2.50)$$

Remember that this rule tells us how to correctly classify the observed intensity value x as being either of the background class \mathcal{C}_0 or the foreground class \mathcal{C}_1 , assuming that the underlying distributions are really Gaussian and their parameters are well estimated.

If we apply a threshold q , all pixel values $g \leq q$ are implicitly classified as \mathcal{C}_0 (background) and all $g > q$ as \mathcal{C}_1 (foreground). The goodness of this classification by q over all N image pixels $I(u, v)$ can be measured with the criterion function

$$\begin{aligned} \mathbf{e}(q) &= \frac{1}{N} \cdot \sum_{u,v} \begin{cases} \varepsilon_0(I(u,v)) & \text{for } I(u,v) \leq q \\ \varepsilon_1(I(u,v)) & \text{for } I(u,v) > q \end{cases} \\ &= \frac{1}{N} \cdot \sum_{g=0}^q \mathbf{h}(g) \cdot \varepsilon_0(g) + \frac{1}{N} \cdot \sum_{g=q+1}^{K-1} \mathbf{h}(g) \cdot \varepsilon_1(g) \\ &= \sum_{g=0}^q \mathbf{p}(g) \cdot \varepsilon_0(g) + \sum_{g=q+1}^{K-1} \mathbf{p}(g) \cdot \varepsilon_1(g), \end{aligned} \quad (2.51)$$

with the normalized frequencies $\mathbf{p}(g) = \mathbf{h}(g)/N$ and the function $\varepsilon_j(g)$ as defined in Eqn. (2.49). By substituting $\varepsilon_j(g)$ from Eqn. (2.49) and some mathematical gymnastics, $\mathbf{e}(q)$ can be written as

$$\begin{aligned} \mathbf{e}(q) &= 1 + P_0(q) \cdot \ln(\sigma_0^2(q)) + P_1(q) \cdot \ln(\sigma_1^2(q)) \\ &\quad - 2 \cdot P_0(q) \cdot \ln(P_0(q)) - 2 \cdot P_1(q) \cdot \ln(P_1(q)). \end{aligned} \quad (2.52)$$

The remaining task is to find the threshold q that *minimizes* $\mathbf{e}(q)$ (where the constant 1 in Eqn. (2.52) can be omitted, of course). For each possible threshold

q , we only need to estimate (from the image's histogram, as in Eqn. (2.32)) the ‘prior’ probabilities $P_0(q)$, $P_1(q)$ and the corresponding within-class variances $\sigma_0(q)$, $\sigma_1(q)$. The prior probabilities for the background and foreground classes are estimated as

$$\begin{aligned} P_0(q) &\approx \sum_{g=0}^q p(g) = \frac{1}{N} \cdot \sum_{g=0}^q h(g) = \frac{n_0(q)}{N}, \\ P_1(q) &\approx \sum_{g=q+1}^{K-1} p(g) = \frac{1}{N} \cdot \sum_{g=q+1}^{K-1} h(g) = \frac{n_1(q)}{N}, \end{aligned} \quad (2.53)$$

where $n_0(q) = \sum_{i=0}^q h(i)$, $n_1(q) = \sum_{i=q+1}^{K-1} h(i)$, and $N = n_0(q) + n_1(q)$ is the total number of image pixels. Estimates for background and foreground variances $\sigma_0^2(q)$, $\sigma_1^2(q)$, defined in Eqns. (2.11–2.12), can be calculated efficiently by expressing them in the form

$$\begin{aligned} \sigma_0^2(q) &\approx \frac{1}{n_0(q)} \cdot \left[\underbrace{\sum_{g=0}^q h(g) \cdot g^2}_{B_0(q)} - \frac{1}{n_0(q)} \cdot \underbrace{\left(\sum_{g=0}^q h(g) \cdot g \right)^2}_{A_0(q)} \right] \\ &= \frac{1}{n_0(q)} \cdot \left[B_0(q) - \frac{1}{n_0(q)} \cdot A_0^2(q) \right], \end{aligned} \quad (2.54)$$

$$\begin{aligned} \sigma_1^2(q) &\approx \frac{1}{n_1(q)} \cdot \left[\underbrace{\sum_{g=q+1}^{K-1} h(g) \cdot g^2}_{B_1(q)} - \frac{1}{n_1(q)} \cdot \underbrace{\left(\sum_{g=q+1}^{K-1} h(g) \cdot g \right)^2}_{A_1(q)} \right] \\ &= \frac{1}{n_1(q)} \cdot \left[B_1(q) - \frac{1}{n_1(q)} \cdot A_1^2(q) \right], \end{aligned} \quad (2.55)$$

with the quantities

$$\begin{aligned} A_0(q) &= \sum_{g=0}^q h(g) \cdot g, & B_0(q) &= \sum_{g=0}^q h(g) \cdot g^2, \\ A_1(q) &= \sum_{g=q+1}^{K-1} h(g) \cdot g, & B_1(q) &= \sum_{g=q+1}^{K-1} h(g) \cdot g^2. \end{aligned} \quad (2.56)$$

Furthermore, the values $\sigma_0^2(q)$, $\sigma_1^2(q)$ can be tabulated for every possible q in only two passes over the histogram, using the following recurrence relations:

$$A_0(q) = \begin{cases} 0 & \text{for } q = 0, \\ A_0(q-1) + h(q) \cdot q & \text{for } 1 \leq q \leq K-1, \end{cases} \quad (2.57)$$

$$B_0(q) = \begin{cases} 0 & \text{for } q = 0, \\ B_0(q-1) + h(q) \cdot q^2 & \text{for } 1 \leq q \leq K-1, \end{cases} \quad (2.58)$$

$$A_1(q) = \begin{cases} 0 & \text{for } q = K-1, \\ A_1(q+1) + h(q+1) \cdot (q+1) & \text{for } 0 \leq q \leq K-2, \end{cases} \quad (2.59)$$

$$B_1(q) = \begin{cases} 0 & \text{for } q = K-1, \\ B_1(q+1) + h(q+1) \cdot (q+1)^2 & \text{for } 0 \leq q \leq K-2. \end{cases} \quad (2.60)$$

The complete minimum-error threshold calculation is summarized in [Alg. 2.6](#). First, the tables σ_0^2, σ_1^2 are set up and initialized with the values of $\sigma_0^2(q), \sigma_1^2(q)$, respectively, for $0 \leq q < K$, following the recursive scheme in Eqns. (2.57–2.60). Subsequently, the error value $e(q)$ is calculated for every possible threshold value q to find the global minimum. Again $e(q)$ can only be calculated for those values of q , for which both resulting partitions are non-empty (i.e., with $n_0(q), n_1(q) > 0$). Note that, in lines 22 and 30 of [Alg. 2.6](#), a small constant ($\frac{1}{12}$) is added to the variance to avoid zero values when the corresponding class population is homogeneous (i.e., only contains a single intensity value).⁸ This ensures that the algorithm works properly on images with only two distinct gray values. The algorithm computes the optimal threshold by performing three passes over the histogram (two for initializing the tables and one for finding the minimum); it thus has the same time complexity of $\mathcal{O}(K)$ as the algorithms described before.

[Figure 2.7](#) shows the results of minimum-error thresholding on our set of test images. It also shows the fitted pair of Gaussian distributions for the background and the foreground pixels, respectively, for the optimal threshold as well as the graphs of the error function $e(q)$, which is minimized over all threshold values q . Obviously the error function is quite flat in certain cases, indicating that similar scores are obtained for a wide range of threshold values and the optimal threshold is not very distinct. We can also see that the estimate is quite accurate in the case of the synthetic test image in [Fig. 2.7 \(d\)](#), which is actually generated as a mixture of two Gaussians (with parameters $\mu_0 = 80$, $\mu_1 = 170$ and $\sigma_0 = \sigma_1 = 20$).

A minor theoretical problem with the minimum error technique is that the parameters of the Gaussian distributions are always estimated from *truncated* samples. This means that, for any threshold q , only the intensity values smaller than q are used to estimate the parameters of the background distribution, and only the intensities greater than q contribute to the foreground parameters. In practice, this problem is of minor importance, since the distributions are typically not strictly Gaussian either.

⁸ This is justified by the fact that each histogram bin $h(i)$ represents intensities in the continuous range $[i \pm 0.5]$ and the variance of uniformly distributed values in the unit interval is $\frac{1}{12}$.

Algorithm 2.6 Minimum error threshold selection based on a Gaussian mixture model (after Kittler and Illingworth [57]).

```

1: MINIMUMERRORTHRESHOLD( $h$ )
   Input:  $h : [0, K-1] \mapsto \mathbb{N}$ , a grayscale histogram.
   Returns the optimal threshold value or  $-1$  if no threshold is found.

2:  $K \leftarrow \text{Size}(h)$ 
3:  $(\sigma_0^2, \sigma_1^2, N) \leftarrow \text{MAKESIGMATABLES}(h, K)$ 
4:  $n_0 \leftarrow 0, q_{\min} \leftarrow -1, e_{\min} \leftarrow \infty$ 
5: for  $q \leftarrow 0, \dots, K-2$  do            $\triangleright$  evaluate all possible thresholds  $q$ 
6:    $n_0 \leftarrow n_0 + h(q)$                    $\triangleright$  background population
7:    $n_1 \leftarrow N - n_0$                     $\triangleright$  foreground population
8:   if  $(n_0 > 0) \wedge (n_1 > 0)$  then
9:      $P_0 \leftarrow n_0/N$                    $\triangleright$  prior probability of  $C_0$ 
10:     $P_1 \leftarrow n_1/N$                    $\triangleright$  prior probability of  $C_1$ 
11:     $e \leftarrow P_0 \cdot \ln(\sigma_0^2(q)) + P_1 \cdot \ln(\sigma_1^2(q))$ 
         $- 2 \cdot (P_0 \cdot \ln(P_0) + P_1 \cdot \ln(P_1))$        $\triangleright$  Eqn. (2.52)
12:    if  $e < e_{\min}$  then                 $\triangleright$  minimize error for  $q$ 
13:       $e_{\min} \leftarrow e, q_{\min} \leftarrow q$ 
14: return  $q_{\min}$ 

15: MAKESIGMATABLES( $h, K$ )
16: Create maps  $\sigma_0^2, \sigma_1^2 : [0, K-1] \mapsto \mathbb{R}$ 
17:  $n_0 \leftarrow 0, A_0 \leftarrow 0, B_0 \leftarrow 0$ 
18: for  $q \leftarrow 0, \dots, K-1$  do            $\triangleright$  tabulate  $\sigma_0^2(q)$ 
19:    $n_0 \leftarrow n_0 + h(q)$ 
20:    $A_0 \leftarrow A_0 + h(q) \cdot q$            $\triangleright$  Eqn. (2.57)
21:    $B_0 \leftarrow B_0 + h(q) \cdot q^2$         $\triangleright$  Eqn. (2.58)
22:    $\sigma_0^2(q) \leftarrow \begin{cases} \frac{1}{12} + (B_0 - A_0^2/n_0)/n_0 & \text{for } n_0 > 0 \\ 0 & \text{otherwise} \end{cases}$        $\triangleright$  Eqn. (2.54)
23:    $N \leftarrow n_0$ 
24:    $n_1 \leftarrow 0, A_1 \leftarrow 0, B_1 \leftarrow 0$ 
25:    $\sigma_1^2(K-1) \leftarrow 0$ 
26:   for  $q \leftarrow K-2, \dots, 0$  do            $\triangleright$  tabulate  $\sigma_1^2(q)$ 
27:      $n_1 \leftarrow n_1 + h(q+1)$ 
28:      $A_1 \leftarrow A_1 + h(q+1) \cdot (q+1)$        $\triangleright$  Eqn. (2.59)
29:      $B_1 \leftarrow B_1 + h(q+1) \cdot (q+1)^2$        $\triangleright$  Eqn. (2.60)
30:      $\sigma_1^2(q) \leftarrow \begin{cases} \frac{1}{12} + (B_1 - A_1^2/n_1)/n_1 & \text{for } n_1 > 0 \\ 0 & \text{otherwise} \end{cases}$        $\triangleright$  Eqn. (2.55)
31: return  $(\sigma_0^2, \sigma_1^2, N)$ 
```

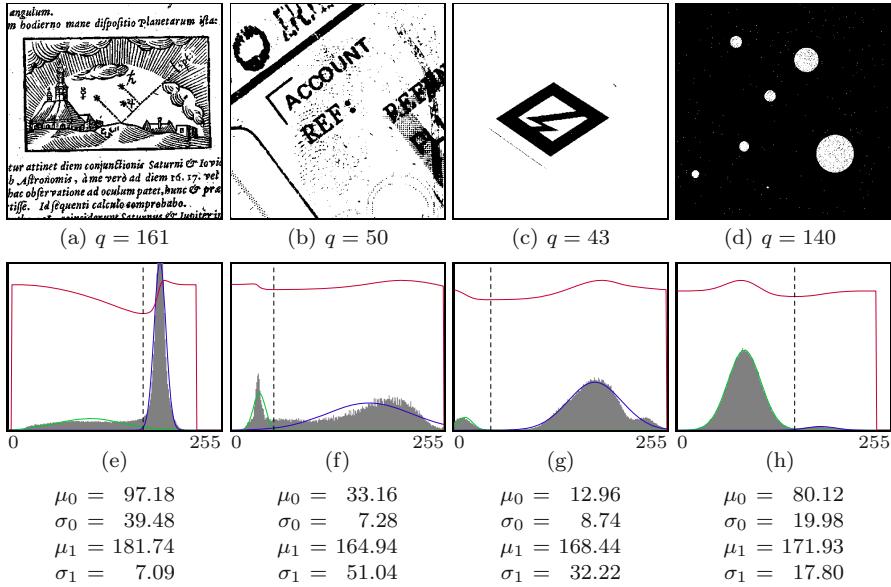


Figure 2.7 Results from minimum-error thresholding. Calculated threshold values q and resulting binary images (a–d). The green and blue graphs in (e–h) show the fitted Gaussian background and foreground distributions $\mathcal{N}_0 = (\mu_0, \sigma_0)$ and $\mathcal{N}_1 = (\mu_1, \sigma_1)$, respectively. The red graph corresponds to the error quantity $e(q)$ for varying threshold values $q = 0, \dots, 255$ (see Eqn. (2.52)). The optimal threshold q_{\min} is located at the *minimum* of $e(q)$ (dashed vertical line). The estimated parameters of the background/foreground Gaussians are listed at the bottom.

2.2 Local adaptive thresholding

In many situations, a fixed threshold is not appropriate to classify the pixels in the entire image, for example, when confronted with stained backgrounds, uneven lighting or exposure. [Figure 2.8](#) shows a typical, unevenly exposed document image and the results obtained with some global thresholding methods described in the previous sections.

Instead of using a single threshold value for the whole image, adaptive thresholding specifies a *varying* threshold value $Q(u, v)$ for each image position that is used to classify the corresponding pixel $I(u, v)$ in the same way as described in Eqn. (2.1) for a global threshold. The following approaches differ only with regard to how the variable threshold Q is derived from the input image.

2.2.1 Bernsen's method

The method proposed by Bernsen [12] specifies a dynamic threshold for each image position (u, v) , based on the minimum and maximum intensity found in

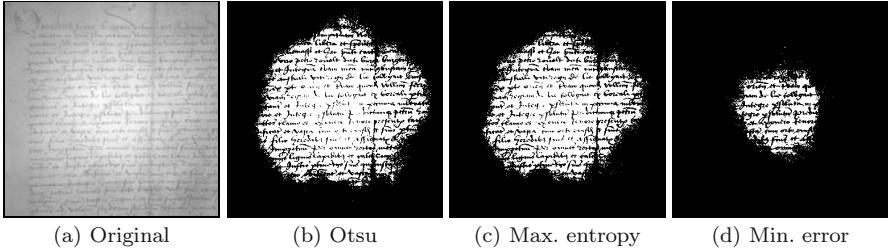


Figure 2.8 Global thresholding methods fail under uneven lighting or exposure. Original image (a), results from global thresholding with various methods described above (b–d).

a local neighborhood $R(u, v)$. If

$$\begin{aligned} I_{\min}(u, v) &= \min_{(i,j) \in R(u,v)} I(i, j), \\ I_{\max}(u, v) &= \max_{(i,j) \in R(u,v)} I(i, j) \end{aligned} \quad (2.61)$$

are the minimum and maximum intensity values within a fixed-size neighborhood region R centered at position (u, v) , the space-varying threshold is simply calculated as the *mid-range* value

$$Q(u, v) \leftarrow \frac{I_{\min}(u, v) + I_{\max}(u, v)}{2}. \quad (2.62)$$

This is done as long as the local contrast $c(u, v) = I_{\max}(u, v) - I_{\min}(u, v)$ is above some predefined limit c_{\min} . If $c(u, v) < c_{\min}$, the pixels in the corresponding image region are assumed to belong to a single class and are (by default) assigned to the background.

The whole process is summarized in [Alg. 2.7](#). The main function provides a control parameter bg to select the proper default threshold \bar{q} , which is set to K in case of a dark background ($bg = \text{dark}$) and to 0 for a bright background ($bg = \text{bright}$). The support region R may be square or circular, typically with a radius $r = 15$. The choice of the minimum contrast limit c_{\min} depends on the type of imagery and the noise level ($c_{\min} = 15$ is a suitable value to start with).

[Figure 2.9](#) shows the results of Bernsen’s method on the uneven test image used in [Fig. 2.8](#) for different settings of the region’s radius r . Due to the non-linear min- and max-operation, the resulting threshold surface is not smooth. The minimum contrast is set to $c_{\min} = 15$, which is too low to avoid thresholding low-contrast noise visible along the left image margin. By increasing the minimum contrast c_{\min} , more neighborhoods are considered “flat” and thus ignored, i. e., classified as background. This is demonstrated in [Fig. 2.10](#). While larger values of c_{\min} effectively eliminate low-contrast noise, relevant structures are also lost, which illustrates the difficulty of finding a suitable global value

Algorithm 2.7 Adaptive thresholding using local contrast (after Bernsen [12]). The argument to *bg* should be set to **dark** if the image background is darker than the structures of interest, and to **bright** if the background is brighter than the objects.

```

1: BERNSENTHRESHOLD( $I, r, c_{\min}, bg$ )
   Input:  $I$ , intensity image of size  $M \times N$ ;  $r$ , radius of support region;
           $c_{\min}$ , minimum contrast;  $bg$ , background type (dark or bright).
   Returns a map with an individual threshold value for each image
   position.
2:  $(M, N) \leftarrow \text{Size}(I)$ 
3: Create map  $Q : M \times N \mapsto \mathbb{R}$ 
4:  $\bar{q} \leftarrow \begin{cases} K & \text{if } bg = \text{dark} \\ 0 & \text{if } bg = \text{bright} \end{cases}$ 
5: for all image coordinates  $(u, v) \in M \times N$  do
6:    $R \leftarrow \text{MAKECIRCULARREGION}(u, v, r)$ 
7:    $I_{\min} \leftarrow \min_{(i,j) \in R} I(i, j)$ 
8:    $I_{\max} \leftarrow \max_{(i,j) \in R} I(i, j)$ 
9:    $c \leftarrow I_{\max} - I_{\min}$ 
10:   $Q(u, v) \leftarrow \begin{cases} (I_{\min} + I_{\max})/2 & \text{if } c \geq c_{\min} \\ \bar{q} & \text{otherwise} \end{cases}$ 
11: return  $Q$ 


---


12: MAKECIRCULARREGION( $u, v, r$ )
   Returns the set of pixel coordinates within the circle of radius  $r$ ,
   centered at  $(u, v)$ 
13: return  $\{(i, j) \in \mathbb{Z}^2 \mid (u - i)^2 + (v - j)^2 \leq r^2\}$ 
```

for c_{\min} . Additional examples, using the test images previously used for global thresholding, are shown in Fig. 2.11.

What Alg. 2.7 describes formally can be implemented quite efficiently, noting that the calculation of local minima and maxima over a sliding window (lines 6–8) corresponds to a simple nonlinear filter operation (see Vol. 1, Sec. 5.4 [20]). To perform these calculations, we can use a *minimum* and *maximum* filter with radius r , as provided by virtually every image processing environment. For example, the Java implementation of the Bernsen thresholding in Prog. 2.1 uses ImageJ’s built-in **RankFilters** class for this purpose. The complete implementation can be found on the book’s website (see Sec. 2.3 for additional details on the corresponding API).

```

1 package threshold;
2
3 import ij.plugin.filter.RankFilters;
4 import ij.process.ByteProcessor;
5
6 public class BernsenThresholder extends AdaptiveThresholder {
7     private int radius = 15;
8     private int cmin = 15;
9     private BackgroundMode bgMode = BackgroundMode.DARK;
10
11    // default constructor:
12    public BernsenThresholder() {
13    }
14
15    // specific constructor:
16    public BernsenThresholder(int radius, int cmin,
17        BackgroundMode bgMode) {
18        this.radius = radius;
19        this.cmin = cmin;
20        this.bgMode = bgMode;
21    }
22
23    public ByteProcessor getThreshold(ByteProcessor I) {
24        int width = I.getWidth();
25        int height = I.getHeight();
26        ByteProcessor Imin = (ByteProcessor) I.duplicate();
27        ByteProcessor Imax = (ByteProcessor) I.duplicate();
28
29        RankFilters rf = new RankFilters();
30        rf.rank(Imin, radius, RankFilters.MIN); //  $I_{\min}(u, v)$ 
31        rf.rank(Imax, radius, RankFilters.MAX); //  $I_{\max}(u, v)$ 
32
33        int q = (bgMode == BackgroundMode.DARK) ? 256 : 0;
34        ByteProcessor Q = new ByteProcessor(width, height);
35
36        for (int v = 0; v < height; v++) {
37            for (int u = 0; u < width; u++) {
38                int gMin = Imin.get(u, v);
39                int gMax = Imax.get(u, v);
40                int c = gMax - gMin;
41                if (c >= cmin)
42                    Q.set(u, v, (gMin + gMax) / 2);
43                else
44                    Q.set(u, v, q);
45            }
46        }
47        return Q; //  $Q(u, v)$ 
48    }
49 }
```

Program 2.1 Bernsen's thresholder (ImageJ plugin implementation of [Alg. 2.7](#)). Note the use of ImageJ's `RankFilters` class (lines 29–31) for calculating the local minimum (`Imin`) and maximum (`Imax`) maps inside the `getThreshold()` method.

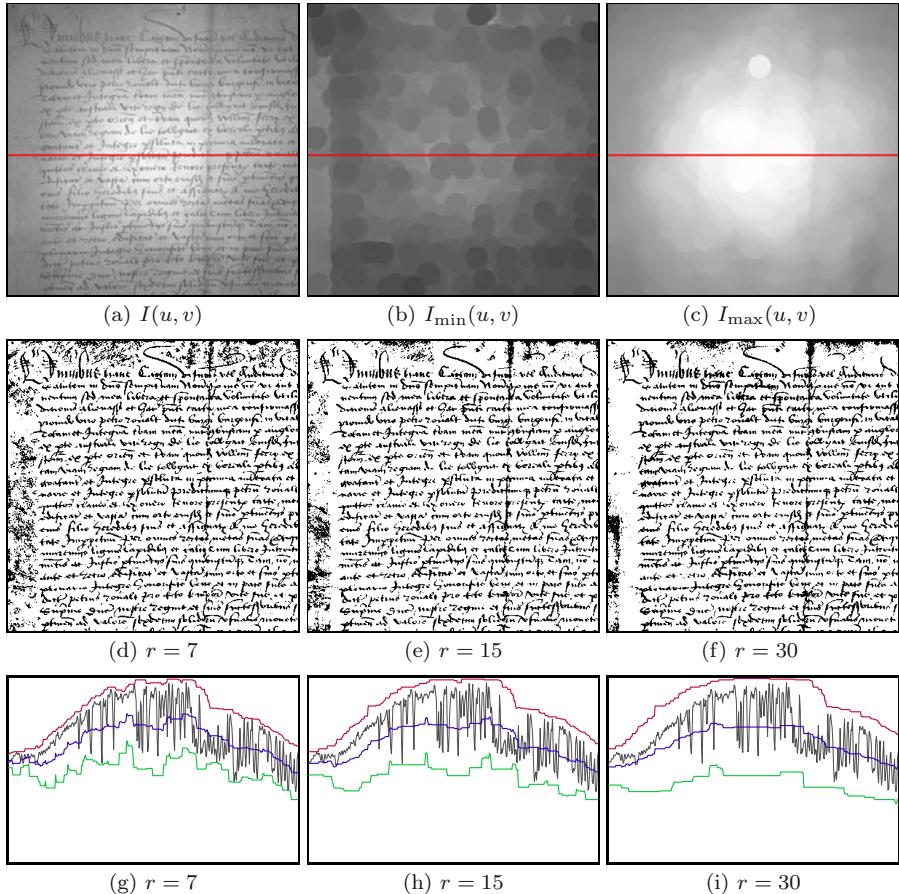


Figure 2.9 Adaptive thresholding using Bernsen’s method. Original image (a), local minimum (b) and maximum (c). The center row shows the binarized images for different settings of r (d–f). The corresponding curves in (g–i) show the original intensity (gray), local minimum (green), maximum (red), and the actual threshold (blue) along the horizontal line marked in (a–c). The region radius r is 15 pixels, the minimum contrast c_{\min} is 15 intensity units.

2.2.2 Niblack’s method

In this approach, originally presented in [91, Sec. 5.1], the threshold $Q(u, v)$ is varied across the image as a function of the local intensity average $\mu_R(u, v)$ and standard deviation⁹ $\sigma_R(u, v)$ in the form

$$Q(u, v) = \mu_R(u, v) + \kappa \cdot \sigma_R(u, v). \quad (2.63)$$

⁹ The standard deviation σ is the square root of the variance σ^2 .

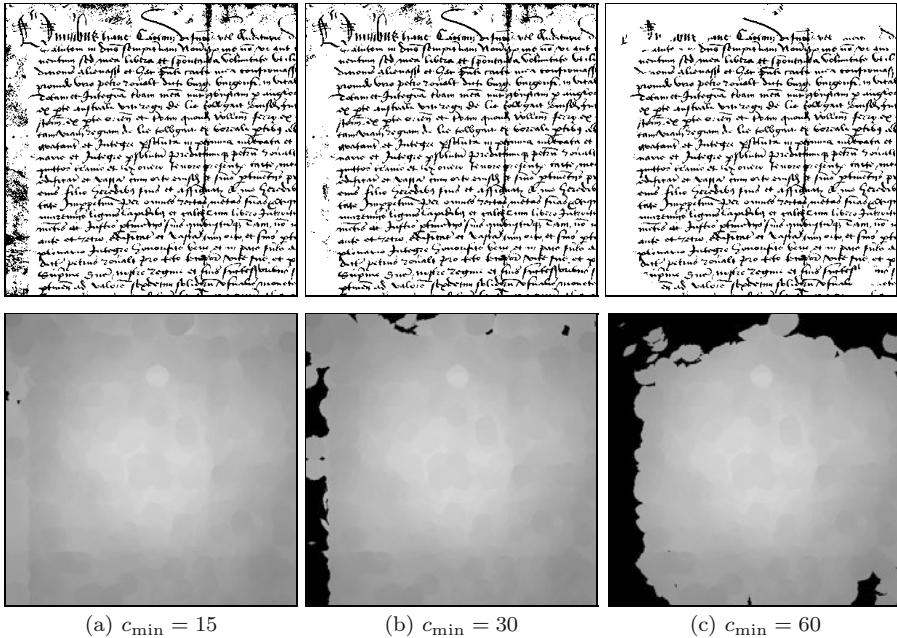


Figure 2.10 Adaptive thresholding using Bernsen’s method with different settings of c_{\min} . Binarized images (top row) and threshold surface $Q(u, v)$ (bottom row). Black areas in the threshold functions indicate that the local contrast is below c_{\min} ; the corresponding pixels are classified as background (white in this case).

Thus the local threshold $Q(u, v)$ is determined by adding a constant portion ($\kappa \geq 0$) of the local standard deviation σ_R to the local mean μ_R . μ_R and σ_R are calculated over a square support region R centered at (u, v) . The size (radius) of the averaging region R should be as large as possible, at least larger than the size of the structures to be detected, but small enough to capture the variations (unevenness) of the background. A size of 31×31 pixels (or radius $r = 15$) is suggested in [91] and $\kappa = 0.18$, though the latter does not seem to be critical.

One problem is that, for small values of σ_R (as obtained in “flat” image regions of approximately constant intensity), the threshold will be close to the local average, which makes the segmentation quite sensitive to low-amplitude noise (“ghosting”). A simple improvement is to secure a minimum distance from the mean by adding a constant offset d , i. e., replacing Eqn. (2.63) by

$$Q(u, v) = \mu_R(u, v) + \kappa \cdot \sigma_R(u, v) + d, \quad (2.64)$$

with $d \geq 0$, in the range $2, \dots, 20$ for typical 8-bit images.

The original formulation (Eqn. (2.63)) is aimed at situations where the foreground structures are *brighter* than the background (Fig. 2.12 (a)) but does not work if the images are set up the other way round (Fig. 2.12 (b)). In the

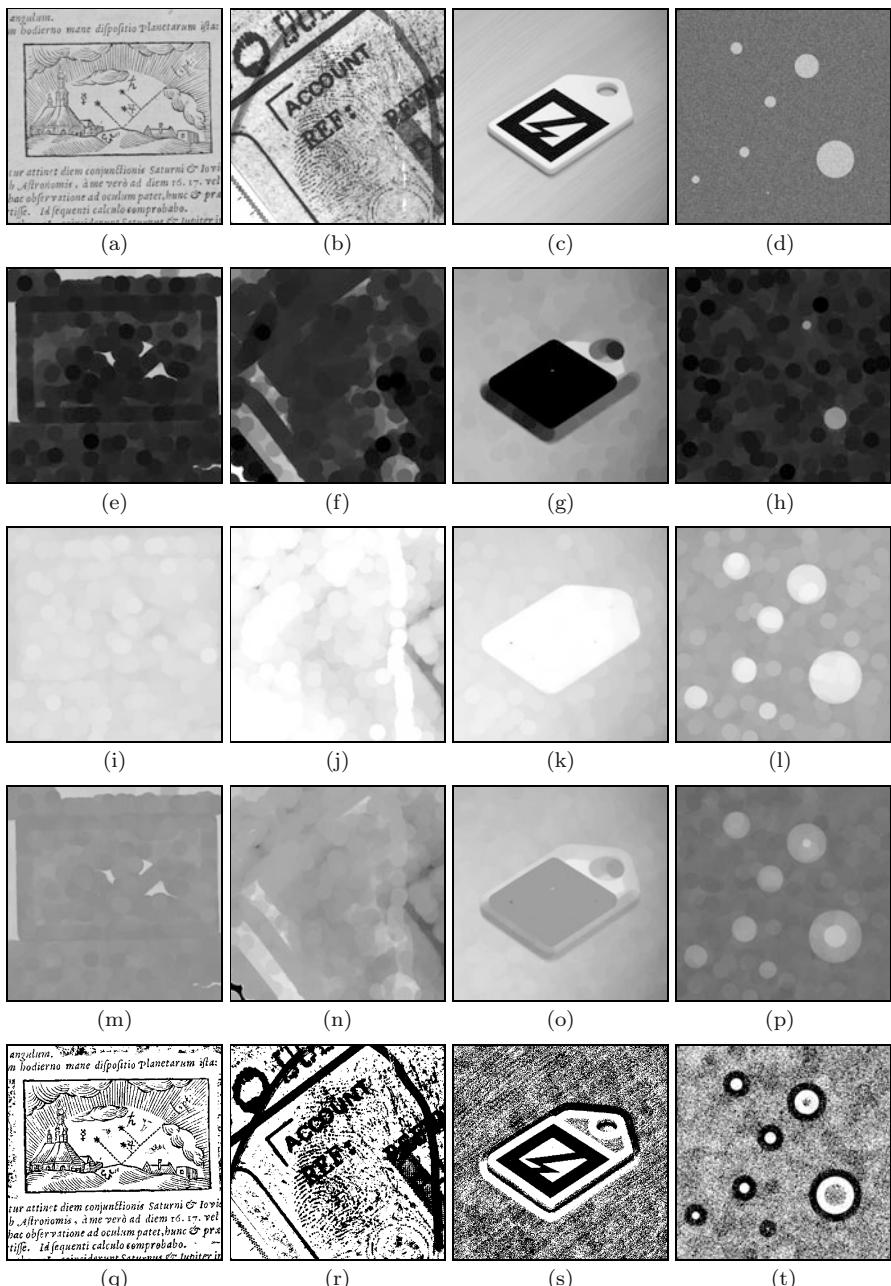


Figure 2.11 Additional examples for Bernsen’s method. Original images (a–d), local minimum I_{\min} (e–h), maximum I_{\max} (i–l), and threshold map Q (m–p); results after thresholding the images (q–t). Settings are $r = 15$, $c_{\min} = 15$. A bright background is assumed for all images ($bg = \text{bright}$), except for image (d).

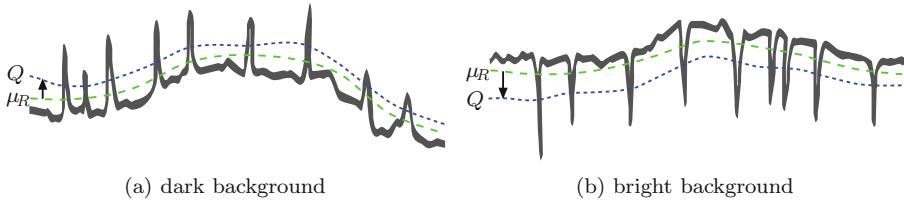


Figure 2.12 Adaptive thresholding based on average local intensity. The illustration shows a line profile as typically found in document imaging. The space-variant threshold Q (dotted blue line) is chosen as the local average μ_R (dashed green line) and subsequently offset by a multiple of the local intensity variation σ_R . The offset is chosen to be *positive* for images with a dark background and bright structures (a) and *negative* if the background is brighter than the contained structures of interest (b).

case that the structures of interest are *darker* than the background (as, for example, in typical OCR applications), one could either work with inverted images or modify the calculation of the threshold to

$$Q(u, v) = \begin{cases} \mu_R(u, v) + (\kappa \cdot \sigma_R(u, v) + d) & \text{for a } \textit{dark} \text{ background,} \\ \mu_R(u, v) - (\kappa \cdot \sigma_R(u, v) + d) & \text{for a } \textit{bright} \text{ background.} \end{cases} \quad (2.65)$$

The modified procedure is detailed in [Alg. 2.8](#). The example in [Fig. 2.13](#) shows results obtained with this method on an image with a bright background containing dark structures, for $\kappa = 0.3$ and varying settings of d . Note that setting $d = 0$ ([Fig. 2.13 \(d, g\)](#)) corresponds to Niblack's original method. For these examples, a circular window of radius $r = 15$ was used to compute the local mean $\mu_R(u, v)$ and variance $\sigma_R(u, v)$. Additional examples are shown in [Fig. 2.14](#). Note that the selected radius r is obviously too small for the structures in images [Fig. 2.14 \(c, d\)](#), which are thus not segmented cleanly. Better results can be expected with a larger radius.

With the intent to improve upon Niblack's method, particularly for thresholding deteriorated text images, Sauvola and Pietikäinen [116] proposed setting the threshold to

$$Q(u, v) = \mu_R(u, v) \cdot \left[1 + \kappa \cdot \left(\frac{\sigma_R(u, v)}{\sigma_{\max}} - 1 \right) \right], \quad (2.66)$$

with $\kappa = 0.5$ and $\sigma_{\max} = 128$ (the “dynamic range of the standard deviation” for 8-bit images) as suggested parameter values. In this approach, the offset between the threshold and the local average not only depends on the local variation σ_R (as in Eqn. (2.63)), but also on the magnitude of the local *mean* μ_R . Thus, changes in absolute brightness lead to modified relative threshold values, even when the image contrast remains constant. Though this technique is frequently referenced in the literature it appears questionable if this behavior is generally desirable.

Algorithm 2.8 Adaptive thresholding using local mean and variance (modified version of Niblack's method [91]). The argument to bg should be `dark` if the image background is darker than the structures of interest, `bright` if the background is brighter than the objects.

```

1: NIBLACKTHRESHOLD( $I, r, \kappa, d, bg$ )
   Input:  $I$ , intensity image of size  $M \times N$ ;  $r$ , radius of support region;
           $\kappa$ , variance control parameter;  $d$ , minimum offset;  $bg \in \{\text{dark}, \text{bright}\}$ ,
          background type.
   Returns a map with an individual threshold value for each image
   position.
2:  $(M, N) \leftarrow \text{Size}(I)$ 
3: Create map  $Q : M \times N \mapsto \mathbb{R}$ 
4: for all image coordinates  $(u, v) \in M \times N$  do
   Define a support region of radius  $r$ , centered at  $(u, v)$ :
5:  $(\mu, \sigma^2) \leftarrow \text{GETLOCALMEANANDVARIANCE}(I, u, v, r)$ 
6:  $\sigma \leftarrow \sqrt{\sigma^2}$                                  $\triangleright$  local standard deviation  $\sigma_R$ 
7:  $Q(u, v) \leftarrow \begin{cases} \mu + (\kappa \cdot \sigma + d) & \text{if } bg = \text{dark} \\ \mu - (\kappa \cdot \sigma + d) & \text{if } bg = \text{bright} \end{cases}$        $\triangleright$  Eqn. (2.65)
8: return  $Q$ 


---


9: GETLOCALMEANANDVARIANCE( $I, u, v, r$ )
   Returns the local mean and variance of the image pixels  $I(i, j)$  within
   the disk-shaped region  $R$ .
10:  $R \leftarrow \text{MAKECIRCULARREGION}(u, v, r)$             $\triangleright$  see Alg. 2.7
11:  $N \leftarrow 0, A \leftarrow 0, B \leftarrow 0$ 
12: for all  $(i, j) \in R$  do
13:    $N \leftarrow N + 1$ 
14:    $A \leftarrow A + I(i, j)$ 
15:    $B \leftarrow B + I^2(i, j)$ 
16:    $\mu \leftarrow \frac{1}{N} \cdot A$ 
17:    $\sigma^2 \leftarrow \frac{1}{N} \cdot (B - \frac{1}{N} \cdot A^2)$ 
18: return  $(\mu, \sigma^2)$ 
```

Computing local average and variance

Algorithm 2.8 shows the principle operation of Niblack's method and also illustrates how to efficiently calculate the local average and variance. Given the image I and the averaging region R , we can use the shortcut suggested in Eqn. (2.4) to obtain these quantities as

$$\mu_R = \frac{1}{N} \cdot A \quad \text{and} \quad \sigma_R^2 = \frac{1}{N} \cdot \left(B - \frac{1}{N} \cdot A^2 \right), \quad (2.67)$$

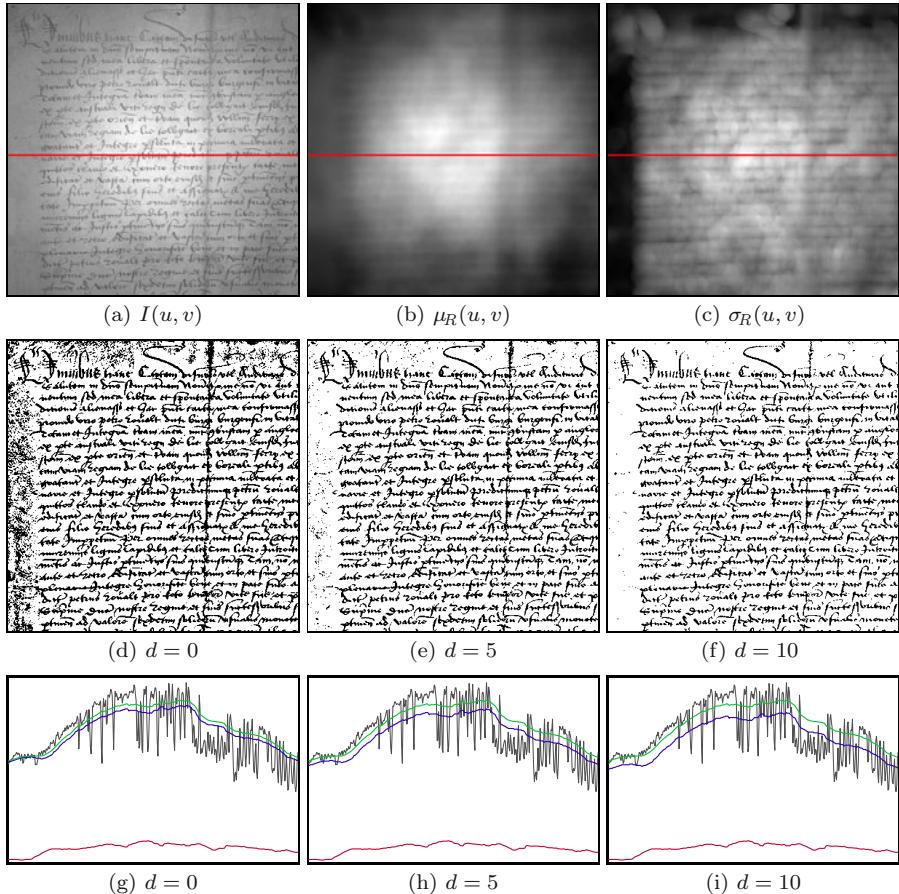


Figure 2.13 Adaptive thresholding using Niblack’s method (with $r = 15$, $\kappa = 0.3$). Original image (a), local mean μ_R (b) and standard deviation σ_R (c). The result for $d = 0$ in (d) corresponds to Niblack’s original formulation. Increasing the value of d reduces the amount of clutter in regions with low variance (e, f). The curves in (g–i) show the local intensity (gray), mean (green), variance (red), and the actual threshold (blue) along the horizontal line marked in (a–c).

respectively, with $A = \sum_{(i,j) \in R} I(i, j)$, $B = \sum_{(i,j) \in R} I^2(i, j)$, and $N = |R|$. Procedure GETLOCALMEANANDVARIANCE() in Alg. 2.8 shows this calculation in full detail. When computing the local average and variance, attention must be paid to the situation at the image borders, as illustrated in Fig. 2.15. Two approaches are frequently used. In the first approach (following the common practice for implementing filter operations), all outside pixel values are replaced by the closest inside pixel, which is always a border pixel. Thus the border pixel values are effectively replicated outside the image boundaries and thus these pixels have a strong influence on the local results. The second approach is to

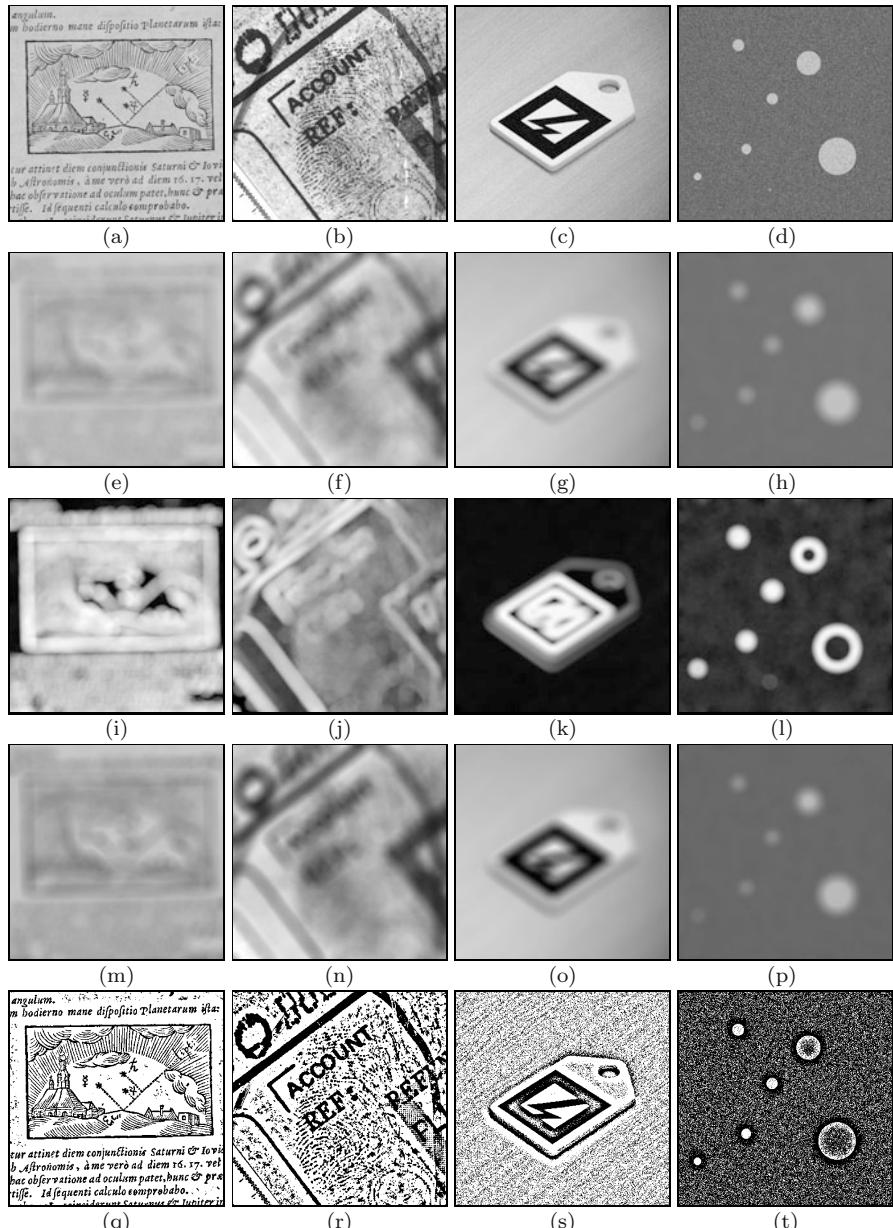


Figure 2.14 Additional examples for thresholding with Niblack’s method using a disk-shaped support region of radius $r = 15$. Original images (a–d), local mean μ_R (e–h), standard deviation σ_R (i–l), and threshold Q (m–p); results after thresholding the images (q–t). The background is assumed to be brighter than the structures of interest, except for image (d), which has a dark background. Settings are $\kappa = 0.3$, $d = 5$.

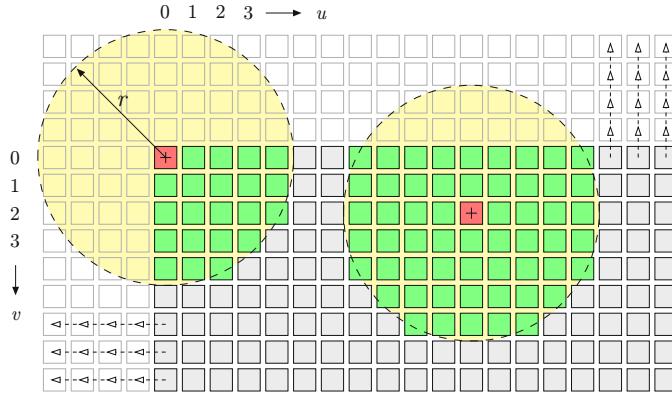


Figure 2.15 Calculating local statistics at image boundaries. The illustration shows a disk-shaped support region with radius r , placed at the image border. Pixel values outside the image can be replaced (“filled-in”) by the closest border pixel, as is common in many filter operations. Alternatively, the calculation of the local statistics can be confined to include only those pixels inside the image that are actually covered by the support region. At any border pixel, the number of covered elements (N) is still more than $\approx 1/4$ of the full region size. In this particular case, the circular region covers a maximum of $N = 69$ pixels when fully embedded and $N = 22$ when positioned at an image corner.

perform the calculation of the average and variance on only those image pixels that are actually covered by the support region. In this case, the number of pixels (N) is reduced at the image borders to about $1/4$ of the full region size.

Although the calculation of the local mean and variance outlined by function `GETLOCALMEANANDVARIANCE()` in Alg. 2.8 is definitely more efficient than a brute-force approach, additional optimizations are possible. Most image processing environments have suitable routines already built in. With ImageJ, for example, we can again use the `RankFilters` class (as with the *min-* and *max-filters* in the *Bernsen* approach, see Sec. 2.2.1). Instead of performing the computation for each pixel individually, the following ImageJ code segment uses predefined filters to compute two separate images `Imean` (μ_R) and `Ivar` (σ_R^2) containing the local mean and variance values, respectively, with a disk-shaped support region of radius 15:

```

1 ByteProcessor I; // original image I(u, v)
2 int radius = 15;
3
4 FloatProcessor Imean = (FloatProcessor) I.convertToFloat();
5 FloatProcessor Ivar = (FloatProcessor) Imean.duplicate();
6
7 RankFilters rf = new RankFilters();
8 rf.rank(Imean, radius, RankFilters.MEAN);           //  $\mu_R(u, v)$ 
9 rf.rank(Ivar, radius, RankFilters.VARIANCE);        //  $\sigma_R^2(u, v)$ 
10 ...

```

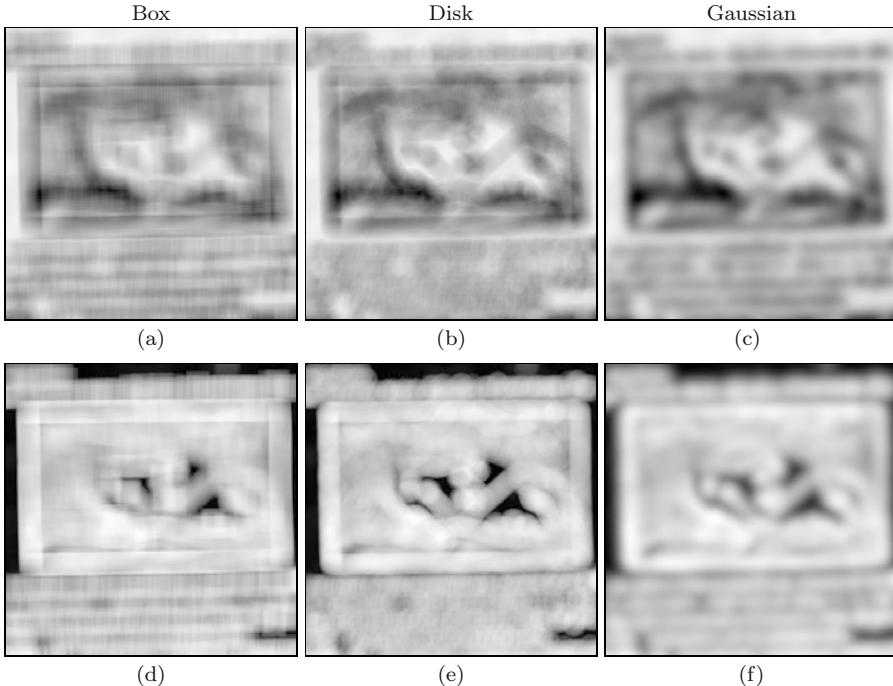


Figure 2.16 Local average (a–c) and variance (d–f) obtained with different smoothing kernels. 31×31 box filter (a, d), disk filter with radius $r = 15$ (b, e), Gaussian kernel with $\sigma = 0.6 \cdot 15 = 9.0$ (c, f). Both the box and disk filter show strong truncation effects (ringing), the box filter is also highly non-isotropic. All images are contrast-enhanced for better visibility.

See Section 2.3 and the online code for additional implementation details. Note that the filter methods implemented in `RankFilters` perform replication of border pixels as the border handling strategy, as discussed above.

Local average and variance with Gaussian kernels

The purpose of taking the local average is to smooth the image to obtain an estimate of the varying background intensity. In case of a square or circular region, this is equivalent to convolving the image with a box- or disk-shaped kernel, respectively. Kernels of this type, however, are not well suited for image smoothing because they create strong ringing and truncating effects, as demonstrated in Fig. 2.16. Moreover, convolution with a box-shaped (rectangular) kernel is a non-isotropic operation, i. e., the results are orientation-dependent. From this perspective alone it seems appropriate to consider other smoothing kernels, Gaussian kernels in particular.

Using a Gaussian kernel H^G for smoothing is equivalent to calculating a

weighted average of the corresponding image pixels, with the weights being the coefficients of the kernel. Calculating this weighted local average can be expressed as

$$\mu_G(u, v) = \frac{1}{\sum H^G} \cdot (I * H^G)(u, v), \quad (2.68)$$

where $\sum H^G$ is the sum of the coefficients in the kernel H^G , and $*$ denotes the linear convolution operator.¹⁰ Analogously, there is also a weighted *variance* σ_G^2 which can be calculated jointly with the local average μ_G (as in Eqn. (2.67)) in the form

$$\mu_G(u, v) = \frac{1}{\sum H^G} \cdot A_G(u, v), \quad (2.69)$$

$$\sigma_G^2(u, v) = \frac{1}{\sum H^G} \cdot (B_G(u, v) - \frac{1}{\sum H^G} \cdot A_G^2(u, v)), \quad (2.70)$$

with $A_G = I * H^G$ and $B_G = I^2 * H^G$.

Thus all we need are two filter operations, one applied to the original image ($I * H^G$) and another applied to the squared image ($I^2 * H^G$), using the same 2D Gaussian kernel H^G (or any other suitable smoothing kernel). If the kernel H^G is *normalized* (i.e., $\sum H^G = 1$), Eqns. (2.69–2.70) reduce to

$$\mu_G(u, v) = A_G(u, v), \quad (2.71)$$

$$\sigma_G^2(u, v) = B_G(u, v) - A_G^2(u, v), \quad (2.72)$$

with A_G, B_G as defined above.

This suggests a very simple process for computing the local average and variance by Gaussian filtering, as summarized in [Alg. 2.9](#). The width (standard deviation σ) of the Gaussian kernel is set to 0.6 times the radius r of the corresponding disk filter to produce a similar effect as [Alg. 2.8](#). The Gaussian approach has two advantages. First, the Gaussian makes a much superior low-pass filter, compared to the box or disk kernels. Second, the 2D Gaussian is (unlike the circular disk kernel) separable in the x - and y -direction, which permits a very efficient implementation of the 2D filter using only a pair of 1D convolutions (see Vol. 1, Sec. 5.2 [20]). Note that separability of the Gaussian is not used in [Alg. 2.9](#), which is meant for demonstration only.

In practice, A_G, B_G can be represented as (floating-point) images and most modern image processing environments provide efficient (multi-scale) implementations of Gaussian filters with large-size kernels. In ImageJ, fast Gaussian filtering is implemented by the class `GaussianBlur` with the public methods `blur()`, `blurGaussian()`, and `blurFloat()`, which all use normalized filter kernels by default. Programs 2.2–2.3 show the complete ImageJ implementation of Niblack's thresholder using Gaussian smoothing kernels.

¹⁰ Also see Vol. 1 [20, Sec. 5.3.1].

Algorithm 2.9 Adaptive thresholding using Gaussian averaging (extended from Alg. 2.8). Parameters are the original image I , the radius r of the Gaussian kernel, variance control k , and minimum offset d . The argument to bg should be **dark** if the image background is darker than the structures of interest, **bright** if the background is brighter than the objects. The procedure $\text{MAKEGAUSSIANKERNEL2D}(\sigma)$ creates a discrete, normalized 2D Gaussian kernel with standard deviation σ .

```

1: ADAPTIVETHRESHOLDGAUSS( $I, r, \kappa, d, bg$ )
   Input:  $I$ , intensity image of size  $M \times N$ ;  $r$ , support region radius;
           $\kappa$ , variance control parameter;  $d$ , minimum offset;  $bg \in \{\text{dark}, \text{bright}\}$ ,
          background type.
   Returns a map  $Q$  of local thresholds for the grayscale image  $I$ .
2:  $(M, N) \leftarrow \text{Size}(I)$ 
3: Create maps  $A_G, B_G, Q : M \times N \mapsto \mathbb{R}$ 
4: for all image coordinates  $(u, v) \in M \times N$  do
5:    $A_G(u, v) \leftarrow I(u, v)$ 
6:    $B_G(u, v) \leftarrow (I(u, v))^2$ 
7:    $H^G \leftarrow \text{MAKEGAUSSIANKERNEL2D}(0.6 \cdot r)$ 
8:    $A_G \leftarrow A_G * H^G$                                  $\triangleright$  filter the original image with  $H^G$ 
9:    $B_G \leftarrow B_G * H^G$                                  $\triangleright$  filter the squared image with  $H^G$ 
10:  for all image coordinates  $(u, v) \in M \times N$  do
11:     $\mu_G \leftarrow A_G(u, v)$                                  $\triangleright$  Eqn. (2.71)
12:     $\sigma_G \leftarrow \sqrt{B_G(u, v) - A_G^2(u, v)}$                  $\triangleright$  Eqn. (2.72)
13:     $Q(u, v) \leftarrow \begin{cases} \mu_G + (\kappa \cdot \sigma_G + d) & \text{if } bg = \text{dark} \\ \mu_G - (\kappa \cdot \sigma_G + d) & \text{if } bg = \text{bright} \end{cases}$        $\triangleright$  Eqn. (2.65)
14:  return  $Q$ 


---


15: MAKEGAUSSIANKERNEL2D( $\sigma$ )
   Returns a discrete 2D Gaussian kernel  $H$  with standard deviation  $\sigma$ ,
   sized sufficiently large to avoid truncation effects.
16:  $r \leftarrow \max(1, \lceil 3.5 \cdot \sigma \rceil)$                    $\triangleright$  size the kernel sufficiently large
17: Create map  $H : [-r, r]^2 \mapsto \mathbb{R}$ 
18:  $s \leftarrow 0$ 
19: for  $x \leftarrow -r, \dots, r$  do
20:   for  $y \leftarrow -r, \dots, r$  do
21:      $H(x, y) \leftarrow e^{-\frac{x^2+y^2}{2 \cdot \sigma^2}}$              $\triangleright$  unnormalized 2D Gaussian
22:      $s \leftarrow s + H(x, y)$ 
23:   for  $x \leftarrow -r, \dots, r$  do
24:     for  $y \leftarrow -r, \dots, r$  do
25:        $H(x, y) \leftarrow \frac{1}{s} \cdot H(x, y)$                    $\triangleright$  normalize  $H$ 
26:   return  $H$ 

```

```

1 package threshold;
2 import ij.plugin.filter.GaussianBlur;
3 import ij.process.ByteProcessor;
4 import ij.process.FloatProcessor;
5
6 public class NiblackThresholderGauss extends AdaptiveThresholder {
7
8     private int radius = 15;
9     private double kappa = 0.18;
10    private int dmin = 0;
11    private BackgroundMode bgMode = BackgroundMode.BRIGHT; // or DARK;
12
13    private FloatProcessor Imu; // =  $\mu_G(u, v)$ 
14    private FloatProcessor Isigma; // =  $\sigma_G(u, v)$ 
15
16    // default constructor:
17    public NiblackThresholderGauss() {
18    }
19
20    // specific constructor:
21    public NiblackThresholderGauss(int radius, double kappa, int dmin,
22        BackgroundMode bgMode) {
23        this.radius = radius; // = r
24        this.kappa = kappa; // =  $\kappa$ 
25        this.dmin = dmin; // =  $d_{\min}$ 
26        this.bgMode = bgMode;
27    }
28
29    public ByteProcessor getThreshold(ByteProcessor ip) {
30        int w = ip.getWidth();
31        int h = ip.getHeight();
32        makeMeanAndVariance(ip, radius); // see Prog. 2.3
33        ByteProcessor Q = new ByteProcessor(w, h);
34
35        for (int v = 0; v < h; v++) {
36            for (int u = 0; u < w; u++) {
37                double sigma = Isigma.getf(u, v);
38                double mu = Imu.getf(u, v);
39                double diff = kappa * sigma + dmin;
40                int q = (int) Mathrint((bgMode == BackgroundMode.DARK) ?
41                    mu + diff : mu - diff);
42                if (q < 0) q = 0;
43                if (q > 255) q = 255;
44                Q.set(u, v, q);
45            }
46        }
47    }
48    // continues in Prog. 2.3

```

Program 2.2 Niblack's thresholder using Gaussian smoothing kernels (ImageJ implementation of Alg. 2.9, part 1).

```

49  // continued from Prog. 2.2
50
51  private void makeMeanAndVariance(ByteProcessor ip, int rad) {
52      int width = ip.getWidth();
53      int height = ip.getHeight();
54
55      Imu = new FloatProcessor(width,height);
56      Isigma = new FloatProcessor(width,height);
57
58      FloatProcessor AG = (FloatProcessor) ip.convertToFloat(); // = I
59      FloatProcessor BG = (FloatProcessor) AG.duplicate(); // = I
60      BG.sqr(); // = I2
61
62      GaussianBlur gb = new GaussianBlur();
63
64      // σ of Gaussian filter should be approx. 0.6 of the disk's radius:
65      double sigma = rad * 0.6;
66
67      gb.blurFloat(AG, sigma, sigma, 0.002); // → AG
68      gb.blurFloat(BG, sigma, sigma, 0.002); // → BG
69
70      for (int v=0; v<height; v++) {
71          for (int u=0; u<width; u++) {
72              float a = AG.getf(u,v);
73              float b = BG.getf(u,v);
74              float muG = a;
75              float sigmaG = (float) Math.sqrt(b-a*a); // see Eqn. (2.72)
76              Imu.setf(u, v, muG); // = μG(u, v)
77              Isigma.setf(u, v, sigmaG); // = σG(u, v)
78          }
79      }
80  }
81 }
```

Program 2.3 Niblack’s thresholder using Gaussian smoothing kernels (part 2). The floating-point images A_G and B_G correspond to the maps A_G (filtered original image) and B_G (filtered squared image) in Alg. 2.9. An instance of the ImageJ class `GaussianBlur` is created in line 62 and subsequently used to filter both images in lines 67–68. The last argument to the ImageJ method `blurFloat` (0.002) specifies the accuracy of the Gaussian kernel.

2.3 Java implementation

All thresholding methods described in this chapter have been implemented in a small Java library that is available with full source code at the book’s website. The top class in this library¹¹ is `Thresholder` with the sub-classes `GlobalThresholder` and `AdaptiveThresholder` for the methods described in Sections 2.1 and 2.2, respectively.

¹¹ Package `imagingbook.threshold`

Threshold (class)

Class **Threshold** itself is abstract and only defines a set of non-public utility methods for histogram analysis and the public enumeration type

```
enum BackgroundMode {BRIGHT, DARK}
```

for specifying the assumed type of image background.

GlobalThreshold (class)

This abstract sub-class of **Threshold** is the super-class for the implementations of the global thresholding methods described in Section 2.1. It provides a single public method,

```
int getThreshold (ByteProcessor bp)
```

Returns the optimal threshold q for the 8-bit grayscale image in **bp** as an **int** value.

The following example demonstrates the typical use of this method for a given **ByteProcessor** object **bp**:

```
1  ...
2  GlobalThreshold thr = new IsodataThreshold();
3  int q = thr.getThreshold(bp);
4  bp.threshold(q);
5  ...
```

Available real sub-classes of **GlobalThreshold** that can be instantiated (as in line 2 above) are

IsodataThreshold (class)

ISODATA thresholding, see Section 2.1.3 and [Alg. 2.2](#) (p. 13).

MaxEntropyThreshold (class)

Maximum entropy thresholding, see Section 2.1.5 and [Alg. 2.5](#) (p. 23).

MinErrorThreshold (class)

Minimum error thresholding, see Section 2.1.6 and [Alg. 2.6](#) (p. 29).

OtsuThreshold (class)

Otsu thresholding, see Section 2.1.4 and [Alg. 2.4](#) (p. 17).

QuantileThreshold (class)

Simple threshold selection algorithm, see Section 2.1.2 and [Alg. 2.1](#) (p. 11).

AdaptiveThreshold (class)

This abstract sub-class of **Threshold** is the super-class for the implementations of the local adaptive thresholding methods described in Section 2.2. It provides the following public methods:

```
ByteProcessor getThreshold (ByteProcessor bp)
    Returns the threshold  $Q(u, v)$  for each pixel of the 8-bit grayscale input
    image bp as a ByteProcessor of identical size.

void threshold (ByteProcessor bp, ByteProcessor Q)
    Thresholds the image bp adaptively by applying the threshold map Q.
    bp and Q must be of identical size, bp is destructively modified.

void threshold (ByteProcessor bp)
    Adaptively thresholds the 8-bit grayscale image bp using a threshold
    map calculated with getThreshold(bp).
```

The following example demonstrates the typical use of these methods for a given **ByteProcessor** object **bp**:

```
1  ...
2  AdaptiveThresholder thr = new BernsenThresholder();
3  ByteProcessor Q = thr.getThreshold(bp);
4  thr.threshold(bp, Q);
5  ...
```

Alternatively, the same operation can be defined without making **Q** explicit as

```
6  ...
7  AdaptiveThresholder thr = new BernsenThresholder();
8  thr.threshold(bp);
9  ...
```

Available real sub-classes of **AdaptiveThresholder** that can be instantiated (as in line 2 above) are

BernsenThresholder (class)

Implements Bernsen's adaptive thresholding method, see Section 2.2.1 and [Alg. 2.7](#) (on p. 32).

NiblackThresholderBox (class)

Implements Niblack's adaptive thresholding method using a square support region for calculating the local mean and average, see Section 2.2.2 and [Alg. 2.8](#) (p. 38).

NiblackThresholderDisk (class)

Implements Niblack's adaptive thresholding method using a disk-shaped support region.

NiblackThresholderGauss (class)

Implements Niblack's adaptive thresholding method using Gaussian kernels for calculating the local mean and average, see [Alg. 2.9](#) (p. 44) and the implementation in Progs. 2.2–2.3.

SauvolaThresholder (class)

Implements the adaptive thresholding method proposed by Sauvola and Pietikäinen [116], see Eqn. (2.66).

2.4 Summary and further reading

The intention of this chapter is to give an overview of established methods for automatic image thresholding. A vast body of relevant literature exists and thus only a fraction of the proposed techniques could be discussed here. For additional approaches and references, several excellent surveys are available, including [46, 96, 113, 128] and, more recently, [118].

Given the obvious limitations of global techniques, adaptive thresholding methods have received continued interest and are still a focus of ongoing research. Another popular approach is to calculate an adaptive threshold through image decomposition. In this case, the image is partitioned into (possibly overlapping) tiles, an “optimal” threshold is calculated for each tile and the adaptive threshold is obtained by interpolation between adjacent tiles. Another interesting idea, proposed in [149], is to specify a “threshold surface” by sampling the image at specific points that exhibit a high gradient, with the assumption that these points are at transitions between the background and the foreground. Interpolation between these irregularly spaced point samples is done by solving a Laplacian difference equation to obtain a continuous “potential surface”. This is accomplished with the so-called “successive over-relaxation” method, which requires about N scans over an image of size $N \times N$ to converge, so its time complexity is an expensive $\mathcal{O}(N^3)$. A more efficient approach was proposed in [14], which uses a hierarchical, multi-scale algorithm for interpolating the threshold surface. Similarly, a quad-tree representation was used for this purpose in [27]. Another concept relevant in this context is “kriging” [94], which was originally developed for interpolating two-dimensional geological data [109, Sec. 3.7.4].

In the case of color images, simple thresholding is often applied individually to each color channel and the results are subsequently merged using a suitable logical operation. Transformation to a non-RGB color space (such as HSV or CIELAB) might be helpful for this purpose. For a binarization method aimed specifically at vector-valued images, see [84], for example. Since thresholding can be viewed as a specific form of segmentation, color segmentation methods [28, 30, 45, 119] are also relevant for binarizing color images.

2.5 Exercises

Exercise 2.1

Define a procedure for estimating the minimum and maximum pixel value of an image from its histogram. Threshold the image at the resulting mid-range value (see Eqn. (2.17)). Can anything be said about the size of the resulting partitions?

Exercise 2.2

Define a procedure for estimating the median of an image from its histogram. Threshold the image at the resulting median value (see Eqn. (2.16)) and verify that the foreground and background partitions are of approximately equal size.

Exercise 2.3

The algorithms described in this chapter assume 8-bit grayscale input images (of type `ByteProcessor` in ImageJ). Adopt the current implementations to work with 16-bit integer image (of type `ShortProcessor`). Images of this type may contain pixel values in the range $[0, 2^{16} - 1]$ and the `getHistogram()` method returns the histogram as an integer array of length 65536.

Exercise 2.4

Implement simple thresholding for RGB color images by thresholding each (scalar-valued) color channel individually and then merging the results by performing a pixel-wise AND operation. Compare the results to those obtained by thresholding the corresponding grayscale (luminance) images.

3

Filters for Color Images

Color images are everywhere and filtering them is such a common task that it does not seem to require much attention at all. In this chapter, we describe how classical linear and non-linear filters, which we covered before in the context of grayscale images, can be either used directly or adapted for the processing of color images. Often color images are treated as stacks of intensity images and existing monochromatic filters are simply applied independently to the individual color channels. While this is straightforward and performs satisfactorily in many situations, it does not take into account the vector-valued nature of color pixels as samples taken in a specific, multi-dimensional color space. As we show in this chapter, the outcome of filter operations depends strongly on the working color space and the variations between different color spaces may be substantial. Although this may not be apparent in many situations, it should be of concern if high-quality color imaging is an issue.

3.1 Linear filters

Linear filters are important in many applications, such as smoothing, noise removal, interpolation for geometric transformations, decimation in scale-space transformations, image compression, reconstruction and edge enhancement. The general properties of linear filters and their use on scalar-valued grayscale images are detailed in Section 5.2 of Volume 1 [20]. For color images, it is common practice to apply these monochromatic filters separately to each color channel, thereby treating the image as a stack of scalar-valued images. As we

describe in the following section, this approach is simple as well as efficient, since existing implementations for grayscale images can be reused without any modification. However, the outcome depends strongly on the choice of the color space in which the filter operation is performed. For example, it makes a great difference if the channels of an RGB image contain linear or nonlinear component values. This issue is discussed in more detail in Section 3.1.2.

3.1.1 Using monochromatic linear filters on color images

Given a discrete *scalar* (grayscale) image with elements $I(u, v) \in \mathbb{R}$, the application of a linear filter can be expressed as a linear 2D convolution¹

$$\bar{I}(u, v) = [I * H](u, v) = \sum_{(i,j) \in \mathcal{R}_H} I(u-i, v-j) \cdot H(i, j), \quad (3.1)$$

where H denotes a discrete filter kernel defined over the (usually rectangular) region \mathcal{R}_H , with $H(i, j) \in \mathbb{R}$. For a *vector*-valued image \mathbf{I} with K components (i.e., $\mathbf{I}(u, v) \in \mathbb{R}^K$), the above linear filter can be written in the form

$$\bar{\mathbf{I}}(u, v) = [\mathbf{I} * H](u, v) = \sum_{(i,j) \in \mathcal{R}_H} \mathbf{I}(u-i, v-j) \cdot H(i, j), \quad (3.2)$$

with the same scalar-valued filter kernel H as in Eqn. (3.1). Thus the k th element of the resulting pixels,

$$\bar{I}_k(u, v) = \sum_{(i,j) \in \mathcal{R}_H} I_k(u-i, v-j) \cdot H(i, j) = [I_k * H](u, v), \quad (3.3)$$

is simply the result of scalar convolution (Eqn. (3.1)) applied to the corresponding component plane I_k . In the case of an RGB color image (with $K = 3$ components), the filter kernel H is applied separately to the scalar-valued R , G and B planes (I_R, I_G, I_B), that is,

$$\bar{\mathbf{I}}(u, v) = \begin{pmatrix} \bar{I}_R(u, v) \\ \bar{I}_G(u, v) \\ \bar{I}_B(u, v) \end{pmatrix} = \begin{pmatrix} [I_R * H](u, v) \\ [I_G * H](u, v) \\ [I_B * H](u, v) \end{pmatrix}. \quad (3.4)$$

[Figure 3.1](#) illustrates how linear filters for color images are typically implemented.

¹ See also Vol. 1, Sec. 5.3.1 [20].

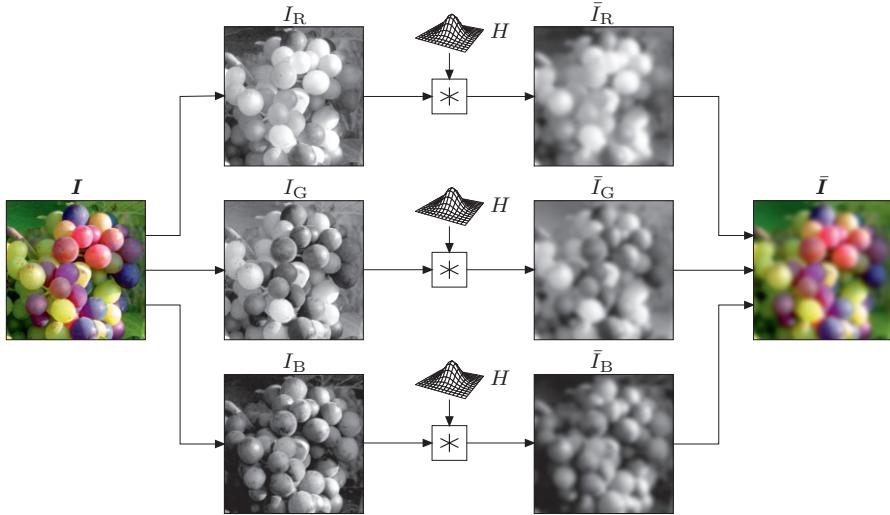


Figure 3.1 Common implementation of linear filters for RGB color images. The filter, specified by the kernel H , is applied separately to each of the scalar-valued color channels I_R , I_G , I_B . Combining the filtered component channels \bar{I}_R , \bar{I}_G , \bar{I}_B produces the filtered color image \bar{I} .

Linear smoothing filters

Smoothing filters are a particular class of linear filters that are found in many applications and characterized by positive-only filter coefficients. Let $C_{u,v} = (\mathbf{c}_1, \dots, \mathbf{c}_n)$ denote the vector of color pixels $\mathbf{c}_m \in \mathbb{R}^K$ contained in the spatial support region of the kernel H , placed at position (u, v) in the original image \mathbf{I} , where n is the size of H . With arbitrary kernel coefficients $H(i, j) \in \mathbb{R}$, the resulting color pixel $\bar{\mathbf{c}}(u, v) = \bar{\mathbf{c}}$ in the filtered image is a *linear combination* of the original colors in $C_{u,v}$, that is,

$$\bar{\mathbf{c}} = \alpha_1 \cdot \mathbf{c}_1 + \alpha_2 \cdot \mathbf{c}_2 + \cdots + \alpha_n \cdot \mathbf{c}_n, \quad (3.5)$$

where α_m is the coefficient in H that corresponds to pixel \mathbf{c}_m . If the kernel is *normalized* (i.e., $\sum H(i, j) = \sum \alpha_m = 1$), the result is an *affine combination* of the original colors. In case of a typical smoothing filter, with H normalized and all coefficients $H(i, j)$ being *positive*, any resulting color $\bar{\mathbf{c}}$ is a *convex combination* of the original color vectors $\mathbf{c}_1, \dots, \mathbf{c}_n$.

Geometrically this means that the mixed color $\bar{\mathbf{c}}$ is contained within the *convex hull* of the contributing colors $\mathbf{c}_1, \dots, \mathbf{c}_n$, as illustrated in Fig. 3.2. In the special case that only *two* original colors $\mathbf{c}_1, \mathbf{c}_2$ are involved,² the result $\bar{\mathbf{c}}$

² The convex hull of *two* points $\mathbf{c}_1, \mathbf{c}_2$ consists of the straight line segment between them.

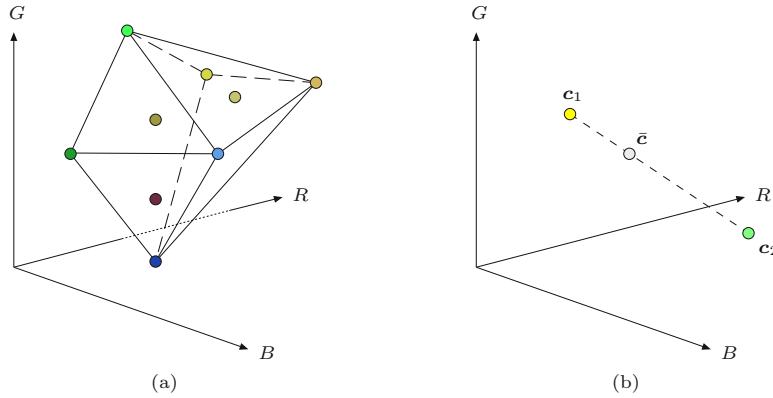


Figure 3.2 Convex linear color mixtures. The result of the convex combination (mixture) of n color vectors $\mathcal{C} = \{c_1, \dots, c_n\}$ is confined to the convex hull of \mathcal{C} (a). In the special case of only two initial colors c_1 and c_2 , any mixed color \bar{c} is located on the straight line segment connecting c_1 and c_2 (b).

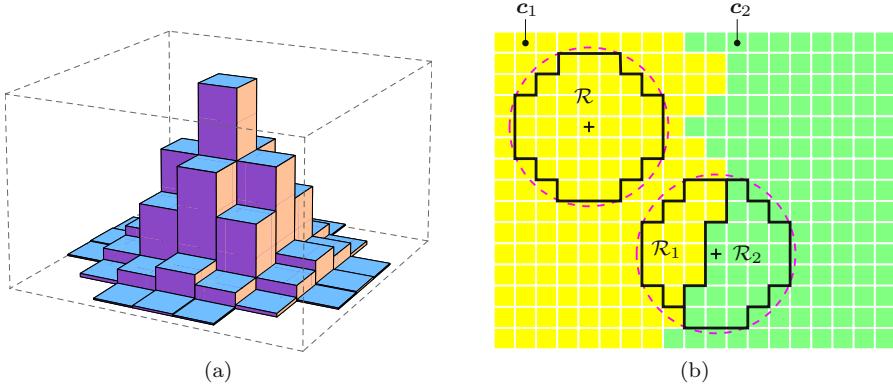


Figure 3.3 Linear smoothing filter at a color edge. Discrete filter kernel with positive-only elements and support region \mathcal{R} (a). Filter kernel positioned over a region of constant color c_1 and over a color step edge c_1/c_2 , respectively (b). If the (normalized) filter kernel of extent \mathcal{R} is completely embedded in a region of constant color (c_1), the result of filtering is exactly that same color. At a step edge between two colors c_1, c_2 , one part of the kernel (\mathcal{R}_1) covers pixels of color c_1 and the remaining part (\mathcal{R}_2) covers pixels of color c_2 . In this case, the result is a *linear mixture* of the colors c_1, c_2 , as illustrated in Fig. 3.2 (b).

is located on the straight line segment connecting c_1 and c_2 (Fig. 3.2 (b)).

Response to a color step edge

Assume, as a special case, that the original RGB image \mathbf{I} contains a *step edge* separating two regions of constant colors c_1 and c_2 , respectively, as illustrated in Fig. 3.3 (b). If the normalized smoothing kernel H is placed at some position

(u, v) , where it is fully supported by pixels of identical color \mathbf{c}_1 , the (trivial) response of the filter is

$$\bar{I}(u, v) = \sum_{(i,j) \in \mathcal{R}_H} \mathbf{c}_1 \cdot H(i, j) = \mathbf{c}_1 \cdot \sum_{(i,j) \in \mathcal{R}_H} H(i, j) = \mathbf{c}_1 \cdot 1 = \mathbf{c}_1. \quad (3.6)$$

Thus the result at this position is the original color \mathbf{c}_1 . Alternatively, if the filter kernel is placed at some position *on* a color edge (between two colors $\mathbf{c}_1, \mathbf{c}_2$, see again [Fig. 3.3 \(b\)](#)), a subset of its coefficients (\mathcal{R}_1) is supported by pixels of color \mathbf{c}_1 , while the other coefficients (\mathcal{R}_2) overlap with pixels of color \mathbf{c}_2 . Since $\mathcal{R}_1 \cup \mathcal{R}_2 = \mathcal{R}$ and the kernel is normalized, the resulting color is

$$\begin{aligned} \bar{\mathbf{c}} &= \sum_{(i,j) \in \mathcal{R}_1} \mathbf{c}_1 \cdot H(i, j) + \sum_{(i,j) \in \mathcal{R}_2} \mathbf{c}_2 \cdot H(i, j) \\ &= \mathbf{c}_1 \cdot \underbrace{\sum_{(i,j) \in \mathcal{R}_1} H(i, j)}_{s_1} + \mathbf{c}_2 \cdot \underbrace{\sum_{(i,j) \in \mathcal{R}_2} H(i, j)}_{s_2} \\ &= \mathbf{c}_1 \cdot s_1 + \mathbf{c}_2 \cdot s_2 = \mathbf{c}_1 \cdot (1 - s_2) + \mathbf{c}_2 \cdot s_2 \\ &= \mathbf{c}_1 + s_2 \cdot (\mathbf{c}_2 - \mathbf{c}_1), \end{aligned} \quad (3.7)$$

for some $s_2 \in [0, 1]$. As we see, the resulting color coordinate $\bar{\mathbf{c}}$ lies on the straight line segment connecting the original colors \mathbf{c}_1 and \mathbf{c}_2 in the respective color space. Thus, at a step edge between two colors $\mathbf{c}_1, \mathbf{c}_2$, the intermediate colors produced by a (normalized) smoothing filter are located on the straight line between the two original color coordinates. Note that this relationship between linear filtering and linear color mixtures is independent of the particular color space in which the operation is performed.

3.1.2 Color space considerations

Obviously, a linear filter implies certain “metric” properties of the underlying color space. The question is, does it matter which color space is used for linear filtering and do the results differ significantly? Which is a good color space to use and is there maybe an “ideal” color space for linear filtering? Is it acceptable to filter in the non-linear sRGB space or should the operation be performed in linear RGB, CIEXYZ, or even in a perceptually uniform color space, such as CIELAB? [Figure 3.4](#) illustrates how a linear filter operation could be performed in a different color space instead, i. e., in CIELAB.

In the following, we investigate the nature of scalar, linear filters for color images and derive requirements for the results, based on colorimetric and perceptual arguments. Four popular color spaces are evaluated: non-linear sRGB, linear RGB, CIELUV, and CIELAB (see also Vol. 2, Ch. 6 [21]). Note that

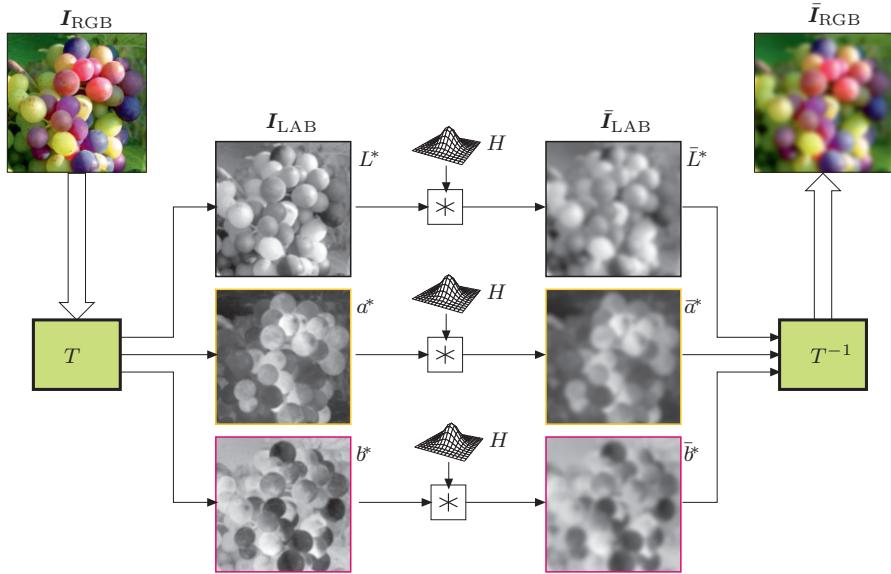


Figure 3.4 Linear filter operation performed in a different color space. The original RGB image I_{RGB} is first transformed to CIELAB (by T), where the linear filter is applied separately to the three channels L^* , a^* , b^* . The filtered RGB image \bar{I}_{RGB} is obtained by transforming back from CIELAB to RGB (by T^{-1}).

we do not consider cylindrical color spaces (such as HSV or HLS) here because they lack a precise photometric specification.

Different color spaces are usually related to each other by some well-defined non-linear transformation as, for example, between RGB and sRGB.³ Obviously, the colors obtained by linear interpolation in some color space A will not map to linearly interpolated colors in space B , if the transformation between A and B is non-linear (see Fig. 3.5). Thus, if the color mixture is assumed to be correct in one color space, it cannot be correct in the other space, unless the mapping between the color spaces is linear (such as between RGB and CIEXYZ).

The example in Fig. 3.6 (a) illustrates the results of linear interpolation between pure red and green, calculated in four different color spaces: sRGB, linear RGB, CIELUV, and CIELAB. The graphs show the resulting component values in linear RGB space, as well as the intermediate luminance (Y) values. Notice, in particular, how strongly the results from filtering in sRGB deviate from the other color spaces (component as well as luminance values). Similarly, Fig. 3.7 shows the transient colors between yellow and blue produced by

³ See Appendix E.1 (p. 341) for the precise transformations from XYZ to RGB and sRGB.

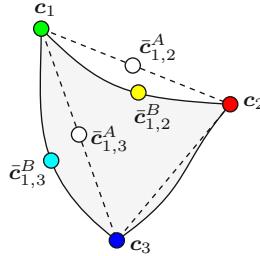


Figure 3.5 Linear interpolation in different color spaces. If two (hypothetical) color spaces A and B are related by some non-linear transformation, linear interpolation performed in either color space leads to different results. For example, interpolation between colors c_1 , c_2 in color space A (dashed lines) gives $\bar{c}_{1,2}^A$, while the result of interpolation in space B (solid lines) is $\bar{c}_{1,2}^B$.

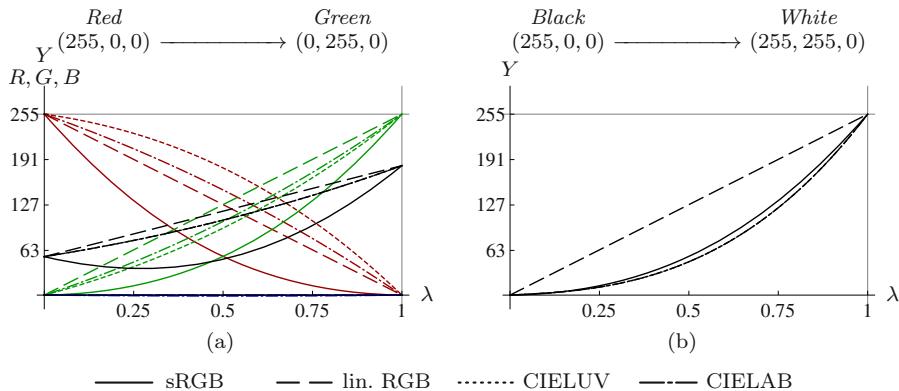


Figure 3.6 Results of linear interpolation between pairs of colors in different color spaces. Linear R, G, B components (colored) and corresponding luminance (Y) values are shown for the transition between pure red and green (a). Luminance (Y) values for the transition between black and white (b) show that CIELUV and CIELAB are identical and similar to sRGB, while the transition in linear RGB (by definition of luminance) is a straight line.

linear interpolation in different color spaces. For comparison, the results are all mapped to linear RGB as a reference space. Note the large differences not only between sRGB and linear RGB, but also between the two CIE color spaces. Since the results are shown in linear RGB space, the interpolation for this color space is necessarily a straight line.

Figure 3.8 shows the results of filtering a synthetic test image with a normalized Gaussian kernel of radius $\sigma = 3$. Included are the grayscale images obtained from the resulting luminance (Y) values (see Sec. 3.1.2). The colors inside the horizontal bar at the center of the test image are desaturated but have the same luminance as their neighboring colors. The bar should thus not

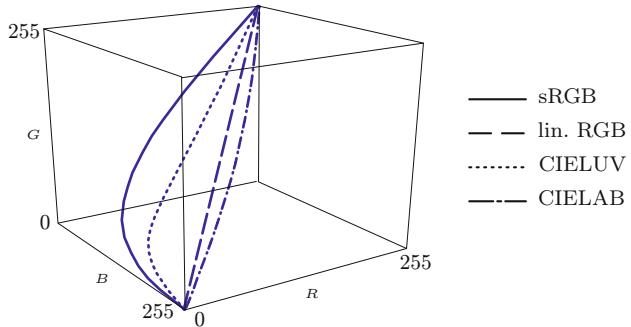


Figure 3.7 Transient colors produced by linear interpolation between *yellow* and *blue*, performed in different color spaces. The 3D plot shows the resulting colors in linear RGB space.

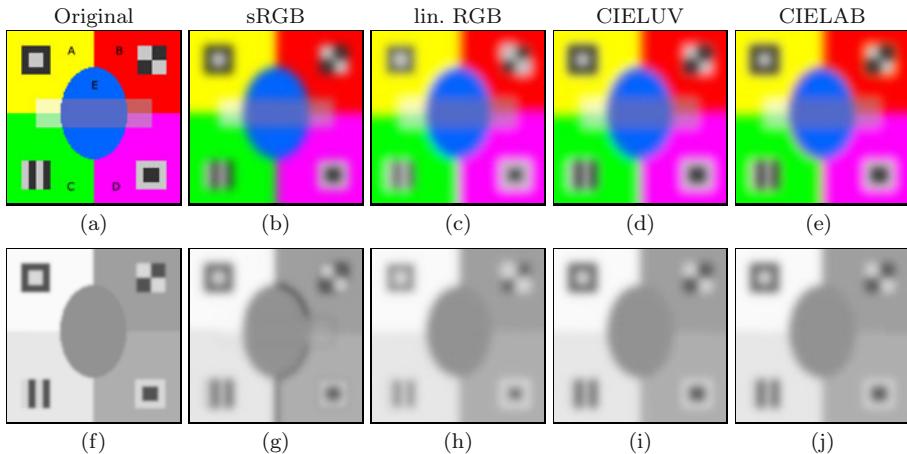


Figure 3.8 Gaussian smoothing performed in different color spaces. Synthetic color image (a) and corresponding luminance image (f); Gaussian smoothing filter applied in different color spaces: sRGB (b), linear RGB (c), CIELUV (d) and CIELAB (e). The bottom row (f–j) shows the corresponding luminance (Y) images. Note the dark bands in the sRGB result (b), particularly along the color boundaries between regions B-E, C-D, and D-E, which stand out clearly in the corresponding luminance image (g). Filtering in linear RGB space gives good results between highly saturated colors, but too high luminance in unsaturated regions, which is apparent around the gray markers (c, h). Results with CIELUV (d, i) and CIELAB color spaces (e, j) are most consistent as far as the preservation of luminance is concerned.

show in the filtered grayscale images, unless incorrect luminance values are produced. This happens in the sRGB results (Fig. 3.8 (b, g)), which also show strong dark bands, particularly along the red-blue, magenta-blue, and magenta-green edges. In these color transitions, the intermediate luminance drops below the luminance of both contributing colors, an effect that could have been predicted from the unusual luminance curve for the red-green transition in sRGB (see Fig. 3.6 (a)).

Preserving brightness

At this point we cannot say much in general about the mathematical, physical or perceptual correctness of linear filtering in any of these color spaces. However, the preservation of local brightness appears to be a minimal requirement. As a fundamental requirement we state that the local brightness β of the filtered color image should be identical to the result of filtering only the (scalar) brightness of the original image with the same kernel H . Thus, if $\beta(\mathbf{I})$ denotes the brightness of the original color image and $\beta(\mathbf{I} * H)$ is the brightness of the filtered image, we want to make sure that

$$\beta(\mathbf{I} * H) \equiv \beta(\mathbf{I}) * H. \quad (3.8)$$

If this condition holds, the brightness $\beta(\mathbf{I} * H)$ of the filtered color image must at least be bounded by the brightness values of the original colors in the filter region $\mathcal{R}_{u,v}$, that is,

$$\min_{(i,j) \in \mathcal{R}_{u,v}} \beta(\mathbf{I}(i,j)) \leq [\beta(\mathbf{I}) * H](u,v) \leq \max_{(i,j) \in \mathcal{R}_{u,v}} \beta(\mathbf{I}(i,j)), \quad (3.9)$$

because $\beta(\mathbf{I}) * H$ is again a convex linear mixture of the original (scalar) brightness values $\beta(\mathbf{I})$. This implies that none of the resulting intermediate colors produced by the filter may be brighter than the brightest contributing color, and none may be darker than the darkest contributing color. Consequently, when a smoothing filter kernel moves continuously across a step edge separating two colors $\mathbf{c}_1, \mathbf{c}_2$, the resulting intermediate brightness values must increase (or decrease) *monotonically* between the brightness of \mathbf{c}_1 and \mathbf{c}_2 .

Luminance, luma or lightness?

Whether the condition in (3.8) is satisfied for a particular color space depends on the definition of the color brightness function $\beta()$. It holds if brightness is defined as a linear combination of the component values or is an independent component in the working color space. For example, the brightness of a linear RGB color sample $\mathbf{c} = (R, G, B)$ is usually calculated as the weighted sum of the component values in the familiar form

$$Y(\mathbf{c}) = \mathbf{w} \cdot \mathbf{c} = w_R \cdot R + w_G \cdot G + w_B \cdot B, \quad (3.10)$$

with weights $w_k > 0$, and $\sum w_k = 1$. The same can be done with non-linear sRGB colors $\mathbf{c}' = (R', G', B')$ using a different set of weights w'_k , that is,

$$Y'(\mathbf{c}') = \mathbf{w}' \cdot \mathbf{c}' = w'_R \cdot R' + w'_G \cdot G' + w'_B \cdot B'. \quad (3.11)$$

The quantity Y in Eqn. (3.10) is called *luminance*, while Y' in Eqn. (3.11) is referred to as *luma*.⁴ Note that the luminance Y is the same as the corresponding coordinate in CIEXYZ space and is thus relevant in colorimetric terms. Since $Y(\mathbf{c})$ and $Y'(\mathbf{c}')$ are both linear, convex combinations of the underlying component values, it is easy to show that

$$Y(\mathbf{I} * H) \equiv Y(\mathbf{I}) * H \quad \text{and} \quad Y'(\mathbf{I}' * H) \equiv Y'(\mathbf{I}') * H, \quad (3.12)$$

as required by Eqn. (3.8). In other words, *luminance* (Y) is preserved by filtering in (linear) RGB space, while *luma* (Y') is preserved by linear filtering in (non-linear) sRGB space. However, luminance is *not* preserved when applying a linear filter to a non-linear sRGB image \mathbf{I}' and vice versa, i. e.,

$$Y(\mathbf{I}' * H) \not\equiv Y(\mathbf{I}') * H \quad \text{and} \quad Y'(\mathbf{I} * H) \not\equiv Y'(\mathbf{I}) * H. \quad (3.13)$$

In the colorimetric color models CIELUV and CIELAB, the brightness of a color is directly expressed by the value of its L^* (lightness) component, roughly defined as⁵

$$L^* \approx Y^{1/2.38}, \quad (3.14)$$

where Y is the corresponding luminance (CIE-XYZ coordinate), as defined in Eqn. (3.10). Since L^* is a separate component in these color spaces, linear filtering an image $\mathbf{I}_{\text{CIE}} = (L^*, \cdot, \cdot)$ in either CIELUV or CIELAB space naturally preserves this quantity, that is,

$$L(\mathbf{I}_{\text{CIE}} * H) \equiv L(\mathbf{I}_{\text{CIE}}) * H. \quad (3.15)$$

As one might expect, lightness (L) is *not* preserved if the filter operation is performed in linear RGB or sRGB space.

[Figure 3.9](#) shows the transient colors and luminance values obtained by linear interpolation (filtering) in different color spaces. In summary, filtering in any of the described color spaces preserves a particular brightness quantity: *luma* (Y') for sRGB, *luminance* (Y) for linear RGB, and *lightness* (L) for CIELUV and CIELAB.

⁴ These terms are often confused or used incorrectly (see [107] for a detailed discussion). Surprisingly, ITU Rec. 709 [58] specifies the same weights $\mathbf{w} = \mathbf{w}' = (0.2126, 0.7152, 0.0722)$ to be applied for both luminance and luma calculations.

⁵ See Appendix E.2 (p. 342) for the precise transformations from XYZ to CIELUV and CIELAB.

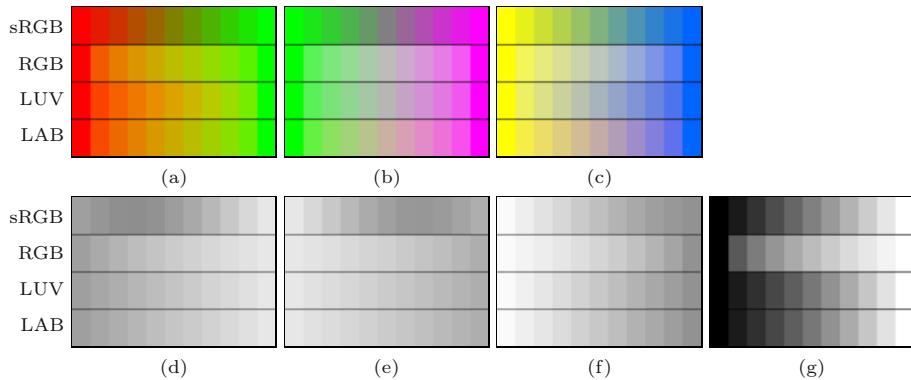


Figure 3.9 Transient colors and luminance values obtained by linear interpolation between high-chroma colors in different color spaces. Color transitions between *red/green* (a), *red/magenta* (b), and *yellow/blue* (c). Corresponding luminance (Y) values (d–f), *black/white* transition (g). Note the non-monotonic luminance curves for sRGB in (d, e). Colors may not appear authentic in print.

Perceptually uniform color spaces: CIELUV and CIELAB

Perceptually uniform color spaces are designed such that the difference between any two colors is perceived proportional to their distance in the color space. This requirement implies that the resulting color space is in some sense “metric” with respect to human perception and color differences can be easily measured with the Euclidean distance. Consequently, any intermediate colors produced by linear color interpolation in a uniform color space should also be visually “correct”. Perceptually uniform color spaces thus should be best suited for performing linear filter operations.

The construction of a perceptually uniform color space is difficult, given that the transformations (usually to CIEXYZ) should be simple and easily invertible. The original CIEXYZ color space, although based on psychophysical measurements, is perceptually *non-uniform*, and so is any color space that maps linearly to CIEXYZ (including RGB). Similarly, sRGB is partially uniform along the RGB color axes, but highly non-uniform in other parts of its gamut volume. The CIELUV and CIELAB color spaces are specified by simple nonlinear transformations to and from CIEXYZ to obtain perceptual uniformity.⁶ Figure 3.10 shows the (uniformly tessellated) sRGB color cube transformed to the CIEXYZ, CIELUV, and CIELAB spaces. While the lightness component (L^* , vertical axis) is defined identically in CIELUV and CIELAB,

⁶ CIELAB was primarily designed for applications dealing with reflective and transmissive products, such as the graphic arts industry, while CIELUV aims at emissive color applications, including video, television broadcasting, and computer graphics.

their chroma components are defined differently.⁷ In CIELAB, the a^*, b^* components are color differences that are also gamma-mapped, i.e., non-linearly related to differences in XYZ space. This is not the case in CIELUV, where the u^*, v^* components are obtained by a central projection from X - and Z -coordinates, respectively. Although this is also a non-linear transformation, straight lines in CIELUV map to straight lines in the CIE chromaticity diagram. Thus the intermediate chromaticities produced by a linear interpolation in CIELUV are located on the connecting straight lines in the xy -diagram, at least for constant luminance Y .

While CIELUV and CIELAB certainly show better uniformity than CIE-XYZ, both color spaces are far from being really uniform, even for small color differences [147]. Uniformity over large color distances can thus not be expected. Also, the results from CIELUV and CIELAB differ quite strongly, as the example in Fig. 3.7 shows for the yellow-blue interpolation. Thus, even for CIELUV and CIELAB we cannot say that the perceptually “best” intermediate colors are found on a straight line between the original color points, as would be implied by a linear filter.

Out-of-gamut colors

If we apply a linear filter in RGB or sRGB space, the resulting intermediate colors are always valid RGB colors again and contained in the original RGB gamut volume. Since the gamut volume of both color spaces is convex (see Fig. 3.10(a, b)), the convex hull of any subset of colors is also contained in the original gamut and thus no out-of-gamut colors can result from a linear interpolation in RGB or sRGB.

However, transformed to CIELUV or CIELAB, the set of possible RGB or sRGB colors forms a non-convex shape, as shown in Fig. 3.10(c-f). A linear interpolation in any of these spaces may result in new colors which, when mapped back to RGB or sRGB, are not contained in the gamut of the original color space. Particularly critical (in both CIELUV and CIELAB) are the *red-white*, *red-yellow* and *red-magenta* transitions, as well as *yellow-green* in CIELAB, where the resulting distances from the gamut surface can be quite large (see Fig. 3.11). Figure 3.12 shows the results of filtering the previous color test image in CIELUV and CIELAB, respectively, with the detected out-of-gamut colors marked white. Also note that in Fig. 3.6(a) the *blue* component for CIELAB interpolation is slightly negative and thus outside the RGB gamut.

Correcting out-of-gamut errors is a non-trivial problem that is highly relevant in the context of color management and gamut mapping. Unfortunately, the classic algorithms for gamut mapping [87] are complex and simply clip-

⁷ See Appendix E (pp. 341–345) for the precise color space definitions.

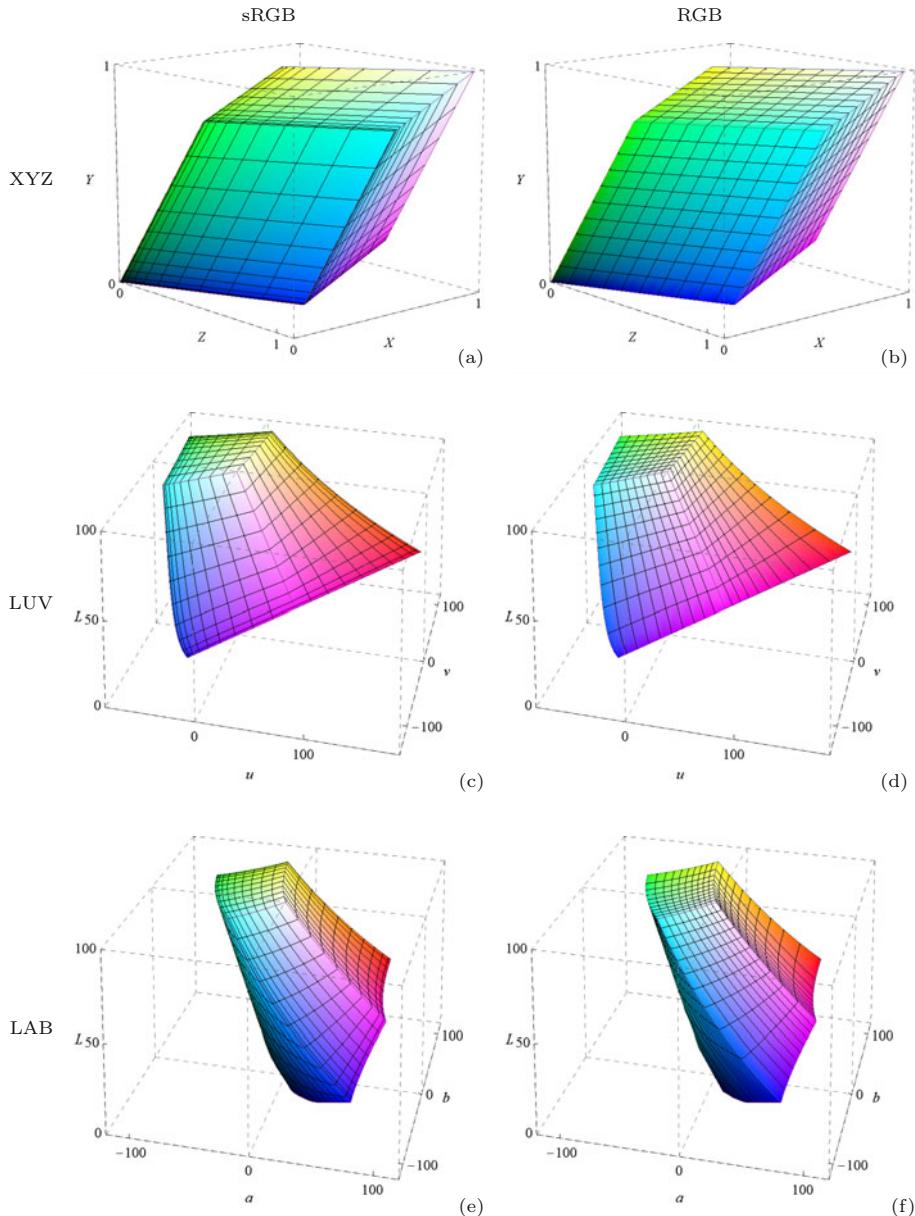


Figure 3.10 Uniformly tessellated sRGB and linear RGB color cubes mapped to CIE-XYZ (a, b), CIELUV (c, d) and CIELAB (e, f). Out-of-gamut colors may occur from linear interpolation in LUV or LAB due to the non-convexity of the corresponding volumes.

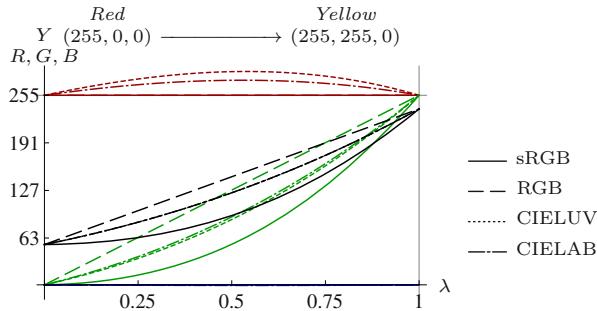


Figure 3.11 Out-of-gamut colors produced by linear interpolation between *red* and *yellow* in different color spaces. The graphs show that the red component runs significantly outside the admissible range during interpolation between red and yellow in both CIELUV and CIELAB.

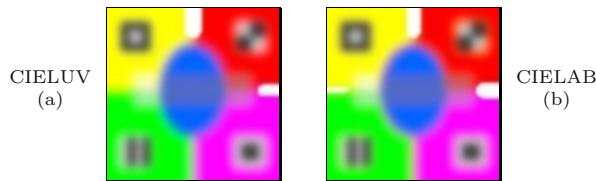


Figure 3.12 Out-of-gamut colors produced by linear filtering in CIELUV and CIELAB. Images were filtered in CIELUV (a) and CIELAB (b) and transformed back to RGB. White pixels indicate that at least one of the RGB components is outside its permissible range by more than 1%.

pling the individual out-of-gamut components to their admissible range can potentially cause color artifacts. Fortunately, out-of-gamut colors appear infrequently in real images and only arise from extremely saturated colors (e.g., around specular highlights), such that the errors caused by component clipping are usually tolerable. For high quality applications, however, the problem of handling out-of-gamut colors should not be ignored.

Implications and further reading

Applying a linear filter to the individual component channels of a color image presumes a certain “linearity” of the underlying color space. Smoothing filters implicitly perform additive linear mixing and interpolation. Despite common practice (and demonstrated by the results), there is no justification for performing a linear filter operation directly on gamma-mapped sRGB components. However, unlike one may expect, filtering in linear RGB does not yield better overall results either. In summary, both non-linear sRGB and linear RGB color spaces are unsuitable for linear filtering if perceptually accurate results are desired. Perceptually uniform color spaces, such as CIELUV and CIELAB,

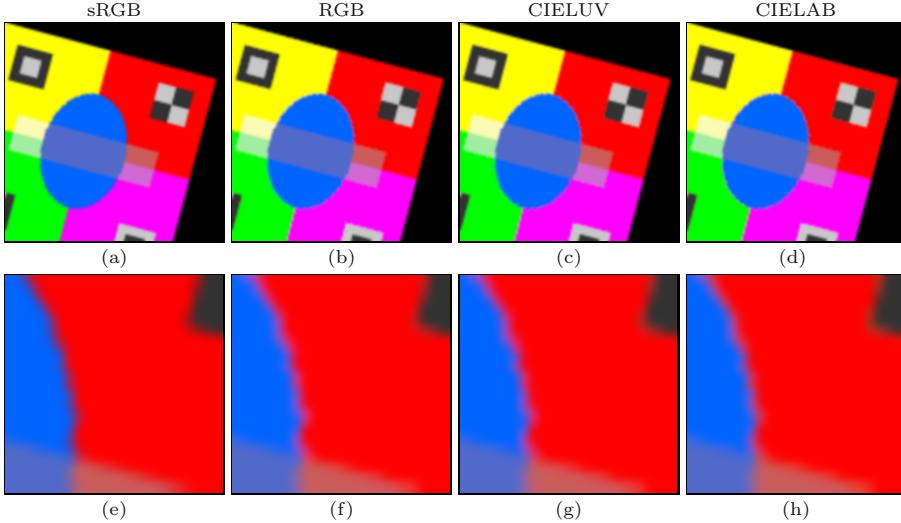


Figure 3.13 Image rotation with bilinear interpolation performed in different color spaces. Interpolation in sRGB (a), linear RGB (b), CIELUV (c) and CIELAB (d). Again we can see that performing the operation in sRGB introduces intermediate colors that are generally too dark. Enlarged detail in (e–h).

are good choices for linear filtering because of their metric properties, with CIELUV being perhaps slightly superior when it comes to interpolation over large color distances. When using CIELUV or CIELAB as intermediate color spaces for filtering RGB images, one must consider that out-of-gamut colors may be produced that must be handled properly. Thus none of the existing standard color spaces is universally suited or even “ideal” with respect to linear filtering.

The proper choice of the working color space is relevant not only to smoothing filters, but also to other types of filters, such as linear interpolation filters for geometric image transformations (see Fig. 3.13), decimation filters used in multi-scale techniques, and also non-linear filters that involve averaging colors or calculation of color distances, such as the vector median filter (see Sec. 3.2.2). While complex color space transformations in the context of filtering (e.g., sRGB \leftrightarrow CIELUV) are usually avoided for performance reasons, they should certainly be considered when high-quality results are important.

Although the issues related to color mixtures and interpolation have been investigated for some time (see, e.g., [76, 147]), their relevance to image filtering has not received much attention in the literature. Most image processing tools (including commercial software) apply linear filters directly to color images, without proper linearization or color space conversion. Lindbloom [76] was among the first to describe the problem of accurate color reproduction,

particularly in the context of computer graphics and photo-realistic imagery. He also emphasized the relevance of perceptual uniformity for color processing and recommended the use of CIELUV as a suitable (albeit not perfect) processing space. Tomasi and Manduchi [126] suggested the use of the Euclidean distance in CIELAB space as “most natural” for bilateral filtering applied to color images (see also Sec. 5.2) and similar arguments are put forth in [55]. De Weijer [134] notes that the additional chromaticities introduced by linear smoothing are “visually unacceptable” and argues for the use of non-linear operators as an alternative. Lukac et al. [83] mention “certain inaccuracies” and color artifacts related to the application of scalar filters and discuss the issue of choosing a proper distance metric for vector-based filters. The practical use of alternative color spaces for image filtering is described in [69, Ch. 5].

3.2 Non-linear color filters

In many practical image processing applications, linear filters are of limited use and non-linear filters, such as the median filter, are applied instead.⁸ In particular, for effective noise removal, non-linear filters are usually the better choice. However, as with linear filters, the techniques originally developed for scalar (grayscale) images do not transfer seamlessly to vector-based color data. One reason is that, unlike in scalar data, no natural ordering relation exists for multi-dimensional data. As a consequence, non-linear filters of the scalar type are often applied separately to the individual color channels, and again one must be cautious about the intermediate colors being introduced by these types of filters.

In the remainder of this section we describe the application of the classic (scalar) median filter to color images, a vector-based version of the median filter, and edge-preserving smoothing filters designed for color images. Additional filters for color images are presented in Chapter 5.

3.2.1 Scalar median filter

Applying a median filter with support region \mathcal{R} (for example, a disk-shaped region) at some image position (u, v) means to select one pixel value that is the most representative of the pixels in \mathcal{R} to replace the current center pixel (*hot spot*). In case of a median filter, the statistical *median* of the pixels in \mathcal{R} is taken as that representative. Since we always select the value of one of the existing image pixels, the median filter does not introduce any new pixel values that were not contained in the original image.

⁸ See also Vol. 1, Sec. 5.4 [20].

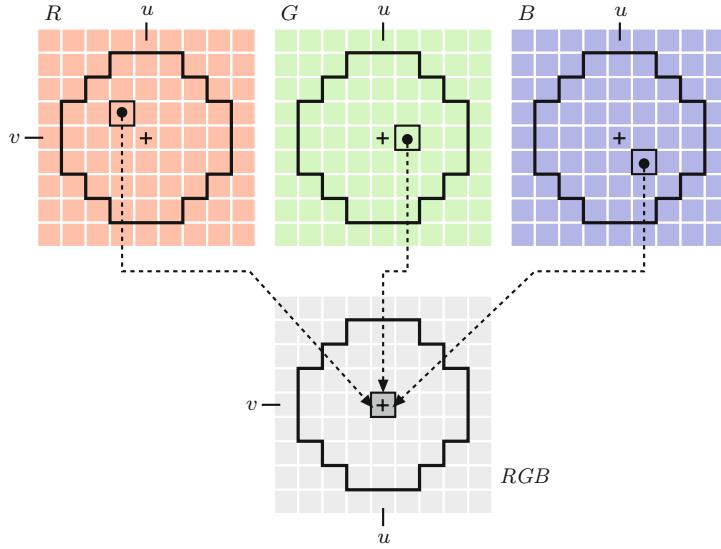


Figure 3.14 Scalar median filter applied separately to color channels. With the filter region \mathcal{R} centered at some point (u, v) , the median pixel value is generally found at different locations in the R, G, B channels of the original image. The components of the resulting RGB color vector are collected from spatially separated pixels. It thus may not match any of the colors in the original image.

If a median filter is applied independently to the components of a color image, each channel is treated as a scalar image, like a single grayscale image. In this case, with the support region \mathcal{R} centered at some point (u, v) , the median for each color channel will typically originate from a *different* spatial position in \mathcal{R} , as illustrated in Fig. 3.14. Thus the components of the resulting color vector are generally collected from more than one pixel in \mathcal{R} , therefore the color placed in the filtered image may not match any of the original colors and new colors may be generated that were not contained in the original image.

Despite its obvious deficiencies, the scalar (monochromatic) median filter is used in many popular image processing environments (including Photoshop and ImageJ) as the standard median filter for color images.

3.2.2 Vector median filter

The scalar median filter is based on the concept of *rank ordering*, i. e., it assumes that the underlying data can be ordered and sorted. However, no such natural ordering exists for data elements that are vectors. Although vectors can be sorted in many different ways, for example by length or lexicographically along their dimensions, it is usually impossible to define a useful greater-than relation

between any pair of vectors.

One can show, however, that the median of a sequence of n scalar values $P = (p_1, \dots, p_n)$ can also be defined as the value p_m selected from P , such that

$$\sum_{i=1}^n |p_m - p_i| \leq \sum_{i=1}^n |p_j - p_i| \quad (3.16)$$

holds for any $p_j \in P$. In other words, the median value $p_m = \text{median}(P)$ is the one for which the sum of the differences to *all other* elements in the sequence P is the smallest.

With this definition, the concept of the median can be easily extended from the scalar situation to the case of multi-dimensional data. Given a sequence of vector-valued samples $\mathbf{P} = (\mathbf{p}_1, \dots, \mathbf{p}_n)$, with $\mathbf{p}_i \in \mathbb{R}^K$, we define the median element \mathbf{p}_m to satisfy

$$\sum_{i=1}^n \|\mathbf{p}_m - \mathbf{p}_i\| \leq \sum_{i=1}^n \|\mathbf{p}_j - \mathbf{p}_i\|, \quad (3.17)$$

for every possible $\mathbf{p}_j \in \mathbf{P}$. This is analogous to Eqn. (3.16), with the exception that the scalar difference $|\cdot|$ has been replaced by the vector norm $\|\cdot\|$ for measuring the distance between two points in the K -dimensional space.⁹ We call

$$D_L(\mathbf{p}, \mathbf{P}) = \sum_{\mathbf{p}_i \in \mathbf{P}} \|\mathbf{p} - \mathbf{p}_i\|_L \quad (3.18)$$

the “aggregate distance” of the sample vector \mathbf{p} with respect to all samples \mathbf{p}_i in \mathbf{P} under the distance norm L. Common choices for the distance norm are the L_1 , L_2 and L_∞ norms, that is,

$$\begin{aligned} L_1: \quad \|\mathbf{p} - \mathbf{q}\|_1 &= \sum_{k=1}^K |p_k - q_k|, \\ L_2: \quad \|\mathbf{p} - \mathbf{q}\|_2 &= \left[\sum_{k=1}^K (p_k - q_k)^2 \right]^{1/2}, \\ L_\infty: \quad \|\mathbf{p} - \mathbf{q}\|_\infty &= \max_{1 \leq k \leq K} |p_k - q_k|. \end{aligned} \quad (3.19)$$

The vector median of the sequence \mathbf{P} can thus be defined as

$$\text{median}(\mathbf{P}) = \underset{\mathbf{p} \in \mathbf{P}}{\operatorname{argmin}} D_L(\mathbf{p}, \mathbf{P}), \quad (3.20)$$

⁹ K denotes the dimensionality of the samples in \mathbf{p}_i , for example, $K = 3$ for RGB color samples.

that is, the sample \mathbf{p} with the smallest aggregate distance to all other elements in \mathbf{P} .

A straight forward implementation of the vector median filter for RGB images is given in [Alg. 3.1](#). The calculation of the aggregate distance $D_L(\mathbf{p}, \mathbf{P})$ is performed by the function `AGGREGATEDISTANCE(\mathbf{p}, \mathbf{P})`. At any position (u, v) , the center pixel is replaced by the neighborhood pixel with the smallest aggregate distance D_{\min} , but only if it is smaller than the center pixel's aggregate distance D_{ctr} (line 15). Otherwise, the center pixel is left unchanged (line 17). This is to prevent the center pixel being unnecessarily changed to another color which incidentally has the same aggregate distance.

The optimal choice of the norm L for calculating the distances between color vectors in Eqn. (3.18) depends on the assumed noise distribution of the underlying signal [6]. The effects of using different norms (L_1 , L_2 , L_∞) are shown in [Fig. 3.16](#) (see [Fig. 3.15](#) for the original images). Although the results for these norms show numerical differences, they are hardly noticeable in real images (particularly in print). Unless otherwise noted, the L_1 norm is used in all subsequent examples.

Results of the scalar median filter and the vector median filter are compared in [Fig. 3.17](#). Note how new colors are introduced by the scalar filter at certain locations ([Fig. 3.17 \(a, c\)](#)), as illustrated in [Fig. 3.14](#). In contrast, the vector median filter ([Fig. 3.17 \(b, d\)](#)) can only produce colors that already exist in the original image. [Figure 3.18](#) shows the results of applying the vector median filter to real color images while varying the filter radius.

Since the vector median filter relies on measuring the distance between pairs of colors, the considerations in Section 3.1.2 regarding the metric properties of the color space do apply here as well. It is thus not uncommon to perform this filter operation not in RGB but in a perceptual uniform color space, such as CIELUV or CIELAB [63, 135, 145].

The vector median filter is computationally expensive. Calculating the aggregate distance for all sample vectors \mathbf{p}_i in \mathbf{P} requires $\mathcal{O}(n^2)$ steps, for a support region of size n . Finding the candidate neighborhood pixel with the minimum aggregate distance in \mathbf{P} can be done in $\mathcal{O}(n)$. Since n is proportional to the square of the filter radius r , the number of steps required for calculating a single image pixel is roughly $\mathcal{O}(r^4)$. While faster implementations have been proposed [6, 9, 121], calculating the vector median filter remains computationally demanding.

3.2.3 Sharpening vector median filter

Although the vector median filter is a good solution for suppressing impulse noise and additive Gaussian noise in color images, it does tend to blur or even eliminate relevant structures, such as lines and edges. The *sharpening*

Algorithm 3.1 Vector median filter for color images.

1: **VECTORMEDIANFILTER**(\mathbf{I}, r)
 Input: $\mathbf{I} = (I_R, I_G, I_B)$, a color image of size $M \times N$;
 r , filter radius ($r \geq 1$).
 Returns a new (filtered) color image of size $M \times N$.

2: $(M, N) \leftarrow \text{SIZE}(\mathbf{I})$
 3: $\mathbf{I}' \leftarrow \text{DUPLICATE}(\mathbf{I})$
 4: **for all** image coordinates $(u, v) \in M \times N$ **do**
 5: $\mathbf{p}_{\text{ctr}} \leftarrow \mathbf{I}(u, v)$ ▷ center pixel of support region
 6: $\mathbf{P} \leftarrow \text{GETSUPPORTREGION}(\mathbf{I}, u, v, r)$
 7: $d_{\text{ctr}} \leftarrow \text{AGGREGATEDISTANCE}(\mathbf{p}_{\text{ctr}}, \mathbf{P})$
 8: $d_{\min} \leftarrow \infty$
 9: **for all** $\mathbf{p} \in \mathbf{P}$ **do**
 10: $d \leftarrow \text{AGGREGATEDISTANCE}(\mathbf{p}, \mathbf{P})$
 11: **if** $d < d_{\min}$ **then**
 12: $\mathbf{p}_{\min} \leftarrow \mathbf{p}$
 13: $d_{\min} \leftarrow d$
 14: **if** $d_{\min} < d_{\text{ctr}}$ **then**
 15: $\mathbf{I}'(u, v) \leftarrow \mathbf{p}_{\min}$ ▷ modify this pixel
 16: **else**
 17: $\mathbf{I}'(u, v) \leftarrow \mathbf{I}(u, v)$ ▷ keep the original pixel value
 18: **return** \mathbf{I}' .

19: **GETSUPPORTREGION**(\mathbf{I}, u, v, r)
 Returns a vector of n pixel values $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n)$ from image \mathbf{I}
 that are inside a disk of radius r , centered at position (u, v) .

20: $\mathbf{P} \leftarrow ()$
 21: **for** $i \leftarrow \lfloor u - r \rfloor, \dots, \lceil u + r \rceil$ **do**
 22: **for** $j \leftarrow \lfloor v - r \rfloor, \dots, \lceil v + r \rceil$ **do**
 23: **if** $(u - i)^2 + (v - j)^2 \leq r^2$ **then**
 24: $\mathbf{p} \leftarrow \mathbf{I}(i, j)$
 25: $\mathbf{P} \leftarrow \mathbf{P} \cup (\mathbf{p})$
 26: **return** \mathbf{P} ▷ $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n)$

27: **AGGREGATEDISTANCE**(\mathbf{p}, \mathbf{P})
 Returns the aggregate distance $D_L(\mathbf{p}, \mathbf{P})$ of the sample vector \mathbf{p} over
 all elements $\mathbf{p}_i \in \mathbf{P}$ (see Eqn. (3.18)).

28: $d \leftarrow 0$
 29: **for all** $\mathbf{q} \in \mathbf{P}$ **do**
 30: $d \leftarrow d + \|\mathbf{p} - \mathbf{q}\|_L$ ▷ choose any distance norm L
 31: **return** d .

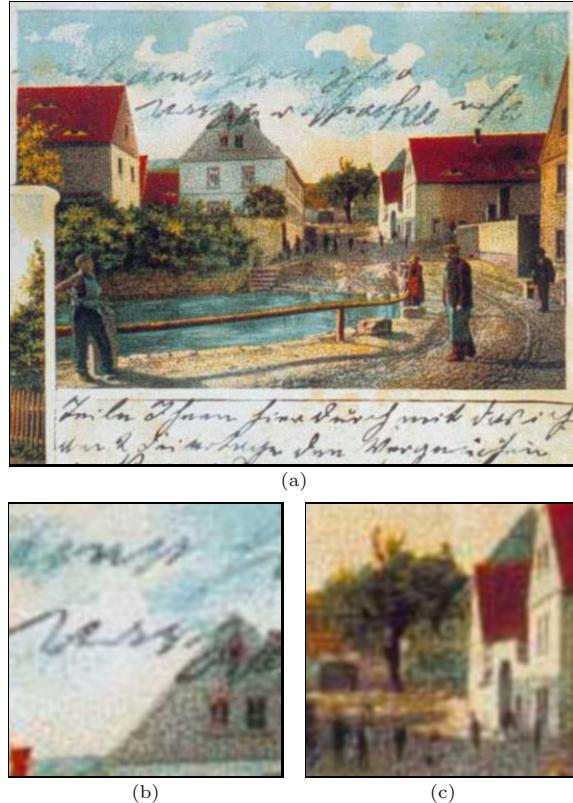


Figure 3.15 Noisy test image (a) with enlarged details (b, c), used in the following examples.

vector median filter, proposed in [82], aims at improving the edge preservation properties of the standard vector median filter described above. The key idea is not to calculate the aggregate distances against *all* other samples in the neighborhood but only against the *most similar* ones. The rationale is that the samples deviating strongly from their neighbors tend to be *outliers* (e.g., caused by nearby edges) and should be excluded from the median calculation to avoid blurring of structural details.

The operation of the sharpening vector median filter is summarized in [Alg. 3.2](#). For calculating the aggregate distance $D_L(\mathbf{p}, \mathbf{P})$ of a given sample vector \mathbf{p} (see Eqn. (3.18)), not all samples in \mathbf{P} are considered, but only those a samples that are *closest* to \mathbf{p} in the 3D color space (a being a fixed fraction of the support region size). The subsequent minimization is performed over what is called the “trimmed aggregate distance”. Thus, only a fixed number (a) of neighborhood pixels is included in the calculation of the aggregate distances. As a consequence, the sharpening vector median filter provides good

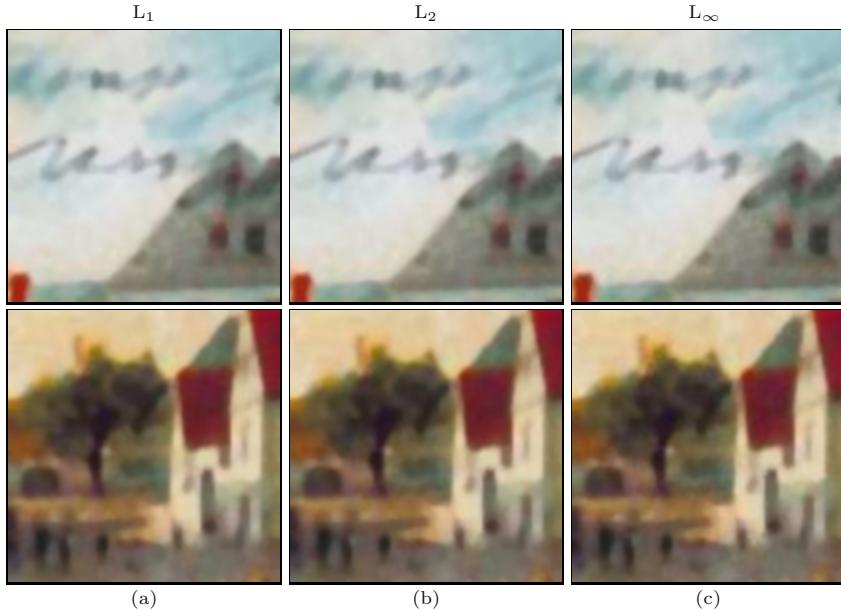


Figure 3.16 Results of vector median filtering using different color distance norms: L_1 norm (a), L_2 norm (b), L_∞ norm (c). The results for the three norms are not the same but the differences are almost imperceptible for real images. Filter radius $r = 2.0$.

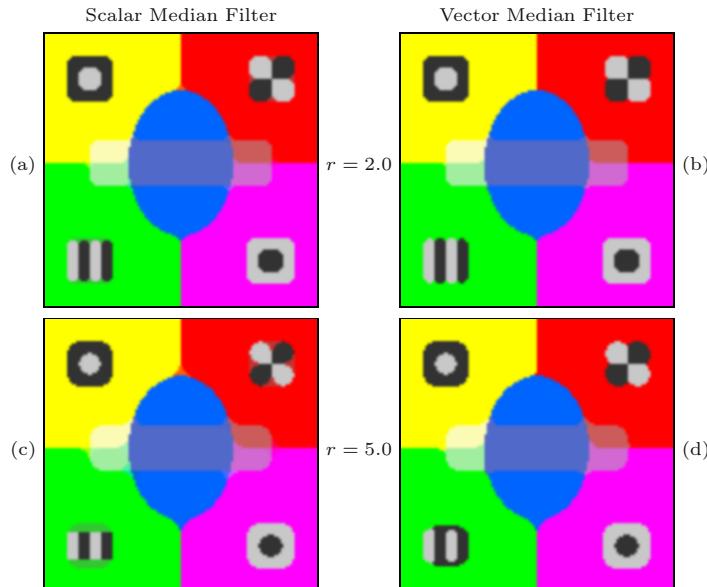


Figure 3.17 Scalar median vs. vector median filter applied to a color test image, with filter radius $r = 2.0$ (a, b) and $r = 5.0$ (c, d). Note how the scalar median filter (a, c) introduces new colors that are not contained in the original image.



Figure 3.18 Vector median filter of varying radius applied to a real color image.

noise removal while at the same time leaving edge structures intact.

Typically, the aggregate distance of \mathbf{p} to the a closest neighborhood samples is found by first calculating the distances between \mathbf{p} and all other samples in \mathbf{P} , then sorting the result, and finally adding up only the a initial elements of the sorted distances (see procedure TRIMMEDAGGREGATEDISTANCE($\mathbf{p}, \mathbf{P}, a$) in [Alg. 3.2](#)). Thus the sharpening median filter requires an additional sorting step over $n \propto r^2$ elements at each pixel, which again adds to its time complexity.

The parameter s in [Alg. 3.2](#) controls the amount of sharpening by limiting the number of neighborhood pixels $a = 1, \dots, |\mathbf{P}|$ to be included in the median calculation. For $a = 1$, the median is the value of the center pixel itself and none of the surrounding pixels are taken into account; in this case, the filter leaves the image unmodified. If $a = |\mathbf{P}| = n$, all samples in the support region \mathbf{P} are taken into account and the filter behaves like the ordinary vector median filter described in [Alg. 3.1](#). In [Alg. 3.2](#) (line 7), a is related to the sharpening parameter s by

$$a = \text{round}(n - s \cdot (n - 2)), \quad (3.21)$$

such that $a = n, \dots, 2$ for $s = 0, \dots, 1$.

The calculation of the “trimmed aggregate distance” is shown in [Alg. 3.2](#) (lines 20–29). The function TRIMMEDAGGREGATEDISTANCE($\mathbf{p}, \mathbf{P}, a$) calculates the aggregate distance for a given vector (color sample) \mathbf{p} over the a closest samples in the support region \mathbf{P} . Initially (in line 24), the n distances $D(i)$ between \mathbf{p} and all elements in \mathbf{P} are calculated, with $D(i) = \|\mathbf{p} - \mathbf{P}(i)\|_L$ (see Eqn. (3.19)). These are subsequently sorted by increasing value (line 25) and the sum of the a smallest values $D'(1), \dots, D'(a)$ (line 28) is returned.¹⁰

The effects of varying the sharpening parameter s (which controls a , the number of included neighborhood pixels, as given in Eqn. (3.21)) are shown in [Fig. 3.19](#), with a fixed filter radius $r = 2.0$ and threshold $t = 0$. For $s = 0.0$ ([Fig. 3.19 \(a\)](#)), the result is the same as that of the ordinary vector median filter (see [Fig. 3.18 \(b\)](#)).

The value of the current center pixel is only replaced by a neighboring pixel value if the corresponding minimal (trimmed) aggregate distance d_{\min} is significantly smaller than the center pixel's aggregate distance d_{ctr} . In [Alg. 3.2](#), this is controlled by the threshold t . If the condition

$$(d_{\text{ctr}} - d_{\min}) > t \cdot a \quad (3.22)$$

holds ([Alg. 3.2](#), line 15), the minimum aggregate distance d_{\min} in the neighborhood \mathbf{P} is less than the center pixel's aggregate distance d_{ctr} by more than $t \cdot a$ (where a is the number of included pixels, see Eqn. (3.21)). Only in this case

¹⁰ $D'(1)$ is zero because it is the distance between \mathbf{p} and itself.

Algorithm 3.2 Sharpening vector median filter for RGB color images (extension of Alg. 3.1). The *sharpening parameter* $s \in [0, 1]$ controls the number of most-similar neighborhood pixels included in the median calculation. For $s = 0$, all pixels in the given support region are included and no sharpening occurs; setting $s = 1$ leads to maximum sharpening. The *threshold parameter* t controls how much smaller the aggregate distance of any neighborhood pixel must be to replace the current center pixel.

```

1: SHARPENINGVECTORMEDIANFILTER( $\mathbf{I}, r, s, t$ )
   Input:  $\mathbf{I}$ , a color image of size  $M \times N$ ,  $\mathbf{I}(u, v) \in \mathbb{R}^3$ ;  $r$ , filter radius
          ( $r \geq 1$ );  $s$ , sharpening parameter ( $0 \leq s \leq 1$ );  $t$ , threshold ( $t \geq 0$ ).
          Returns a new (filtered) color image of size  $M \times N$ .
2:  $(M, N) \leftarrow \text{SIZE}(\mathbf{I})$ 
3:  $\mathbf{I}' \leftarrow \text{DUPLICATE}(\mathbf{I})$ 
4: for all image coordinates  $(u, v) \in M \times N$  do
5:    $\mathbf{P} \leftarrow \text{GETSUPPORTREGION}(\mathbf{I}, u, v, r)$                                  $\triangleright$  see Alg. 3.1
6:    $n \leftarrow |\mathbf{P}|$                                                                 $\triangleright$  size of  $\mathbf{P}$ 
7:    $a \leftarrow \text{round}(n - s \cdot (n - 2))$                                           $\triangleright a = 2, \dots, n$ 
8:    $d_{\text{ctr}} \leftarrow \text{TRIMMEDAGGREGATEDISTANCE}(\mathbf{I}(u, v), \mathbf{P}, a)$ 
9:    $d_{\min} \leftarrow \infty$ 
10:  for all  $\mathbf{p} \in \mathbf{P}$  do
11:     $d \leftarrow \text{TRIMMEDAGGREGATEDISTANCE}(\mathbf{p}, \mathbf{P}, a)$ 
12:    if  $d < d_{\min}$  then
13:       $\mathbf{p}_{\min} \leftarrow \mathbf{p}$ 
14:       $d_{\min} \leftarrow d$ 
15:    if  $(d_{\text{ctr}} - d_{\min}) > t \cdot a$  then
16:       $\mathbf{I}'(u, v) \leftarrow \mathbf{p}_{\min}$                                                $\triangleright$  replace the center pixel
17:    else
18:       $\mathbf{I}'(u, v) \leftarrow \mathbf{I}(u, v)$                                           $\triangleright$  keep the original center pixel
19:  return  $\mathbf{I}'$ .
20: TRIMMEDAGGREGATEDISTANCE( $\mathbf{p}, \mathbf{P}, a$ )
   Returns the aggregate distance from  $\mathbf{p}$  to the  $a$  most similar elements
   in  $\mathbf{P} = (\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n)$ .
21:  $n \leftarrow |\mathbf{P}|$                                                                 $\triangleright$  size of  $\mathbf{P}$ 
22: Create map  $D : [1, n] \mapsto \mathbb{R}$ 
23: for  $i \leftarrow 1, \dots, n$  do
24:    $D(i) \leftarrow \|\mathbf{p} - \mathbf{P}(i)\|_L$                                           $\triangleright$  choose any distance norm L
25:    $D' \leftarrow \text{SORT}(D)$                                                   $\triangleright D'(1) \leq D'(2) \leq \dots \leq D'(n)$ 
26:    $d \leftarrow 0$ 
27:   for  $i \leftarrow 2, \dots, a$  do                                                $\triangleright D'(1) = 0$ , thus skipped
28:      $d \leftarrow d + D'(i)$ 
29:   return  $d$ .
```

is the center pixel replaced, otherwise it remains unmodified. Note that the distance limit is proportional to a and thus t really specifies the minimum “average” pixel distance; it is independent of the filter radius r and the sharpening parameter s . Results for typical values of t (in the range 0 … 10) are shown in Figs. 3.20–3.21. To illustrate the effect, the images in Fig. 3.21 only display those pixels that were *not* replaced by the filter, while all modified pixels are set to black. As one would expect, increasing the threshold t leads to fewer pixels being modified. Of course, the same thresholding scheme may also be used with the ordinary vector median filter (see Exercise 3.2).

3.3 Java implementation

Implementations of the scalar and vector median filter as well as the sharpening vector median filter are available with full Java source code at the book’s website.¹¹ The corresponding classes `ScalarMedianFilter`, `VectorMedianFilter` and `VectorMedianFilterSharpen` are based on the common super-class `GenericFilter`, which provides the abstract methods

```
float filterPixel (ImageAccessor.Gray source, int u, int v)
float[] filterPixel (ImageAccessor.Color source, int u, int v)
```

for filtering grayscale and color images, respectively.¹² Moreover, class `GenericFilter` provides the method

```
void applyTo (ImageProcessor ip),
```

which greatly simplifies the use of these filters. The following code segment demonstrates the use of the class `VectorMedianFilter` (with radius 3.0 and L₁-norm) for RGB color images in an ImageJ plugin:

```
1 import ...
2
3 public class MedianFilter_Color_Vector implements PlugInFilter {
4
5     public int setup(String arg, ImagePlus imp) {
6         return DOES_RGB;
7     }
8
9     public void run(ImageProcessor ip) {
10        double radius = 3.0;
11        DistanceNorm norm = DistanceNorm.L1;
12        VectorMedianFilter filter =
13            new VectorMedianFilter(radius, norm);
14        filter.applyTo(ip);
15    }
16 }
```

¹¹ See packages `imagingbook.filters` and `imagingbook.colorfilters`.

¹² `ImageAccessor` is contained in package `imagingbook.image`.



Figure 3.19 Sharpening vector median filter with different sharpness values s . The filter radius is $r = 2.0$ and the corresponding filter mask contains $n = 21$ pixels. At each pixel, only the $a = 21, 17, 12, 6$ closest color samples (for sharpness $s = 0.0, 0.2, 0.5, 0.8$, respectively) are considered when calculating the local vector median.



Figure 3.20 Sharpening vector median filter with different threshold values $t = 0, 2, 5, 10$. The filter radius and sharpening factor are fixed at $r = 2.0$ and $s = 0.0$, respectively.

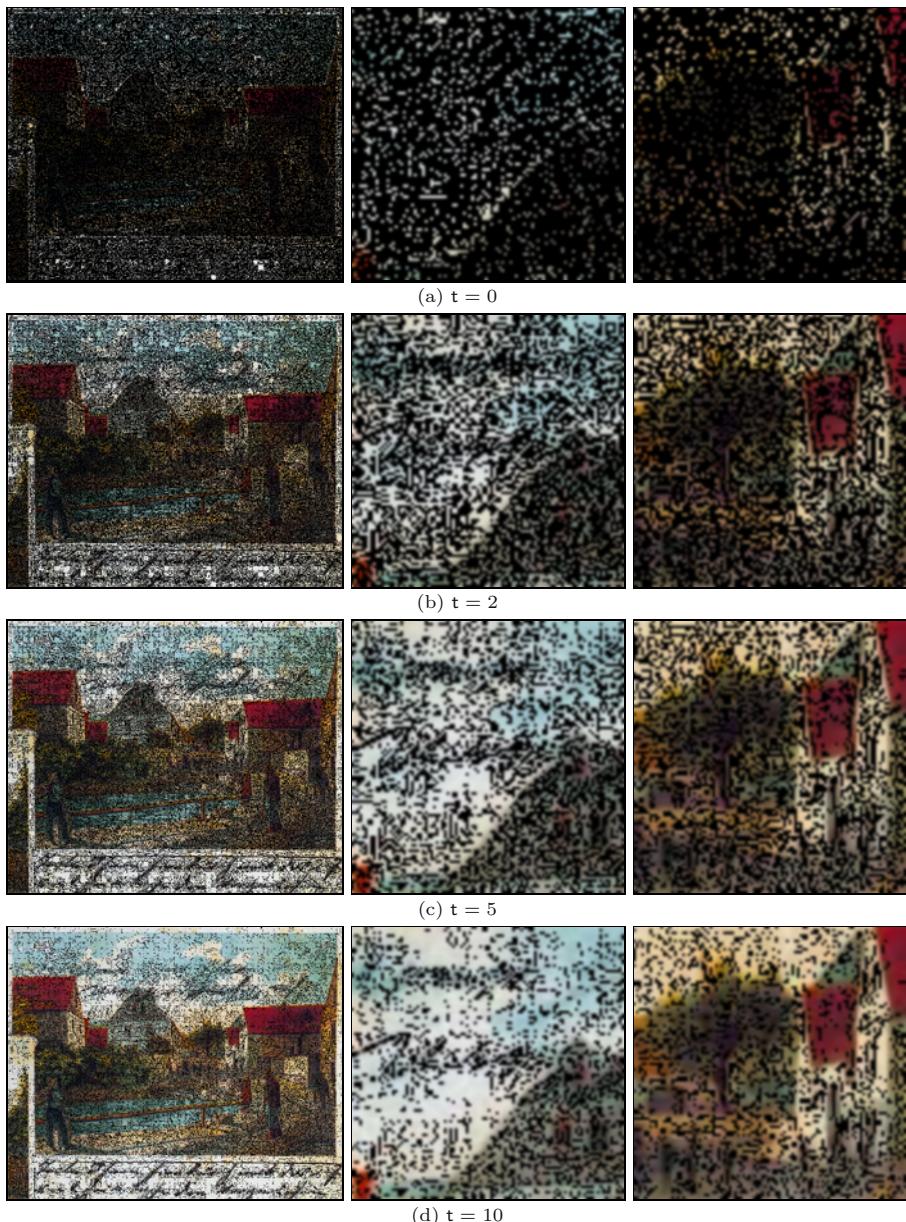


Figure 3.21 Sharpening vector median filter with different threshold values $t = 0, 2, 5, 10$ (also see Fig. 3.20). Only the unmodified pixels are shown, while all modified pixels are set to black. The filter radius and sharpening factor are fixed at $r = 2.0$ and $s = 0.0$, respectively.

A new instance of `VectorMedianFilter` is created in line 13, which is subsequently applied to the color image `ip` in line 14. For the specific filter types, the following constructors are provided:

`ScalarMedianFilter (double r)`

Creates a scalar median filter, as described in Section 3.2.1, with radius `r` (default 3.0).

`VectorMedianFilter (double r, DistanceNorm norm)`

Creates a vector median filter, as described in Section 3.2.2, with radius `r` (default 3.0) and the distance norm `norm` (L1, L2 or Lmax, default is L1).

`VectorMedianFilterSharpen (double r, DistanceNorm norm,`

`double s, double thr)`

Creates a sharpening vector median filter, as described in Section 3.2.3, with radius `r` (default 3.0), distance norm `norm` (default L1)), sharpening parameter `s` (in [0, 1], default 0.5) and threshold `thr` (default 0.0).

The listed default values pertain to the parameter-less constructors that are also available. Note that the created filter objects are generic and can be applied to either grayscale and color images without any modification.

3.4 Further reading

A good overview of different linear and non-linear filtering techniques for color images can be found in [69]. In [105, Ch. 2], the authors give a concise treatment of color image filtering, including statistical noise models, vector ordering schemes and different color similarity measures. Several variants of weighted median filters for color images and multi-channel data in general are described in [4, Sec. 2.4]. A very readable and up-to-date survey of important color issues in computer vision, such as color constancy, photometric invariance and color feature extraction, can be found in [43]. In addition to the techniques discussed in this chapter, most of the filters described in Chapter 5 are either directly applicable to color images or can be easily modified for this purpose.

3.5 Exercises

Exercise 3.1

Verify Eqn. (3.16) by showing (formally or experimentally) that the usual calculation of the scalar median (by sorting a sequence and selecting the center value) indeed gives the value with the smallest sum of differences from all other values in the same sequence.

Exercise 3.2

Modify the ordinary vector median filter described in [Alg. 3.1](#) to incorporate

a threshold t for deciding whether to modify the current center pixel or not, analogous to the approach taken in the sharpening vector median filter in [Alg. 3.2](#).

4

Edge Detection in Color Images

Edge information is essential in many image analysis and computer vision applications and thus the ability to locate and characterize edges robustly and accurately is an important task. Basic techniques for edge detection in *grayscale* images are discussed in Chapter 6 of Volume 1 [20]. *Color* images contain richer information than grayscale images and it appears natural to assume that edge detection methods based on color should outperform their monochromatic counterparts. For example, locating an edge between two image regions of different hue but similar brightness is difficult with an edge detector that only looks for changes in image intensity. In this chapter, we first look at the use of “ordinary” (i.e., monochromatic) edge detectors for color images and then discuss dedicated detectors that are specifically designed for color images.

Although the problem of color edge detection has been pursued for a long time (see [68, 152] for a good overview), most image processing textbooks do not treat this subject in much detail. One reason could be that, in practice, edge detection in color images is often accomplished by using “monochromatic” techniques on the intensity channel or the individual color components. We discuss these simple methods—which nevertheless give satisfactory results in many situations—in Section 4.1.

Unfortunately, monochromatic techniques do not extend naturally to color images and other “multi-channel” data, since edge information in the different color channels may be ambiguous or even contradictory. For example, multiple edges running in different directions may coincide at a given image location, edge gradients may cancel out or edges in different channels may be slightly displaced. In Section 4.2, we describe how local gradients can be calculated

for edge detection by treating the color image as a two-dimensional *vector field*. Finally, in Section 4.3 we show how the popular Canny edge detector, originally designed for monochromatic images, can be adapted for color images. Implementations of the discussed algorithms are described in Section 4.4, with complete source code available on the book’s website.

4.1 Monochromatic techniques

Linear filters are the basis of most edge enhancement and edge detection operators for scalar-valued grayscale images, particularly the gradient filters described in Section 6.3 of Volume 1 [20]. Again it is quite common to apply these scalar filters separately to the individual color channels of RGB images. A popular example is the Sobel operator with the filter kernels

$$H_x^S = \frac{1}{8} \cdot \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad H_y^S = \frac{1}{8} \cdot \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (4.1)$$

for the x - and y -direction, respectively. Applied to a grayscale image I , with $I_x = I * H_x^S$ and $I_y = I * H_y^S$, these filters give a reasonably good estimate of the local gradient vector,

$$\nabla I(u, v) = \begin{pmatrix} I_x(u, v) \\ I_y(u, v) \end{pmatrix}. \quad (4.2)$$

The local edge *strength* of the grayscale image is then taken as

$$E_{\text{gray}}(u, v) = \|\nabla I(u, v)\| = \sqrt{I_x^2(u, v) + I_y^2(u, v)}, \quad (4.3)$$

and the corresponding edge *orientation* is calculated as

$$\Phi(u, v) = \angle \nabla I(u, v) = \tan^{-1} \left(\frac{I_y(u, v)}{I_x(u, v)} \right). \quad (4.4)$$

The angle $\Phi(u, v)$ gives the direction of maximum intensity change on the 2D image surface at position (u, v) , which is the normal to the edge tangent.

Analogously, to apply this technique to a color image $\mathbf{I} = (I_R, I_G, I_B)$, each color plane is first filtered individually with the two gradient kernels given in Eqn. (4.1), resulting in

$$\begin{aligned} \nabla I_R &= \begin{pmatrix} I_{R,x} \\ I_{R,y} \end{pmatrix} = \begin{pmatrix} I_R * H_x^S \\ I_R * H_y^S \end{pmatrix}, \\ \nabla I_G &= \begin{pmatrix} I_{G,x} \\ I_{G,y} \end{pmatrix} = \begin{pmatrix} I_G * H_x^S \\ I_G * H_y^S \end{pmatrix}, \\ \nabla I_B &= \begin{pmatrix} I_{B,x} \\ I_{B,y} \end{pmatrix} = \begin{pmatrix} I_B * H_x^S \\ I_B * H_y^S \end{pmatrix}. \end{aligned} \quad (4.5)$$

The local edge strength in each color channel is calculated as $E_c(u, v) = \|\nabla I_c(u, v)\|$ (for $c = R, G, B$), yielding a vector

$$\mathbf{E}(u, v) = \begin{pmatrix} E_R(u, v) \\ E_G(u, v) \\ E_B(u, v) \end{pmatrix} = \begin{pmatrix} \|\nabla I_R(u, v)\| \\ \|\nabla I_G(u, v)\| \\ \|\nabla I_B(u, v)\| \end{pmatrix} \quad (4.6)$$

$$= \begin{pmatrix} [I_{R,x}^2(u, v) + I_{R,y}^2(u, v)]^{1/2} \\ [I_{G,x}^2(u, v) + I_{G,y}^2(u, v)]^{1/2} \\ [I_{B,x}^2(u, v) + I_{B,y}^2(u, v)]^{1/2} \end{pmatrix} \quad (4.7)$$

for each image position (u, v) . These vectors could be combined into a new color image $\mathbf{E} = (E_R, E_G, E_B)$, although such a “color edge image” has no particularly useful interpretation.¹ Finally, a scalar quantity of *combined edge strength* over all color planes can be obtained, for example, by calculating the Euclidean (L_2) norm of \mathbf{E} as

$$\begin{aligned} E_2(u, v) &= \|\mathbf{E}(u, v)\|_2 = [E_R^2(u, v) + E_G^2(u, v) + E_B^2(u, v)]^{1/2} \\ &= [I_{R,x}^2 + I_{R,y}^2 + I_{G,x}^2 + I_{G,y}^2 + I_{B,x}^2 + I_{B,y}^2]^{1/2} \end{aligned} \quad (4.8)$$

(coordinates (u, v) are omitted in the second line) or with the L_1 norm,

$$E_1(u, v) = \|\mathbf{E}(u, v)\|_1 = |E_R(u, v)| + |E_G(u, v)| + |E_B(u, v)|. \quad (4.9)$$

Another alternative for calculating a combined edge strength is to take the *maximum* magnitude of the RGB gradients (i. e., the L_∞ norm),

$$E_\infty(u, v) = \|\mathbf{E}(u, v)\|_\infty = \max(|E_R(u, v)|, |E_G(u, v)|, |E_B(u, v)|). \quad (4.10)$$

An example using the test image from Chapter 3 is given in Fig. 4.1. It shows the edge magnitude of the corresponding grayscale image and the combined color edge magnitude calculated with the different norms defined in Eqns. (4.8–4.10).

As far as edge *orientation* is concerned, there is no simple extension of the grayscale case. While edge orientation can easily be calculated for each individual color component (using Eqn. (4.4)), the gradients are generally different (or even contradictory) and there is no obvious way of combining them. A simple ad hoc approach is to choose, at each image position (u, v) , the gradient direction from the color channel of maximum edge strength, i. e.,

$$\varPhi_{\text{col}}(u, v) = \tan^{-1}\left(\frac{I_{m,y}(u, v)}{I_{m,x}(u, v)}\right), \quad \text{with } m = \underset{j=R,G,B}{\operatorname{argmax}} E_j(u, v). \quad (4.11)$$

¹ Such images are nevertheless produced by the “Find Edges” command in ImageJ and the filter of the same name in Photoshop (showing inverted components).

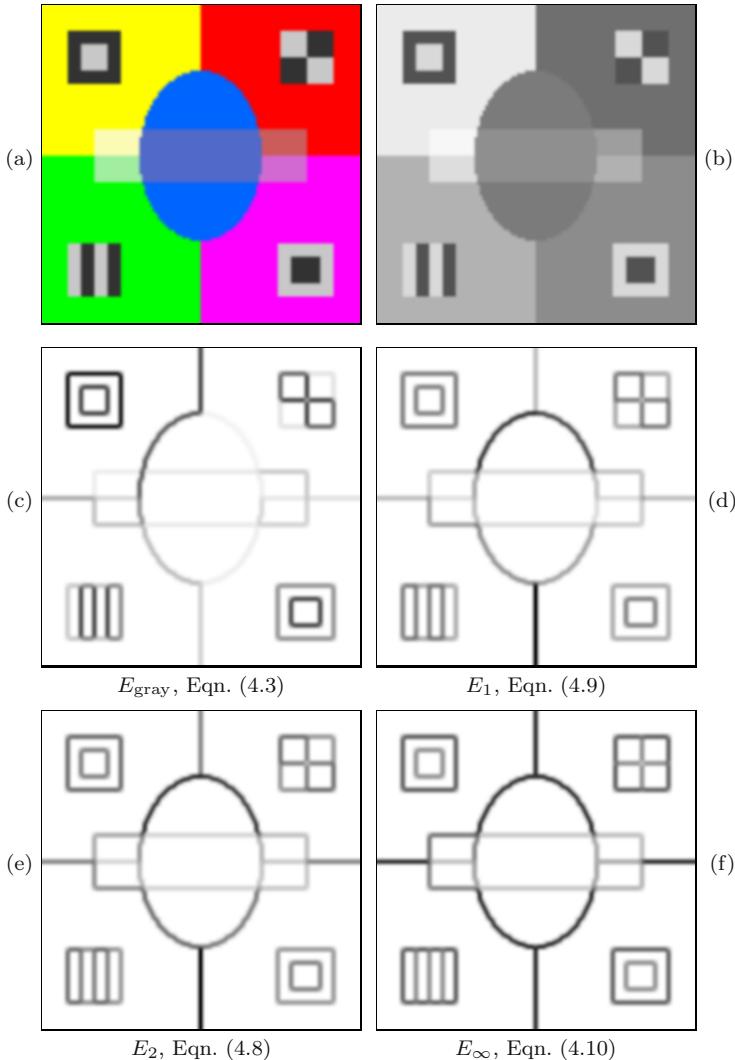


Figure 4.1 Color edge enhancement with monochromatic methods. Original color image (a) and corresponding grayscale image (b); edge magnitude from the grayscale image (c). Color edge magnitude calculated with different norms: L_1 (d), L_2 (e), and L_∞ (f). The images in (c–f) are inverted for better viewing.

This monochromatic edge detection method for color images is summarized in [Alg. 4.1](#) (see Sec. 4.4 for the corresponding Java implementation). Results obtained with this procedure are shown in [Figs. 4.2–4.5 \(c\)](#). For comparison, these figures also show the edge maps obtained by first converting the color image to a grayscale image and then applying the Sobel operator. The edge magnitude in all examples is normalized; it is shown inverted and contrast-

Algorithm 4.1 Monochromatic color edge operator. A pair of Sobel-type filter kernels (H_x^S, H_y^S) is used to estimate the local x/y gradients of each component of the RGB input image \mathbf{I} . Color edge magnitude is calculated as the L₂ norm of the color gradient vector (see Eqn. (4.8)). The procedure returns a pair of maps, holding the edge magnitude E_2 and the edge orientation Φ , respectively.

```

1: MONOCHROMATICCOLOREDGE( $\mathbf{I}$ )
   Input:  $\mathbf{I} = (I_R, I_G, I_B)$ , an RGB color image of size  $M \times N$ .
   Returns a pair of maps  $(E, \Phi)$  for edge magnitude and orientation.

2:  $H_x^S \leftarrow \frac{1}{8} \cdot \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_y^S \leftarrow \frac{1}{8} \cdot \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$   $\triangleright x/y$  gradient kernels

3:  $(M, N) \leftarrow \text{SIZE}(\mathbf{I})$ 
4: Create maps  $E, \Phi : M \times N \rightarrow \mathbb{R}$   $\triangleright$  edge magnitude/orientation
5:  $I_{R,x} \leftarrow I_R * H_x^S, \quad I_{R,y} \leftarrow I_R * H_y^S$   $\triangleright$  apply gradient filters
6:  $I_{G,x} \leftarrow I_G * H_x^S, \quad I_{G,y} \leftarrow I_G * H_y^S$ 
7:  $I_{B,x} \leftarrow I_B * H_x^S, \quad I_{B,y} \leftarrow I_B * H_y^S$ 

8: for all image coordinates  $(u, v) \in M \times N$  do
9:    $(r_x, g_x, b_x) \leftarrow (I_{R,x}(u, v), I_{G,x}(u, v), I_{B,x}(u, v))$ 
10:   $(r_y, g_y, b_y) \leftarrow (I_{R,y}(u, v), I_{G,y}(u, v), I_{B,y}(u, v))$ 
11:   $e_R^2 \leftarrow r_x^2 + r_y^2$ 
12:   $e_G^2 \leftarrow g_x^2 + g_y^2$ 
13:   $e_B^2 \leftarrow b_x^2 + b_y^2$ 
14:   $e_{\max}^2 \leftarrow e_R^2$   $\triangleright$  find maximum gradient channel
15:   $c_x \leftarrow r_x, \quad c_y \leftarrow r_y$ 
16:  if  $e_G^2 > e_{\max}^2$  then
17:     $e_{\max}^2 \leftarrow e_G^2, \quad c_x \leftarrow g_x, \quad c_y \leftarrow g_y$ 
18:  if  $e_B^2 > e_{\max}^2$  then
19:     $e_{\max}^2 \leftarrow e_B^2, \quad c_x \leftarrow b_x, \quad c_y \leftarrow b_y$ 
20:   $E(u, v) \leftarrow \sqrt{e_R^2 + e_G^2 + e_B^2}$   $\triangleright$  edge magnitude (L2 norm)
21:   $\Phi(u, v) \leftarrow \text{Arctan}(c_y, c_x)$   $\triangleright$  edge orientation
22: return  $(E, \Phi)$ .
```

enhanced to increase the visibility of low-contrast edges.

The binary images in the right columns of Figs. 4.2–4.5 were obtained by thresholding the resulting edge strength using the ISODATA method (see Sec. 2.1.3). As expected and apparent from the examples, even simple monochromatic techniques applied to color images perform better than edge detection on the corresponding grayscale images. In particular, edges between color regions of similar brightness are not detectable in this way, so using color information for edge detection is generally more powerful than relying on in-

tensity alone. Among the simple color techniques, the maximum channel edge strength E_∞ (Eqn. (4.10)) seems to give the most consistent results with the fewest edges getting lost.

However, none of the monochromatic detection techniques can be expected to work reliably under these circumstances. While the threshold for binarizing the edge magnitude could be tuned manually to give more pleasing results on specific images, it is difficult in practice to achieve consistently good results over a wide range of images. Methods for determining the optimal edge threshold dynamically, i. e., depending on the image content, have been proposed, typically based on the statistical variability of the color gradients. Additional details can be found in [44, 90, 110].

4.2 Edges in vector-valued images

In the “monochromatic” scheme described above, the edge magnitude in each color channel is calculated separately and thus no use is made of the potential coupling between color channels. Only in a subsequent step are the individual edge responses in the color channels combined, albeit in an ad hoc fashion. In other words, the color data are not treated as vectors, but merely as separate and unrelated scalar values. To obtain better insight into this problem it is helpful to treat the color image as a *vector field*, a standard construct in vector calculus [17, 122].² Assuming continuous coordinates \mathbf{x} , a three-channel RGB color image $\mathbf{I}(\mathbf{x}) = (I_R(\mathbf{x}), I_G(\mathbf{x}), I_B(\mathbf{x}))$ can be modeled as a two-dimensional vector field, which is a mapping

$$\mathbf{I} : \mathbb{R}^2 \mapsto \mathbb{R}^3, \quad (4.12)$$

i. e., a function whose coordinates $\mathbf{x} = (x, y)$ are two-dimensional and whose values are three-dimensional vectors. Similarly, a continuous grayscale image can be described as a *scalar field*, since its pixel values are only one-dimensional.

4.2.1 Multi-dimensional gradients

As noted in the previous section, the gradient of a continuous scalar image I at a specific position $\dot{\mathbf{x}} = (\dot{x}, \dot{y})$ is defined as

$$\nabla I(\dot{\mathbf{x}}) = \begin{pmatrix} \frac{\partial I}{\partial x}(\dot{\mathbf{x}}) \\ \frac{\partial I}{\partial y}(\dot{\mathbf{x}}) \end{pmatrix}, \quad (4.13)$$

² See Appendix B.6 for some general properties of vector fields.

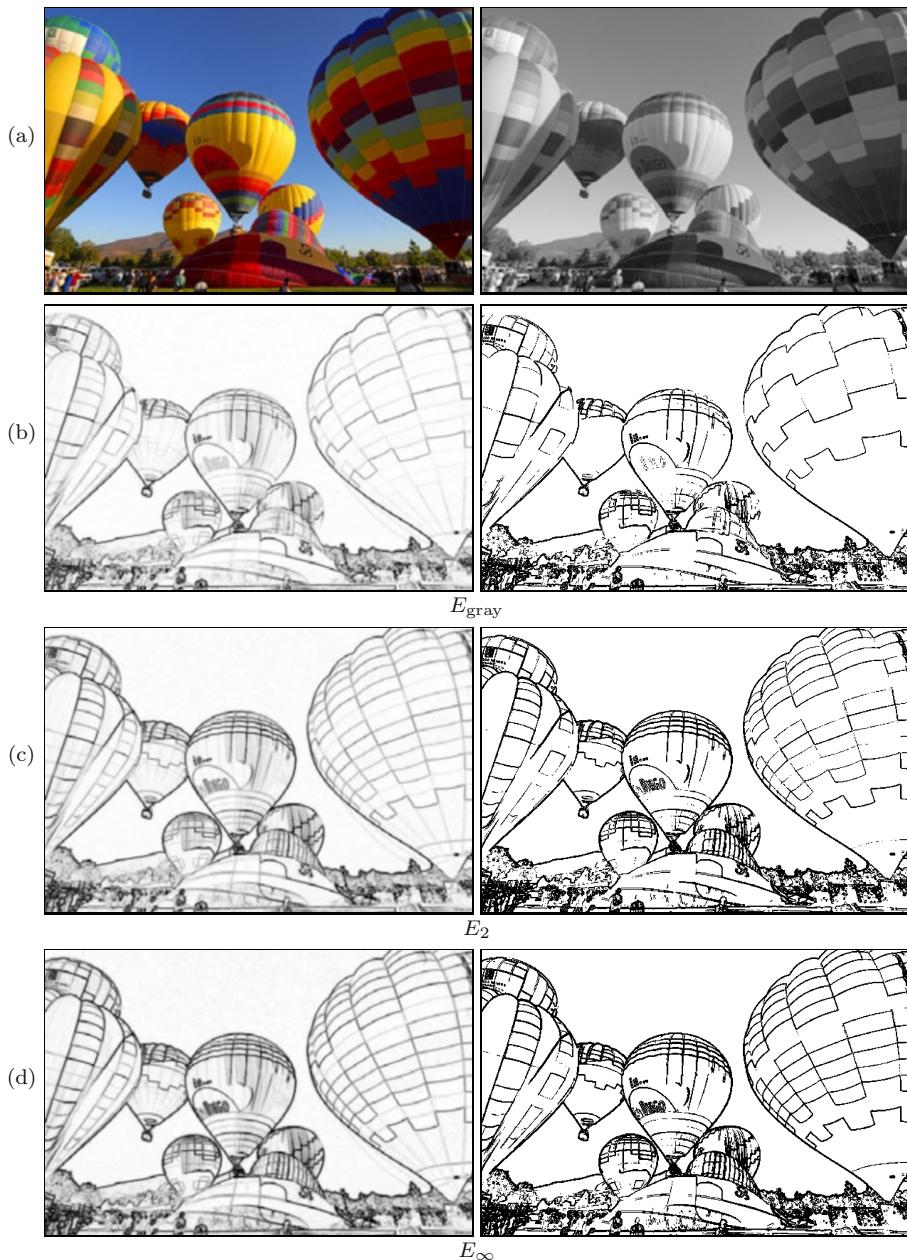


Figure 4.2 Example of color edge enhancement with monochromatic techniques (balloons image). Original color image and corresponding grayscale image (a), edge magnitude obtained from the grayscale image (b), color edge magnitude calculated with the L_2 norm (c) and the L_∞ norm (d). Binary images in the right column were obtained by thresholding the resulting edge strength.

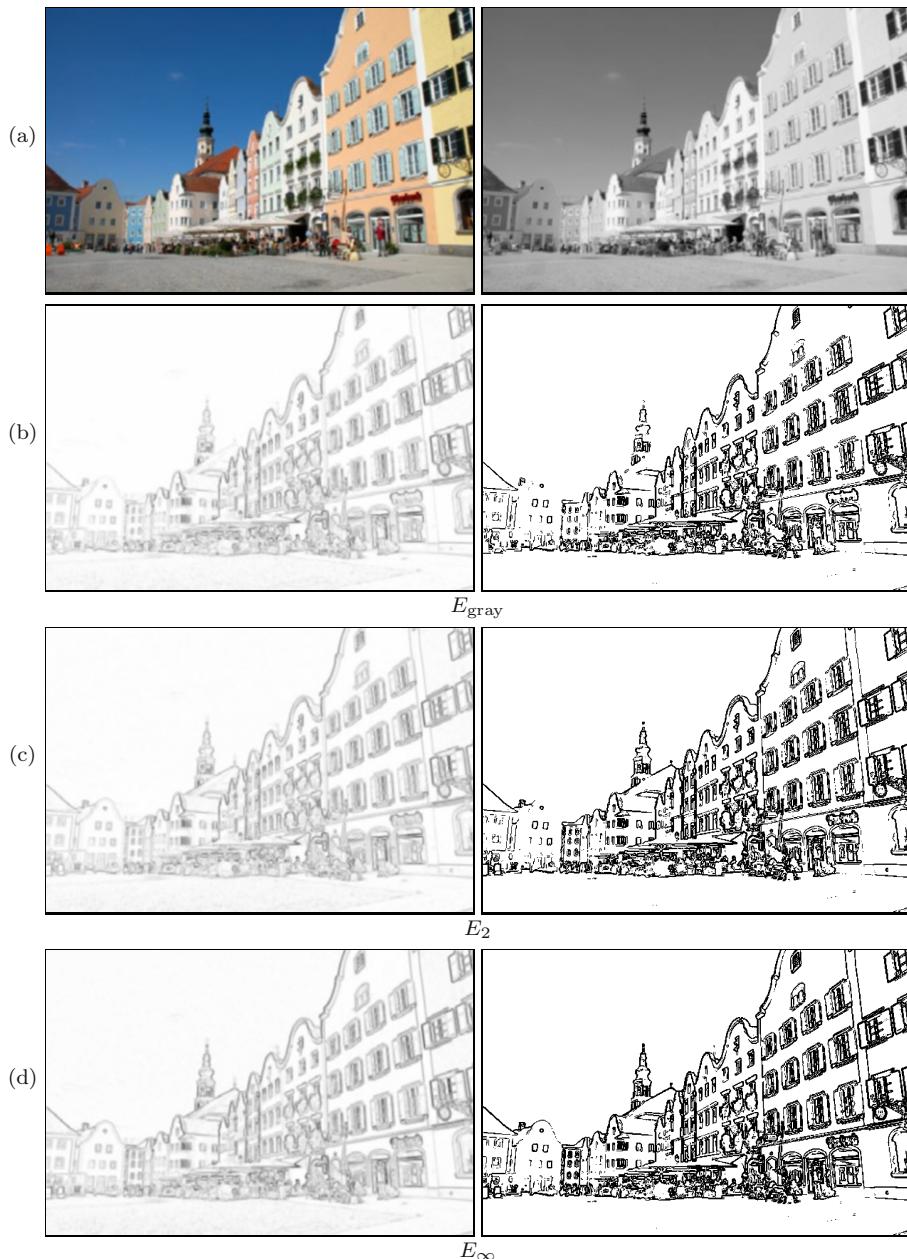


Figure 4.3 Example of color edge enhancement with monochromatic techniques (town01 image). Original color image and corresponding grayscale image (a), edge magnitude obtained from the grayscale image (b), color edge magnitude calculated with the L_2 norm (c) and the L_∞ norm (d). Binary images in the right column were obtained by thresholding the resulting edge strength.

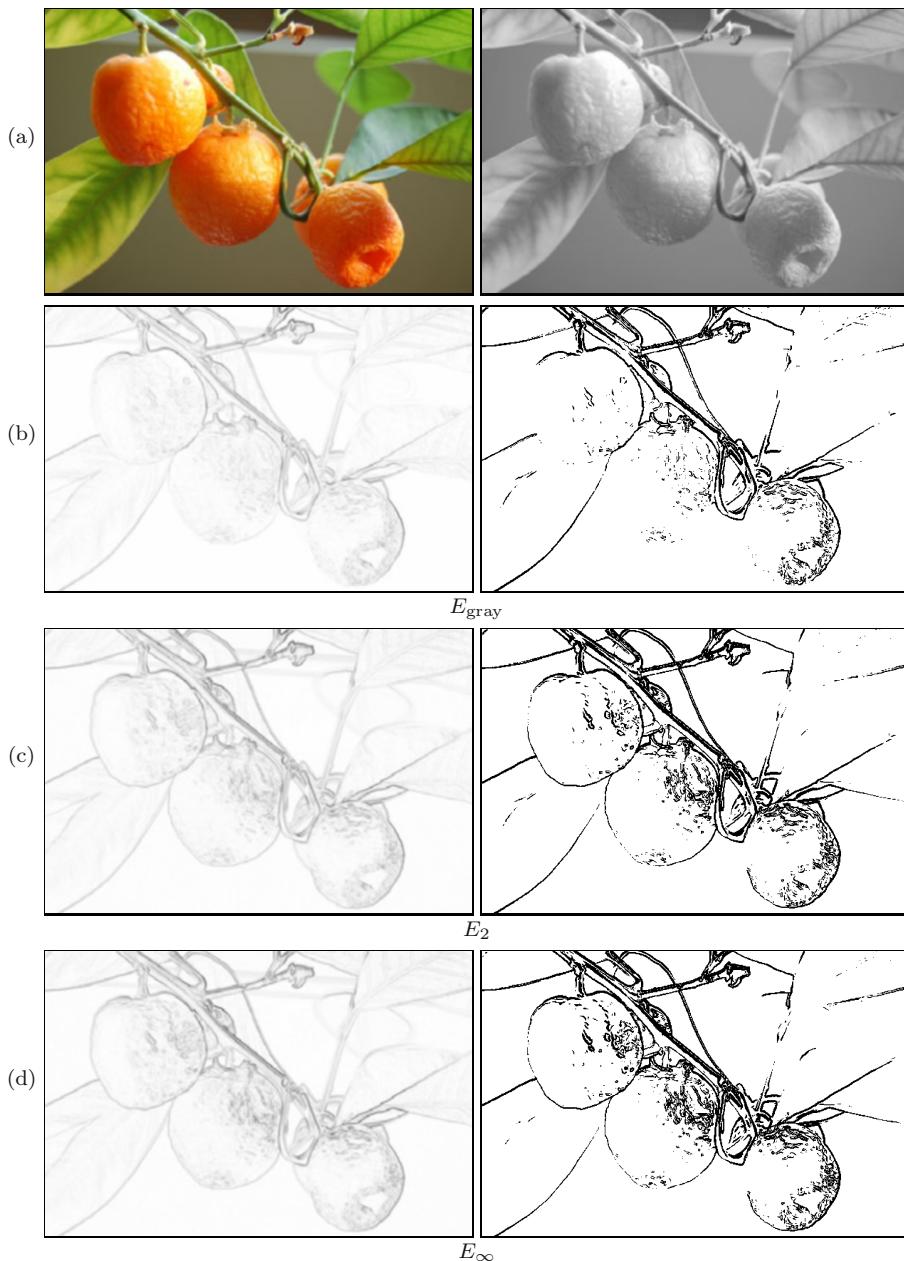


Figure 4.4 Example of color edge enhancement with monochromatic techniques (mandarins image). Original color image and corresponding grayscale image (a), edge magnitude obtained from the grayscale image (b), color edge magnitude calculated with the L_2 norm (c) and the L_∞ norm (d). Binary images in the right column were obtained by thresholding the resulting edge strength.

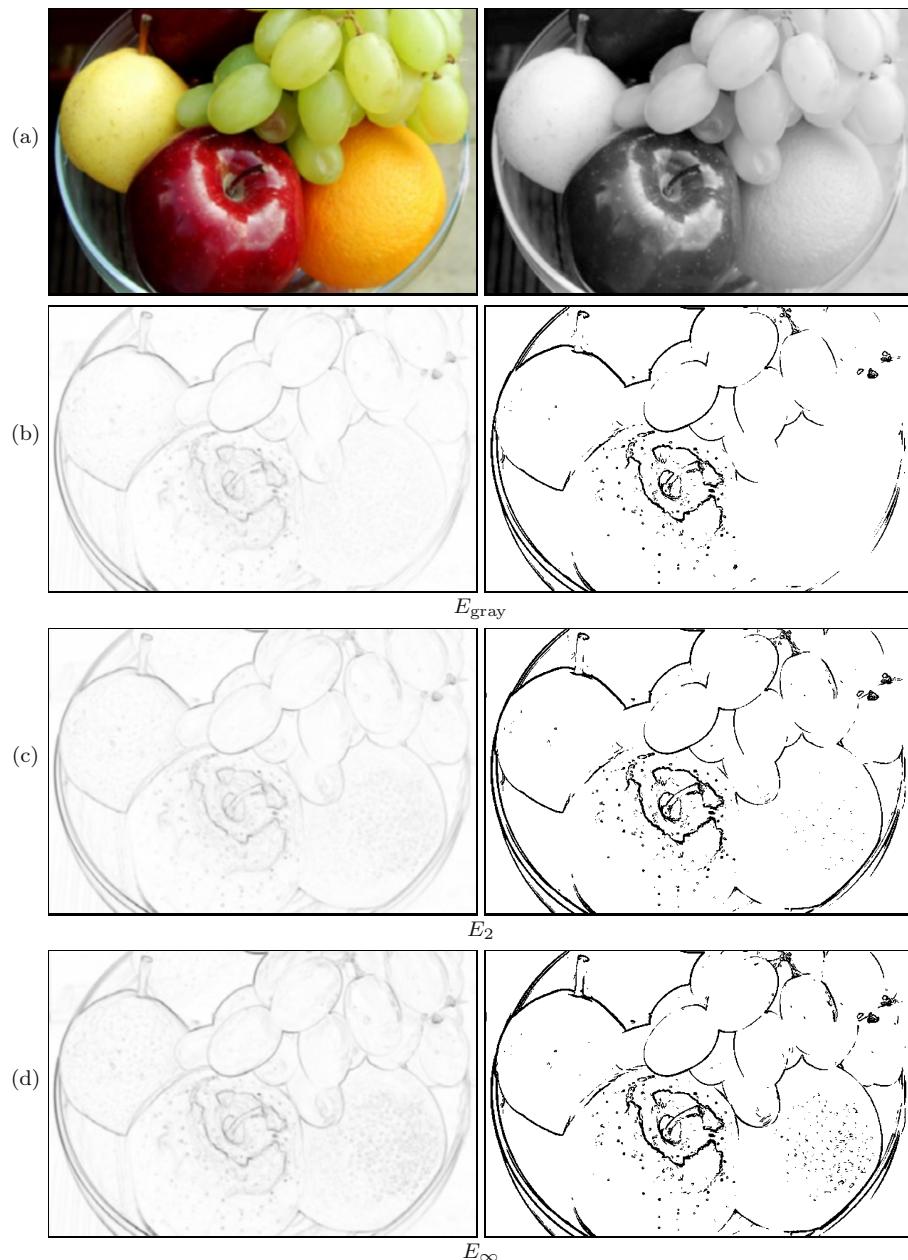


Figure 4.5 Example of color edge enhancement with monochromatic techniques (**fruits2** image). Original color image and corresponding grayscale image (a), edge magnitude obtained from the grayscale image (b), color edge magnitude calculated with the L_2 norm (c) and the L_∞ norm (d). Binary images in the right column were obtained by thresholding the resulting edge strength.

that is, the vector of the partial derivatives of the function I in the x - and y -direction, respectively.³ Obviously, the gradient of a scalar image is a two-dimensional vector field.

In the case of a color image $\mathbf{I} = (I_R, I_G, I_B)$, we can treat the three color channels as separate scalar images and obtain their gradients analogously as

$$\nabla I_R(\dot{\mathbf{x}}) = \begin{pmatrix} \frac{\partial I_R}{\partial x}(\dot{\mathbf{x}}) \\ \frac{\partial I_R}{\partial y}(\dot{\mathbf{x}}) \end{pmatrix}, \quad \nabla I_G(\dot{\mathbf{x}}) = \begin{pmatrix} \frac{\partial I_G}{\partial x}(\dot{\mathbf{x}}) \\ \frac{\partial I_G}{\partial y}(\dot{\mathbf{x}}) \end{pmatrix}, \quad \nabla I_B(\dot{\mathbf{x}}) = \begin{pmatrix} \frac{\partial I_B}{\partial x}(\dot{\mathbf{x}}) \\ \frac{\partial I_B}{\partial y}(\dot{\mathbf{x}}) \end{pmatrix}, \quad (4.14)$$

which is exactly what we did before in Eqn. (4.5). Before we can take the next steps, we need to introduce a standard tool for the analysis of vector fields.

4.2.2 The Jacobian matrix

The *Jacobian* matrix⁴ $\mathbf{J}_I(\dot{\mathbf{x}})$ combines all first partial derivatives of a vector field \mathbf{I} at a given position $\dot{\mathbf{x}}$, its row vectors being the gradients of the scalar component functions. In particular, for our RGB color image \mathbf{I} , the Jacobian matrix is defined as

$$\mathbf{J}_I(\dot{\mathbf{x}}) = \begin{pmatrix} (\nabla I_R)^\top(\dot{\mathbf{x}}) \\ (\nabla I_G)^\top(\dot{\mathbf{x}}) \\ (\nabla I_B)^\top(\dot{\mathbf{x}}) \end{pmatrix} = \begin{pmatrix} \frac{\partial I_R}{\partial x}(\dot{\mathbf{x}}) & \frac{\partial I_R}{\partial y}(\dot{\mathbf{x}}) \\ \frac{\partial I_G}{\partial x}(\dot{\mathbf{x}}) & \frac{\partial I_G}{\partial y}(\dot{\mathbf{x}}) \\ \frac{\partial I_B}{\partial x}(\dot{\mathbf{x}}) & \frac{\partial I_B}{\partial y}(\dot{\mathbf{x}}) \end{pmatrix} = \begin{pmatrix} \mathbf{I}_x(\dot{\mathbf{x}}) & \mathbf{I}_y(\dot{\mathbf{x}}) \end{pmatrix}, \quad (4.15)$$

with $\nabla I_R, \nabla I_G, \nabla I_B$ as defined in Eqn. (4.14). We see that the two-dimensional gradient vectors $(\nabla I_R)^\top, (\nabla I_G)^\top, (\nabla I_B)^\top$ constitute the rows of the resulting 3×2 matrix \mathbf{J}_I . The two three-dimensional column vectors of this matrix,

$$\mathbf{I}_x(\dot{\mathbf{x}}) = \frac{\partial \mathbf{I}}{\partial x}(\dot{\mathbf{x}}) = \begin{pmatrix} \frac{\partial I_R}{\partial x}(\dot{\mathbf{x}}) \\ \frac{\partial I_G}{\partial x}(\dot{\mathbf{x}}) \\ \frac{\partial I_B}{\partial x}(\dot{\mathbf{x}}) \end{pmatrix} \quad \text{and} \quad \mathbf{I}_y(\dot{\mathbf{x}}) = \frac{\partial \mathbf{I}}{\partial y}(\dot{\mathbf{x}}) = \begin{pmatrix} \frac{\partial I_R}{\partial y}(\dot{\mathbf{x}}) \\ \frac{\partial I_G}{\partial y}(\dot{\mathbf{x}}) \\ \frac{\partial I_B}{\partial y}(\dot{\mathbf{x}}) \end{pmatrix}, \quad (4.16)$$

are the partial derivatives of the color components along the x - and y -axis, respectively. At a given position $\dot{\mathbf{x}}$, the total amount of change over all three color channels in the horizontal direction can be quantified by the norm of the corresponding column vector $\|\mathbf{I}_x(\dot{\mathbf{x}})\|$. Analogously, $\|\mathbf{I}_y(\dot{\mathbf{x}})\|$ gives the total amount of change over all three color channels along the vertical axis.

³ Of course, images are discrete functions and the partial derivatives are estimated from finite differences (see Sec. B.7.1 in the Appendix).

⁴ See also Sec. B.6.1 in the Appendix.

4.2.3 Squared local contrast

Now that we can quantify the change along the horizontal and vertical axes at any position $\dot{\mathbf{x}}$, the next task is to find out the direction of the *maximum* change to find the angle of the edge normal, which we then use to derive the local edge strength. How can we calculate the gradient in some direction θ other than horizontal and vertical? For this purpose, we use the product of the unit vector oriented at angle θ ,

$$\mathbf{e}_\theta = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix}, \quad (4.17)$$

and the Jacobian matrix \mathbf{J}_I (Eqn. (4.15)) in the form

$$\begin{aligned} (\text{grad}_\theta \mathbf{I})(\dot{\mathbf{x}}) &= \mathbf{J}_I(\dot{\mathbf{x}}) \cdot \mathbf{e}_\theta = \left(\mathbf{I}_x(\dot{\mathbf{x}}) \mid \mathbf{I}_y(\dot{\mathbf{x}}) \right) \cdot \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} \\ &= \mathbf{I}_x(\dot{\mathbf{x}}) \cdot \cos(\theta) + \mathbf{I}_y(\dot{\mathbf{x}}) \cdot \sin(\theta). \end{aligned} \quad (4.18)$$

The resulting three-dimensional vector $(\text{grad}_\theta \mathbf{I})(\dot{\mathbf{x}})$ is called the *directional gradient*⁵ of the color image \mathbf{I} in the direction θ at position $\dot{\mathbf{x}}$. By taking the squared norm of this vector,

$$\begin{aligned} S_\theta(\mathbf{I}, \dot{\mathbf{x}}) &= \|(\text{grad}_\theta \mathbf{I})(\dot{\mathbf{x}})\|_2^2 = \|\mathbf{I}_x(\dot{\mathbf{x}}) \cdot \cos(\theta) + \mathbf{I}_y(\dot{\mathbf{x}}) \cdot \sin(\theta)\|_2^2 \\ &= \mathbf{I}_x^2(\dot{\mathbf{x}}) \cdot \cos^2(\theta) + 2 \cdot \mathbf{I}_x(\dot{\mathbf{x}}) \cdot \mathbf{I}_y(\dot{\mathbf{x}}) \cdot \cos(\theta) \cdot \sin(\theta) + \mathbf{I}_y^2(\dot{\mathbf{x}}) \cdot \sin^2(\theta), \end{aligned} \quad (4.19)$$

we obtain what is called the *squared local contrast* of the vector-valued image \mathbf{I} at position $\dot{\mathbf{x}}$ in direction θ .⁶ For an RGB image $\mathbf{I} = (I_R, I_G, I_B)$, the squared local contrast in Eqn. (4.19) is, explicitly written,

$$\begin{aligned} S_\theta(\mathbf{I}, \dot{\mathbf{x}}) &= \left\| \begin{pmatrix} I_{R,x}(\dot{\mathbf{x}}) \\ I_{G,x}(\dot{\mathbf{x}}) \\ I_{B,x}(\dot{\mathbf{x}}) \end{pmatrix} \cdot \cos(\theta) + \begin{pmatrix} I_{R,y}(\dot{\mathbf{x}}) \\ I_{G,y}(\dot{\mathbf{x}}) \\ I_{B,y}(\dot{\mathbf{x}}) \end{pmatrix} \cdot \sin(\theta) \right\|_2^2 \\ &= [I_{R,x}^2(\dot{\mathbf{x}}) + I_{G,x}^2(\dot{\mathbf{x}}) + I_{B,x}^2(\dot{\mathbf{x}})] \cdot \cos^2(\theta) + [I_{R,y}^2(\dot{\mathbf{x}}) + I_{G,y}^2(\dot{\mathbf{x}}) + I_{B,y}^2(\dot{\mathbf{x}})] \cdot \sin^2(\theta) \\ &\quad + 2 \cdot [I_{R,x}(\dot{\mathbf{x}}) \cdot I_{R,y}(\dot{\mathbf{x}}) + I_{G,x}(\dot{\mathbf{x}}) \cdot I_{G,y}(\dot{\mathbf{x}}) + I_{B,x}(\dot{\mathbf{x}}) \cdot I_{B,y}(\dot{\mathbf{x}})] \cdot \cos(\theta) \cdot \sin(\theta). \end{aligned} \quad (4.20)$$

Note that, in the case that I is a *scalar* image, the squared local contrast reduces to

$$\begin{aligned} S_\theta(I, \dot{\mathbf{x}}) &= \|(\text{grad}_\theta I)(\dot{\mathbf{x}})\|_2^2 = \left\| \begin{pmatrix} I_x(\dot{\mathbf{x}}) \\ I_y(\dot{\mathbf{x}}) \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} \right\|_2^2 \\ &= [I_x(\dot{\mathbf{x}}) \cdot \cos(\theta) + I_y(\dot{\mathbf{x}}) \cdot \sin(\theta)]^2. \end{aligned} \quad (4.21)$$

⁵ See also Eqn. (B.49) on p. 316 in Appendix B.

⁶ Note that $\mathbf{I}_x^2 = \mathbf{I}_x \cdot \mathbf{I}_x$, $\mathbf{I}_y^2 = \mathbf{I}_y \cdot \mathbf{I}_y$ and $\mathbf{I}_x \cdot \mathbf{I}_y$ in Eqn. (4.19) are dot products and thus the results are scalar values.

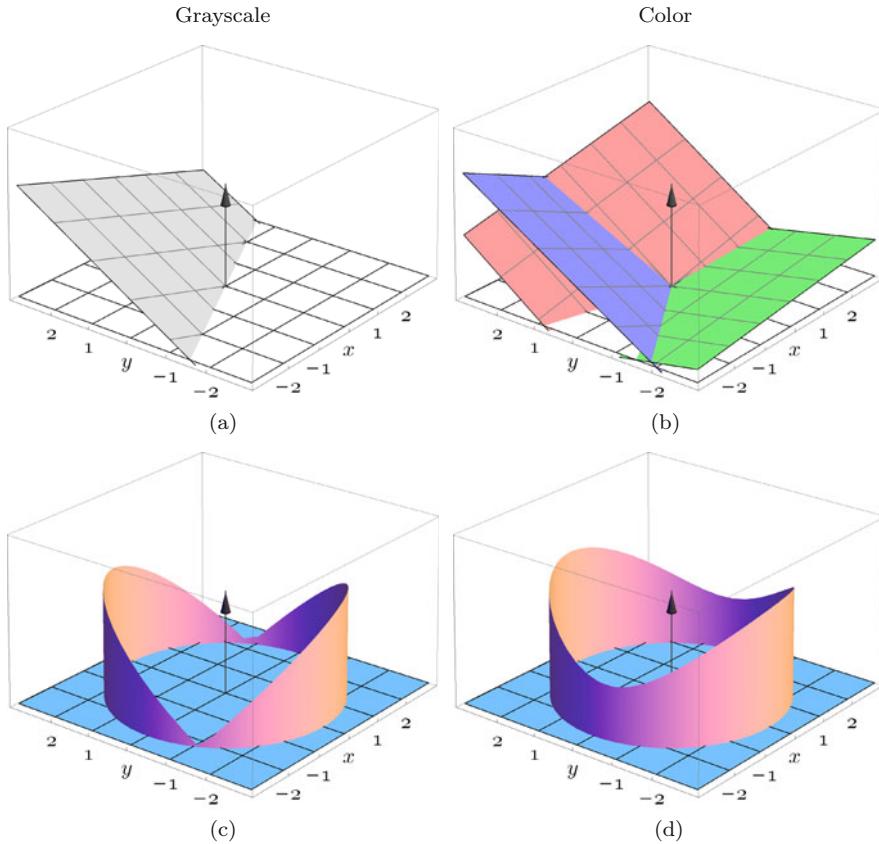


Figure 4.6 Local image gradients and squared local contrast. In case of a scalar (grayscale) image (a), the local gradient $\nabla I = (\partial I / \partial x, \partial I / \partial y)$ defines a single plane that is tangential to the image function I at position $\hat{\mathbf{x}} = (\hat{x}, \hat{y})$. In case of an RGB color image (b), the local gradients $\nabla I_R, \nabla I_G, \nabla I_B$ for each color channel define three tangent planes. The vertical axes in graphs (c, d) show the corresponding local contrast value $\sqrt{S_\theta(\mathbf{I}, \hat{\mathbf{x}})}$ (see Eqns. 4.19–4.20) for directions $\theta = 0 \dots 2\pi$, with two opposing minima along the edge tangent and maxima in the direction of the edge normal.

Figure 4.6 illustrates the meaning of the squared local contrast in relation to the local image gradients. At a given image position $\hat{\mathbf{x}}$, the local gradient $\nabla I(\hat{\mathbf{x}})$ in a grayscale image ([Fig. 4.6 \(a\)](#)) defines a single plane that is tangential to the image function I at position $\hat{\mathbf{x}}$. In case of an RGB image ([Fig. 4.6 \(b\)](#)), each color channel defines an individual tangent plane.

4.2.4 Color edge magnitude

The directions that *maximize* $S_\theta(\mathbf{I}, \hat{\mathbf{x}})$ in Eqn. (4.19) can be found analytically as the roots of the first partial derivative of S with respect to the angle θ ,

as originally suggested by Di Zenzo [36], and the resulting quantity is called *maximum local contrast*. As shown in [33], the maximum local contrast can also be found as the largest eigenvalue of the 2×2 matrix

$$\mathbf{M} = \mathbf{J}_I^\top \cdot \mathbf{J}_I = \begin{pmatrix} \mathbf{I}_x \\ \mathbf{I}_y \end{pmatrix} \cdot (\mathbf{I}_x \mid \mathbf{I}_y) = \begin{pmatrix} \mathbf{I}_x^2 & \mathbf{I}_x \mathbf{I}_y \\ \mathbf{I}_y \mathbf{I}_x & \mathbf{I}_y^2 \end{pmatrix} = \begin{pmatrix} A & C \\ C & B \end{pmatrix}, \quad (4.22)$$

with elements⁷

$$A = \mathbf{I}_x^2 = \mathbf{I}_x \cdot \mathbf{I}_x, \quad B = \mathbf{I}_y^2 = \mathbf{I}_y \cdot \mathbf{I}_y, \quad C = \mathbf{I}_x \cdot \mathbf{I}_y = \mathbf{I}_y \cdot \mathbf{I}_x. \quad (4.23)$$

Note that the matrix \mathbf{M} is the color equivalent to the *local structure matrix* used for corner detection on grayscale images (see Vol. 2, Ch. 4 [21]). The eigenvalues of \mathbf{M} can be found in closed form as⁸

$$\lambda_{1,2} = \frac{A + B \pm \sqrt{(A - B)^2 + 4C^2}}{2}. \quad (4.24)$$

Since \mathbf{M} is symmetric, the expression under the square root is positive and thus all eigenvalues are real. In addition, A, B are both positive and therefore

$$\lambda_1 = \frac{A + B + \sqrt{(A - B)^2 + 4C^2}}{2} \quad (4.25)$$

is always the *larger* of the two eigenvalues.⁹ It is equivalent to the maximum squared local contrast (Eqn. (4.19)), that is

$$\lambda_1(\dot{\mathbf{x}}) = \max_{0 \leq \theta < 2\pi} S_\theta(\mathbf{I}, \dot{\mathbf{x}}) \quad (4.26)$$

and thus $\sqrt{\lambda_1}$ can be used directly to quantify the local edge strength. One of the *eigenvectors* associated with λ_1 is

$$\mathbf{x}_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} A - B + \sqrt{(A - B)^2 + 4C^2} \\ 2C \end{pmatrix}, \quad (4.27)$$

or, equivalently, any multiple of \mathbf{x}_1 .¹⁰ Thus the rate of change along the vector \mathbf{x}_1 is the same as in the opposite direction $-\mathbf{x}_1$ and it follows that the squared

⁷ For brevity, we omit the position argument $(\dot{\mathbf{x}})$ and use only $\mathbf{I}_x, \mathbf{I}_y$ instead of $\mathbf{I}_x(\dot{\mathbf{x}}), \mathbf{I}_y(\dot{\mathbf{x}})$ etc. in the following.

⁸ See Sec. B.4 in the Appendix for how to calculate the eigenvalues and eigenvectors of a 2×2 matrix.

⁹ Note that, in Eqn. (4.25), $C^2 = (\mathbf{I}_x \cdot \mathbf{I}_y)^2 \neq \mathbf{I}_x^2 \cdot \mathbf{I}_y^2$ and therefore $(A - B)^2 + 4C^2 \neq (A + B)^2$. Thus, in general, $\lambda_1, \lambda_2 \geq 0$ for vector-valued images.

¹⁰ The eigenvalues of a matrix are unique, but the corresponding eigenvectors are not. If some “eigenpair” $\langle \lambda, \mathbf{x} \rangle$, consisting of the *eigenvalue* λ and the corresponding *eigenvector* \mathbf{x} , satisfies $\mathbf{A} \cdot \mathbf{x} = \lambda \cdot \mathbf{x}$, then any scaled vector $\mathbf{x}' = s\mathbf{x}$ (with $s \neq 0$) is also a valid eigenvector for λ (also see Appendix B.4).

local contrast $S(\hat{\mathbf{x}}, \theta)$ at orientation θ is the same at orientation $\theta + k\pi$ (for any $k \in \mathbb{Z}$). Therefore, under the given definition of contrast and in the absence of additional constraints, the orientation of maximum change is inherently ambiguous [34]. While \mathbf{x}_1 (the eigenvector associated with the greater eigenvalue of \mathbf{M}) points in the direction of maximum change, the second eigenvector \mathbf{x}_2 is orthogonal to \mathbf{x}_1 , i.e., has the same direction as the local edge tangent. The *unit vector* corresponding to \mathbf{x}_1 is obtained by scaling \mathbf{x}_1 by its magnitude, that is

$$\hat{\mathbf{x}}_1 = \begin{pmatrix} \hat{x}_1 \\ \hat{y}_1 \end{pmatrix} = \frac{1}{\|\mathbf{x}_1\|_2} \cdot \mathbf{x}_1. \quad (4.28)$$

An alternative method, proposed in [34], is to calculate the unit eigenvector $\hat{\mathbf{x}}_1 = (\hat{x}_1, \hat{y}_1)^\top$ in the form

$$\hat{x}_1 = \sqrt{\frac{1+c}{2}}, \quad \hat{y}_1 = \text{sgn}(C) \cdot \sqrt{\frac{1-c}{2}}, \quad (4.29)$$

with $c = (A - B) / \sqrt{(A - B)^2 + 4C^2}$, directly from the matrix elements A, B, C defined in Eqn. (4.23).

4.2.5 Color edge orientation

The local orientation of the edge (i.e., the *edge normal*) can be obtained from the eigenvector $\mathbf{x}_1 = (x_1, y_1)^\top$ (Eqn. (4.27)) using the relation

$$\tan(\theta_1) = \frac{y_1}{x_1} = \frac{2C}{A - B + \sqrt{(A - B)^2 + 4C^2}}, \quad (4.30)$$

which can be simplified¹¹ to $\tan(2\theta_1) = (2C)/(A - B)$. Unless both $A = B$ and $C = 0$, in which case the edge orientation is undetermined, the angle of maximum local contrast (color edge orientation) can be calculated as

$$\theta_1 = \frac{1}{2} \cdot \tan^{-1}\left(\frac{2C}{A - B}\right). \quad (4.31)$$

The above steps are summarized in [Alg. 4.2](#), which is a color edge operator based on the first derivatives of the image function (see Sec. 4.4 for the corresponding Java implementation). It is similar to the algorithm proposed by Di Zenzo [36] but uses the eigenvalues of the local structure matrix $\mathbf{M} = \begin{pmatrix} A & C \\ C & B \end{pmatrix}$ for calculating edge magnitude and orientation, as suggested in [33] (see Eqn. (4.22)).

Results of the monochromatic edge operator in [Alg. 4.1](#) and the multi-gradient operator in [Alg. 4.2](#) are compared in [Fig. 4.7](#). The test image has

¹¹ Using the trigonometric relation $\tan(2\theta) = (2\tan(\theta)) / (1 - \tan^2(\theta))$.

Algorithm 4.2 “DiZenzo/Cumani-style” multi-gradient color edge operator. A pair of Sobel-type filters (H_x^S, H_y^S) is used for estimating the local x/y gradients in each component of the RGB input image \mathbf{I} . The procedure returns a pair of maps, holding the edge magnitude E and the edge orientation Φ , respectively.

```

1: MULTIGRADIENTCOLOREDGE( $\mathbf{I}$ )
   Input:  $\mathbf{I} = (I_R, I_G, I_B)$ , an RGB color image of size  $M \times N$ .
   Returns a pair of maps  $(E, \Phi)$  for edge magnitude and orientation.

2:  $H_x^S \leftarrow \frac{1}{8} \cdot \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad H_y^S \leftarrow \frac{1}{8} \cdot \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$   $\triangleright x/y$  gradient kernels
3:  $(M, N) \leftarrow \text{SIZE}(\mathbf{I})$ 
4: Create maps  $E, \Phi : M \times N \mapsto \mathbb{R}$   $\triangleright$  edge magnitude/orientation
5:  $I_{R,x} \leftarrow I_R * H_x^S, \quad I_{R,y} \leftarrow I_R * H_y^S$   $\triangleright$  apply gradient filters
6:  $I_{G,x} \leftarrow I_G * H_x^S, \quad I_{G,y} \leftarrow I_G * H_y^S$ 
7:  $I_{B,x} \leftarrow I_B * H_x^S, \quad I_{B,y} \leftarrow I_B * H_y^S$ 

8: for all image coordinates  $(u, v) \in M \times N$  do
9:    $(r_x, g_x, b_x) \leftarrow (I_{R,x}(u, v), I_{G,x}(u, v), I_{B,x}(u, v))$ 
10:   $(r_y, g_y, b_y) \leftarrow (I_{R,y}(u, v), I_{G,y}(u, v), I_{B,y}(u, v))$ 
11:   $A \leftarrow r_x^2 + g_x^2 + b_x^2$   $\triangleright A = \mathbf{I}_x \cdot \mathbf{I}_x$ 
12:   $B \leftarrow r_y^2 + g_y^2 + b_y^2$   $\triangleright B = \mathbf{I}_y \cdot \mathbf{I}_y$ 
13:   $C \leftarrow r_x \cdot r_y + g_x \cdot g_y + b_x \cdot b_y$   $\triangleright C = \mathbf{I}_x \cdot \mathbf{I}_y$ 
14:   $\lambda_1 \leftarrow \frac{1}{2} \cdot (A+B+\sqrt{(A-B)^2+4C^2})$   $\triangleright$  Eqn. (4.25)
15:   $E(u, v) \leftarrow \sqrt{\lambda_1}$   $\triangleright$  edge magnitude,  $\lambda_1 = \max_\theta S_\theta(\mathbf{I}, \dot{\mathbf{x}}, \theta)$ 
16:   $\Phi(u, v) \leftarrow \frac{1}{2} \cdot \text{Arctan}(2C, A-B)$   $\triangleright$  edge orientation (normal)
17: return  $(E, \Phi)$ .
```

constant luminance ($L = 65$) and thus no gray-value operator would be able to detect edges in this image. Before applying the edge operators, the 50×50 image was smoothed with a Gaussian filter with $\sigma = 2.0$. The local edge strength $E(u, v)$ is identical for both operators (Fig. 4.7 (b)). The vectors in Fig. 4.7 (b–f) show the orientation of the edge tangents that are normals to the direction of maximum color contrast, $\Phi(u, v)$. The length of each tangent vector is proportional to the local edge strength. In the case of the monochromatic operator (Fig. 4.7 (c, e)), the tangent vectors are normal to the gradient direction of the color channel with maximum local contrast. For the multi-gradient operator (Fig. 4.7 (d, f)) the orientation is derived from all three color channels, as defined in Eqn. (4.31). Although edge orientations are more accurate with the multi-gradient approach, the results of the simpler monochromatic operator do not fall behind much and may be sufficient for most practical applications. Another synthetic example calculated with the multi-gradient algorithm is shown

in Fig. 4.8.

Figure 4.9 shows results from the “Di Zenzo/Cumani-style” multi-gradient edge operator (Alg. 4.2) for real images. Note that the multi-gradient edge magnitude (calculated from eigenvalue λ_1) is virtually identical to the monochromatic edge magnitude E_{mag} shown in Figures 4.2–4.5 (e). Larger differences may occur locally at multi-color junctions but are usually not noticeable. Thus, considering only edge *magnitude*, the multi-gradient operator has no significant advantage over the simpler, monochromatic techniques, particularly compared to the maximum channel gradient technique (cf. Figs. 4.2–4.5 (g, h)). However, if edge *orientation* is important (as in the color version of the Canny operator described in Sec. 4.3.2), the multi-gradient technique is certainly more reliable and consistent.

4.2.6 Grayscale gradients revisited

For a scalar-valued (monochromatic) image, the usual gradient-based calculation of the edge orientation (see Vol. 1, Sec. 6.2 [20]) is only a special case of the multi-dimensional gradient calculation shown above. Given a scalar image I , the intensity gradient vector $(\nabla I)(\dot{\mathbf{x}}) = (I_x(\dot{\mathbf{x}}), I_y(\dot{\mathbf{x}}))^T$ defines a single plane that is tangential to the image function at position $\dot{\mathbf{x}}$, as illustrated in Fig. 4.6 (a). With

$$A(\dot{\mathbf{x}}) = I_x^2(\dot{\mathbf{x}}), \quad B(\dot{\mathbf{x}}) = I_y^2(\dot{\mathbf{x}}), \quad C(\dot{\mathbf{x}}) = I_x(\dot{\mathbf{x}}) \cdot I_y(\dot{\mathbf{x}}), \quad (4.32)$$

(analogous to Eqn. (4.23)) the squared local contrast at position $\dot{\mathbf{x}}$ in direction θ (as defined in Eqn. (4.19)) is

$$S_\theta(I, \dot{\mathbf{x}}) = (I_x(\dot{\mathbf{x}}) \cdot \cos(\theta) + I_y(\dot{\mathbf{x}}) \cdot \sin(\theta))^2. \quad (4.33)$$

From Eqn. (4.24), the eigenvalues of the local structure matrix $\mathbf{M} = (\begin{smallmatrix} A & C \\ C & B \end{smallmatrix})$ at position $\dot{\mathbf{x}}$ are

$$\lambda_{1,2} = \frac{A + B \pm \sqrt{(A - B)^2 + 4C^2}}{2}, \quad (4.34)$$

but here, with I_x , I_y being no vectors but scalar values, we get $C^2 = (I_x \cdot I_y)^2 = I_x^2 \cdot I_y^2$, such that $(A - B)^2 + 4C^2 = (A + B)^2$, and therefore

$$\lambda_{1,2} = \frac{A + B \pm (A + B)}{2}. \quad (4.35)$$

We see that, for a scalar-valued image, the dominant eigenvalue,

$$\lambda_1 = A + B = I_x^2 + I_y^2 = \|\nabla I\|_2^2, \quad (4.36)$$

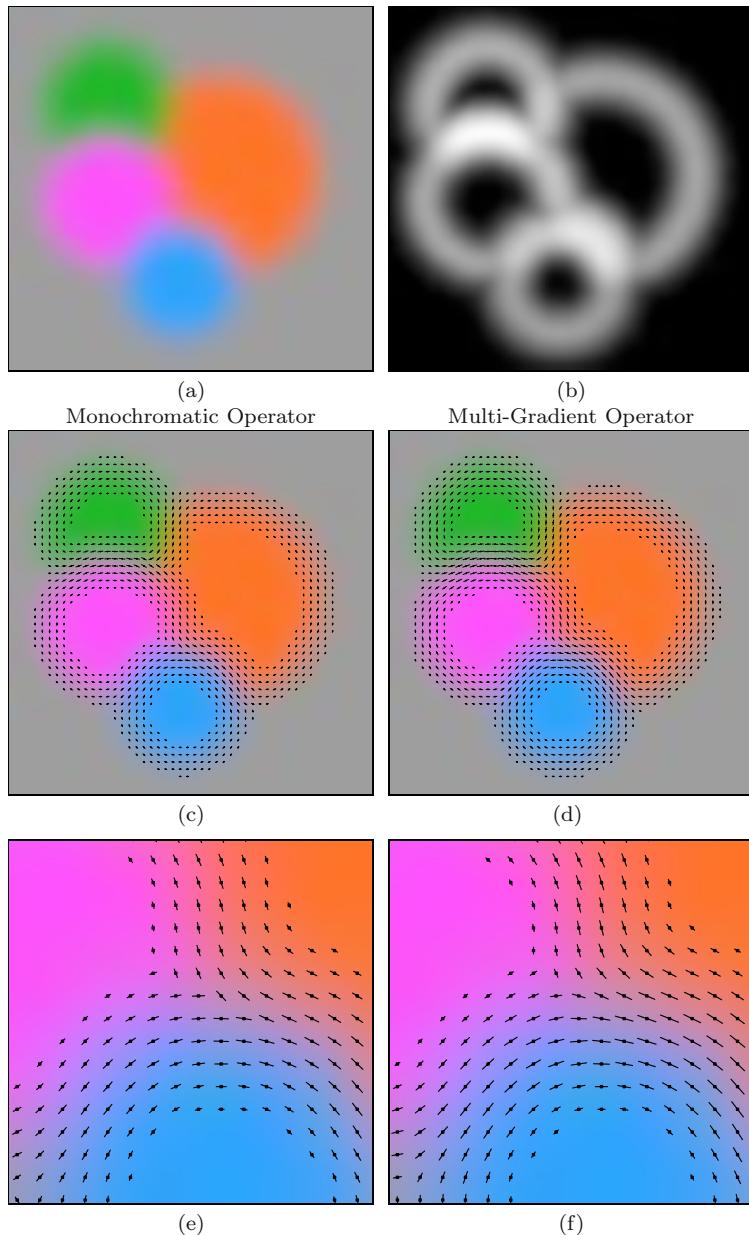


Figure 4.7 Results from the monochromatic (Alg. 4.1) and the multi-gradient color edge operators (Alg. 4.2). The original color image (a) has *constant luminance*, i.e., the intensity gradient is zero and thus a simple grayscale operator would not detect any edges at all. The local edge strength $E(u, v)$ is almost identical for both color edge operators (b). Edge tangent orientation vectors (normal to $\Phi(u, v)$) for the monochromatic and multi-gradient operators (c, d), with enlarged details in (e, f).

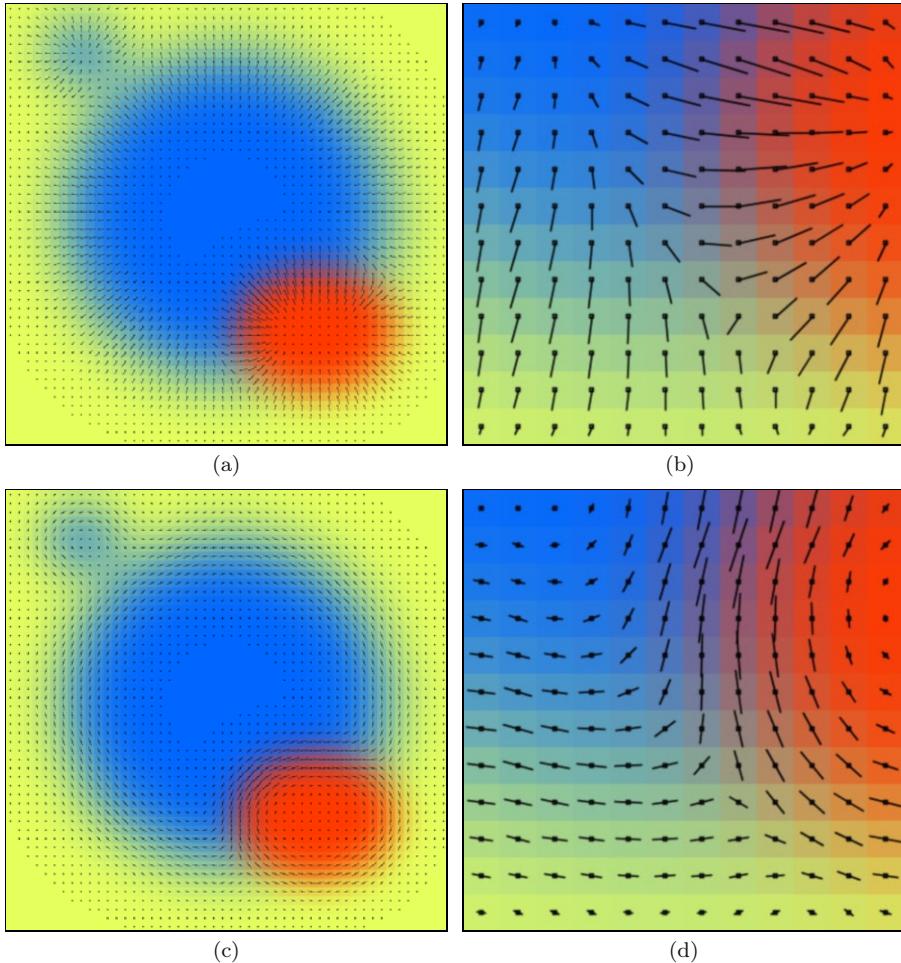


Figure 4.8 Edge magnitude and orientation from multi-gradient color edge operator. Directions of the maximum color gradient (a, b) and perpendicular edge tangents (c, d). The length of the displayed vectors is proportional to the local edge strength.

is simply the squared norm of the local gradient vector, while the smaller eigenvalue λ_2 is always zero. Thus, for a grayscale image, the maximum edge strength $\sqrt{\lambda_1} = \|\nabla I\|_2$ is equivalent to the *magnitude* of the local intensity gradient.¹² The fact that $\lambda_2 = 0$ indicates that the local contrast in the orthogonal direction (i.e., along the edge tangent) is zero (see Fig. 4.6 (c)). To calculate the local edge *orientation*, we use Eqn. (4.30) to get

$$\tan(\theta_1) = \frac{2C}{A - B + (A + B)} = \frac{2C}{2A} = \frac{I_x I_y}{I_x^2} = \frac{I_y}{I_x} \quad (4.37)$$

¹² See Eqns. (6.5) and (6.13) in Vol. 1 [20].

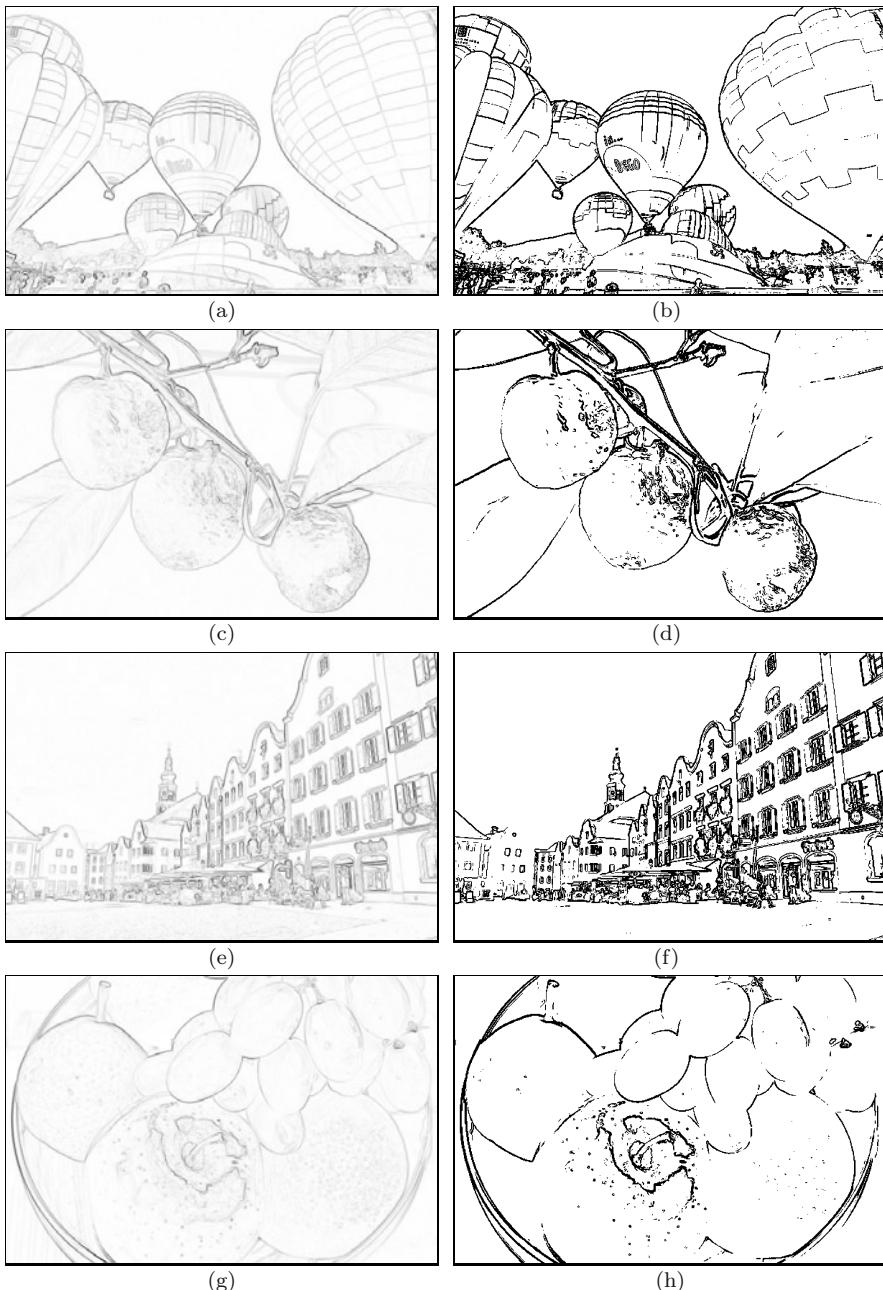


Figure 4.9 Results of multi-gradient edge operator (Alg. 4.2) on real images. Inverted edge magnitude (left column); the binary images (right column) were obtained by thresholding the corresponding edge strength using the ISODATA algorithm (described in Sec. 2.1.3). See Figs. 4.2–4.5 for the original color images and results from monochromatic edge detection.

and the direction of maximum contrast¹³ is then found as

$$\theta_1 = \tan^{-1}\left(\frac{I_y}{I_x}\right). \quad (4.38)$$

Thus, for grayscale images, the results are exactly the same as those obtained with traditional gradient-based edge detection, as described in Volume 1.

4.3 Canny edge operator

The edge operator proposed in [23] (and briefly described in Sec. 6.4.3 of Vol. 1 [20]) is still considered state-of-the-art and used in many applications today. The operator was designed to combine good detection of real edges, precise localization, and minimal response to spurious edges. These are desirable properties which simple edge operators—based on first-order derivative filters and global thresholding—do not provide. While the Canny operator also builds on first-order gradient filters, it locates edges by isolating the local maxima of the gradient magnitude and in this aspect resembles operators based on the detection of zero-crossings in second-order derivatives¹⁴ [85].

4.3.1 Canny edge detector for grayscale images

Although originally intended to detect edges at multiple spatial scales, most implementations apply the Canny detector at a single scale level only, specified by the width (σ) of its Gaussian smoothing filter. In its basic (single-scale) form, the Canny operator performs the following steps (stated more precisely in [Algs. 4.3–4.4](#)):

1. **Pre-processing:** Smooth the image with a Gaussian filter of width σ , which specifies the scale level of the edge detector. Calculate the x/y gradient vector at each position of the filtered image and determine the local gradient magnitude and orientation.
2. **Edge localization:** Isolate local maxima of gradient magnitude by “non-maximum suppression” along the local gradient direction.
3. **Edge tracing and hysteresis thresholding:** Collect sets of connected edge pixels from the local maxima by applying “hysteresis thresholding”.

¹³ See Eqn. (6.14) in Vol. 1 [20].

¹⁴ Zero-crossings of the second-order derivative of a function are found where the first-order derivative is either a local maximum or minimum.

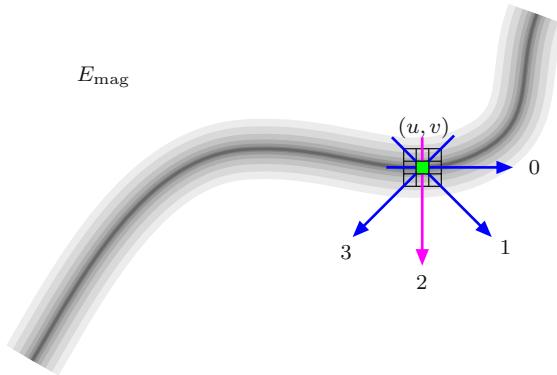


Figure 4.10 Non-maximum suppression of gradient magnitude. Only pixels where the gradient magnitude $E_{\text{mag}}(u, v)$ is a local maximum in the gradient direction (i.e., perpendicular to the edge tangent) are preserved as candidate edge points. The gradient magnitude at all other points is set (suppressed) to zero. For ease of processing, the gradient orientation is quantized into four octants $s_\theta = 0, \dots, 3$.

Pre-processing

The original intensity image I is first smoothed with a Gaussian filter kernel $H^{G,\sigma}$; its width σ specifies the spatial scale at which edges are to be detected (see [Alg. 4.3](#), lines 2–10). Subsequently, first-order difference filters are applied to the smoothed image \bar{I} to calculate the components \bar{I}_x, \bar{I}_y of the local gradient vectors. Then the local magnitude E_{mag} is calculated as the norm of the corresponding gradient vector. In view of the subsequent thresholding it may be helpful to normalize the edge magnitude values to a standard range (e.g., to $[0, 100]$).

Edge localization

Candidate edge pixels are isolated by local “non-maximum suppression” of the edge magnitude E_{mag} . In this step, only those pixels are preserved that represent a local maximum along the 1D-profile in the direction of the gradient, i.e., perpendicular to the edge tangent (see [Fig. 4.10](#)). While the gradient may point in any continuous direction, only *four* discrete directions are typically used to facilitate efficient processing. The pixel at position (u, v) is only retained as an edge candidate if its gradient magnitude is greater than both its immediate neighbors in the direction specified by the gradient vector (d_x, d_y) at position (u, v) . If a pixel is not a local maximum, its edge magnitude value is set to zero (i.e., “suppressed”). The non-maximum suppressed edge values are stored in the map E_{nms} .

The problem of finding the discrete orientation $s_\theta = 0, \dots, 3$ for a given gradient vector $\mathbf{d} = (d_x, d_y)$ is illustrated in Fig. 4.11. This task is simple if the corresponding angle $\theta = \tan^{-1}(d_y/d_x)$ is known, but in such a situation the use of the trigonometric functions is typically avoided for efficiency reasons. The octant that corresponds to \mathbf{d} can be inferred directly from the signs and magnitude of the components d_x, d_y , however, the necessary decision rules are quite complex. Much simpler rules apply if the coordinate system and gradient vector \mathbf{d} are rotated by $\frac{\pi}{8}$, as illustrated in Fig. 4.11(b). This step is implemented by the function GETORIENTATIONSECTOR() in Alg. 4.4.¹⁵

Edge tracing and hysteresis thresholding

In the final step, sets of connected edge points are collected from the magnitude values that remained unsuppressed in the previous operation. This is done with a technique called “hysteresis thresholding” using two different threshold values t_{hi} , t_{lo} (with $t_{hi} > t_{lo}$). The image is scanned for pixels with edge magnitude $E_{nms}(u, v) \geq t_{hi}$. Whenever such a (previously unvisited) location is found, a new *edge trace* is started and all connected edge pixels (u', v') are added to it as long as $E_{nms}(u', v') \geq t_{lo}$. Only those edge traces remain that contain at least one pixel with edge magnitude greater than t_{hi} and no pixels with edge magnitude less than t_{lo} . This process (which is similar to flood-fill region growing) is detailed in procedure GETORIENTATIONSECTOR in Alg. 4.4. Typical threshold values for 8-bit grayscale images are $t_{hi} = 5.0$ and $t_{lo} = 2.5$.

Figure 4.12 illustrates the effectiveness of non-maximum suppression for localizing the edge centers and edge-linking with hysteresis thresholding. Results from the single-scale Canny detector are shown in Fig. 4.13 for different settings of σ and fixed upper/lower threshold values $t_{hi} = 20\%$, $t_{lo} = 5\%$ (relative to the maximum gradient magnitude).

Due to the long-lasting popularity of the Canny operator, additional descriptions and some excellent illustrations can be found at various places in the literature, including [47, p. 719], [129, pp. 71–80], and [86, pp. 548–549]. An edge operator similar to the Canny detector, but based on a set of recursive filters, is described in [35]. While the Canny detector was originally designed for grayscale images, modified versions for color images exist, including the one we describe in the next section.

4.3.2 Canny edge detector for color images

Like most other edge operators, the Canny detector was originally designed for grayscale (i.e., scalar-valued) images. To use it on color images, a trivial

¹⁵ Note that the elements of the rotation matrix in Alg. 4.4 (line 2) are constants and thus no repeated use of trigonometric functions is required.

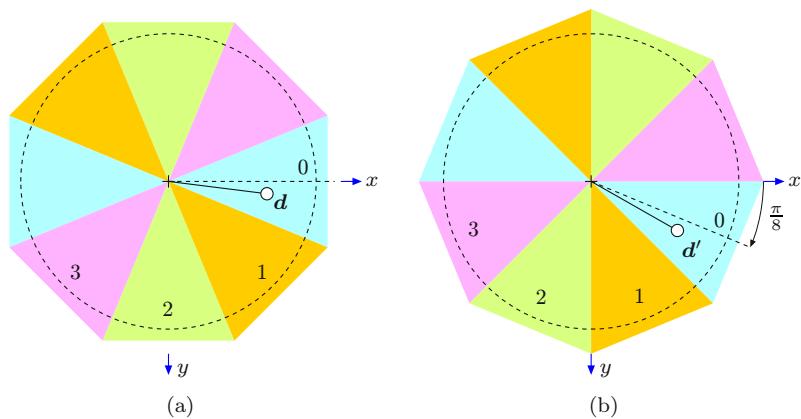


Figure 4.11 Discrete orientation sectors. In (a), finding the octant for a given orientation vector $\mathbf{d} = (d_x, d_y)$ requires a complex decision. Alternatively (b), if \mathbf{d} is rotated by $\frac{\pi}{8}$ to \mathbf{d}' , the corresponding octant can be found directly from the components of $\mathbf{d}' = (d'_x, d'_y)$ without the need to calculate the actual angle. Orientation vectors in the other octants are mirrored to octants $s_\theta = 0, 1, 2, 3$.

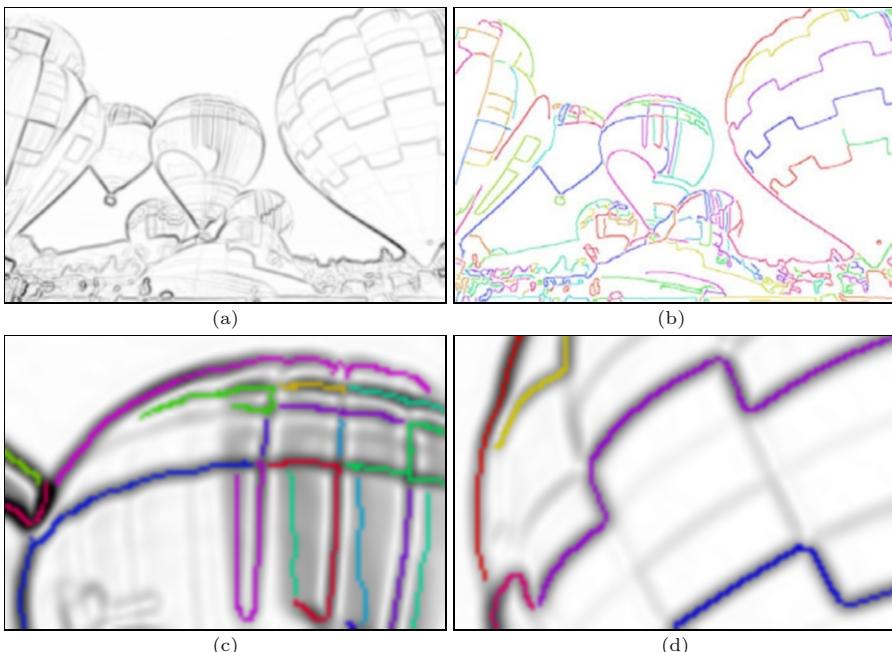


Figure 4.12 Grayscale Canny edge operator details. Inverted gradient magnitude (a), detected edge points with connected edge tracks shown in distinctive colors (b). Details with gradient magnitude and detected edge points overlaid (c, d). Settings: $\sigma = 2.0$, $t_{hi} = 20\%$, $t_{lo} = 5\%$.

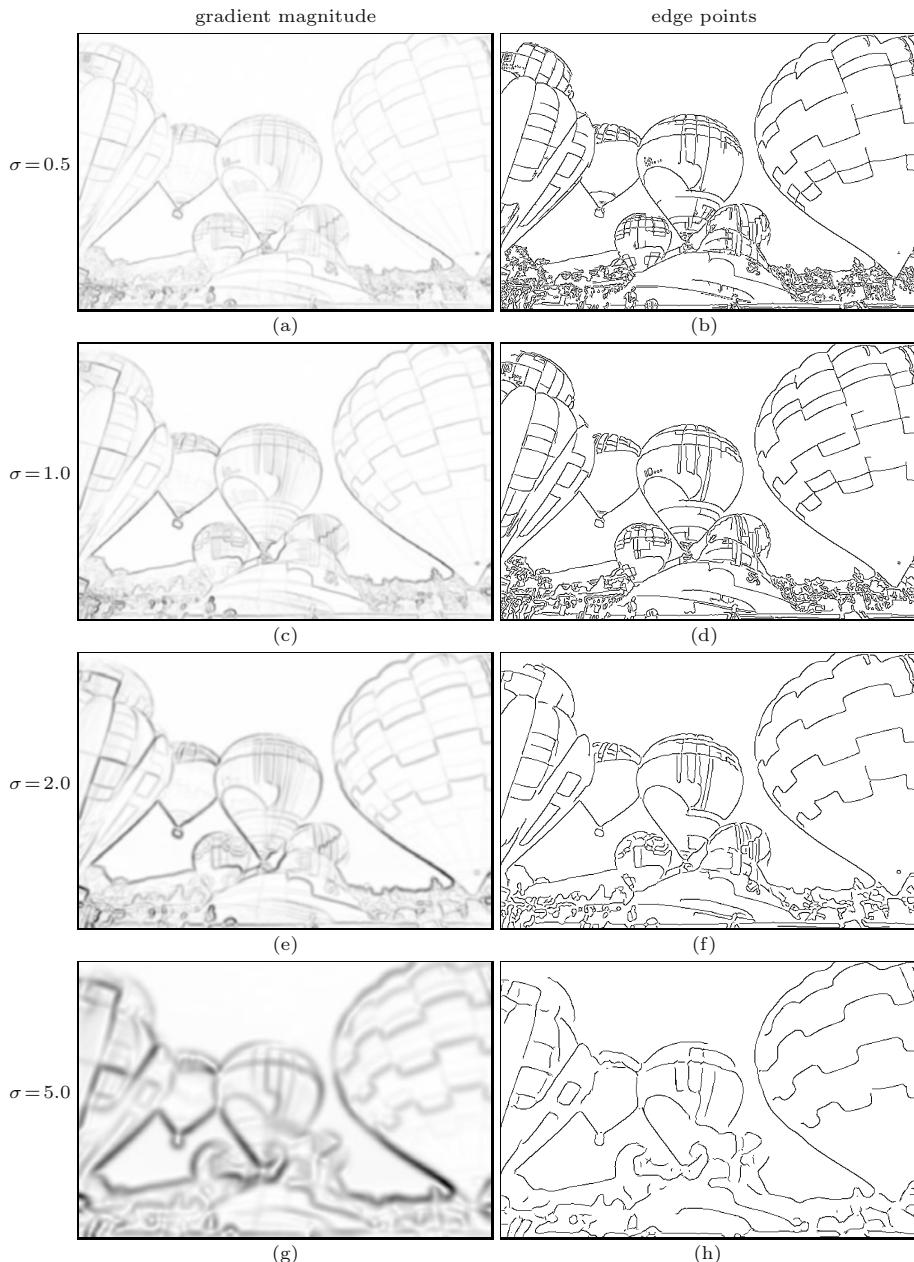


Figure 4.13 Results from the single-scale grayscale Canny edge operator (Algs. 4.3–4.4) for different values of σ ($t_{hi} = 20\%$, $t_{lo} = 5\%$); inverted gradient magnitude (left column) and detected edge points (right column).

Algorithm 4.3 Canny edge detector for grayscale images. Note that pixels at the image boundaries are excluded from being edge points.

```

1: CANNYEDGEDETECTOR( $I, \sigma, t_{hi}, t_{lo}$ )
   Input:  $I$ , a grayscale image of size  $M \times N$ ;
           $\sigma$ , radius of Gaussian filter  $H^{G,\sigma}$ ;
           $t_{hi}, t_{lo}$ , hysteresis thresholds ( $t_{hi} > t_{lo}$ ).
   Returns a binary edge image of size  $M \times N$ .
2:  $\bar{I} \leftarrow I * H^{G,\sigma}$                                  $\triangleright$  blur with Gaussian of width  $\sigma$ 
3:  $\bar{I}_x \leftarrow \bar{I} * [-0.5 \ 0 \ 0.5]$                  $\triangleright$   $x$ -gradient
4:  $\bar{I}_y \leftarrow \bar{I} * [-0.5 \ 0 \ 0.5]^T$              $\triangleright$   $y$ -gradient
5:  $(M, N) \leftarrow \text{SIZE}(I)$ 
6: Create maps:
7:    $E_{mag} : M \times N \mapsto \mathbb{R}$                    $\triangleright$  gradient magnitude
8:    $E_{nms} : M \times N \mapsto \mathbb{R}$                    $\triangleright$  maximum magnitude
9:    $E_{bin} : M \times N \mapsto \{0, 1\}$                  $\triangleright$  binary edge pixels
10:  for all image coordinates  $(u, v) \in M \times N$  do
11:     $E_{mag}(u, v) \leftarrow [\bar{I}_x^2(u, v) + \bar{I}_y^2(u, v)]^{1/2}$ 
12:     $E_{nms}(u, v) \leftarrow 0$ 
13:     $E_{bin}(u, v) \leftarrow 0$ 
14:  for  $u \leftarrow 1, \dots, M-2$  do
15:    for  $v \leftarrow 1, \dots, N-2$  do
16:       $d_x \leftarrow \bar{I}_x(u, v), d_y \leftarrow \bar{I}_y(u, v)$ 
17:       $s_\theta \leftarrow \text{GETORIENTATIONSECTOR}(d_x, d_y)$          $\triangleright$  Alg. 4.4
18:      if ISLOCALMAX( $E_{mag}, u, v, s_\theta, t_{lo}$ ) then         $\triangleright$  Alg. 4.4
19:         $E_{nms}(u, v) \leftarrow E_{mag}(u, v)$                  $\triangleright$  only keep local maxima
20:  for  $u \leftarrow 1, \dots, M-2$  do
21:    for  $v \leftarrow 1, \dots, N-2$  do
22:      if  $(E_{nms}(u, v) \geq t_{hi}) \wedge (E_{bin}(u, v) = 0)$  then
23:        TRACEANDTHRESHOLD( $E_{nms}, E_{bin}, u, v, t_{lo}$ )       $\triangleright$  Alg. 4.4
24:  return  $E_{bin}$ .
```

approach is to apply the monochromatic operator separately to each of the color channels and subsequently merge the results into a single edge map. However, since edges within the different color channels rarely occur in the same places, the result will usually contain multiple edge marks and undesirable clutter (see [Fig. 4.16](#) on page 114 for an example).

Fortunately, the original grayscale version of the Canny edge detector can be easily adapted to color imagery using the multi-gradient concept described in Section 4.2.1. The only changes required in [Alg. 4.3](#) are the calculation of

Algorithm 4.4 Procedures used in [Alg. 4.3](#) (Canny edge detector).

1: GETORIENTATIONSECTOR(d_x, d_y)
 Returns the orientation sector s_θ for the 2D vector $(d_x, d_y)^\top$. See [Fig. 4.11](#) for an illustration.

2: $\begin{pmatrix} d'_x \\ d'_y \end{pmatrix} \leftarrow \begin{pmatrix} \cos(\pi/8) & -\sin(\pi/8) \\ \sin(\pi/8) & \cos(\pi/8) \end{pmatrix} \cdot \begin{pmatrix} d_x \\ d_y \end{pmatrix}$ \triangleright rotate $\begin{pmatrix} d_x \\ d_y \end{pmatrix}$ by $\frac{\pi}{8}$

3: **if** $d'_y < 0$ **then**

4: $d'_x \leftarrow -d'_x, \quad d'_y \leftarrow -d'_y$ \triangleright mirror to octants $0, \dots, 3$

5: $s_\theta \leftarrow \begin{cases} 0 & \text{if } (d'_x \geq 0) \wedge (d'_x \geq d'_y) \\ 1 & \text{if } (d'_x \geq 0) \wedge (d'_x < d'_y) \\ 2 & \text{if } (d'_x < 0) \wedge (-d'_x < d'_y) \\ 3 & \text{if } (d'_x < 0) \wedge (-d'_x \geq d'_y) \end{cases}$

6: **return** s_θ . \triangleright sector index $s_\theta \in \{0, 1, 2, 3\}$

7: ISLOCALMAX($E_{\text{mag}}, u, v, s_\theta, t_{\text{lo}}$)
 Determines if the gradient magnitude E_{mag} is a local maximum at position (u, v) in direction $s_\theta \in \{0, 1, 2, 3\}$.

8: $m_C \leftarrow E_{\text{mag}}(u, v)$

9: **if** $m_C < t_{\text{lo}}$ **then**

10: **return** false

11: **else**

12: $(m_L, m_R) \leftarrow \begin{cases} (E_{\text{mag}}(u-1, v), E_{\text{mag}}(u+1, v)) & \text{if } s_\theta = 0 \\ (E_{\text{mag}}(u-1, v-1), E_{\text{mag}}(u+1, v+1)) & \text{if } s_\theta = 1 \\ (E_{\text{mag}}(u, v-1), E_{\text{mag}}(u, v+1)) & \text{if } s_\theta = 2 \\ (E_{\text{mag}}(u-1, v+1), E_{\text{mag}}(u+1, v-1)) & \text{if } s_\theta = 3 \end{cases}$

13: **return** $(m_L \leq m_C) \wedge (m_C \geq m_R)$.

14: TRACEANDTHRESHOLD($E_{\text{nms}}, E_{\text{bin}}, u_0, v_0, t_{\text{lo}}$)
 Recursively collects and marks all pixels of an edge that are 8-connected to (u_0, v_0) and exhibit a gradient magnitude above t_{lo} .

15: $E_{\text{bin}}(u_0, v_0) \leftarrow 1$ \triangleright mark (u_0, v_0) as an edge pixel

16: $u_L \leftarrow \max(u_0 - 1, 0)$ \triangleright limit to image bounds

17: $u_R \leftarrow \min(u_0 + 1, M - 1)$

18: $v_T \leftarrow \max(v_0 - 1, 0)$

19: $v_B \leftarrow \min(v_0 + 1, N - 1)$

20: **for** $u \leftarrow u_L, \dots, u_R$ **do**

21: **for** $v \leftarrow v_T, \dots, v_B$ **do**

22: **if** $(E_{\text{nms}}(u, v) \geq t_{\text{lo}}) \wedge (E_{\text{bin}}(u, v) = 0)$ **then**

23: TRACEANDTHRESHOLD($E_{\text{nms}}, E_{\text{bin}}, u, v, t_{\text{lo}}$)

24: **return**

the local gradients and the edge magnitude E_{mag} . The modified procedure is shown in [Alg. 4.5](#) (see Sec. 4.4 for the corresponding Java implementation).

In the pre-processing step, each of the three color channels is individually smoothed by a Gaussian filter of width σ , before calculating the gradient vectors ([Alg. 4.5](#), lines 2–9). As in [Alg. 4.2](#), the color edge magnitude is calculated as the squared local contrast, obtained from the dominant eigenvalue of the structure matrix \mathbf{M} (Eqns. 4.22–4.26). The local gradient vector (E_x, E_y) is calculated from the elements A, B, C , of the matrix \mathbf{M} , as given in Eqn. (4.27). The corresponding steps are found in [Alg. 4.5](#), lines 14–21. The remaining steps, including non-maximum suppression, edge tracing and thresholding, are exactly the same as in [Alg. 4.3](#).

Results from the grayscale and color version of the Canny edge detector are compared in [Figs. 4.14–4.15](#) for varying values of σ and t_{hi} , respectively. Evidently, the color detector gives the more consistent results, particularly at color edges with low intensity difference. In all cases, the gradient magnitude was normalized and the threshold values $t_{\text{hi}}, t_{\text{lo}}$ are given as a percentage of the maximum edge magnitude.

For comparison, [Fig. 4.16](#) shows the results of applying the monochromatic Canny operator separately to each color channel and subsequently merging the edge pixels into a combined edge map, as mentioned at the beginning of this section. We see that this leads to multiple responses and cluttered edges, since maximum gradient positions in the different color channels are generally not collocated.

In summary, the Canny edge detector is superior to simpler schemes based on first-order gradients and global thresholding, in terms of extracting clean and well-located edges that are immediately useful for subsequent processing. The results in [Figs. 4.14–4.15](#) demonstrate that the use of color gives additional improvements over the grayscale approach, since edges with insufficient brightness gradients can still be detected from local color differences. Essential for the good performance of the color Canny edge detector, however, is the reliable calculation of the gradient direction, based on the multi-dimensional squared local contrast formulation given in Section 4.2.3. Quite a few variations of Canny detectors for color images have been proposed in the literature, including the one attributed to Kanade (in [68]), which is similar to the algorithm described here. Other approaches of adapting the Canny detector for color images can be found in [45].

Algorithm 4.5 Canny edge detector for color images. Structure and parameters are identical to the grayscale version in Alg. 4.3 (p. 108). In the algorithm below, edge magnitude (E_{mag}) and orientation (E_x, E_y) are obtained from the gradients of the individual color channels (as described in Sec. 4.2.1).

```

1: COLORCANNYEDGEDETECTOR( $\mathbf{I}, \sigma, t_{\text{hi}}, t_{\text{lo}}$ )
   Input:  $\mathbf{I} = (I_R, I_G, I_B)$ , an RGB color image of size  $M \times N$ ;  $\sigma$ , radius
         of Gaussian filter  $H^{G,\sigma}$ ;  $t_{\text{hi}}$ ,  $t_{\text{lo}}$ , hysteresis thresholds ( $t_{\text{hi}} > t_{\text{lo}}$ ).
         Returns a binary edge image of size  $M \times N$ .
2:  $\bar{I}_R \leftarrow I_R * H^{G,\sigma}$             $\triangleright$  blur components with Gaussian of width  $\sigma$ 
3:  $\bar{I}_G \leftarrow I_G * H^{G,\sigma}$ 
4:  $\bar{I}_B \leftarrow I_B * H^{G,\sigma}$ 
5:  $H_x^\nabla \leftarrow [-0.5 \ 0 \ 0.5]$            $\triangleright x$  gradient filter
6:  $H_y^\nabla \leftarrow [-0.5 \ 0 \ 0.5]^\top$          $\triangleright y$  gradient filter
7:  $\bar{I}_{R,x} \leftarrow \bar{I}_R * H_x^\nabla, \quad \bar{I}_{R,y} \leftarrow \bar{I}_R * H_y^\nabla$ 
8:  $\bar{I}_{G,x} \leftarrow \bar{I}_G * H_x^\nabla, \quad \bar{I}_{G,y} \leftarrow \bar{I}_G * H_y^\nabla$ 
9:  $\bar{I}_{B,x} \leftarrow \bar{I}_B * H_x^\nabla, \quad \bar{I}_{B,y} \leftarrow \bar{I}_B * H_y^\nabla$ 
10:  $(M, N) \leftarrow \text{SIZE}(\mathbf{I})$ 
11: Create maps:
12:    $E_{\text{mag}}, E_{\text{nms}}, E_x, E_y : M \times N \rightarrow \mathbb{R}$ 
13:    $E_{\text{bin}} : M \times N \rightarrow \{0, 1\}$ 
14:   for all image coordinates  $(u, v) \in M \times N$  do
15:      $(r_x, g_x, b_x) \leftarrow (I_{R,x}(u, v), I_{G,x}(u, v), I_{B,x}(u, v))$ 
16:      $(r_y, g_y, b_y) \leftarrow (I_{R,y}(u, v), I_{G,y}(u, v), I_{B,y}(u, v))$ 
17:      $A \leftarrow r_x^2 + g_x^2 + b_x^2, \quad B \leftarrow r_y^2 + g_y^2 + b_y^2$ 
18:      $C \leftarrow r_x \cdot r_y + g_x \cdot g_y + b_x \cdot b_y$ 
19:      $D \leftarrow [(A - B)^2 + 4C^2]^{1/2}$ 
20:      $E_{\text{mag}}(u, v) \leftarrow [0.5 \cdot (A + B + D)]^{1/2}$            $\triangleright \sqrt{\lambda_1}$ , Eqn. (4.26)
21:      $E_x(u, v) \leftarrow A - B + D, \quad E_y(u, v) \leftarrow 2C$            $\triangleright \mathbf{x}_1$ , Eqn. (4.27)
22:      $E_{\text{nms}}(u, v) \leftarrow 0, \quad E_{\text{bin}}(u, v) \leftarrow 0$ 
23:   for  $u \leftarrow 1, \dots, M - 2$  do
24:     for  $v \leftarrow 1, \dots, N - 2$  do
25:        $d_x \leftarrow E_x(u, v), \quad d_y \leftarrow E_y(u, v)$ 
26:        $s \leftarrow \text{GETORIENTATIONSECTOR}(d_x, d_y)$            $\triangleright$  see Alg. 4.4
27:       if IsLOCALMAX( $E_{\text{mag}}, u, v, s, t_{\text{lo}}$ ) then           $\triangleright$  see Alg. 4.4
28:          $E_{\text{nms}}(u, v) \leftarrow E_{\text{mag}}(u, v)$ 
29:   for  $u \leftarrow 1, \dots, M - 2$  do
30:     for  $v \leftarrow 1, \dots, N - 2$  do
31:       if  $(E_{\text{nms}}(u, v) \geq t_{\text{hi}} \wedge E_{\text{bin}}(u, v) = 0)$  then
32:         TRACEANDTHRESHOLD( $E_{\text{nms}}, E_{\text{bin}}, u, v, t_{\text{lo}}$ )  $\triangleright$  see Alg. 4.4
33:   return  $E_{\text{bin}}$ .
```

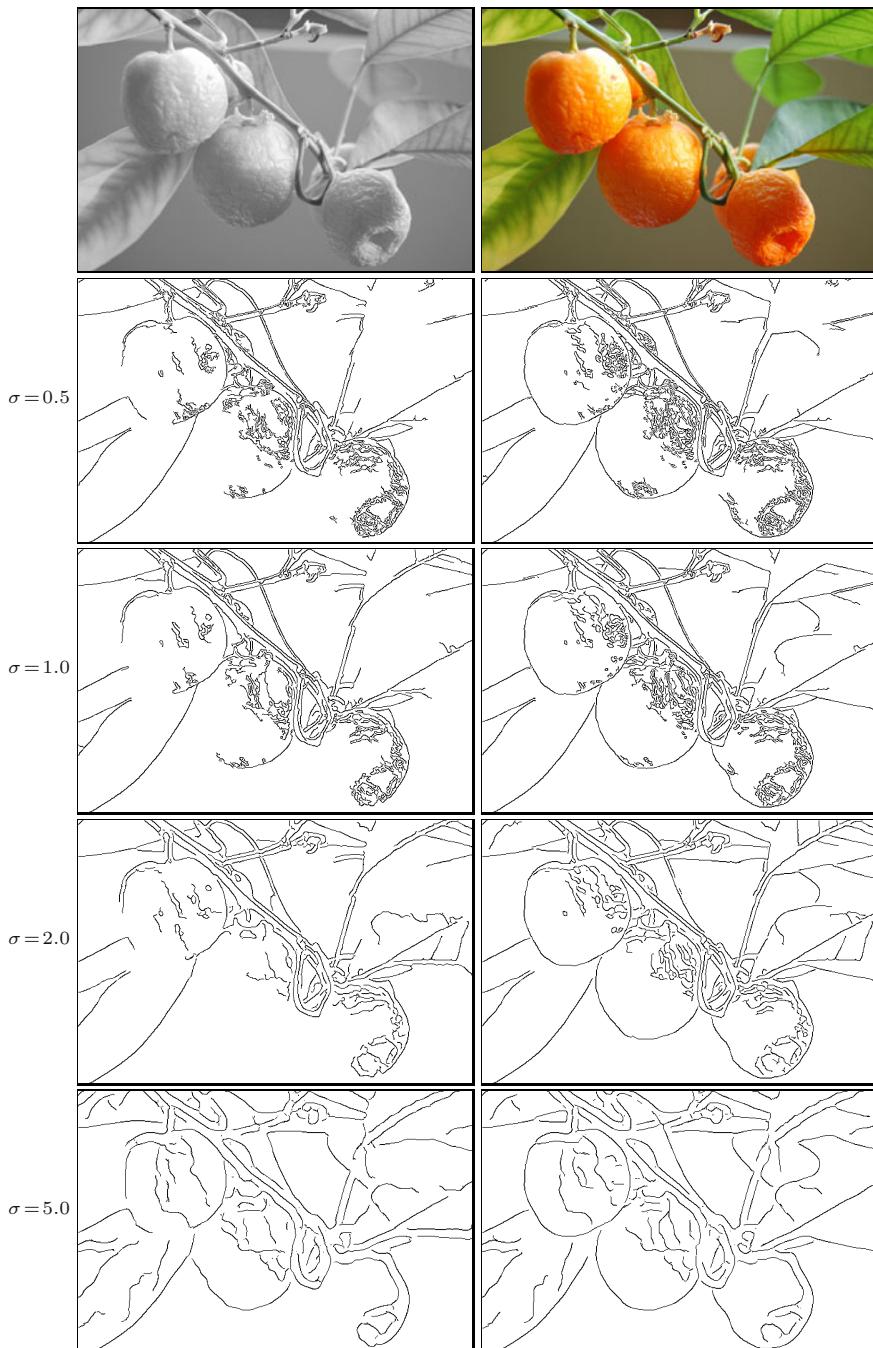


Figure 4.14 Results from grayscale Canny operator (left) and color Canny operator (right) for different values of σ ($t_{hi} = 20\%$, $t_{lo} = 5\%$ of max. edge magnitude).

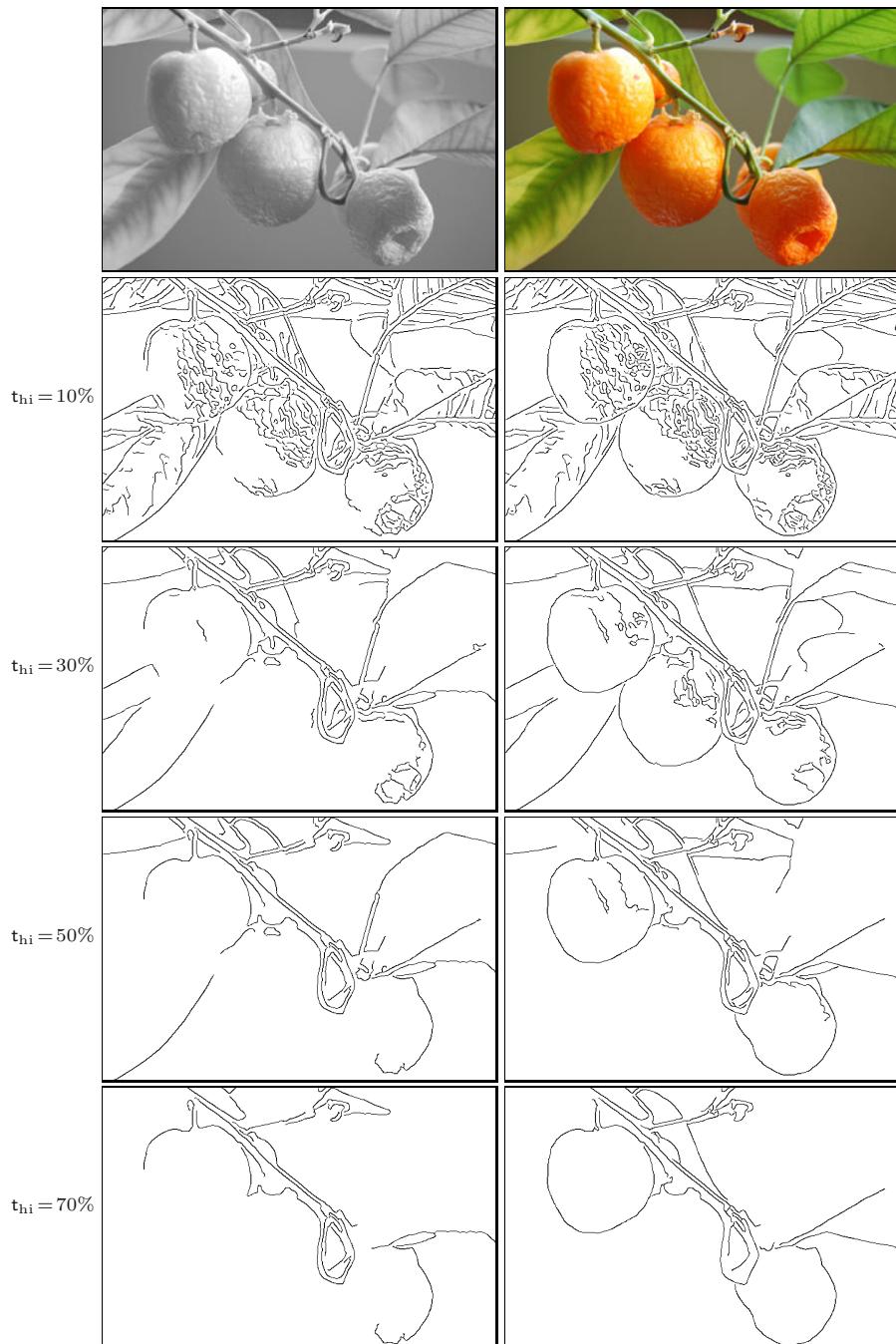


Figure 4.15 Results from grayscale Canny operator (left) and color Canny operator (right) for different threshold values t_{hi} , given in % of max. edge magnitude ($t_{lo} = 5\%$, $\sigma = 2.0$).

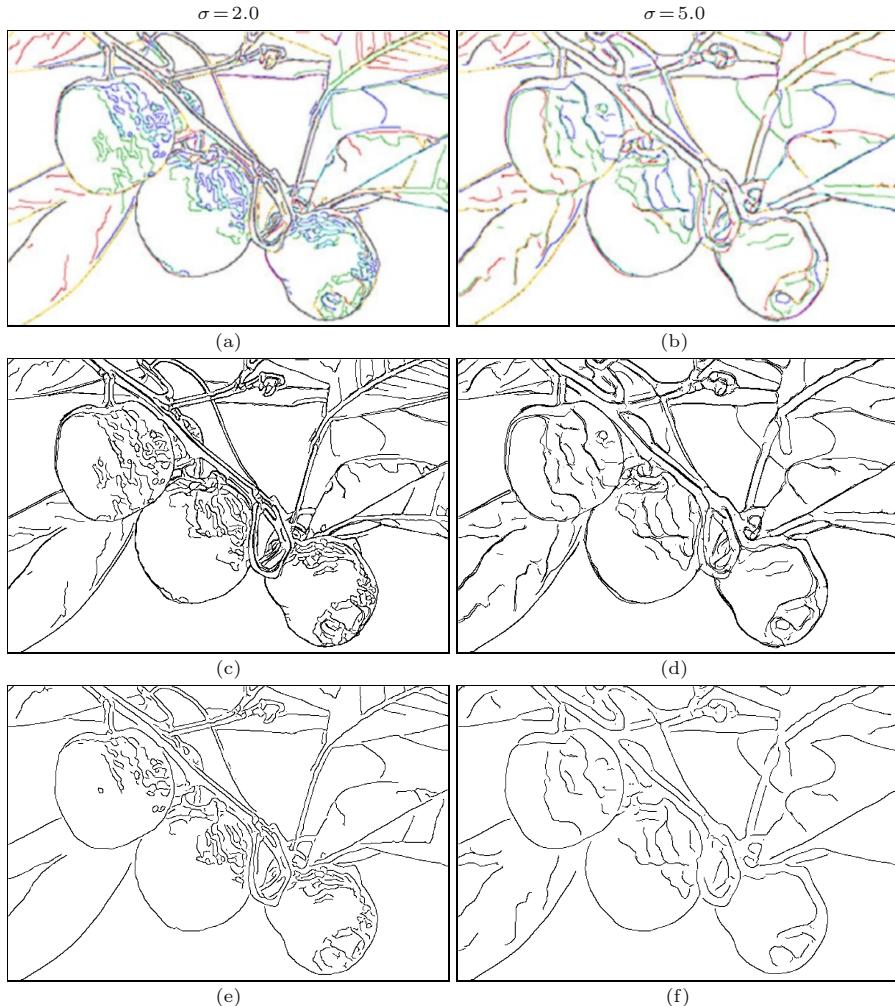


Figure 4.16 Scalar vs. vector-based color Canny operator. Results from the scalar Canny operator applied separately to each color channel (a, b). Channel edges are shown in corresponding colors, with mixed colors indicating that edge points were detected in multiple channels (e.g., yellow marks overlapping points from the red and the green channel). A black pixel indicates that an edge point was detected in all three color channels. Channel edges combined into a joint edge map (c, d). For comparison, the result of the vector-based color Canny operator (e, f). Common parameter settings are $\sigma = 2.0$ and 5.0 , $t_{hi} = 20\%$, $t_{lo} = 5\%$ of max. edge magnitude.

4.4 Implementation

The following Java implementations of the color edge detection algorithms described in this chapter can be found in the source code section¹⁶ of the book's accompanying website.

ColorEdgeDetector (class)

This is the abstract super-class of all concrete color edge detection classes described below. It provides the following public methods:

`FloatProcessor getEdgeMagnitude ()`

Returns the resulting edge magnitude map $E(u, v)$ as a `FloatProcessor` object.

`FloatProcessor getEdgeOrientation ()`

Returns the resulting edge orientation map $\Phi(u, v)$ as a `FloatProcessor` object, with values in the range $[-\pi, \pi]$.

MonochromaticEdgeDetector (class)

This sub-class of class `ColorEdgeDetector` implements the monochromatic color edge detector described in [Alg. 4.1](#).

`MonochromaticEdgeDetector (ColorProcessor cp)`

Constructor, creates a new detector for the RGB color image `cp` and performs edge detection. Results can be retrieved using the methods `getEdgeMagnitude()` and `getEdgeOrientation()` defined by super-class `ColorEdgeDetector`.

DiZzenzoCumaniEdgeDetector (class)

This sub-class of class `ColorEdgeDetector` implements the “Di Zenzo/Cumani-style” multi-gradient color edge detector described in [Alg. 4.2](#).

`DiZzenzoCumaniEdgeDetector (ColorProcessor cp)`

Constructor, creates a new detector for the RGB color image `cp` and performs edge detection. Results can be retrieved using the methods `getEdgeMagnitude()` and `getEdgeOrientation()` defined by super-class `ColorEdgeDetector`.

CannyEdgeDetector (class)

This sub-class of class `ColorEdgeDetector` implements the Canny edge detector for color and grayscale images described in [Algs. 4.3–4.5](#).

¹⁶ See package `imagingbook.colorededge`.

`CannyEdgeDetector (ImageProcessor ip)`

Constructor, creates a new detector for the RGB color image `cp` and performs edge detection, using default parameters settings.

`CannyEdgeDetector (ImageProcessor ip, double gSigma, double`

`hiThr, double loThr)`

Constructor with explicit parameters `gSigma` (σ , default 2.0), `hiThr` (t_{hi} , default 20.0), `loThr` (t_{lo} , default 5.0).

`CannyEdgeDetector (ImageProcessor ip, Parameters params)`

Alternative constructor accepting a configurable parameter object of type `CannyEdgeDetector.Parameters` (see the example in Prog. 4.1).

`ByteProcessor getEdgeBinary ()`

Returns the binary edge map $E_{bin}(u, v)$ as a `ByteProcessor` object.

`List<List<java.awt.Point>> getEdgeTraces ()`

Returns a list of connected edge traces. Each trace is again a list of `java.awt.Point` objects.

Program 4.1 shows a simple example for the use of the class `CannyEdgeDetector` in the context of an ImageJ plugin.

4.5 Other color edge operators

The idea of using a vector field model in the context of color edge detection was first presented by Di Zenzo [36], who suggested finding the orientation of maximum change by maximizing $S(\dot{x}, \theta)$ in Eqn. (4.19) over the angle θ . Later Cumani [33,34] proposed directly using the eigenvalues and -vectors of the local structure matrix \mathbf{M} (Eqn. (4.22)) for calculating edge strength and orientation. He also proposed using the zero-crossings of the second-order gradients along the direction of maximum contrast to precisely locate edges, which is a general problem with first-order techniques. Both Di Zenzo and Cumani used only the dominant eigenvalue, indicating the edge strength perpendicular to the edge (if an edge existed at all), and then discarded the smaller eigenvalue proportional to the edge strength in the perpendicular (i.e., tangential) direction. Real edges only exist where the larger eigenvalue is considerably greater than the smaller one. If both eigenvalues have similar values, this indicates that the local image surface exhibits change in all directions, which is not typically true at an edge but quite characteristic of flat, noisy regions and corners. One solution therefore is to use the difference between the eigenvalues, $\lambda_1 - \lambda_2$, to quantify edge strength [115].

Other types of color edge detectors exist and have been used successfully, including techniques based on vector order statistics and color difference vectors. Excellent overviews of the various color edge detection schemes can be

```
1 import ij.ImagePlus;
2 import ij.plugin.filter.PlugInFilter;
3 import ij.process.ByteProcessor;
4 import ij.process.FloatProcessor;
5 import ij.process.ImageProcessor;
6
7 import java.awt.Point;
8 import java.util.List;
9
10
11 public class Canny_Edge_Demo implements PlugInFilter {
12
13     public int setup(String arg0, ImagePlus imp) {
14         return DOES_ALL + NO_CHANGES;
15     }
16
17     public void run(ImageProcessor ip) {
18
19         CannyEdgeDetector.Parameters params =
20             new CannyEdgeDetector.Parameters();
21
22         params.gSigma = 3.0f; //  $\sigma$  of Gaussian
23         params.hiThr = 20.0f; // 20% of max. edge magnitude
24         params.loThr = 5.0f; // 5% of max. edge magnitude
25
26         CannyEdgeDetector detector = new CannyEdgeDetector(ip, params);
27
28         FloatProcessor eMag = detector.getEdgeMagnitude();
29         FloatProcessor eOrt = detector.getEdgeOrientation();
30         ByteProcessor eBin = detector.getEdgeBinary();
31         List<List<Point>> edgeTraces = detector.getEdgeTraces();
32
33         (new ImagePlus("Canny Edges", eBin)).show();
34
35         // process edge detection results ...
36
37     }
38 }
```

Program 4.1 Use of the `CannyEdgeDetector` class in an ImageJ plugin. A parameter object (`params`) is created in line 20, subsequently configured (in lines 22–24) and finally used to construct a `CannyEdgeDetector` object in line 26. Note that edge detection is performed within the constructor method. Lines 28–31 demonstrate how different types of edge detection results can be retrieved. The binary edge map `eBin` is displayed in line 33. As indicated in the `setup()` method, this plugin works with any type of image.

found in [152] and [69, Ch. 6].

5

Edge-Preserving Smoothing Filters

Noise reduction in images is a common objective in image processing, not only for producing pleasing results for human viewing but also to facilitate easier extraction of meaningful information in subsequent steps, for example, in segmentation or feature detection. Simple smoothing filters, such as the Gaussian filter¹ and the filters discussed in Chapter 3 of this volume effectively perform low-pass filtering and thus remove high-frequency noise. However, they also tend to suppress high-rate intensity variations that are part of the original signal, thereby destroying image structures that are visually important. The filters described in this chapter are “edge preserving” in the sense that they change their smoothing behavior adaptively depending upon the local image structure. In general, maximum smoothing is performed over “flat” (uniform) image regions, while smoothing is reduced near or across edge-like structures, typically characterized by high intensity gradients.

In the following, three classical types of edge preserving filters are presented, which are largely based on different strategies. The *Kuwahara-type* filters described in Section 5.1 partition the filter kernel into smaller sub-kernels and select the most “homogeneous” of the underlying image regions for calculating the filter’s result. In contrast, the *Bilateral* filter in Section 5.2 uses the differences between pixel *values* to control how much each individual pixel in the filter region contributes to the local average. Pixels which are similar to the current center pixel contribute strongly, while highly different pixels add little to the result. Thus, in a sense, the Bilateral filter is a non-homogeneous linear

¹ See Sec. 5.2 in Vol. 1 [20].

filter with a convolution kernel that is adaptively controlled by the local image content. Finally, the *anisotropic diffusion* filters in Section 5.3 iteratively smooth the image similar to the process of thermal diffusion, using the image gradient to block the local diffusion at edges and similar structures.

5.1 Kuwahara-type filters

The filters described in this section are all based on a similar concept that has its early roots in the work of Kuwahara et al. [71]. Although many variations have been proposed by other authors, we summarize them here under the term ‘‘Kuwahara-type’’ to indicate their origin and algorithmic similarities.

In principle, these filters work by calculating the mean and variance within neighboring image regions and selecting the mean value of the most ‘‘homogeneous’’ region, i. e., the one with the smallest variance, to replace the original (center) pixel. For this purpose, the filter region \mathcal{R} is divided into K partially overlapping subregions $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_K$. At every image position (u, v) , the *mean* μ_k and the *variance* σ_k^2 of each subregion \mathcal{R}_k are calculated from the corresponding pixel values in I as

$$\mu_k(u, v) = \frac{1}{|\mathcal{R}_k|} \cdot \sum_{(i,j) \in \mathcal{R}_k} I(u+i, v+j) = \frac{1}{n_k} \cdot S_{1,k}(u, v), \quad (5.1)$$

$$\sigma_k^2(u, v) = \frac{1}{|\mathcal{R}_k|} \cdot \sum_{(i,j) \in \mathcal{R}_k} (I(u+i, v+j) - \mu_k(u, v))^2 \quad (5.2)$$

$$= \frac{1}{|\mathcal{R}_k|} \cdot \left[S_{2,k}(u, v) - \frac{S_{1,k}^2(u, v)}{|\mathcal{R}_k|} \right], \quad (5.3)$$

for $k = 1, \dots, K$, with

$$S_{1,k}(u, v) = \sum_{(i,j) \in \mathcal{R}_k} I(u+i, v+j), \quad (5.4)$$

$$S_{2,k}(u, v) = \sum_{(i,j) \in \mathcal{R}_k} I^2(u+i, v+j). \quad (5.5)$$

The mean (μ) of the subregion with the smallest variance (σ^2) is selected as the new image value, that is,

$$I'(u, v) \leftarrow \mu_{k'}(u, v), \quad \text{with } k' = \operatorname{argmin}_{k=1, \dots, K} \sigma_k^2(u, v). \quad (5.6)$$

This process is summarized in [Alg. 5.1](#). The subregion structure originally proposed by Kuwahara et al. [71] is shown in [Fig. 5.1 \(a\)](#) for a 3×3 filter ($r = 1$). It uses four square subregions of size $(r+1) \times (r+1)$ that overlap at the center. In general, the size of the whole filter is $(2r+1) \times (2r+1)$. Note that this filter

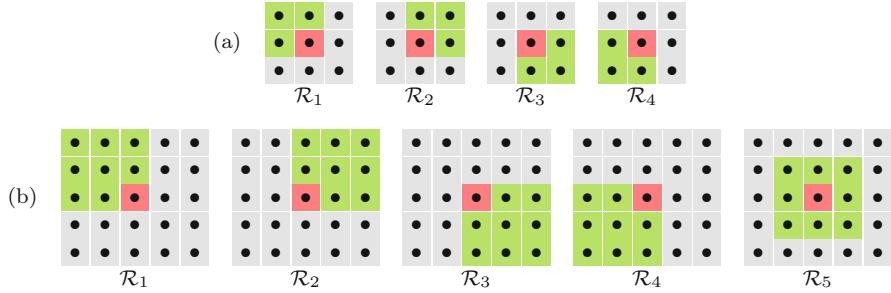


Figure 5.1 Subregion structures for Kuwahara-type filters. The original *Kuwahara-Hachimura* filter (a) considers four square, overlapping subregions [71]. The center pixel (red) is included in all subregions $\mathcal{R}_1, \dots, \mathcal{R}_4$. The *Tomita-Tsuji* filter (b) follows the same construction principle but consists of five subregions whose side length is odd [127]. In both types of filters, all subregions have the same size; $r = 1$ (a) and $r = 2$ (b), respectively.

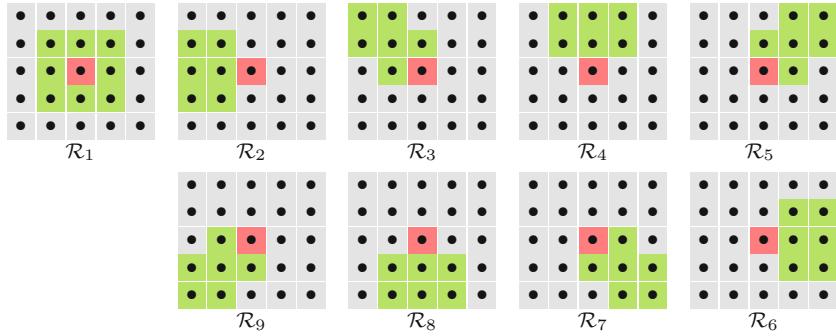


Figure 5.2 Subregions for the 5×5 ($r = 2$) *Nagao-Matsuyama* filter [89]. Note that the centered subregion (\mathcal{R}_1) has a different size than the remaining subregions ($\mathcal{R}_2, \dots, \mathcal{R}_9$).

does have a centered subregion, which means that the center pixel is always replaced by the mean of one of the neighboring regions, even if it had perfectly fit the surrounding values. Thus the filter always performs a spatial shift, which introduces jitter and banding artifacts in regions of smooth intensity change. This effect is reduced with the filter proposed by Tomita and Tsuji [127], which is similar but includes a fifth subregion at its center (Fig. 5.1 (b)). Filters of arbitrary size can be built by simply scaling the corresponding structure. In the case of the *Tomita-Tsuji* filter, the side length of the subregions should be odd.

Note that replacing a pixel value by the mean of a square neighborhood is equivalent to linear filtering with a simple box kernel, which is not an optimal smoothing operator. To reduce the artifacts caused by the square subregions, alternative filter structures have been proposed, such as the 5×5 *Nagao-Matsuyama* filter [89] shown in Fig. 5.2.

Algorithm 5.1 General form of the *Kuwahara-type* filter for arbitrary filter size and subregion structure. The parameter r specifies the radius of the filter with the total size $(2r+1) \times (2r+1)$.

```

1: KUWAHARAFILTER( $I, r$ )
   Input:  $I$ , a grayscale image of size  $M \times N$ ;  $r$ , filter radius ( $r \geq 1$ ).
   Returns a new (filtered) image of size  $M \times N$ .
2:  $\bar{\mathcal{R}} \leftarrow [-r, r] \times [-r, r]$      $\triangleright$  comp. filter region of size  $(2r+1) \times (2r+1)$ 
3: for  $k = 1$  to  $K$  do            $\triangleright$  define  $K$  subregions  $\mathcal{R}_1, \dots, \mathcal{R}_K$ 
4:    $\mathcal{R}_k \leftarrow \{\mathbf{p}_{k,1}, \dots, \mathbf{p}_{k,n_k}\}$ , with  $\mathbf{p}_{k,j} \in \bar{\mathcal{R}}$ 
5:    $(M, N) \leftarrow \text{SIZE}(I)$ 
6:    $I' \leftarrow \text{DUPLICATE}(I)$ 
7:   for all image coordinates  $(u, v) \in M \times N$  do
8:      $\sigma_{\min}^2 \leftarrow \infty, \mu_{\min} \leftarrow 0$ 
9:     for  $k = 1$  to  $K$  do
10:       $(\sigma^2, \mu) \leftarrow \text{EVALSUBREGION}(I, \mathcal{R}_k, u, v)$ 
11:      if  $\sigma^2 < \sigma_{\min}^2$  then
12:         $\sigma_{\min}^2 \leftarrow \sigma^2$ 
13:         $\mu_{\min} \leftarrow \mu$ 
14:      $I'(u, v) \leftarrow \mu_{\min}$ 
15:   return  $I'$ .


---


16: EVALSUBREGION( $I, \mathcal{R}, u, v$ )
17:    $n \leftarrow \text{Size}(\mathcal{R})$ 
18:    $S_1 \leftarrow 0, S_2 \leftarrow 0$ 
19:   for all  $(i, j) \in \mathcal{R}$  do
20:      $a \leftarrow I(u+i, v+j)$ 
21:      $S_1 \leftarrow S_1 + a$             $\triangleright$  Eqn. (5.4)
22:      $S_2 \leftarrow S_2 + a^2$           $\triangleright$  Eqn. (5.5)
23:    $\sigma^2 \leftarrow (S_2 - S_1^2/n)/n$      $\triangleright$  variance of subregion  $\mathcal{R}$ , see Eqn. (5.1)
24:    $\mu \leftarrow S_1/n$                   $\triangleright$  mean of subregion  $\mathcal{R}$ , see Eqn. (5.3)
25:   return  $(\sigma^2, \mu)$ .

```

If all subregions are of identical size $|\mathcal{R}_k| = n$, the quantities

$$\sigma_k^2(u, v) \cdot n = S_{2,k}(u, v) - S_{1,k}^2(u, v)/n \quad \text{and} \quad (5.7)$$

$$\sigma_k^2(u, v) \cdot n^2 = S_{2,k}(u, v) \cdot n - S_{1,k}^2(u, v) \quad (5.8)$$

can be used to measure the amount of variation within the corresponding subregion. Both expressions require calculating one multiplication less for each pixel than the “real” variance σ_k^2 in Eqn. (5.3).



Figure 5.3 Noisy test image with selected details, as used for the subsequent color examples.

Moreover, if all subregions have the same *shape* (such as the filters in Fig. 5.1), additional optimizations are possible that substantially improve the performance. In this case, the local mean and variance need to be calculated only once over a fixed neighborhood for each image position. This type of filter can be efficiently implemented by using a set of pre-calculated maps for the local variance and mean values, as described in Alg. 5.2. As before, the parameter r specifies the radius of the composite filter, which is of size $(2r+1) \times (2r+1)$. The individual subregions are of size $(r+1) \times (r+1)$; for example, $r = 2$ for the 5×5 filter shown in Fig. 5.1 (b).

All these filters tend to generate banding artifacts in smooth image regions due to erratic spatial displacements, which become worse with increasing filter size. If a centered subregion is used (such as \mathcal{R}_5 in Fig. 5.1 or \mathcal{R}_1 in Fig. 5.2), one could reduce this effect by applying a threshold (t_σ) to select any off-center subregion \mathcal{R}_k *only* if its variance is significantly smaller than the variance of the center region \mathcal{R}_1 (see Alg. 5.2, line 13).

5.1.1 Application to color images

While all of the above filters were originally designed for grayscale images, they are easily modified to work with color images. We only need to specify how to calculate the variance and mean for any subregion; the decision and replacement mechanisms then remain the same.

Given an RGB color image $\mathbf{I} = (I_R, I_G, I_B)$ with a subregion \mathcal{R}_k , we can calculate the local mean and variance for each color channel as

$$\begin{aligned}\mu_{k,R}(\mathbf{I}, u, v) &= \mu_k(I_R, u, v), & \sigma_{k,R}^2(\mathbf{I}, u, v) &= \sigma_k^2(I_R, u, v), \\ \mu_{k,G}(\mathbf{I}, u, v) &= \mu_k(I_G, u, v), & \sigma_{k,G}^2(\mathbf{I}, u, v) &= \sigma_k^2(I_G, u, v), \\ \mu_{k,B}(\mathbf{I}, u, v) &= \mu_k(I_B, u, v), & \sigma_{k,B}^2(\mathbf{I}, u, v) &= \sigma_k^2(I_B, u, v),\end{aligned}\quad (5.9)$$

with $\mu_k()$, $\sigma_k^2()$ as defined in Eqns. (5.1) and (5.3), respectively. Analogous to the grayscale case, each pixel is then replaced by the average color in the

Algorithm 5.2 Fast Kuwahara-type (Tomita-Tsuji) filter with fixed subregion structure. The filter uses five square subregions of size $(r+1) \times (r+1)$, with a composite filter of $(2r+1) \times (2r+1)$, as shown in Fig. 5.1 (b). The purpose of the variance threshold t_σ is to reduce banding effects in smooth image regions (typically $t_\sigma = 5, \dots, 50$ for 8-bit images).

```

1:  FASTKUWAHARAFILTER( $I, r, t_\sigma$ )
   Input:  $I$ , a grayscale image of size  $M \times N$ ;
           $r$ , filter radius ( $r \geq 1$ );  $t_\sigma$ , variance threshold.
   Returns a new (filtered) image of size  $M \times N$ .
2:   $(M, N) \leftarrow \text{SIZE}(I)$ 
3:  Create maps:
    $\mathsf{S} : M \times N \rightarrow \mathbb{R}$             $\triangleright$  local variance  $\mathsf{S}(u, v) \equiv n \cdot \sigma^2(I, u, v)$ 
    $\mathsf{A} : M \times N \rightarrow \mathbb{R}$             $\triangleright$  local mean  $\mathsf{A}(u, v) \equiv \mu(I, u, v)$ 
4:   $d_- \leftarrow (r \div 2) - r$             $\triangleright$  subregions' left/top position
5:   $d_+ \leftarrow d_- + r$             $\triangleright$  subregions' right/bottom position
6:  for all image coordinates  $(u, v) \in M \times N$  do
7:     $(s, \mu) \leftarrow \text{EVALSQUARESUBREGION}(I, u, v, d_-, d_+)$ 
8:     $\mathsf{S}(u, v) \leftarrow s$ 
9:     $\mathsf{A}(u, v) \leftarrow \mu$ 
10:    $n \leftarrow (r+1)^2$             $\triangleright$  fixed subregion size
11:    $I' \leftarrow \text{DUPLICATE}(I)$ 
12:   for all image coordinates  $(u, v) \in M \times N$  do
13:      $s_{\min} \leftarrow \mathsf{S}(u, v) - t_\sigma \cdot n$             $\triangleright$  variance of center region
14:      $\mu_{\min} \leftarrow \mathsf{A}(u, v)$             $\triangleright$  mean of center region
15:     for all  $p \leftarrow d_-, \dots, d_+$  do
16:       for all  $q \leftarrow d_-, \dots, d_+$  do
17:         if  $\mathsf{S}(u+p, v+q) < s_{\min}$  then
18:            $s_{\min} \leftarrow \mathsf{S}(u+p, v+q)$ 
19:            $\mu_{\min} \leftarrow \mathsf{A}(u+p, v+q)$ 
20:      $I'(u, v) \leftarrow \mu_{\min}$ 
21:   return  $I'$ .


---


22: EVALSQUARESUBREGION( $I, u, v, d_-, d_+$ )
23:    $S_1 \leftarrow 0, S_2 \leftarrow 0$ 
24:   for  $i \leftarrow d_-, \dots, d_+$  do
25:     for  $j \leftarrow d_-, \dots, d_+$  do
26:        $a \leftarrow I(u+i, v+j)$ 
27:        $S_1 \leftarrow S_1 + a$             $\triangleright$  Eqn. (5.4)
28:        $S_2 \leftarrow S_2 + a^2$             $\triangleright$  Eqn. (5.5)
29:      $s \leftarrow S_2 - S_1^2/n$             $\triangleright$  subregion variance ( $s \equiv n \cdot \sigma^2$ )
30:      $\mu \leftarrow S_1/n$             $\triangleright$  subregion mean ( $\mu$ )
31:   return  $(s, \mu)$ .

```

Algorithm 5.3 Color version of the *Kuwahara-type* filter (adapted from Alg. 5.1). The algorithm uses the definition in Eqn. (5.11) for the total variance σ^2 in the subregion \mathcal{R} (see line 24). The vector μ (calculated in line 25) is the average color of the subregion.

```

1: KUWAHARAFILTERCOLOR( $I, r$ )
   Input:  $I$ , an RGB image of size  $M \times N$ ;  $r$ , filter radius ( $r \geq 1$ ).
   Returns a new (filtered) color image of size  $M \times N$ .
2:  $\bar{\mathcal{R}} \leftarrow [-r, r] \times [-r, r]$      $\triangleright$  comp. filter region of size  $(2r+1) \times (2r+1)$ 
3: for  $k = 1$  to  $K$  do            $\triangleright$  define  $K$  subregions  $\mathcal{R}_1, \dots, \mathcal{R}_K$ 
4:    $\mathcal{R}_k \leftarrow \{p_{k,1}, \dots, p_{k,n_k}\}$ , with  $p_{k,j} \in \bar{\mathcal{R}}$ 
5:    $(M, N) \leftarrow \text{SIZE}(I)$ 
6:    $I' \leftarrow \text{DUPLICATE}(I)$ 
7:   for all image coordinates  $(u, v) \in M \times N$  do
8:      $\sigma_{\min}^2 \leftarrow \infty$ ,  $\mu_{\min} \leftarrow \mathbf{0}$ 
9:     for  $k = 1$  to  $K$  do
10:       $(\sigma^2, \mu) \leftarrow \text{EVALSUBREGION}(I, \mathcal{R}_k, u, v)$ 
11:      if  $\sigma^2 < \sigma_{\min}^2$  then
12:         $\sigma_{\min}^2 \leftarrow \sigma^2$ 
13:         $\mu_{\min} \leftarrow \mu$ 
14:      $I'(u, v) \leftarrow \mu_{\min}$ 
15:   return  $I'$ .


---


16: EVALSUBREGION( $I, \mathcal{R}, u, v$ )
17:    $n \leftarrow \text{Size}(\mathcal{R})$ 
18:    $S_1 \leftarrow \mathbf{0}$ ,  $S_2 \leftarrow \mathbf{0}$             $\triangleright S_1, S_2 \in \mathbb{R}^3$ 
19:   for all  $(i, j) \in \mathcal{R}$  do
20:      $a \leftarrow I(u+i, v+j)$             $\triangleright a \in \mathbb{R}^3$ 
21:      $S_1 \leftarrow S_1 + a$ 
22:      $S_2 \leftarrow S_2 + a^2$             $\triangleright a^2 = a \cdot a$  (dot product)
23:      $S \leftarrow (S_2 - S_1^2 \cdot \frac{1}{n}) \cdot \frac{1}{n}$             $\triangleright S = (\sigma_R^2, \sigma_G^2, \sigma_B^2)$ 
24:    $\sigma^2 \leftarrow \Sigma S$             $\triangleright \sigma^2 = \sigma_R^2 + \sigma_G^2 + \sigma_B^2$ , total color variance in  $\mathcal{R}$ 
25:    $\mu \leftarrow \frac{1}{n} \cdot S_1$             $\triangleright \mu \in \mathbb{R}^3$ , average color in subregion  $\mathcal{R}$ 
26:   return  $(\sigma^2, \mu)$ .
```

subregion with the smallest variance, i. e.,

$$I'(u, v) \leftarrow \begin{pmatrix} \mu_{k'}(I_R, u, v) \\ \mu_{k'}(I_G, u, v) \\ \mu_{k'}(I_B, u, v) \end{pmatrix}, \quad \text{where } k' = \underset{k=1, \dots, K}{\operatorname{argmin}} \sigma_k^2(I, u, v). \quad (5.10)$$

The overall variance $\sigma_k^2(I, u, v)$, used to determine k' in Eqn. (5.10), can be defined in different ways, for example, as the sum of the variances in the

individual color channels, that is

$$\sigma_k^2(\mathbf{I}, u, v) = \sigma_k^2(I_R, u, v) + \sigma_k^2(I_G, u, v) + \sigma_k^2(I_B, u, v). \quad (5.11)$$

This is sometimes called the “total variance” and the resulting filter process is summarized in [Alg. 5.3](#). Color examples produced with this algorithm are shown in [Figs. 5.4](#) and [5.5](#).

Alternatively (as in [55]), one could define the combined color variance as the norm of the *color covariance matrix*² for the subregion \mathcal{R}_k ,

$$\Sigma_k(\mathbf{I}, u, v) = \begin{pmatrix} \sigma_{k,RR} & \sigma_{k,RG} & \sigma_{k,RB} \\ \sigma_{k,GR} & \sigma_{k,GG} & \sigma_{k,GB} \\ \sigma_{k,BR} & \sigma_{k,BG} & \sigma_{k,BB} \end{pmatrix}, \quad (5.12)$$

with

$$\sigma_{k,pq} = \frac{\sum_{(i,j) \in \mathcal{R}_k} [I_p(u+i, v+j) - \mu_k(I_p, u, v)] \cdot [I_q(u+i, v+j) - \mu_k(I_q, u, v)]}{|\mathcal{R}_k|},$$

for all possible color pairs $(p, q) \in \{R, G, B\}^2$. Note that $\sigma_{k,pp} = \sigma_{k,p}^2$ and $\sigma_{k,pq} = \sigma_{k,qp}$, and thus the matrix Σ_k is symmetric and only 6 of its 9 entries need to be calculated. The (Frobenius) *norm* of the 3×3 color covariance matrix is defined as

$$\|\Sigma_k(\mathbf{I}, u, v)\|_2 = \left(\sum_{\substack{p=1 \\ R,G,B}} \sum_{\substack{q=1 \\ R,G,B}} (\sigma_{k,pq})^2 \right)^{1/2}. \quad (5.13)$$

Note that the total variance in Eqn. (5.11)—which is simpler to calculate than this norm—is equivalent to the *trace* of the covariance matrix Σ_k .

Since each pixel of the filtered image is calculated as the *mean* (i. e., a linear combination) of a set of original color pixels, the results depend on the color space used, as discussed in [Section 3.1.2](#).

5.2 Bilateral filter

Traditional linear smoothing filters operate by convolving the image with a kernel, whose coefficients act as weights for the corresponding image pixels and only depend on the spatial distance from the center coordinate. Pixels close to the filter center are typically given larger weights while pixels at a greater distance carry smaller weights. Thus the convolution kernel effectively encodes the closeness of the underlying pixels in space. In the following, a filter whose weights depend only on the distance in the spatial domain is called a *domain*

² See [Appendix C.2](#) (p. 330) for details.

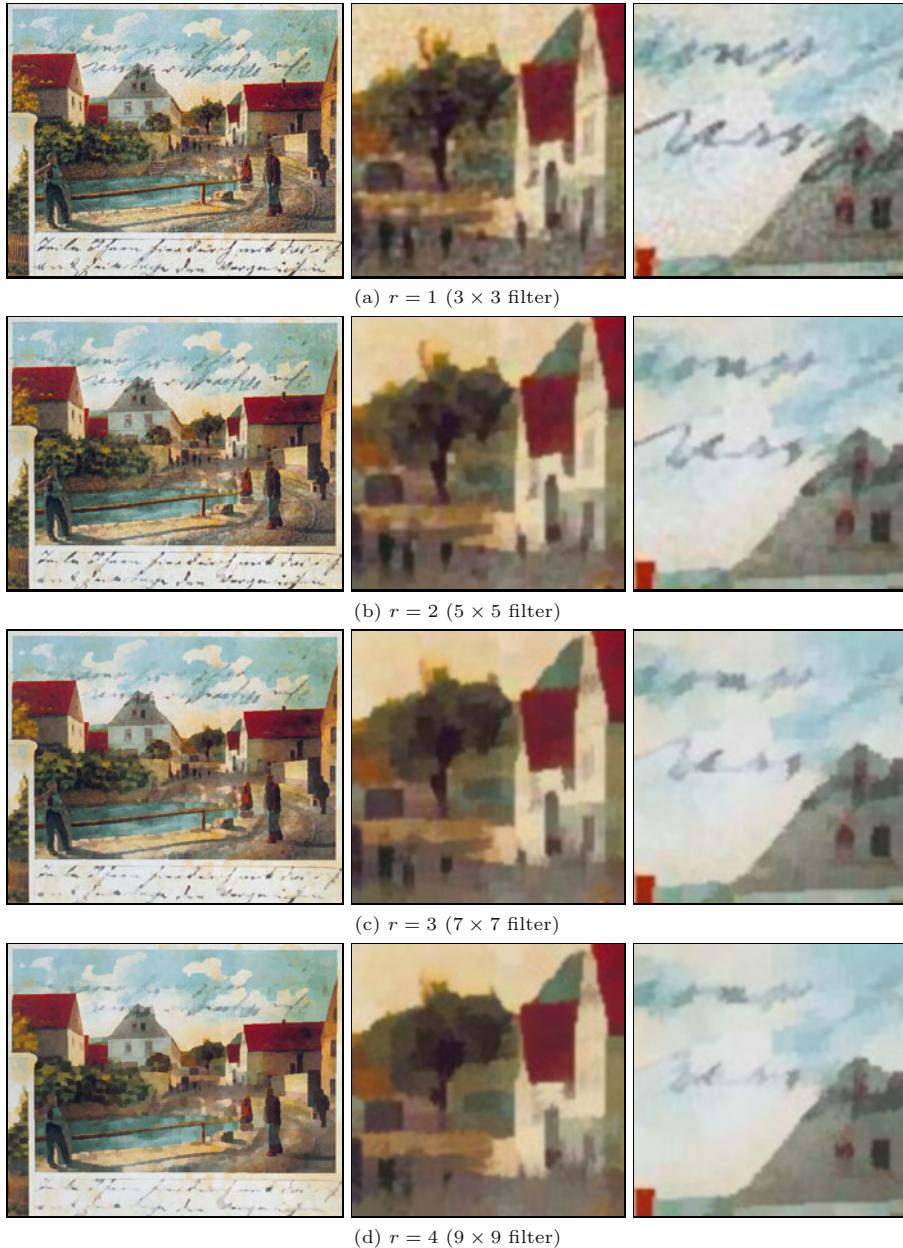


Figure 5.4 Kuwahara-type (*Tomita-Tsuji*) filter—color example using the variance definition in Eqn. (5.11). The filter radius is varied from $r = 1$ (filter size 3×3) to $r = 4$ (filter size 9×9). Original image in Fig. 5.3.

(a) 5×5 Tomita-Tsuji filter ($r = 2$)(b) 5×5 Nagao-Matsuyama filter

Figure 5.5 Color versions of the *Tomita-Tsuji* and *Nagao-Matsuyama* filter. Both filters are of size 5×5 and use the variance definition in Eqn. (5.11). Results are visually similar, but in general the *Nagao-Matsuyama* filter is slightly less destructive on diagonal structures. Original image in Fig. 5.3.

filter. To make smoothing filters less destructive on edges, a typical strategy is to exclude individual pixels from the filter operation or to reduce the weight of their contribution if they are very dissimilar *in value* to the pixel found at the center position. This operation too can be formulated as a filter, but this time the kernel coefficients depend only upon the differences in pixel *values* or *range*. Therefore, this is called a *range filter*, as explained in more detail below. The idea of the *Bilateral* filter, proposed by Tomasi and Manduchi in [126], is to *combine* both domain and range filtering into a common, edge-preserving smoothing filter.

5.2.1 Domain vs. range filters

In a 2D linear filter (or “convolution”) operation,³

$$I'(u, v) \leftarrow \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} I(u+m, v+n) \cdot H(m, n) \quad (5.14)$$

³ See also Eqn. (5.5) in Vol. 2 [20, p. 97].

$$= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i, j) \cdot H(i - u, j - v), \quad (5.15)$$

every new pixel value $I'(u, v)$ is the weighted average of the original image pixels I in a certain neighborhood, with the weights specified by the elements of the filter kernel H . The weight assigned to each pixel only depends on its spatial position relative to the current center coordinate (u, v) . In particular, $H(0, 0)$ specifies the weight of the center pixel $I(u, v)$, and $H(m, n)$ is the weight assigned to a pixel displaced by (m, n) from the center. Since only the spatial image coordinates are relevant, such a filter is called a *domain filter*. Obviously, ordinary filters as we know them are *all* domain filters.

Although the idea may appear strange at first, one could also apply a linear filter to the pixel *values* or *range* of an image in the form

$$I'_r(u, v) \leftarrow \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i, j) \cdot H_r(I(i, j) - I(u, v)). \quad (5.16)$$

The contribution of each pixel is specified by the function H_r and depends on the difference between its own *value* $I(i, j)$ and the value at the current center pixel $I(u, v)$. The operation in Eqn. (5.16) is called a *range filter*, where the spatial position of a contributing pixel is irrelevant and only the difference in values is considered. For a given position (u, v) , all surrounding image pixels $I(i, j)$ with the same value contribute equally to the result $I'_r(u, v)$. Consequently, the application of a *range filter* has no *spatial* effect upon the image—in contrast to a *domain filter*, no blurring or sharpening will occur. However, a global *range filter* by itself is of little use, since it combines pixels from the entire image and only changes the intensity or color map of the image, equivalent to a non-linear, image-dependent point operation.

The key idea behind the Bilateral filter is to *combine* domain filtering (Eqn. (5.15)) *and* range filtering (Eqn. (5.16)) in the form

$$I'(u, v) \leftarrow \frac{1}{W_{u,v}} \cdot \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i, j) \cdot \underbrace{H_d(i-u, j-v) \cdot H_r(I(i, j) - I(u, v))}_{w_{i,j}}, \quad (5.17)$$

where H_d , H_r are the domain and range kernels, respectively, $w_{i,j}$ are the composite weights, and

$$W_{u,v} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} w_{i,j} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} H_d(i-u, j-v) \cdot H_r(I(i, j) - I(u, v)) \quad (5.18)$$

is the (position-dependent) sum of the weights $w_{i,j}$ used to normalize the combined filter kernel.

In this form, the scope of range filtering is constrained to the spatial neighborhood defined by the domain kernel H_d . At a given filter position (u, v) , the weight $w_{i,j}$ assigned to each contributing pixel depends upon (1) its spatial position relative to (u, v) , and (2) the similarity of its pixel value to the value at the center position (u, v) . In other words, the resulting pixel is the weighted average of pixels that are nearby *and* similar to the original pixel. In a flat image region, where most surrounding pixels having values similar to the center pixel, the Bilateral filter acts as a conventional smoothing filter, controlled only by the domain kernel H_d . However, when placed near a step edge or on an intensity ridge, only those pixels are included in the smoothing process that are similar in value to the center pixel, thus avoiding blurring the edges. The Bilateral filter is space-variant and non-linear, thus no general statement can be made about its effects in the spectral domain.

If the domain kernel H_d has a limited radius K , or size $(2K+1) \times (2K+1)$, the Bilateral filter defined in Eqn. (5.17) can be written as

$$I'(u, v) \leftarrow \frac{\sum_{\substack{i= \\ u-K}}^{u+K} \sum_{\substack{j= \\ v-K}}^{v+K} I(i, j) \cdot H_d(i-u, j-v) \cdot H_r(I(i, j) - I(u, v))}{\sum_{\substack{i= \\ u-K}}^{u+K} \sum_{\substack{j= \\ v-K}}^{v+K} H_d(i-u, j-v) \cdot H_r(I(i, j) - I(u, v))} \quad (5.19)$$

$$= \frac{\sum_{\substack{m= \\ -K}}^K \sum_{\substack{n= \\ -K}}^K I(u+m, v+n) \cdot H_d(m, n) \cdot H_r(I(u+m, v+n) - I(u, v))}{\sum_{\substack{m= \\ -K}}^K \sum_{\substack{n= \\ -K}}^K H_d(m, n) \cdot H_r(I(u+m, v+n) - I(u, v))} \quad (5.20)$$

(by substituting $(i-u) \rightarrow m$ and $(j-v) \rightarrow n$). The effective, space variant filter kernel for the image I at position (u, v) then is

$$\bar{H}_{I,u,v}(i, j) = \frac{H_d(i, j) \cdot H_r(I(u+i, v+j) - I(u, v))}{\sum_{\substack{m= \\ -K}}^K \sum_{\substack{n= \\ -K}}^K H_d(m, n) \cdot H_r(I(u+m, v+n) - I(u, v))}, \quad (5.21)$$

for $-K \leq i, j \leq K$, otherwise $\bar{H}_{I,u,v}(i, j) = 0$. This quantity specifies the contribution of the original image pixels $I(u+i, v+j)$ to the resulting new pixel value $I'(u, v)$.

5.2.2 Bilateral filter with Gaussian kernels

A special (but common) case is the use of Gaussian kernels for both the domain and the range parts of the Bilateral filter. The discrete 2D Gaussian *domain kernel* of width σ_d is defined as

$$H_d^{G,\sigma_d}(m, n) = \frac{1}{2\pi\sigma_d^2} \cdot e^{-\frac{\rho^2}{2\sigma_d^2}} = \frac{1}{2\pi\sigma_d^2} \cdot e^{-\frac{m^2+n^2}{2\sigma_d^2}} \quad (5.22)$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma_d} \cdot e^{-\frac{m^2}{2\sigma_d^2}} \right) \cdot \left(\frac{1}{\sqrt{2\pi}\sigma_d} \cdot e^{-\frac{n^2}{2\sigma_d^2}} \right), \quad (5.23)$$

for $m, n \in \mathbb{Z}$. It has its maximum at the center ($m = n = 0$) and falls off smoothly and isotropically with increasing radius $\rho = \sqrt{m^2 + n^2}$; for $\rho > 3.5\sigma_d$, $H_d^{G,\sigma_d}(m, n)$ is practically zero. The factorization in Eqn. (5.23) indicates that the Gaussian 2D kernel can be separated into one-dimensional Gaussians, allowing for a more efficient implementation.⁴ The constant factors $1/(\sqrt{2\pi}\sigma_d)$ can be omitted in practice, since the Bilateral filter requires individual normalization at each image position (Eqn. (5.18)).

Similarly, the corresponding *range filter kernel* is defined as a (continuous) one-dimensional Gaussian of width σ_r ,

$$H_r^{G,\sigma_r}(x) = \frac{1}{\sqrt{2\pi}\sigma_r} \cdot e^{-\frac{x^2}{2\sigma_r^2}}, \quad (5.24)$$

for $x \in \mathbb{R}$. The constant factor $1/(\sqrt{2\pi}\sigma_r)$ may again be omitted and the resulting composite filter (Eqn. (5.17)) can thus be written as

$$\begin{aligned} I'(u, v) &\leftarrow \frac{1}{W_{u,v}} \cdot \sum_{\substack{i=-K \\ u-K}}^{u+K} \sum_{\substack{j=-K \\ v-K}}^{v+K} I(i, j) \cdot H_d^{G,\sigma_d}(i-u, j-v) \cdot H_r^{G,\sigma_r}(I(i, j) - I(u, v)) \\ &= \frac{1}{W_{u,v}} \cdot \sum_{\substack{m=-K \\ -K}}^K \sum_{\substack{n=-K \\ -K}}^K I(u+m, v+n) \cdot H_d^{G,\sigma_d}(m, n) \cdot H_r^{G,\sigma_r}(I(u+m, v+n) - I(u, v)) \\ &= \frac{1}{W_{u,v}} \cdot \sum_{\substack{m=-K \\ -K}}^K \sum_{\substack{n=-K \\ -K}}^K I(u+m, v+n) \cdot e^{-\frac{m^2+n^2}{2\sigma_d^2}} \cdot e^{-\frac{(I(u+m, v+n) - I(u, v))^2}{2\sigma_r^2}}, \end{aligned} \quad (5.25)$$

with $K = \lceil 3.5 \cdot \sigma_d \rceil$ and $W_{u,v} = \sum_{\substack{m=-K \\ -K}}^K \sum_{\substack{n=-K \\ -K}}^K e^{-\frac{m^2+n^2}{2\sigma_d^2}} \cdot e^{-\frac{(I(u+m, v+n) - I(u, v))^2}{2\sigma_r^2}}$.

For 8-bit grayscale images, with pixel values in the range [0, 255], the width of the range kernel is typically set to $\sigma_r = 10, \dots, 50$. The width of the domain kernel (σ_d) depends on the desired amount of spatial smoothing. [Algorithm 5.4](#)

⁴ See also Vol. 1, Sec. 5.3.3 [20, p. 113].

Algorithm 5.4 Bilateral filter with Gaussian kernels (grayscale version).

```

1: BILATERALFILTERGRAY( $I, \sigma_d, \sigma_r$ )
   Input:  $I$ , a grayscale image of size  $M \times N$ ;  $\sigma_d$ , width of the 2D Gaussian
         domain kernel;  $\sigma_r$ , width of the 1D Gaussian range kernel;
         Returns a new filtered image of size  $M \times N$ .
2:  $(M, N) \leftarrow \text{SIZE}(I)$ 
3:  $K \leftarrow \lceil 3.5 \cdot \sigma_d \rceil$                                  $\triangleright$  width of domain filter kernel
4:  $I' \leftarrow \text{DUPLICATE}(I)$ 
5: for all image coordinates  $(u, v) \in M \times N$  do
6:    $S \leftarrow 0$                                           $\triangleright$  sum of weighted pixel values
7:    $W \leftarrow 0$                                           $\triangleright$  sum of weights
8:    $a \leftarrow I(u, v)$                                       $\triangleright$  center pixel value
9:   for  $m \leftarrow -K, \dots, K$  do
10:    for  $n \leftarrow -K, \dots, K$  do
11:       $b \leftarrow I(u + m, v + n)$                           $\triangleright$  off-center pixel value
12:       $w_d \leftarrow e^{-\frac{m^2 + n^2}{2\sigma_d^2}}$            $\triangleright$  domain weight
13:       $w_r \leftarrow e^{-\frac{(a-b)^2}{2\sigma_r^2}}$            $\triangleright$  range weight
14:       $w \leftarrow w_d \cdot w_r$                              $\triangleright$  composite weight
15:       $S \leftarrow S + w \cdot b$ 
16:       $W \leftarrow W + w$ 
17:    $I'(u, v) \leftarrow \frac{1}{W} \cdot S$ 
18: return  $I'$ .

```

gives a summary of the steps involved in Bilateral filtering for grayscale images. Figures 5.6–5.8 show the effective, space-variant filter kernels (see Eqn. (5.21)) and the results of applying a Bilateral filter with Gaussian domain and range kernels in different situations.

5.2.3 Application to color images

Linear smoothing filters are typically used on color images by separately applying the same filter to the individual color channels. As discussed in Section 3.1, this is legitimate if a suitable working color space is used to avoid the introduction of unnatural intensity and chromaticity values. Thus, for the domain-part of the Bilateral filter, the same considerations apply as for any linear smoothing filter. However, as described below, the Bilateral filter as a whole cannot be implemented by filtering the color channels separately.

In the *range* part of the filter, the weight assigned to each contributing

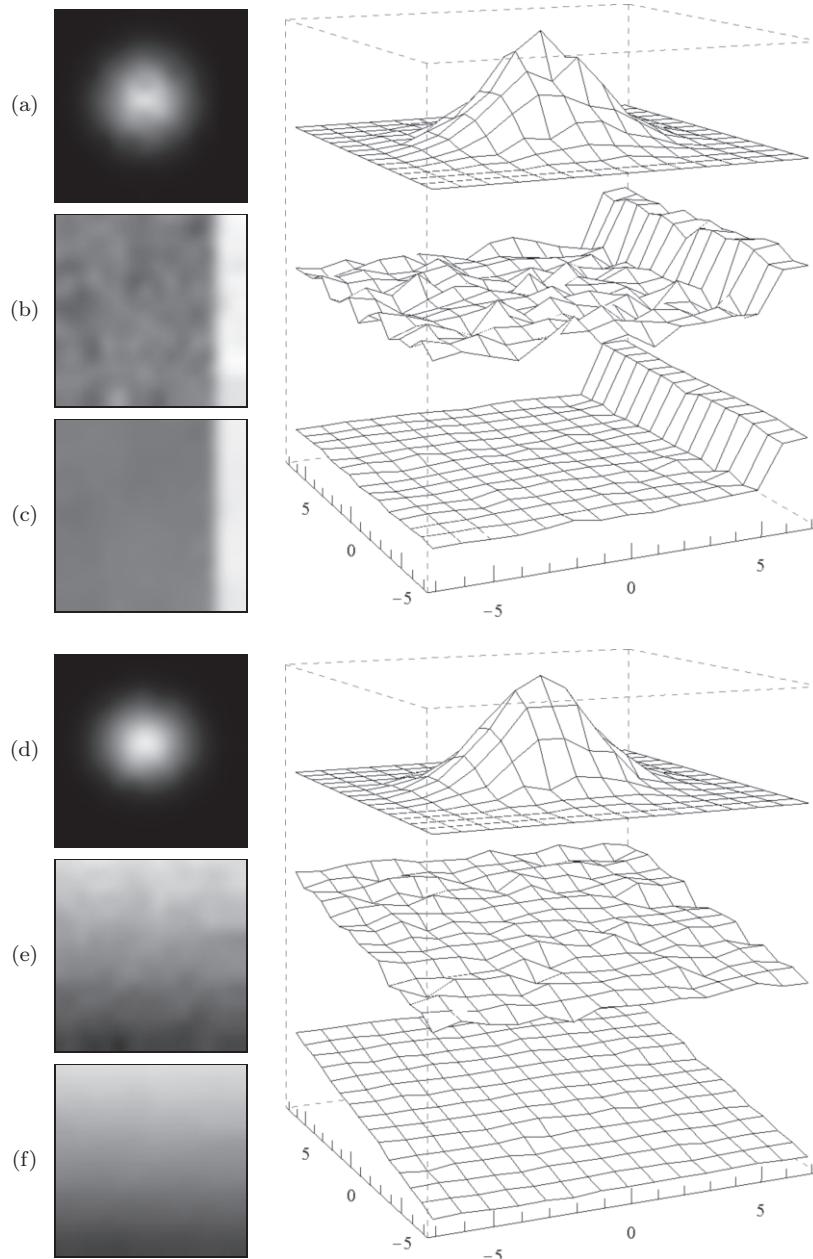


Figure 5.6 Bilateral filter example with Gaussian kernels ($\sigma_d = 2.0$, $\sigma_r = 50$). Response with the filter centered in a flat image region (a–c) and on a linear ramp (d–f). Effective filter kernels (a, d), original image data (b, e), filtered image data (c, f). The size of the domain kernel H_d is 15×15 .

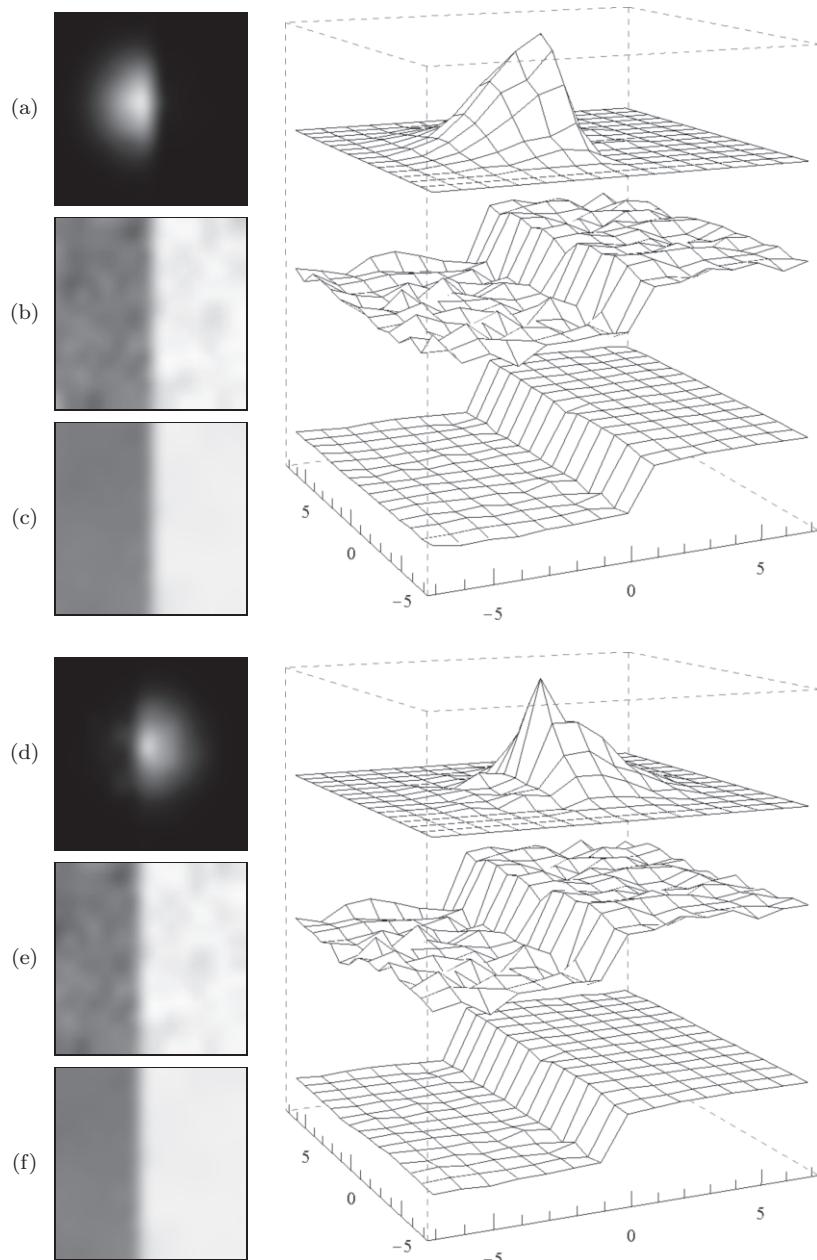


Figure 5.7 Bilateral filter example with Gaussian kernels ($\sigma_d = 2.0$, $\sigma_r = 50$). Response with the filter positioned left to a vertical step edge (a–c) and right to a vertical step edge (d–f). Effective filter kernels (a, d), original image data (b, e), filtered image data (c, f). The size of the domain kernel H_d is 15×15 .

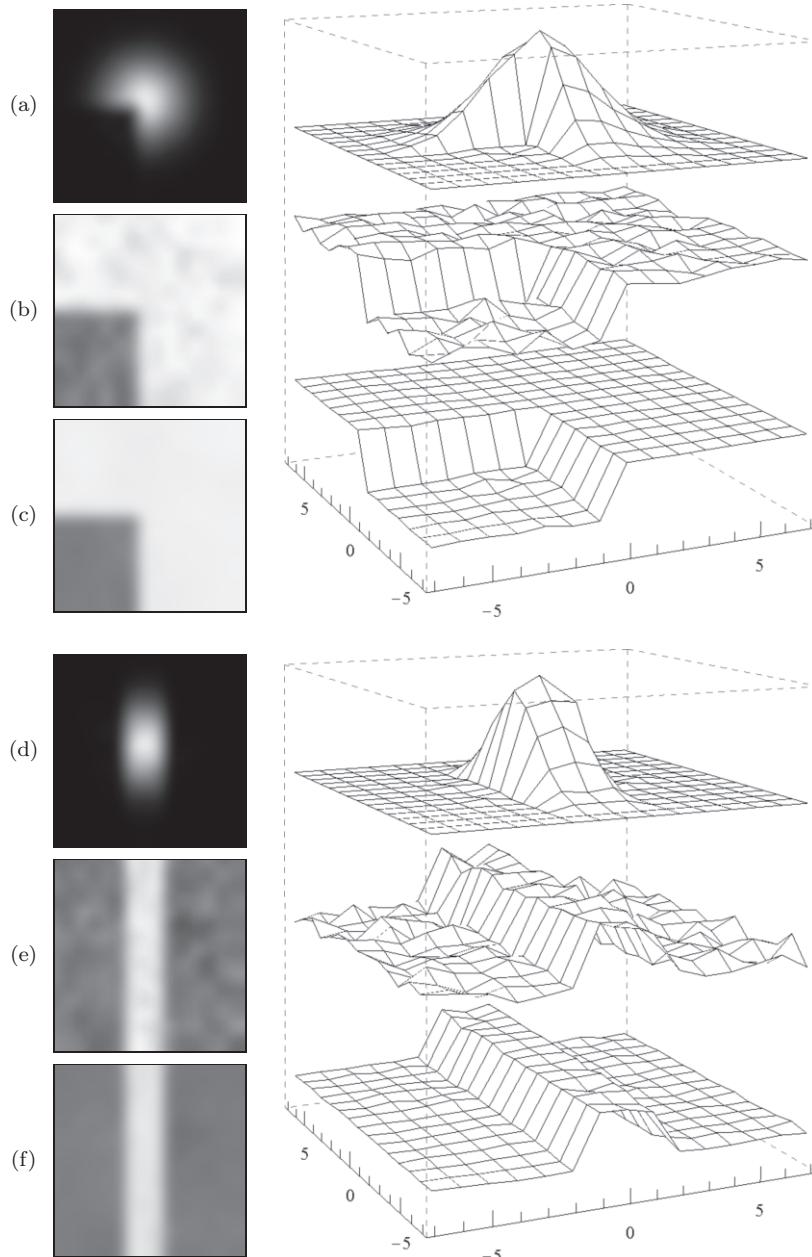


Figure 5.8 Bilateral filter example with Gaussian kernels ($\sigma_d = 2.0$, $\sigma_r = 50$). Response with the filter centered at a corner (a–c) and on a vertical ridge (d–f). Effective filter kernels (a, d), original image data (b, e), filtered image data (c, f). The size of the domain kernel H_d is 15×15 .

pixel depends on its difference to the value of the center pixel. Given a suitable distance measure $\text{dist}(\mathbf{a}, \mathbf{b})$ between two color vectors \mathbf{a}, \mathbf{b} , the Bilateral filter in Eqn. (5.17) can be easily modified for a color image \mathbf{I} to

$$\mathbf{I}'(u, v) \leftarrow \frac{1}{W_{u,v}} \cdot \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} \mathbf{I}(i, j) \cdot H_d(i-u, j-v) \cdot H_r(\text{dist}(\mathbf{I}(i, j), \mathbf{I}(u, v))),$$

$$\text{with } W_{u,v} = \sum_{i,j} H_d(i-u, j-v) \cdot H_r(\text{dist}(\mathbf{I}(i, j), \mathbf{I}(u, v))). \quad (5.26)$$

It is common to use one of the popular norms for measuring color distances, such as the L_1 , L_2 (Euclidean) or the L_∞ (maximum) norms, for example,

$$\text{dist}_1(\mathbf{a}, \mathbf{b}) \equiv \frac{1}{3} \cdot \|\mathbf{a} - \mathbf{b}\|_1 = \frac{1}{3} \cdot \sum_k |a_k - b_k|, \quad (5.27)$$

$$\text{dist}_2(\mathbf{a}, \mathbf{b}) \equiv \frac{1}{\sqrt{3}} \cdot \|\mathbf{a} - \mathbf{b}\|_2 = \frac{1}{\sqrt{3}} \cdot (\sum_k (a_k - b_k)^2)^{1/2}, \quad (5.28)$$

$$\text{dist}_\infty(\mathbf{a}, \mathbf{b}) \equiv \|\mathbf{a} - \mathbf{b}\|_\infty = \max_k |a_k - b_k|. \quad (5.29)$$

The normalizing factors $1/3$ and $1/\sqrt{3}$ in Eqns. (5.27–5.28) are necessary to obtain results comparable in magnitude to those of grayscale images when using the same range kernel H_r .⁵ Of course in most color spaces, none of these norms measures perceived color difference.⁶ However, the distance function itself is not really critical since it only affects the relative *weights* assigned to the contributing color pixels. Regardless of the distance function used, the resulting chromaticities are linear, convex combinations of the original colors in the filter region, and thus the choice of the working color space is more important (see Sec. 3.1).

The process of Bilateral filtering for color images (again using Gaussian kernels for the domain and the range filters) is summarized in Alg. 5.5. The Euclidean distance (L_2 norm) is used to measure the difference between color vectors. The examples in Fig. 5.9 were produced using sRGB as the color working space.

5.2.4 Separable implementation

The Bilateral filter, if implemented in the way described in Algs. 5.4–5.5, is computationally expensive, with a time complexity of $\mathcal{O}(K^2)$ for each pixel, where K denotes the radius of the filter. Some mild speedup is possible by tabulating the domain and range kernels, but the performance of the brute-force implementation is usually not acceptable for practical applications. In [104] a

⁵ For example, with 8-bit RGB color images, $\text{dist}(\mathbf{a}, \mathbf{b})$ is always in the range $[0, 255]$.

⁶ The CIELAB and CIELUV color spaces are designed to use the Euclidean distance (L_2 norm) as a valid metric for color difference.



Figure 5.9 Bilateral filter—color example. A Gaussian kernel with $\sigma_d = 2.0$ (kernel size 15×15) is used for the domain part of the filter; working color space is sRGB. The width of the range filter is varied from $\sigma_r = 10$ to 100. Original image in Fig. 5.3.

Algorithm 5.5 Bilateral filter with Gaussian kernels (color version). The function $\text{dist}(\mathbf{a}, \mathbf{b})$ measures the distance between two colors \mathbf{a} and \mathbf{b} , for example, using the L₂ norm (line 5, see Eqns. (5.27–5.29) for other options).

```

1: BILATERALFILTERCOLOR( $\mathbf{I}$ ,  $\sigma_d$ ,  $\sigma_r$ )
   Input:  $\mathbf{I}$ , an RGB color image of size  $M \times N$ ;  $\sigma_d$ , width of the 2D
         Gaussian domain kernel;  $\sigma_r$ , width of the 1D Gaussian range kernel;
         Returns a new filtered color image of size  $M \times N$ .
2:  $(M, N) \leftarrow \text{SIZE}(I)$ 
3:  $K \leftarrow \lceil 3.5 \cdot \sigma_d \rceil$                                  $\triangleright$  width of domain filter kernel
4:  $\mathbf{I}' \leftarrow \text{DUPLICATE}(\mathbf{I})$ 
5:  $\text{dist}(\mathbf{a}, \mathbf{b}) := \frac{1}{\sqrt{3}} \cdot \|\mathbf{a} - \mathbf{b}\|_2$        $\triangleright$  color distance (e.g., Euclidean)
6: for all image coordinates  $(u, v) \in (M \times N)$  do
7:    $\mathbf{S} \leftarrow \mathbf{0}$                                       $\triangleright \mathbf{S} \in \mathbb{R}^3$ , sum of weighted pixel vectors
8:    $W \leftarrow 0$                                           $\triangleright$  sum of pixel weights (scalar)
9:    $\mathbf{a} \leftarrow \mathbf{I}(u, v)$                                 $\triangleright \mathbf{a} \in \mathbb{R}^3$ , center pixel vector
10:  for  $m \leftarrow -K, \dots, K$  do
11:    for  $n \leftarrow -K, \dots, K$  do
12:       $\mathbf{b} \leftarrow \mathbf{I}(u + m, v + n)$            $\triangleright \mathbf{b} \in \mathbb{R}^3$ , off-center pixel vector
13:       $w_d \leftarrow e^{-\frac{m^2+n^2}{2\sigma_d^2}}$             $\triangleright$  domain weight
14:       $w_r \leftarrow e^{-\frac{(\text{dist}(\mathbf{a}, \mathbf{b}))^2}{2\sigma_r^2}}$         $\triangleright$  range weight
15:       $w \leftarrow w_d \cdot w_r$                        $\triangleright$  composite weight
16:       $\mathbf{S} \leftarrow \mathbf{S} + w \cdot \mathbf{b}$ 
17:       $W \leftarrow W + w$ 
18:     $\mathbf{I}'(u, v) \leftarrow \frac{1}{W} \cdot \mathbf{S}$ 
19:  return  $\mathbf{I}'$ .

```

separable *approximation* of the Bilateral filter is proposed that brings about a significant performance increase. In this implementation, a *one-dimensional* Bilateral filter is first applied in the horizontal direction only, which uses one-dimensional domain and range kernels H_d and H_r , respectively, and produces the intermediate image I^\triangleright , that is,

$$I^\triangleright(u, v) \leftarrow \frac{\sum_{m=-K}^K I(u+m, v) \cdot H_d(m) \cdot H_r(I(u+m, v) - I(u, v))}{\sum_{m=-K}^K H_d(m) \cdot H_r(I(u+m, v) - I(u, v))}. \quad (5.30)$$

In the next (second) pass, the *same* filter is applied to the intermediate result I^\triangleright in the vertical direction to obtain the final result I' as

$$I'(u, v) \leftarrow \frac{\sum_{n=-K}^K I^\triangleright(u, v+n) \cdot H_d(n) \cdot H_r(I^\triangleright(u, v+n) - I^\triangleright(u, v))}{\sum_{n=-K}^K H_d(n) \cdot H_r(I^\triangleright(u, v+n) - I^\triangleright(u, v))}, \quad (5.31)$$

using the same 1D domain kernels H_d and H_r as in Eqn. (5.30). Alternatively, the vertical filter could be applied first, followed by the horizontal filter. [Algorithm 5.6](#) shows a direct implementation of the separable Bilateral filter for grayscale images, using Gaussian kernels for both the domain and the range parts of the filter. Again, the extension to color images is straightforward (see Eqn. (5.26) and Exercise 5.5).

For the *horizontal* part of the filter, the effective space-variant kernel at image position (u, v) is

$$\bar{H}_{I,u,v}^\triangleright(i) = \frac{H_d(i) \cdot H_r(I(u+i, v) - I(u, v))}{\sum_{m=-K}^K H_d(m) \cdot H_r(I(u+m, v) - I(u, v))}, \quad (5.32)$$

for $-K \leq i \leq K$ (zero otherwise). Analogously, the effective kernel for the *vertical* filter is

$$\bar{H}_{I,u,v}^\nabla(j) = \frac{H_d(j) \cdot H_r(I(u, v+j) - I(u, v))}{\sum_{n=-K}^K H_d(n) \cdot H_r(I(u, v+n) - I(u, v))} \quad (5.33)$$

again for $-K \leq j \leq K$. For the combined filter, the effective 2D kernel at position (u, v) then is

$$\bar{H}_{I,u,v}(i, j) = \begin{cases} \bar{H}_{I,u,v}^\triangleright(i) \cdot \bar{H}_{I^\triangleright,u,v}^\nabla(j) & \text{for } -K \leq i, j \leq K, \\ 0 & \text{otherwise,} \end{cases} \quad (5.34)$$

where I is the original image and I^\triangleright denotes the intermediate image, as defined in Eqn. (5.30).

As intended, the advantage of the separable filter is performance. For a given kernel radius K , the original (non-separable) requires $\mathcal{O}(K^2)$ calculations for each pixel, while the separable version takes only $\mathcal{O}(K)$ steps. This means a substantial saving and speed increase, particularly for large filters.

[Figure 5.10](#) shows the response of the 1D separable Bilateral filter in various situations. The results produced by the separable filter are very similar to those obtained with the original filter in [Figs. 5.6–5.8](#), partly because the local structures in these images are parallel to the coordinate axes. In general, the results are different, as demonstrated for a diagonal step edge in [Fig. 5.11](#). The effective filter kernels are shown in [Fig. 5.11 \(g, h\)](#) for an anchor point positioned on the bright side of the edge. It can be seen that, while the kernel of the full

Algorithm 5.6 Separable Bilateral filter with Gaussian kernels (adapted from Alg. 5.4). The input image is processed in two passes. In each pass, a 1D kernel is applied in horizontal or vertical direction, respectively (see Eqns. (5.30–5.31)). Note that this is only an approximation of the non-separable Bilateral filter.

```

1: BILATERALFILTERGRAYSEPARABLE( $I, \sigma_d, \sigma_r$ )
   Input:  $I$ , a grayscale image of size  $M \times N$ ;  $\sigma_d$ , width of the 2D Gaussian
   domain kernel;  $\sigma_r$ , width of the 1D Gaussian range kernel;
   Returns a new filtered image of size  $M \times N$ .
2:  $(M, N) \leftarrow \text{SIZE}(I)$ 
3:  $K \leftarrow \lceil 3.5 \cdot \sigma_d \rceil$                                  $\triangleright$  width of domain filter kernel
4:  $I^\triangleright \leftarrow \text{DUPLICATE}(I)$ 
5: for all coordinates  $(u, v) \in M \times N$  do            $\triangleright$  Pass 1 (horizontal)
6:    $a \leftarrow I(u, v)$ 
7:    $S \leftarrow 0, W \leftarrow 0$ 
8:   for  $m \leftarrow -K, \dots, K$  do
9:      $b \leftarrow I(u + m, v)$ 
10:     $w_d \leftarrow e^{-\frac{m^2}{2\sigma_d^2}}$            $\triangleright$  domain weight  $H_d(m)$ 
11:     $w_r \leftarrow e^{-\frac{(a-b)^2}{2\sigma_r^2}}$        $\triangleright$  range weight  $H_r$ 
12:     $w \leftarrow w_d \cdot w_r$                        $\triangleright$  composite weight
13:     $S \leftarrow S + w \cdot b$ 
14:     $W \leftarrow W + w$ 
15:     $I^\triangleright(u, v) \leftarrow \frac{1}{W} \cdot S$          $\triangleright$  see Eqn. (5.30)
16:    $I' \leftarrow \text{DUPLICATE}(I)$ 
17:   for all coordinates  $(u, v) \in M \times N$  do        $\triangleright$  Pass 2 (vertical)
18:      $a \leftarrow I^\triangleright(u, v)$ 
19:      $S \leftarrow 0, W \leftarrow 0$ 
20:     for  $n \leftarrow -K, \dots, K$  do
21:        $b \leftarrow I^\triangleright(u, v + n)$ 
22:        $w_d \leftarrow e^{-\frac{n^2}{2\sigma_d^2}}$            $\triangleright$  domain weight  $H_d(n)$ 
23:        $w_r \leftarrow e^{-\frac{(a-b)^2}{2\sigma_r^2}}$        $\triangleright$  range weight  $H_r$ 
24:        $w \leftarrow w_d \cdot w_r$                        $\triangleright$  composite weight
25:        $S \leftarrow S + w \cdot b$ 
26:        $W \leftarrow W + w$ 
27:        $I'(u, v) \leftarrow \frac{1}{W} \cdot S$          $\triangleright$  see Eqn. (5.31)
28:   return  $I'$ .

```

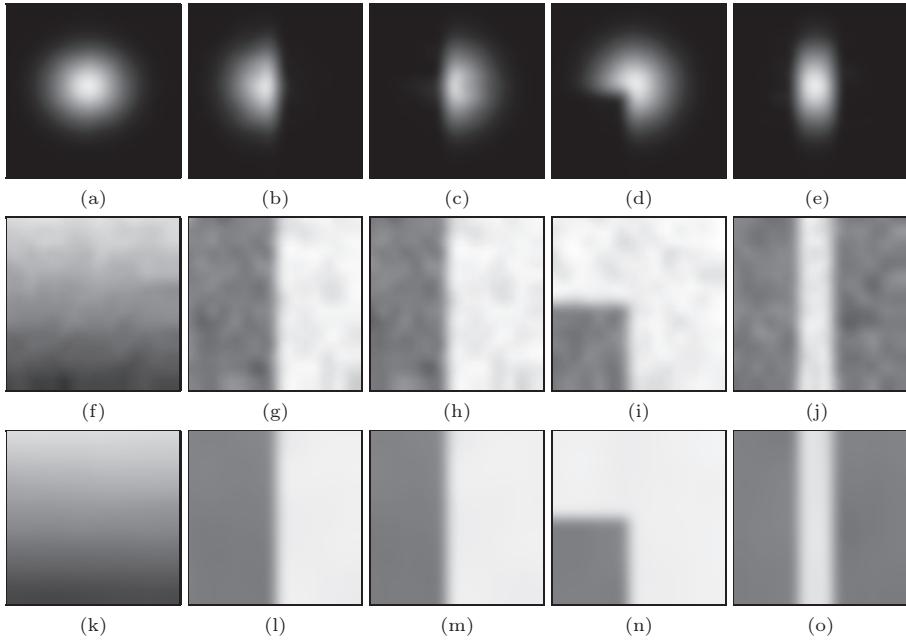


Figure 5.10 Response of a separable Bilateral filter in various situations. Effective kernel at the center pixel (a–e), original image data (f–j), filtered image data (k–o). Settings are the same as in Figs. 5.6–5.8.

filter Fig. 5.11 (g) is orientation-insensitive, the upper part of the separable kernel is clearly truncated Fig. 5.11 (h). But although the separable Bilateral filter is sensitive to local structure orientation, it performs well and is usually a sufficient substitute for the non-separable version [104]. The color examples shown in Fig. 5.12 demonstrate the effects of one-dimensional Bilateral filtering in the x - and y -directions. Note that the results are not exactly the same if the filter is first applied in the x - or in y -direction, but usually the differences are negligible.

5.2.5 Other implementations and improvements

A thorough analysis of the Bilateral filter as well as its relationship to adaptive smoothing and nonlinear diffusion can be found in [8] and [39]. In addition to the simple separable implementation described above, several other fast versions of the Bilateral filter have been proposed. The method described in [38] approximates the Bilateral by filtering sub-sampled copies of the image with discrete intensity kernels, and recombining the results using linear interpolation. An improved and theoretically well-grounded version of this method was presented in [97]. The fast technique proposed in [144] eliminates the redund-

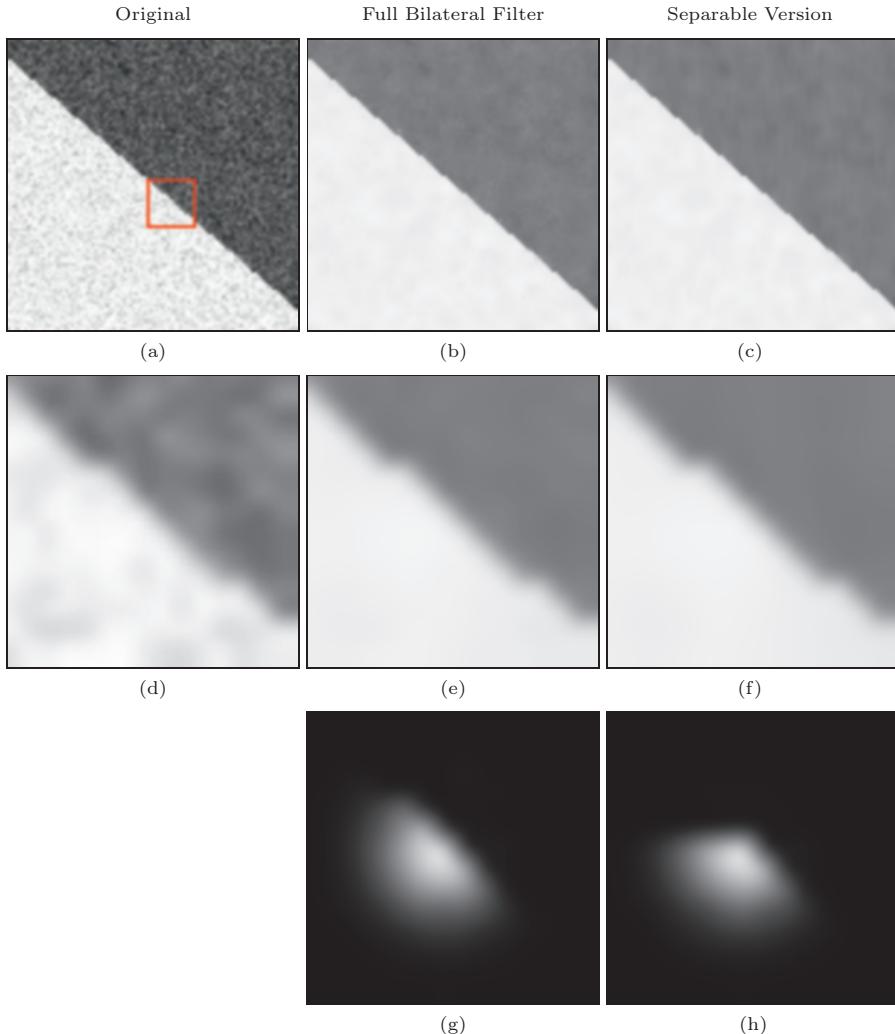


Figure 5.11 Bilateral filter—full vs. separable version. Original image (a) and enlarged detail (d). Results of the full Bilateral filter (b, e) and the separable version (c, f). The corresponding local filter kernels for the center pixel (positioned on the bright side of the step edge) for the full filter (g) and the separable version (h). Note how the upper part of the kernel in (h) is truncated along the horizontal axis, which shows that the separable filter is orientation-sensitive. In both cases, $\sigma_d = 2.0$, $\sigma_r = 25$.

dant calculations performed in partly overlapping image regions, albeit it is restricted to the use of box-shaped domain kernels. As demonstrated in [106,148], real-time performance using arbitrary-shaped kernels can be obtained by decomposing the filter into a set of smaller spatial filters.

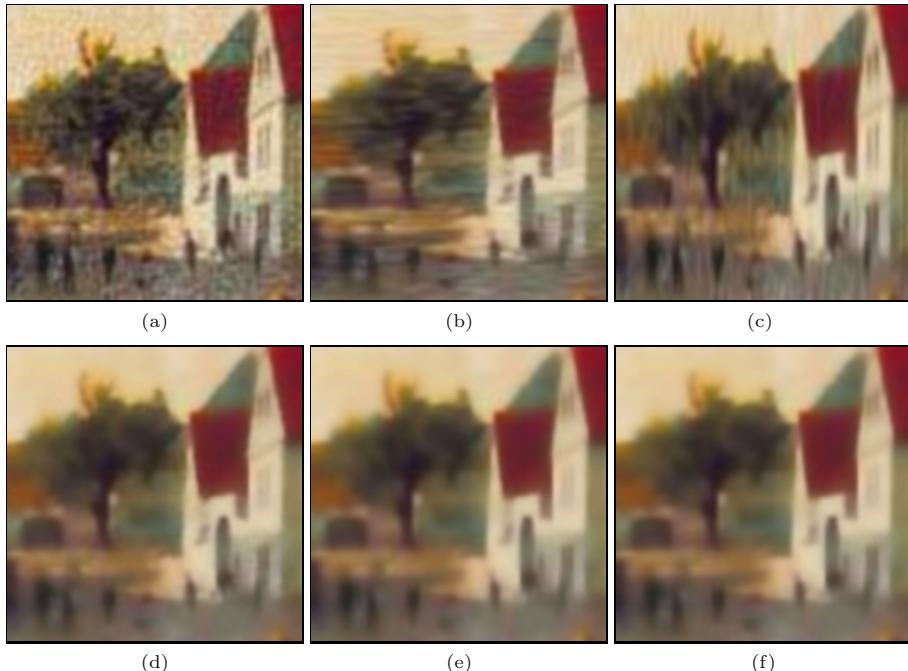


Figure 5.12 Separable Bilateral filter (color example). Original image (a), Bilateral filter applied only in the x -direction (b) and only in the y -direction (c). Note the resulting horizontal and vertical structures, respectively. Result of applying the *full* Bilateral filter (d) and the *separable* Bilateral filter applied in x/y order (e) and y/x order (f). Settings: $\sigma_d = 2.0$, $\sigma_r = 50$, L₂ color distance.

5.3 Anisotropic diffusion filters

Diffusion is a concept adopted from physics that models the spatial propagation of particles or state properties within substances. In the real world, certain physical properties (such as temperature) tend to diffuse homogeneously through a physical body, i. e., equally in all directions. The idea viewing image smoothing as a diffusion process has a long history in image processing (see, for example [7, 67]). To smooth an image and, at the same time, preserve edges or other “interesting” image structures, the diffusion process must somehow be made locally *non-homogeneous*; otherwise the entire image would come out equally blurred. Typically, the dominant smoothing direction is chosen to be *parallel* to nearby image contours, while smoothing is inhibited in the perpendicular direction, i. e., *across* the contours.

Since the pioneering work by Perona and Malik [101], anisotropic diffusion has seen continued interest in the image processing community and research in this area is still strong today. The main elements of their approach are outlined

in Section 5.3.2. While various other formulations have been proposed since, it was a key contribution by Weickert [141, 142] and Tschumperlé [130, 133] who unified them into a common framework and demonstrated their extension to color images. In Section 5.3.4 we give a brief introduction to the approach proposed by Tschumperlé and Deriche, as initially described in [130]. Beyond these selected examples, a vast literature exists on this topic, including excellent reviews [50, 141], textbook material [59, 114] and a plethora of journal articles (see [1, 24, 29, 92, 115, 123], for example).

5.3.1 Homogeneous diffusion and the heat equation

Assume that in a homogeneous, three-dimensional volume some physical property (e.g., temperature) is specified by a continuous function $f(\mathbf{x}, t)$ at position $\mathbf{x} = (x, y, z)$ and time t . With the system left to itself, the local differences in the property f will equalize over time until a global equilibrium is reached. This *diffusion process* in 3D space (x, y, z) and time (t) can be expressed using a partial differential equation (PDE),

$$\frac{\partial f}{\partial t} = c \cdot [\nabla^2 f] = c \cdot \left[\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \right]. \quad (5.35)$$

This is the so-called *heat equation*, where $\nabla^2 f$ denotes the *Laplace operator*⁷ applied to the scalar-valued function f , and c is a constant which describes the (thermal) *conductivity* or *conduction coefficient* of the material. Since the conductivity is independent of position and orientation (c is constant), the resulting process is *isotropic*, i.e., the heat spreads evenly in all directions. For simplicity, we assume $c = 1$. Since f is a multi-dimensional function in space and time, we make this fact a bit more transparent by attaching explicit space and time coordinates \mathbf{x} and τ to Eqn. (5.35), that is,

$$\frac{\partial f}{\partial t}(\mathbf{x}, \tau) = \frac{\partial^2 f}{\partial x^2}(\mathbf{x}, \tau) + \frac{\partial^2 f}{\partial y^2}(\mathbf{x}, \tau) + \frac{\partial^2 f}{\partial z^2}(\mathbf{x}, \tau) \quad (5.36)$$

or, written more compactly,

$$f_t(\mathbf{x}, \tau) = f_{xx}(\mathbf{x}, \tau) + f_{yy}(\mathbf{x}, \tau) + f_{zz}(\mathbf{x}, \tau). \quad (5.37)$$

⁷ Remember that ∇f denotes the *gradient* of the function f , which is a vector for any multi-dimensional function. The Laplace operator (or *Laplacian*) $\nabla^2 f$ corresponds to the *divergence* of the *gradient* of f , denoted $\text{div } \nabla f$, which is a scalar value (see Sec. B.6.5 and B.6.4). Other notations for the Laplacian are $\nabla \cdot (\nabla f)$, $(\nabla \cdot \nabla) f$, $\nabla \cdot \nabla f$, $\nabla^2 f$, or Δf .

Diffusion in images

A continuous, time-varying image I may be treated analogously to the function $f(\mathbf{x}, \tau)$, with the local intensities taking on the role of the temperature values in Eqn. (5.37). In this two-dimensional case, the isotropic diffusion equation can be written as

$$\frac{\partial I}{\partial t} = \nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}, \quad \text{or} \quad (5.38)$$

$$I_t(\mathbf{x}, \tau) = I_{xx}(\mathbf{x}, \tau) + I_{yy}(\mathbf{x}, \tau), \quad (5.39)$$

with the derivatives $I_t = \partial I / \partial t$, $I_{xx} = \partial^2 I / \partial x^2$, and $I_{yy} = \partial^2 I / \partial y^2$.

An approximate, numerical solution of such a PDE can be obtained by replacing the derivatives with finite differences. Starting with the initial (noisy) image $I^{(0)} = I$, the solution is calculated iteratively in the form

$$I^{(n)}(\mathbf{u}) \leftarrow \begin{cases} I(\mathbf{u}) & \text{for } n = 0, \\ I^{(n-1)}(\mathbf{u}) + \alpha \cdot [\nabla^2 I^{(n-1)}(\mathbf{u})] & \text{for } n > 0, \end{cases} \quad (5.40)$$

for each image position $\mathbf{u} = (u, v)$. This is called the “direct” solution method (there are other methods but this is the simplest). The constant α in Eqn. (5.40) is the time increment, which controls the speed of the diffusion process. Its value should be in the range $(0, 0.25]$ for the numerical scheme to be stable. At each iteration n , the variations in the image function are reduced and (depending on the boundary conditions) the image function should eventually flatten out to a constant plane as n approaches infinity.

For a discrete image I , the Laplacian $\nabla^2 I$ in Eqn. (5.40) can be approximated by a linear 2D filter,

$$\nabla^2 I \approx I * H^L = I * \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad (5.41)$$

as described earlier.⁸ An essential property of isotropic diffusion is that it has the same effect as a Gaussian filter whose width grows with the elapsed time. For a discrete 2D image, in particular, the result obtained after n diffusion steps (Eqn. (5.40)),

$$I^{(n)} = I * H^{G, \sigma_n}, \quad (5.42)$$

is the same as filtering the original image I with the normalized Gaussian kernel

$$H^{G, \sigma_n}(x, y) = \frac{1}{2\pi\sigma_n^2} \cdot e^{-\frac{x^2+y^2}{2\sigma_n^2}} \quad (5.43)$$

of width $\sigma_n = \sqrt{2t} = \sqrt{2n/\alpha}$. The examples in Figs. 5.13 and 5.14 illustrate this Gaussian smoothing behavior obtained with discrete isotropic diffusion.

⁸ See also Vol. 1, Sec. 6.6.1 [20, p. 147].

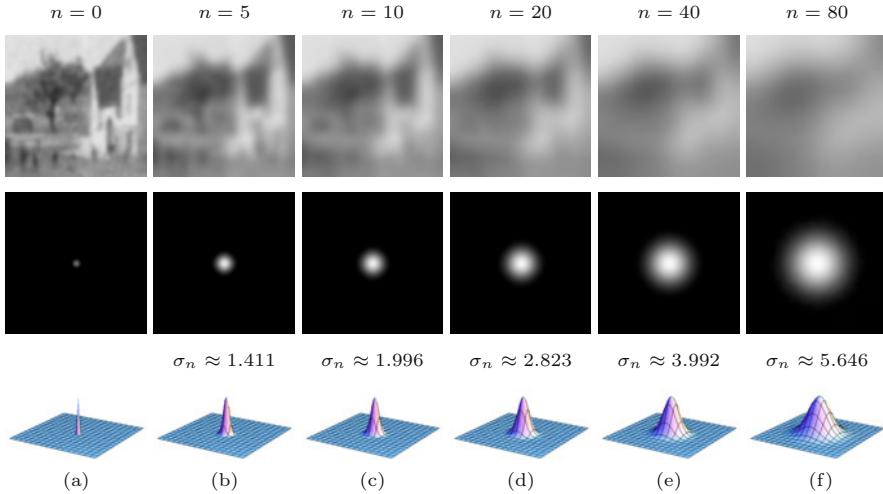


Figure 5.13 Discrete isotropic diffusion. Blurred images and impulse response obtained after T iterations, with $\alpha = 0.20$ (see Eqn. (5.40)). The size of the images is 50×50 . The width of the equivalent Gaussian kernel (σ_n) grows with the square root of n (the number of iterations). Impulse response plots are normalized to identical peak values.

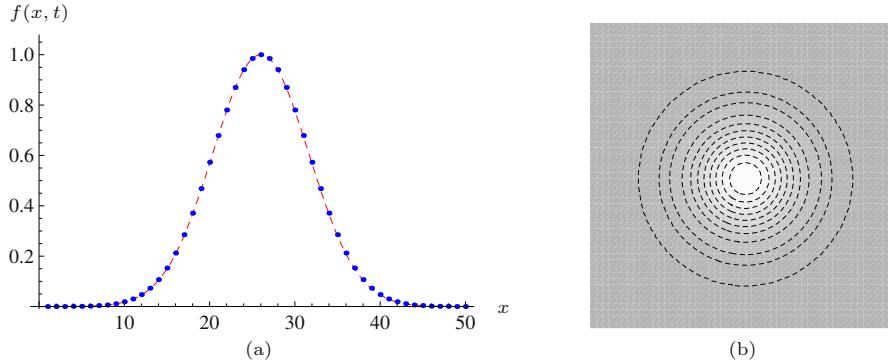


Figure 5.14 Result of discrete isotropic diffusion (blue dots) compared to the equivalent continuous Gaussian (dashed line) for $T = 80$ (a). The data correspond to a centered cross section of Fig. 5.13(f). The contour plot in (b) demonstrates that the discrete diffusion process is indeed isotropic.

5.3.2 Perona-Malik filter

Isotropic diffusion, as described above, is a homogeneous operation that is independent of the underlying image content. Like any Gaussian filter, it effectively suppresses image noise but also tends to blur away sharp boundaries and detailed structures, a property that is often undesirable. The idea proposed in [101] is to make the conductivity coefficient *variable* and dependent on the local image structure. This is done by replacing the conductivity constant c in

Eqn. (5.35), which can be written as

$$\frac{\partial I}{\partial t} = c \cdot (\nabla^2 I) = \operatorname{div}(c \cdot \nabla I), \quad (5.44)$$

by a function $c(\mathbf{x}, t)$ that varies over space \mathbf{x} and time t , that is,⁹

$$\frac{\partial I}{\partial t} = \operatorname{div}(c(\mathbf{x}, t) \cdot \nabla I). \quad (5.45)$$

If the conductivity function $c()$ is constant, then the equation reduces to the isotropic diffusion model in Eqn. (5.39).

Different behaviors can be implemented by selecting a particular function $c()$. To achieve edge-preserving smoothing, the conductivity $c()$ is chosen as a function of the magnitude of the local gradient vector ∇I , that is,

$$c(\mathbf{x}, t) = g(\|\nabla I(\mathbf{x}, t)\|). \quad (5.46)$$

To preserve edges, the function $g(d) : \mathbb{R} \rightarrow [0, 1]$ should return high values in areas of low image gradient, enabling smoothing of homogeneous regions, but return low values (and thus inhibit smoothing) where the local brightness changes rapidly. Commonly used conductivity functions $g(d)$ are, for example [26, 101],

$$\begin{aligned} g^{(1)}(d) &= e^{-(d/\kappa)^2}, & g^{(2)}(d) &= \frac{1}{1+(d/\kappa)^2}, \\ g^{(3)}(d) &= \frac{1}{\sqrt{1+(d/\kappa)^2}}, & g^{(4)}(d) &= \begin{cases} (1-(d/2\kappa)^2)^2 & \text{for } d \leq 2\kappa, \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \quad (5.47)$$

where $\kappa > 0$ is a constant that is either set manually (typically in the range [5, 50] for 8-bit images) or adjusted to the amount of image noise. The Gaussian conductivity function $g^{(1)}$ tends to promote high-contrast edges, whereas $g^{(2)}$ and even more $g^{(3)}$ prefer wide, flat regions over smaller ones. Function $g^{(4)}$, which corresponds to Tukey's *biweight* function known from robust statistics [114, p. 230], is strictly zero for any argument $d > 2\kappa$. Graphs of the four functions in Eqn. (5.47) are shown in Fig. 5.15 for selected values of κ . The exact shape of the function $g()$ does not appear to be critical; other functions with similar properties (e.g., with a linear cutoff) are sometimes used instead.

As an approximate discretization of Eqn. (5.45), Perona and Malik [101] proposed the simple iterative scheme

$$I^{(n)}(\mathbf{u}) \leftarrow I^{(n-1)}(\mathbf{u}) + \alpha \cdot \sum_{i=0}^3 g(|\delta_i(I^{(n-1)}, \mathbf{u})|) \cdot \delta_i(I^{(n-1)}, \mathbf{u}) \quad (5.48)$$

⁹ “div” denotes the divergence operator (see Appendix B.6.4).

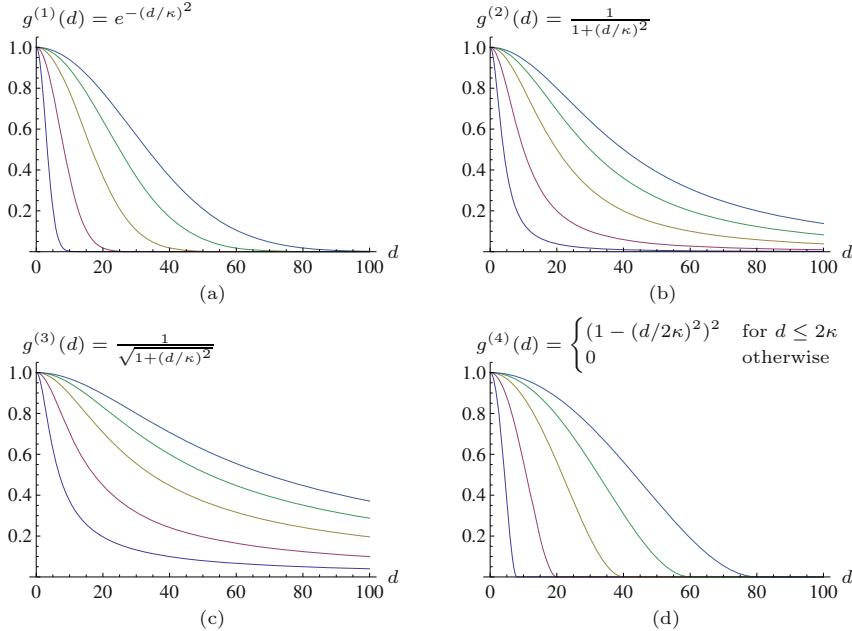


Figure 5.15 Typical conductivity functions $g^{(1)}(\cdot), \dots, g^{(4)}(\cdot)$ for $\kappa = 4, 10, 20, 30, 40$ (see Eqn. (5.47)). If the magnitude of the local gradient d is small (near zero), smoothing amounts to a maximum (1.0), whereas diffusion is reduced where the gradient is high, e.g., at or near edges. Smaller values of κ result in narrower curves, thereby restricting the smoothing operation to image areas with only small variations.

where $I^{(0)} = I$ is the original image and

$$\delta_i(I, \mathbf{u}) = I(\mathbf{u} + \mathbf{d}_i) - I(\mathbf{u}) \quad (5.49)$$

denotes the difference between the pixel $I(\mathbf{u})$ and one of its four direct neighbors (see Fig. 5.16),

$$\mathbf{d}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \mathbf{d}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad \mathbf{d}_2 = -\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad \mathbf{d}_3 = -\begin{pmatrix} 0 \\ 1 \end{pmatrix}. \quad (5.50)$$

The procedure for computing the Perona-Malik filter for scalar-valued images is summarized in Alg. 5.7. The examples in Fig. 5.17 demonstrate how this filter performs along a step edge in a noisy grayscale image compared to isotropic (i.e., Gaussian) filtering.

In summary, the principle operation of this filter is to inhibit smoothing in the direction of strong local gradient vectors. Wherever the local contrast (and thus the gradient) is small, diffusion occurs uniformly in all directions, effectively implementing a Gaussian smoothing filter. However, in locations of high gradients, smoothing is inhibited along the gradient direction and allowed only in the direction perpendicular to it. If viewed as a heat diffusion process, a

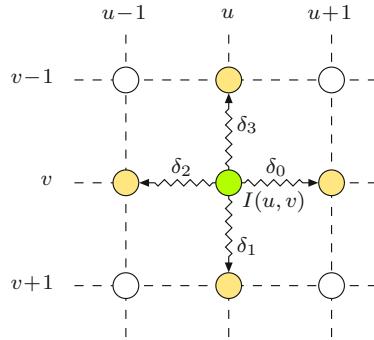


Figure 5.16 Discrete lattice used in the Perona-Malik algorithm. The green element represents the current image pixel at position $\mathbf{u} = (u, v)$ and the yellow elements are its direct 4-neighbors.

high-gradient brightness edge in an image acts like an insulating layer between areas of different temperatures. While temperatures continuously level out in the homogeneous regions on either side of an edge, thermal energy does not diffuse across the edge itself. Note that technically the Perona-Malik filter (as defined in Eqn. (5.45)) is considered a *non-linear* but not *anisotropic* diffusion filter because the conductivity function $g()$ is only a scalar and not a (directed) vector-valued function [141]. However, the (inexact) discretization used in Eqn. (5.48), where each lattice direction is attenuated individually, makes the filter appear to perform in an anisotropic fashion.

5.3.3 Perona-Malik filter for color images

The original Perona-Malik filter is not explicitly designed for color images or vector-valued images in general. The simplest way to apply this filter to a color image is (as usual) to treat the color channels as a set of independent scalar images and filter them separately. Edges should be preserved, since they occur only where at least one of the color channels exhibits a strong variation. However, different filters are applied to the color channels and thus new chromaticities may be produced that were not contained in the original image. Nevertheless, the results obtained (see the examples in Fig. 5.18 (b-d)) are often satisfactory and the approach is frequently used because of its simplicity.

Color diffusion based on the brightness gradient

As an alternative to filtering each color channel separately, it has been proposed to use only the brightness (intensity) component to control the diffusion

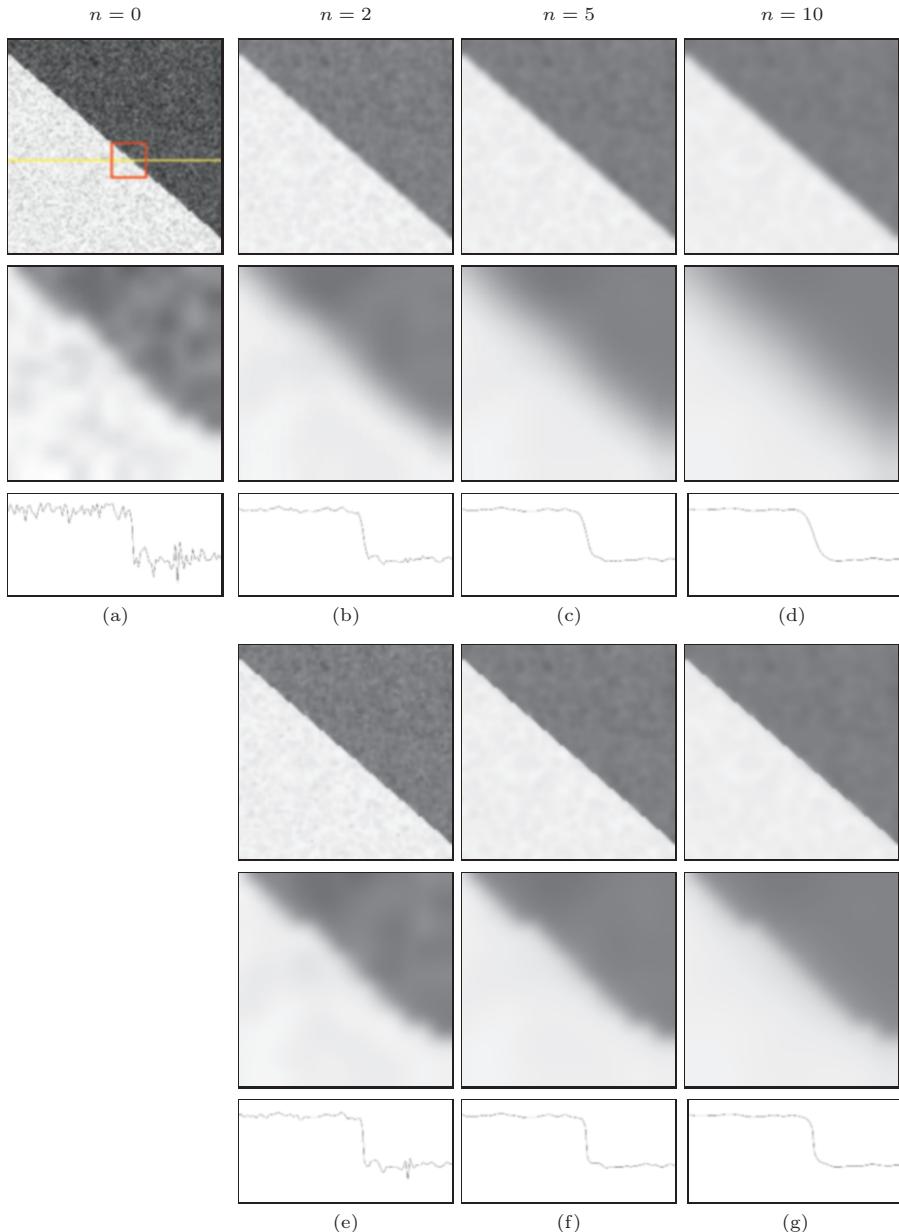


Figure 5.17 Isotropic vs. anisotropic diffusion applied to a noisy step edge. Original image, enlarged detail and horizontal profile (a), results of isotropic diffusion (b-d), results of anisotropic diffusion (e-g) after $n = 2, 5, 10$ iterations, respectively ($\alpha = 0.20$, $\kappa = 40$).

Algorithm 5.7 Perona-Malik anisotropic diffusion filter for scalar (grayscale) images. The input image I is assumed to be real-valued (floating-point). Temporary real-valued maps D_x, D_y are used to hold the directional gradient values, which are then re-calculated in every iteration. The conductivity function $g(d)$ can be one of the functions defined in Eqn. (5.47), or any similar function.

```

1: PERONAMALIKGRAY( $I, \alpha, \kappa, T$ )
   Input:  $I$ , a grayscale image of size  $M \times N$ ;  $\alpha$ , update rate;  $\kappa$ , smooth-
          ness parameter;  $T$ , number of iterations.
   Returns the modified image  $I$ .
   Specify the conductivity function:
2:  $g(d) := e^{-(d/\kappa)^2}$            ▷ for example, see alternatives in Eqn. (5.47)
3:  $(M, N) \leftarrow \text{SIZE}(I)$ 
4: Create maps  $D_x, D_y: M \times N \rightarrow \mathbb{R}$ 
5: for  $n \leftarrow 1, \dots, T$  do           ▷ perform  $T$  iterations
6:   for all coordinates  $(u, v) \in M \times N$  do     ▷ re-calculate gradients
7:      $D_x(u, v) \leftarrow \begin{cases} I(u+1, v) - I(u, v) & \text{if } u < M-1 \\ 0 & \text{otherwise} \end{cases}$ 
8:      $D_y(u, v) \leftarrow \begin{cases} I(u, v+1) - I(u, v) & \text{if } v < N-1 \\ 0 & \text{otherwise} \end{cases}$ 
9:   for all coordinates  $(u, v) \in M \times N$  do           ▷ update the image
10:     $\delta_0 \leftarrow D_x(u, v)$ 
11:     $\delta_1 \leftarrow D_y(u, v)$ 
12:     $\delta_2 \leftarrow \begin{cases} -D_x(u-1, v) & \text{if } u > 0 \\ 0 & \text{otherwise} \end{cases}$ 
13:     $\delta_3 \leftarrow \begin{cases} -D_y(u, v-1) & \text{if } v > 0 \\ 0 & \text{otherwise} \end{cases}$ 
14:     $I(u, v) \leftarrow I(u, v) + \alpha \cdot \sum_{k=0}^3 g(|\delta_k|) \cdot \delta_k$ 
15: return  $I$ .
```

process of all color channels. Given an RGB color image $\mathbf{I} = (I_R, I_G, I_B)$ and a brightness function $\beta(\mathbf{I})$, the iterative scheme in Eqn. (5.48) could be modified to

$$\mathbf{I}^{(n)}(\mathbf{u}) \leftarrow \mathbf{I}^{(n-1)}(\mathbf{u}) + \alpha \cdot \sum_{i=0}^3 g(|\beta_i(\mathbf{I}^{(n-1)}, \mathbf{u})|) \cdot \delta_i(\mathbf{I}^{(n-1)}, \mathbf{u}), \quad (5.51)$$

where

$$\beta_i(\mathbf{I}, \mathbf{u}) = \beta(\mathbf{I}(\mathbf{u} + \mathbf{d}_i)) - \beta(\mathbf{I}(\mathbf{u})), \quad (5.52)$$

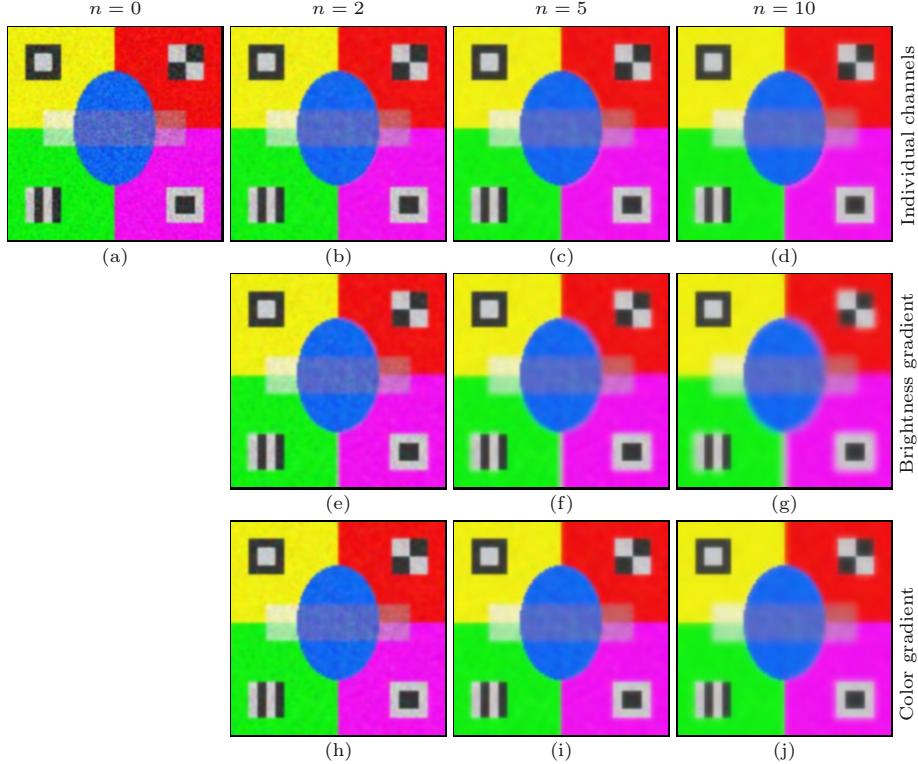


Figure 5.18 Anisotropic diffusion filter (color). Noisy test image (a). Anisotropic diffusion filter applied to *individual color channels* (b–d), diffusion controlled by *brightness gradient* (e–g), diffusion controlled by *color gradient* (h–j), after 2, 5, and 10 iterations, respectively ($\alpha = 0.20$, $\kappa = 40$). With diffusion controlled by the brightness gradient, strong blurring occurs between regions of different color but similar brightness (e–g). The most consistent results are obtained by diffusion controlled by the *color gradient* (h–j). Filtering was performed in linear RGB color space.

with \mathbf{d}_i as defined in Eqn. (5.50), is the local brightness difference and

$$\delta_i(\mathbf{I}, \mathbf{u}) = \begin{pmatrix} I_R(\mathbf{u} + \mathbf{d}_i) - I_R(\mathbf{u}) \\ I_G(\mathbf{u} + \mathbf{d}_i) - I_G(\mathbf{u}) \\ I_B(\mathbf{u} + \mathbf{d}_i) - I_B(\mathbf{u}) \end{pmatrix} = \begin{pmatrix} \delta_i(I_R, \mathbf{u}) \\ \delta_i(I_G, \mathbf{u}) \\ \delta_i(I_B, \mathbf{u}) \end{pmatrix} \quad (5.53)$$

is the local color difference vector for the neighboring pixels in directions $i = 0, \dots, 3$ (see Fig. 5.16). Typical choices for the brightness function $\beta()$ are the *luminance* Y (calculated as a weighted sum of the linear R, G, B components), *luma* Y' (from nonlinear R', G', B' components), or the lightness component L of the CIELAB and CIELUV color spaces (see Sec. 3.1 for a detailed discussion).

Algorithm 5.7 can be easily adapted to implement this type of color filter. An obvious disadvantage of this method is that it naturally blurs across color

edges if the neighboring colors are of similar brightness, as the examples in [Fig. 5.18 \(e–g\)](#) demonstrate. This limits its usefulness for practical applications.

Using the color gradient

A better option for controlling the diffusion process in all three color channels is to use the color gradient (see Sec. 4.2.1). As defined in Eqn. (4.18), the color gradient

$$(\text{grad}_\theta \mathbf{I})(\mathbf{u}) = \mathbf{I}_x(\mathbf{u}) \cdot \cos(\theta) + \mathbf{I}_y(\mathbf{u}) \cdot \sin(\theta) \quad (5.54)$$

is a three-dimensional vector, representing the combined variations of the color image \mathbf{I} at position \mathbf{u} in a given direction θ . The squared norm of this vector, $S_\theta(\mathbf{I}, \mathbf{u}) = \|(\text{grad}_\theta \mathbf{I})(\mathbf{u})\|^2$, called the *squared local contrast*, is a scalar quantity useful for color edge detection. Along the horizontal and vertical directions of the discrete diffusion lattice (see [Fig. 5.16](#)), the angle θ is a multiple of $\pi/2$, and thus one of the cosine/sine terms in Eqn. (5.54) vanishes, that is,

$$\|(\text{grad}_\theta \mathbf{I})(\mathbf{u})\| = \|(\text{grad}_{i\pi/2} \mathbf{I})(\mathbf{u})\| = \begin{cases} \|\mathbf{I}_x(\mathbf{u})\| & \text{for } i = 0, 2, \\ \|\mathbf{I}_y(\mathbf{u})\| & \text{for } i = 1, 3. \end{cases} \quad (5.55)$$

Taking δ_i (Eqn. (5.53)) as an estimate for the horizontal and vertical derivatives \mathbf{I}_x , \mathbf{I}_y , the diffusion iteration (adapted from Eqn. (5.48)) thus becomes

$$\mathbf{I}^{(n)}(\mathbf{u}) \leftarrow \mathbf{I}^{(n-1)}(\mathbf{u}) + \alpha \cdot \sum_{i=0}^3 g(\|\delta_i(\mathbf{I}^{(n-1)}, \mathbf{u})\|) \cdot \delta_i(\mathbf{I}^{(n-1)}, \mathbf{u}), \quad (5.56)$$

with $g()$ chosen from one of the conductivity functions in Eqn. (5.47). Note that this is almost identical to the formulation in Eqn. (5.48), except for the use of vector-valued images and the absolute values $|\cdot|$ being replaced by the vector norm $\|\cdot\|$. The diffusion process is coupled between color channels, because the local diffusion strength depends on the combined color difference vectors. Thus, unlike in the brightness-governed diffusion scheme in Eqn. (5.51), opposing variations in different color do not cancel out and edges between colors of similar brightness are preserved (see the examples in [Fig. 5.18 \(h–j\)](#)).

The resulting process is summarized in [Alg. 5.8](#). The algorithm assumes that the components of the color image \mathbf{I} are real-valued. In practice, integer-valued images must be converted to floating point before this procedure can be applied and integer results should be recovered by appropriate rounding.

Algorithm 5.8 Anisotropic diffusion filter for color images based on the color gradient (see Sec. 4.2.1). The input image \mathbf{I} is assumed to contain real-valued (floating-point) color vectors.

```

1: PERONAMALIKCOLOR( $\mathbf{I}, \alpha, \kappa, T$ )
   Input:  $\mathbf{I}$ , an RGB color image of size  $M \times N$ ;  $\alpha$ , update rate;  $\kappa$ , smoothness parameter;  $T$ , number of iterations.
   Returns the modified image  $\mathbf{I}$ .
   Specify the conductivity function:
2:  $g(d) := e^{-(d/\kappa)^2}$             $\triangleright$  for example, see alternatives in Eqn. (5.47)
3:  $(M, N) \leftarrow \text{SIZE}(\mathbf{I})$ 
4: Create maps  $\mathbf{D}_x, \mathbf{D}_y: M \times N \rightarrow \mathbb{R}^3$ ;  $S_x, S_y: M \times N \rightarrow \mathbb{R}$ 
5: for  $t \leftarrow 1, \dots, T$  do            $\triangleright$  perform  $T$  iterations
6:   for all  $(u, v) \in M \times N$  do        $\triangleright$  re-calculate gradients
7:      $\mathbf{D}_x(u, v) \leftarrow \begin{cases} \mathbf{I}(u+1, v) - \mathbf{I}(u, v) & \text{if } u < M-1 \\ \mathbf{0} & \text{otherwise} \end{cases}$ 
8:      $\mathbf{D}_y(u, v) \leftarrow \begin{cases} \mathbf{I}(u, v+1) - \mathbf{I}(u, v) & \text{if } v < N-1 \\ \mathbf{0} & \text{otherwise} \end{cases}$ 
9:      $S_x(u, v) \leftarrow (\mathbf{D}_x(u, v))^2$             $\triangleright = I_{R,x}^2 + I_{G,x}^2 + I_{B,x}^2$ 
10:     $S_y(u, v) \leftarrow (\mathbf{D}_y(u, v))^2$             $\triangleright = I_{R,y}^2 + I_{G,y}^2 + I_{B,y}^2$ 
11:    for all  $(u, v) \in M \times N$  do        $\triangleright$  update the image
12:       $s_0 \leftarrow S_x(u, v)$ ,  $\Delta_0 \leftarrow \mathbf{D}_x(u, v)$ 
13:       $s_1 \leftarrow S_y(u, v)$ ,  $\Delta_1 \leftarrow \mathbf{D}_y(u, v)$ 
14:       $s_2 \leftarrow 0$ ,  $\Delta_2 \leftarrow \mathbf{0}$ 
15:       $s_3 \leftarrow 0$ ,  $\Delta_3 \leftarrow \mathbf{0}$ 
16:      if  $u > 0$  then
17:         $s_2 \leftarrow S_x(u-1, v)$ 
18:         $\Delta_2 \leftarrow -\mathbf{D}_x(u-1, v)$ 
19:      if  $v > 0$  then
20:         $s_3 \leftarrow S_y(u, v-1)$ 
21:         $\Delta_3 \leftarrow -\mathbf{D}_y(u, v-1)$ 
22:       $\mathbf{I}(u, v) \leftarrow \mathbf{I}(u, v) + \alpha \cdot \sum_{k=0}^3 g(|s_k|) \cdot \Delta_k$ 
23: return  $\mathbf{I}$ .
```

Examples

Figure 5.19 shows the results of applying the Perona-Malik filter to a color image, using different modalities to control the diffusion process. In Fig. 5.19 (a) the *scalar* (grayscale) diffusion filter (described in Alg. 5.7) is applied *separately* to each color channel. In Fig. 5.19 (b) the diffusion process is coupled over all



Figure 5.19 Perona-Malik color example. Scalar diffusion filter applied separately to each color channel (a), diffusion controlled by the brightness gradient (b), diffusion controlled by color gradient (c). Common settings are $T = 10$, $\alpha = 0.20$, $g(d) = g^{(1)}(d)$, $\kappa = 25$; original image in Fig. 5.3.

three color channels and controlled by the *brightness gradient*, as specified in Eqn. (5.51). Finally, in Fig. 5.19 (c) the *color gradient* is used to control the common diffusion process, as defined in Eqn. (5.56) and Alg. 5.8. In each case, $T = 10$ diffusion iterations were applied, with update rate $\alpha = 0.20$, smoothness $\kappa = 25$ and conductivity function $g^{(1)}(d)$. The example demonstrates that, under otherwise equal conditions, edges and line structures are best preserved by the filter if the diffusion process is controlled by the color gradient.

5.3.4 Geometry-preserving anisotropic diffusion

The publication by Perona and Malik [101] was followed by increased interest in the use of diffusion filters based on partial differential equations. Numerous different schemes were proposed, mainly with the aim to better adapt the diffusion process to the underlying image geometry.

Generalized divergence-based formulation

Weickert [140,141] generalized the divergence-based formulation of the Perona-Malik approach (see Eqn. (5.44)), that is,

$$\frac{\partial I}{\partial t} = \operatorname{div}(c \cdot \nabla I),$$

by replacing the time-varying, scalar diffusivity field $c(\mathbf{x}, \tau) \in \mathbb{R}$ by a *diffusion tensor* field $\mathbf{D}(\mathbf{x}, \tau) \in \mathbb{R}^{2 \times 2}$ in the form

$$\frac{\partial I}{\partial t} = \operatorname{div}(\mathbf{D} \cdot \nabla I). \quad (5.57)$$

The time-varying tensor field $\mathbf{D}(\mathbf{x}, \tau)$ specifies a symmetric, positive-definite 2×2 matrix for each 2D image position \mathbf{x} and time τ (i.e., $\mathbf{D} : \mathbb{R}^3 \rightarrow \mathbb{R}^{2 \times 2}$ in the continuous case). Geometrically, \mathbf{D} specifies an oriented, stretched ellipse which controls the local diffusion process. \mathbf{D} may be independent of the image I but is typically derived from it. For example, the original Perona-Malik diffusion equation could be (trivially) written in the form¹⁰

$$\frac{\partial I}{\partial t} = \operatorname{div} \left[\underbrace{(c \cdot \mathbf{I}_2)}_{\mathbf{D}} \cdot \nabla I \right] = \operatorname{div} \left[\begin{pmatrix} c & 0 \\ 0 & c \end{pmatrix} \cdot \nabla I \right], \quad (5.58)$$

where $c = g(\|\nabla I(\mathbf{x}, t)\|)$ (see Eqn. (5.46)), and thus \mathbf{D} is coupled to the image content. In Weickert's approach, \mathbf{D} is constructed from the eigenvalues of the local “image structure tensor” [142], which we have encountered under different names in several places. This approach was also adapted to work with color images [143].

Trace-based formulation

Similar to the work of Weickert, the approach proposed by Tschumperlé and Deriche [130,132] also pursues a geometry-oriented generalization of anisotropic diffusion. The approach is directly aimed at vector-valued (color) images, but

¹⁰ \mathbf{I}_2 denotes the 2×2 identity matrix.

can also be applied to single-channel (scalar-valued) images. For a vector-valued image $\mathbf{I} = (I_1, \dots, I_n)$, the smoothing process is specified as

$$\frac{\partial I_k}{\partial t} = \text{trace}(\mathbf{A} \cdot \mathbf{H}_k), \quad (5.59)$$

for each channel k , where \mathbf{H}_k denotes the *Hessian* matrix of the scalar-valued image function of channel I_k , and \mathbf{A} is a square (2×2 for 2D images) matrix that depends on the complete image \mathbf{I} and adapts the smoothing process to the local image geometry. Note that \mathbf{A} is the same for all image channels. Since the trace of the Hessian matrix¹¹ is the Laplacian of the corresponding function (i.e., $\text{trace}(\mathbf{H}_I) = \nabla^2 I$) the diffusion equation for the Perona-Malik filter (Eqn. (5.44)) can be written as

$$\begin{aligned} \frac{\partial I}{\partial t} &= c \cdot (\nabla^2 I) = \text{div}(c \cdot \nabla I) \\ &= \text{trace}((c \cdot \mathbf{I}_2) \cdot \mathbf{H}_I) = \text{trace}(c \cdot \mathbf{H}_I). \end{aligned} \quad (5.60)$$

In this case, $\mathbf{A} = c \cdot \mathbf{I}_2$, which merely applies the constant scalar factor c to the Hessian matrix \mathbf{H}_I (and thus to the resulting Laplacian) that is derived from the local image (since $c = g(\|\nabla I(\mathbf{x}, t)\|)$) and does not represent any geometric information.

5.3.5 Tschumperlé-Deriche algorithm

This is different in the trace-based approach proposed by Tschumperlé and Deriche [130, 132], where the matrix \mathbf{A} in Eqn. (5.59) is composed by the expression

$$\mathbf{A} = f_1(\lambda_1, \lambda_2) \cdot (\hat{\mathbf{e}}_2 \cdot \hat{\mathbf{e}}_2^\top) + f_2(\lambda_1, \lambda_2) \cdot (\hat{\mathbf{e}}_1 \cdot \hat{\mathbf{e}}_1^\top), \quad (5.61)$$

where λ_1, λ_2 and $\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2$ are the eigenvalues and normalized eigenvectors, respectively, of the (smoothed) 2×2 structure matrix

$$\mathbf{G} = \sum_{k=1}^K (\nabla I_k) \cdot (\nabla I_k)^\top, \quad (5.62)$$

with ∇I_k denoting the local gradient vector in image channel I_k . The functions $f_1()$, $f_2()$, which are defined in Eqn. (5.73) below, use the two eigenvalues to control the diffusion strength along the dominant direction of the contours (f_1) and perpendicular to it (f_2). Since the resulting algorithm is more involved than most previous ones, we describe it in more detail than usual.

Given a vector-valued image $\mathbf{I}: M \times N \rightarrow \mathbb{R}^n$, the following steps are performed in each iteration of the algorithm:

¹¹ See Appendix B.6.6 for details.

Step 1: Calculate the gradient at each image position $\mathbf{u} = (u, v)$,

$$\nabla I_k(\mathbf{u}) = \begin{pmatrix} \frac{\partial I_k}{\partial x}(\mathbf{u}) \\ \frac{\partial I_k}{\partial y}(\mathbf{u}) \end{pmatrix} = \begin{pmatrix} I_{k,x}(\mathbf{u}) \\ I_{k,y}(\mathbf{u}) \end{pmatrix} = \begin{pmatrix} (I_k * H_x^\nabla)(\mathbf{u}) \\ (I_k * H_y^\nabla)(\mathbf{u}) \end{pmatrix}, \quad (5.63)$$

for each color channel $k = 1, \dots, K$.¹² The first derivatives of the gradient vector ∇I_k are estimated by convolving the image with the kernels

$$H_x^\nabla = \begin{bmatrix} -a & 0 & a \\ -b & 0 & b \\ -a & 0 & a \end{bmatrix} \quad \text{and} \quad H_y^\nabla = \begin{bmatrix} -a & -b & -a \\ 0 & 0 & 0 \\ a & b & a \end{bmatrix}, \quad (5.64)$$

with $a = (2 - \sqrt{2})/4$ and $b = (\sqrt{2} - 1)/2$ (such that $2a + b = 1/2$).¹³

Step 2: Smooth the channel gradients $I_{k,x}$, $I_{k,y}$ with an isotropic 2D Gaussian filter kernel H^{G,σ_d} of radius σ_d ,

$$\overline{\nabla I}_k = \begin{pmatrix} \bar{I}_{k,x} \\ \bar{I}_{k,y} \end{pmatrix} = \begin{pmatrix} I_{k,x} * H^{G,\sigma_d} \\ I_{k,y} * H^{G,\sigma_d} \end{pmatrix}, \quad (5.65)$$

for each image channel $k = 1, \dots, K$. In practice, this step is usually skipped by setting $\sigma_d = 0$.

Step 3: Calculate the *Hessian matrix* (see Appendix B.6.6) for each image channel I_k , $k = 1, \dots, K$, that is,

$$\mathbf{H}_k = \begin{pmatrix} \frac{\partial^2 I_k}{\partial x^2} & \frac{\partial^2 I_k}{\partial x \partial y} \\ \frac{\partial^2 I_k}{\partial y \partial x} & \frac{\partial^2 I_k}{\partial y^2} \end{pmatrix} = \begin{pmatrix} I_{k,xx} & I_{k,xy} \\ I_{k,xy} & I_{k,yy} \end{pmatrix} = \begin{pmatrix} I_k * H_{xx}^\nabla & I_k * H_{xy}^\nabla \\ I_k * H_{xy}^\nabla & I_k * H_{yy}^\nabla \end{pmatrix}, \quad (5.66)$$

using the filter kernels

$$H_{xx}^\nabla = [1 \ -2 \ 1], \quad H_{yy}^\nabla = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}, \quad H_{xy}^\nabla = \begin{bmatrix} 0.25 & 0 & -0.25 \\ 0 & 0 & 0 \\ -0.25 & 0 & 0.25 \end{bmatrix}. \quad (5.67)$$

¹² Note that $\nabla I_k(\mathbf{u})$ in Eqn. (5.63) is a two-dimensional, vector-valued function, i.e., a dedicated vector is calculated for every image position $\mathbf{u} = (u, v)$. For better readability, we omit the spatial coordinate (\mathbf{u}) in the following and simply write ∇I_k instead of $\nabla I_k(\mathbf{u})$. Analogously, all related vectors and matrices defined below (including the vectors $\mathbf{e}_1, \mathbf{e}_2$ and the matrices $\mathbf{G}, \bar{\mathbf{G}}, \mathbf{A}$, and \mathbf{H}_k) are also calculated for each image point \mathbf{u} , without the spatial coordinate being explicitly given.

¹³ Any other common set of x/y gradient kernels (e.g., Sobel masks) could be used instead, but these filters have better rotation invariance than their traditional counterparts. Similar kernels (with $a = 3/32$, $b = 10/32$) were proposed by Jähne in [60, p. 353].

Step 4: Calculate the local variation (structure) matrix as

$$\begin{aligned} \mathbf{G} &= \begin{pmatrix} G_0 & G_1 \\ G_1 & G_2 \end{pmatrix} = \sum_{k=1}^K (\nabla \bar{I}_k) \cdot (\nabla \bar{I}_k)^T \\ &= \sum_{k=1}^K \begin{pmatrix} \bar{I}_{k,x}^2 & \bar{I}_{k,x} \cdot \bar{I}_{k,y} \\ \bar{I}_{k,x} \cdot \bar{I}_{k,y} & \bar{I}_{k,y}^2 \end{pmatrix} = \begin{pmatrix} \sum_{k=1}^K \bar{I}_{k,x}^2 & \sum_{k=1}^K \bar{I}_{k,x} \cdot \bar{I}_{k,y} \\ \sum_{k=1}^K \bar{I}_{k,x} \cdot \bar{I}_{k,y} & \sum_{k=1}^K \bar{I}_{k,y}^2 \end{pmatrix}, \end{aligned} \quad (5.68)$$

for each image position \mathbf{u} . Note that the matrix \mathbf{G} is symmetric (and positive semidefinite). In particular, for an RGB color image this is (coordinates \mathbf{u} again omitted)

$$\begin{aligned} \mathbf{G} &= \begin{pmatrix} \bar{I}_{R,x}^2 & \bar{I}_{R,x} \bar{I}_{R,y} \\ \bar{I}_{R,x} \bar{I}_{R,y} & \bar{I}_{R,y}^2 \end{pmatrix} + \begin{pmatrix} \bar{I}_{G,x}^2 & \bar{I}_{G,x} \bar{I}_{G,y} \\ \bar{I}_{G,x} \bar{I}_{G,y} & \bar{I}_{G,y}^2 \end{pmatrix} + \begin{pmatrix} \bar{I}_{B,x}^2 & \bar{I}_{B,x} \bar{I}_{B,y} \\ \bar{I}_{B,x} \bar{I}_{B,y} & \bar{I}_{B,y}^2 \end{pmatrix} \\ &= \begin{pmatrix} \bar{I}_{R,x}^2 + \bar{I}_{G,x}^2 + \bar{I}_{B,x}^2 & \bar{I}_{R,x} \bar{I}_{R,y} + \bar{I}_{G,x} \bar{I}_{G,y} + \bar{I}_{B,x} \bar{I}_{B,y} \\ \bar{I}_{R,x} \bar{I}_{R,y} + \bar{I}_{G,x} \bar{I}_{G,y} + \bar{I}_{B,x} \bar{I}_{B,y} & \bar{I}_{R,y}^2 + \bar{I}_{G,y}^2 + \bar{I}_{B,y}^2 \end{pmatrix}. \end{aligned} \quad (5.69)$$

Step 5: Smooth the elements of the structure matrix \mathbf{G} using an isotropic Gaussian filter kernel H^{G,σ_g} of radius σ_g , that is,

$$\bar{\mathbf{G}} = \begin{pmatrix} \bar{G}_0 & \bar{G}_1 \\ \bar{G}_1 & \bar{G}_2 \end{pmatrix} = \begin{pmatrix} G_0 * H^{G,\sigma_g} & G_1 * H^{G,\sigma_g} \\ G_1 * H^{G,\sigma_g} & G_2 * H^{G,\sigma_g} \end{pmatrix}. \quad (5.70)$$

Step 6: For each image position \mathbf{u} , calculate the eigenvalues

$$\lambda_1, \lambda_2,$$

for the smoothed 2×2 matrix $\bar{\mathbf{G}}$, such that $\lambda_1 \geq \lambda_2$, and the corresponding normalized eigenvectors¹⁴

$$\hat{\mathbf{e}}_1 = \begin{pmatrix} \hat{x}_1 \\ \hat{y}_1 \end{pmatrix}, \quad \hat{\mathbf{e}}_2 = \begin{pmatrix} \hat{x}_2 \\ \hat{y}_2 \end{pmatrix},$$

such that $\|\hat{\mathbf{e}}_1\| = \|\hat{\mathbf{e}}_2\| = 1$. Note that $\hat{\mathbf{e}}_1$ points in the direction of maximum change and $\hat{\mathbf{e}}_2$ points in the perpendicular direction, i.e., along the edge tangent. Thus, smoothing should occur predominantly along $\hat{\mathbf{e}}_2$. Since $\hat{\mathbf{e}}_1$ and $\hat{\mathbf{e}}_2$ are normal to each other, we can express $\hat{\mathbf{e}}_2$ in terms of $\hat{\mathbf{e}}_1$, for example,

$$\hat{\mathbf{e}}_2 \equiv \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \cdot \hat{\mathbf{e}}_1 = \begin{pmatrix} -\hat{y}_1 \\ \hat{x}_1 \end{pmatrix}. \quad (5.71)$$

¹⁴ See Appendix B.4.1 for details on calculating the eigensystem of a 2×2 matrix.

Step 7: From the eigenvalues (λ_1, λ_2) and the normalized eigenvectors (\hat{e}_1, \hat{e}_2) of $\tilde{\mathbf{G}}$, compose the symmetric matrix \mathbf{A} in the form

$$\begin{aligned}\mathbf{A} &= \begin{pmatrix} A_0 & A_1 \\ A_1 & A_2 \end{pmatrix} = \underbrace{f_1(\lambda_1, \lambda_2) \cdot (\hat{e}_2 \cdot \hat{e}_2^\top)}_{c_1} + \underbrace{f_2(\lambda_1, \lambda_2) \cdot (\hat{e}_1 \cdot \hat{e}_1^\top)}_{c_2} \\ &= c_1 \cdot \begin{pmatrix} \hat{y}_1^2 & -\hat{x}_1 \cdot \hat{y}_1 \\ -\hat{x}_1 \cdot \hat{y}_1 & \hat{x}_1^2 \end{pmatrix} + c_2 \cdot \begin{pmatrix} \hat{x}_1^2 & \hat{x}_1 \cdot \hat{y}_1 \\ \hat{x}_1 \cdot \hat{y}_1 & \hat{y}_1^2 \end{pmatrix} \quad (5.72) \\ &= \begin{pmatrix} c_1 \cdot \hat{y}_1^2 + c_2 \cdot \hat{x}_1^2 & (c_2 - c_1) \cdot \hat{x}_1 \cdot \hat{y}_1 \\ (c_2 - c_1) \cdot \hat{x}_1 \cdot \hat{y}_1 & c_1 \cdot \hat{x}_1^2 + c_2 \cdot \hat{y}_1^2 \end{pmatrix},\end{aligned}$$

using the conductivity coefficients

$$\begin{aligned}c_1 &= f_1(\lambda_1, \lambda_2) = \frac{1}{(1 + \lambda_1 + \lambda_2)^{a_1}}, \\ c_2 &= f_2(\lambda_1, \lambda_2) = \frac{1}{(1 + \lambda_1 + \lambda_2)^{a_2}},\end{aligned} \quad (5.73)$$

with fixed parameters $a_1, a_2 > 0$ to control the non-isotropy of the filter: a_1 specifies the amount of smoothing along contours, a_2 in perpendicular direction (along the gradient). Small values of a_1, a_2 facilitate diffusion in the corresponding direction, while larger values inhibit smoothing. With a_1 close to zero, diffusion is practically unconstrained along the tangent direction. Typical default values are $a_1 = 0.5$ and $a_2 = 0.9$; results from other settings are shown in the examples.

Step 8: Finally, each image channel I_k is updated using the recurrence relation

$$\begin{aligned}I_k &\leftarrow I_k + \alpha \cdot \text{trace}(\mathbf{A} \cdot \mathbf{H}_k) = I_k + \alpha \cdot \beta_k \\ &= I_k + \alpha \cdot (A_0 \cdot I_{k,xx} + A_1 \cdot I_{k,xy} + A_1 \cdot I_{k,yx} + A_2 \cdot I_{k,yy}) \\ &= I_k + \alpha \cdot \underbrace{(A_0 \cdot I_{k,xx} + 2 \cdot A_1 \cdot I_{k,xy} + A_2 \cdot I_{k,yy})}_{\beta_k}\end{aligned} \quad (5.74)$$

(since $I_{k,xy} = I_{k,yx}$). The term $\beta_k = \text{trace}(\mathbf{A} \cdot \mathbf{H}_k)$ represents the local image *velocity* in channel k . Note that, although a separate Hessian matrix \mathbf{H}_k is calculated for each channel, the structure matrix \mathbf{A} is the same for all image channels. The image is thus smoothed along a common image geometry which considers the correlation between color channels, since \mathbf{A} is derived from the joint structure matrix \mathbf{G} (Eqn. (5.69)) and therefore combines all K color channels.

In each iteration, the factor α in Eqn. (5.74) is adjusted dynamically to the maximum current velocity β_k in all channels in the form

$$\alpha = \frac{d_t}{\max \beta_k} = \frac{d_t}{\max_{k,u} |\text{trace}(\mathbf{A} \cdot \mathbf{H}_k)|}, \quad (5.75)$$

where d_t is the (constant) “time increment” parameter. Thus the time step α is kept small as long as the image gradients (vector field velocities) are large. As smoothing proceeds, image gradients are reduced and thus α typically increases over time. In the actual implementation, the values of I_k (in Eqn. (5.74)) are hard-limited to the initial minimum and maximum.

The above steps (1–8) are repeated for the specified number of iterations. The complete procedure is summarized in [Alg. 5.9](#) and a corresponding Java implementation can be found on the book’s website (see Sec. 5.5).

Beyond this baseline algorithm, several variations and extensions of this filter exist, including the use of spatially-adaptive, oriented smoothing filters.¹⁵ This type of filter has also been used with good results for *image inpainting* [131], where diffusion is applied to fill out only selected (masked) parts of the image where the content is unknown or should be removed.

Example

The example in [Fig. 5.20](#) demonstrates the influence of image geometry and how the non-isotropy of the Tschumperlé-Deriche filter can be controlled by varying the diffusion parameters a_1, a_2 (see Eqn. (5.73)). Parameter a_1 , which specifies the diffusion in the direction of contours, is changed while a_2 (controlling the diffusion in the gradient direction) is held constant. In [Fig. 5.20 \(a\)](#), smoothing along contours is modest and very small across edges with the default settings $a_1 = 0.5$ and $a_2 = 0.9$. With lower values of a_1 , increased blurring occurs in the direction of the contours, as shown in [Figs. 5.20 \(b, c\)](#).

5.4 Measuring image quality

Assessing the effectiveness of a noise-removal filter in terms of image quality is difficult. The criteria commonly applied are either *subjective* or *objective* [47, Sec. 8.1]. Subjective assessment relies on the qualitative judgment of human observers, while objective criteria try to quantify the resulting image quality in mathematical terms. Objective quality measures are usually obtained by adding a certain quantity of synthetic noise to an undisturbed reference image I , that is,

$$\tilde{I} \leftarrow I + I_{\text{noise}}, \quad (5.76)$$

then filtering the noisy image \tilde{I} ,

$$\bar{I} \leftarrow \text{Filter}(\tilde{I}), \quad (5.77)$$

¹⁵ A recent version was released by the original authors as part of the “GREYC’s Magic Image Converter” open-source framework, which is also available as a GIMP plugin (<http://gmic.sourceforge.net>).

Algorithm 5.9 Tschumperlé-Deriche anisotropic diffusion filter for vector-valued (color) images. Typical settings are $T = 5, \dots, 20$, $d_t = 20$, $\sigma_g = 0$, $\sigma_s = 0.5$, $a_1 = 0.5$, $a_2 = 0.9$. See Sec. B.4.1 for a description of the procedure REALEIGENVALUES2X2 (used in line 12).

```

1: TSCHUMPERLEDERICHEFILTER( $\mathbf{I}, T, d_t, \sigma_g, \sigma_s, a_1, a_2$ )
   Input:  $\mathbf{I} = (I_1, \dots, I_K)$ , color image of size  $M \times N$  with  $K$  channels;
           $T$ , number of iterations;  $d_t$ , time increment;  $\sigma_g$ , width of Gaussian
          for smoothing the gradient;  $\sigma_s$ , width of Gaussian for smoothing the
          structure matrix;  $a_1, a_2$ , diffusion parameters for directions of min./
          max. variation, respectively. Returns the modified image  $\mathbf{I}$ .
2: Create maps:
    $D : K \times M \times N \rightarrow \mathbb{R}^2$             $\triangleright D(k, u, v) \equiv \nabla I_k(u, v)$ , grad. vector
    $H : K \times M \times N \rightarrow \mathbb{R}^{2 \times 2}$       $\triangleright H(k, u, v) \equiv \mathbf{H}_k(u, v)$ , Hess. matrix
    $G : M \times N \rightarrow \mathbb{R}^{2 \times 2}$             $\triangleright G(u, v) \equiv \mathbf{G}(u, v)$ , structure matrix
    $A : M \times N \rightarrow \mathbb{R}^{2 \times 2}$             $\triangleright A(u, v) \equiv \mathbf{A}(u, v)$ , geometry matrix
    $B : K \times M \times N \rightarrow \mathbb{R}$             $\triangleright B(k, u, v) \equiv \beta_k(u, v)$ , velocity
3: for  $t \leftarrow 1, \dots, T$  do            $\triangleright$  perform  $T$  iterations
4:   for  $k \leftarrow 1, \dots, K$  and all coordinates  $(u, v) \in M \times N$  do
5:      $D(k, u, v) \leftarrow \begin{pmatrix} (I_k * H_x^\nabla)(u, v) \\ (I_k * H_y^\nabla)(u, v) \end{pmatrix}$             $\triangleright$  Eqns. (5.63–5.64)
6:      $H(k, u, v) \leftarrow \begin{pmatrix} (I_k * H_{xx}^\nabla)(u, v) & (I_k * H_{xy}^\nabla)(u, v) \\ (I_k * H_{xy}^\nabla)(u, v) & (I_k * H_{yy}^\nabla)(u, v) \end{pmatrix}$             $\triangleright$  Eqns. (5.66–5.67)
7:      $D \leftarrow D * H_G^{\sigma_d}$             $\triangleright$  smooth elements of  $D$  over  $(u, v)$ 
8:     for all coordinates  $(u, v) \in M \times N$  do
9:        $G(u, v) \leftarrow \sum_{k=1}^K \begin{pmatrix} (D_x(k, u, v))^2 & D_x(k, u, v) \cdot D_y(k, u, v) \\ D_x(k, u, v) \cdot D_y(k, u, v) & (D_y(k, u, v))^2 \end{pmatrix}$ 
10:       $G \leftarrow G * H_G^{\sigma_g}$             $\triangleright$  smooth elements of  $G$  over  $(u, v)$ 
11:      for all coordinates  $(u, v) \in M \times N$  do
12:         $(\lambda_1, \lambda_2, e_1, e_2) \leftarrow \text{REALEigenvalues2x2}(G(u, v))$             $\triangleright$  p. 310
13:         $\hat{e}_1 \leftarrow \begin{pmatrix} \hat{x}_1 \\ \hat{y}_1 \end{pmatrix} = \frac{e_1}{\|e_1\|}$             $\triangleright$  normalize 1st eigenvector ( $\lambda_1 \geq \lambda_2$ )
14:         $c_1 \leftarrow \frac{1}{(1+\lambda_1+\lambda_2)^{a_1}}, \quad c_2 \leftarrow \frac{1}{(1+\lambda_1+\lambda_2)^{a_2}}$             $\triangleright$  Eqn. (5.73)
15:         $A(u, v) \leftarrow \begin{pmatrix} c_1 \cdot \hat{y}_1^2 + c_2 \cdot \hat{x}_1^2 & (c_2 - c_1) \cdot \hat{x}_1 \cdot \hat{y}_1 \\ (c_2 - c_1) \cdot \hat{x}_1 \cdot \hat{y}_1 & c_1 \cdot \hat{x}_1^2 + c_2 \cdot \hat{y}_1^2 \end{pmatrix}$             $\triangleright$  Eqn. (5.72)
16:         $\beta_{\max} \leftarrow -\infty$ 
17:        for  $k \leftarrow 1, \dots, K$  and all coordinates  $(u, v) \in M \times N$  do
18:           $B(k, u, v) \leftarrow \text{trace}(A(u, v) \cdot H(k, u, v))$             $\triangleright \beta_k$ , Eqn. (5.74)
19:           $\beta_{\max} \leftarrow \max(\beta_{\max}, |B(k, u, v)|)$ 
20:         $\alpha \leftarrow d_t / \beta_{\max}$             $\triangleright$  Eqn. (5.75)
21:        for  $k \leftarrow 1, \dots, K$  and all coordinates  $(u, v) \in M \times N$  do
22:           $I_k(u, v) \leftarrow I_k(u, v) + \alpha \cdot B(k, u, v)$             $\triangleright$  update the image
23: return  $\mathbf{I}$ .

```

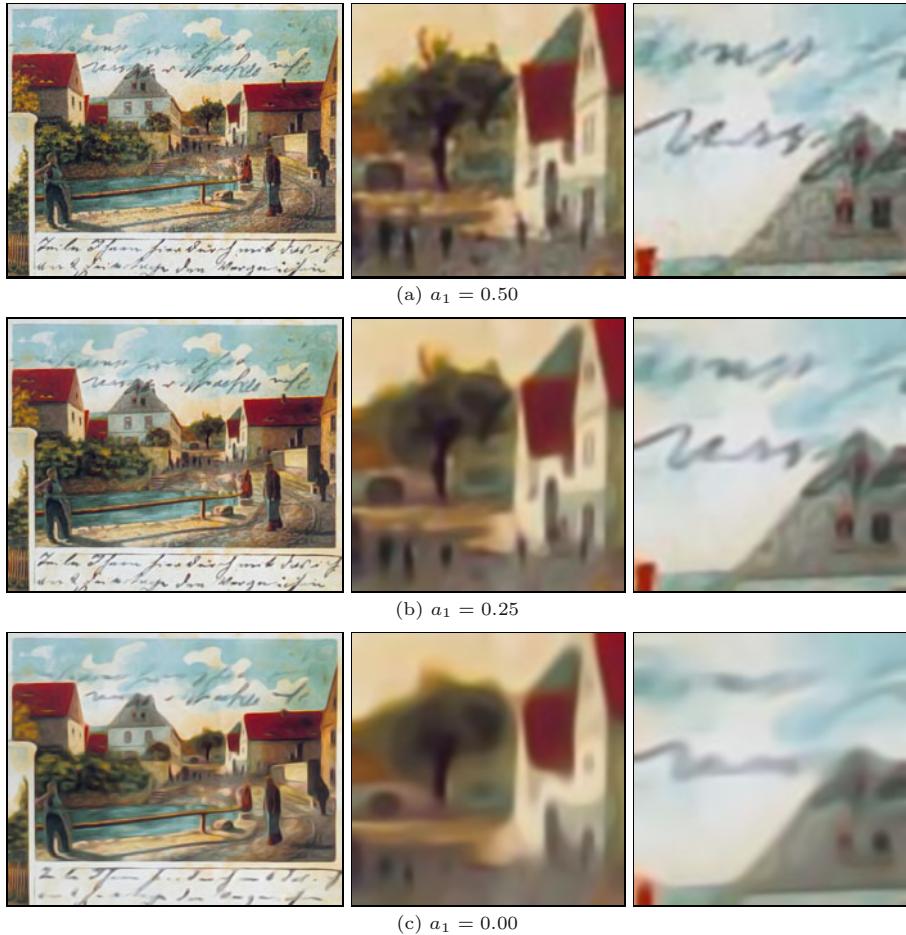


Figure 5.20 Tschumperlé-Deriche filter example. The non-isotropy of the filter can be adjusted by changing parameter a_1 , which controls the diffusion along contours (see Eqn. (5.73)): $a_1 = 0.50, 0.25, 0.00$ (a–c). Parameter $a_2 = 0.90$ (constant) controls the diffusion in the direction of the gradient (perpendicular to contours). Remaining settings are $T = 20$, $d_t = 20$, $\sigma_g = 0.5$, $\sigma_s = 0.5$ (see the description of Alg. 5.9); original image in Fig. 5.3.

and finally comparing the resulting image \bar{I} to the original image I . In this way, the effectiveness of the filter on particular types of noise can be estimated under controlled conditions. Frequently used noise models are *spot noise* and *Gaussian noise* with suitable parameter settings. An example for adding synthetic Gaussian noise to an image is given in Section C.3.3 of the Appendix.

The most common quantity used for measuring image fidelity is the so-called *signal-to-noise ratio* (SNR), which is defined as the ratio between the average signal power P_{signal} and the average noise power P_{noise} . The signal

corresponds to the original image I and the noise is taken as the difference between the “corrupted” image \bar{I} and the original image I . The SNR is then calculated as

$$\text{SNR}(I, \bar{I}) = \frac{P_{\text{signal}}}{P_{\text{noise}}} = \frac{\sum_{u=0}^{M-1} \sum_{v=0}^{N-1} I^2(u, v)}{\sum_{u=0}^{M-1} \sum_{v=0}^{N-1} |I(u, v) - \bar{I}(u, v)|^2}. \quad (5.78)$$

A small difference between \bar{I} and the clean image I results in a higher signal-to-noise ratio, indicating a lower amount of noise. Thus, *large* SNR values indicate that the filter performs well in removing the given type of noise and reconstructing the image. The SNR is commonly given on a logarithmic scale in *decibel* (dB) units, that is,

$$\text{SNR}_{[\text{dB}]}(I, \bar{I}) = 10 \cdot \log_{10}(\text{SNR}(I, \bar{I})). \quad (5.79)$$

A small collection of common quality measures is given in Section C.4 of the Appendix. Note, however, that these quantitative error criteria usually do not correlate well with the subjective judgment of human observers. In particular, the goodness of edge preservation and the impression of sharpness is hardly captured by quantitative error measures and thus their practical relevance is limited.

5.5 Implementation

Implementations of the filters described in this chapter are available with full Java source code at the book’s website.¹⁶ The corresponding classes `KuwaharaFilter`, `NagaoMatsuyamaFilter`, `PeronaMalikFilter` and `Tschumperle-DericheFilter` are based on the common super-class `GenericFilter`:¹⁷

`KuwaharaFilter(int r, double tsigma)`

Creates a Kuwahara-type filter for grayscale and color images, as described in Section 5.1 (Alg. 5.2), with radius `r` (default 2) and variance threshold `tsigma` (denoted t_σ in Alg. 5.2, default 0.0). The size of the resulting filter is $(2r+1) \times (2r+1)$.

`BilateralFilter(double sigmaD, double sigmaR)`

Creates a Bilateral filter for grayscale and color images using Gaussian kernels, as described in Section 5.2 (Algs. 5.4 and 5.5). Parameters `sigmaD` (σ_d , default 2.0) and `sigmaR` (σ_r , default 50.0) specify the widths

¹⁶ Package `imagingbook.edgepreservingfilters`

¹⁷ Package `imagingbook.filters`. Filters of this type can be applied to images using the method `applyTo(ImageProcessor ip)`, as described in Section 3.3.

of the domain and the range kernels, respectively. The distance norm to be used for measuring color differences can be specified (after instantiation) with `setColorDistanceNorm()`.

`BilateralFilterSeparable(double sigmaD, double sigmaR)`

Creates a x/y -separable Bilateral filter for grayscale and color images, as described in Section 5.2.4 (Alg. 5.6). Constructor parameters are the same as for the class `BilateralFilter` above.

`PeronaMalikFilter(int iterations, double alpha, double kappa, boolean smoothRegions)`

Creates an anisotropic diffusion filter for grayscale and color images, as described in Section 5.3.2 (Algs. 5.7 and 5.8). Parameters are `iterations` (T , default 10), `alpha` (α , default 0.2), `kappa` (κ , default 25). If `smoothRegions` is set false, $g_\kappa^{(1)}$ is used as the conductivity function, otherwise $g_\kappa^{(2)}$ (see Eqn. (5.47)). For filtering color images, three different color modes can be specified for diffusion control: `SeparateChannels`, `BrightnessGradient` or `ColorGradient`. See Prog. 5.1 for an example of using this class in a simple ImageJ plugin.

`TschumperleDericheFilter(int iterations, double dt, double sigmaG, double sigmaS, float a1, float a2)`

Creates an anisotropic diffusion filter for color images, as described in Section 5.3.4 (Alg. 5.9). Parameters are `iterations` (T , default 5), `dt` (d_t , default 20), `sigmaG` (σ_g , default 0.0), `sigmaS` (σ_s , default 0.5), `a1` (a_1 , default 0.5), `a2` (a_2 , default 0.9). Otherwise the usage of this class is analogous to the example in Prog. 5.1.

All default values pertain to the parameter-less constructors that are also available. Note that most of these filters are generic and can be applied to grayscale and color images without any modification.

5.6 Exercises

Exercise 5.1

Implement a pure *range filter* (Eqn. (5.16)) for grayscale images, using a 1D Gaussian kernel $H_r(x) = \frac{1}{\sqrt{2\pi}\cdot\sigma} \cdot \exp(-\frac{x^2}{2\sigma^2})$. Investigate the effects of this filter for $\sigma = 10, 20$, and 25 upon the image and its histogram.

Exercise 5.2

Create an ImageJ plugin (or other suitable implementation) to produce a Gaussian noise image with mean value m and standard deviation s , based on the description in Section C.3.3 of the Appendix. Inspect the histogram of the resulting image to see if the distribution of pixel values is Gaussian. Calculate the mean (μ) and the variance (σ^2) of the resulting pixel values and verify that $\mu \approx m$ and $\sigma^2 \approx s^2$.

```

1 import ij.ImagePlus;
2 import ij.plugin.filter.PlugInFilter;
3 import ij.process.ImageProcessor;
4 import edgepreservingfilters.PeronaMalikFilter;
5 import edgepreservingfilters.PeronaMalikFilter.ColorMode;
6 import edgepreservingfilters.PeronaMalikFilter.Parameters;
7
8 public class AD_Filter_PeronaMalik_Book implements PlugInFilter {
9
10    public int setup(String arg0, ImagePlus imp) {
11        return DOES_ALL + DOES_STACKS;
12    }
13
14    public void run(ImageProcessor ip) {
15        // create a parameter object:
16        Parameters params = new Parameters();
17
18        // modify filter settings if needed:
19        params.iterations = 20;
20        params.alpha = 0.15f;
21        params.kappa = 20.0f;
22        params.smoothRegions = true;
23        params.colorMode = ColorMode.ColorGradient;
24        params.useLinearRgb = false;
25
26        // instantiate the filter object:
27        PeronaMalikFilter filter = new PeronaMalikFilter(params);
28
29        // apply the filter:
30        filter.applyTo(ip);
31    }
32 }
```

Program 5.1 Perona-Malik filter (complete ImageJ plugin). Inside the `run()` method, a parameter object (instance of class `PeronaMalikFilter.Parameters`) is created in line 16. Individual parameters may then be modified, as shown in lines 19–24. This would typically be done by querying the user (e.g., with ImageJ’s `GenericDialog` class). In line 27, a new instance of `PeronaMalikFilter` is created, the parameter object (`params`) being passed to the constructor as the only argument. Finally, in line 30, the filter is (destructively) applied to the input image, i.e., `ip` is modified. `ColorMode` (in line 23) is implemented as an enumeration type within class `PeronaMalikFilter`, providing the options `SeparateChannels` (default), `BrightnessGradient` and `ColorGradient`. Note that, as specified in the `setup()` method, this plugin works for any type of image and image stacks.

Exercise 5.3

Apply the tool developed in Exercise 5.2 to create two Gaussian noise images: $I_1 \sim \mathcal{N}(\mu_1, \sigma_1)$ and $I_2 \sim \mathcal{N}(\mu_2, \sigma_2)$. The sum of these two images, $I_3 = I_1 + I_2$, should again show a Gaussian distribution $\mathcal{N}(\mu_3, \sigma_3)$. Use Eqn. (C.17) to predict the parameters μ_3, σ_3 , given $\mu_1 = 40$, $\sigma_1 = 6$ and $\mu_2 = 70$, $\sigma_2 = 10$. Inspect the histogram of I_3 and verify the result by calculating the mean and variance of the image data.

Exercise 5.4

Modify the Kuwahara-type filter for color images in [Alg. 5.3](#) to use the *norm of the color covariance matrix* (as defined in Eqn. (5.12)) for quantifying the amount of variation in each subregion. Estimate the number of additional calculations required for processing each image pixel. Implement the modified algorithm, compare the results and performance.

Exercise 5.5

Modify the separable Bilateral filter algorithm (given in [Alg. 5.6](#)) to handle color images, using [Alg. 5.5](#) as a starting point. Implement and test your algorithm, compare the results (see also [Fig. 5.12](#)) and performance.

Exercise 5.6 (experimental)

Use the signal-to-noise ratio (SNR) to measure the effectiveness of noise suppression by edge-preserving smoothing filters on grayscale images. Add synthetic Gaussian noise (see Sec. C.3.3) to the original image I to create a corrupted image \tilde{I} . Then apply the filter to \tilde{I} to obtain \bar{I} . Finally, calculate $\text{SNR}(I, \bar{I})$ as defined in Eqn. (5.78). Compare the SNR values obtained with various types of filters and different parameter settings, for example, for the *Kuwahara filter* ([Alg. 5.2](#)), the *Bilateral filter* ([Alg. 5.4](#)), and the *Perona-Malik* anisotropic diffusion filter ([Alg. 5.7](#)). Analyze if and how the SNR values relate to the perceived image quality.

6

Fourier Shape Descriptors

Fourier descriptors are an interesting method for modeling 2D shapes that are described as closed contours. Unlike polylines or splines, which are explicit and local descriptions of the contour, Fourier descriptors are *global* shape representations, i.e., each component stands for a particular characteristic of the entire shape. If one component is changed, the whole shape will change. The advantage is that it is possible to capture coarse shape properties with only a few numeric values, and the level of detail can be increased (or decreased) by adding (or removing) descriptor elements. In the following, we describe what is called “cartesian” (or “elliptical”) Fourier descriptors, how they can be used to model the shape of closed 2D contours and how they can be adapted to compare shapes in a translation-, scale- and rotation-invariant fashion.

6.1 2D boundaries in the complex plane

6.1.1 Parameterized boundary curves

Any continuous curve C in the 2D plane can be expressed as a function $f: \mathbb{R} \rightarrow \mathbb{R}^2$, with

$$f(t) = \begin{pmatrix} x_t \\ y_t \end{pmatrix} = \begin{pmatrix} f_x(t) \\ f_y(t) \end{pmatrix}, \quad (6.1)$$

with the continuous parameter t being varied over the range $[0, t_{\max}]$. If the curve is closed, then $f(0) = f(t_{\max})$ and $f(t) = f(t + t_{\max})$. Note that $f_x(t)$, $f_y(t)$ are independent, real-valued functions, and t is typically the *path length* along the curve.

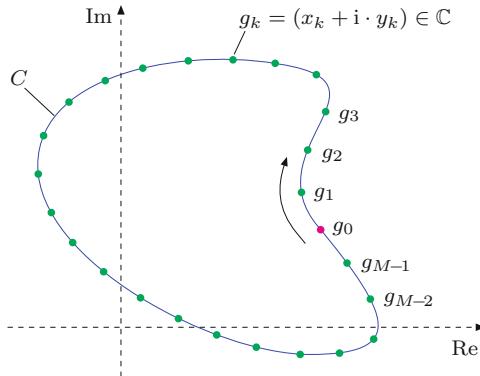


Figure 6.1 A closed, continuous 2D curve C , represented as a sequence of M uniformly placed samples $\mathbf{g} = (g_0, g_1, \dots, g_{M-1})$ in the complex plane.

6.1.2 Discrete 2D boundaries

Sampling a closed curve C at M regularly spaced positions t_0, t_1, \dots, t_{M-1} , with $t_i - t_{i-1} = \Delta_t = \text{Length}(C)/M$, results in a sequence (vector) of discrete 2D coordinates $V = (\mathbf{v}_0, \mathbf{v}_1, \dots, \mathbf{v}_{M-1})$, with

$$\mathbf{v}_k = (x_k, y_k) = f(t_k). \quad (6.2)$$

Since the curve C is closed, the vector V represents a discrete function that is infinite and periodic, i.e.,

$$\mathbf{v}_k = \mathbf{v}_{k+pM}, \quad (6.3)$$

for $0 \leq k < M$ and any $p \in \mathbb{Z}$.

Points in the complex plane

Any 2D contour sample $\mathbf{v}_k = (x_k, y_k)$ can be interpreted as a point g_k in the complex plane,

$$g_k = x_k + i \cdot y_k, \quad (6.4)$$

with x_k and y_k taken as the real and imaginary components, respectively.¹ The result is a sequence (vector) of complex values

$$\mathbf{g} = (g_0, g_1, \dots, g_{M-1}), \quad (6.5)$$

representing the discrete 2D contour (see Fig. 6.1).

¹ Instead of $g \leftarrow x + i \cdot y$, we sometimes use the short notation $g \leftarrow (x, y)$ or $g \leftarrow \mathbf{v}$ for assigning the components of a 2D vector $\mathbf{v} = (x, y) \in \mathbb{R}^2$ to a complex variable $g \in \mathbb{C}$.

Regular sampling

The assumption of input data being obtained by regular sampling is quite fundamental in traditional discrete Fourier analysis. In practice, contours of objects are typically not available as regularly sampled point sequences. For example, if an object has been segmented as a binary region, the coordinates of its boundary pixels could be used as the original contour sequence. However, the number of boundary pixels is usually too large to be used directly and their positions are not strictly uniformly spaced (at least under 8-connectivity). To produce a useful contour sequence from a region boundary, one could choose an arbitrary contour point as the start position x_0 and then sample the x/y positions along the contour at regular (equidistant) steps, treating the centers of the boundary pixels as the vertices of a closed polygon. [Algorithm 6.1](#) shows how to calculate a predefined number of contour points on an arbitrary polygon, such that the path length between the sample points is uniform. This algorithm is used in all examples involving contours obtained from binary regions.

Note that if the shape is given as an arbitrary polygon, the corresponding Fourier descriptor can also be calculated directly (and exactly) from the vertices of the polygon, without sub-sampling the polygon contour path at all. This “trigonometric” variant of the Fourier descriptor calculation is described in Section 6.3.7.

6.2 Discrete Fourier transform

Fourier descriptors are obtained by applying the one-dimensional Discrete Fourier Transform (DFT)² to the complex-valued vector \mathbf{g} of 2D contour points (Eqn. (6.5)). The DFT is a transformation of a finite, complex-valued *signal* vector $\mathbf{g} = (g_0, g_1, \dots, g_{M-1})$ to a complex-valued *spectrum* $\mathbf{G} = (G_0, G_1, \dots, G_{M-1})$.³ Both the signal and the spectrum are of the same length (M) and periodic. In the following, we typically use k to denote the index in the time or space domain,⁴ and m for a frequency index in the spectral domain.

² See Vol. 2 [21, Sec. 7.3] for a basic introduction.

³ In most traditional applications of the DFT (e.g. in acoustic processing), the signals are real-valued, i.e., the imaginary components of the samples are zero. The Fourier spectrum is generally complex-valued, but it is symmetric for real-valued signals.

⁴ We use k instead of the usual i as the running index to avoid confusion with the imaginary constant “i” (despite the deliberate use of different glyphs).

Algorithm 6.1 Regular sampling of a polygon path. Given a sequence V of 2D points representing the vertices of a closed polygon, SAMPLEPOLYGONUNIFORMLY(V, M) returns a sequence of M complex values \mathbf{g} on the polygon V , such that $\mathbf{g}(0) \equiv V(0)$ and all remaining points $\mathbf{g}(k)$ are uniformly positioned along the polygon path. See Alg. 6.9 (p. 226) for an alternate solution.

```

1: SAMPLEPOLYGONUNIFORMLY( $V, M$ )
   Input:  $V = (\mathbf{v}_0, \dots, \mathbf{v}_{N-1})$ , a sequence of  $N$  points representing the
         vertices of a 2D polygon;  $M$ , number of desired sample points.
   Returns a sequence  $\mathbf{g} = (g_0, \dots, g_{M-1})$  of complex values representing
         points sampled uniformly along the path of the input polygon  $V$ .
2:  $N \leftarrow |V|$ 
3:  $\Delta \leftarrow \frac{1}{M} \cdot \text{PATHLENGTH}(V)$                                  $\triangleright$  const. segment length  $\Delta$ 
4: Create map  $\mathbf{g}: [0, M-1] \rightarrow \mathbb{C}$                                  $\triangleright$  complex point sequence  $\mathbf{g}$ 
5:  $\mathbf{g}(0) \leftarrow \text{COMPLEX}(V(0))$ 
6:  $i \leftarrow 0$                                           $\triangleright$  index of polygon segment  $\langle \mathbf{v}_i, \mathbf{v}_{i+1} \rangle$ 
7:  $k \leftarrow 1$                                           $\triangleright$  index of next point to be added to  $\mathbf{g}$ 
8:  $\alpha \leftarrow 0$                                           $\triangleright$  path position of polygon vertex  $\mathbf{v}_i$ 
9:  $\beta \leftarrow \Delta$                                           $\triangleright$  path position of next point to be added to  $\mathbf{g}$ 
10: while ( $i < N$ )  $\wedge$  ( $k < M$ ) do
11:    $\mathbf{v}_A \leftarrow V(i)$ 
12:    $\mathbf{v}_B \leftarrow V((i + 1) \bmod N)$ 
13:    $\delta \leftarrow \|\mathbf{v}_B - \mathbf{v}_A\|$                                  $\triangleright$  length of segment  $\langle \mathbf{v}_A, \mathbf{v}_B \rangle$ 
14:   while ( $\beta \leq \alpha + \delta$ )  $\wedge$  ( $k < M$ ) do
15:      $\mathbf{x} \leftarrow \mathbf{v}_A + \frac{\beta - \alpha}{\delta} \cdot (\mathbf{v}_B - \mathbf{v}_A)$        $\triangleright$  linear path interpolation
16:      $\mathbf{g}(k) \leftarrow \text{COMPLEX}(\mathbf{x})$ 
17:      $k \leftarrow k + 1$ 
18:      $\beta \leftarrow \beta + \Delta$ 
19:      $\alpha \leftarrow \alpha + \delta$ 
20:    $i \leftarrow i + 1$ 
21: return  $\mathbf{g}$ .
```

```

22: PATHLENGTH( $V$ )     $\triangleright$  returns the path length of the closed polygon  $V$ 
23:    $N \leftarrow |V|$ 
24:    $L \leftarrow 0$ 
25:   for  $i \leftarrow 0, \dots, N-1$  do
26:      $\mathbf{v}_A \leftarrow V(i)$ 
27:      $\mathbf{v}_B \leftarrow V((i + 1) \bmod N)$ 
28:      $L \leftarrow L + \|\mathbf{v}_B - \mathbf{v}_A\|$ 
29: return  $L$ .
```

6.2.1 Forward transform

The discrete Fourier spectrum $\mathbf{G} = (G_0, G_1, \dots, G_{M-1})$ is computed from the discrete, complex-valued signal $\mathbf{g} = (g_0, g_1, \dots, g_{M-1})$ using the forward DFT, defined as⁵

$$G_m = \frac{1}{M} \cdot \sum_{k=0}^{M-1} g_k \cdot e^{-i2\pi m \frac{k}{M}} = \frac{1}{M} \cdot \sum_{k=0}^{M-1} g_k \cdot e^{-i\omega_m \frac{k}{M}} \quad (6.6)$$

$$= \frac{1}{M} \cdot \sum_{k=0}^{M-1} \underbrace{\left[x_k + i \cdot y_k \right]}_{g_k} \cdot \left[\cos\left(\underbrace{2\pi m \frac{k}{M}}_{\omega_m}\right) - i \cdot \sin\left(\underbrace{2\pi m \frac{k}{M}}_{\omega_m}\right) \right] \quad (6.7)$$

$$= \frac{1}{M} \cdot \sum_{k=0}^{M-1} \left[x_k + i \cdot y_k \right] \cdot \left[\cos\left(\omega_m \frac{k}{M}\right) - i \cdot \sin\left(\omega_m \frac{k}{M}\right) \right], \quad (6.8)$$

for $0 \leq m < M$.⁶ Note that $\omega_m = 2\pi m$ denotes the *angular frequency* for the frequency index m . By applying the usual rules of complex multiplication, we obtain the *real* (Re) and *imaginary* (Im) parts of the spectral coefficients $G_m = (A_m + i \cdot B_m)$ explicitly as

$$\begin{aligned} A_m &= \text{Re}(G_m) = \frac{1}{M} \sum_{k=0}^{M-1} \left[x_k \cdot \cos\left(\omega_m \frac{k}{M}\right) + y_k \cdot \sin\left(\omega_m \frac{k}{M}\right) \right], \\ B_m &= \text{Im}(G_m) = \frac{1}{M} \sum_{k=0}^{M-1} \left[y_k \cdot \cos\left(\omega_m \frac{k}{M}\right) - x_k \cdot \sin\left(\omega_m \frac{k}{M}\right) \right]. \end{aligned} \quad (6.9)$$

The DFT is defined for any signal length $M \geq 1$. If the signal length M is a power of two (i.e., $M = 2^n$ for some $n \in \mathbb{N}$), the Fast Fourier Transform (FFT)⁷ can be used in place of the DFT for improved performance.

6.2.2 Inverse Fourier transform (reconstruction)

The inverse DFT reconstructs the original signal \mathbf{g} from a given spectrum \mathbf{G} . The formulation is almost symmetrical (except for the scale factor and the different signs in the exponent) to the forward transformation in Eqns. (6.6–6.8); its full expansion is

$$g_k = \sum_{m=0}^{M-1} G_m \cdot e^{i2\pi m \frac{k}{M}} = \sum_{m=0}^{M-1} G_m \cdot e^{i\omega_m \frac{k}{M}} \quad (6.10)$$

⁵ This definition deviates slightly from the one used in Vol. 2 [21, Sec. 7.3] but is otherwise equivalent.

⁶ Recall that $z = x + iy = |z| \cdot (\cos \psi + i \cdot \sin \psi) = |z| \cdot e^{i\psi}$, with $\psi = \tan^{-1}(y/x)$.

⁷ See Vol. 2 [21, Sec. 7.4.2].

Algorithm 6.2 Calculating the Fourier descriptor for a sequence of uniformly sampled contour points. The complex-valued contour points in C represent 2D positions sampled uniformly along the contour path. Applying the DFT to \mathbf{g} yields the raw Fourier descriptor \mathbf{G} .

```

1: FOURIERDESCRIPTORUNIFORM( $\mathbf{g}$ )
   Input:  $\mathbf{g} = (g_0, \dots, g_{M-1})$ , a sequence of  $M$  complex values, representing regularly sampled 2D points along a contour path.
   Returns a Fourier descriptor  $\mathbf{G}$  of length  $M$ .
2:  $M \leftarrow |\mathbf{g}|$ 
3: Create map  $\mathbf{G}: [0, M-1] \rightarrow \mathbb{C}$ 
4: for  $m \leftarrow 0, \dots, M-1$  do
5:    $A \leftarrow 0, B \leftarrow 0$             $\triangleright$  real/imag. part of coefficient  $G_m$ 
6:   for  $k \leftarrow 0, \dots, M-1$  do
7:      $g \leftarrow \mathbf{g}(k)$ 
8:      $x \leftarrow \text{Re}(g), y \leftarrow \text{Im}(g)$ 
9:      $\phi \leftarrow 2 \cdot \pi \cdot m \cdot \frac{k}{M}$ 
10:     $A \leftarrow A + x \cdot \cos(\phi) + y \cdot \sin(\phi)$             $\triangleright$  Eqn. (6.9)
11:     $B \leftarrow B - x \cdot \sin(\phi) + y \cdot \cos(\phi)$ 
12:     $\mathbf{G}(m) \leftarrow \frac{1}{M} \cdot (A + i \cdot B)$ 
13: return  $\mathbf{G}$ .

```

$$= \sum_{m=0}^{M-1} \underbrace{\left[\text{Re}(G_m) + i \cdot \text{Im}(G_m) \right]}_{G_m} \cdot \left[\cos\left(2\pi m \frac{k}{M}\right) + i \cdot \sin\left(2\pi m \frac{k}{M}\right) \right] \quad (6.11)$$

$$= \sum_{m=0}^{M-1} \left[A_m + i \cdot B_m \right] \cdot \left[\cos\left(\omega_m \frac{k}{M}\right) + i \cdot \sin\left(\omega_m \frac{k}{M}\right) \right]. \quad (6.12)$$

Again we can expand Eqn. (6.12) to obtain the real and imaginary parts of the reconstructed signal, i. e., the x/y -components of the corresponding curve points $g_k = (x_k, y_k)$ as

$$\begin{aligned} x_k &= \text{Re}(g_k) = \sum_{m=0}^{M-1} \left[\text{Re}(G_m) \cdot \cos\left(2\pi m \frac{k}{M}\right) - \text{Im}(G_m) \cdot \sin\left(2\pi m \frac{k}{M}\right) \right], \\ y_k &= \text{Im}(g_k) = \sum_{m=0}^{M-1} \left[\text{Im}(G_m) \cdot \cos\left(2\pi m \frac{k}{M}\right) + \text{Re}(G_m) \cdot \sin\left(2\pi m \frac{k}{M}\right) \right], \end{aligned} \quad (6.13)$$

for $0 \leq k < M$. If *all* coefficients of the spectrum are used, this reconstruction is *exact*, i. e., the resulting discrete points g_k are identical to the original contour points.⁸

⁸ Apart from inaccuracies caused by finite floating-point precision.

With the above formulation we cannot only reconstruct the discrete contour points g_k from the DFT spectrum, but also a smooth, interpolating curve as the sum of continuous sine and cosine components. To calculate *arbitrary* points on this curve, we replace the discrete quantity $\frac{k}{M}$ in Eqn. (6.13) by the continuous parameter t in the range $[0, 1]$. We must be careful about the frequencies, though. To get the desired *smooth* interpolation, the set of *lowest* possible frequencies ω_m must be used,⁹ that is,

$$\begin{aligned} x(t) &= \sum_{m=0}^{M-1} [\operatorname{Re}(G_m) \cdot \cos(\omega_m \cdot t) - \operatorname{Im}(G_m) \cdot \sin(\omega_m \cdot t)], \\ y(t) &= \sum_{m=0}^{M-1} [\operatorname{Im}(G_m) \cdot \cos(\omega_m \cdot t) + \operatorname{Re}(G_m) \cdot \sin(\omega_m \cdot t)], \end{aligned} \quad (6.14)$$

with

$$\omega_m = \begin{cases} 2\pi m & \text{for } m \leq (M \div 2), \\ 2\pi(m-M) & \text{for } m > (M \div 2), \end{cases} \quad (6.15)$$

where \div denotes the quotient (integer division). Alternatively, we can write Eqn. (6.14) in the form

$$\begin{aligned} x(t) &= \sum_{\substack{m=-(M-1)\div 2 \\ m=-\frac{M}{2}}}^{\frac{M}{2}} [\operatorname{Re}(G_{m \bmod M}) \cdot \cos(2\pi mt) - \operatorname{Im}(G_{m \bmod M}) \cdot \sin(2\pi mt)], \\ y(t) &= \sum_{\substack{m=-(M-1)\div 2 \\ m=-\frac{M}{2}}}^{\frac{M}{2}} [\operatorname{Im}(G_{m \bmod M}) \cdot \cos(2\pi mt) + \operatorname{Re}(G_{m \bmod M}) \cdot \sin(2\pi mt)]. \end{aligned} \quad (6.16)$$

This formulation is used for the shape reconstruction from Fourier descriptors in [Alg. 6.4](#) (see p. 193).

[Figure \(6.2\)](#) shows the reconstruction of the discrete contour points as well as the calculation of a continuous outline from the DFT spectrum obtained from a sequence of discrete contour positions. The original sample points were taken at $M = 25$ uniformly spaced positions along the region's contour. The discrete points in [Fig. 6.2 \(b\)](#) are exactly reconstructed from the complete DFT spectrum, as specified in Eqn. (6.13). The interpolated (green) outline in [Fig. 6.2 \(c\)](#) was calculated with Eqn. (6.13) for continuous positions, based on the frequencies $m = 0, \dots, M-1$. The oscillations of the resulting curve are explained

⁹ Due to the periodicity of the discrete spectrum, any summation over M successive frequencies ω_m can be used to reconstruct the original discrete x/y samples. However, a smooth interpolation between the discrete x/y samples can only be obtained from the set of *lowest* frequencies in the range $[-\frac{M}{2}, +\frac{M}{2}]$ centered around the zero frequency, as in Eqns. (6.14) and (6.16).

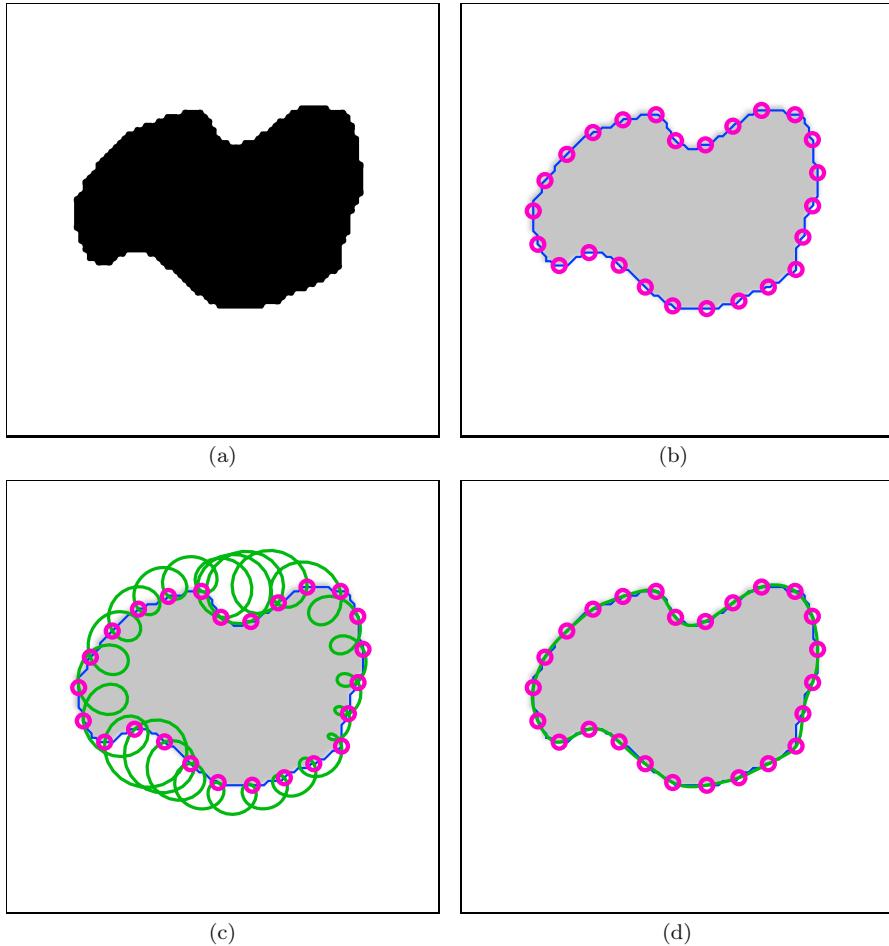


Figure 6.2 Contour reconstruction by inverse DFT. Original image (a), $M = 25$ uniformly spaced sample points on the region's contour (b). Continuous contour (green line) reconstructed by using frequencies ω_m with $m = 0, \dots, 24$ (c). Note that despite the oscillations introduced by the high frequencies, the continuous contour passes exactly through the original sample points. Smooth interpolation reconstructed with Eqn. (6.14) from the lowest-frequency coefficients in the symmetric range $m = -(M - 1) \div 2, \dots, M \div 2$ (d).

by the high-frequency components. Note that the curve still passes exactly through each of the original sample points, in fact, these can be perfectly reconstructed from *any* contiguous range of M coefficients and the corresponding harmonic frequencies. The smooth interpolation in Fig. 6.2(d), based on the symmetric low-frequency coefficients $m = -(M - 1) \div 2, \dots, M \div 2$ (see Eqn. (6.16)) shows no such oscillations, since no high-frequency components are included.

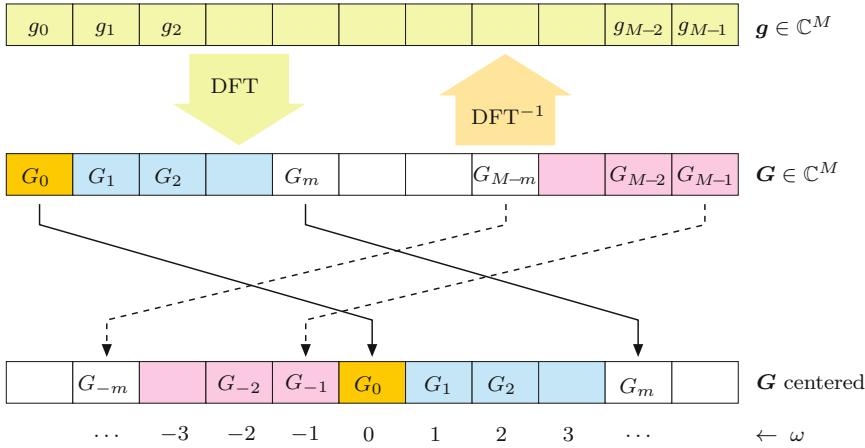


Figure 6.3 Applying the DFT to a complex-valued vector \mathbf{g} of length M yields the complex-valued spectrum \mathbf{G} that is also of length M . The DFT spectrum is infinite and periodic with M , thus $G_{-m} = G_{M-m}$, as illustrated by the centered representation of the DFT spectrum (bottom). ω at the bottom denotes the harmonic number (multiple of the fundamental frequency) associated with each coefficient.

6.2.3 Periodicity of the DFT spectrum

When we apply the DFT, we implicitly assume that both the signal vector $\mathbf{g} = (g_0, g_1, \dots, g_{M-1})$ and the spectral vector $\mathbf{G} = (G_0, G_1, \dots, G_{M-1})$ represent discrete, periodic functions of infinite extent (see [19, Ch. 13] for details). Due to this periodicity, $\mathbf{G}(0) = \mathbf{G}(M)$, $\mathbf{G}(1) = \mathbf{G}(M+1)$, etc. In general

$$\mathbf{G}(q \cdot M + m) = \mathbf{G}(m) \quad \text{and} \quad \mathbf{G}(m) = \mathbf{G}(m \bmod M), \quad (6.17)$$

for arbitrary integers $q, m \in \mathbb{Z}$. Also, since $(-m \bmod M) = (M-m) \bmod M$, we can state that

$$\mathbf{G}(-m) = \mathbf{G}(M-m), \quad (6.18)$$

for any $m \in \mathbb{Z}$, such that $\mathbf{G}(-1) = \mathbf{G}(M-1)$, $\mathbf{G}(-2) = \mathbf{G}(M-2)$, \dots , as illustrated in Fig. 6.3.

6.2.4 Truncating the DFT spectrum

In the original formulation in Eqns. (6.6–6.8), the DFT is applied to a signal \mathbf{g} of length M and yields a discrete Fourier spectrum \mathbf{G} with M coefficients. Thus the signal and the spectrum have the same length. For shape representation, it is often useful to work with a truncated spectrum, i. e., a reduced number of low-frequency Fourier coefficients.

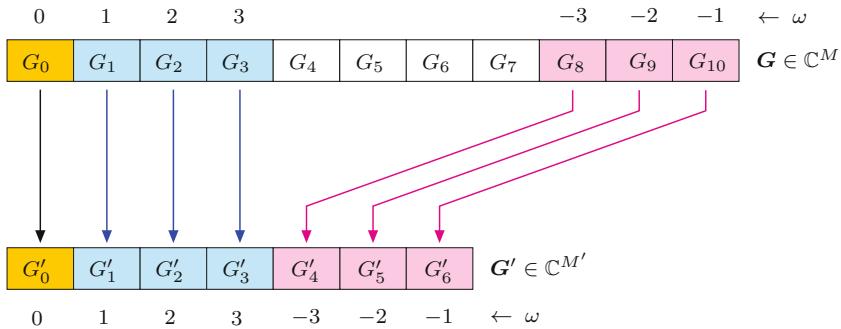


Figure 6.4 Truncating a DFT spectrum from $M = 11$ to $M' = 7$ coefficients, as specified in Eqns. (6.19–6.20). Coefficients G_4, \dots, G_7 are discarded ($M' \div 2 = 3$). Note that the associated harmonic number ω remains the same for each coefficient.

By truncating a spectrum we mean the removal of coefficients above a certain harmonic number, which are (considering positive and negative frequencies) located around the center of the coefficient vector. Truncating a given spectrum \mathbf{G} of length $|\mathbf{G}| = M$ to a shorter spectrum \mathbf{G}' of length $M' \leq M$ is done as

$$\mathbf{G}'(m) \leftarrow \begin{cases} \mathbf{G}(m) & \text{for } 0 \leq m \leq M' \div 2, \\ \mathbf{G}(M - M' + m) & \text{for } M' \div 2 < m < M', \end{cases} \quad (6.19)$$

or, alternatively,

$$\mathbf{G}'(m \bmod M') \leftarrow \mathbf{G}(m \bmod M), \quad (6.20)$$

for $(M' \div 2 - M' + 1) \leq m \leq (M' \div 2)$. This works for M and M' being even or odd. The example in Fig. 6.4 illustrates how an original DFT spectrum \mathbf{G} of length $M = 11$ is truncated to \mathbf{G}' with only $M' = 7$ coefficients.

Of course it is also possible to calculate the truncated spectrum directly from the contour samples, without going through the full DFT spectrum. With M being the length of the signal vector \mathbf{g} and $M' \leq M$ the desired length of the (truncated) spectrum \mathbf{G}' , Eqn. (6.6) modifies to

$$\mathbf{G}'(m \bmod M') = \frac{1}{M} \cdot \sum_{k=0}^{M-1} g_k \cdot e^{-i2\pi m \frac{k}{M}}, \quad (6.21)$$

for m in the same range as in Eqn. (6.20). This approach is more efficient than truncating the complete spectrum, since unneeded coefficients are never calculated. Algorithm 6.3, which is a modified version of Alg. 6.2, summarizes the steps described above.

Algorithm 6.3 Calculating a truncated Fourier descriptor for a sequence of uniformly sampled contour points (adapted from Alg. 6.2). The M complex-valued contour points in \mathbf{g} represent 2D positions sampled uniformly along the contour path. The resulting Fourier descriptor \mathbf{G} contains only M' coefficients for the M' lowest harmonic frequencies.

```

1: FOURIERDESCRIPTORUNIFORM( $\mathbf{g}, M'$ )
   Input:  $\mathbf{g} = (g_0, \dots, g_{M-1})$ , a sequence of  $M$  complex values, representing regularly sampled 2D points along a contour path.
           $M'$ , the number of Fourier coefficients ( $M' \leq M$ ).
          Returns a truncated Fourier descriptor  $\mathbf{G}$  of length  $M'$ .
2:  $M \leftarrow |\mathbf{g}|$ 
3: Create map  $\mathbf{G}: [0, M'-1] \rightarrow \mathbb{C}$ 
4: for  $m \leftarrow (M' \div 2 - M' + 1), \dots, (M' \div 2)$  do
5:    $A \leftarrow 0, B \leftarrow 0$                                  $\triangleright$  real/imag. part of coefficient  $G_m$ 
6:   for  $k \leftarrow 0, \dots, M-1$  do
7:      $g \leftarrow \mathbf{g}(k)$ 
8:      $x \leftarrow \text{Re}(g), y \leftarrow \text{Im}(g)$ 
9:      $\phi \leftarrow 2 \cdot \pi \cdot m \cdot \frac{k}{M}$ 
10:     $A \leftarrow A + x \cdot \cos(\phi) + y \cdot \sin(\phi)$             $\triangleright$  Eqn. (6.9)
11:     $B \leftarrow B - x \cdot \sin(\phi) + y \cdot \cos(\phi)$ 
12:     $\mathbf{G}(m \bmod M') \leftarrow \frac{1}{M} \cdot (A + i \cdot B)$ 
13: return  $\mathbf{G}$ .
```

Since some of the coefficients are missing, it is not possible to reconstruct the original signal vector \mathbf{g} from the truncated DFT spectrum \mathbf{G}' . However, the calculation of a partial reconstruction is possible, for example, using the formulation in Eqn. (6.16). In this case, the discarded (high-frequency) coefficients are simply assumed to have zero values (see Sec. 6.3.6 for more details).

6.3 Geometric interpretation of Fourier coefficients

The contour reconstructed by the inverse transformation (Eqn. (6.13)) is the sum of M terms, one for each Fourier coefficient $G_m = (A_m, B_m)$. Each of these M terms represents a particular 2D shape in the spatial domain and the original contour can be obtained by point-wise addition of the individual shapes. So what are the spatial shapes that correspond to the individual Fourier coefficients?

6.3.1 Coefficient G_0 corresponds to the contour's centroid

We first look only at the specific Fourier coefficient G_0 with frequency index $m = 0$. Substituting $m = 0$ and $\omega_0 = 0$ in Eqn. (6.9), we get

$$\begin{aligned} A_0 &= \frac{1}{M} \sum_{k=0}^{M-1} [x_k \cdot \cos(0) + y_k \cdot \sin(0)] \\ &= \frac{1}{M} \sum_{k=0}^{M-1} [x_k \cdot 1 + y_k \cdot 0] = \frac{1}{M} \sum_{k=0}^{M-1} x_k = \bar{x}, \end{aligned} \quad (6.22)$$

$$\begin{aligned} B_0 &= \frac{1}{M} \sum_{k=0}^{M-1} [y_k \cdot \cos(0) - x_k \cdot \sin(0)] \\ &= \frac{1}{M} \sum_{k=0}^{M-1} [y_k \cdot 1 - x_k \cdot 0] = \frac{1}{M} \sum_{k=0}^{M-1} y_k = \bar{y}. \end{aligned} \quad (6.23)$$

Thus, $G_0 = (A_0, B_0) = (\bar{x}, \bar{y})$ is simply the average of the x/y -coordinates i.e., the *centroid* of the original contour points g_k (see Fig. 6.5).¹⁰ If we apply the *inverse Fourier transform* (Eqn. (6.13)) by ignoring (i.e., zeroing) all coefficients except G_0 , we get the *partial reconstruction*¹¹ of the 2D contour coordinates $g_k^{(0)} = (x_k^{(0)}, y_k^{(0)})$ as

$$\begin{aligned} x_k^{(0)} &= \left[A_0 \cdot \cos\left(\omega_0 \frac{k}{M}\right) - B_0 \cdot \sin\left(\omega_0 \frac{k}{M}\right) \right] \\ &= \bar{x} \cdot \cos(0) - \bar{y} \cdot \sin(0) = \bar{x} \cdot 1 - \bar{y} \cdot 0 = \bar{x}, \end{aligned} \quad (6.24)$$

$$\begin{aligned} y_k^{(0)} &= \left[B_0 \cdot \cos\left(\omega_0 \frac{k}{M}\right) + A_0 \cdot \sin\left(\omega_0 \frac{k}{M}\right) \right] \\ &= \bar{y} \cdot \cos(0) + \bar{x} \cdot \sin(0) = \bar{y} \cdot 1 + \bar{x} \cdot 0 = \bar{y}. \end{aligned} \quad (6.25)$$

Thus the contribution of G_0 to the reconstructed shape is its *centroid* (see Fig. 6.5). If we perform a partial reconstruction of the contour using only the spectral coefficient G_0 , then all contour points

$$g_0^{(0)} = g_1^{(0)} = \dots = g_k^{(0)} = \dots = g_{M-1}^{(0)} = (\bar{x}, \bar{y})$$

would have the same (centroid) coordinate. This is because G_0 is the coefficient for the zero frequency and thus the sine- and cosine-terms in Eqns. (6.22–6.23)

¹⁰ Note that the centroid of a boundary is generally not the same as the centroid of the enclosed region.

¹¹ We use the notation $\mathbf{g}^{(m)} = (g_0^{(m)}, g_1^{(m)}, \dots, g_{M-1}^{(m)})$ for the *partial reconstruction* of the contour \mathbf{g} from only a single Fourier coefficient G_m . For example, $\mathbf{g}^{(0)}$ is the reconstruction from the zero-frequency coefficient G_0 only. Analogously, we use $\mathbf{g}^{(a,b,c)}$ to denote a partial reconstruction based on selected Fourier coefficients G_a, G_b, G_c .

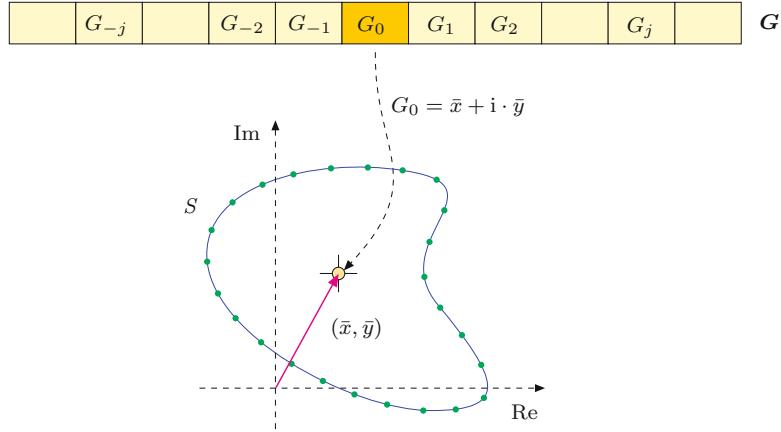


Figure 6.5 DFT coefficient G_0 corresponds to the centroid of the contour points.

are constant. Alternatively, if we reconstruct the signal by *omitting* G_0 (i.e., $\mathbf{g}^{(1,\dots,M-1)}$), the resulting contour is identical to the original shape, except that it is centered at the coordinate origin.

6.3.2 Coefficient G_1 corresponds to a circle

Next, we look at the geometric interpretation of $G_1 = (A_1, B_1)$, i.e., the coefficient with frequency index $m = 1$, which corresponds to the angular frequency $\omega_1 = 2\pi$. Assuming that all coefficients G_m in the DFT spectrum are set to zero, except the single coefficient G_1 , we get the partially reconstructed contour points $\mathbf{g}^{(1)}$ by Eqn. (6.10) as

$$g_k^{(1)} = G_1 \cdot e^{i \cdot 2\pi \frac{k}{M}} = (A_1 + i \cdot B_1) \cdot \left(\cos\left(2\pi \frac{k}{M}\right) + i \cdot \sin\left(2\pi \frac{k}{M}\right) \right), \quad (6.26)$$

for $0 \leq k < M$. Remember that the complex values of $e^{i\varphi}$ describe a *unit circle* in the complex plane that performs one full (counter-clockwise) revolution, as the angle φ runs from $0, \dots, 2\pi$. Analogously, $e^{i2\pi t}$ also describes a complete unit circle as t goes from 0 to 1. Since the term $\frac{k}{M}$ (for $0 \leq k < M$) also varies from 0 to 1 in Eqn. (6.26), the M reconstructed contour points are placed on a circle at equal angular steps. Multiplying $e^{i2\pi t}$ by a complex factor z stretches the *radius* of the circle by $|z|$, and also changes the *phase* (starting angle) of the circle by an angle θ , that is,

$$z \cdot e^{i\varphi} = |z| \cdot e^{i(\varphi+\theta)}, \quad (6.27)$$

with $\theta = \arg(z) = \tan^{-1}(\text{Im}(z)/\text{Re}(z))$.

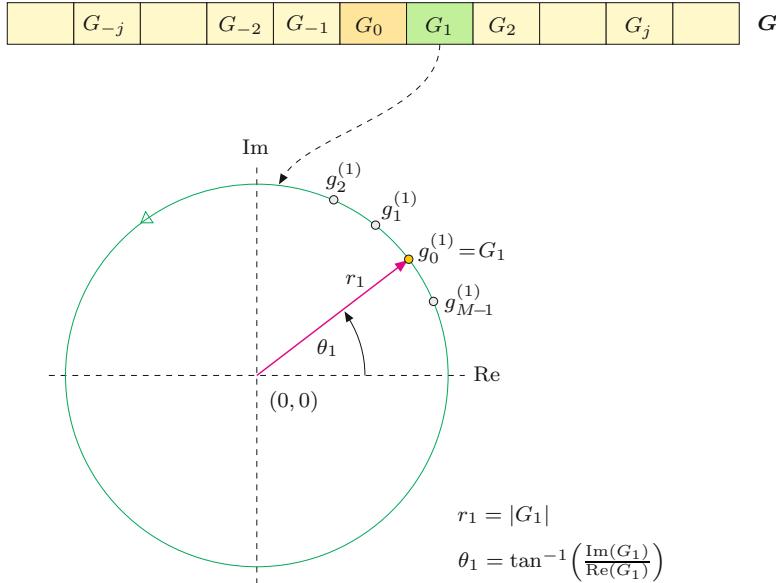


Figure 6.6 A single DFT coefficient corresponds to a circle. The partial reconstruction from the single DFT coefficient G_m yields a sequence of M points $g_0^{(m)}, \dots, g_{M-1}^{(m)}$ on a circle centered at the coordinate origin, with radius r_m and starting angle (phase) θ_m .

We now see that the points $g_k^{(1)} = G_1 \cdot e^{i2\pi k/M}$, generated by Eqn. (6.26), are positioned uniformly on a circle with radius $r_1 = |G_1|$ and starting angle (phase)

$$\theta_1 = \arg G_1 = \tan^{-1}\left(\frac{\text{Im}(G_1)}{\text{Re}(G_1)}\right) = \tan^{-1}\left(\frac{B_1}{A_1}\right). \quad (6.28)$$

This point sequence is traversed in counter-clockwise direction for $k = 0, \dots, M-1$ at frequency $m = 1$, that is, the circle performs one full revolution while the contour is traversed once. The circle is centered at the coordinate origin $(0,0)$, its radius is $|G_1|$, and its starting point (Eqn. (6.26) for $k = 0$) is

$$g_0^{(1)} = G_1 \cdot e^{i2\pi m \frac{k}{M}} = G_1 \cdot e^{i2\pi 1 \frac{0}{M}} = G_1 \cdot e^0 = G_1, \quad (6.29)$$

as illustrated in Fig. 6.6.

6.3.3 Coefficient G_m corresponds to a circle with frequency m

Based on the above result for the frequency index $m = 1$, we can easily generalize the geometric interpretation of Fourier coefficients with arbitrary index

$m > 0$. Using Eqn. (6.10), the partial reconstruction for the single Fourier coefficient $G_m = (A_m, B_m)$ is the contour $\mathbf{g}^{(m)}$, with coordinates

$$\begin{aligned} g_k^{(m)} &= G_m \cdot e^{i2\pi m \frac{k}{M}} \\ &= [A_m + i \cdot B_m] \cdot [\cos(2\pi m \frac{k}{M}) + i \cdot \sin(2\pi m \frac{k}{M})], \end{aligned} \quad (6.30)$$

which again describe a circle with radius $r_m = |G_m|$, phase $\theta_m = \arg(G_m) = \tan^{-1}(B_m/A_m)$, and starting point $g_0^{(m)} = G_m$. In this case, however, the angular velocity is scaled by m , i. e., the resulting circle revolves m times faster than the circle for G_1 . In other words, while the contour is traversed once, this circle performs m full revolutions.

Note that G_0 (see Sec. 6.3.1) does not really constitute a special case at all. Formally, it also describes a circle but one that oscillates with zero frequency, that is, all points have the same (constant) position

$$g_k^{(0)} = G_0 \cdot e^{i2\pi m \frac{k}{M}} = G_0 \cdot e^{i2\pi 0 \frac{k}{M}} = G_0 \cdot e^0 = G_0, \quad (6.31)$$

for $k = 0, \dots, M-1$, which is equivalent to the curve's centroid $G_0 = (\bar{x}, \bar{y})$, as shown in Eqns. (6.22–6.23). Since the corresponding frequency is zero, the point never moves away from G_0 .

6.3.4 Negative frequencies

The DFT spectrum is periodic and defined for all frequencies $m \in \mathbb{Z}$, including negative frequencies. From Eqn. (6.17) we know that for any DFT coefficient with negative index G_{-m} there is an equivalent coefficient G_n whose index n is in the range $0, \dots, M-1$. The partial reconstruction of the spectrum with the single coefficient G_{-m} is

$$g_k^{(-m)} = G_{-m} \cdot e^{-i2\pi m \frac{k}{M}} = G_n \cdot e^{-i2\pi m \frac{k}{M}}, \quad (6.32)$$

with $n = -m \bmod M$, which is again a sequence of points on the circle with radius $r_{-m} = r_n = |G_n|$ and phase $\theta_{-m} = \theta_n = \arg(G_n)$. The absolute rotation frequency is m , but this circle spins in the opposite, i. e., *clockwise* direction, since angles become increasingly negative with growing k .

6.3.5 Fourier descriptor pairs correspond to ellipses

It follows from the above that the space-domain circles for the Fourier coefficients G_m and G_{-m} rotate with the same absolute frequency m but with different phase angles θ_m, θ_{-m} and in opposite directions. We denote the tuple

$$\text{FP}_m = \langle G_{-m}, G_{+m} \rangle$$

the “Fourier descriptor pair” (or “FD pair”) for the frequency index m . If we perform a partial reconstruction from only the two Fourier coefficients G_{-m}, G_{+m} of this FD pair, we obtain the spatial points

$$\begin{aligned} g_k^{(\pm m)} &= g_k^{(-m)} + g_k^{(+m)} \\ &= G_{-m} \cdot e^{-i2\pi m \frac{k}{M}} + G_m \cdot e^{i2\pi m \frac{k}{M}} \\ &= G_{-m} \cdot e^{-i\omega_m \frac{k}{M}} + G_m \cdot e^{i\omega_m \frac{k}{M}}. \end{aligned} \quad (6.33)$$

From Eqn. (6.13) we can expand the result from Eqn. (6.33) to cartesian x/y coordinates as¹²

$$\begin{aligned} x_k^{(\pm m)} &= A_{-m} \cdot \cos(-\omega_m \frac{k}{M}) - B_{-m} \cdot \sin(-\omega_m \frac{k}{M}) + \\ &\quad A_m \cdot \cos(\omega_m \frac{k}{M}) - B_m \cdot \sin(\omega_m \frac{k}{M}) \\ &= (A_{-m} + A_m) \cdot \cos(\omega_m \frac{k}{M}) + (B_{-m} - B_m) \cdot \sin(\omega_m \frac{k}{M}), \end{aligned} \quad (6.34)$$

$$\begin{aligned} y_k^{(\pm m)} &= B_{-m} \cdot \cos(-\omega_m \frac{k}{M}) + A_{-m} \cdot \sin(-\omega_m \frac{k}{M}) + \\ &\quad B_m \cdot \cos(\omega_m \frac{k}{M}) + A_m \cdot \sin(\omega_m \frac{k}{M}) \\ &= (B_{-m} + B_m) \cdot \cos(\omega_m \frac{k}{M}) - (A_{-m} - A_m) \cdot \sin(\omega_m \frac{k}{M}), \end{aligned} \quad (6.35)$$

for $k = 0, \dots, M-1$. The 2D point sequence $\mathbf{g}^{(\pm m)} = (g_0^{(\pm m)}, \dots, g_{M-1}^{(\pm m)})$, obtained with the above pair of equations, describes an oriented *ellipse* that is centered at the origin (see Fig. 6.7). The parametric equation for this ellipse is

$$\begin{aligned} x_t^{(\pm m)} &= (A_{-m} + A_m) \cdot \cos(\omega_m \cdot t) + (B_{-m} - B_m) \cdot \sin(\omega_m \cdot t), \\ &= (A_{-m} + A_m) \cdot \cos(2\pi mt) + (B_{-m} - B_m) \cdot \sin(2\pi mt), \\ y_t^{(\pm m)} &= (B_{-m} + B_m) \cdot \cos(\omega_m \cdot t) - (A_{-m} - A_m) \cdot \sin(\omega_m \cdot t) \\ &= (B_{-m} + B_m) \cdot \cos(2\pi mt) - (A_{-m} - A_m) \cdot \sin(2\pi mt), \end{aligned} \quad (6.36)$$

for $t = 0, \dots, 1$.

Ellipse parameters

In general, the parametric equation of an ellipse with radii a, b , centered at (x_c, y_c) and oriented at an angle α is

$$\begin{aligned} x(\psi) &= x_c + a \cdot \cos(\psi) \cdot \cos(\alpha) - b \cdot \sin(\psi) \cdot \sin(\alpha), \\ y(\psi) &= y_c + a \cdot \cos(\psi) \cdot \sin(\alpha) + b \cdot \sin(\psi) \cdot \cos(\alpha), \end{aligned} \quad (6.37)$$

¹² Using the relations $\sin(-a) = -\sin(a)$ and $\cos(-a) = \cos(a)$.

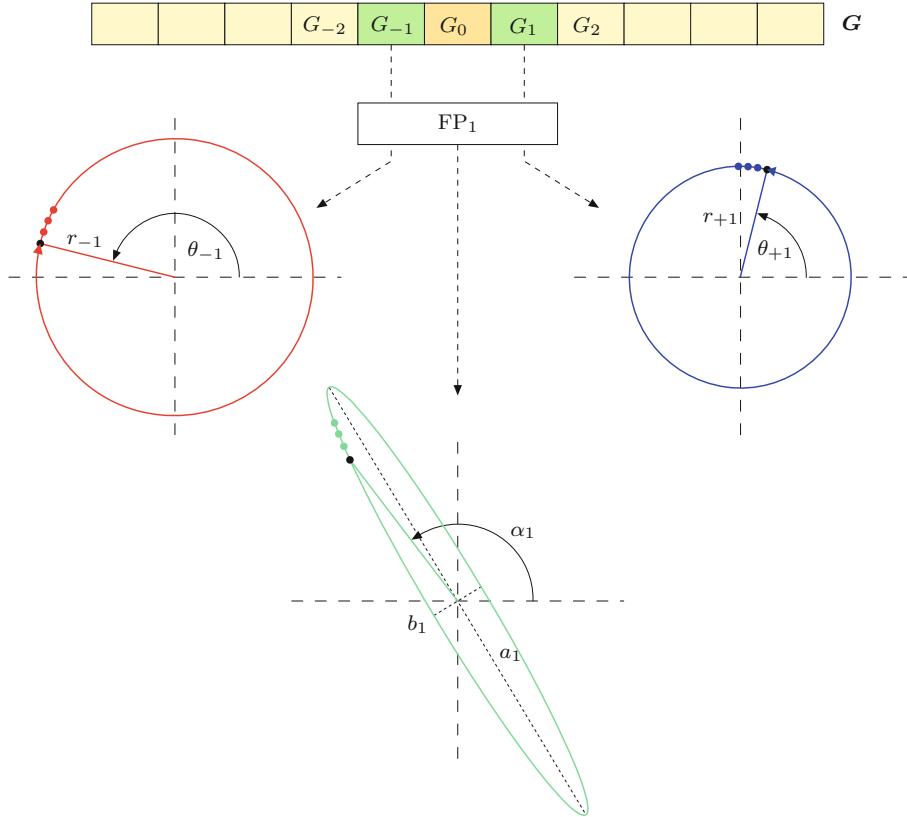


Figure 6.7 DFT coefficients G_{-m}, G_{+m} form a Fourier descriptor pair FP_m . Each of the two descriptors corresponds to M points on a circle of radius r_{-m}, r_{+m} and phase θ_{-m}, θ_{+m} , respectively, revolving with the same frequency m but in opposite directions. The sum of each point pair is located on an ellipse with radii a_m, b_m and orientation α_m . The orientation α_m of the ellipse's major axis is centered between the starting angles of the circles defined by G_{-m} and G_{+m} ; its radii are $a_m = r_{-m} + r_{+m}$ for the major axis and $b_m = |r_{-m} - r_{+m}|$ for the minor axis. The figure shows the situation for $m = 1$.

with $\psi = 0, \dots, 2\pi$. From Eqns. (6.34–6.35) we see that the parameters a_m, b_m, α_m of the ellipse for a single Fourier descriptor pair $\text{FP}_m = \langle G_{-m}, G_{+m} \rangle$ are

$$a_m = r_{-m} + r_{+m} = |G_{-m}| + |G_{+m}|, \quad (6.38)$$

$$b_m = |r_{-m} - r_{+m}| = ||G_{-m}| - |G_{+m}||, \quad (6.39)$$

$$\alpha_m = \frac{1}{2} \cdot \left(\underbrace{\arg G_{-m}}_{\theta_{-m}} + \underbrace{\arg G_{+m}}_{\theta_{+m}} \right) = \frac{1}{2} \cdot \left[\tan^{-1} \left(\frac{B_{-m}}{A_{-m}} \right) + \tan^{-1} \left(\frac{B_{+m}}{A_{+m}} \right) \right]. \quad (6.40)$$

Like its constituting circles, this ellipse is centered at $(x_c, y_c) = (0, 0)$ and performs m revolutions for one traversal of the contour. G_{-m} specifies the circle

$$z_{-m}(\varphi) = G_{-m} \cdot e^{i(-\varphi)} = r_{-m} \cdot e^{i(\theta_{-m}-\varphi)}, \quad (6.41)$$

for $\varphi \in [0, 2\pi]$, with starting angle θ_{-m} and radius r_{-m} , rotating in clockwise direction. Similarly, G_{+m} specifies the circle

$$z_{+m}(\varphi) = G_{+m} \cdot e^{i(\varphi)} = r_{+m} \cdot e^{i(\theta_{+m}+\varphi)}, \quad (6.42)$$

with starting angle θ_{+m} and radius r_{+m} , rotating in counter-clockwise direction. Both circles thus rotate at the same angular velocity but in opposite directions, as mentioned before. The corresponding (complex-valued) ellipse points are

$$z_m(\varphi) = z_{-m}(\varphi) + z_{+m}(\varphi). \quad (6.43)$$

The ellipse radius $|z_m(\varphi)|$ is a *maximum* at position $\varphi = \varphi_{\max}$, where the angles on both circles are identical (i.e., the corresponding vectors have the same direction). This occurs when

$$\theta_{-m} - \varphi_{\max} = \theta_{+m} + \varphi_{\max} \quad \text{or} \quad \varphi_{\max} = \frac{1}{2} \cdot (\theta_{-m} - \theta_{+m}), \quad (6.44)$$

i.e., at mid-angle between the two starting angles θ_{-m} and θ_{+m} . Therefore, the orientation of the ellipse's major axis is

$$\alpha_m = \theta_{+m} + \frac{\theta_{-m} - \theta_{+m}}{2} = \frac{1}{2} \cdot (\theta_{-m} + \theta_{+m}), \quad (6.45)$$

as already stated in Eqn. (6.40). At $\varphi = \varphi_{\max}$ the two radial vectors align, and thus the radius of the ellipse's major axis a_m is the sum of the two circle radii, that is,

$$a_m = r_{-m} + r_{+m} \quad (6.46)$$

(cf. Eqn. (6.38)). Analogously, the ellipse radius is *minimized* at position $\varphi = \varphi_{\min}$, where the $z_{-m}(\varphi_{\min})$ and $z_{+m}(\varphi_{\min})$ lie on opposite sides of the circle. This occurs at angle

$$\varphi_{\min} = \varphi_{\max} + \frac{\pi}{2} = \frac{\pi + \theta_{-m} - \theta_{+m}}{2} \quad (6.47)$$

and the corresponding radius for the ellipse's minor axis is (cf. Eqn. (6.39))

$$b_m = r_{+m} - r_{-m}. \quad (6.48)$$

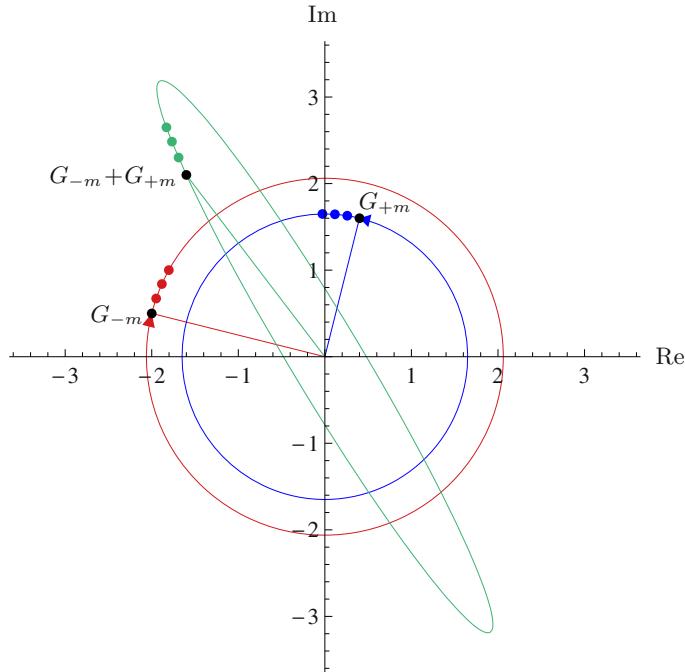


Figure 6.8 Ellipse created by partial reconstruction from a single Fourier descriptor pair $\text{FP}_m = \langle G_{-m}, G_{+m} \rangle$. The two complex-valued Fourier coefficients $G_{-m} = (-2, 0.5)$ and $G_{+m} = (0.4, 1.6)$ represent circles with starting points G_{-m} and G_{+m} , respectively. The circle for G_{-m} (red) rotates in clockwise direction, the circle for G_{+m} (blue) rotates in counter-clockwise direction. The ellipse (green) is the result of point-wise addition of the two circles, as shown for four successive points, starting with point $G_{-m} + G_{+m}$.

Figure 6.8 illustrates this situation for a specific Fourier descriptor pair $\text{FP}_m = \langle G_{-m}, G_{+m} \rangle = \langle -2 + i \cdot 0.5, 0.4 + i \cdot 1.6 \rangle$. Note that the ellipse parameters a_m, b_m, α_m (Eqns. (6.38–6.40)) are not explicitly required for reconstructing (drawing) the contour, since the ellipse can also be generated by simply adding the x/y -coordinates of the two counter-revolving circles for the participating Fourier descriptors, as given in Eqn. (6.43).

6.3.6 Shape reconstruction from truncated Fourier descriptors

Due to the periodicity of the DFT spectrum, the complete reconstruction of the contour points g_k from the Fourier coefficients G_m (see Eqn. (6.10)), could also be written with a different summation range, as long as all spectral coefficients

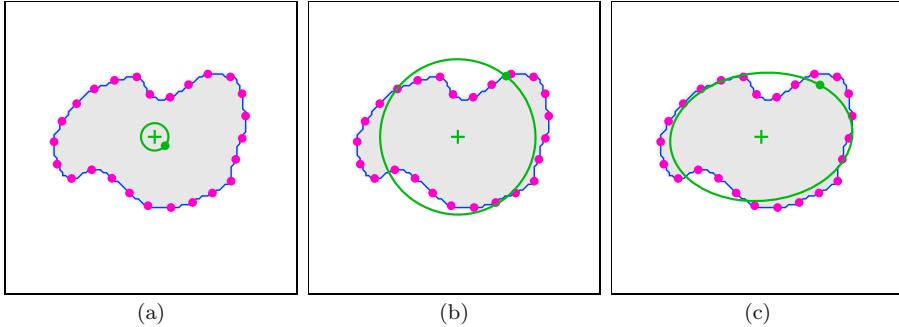


Figure 6.9 Partial reconstruction from single coefficients and an FD descriptor pair. The two circles reconstructed from DFT coefficient G_{-1} (a) and coefficient G_{+1} (b) are positioned at the centroid of the contour (G_0). The combined reconstruction for $\langle G_{-1}, G_{+1} \rangle$ produces the ellipse in (c). The dots on the green curves show the path position for $t = 0$.

are included, that is,

$$g_k = \sum_{m=0}^{M-1} G_m \cdot e^{i2\pi m \frac{k}{M}} = \sum_{m=m_0}^{m_0+M-1} G_m \cdot e^{i2\pi m \frac{k}{M}}, \quad (6.49)$$

for any start index $m_0 \in \mathbb{Z}$. As a special (but important) case we can perform the summation symmetrically around the zero index and write

$$g_k = \sum_{m=0}^{M-1} G_m \cdot e^{i2\pi m \frac{k}{M}} = \sum_{\substack{m=-(M-1)/2 \\ m=m_0}}^{M/2} G_m \cdot e^{i2\pi m \frac{k}{M}}. \quad (6.50)$$

To understand the reconstruction in terms of Fourier descriptor pairs, it is helpful to distinguish whether M (the number of contour points and Fourier coefficients) is *even* or *odd*.

Odd number of contour points

If M is *odd*, then the spectrum consists of G_0 (representing the contour's centroid) plus exactly $M/2$ Fourier descriptor pairs FP_m , with $m = 1, \dots, M/2$.¹³ We can thus rewrite Eqn. (6.49) as

$$\begin{aligned} g_k &= \sum_{m=0}^{M-1} G_m \cdot e^{i2\pi m \frac{k}{M}} = \underbrace{G_0}_{g_k^{(0)}} + \sum_{m=1}^{M/2} \left[\underbrace{G_{-m} \cdot e^{-i2\pi m \frac{k}{M}} + G_m \cdot e^{i2\pi m \frac{k}{M}}}_{g_k^{(\pm m)} = g_k^{(-m)} + g_k^{(m)}} \right] \\ &= g_k^{(0)} + \sum_{m=1}^{M/2} g_k^{(\pm m)} = g_k^{(0)} + g_k^{(\pm 1)} + g_k^{(\pm 2)} + \dots + g_k^{(\pm M/2)}, \end{aligned} \quad (6.51)$$

¹³ If M is odd, then $M = 2 \cdot (M/2) + 1$.

where $g_k^{(\pm m)}$ denotes the partial reconstruction from the single Fourier descriptor pair FP_m (see Eqn. (6.33)).

As we already know, the partial reconstruction $g_k^{(\pm m)}$ of an individual Fourier descriptor pair FP_m is a set of points on an ellipse that is centered at the origin $(0, 0)$. The partial reconstruction of the *three* DFT coefficients G_0, G_{-m}, G_{+m} (i. e., FP_m plus the single coefficient G_0) is the point sequence

$$g_k^{(-m,0,m)} = g_k^{(0)} + g_k^{(\pm m)}, \quad (6.52)$$

which is the ellipse for $g_k^{(\pm m)}$ shifted to $g_k^{(0)} = (\bar{x}, \bar{y})$, the centroid of the original contour. For example, the partial reconstruction from the coefficients G_{-1}, G_0, G_{+1} ,

$$g_k^{(-1,0,1)} = g_k^{(-1,\dots,1)} = g_k^{(0)} + g_k^{(\pm 1)}, \quad (6.53)$$

is an ellipse with frequency $m = 1$ that revolves around the (fixed) centroid of the original contour. If we add another Fourier descriptor pair FP_2 , the resulting reconstruction is

$$g_k^{(-2,\dots,2)} = \underbrace{g_k^{(0)} + g_k^{(\pm 1)}}_{\text{ellipse 1}} + \underbrace{g_k^{(\pm 2)}}_{\text{ellipse 2}}. \quad (6.54)$$

The resulting ellipse $g_k^{(\pm 2)}$ has the frequency $m = 2$, but note that it is centered at a moving point on the “slower” ellipse (with frequency $m = 1$), that is, ellipse 2 effectively “rides” on ellipse 1. If we add FP_3 , its ellipse is again centered at a point on ellipse 2, and so on. For an illustration, see the examples in Figs. 6.11–6.12. In general, the ellipse for descriptor pair FP_j revolves around the (moving) center obtained as the superposition of $j - 1$ “slower” ellipses,

$$g_k^{(0)} + \sum_{m=1}^{j-1} g_k^{(\pm m)}. \quad (6.55)$$

Consequently, the curve obtained by the partial reconstruction from descriptor pairs $\text{FP}_1, \dots, \text{FP}_j$ (for $j \leq M \div 2$) is the point sequence

$$g_k^{(-j,\dots,j)} = g_k^{(0)} + \sum_{m=1}^j g_k^{(\pm m)}, \quad (6.56)$$

for $k = 0, \dots, M - 1$. The fully reconstructed shape is the sum of the centroid (defined by G_0) and $M \div 2$ ellipses, one for each Fourier descriptor pair $\text{FP}_1, \dots, \text{FP}_{M \div 2}$.

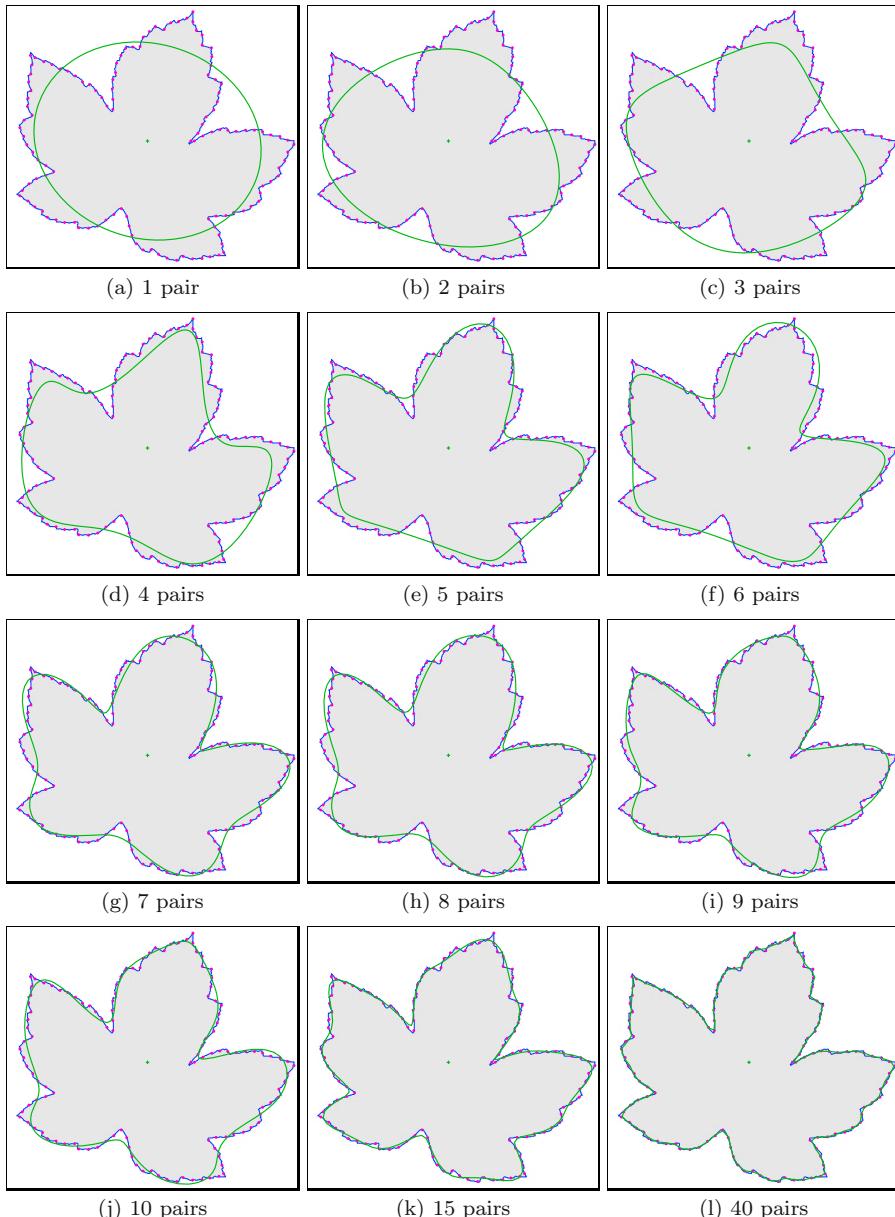


Figure 6.10 Partial shape reconstruction from a limited set of Fourier descriptor pairs. The full descriptor contains 125 coefficients (G_0 plus 62 FD pairs).

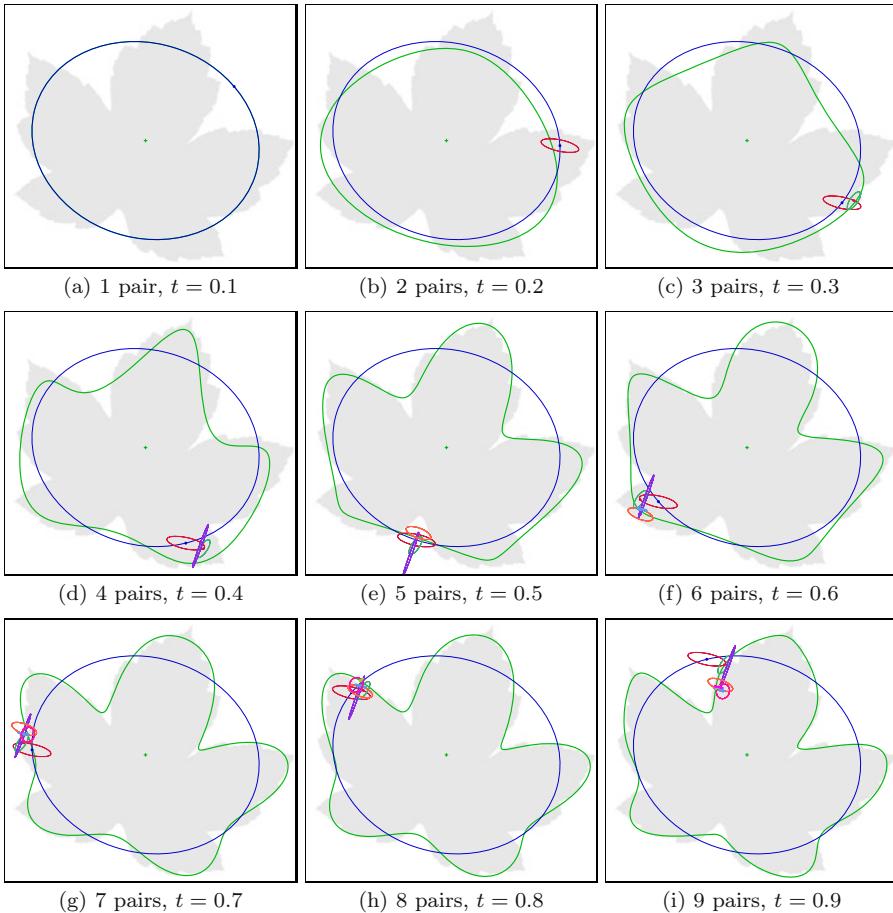


Figure 6.11 Partial reconstruction by ellipse superposition (details). The green curve shows the partial reconstruction from $1, \dots, 9$ FD pairs. This curve performs one full revolution as the path parameter t runs from 0 to 1. Subfigures (a–i) depict the situation for $1, \dots, 9$ FD pairs and different path positions $t = 0.1, 0.2, \dots, 0.9$. Each Fourier descriptor pair corresponds to an ellipse that is centered at the current position t on the previous ellipse. The individual Fourier descriptor pair FP_1 in (a) corresponds to a single ellipse. In (b), the point for $t = 0.2$ on the blue ellipse (for FP_1) is the center of the red ellipse (for FP_2). In (c), the green ellipse (for FP_3) is centered at the point marked on the previous ellipse, and so on. The reconstructed shape is obtained by superposition of all ellipses. See Fig. 6.12 for a detailed view.

Even number of contour points

If M is even,¹⁴ then the reconstructed shape is a superposition of the centroid (defined by G_0), $(M - 1) \div 2$ ellipses from the Fourier descriptor pairs $\text{FP}_1, \dots, \text{FP}_{(M-1)\div 2}$, plus one additional circle specified by the single (high-

¹⁴ In this case, $M = 2 \cdot (M \div 2) = (M - 1) \div 2 + 1 + M \div 2$.

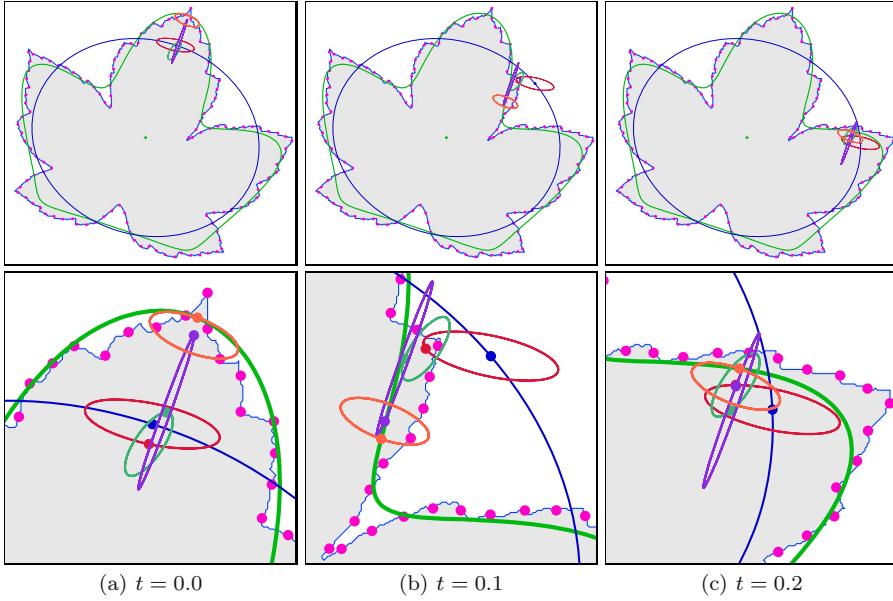


Figure 6.12 Partial reconstruction by ellipse superposition (details). The green curve shows the partial reconstruction from 5 FD pairs $\text{FP}_1, \dots, \text{FP}_5$. This curve performs one full revolution as the path parameter t runs from 0 to 1. Subfigures (a–c) show the composition of the contour by superposition of the 5 ellipses, each corresponding to one FD pair, at selected positions $t = 0.0, 0.1, 0.2$. The blue ellipse corresponds to FP_1 and revolves once for $t = 0, \dots, 1$. The blue dot on this ellipse marks the position t , which serves as the center of the next (red) ellipse corresponding to FP_2 . This ellipse makes 2 revolutions for $t = 0, \dots, 1$ and the red dot for position t is again the center of green ellipse (for FP_3), and so on. Position t on the orange ellipse (for FP_1) coincides with the final reconstruction (green curve). The original contour was sampled at 125 equidistant points.

est frequency) Fourier coefficient $G_{M \div 2}$. The complete reconstruction from an even-length Fourier descriptor can thus be written as

$$g_k = \sum_{m=0}^{M-1} G_m \cdot e^{i2\pi m \frac{k}{M}} = \underbrace{g_k^{(0)} \text{ center}}_{\text{(}M-1\text{)}} + \underbrace{\sum_{m=1}^{(M-1)\div 2} g_k^{(\pm m)}}_{(M-1)\div 2 \text{ ellipses}} + \underbrace{g_k^{(M \div 2)}}_{1 \text{ circle}}. \quad (6.57)$$

The single high-frequency circle corresponding to $g_k^{(M \div 2)}$ has its (moving) center at the sum of all lower-frequency ellipses that correspond to the Fourier coefficients G_{-m}, \dots, G_{+m} , with $m < (M \div 2)$.

Reconstruction algorithm

Algorithm 6.4 describes the reconstruction of shapes from a Fourier descriptor using only a specified number (M_p) of Fourier descriptor pairs. The number of

Algorithm 6.4 Partial shape reconstruction from a truncated Fourier descriptor \mathbf{G} . The shape is reconstructed by considering up to M_p Fourier descriptor pairs. The resulting sequence of contour points may be of arbitrary length (N). See Figs. (6.10–6.12) for examples.

```

1: GETPARTIALRECONSTRUCTION( $\mathbf{G}, M_p, N$ )
   Input:  $\mathbf{G} = (G_0, \dots, G_{M-1})$ , Fourier descriptor with  $M$  coefficients;
           $M_p$ , number of Fourier descriptor pairs to consider;  $N$ , number of
          points on the reconstructed shape. Returns the reconstructed contour
          as a sequence of  $N$  complex values.
2: Create map  $\mathbf{g}: [0, N-1] \rightarrow \mathbb{C}$ 
3:  $M \leftarrow |\mathbf{G}|$                                  $\triangleright$  total number of Fourier coefficients
4:  $M_p \leftarrow \min(M_p, (M-1) \div 2)$             $\triangleright$  available Fourier coefficient pairs
5: for  $k \leftarrow 0, \dots, N-1$  do
6:    $t \leftarrow k/N$                                 $\triangleright$  continuous path position  $t \in [0, 1]$ 
7:    $\mathbf{g}(k) \leftarrow \text{GETSINGLEPOINT}(\mathbf{G}, -M_p, M_p, t)$        $\triangleright$  see below
8: return  $\mathbf{g}$ .


---


9: GETSINGLEPOINT( $\mathbf{G}, m_-, m_+, t$ )
   Returns a single point (as a complex value) on the reconstructed shape
   for the continuous path position  $t \in [0, 1]$ , based on the Fourier coef-
   ficients  $\mathbf{G}(m_-), \dots, \mathbf{G}(m_+)$ .
10:  $M \leftarrow |\mathbf{G}|$ 
11:  $x \leftarrow 0, y \leftarrow 0$ 
12: for  $m \leftarrow m_-, \dots, m_+$  do
13:    $\phi \leftarrow 2 \cdot \pi \cdot m \cdot t$ 
14:    $G \leftarrow \mathbf{G}(m \bmod M)$ 
15:    $A \leftarrow \text{Re}(G), B \leftarrow \text{Im}(G)$ 
16:    $x \leftarrow x + A \cdot \cos(\phi) - B \cdot \sin(\phi)$ 
17:    $y \leftarrow y + A \cdot \sin(\phi) + B \cdot \cos(\phi)$ 
18: return  $(x + i y)$ .
```

points on the reconstructed contour (N) can be freely chosen.

6.3.7 Fourier descriptors from arbitrary polygons

The requirement to distribute sample points uniformly along the contour path stems from classical signal processing and Fourier theory, where uniform sampling is a common assumption. However, as shown in [70] (see also [102, 151]), the Fourier descriptors for a polygonal shape can be calculated directly from the original polygon vertices without uniform sub-sampling. This “trigonometric” approach, described in the following, works for arbitrary (convex and non-convex) polygons.

We assume that the shape is specified as a sequence of P points $V = (\mathbf{v}_0, \dots, \mathbf{v}_{P-1})$, with $V(i) = \mathbf{v}_i = (x_i, y_i)$ representing the 2D vertices of a closed polygon. We define the quantities

$$\mathbf{d}(i) = \mathbf{v}_{(i+1) \bmod P} - \mathbf{v}_i \quad \text{and} \quad \lambda(i) = \|\mathbf{d}(i)\|, \quad (6.58)$$

for $i = 0, \dots, P-1$, where $\mathbf{d}(i)$ is the vector representing the polygon segment between the vertices $\mathbf{v}_i, \mathbf{v}_{i+1}$, and $\lambda(i)$ is the length of that segment. We also define

$$L(i) = \sum_{j=0}^{i-1} \lambda(j), \quad (6.59)$$

for $i = 0, \dots, P$, which is the cumulative length of the polygon path from the start vertex \mathbf{v}_0 to vertex \mathbf{v}_i , such that $L(0)$ is zero and $L(P)$ is the closed path length of the polygon V .

For a (freely chosen) number of Fourier descriptor pairs (M_p) , the corresponding Fourier descriptor $\mathbf{G} = (G_{-M_p}, \dots, G_0, \dots, G_{+M_p})$, has $2M_p + 1$ complex-valued coefficients G_m , where

$$G_0 = a_0 + i \cdot c_0 \quad (6.60)$$

and the remaining coefficients are calculated as

$$G_{+m} = (a_m + d_m) + i \cdot (c_m - b_m), \quad (6.61)$$

$$G_{-m} = (a_m - d_m) + i \cdot (c_m + b_m), \quad (6.62)$$

from the “trigonometric coefficients” a_m, b_m, c_m, d_m . As described in [70], these coefficients are obtained directly from the P polygon vertices \mathbf{v}_i as

$$\begin{pmatrix} a_0 \\ c_0 \end{pmatrix} = \mathbf{v}_0 + \frac{\sum_{i=0}^{P-1} \left[\frac{L^2(i+1) - L^2(i)}{2\lambda(i)} \cdot \mathbf{d}(i) + \lambda(i) \cdot \sum_{j=0}^{i-1} \mathbf{d}(j) - \mathbf{d}(i) \cdot \sum_{j=0}^{i-1} \lambda(j) \right]}{L(P)} \quad (6.63)$$

(representing the shape’s center), with \mathbf{d}, λ, L as defined in Eqns. (6.58–6.59). This can be simplified to

$$\begin{pmatrix} a_0 \\ c_0 \end{pmatrix} = \mathbf{v}_0 + \frac{\sum_{i=0}^{P-1} \left[\left(\frac{L^2(i+1) - L^2(i)}{2\lambda(i)} - L(i) \right) \cdot \mathbf{d}(i) + \lambda(i) \cdot (\mathbf{v}_i - \mathbf{v}_0) \right]}{L(P)}. \quad (6.64)$$

The remaining coefficients a_m, b_m, c_m, d_m ($m = 1, \dots, M_p$) are calculated as

$$\begin{pmatrix} a_m \\ c_m \end{pmatrix} = \frac{L(P)}{(2\pi m)^2} \cdot \sum_{i=0}^{P-1} \left[\frac{\cos(2\pi m \frac{L(i+1)}{L(P)}) - \cos(2\pi m \frac{L(i)}{L(P)})}{\lambda(i)} \cdot \mathbf{d}(i) \right], \quad (6.65)$$

$$\begin{pmatrix} b_m \\ d_m \end{pmatrix} = \frac{L(P)}{(2\pi m)^2} \cdot \sum_{i=0}^{P-1} \left[\frac{\sin(2\pi m \frac{L(i+1)}{L(P)}) - \sin(2\pi m \frac{L(i)}{L(P)})}{\lambda(i)} \cdot \mathbf{d}(i) \right], \quad (6.66)$$

respectively. The complete calculation of a Fourier descriptor from trigonometric coordinates (arbitrary polygons) is summarized in [Alg. 6.5](#).

An approximate reconstruction of the original shape can be obtained directly from the trigonometric coefficients a_m, b_m, c_m, d_m defined in Eqns. (6.64–6.64) as¹⁵

$$\mathbf{x}(t) = \begin{pmatrix} a_0 \\ c_0 \end{pmatrix} + \sum_{m=1}^{M_p} \left[\begin{pmatrix} a_m \\ c_m \end{pmatrix} \cdot \cos(2\pi m t) + \begin{pmatrix} b_m \\ d_m \end{pmatrix} \cdot \sin(2\pi m t) \right], \quad (6.67)$$

for $t = 0, \dots, 1$. Of course, this reconstruction can also be calculated from the actual DFT coefficients \mathbf{G} , as described in Eqn. (6.16). Again the reconstruction error is reduced by increasing the number of Fourier descriptor pairs (M_p), as demonstrated in [Fig. 6.13](#).¹⁶ The reconstruction is theoretically perfect as M_p goes to infinity.

Working with the trigonometric technique is an advantage, in particular, if the boundary curvature along the outline varies strongly. For example, the silhouette of a human hand typically exhibits high curvature along the fingertips while other contour sections are almost straight. Capturing the high-curvature parts requires a significantly higher density of samples than in the smooth sections, as illustrated in [Fig. 6.14](#). This figure compares the partial shape reconstructions obtained from Fourier descriptors calculated with uniform and non-uniform contour sampling, using identical numbers of Fourier descriptor pairs (M_p). Note that the coefficients (and thus the reconstructions) are very similar, although considerably fewer samples were used for the trigonometric approach.

6.4 Effects of geometric transformations

To be useful for comparing shapes, a representation should be invariant against a certain set of geometric transformations. Typically, a minimal requirement for robust 2D shape matching is invariance to translation, scale changes and rotation. Fourier shape descriptors in their basic form are *not* invariant under any of these transformations but they can be modified to satisfy these requirements. In this section, we discuss the effects of such transformations upon

¹⁵ Note the analogy to the elliptical reconstruction in Eqn. (6.36).

¹⁶ Most test images used in this chapter were taken from the KIMIA dataset [65]. A selected subset of modified images taken from this dataset is available at the book's website.

Algorithm 6.5 Fourier descriptor from trigonometric data (arbitrary polygons). Parameter M_p specifies the number of Fourier coefficient pairs.

```

1: FOURIERDESCRIPTORFROMPOLYGON( $V, M_p$ )
   Input:  $V = (\mathbf{v}_0, \dots, \mathbf{v}_{P-1})$ , a sequence of  $P$  points representing the
          vertices of a closed 2D polygon;  $M_p$ , the desired number of FD pairs.
          Returns a new Fourier descriptor of length  $2M_p+1$ .
2:  $P \leftarrow |V|$                                  $\triangleright$  number of polygon vertices in  $V$ 
3:  $M \leftarrow 2 \cdot M_p + 1$                        $\triangleright$  number of Fourier coefficients in  $\mathbf{G}$ 
4: Create maps  $\mathbf{d}: [0, P-1] \rightarrow \mathbb{R}^2$ ,  $\lambda: [0, P-1] \rightarrow \mathbb{R}$ ,
    $L: [0, P] \rightarrow \mathbb{R}$ ,  $\mathbf{G}: [0, M-1] \rightarrow \mathbb{C}$ 
5:
6:  $L(0) \leftarrow 0$ 
7: for  $i \leftarrow 0, \dots, P-1$  do
8:    $\mathbf{d}(i) \leftarrow V((i+1) \bmod P) - V(i)$             $\triangleright$  Eqn. (6.58)
9:    $\lambda(i) \leftarrow \|\mathbf{d}(i)\|$ 
10:   $L(i+1) \leftarrow L(i) + \lambda(i)$ 
11:   $\begin{pmatrix} a \\ c \end{pmatrix} \leftarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix}$             $\triangleright a = a_0, c = c_0$ 
12:  for  $i \leftarrow 0, \dots, P-1$  do
13:     $s \leftarrow \frac{L^2(i+1) - L^2(i)}{2 \cdot \lambda(i)} - L(i)$ 
14:     $\begin{pmatrix} a \\ c \end{pmatrix} \leftarrow \begin{pmatrix} a \\ c \end{pmatrix} + s \cdot \mathbf{d}(i) + \lambda(i) \cdot (V(i) - V(0))$       $\triangleright$  Eqn. (6.64)
15:   $\mathbf{G}(0) \leftarrow \mathbf{v}_0 + \frac{1}{L(P)} \cdot \begin{pmatrix} a \\ c \end{pmatrix}$             $\triangleright$  Eqn. (6.60)
16:  for  $m \leftarrow 1, \dots, M_p$  do            $\triangleright$  for FD-pairs  $G_{\pm 1}, \dots, G_{\pm M_p}$ 
17:     $\begin{pmatrix} a \\ c \end{pmatrix} \leftarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad \begin{pmatrix} b \\ d \end{pmatrix} \leftarrow \begin{pmatrix} 0 \\ 0 \end{pmatrix}$             $\triangleright a_m, b_m, c_m, d_m$ 
18:    for  $i \leftarrow 0, \dots, P-1$  do
19:       $\omega_0 \leftarrow 2\pi m \cdot \frac{L(i)}{L(P)}$ 
20:       $\omega_1 \leftarrow 2\pi m \cdot \frac{L((i+1) \bmod P)}{L(P)}$ 
21:       $\begin{pmatrix} a \\ c \end{pmatrix} \leftarrow \begin{pmatrix} a \\ c \end{pmatrix} + \frac{\cos(\omega_1) - \cos(\omega_0)}{\lambda(i)} \cdot \mathbf{d}(i)$       $\triangleright$  Eqn. (6.65)
22:       $\begin{pmatrix} b \\ d \end{pmatrix} \leftarrow \begin{pmatrix} b \\ d \end{pmatrix} + \frac{\sin(\omega_1) - \sin(\omega_0)}{\lambda(i)} \cdot \mathbf{d}(i)$       $\triangleright$  Eqn. (6.66)
23:     $\mathbf{G}(m) \leftarrow \frac{L(P)}{(2\pi m)^2} \cdot \begin{pmatrix} a+d \\ c-b \end{pmatrix}$             $\triangleright$  Eqn. (6.61)
24:     $\mathbf{G}(-m \bmod M) \leftarrow \frac{L(P)}{(2\pi m)^2} \cdot \begin{pmatrix} a-d \\ c+b \end{pmatrix}$             $\triangleright$  Eqn. (6.62)
25: return  $\mathbf{G}$ .

```

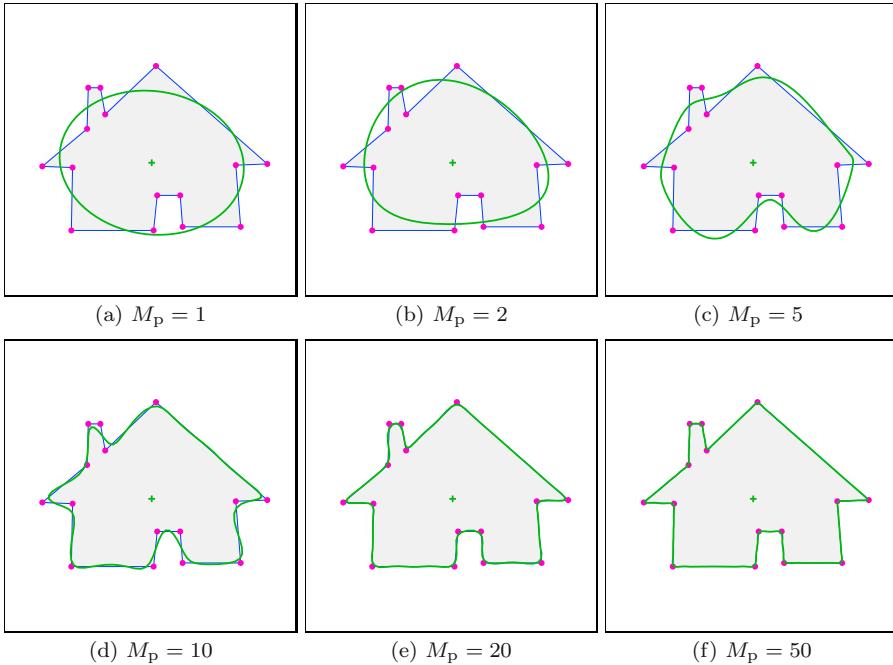


Figure 6.13 Fourier descriptors calculated from trigonometric data (arbitrary polygons). Shape reconstructions with different numbers of Fourier descriptor pairs (M_p).

the corresponding Fourier descriptors. The steps involved for making Fourier descriptors invariant are discussed subsequently in Section 6.5.

6.4.1 Translation

As described in Section 6.3.1, the coefficient G_0 of a Fourier descriptor \mathbf{G} corresponds to the centroid of the encoded contour. Moving the points g_k of a shape \mathbf{g} in the complex plane by some constant $z \in \mathbb{C}$,

$$g'_k = g_k + z, \quad (6.68)$$

for $k = 0, \dots, M-1$, only affects Fourier coefficient G_0 , that is,

$$G'_m = \begin{cases} G_m + z & \text{for } m = 0, \\ G_m & \text{for } m \neq 0. \end{cases} \quad (6.69)$$

To make an FD invariant against translation, it is thus sufficient to zero its G_0 coefficient, thereby shifting the shape's center to the origin of the coordinate system. Alternatively, translation invariant matching of Fourier descriptors is achieved by simply ignoring coefficient G_0 .

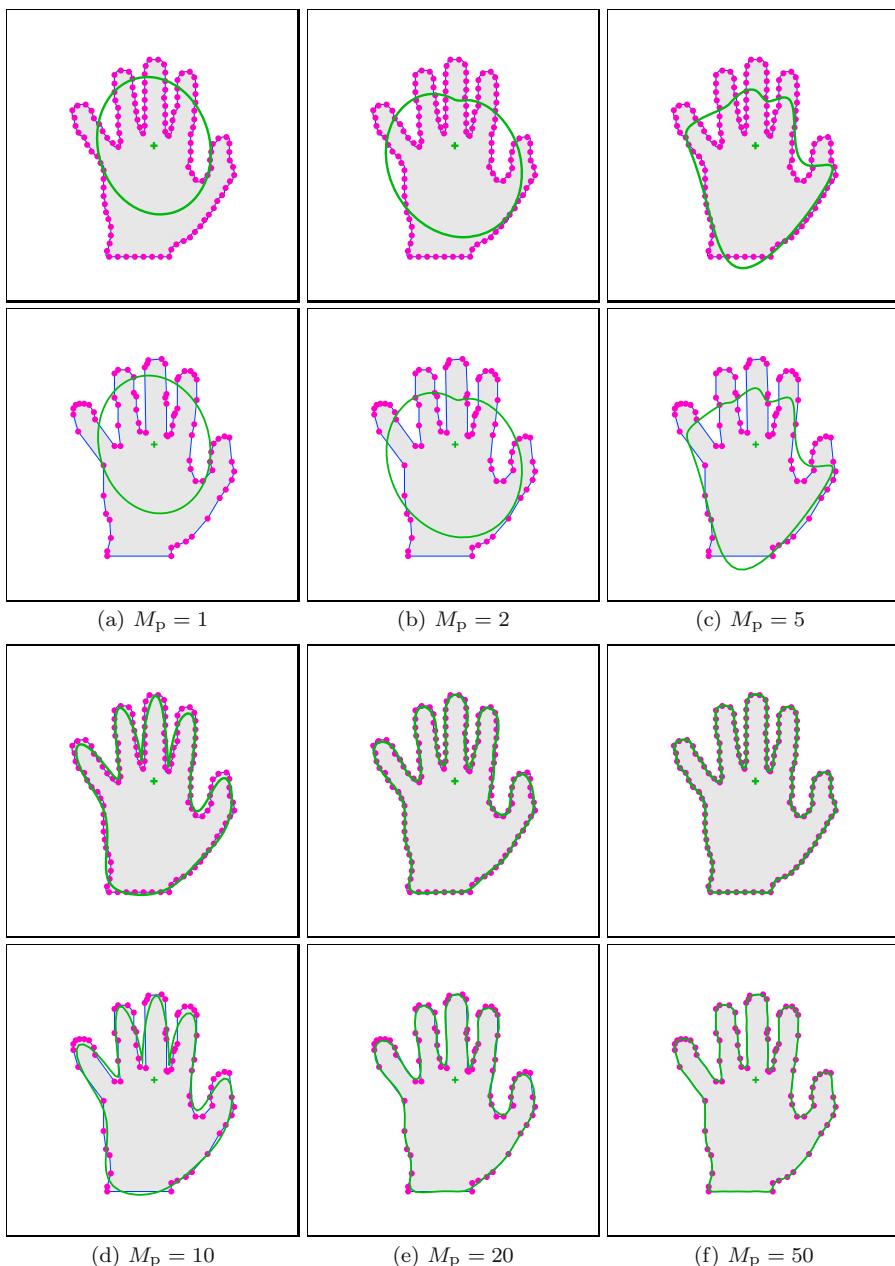


Figure 6.14 Fourier descriptors from uniformly sampled vs. non-uniformly sampled (trigonometric) contours. Partial constructions from Fourier descriptors obtained from uniformly sampled contours (rows 1, 3) and non-uniformly sampled contours (rows 2, 4), for different numbers of Fourier descriptor pairs (M_P).

6.4.2 Scale change

Since the Fourier transform is a linear operation, scaling a 2D shape \mathbf{g} uniformly by a real-valued factor s ,

$$g'_k = s \cdot g_k, \quad (6.70)$$

also scales the corresponding Fourier spectrum by the same factor, that is,

$$G'_m = s \cdot G_m, \quad (6.71)$$

for $m = 1, \dots, M-1$. Note that scaling by $s = -1$ (or any other negative factor) corresponds to *reversing* the ordering of the samples along the contour (see also Sec. 6.4.6). Given the fact that the DFT coefficient G_1 represents a circle whose radius $r_1 = |G_1|$ is proportional to the size of the original shape (see Sec. 6.3.2), the Fourier descriptor \mathbf{G} could be normalized for scale by setting

$$G_m^S = \frac{1}{|G_1|} \cdot G_m, \quad (6.72)$$

for $m = 1, \dots, M-1$, such that $|G_1^S| = 1$. Although it is common to use only G_1 for scale normalization, this coefficient may be relatively small (and thus unreliable) for certain shapes. We therefore prefer to normalize the complete Fourier coefficient vector to achieve scale invariance (see Sec. 6.5.1).

6.4.3 Shape rotation

If a given shape is rotated about the origin by some angle β , then each contour point $\mathbf{v}_k = (x_k, y_k)$ moves to a new position

$$\mathbf{v}'_k = \begin{pmatrix} x'_k \\ y'_k \end{pmatrix} = \begin{pmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{pmatrix} \cdot \begin{pmatrix} x_k \\ y_k \end{pmatrix}. \quad (6.73)$$

If the 2D contour samples are represented as complex values $g_k = x_k + i \cdot y_k$, this rotation can be expressed as a multiplication

$$g'_k = e^{i\beta} \cdot g_k, \quad (6.74)$$

with the complex factor $e^{i\beta} = \cos(\beta) + i \cdot \sin(\beta)$. As in Eqn. (6.71), we can use the linearity of the DFT to predict the effects of rotating the shape \mathbf{g} by angle β as

$$G'_m = e^{i\beta} \cdot G_m, \quad (6.75)$$

for $m = 0, \dots, M-1$. Thus, the spatial rotation in Eqn. (6.74) multiplies each DFT coefficient G_m by the *same* complex factor $e^{i\beta}$, which has unit magnitude. Since

$$e^{i\beta} \cdot G_m = e^{i(\theta_m + \beta)} \cdot |G_m|, \quad (6.76)$$

this only rotates the *phase* $\theta_m = \angle G_m$ of each coefficient by the *same* angle β , without changing its *magnitude* $|G_m|$.

6.4.4 Shifting the contour start position

Despite the implicit periodicity of the boundary sequence and the corresponding DFT spectrum, Fourier descriptors are generally not the same if sampling starts at different positions along the contour. Given a periodic sequence of M discrete contour samples $\mathbf{g} = (g_0, g_1, \dots, g_{M-1})$, we select another sequence $\mathbf{g}' = (g'_0, g'_1, \dots) = (g_{k_s}, g_{k_s+1}, \dots)$, again of length M , from the same set of samples but starting at point k_s , that is,

$$g'_k = g_{(k+k_s) \bmod M}. \quad (6.77)$$

This is equivalent to *shifting* the original signal \mathbf{g} circularly by $-k_s$ positions. The well-known “shift property” of the Fourier transform¹⁷ states that such a change to the “signal” \mathbf{g} modifies the corresponding DFT coefficients G_m (for the original contour sequence) to

$$G'_m = e^{i \cdot m \cdot \frac{2\pi k_s}{M}} \cdot G_m = e^{i \cdot m \cdot \varphi_s} \cdot G_m, \quad (6.78)$$

where $\varphi_s = \frac{2\pi k_s}{M}$ is a constant phase angle that is obviously proportional to the chosen start position k_s . Note that, in Eqn. (6.78), each DFT coefficient G_m is multiplied by a *different* complex quantity $e^{i \cdot m \cdot \varphi_s}$, which is of unit magnitude and varies with the frequency index m . In other words, the *magnitude* of any DFT coefficient G_m is again preserved but its *phase* changes individually. The coefficients of any Fourier descriptor pair $\text{FP}_m = \langle G_{-m}, G_{+m} \rangle$ thus become

$$G'_{-m} = e^{-i \cdot m \cdot \varphi_s} \cdot G_{-m} \quad \text{and} \quad G'_{+m} = e^{i \cdot m \cdot \varphi_s} \cdot G_{+m}, \quad (6.79)$$

that is, coefficient G_{-m} is rotated by the angle $-m \cdot \varphi_s$ and G_{+m} is rotated by $m \cdot \varphi_s$. In other words, a circular shift of the signal by $-k_s$ samples rotates the coefficients G_{-m}, G_{+m} by the same angle $m \cdot \varphi_s$ but in *opposite* directions. Therefore, the sum of both angles stays the same, i. e.,

$$\angle G'_{-m} + \angle G'_{+m} \equiv \angle G_{-m} + \angle G_{+m}. \quad (6.80)$$

In particular, we see from Eqn. (6.79) that shifting the start position modifies the coefficients of the *first* descriptor pair $\text{FP}_1 = \langle G_{-1}, G_{+1} \rangle$ to

$$G'_{-1} = e^{-i \cdot \varphi_s} \cdot G_{-1} \quad \text{and} \quad G'_{+1} = e^{i \cdot \varphi_s} \cdot G_{+1}. \quad (6.81)$$

The resulting *absolute* phase change of the coefficients G_{-1}, G_{+1} is $-\varphi_s, +\varphi_s$, respectively, and thus the change in phase *difference* is $2 \cdot \varphi_s$, i. e., the phase difference between the coefficients G_{-1}, G_{+1} is proportional to the chosen start position k_s (see Fig. 6.15).

¹⁷ See Vol. 2, Sec. 7.1.6 [21, p. 136].

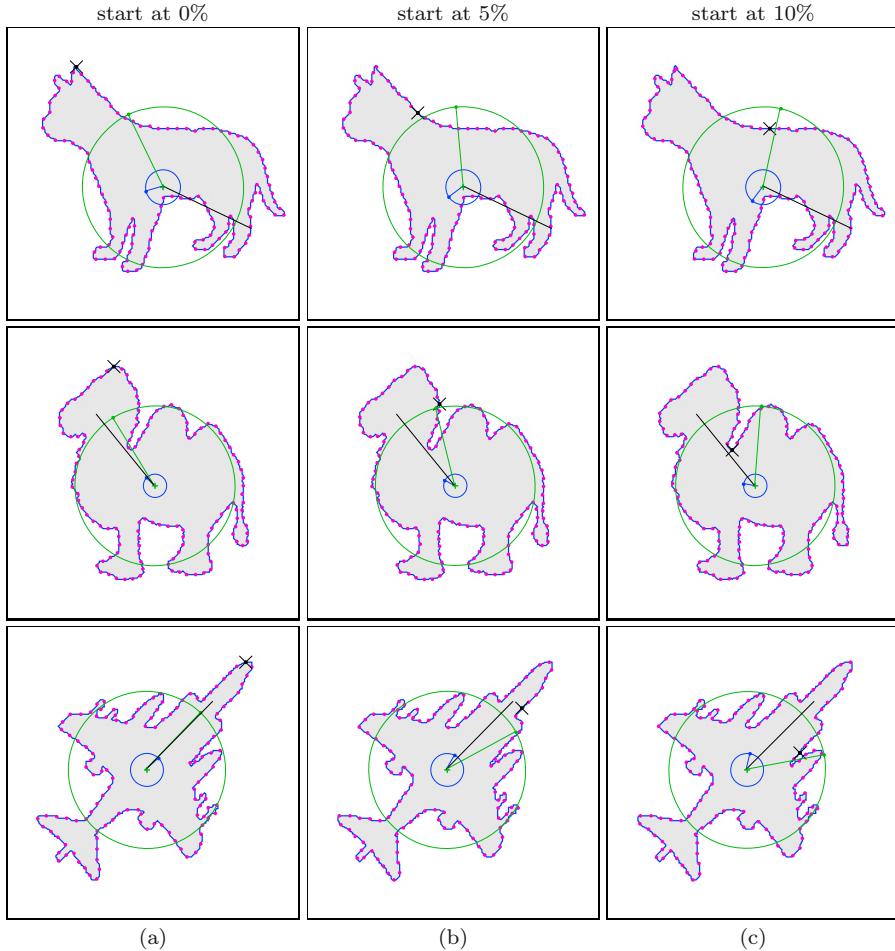


Figure 6.15 Effects of choosing different start points for contour sampling. The start point (marked \times on the contour) is set to 0%, 5%, 10% of the contour path length. The blue and green circles represent the partial reconstruction from single DFT coefficients G_{-1} and G_{+1} , respectively. The dot on each circle and the associated radial line shows the phase of the corresponding coefficient. The black line indicates the average orientation $(\angle G_{-1} + \angle G_{+1})/2$. It can be seen that the phase difference of G_{-1} and G_{+1} is directly related to the start position, but the average *orientation* (black line) remains unchanged.

6.4.5 Effects of phase removal

As described in the two previous sections, shape rotation (Sec. 6.4.3) and shift of start point (Sec. 6.4.4) both affect the phase of the Fourier coefficients but not their magnitude. The fact that magnitude is preserved suggests a simple solution for rotation invariant shape matching by simply ignoring the phase of the coefficients and comparing only their magnitude (see Sec. 6.6). Although

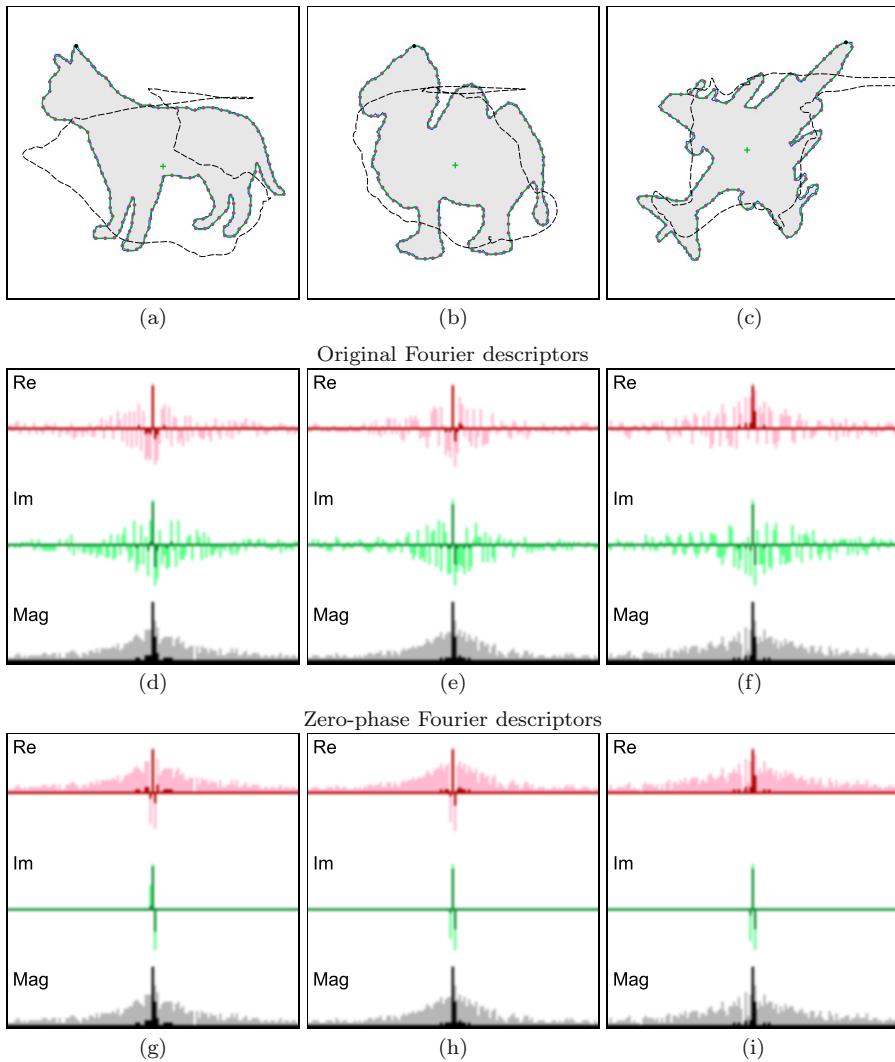


Figure 6.16 Effects of removing phase information. Original shapes and reconstruction after phase removal (a–c). Original Fourier coefficients (d–f) and zero-phase coefficients (g–i). The red and green plots in (d–i) show the real and imaginary components, respectively; gray plots show the coefficient magnitude. Dark-shaded bars correspond to the actual values, light-shaded bars are logarithmic values. The magnitude of the coefficients in (d–f) is the same as in (g–i).

this comes at the price of losing shape descriptiveness, magnitude-only descriptors are often used for shape matching. Clearly, the original shape cannot be reconstructed from a magnitude-only Fourier descriptor, as demonstrated in Fig. 6.16. It shows the reconstruction of shapes from Fourier descriptors with

the phase of all coefficients set to zero, except for G_{-1} , G_0 and G_{+1} (to preserve the shape's center and main orientation).

6.4.6 Direction of contour traversal

If the traversal direction of the contour samples is reversed, the coefficients of all Fourier descriptor pairs are exchanged, that is

$$G'_m = G_{-m \bmod M}. \quad (6.82)$$

This is equivalent to scaling the original shape by $s = -1$, as pointed out in Section 6.4.2. However, this is typically of no relevance in matching, since we can specify all contours to be sampled in either clockwise or counter-clockwise direction.

6.4.7 Reflection (symmetry)

Mirroring or reflecting a contour about the x -axis is equivalent to replacing each complex-valued point $g_k = x_k + i \cdot y_k$ by its *complex conjugate* g_k^* , that is,

$$g'_k = g_k^* = x_k - i \cdot y_k. \quad (6.83)$$

This change to the “signal” results in a modified DFT spectrum with coefficients

$$G'_m = G_{-m \bmod M}^*, \quad (6.84)$$

where G^* denotes the complex conjugate of the original DFT coefficients. Reflections about arbitrary axes can be described in the same way with additional rotations. Fourier descriptors can be made invariant against reflections, such that symmetric contours map to equivalent descriptors [139]. Note however that invariance to symmetry is not always desirable, for example, for distinguishing the silhouettes of left and right hands.

The relations between 2D point coordinates and the Fourier spectrum, as well as the effects of the above mentioned geometric shape transformations upon the DFT coefficients are compactly summarized in [Table 6.1](#).

6.5 Making Fourier descriptors invariant

As mentioned before, making a Fourier descriptor invariant to *translation* or absolute shape position is easy because the only affected coefficient is G_0 . Thus, setting coefficient G_0 to zero implicitly moves the center of the corresponding shape to the coordinate origin and thus creates a descriptor that is invariant to shape translation.

Table 6.1 Effects of spatial transformations upon the corresponding DFT spectrum. The original contour samples are denoted g_k , the DFT coefficients are G_m .

Operation	Contour samples	DFT coefficients
Forward transformation	g_k , for $k=0, \dots, M-1$	$G_m = \frac{1}{M} \cdot \sum_{k=0}^{M-1} g_k \cdot e^{-i2\pi m \frac{k}{M}}$
Backward transformation	$g_k = \sum_{m=0}^{M-1} G_m \cdot e^{i2\pi m \frac{k}{M}}$	G_m , for $m=0, \dots, M-1$
Translation (by $z \in \mathbb{C}$)	$g'_k = g_k + z$	$G'_m = \begin{cases} G_m + z & \text{for } m = 0 \\ G_m & \text{otherwise} \end{cases}$
Uniform scaling (by $s \in \mathbb{R}$)	$g'_k = s \cdot g_k$	$G'_m = s \cdot G_m$
Rotation about the origin (by β)	$g'_k = e^{i \cdot \beta} \cdot g_k$	$G'_m = e^{i \cdot \beta} \cdot G_m$
Shift of start position (by k_s)	$g'_k = g_{(k+k_s) \bmod M}$	$G'_m = e^{i \cdot m \cdot \frac{2\pi k_s}{M}} \cdot G_m$
Direction of contour traversal	$g'_k = g_{-k \bmod M}$	$G'_m = G_{-m \bmod M}$
Reflection about the x -axis	$g'_k = g_k^*$	$G'_m = G_{-m \bmod M}^*$

Invariance against a change in *scale* is also a simple issue because it only multiplies the magnitude of all Fourier coefficients by the same real-valued scale factor, which can be easily normalized.

A more challenging task is to make Fourier descriptors invariant against shape *rotation* and shift of the contour *starting point*, because they jointly affect the phase of the Fourier coefficients. If matching is to be based on the complex-valued Fourier descriptors (not on coefficient magnitude only) to achieve better shape discrimination, the phase changes introduced by shape rotation and start point shifts must be eliminated first. However, due to noise and possible ambiguities, this is not a trivial problem (see also [102, 103, 108, 139]).

6.5.1 Scale invariance

As mentioned in Section 6.4.2, the magnitude G_{+1} is often used as a reference to normalize for scale, since G_{+1} is typically (though not always) the Fourier coefficient with the largest magnitude. Alternatively, one could use the size of the fundamental ellipse, defined by the Fourier descriptor pair FP_1 , to measure the overall scale, for example, by normalizing to

$$G_m^S \leftarrow \frac{1}{|G_{-1}| + |G_{+1}|} \cdot G_m, \quad (6.85)$$

which normalizes the *length* of the major axis $a_1 = |G_{-1}| + |G_{+1}|$ (see Eqn. (6.46)) of the fundamental ellipse to unity. Another alternative is

$$G_m^S \leftarrow \frac{1}{(|G_{-1}| \cdot |G_{+1}|)^{1/2}} \cdot G_m, \quad (6.86)$$

which normalizes the *area* of the fundamental ellipse. Since all variants in Eqns. (6.72, 6.85, 6.86) scale the coefficients G_m by a fixed (real-valued) factor, the shape information contained in the Fourier descriptor remains unchanged.

There are shapes, however, where coefficients G_{+1} and/or G_{-1} are small or almost vanish to zero, such that they are not always a reliable reference for scale. An obvious solution is to include the complete set of Fourier coefficients by standardizing the *norm* of the coefficient vector \mathbf{G} to unity in the form

$$G_m^S \leftarrow \frac{1}{\|\mathbf{G}\|} \cdot G_m, \quad (6.87)$$

(assuming that $G_0 = 0$). In general, the L_2 norm of a complex-valued vector $Z = (z_0, z_1, \dots, z_{M-1})$, $z_i \in \mathbb{C}$, is defined as

$$\|Z\| = \left(\sum_{i=1}^{M-1} |z_i|^2 \right)^{1/2} = \left(\sum_{i=1}^{M-1} \operatorname{Re}(z_i)^2 + \operatorname{Im}(z_i)^2 \right)^{1/2}. \quad (6.88)$$

Scaling the vector Z by the reciprocal of its norm yields a vector with unit norm, that is,

$$\left\| \frac{1}{\|Z\|} \cdot Z \right\| = 1. \quad (6.89)$$

To normalize a given Fourier descriptor \mathbf{G} , we use all elements except G_0 (which relates to the absolute position of the shape and is not relevant for its shape). The following substitution makes \mathbf{G} scale invariant by normalizing the remaining sub-vector $(G_1, G_2, \dots, G_{M-1})$ to

$$G_m^S \leftarrow \begin{cases} G_m & \text{for } m = 0, \\ \frac{1}{\sqrt{\nu}} \cdot G_m & \text{for } 1 \leq m < M, \end{cases} \quad \text{with } \nu = \sum_{m=1}^{M-1} |G_m|^2. \quad (6.90)$$

See procedure `MAKESCALEINVARIANT(\mathbf{G})` in [Alg. 6.6](#) (lines 7–15) for a summary of this step.

6.5.2 Start point invariance

As discussed in Sections 6.4.3 and 6.4.4, respectively, shape rotation and shift of start point both affect the phase of the Fourier coefficients in a combined manner, without altering their magnitude. In particular, if the shape is rotated by some angle β (see Eqn. (6.78)) and the start position is shifted by k_s samples (see Eqn. (6.75)), then each Fourier coefficient G_m is modified to

$$G'_m = e^{i \cdot \beta} \cdot e^{i \cdot m \cdot \varphi_s} \cdot G_m = e^{i \cdot (\beta + m \cdot \varphi_s)} \cdot G_m, \quad (6.91)$$

where $\varphi_s = \frac{2\pi k_s}{M}$ is the corresponding *start point phase*. Thus, the incurred phase shift is not only different for each coefficient but simultaneously depends on the rotation angle β and the start point phase φ_s . Normalization in this case means to remove these phase shifts, which would be straightforward if β and φ_s were known. We derive these two parameters one after the other, starting with the calculation of the start point phase φ_s , which we describe below, followed by the estimation of the rotation β , shown subsequently in Section 6.5.3.

To normalize the Fourier descriptor of a particular shape to a “canonical” start point, we need a quantity that can be calculated from the Fourier spectrum and only depends on the start point phase φ_s but is independent of the rotation β . From Eqn. (6.79) and Fig. 6.15 we see that the phase *difference* within any Fourier descriptor pair $\langle G_{-m}, G_m \rangle$ is proportional to the start point phase φ_s and independent to shape rotation β , since the latter rotates all coefficients by the same angle. Thus, we look for a quantity that depends only on the phase *differences* within Fourier descriptor pairs. This is accomplished, for example, by the function

$$f_p(\varphi) = \sum_{m=1}^{M_p} [e^{-i \cdot m \cdot \varphi} \cdot G_{-m}] \otimes [e^{i \cdot m \cdot \varphi} \cdot G_m], \quad (6.92)$$

where parameter φ is an arbitrary start point phase, M_p is the number of coefficient pairs, and \otimes denotes the “cross product” between two Fourier coefficients.¹⁸ Given a particular start point phase φ , the function in Eqn. (6.92) yields the sum of the cross products of each coefficient pair $\langle G_{-m}, G_m \rangle$, for $m = 1, \dots, M_p$. If each of the complex-valued coefficients is interpreted as a vector in the 2D plane, the magnitude of their cross product is proportional to the *area* of the enclosed parallelogram. The enclosed area is potentially large only if *both* vectors are of significant length, which means that the corresponding ellipse has a distinct eccentricity and orientation. Note that the sign of the cross product may be positive or negative and depends on the relative orientation or “handedness” of the two vectors.

Since the function $f_p(\varphi)$ is based only on the *relative* orientation (phase) of the involved coefficients, it is invariant to a shape rotation β , which shifts all coefficients by the same angle (see Eqn. (6.75)). As shown in Fig. 6.17, $f_p(\varphi)$ is periodic with π and its phase is proportional to the actual start point shift. We choose the angle φ that *maximizes* $f_p(\varphi)$ as the “canonical” start point phase φ_A , that is,

$$\varphi_A = \underset{0 \leq \varphi < \pi}{\operatorname{argmax}} f_p(\varphi). \quad (6.93)$$

¹⁸ In analogy to 2D vector notation, we define the “cross product” of two complex quantities $z_1 = (a_1, b_1)$ and $z_2 = (a_2, b_2)$ as $z_1 \otimes z_2 = a_1 \cdot b_2 - b_1 \cdot a_2 = |z_1| \cdot |z_2| \cdot \sin(\theta_2 - \theta_1)$.

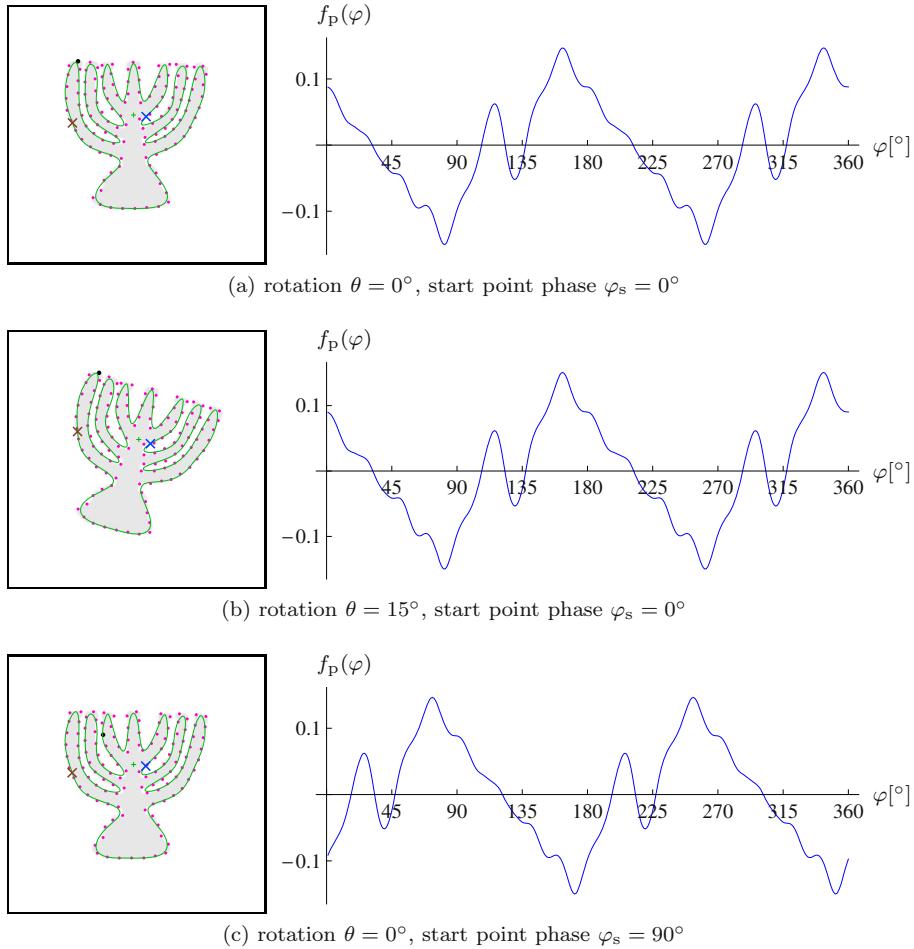


Figure 6.17 Plot of function $f_p(\varphi)$ used for start point normalization. In the figures on the left, the real start point is marked by a black dot. The normalized start points φ_A and $\varphi_B = \varphi_A + \pi$ are marked by a blue and a brown cross, respectively. They correspond to the two peak positions of the function $f_p(\varphi)$, as defined in Eqn. (6.92), separated by a fixed phase shift of $\pi = 180^\circ$ (right). The function is invariant under shape rotation, as demonstrated in (b), where the shape is rotated by 15° but sampled from the same start point as in (a). However, the phase of $f_p(\varphi)$ is proportional to the start point shift, as shown in (c), where the start point is chosen at 25% ($\varphi_s = 90^\circ$) of the boundary path length. The functions were calculated after scale normalization, using $M_p=25$ Fourier coefficient pairs.

However, since $f_p(\varphi) = f_p(\varphi + \pi)$, there is also a second candidate phase

$$\varphi_B = \varphi_A + \pi, \quad (6.94)$$

displaced by $\pi = 180^\circ$. The two “canonical” start points corresponding to φ_A and φ_B , respectively, are marked on the reconstructed shapes in Fig. 6.17.

Although it might seem easy at first to resolve this 180° ambiguity of the start point phase, this turns out to be difficult to achieve in general from the Fourier coefficients alone. Several functions have been proposed for this purpose that work well for certain shapes but fail on others, including the “positive real energy” function suggested in [139]. In particular, any decision based on the magnitude or phase of a *single* coefficient (or a single coefficient pair) must eventually fail, since none of the coefficients is guaranteed to have a significant magnitude. With vanishing coefficient magnitude, phase measurements become unreliable and may be very susceptible to noise.

The complete process of start point normalization is summarized in [Alg. 6.7](#). The start point phase φ_A is found numerically by evaluating the function $f_p(\varphi)$ at 400 discrete steps for $\varphi = 0, \dots, \pi$ (lines 6–16). For practical use, this exhaustive method should be substituted by a more efficient and accurate optimization technique (for example, using Brent’s method [109, Ch. 10]).¹⁹ Given the estimated start point phase φ_A for the Fourier descriptor \mathbf{G} , two normalized versions $\mathbf{G}^A, \mathbf{G}^B$ are calculated as

$$\begin{aligned}\mathbf{G}^A: G_m^A &\leftarrow G_m \cdot e^{i \cdot m \cdot \varphi_A}, \\ \mathbf{G}^B: G_m^B &\leftarrow G_m \cdot e^{i \cdot m \cdot (\varphi_A + \pi)},\end{aligned}\tag{6.95}$$

for $m = -M_p, \dots, M_p, m \neq 0$. Note that start point normalization does not require the Fourier descriptor \mathbf{G} to be normalized for translation and scale (see Sec. 6.5.1).

6.5.3 Rotation invariance

After normalizing for starting point, the orientation of the fundamental ellipse (formed by the descriptor pair $\langle G_{-1}, G_{+1} \rangle$) could be assumed to be a reliable reference for global shape rotation. However, for certain shapes (for example, regular polyhedra with an even number of faces), G_{-1} may vanish. Therefore, we recover the overall shape orientation from the vector obtained as the weighted sum of *all* Fourier coefficients, that is,

$$z = \sum_{m=1}^{M_p} \frac{1}{m} \cdot (G_{-m} + G_{+m}),\tag{6.96}$$

where the $1/m$ serves as a weighting factor, giving stronger emphasis to the low-frequency coefficients and attenuating the influence of the high-frequency coefficients. The resulting shape orientation estimate is

$$\beta = \arg z = \tan^{-1} \left(\frac{\text{Im}(z)}{\text{Re}(z)} \right).\tag{6.97}$$

¹⁹ The accompanying Java implementation uses the class `BrentOptimizer` from the *Apache Commons Math Library* [2] for this purpose.

To normalize $\mathbf{G}^A, \mathbf{G}^B$ (obtained in Eqn. (6.95)) for shape orientation, we rotate each coefficient (except G_0) by $-\beta$, that is,

$$\begin{aligned}\mathbf{G}^A: G_m^A &\leftarrow G_m^A \cdot e^{-i\cdot\beta}, \\ \mathbf{G}^B: G_m^B &\leftarrow G_m^B \cdot e^{-i\cdot\beta},\end{aligned}\quad (6.98)$$

for $m = -M_p, \dots, M_p, m \neq 0$. See procedure MAKEROTATIONINVARIANT(\mathbf{G}) in Alg. 6.6 (lines 16–24) for a summary of these steps.

6.5.4 Other approaches

The above normalization for making Fourier descriptors invariant to geometric transformations deviates from the published “classic” techniques in certain ways, but also adopts some common elements. As representative examples, we briefly discuss two of these techniques (already referenced earlier) in the following:

Persoon and Fu [102, 103] proposed (in what they call the “suboptimal” approach) to choose the parameters s (common scale factor), β (shape rotation), and φ_s (start point phase) such that the modified coefficients G'_{-1}, G'_{+1} are both imaginary and $|G_{-1} + G_{+1}| = 1$. As argued in [139], this method leaves a $\pm 180^\circ$ ambiguity for the shape orientation. Also, it requires that both G_{-1}, G_{+1} have significant magnitude, which may not be true for G_{-1} in case of shapes that are circularly symmetric (e.g., equilateral triangles, squares, pentagons etc.).

Wallace and Wintz [139] use $|G_{+1}|$ as the common scale factor, because the coefficient G_{+1} typically has the largest magnitude. The phase of G_{+1} , denoted $\phi_1 = \angle G_{+1}$, and the phase of another coefficient G_k ($k > 0$) with the second-largest magnitude and phase $\phi_k = \angle G_k$ are used to compensate for rotation and starting point. Coefficients are phase shifted such that both G'_{+1} and G'_k have zero phase. This is accomplished by multiplying all coefficients in the form

$$G'_m = G_m \cdot e^{i \cdot [(m-k) \cdot \phi_1 + (1-m) \cdot \phi_k] \cdot (k-1)}, \quad (6.99)$$

for $-\frac{M}{2} + 1 \leq m \leq \frac{M}{2}$ (also used in [108]). Depending on the index k of the second-largest coefficient, there exist $|k-1|$ different orientation/start point combinations to obtain zero-phase in G'_{+1} and G'_k . If $k = 2$, then $|k-1| = 1$, thus the solution is unique and Eqn. (6.99) simplifies to

$$G'_m = G_m \cdot e^{i \cdot [(m-2) \cdot \phi_1 + (1-m) \cdot \phi_2]}, \quad (6.100)$$

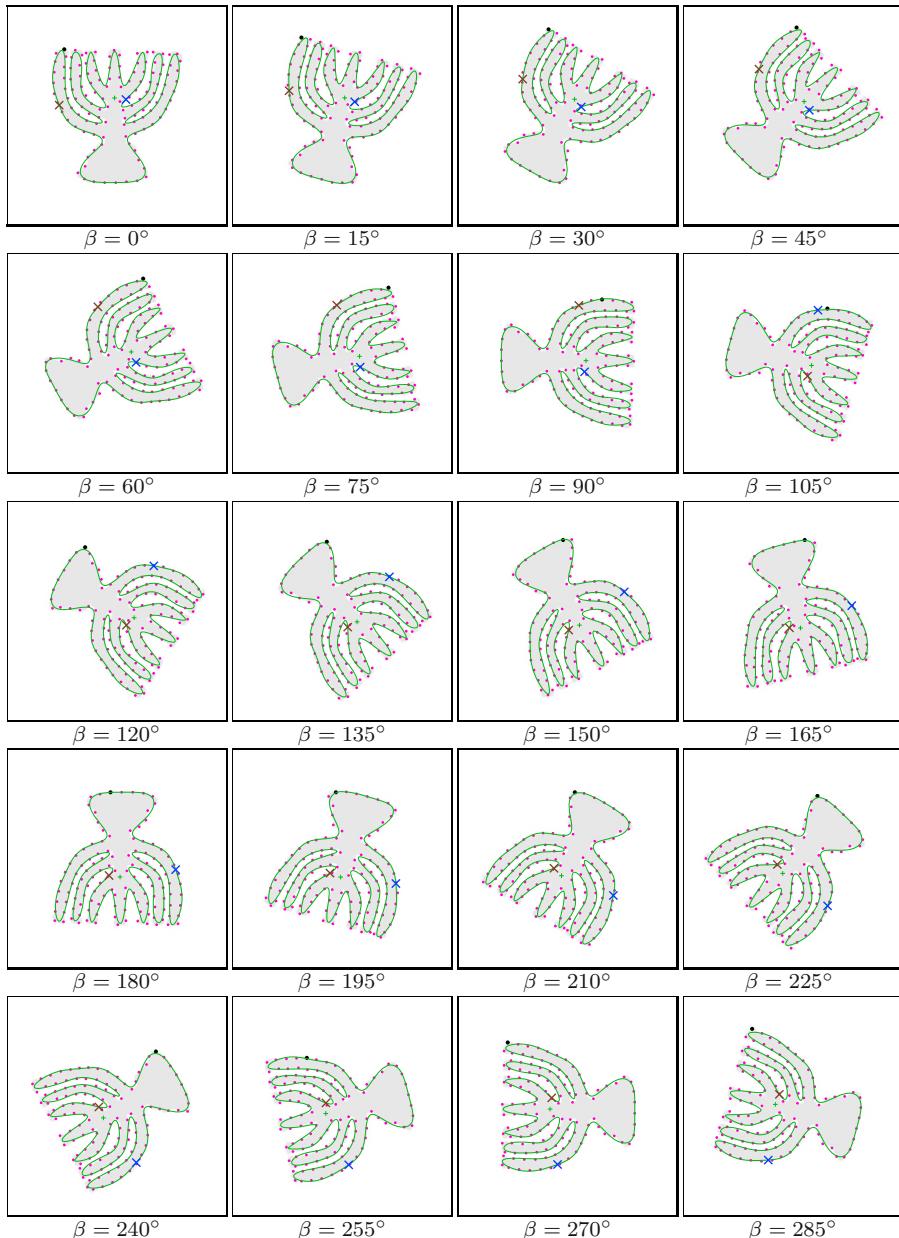


Figure 6.18 Start point normalization under varying shape rotation (β). The real start point (which varies with shape rotation) is marked by a black dot. The two normalized start points φ_A and $\varphi_B = \varphi_A + \pi$ (calculated with the procedure in Alg. 6.7) are marked by a blue and a brown \times , respectively. 25 Fourier coefficient pairs are used for the normalization and shape reconstruction. Inaccuracies are due to shape variations caused by the use of nearest-neighbor interpolation for the image rotation.

Algorithm 6.6 Making Fourier descriptors invariant against scale, shift of start point, and shape rotation. For a given Fourier descriptor \mathbf{G} , procedure **MAKESTARTPOINTINVARIANT(\mathbf{G})** returns a pair of normalized Fourier descriptors $(\mathbf{G}^A, \mathbf{G}^B)$, one for each normalized start point phase φ_A and $\varphi_B = \varphi_A + \pi$.

```

1: MAKEINVARIANT( $\mathbf{G}$ )
   Input:  $\mathbf{G}$ , Fourier descriptor with  $M_p$  coefficient pairs.
   Returns a pair of normalized Fourier descriptors  $\mathbf{G}^A, \mathbf{G}^B$ , with a start
   point phase offset by  $180^\circ$ .
2: MAKESCALEINVARIANT( $\mathbf{G}$ )                                 $\triangleright$  see below
3:  $(\mathbf{G}^A, \mathbf{G}^B) \leftarrow \text{MAKESTARTPOINTINVARIANT}(\mathbf{G})$        $\triangleright$  see Alg. 6.7
4: AKEROTATIONINVARIANT( $\mathbf{G}^A$ )                                 $\triangleright$  see below
5: AKEROTATIONINVARIANT( $\mathbf{G}^B$ )
6: return  $(\mathbf{G}^A, \mathbf{G}^B)$ .
```

```

7: MAKESCALEINVARIANT( $\mathbf{G}$ )
   Modifies  $\mathbf{G}$  by unifying its norm and returns the scale factor  $\nu$ .
8:  $s \leftarrow 0$                                           $\triangleright s \in \mathbb{R}$ 
9: for  $m \leftarrow 1, \dots, M_p$  do
10:     $s \leftarrow s + |\mathbf{G}(-m)|^2 + |\mathbf{G}(m)|^2$ 
11:     $\nu \leftarrow 1/\sqrt{s}$ 
12:    for  $m \leftarrow 1, \dots, M_p$  do
13:        $\mathbf{G}(-m) \leftarrow \nu \cdot \mathbf{G}(-m)$ 
14:        $\mathbf{G}(m) \leftarrow \nu \cdot \mathbf{G}(m)$ 
15:    return  $\nu$ .
```

```

16: AKEROTATIONINVARIANT( $\mathbf{G}$ )
   Modifies  $\mathbf{G}$  and returns the estimated rotation angle  $\beta$ .
17:  $z \leftarrow 0 + i \cdot 0$                                           $\triangleright z \in \mathbb{C}$ 
18: for  $m \leftarrow 1, \dots, M_p$  do
19:     $z \leftarrow z + \frac{1}{m} \cdot (\mathbf{G}(-m) + \mathbf{G}(m))$            $\triangleright$  complex addition!
20:     $\beta \leftarrow \arg z$ 
21: for  $m \leftarrow 1, \dots, M_p$  do                                 $\triangleright$  rotate all coefficients by  $-\beta$ 
22:     $\mathbf{G}(-m) \leftarrow e^{-i \cdot \beta} \cdot \mathbf{G}(-m)$ 
23:     $\mathbf{G}(m) \leftarrow e^{-i \cdot \beta} \cdot \mathbf{G}(m)$ 
24: return  $\beta$ .
```

with $\phi_2 = \arg G_2$.²⁰ Otherwise the ambiguity is resolved by calculating an “ambiguity-resolving” criterion for each of the $|k - 1|$ solutions, for example,

²⁰ Unfortunately, the general use of coefficient G_2 as a phase reference is critical, because the magnitude of G_2 may be small or even zero for certain symmetrical shapes (including all regular polygons with an even number of faces).

Algorithm 6.7 Making Fourier descriptors invariant to the shift of start point. Since the result is ambiguous by 180° , two normalized descriptors ($\mathbf{G}^A, \mathbf{G}^B$) are returned, with the start point phase set to φ_A and $\varphi_A + \pi$, respectively.

```

1: MAKESTARTPOINTINVARIANT( $\mathbf{G}$ )
   Input:  $\mathbf{G}$ , Fourier descriptor with  $M_p$  coefficient pairs.
   Returns a pair of new Fourier descriptors  $\mathbf{G}^A, \mathbf{G}^B$ , normalized to the
   start point phase  $\varphi_A$  and  $\varphi_A + \pi$ , respectively.

2:  $\varphi_A \leftarrow \text{GETSTARTPOINTPHASE}(\mathbf{G})$                                 ▷ see below
3:  $\mathbf{G}^A \leftarrow \text{SHIFTSTARTPOINTPHASE}(\mathbf{G}, \varphi_A)$                   ▷ see below
4:  $\mathbf{G}^B \leftarrow \text{SHIFTSTARTPOINTPHASE}(\mathbf{G}, \varphi_A + \pi)$ 
5: return  $(\mathbf{G}^A, \mathbf{G}^B)$ .
```

```

6: GETSTARTPOINTPHASE( $\mathbf{G}$ )
   Returns  $\varphi$  maximizing  $f_p(\mathbf{G}, \varphi)$ , with  $\varphi \in [0, \pi]$ . The maximum is
   found by simple brute-force search (for illustration only).

7:  $c_{\max} \leftarrow -\infty$ 
8:  $\varphi_{\max} \leftarrow 0$ 
9:  $K \leftarrow 400$                                          ▷ do  $K$  search steps over  $0, \dots, \pi$ 
10: for  $k \leftarrow 0, \dots, K-1$  do                      ▷ find  $\varphi$  maximizing  $f_p(\mathbf{G}, \varphi)$ 
11:    $\varphi \leftarrow \pi \cdot \frac{k}{K}$ 
12:    $c \leftarrow f_p(\mathbf{G}, \varphi)$ 
13:   if  $c > c_{\max}$  then
14:      $c_{\max} \leftarrow c$ 
15:      $\varphi_{\max} \leftarrow \varphi$ 
16: return  $\varphi_{\max}$ .
```

```

17:  $f_p(\mathbf{G}, \varphi)$                                      ▷ see Eqn. (6.92)
18:    $s \leftarrow 0$ 
19:   for  $m \leftarrow 1, \dots, M_p$  do
20:      $z_1 \leftarrow \mathbf{G}(-m) \cdot e^{-i \cdot m \cdot \varphi}$ 
21:      $z_2 \leftarrow \mathbf{G}(m) \cdot e^{i \cdot m \cdot \varphi}$ 
22:      $s \leftarrow s + \text{Re}(z_1) \cdot \text{Im}(z_2) - \text{Im}(z_1) \cdot \text{Re}(z_2)$     ▷  $= s + (z_1 \otimes z_2)$ 
23:   return  $s$ .
```

```

24: SHIFTSTARTPOINTPHASE( $\mathbf{G}, \varphi$ )           ▷ start-point normalize  $\mathbf{G}$  by  $\varphi$ 
25:    $\mathbf{G}' \leftarrow \text{DUPLICATE}(\mathbf{G})$ 
26:   for  $m \leftarrow 1, \dots, M_p$  do
27:      $\mathbf{G}'(-m) \leftarrow \mathbf{G}(-m) \cdot e^{-i \cdot m \cdot \varphi}$ 
28:      $\mathbf{G}'(m) \leftarrow \mathbf{G}(m) \cdot e^{i \cdot m \cdot \varphi}$ 
29:   return  $\mathbf{G}'$ .
```

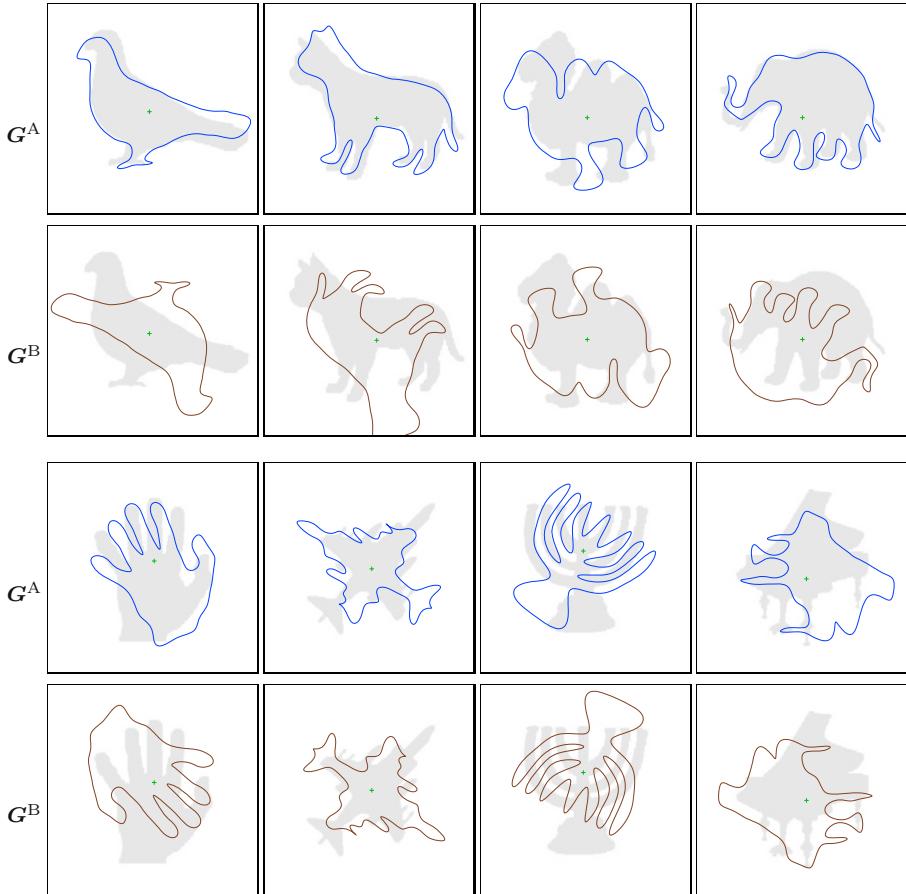


Figure 6.19 Reconstruction of various shapes from Fourier descriptors normalized for start point shift and shape rotation. The blue shapes (rows 1, 3) correspond to the normalized Fourier descriptors \mathbf{G}^A with start point phase φ_A . The brown shapes (rows 2, 4) correspond to the normalized Fourier descriptors \mathbf{G}^B with start point phase $\varphi_B = \varphi_A + \pi$. No scale normalization was applied for better visualization.

the amount of “positive real energy”,

$$\sum_{m=1}^{N-1} \operatorname{Re}(G'_m) \cdot |\operatorname{Re}(G'_m)|,$$

as defined in [139] (alternative functions are suggested in [108]). This leaves the problem that, for matching, the normalization of the investigated shape descriptor must be based on the same set of dominant coefficients as the reference descriptor. Alternatively, one could memorize the relevant coefficient indexes for every reference descriptor, but then different normalizations must

be applied for matching against multiple models in a database.

6.6 Shape matching with Fourier descriptors

A typical use of Fourier descriptors is to see if a given shape is identical or similar to an exemplar contained in a database of reference shapes. For this purpose, we need to define a distance measure that quantifies the difference between two Fourier shape descriptors \mathbf{G}_1 and \mathbf{G}_2 . In the following, we assume that the Fourier descriptors $\mathbf{G}_1, \mathbf{G}_2$ are at least scale-normalized (as described in [Alg. 6.6](#)) and of identical length, each with M_p coefficient pairs.

6.6.1 Magnitude-only matching

In the simplest case, we only use the *magnitude* of the Fourier coefficients for comparison and entirely ignore their phase, using the distance function

$$\begin{aligned} \text{dist}_M(\mathbf{G}_1, \mathbf{G}_2) &= \left(\sum_{\substack{m=-M_p, \\ m \neq 0}}^{M_p} (|\mathbf{G}_1(m)| - |\mathbf{G}_2(m)|)^2 \right)^{1/2} \\ &= \left(\sum_{m=1}^{M_p} (|\mathbf{G}_1(-m)| - |\mathbf{G}_2(-m)|)^2 + (|\mathbf{G}_1(m)| - |\mathbf{G}_2(m)|)^2 \right)^{1/2}, \end{aligned} \quad (6.101)$$

where M_p denotes the number of FD pairs used for matching. Note that Eqn. (6.101) is simply the L_2 norm of the magnitude difference vector, and of course other norms (such as L_1 or L_∞) could be used as well. The advantage of the magnitude-only approach is that no normalization (except for scale) is required. Its drawback is that even highly dissimilar shapes might be mistakenly matched, since the removal of phase naturally eliminates shape information that is possibly essential for discrimination. As demonstrated in [Fig. 6.20](#), a given Fourier magnitude vector may correspond to a great diversity of shapes, and thus the subspace of “equivalent” shapes defined by the magnitude-only distance dist_M is quite large.

Nevertheless, magnitude-only matching may be sufficient in situations where the reference shapes are not too similar. In a sense, the operation of reducing the complex-valued Fourier descriptors to their magnitude vectors can be viewed as a *hash* function. While potentially many different shapes may produce (i. e., “hash to”) similar Fourier magnitude vectors, the chance of two real shapes mapping to the same vector (and thus being confused) may be relatively small. Thus, particularly considering its simplicity (only scale-normalization of descriptors is required), magnitude-based matching can be quite effective in practice.

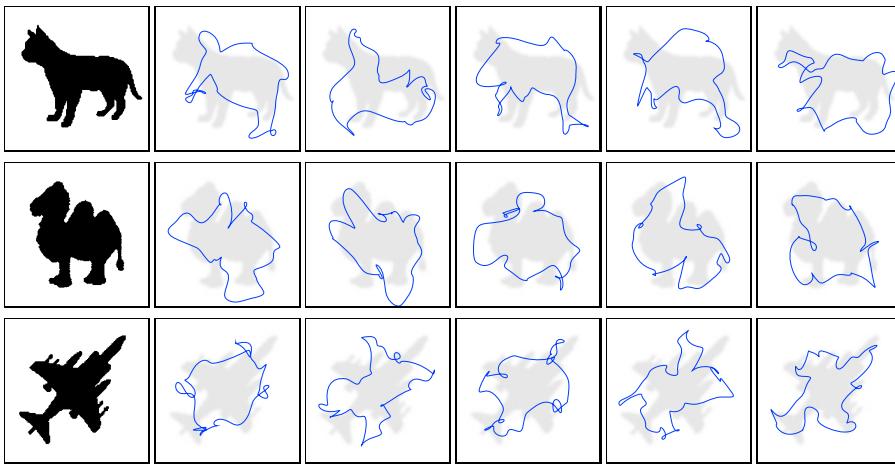


Figure 6.20 Randomized phase. Reconstruction of shapes from Fourier descriptors with the phase of all coefficients (except G_{-1} , G_0 and G_{+1}) individually randomized. Note that the magnitude of the coefficients is exactly the same for each shape category, so all blue shapes would be considered “equivalent” to the original shape at the left by a magnitude-only matcher.

Figure 6.21 shows the pair-wise magnitude-only distances (blue cells, values are $10 \times \text{dist}_M$) between various sample shapes. The corresponding intra-class distances, given in [Fig. 6.22](#), are typically more than one order of magnitude smaller, indicating that shape discrimination based on this measure should be fairly reliable.

	bird	cat	camel	elephant	hand	harrier	menora	piano	creature
bird									
	0.000	4.529	4.482	5.007	5.525	4.314	7.554	5.174	7.076
	3.156	0.000	5.788	4.708	5.711	5.701	7.181	5.543	7.677
	2.648	3.005	0.000	4.429	5.573	3.726	7.014	4.013	8.480
	3.487	1.933	2.549	0.000	6.100	4.618	5.338	4.369	8.743
	4.627	3.146	3.132	2.372	0.000	6.079	8.540	5.580	7.136
	3.712	3.707	2.687	3.553	4.294	0.000	6.818	4.958	8.284
	5.835	4.893	4.563	4.162	3.788	5.775	0.000	6.826	11.072
	4.037	2.426	2.610	1.876	1.848	3.405	4.315	0.000	7.666
	6.030	6.261	5.554	5.492	5.955	5.914	5.190	6.049	0.000

 $\text{dist}_M(\mathbf{G}_1, \mathbf{G}_2)$ $\text{dist}_C(\mathbf{G}_1, \mathbf{G}_2)$

Figure 6.21 Inter-class Fourier descriptor distances (magnitude-only and complex-valued). Numbers inside the green fields (lower-left half of the matrix) are the magnitude-only distances dist_M (see Eqn. (6.101)). Numbers in blue fields (upper-right half of the matrix) are the complex-valued distances dist_C (see Eqn. (6.103)). Shapes were sampled uniformly at 125 contour positions, with 25 coefficient pairs. Fourier descriptors were normalized for scale, start point and rotation. All distance values are multiplied by 10.

$\alpha =$	0°	17°	34°	51°	68°	85°	102°	119°	136°	153°	170°	187°	204°
dist _M	0.000	0.070	0.126	0.151	0.103	0.058	0.143	0.107	0.195	0.190	0.105	0.078	0.053
dist _C	0.000	0.141	0.222	0.299	0.198	0.111	0.274	0.159	0.313	0.400	0.142	0.162	0.092
dist _M	0.000	0.134	0.144	0.176	0.167	0.055	0.104	0.206	0.227	0.135	0.164	0.083	0.174
dist _C	0.000	0.222	0.214	0.252	0.244	0.081	0.141	0.310	0.339	0.197	0.231	0.157	0.281
dist _M	0.000	0.117	0.346	0.147	0.142	0.141	0.109	0.100	0.125	0.163	0.099	0.147	0.106
dist _C	0.000	0.229	0.728	0.367	0.310	0.386	0.161	0.186	0.202	0.252	0.141	0.191	0.271
dist _M	0.000	0.121	0.195	0.272	0.170	0.057	0.135	0.175	0.216	0.176	0.092	0.112	0.160
dist _C	0.000	0.180	0.317	0.392	0.278	0.080	0.218	0.257	0.307	0.266	0.160	0.198	0.248
dist _M	0.000	0.127	0.138	0.179	0.130	0.048	0.131	0.115	0.329	0.173	0.202	0.109	0.132
dist _C	0.000	0.179	0.186	0.361	0.180	0.085	0.234	0.188	0.496	0.263	0.313	0.182	0.195
dist _M	0.000	0.234	0.171	0.224	0.095	0.090	0.106	0.189	0.228	0.170	0.079	0.121	0.213
dist _C	0.000	0.433	0.290	0.317	0.147	0.129	0.197	0.276	0.344	0.251	0.146	0.197	0.308
dist _M	0.000	0.163	0.148	0.131	0.213	0.116	0.228	0.322	0.334	0.205	0.253	0.108	0.122
dist _C	0.000	0.570	0.330	0.395	0.456	0.169	0.271	0.401	0.465	0.295	0.440	0.149	0.251
dist _M	0.000	0.164	0.186	0.161	0.186	0.101	0.112	0.252	0.159	0.150	0.169	0.104	0.201
dist _C	0.000	0.264	0.362	0.311	0.255	0.175	0.148	0.576	0.230	0.267	0.232	0.142	0.284
dist _M	0.000	0.154	0.190	0.167	0.103	0.084	0.180	0.390	0.210	0.123	0.194	0.084	0.131
dist _C	0.000	0.203	0.260	0.248	0.141	0.108	0.232	0.447	0.308	0.171	0.234	0.120	0.160

Figure 6.22 Intra-class Fourier descriptor distances (magnitude-only and complex-valued). The reference images (0° column) were rotated by angle α (multiples of 17°), using no (i.e., nearest-neighbor) interpolation. Numbers inside the blue fields are the magnitude-only distances dist_M (see Eqn. (6.101)). Numbers inside the green fields are the complex-valued distances dist_C (see Eqn. (6.103)). Shapes were sampled uniformly at 125 contour positions, with 25 coefficient pairs. Fourier descriptors were normalized for scale, start point shift and shape rotation. All distance values are multiplied by 10. Note that all *intra*-class distances are roughly one order of magnitude smaller than the *inter*-class distances shown in Fig. 6.21.

6.6.2 Complex (phase-preserving) matching

Assuming that the Fourier descriptors \mathbf{G}_1 and \mathbf{G}_2 have been normalized for scale, start point shift and shape rotation (see [Alg. 6.6](#)), we can use the following function to measure their mutual distance:

$$\text{dist}_C(\mathbf{G}_1, \mathbf{G}_2) = \left(\sum_{\substack{m=-M_p, \\ m \neq 0}}^{M_p} |\mathbf{G}_1(m) - \mathbf{G}_2(m)|^2 \right)^{1/2} \quad (6.102)$$

$$= \left(\sum_{m=1}^{M_p} |\mathbf{G}_1(-m) - \mathbf{G}_2(-m)|^2 + |\mathbf{G}_1(m) - \mathbf{G}_2(m)|^2 \right)^{1/2} \quad (6.103)$$

$$= \left(\sum_{\substack{m=-M_p, \\ m \neq 0}}^{M_p} [\text{Re}(\mathbf{G}_1(m)) - \text{Re}(\mathbf{G}_2(m))]^2 + [\text{Im}(\mathbf{G}_1(m)) - \text{Im}(\mathbf{G}_2(m))]^2 \right)^{1/2}.$$

Again, this is simply the L_2 norm of the complex-valued difference vector $\mathbf{G}_1 - \mathbf{G}_2$ (ignoring the coefficients at $m = 0$), which could be substituted by some other norm. Since the phase of the involved coefficients is fully preserved, a zero distance between two Fourier descriptors means that they represent the very same shape. Thus the set of equivalent shapes defined by the distance function in Eqn. (6.103) is much smaller than the one defined by the magnitude-only distance in Eqn. (6.101). Consequently, the probability of two different shapes being confused for the same is also significantly smaller with this distance measure.

Complex inter-class and intra-class distance values for the set of sample shapes are listed in [Figs. 6.21–6.22](#). Notice that, with the normalization described in [Alg. 6.6](#), the complex intra-class distance values in [Fig. 6.22](#) (which should be as small as possible) are typically about twice as large as the corresponding magnitude-only distance values, but still an order of magnitude smaller than comparable inter-class values in [Fig. 6.21](#), so reliable shape discrimination should be possible.

The price paid for the increased discriminative power is the extra work necessary for normalizing the Fourier descriptors for start point and shape rotation (in addition to scale), as described in [Alg. 6.6](#). Note that this involves the comparison with *two* normalized descriptors to cope with the unresolved 180° ambiguity of the start point normalization (see Eqns. (6.93–6.94)). For example, assume we wish to compare two shapes V_1, V_2 with Fourier descriptors $\mathbf{G}_1, \mathbf{G}_2$, respectively. We first calculate the corresponding invariant descriptors (as described in [Alg. 6.6](#)),

$$(\mathbf{G}_1^A, \mathbf{G}_1^B) \leftarrow \text{MAKEINVARIANT}(\mathbf{G}_1),$$

$$(\mathbf{G}_2^A, \mathbf{G}_2^B) \leftarrow \text{MAKEINVARIANT}(\mathbf{G}_2).$$

Now we use Eqn. (6.103) to calculate the complex-valued distance as

$$d_{\min} = \min(\text{dist}_C(\mathbf{G}_1^A, \mathbf{G}_2^A), \text{dist}_C(\mathbf{G}_1^A, \mathbf{G}_2^B)) \quad (6.104)$$

or, alternatively, as

$$d_{\min} = \min(\text{dist}_C(\mathbf{G}_1^A, \mathbf{G}_2^A), \text{dist}_C(\mathbf{G}_1^B, \mathbf{G}_2^A)). \quad (6.105)$$

Note that, in any case, the resulting distance d_{\min} will be small only if the two shapes V_1, V_2 are really similar. This also means that we only need to store *one* of the two normalized Fourier descriptors—for example, $\mathbf{G}_{\text{ref}}^A$ —for each reference shape V_{ref} and then (following Eqn. (6.104)) compare it to *both* normalized descriptors $\mathbf{G}_{\text{new}}^A$ and $\mathbf{G}_{\text{new}}^B$ of any new shape V_{new} .²¹

To illustrate this idea, [Alg. 6.8](#) shows the construction of a simple Fourier descriptor database from a set of reference shapes and its subsequent use for classifying unknown shapes. First, procedure `MAKEFDDATABASE(V)` returns a map D holding a normalized Fourier descriptor for each of the reference shapes given in V . Matching a new shape V_{new} to the entries in the database D is accomplished by procedure `FINDBESTMATCH(V_{new} , D, d_{\max})`, which returns the index of the best-fitting shape in D , or `nil` if the distance of the closest match exceeds the predefined threshold d_{\max} . As common in this situation, we use *squared* distance values (i.e., dist_C^2) for matching in [Alg. 6.8](#) (lines 15–18), thereby avoiding the square root operations in Eqns. (6.101) and (6.103).

6.7 Java implementation

The algorithms described in this chapter have been implemented in a matching Java API, with is available at the book's accompanying website. As usual, most Java methods are named and structured identically to the procedures defined in the various algorithms for easy identification. The relevant Java classes are contained in package `imagingbook.fd`.

`FourierDescriptor` (class)

This is the main class in this package; it holds all data structures and implements the functionality common to all Fourier descriptors, including methods for shape reconstruction, invariance, and matching (see below).

²¹ The justification for keeping only *one* of the two normalized descriptors $\mathbf{G}_{\text{ref}}^A$, $\mathbf{G}_{\text{ref}}^B$ of each reference shape V_{ref} is that if two candidate shapes V_1, V_2 are similar, then the normalization will produce pairs of Fourier descriptors $(\mathbf{G}_1^A, \mathbf{G}_1^B)$ and $(\mathbf{G}_2^A, \mathbf{G}_2^B)$ that are also similar but not necessarily in the same order. Therefore, \mathbf{G}_1^A must only match with *either* \mathbf{G}_2^A or \mathbf{G}_2^B to detect the similarity of V_1 and V_2 .

Algorithm 6.8 Simple shape matching with a database of Fourier descriptors. MAKEFDDATABASE(V_{ref}, M') creates and returns a new database (map) R from a sequence of reference shapes V_{ref} . R can then be passed to FINDBESTMATCH($V_{\text{new}}, M', R, d_{\max}$) for classifying a new shape V_{new} , where d_{\max} is a predefined distance threshold.

```

1: MAKEFDDATABASE( $V_{\text{ref}}, M'$ )
   Input:  $V_{\text{ref}} = (V_0, V_1, \dots, V_{N_R})$ , a sequence of reference shapes;  $M'$ ,  

          the number of Fourier coefficients. Returns a sequence of model  

          Fourier descriptors for the reference shapes in  $V_{\text{ref}}$ .
2:  $N_R \leftarrow |V_{\text{ref}}|$ 
3:  $R \leftarrow$  new map of Fourier descriptors over  $[0, N_R - 1]$ 
4: for  $i \leftarrow 0, \dots, N_R - 1$  do
5:    $\mathbf{G} \leftarrow \text{FOURIERDESCRIPTORUNIFORM}(V_{\text{ref}}(i), M')$             $\triangleright$  Alg. 6.3
6:    $(\mathbf{G}^A, \mathbf{G}^B) \leftarrow \text{MAKEINVARIANT}(\mathbf{G})$                     $\triangleright$  Alg. 6.6
7:    $R(i) \leftarrow \mathbf{G}^A$             $\triangleright$  store only one normalized descriptor ( $\mathbf{G}^A$ )
8: return  $R$ .
```

```

9: FINDBESTMATCH( $V_{\text{new}}, M', R, d_{\max}$ )
   Input:  $V_{\text{new}}$ , a new shape;  $M'$ , the number of Fourier coefficients;  $R$ ,  

          a sequence of reference Fourier descriptors;  $d_{\max}$ , maximum squared  

          distance acceptable for a positive match. Returns the best-matching  

          shape index  $i_{\min}$  or nil if no acceptable match was found.
10:  $\mathbf{G}_{\text{new}} \leftarrow \text{FOURIERDESCRIPTORUNIFORM}(V_{\text{new}}, M')$             $\triangleright$  Alg. 6.3
11:  $(\mathbf{G}_{\text{new}}^A, \mathbf{G}_{\text{new}}^B) \leftarrow \text{MAKEINVARIANT}(\mathbf{G}_{\text{new}})$             $\triangleright$  Alg. 6.6
12:  $d_{\min} \leftarrow \infty, i_{\min} \leftarrow -1$ 
13: for  $i \leftarrow 0, \dots, |R| - 1$  do
14:    $\mathbf{G}_{\text{ref}}^A \leftarrow R(i)$ 
15:    $d_2 \leftarrow \min(D2(\mathbf{G}_{\text{new}}^A, \mathbf{G}_{\text{ref}}^A), D2(\mathbf{G}_{\text{new}}^B, \mathbf{G}_{\text{ref}}^A))$             $\triangleright$  Eqn. (6.105)
16:   if  $d_2 < d_{\min}$  then
17:      $d_{\min} \leftarrow d_2$ 
18:      $i_{\min} \leftarrow i$ 
19:   if  $d_{\min} \leq d_{\max}$  then
20:     return  $i_{\min}$             $\triangleright$  best match index is  $i_{\min}$ 
21:   else
22:     return nil.            $\triangleright$  no matching shape found in  $R$ 
```

```

23: D2( $\mathbf{G}_1, \mathbf{G}_2$ )
   Returns the squared complex distance  $\text{dist}_C^2(\mathbf{G}_1, \mathbf{G}_2)$  between the  

   Fourier descriptors  $\mathbf{G}_1, \mathbf{G}_2$  (see Eqn. (6.103)).
24:  $d \leftarrow 0, M_p \leftarrow (\min(|\mathbf{G}_1|, |\mathbf{G}_2|) - 1) \div 2$ 
25: for  $m \leftarrow -M_p, \dots, M_p, m \neq 0$  do
26:    $d \leftarrow d + [\text{Re}(\mathbf{G}_1(m)) - \text{Re}(\mathbf{G}_2(m))]^2 + [\text{Im}(\mathbf{G}_1(m)) - \text{Im}(\mathbf{G}_2(m))]^2$ 
27: return  $d$ .            $\triangleright d \equiv (\text{dist}_C(\mathbf{G}_1, \mathbf{G}_2))^2$ 
```

`FourierDescriptor` is an abstract class and thus cannot be instantiated. To create Fourier descriptor objects, one of the concrete sub-classes `FourierDescriptorUniform` or `FourierDescriptorFromPolygon` (see page 223) may be used, which provide the appropriate constructors. `FourierDescriptor` provides the following methods for both types of Fourier descriptors:

Access to Fourier coefficients

`Complex[] getCoefficients ()`

Returns the complete vector of complex-valued Fourier coefficients.²²

`Complex getCoefficient (int m)`

Returns the value of the Fourier coefficient $\mathbf{G}(m \bmod M)$, with $M = |\mathbf{G}|$ as above.

`Complex setCoefficient (int m, Complex z)`

Replaces the Fourier coefficient $\mathbf{G}(m \bmod M)$ by the complex value z , with $M = |\mathbf{G}|$ as above.

`Complex setCoefficient (int m, double a, double b)`

Replaces the Fourier coefficient $\mathbf{G}(m \bmod M)$ by the complex value $z = a + i \cdot b$, with $M = |\mathbf{G}|$ as above.

`int size ()`

Returns the length (M) of the Fourier descriptor.

`int getMaxNegHarmonic ()`

Returns the max. negative harmonic $m = -(M - 1) \div 2$ for this Fourier descriptor (of length M).

`int getMaxPosHarmonic ()`

Returns the max. positive harmonic $m = M \div 2$ for this Fourier descriptor (of length M).

`int getMaxCoefficientPairs ()`

Returns the max. number of coefficient pairs, $(M - 1) \div 2$, for this Fourier descriptor (of length M).

`void truncate (int P)`

Truncates this Fourier descriptor to the P lowest-frequency coefficients (see Eqn. (6.19)).

Comparing Fourier descriptors

`double distanceComplex (FourierDescriptor fd2)`

Returns the complex-valued distance ($\text{dist}_C(\mathbf{G}_1, \mathbf{G}_2)$, see Eqn. (6.103)) between *this* Fourier descriptor (\mathbf{G}_1) and another Fourier descriptor `fd2` (\mathbf{G}_2). The zero-coefficients are ignored.

²² The class `Complex` is defined in the `imagingbook.math` package.

`double distanceComplex (FourierDescriptor fd2, int Mp)`

As above, but using only M_p coefficient pairs (see Eqn. (6.103)).

`double distanceMagnitude (FourierDescriptor fd2)`

Returns the magnitude-only distance ($\text{dist}_M(\mathbf{G}_1, \mathbf{G}_2)$, see Eqn. (6.101)) between *this* Fourier descriptor (\mathbf{G}_1) and another Fourier descriptor `fd2` (\mathbf{G}_2). The zero-coefficients are ignored.

`double distanceMagnitude (FourierDescriptor fd2, int Mp)`

As above, but using only M_p coefficient pairs (see Eqn. (6.101)).

Shape reconstruction

`Complex[] getReconstruction (int N)`

Returns the shape reconstructed from the complete Fourier descriptor as a sequence of N complex-valued contour points. The contour points are obtained by evaluating `getReconstructionPoint(t)` at uniformly spaced positions $t \in [0, 1]$.

`Complex[] getReconstruction (int N, int Mp)`

Returns a partial shape reconstruction from M_p Fourier coefficient pairs as a sequence of N complex-valued contour points.

`Complex getReconstructionPoint (double t)`

Returns a single point (as a complex value) on the continuous contour for path parameter $t \in [0, 1]$, reconstructed from the complete Fourier descriptor (see Eqn. (6.16)).

`Complex getReconstructionPoint (double t, int Mp)`

Returns a single point (as a complex value) on the continuous contour for path parameter $t \in [0, 1]$, reconstructed from M_p Fourier coefficient pairs.

Normalization

`FourierDescriptor[] makeInvariant ()`

Returns a pair of Fourier descriptors ($\mathbf{G}^A, \mathbf{G}^B$) that are normalized for scale, start point shift and shape rotation (see [Alg. 6.6](#)).

`double makeRotationInvariant ()`

Normalizes the Fourier descriptor for shape rotation by phase-shifting all coefficients (see [Alg. 6.6](#)). Returns the estimated rotation angle β .

`double makeScaleInvariant ()`

Normalizes the Fourier descriptor for scale by multiplying with a common factor, such that the L_2 norm of the resulting vector is 1. Returns the scale factor that was applied for normalization.

```
FourierDescriptor[] makeStartPointInvariant ()
```

Returns a pair of normalized Fourier descriptors ($\mathbf{G}^A, \mathbf{G}^B$), one for each start point normalization angles φ_A and $\varphi_B = \varphi_A + \pi$, respectively (see [Alg. 6.7](#)).

```
void makeTranslationInvariant ()
```

Modifies this Fourier descriptor by setting the zero-coefficient $\mathbf{G}(0)$ to zero. This method is rarely needed because coefficient $\mathbf{G}(0)$ is ignored for matching.

FourierDescriptorUniform (class)

This sub-class of `FourierDescriptor` represents Fourier descriptors obtained from uniformly sampled contours, as described in [Alg. 6.2](#). It only provides the constructor methods

```
FourierDescriptorUniform (Point2D[] V),
```

```
FourierDescriptorUniform (Point2D[] V, int Mp),
```

where V is a sequence of M contour points (`Point2D`), assumed to be uniformly sampled. The first constructor creates a full Fourier descriptor with M coefficients (see [Alg. 6.2](#)). The second constructor creates a Fourier descriptor with M_p coefficient pairs (i.e., $2 \cdot M_p + 1$ coefficients), as described in [Alg. 6.3](#)

FourierDescriptorFromPolygon (class)

This sub-class of `FourierDescriptor` represents Fourier descriptors obtained directly from polygons (without contour sampling, see [Alg. 6.5](#)). It only provides the constructor method

```
FourierDescriptorFromPolygon (Point2D[] V, int Mp),
```

where V is a sequence of polygon vertices and M_p specifies the number of Fourier coefficient pairs.

PolygonSampler (Class)

Instances of this class can be used to produce uniformly sampled polygons.

```
Point2D[] samplePolygonUniformly (Point2D[] V, int M)
```

Samples the closed polygon path specified by the vertices in V at M equidistant positions and returns the resulting point sequence (see [Alg. 6.1](#)).

Example

The code example in Prog. 6.1 demonstrates the use of the Fourier descriptor API. It assumes that the binary input image (`ip`) contains at least one connected foreground region. Region labeling and contour extraction

```

1 import imagingbook.fd.*;
2 import imagingbook.regions.*;
3 import imagingbook.contours.*;
4 import imagingbook.math.Complex;
5
6 ByteProcessor ip ...; // assumed to contain a binary image
7
8 // segment ip and select the longest outer region contour:
9 RegionContourLabeling labeling = new RegionContourLabeling(ip);
10 List<Contour> outerContours = labeling.getAllOuterContours(true);
11 Contour contr = outerContours.get(0); // select the longest contour
12 Point2D[] V = contr.getPointArray();
13
14 // create the Fourier descriptor for V with 15 coefficient pairs:
15 FourierDescriptor fd = new FourierDescriptorUniform(V, 15);
16
17 // reconstruct the corresponding shape with 100 contour points:
18 Complex[] R = fd.getReconstruction(100);
19
20 // create a pair of invariant descriptors ( $G^A, G^B$ ):
21 FourierDescriptor[] fdAB = fd.makeInvariant();
22 FourierDescriptor fdA = fdAB[0]; // =  $G^A$ 
23 FourierDescriptor fdb = fdAB[1]; // =  $G^B$ 
24 ...

```

Program 6.1 Fourier descriptor code example. The input image `ip` is assumed to contain a binary image (line 6). The class `RegionContourLabeling` is used to find connected regions (line 9). Then the list of outer contours is retrieved (line 10) and the longest contour is assigned to `V` as an array of type `Point2D` (line 11–12). In line 15, the contour `V` is used to create a Fourier descriptor with 15 coefficient pairs. Alternatively, we could have created a Fourier descriptor of the same length (number of coefficients) as the contour and then truncated it (using the `truncate()` method) to the specified number of coefficient pairs. A partial reconstruction of the contour (with 100 sample points) is calculated from the Fourier descriptor `fd` in line 18. Finally, a pair of invariant descriptors (contained in the array `fdAB`) is calculated in line 21.

is applied first, using methods provided by the `imagingbook.regions` and `imagingbook.contours` packages.²³ Subsequently, the longest region contour (`C`) is used to create a Fourier descriptor (`fd`) with $M_P = 15$ coefficient pairs. A partial reconstruction is calculated from the original Fourier descriptor with 100 sample points along the contour. The last lines show how a pair of invariant descriptors (G^A, G^B) is obtained by applying the `makeInvariant()` method. Note that the code fragment in Prog. 6.1 is not complete but would typically be part of the `run()` method in an ImageJ plugin. The full version and additional code examples can be found on the book's website.

²³ See Vol. 2, Ch. 2, “Regions in Binary Images” [21].

6.8 Summary and further reading

The use of Fourier descriptors for shape description and matching dates back to the early 1960's [31, 42], advanced by the work of Zahn and Roskies [151], Granlund [49], Richard and Hemami [111], and Persoon and Fu [102, 103] in the 1970s, particularly in the context of character recognition and aircraft identification. Making Fourier descriptors invariant against various geometric transformations was a key issue from the very beginning, and several relevant contributions were published in the 1980s, including [139], [32] [70], and [108]. Unfortunately, as illustrated in this chapter, to achieve robust invariance and uniqueness of representation in practice is not as easy as sometimes suggested in the literature, despite the simplicity and elegance of the underlying theory. In practice, normalization for descriptor invariance is quite difficult for arbitrary shapes because of possibly vanishing Fourier coefficients and the resulting sensitivity to noise.

Fourier descriptors have nevertheless become popular in a wide range of applications, including geology and, in particular, biological imaging, as documented by the work of Lestrel and others in [73]. Fourier descriptors have been extended to accommodate affine transformations and applied to 3D object identification [3] and stereo matching [146].

Although Fourier descriptors have been investigated to handle open contours and partial shapes [75], they are naturally best suited for dealing with closed contours, as described above. Of course, this is a limitation if shapes are only partially visible or occluded. The presentation in this chapter was limited to what is frequently called "elliptical" Fourier descriptors [49], since they are most popular and well known. Other types of Fourier descriptors have been proposed, which are not covered here but can be found elsewhere in the literature (see, for example [60, p. 534] and [93, Ch. 7]).

6.9 Exercises

Exercise 6.1

Verify that the DFT spectrum is periodic, i. e., that $\mathbf{G}(-m) = \mathbf{G}(M-m)$ holds for arbitrary $m \in \mathbb{Z}$ (as claimed in Eqn. (6.18)).

Exercise 6.2

[Algorithm 6.9](#) shows an alternative solution to uniform polygon sampling. Implement this algorithm and verify that it is equivalent to [Alg. 6.1](#) (implemented as method `samplePolygonUniformly()` in class `PolygonSampler`, see p. 223).

Exercise 6.3

Assume that the complete outer contour of a binary region is given as a

Algorithm 6.9 Uniform sampling of a polygon path (alternative to [Alg. 6.1](#), proposed by J. Heinzelreiter).

```

1: SAMPLEPOLYGONUNIFORMLY( $V, M$ )
   Input:  $V = (\mathbf{v}_0, \dots, \mathbf{v}_{N-1})$ , a sequence of  $N$  points representing the
         vertices of a closed 2D polygon;  $M$ , number of desired sample points.
   Returns a new sequence  $\mathbf{g} = (g_0, \dots, g_{M-1})$  of complex values repre-
         senting sample points sampled uniformly along the path of the input
         polygon  $V$ .
2:  $N \leftarrow |V|$ 
3:  $\Delta \leftarrow \frac{1}{M} \cdot \text{PATHLENGTH}(V)$             $\triangleright$  segment length  $\Delta$ , see Alg. 6.1
4: Create map  $\mathbf{g}: [0, M-1] \rightarrow \mathbb{C}$             $\triangleright$  complex point sequence  $\mathbf{g}$ 
5:  $\mathbf{g}(0) \leftarrow \text{COMPLEX}(V(0))$ 
6:  $i \leftarrow 0$                                       $\triangleright$  index of path segment  $\langle V_i, V_{i+1} \rangle$ 
7:  $k \leftarrow 1$                                       $\triangleright$  index of first unassigned point in  $\mathbf{g}$ 
8:  $d_p \leftarrow 0$                                       $\triangleright$  path distance between  $V(i)$  and  $V(k-1)$ 
9: while ( $i < N$ )  $\wedge$  ( $k < M$ ) do
10:    $\mathbf{v}_A \leftarrow V(i)$ 
11:    $\mathbf{v}_B \leftarrow V((i + 1) \bmod N)$ 
12:    $\delta \leftarrow \|\mathbf{v}_B - \mathbf{v}_A\|$                     $\triangleright$  Euclidean distance
13:   if ( $\Delta - d_p \leq \delta$ ) then
14:      $\mathbf{x} \leftarrow \mathbf{v}_A + \frac{\Delta - d_p}{\delta} \cdot (\mathbf{v}_B - \mathbf{v}_A)$        $\triangleright \mathbf{x}_k$  by lin. interpolation
15:      $\mathbf{g}(k) \leftarrow \text{COMPLEX}(\mathbf{x})$ 
16:      $d_p \leftarrow d_p - \Delta$ 
17:      $k \leftarrow k + 1$ 
18:   else
19:      $d_p \leftarrow d_p + \delta$ 
20:      $i \leftarrow i + 1$ 
21: return  $\mathbf{g}.$ 
```

sequence of P boundary pixels with coordinates $V = (\mathbf{p}_0, \dots, \mathbf{p}_{P-1})$. To produce a Fourier descriptor of length $M < P$ there are several options:

1. Sample the original contour V at M uniformly-spaced positions (see [Alg. 6.1](#)) and then calculate the Fourier descriptor of length M using [Alg. 6.2](#).
2. Calculate a partial Fourier descriptor of length M from the original contour V using [Alg. 6.3](#).
3. Calculate the full Fourier descriptor (of length P) from the original

contour V (using [Alg. 6.2](#)) and subsequently truncate²⁴ the Fourier descriptor to length M , as described in Eqns. (6.19–6.20).

4. Treat the original boundary coordinates V as the vertices of a closed polygon and calculate a Fourier descriptor with $M_P = M \div 2$ coefficient pairs, using the trigonometric method described in [Alg. 6.5](#).

Compare these approaches and discuss their individual merits or disadvantages in terms of efficiency and accuracy.

Exercise 6.4

Test the Fourier descriptor normalization described in [Algs. 6.6–6.7](#) (implemented by method `makeInvariant()` in the Java API) for changes in scale, start point shift and shape rotation on a suitable set of binary shapes (e.g., images from the KIMIA dataset [65]). See the examples for shape rotation and (implicit) start point shifts in [Fig. 6.22](#). How reliably do the normalized Fourier descriptors of the modified shapes match to their corresponding originals?

Exercise 6.5

Magnitude-only matching (see Sec. 6.6.1) is much simpler than complex-valued matching (see Sec. 6.6.2) of Fourier descriptors, since no normalization for phase (start point shift and shape rotation) is required. However, it can be assumed that different shapes are more likely to be confused if the phase information is ignored. Test this hypothesis on a large number and variety of different shapes. Compare the confusion probability for magnitude-only vs. complex-valued matching.

²⁴ See method `truncate(int P)` on p. 221 for a Java implementation.

SIFT—Scale-Invariant Local Features

Many real applications require the localization of reference positions in one or more images, for example, for image alignment, removing distortions, object tracking, 3D reconstruction etc. We have seen that corner points¹ can be located quite reliably and independent to orientation. However, typical corner detectors only provide the position and strength of each candidate point but do not provide any information about its characteristic or “identity” that could be used for matching. Another limitation is that most corner detectors only operate at a particular scale or resolution, since they are based on a rigid set of filters.

This chapter describes the SIFT technique for local feature detection, which was originally proposed by David Lowe [79] and has since become a “workhorse” method in the imaging industry. Its goal is to locate image features that can be identified robustly to facilitate matching in multiple images and image sequences as well as object recognition under different viewing conditions. SIFT employs the concept of “scale space” [78] to capture features at *multiple* scale levels or image resolutions, which not only increases the number of available features but also makes the method highly tolerant to scale changes. This makes it possible, for example, to track features on objects that move towards the camera and thereby change their scale continuously or to stitch together images taken with widely different zoom settings.

Many implementations of the original SIFT technique and modified versions exist, including those described in [54, 136] and libraries included in popular software such as *OpenCV*, *AutoPano* and *Hugin*. Accelerated variants of the

¹ See Vol. 2 [21, Ch. 4].

SIFT algorithm have been implemented by streamlining the scale space calculation and feature detection or the use of GPU hardware [10, 48, 120].²

In principle, SIFT works like a multi-scale corner detector with sub-pixel positioning accuracy and a rotation-invariant feature descriptor attached to each candidate point. This (typically 128-dimensional) feature descriptor summarizes the distribution of the gradient directions in a spatial neighborhood around the corresponding feature point and can thus be used like a “fingerprint”. The main steps involved in the calculation of SIFT features are:

1. Extrema detection in a Laplacian-of-Gaussian (LoG) scale space to locate potential interest points.
2. Key point refinement by fitting a continuous model to determine precise location and scale.
3. Orientation assignment by the dominant orientation of the feature point from the directions of the surrounding image gradients.
4. Formation of the feature descriptor by normalizing the local gradient histogram.

These steps are all described in the remaining parts of this chapter. There are several reasons why we explain the SIFT technique here in such great detail. For one, it is by far the most complex algorithm that we have looked at so far, its individual steps are carefully designed and delicately interdependent, with numerous parameters that need to be considered. A good understanding of the inner workings and limitations is thus important for a successful use as well as for analyzing problems if the results are not as expected.

7.1 Interest points at multiple scales

The first step in detecting interest points is to find locations with stable features that can be localized under a wide range of viewing conditions and different scales. In the SIFT approach, interest point detection is based on Laplacian-of-Gaussian (LoG) filters, which respond primarily to distinct bright blobs surrounded by darker regions, or vice versa. Unlike the filters used in popular corner detectors,³ LoG filters are *isotropic*, i.e., insensitive to orientation. To locate interest points over multiple scales, a scale space representation of the input image is constructed by recursively smoothing the image with a sequence of small Gaussian filters. The difference between the images in adjacent scale

² Note that the SIFT technique has been patented [81] and only a binary executable has been made available with the original publication (see <http://www.cs.ubc.ca/~lowe/keypoints/>).

³ See Ch. 4 of Volume 2 [21].

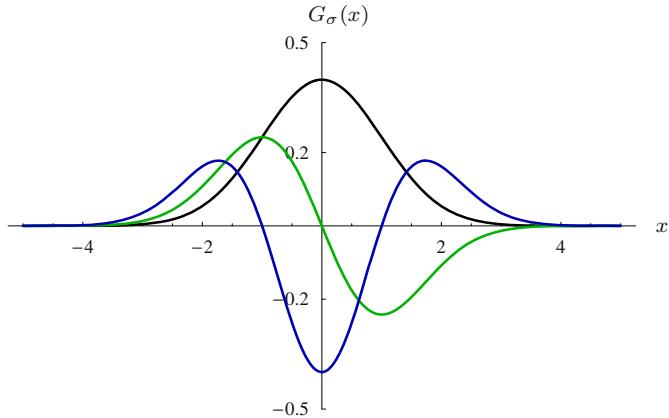


Figure 7.1 One-dimensional Gaussian function $G_\sigma(x)$ with $\sigma = 1$ (black), its first derivative $G'_\sigma(x)$ (green) and second derivative $G''_\sigma(x)$ (blue).

layers is used to approximate the LoG filter at each scale. Interest points are finally selected by finding the local maxima in the three-dimensional LoG scale space.

In this section, we first outline LoG filters and the basic construction of a Gaussian scale space, followed by a detailed description of the actual implementation and the parameters used in the SIFT approach.

7.1.1 The Laplacian-of-Gaussian (LoG) filter

The LoG is a so-called *center-surround* operator, which most strongly responds to isolated local intensity peaks, edge, and corner-like image structures. The corresponding filter kernel is based on the second derivative of the Gaussian function, as illustrated in Fig. 7.1 for the one-dimensional case. The one-dimensional Gaussian function of width σ is defined as

$$G_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{-\frac{x^2}{2\sigma^2}}, \quad (7.1)$$

and its *first* derivative is

$$G'_\sigma(x) = \frac{dG}{dx}(x) = -\frac{x}{\sqrt{2\pi}\sigma^3} \cdot e^{-\frac{x^2}{2\sigma^2}}. \quad (7.2)$$

Analogously, the *second* derivative of the 1D-Gaussian is

$$G''_\sigma(x) = \frac{d^2G}{dx^2}(x) = \frac{x^2 - \sigma^2}{\sqrt{2\pi}\sigma^5} \cdot e^{-\frac{x^2}{2\sigma^2}}. \quad (7.3)$$

The *Laplacian* (denoted ∇^2) of a continuous, two-dimensional function $f(x, y)$ is defined as the sum of the second partial derivatives for the x - and y -directions, traditionally written as

$$(\nabla^2 f)(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}. \quad (7.4)$$

Note that, unlike the *gradient*⁴ of a 2D function, the result of the Laplacian is not a vector but a *scalar* quantity. Its value is invariant against rotations of the coordinate system, i.e., the Laplacian operator has the important property of being *isotropic*.

By applying the *Laplacian* operator to a rotationally symmetric 2D Gaussian,

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}, \quad (7.5)$$

with identical widths $\sigma = \sigma_x = \sigma_y$ in the x/y directions (see Fig. 7.2(a)), we obtain the *Laplacian-of-Gaussian* or “LoG” function

$$\begin{aligned} L_\sigma(x, y) &= (\nabla^2 G_\sigma)(x, y) = \frac{\partial^2 G_\sigma(x, y)}{\partial x^2} + \frac{\partial^2 G_\sigma(x, y)}{\partial y^2} \\ &= \frac{(x^2 - \sigma^2)}{2\pi\sigma^6} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} + \frac{(y^2 - \sigma^2)}{2\pi\sigma^6} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \\ &= \frac{1}{\pi\sigma^4} \cdot \left(\frac{x^2 + y^2 - 2\sigma^2}{2\sigma^2} \right) \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}. \end{aligned} \quad (7.6)$$

The continuous LoG function in Eqn. (7.6) has the absolute value integral

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |L_\sigma(x, y)| \, dx \, dy = \frac{4}{\sigma^2 e} \quad (7.7)$$

and zero average, that is,

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} L_\sigma(x, y) \, dx \, dy = 0. \quad (7.8)$$

When used as the kernel of a linear filter,⁵ the LoG responds maximally to circular spots that are *darker* than the surrounding background and have a radius of approximately σ . Blobs that are *brighter* than the surrounding background are enhanced by filtering with the negative LoG kernel, i.e., $-L_\sigma$, which is often referred to as the “Mexican hat” filter (see Fig. 7.2).⁶ Both types of blobs can be detected simultaneously by simply taking the absolute value of the filter response (see Fig. 7.3).

⁴ See Vol. 1 [20, Sec. 6.2.1].

⁵ To produce a sufficiently accurate discrete LoG filter kernel, the support radius

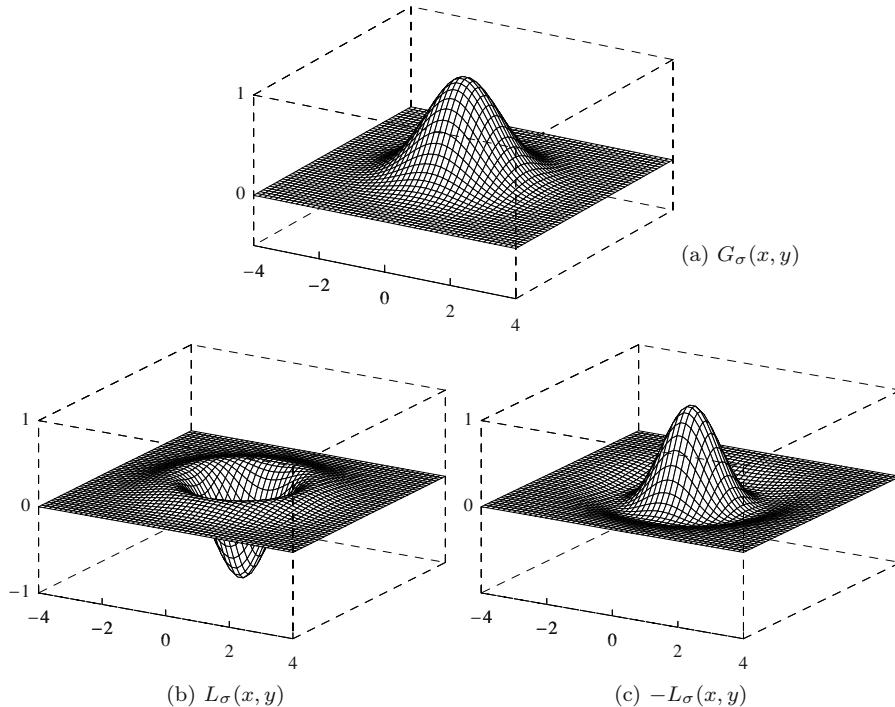


Figure 7.2 Two-dimensional Gaussian and Laplacian-of-Gaussian (LoG). Gaussian function $G_\sigma(x, y)$ with $\sigma = 1$ (a); the corresponding LoG function $L_\sigma(x, y)$ in (b), and the inverted function (“Mexican hat” or “Sombrero” kernel) $-L_\sigma(x, y)$ in (c). For illustration, all three functions are normalized to an absolute value of 1 at the origin.

Since the LoG function is based on derivatives, its magnitude strongly depends on the steepness of the Gaussian slope which is controlled by σ . To obtain responses of comparable magnitude over multiple scales, a *scale normalized* LoG kernel [78] can be defined in the form

$$\begin{aligned}\hat{L}_\sigma(x, y) &= \sigma^2 \cdot (\nabla^2 G_\sigma)(x, y) = \sigma^2 \cdot L_\sigma(x, y) \\ &= \frac{1}{\pi\sigma^2} \cdot \left(\frac{x^2 + y^2 - 2\sigma^2}{2\sigma^2} \right) \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}.\end{aligned}\quad (7.9)$$

should be set to at least 4σ (kernel diameter $\geq 8\sigma$).

⁶The LoG is often used as a model for early processes in biological vision systems [85], particularly to describe the center-surround response of receptive fields. In this model, an “on-center” cell is *stimulated* when the center of its receptive field is exposed to light, and is *inhibited* when light falls on its surround. Conversely, an “off-center” cell is stimulated by light falling on its surround. Thus filtering with the original LoG L_σ (Eqn. (7.6)) corresponds to the behavior of off-center cells, while the response to the negative LoG kernel $-L_\sigma$ is that of an on-center cell.

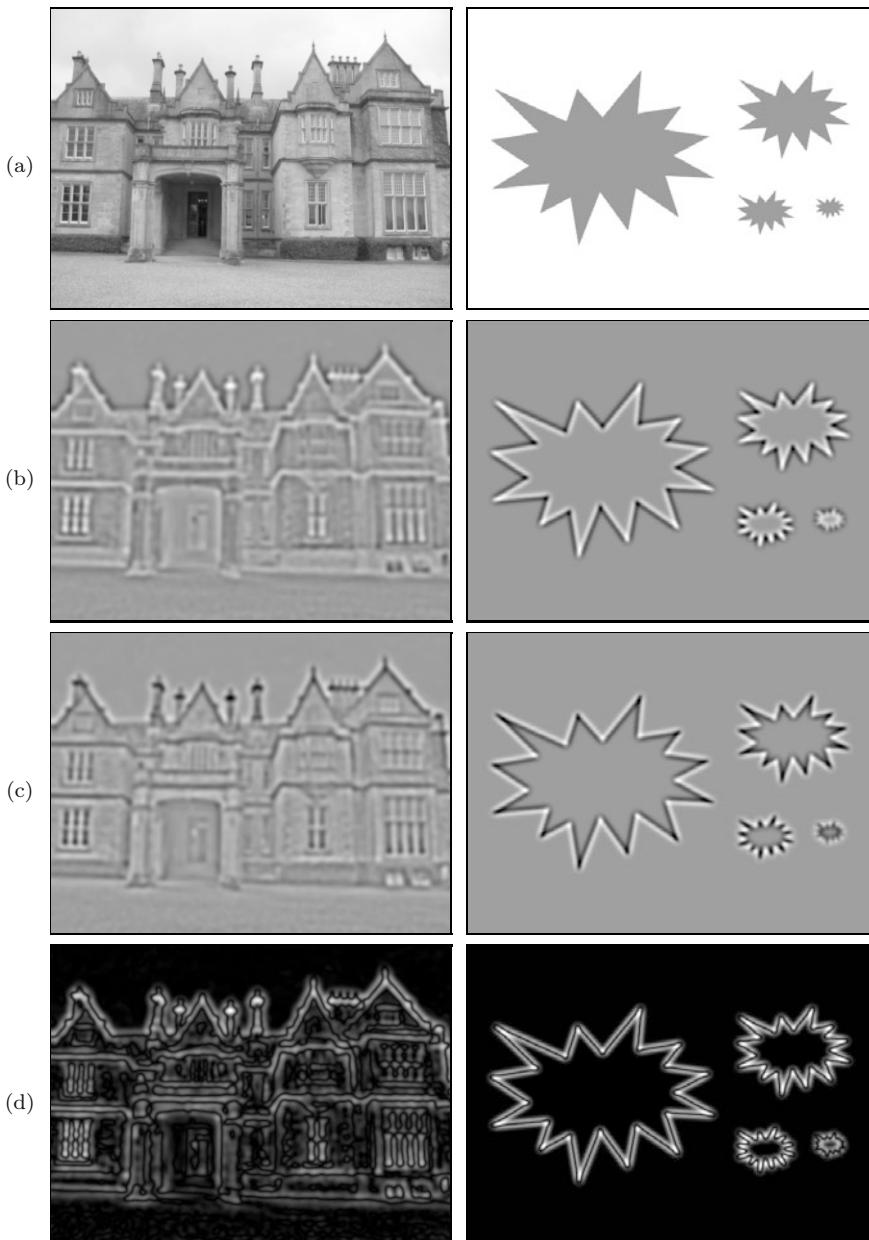


Figure 7.3 Filtering with the LoG kernel (with $\sigma = 3$). Original images (a). A linear filter with the LoG kernel $L_\sigma(x, y)$ responds strongest to dark spots in a bright surround (b), while the inverted kernel $-L_\sigma(x, y)$ responds strongest to bright spots in a dark surround (c). In (b, c), zero values are shown as medium gray, negative values are dark, positive values are bright. The absolute value of either result (d) combines the responses from both dark and bright spots.

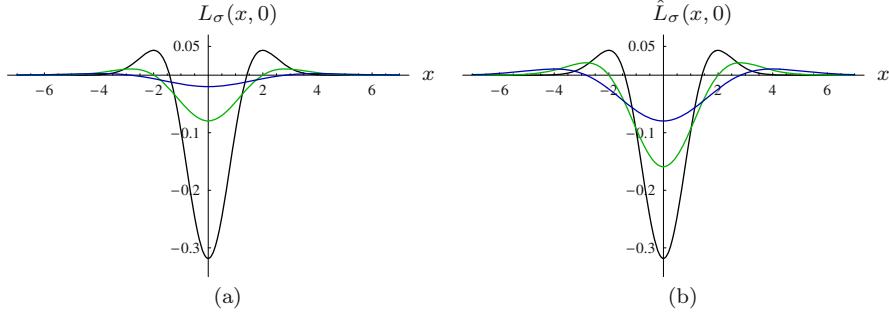


Figure 7.4 Unnormalized LoG vs. scale-normalized LoG function for different values of σ . Cross section of LoG function $L_\sigma(x, y)$ as defined in Eqn. (7.6) (a), scale-normalized LoG (b) as defined in Eqn. (7.9). $\sigma = 1.0$ (black), $\sigma = \sqrt{2}$ (green), $\sigma = 2.0$ (blue). All three functions in (b) have the same absolute value integral that is independent of σ (see Eqn. (7.10)).

Note that the integral of this function,

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |\hat{L}_\sigma(x, y)| dx dy = \frac{4}{e}, \quad (7.10)$$

is constant and thus independent of the scale variable σ (see Fig. 7.4).

Approximating the LoG by the difference of two Gaussians (DoG)

Although the LoG is “quasi-separable” [56, 138] and can thus be calculated efficiently, the most common method for implementing the LoG filter is to approximate it by the *difference of two Gaussians* (DoG) of widths σ and $\kappa\sigma$, respectively, that is,

$$D_{\sigma, \kappa}(x, y) = G_{\kappa\sigma}(x, y) - G_\sigma(x, y), \quad (7.11)$$

with the parameter $\kappa > 1$ specifying the relative width of the two Gaussians (defined in Eqn. (7.5)). Properly scaled (by some factor λ , see below), the DoG function $D_{\sigma, \kappa}(x, y)$ approximates the LoG function $L_\sigma(x, y)$ in Eqn. (7.6) with arbitrary precision, i.e.,

$$L_\sigma(x, y) \approx \lambda \cdot D_{\sigma, \kappa}(x, y), \quad (7.12)$$

as κ approaches 1 ($\kappa = 1$ being excluded, of course). In practice, values of κ in the range $1.1, \dots, 1.3$ yield sufficiently accurate results. As an example, Fig. 7.5 shows the cross-section of the two-dimensional DoG function for $\kappa = 2^{1/3} \approx 1.25992$.⁷ The factor $\lambda \in \mathbb{R}$ in Eqn. (7.12) controls the magnitude of

⁷ The factor $\kappa = 2^{1/3}$ originates from splitting the scale interval 2 (i.e., one scale octave) into 3 equal intervals, as described later on. Another factor mentioned frequently in the literature is 1.6 which, however, does not yield a satisfactory approximation. Possibly that value refers to the squared ratio κ^2 , i.e., the ratio of the variances σ_2^2/σ_1^2 and not the ratio of the standard deviations σ_2/σ_1 .

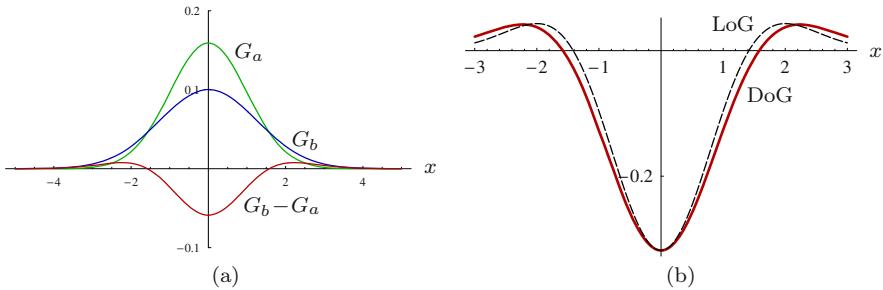


Figure 7.5 Approximating the LoG by the difference of two Gaussians (DoG). The two original Gaussians, $G_a(x)$ with $\sigma_a = 1.0$ and $G_b(x)$ with $\sigma_b = \sigma_a \cdot \kappa = \kappa$, shown by the green and blue curves, respectively (a). The red curve shows $\text{DoG}_{\sigma,\kappa}(x, y) = G_b(x) - G_a(x)$. In (b), the dashed line shows the reference LoG function $L_\sigma(x)$ in comparison to the DoG (red). The DoG is scaled to match the magnitude of the LoG function.

the DoG function; it depends on both the ratio κ and the scale parameter σ . To match the magnitude of the original LoG (Eqn. (7.6)) at the origin, it must be set to

$$\lambda = \frac{2\kappa^2}{\sigma^2 \cdot (\kappa^2 - 1)} \quad (7.13)$$

(see Appendix D.3 for details). Similarly, the *scale-normalized* LoG \hat{L}_σ (Eqn. (7.9)) can be approximated by the DoG $D_{\sigma,\kappa}$ as

$$\hat{L}_\sigma(x, y) = \sigma^2 L_\sigma(x, y) \approx \underbrace{\sigma^2 \cdot \lambda}_{\hat{\lambda}} \cdot D_{\sigma,\kappa}(x, y) = \frac{2\kappa^2}{\kappa^2 - 1} \cdot D_{\sigma,\kappa}(x, y), \quad (7.14)$$

with the factor $\hat{\lambda} = \sigma^2 \cdot \lambda = \frac{2\kappa^2}{\kappa^2 - 1}$ being constant and therefore independent of the scale σ . Thus, as pointed out in [80], with a fixed scale increment κ , the DoG already approximates the scale-normalized LoG up to a constant factor, and thus no additional scaling is required to compare the magnitudes of the DoG responses obtained at different scales.

In the SIFT approach, the DoG is used as an approximation of the (scale-normalized) LoG filter at multiple scales, based on a Gaussian scale space representation of the input image that is described next.⁸

⁸ See [Table 7.3](#) on p. 263 for a summary of the most important scale space-related symbols used in this chapter.

7.1.2 Gaussian scale space

The concept of scale space is motivated by the observation that real-world scenes exhibit relevant image features over a large range of sizes and, depending on the particular viewing situation, at various different scales. To handle structures at different and unknown sizes, it is useful to represent the image data over a large range of scales.

Continuous Gaussian scale space

The scale space representation of an image is obtained by filtering the image with a kernel that is parameterized to the desired scale (see [77] for an extensive treatment of scale space theory). Because of its unique properties [7, 40], the most common type of scale space is based on successive filtering with Gaussian kernels. Conceptually, given a continuous, two dimensional function $F(x, y)$, its Gaussian scale space representation is a three-dimensional function

$$\mathcal{G}(x, y, \sigma) = (F * H^{G, \sigma})(x, y), \quad (7.15)$$

where $H^{G, \sigma} \equiv G_\sigma(x, y)$ is a 2D Gaussian kernel (see Eqn. (7.5)) with unit integral, and $*$ denotes the linear convolution over x, y . Note that $\sigma \geq 0$ serves as both the continuous scale parameter and the width of the corresponding Gaussian filter kernel.

A fully continuous Gaussian scale space $\mathcal{G}(x, y, \sigma)$ covers a three-dimensional volume and represents the original function $F(x, y)$ at varying scales σ . For $\sigma = 0$, the Gaussian kernel $H^{G, 0}$ has zero width, which makes it equivalent to an impulse or Dirac function $\delta(x, y)$.⁹ This is the neutral element of linear convolution, that is,

$$\mathcal{G}(x, y, 0) = (F * H^{G, 0})(x, y) = (F * \delta)(x, y) = F(x, y) \quad (7.16)$$

and thus filtering with the Dirac function does not modify the input signal.

Discrete Gaussian scale space

In general (with $\sigma > 0$), the Gaussian kernel $H^{G, \sigma}$ acts as a low-pass filter with a cutoff frequency proportional to $1/\sigma$ (see Appendix D.2), the maximum frequency (or bandwidth) of the original “signal” $F(x, y)$ being potentially unlimited. This is different for a *discrete* input function $I(u, v)$, whose bandwidth is implicitly limited to half the sampling frequency, as mandated by the sampling theorem to avoid aliasing.¹⁰ Thus, in the discrete case, the lowest level $\mathcal{G}(x, y, 0)$ of the Gaussian scale space is not accessible! To model the implicit

⁹ See Vol. 1 [20, Sec. 5.3.4].

¹⁰ See Vol. 2 [21, Sec. 7.2.1].

bandwidth limitations of the sampling process, the discrete input image $I(u, v)$ is assumed to be pre-filtered (with respect to the underlying continuous signal) with a Gaussian kernel of width $\sigma_s \geq 0.5$ [80], that is,

$$\mathcal{G}(u, v, \sigma_s) \equiv I(u, v). \quad (7.17)$$

Thus the discrete input image $I(u, v)$ is implicitly placed at some initial level σ_s of the Gaussian scale space, and the lower levels with $\sigma < \sigma_s$ are not available.

Any higher level $\sigma_h > \sigma_s$ of the Gaussian scale space can be derived from the original image $I(u, v)$ by filtering with Gaussian kernel $H^{G, \hat{\sigma}}$, i.e.,

$$\mathcal{G}(u, v, \sigma_h) = (I * H^{G, \hat{\sigma}})(u, v), \quad \text{with } \hat{\sigma} = \sqrt{\sigma_h^2 - \sigma_s^2}. \quad (7.18)$$

This is due to the fact that applying two Gaussian filters of widths σ_1 and σ_2 , one after the other, is equivalent to a single convolution with a Gaussian kernel of width $\sigma_{1,2}$, that is,¹¹

$$(I * H^{G, \sigma_1}) * H^{G, \sigma_2} \equiv I * H^{G, \sigma_{1,2}}, \quad \text{with } \sigma_{1,2} = \sqrt{\sigma_1^2 + \sigma_2^2}. \quad (7.19)$$

We define the *discrete Gaussian scale space* representation of an image I as a vector of M images, one for each scale level m :

$$\mathbf{G} = (\mathbf{G}_0, \mathbf{G}_1, \dots, \mathbf{G}_{M-1}). \quad (7.20)$$

Associated with each level \mathbf{G}_m is its absolute scale $\sigma_m > 0$, and each level \mathbf{G}_m represents a blurred version of the original image, that is, $\mathbf{G}_m(u, v) \equiv \mathcal{G}(u, v, \sigma_m)$ in the notation introduced in Eqn. (7.15). The scale ratio between adjacent levels,

$$\Delta_\sigma = \frac{\sigma_{m+1}}{\sigma_m}, \quad (7.21)$$

is constant. Usually, Δ_σ is specified such that the absolute scale σ_m doubles with a given number of levels Q , called an *octave*. In this case, the resulting scale increment is $\Delta_\sigma = 2^{1/Q}$ with (typically) $Q = 3, \dots, 6$.

In addition, a *base scale* $\sigma_0 > \sigma_s$ is specified for the initial level \mathbf{G}_0 , with σ_s denoting the smoothing of the discrete image implied by the sampling process, as discussed above. Based on empirical results, a base scale of $\sigma_0 = 1.6$ is recommended in [80] to achieve reliable interest point detection. Given Q and the base scale σ_0 , the absolute scale at an arbitrary scale space level \mathbf{G}_m is

$$\sigma_m = \sigma_0 \cdot \Delta_\sigma^m = \sigma_0 \cdot 2^{m/Q}, \quad (7.22)$$

¹¹ See Sec. D.1 in the Appendix for additional details on combining Gaussian filters.

for $m = 0, \dots, M - 1$. As follows from Eqn. (7.18), each scale level G_m can be obtained directly from the discrete input image I by a filter operation

$$G_m = I * H^{G, \hat{\sigma}_m}, \quad (7.23)$$

with a Gaussian kernel $H^{G, \hat{\sigma}_m}$ of width

$$\hat{\sigma}_m = \sqrt{\sigma_m^2 - \sigma_s^2} = \sqrt{\sigma_0^2 \cdot 2^{2m/Q} - \sigma_s^2}. \quad (7.24)$$

In particular, the initial scale space level G_0 , (with the specified base scale σ_0) is obtained from the discrete input image I by linear filtering

$$G_0 = I * H^{G, \hat{\sigma}_0}, \quad (7.25)$$

using a Gaussian kernel of width

$$\hat{\sigma}_0 = \sqrt{\sigma_0^2 - \sigma_s^2} \quad (7.26)$$

Alternatively, using the relation $\sigma_m = \sigma_{m-1} \cdot \Delta_\sigma$ (from Eqn. (7.21)), the scale levels G_1, \dots, G_{M-1} could be calculated recursively from the base level G_0 in the form

$$G_m = G_{m-1} * H^{G, \sigma'_m}, \quad (7.27)$$

for $m > 0$, with a sequence of Gaussian kernels H^{G, σ'_m} of width

$$\sigma'_m = \sqrt{\sigma_m^2 - \sigma_{m-1}^2} = \sigma_0 \cdot 2^{m/Q} \cdot \sqrt{1 - 1/\Delta_\sigma^2}. \quad (7.28)$$

Table 7.1 lists the resulting kernel widths for $Q = 3$ levels per octave and base scale $\sigma_0 = 1.6$ over a scale range of 6 octaves. The value $\hat{\sigma}_m$ denotes the size of the Gaussian kernel required to compute the image at scale m from the discrete input image I (assumed to be sampled with $\sigma_s = 0.5$). σ'_m is the width of the Gaussian kernel to compute level m recursively from the previous level $m - 1$. Apparently (though perhaps unexpectedly), the kernel size required for recursive filtering (σ'_m) grows at the same (exponential) rate as the absolute kernel size $\hat{\sigma}_m$.¹² At scale level $m = 16$ and absolute scale $\sigma_{16} = 1.6 \cdot 2^{16/3} \approx 64.5$, for example, the Gaussian filter required to compute G_{16} directly from the input image I has the width $\hat{\sigma}_{16} = \sqrt{\sigma_{16}^2 - \sigma_s^2} = \sqrt{64.5080^2 - 0.5^2} \approx 64.5$, while the filter to blur incrementally from the previous scale level has the width $\sigma'_{16} = \sqrt{\sigma_{16}^2 - \sigma_{15}^2} = \sqrt{64.5080^2 - 51.1976^2} \approx 39.2$. Thus, in terms of the filter sizes required, it generally makes little difference if the scale space images are computed individually from the original image or incrementally from one level to the next. Since recursive filtering also tends to accrue numerical inaccuracies,

¹² The ratio of the kernel sizes $\hat{\sigma}_m / \sigma'_m$ converges to $\sqrt{1 - 1/\Delta_\sigma^2}$ (≈ 1.64 for $Q = 3$) and is thus practically constant for larger values of m .

Table 7.1 Filter sizes required for calculating Gaussian scale levels G_m for the first 6 octaves. Each octave consists of $Q = 3$ levels, spaced at Δ_σ in scale. The discrete input image I is assumed to be pre-filtered with σ_s . Column σ_m denotes the absolute scale at level m , starting with the specified base offset scale σ_0 . $\hat{\sigma}_m$ is the width of the Gaussian filter required to calculate level G_m directly from the input image I . Values σ'_m are the widths of the Gaussian kernels required to calculate level G_m from the previous level G_{m-1} . Note that the width of the Gaussian kernels needed for recursive filtering (σ'_m) grows at the same exponential rate as the size of the direct filter ($\hat{\sigma}_m$).

m	σ_m	$\hat{\sigma}_m$	σ'_m
18	102.4000	102.3988	62.2908
17	81.2749	81.2734	49.4402
16	64.5080	64.5060	39.2408
15	51.2000	51.1976	31.1454
14	40.6375	40.6344	24.7201
13	32.2540	32.2501	19.6204
12	25.6000	25.5951	15.5727
11	20.3187	20.3126	12.3601
10	16.1270	16.1192	9.8102
9	12.8000	12.7902	7.7864
8	10.1594	10.1471	6.1800
7	8.0635	8.0480	4.9051
6	6.4000	6.3804	3.8932
5	5.0797	5.0550	3.0900
4	4.0317	4.0006	2.4525
3	3.2000	3.1607	1.9466
2	2.5398	2.4901	1.5450
1	2.0159	1.9529	1.2263
0	1.6000	1.5199	—

m ... linear scale index

σ_m ... absolute scale at level m (Eqn. (7.22))

$\hat{\sigma}_m$... relative scale at level m w.r.t. the original image (Eqn. (7.24))

σ'_m ... relative scale at level m w.r.t. the previous level $m - 1$ (Eqn. (7.28))

$\sigma_s = 0.5$ (sampling scale)

$\sigma_0 = 1.6$ (base scale)

$Q = 3$ (number of levels per octave)

$\Delta_\sigma = 2^{1/Q} \approx 1.256$

this approach does not offer a significant advantage in general. Fortunately, the growth of the Gaussian kernels can be kept small by spatially sub-sampling after each octave, as described in Section 7.1.4.

The process of constructing a discrete Gaussian scale space using the same parameters as in Table 7.1 is illustrated in Fig. 7.6. Again the input image I is assumed to be pre-filtered at $\sigma_s = 0.5$ due to sampling and the absolute scale of the first level G_0 is set to $\sigma_0 = 1.6$. The scale ratio between successive levels is fixed at $\Delta_\sigma = 2^{1/3} \approx 1.25992$, i.e., each octave spans three discrete scale levels. As shown in this figure, each scale level G_m can be calculated either directly from the input image I by filtering with a Gaussian of width $\hat{\sigma}_m$, or recursively from the previous level by filtering with σ'_m .

7.1.3 LoG/DoG scale space

Interest point detection in the SIFT approach is based on finding local maxima in the output of Laplacian-of-Gaussian (LoG) filters over multiple scales. Analogous to the discrete Gaussian scale space described above, an LoG scale

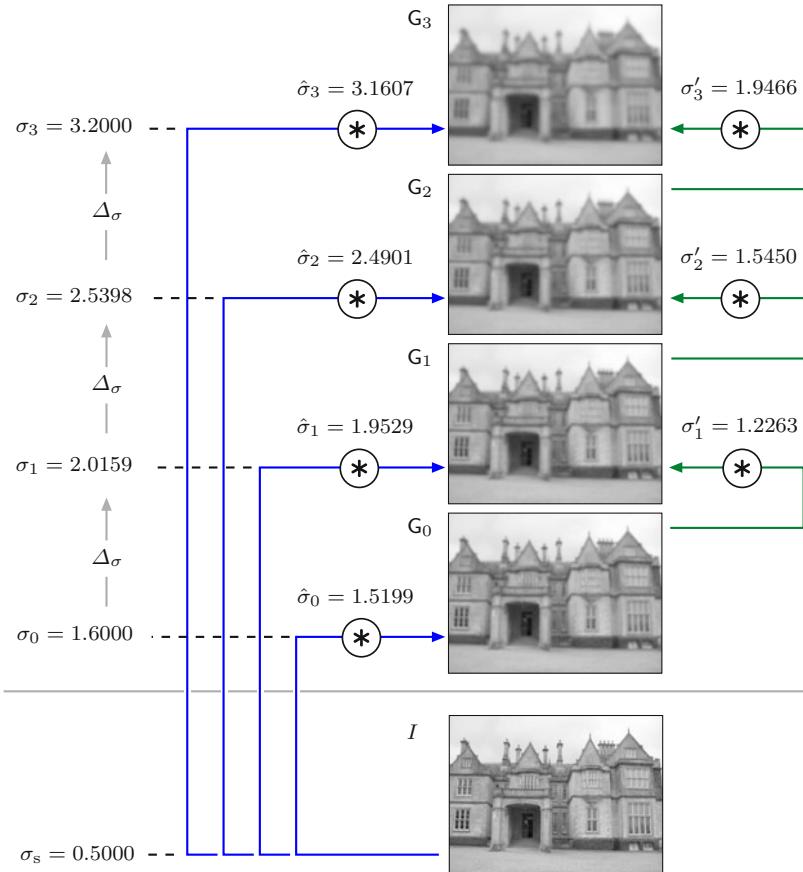


Figure 7.6 Gaussian scale space construction (first four levels). Parameters are the same as in Table 7.1. The discrete input image I is assumed to be pre-filtered with a Gaussian of width $\sigma_s = 0.5$; the scale of the initial level (base scale offset) is set to $\sigma_0 = 1.6$. The discrete scale space levels G_0, G_1, \dots (at absolute scales $\sigma_0, \sigma_1, \dots$) are slices through the continuous scale space. Scale levels can either be calculated by filtering directly from the discrete image I with Gaussian kernels of width $\hat{\sigma}_0, \hat{\sigma}_1, \dots$ (blue arrows) or, alternatively, by recursively filtering with $\sigma'_1, \sigma'_2, \dots$ (green arrows).

space representation of an image I can be defined as

$$\mathcal{L} = (\mathcal{L}_0, \mathcal{L}_1, \dots, \mathcal{L}_{M-1}), \quad (7.29)$$

with levels $\mathcal{L}_m = I * H^{\mathcal{L}, \sigma_m}$, where $H^{\mathcal{L}, \sigma_m}(x, y) \equiv \hat{L}_{\sigma_m}(x, y)$ is a scale-normalized LoG kernel of width σ_m (see Eqn. (7.9)).

As demonstrated in Eqn. (7.12), the LoG kernel can be approximated by the difference of two Gaussians (DoG) whose widths differ by a certain ratio κ .

Since pairs of adjacent scale layers in the Gaussian scale space are also separated by a fixed scale ratio, it is straightforward to construct a multi-scale DoG representation,

$$\mathbf{D} = (\mathbf{D}_0, \mathbf{D}_1, \dots, \mathbf{D}_{M-2}) \quad (7.30)$$

from an existing Gaussian scale space $\mathbf{G} = (\mathbf{G}_0, \mathbf{G}_1, \dots, \mathbf{G}_{M-1})$. The individual levels in the DoG scale space are defined as

$$\mathbf{D}_m = \hat{\lambda} \cdot (\mathbf{G}_{m+1} - \mathbf{G}_m) \approx \mathbf{L}_m, \quad (7.31)$$

for $m = 0, \dots, M-2$. The constant factor $\hat{\lambda}$ (defined in Eqn. (7.14)) can be omitted in the above expression, as the relative width of the involved Gaussians,

$$\kappa = \Delta_\sigma = \frac{\sigma_{m+1}}{\sigma_m} = 2^{1/Q}, \quad (7.32)$$

is simply the fixed scale ratio Δ_σ between successive scale space levels. Note that the DoG approximation does not require any additional normalization to approximate a scale-normalized LoG representation (see Eqns. 7.9 and 7.14). The process of calculating a DoG scale space from a discrete Gaussian scale space is illustrated in Fig. 7.7, using the same parameters as in Table 7.1 and Fig. 7.6.

7.1.4 Hierarchical scale space

Despite the fact that 2D Gaussian filter kernels are separable into one-dimensional kernels,¹³ the size of the required filter grows quickly with increasing scale, regardless if a direct or recursive approach is used (as shown in Table 7.1). However, each Gaussian filter operation reduces the bandwidth of the signal inversely proportional to the width of the kernel (see Sec. D.2). If the image size is kept constant over all scales, the images become increasingly oversampled at higher scale levels. In other words, the sampling rate can be reduced with increasing scale without losing relevant signal information.

Octaves and sub-sampling (decimation)

In particular, doubling the scale cuts the bandwidth by half, i. e., the signal at scale level 2σ has only half the bandwidth of the signal at level σ . In a Gaussian scale space representation it is thus safe to down-sample the image to half the sample rate after each octave without any loss of information. This suggests a very efficient, “pyramid-like” approach for constructing a DoG scale space, as illustrated in Fig. 7.8.¹⁴ At the start (bottom) of each octave, the

¹³ See also Vol. 1 [20, Sec. 5.3.3].

¹⁴ Successive reduction of image resolution by sub-sampling is the core concept of “image pyramid” methods [22].

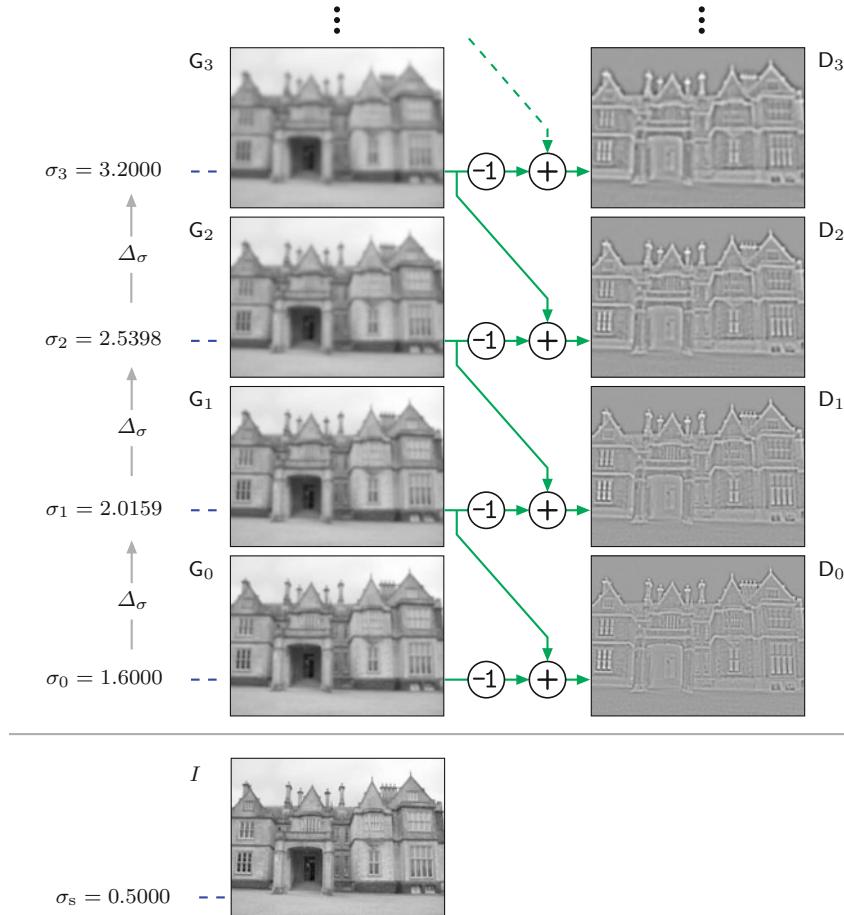


Figure 7.7 Difference-of-Gaussians (DoG) scale space construction. The differences of successive levels G_0, G_1, \dots of the Gaussian scale space (see Fig. 7.6) are used to approximate an LoG scale space. Each DoG level D_m is calculated as the point-wise difference between Gaussian levels $G_{m+1} - G_m$. The values in D_0, \dots, D_3 are scale-normalized (see Eqn. (7.14)) and mapped to a uniform intensity range for viewing.

image is down-sampled to half the resolution, i.e., each pixel in the new octave covers twice the distance of the pixels in the previous octave in every spatial direction. Within each octave, the same small Gaussian kernels can be used for successive filtering, since their relative widths (with respect to the original sampling lattice) also implicitly double at each octave. To describe these relations formally, we use

$$\mathbf{G} = (\mathbf{G}_0, \mathbf{G}_1, \dots, \mathbf{G}_{P-1}) \quad (7.33)$$

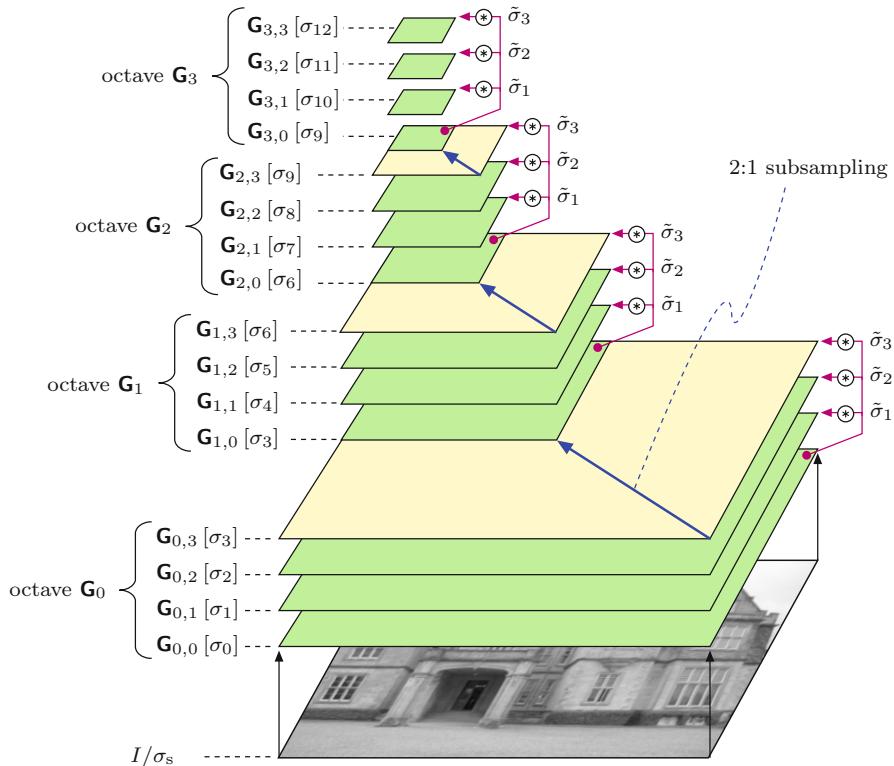


Figure 7.8 Hierarchical Gaussian scale space. Each octave extends over $Q = 3$ scale steps. The base level $\mathbf{G}_{p,0}$ of each octave $p > 0$ is obtained by 2:1 sub-sampling of the top level $\mathbf{G}_{p-1,3}$ of the next-lower octave. At the transition between octaves, the resolution (image size) is cut in half. The absolute scale at octave level $\mathbf{G}_{p,q}$ is σ_m , with $m = Qp + q$. Within each octave, the same set of Gaussian kernels ($\tilde{\sigma}_1, \tilde{\sigma}_2, \tilde{\sigma}_3$) is used to calculate the following levels from the octave's base level $\mathbf{G}_{p,0}$.

to denote a *hierarchical Gaussian scale space* consisting of P octaves. Each octave

$$\mathbf{G}_p = (\mathbf{G}_{p,0}, \mathbf{G}_{p,1}, \dots, \mathbf{G}_{p,Q}), \quad (7.34)$$

consists of $Q+1$ scale levels $\mathbf{G}_{p,q}$, where $p \in [0, P-1]$ is the octave index and $q \in [0, Q]$ is the level index within the containing octave \mathbf{G}_p . With respect to *absolute scale*, a level $\mathbf{G}_{p,q} = \mathbf{G}_p(q)$ in the hierarchical Gaussian scale space corresponds to the level \mathbf{G}_m in the non-hierarchical Gaussian scale space (see Eqn. (7.20)), with the level index

$$m = Qp + q. \quad (7.35)$$

As follows from Eqn. (7.22), the **absolute scale** at level $\mathbf{G}_{p,q}$ then is

$$\begin{aligned}\sigma_{p,q} &= \sigma_m = \sigma_0 \cdot \Delta_\sigma^m = \sigma_0 \cdot 2^{m/Q} \\ &= \sigma_0 \cdot 2^{(Qp+q)/Q} = \sigma_0 \cdot 2^{p+q/Q},\end{aligned}\quad (7.36)$$

where $\sigma_0 = \sigma_{0,0}$ denotes the predefined base scale offset (e.g., $\sigma_0 = 1.6$ in [Table 7.1](#)). In particular, the absolute scale of the base level $\mathbf{G}_{p,0}$ of *any* octave \mathbf{G}_p is

$$\sigma_{p,0} = \sigma_0 \cdot 2^p. \quad (7.37)$$

The **decimated scale** $\dot{\sigma}_{p,q}$ is the absolute scale $\sigma_{p,q}$ (Eqn. (7.36)) expressed in the coordinate units of octave \mathbf{G}_p , that is,

$$\dot{\sigma}_{p,q} = \dot{\sigma}_q = \sigma_{p,q} \cdot 2^{-p} = \sigma_0 \cdot 2^{p+q/Q} \cdot 2^{-p} = \sigma_0 \cdot 2^{q/Q}. \quad (7.38)$$

Note that the decimated scale $\dot{\sigma}_{p,q}$ is independent of the octave index p and therefore $\dot{\sigma}_{p,q} \equiv \dot{\sigma}_q$, for any level index q .

From the octave's base level $\mathbf{G}_{p,0}$, the subsequent levels in the same octave can be calculated by filtering with relatively small Gaussian kernels. The size of the kernel needed to calculate scale level $\mathbf{G}_{p,q}$ from the octave's base level $\mathbf{G}_{p,0}$ is obtained from the corresponding decimated scales (Eqn. (7.38)) as

$$\tilde{\sigma}_q = \sqrt{\dot{\sigma}_q^2 - \dot{\sigma}_0^2} = \sqrt{(\sigma_0 \cdot 2^{q/Q})^2 - \sigma_0^2} = \sigma_0 \cdot \sqrt{2^{2q/Q} - 1}. \quad (7.39)$$

Note that $\tilde{\sigma}_q$ is independent of the octave index p and thus the *same* filter kernels can be used at each octave. For example, with $Q = 3$ and $\sigma_0 = 1.6$ (as used in [Table 7.1](#)) the resulting kernel widths are

$$\tilde{\sigma}_1 = 1.2263 \quad \tilde{\sigma}_2 = 1.9725 \quad \tilde{\sigma}_3 = 2.7713. \quad (7.40)$$

Also note that, instead of filtering all scale levels $\mathbf{G}_{p,q}$ in an octave from the corresponding base level $\mathbf{G}_{p,0}$, we could calculate them recursively from the next-lower level $\mathbf{G}_{p,q-1}$. While this approach requires even smaller Gaussian kernels, recursive filtering tends to accrue numerical inaccuracies. Nevertheless, the method is used frequently in scale-space implementations.

Decimation between successive octaves

With $M \times N$ being the size of the original image I , every sub-sampling step between octaves cuts the size of the image by half, i.e.,

$$M_{p+1} \times N_{p+1} = \left\lfloor \frac{M_p}{2} \right\rfloor \times \left\lfloor \frac{N_p}{2} \right\rfloor, \quad (7.41)$$

for octave index $p \geq 0$, with $M_0 = M$ and $N_0 = N$. The resulting image size at octave \mathbf{G}_p is thus¹⁵

$$M_p \times N_p = \left\lfloor \frac{M}{2^p} \right\rfloor \times \left\lfloor \frac{N}{2^p} \right\rfloor. \quad (7.42)$$

The base level $\mathbf{G}_{p,0}$ of each octave \mathbf{G}_p , $p > 0$ is obtained by sub-sampling the top level $\mathbf{G}_{p-1,Q}$ of the next-lower octave, that is,

$$\mathbf{G}_{p,0} = \text{DECIMATE}(\mathbf{G}_{p-1,Q}), \quad (7.43)$$

where $\text{DECIMATE}(G)$ denotes the 2:1 sub-sampling operation, that is,

$$\mathbf{G}_{p,0}(u, v) \leftarrow \mathbf{G}_{p-1,Q}(2u, 2v), \quad (7.44)$$

for each sample position $(u, v) \in [0, M_p - 1] \times [0, N_p - 1]$. Additional low-pass filtering is not required prior to sub-sampling since the Gaussian smoothing performed in each octave also cuts the bandwidth by half.

The main steps involved in constructing a hierarchical Gaussian scale space are summarized in [Alg. 7.1](#). In summary, the input image I is first blurred to scale σ_0 by filtering with a Gaussian kernel of width $\hat{\sigma}_0$. Within each octave \mathbf{G}_p , the scale levels $\mathbf{G}_{p,q}$ are calculated from the base level $\mathbf{G}_{p,0}$ by filtering with a set of Gaussian filters of width $\tilde{\sigma}_q$ ($q = 1, \dots, Q$). Note that the values $\tilde{\sigma}_q$ and the corresponding Gaussian kernels $H^{G, \tilde{\sigma}_q}$ can be pre-calculated once since they are independent of the octave index p ([Alg. 7.1](#), lines 13–14). The base level $\mathbf{G}_{p,0}$ of each higher octave \mathbf{G}_p is obtained by decimating the top level $\mathbf{G}_{p-1,Q}$ of the previous octave \mathbf{G}_{p-1} . Typical parameter values are $\sigma_s = 0.5$, $\sigma_0 = 1.6$, $Q = 3$, $P = 4$.

Spatial positions in the hierarchical scale space

To properly associate the spatial positions of features detected in different octaves of the hierarchical scale space we define the function

$$\mathbf{x} = \text{REALPOS}(\mathbf{x}_p, p)$$

that maps the continuous position $\mathbf{x}_p = (x_p, y_p)$ in the local coordinate system of octave \mathbf{G}_p to the corresponding position $\mathbf{x} = (x, y)$ in the coordinate system of the original full-resolution image. The function REALPOS can be defined recursively by relating the positions in successive octaves as

$$\text{REALPOS}(\mathbf{x}_p, p) = \begin{cases} \mathbf{x}_p & \text{for } p = 0, \\ \text{REALPOS}(2 \cdot \mathbf{x}_p, p-1) & \text{for } p > 0, \end{cases} \quad (7.45)$$

¹⁵ This conclusion might not seem obvious but is fairly easy to show. It means that, given some initial number $n_0 \in \mathbb{N}_0$ and the relation $n_{i+1} = \left\lfloor \frac{n_i}{2} \right\rfloor$, then $n_i = \left\lfloor \frac{n_0}{2^i} \right\rfloor$.

Algorithm 7.1 Building a hierarchical Gaussian scale space. The input image I is first blurred to scale σ_0 by filtering with a Gaussian kernel of width $\hat{\sigma}_0$. In each octave \mathbf{G}_p , the scale levels $\mathbf{G}_{p,q}$ are calculated from the base level $\mathbf{G}_{p,0}$ by filtering with a set of Gaussian filters of width $\tilde{\sigma}_1, \dots, \tilde{\sigma}_Q$. The base level $\mathbf{G}_{p,0}$ of each higher octave is obtained by sub-sampling the top level $\mathbf{G}_{p-1,Q}$ of the previous octave.

```

1: BUILDGAUSSIANSCALESPACE( $I, \sigma_s, \sigma_0, Q, P$ )
   Input:  $I$ , source image;  $\sigma_s$ , sampling scale;  $\sigma_0$ , reference scale of the
          first octave;  $Q$ , number of scale steps per octave;  $P$ , number of octaves.
          Returns a hierarchical Gaussian scale space representation  $\mathbf{G}$  of the
          image  $I$ .
2:  $\hat{\sigma}_0 \leftarrow (\sigma_0^2 - \sigma_s^2)^{1/2}$                                 ▷ Eqn. (7.26)
3:  $\mathbf{G}_{0,0} \leftarrow I * H^{G, \hat{\sigma}_0}$            ▷ apply 2D Gaussian filter of width  $\hat{\sigma}_0$ 
4:  $\mathbf{G}_0 \leftarrow \text{MAKEGAUSSIANOCTAVE}(\mathbf{G}_{0,0}, 0)$       ▷ create octave  $\mathbf{G}_0$ 
5: for  $p \leftarrow 1, \dots, P-1$  do                                     ▷ octave index  $p$ 
6:    $\mathbf{G}_{p-1,Q} \leftarrow \mathbf{G}_{p-1}(Q)$ 
7:    $\mathbf{G}_{p,0} \leftarrow \text{DECIMATE}(\mathbf{G}_{p-1,Q})$ 
8:    $\mathbf{G}_p \leftarrow \text{MAKEGAUSSIANOCTAVE}(\mathbf{G}_{p,0}, p)$       ▷ create octave  $\mathbf{G}_p$ 
9:  $\mathbf{G} \leftarrow (\mathbf{G}_0, \dots, \mathbf{G}_{P-1})$ 
10: return  $\mathbf{G}$ .                                         ▷ hierarchical Gaussian scale space  $\mathbf{G}$ 


---


11: MAKEGAUSSIANOCTAVE( $\mathbf{G}_{p,0}, p$ )
   Input:  $\mathbf{G}_{p,0}$ , octave base level;  $p$ , octave index.
12: for  $q \leftarrow 1, \dots, Q$  do                                     ▷ level index  $q$ 
13:    $\tilde{\sigma}_q \leftarrow \sigma_0 \cdot \sqrt{2^{2q}/Q} - 1$            ▷ Eqn. (7.39)
14:    $\mathbf{G}_{p,q} \leftarrow \mathbf{G}_{p,0} * H^{G, \tilde{\sigma}_q}$        ▷ apply 2D Gaussian filter of width  $\tilde{\sigma}_q$ 
15:    $\mathbf{G}_p \leftarrow (\mathbf{G}_{p,0}, \dots, \mathbf{G}_{p,Q})$ 
16: return  $\mathbf{G}_p$ .                                         ▷ scale space octave  $\mathbf{G}_p$ 


---


17: DECIMATE( $\mathbf{G}_{\text{in}}$ )
   Input:  $\mathbf{G}_{\text{in}}$ , Gaussian octave level of size  $M \times N$ .
18:  $(M, N) \leftarrow \text{SIZE}(\mathbf{G}_{\text{in}})$ 
19:  $M' \leftarrow \lfloor \frac{M}{2} \rfloor, N' \leftarrow \lfloor \frac{N}{2} \rfloor$ 
20:  $\mathbf{G}_{\text{out}} \leftarrow \text{new image of size } M' \times N'$ 
21: for all image coordinates  $(u, v) \in M' \times N'$  do
22:    $\mathbf{G}_{\text{out}}(u, v) \leftarrow \mathbf{G}_{\text{in}}(2u, 2v)$            ▷ 2:1 sub-sampling
23: return  $\mathbf{G}_{\text{out}}$ .                                         ▷ decimated octave level  $\mathbf{G}_{\text{out}}$ 
```

which yields the position in the reference coordinate system of the original image (equivalent to octave \mathbf{G}_0) as

$$\mathbf{x} = \text{REALPOS}(\mathbf{x}_p, p) = \text{REALPOS}(2^p \cdot \mathbf{x}_p, 0) = 2^p \cdot \mathbf{x}_p. \quad (7.46)$$

Hierarchical LoG/DoG scale space

Analogous to the scheme shown in Fig. 7.7, a *hierarchical* Difference-of-Gaussian (DoG) scale space representation is obtained by calculating the difference of adjacent scale levels within each octave \mathbf{G}_p of the hierarchical Gaussian scale space, that is,

$$\mathbf{D}_{p,q} = \mathbf{G}_{p,q+1} - \mathbf{G}_{p,q}, \quad (7.47)$$

for level numbers $q \in [0, Q-1]$. Figure 7.9 shows the corresponding Gaussian and DoG scale levels for the previous example over a range of three octaves. To demonstrate the effects of sub-sampling, the same information is shown in Figs. 7.10 and 7.11, with all level images scaled to the same size. Figure 7.11 also shows the absolute values of the DoG response, which are effectively used for detecting interest points at different scale levels. Note how blob-like features stand out and disappear again as the scale varies from fine to coarse. Analogous results obtained from a different image are shown in Figs. 7.12 and 7.13.

7.1.5 Scale space implementation in SIFT

In the SIFT approach, the absolute value of the DoG response is used to localize interest points at different scales. For this purpose, local maxima are detected in the three-dimensional space spanned by the spatial x/y -positions and the scale coordinate. To determine local maxima along the scale dimension over a full octave, two additional DoG levels, $\mathbf{D}_{p,-1}$ and $\mathbf{D}_{p,Q}$, and two additional Gaussian scale levels, $\mathbf{G}_{p,-1}$ and $\mathbf{G}_{p,Q+1}$, are required in each octave.

In total, each octave \mathbf{G}_p then consists of $Q + 3$ Gaussian scale levels $\mathbf{G}_{p,q}$ ($q = -1, \dots, Q + 1$) and $Q + 2$ DoG levels $\mathbf{D}_{p,q}$ ($q = -1, \dots, Q$), as shown in Fig. 7.14. For the base level $\mathbf{G}_{0,-1}$, the scale index is $m = -1$ and its absolute scale (see Eqns. (7.22) and (7.36)) is

$$\sigma_{0,-1} = \sigma_0 \cdot 2^{-1/Q} = \sigma_0 \cdot \frac{1}{\Delta_\sigma}. \quad (7.48)$$

Thus, with the usual settings ($\sigma_0 = 1.6$ and $Q = 3$), the *absolute* scale values for the six levels of the first octave are

$$\begin{aligned} \sigma_{0,-1} &= 1.2699, & \sigma_{0,0} &= 1.6000, & \sigma_{0,1} &= 2.0159, \\ \sigma_{0,2} &= 2.5398, & \sigma_{0,3} &= 3.2000, & \sigma_{0,4} &= 4.0317. \end{aligned}$$

(see Table 7.2). The complete set of scale values for a SIFT scale space with four octaves ($p = 0, \dots, 3$) is listed in Table 7.2.

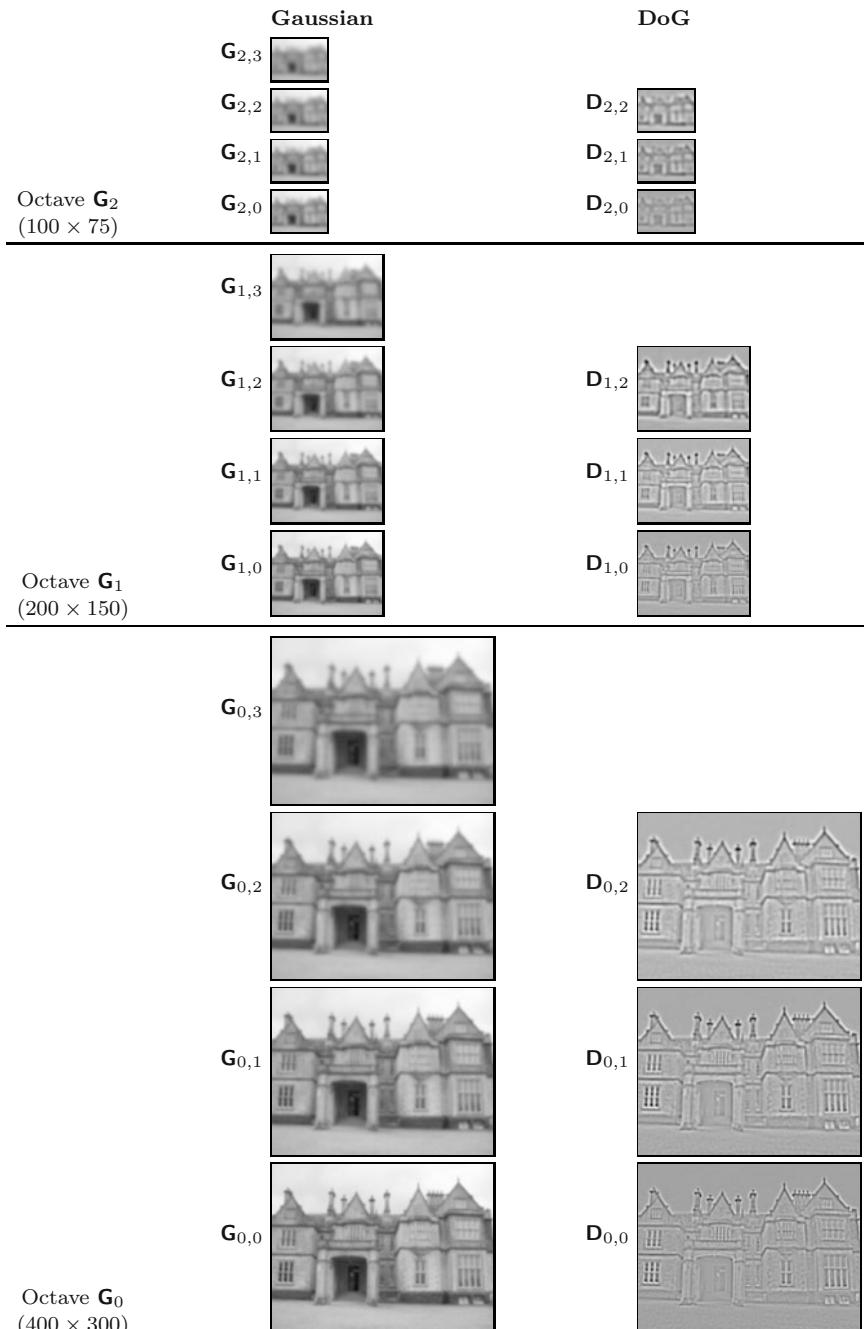


Figure 7.9 Hierarchical Gaussian and DoG scale space example, with $P = Q = 3$. Gaussian scale space levels $\mathbf{G}_{p,q}$ are shown in the left column, DoG levels $\mathbf{D}_{p,q}$ in the right column. All images are shown at their real scale.

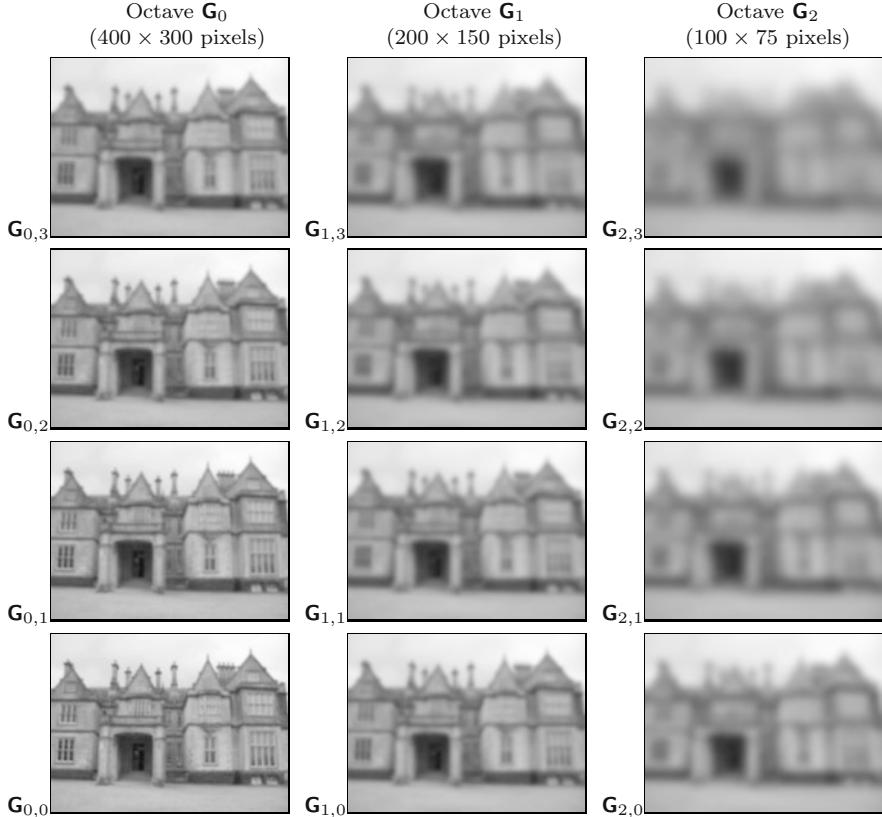


Figure 7.10 Hierarchical Gaussian scale space example (`castle` image). All images are scaled to the same size. Note that $\mathbf{G}_{1,0}$ is merely a sub-sampled copy of $\mathbf{G}_{0,3}$; analogously, $\mathbf{G}_{2,0}$ is sub-sampled from $\mathbf{G}_{1,3}$.

To construct the Gaussian part of the first scale space octave \mathbf{G}_0 , the initial level $\mathbf{G}_{0,-1}$ is obtained by filtering the input image I with a Gaussian kernel of width

$$\hat{\sigma}_{0,-1} = \sqrt{\sigma_{0,-1}^2 - \sigma_s^2} \approx 1.1673. \quad (7.49)$$

For the higher octaves \mathbf{G}_p , the initial level ($q = -1$) is obtained by sub-sampling (decimating) level $Q-1$ of the next-lower octave \mathbf{G}_{p-1} , that is,

$$\mathbf{G}_{p,-1} \leftarrow \text{DECIMATE}(\mathbf{G}_{p-1,Q-1}), \quad (7.50)$$

for $p > 0$, analogous to Eqn. (7.43).

The remaining levels $\mathbf{G}_{p,0}, \dots, \mathbf{G}_{p,Q+1}$ of the octave are either calculated by incremental filtering (as described in Fig. 7.6) or by filtering from the octave's initial level $\mathbf{G}_{p,-1}$ with a Gaussian of width $\tilde{\sigma}_{p,q}$ (see Eqn. (7.39)). The

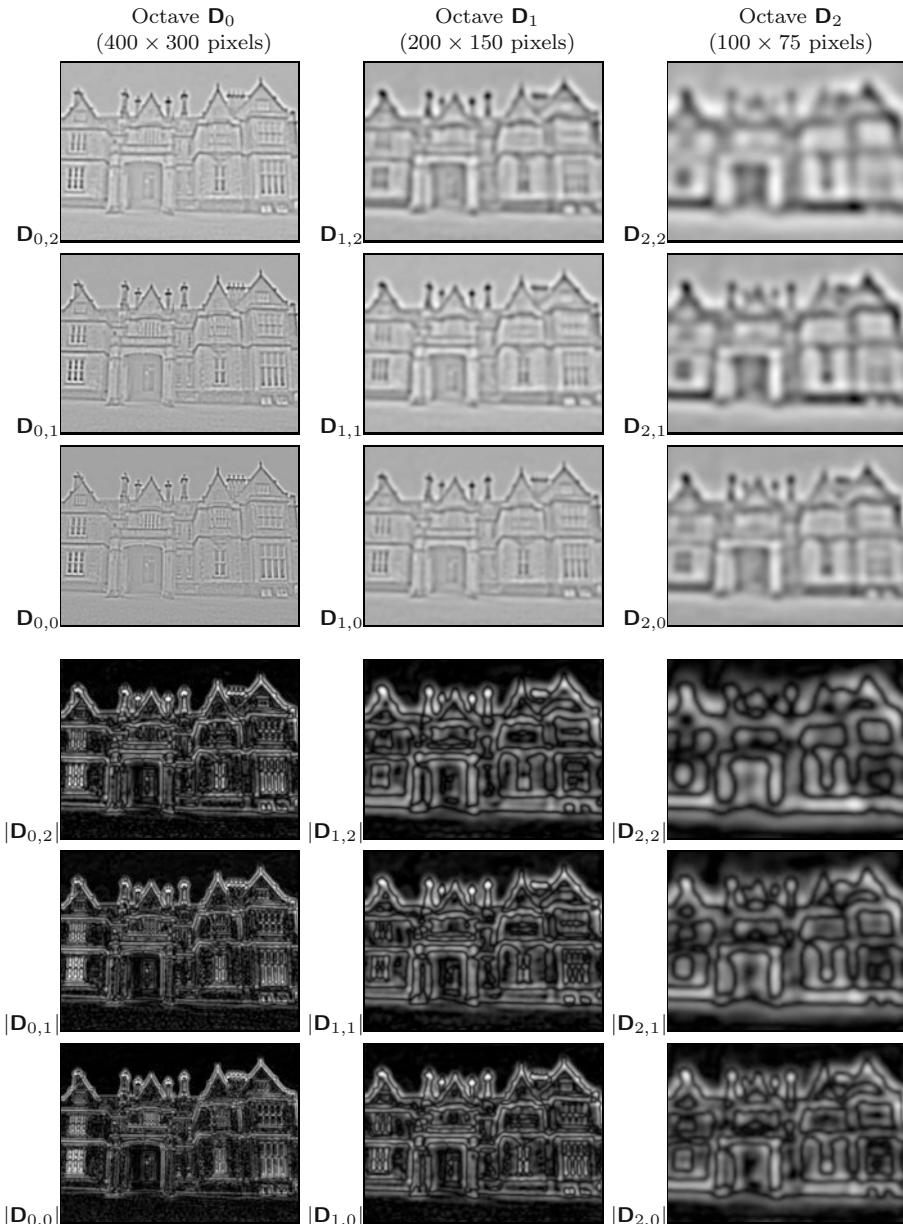


Figure 7.11 Hierarchical DoG scale space example (`castle` image). The three top rows show the positive and negative DoG values (zero is mapped to intermediate gray). The three bottom rows show the absolute values of the DoG results (zero is mapped to black, maximum values to white). All images are scaled to the size of the original image.

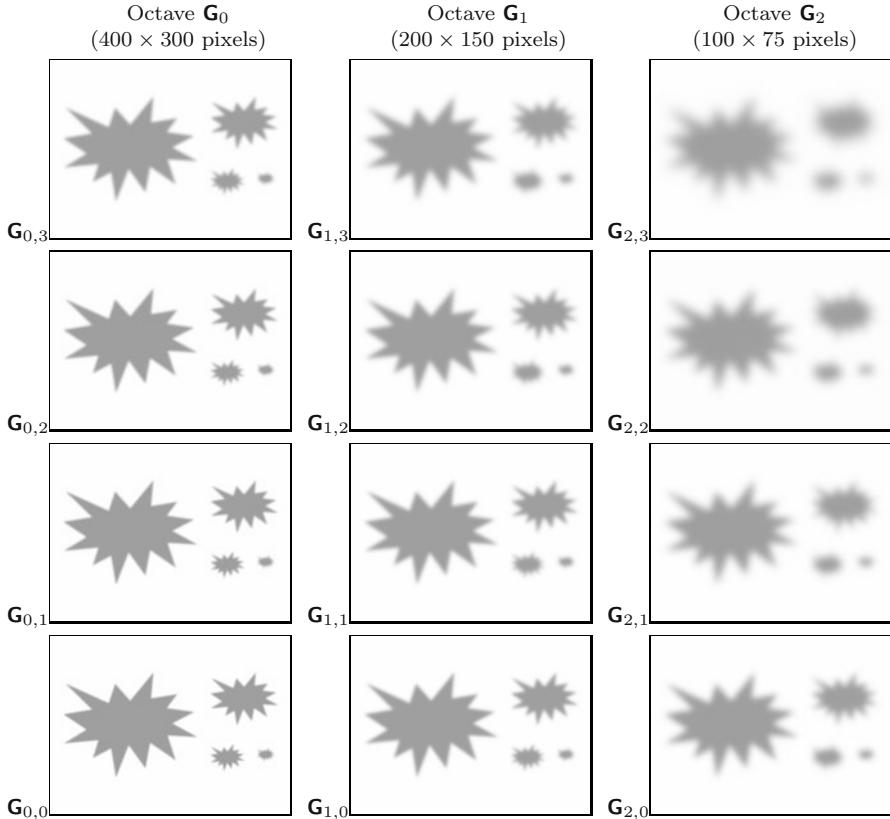


Figure 7.12 Hierarchical Gaussian scale space example (`stars` image).

advantage of the direct approach is that numerical errors do not accrue across the scale space; the disadvantage is that the kernels are up to 50 % larger than those needed for the incremental approach ($\tilde{\sigma}_{0,4} = 3.8265$ vs. $\sigma'_{0,4} = 2.4525$). Note that the inner levels $\mathbf{G}_{p,q}$ of all higher octaves (i.e., $p > 0, q \geq 0$) are calculated from the base level $\mathbf{G}_{p,-1}$, using the *same* set of kernels as for the first octave, as listed in [Table 7.2](#). Thus, since no larger kernels are needed to build the entire scale space and since the Gaussian is a separable filter, the penalty associated with the direct approach is usually acceptable.

7.2 Key point selection and refinement

Key points are identified in three steps: (1) detection of extremal points in the DoG scale space, (2) position refinement by local interpolation, and (3) elimination of edge responses. These steps are detailed in the following and sum-

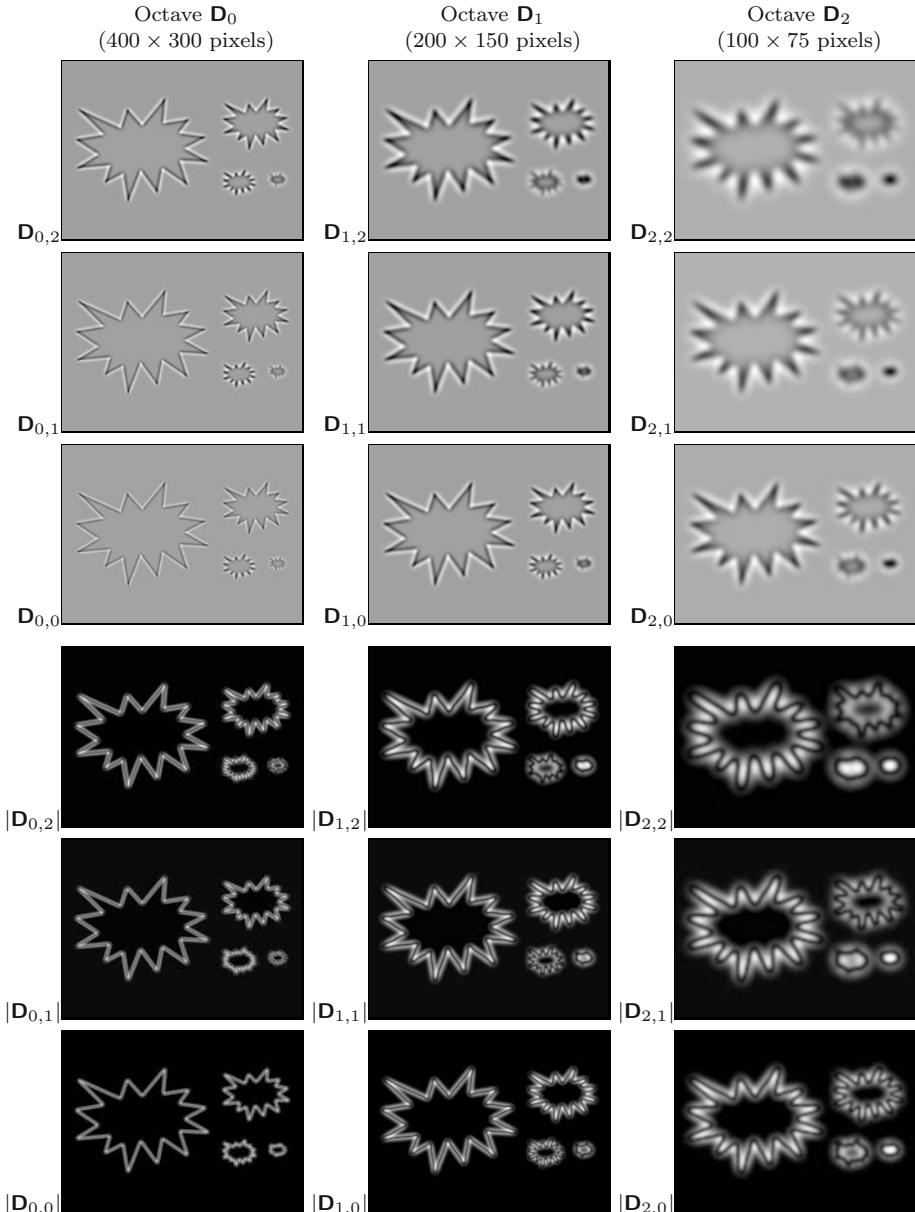


Figure 7.13 Hierarchical DoG scale space example (`stars` image). The three top rows show the positive and negative DoG values (zero is mapped to intermediate gray). The three bottom rows show the absolute values of the DoG results (zero is mapped to black, maximum values to white). All images are scaled to the size of the original image.

Algorithm 7.2 Building a SIFT scale space. This procedure is an extension of [Alg. 7.1](#) and takes the same parameters. The SIFT scale space consists of two components: a hierarchical Gaussian scale space $\mathbf{G} = (\mathbf{G}_0, \dots, \mathbf{G}_{P-1})$ and a hierarchical DoG scale space $\mathbf{D} = (\mathbf{D}_0, \dots, \mathbf{D}_{P-1})$. Each Gaussian octave \mathbf{G}_p holds $Q+3$ levels $(\mathbf{G}_{p,-1}, \dots, \mathbf{G}_{p,Q+1})$. For each Gaussian octave, the first level $\mathbf{G}_{p,-1}$ is obtained by decimating level $\mathbf{G}_{p-1,Q}$ of the previous octave. A DoG octave \mathbf{D}_p contains $Q+2$ levels $(\mathbf{D}_{p,-1}, \dots, \mathbf{D}_{p,Q})$. Each DoG level $\mathbf{D}_{p,q}$ is calculated as the pointwise difference of two adjacent Gaussian levels $\mathbf{G}_{p,q+1}$ and $\mathbf{G}_{p,q}$. Typical settings are $\sigma_s = 0.5$, $\sigma_0 = 1.6$, $Q = 3$, $P = 4$.

```

1: BUILDSCALESPACE( $I, \sigma_s, \sigma_0, P, Q$ )
   Input:  $I$ , source image;  $\sigma_s$ , sampling scale;  $\sigma_0$ , reference scale of the
         first octave;  $P$ , number of octaves;  $Q$ , number of scale steps per octave;
         Returns a SIFT scale space representation  $\langle \mathbf{G}, \mathbf{D} \rangle$  of the image  $I$ .
2:  $\sigma_{0,-1} \leftarrow \sigma_0 \cdot 2^{-1/Q}$                                 ▷ Eqn. (7.48)
3:  $\hat{\sigma}_{0,-1} \leftarrow (\sigma_{0,-1}^2 - \sigma_s^2)^{1/2}$                 ▷ Eqn. (7.49)
4:  $\mathbf{G}_{0,-1} \leftarrow I * H^{\mathbf{G}, \hat{\sigma}_{0,-1}}$       ▷ apply 2D Gaussian filter of width  $\hat{\sigma}_{0,-1}$ 
5:  $\mathbf{G}_0 \leftarrow \text{MAKEGAUSSIANOCTAVE}(\mathbf{G}_{0,-1}, 0)$     ▷ create Gauss. octave 0
6: for  $p \leftarrow 1, \dots, P-1$  do                                ▷ for octaves 1, ...,  $P-1$ 
7:    $\mathbf{G}_{p-1,Q} \leftarrow \mathbf{G}_{p-1}(Q)$ 
8:    $\mathbf{G}_{p,-1} \leftarrow \text{DECIMATE}(\mathbf{G}_{p-1,Q-1})$           ▷ see Alg. 7.1
9:    $\mathbf{G}_p \leftarrow \text{MAKEGAUSSIANOCTAVE}(\mathbf{G}_{p,-1}, p)$     ▷ create octave  $p$ 
10:   $\mathbf{G} \leftarrow (\mathbf{G}_0, \dots, \mathbf{G}_{P-1})$            ▷ assemble the Gaussian scale space  $\mathbf{G}$ 
11:  for  $p \leftarrow 0, \dots, P-1$  do
12:     $\mathbf{D}_p \leftarrow \text{MAKEDOGOCTAVE}(\mathbf{G}_p, p)$ 
13:   $\mathbf{D} \leftarrow (\mathbf{D}_0, \dots, \mathbf{D}_{P-1})$            ▷ assemble the DoG scale space  $\mathbf{D}$ 
14:  return  $\langle \mathbf{G}, \mathbf{D} \rangle$ 



---


15:  $\text{MAKEGAUSSIANOCTAVE}(\mathbf{G}_{p,-1}, p)$           ▷ Gaussian base level  $\mathbf{G}_{p,-1}$ 
16: for  $q \leftarrow 0, \dots, Q+1$  do
17:    $\tilde{\sigma}_q \leftarrow \sigma_0 \cdot \sqrt{2^{2q}/Q - 1}$         ▷ Eqn. (7.39)
18:    $\mathbf{G}_{p,q} \leftarrow \mathbf{G}_{p,-1} * H^{\mathbf{G}, \tilde{\sigma}_q}$     ▷ apply 2D Gaussian filter of width  $\tilde{\sigma}_q$ 
19: return  $(\mathbf{G}_{p,-1}, \dots, \mathbf{G}_{p,Q+1})$ .           ▷ Gaussian octave  $\mathbf{G}_p$ 



---


20:  $\text{MAKEDOGOCTAVE}(\mathbf{G}_p, p)$           ▷ Gaussian octave  $\mathbf{G}_p$ 
21: for  $q \leftarrow -1, \dots, Q$  do
22:    $\mathbf{G}_{p,q} \leftarrow \mathbf{G}_p(q)$ 
23:    $\mathbf{G}_{p,q+1} \leftarrow \mathbf{G}_p(q+1)$ 
24:    $\mathbf{D}_{p,q} \leftarrow \mathbf{G}_{p,q+1} - \mathbf{G}_{p,q}$        ▷ diff. of Gaussians, Eqn. (7.31)
25: return  $(\mathbf{D}_{p,-1}, \mathbf{D}_{p,0}, \dots, \mathbf{D}_{p,Q})$ .       ▷ DoG octave  $\mathbf{D}_p$ 

```

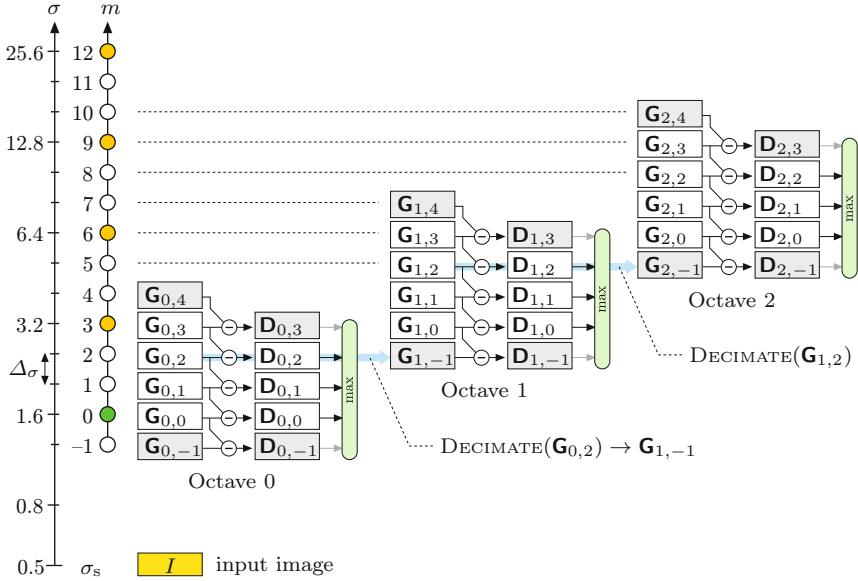


Figure 7.14 Scale space structure for SIFT with $P = 3$ octaves and $Q = 3$ levels per octave. To perform local maximum detection (“max”) over the full octave, $Q+2$ DoG scale space levels ($\mathbf{D}_{p,-1}, \dots, \mathbf{D}_{p,Q}$) are required. The blue arrows indicate the decimation steps between successive Gaussian octaves. Since the DoG levels are obtained by subtracting pairs of Gaussian scale space levels, $Q+3$ such levels ($\mathbf{G}_{p,-1}, \dots, \mathbf{G}_{p,Q+1}$) are needed in each octave \mathbf{G}_p . The two vertical axes on the left show the absolute scale (σ) and the discrete scale index (m), respectively. Note that the values along the absolute scale axis are logarithmic with constant multiplicative scale increments $\Delta\sigma = 2^{1/Q}$. The absolute scale of the input image (I) is assumed as $\sigma_I = 0.5$.

marized in Algs. 7.3–7.6.

7.2.1 Local extrema detection

In the first step, candidate interest points are detected as local extrema in the three-dimensional DoG scale space that we described in the previous section. Extrema detection is performed independently within each octave p . Given a point $\mathbf{c} = (u, v, q)$ at spatial position (u, v) and level q in octave p of the hierarchical DoG scale space \mathbf{D} , we use the function

$$D(\mathbf{c}) = \mathbf{D}_{p,q+k}(u, v) \quad (7.51)$$

in the following as a short notation for accessing samples in the DoG scale space. For accessing the 3D neighborhood of DoG values around each position \mathbf{c} , we define the map

$$\mathbf{N}_{\mathbf{c}}(i, j, k) = D(\mathbf{c} + i \cdot \mathbf{e}_i + j \cdot \mathbf{e}_j + k \cdot \mathbf{e}_k), \quad (7.52)$$

Table 7.2 Absolute and relative scale values for a SIFT scale space with four octaves. Each octave with index $p = 0, \dots, 3$ consists of 6 Gaussian scale layers $\mathbf{G}_{p,q}$, with $q = -1, \dots, 4$. For each scale layer, m is the scale index and $\sigma_{p,q}$ is the corresponding *absolute* scale. Within each octave p , $\tilde{\sigma}_{p,q}$ denotes the *relative* scale with respect to the octave's base layer $\mathbf{G}_{p,-1}$. Each base layer $\mathbf{G}_{p,-1}$ is obtained by sub-sampling (decimating) layer $q = Q - 1 = 2$ in the previous octave, i.e., $\mathbf{G}_{p,-1} = \text{DECIMATE}(\mathbf{G}_{p-1,Q-1})$, for $p > 0$. The base layer $\mathbf{G}_{0,-1}$ in the bottom octave is derived by Gaussian smoothing of the original image. Note that the relative scale values $\tilde{\sigma}_{p,q} = \tilde{\sigma}_q$ are the same inside every octave (independent of p) and thus the same Gaussian filter kernels can be used for calculating all octaves.

p	q	m	d	$\sigma_{p,q}$	$\tilde{\sigma}_q$	$\tilde{\sigma}_q$
3	4	13	8	32.2540	4.0317	3.8265
3	3	12	8	25.6000	3.2000	2.9372
3	2	11	8	20.3187	2.5398	2.1996
3	1	10	8	16.1270	2.0159	1.5656
3	0	9	8	12.8000	1.6000	0.9733
3	-1	8	8	10.1594	1.2699	0.0000
2	4	10	4	16.1270	4.0317	3.8265
2	3	9	4	12.8000	3.2000	2.9372
2	2	8	4	10.1594	2.5398	2.1996
2	1	7	4	8.0635	2.0159	1.5656
2	0	6	4	6.4000	1.6000	0.9733
2	-1	5	4	5.0797	1.2699	0.0000
1	4	7	2	8.0635	4.0317	3.8265
1	3	6	2	6.4000	3.2000	2.9372
1	2	5	2	5.0797	2.5398	2.1996
1	1	4	2	4.0317	2.0159	1.5656
1	0	3	2	3.2000	1.6000	0.9733
1	-1	2	2	2.5398	1.2699	0.0000
0	4	4	1	4.0317	4.0317	3.8265
0	3	3	1	3.2000	3.2000	2.9372
0	2	2	1	2.5398	2.5398	2.1996
0	1	1	1	2.0159	2.0159	1.5656
0	0	0	1	1.6000	1.6000	0.9733
0	-1	-1	1	1.2699	1.2699	0.0000

p ... octave index

q ... level index

m ... linear scale index ($m = Qp + q$)

d ... decimation factor ($d = 2^p$)

$\sigma_{p,q}$... absolute scale (Eqn. (7.36))

$\tilde{\sigma}_q$... decimated scale (Eqn. (7.38))

$\tilde{\sigma}_q$... relative decimated scale w.r.t.
octave's base level $\mathbf{G}_{p,-1}$ (Eqn.
(7.39))

$P = 3$ (number of octaves)

$Q = 3$ (levels per octave)

$\sigma_0 = 1.6$ (base scale)

with $i, j, k \in \{-1, 0, 1\}$ and the 3D unit vectors

$$\mathbf{e}_i = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{e}_j = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{e}_k = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \quad (7.53)$$

The neighborhood \mathbf{N}_c includes the center value $D(c)$ and the 26 values of its immediate neighbors. These values are used to estimate the 3D gradient vector and the Hessian matrix for the three-dimensional scale space position c , as described below.

For extrema detection, a position c in the DoG scale space is taken as a local extremum (minimum or maximum) if its value is smaller or greater than all other values of its neighborhood (see procedure `ISEXTREMUM(N)` in [Alg. 7.5](#) on p. 279). As illustrated in [Fig. 7.15](#), alternative neighborhoods with 18 or 10 cells may be specified for extrema detection.

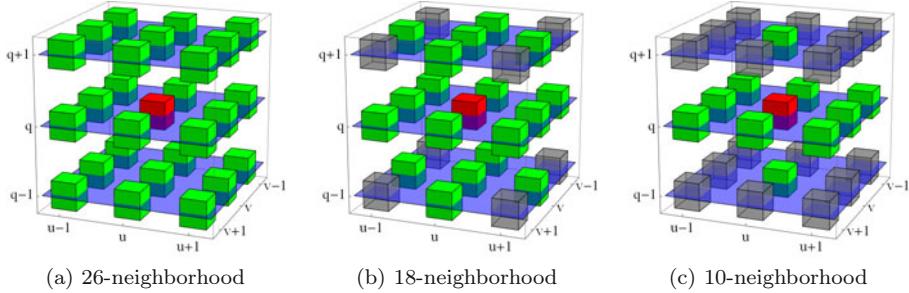


Figure 7.15 Three-dimensional neighborhoods for detecting local extrema in the DoG scale space. For a given position \mathbf{c} at some octave p and level q in the hierarchical DoG scale space \mathbf{D} , the set $\mathcal{N}_D(\mathbf{c})$ contains the complete 3D neighborhood of 27 cells (see Eqn. (7.52)). The red cube marks the center coordinate $\mathbf{c} = (u, v, q)$ in scale space octave \mathbf{D}_p (u, v are the relative space coordinates and q is the scale level index). Full $3 \times 3 \times 3$ neighborhood with 26 elements (a); alternative neighborhood ignoring the outer corners of the cube (b); neighborhood that includes only two adjacent cells along the scale axis (c). A local maximum/minimum is detected if the DoG value at the center is greater/smaller than any of the neighborhood values (green cubes).

7.2.2 Position refinement

After a local extremum is found in the DoG scale space, only its *discrete* 3D position is known. In the second step, a more accurate, *continuous* position for each candidate key point is estimated by fitting a quadratic function to the local neighborhood, as proposed in [18]. This is particularly important at the higher octaves of the scale space, where the spatial resolution becomes increasingly coarse due to successive decimation. Position refinement is based on a local second-order Taylor expansion of the discrete DoG function, which yields a continuous approximation function whose maximum or minimum can be found analytically. Additional details and illustrative examples are provided in Section B.7.2 of the Appendix.

At any extremal position $\mathbf{c} = (u, v, q)$ in octave p of the hierarchical DoG scale space \mathbf{D} , the corresponding $3 \times 3 \times 3$ neighborhood $\mathcal{N}_D(\mathbf{c})$ is used to estimate the elements of the continuous 3D gradient, that is,

$$\nabla_D(\mathbf{c}) = \begin{pmatrix} d_x \\ d_y \\ d_\sigma \end{pmatrix} = 0.5 \cdot \begin{pmatrix} D(\mathbf{c} + \mathbf{e}_i) - D(\mathbf{c} - \mathbf{e}_i) \\ D(\mathbf{c} + \mathbf{e}_j) - D(\mathbf{c} - \mathbf{e}_j) \\ D(\mathbf{c} + \mathbf{e}_k) - D(\mathbf{c} - \mathbf{e}_k) \end{pmatrix}, \quad (7.54)$$

with $D(\mathbf{c})$ as defined in Eqn. (7.51). Similarly, the 3×3 Hessian matrix for position \mathbf{c} is obtained as

$$\mathbf{H}_D(\mathbf{c}) = \begin{pmatrix} d_{xx} & d_{xy} & d_{x\sigma} \\ d_{xy} & d_{yy} & d_{y\sigma} \\ d_{x\sigma} & d_{y\sigma} & d_{\sigma\sigma} \end{pmatrix}, \quad (7.55)$$

with the required second order derivatives estimated as

$$\begin{aligned} d_{xx} &= D(\mathbf{c}-\mathbf{e}_i) - 2 \cdot D(\mathbf{c}) + D(\mathbf{c}+\mathbf{e}_i), \\ d_{yy} &= D(\mathbf{c}-\mathbf{e}_j) - 2 \cdot D(\mathbf{c}) + D(\mathbf{c}+\mathbf{e}_j), \\ d_{\sigma\sigma} &= D(\mathbf{c}-\mathbf{e}_k) - 2 \cdot D(\mathbf{c}) + D(\mathbf{c}+\mathbf{e}_k), \\ d_{xy} &= \frac{1}{4} \cdot [D(\mathbf{c}+\mathbf{e}_i+\mathbf{e}_j) - D(\mathbf{c}-\mathbf{e}_i+\mathbf{e}_j) - D(\mathbf{c}+\mathbf{e}_i-\mathbf{e}_j) + D(\mathbf{c}-\mathbf{e}_i-\mathbf{e}_j)], \\ d_{x\sigma} &= \frac{1}{4} \cdot [D(\mathbf{c}+\mathbf{e}_i+\mathbf{e}_k) - D(\mathbf{c}-\mathbf{e}_i+\mathbf{e}_k) - D(\mathbf{c}+\mathbf{e}_i-\mathbf{e}_k) + D(\mathbf{c}-\mathbf{e}_i-\mathbf{e}_k)], \\ d_{y\sigma} &= \frac{1}{4} \cdot [D(\mathbf{c}+\mathbf{e}_j+\mathbf{e}_k) - D(\mathbf{c}-\mathbf{e}_j+\mathbf{e}_k) - D(\mathbf{c}+\mathbf{e}_j-\mathbf{e}_k) + D(\mathbf{c}-\mathbf{e}_j-\mathbf{e}_k)]. \end{aligned} \quad (7.56)$$

See the procedures $\text{GRADIENT}(\mathcal{N})$ and $\text{HESSIAN}(\mathcal{N})$ in [Alg. 7.5](#) (p. 279) for additional details. From the gradient vector $\nabla_D(\mathbf{c})$ and the Hessian matrix $\mathbf{H}_D(\mathbf{c})$, the second order Taylor expansion around point \mathbf{c} is

$$\tilde{D}(\mathbf{c}, \mathbf{x}) = D(\mathbf{c}) + \nabla_D^\top(\mathbf{c}) \cdot (\mathbf{x} - \mathbf{c}) + \frac{1}{2} \cdot (\mathbf{x} - \mathbf{c})^\top \cdot \mathbf{H}_D(\mathbf{c}) \cdot (\mathbf{x} - \mathbf{c}). \quad (7.57)$$

The scalar-valued function $\tilde{D}(\mathbf{c}, \mathbf{x}) \in \mathbb{R}$, with $\mathbf{c} = (u, v, q)^\top$ and $\mathbf{x} = (x, y, \sigma)^\top$, is a local, *continuous* approximation of the discrete DoG function $\mathbf{D}_{p,q}(u, v)$ at octave p , scale level q , and spatial position u, v . This is a quadratic function with an extremum (maximum or minimum) at position

$$\check{\mathbf{x}} = \begin{pmatrix} \check{x} \\ \check{y} \\ \check{\sigma} \end{pmatrix} = \mathbf{c} + \mathbf{d} = \mathbf{c} - \underbrace{\mathbf{H}_D^{-1}(\mathbf{c}) \cdot \nabla_D(\mathbf{c})}_{\mathbf{d} = \check{\mathbf{x}} - \mathbf{c}}, \quad (7.58)$$

under the assumption that the inverse of the Hessian matrix \mathbf{H}_D exists. By inserting the extremal position $\check{\mathbf{x}}$ into Eqn. (7.57), the peak (minimum or maximum) *value* of the continuous approximation function \tilde{D} is found as¹⁶

$$\begin{aligned} D_{\text{peak}}(\mathbf{c}) &= \tilde{D}(\mathbf{c}, \check{\mathbf{x}}) = D(\mathbf{c}) + \frac{1}{2} \cdot \nabla_D^\top(\mathbf{c}) \cdot (\check{\mathbf{x}} - \mathbf{c}) \\ &= D(\mathbf{c}) + \frac{1}{2} \cdot \nabla_D^\top(\mathbf{c}) \cdot \mathbf{d}, \end{aligned} \quad (7.59)$$

where $\mathbf{d} = \check{\mathbf{x}} - \mathbf{c}$ (cf. Eqn. (7.58)) denotes the 3D vector between the neighborhood's discrete center position \mathbf{c} and the continuous extremal position $\check{\mathbf{x}}$. A scale space location \mathbf{c} is only retained as a candidate interest point if the estimated magnitude of the DoG exceeds a given threshold t_{peak} , that is, if

$$|D_{\text{peak}}(\mathbf{c})| > t_{\text{peak}}. \quad (7.60)$$

¹⁶ See Eqn. (B.90) in Sec. B.7.3 of the Appendix for details.

If the distance $\mathbf{d} = (x', y', \sigma')^\top$ from \mathbf{c} to the estimated (continuous) peak position $\check{\mathbf{x}}$ in Eqn. (7.58) is greater than a predefined limit (typically 0.5) in any spatial direction, the center point $\mathbf{c} = (u, v, q)^\top$ is moved to one of the neighboring DoG cells by maximally ± 1 unit steps along the u, v axes, i. e.,

$$\mathbf{c} \leftarrow \mathbf{c} + \begin{pmatrix} \min(1, \max(-1, \text{round}(x'))) \\ \min(1, \max(-1, \text{round}(y'))) \\ 0 \end{pmatrix}, \quad (7.61)$$

with x', y' being the first two components of \mathbf{d} , as defined in Eqn. (7.58). Based on the surrounding 3D neighborhood of this new point, a Taylor expansion (Eqn. (7.58)) is again performed to estimate a new peak location. This is repeated until either the peak location is inside the current DoG cell or the allowed number of repositioning steps n_{refine} is reached (typically n_{refine} is set to 4 or 5). The result of this step is a refined candidate point with optimized (continuous) spatial coordinates \check{x}, \check{y} ,

$$\check{\mathbf{c}} = \begin{pmatrix} \check{x} \\ \check{y} \\ \check{q} \end{pmatrix} = \mathbf{c} + \begin{pmatrix} x' \\ y' \\ 0 \end{pmatrix}. \quad (7.62)$$

Notice that (in this implementation) the scale level q remains unchanged even if the 3D Taylor expansion indicates that the estimated peak is located at another scale level. See procedure `REFINEKEYPOSITION()` in [Alg. 7.4](#) (p. 278) for a concise summary of the above steps.

Note. It should be mentioned that the original publication [80] is not particularly explicit about the above position refinement process and thus slightly different approaches are used in various open-source SIFT implementations. For example, the implementation in *VLFeat*¹⁷ [136] moves to one of the direct neighbors at the same scale level as described above, as long as $|x'|$ or $|y'|$ is greater than 0.6. *AutoPano-SIFT*¹⁸ by S. Nowozin calculates the length of the spatial displacement $d = \|(x', y')\|$ and discards the current point if $d > 2$. Otherwise it moves by $\Delta_u = \text{round}(x')$, $\Delta_v = \text{round}(y')$ without limiting the displacement to ± 1 . The *Open-Source SIFT Library*¹⁹ [54] used in *OpenCV* also makes full moves in the spatial directions and, in addition, potentially also changes the scale level by $\Delta_q = \text{round}(\sigma')$ in each iteration.

¹⁷ <http://www.vlfeat.org/overview/sift.html>

¹⁸ <http://sourceforge.net/projects/hugin/files/autopano-sift-C/>

¹⁹ <http://blogs.oregonstate.edu/hess/code/sift/>

7.2.3 Suppressing responses to edge-like structures

In the previous step, candidate interest points were selected as those locations in the DoG scale space where the Taylor approximation had a local maximum and the extrapolated DoG value was above a given threshold (t_{peak}). However, the DoG filter also responds strongly to edge-like structures. At such positions, interest points cannot be located with sufficient stability and repeatability. To eliminate the responses near edges, Lowe suggests the use of the principal curvatures of the two-dimensional DoG result along the spatial x, y axes, using the fact that the principal curvatures of a function are proportional to the eigenvalues of the function's Hessian matrix at a given point.

For a particular lattice point $\mathbf{c} = (u, v, q)$ in DoG scale space, with neighborhood $\mathcal{N}_D(\mathbf{c})$, the 2×2 Hessian matrix for the spatial coordinates is

$$\mathbf{H}_{xy}(\mathbf{c}) = \begin{pmatrix} d_{xx} & d_{xy} \\ d_{xy} & d_{yy} \end{pmatrix}, \quad (7.63)$$

with d_{xx} , d_{xy} , d_{yy} as defined in Eqn. (7.56), i.e., these values can be extracted from the corresponding 3×3 Hessian matrix $\mathbf{H}_D(\mathbf{c})$ (see Eqn. (7.55)).

The matrix $\mathbf{H}_{xy}(\mathbf{c})$ has two eigenvalues λ_1, λ_2 , which we define as being ordered, such that λ_1 has the greater magnitude ($|\lambda_1| \geq |\lambda_2|$). If both eigenvalues for a point \mathbf{c} are of similar magnitude, the function exhibits a high curvature along two orthogonal directions and in this case \mathbf{c} is likely to be a good reference point that can be located reliably. In the optimal situation (for example, near a corner), the ratio of the eigenvalues $\rho = \lambda_1/\lambda_2$ is close to one. Alternatively, if the ratio ρ is high it can be concluded that a single orientation dominates at this position, as is typically the case in the neighborhood of edges.

To estimate the ratio ρ it is not necessary to calculate the actual eigenvalues. Following the description in [80], the sum and product of the eigenvalues λ_1, λ_2 can be found as

$$\lambda_1 + \lambda_2 = \text{tr}(\mathbf{H}_{xy}(\mathbf{c})) = d_{xx} + d_{yy}, \quad (7.64)$$

$$\lambda_1 \cdot \lambda_2 = \det(\mathbf{H}_{xy}(\mathbf{c})) = d_{xx} \cdot d_{yy} - d_{xy}^2. \quad (7.65)$$

If the determinant $\det(\mathbf{H}_{xy})$ is *negative*, the principal curvatures of the underlying 2D function have opposite signs and thus point \mathbf{c} can be discarded as not being an extremum. Otherwise, with

$$\rho_{1,2} = \frac{\lambda_1}{\lambda_2} \quad (7.66)$$

being the ratio of the two eigenvalues (and thus $\lambda_1 = \rho_{1,2} \cdot \lambda_2$), the expression

$$\alpha = \frac{[\text{tr}(\mathbf{H}_{xy}(\mathbf{c}))]^2}{\det(\mathbf{H}_{xy}(\mathbf{c}))} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 \cdot \lambda_2} \quad (7.67)$$

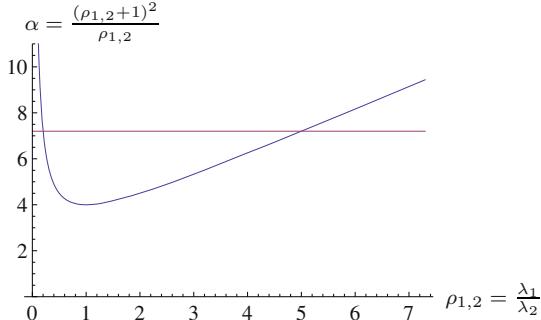


Figure 7.16 Limiting the ratio of principal curvatures (edge ratio). The quantity α (blue line) has a minimum when the eigenvalue ratio $\rho_{1,2} = \frac{\lambda_1}{\lambda_2}$ is one, i.e., when the two eigenvalues λ_1, λ_2 are equal, indicating a corner-like event. Line-like structures, where one of the eigenvalues is dominant, are characterized by greater $\rho_{1,2}$ and α values. For example, the principal curvature ratio $\rho_{1,2}$ is limited to 5 by setting $\alpha_{\max} = (5+1)^2/5 = 7.2$ (purple line).

$$= \frac{(\rho_{1,2} \cdot \lambda_2 + \lambda_2)^2}{\rho_{1,2} \cdot \lambda_2^2} = \frac{\lambda_2^2(\rho_{1,2} + 1)^2}{\rho_{1,2} \cdot \lambda_2^2} = \frac{(\rho_{1,2} + 1)^2}{\rho_{1,2}} \quad (7.68)$$

depends only on the ratio $\rho_{1,2}$. Thus, given that the determinant of \mathbf{H}_{xy} is positive, the quantity α has a minimum (4.0) at $\rho_{1,2} = 1$, if the two eigenvalues are equal (see Fig. 7.16). Note that the ratio α is the same for $\rho_{1,2} = \lambda_1/\lambda_2$ or $\rho_{1,2} = \lambda_2/\lambda_1$, since

$$\alpha = \frac{(\rho_{1,2} + 1)^2}{\rho_{1,2}} = \frac{\left(\frac{1}{\rho_{1,2}} + 1\right)^2}{\frac{1}{\rho_{1,2}}} . \quad (7.69)$$

To verify that the eigenvalue ratio $\rho_{1,2}$ at a given position \mathbf{c} is *below* a specified limit $\bar{\rho}_{1,2}$ (making \mathbf{c} a good candidate), it is thus sufficient to check the condition

$$\alpha \leq \alpha_{\max}, \quad \text{with} \quad \alpha_{\max} = \frac{(\bar{\rho}_{1,2} + 1)^2}{\bar{\rho}_{1,2}}, \quad (7.70)$$

without the need to actually calculate the individual eigenvalues λ_1 and λ_2 .²⁰ $\bar{\rho}_{1,2}$ should be greater than 1 and is typically chosen to be in the range $3, \dots, 10$ ($\bar{\rho}_{1,2} = 10$ is suggested in [80]). The resulting value of α_{\max} in Eqn. (7.70) is constant and only needs to be calculated once (see Alg. 7.3). Detection examples for varying values of $\bar{\rho}_{1,2}$ are shown in Fig. 7.17. Note that considerably more candidates appear near edges as $\bar{\rho}_{1,2}$ is raised from 3 to 40.

²⁰ Note that a similar trick is used in the *Harris* corner detection algorithm (see Vol. 2, Ch. 4 [21]).

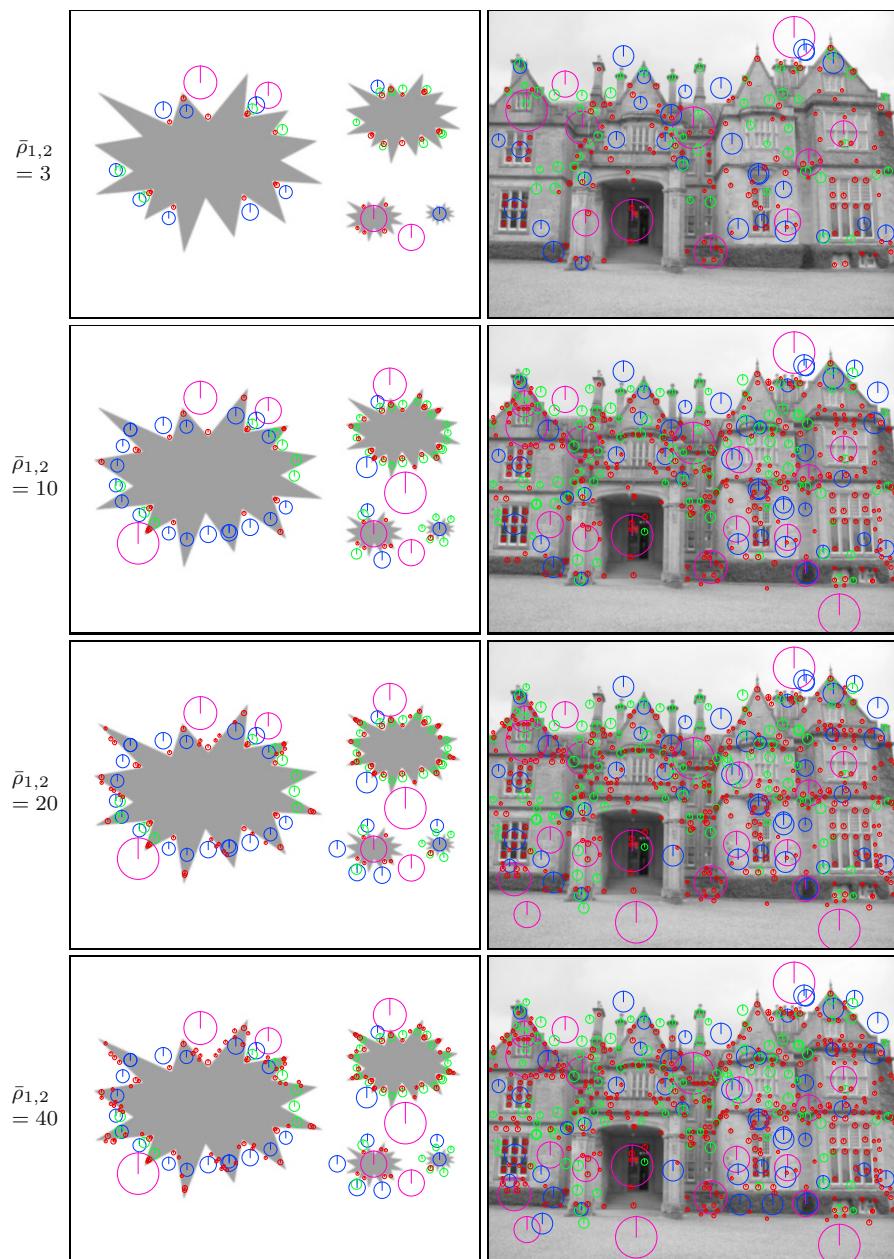


Figure 7.17 Rejection of edge-like features by controlling the max. curvature ratio $\bar{\rho}_{1,2}$. The size of the circles is proportional to the scale level at which the corresponding key point was detected, the color indicating the containing octave (0 = red, 1 = green, 2 = blue, 3 = magenta).

Table 7.3 Scale space-related symbols used in this chapter.

$\mathcal{G}(x, y, \sigma)$	continuous Gaussian scale space
\mathbf{G}	discrete Gaussian scale space
$\mathbf{G}_k = \mathbf{G}(k)$	discrete Gaussian scale space level
\mathbf{L}	LoG scale space
$\mathbf{L}_k = \mathbf{L}(k)$	LoG scale space level
\mathbf{D}	DoG scale space
$\mathbf{D}_k = \mathbf{D}(k)$	DoG scale space level
\mathbf{G}	hierarchical Gaussian scale space
$\mathbf{G}_p = \mathbf{G}(p)$	hierarchical Gaussian scale space octave
$\mathbf{G}_{p,q} = \mathbf{G}(p, q)$	hierarchical Gaussian scale space level
\mathbf{D}	hierarchical DoG scale space
$\mathbf{D}_p = \mathbf{D}(p)$	hierarchical DoG scale space octave
$\mathbf{D}_{p,q} = \mathbf{D}(p, q)$	hierarchical DoG scale space level
$\mathbf{k} = (p, q, u, v)$	discrete key point position in hierarchical scale space
$\mathbf{k}' = (p, q, x, y)$	continuous (refined) key point position in hierarchical scale space

7.3 Creating Local Descriptors

For each local maximum detected in the hierarchical DoG scale space, a candidate key point $\mathbf{k} = (p, q, u, v)$ is created, which is subsequently refined to a continuous position $\mathbf{k}' = (p, q, x, y)$, following the steps described above (see Eqns. (7.54–7.62)). Then, for each refined key point \mathbf{k}' , one or more (up to 4) local descriptors are calculated. This process involves the following steps:

1. Find the *dominant* orientation(s) of the key point \mathbf{k}' from the distribution of the gradients at the corresponding Gaussian scale space level.
2. For each dominant orientation, create a separate SIFT *descriptor* at the key point \mathbf{k}' .

7.3.1 Finding dominant orientations

Orientation from Gaussian scale space

Orientation vectors are obtained by sampling the *gradient* values of the hierarchical Gaussian scale space $\mathbf{G}_{p,q}(u, v)$ (see Eqn. (7.33)). For any lattice position (u, v) at octave p and scale level q , the local gradient is calculated as

$$\nabla_{p,q}(u, v) = \begin{pmatrix} A_{p,q}(u, v) \\ B_{p,q}(u, v) \end{pmatrix} = \frac{1}{2} \cdot \begin{pmatrix} \mathbf{G}_{p,q}(u+1, v) - \mathbf{G}_{p,q}(u-1, v) \\ \mathbf{G}_{p,q}(u, v+1) - \mathbf{G}_{p,q}(u, v-1) \end{pmatrix}. \quad (7.71)$$

From these gradient vectors, the gradient *magnitude* and *orientation* (i. e., polar coordinates) is obtained as

$$R_{p,q}(u, v) = \|\nabla_{p,q}(u, v)\| = \sqrt{A_{p,q}^2(u, v) + B_{p,q}^2(u, v)}, \quad (7.72)$$

$$\phi_{p,q}(u, v) = \angle \nabla_{p,q}(u, v) = \tan^{-1}\left(\frac{B_{p,q}(u, v)}{A_{p,q}(u, v)}\right). \quad (7.73)$$

In practice, gradient magnitude and orientation fields are usually pre-calculated for all relevant levels of the Gaussian scale space \mathbf{G} .

Orientation histograms

To find the dominant orientations for a given keypoint, a histogram \mathbf{h}_ϕ of the orientation angles is calculated for the gradient vectors collected from a square window around the key point center. Typically the histogram has $n_{\text{orient}} = 36$ bins, i. e., the angular resolution is 10° . The orientation histogram is collected from a square region using an isotropic Gaussian weighting function whose width σ_w is proportional to the *decimated scale* $\dot{\sigma}_q$ (see Eqn. (7.38)) of the keypoints's scale level q . Typically a Gaussian weighting function “with a σ that is 1.5 times that of the scale of the keypoint” [80] is used, i. e.,

$$\sigma_w = 1.5 \cdot \dot{\sigma}_q = 1.5 \cdot \sigma_0 \cdot 2^{q/Q}. \quad (7.74)$$

Note that σ_w is independent of the octave index p and thus the same weighting functions are used in each octave. To calculate the *orientation histogram*, the Gaussian gradients around the given key point are collected from a square region of size $2r_w \times 2r_w$, with

$$r_w = \lceil 2.5 \cdot \sigma_w \rceil \quad (7.75)$$

to avoid truncation effects. For the parameters listed in [Table 7.2](#) ($\sigma_0 = 1.6$, $Q = 3$), the values for σ_w (expressed in the octave's coordinate units) are

q	0	1	2	3
σ_w	1.6000	2.0159	2.5398	3.2000
r_w	4	5	6	7

.

In [Alg. 7.7](#), σ_w and r_w of the Gaussian weighting function are calculated in lines 7 and 8, respectively. At each lattice point (u, v) , the gradient vector $\nabla_{p,q}(u, v)$ is calculated in octave p and level q of the Gaussian scale space \mathbf{G} ([Alg. 7.7](#), line 15). From this, the gradient magnitude $E_{p,q}(u, v)$ and orientation $\phi_{p,q}(u, v)$ are obtained (lines 27–28). The corresponding Gaussian weight is calculated (in line 16) from the spatial distance between the grid point (u, v) and the interest point (x, y) as

$$w_G(u, v) = e^{-\frac{(u-x)^2+(v-y)^2}{2 \cdot \sigma_w^2}}. \quad (7.76)$$

For the grid point (u, v) , the quantity to be accumulated into the orientation histogram is

$$z = R_{p,q}(u, v) \cdot w_G(u, v), \quad (7.77)$$

i.e., the local gradient magnitude weighted by the Gaussian window function ([Alg. 7.7](#), line 17). The orientation histogram \mathbf{h}_ϕ consists of n_{orient} bins and thus the *continuous* bin number for the angle $\phi(u, v)$ is

$$\kappa_\phi = n_{\text{orient}} \cdot \frac{\phi(u, v)}{2\pi} \quad (7.78)$$

(see [Alg. 7.7](#), line 18). To collect the *continuous* orientations into a histogram with discrete bins, quantization must be performed. The simplest approach is to select the “nearest” bin (by rounding) and to add the associated quantity (denoted z) entirely to the selected bin. Alternatively, to reduce quantization effects, a common technique is to *split* the quantity z onto the two closest bins. Given the continuous bin value κ_ϕ , the indexes of the two closest discrete bins are

$$k_0 = \lfloor \kappa_\phi \rfloor \bmod n_{\text{orient}} \quad \text{and} \quad k_1 = (\lfloor \kappa_\phi \rfloor + 1) \bmod n_{\text{orient}}, \quad (7.79)$$

respectively. The quantity z is then partitioned and accumulated into the adjacent bins k_0, k_1 of the orientation histogram \mathbf{h}_ϕ in the form

$$\mathbf{h}_\phi(k_0) \leftarrow \mathbf{h}_\phi(k_0) + (1 - \alpha) \cdot z \quad \text{and} \quad \mathbf{h}_\phi(k_1) \leftarrow \mathbf{h}_\phi(k_1) + \alpha \cdot z, \quad (7.80)$$

with $\alpha = \kappa_\phi - \lfloor \kappa_\phi \rfloor$ (see [Alg. 7.7](#), lines 19–23). This process is illustrated in [Fig. 7.18](#) and by the example in [Fig. 7.19](#), showing a three-dimensional orientation histogram \mathbf{h}_ϕ with 36 bins.

Orientation histogram smoothing.

Before calculating the dominant orientations, the raw orientation histogram \mathbf{h}_ϕ is usually smoothed by applying a (circular) low-pass filter, typically a simple 3-tap Gaussian or box-type filter (see procedure `SMOOTHCIRCULAR()` in [Alg. 7.7](#), lines 6–15).²¹ Stronger smoothing is achieved by applying the filter multiple times, as illustrated in [Fig. 7.20](#). In practice, 2–3 smoothing iterations appear to be sufficient.

²¹ Histogram smoothing is not mentioned in the original SIFT publication [80] but used in most implementations.

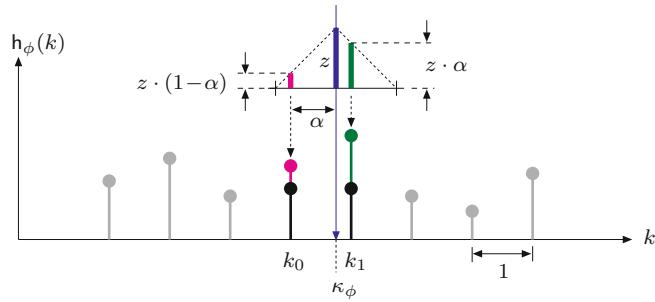


Figure 7.18 Accumulating into multiple histogram bins by linear interpolation. Assume that some quantity z (blue bar) is to be added to the *discrete* histogram h_ϕ at the *continuous* position κ_ϕ . The histogram bins adjacent to κ_ϕ are $k_0 = \lfloor \kappa_\phi \rfloor$ and $k_1 = \lfloor \kappa_\phi \rfloor + 1$. The fraction of z accumulated into bin k_1 is $z_1 = z \cdot \alpha$ (green bar), with $\alpha = \kappa_\phi - k_0$. Analogously, the quantity added to bin k_0 is $z_0 = z \cdot (1 - \alpha)$ (magenta bar).

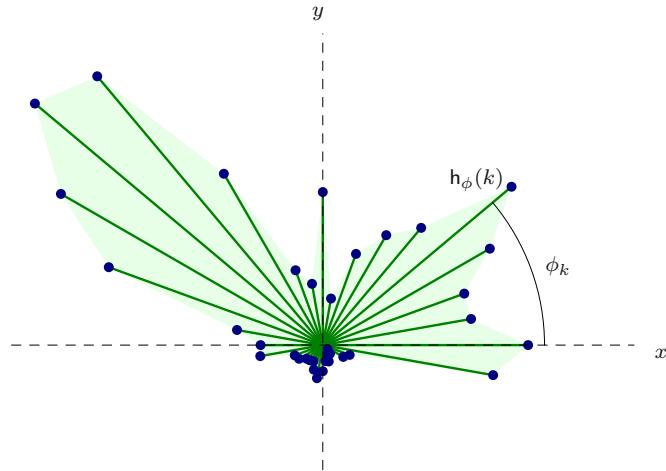


Figure 7.19 Orientation histogram example. Each of the 36 radial bars corresponds to one entry in the orientation histogram h_ϕ . The *length* (radius) of each radial bar with index k is proportional to the accumulated value in the corresponding bin $h_\phi(k)$ and its orientation is ϕ_k .

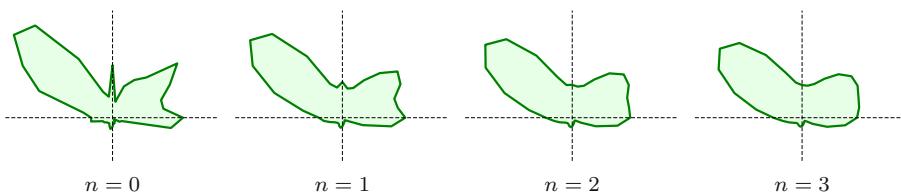


Figure 7.20 Smoothing the orientation histogram by repeatedly applying a circular low-pass filter with the kernel $\frac{1}{4} \cdot (1, 2, 1)$. For $n = 0$, the figure shows the original (unsmoothed) orientation histogram, identical to Fig. 7.19.

Locating and interpolating orientation peaks.

After smoothing the orientation histogram, the next step is to detect the peak entries in \mathbf{h}_ϕ . A bin k is considered a significant orientation peak if $\mathbf{h}_\phi(k)$ is a local maximum and its value is not less than a certain fraction of the maximum histogram entry, that is, only if

$$\mathbf{h}_\phi(k) > \mathbf{h}_\phi((k \pm 1) \bmod n_{\text{orient}}) \quad \wedge \quad \mathbf{h}_\phi(k) > t_{\text{domor}} \cdot \max_k \mathbf{h}_\phi(k), \quad (7.81)$$

with $t_{\text{domor}} = 80\%$ as a typical limit.

To achieve a finer angular resolution than provided by the orientation histogram bins (typically spaced at 10° steps) alone, a continuous peak orientation is calculated by quadratic interpolation of the neighboring histogram values. Given a discrete peak index k , the interpolated (continuous) peak position \check{k} is obtained by fitting a quadratic function to the three successive histogram values $\mathbf{h}_\phi(k-1)$, $\mathbf{h}_\phi(k)$, $\mathbf{h}_\phi(k+1)$ as²²

$$\check{k} = k + \frac{\mathbf{h}_\phi(k-1) - \mathbf{h}_\phi(k+1)}{2 \cdot (\mathbf{h}_\phi(k-1) - 2\mathbf{h}_\phi(k) + \mathbf{h}_\phi(k+1))}, \quad (7.82)$$

with all indexes taken modulo n_{orient} . From Eqn. (7.78), the (continuous) dominant orientation angle $\theta \in [0, 2\pi)$ is then obtained as

$$\theta = (\check{k} \bmod n_{\text{orient}}) \cdot \frac{2\pi}{n_{\text{orient}}}. \quad (7.83)$$

In this way, the dominant orientation can be estimated with accuracy much beyond the coarse resolution of the orientation histogram. Note that, in some cases, multiple histogram peaks are obtained for a given key point (see procedure FINDPEAKORIENTATIONS() in [Alg. 7.6](#), lines 17–29). In this event, individual SIFT descriptors are created for each dominant orientation at the same key point position.

[Figure 7.21](#) shows the orientation histograms for a set of detected key points in two different images after applying a varying number of smoothing steps. It also shows the interpolated dominant orientations calculated from the orientation histograms, as described above.

7.3.2 Descriptor formation

For each key point $\mathbf{k}' = (p, q, x, y)$ and each dominant orientation θ , a corresponding SIFT descriptor is obtained by sampling the surrounding gradients at octave p and level q of the Gaussian scale space \mathbf{G} .

²² See Appendix B.5.2 for details.

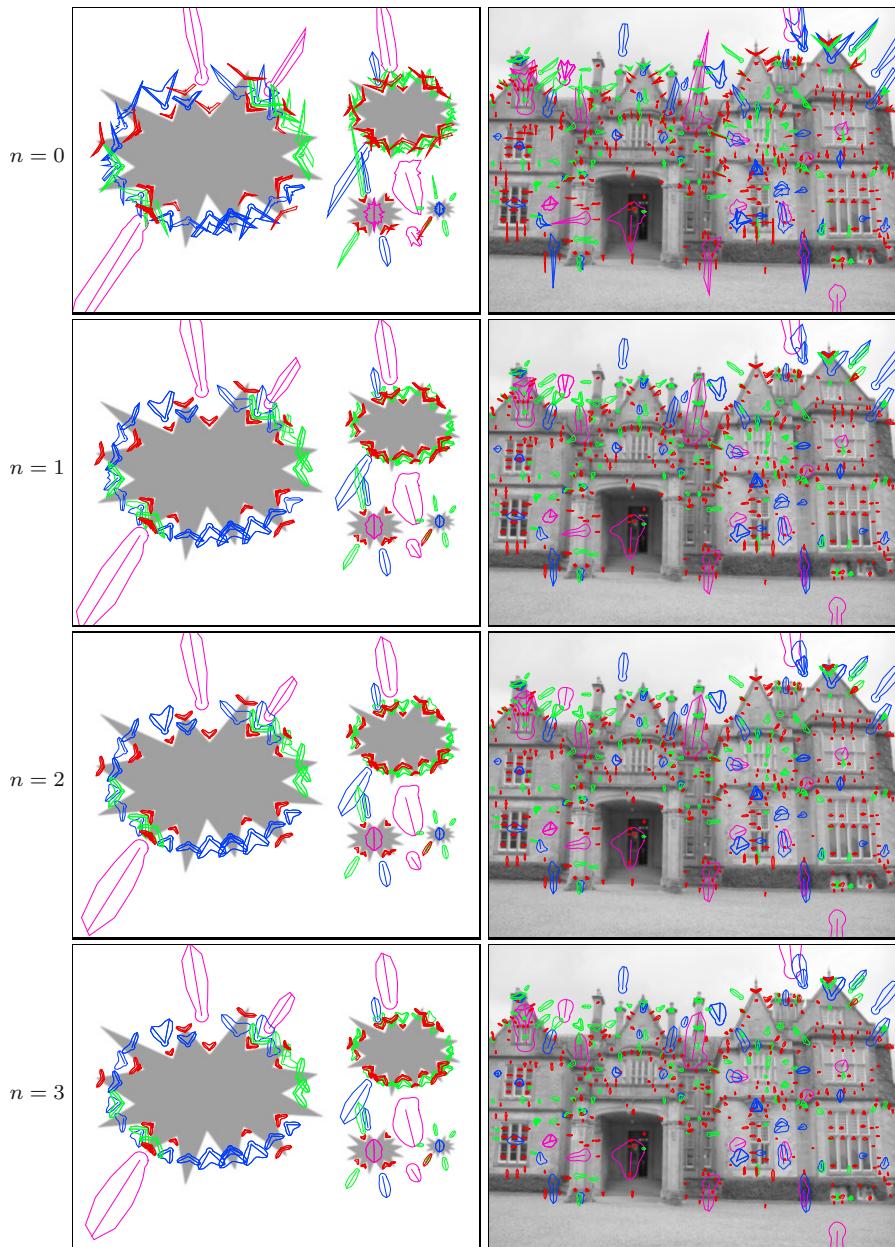


Figure 7.21 Orientation histograms and dominant orientations (examples). $n = 0, \dots, 3$ smoothing iterations were applied to the orientation histograms. The (interpolated) dominant orientations are shown as radial lines that emanate from each feature's center point. The size of the histogram graphs is proportional to the absolute scale ($\sigma_{p,q}$, see Table 7.2) at which the corresponding key point was detected. The colors indicate the index of the containing scale space octave: $p = 0$ (red), 1 (green), 2 (blue), 3 (magenta).

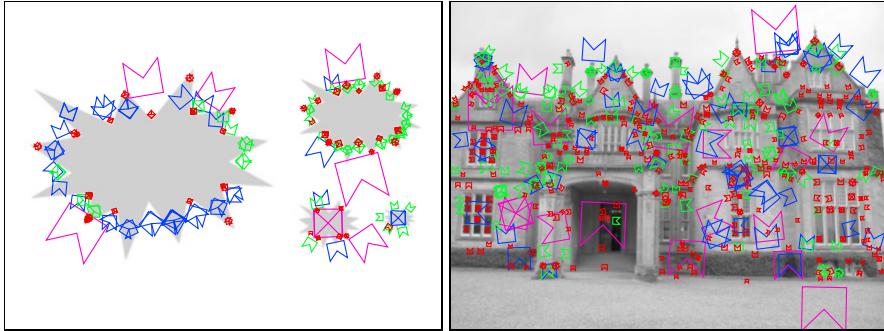


Figure 7.22 Marked key points aligned to their dominant orientation. The size of the markers is proportional to the absolute scale ($\sigma_{p,q}$, see Table 7.2) at which the corresponding key point was detected. Note that multiple feature instances are inserted at key point positions with more than one dominant orientation. The colors indicate the index of the scale space containing octave: $p = 0$ (red), 1 (green), 2 (blue), 3 (magenta).

Descriptor geometry

The geometry underlying the calculation of SIFT descriptors is illustrated in Fig. 7.23. The descriptor combines the gradient orientations over a square region of size $w_d \times w_d$ that is partitioned into $n_{\text{spat}} \times n_{\text{spat}}$ sub-squares (typ. $n_{\text{spat}} = 4$, see Table 7.5). To achieve rotation invariance, the descriptor region is aligned to the key point's dominant orientation θ , as determined in the previous steps. To make the descriptor invariant to scale changes, its size w_d (expressed in the grid coordinate units of octave p) is set proportional to the key point's *decimated scale* $\dot{\sigma}_q$ (see Eqn. (7.38)), that is,

$$w_d = s_d \cdot \dot{\sigma}_q = s_d \cdot \sigma_0 \cdot 2^{q/Q}, \quad (7.84)$$

where s_d is a constant size factor. For $s_d = 10$ (see Table 7.5), the descriptor size w_d ranges from 16.0 (at level 0) to 25.4 (at level 2), as listed in Table 7.4. Note that the descriptor size w_d only depends on the scale level index q and is independent of the octave index p . Thus the same descriptor geometry applies to all octaves of the scale space.

The descriptor's *spatial resolution* is specified by the parameter n_{spat} . Typically $n_{\text{spat}} = 4$ (as shown in Fig. 7.23) and thus the total number of spatial bins is $n_{\text{spat}} \times n_{\text{spat}} = 16$ (in this case). Each spatial descriptor bin relates to an area of size $(w_d/n_{\text{spat}}) \times (w_d/n_{\text{spat}})$. For example, at scale level $q = 0$ of any octave, $\dot{\sigma}_0 = 1.6$ and the corresponding descriptor size is $w_d = s_d \cdot \dot{\sigma}_0 = 10 \cdot 1.6 = 16.0$ (see Table 7.4). In this case (illustrated in Fig. 7.24), the descriptor covers 16×16 gradient samples, as suggested in [80].

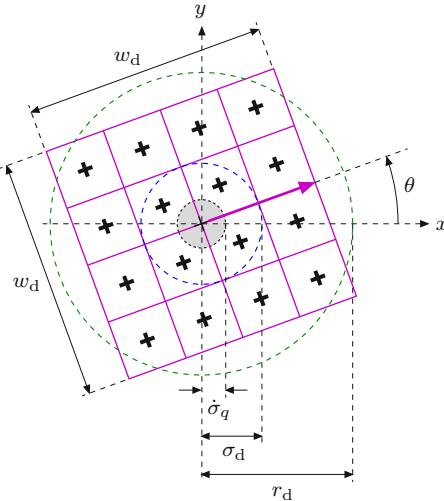


Figure 7.23 Geometry of a SIFT descriptor. The descriptor is calculated from a square support region that is centered at the keypoint’s position (x, y) , aligned to the key point’s dominant orientation θ , and partitioned into $n_{\text{spat}} \times n_{\text{spat}}$ (4×4) sub-squares. The side length of the descriptor is set to $w_d = 10 \cdot \dot{\sigma}_q$, where $\dot{\sigma}_q$ denotes the keypoint’s decimated scale (radius of the inner circle). It depends on the key point’s scale level q (see Table 7.4). The descriptor covers a region of size $w_d \times w_d$. The contribution of each gradient sample is attenuated by a circular Gaussian function of width $\sigma_d = 0.25 \cdot w_d$ (blue circle). The weights drop off radially and are practically zero at $r_d = 2.5 \cdot \sigma_d$ (green circle). Thus only samples within the outer (green) circle need to be included for calculating the descriptor statistics.

Table 7.4 SIFT descriptor dimensions for different scale levels q (for size factor $s_d = 10$ and $Q = 3$ levels per octave). $\dot{\sigma}_q$ is the keypoint’s decimated scale, w_d is the descriptor size, σ_d is the width of the Gaussian weighting function, and r_d is the radius of the descriptor’s support region. For $Q = 3$, only scale levels $q = 0, 1, 2$ are relevant. All lengths are expressed in the octave’s (i.e., decimated) coordinate units.

q	$\dot{\sigma}_q$	$w_d = s_d \cdot \dot{\sigma}_q$	$\sigma_d = 0.25 \cdot w_d$	$r_d = 2.5 \cdot \sigma_d$
3	3.2000	32.000	8.0000	20.0000
2	2.5398	25.398	6.3495	15.8738
1	2.0159	20.159	5.0398	12.5994
0	1.6000	16.000	4.0000	10.0000
-1	1.2699	12.699	3.1748	7.9369

Gradient features

The actual SIFT descriptor is a feature vector obtained by histogramming the gradient orientations of the Gaussian scale level within the descriptors spatial support region. This requires a three-dimensional histogram $h_V(i, j, k)$, with two spatial dimensions (i, j) for the $n_{\text{spat}} \times n_{\text{spat}}$ sub-regions and one additional dimension (k) for n_{angl} gradient orientations. This histogram thus contains

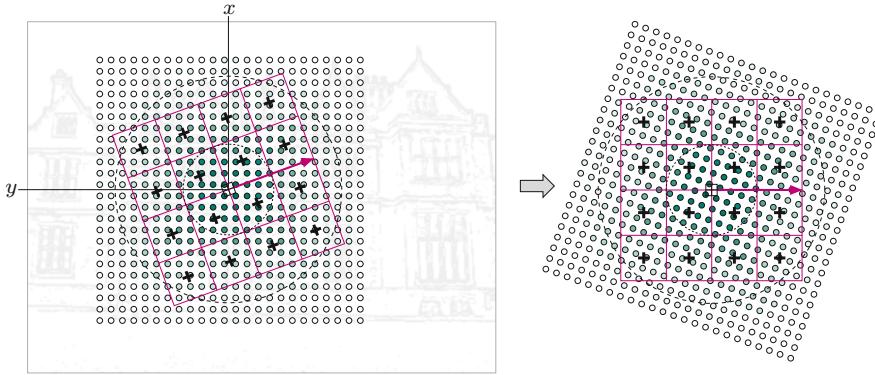


Figure 7.24 SIFT descriptor region placed over discrete lattice points at scale level $q = 0$. In this case, the decimated scale is $\dot{\sigma}_0 = 1.6$ and the width of the descriptor is $w_d = s_d \cdot \dot{\sigma}_0 = 10 \cdot 1.6 = 16.0$. Each circle represents a gradient sample in the Gaussian scale space at the key point's scale level q . At level 0 the descriptor thus covers 16×16 scale space samples. Assignment to the corresponding histogram bins is done by rotating each gradient sample position by the dominant orientation angle θ to the canonical coordinate frame shown on the right.

$n_{\text{spat}} \times n_{\text{spat}} \times n_{\text{angl}}$ bins.

Figure 7.25 illustrates this structure for the typical setup, with $n_{\text{spat}} = 4$ and $n_{\text{angl}} = 8$ (see Table 7.5). In this case, eight orientation bins are provided for each of the 16 spatial bins $A1, \dots, D4$, which makes a total of 128 histogram bins.

For a given key point $k' = (p, q, x, y)$, the histogram h_∇ accumulates the orientations (angles) of the gradients at the Gaussian scale space level $\mathbf{G}_{p,q}$ within the support region around the center point (x, y) . For each sample point (u, v) in this region, the gradient vector of the Gaussian ∇_G is estimated as

$$\nabla_G(u, v) = \begin{pmatrix} d_x \\ d_y \end{pmatrix} = 0.5 \cdot \begin{pmatrix} \mathbf{G}_{p,q}(u+1, v) - \mathbf{G}_{p,q}(u-1, v) \\ \mathbf{G}_{p,q}(u, v+1) - \mathbf{G}_{p,q}(u, v-1) \end{pmatrix}, \quad (7.85)$$

from which the gradient magnitude E and orientation ϕ are determined as

$$R(u, v) = \sqrt{d_x^2 + d_y^2} \quad \text{and} \quad \phi(u, v) = \tan^{-1}\left(\frac{d_y}{d_x}\right), \quad (7.86)$$

respectively (see Alg. 7.7 on p. 281, lines 25–29).²³ Each gradient sample contributes to the gradient histogram h_∇ a particular quantity z that depends on the gradient magnitude R and the distance of the sample point (u, v) from

²³ For efficiency reasons, $R(u, v)$ and $\phi(u, v)$ are typically pre-calculated for all relevant scale levels.

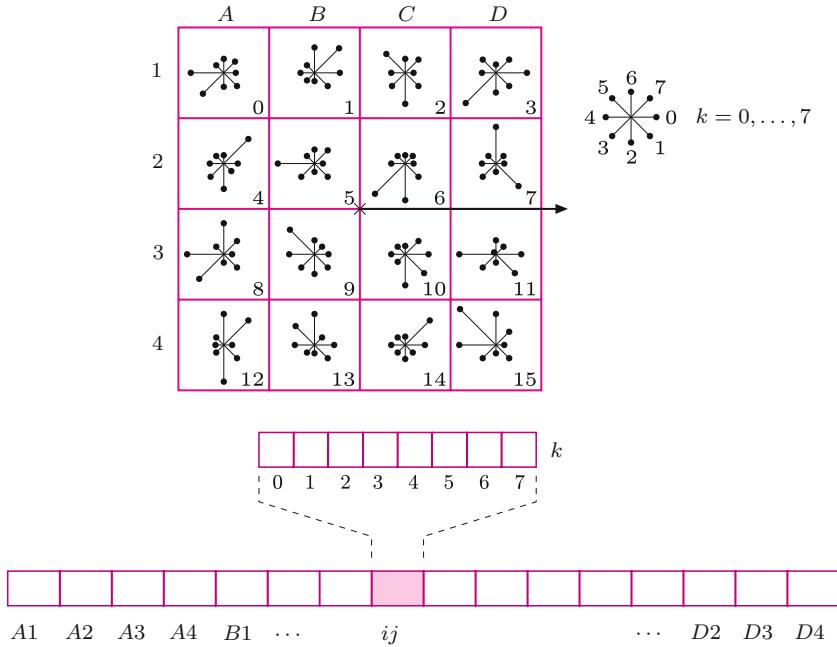


Figure 7.25 SIFT descriptor structure for $n_{\text{spat}} = 4$ and $n_{\text{angl}} = 8$. Eight orientation bins $k = 0, \dots, 7$ are provided for each of the 16 spatial bins $ij = A1, \dots, D4$. Thus the gradient histogram h_V holds 128 bins in total, which is also the size of the final feature vector.

the key point's center (x, y) . A Gaussian weighting function is used to attenuate samples with increasing distance, such that the accumulated quantity is

$$z(u, v) = R(u, v) \cdot w_G = R(u, v) \cdot e^{-\frac{(u-x)^2 + (v-y)^2}{2\sigma_d^2}}. \quad (7.87)$$

The width σ_d of this weighting function is proportional to the size of the descriptor region, that is,

$$\sigma_d = 0.25 \cdot w_d = 0.25 \cdot s_d \cdot \dot{\sigma}_q. \quad (7.88)$$

The weighting function drops off radially from the center and is practically zero at distance $r_d = 2.5 \cdot \sigma_d$. Therefore, only gradient samples that are closer to the key point's center than r_d (green circle in Fig. 7.23) need to be considered in the gradient histogram calculation (see Alg. 7.8, lines 7, 15). For a given key point $\mathbf{k}' = (p, q, x, y)$, sampling of the Gaussian gradients can thus be confined to the grid points (u, v) inside the square region bounded by $x \pm r_d$ and $y \pm r_d$ (see Alg. 7.8, lines 8–9, 13–14). Each sample point (u, v) is then subjected to the affine transformation

$$\begin{pmatrix} u' \\ v' \end{pmatrix} \leftarrow \frac{1}{w_d} \cdot \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix} \cdot \begin{pmatrix} u-x \\ v-y \end{pmatrix}, \quad (7.89)$$

which performs a rotation by the dominant orientation θ and maps the original (rotated) square of size $w_d \times w_d$ to the unit square with coordinates $u', v' \in [-0.5, +0.5]$ (see Fig. 7.24).

To make feature vectors rotation-invariant, the individual gradient orientations $\phi(u, v)$ (Eqn. (7.86)) are rotated by

$$\phi'(u, v) = (\phi(u, v) - \theta) \bmod 2\pi, \quad (7.90)$$

such that $\phi'(u, v) \in [0, 2\pi]$, to align with the reference orientation θ . For each gradient sample, with the continuous coordinates (u', v', ϕ') , the corresponding quantity $z(u, v)$ (Eqn. (7.87)) is accumulated into the three-dimensional gradient histogram h_{∇} . For a complete description of this step see procedure UPDATEGRADIENTHISTOGRAM() in Alg. 7.9. It first maps the coordinates (u', v', ϕ') (see Eqn. (7.89)) to the continuous histogram position (i', j', k') by

$$\begin{aligned} i' &= n_{\text{spat}} \cdot u' + 0.5 \cdot (n_{\text{spat}} - 1), \\ j' &= n_{\text{spat}} \cdot v' + 0.5 \cdot (n_{\text{spat}} - 1), \\ k' &= \phi' \cdot \frac{n_{\text{angl}}}{2\pi}, \end{aligned} \quad (7.91)$$

such that $i', j' \in [-0.5, n_{\text{spat}} - 0.5]$ and $k' \in [0, n_{\text{angl}}]$.

Analogous to inserting into a continuous position of a one-dimensional histogram by linear interpolation over *two* bins (see Fig. 7.18), the quantity z is distributed over *eight* neighboring histogram bins by *tri-linear* interpolation. The quantiles of z contributing to the individual histogram bins are determined by the distances of the coordinates (i', j', k') from the discrete indexes (i, j, k) of the affected histogram bins. The bin indexes are found as the possible combinations (i, j, k) , with $i \in \{i_0, i_1\}$, $j \in \{j_0, j_1\}$, $k \in \{k_0, k_1\}$, and

$$\begin{aligned} i_0 &= \lfloor i' \rfloor, & i_1 &= (i_0 + 1), \\ j_0 &= \lfloor j' \rfloor, & j_1 &= (j_0 + 1), \\ k_0 &= \lfloor k' \rfloor \bmod n_{\text{angl}}, & k_1 &= (k_0 + 1) \bmod n_{\text{angl}}. \end{aligned} \quad (7.92)$$

The corresponding quantiles (weights) are

$$\begin{aligned} \alpha_0 &= \lfloor i' \rfloor + 1 - i' = i_1 - i', & \alpha_1 &= 1 - \alpha_0, \\ \beta_0 &= \lfloor j' \rfloor + 1 - j' = j_1 - j', & \beta_1 &= 1 - \beta_0, \\ \gamma_0 &= \lfloor k' \rfloor + 1 - i', & \gamma_1 &= 1 - \gamma_0, \end{aligned} \quad (7.93)$$

and the 8 affected bins of the gradient histogram h_{∇} are finally updated as

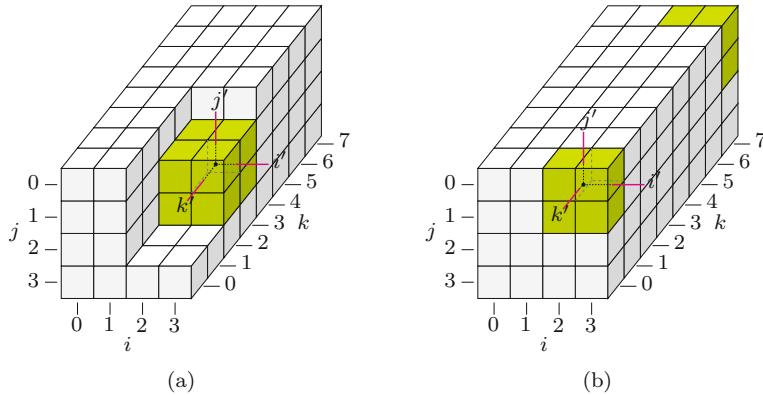


Figure 7.26 Structure of the three-dimensional gradient histogram, with $n_{\text{spat}} \times n_{\text{spat}} = 4 \times 4$ bins for the spatial dimensions (i, j) and $n_{\text{angl}} = 8$ bins along the orientation axis (k) . For the histogram to accumulate a quantity z into some continuous position (i', j', k') , eight adjacent bins receive different quantiles of z that are determined by tri-linear interpolation (a). Note that the bins along the orientation axis ϕ are treated cyclically, e.g., bins at $k = 0$ are also considered adjacent to the bins at $k = 7$ (b).

$$\begin{aligned}
 h_{\nabla}(i_0, j_0, k_0) &\leftarrow z \cdot \alpha_0 \cdot \beta_0 \cdot \gamma_0, \\
 h_{\nabla}(i_1, j_0, k_0) &\leftarrow z \cdot \alpha_1 \cdot \beta_0 \cdot \gamma_0, \\
 h_{\nabla}(i_0, j_1, k_0) &\leftarrow z \cdot \alpha_0 \cdot \beta_1 \cdot \gamma_0, \\
 &\vdots && \vdots \\
 h_{\nabla}(i_1, j_1, k_1) &\leftarrow z \cdot \alpha_1 \cdot \beta_1 \cdot \gamma_1.
 \end{aligned} \tag{7.94}$$

Of course, care must be taken that all indexes are within the bounds of the discrete histogram (see [Alg. 7.9](#), lines 17–22). Also, attention must be paid to the fact that the coordinate k represents an orientation and must therefore be treated in a cyclic manner, as illustrated in [Fig. 7.26](#) (also see [Alg. 7.9](#), lines 9–10). For each histogram bin, the range of contributing gradient samples covers half of each neighboring bin, i.e., the support regions of neighboring bins overlap, as illustrated in [Fig. 7.27](#).

SIFT descriptors

The elements of the gradient histogram h_{∇} are the raw material for the SIFT feature vectors f_{sift} . The process of calculating the feature vectors from the gradient histogram is described in [Alg. 7.10](#) (see procedure `MAKESIFTFEATUREVECTOR()`). Initially, the three-dimensional gradient histogram h_{∇} (which contains continuous values) of size $n_{\text{spat}} \times n_{\text{spat}} \times n_{\text{angl}}$ is flattened to a one-dimensional vector f of length $n_{\text{spat}}^2 \cdot n_{\text{angl}}$ (typ. 128), with

$$f((i \cdot n_{\text{spat}} + j) \cdot n_{\text{angl}} + k) = h_{\nabla}(i, j, k), \tag{7.95}$$

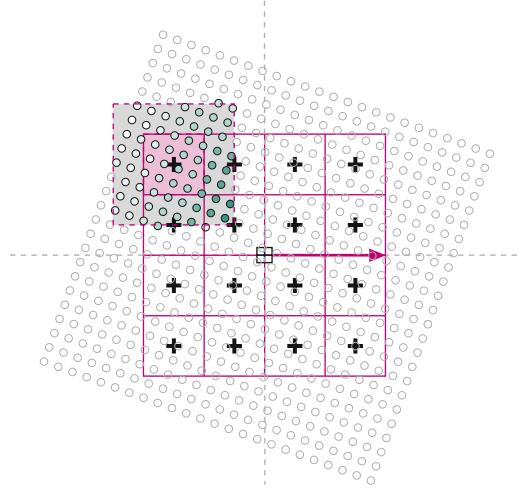


Figure 7.27 Support regions for gradient histogram bins overlap. The range of contributing gradient samples (small circles inside the shaded square) covers half of each neighboring bin along each direction, due to tri-linear interpolation. The shading of the circles indicates the weight w_G assigned to each sample by the Gaussian weighting function, whose value depends on the distance of each sample from the key point’s center (see Eqn. (7.87)).

for $i, j = 0, \dots, n_{\text{spat}} - 1$ and $k = 0, \dots, n_{\text{angl}} - 1$. The elements in \mathbf{f} are thus arranged in the same order as shown in Fig. 7.25, with the orientation index k being the fastest moving and the spatial index i being the slowest (see Alg. 7.10, lines 3–8).²⁴

Changes in image contrast have a linear impact upon the gradient magnitude and thus also upon the values of the feature vector \mathbf{f} . To eliminate these effects, the vector \mathbf{f} is subsequently normalized to

$$\mathbf{f}(m) \leftarrow \frac{1}{\|\mathbf{f}\|} \cdot \mathbf{f}(m), \quad (7.96)$$

for all m , such that \mathbf{f} has unit norm (see Alg. 7.10, line 9). Since the gradient is calculated from local pixel differences, changes in absolute brightness do not affect the gradient magnitude, unless saturation occurs. Such non-linear illumination changes tend to produce peak gradient values, which are compensated for by clipping the values of \mathbf{f} to a predefined maximum t_{fclip} , that is,

$$\mathbf{f}(m) = \min(\mathbf{f}(m), t_{\text{fclip}}), \quad (7.97)$$

with typically $t_{\text{fclip}} = 0.2$, as suggested in [80] (see Alg. 7.10, line 10). After this step, \mathbf{f} is normalized once again, as in Eqn. (7.96). Finally, the real-valued

²⁴ Note that different ordering schemes for arranging the elements of the feature vector are used in various SIFT implementations. For successful matching, the ordering of the elements must be identical, of course.

feature vector \mathbf{f} is converted to an integer-valued vector by

$$\mathbf{f}_{\text{sift}}(m) = \min(\text{round}(\mathsf{s}_{\text{fscale}} \cdot \mathbf{f}(m)), 255), \quad (7.98)$$

with $\mathsf{s}_{\text{fscale}}$ being a predefined constant (typ. $\mathsf{s}_{\text{fscale}} = 512$). The elements of \mathbf{f}_{sift} are in the range $[0, 255]$ to be conveniently encoded and stored as a byte sequence (see [Alg. 7.10](#), line 12). The final SIFT descriptor for a given key point $\mathbf{k}' = (p, q, x, y)$ is a tuple

$$\mathbf{s} = \langle x', y', \sigma, \theta, \mathbf{f}_{\text{sift}} \rangle, \quad (7.99)$$

which contains the key point's interpolated position x', y' (in original image coordinates), the absolute scale σ , its dominant orientation θ , and the corresponding integer-valued gradient feature vector \mathbf{f}_{sift} (see [Alg. 7.8](#) on 282, line 25).

Remember that multiple SIFT descriptors may be produced for different dominant orientations located at the same key point position. These will have the same position and scale values but different θ and \mathbf{f}_{sift} data.

7.4 SIFT algorithm summary

This section contains a collection of algorithms that summarizes the SIFT feature extraction process described in the previous sections of this chapter.

[Algorithm 7.3](#) shows the top-level procedure `GETSIFTFEATURES(I)`, which returns a sequence of SIFT feature descriptors for the given image I . The remaining parts of [Alg. 7.3](#) describe the key point detection as extrema of the DoG scale space. The refinement of key point positions is covered in [Alg. 7.4](#). [Algorithm 7.5](#) contains the procedures used for neighborhood operations, detecting local extrema, and the calculation of the gradient and Hessian matrix in 3D. [Algorithm 7.6](#) covers the operations related to finding the dominant orientations at a given key point location, based on the orientation histogram that is calculated in [Alg. 7.7](#). The final formation of the SIFT descriptors is described in [Alg. 7.8](#), which is based on the procedures defined in [Algs. 7.9–7.10](#). The global constants used throughout these algorithms are listed in [Table 7.5](#), together with the corresponding Java identifiers in the associated source code (see Sec. 7.7).

7.5 Matching SIFT Features

Most applications of SIFT features aim at locating corresponding interest points in two or more images of the same scene, for example, for matching stereo pairs, panorama stitching, or feature tracking. Other applications like

Algorithm 7.3 SIFT feature extraction (Part 1). Top-level SIFT procedure. Global parameters: σ_s , σ_0 , t_{mag} , Q , P (see Table 7.5).

```

1: GETSIFTFEATURES( $I$ )
   Input:  $I$ , the source image (scalar-valued).
   Returns a sequence of SIFT feature descriptors detected in  $I$ .
2:  $\langle \mathbf{G}, \mathbf{D} \rangle \leftarrow \text{BUILDSCALESPACE}(I, \sigma_s, \sigma_0, P, Q)$            ▷ Alg. 7.2
3:  $C \leftarrow \text{GETKEYPOINTS}(\mathbf{D})$ 
4:  $S \leftarrow ()$                                 ▷ empty list of SIFT descriptors
5: for all  $k' \in C$  do                      ▷  $k' = (p, q, x, y)$ 
6:    $A \leftarrow \text{GETDOMINANTORIENTATIONS}(\mathbf{G}, k')$            ▷ Alg. 7.6
7:   for all  $\theta \in A$  do
8:      $s \leftarrow \text{MAKESIFTDESCRIPTOR}(\mathbf{G}, k', \theta)$            ▷ Alg. 7.8
9:      $S \leftarrow S \cup (s)$ 
10:  return  $S$ 

11: GETKEYPOINTS( $\mathbf{D}$ )
     $\mathbf{D}$ : DoG scale space (with  $P$  octaves, each containing  $Q$  levels).
    Returns a set of keypoints located in  $\mathbf{D}$ .
12:  $C \leftarrow ()$                                 ▷ empty list of key points
13: for  $p \leftarrow 0, \dots, P-1$  do           ▷ for all octaves  $p$ 
14:   for  $q \leftarrow 0, \dots, Q-1$  do           ▷ for all scale levels  $q$ 
15:      $E \leftarrow \text{FINDEXTREMA}(\mathbf{D}, p, q)$ 
16:     for all  $k \in E$  do                  ▷  $k = (p, q, u, v)$ 
17:        $k' \leftarrow \text{REFINEKEYPOSITION}(\mathbf{D}, k)$            ▷ Alg. 7.4
18:       if  $k' \neq \text{nil}$  then          ▷  $k' = (p, q, x, y)$ 
19:          $C \leftarrow C \cup (k')$            ▷ add refined key point  $k'$ 
20:  return  $C$ 

21: FINDEXTREMA( $\mathbf{D}, p, q$ )
22:  $\mathbf{D}_{p,q} \leftarrow \text{GETSCALELEVEL}(\mathbf{D}, p, q)$ 
23:  $(M, N) \leftarrow \text{SIZE}(\mathbf{D}_{p,q})$ 
24:  $E \leftarrow ()$                                 ▷ empty list of extrema
25: for  $u \leftarrow 1, \dots, M-2$  do
26:   for  $v \leftarrow 1, \dots, N-2$  do
27:     if  $|\mathbf{D}_{p,q}(u, v)| > t_{mag}$  then
28:        $k \leftarrow (p, q, u, v)$ 
29:        $N \leftarrow \text{GETNEIGHBORHOOD}(\mathbf{D}, k)$            ▷ Alg. 7.5
30:       if  $\text{ISEXTREMUM}(N)$  then          ▷ Alg. 7.5
31:          $E \leftarrow E \cup (k)$            ▷ add  $k$  to  $E$ 
32:  return  $E$ 
```

Algorithm 7.4 SIFT feature extraction (Part 2). Position refinement. Global parameters: n_{refine} , t_{peak} , $\bar{\rho}_{1,2}$ (see Table 7.5).

```

1:  REFINEKEYPOSITION( $\mathbf{D}, \mathbf{k}$ )
   Input:  $\mathbf{D}$ , hierarchical DoG scale space;  $\mathbf{k} = (p, q, u, v)$ , candidate
         (extremal) position.
   Returns a refined key point  $\mathbf{k}'$  or nil if no proper key point could be
         localized at or near the extremal position  $\mathbf{k}$ .
2:   $\alpha_{\max} \leftarrow \frac{(\bar{\rho}_{1,2}+1)^2}{\bar{\rho}_{1,2}}$                                  $\triangleright$  see Eqn. (7.70)
3:   $\mathbf{k}' \leftarrow \text{nil}$                                           $\triangleright$  refined key point
4:   $n \leftarrow 1$                                           $\triangleright$  number of repositioning steps
5:  while  $\neg \text{done} \wedge n \leq n_{\text{refine}} \wedge \text{ISINSIDE}(\mathbf{D}, \mathbf{k})$  do
6:     $\mathbf{N} \leftarrow \text{GETNEIGHBORHOOD}(\mathbf{D}, \mathbf{k})$                        $\triangleright$  Alg. 7.5
7:     $\nabla = \begin{pmatrix} d_x \\ d_x \\ d_\sigma \end{pmatrix} \leftarrow \text{GRADIENT}(\mathbf{N})$        $\triangleright$  Alg. 7.5
8:     $\mathbf{H}_D = \begin{pmatrix} d_{xx} & d_{xy} & d_{x\sigma} \\ d_{xy} & d_{yy} & d_{y\sigma} \\ d_{x\sigma} & d_{y\sigma} & d_{\sigma\sigma} \end{pmatrix} \leftarrow \text{HESSIAN}(\mathbf{N})$        $\triangleright$  Alg. 7.5
9:    if  $\det(\mathbf{H}_D) = 0$  then                                $\triangleright \mathbf{H}_D$  is not invertible
10:    $\text{done} \leftarrow \text{true}$                           $\triangleright$  ignore this point and finish
11:   else
12:      $\mathbf{d} = (x', y', \sigma')^\top \leftarrow -\mathbf{H}_D^{-1} \cdot \nabla$            $\triangleright$  Eqn. (7.58)
13:     if  $|x'| < 0.5 \wedge |y'| < 0.5$  then       $\triangleright$  stay in the same DoG cell
14:        $\text{done} \leftarrow \text{true}$ 
15:        $D_{\text{peak}} \leftarrow \mathbf{N}(0, 0, 0) + \frac{1}{2} \cdot \nabla^\top \cdot \mathbf{d}$        $\triangleright$  Eqn. (7.59)
16:        $\mathbf{H}_{xy} \leftarrow \begin{pmatrix} d_{xx} & d_{xy} \\ d_{xy} & d_{yy} \end{pmatrix}$             $\triangleright$  extract 2D Hessian from  $\mathbf{H}_D$ 
17:       if  $|D_{\text{peak}}| > t_{\text{peak}} \wedge \det(\mathbf{H}_{xy}) > 0$  then
18:          $\alpha \leftarrow \frac{[\text{tr}(\mathbf{H}_{xy})]^2}{\det(\mathbf{H}_{xy})}$            $\triangleright$  Eqn. (7.68)
19:         if  $\alpha \leq \alpha_{\max}$  then   $\triangleright$  suppress if edge-like, Eqn. (7.70)
20:            $\mathbf{k}' \leftarrow \mathbf{k} + (0, 0, x', y')^\top$            $\triangleright$  final key point
21:         else
22:           Move to a neighboring DoG position at same level  $p, q$ :
23:            $u' \leftarrow \min(1, \max(-1, \text{round}(x')))$        $\triangleright$  move by max.  $\pm 1$ 
24:            $v' \leftarrow \min(1, \max(-1, \text{round}(y')))$        $\triangleright$  move by max.  $\pm 1$ 
25:            $\mathbf{k} \leftarrow \mathbf{k} + (0, 0, u', v')^\top$ 
26:            $n \leftarrow n + 1$ 
return  $\mathbf{k}'$ .           $\triangleright \mathbf{k}'$  is either a refined key point position or nil

```

Algorithm 7.5 SIFT feature extraction (Part 3): neighborhood operations. Global parameters: Q , t_{extrm} (see Table 7.5).

```

1: ISINSIDE( $\mathbf{D}, \mathbf{k}$ )
   Checks if coordinate  $\mathbf{k} = (p, q, u, v)$  is inside the DoG scale space  $\mathbf{D}$ .
2:  $(p, q, u, v) \leftarrow \mathbf{k}$ 
3:  $(M, N) \leftarrow \text{SIZE}(\text{GETSCALELEVEL}(\mathbf{D}, p, q))$ 
4: return  $(0 < u < M-1) \wedge (0 < v < N-1) \wedge (0 \leq q < Q)$ 


---


5: GETNEIGHBORHOOD( $\mathbf{D}, \mathbf{k}$ )  $\triangleright \mathbf{k} = (p, q, u, v)$ 
   Collects and returns the  $3 \times 3 \times 3$  neighborhood values around position  $\mathbf{k}$  in the hierarchical DoG scale space  $\mathbf{D}$ .
6: Create map  $\mathbf{N} : \{-1, 0, 1\}^3 \mapsto \mathbb{R}$ 
7: for all  $(i, j, k) \in \{-1, 0, 1\}^3$  do  $\triangleright$  collect  $3 \times 3 \times 3$  neighborhood
8:    $\mathbf{N}(i, j, k) \leftarrow \mathbf{D}_{p, q+k}(u+i, v+j)$ 
9: return  $\mathbf{N}$ .


---


10: ISEXTREMUM( $\mathbf{N}$ )  $\triangleright \mathbf{N}$  is a  $3 \times 3 \times 3$  neighborhood
   Determines if the center of the neighborhood  $\mathbf{N}$  is either a local minimum or maximum by the threshold  $t_{\text{extrm}} \geq 0$ . Returns a boolean value (true or false).
11:  $c \leftarrow \mathbf{N}(0, 0, 0)$   $\triangleright$  center value
12:  $isMin \leftarrow (c + t_{\text{extrm}}) < \min_{(i,j,k) \neq (0,0,0)} \mathbf{N}(i, j, k)$ 
13:  $isMax \leftarrow (c - t_{\text{extrm}}) > \max_{(i,j,k) \neq (0,0,0)} \mathbf{N}(i, j, k)$ 
14: return  $isMin \vee isMax$ .


---


15: GRADIENT( $\mathbf{N}$ )  $\triangleright \mathbf{N}$  is a  $3 \times 3 \times 3$  neighborhood
   Returns the estim. gradient vector ( $\nabla$ ) for the 3D neighborhood  $\mathbf{N}$ .
16:  $\nabla \leftarrow 0.5 \cdot \begin{pmatrix} \mathbf{N}(1, 0, 0) - \mathbf{N}(-1, 0, 0) \\ \mathbf{N}(0, 1, 0) - \mathbf{N}(0, -1, 0) \\ \mathbf{N}(0, 0, 1) - \mathbf{N}(0, 0, -1) \end{pmatrix}$   $\triangleright \nabla = \begin{pmatrix} d_x \\ d_y \\ d_\sigma \end{pmatrix}$ , see Eqn. (7.54)
17: return  $\nabla$ .


---


18: HESSIAN( $\mathbf{N}$ )  $\triangleright \mathbf{N}$  is a  $3 \times 3 \times 3$  neighborhood
   Returns the estim. Hessian matrix ( $\mathbf{H}$ ) for the neighborhood  $\mathbf{N}$ .
19:  $d_{xx} \leftarrow \mathbf{N}(-1, 0, 0) - 2 \cdot \mathbf{N}(0, 0, 0) + \mathbf{N}(1, 0, 0)$   $\triangleright$  see Eqn. (7.56)
20:  $d_{yy} \leftarrow \mathbf{N}(0, -1, 0) - 2 \cdot \mathbf{N}(0, 0, 0) + \mathbf{N}(0, 1, 0)$ 
21:  $d_{\sigma\sigma} \leftarrow \mathbf{N}(0, 0, -1) - 2 \cdot \mathbf{N}(0, 0, 0) + \mathbf{N}(0, 0, 1)$ 
22:  $d_{xy} \leftarrow [\mathbf{N}(1, 1, 0) - \mathbf{N}(-1, 1, 0) - \mathbf{N}(1, -1, 0) + \mathbf{N}(-1, -1, 0)] / 4$ 
23:  $d_{x\sigma} \leftarrow [\mathbf{N}(1, 0, 1) - \mathbf{N}(-1, 0, 1) - \mathbf{N}(1, 0, -1) + \mathbf{N}(-1, 0, -1)] / 4$ 
24:  $d_{y\sigma} \leftarrow [\mathbf{N}(0, 1, 1) - \mathbf{N}(0, -1, 1) - \mathbf{N}(0, 1, -1) + \mathbf{N}(0, -1, -1)] / 4$ 
25:  $\mathbf{H} \leftarrow \begin{pmatrix} d_{xx} & d_{xy} & d_{x\sigma} \\ d_{xy} & d_{yy} & d_{y\sigma} \\ d_{x\sigma} & d_{y\sigma} & d_{\sigma\sigma} \end{pmatrix}$ 
26: return  $\mathbf{H}$ .
```

Algorithm 7.6 SIFT feature extraction (Part 4): keypoint orientation assignment. Global parameters: n_{smooth} , t_{domor} (see Table 7.5).

```

1: GETDOMINANTORIENTATIONS( $\mathbf{G}, \mathbf{k}'$ )
   Input:  $\mathbf{G}$ , hierarchical Gaussian scale space;  $\mathbf{k}' = (p, q, x, y)$ , refined
   key point at octave  $p$ , scale level  $q$  and spatial position  $x, y$  (in octave's
   coordinates).
   Returns a list of dominant orientations for the key point  $\mathbf{k}'$ .
2:  $\mathbf{h}_\phi \leftarrow \text{GETORIENTATIONHISTOGRAM}(\mathbf{G}, \mathbf{k}')$                                  $\triangleright$  Alg. 7.7
3: SMOOTHCIRCULAR( $\mathbf{h}_\phi, n_{\text{smooth}}$ )
4:  $A \leftarrow \text{FINDPEAKORIENTATIONS}(\mathbf{h}_\phi)$ 
5: return  $A$ .


---


6: SMOOTHCIRCULAR( $\mathbf{x}, iter$ )
   Smooths the real-valued vector  $\mathbf{x} = (x_0, \dots, x_{n-1})$  circularly using
   the 3-element kernel  $H = (h_0, h_1, h_2)$ , with  $h_1$  as the hot-spot. The
   filter operation is applied  $iter$  times and “in place”, i. e., the vector  $\mathbf{x}$ 
   is modified.
7:  $(h_0, h_1, h_2) \leftarrow \frac{1}{4}(1, 2, 1)$                                                $\triangleright$  1D filter kernel
8:  $n \leftarrow \text{SIZE}(\mathbf{x})$ 
9: for  $i \leftarrow 1, \dots, iter$  do
10:     $s \leftarrow \mathbf{x}(0), p \leftarrow \mathbf{x}(n-1)$ 
11:    for  $j \leftarrow 0, \dots, n-2$  do
12:        $c \leftarrow \mathbf{x}(j)$ 
13:        $\mathbf{x}(j) \leftarrow h_0 \cdot p + h_1 \cdot \mathbf{x}(j) + h_2 \cdot \mathbf{x}(j+1)$ 
14:        $p \leftarrow c$ 
15:     $\mathbf{x}(n-1) \leftarrow h_0 \cdot p + h_1 \cdot \mathbf{x}(n-1) + h_2 \cdot s$ 
16: return.


---


17: FINDPEAKORIENTATIONS( $\mathbf{h}_\phi$ )
18:  $n \leftarrow \text{SIZE}(\mathbf{h}_\phi), A \leftarrow ()$ 
19:  $h_{\max} \leftarrow \max_{0 \leq i < n} \mathbf{h}_\phi(i)$ 
20: for  $k \leftarrow 0, \dots, n-1$  do
21:     $h_c \leftarrow \mathbf{h}(k)$ 
22:    if  $h_c > t_{\text{domor}} \cdot h_{\max}$  then           $\triangleright$  only accept dominant peaks
23:        $h_p \leftarrow \mathbf{h}_\phi((k-1) \bmod n)$ 
24:        $h_n \leftarrow \mathbf{h}_\phi((k+1) \bmod n)$ 
25:       if  $(h_c > h_p) \wedge (h_c > h_n)$  then       $\triangleright$  local max. at index  $k$ 
26:           $\check{k} \leftarrow k + \frac{h_p - h_n}{2 \cdot (h_p - 2 \cdot h_c + h_n)}$      $\triangleright$  quadr. interpol., Eqn. (7.82)
27:           $\theta \leftarrow (\check{k} \cdot \frac{2\pi}{n}) \bmod 2\pi$          $\triangleright$  domin. orientation, Eqn. (7.83)
28:           $A \leftarrow A \cup (\theta)$ 
29: return  $A$ .
```

Algorithm 7.7 SIFT feature extraction (Part 5): calculation of the orientation histogram and gradients from Gaussian scale levels. Global parameters: n_{orient} (see Table 7.5).

```

1: GETORIENTATIONHISTOGRAM( $\mathbf{G}, \mathbf{k}'$ )
   Input:  $\mathbf{G}$ , hierarchical Gaussian scale space;  $\mathbf{k}' = (p, q, x, y)$ , refined
         key point at octave  $p$ , scale level  $q$  and relative position  $x, y$ .
         Returns the gradient orientation histogram for key point  $\mathbf{k}'$ .
2:  $\mathbf{G}_{p,q} \leftarrow \text{GETSCALELEVEL}(\mathbf{G}, p, q)$ 
3:  $(M, N) \leftarrow \text{SIZE}(\mathbf{G}_{p,q})$ 
4: Create a new map  $h_\phi : [0, n_{\text{orient}} - 1] \mapsto \mathbb{R}$ .            $\triangleright$  new histogram  $h_\phi$ 
5: for  $i \leftarrow 0, \dots, n_{\text{orient}} - 1$  do                                 $\triangleright$  initialize  $h_\phi$  to zero
6:    $h_\phi(i) \leftarrow 0$ 
7:  $\sigma_w \leftarrow 1.5 \cdot \sigma_0 \cdot 2^{q/Q}$        $\triangleright \sigma$  of Gaussian weight fun., see Eqn. (7.74)
8:  $r_w \leftarrow \max(1, 2.5 \cdot \sigma_w)$            $\triangleright$  rad. of weight fun., see Eqn. (7.75)
9:  $u_{\min} \leftarrow \max(\lfloor x - r_w \rfloor, 1), u_{\max} \leftarrow \min(\lceil x + r_w \rceil, M - 2)$ 
10:  $v_{\min} \leftarrow \max(\lfloor y - r_w \rfloor, 1), v_{\max} \leftarrow \min(\lceil y + r_w \rceil, N - 2)$ 
11: for  $u \leftarrow u_{\min}, \dots, u_{\max}$  do
12:   for  $v \leftarrow v_{\min}, \dots, v_{\max}$  do
13:      $r^2 \leftarrow (u - x)^2 + (v - y)^2$ 
14:     if  $r^2 < r_w^2$  then
15:        $(R, \phi) \leftarrow \text{GETGRADIENTPOLAR}(\mathbf{G}_{p,q}, u, v)$      $\triangleright$  see below
16:        $w_G \leftarrow \exp\left(-\frac{(u-x)^2 + (v-y)^2}{2\sigma_w^2}\right)$      $\triangleright$  Gaussian weight
17:        $z \leftarrow R \cdot w_G$                                           $\triangleright$  quantity to accumulate
18:        $\kappa_\phi \leftarrow n_{\text{orient}} \cdot \frac{\phi}{2\pi}$              $\triangleright \kappa_\phi \in [-\frac{n_{\text{orient}}}{2}, +\frac{n_{\text{orient}}}{2}]$ 
19:        $\alpha \leftarrow \kappa_\phi - \lfloor \kappa_\phi \rfloor$                    $\triangleright \alpha \in [0, 1]$ 
20:        $k_0 \leftarrow \lfloor \kappa_\phi \rfloor \bmod n_{\text{orient}}$          $\triangleright$  lower orientation bin
21:        $k_1 \leftarrow (k_0 + 1) \bmod n_{\text{orient}}$          $\triangleright$  upper orientation bin
22:        $h_\phi(k_0) \leftarrow h_\phi(k_0) + (1 - \alpha) \cdot z$      $\triangleright$  distrib.  $z$  onto bins  $k_0, k_1$ 
23:        $h_\phi(k_1) \leftarrow h_\phi(k_1) + \alpha \cdot z$ 
24: return  $h_\phi$ 


---


25: GETGRADIENTPOLAR( $\mathbf{G}_{p,q}, u, v$ )
   Returns the gradient magnitude and orientation at position  $(u, v)$  of
   the Gaussian scale level  $\mathbf{G}_{p,q}$ .
26: 
$$\begin{pmatrix} d_x \\ d_y \end{pmatrix} \leftarrow 0.5 \cdot \begin{pmatrix} \mathbf{G}_{p,q}(u+1, v) - \mathbf{G}_{p,q}(u-1, v) \\ \mathbf{G}_{p,q}(u, v+1) - \mathbf{G}_{p,q}(u, v-1) \end{pmatrix}$$
     $\triangleright$  gradient at  $u, v$ 
27:  $R \leftarrow (d_x^2 + d_y^2)^{1/2}$                        $\triangleright$  gradient magnitude
28:  $\phi \leftarrow \text{Arctan}(d_x, d_y)$                    $\triangleright$  gradient orientation  $(-\pi \leq \phi \leq \pi)$ 
29: return  $(R, \phi)$ .

```

Algorithm 7.8 SIFT feature extraction (Part 6): calculation of SIFT descriptors. Global parameters: Q , σ_0 , s_d , n_{spat} , n_{angl} (see Table 7.5).

```

1: MAKESIFTDESCRIPTOR( $\mathbf{G}, \mathbf{k}', \theta$ )
   Input:  $\mathbf{G}$ , hierarchical Gaussian scale space;  $\mathbf{k}' = (p, q, x, y)$ , refined
         key point;  $\theta$ , dominant orientation.
   Returns a new SIFT descriptor for the key point  $\mathbf{k}'$ .
2:  $\mathbf{G}_{p,q} \leftarrow \text{GETSCALELEVEL}(\mathbf{G}, p, q)$ 
3:  $(M, N) \leftarrow \text{SIZE}(\mathbf{G}_{p,q})$ 
4:  $\dot{\sigma}_q \leftarrow \sigma_0 \cdot 2^{q/Q}$                                  $\triangleright$  decimated scale at level  $q$ 
5:  $w_d \leftarrow s_d \cdot \dot{\sigma}_q$                                 $\triangleright$  descriptor size is prop. to keypoint scale
6:  $\sigma_d \leftarrow 0.25 \cdot w_d$                                   $\triangleright$  width of Gaussian weighting function
7:  $r_d \leftarrow 2.5 \cdot \sigma_d$                                  $\triangleright$  cutoff radius of weighting function
8:  $u_{\min} \leftarrow \max(\lfloor x - r_d \rfloor, 1), \quad u_{\max} \leftarrow \min(\lceil x + r_d \rceil, M - 2)$ 
9:  $v_{\min} \leftarrow \max(\lfloor y - r_d \rfloor, 1), \quad v_{\max} \leftarrow \min(\lceil y + r_d \rceil, N - 2)$ 
10: Create map  $\mathbf{h}_\nabla : n_{\text{spat}} \times n_{\text{spat}} \times n_{\text{angl}} \mapsto \mathbb{R}$      $\triangleright$  gradient histogram  $\mathbf{h}_\nabla$ 
11: for all  $(i, j, k) \in n_{\text{spat}} \times n_{\text{spat}} \times n_{\text{angl}}$  do
12:    $\mathbf{h}_\nabla(i, j, k) \leftarrow 0$                                       $\triangleright$  initialize  $\mathbf{h}_\nabla$  to zero
13:   for  $u \leftarrow u_{\min}, \dots, u_{\max}$  do
14:     for  $v \leftarrow v_{\min}, \dots, v_{\max}$  do
15:        $r^2 \leftarrow (u - x)^2 + (v - y)^2$ 
16:       if  $r^2 < r_d^2$  then
           Map to canonical coord. frame, with  $u', v' \in [-\frac{1}{2}, +\frac{1}{2}]$ :
           
$$\begin{pmatrix} u' \\ v' \end{pmatrix} \leftarrow \frac{1}{w_d} \cdot \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{pmatrix} \cdot \begin{pmatrix} u - x \\ v - y \end{pmatrix}$$

17:        $(R, \phi) \leftarrow \text{GETGRADIENTPOLAR}(\mathbf{G}_{p,q}, u, v)$        $\triangleright$  Alg. 7.7
18:        $\phi' \leftarrow (\phi - \theta) \bmod 2\pi$                              $\triangleright$  normalize gradient angle
19:        $w_G \leftarrow \exp(-\frac{r^2}{2\sigma_d^2})$                           $\triangleright$  Gaussian weight
20:        $z \leftarrow R \cdot w_G$                                           $\triangleright$  quantity to accumulate
21:        $\text{UPDATEGRADIENTHISTOGRAM}(\mathbf{h}_\nabla, u', v', \phi', z)$   $\triangleright$  Alg. 7.9
22:    $\mathbf{f}_{\text{sift}} \leftarrow \text{MAKEFEATUREVECTOR}(\mathbf{h}_\nabla)$            $\triangleright$  see Alg. 7.10
23:    $\sigma \leftarrow \sigma_0 \cdot 2^{p+q/Q}$                                  $\triangleright$  absolute scale, Eqn. (7.36)
24:   
$$\begin{pmatrix} x' \\ y' \end{pmatrix} \leftarrow 2^p \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$
           $\triangleright$  real position, Eqn. (7.46)
25:    $\mathbf{s} \leftarrow \langle x', y', \sigma, \theta, \mathbf{f}_{\text{sift}} \rangle$            $\triangleright$  create a new SIFT descriptor
26: return  $\mathbf{s}$ .
```

Algorithm 7.9 SIFT feature extraction (Part 7): updating the gradient descriptor histogram. The quantity z pertaining to the continuous position (u', v', ϕ') is to be accumulated into the three-dimensional histogram h_{∇} (u', v' are normalized spatial coordinates, ϕ' is the orientation). The quantity z is distributed over up to 8 neighboring histogram bins (see Fig. 7.26) by tri-linear interpolation. Note that the orientation coordinate ϕ' receives special treatment because it is cyclic. Global parameters: n_{spat} , n_{angl} (see Table 7.5).

```

1: UPDATEGRADIENTHISTOGRAM( $h_{\nabla}, u', v', \phi', z$ )
   Input:  $h_{\nabla}$ , gradient histogram of size  $n_{\text{spat}} \times n_{\text{spat}} \times n_{\text{angl}}$ , with
           $h_{\nabla}(i, j, k) \in \mathbb{R}$ ;  $u', v' \in [-\frac{1}{2}, +\frac{1}{2}]$ , normalized spatial position;  $\phi' \in [0, 2\pi)$ , normalized gradient orientation;  $z \in \mathbb{R}$ , quantity to be accumulated into  $h_{\nabla}$ . Returns nothing but modifies the histogram  $h_{\nabla}$ .
2:  $i' \leftarrow n_{\text{spat}} \cdot u' + 0.5 \cdot (n_{\text{spat}} - 1)$                                  $\triangleright$  see Eqn. (7.91)
3:  $j' \leftarrow n_{\text{spat}} \cdot v' + 0.5 \cdot (n_{\text{spat}} - 1)$                                  $\triangleright -0.5 \leq i', j' \leq n_{\text{spat}} - 0.5$ 
4:  $k' \leftarrow n_{\text{angl}} \cdot \frac{\phi'}{2\pi}$                                                $\triangleright -\frac{n_{\text{angl}}}{2} \leq k' \leq \frac{n_{\text{angl}}}{2}$ 
5:  $i_0 \leftarrow \lfloor i' \rfloor$                                                          $\triangleright$  see Eqn. (7.92)
6:  $i_1 \leftarrow i_0 + 1$ 
7:  $j_0 \leftarrow \lfloor j' \rfloor$ 
8:  $j_1 \leftarrow j_0 + 1$ 
9:  $k_0 \leftarrow \lfloor k' \rfloor \bmod n_{\text{angl}}$ 
10:  $k_1 \leftarrow (k_0 + 1) \bmod n_{\text{angl}}$ 
11:  $\alpha_0 \leftarrow i_1 - i'$                                                          $\triangleright$  see Eqn. (7.93)
12:  $\alpha_1 \leftarrow 1 - \alpha_0$ 
13:  $\beta_0 \leftarrow j_1 - j'$ 
14:  $\beta_1 \leftarrow 1 - \beta_0$ 
15:  $\gamma_0 \leftarrow \lfloor k' \rfloor + 1 - i'$ 
16:  $\gamma_1 \leftarrow 1 - \gamma_0$ 
   Distribute  $z$  over up to 8 adjacent spatial/orientation bins:
17: for  $a \in \{0, 1\}$  do
18:   if  $(0 \leq i_a < n_{\text{spat}})$  then
19:     for  $b \in \{0, 1\}$  do
20:       if  $(0 \leq j_b < n_{\text{spat}})$  then
21:         for  $c \in \{0, 1\}$  do                                               $\triangleright$  see Eqn. (7.94)
22:            $h_{\nabla}(i_a, j_b, k_c) \leftarrow h_{\nabla}(i_a, j_b, k_c) + z \cdot \alpha_a \cdot \beta_b \cdot \gamma_c$ 
23: return.

```

Algorithm 7.10 SIFT feature extraction (Part 8): converting the orientation histogram to a SIFT feature vector. Global parameters: n_{spat} , n_{angl} , t_{fclip} , s_{fscale} (see [Table 7.5](#)).

```

1: MAKEFEATUREVECTOR( $h_{\nabla}$ )
   Input:  $h_{\nabla}$ , gradient histogram of size  $n_{\text{spat}} \times n_{\text{spat}} \times n_{\text{angl}}$ .
   Returns a 1D integer (unsigned byte) vector obtained from  $h_{\nabla}$ .
2: Create a map  $f : [0, n_{\text{spat}}^2 \cdot n_{\text{angl}} - 1] \mapsto \mathbb{R}$            ▷ new 1D vector  $f$ 
3:  $m \leftarrow 0$ 
4: for  $i \leftarrow 0, \dots, n_{\text{spat}} - 1$  do                                ▷ flatten  $h_{\nabla}$  into  $f$ 
5:   for  $j \leftarrow 0, \dots, n_{\text{spat}} - 1$  do
6:     for  $k \leftarrow 0, \dots, n_{\text{angl}} - 1$  do
7:        $f(m) \leftarrow h_{\nabla}(i, j, k)$ 
8:        $m \leftarrow m + 1$ 
9: NORMALIZE( $f$ )
10: CLIPPEAKS( $f$ ,  $t_{\text{fclip}}$ )
11: NORMALIZE( $f$ )
12:  $f_{\text{sift}} \leftarrow \text{MAPTOBYTES}(f, s_{\text{fscale}})$ 
13: return  $f_{\text{sift}}$ .


---


14: NORMALIZE( $x$ )
   Scales vector  $x$  to unit norm. Returns nothing, but  $x$  is modified.
15:  $n \leftarrow \text{SIZE}(x)$ 
16:  $s \leftarrow \sum_{i=0}^{n-1} x(i)$ 
17: for  $i \leftarrow 0, \dots, n-1$  do
18:    $x(i) \leftarrow \frac{1}{s} \cdot x(i)$ 
19: return.


---


20: CLIPPEAKS( $x$ ,  $x_{\text{max}}$ )
   Limits the elements of  $x$  to  $x_{\text{max}}$ . Returns nothing, but  $x$  is modified.
21:  $n \leftarrow \text{SIZE}(x)$ 
22: for  $i \leftarrow 0, \dots, n-1$  do
23:    $x(i) \leftarrow \min(x(i), x_{\text{max}})$ 
24: return.


---


25: MAPTOBYTES( $x, s$ )
   Converts the real-valued vector  $x$  to an integer (unsigned byte) valued
   vector with elements in  $[0, 255]$ , using the scale factor  $s > 0$ .
26:  $n \leftarrow \text{SIZE}(x)$ 
27: Create a new map  $x_{\text{int}} : [0, n-1] \mapsto [0, 255]$            ▷ new byte vector
28: for  $i \leftarrow 0, \dots, n-1$  do
29:    $a \leftarrow \text{round}(s \cdot x(i))$                                      ▷  $a \in \mathbb{N}_0$ 
30:    $x_{\text{int}}(i) \leftarrow \min(a, 255)$                                  ▷  $x_{\text{int}}(i) \in [0, 255]$ 
31: return  $x_{\text{int}}$ .
```

Table 7.5 Predefined constants used in the SIFT algorithms (Algs. 7.3–7.11).

Scale space parameters

<i>Symbol</i>	<i>Java var.</i>	<i>Value</i>	<i>Description</i>
Q	<code>Q</code>	3	scale steps (levels) per octave
P	<code>P</code>	4	number of scale space octaves
σ_s	<code>sigma_s</code>	0.5	sampling scale (nominal smoothing of the input image)
σ_0	<code>sigma_0</code>	1.6	base scale of level 0 (base smoothing)
t_{extrm}	<code>t_Extrm</code>	0.0	min. difference w.r.t. any neighbor for extrema detection

Key point detection

<i>Symbol</i>	<i>Java var.</i>	<i>Value</i>	<i>Description</i>
n_{orient}	<code>n_Orient</code>	36	number of orientation bins (angular resolution) used for calculating the dominant keypoint orientation
n_{refine}	<code>n_Refine</code>	5	max. number of iterations for repositioning a key point
n_{smooth}	<code>n_Smooth</code>	2	number of smoothing iterations applied to the orientation histogram
$\bar{\rho}_{1,2}$	<code>reMax</code>	10.0	max. ratio of principal curvatures (3, …, 10)
t_{domor}	<code>t_DomOr</code>	0.8	min. value in orientation histogram for selecting dominant orientations (rel. to max. entry)
t_{mag}	<code>t_Mag</code>	0.01	min. DoG magnitude for initial keypoint candidates
t_{peak}	<code>t_Peak</code>	0.01	min. DoG magnitude at interpolated peaks

Feature descriptor

<i>Symbol</i>	<i>Java var.</i>	<i>Value</i>	<i>Description</i>
n_{spat}	<code>n_Spat</code>	4	number of spatial descriptor bins along each x/y axis
n_{angl}	<code>n_Angl</code>	16	number of angular descriptor bins
s_d	<code>s_Desc</code>	10.0	spatial size factor of descriptor (relative to feature scale)
s_{fscale}	<code>s_Fscale</code>	512.0	scale factor for converting normalized feature values to byte values in [0, 255]
t_{fclip}	<code>t_Fclip</code>	0.2	max. value for clipping elements of normalized feature vectors

Feature matching

<i>Symbol</i>	<i>Java var.</i>	<i>Value</i>	<i>Description</i>
$\bar{\rho}_{\text{match}}$	<code>rmMax</code>	0.8	max. ratio of best and second-best matching feature distance

self-localization or object recognition might use a large database of model descriptors and the task is to match these to the SIFT features detected in a new image or video sequence.

7.5.1 Feature distance and match quality

A typical setup is that a list of SIFT descriptors $S^{(k)} = (s_1^{(k)}, \dots, s_{N_k}^{(k)})$ is calculated for each image I_k and the subsequent task is to find matching descriptors

in pairs of descriptor sets, for example,

$$S^{(a)} = (\mathbf{s}_1^{(a)}, \mathbf{s}_2^{(a)}, \dots, \mathbf{s}_{N_a}^{(a)}) \quad \text{and} \quad S^{(b)} = (\mathbf{s}_1^{(b)}, \mathbf{s}_2^{(b)}, \dots, \mathbf{s}_{N_b}^{(b)}). \quad (7.100)$$

The similarity between a given pair of descriptors, $\mathbf{s}_i = \langle x_i, y_i, \sigma_i, \theta_i, \mathbf{f}_i \rangle$ and $\mathbf{s}_j = \langle x_j, y_j, \sigma_j, \theta_j, \mathbf{f}_j \rangle$, is quantified by the distance between the corresponding feature vectors $\mathbf{f}_i, \mathbf{f}_j$, that is,

$$\text{dist}(\mathbf{s}_i, \mathbf{s}_j) = \|\mathbf{f}_i - \mathbf{f}_j\|, \quad (7.101)$$

where $\|\cdot\|$ denotes an appropriate norm (typically Euclidean, alternatives are discussed below).²⁵

Note that this distance is measured between individual points distributed in a high-dimensional (typ. 128-dimensional) vector space that is only sparsely populated. Since there is always a best-matching counterpart for a given descriptor, matches may occur between unrelated features even if the correct feature is not contained in the target set.

Obviously, significant matches should exhibit small feature distances but setting a *fixed limit* on the acceptable feature distance turns out to be inappropriate in practice, since some descriptors are more discriminative than others. The solution proposed in [80] is to compare the distance obtained for the *best* feature match to that of the *second-best* match. For a given reference descriptor $\mathbf{s}_r \in S^{(a)}$, the best match is defined as the descriptor $\mathbf{s}_1 \in S^{(b)}$ which has the smallest distance from \mathbf{s}_r in the multi-dimensional feature space, that is,

$$\mathbf{s}_1 = \underset{\mathbf{s}_j \in S^{(b)}}{\operatorname{argmin}} \text{dist}(\mathbf{s}_r, \mathbf{s}_j) \quad (7.102)$$

and the primary distance is $d_{r,1} = \text{dist}(\mathbf{s}_r, \mathbf{s}_1)$. Analogously, the second-best matching descriptor is

$$\mathbf{s}_2 = \underset{\substack{\mathbf{s}_j \in S^{(b)}, \\ \mathbf{s}_j \neq \mathbf{s}_1}}{\operatorname{argmin}} \text{dist}(\mathbf{s}_r, \mathbf{s}_j) \quad (7.103)$$

and the corresponding distance is $d_{r,2} = \text{dist}(\mathbf{s}_r, \mathbf{s}_2)$, with $d_{r,1} \leq d_{r,2}$. Reliable matches are expected to have a distance to the primary feature \mathbf{s}_1 that is considerably smaller than the distance to any other feature in the target set. In the case of a weak or ambiguous match, on the other hand, it is likely that other matches exist at a distance similar to $d_{r,1}$, including the second-best match \mathbf{s}_2 . Comparing the best and the second-best distances thus provides information about the likelihood of a false match. For this purpose, we define the *feature distance ratio*

²⁵ See also Sec. B.1.2 in the Appendix.

$$\rho_{\text{match}}(\mathbf{s}_r, \mathbf{s}_1, \mathbf{s}_2) = \frac{\text{dist}(\mathbf{s}_r, \mathbf{s}_1)}{\text{dist}(\mathbf{s}_r, \mathbf{s}_2)} = \frac{d_{r,1}}{d_{r,2}}, \quad (7.104)$$

such that $\rho_{\text{match}} \in [0, 1]$. If the distance $d_{r,1}$ between \mathbf{s}_r and the primary feature \mathbf{s}_1 is small compared to the secondary distance $d_{r,2}$, then the value of ρ_{match} is small as well. Thus, large values of ρ_{match} indicate that the corresponding match is likely to be weak or ambiguous. Matches are only accepted if they are sufficiently distinctive, e.g., by enforcing the condition

$$\rho_{\text{match}}(\mathbf{s}_r, \mathbf{s}_1, \mathbf{s}_2) \leq \bar{\rho}_{\text{match}}, \quad (7.105)$$

where $\bar{\rho}_{\text{match}} \in [0, 1]$ is a constant maximum value. The complete matching process is summarized in [Alg. 7.11](#).

7.5.2 Examples

The following examples were calculated on pairs of stereographic images taken at the beginning of the 20th century.²⁶ In [Figs. 7.28–7.31](#), two sets of SIFT descriptors were extracted individually from the left and the right frame (marked by blue rectangles), respectively. Matching was done by enumerating all possible descriptor pairs from the left and the right image, calculating their (Euclidean) distance, and showing the 25 closest matches obtained from ca. 1,000 detected keypoints in each frame. Matching descriptors are labeled in each frame by their rank, i.e., label “1” marks the descriptor pair with the smallest absolute distance. Selected details from these results are shown in [Fig. 7.29](#). Unless otherwise noted, all SIFT parameters are set to their default values (see [Table 7.5](#)).

Although the use of the Euclidean (L_2) norm for measuring the distances between feature vectors in Eqn. (7.101) is suggested in [80], other norms have been considered [62, 98, 124] to improve the statistical robustness and noise resistance. In [Fig. 7.30](#), matching results are shown using the L_1 , L_2 , and L_∞ norms, respectively. Matches are sorted by feature distance (1 is best). Note that the resulting sets of top-ranking matches are almost the same with different distance norms, but the ordering of the strongest matches does change.

²⁶ The images used in [Figs. 7.28–7.31](#) are historic stereographs made publicly available by the U.S. Library of Congress (www.loc.gov).

Algorithm 7.11 SIFT feature matching using Euclidean feature distance and exhaustive search. The returned sequence of SIFT matches is sorted to ascending distance between corresponding feature pairs. Parameters $\bar{\rho}_{1,2}$ are described in Eqn. (7.105) (see also Table 7.5). FUNCTION DIST(s_a, s_b) demonstrates the calculation of the Euclidean (L_2) feature distance, other options are the L_1 and L_∞ norms.

```

1: MATCHDESCRIPTORS( $S^{(a)}, S^{(b)}, \bar{\rho}_{\text{match}}$ )
   Input:  $S^{(a)}, S^{(b)}$ , two sets of SIFT descriptors;  $\bar{\rho}_{\text{match}}$ , max. ratio of
         best and second-best matching distance.
   Returns a sorted list of matches  $m_{ij} = \langle s_i, s_j, d_{ij} \rangle$ , with  $s_i \in S^{(a)}$ ,
    $s_j \in S^{(b)}$  and  $d_{ij}$  being the distance between  $s_i, s_j$  in feature space.
2:  $M \leftarrow ()$                                  $\triangleright$  empty sequence of matches
3: for all  $s_i \in S^{(a)}$  do
4:    $s_1 \leftarrow \text{nil}, d_{r,1} \leftarrow \infty$            $\triangleright$  best nearest neighbor
5:    $s_2 \leftarrow \text{nil}, d_{r,2} \leftarrow \infty$            $\triangleright$  second-best nearest neighbor
6:   for all  $s_j \in S^{(b)}$  do
7:      $d \leftarrow \text{DIST}(s_i, s_j)$ 
8:     if  $d < d_{r,1}$  then                       $\triangleright d$  is a new ‘best’ distance
9:        $s_2 \leftarrow s_1, d_{r,2} \leftarrow d_{r,1}$ 
10:       $s_1 \leftarrow s_j, d_{r,1} \leftarrow d$ 
11:    else
12:      if  $d < d_{r,2}$  then                       $\triangleright d$  is a new ‘second-best’ distance
13:         $s_2 \leftarrow s_j, d_{r,2} \leftarrow d$ 
14:      if ( $s_2 \neq \text{nil}$ )  $\wedge (\frac{d_{r,1}}{d_{r,2}} \leq \bar{\rho}_{\text{match}})$  then     $\triangleright$  see Eqns. (7.104–7.105)
15:         $m \leftarrow \langle s_i, s_1, d_{r,1} \rangle$             $\triangleright$  add a new match
16:         $M \cup (m)$ 
17:    SORT(M)                                 $\triangleright$  sort M to ascending distance  $d_{r,1}$ 
18: return M.

19: DIST( $s_a, s_b$ )
   Input: SIFT features  $s_a = \langle x_a, y_a, \sigma_a, \theta_a, f_a \rangle$ ,  $s_b = \langle x_b, y_b, \sigma_b, \theta_b, f_b \rangle$ .
   Returns the Euclidean distance between  $s_a$  and  $s_b$ ,  $d_{a,b} = \|f_a - f_b\|$ .
20:  $n \leftarrow |f_a|$                            $\triangleright$  length of feature vectors,  $n = |f_a| = |f_b|$ 
21:  $\Sigma \leftarrow 0$ 
22: for  $i \leftarrow 0, \dots, n-1$  do
23:    $\Sigma \leftarrow \Sigma + [f_a(i) - f_b(i)]^2$ 
24: return  $\sqrt{\Sigma}$ .

```

[Figure 7.31](#) demonstrates the effectiveness of selecting feature matches based on the ratio between the distances to the best and the second-best match (see Eqns. 7.102–7.103). Again the figure shows the 25 top-ranking matches based on the minimum (L_2) feature distance. With the maximum distance ratio $\bar{\rho}_{\text{match}}$ set to 1.0, rejection is practically turned off with the result that several false or ambiguous matches are among the top-ranking feature matches ([Fig. 7.31 \(a\)](#)). With $\bar{\rho}_{\text{match}}$ set to 0.8 and finally 0.5, the number of false matches is effectively reduced ([Fig. 7.31 \(b,c\)](#)).²⁷

7.6 Efficient feature matching

The task of finding the best match based on the minimum distance in feature space is called “nearest-neighbor” search. If performed exhaustively, evaluating all possible matches between two descriptor sets $S^{(a)}$ and $S^{(b)}$ of size N_a and N_b , respectively, requires $N_a \cdot N_b$ feature distance calculations and comparisons. While this may be acceptable for small feature sets (with maybe up to 1,000 descriptors each), this linear (brute-force) approach becomes prohibitively expensive for large feature sets with possibly millions of candidates, as required for example in the context of image database indexing or robot self-localization. Although efficient methods for exact nearest-neighbor search based on tree structures exist, such as the k -d tree method [41], it has been shown that these methods lose their effectiveness with increasing dimensionality of the search space. In fact, no algorithms are known that significantly outperform exhaustive (linear) nearest neighbor search in feature spaces that are more than about 10-dimensional [80]. SIFT feature vectors are 128-dimensional and therefore exact nearest-neighbor search is not a viable option for efficient matching between large descriptor sets.

The approach taken in [11,80] abandons exact nearest-neighbor search in favor of finding an *approximate* solution with substantially reduced effort, based on ideas described in [5]. This so-called “best-bin-first” method uses a modified k -d algorithm which searches neighboring feature space partitions in the order of their closest distance from the given feature vector. To limit the exploration to a small fraction of the feature space, the search is cut off after checking the first 200 candidates, which results in a substantial speedup without compromising the search results, particularly when combined with feature selection based on the ratio of primary and secondary distances (see Eqns. (7.105–7.104)). Additional details can be found in [11].

Approximate nearest-neighbor search in high-dimensional spaces is not only essential for practical SIFT matching in real time, but is a general problem with

²⁷ $\bar{\rho}_{\text{match}} = 0.8$ is recommended in [80].

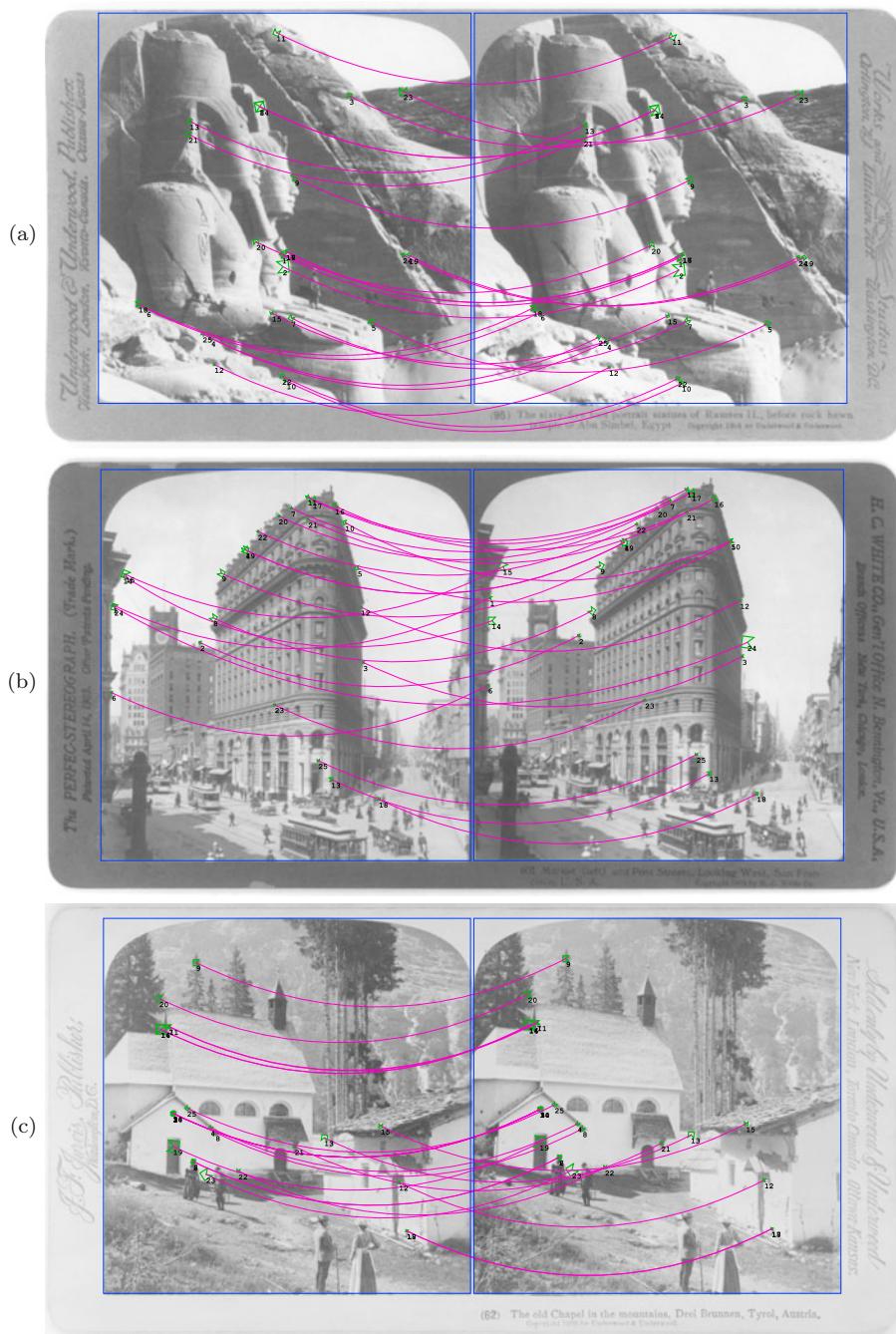


Figure 7.28 SIFT feature matching examples on pairs of stereo images. Shown are the 25 best matches obtained with the L₂ feature distance and $\bar{\rho}_{\text{match}} = 0.8$.

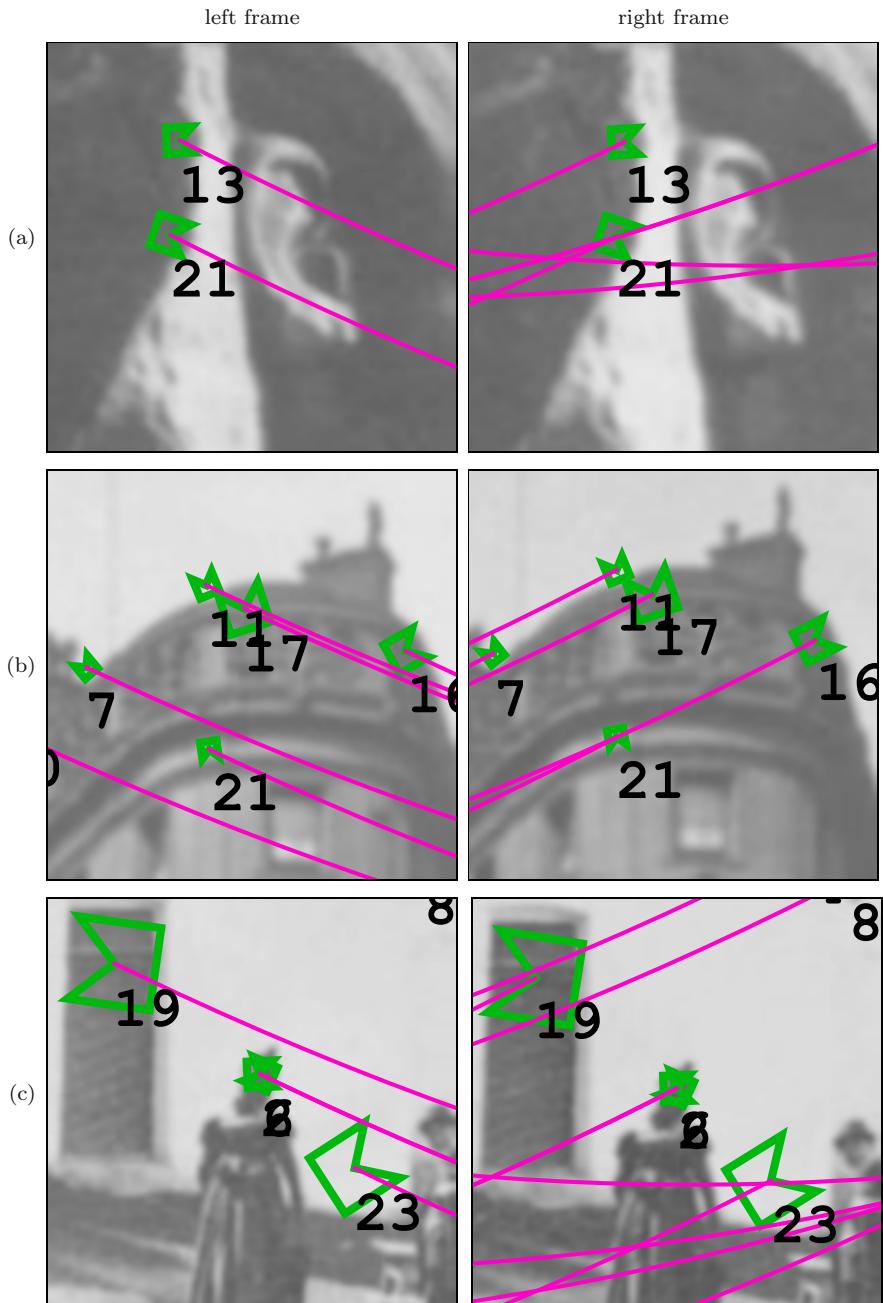


Figure 7.29 Stereo matching examples (enlarged details from Fig. 7.28).

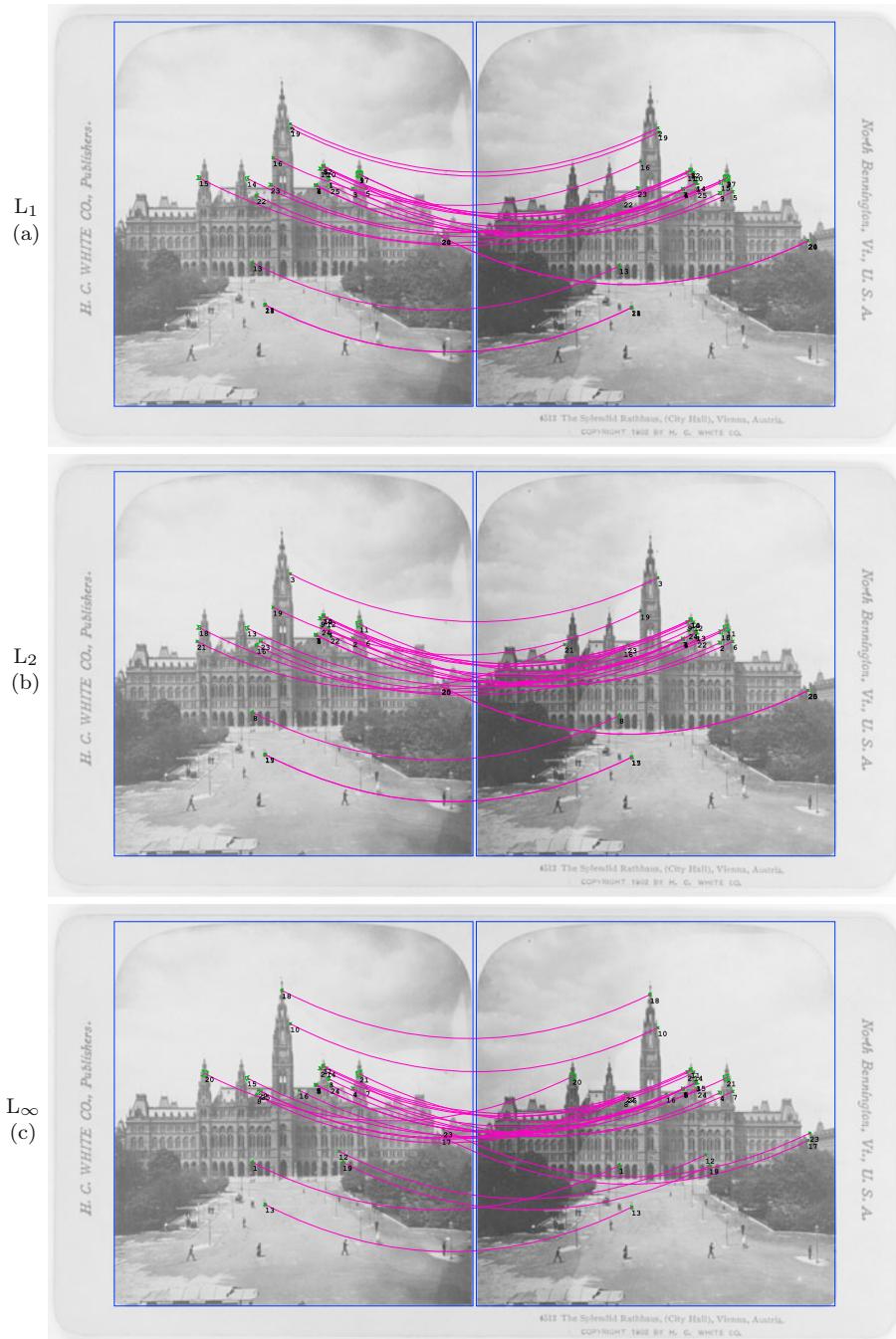


Figure 7.30 Using different distance norms for feature matching. L₁ (a), L₂ (b) and L_∞ norm (c). All other parameters are set to their default values (see Table 7.5).



Figure 7.31 Rejection of weak or ambiguous matches by limiting the ratio of primary and secondary match distance $\bar{\rho}_{\text{match}}$ (see Eqns. 7.104–7.105). $\bar{\rho}_{\text{match}} = 1.0$ (a), 0.8 (b), 0.5 (c).

numerous applications in various disciplines and continued research. Open-source implementations of several different methods are available as software libraries. For further reading, see [72, 88], for example.

7.7 SIFT implementation in Java

A new and complete Java implementation of the SIFT method has been written from ground up to complement the algorithms described in this chapter. Space limitations do not permit a full listing here, but the entire implementation and additional examples can be found in the source code section of this book’s website. Most Java methods are named and structured identically to the procedures listed in the above algorithms for easy identification. Note, however, that this implementation is again written for instructional clarity and readability. The code is neither tuned for efficiency nor is it intended to be used in a production environment.

7.7.1 SIFT feature extraction

The key class in this Java library is `SiftDetector`²⁸ which implements a SIFT detector for a given floating-point image. The following example illustrates its basic use for a given `ImageProcessor` object `ip`:

```
...
FloatProcessor I = (FloatProcessor) ip.convertToFloat();
SiftDetector sd = new SiftDetector(I);
List<SiftDescriptor> S = sd.getSiftFeatures();
...
// process descriptor set S
```

The initial work of setting up the required Gaussian and DoG scale space structures for the given image `I` is accomplished by the constructor in `new SiftDetector(I)`.

The method `getSiftFeatures()` then performs the actual feature detection process and returns a sequence of `SiftDescriptor` objects (`S`) for the image `I`. Each extracted `SiftDescriptor` in `S` holds information about its image position (`x, y`), its absolute scale σ (`scale`) and its dominant orientation θ (`orientation`). It also contains an invariant, 128-element, `int`-type feature vector f_{sift} (see [Alg. 7.8](#)).

The SIFT detector uses a large set of parameters that are set to their default values (see [Table 7.5](#)) if the simple constructor `new SiftDetector(I)` is used, as in the above example. All parameters can be adjusted individually by passing a parameter object to its constructor, as in the following example, which shows feature extraction from two images `ipa`, `ipb` using identical parameters:

²⁸ File `imagingbook.sift.SiftDetector.java`

```

...
FloatProcessor Ia = (FloatProcessor) ipa.convertToFloat();
FloatProcessor Ib = (FloatProcessor) ipb.convertToFloat();
...
SiftDetector.Parameters params = new SiftDetector.Parameters();
params.sigma_s = 0.5; // modify individual parameters
params.sigma_0 = 1.6;
...
SiftDetector sda = new SiftDetector(Ia, params);
SiftDetector sdb = new SiftDetector(Ib, params);
List<SiftDescriptor> Sa = sda.getSiftFeatures();
List<SiftDescriptor> Sb = sdb.getSiftFeatures();
...
// process descriptor sets Sa and Sb

```

7.7.2 SIFT feature matching

Finding matching descriptors from a pair of SIFT descriptor sets S_a , S_b is accomplished by the class `SiftMatcher`.²⁹ One descriptor set (S_a) is considered the “reference” or “model” set and used to initialize a new `SiftMatcher` object, as shown below. The actual matches are then calculated by invoking the method `matchDescriptors()`, which implements the procedure `MATCHDESCRIPTORS()` outlined in Alg. 7.11. It takes the second descriptor set (S_b) as the only argument. The following code segment continues from the previous example:

```

...
SiftMatcher.Parameters params = new SiftMatcher.Parameters();
// set matcher parameters here
SiftMatcher matcher = new SiftMatcher(Sa, params);
List<SiftMatch> matches = matcher.matchDescriptors(Sb);
...
// process matches

```

As noted above, certain parameters of the SIFT matcher can be set individually, for example,

```

params.norm = FeatureDistanceNorm.L1; // L1, L2, or Linf
params.ratioMax = 0.8; // =  $\bar{\rho}_{\text{match}}$ , max. ratio of best and second-best match
params.sort = true; // set to true if sorting of matches is desired

```

The method `matchDescriptors()` in this prototypical implementation performs exhaustive search over all possible descriptor pairs in the two sets S_a and S_b . To implement efficient approximate nearest-neighbor search (see Sec. 7.6), one would pre-calculate the required search tree structures for the model descriptor set once inside `SiftMatcher`’s constructor method. The same matcher object could then be reused to match against multiple descriptor sets without

²⁹ File `imagingbook.sift.SiftMatcher.java`

the need to recalculate the search tree structure over and over again. This is particularly effective when the given model set is large.

7.8 Exercises

Exercise 7.1

As claimed in Eqn. (7.12), the 2D Laplacian-of-Gaussian (LoG) function $L_\sigma(x, y)$ can be approximated by the difference of two Gaussians (DoG) in the form $L_\sigma(x, y) \approx \lambda \cdot (G_{\kappa\sigma}(x, y) - G_\sigma(x, y))$. Create a combined plot, similar to the one in Fig. 7.5 (b), showing the 1D cross sections of the LoG and DoG functions (with $\sigma = 1.0$ and $y = 0$). Compare both functions by varying the values of $\kappa = 2.00, 1.25, 1.10, 1.05$, and 1.01 . How does the approximation change as κ approaches 1, and what happens if κ becomes exactly 1?

Exercise 7.2

Evaluate the SIFT mechanism for tracking features in video sequences. Search for a suitable video sequence with good features to track and process the images frame-by-frame.³⁰ Then match the SIFT features detected in pairs of successive frames by connecting the best-matching features, as long as the “match quality” is above a predefined threshold. Could other properties of the SIFT descriptors (such as position, scale and dominant orientation) be used to improve tracking stability?

³⁰ In ImageJ, choose an AVI video short enough to fit into main memory and open it as an image stack.

Appendix

A

Mathematical Symbols and Notation

- (a_1, a_2, \dots, a_n) A *vector* or *list*, i.e., a sequence of elements of the same type. Unlike a *set* (see below), a list is ordered and may contain the same element more than once. If used to denote a (row) *vector*, $(a_1, \dots, a_n)^\top$ is the *transposed* (column) vector.¹ If used to represent a *list*,² () represents the *empty* list and (a) is a list with a single element a . $|A|$ is the *length* of the sequence A . $A \cup B$ denotes the concatenation of A, B . $A(i) \equiv a_i$ retrieves the i -th element of A . $A(i) \leftarrow x$ means that the i -th element of A is set to x .
- { a, b, c, d, \dots } A *set*, i.e., an unordered collection of distinct elements. A particular element x can be contained in a set at most once. {} denotes the empty set. $|\mathcal{A}|$ is the size (cardinality) of the set \mathcal{A} . $\mathcal{A} \cup \mathcal{B}$ is the union and $\mathcal{A} \cap \mathcal{B}$ is the intersection of two sets \mathcal{A}, \mathcal{B} . $x \in \mathcal{A}$ means that the element x is contained in \mathcal{A} .
- (A, B, C, \dots) A *tuple*, i.e., a fixed-size, ordered list of elements, each possibly of a different type.³

¹ Vectors are usually implemented as one-dimensional arrays, with elements being referred to by position (index).

² Lists are usually implemented with dynamic data structures, such as linked lists. Java's *Collections* framework provides numerous easy-to-use list implementations (see also Vol. 1, Sec. B.2.7 [20, p. 248]).

³ Tuples are typically implemented as *objects* (in Java or C++) or *structures* (in C) with elements being referred to by name.

$[a, b]$	Numeric interval; $x \in [a, b]$ means $a \leq x \leq b$. Similarly, $x \in [a, b)$ says that $a \leq x < b$.
$ A $	Size (cardinality) of a set A , $ A \equiv \text{card } A$.
$ x $	Absolute value of a scalar x .
$ \boldsymbol{x} $	Length (number of elements) of a vector or list \boldsymbol{x} .
$\ \boldsymbol{x}\ $	Euclidean (L_2) norm of a vector \boldsymbol{x} . $\ \boldsymbol{x}\ _n$ denotes the magnitude of \boldsymbol{x} using a particular norm L_n .
$\lceil x \rceil$	“Ceil” of x , the smallest integer $z \in \mathbb{Z}$ greater than $x \in \mathbb{R}$ (i. e., $z = \lceil x \rceil \geq x$). For example, $\lceil 3.141 \rceil = 4$, $\lceil -1.2 \rceil = -1$.
$\lfloor x \rfloor$	“Floor” of x , the largest integer $z \in \mathbb{Z}$ smaller than $x \in \mathbb{R}$ (i. e., $z = \lfloor x \rfloor \leq x$). For example, $\lfloor 3.141 \rfloor = 3$, $\lfloor -1.2 \rfloor = -2$.
$\lfloor \rfloor$	Rounding operator, $\lfloor x \rfloor \equiv \text{round}(x)$.
\div	Integer division operator: $a \div b$ denotes the quotient of the two integers a, b . For example, $5 \div 3 = 1$ and $-13 \div 4 = -3$ (equivalent to Java’s “ <code>/</code> ” operator in the case of integer operands).
$*$	Linear convolution operator (Vol. 1, Sec. 5.3.1).
\oplus	Morphological dilation operator (Vol. 1, Sec. 7.2.3).
\ominus	Morphological erosion operator (Vol. 1, Sec. 7.2.4).
\otimes	Cross product (between vectors or complex quantities, see p. 206).
\smile	List concatenation operator. Given two lists $A \equiv (a, b, c)$ and $B \equiv (d, e)$, $A \smile B$ denotes the concatenation of A and B , with the result (a, b, c, d, e) . Inserting a single element x at the end or front of the list A is written as $A \smile (x)$ or $(x) \smile A$, resulting in (a, b, c, x) or (x, a, b, c) , respectively.
\sim	“Similarity” relation used to denote that a random variable adheres to a particular statistical distribution; for example, $X \sim \mathcal{N}(\mu, \sigma^2)$ means that X is normally distributed with parameters μ and σ^2 (Sec. C.3).
\approx	Approximately equal.
\equiv	Equivalence relation.
\leftarrow	Assignment operator: $a \leftarrow \text{expr}$ means that expression expr is evaluated and subsequently the result is assigned to variable a .
\leftarrow^+	Incremental assignment operator: $a \leftarrow^+ b$ is equivalent to $a \leftarrow a + b$.

$:=$	Function definition operator (used in algorithms). For example, $\text{Foo}(x) := x^2 + 5$ defines a function Foo with the bound variable (formal function argument) x .
\cdots	“upto” (incrementing) iteration, used in loop constructs like for $q \leftarrow 1, \dots, K$.
\cdots	“downto” (decrementing) iteration, e.g., for $q \leftarrow K, \dots, 1$.
∂	Partial derivative operator (Vol. 1, Sec. 6.2.1). For example, $\frac{\partial}{\partial x_i} f$ denotes the <i>first</i> derivative of the multi-dimensional function $f(x_1, x_2, \dots, x_n) : \mathbb{R}^n \rightarrow \mathbb{R}$ along variable x_i , $\frac{\partial^2}{\partial x_i^2} f$ is the <i>second</i> derivative (i.e., differentiating f twice along variable x_i), etc.
∇	Gradient operator. The gradient of a multi-dimensional function $f(x_1, x_2, \dots, x_n) : \mathbb{R}^n \rightarrow \mathbb{R}$, denoted ∇f (also ∇_f or $\text{grad } f$), is the vector of its first partial derivatives (see also Sec. B.6.2).
∇^2	Laplace operator (or <i>Laplacian</i>). The Laplacian of a multi-dimensional function $f(x_1, x_2, \dots, x_n) : \mathbb{R}^n \rightarrow \mathbb{R}$, denoted $\nabla^2 f$ (or ∇_f^2), is the sum of its second partial derivatives (see Sec. B.6.5).
$\text{Arctan}(x, y)$	Inverse tangent function, similar to $\arctan(\frac{y}{x}) = \tan^{-1}(\frac{y}{x})$ but with two arguments and returning angles in the range $[-\pi, +\pi]$ (i.e., covering all four quadrants). $\text{Arctan}(x, y)$ is equivalent to the <code>ArcTan[x, y]</code> function in <i>Mathematica</i> and the standard Java method <code>Math.atan2(y, x)</code> , but note the reversed arguments!
card	Cardinality (size) of a set, $\text{card } \mathcal{A} \equiv \mathcal{A} $.
\mathbb{C}	The set of complex numbers.
dom	The <i>domain</i> of a map or function. For example, the set of possible coordinates of an image I is denoted $\text{dom}(I)$ (see also “ $M \times N$ ” and “ rng ” below).
\mathbf{e}	Unit vector. For example, $\mathbf{e}_x = (1, 0)^\top$ denotes the 2D unit vector in x -direction. $\mathbf{e}_\theta = (\cos \theta, \sin \theta)^\top$ is the 2D unit vector oriented at angle θ . Analogously, $\mathbf{e}_i, \mathbf{e}_j, \mathbf{e}_k$ are the unit vectors along the coordinate axes in 3D.
false	Boolean constant ($\neg \text{true}$).

g	A discrete <i>map</i> , for example, $\mathbf{g} = (g_0, g_1, \dots, g_{N-1})$. The operation $x \leftarrow g_i$ or, equivalently, $x \leftarrow \mathbf{g}(i)$ retrieves the value of g at index i . To <i>set</i> map values, we use $g_i \leftarrow x$ or $\mathbf{g}(i) \leftarrow x$. Note the subtle differences in typography.
grad	Gradient operator (see ∇).
$h(i)$	Histogram entry for pixel value (or bin) i (Vol. 1, Sec. 3.1).
$H(i)$	Cumulative histogram entry for pixel value (or bin) i (Vol. 1, Sec. 3.6).
\mathbf{H}	Hessian matrix (see also Sec. B.6.6).
i	Imaginary unit ($i^2 = -1$).
I	Scalar-valued 2D (intensity) image; $I(u, v) \in \mathbb{R}$.
\mathbf{I}	Vector-valued 2D (color) image; typically $\mathbf{I} = (I_R, I_G, I_B)$ and $\mathbf{I}(u, v) \in \mathbb{R}^3$.
\mathbf{I}_n	Identity matrix of size $n \times n$. For example, $\mathbf{I}_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ is the 2×2 identity matrix.
\mathbf{J}	Jacobian matrix (Sec. B.6.1).
$M \times N$	M, N are the number of columns (width) and rows (height) of an image. $M \times N$ denotes the domain of image coordinates (u, v) , used as a shortcut notation for the set $\{0, \dots, M-1\} \times \{0, \dots, N-1\}$.
mod	Modulus operator: $(a \text{ mod } b)$ is the remainder of the <i>integer</i> division a/b (Vol. 1, Sec. B.1.2).
μ	Mean value.
\mathbb{N}	The set of natural numbers; $\mathbb{N} = \{1, 2, 3, \dots\}$, $\mathbb{N}_0 = \{0, 1, 2, \dots\}$.
nil	Null (“empty”) constant, typically used in algorithms to initialize variables that refer to tuples or other objects (analogous to <code>null</code> in Java).
$p(i)$	Discrete probability density function (Vol. 1, Sec. 4.6.1).
$P(i)$	Discrete probability distribution function or cumulative probability density (Vol. 1, Sec. 4.6.1).
\mathbb{R}	The set of real numbers.
R, G, B	Red, green, blue color channels.
rng	The <i>range</i> of a map or function. For example, the set of possible pixel values of an image I is denoted $\text{rng}(I)$ (see also “dom”).
round	Rounding function: returns the integer closest to the scalar $x \in \mathbb{R}$. $\text{round}(x) = \lfloor x \rfloor = \lfloor x + 0.5 \rfloor$.

σ	Standard deviation (square root of <i>variance</i> σ^2).
sgn	“Sign” or “signum” function: $\text{sgn}(x) = \begin{cases} 1 & \text{for } x > 0 \\ 0 & \text{for } x = 0 \\ -1 & \text{for } x < 0 \end{cases}$
τ	Time period.
t	Continuous time variable.
t	Threshold value.
\top	Transpose operator. \boldsymbol{x}^\top denotes the transpose of the vector \boldsymbol{x} , similarly \boldsymbol{A}^\top is the transpose of the matrix \boldsymbol{A} .
$\text{trace}(\boldsymbol{A})$	Trace (sum of the diagonal elements) of the matrix \boldsymbol{A} .
true	Boolean constant ($\text{true} = \neg\text{false}$).
truncate	Truncation function: truncates x toward zero to the closest integer. For example, $\text{truncate}(3.141) = 3$, $\text{truncate}(-2.5) = -2$.
$\boldsymbol{u} = (u, v)$	Discrete 2D coordinate variable with $u, v \in \mathbb{Z}$. A fixed coordinate point is denoted $\dot{\boldsymbol{u}} = (\dot{u}, \dot{v})$.
$\boldsymbol{x} = (x, y)$	Continuous 2D coordinate variable with $x, y \in \mathbb{R}$. A fixed coordinate point is denoted $\dot{\boldsymbol{x}} = (\dot{x}, \dot{y})$.
\mathbb{Z}	The set of integers.

B

Vector Algebra and Calculus

This part collects a small set of elementary tools and concepts from algebra and calculus that are referenced in the main text.

B.1 Vectors

Here we describe the basic notation for vectors in 2D and 3D. Let

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad (\text{B.1})$$

denote vectors \mathbf{a}, \mathbf{b} in 2D, or similarly

$$\mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \quad (\text{B.2})$$

for vectors in 3D. Vectors are used to describe 2D or 3D points or the displacement between points. We commonly use upper-case letters to denote a *matrix*, for example,

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix}. \quad (\text{B.3})$$

The above matrix consists of 3 rows and 2 columns. In other words, \mathbf{A} is of size 3×2 . Its elements are referred to as A_{ij} , where i is the *row* index (vertical coordinate) and j is the *column* index (horizontal coordinate).¹

The *transpose* of \mathbf{A} , denoted \mathbf{A}^\top , is obtained by exchanging rows and columns, that is

$$\mathbf{A}^\top = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix}^\top = \begin{pmatrix} A_{11} & A_{21} & A_{31} \\ A_{12} & A_{22} & A_{32} \end{pmatrix}. \quad (\text{B.4})$$

B.1.1 Column and row vectors

For practical purposes, a vector can be considered a special case of a matrix. In particular, a *column* vector

$$\mathbf{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix} \quad (\text{B.5})$$

corresponds to a matrix of size $(m, 1)$, while its transpose \mathbf{a}^\top is a *row* vector and is thus equivalent to a matrix of size $(1, m)$. By default and unless otherwise noted any vector is implicitly assumed to be a *column* vector.

B.1.2 Vector length

The *length* or *Euclidean norm* (L_2 norm) of a vector $\mathbf{x} = (x_1, \dots, x_n)^\top$, denoted $\|\mathbf{x}\|$, is defined as

$$\|\mathbf{x}\| = \left(\sum_{i=1}^n x_i^2 \right)^{\frac{1}{2}}. \quad (\text{B.6})$$

For example, for a 3D vector $\mathbf{x} = (x, y, z)^\top$, this can be written as

$$\|\mathbf{x}\| = \sqrt{x^2 + y^2 + z^2}. \quad (\text{B.7})$$

B.2 Matrix multiplication

B.2.1 Scalar multiplication

Multiplication of a real-valued matrix \mathbf{A} and a scalar value $s \in \mathbb{R}$ is defined as

$$s \cdot \mathbf{A} = \mathbf{A} \cdot s = \begin{pmatrix} s \cdot A_{11} & s \cdot A_{12} \\ s \cdot A_{21} & s \cdot A_{22} \\ s \cdot A_{31} & s \cdot A_{32} \end{pmatrix}. \quad (\text{B.8})$$

¹ Note that the usual ordering of matrix coordinates is (unlike image coordinates) vertical-first!

B.2.2 Product of two matrices

We say that a matrix is of size (r, c) if consists of r rows and c columns. Given two matrices, \mathbf{A} , \mathbf{B} of size (m, n) and (p, q) , respectively, the product $\mathbf{A} \cdot \mathbf{B}$ is only defined if $n = p$. Thus the number of columns (n) in \mathbf{A} must always match the number of rows (p) in \mathbf{B} . The result is a new matrix \mathbf{C} of size (m, q) ,

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \underbrace{\begin{pmatrix} A_{11} & \dots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \dots & A_{mn} \end{pmatrix}}_{(m \times n)} \cdot \underbrace{\begin{pmatrix} B_{11} & \dots & B_{1q} \\ \vdots & \ddots & \vdots \\ B_{n1} & \dots & B_{nq} \end{pmatrix}}_{(n \times q)} = \underbrace{\begin{pmatrix} C_{11} & \dots & C_{1q} \\ \vdots & \ddots & \vdots \\ C_{m1} & \dots & C_{mq} \end{pmatrix}}_{(m \times q)}, \quad (\text{B.9})$$

with elements

$$C_{ij} = \sum_{k=1}^n A_{ik} \cdot B_{kj}, \quad (\text{B.10})$$

for $i = 1, \dots, m$ and $j = 1, \dots, q$. Note that this product is *not* commutative, that is, $\mathbf{A} \cdot \mathbf{B} \neq \mathbf{B} \cdot \mathbf{A}$ in general.

B.2.3 Matrix-vector products

The ordinary product $\mathbf{A} \cdot \mathbf{x}$ between a matrix \mathbf{A} and a vector \mathbf{x} is only a special case of a matrix-matrix multiplication given in Eqn. (B.9). In particular, if $\mathbf{x} = (x_1, \dots, x_n)^\top$ is a n -dimensional *column* vector (i.e., a matrix of size $n \times 1$), then the *left* (or *pre-*) multiplication is

$$\underbrace{\mathbf{A}}_{(m \times n)} \cdot \underbrace{\mathbf{x}}_{(n \times 1)} = \underbrace{\mathbf{y}}_{(m \times 1)} \quad (\text{B.11})$$

is only defined if the matrix \mathbf{A} is of size $m \times n$, for arbitrary $m \geq 1$. The result \mathbf{y} is a *column* vector of length m (equivalent to a matrix of size $m \times 1$). For example (with $m = 2$, $n = 3$),

$$\mathbf{A} \cdot \mathbf{x} = \underbrace{\begin{pmatrix} A & B & C \\ D & E & F \end{pmatrix}}_{2 \times 3} \cdot \underbrace{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}_{3 \times 1} = \underbrace{\begin{pmatrix} Ax + By + Cz \\ Dx + Ey + Fz \end{pmatrix}}_{2 \times 1}. \quad (\text{B.12})$$

Similarly, a *right* (or *post-*) multiplication of a *row* vector \mathbf{x}^\top of length m with a matrix \mathbf{B} of size $m \times n$ is accomplished in the form

$$\underbrace{\mathbf{x}^\top}_{1 \times m} \cdot \underbrace{\mathbf{B}}_{m \times n} = \underbrace{\mathbf{z}}_{1 \times n}, \quad (\text{B.13})$$

where the result \mathbf{z} is a n -dimensional *row* vector. For example (again with $m = 2$, $n = 3$),

$$\mathbf{x}^\top \cdot \mathbf{B} = \underbrace{(x, y)}_{1 \times 2} \cdot \underbrace{\begin{pmatrix} A & B & C \\ D & E & F \end{pmatrix}}_{2 \times 3} = \underbrace{(xA+yD, xB+yE, xC+yF)}_{1 \times 3}. \quad (\text{B.14})$$

In general, if $\mathbf{A} \cdot \mathbf{x}$ is defined, then

$$\mathbf{A} \cdot \mathbf{x} = (\mathbf{x}^\top \cdot \mathbf{A}^\top)^\top \quad \text{and} \quad (\mathbf{A} \cdot \mathbf{x})^\top = \mathbf{x}^\top \cdot \mathbf{A}^\top, \quad (\text{B.15})$$

that is, any right-sided matrix-vector product can also be calculated as a left-sided product by transposing the corresponding matrices and vectors.

B.3 Vector products

Products between vectors are a common cause of confusion, mainly because the same symbol (\cdot) is used to denote widely different operators.

B.3.1 Dot product

The *dot* product (also called *scalar* or *inner* product) of two vectors $\mathbf{a} = (a_1, \dots, a_n)^\top$, $\mathbf{b} = (b_1, \dots, b_n)^\top$ of the same length n is defined as

$$x = \mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i \cdot b_i. \quad (\text{B.16})$$

Thus the result x is a scalar value. If we write this as the product of a row and a column vector, as in Eqn. (B.13),

$$\underbrace{x}_{1 \times 1} = \underbrace{\mathbf{a}^\top}_{1 \times n} \cdot \underbrace{\mathbf{b}}_{n \times 1}, \quad (\text{B.17})$$

we conclude that the result x is a matrix of size 1×1 , i. e., a single scalar value.

As a special case, the dot product of a vector with itself gives the square of its length (see Eqn. (B.6)), that is,

$$\mathbf{x} \cdot \mathbf{x} = \mathbf{x}^\top \cdot \mathbf{x} = \sum_{i=1}^n x_i^2 = \|\mathbf{x}\|^2. \quad (\text{B.18})$$

Note that the dot product is zero if the two vectors are orthogonal to each other.

B.3.2 Outer product

The outer product of two vectors $\mathbf{a} = (a_1, \dots, a_m)^\top$, $\mathbf{b} = (b_1, \dots, b_n)^\top$ of length m and n , respectively, is defined as

$$\mathbf{M} = \mathbf{a} \otimes \mathbf{b} = \mathbf{a} \cdot \mathbf{b}^\top = \begin{pmatrix} a_1 b_1 & a_1 b_2 & \dots & a_1 b_n \\ a_2 b_1 & a_2 b_2 & \dots & a_2 b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_m b_1 & a_m b_2 & \dots & a_m b_n \end{pmatrix}. \quad (\text{B.19})$$

Thus the result is a *matrix* \mathbf{M} with m rows and n columns and elements $M_{ij} = a_i \cdot b_j$, for $i = 1, \dots, m$ and $j = 1, \dots, n$. Note that $\mathbf{a} \cdot \mathbf{b}^\top$ in Eqn. (B.19) denotes the ordinary (matrix) product of the column vector \mathbf{a} (of size $m \times 1$) and the row vector \mathbf{b}^\top (of size $1 \times n$), as defined in Eqn. (B.9). The outer product is a special case of the *Kronecker* product, denoted by \otimes , which can be applied to pairs of matrices.

B.4 Eigenvectors and eigenvalues²

In general, the eigenvalue problem is to find solutions $\mathbf{x} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$ for the linear equation

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}, \quad (\text{B.20})$$

with the given square matrix \mathbf{A} of size $n \times n$. Any non-trivial³ solution $\mathbf{x} = \mathbf{e}_j$ is called *eigenvector* and the corresponding scalar $\lambda = \lambda_j$ (which may be complex-valued) is called *eigenvalue*, respectively. Eigenvalue and eigenvectors thus always come in pairs, usually called *eigenpairs*.

Geometrically speaking, applying the matrix \mathbf{A} to an eigenvector only changes the vector's *magnitude* or *length* (by the corresponding λ), but not its orientation in space.

Eqn. (B.20) can be rewritten as $\mathbf{A}\mathbf{x} - \lambda\mathbf{x} = \mathbf{0}$ or

$$(\mathbf{A} - \lambda\mathbf{I}_n)\mathbf{x} = \mathbf{0}, \quad (\text{B.21})$$

where \mathbf{I}_n is the $n \times n$ identity matrix. This homogeneous linear equation has non-trivial solutions only if the matrix $(\mathbf{A} - \lambda\mathbf{I}_n)$ is *singular*, i. e., its rank is *less* than n and thus its determinant $\det()$ is zero, that is,

$$\det(\mathbf{A} - \lambda\mathbf{I}_n) = 0. \quad (\text{B.22})$$

² The content of this section is inspired by the excellent presentations in [15] and [37, Sec. A.2.7].

³ An obvious but trivial solution is $\mathbf{x} = \mathbf{0}$ (where $\mathbf{0}$ denotes the zero-vector).

Eqn. (B.22) is called the “characteristic equation” of the matrix \mathbf{A} , which can be expanded to a polynomial in λ ,

$$\det(\mathbf{A} - \lambda \cdot \mathbf{I}_n) = \begin{vmatrix} a_{11} - \lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} - \lambda \end{vmatrix} \quad (\text{B.23})$$

$$= (-1)^n \cdot [\lambda^n + c_1 \cdot \lambda^{n-1} + c_2 \cdot \lambda^{n-2} + \cdots + c_{n-1} \cdot \lambda^1 + c_n] = 0. \quad (\text{B.24})$$

This function over the variable λ (the coefficients c_i are obtained from the sub-determinants of \mathbf{A}) is a polynomial of n -th degree for a matrix of size $n \times n$. It thus has a maximum of n distinct roots, i. e., solutions of the above equation, which are the eigenvalues of the matrix \mathbf{A} . A matrix of size $n \times n$ thus has up to n non-distinct eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n$, each with an associated eigenvalue $\lambda_1, \lambda_2, \dots, \lambda_n$. As mentioned, eigenvectors and eigenvalues always come in *pairs* $\langle \lambda_j, \mathbf{x}_j \rangle$, called “eigenpairs”. Note that the eigenvalues of an arbitrary matrix may be complex. However (as an important special case), if the matrix \mathbf{A} is *real* and *symmetric*, all its eigenvalues are guaranteed to be *real*.

Example

For the 2×2 matrix

$$\mathbf{A} = \begin{pmatrix} 3 & -2 \\ -4 & 1 \end{pmatrix}, \quad (\text{B.25})$$

the corresponding eigenvalues and the (two possible) eigenvectors are

$$\begin{aligned} \lambda_1 &= 5, & \lambda_2 &= -1, \\ \mathbf{x}_1 &= \begin{pmatrix} 4 \\ -4 \end{pmatrix}, & \mathbf{x}_2 &= \begin{pmatrix} -2 \\ -4 \end{pmatrix}. \end{aligned}$$

The result can be easily verified by inserting pairs λ_k, \mathbf{x}_k into Eqn. (B.20).

B.4.1 Eigenvectors of a 2×2 matrix

In practice, computing the eigenvalues for arbitrary $n \times n$ matrices is done numerically with suitable software packages, e. g., *JAMA*⁴ or the *Apache Commons Math*⁵ library for Java. For the special (but frequent) case of $n = 2$, the

⁴ <http://math.nist.gov/javanumerics/jama/>

⁵ <http://commons.apache.org/proper/commons-math/>

solution can be found in closed form. In this case, the characteristic equation (Eqn. (B.23)) reduces to

$$\det(\mathbf{A} - \lambda \cdot \mathbf{I}_2) = \left| \begin{pmatrix} A & B \\ C & D \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \right| = \left| \begin{matrix} A-\lambda & B \\ C & D-\lambda \end{matrix} \right| = \lambda^2 - (A + D) \cdot \lambda + (AD - BC) = 0. \quad (\text{B.26})$$

The two possible solutions to this equation,

$$\begin{aligned} \lambda_{1,2} &= \frac{A + D}{2} \pm \left[\left(\frac{A + D}{2} \right)^2 - (AD - BC) \right]^{\frac{1}{2}} \\ &= \frac{A + D}{2} \pm \left[\left(\frac{A - D}{2} \right)^2 + BC \right]^{\frac{1}{2}} \\ &= R \pm \sqrt{S^2 + BC} \end{aligned} \quad (\text{B.27})$$

are the eigenvalues of \mathbf{A} , with

$$\begin{aligned} \lambda_1 &= R + \sqrt{S^2 + BC} \\ \lambda_2 &= R - \sqrt{S^2 + BC}. \end{aligned} \quad (\text{B.28})$$

Both λ_1, λ_2 are real-valued if the term under the square root is positive, i. e., if

$$S^2 + BC = \left(\frac{A - D}{2} \right)^2 + BC \geq 0. \quad (\text{B.29})$$

In particular, if the matrix is *symmetric* (i. e., $B = C$), this condition is guaranteed (because $BC \geq 0$). In this case, $\lambda_1 \geq \lambda_2$. [Algorithm B.1](#)⁶ summarizes the closed-form computation of the eigenvalues and eigenvectors of a 2×2 matrix.

B.5 Parabolic fitting

B.5.1 Fitting a parabolic function to three sample points

Given a single-variable (one-dimensional), discrete function $g: \mathbb{Z} \rightarrow \mathbb{R}$, it is sometimes useful to locally fit a quadratic (parabolic) function, for example, for precisely locating a maximum or minimum position. For a quadratic function (second-order polynomial)

$$y = f(x) = a \cdot x^2 + b \cdot x + c \quad (\text{B.30})$$

⁶ See [15] and its reprint in [16, Ch. 5].

Algorithm B.1 Computing the real eigenvalues and eigenvectors for a 2×2 real-valued matrix \mathbf{A} . If the matrix has real eigenvalues, an ordered sequence of two “eigenpairs” $(\lambda_i, \mathbf{x}_i)$, each containing the eigenvalue λ_i and the associated eigenvector \mathbf{x}_i , is returned ($i = 1, 2$). The resulting sequence is ordered by decreasing eigenvalues. If \mathbf{A} has no real eigenvalues, nil is returned.

```

1: REALEIGENVALUES2X2 ( $\mathbf{A}$ )                                      $\triangleright \mathbf{A} = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ 
2:    $R \leftarrow \frac{A+D}{2}$ ,  $S \leftarrow \frac{A-D}{2}$ 
3:   if  $(S^2 + BC) < 0$  then
4:     return nil.                                          $\triangleright \mathbf{A}$  has no real-valued eigenvalues
5:   else
6:      $T \leftarrow \sqrt{S^2 + BC}$ 
7:      $\lambda_1 \leftarrow R + T$ ,  $\lambda_2 \leftarrow R - T$             $\triangleright$  eigenvalues  $\lambda_1, \lambda_2$ 
8:     if  $(A - D) \geq 0$  then                          $\triangleright$  eigenvectors  $\mathbf{x}_1, \mathbf{x}_2$ 
9:        $\mathbf{x}_1 \leftarrow \begin{pmatrix} S+T \\ C \end{pmatrix}$ ,  $\mathbf{x}_2 \leftarrow \begin{pmatrix} B \\ -S-T \end{pmatrix}$ 
10:    else
11:       $\mathbf{x}_1 \leftarrow \begin{pmatrix} B \\ -S+T \end{pmatrix}$ ,  $\mathbf{x}_2 \leftarrow \begin{pmatrix} S-T \\ C \end{pmatrix}$ 
12:    return  $(\langle \lambda_1, \mathbf{x}_1 \rangle, \langle \lambda_2, \mathbf{x}_2 \rangle)$ .           $\triangleright \lambda_1 \geq \lambda_2$ 
```

to pass through a given set of three sample points $\mathbf{p}_i = (x_i, y_i)$, $i = 1, 2, 3$, means that the following three equations must be satisfied:

$$\begin{aligned} y_1 &= a \cdot x_1^2 + b \cdot x_1 + c, \\ y_2 &= a \cdot x_2^2 + b \cdot x_2 + c, \\ y_3 &= a \cdot x_3^2 + b \cdot x_3 + c. \end{aligned} \tag{B.31}$$

Written in the standard matrix form $\mathbf{A} \cdot \mathbf{x} = \mathbf{B}$, or

$$\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}, \tag{B.32}$$

the unknown coefficient vector $\mathbf{x} = (a, b, c)^\top$ is directly found as

$$\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{B} = \begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}, \tag{B.33}$$

assuming that the matrix \mathbf{A} has a non-zero determinant. Geometrically this means that the points \mathbf{p}_i are not collinear.

Example. Fitting the sample points $\mathbf{p}_1 = (-2, 5)^\top$, $\mathbf{p}_2 = (-1, 6)^\top$, $\mathbf{p}_3 = (3, -10)^\top$ to a quadratic function, the equation to solve is

$$\begin{pmatrix} 4 & -2 & 1 \\ 1 & -1 & 1 \\ 9 & 3 & 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 5 \\ 6 \\ -10 \end{pmatrix}, \quad (\text{B.34})$$

with the solution

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 4 & -2 & 1 \\ 1 & -1 & 1 \\ 9 & 3 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 5 \\ 6 \\ -10 \end{pmatrix} = \frac{1}{20} \cdot \begin{pmatrix} 4 & -5 & 1 \\ -8 & 5 & 3 \\ -12 & 30 & 2 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 6 \\ -10 \end{pmatrix} = \begin{pmatrix} -1 \\ -2 \\ 5 \end{pmatrix}. \quad (\text{B.35})$$

Thus $a = -1$, $b = -2$, $c = 5$, and the equation of the quadratic fitting function is $y = -x^2 - 2x + 5$. The result for this example is shown graphically in Fig. B.1.

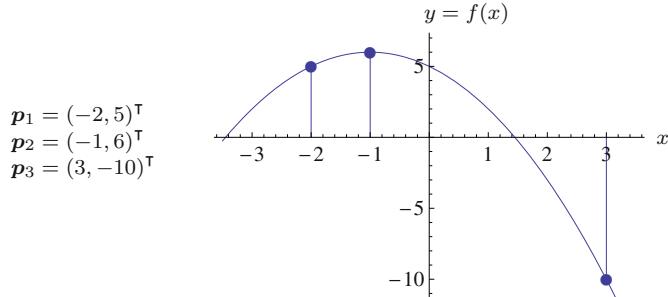


Figure B.1 Fitting a quadratic function to three arbitrary sample points.

B.5.2 Parabolic interpolation

A special situation is when the given points are positioned at $x_1 = -1$, $x_2 = 0$, and $x_3 = +1$. This is useful, for example, to estimate a continuous extremum position from successive discrete function values defined on a regular lattice. Again the objective is to fit a quadratic function

$$y = f(x) = a \cdot x^2 + b \cdot x + c \quad (\text{B.36})$$

to pass through the points $\mathbf{p}_1 = (-1, y_1)^\top$, $\mathbf{p}_2 = (0, y_2)^\top$, and $\mathbf{p}_3 = (1, y_3)^\top$. In this case, the simultaneous equations in Eqn. (B.31) simplify to

$$\begin{aligned} y_1 &= a - b + c, \\ y_2 &= \quad \quad \quad c, \end{aligned} \quad (\text{B.37})$$

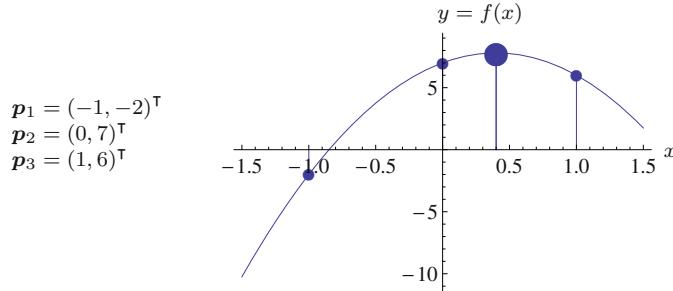


Figure B.2 Fitting a quadratic function to regularly spaced points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$. The interpolated, continuous curve has a maximum at $\check{x} = 0.4$ (large dot).

$$y_3 = a + b + c,$$

with the solution

$$a = \frac{y_1 - 2 \cdot y_2 + y_3}{2}, \quad b = \frac{y_3 - y_1}{2}, \quad c = y_2. \quad (\text{B.38})$$

To estimate a local extremum position, we take the first derivative of the quadratic fitting function (Eqn. (B.30)), which is the linear function $f'(x) = 2a \cdot x + b$, and find the position \check{x} of its (single) root by solving

$$f'(\check{x}) = 2a \cdot \check{x} + b = 0. \quad (\text{B.39})$$

With a, b taken from Eqn. (B.38), the extremal position is thus found at

$$\check{x} = \frac{-b}{2a} = \frac{y_1 - y_3}{2 \cdot (y_1 - 2y_2 + y_3)}. \quad (\text{B.40})$$

The corresponding extremal *value* can then be found by evaluating the quadratic function $f()$ at position \check{x} , i.e.,

$$\check{y} = f(\check{x}) = a \cdot \check{x}^2 + b \cdot \check{x} + c, \quad (\text{B.41})$$

with a, b, c as defined in Eqn. (B.38). [Figure B.2](#) shows an example with sample points $\mathbf{p}_1 = (-1, -2)^T$, $\mathbf{p}_2 = (0, 7)^T$, $\mathbf{p}_3 = (1, 6)^T$. In this case, the interpolated maximum position is at $\check{x} = 0.4$ and $f(\check{x}) = 7.8$.

Using the above scheme, we can interpolate any triplet of successive sample values centered around some position $u \in \mathbb{Z}$, i.e., $\mathbf{p}_1 = (u-1, y_1)^T$, $\mathbf{p}_2 = (u, y_2)^T$, $\mathbf{p}_3 = (u+1, y_3)^T$, with arbitrary values y_1, y_2, y_3 . In this case the estimated position of the extremum is simply

$$\check{x} = u + \frac{y_1 - y_3}{2 \cdot (y_1 - 2 \cdot y_2 + y_3)}. \quad (\text{B.42})$$

B.6 Vector fields

An RGB color image $\mathbf{I}(u, v) = (I_R(u, v), I_G(u, v), I_B(u, v))$ can be considered a two-dimensional function whose values are three-dimensional vectors. Mathematically, this is a special case of a vector-valued function $\mathbf{f}: \mathbb{R}^n \rightarrow \mathbb{R}^m$,

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(x_1, \dots, x_n) = \begin{pmatrix} f_1(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{pmatrix}, \quad (\text{B.43})$$

which is composed of m scalar-valued functions $f_i: \mathbb{R}^n \rightarrow \mathbb{R}$, each being defined on the domain of n -dimensional vectors $\mathbf{x} = (x_1, \dots, x_n)$. A multi-variable, scalar-valued function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is called a *scalar field*, while a vector-valued function $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ is referred to as a *vector field*. For example, a (continuous) two-dimensional grayscale image can be seen as a scalar field, and a two-dimensional RGB color image as a vector field with $n = 2$ and $m = 3$.

B.6.1 Jacobian matrix

Assuming that the function $\mathbf{f}(\mathbf{x})$ is differentiable, the so-called *functional* or *Jacobian* matrix at a given point $\dot{\mathbf{x}} = (\dot{x}_1, \dots, \dot{x}_n)$ is defined as

$$\mathbf{J}_{\mathbf{f}}(\dot{\mathbf{x}}) = \begin{pmatrix} \frac{\partial}{\partial x_1} f_1(\dot{\mathbf{x}}) & \cdots & \frac{\partial}{\partial x_n} f_1(\dot{\mathbf{x}}) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} f_m(\dot{\mathbf{x}}) & \cdots & \frac{\partial}{\partial x_n} f_m(\dot{\mathbf{x}}) \end{pmatrix}. \quad (\text{B.44})$$

The Jacobian matrix $\mathbf{J}_{\mathbf{f}}$ is of size $m \times n$ and composed of the first derivatives of the m component functions with respect to each of the n independent variables x_1, \dots, x_n . Thus each of its elements $\frac{\partial}{\partial x_j} f_i(\dot{\mathbf{x}})$ quantifies how much the value of the scalar-valued component function $f_i(\mathbf{x}) = f_i(x_1, \dots, x_n)$ changes when only variable x_j is varied and all other variables remain fixed. Note that the matrix $\mathbf{J}_{\mathbf{f}}(\mathbf{x})$ is not constant for a given function \mathbf{f} but is different at each position $\dot{\mathbf{x}}$. In general, the Jacobian matrix is neither square (unless $m = n$) nor symmetric.

B.6.2 Gradient

Gradient of a scalar field

The gradient of a *scalar field* $f: \mathbb{R}^n \rightarrow \mathbb{R}$, with $f(\mathbf{x}) = f(x_1, \dots, x_n)$, at some point $\dot{\mathbf{x}} \in \mathbb{R}^n$ is defined as

$$(\nabla f)(\dot{\mathbf{x}}) = (\text{grad } f)(\dot{\mathbf{x}}) = \begin{pmatrix} \frac{\partial}{\partial x_1} f(\dot{\mathbf{x}}) \\ \vdots \\ \frac{\partial}{\partial x_n} f(\dot{\mathbf{x}}) \end{pmatrix}. \quad (\text{B.45})$$

The resulting vector-valued function quantifies the amount of output change with respect to changing any of the input variables x_1, \dots, x_n at position $\dot{\mathbf{x}}$. Thus the gradient of a scalar field is a vector field. The *directional* gradient of a scalar field describes how the (scalar) function value changes when the coordinates are modified along a particular direction, specified by the unit vector \mathbf{e} . We denote the directional gradient as $\nabla_{\mathbf{e}} f$ and define

$$(\nabla_{\mathbf{e}} f)(\dot{\mathbf{x}}) = (\nabla f)(\dot{\mathbf{x}}) \cdot \mathbf{e}, \quad (\text{B.46})$$

where \cdot is the dot (scalar) product. The result is a scalar value that can be interpreted as the slope of the tangent on the n -dimensional surface of the scalar field at position $\dot{\mathbf{x}}$ along the direction specified by the n -dimensional unit vector $\mathbf{e} = (e_1, \dots, e_n)^T$.

Gradient of a vector field

To calculate the gradient of a *vector field* $\mathbf{f}: \mathbb{R}^n \rightarrow \mathbb{R}^m$, we note that each row i in the $m \times n$ Jacobian matrix \mathbf{J}_f (Eqn. (B.44)) is the transposed gradient vector of the corresponding component function f_i , i.e.,

$$\mathbf{J}_f(\dot{\mathbf{x}}) = \begin{pmatrix} (\nabla f_1)(\dot{\mathbf{x}})^T \\ \vdots \\ (\nabla f_m)(\dot{\mathbf{x}})^T \end{pmatrix}, \quad (\text{B.47})$$

and thus the Jacobian matrix is equivalent to the gradient of the vector field \mathbf{f} ,

$$(\text{grad } \mathbf{f})(\dot{\mathbf{x}}) \equiv \mathbf{J}_f(\dot{\mathbf{x}}). \quad (\text{B.48})$$

Analogous to Eqn. (B.46), the *directional* gradient of the vector field is then defined as

$$(\text{grad}_{\mathbf{e}} \mathbf{f})(\dot{\mathbf{x}}) \equiv \mathbf{J}_f(\dot{\mathbf{x}}) \cdot \mathbf{e}, \quad (\text{B.49})$$

where \mathbf{e} is again a unit vector specifying the gradient direction and \cdot is the ordinary matrix-vector product.

B.6.3 Maximum gradient direction

In case of a scalar field $f(\mathbf{x})$, a resulting non-zero gradient vector $(\nabla f)(\dot{\mathbf{x}})$ (Eqn. (B.45)) is also the direction of the steepest ascent of $f(\mathbf{x})$ at position $\dot{\mathbf{x}}$.⁷ In this case, the norm⁸ of the gradient vector $\|(\nabla f)(\dot{\mathbf{x}})\|$ corresponds to the maximum slope of f at point $\dot{\mathbf{x}}$.

⁷ If the gradient vector is zero, i.e., $(\nabla f)(\dot{\mathbf{x}}) = \mathbf{0}$, the direction of the gradient is undefined at position $\dot{\mathbf{x}}$.

⁸ The norm of a vector $\mathbf{x} = (x_1, \dots, x_m)$ is $\|\mathbf{x}\| \equiv \sqrt{\mathbf{x} \cdot \mathbf{x}} = \sqrt{x_1^2 + \dots + x_m^2}$.

In case of a vector field $\mathbf{f}(\mathbf{x})$, the direction of maximum slope cannot be obtained directly, since the gradient is not an n -dimensional vector but its $m \times n$ Jacobian matrix. In this case, the direction of maximum change in the function \mathbf{f} is found as the eigenvector corresponding to the largest eigenvalue of

$$\mathbf{M} = \mathbf{J}_f(\dot{\mathbf{x}})^\top \cdot \mathbf{J}_f(\dot{\mathbf{x}}), \quad (\text{B.50})$$

which is an $n \times n$ matrix.

B.6.4 Divergence

If the vector field \mathbf{f} maps \mathbb{R}^n to itself (i.e., $n = m$), its *divergence* (div) is defined as

$$(\text{div } \mathbf{f})(\dot{\mathbf{x}}) = \frac{\partial}{\partial x_1} f_1(\dot{\mathbf{x}}) + \cdots + \frac{\partial}{\partial x_n} f_n(\dot{\mathbf{x}}) = \sum_{i=1}^n \frac{\partial}{\partial x_i} f_i(\dot{\mathbf{x}}) \in \mathbb{R}, \quad (\text{B.51})$$

for a given point $\dot{\mathbf{x}}$. The result is a scalar value and therefore $(\text{div } \mathbf{f})(\dot{\mathbf{x}})$ yields a scalar field $\mathbb{R}^n \rightarrow \mathbb{R}$. Note that, in this case, the Jacobian matrix \mathbf{J}_f in Eqn. (B.44) is square (of size $n \times n$) and $\text{div } \mathbf{f}$ is equivalent to the trace of \mathbf{J}_f .

B.6.5 Laplacian

The *Laplacian* (or Laplace operator) of a scalar field $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is denoted $\nabla^2 f$, or ∇_f^2 for short. The result of applying ∇^2 to the function f is another function $\mathbb{R}^n \rightarrow \mathbb{R}$, which calculates the sum of all unmixed second partial derivatives of f . Evaluating ∇_f^2 at a particular position $\dot{\mathbf{x}}$,

$$\nabla_f^2(\dot{\mathbf{x}}) = \frac{\partial^2}{\partial x_1^2} f(\dot{\mathbf{x}}) + \cdots + \frac{\partial^2}{\partial x_n^2} f(\dot{\mathbf{x}}) = \sum_{i=1}^n \frac{\partial^2}{\partial x_i^2} f(\dot{\mathbf{x}}) \in \mathbb{R}, \quad (\text{B.52})$$

returns a scalar value. Note that this is identical to the *divergence* (Eqn. (B.51)) of the *gradient* (Eqn. (B.45)) of the scalar-valued function f , that is

$$\nabla_f^2(\dot{\mathbf{x}}) = (\text{div } \nabla f)(\dot{\mathbf{x}}). \quad (\text{B.53})$$

The *Laplacian* is also found as the *trace* of the function's Hessian matrix \mathbf{H}_f (see below).⁹

⁹ Alternative notations for the Laplacian are Δf , $\nabla \cdot (\nabla f)$, and $(\nabla \cdot \nabla) f$, where “ \cdot ” denotes the inner product.

For a *vector*-valued function $\mathbf{f}: \mathbb{R}^n \rightarrow \mathbb{R}^m$, the Laplacian at point $\dot{\mathbf{x}}$ is an m -dimensional vector,

$$\nabla_{\mathbf{f}}^2(\dot{\mathbf{x}}) = \begin{pmatrix} \nabla_{f_1}^2(\dot{\mathbf{x}}) \\ \nabla_{f_2}^2(\dot{\mathbf{x}}) \\ \vdots \\ \nabla_{f_m}^2(\dot{\mathbf{x}}) \end{pmatrix} \in \mathbb{R}^m. \quad (\text{B.54})$$

B.6.6 The Hessian matrix

The Hessian matrix of a n -variable, real-valued function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is the $n \times n$ square matrix composed of its second-order partial derivatives (assuming they all exist), that is

$$\mathbf{H}_f = \begin{pmatrix} H_{11} & H_{12} & \cdots & H_{1n} \\ H_{21} & H_{22} & \cdots & H_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ H_{n1} & H_{n2} & \cdots & H_{nn} \end{pmatrix} = \begin{pmatrix} \frac{\partial^2}{\partial x_1^2} f & \frac{\partial^2}{\partial x_1 \partial x_2} f & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} f \\ \frac{\partial^2}{\partial x_2 \partial x_1} f & \frac{\partial^2}{\partial x_2^2} f & \cdots & \frac{\partial^2}{\partial x_2 \partial x_n} f \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} f & \frac{\partial^2}{\partial x_n \partial x_2} f & \cdots & \frac{\partial^2}{\partial x_n^2} f \end{pmatrix}. \quad (\text{B.55})$$

Since the order of differentiation does not matter ($H_{ij} = H_{ji}$), the matrix \mathbf{H}_f is symmetric. Note that the Hessian of a function f is a matrix of functions. To evaluate the Hessian at a particular point $\dot{\mathbf{x}} \in \mathbb{R}^n$, we write

$$\mathbf{H}_f(\dot{\mathbf{x}}) = \begin{pmatrix} \frac{\partial^2}{\partial x_1^2} f(\dot{\mathbf{x}}) & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} f(\dot{\mathbf{x}}) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} f(\dot{\mathbf{x}}) & \cdots & \frac{\partial^2}{\partial x_n^2} f(\dot{\mathbf{x}}) \end{pmatrix}, \quad (\text{B.56})$$

which is a scalar-valued matrix of size $n \times n$. As mentioned above, the *trace* of the Hessian matrix is the *Laplacian* ∇^2 of the function f ,

$$\nabla^2 f = \text{trace}(\mathbf{H}_f) = \sum_{i=1}^n \frac{\partial^2}{\partial x_i^2} f. \quad (\text{B.57})$$

Example

Given a two-dimensional, continuous, grayscale image or scalar-valued intensity function $I(x, y)$, the corresponding Hessian matrix is

$$\mathbf{H}_I = \begin{pmatrix} \frac{\partial^2}{\partial x^2} I & \frac{\partial^2}{\partial x \partial y} I \\ \frac{\partial^2}{\partial y \partial x} I & \frac{\partial^2}{\partial y^2} I \end{pmatrix} = \begin{pmatrix} I_{xx} & I_{xy} \\ I_{yx} & I_{yy} \end{pmatrix}. \quad (\text{B.58})$$

Note that the Hessian of the function $I(x, y)$ is again a (matrix-valued) function. To be explicit, at a given spatial position $\dot{\mathbf{x}} = (\dot{x}, \dot{y})$, the Hessian matrix $\mathbf{H}_I(\dot{\mathbf{x}})$ is composed of the values of the function's partial derivatives at position $\dot{\mathbf{x}}$, that is,

$$\mathbf{H}_I(\dot{\mathbf{x}}) = \begin{pmatrix} \frac{\partial^2}{\partial x^2} I(\dot{\mathbf{x}}) & \frac{\partial^2}{\partial x \partial y} I(\dot{\mathbf{x}}) \\ \frac{\partial^2}{\partial y \partial x} I(\dot{\mathbf{x}}) & \frac{\partial^2}{\partial y^2} I(\dot{\mathbf{x}}) \end{pmatrix} = \begin{pmatrix} I_{xx}(\dot{\mathbf{x}}) & I_{xy}(\dot{\mathbf{x}}) \\ I_{yx}(\dot{\mathbf{x}}) & I_{yy}(\dot{\mathbf{x}}) \end{pmatrix}. \quad (\text{B.59})$$

B.7 Operations on multi-variable, scalar functions (scalar fields)

B.7.1 Estimating the derivatives of a discrete function

Images are typically discrete functions (i.e., $I: \mathbb{N}^2 \rightarrow \mathbb{R}$) and thus not differentiable. The derivatives can nevertheless be estimated by calculating finite differences from the pixel values in a 3×3 neighborhood, which can be expressed as a linear filter or convolution operation. In particular, first- and second-order derivatives (as required in Eqn. (B.59), e.g.) are usually estimated as

$$\begin{aligned} \frac{\partial I}{\partial x} &\approx I_x = I * \begin{bmatrix} -0.5 & \mathbf{0} & 0.5 \end{bmatrix}, & \frac{\partial^2 I}{\partial x^2} &\approx I_{xx} = I * \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}, \\ \frac{\partial I}{\partial y} &\approx I_y = I * \begin{bmatrix} -0.5 \\ \mathbf{0} \\ 0.5 \end{bmatrix}, & \frac{\partial^2 I}{\partial y^2} &\approx I_{yy} = I * \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}, \\ \frac{\partial^2 I}{\partial x \partial y} &\approx I_{xy} = I_{yx} = I * \begin{bmatrix} -0.5 \\ \mathbf{0} \\ 0.5 \end{bmatrix} * \begin{bmatrix} -0.5 \\ \mathbf{0} \\ 0.5 \end{bmatrix} = I * \begin{bmatrix} 0.25 & 0 & -0.25 \\ 0 & \mathbf{0} & 0 \\ -0.25 & 0 & 0.25 \end{bmatrix}. \end{aligned} \quad (\text{B.60})$$

B.7.2 Taylor series expansion of functions

Single-variable functions

Taylor series expansion (of degree d) of a single-variable function $f: \mathbb{R} \rightarrow \mathbb{R}$ about a point a is

$$\begin{aligned} f(\dot{x}) &= f(a) + f'(a) \cdot (\dot{x} - a) + f''(a) \cdot \frac{(\dot{x} - a)^2}{2} + \cdots + f^{(d)}(a) \cdot \frac{(\dot{x} - a)^d}{d!} + R_d \\ &= f(a) + \sum_{i=1}^d f^{(i)}(a) \cdot \frac{(\dot{x} - a)^i}{i!} + R_d \end{aligned} \quad (\text{B.61})$$

$$= \sum_{i=0}^d f^{(i)}(a) \cdot \frac{(\dot{x} - a)^i}{i!} + R_d, \quad (\text{B.62})$$

where R_d is the remainder term.¹⁰ This means that if the value $f(a)$ and the first d derivatives $f'(a), f''(a), \dots, f^{(d)}(a)$ exist and are known at some position a , the value of f at *another* point \dot{x} can be estimated (up to the remainder R_d) only from the values at point a , without actually evaluating $f(x)$. Omitting the remainder R_d , the result is an *approximation* for $f(\dot{x})$, that is,

$$f(\dot{x}) \approx \sum_{i=0}^d f^{(i)}(a) \cdot \frac{(\dot{x} - a)^i}{i!}, \quad (\text{B.63})$$

whose accuracy depends upon d .

Multi-variable functions

In general, for a real-valued function of n variables,

$$f: \mathbb{R}^n \rightarrow \mathbb{R}, \quad f(\mathbf{x}) = f(x_1, x_2, \dots, x_n),$$

the full Taylor series expansion about a point $\mathbf{a} = (a_1, \dots, a_n)^\top$ is

$$f(\dot{x}_1, \dots, \dot{x}_n) = \sum_{i_1=0}^{\infty} \cdots \sum_{i_n=0}^{\infty} \left[\frac{\partial^{i_1}}{\partial x_1^{i_1}} \cdots \frac{\partial^{i_n}}{\partial x_n^{i_n}} \right] f(\mathbf{a}) \cdot \frac{(\dot{x}_1 - a_1)^{i_1} \cdots (\dot{x}_n - a_n)^{i_n}}{i_1! \cdots i_n!} \quad (\text{B.64})$$

$$= f(\mathbf{a}) + \sum_{i_1=1}^{\infty} \cdots \sum_{i_n=1}^{\infty} \left[\frac{\partial^{i_1}}{\partial x_1^{i_1}} \cdots \frac{\partial^{i_n}}{\partial x_n^{i_n}} \right] f(\mathbf{a}) \cdot \frac{(\dot{x}_1 - a_1)^{i_1} \cdots (\dot{x}_n - a_n)^{i_n}}{i_1! \cdots i_n!}. \quad (\text{B.65})$$

In the above expression,¹¹ the term

$$\left[\frac{\partial^{i_1}}{\partial x_1^{i_1}} \cdots \frac{\partial^{i_n}}{\partial x_n^{i_n}} \right] f(\mathbf{a}) \quad (\text{B.66})$$

means the result of the function f , after applying a sequence of n partial derivatives (one for each dimension $1, \dots, n$), being evaluated at the n -dimensional point \mathbf{a} . To get Eqn. (B.64) into a more compact form, we define the index vector

$$\mathbf{i} = (i_1, i_2, \dots, i_n), \quad \text{with } i_l \in \mathbb{N}_0 \quad (\text{B.67})$$

(and thus $\mathbf{i} \in \mathbb{N}_0^n$), and the associated operations

$$\begin{aligned} \mathbf{i}! &= i_1! \cdot i_2! \cdots i_n!, \\ \mathbf{x}^{\mathbf{i}} &= x_1^{i_1} \cdot x_2^{i_2} \cdots x_n^{i_n}, \\ \Sigma \mathbf{i} &= i_1 + i_2 + \cdots + i_n. \end{aligned} \quad (\text{B.68})$$

¹⁰ Note that $f^{(0)} = f$, $f^{(1)} = f'$, $f^{(2)} = f''$ etc., and $1! = 1$.

¹¹ Note that symbols x_1, \dots, x_n denote the individual variables, while $\dot{x}_1, \dots, \dot{x}_n$ are the coordinates of a particular point in n -dimensional space.

As a shorthand notation for the mixed partial derivative operator in Eqn. (B.66), we define

$$\mathbf{D}^{\mathbf{i}} = \mathbf{D}^{(i_1, \dots, i_n)} = \frac{\partial^{i_1}}{\partial x_1^{i_1}} \frac{\partial^{i_2}}{\partial x_2^{i_2}} \cdots \frac{\partial^{i_n}}{\partial x_n^{i_n}} = \frac{\partial^{i_1+i_2+\dots+i_n}}{\partial x_1^{i_1} \partial x_2^{i_2} \cdots \partial x_n^{i_n}}, \quad (\text{B.69})$$

in which each element $\frac{\partial^j}{\partial x_k^j}$ denotes the j -th order partial derivative operator along the dimension x_k .

With the above definitions, the full Taylor expansion of a multi-variable function about a point \mathbf{a} , as given in Eqn. (B.64), can be elegantly written in the form

$$f(\dot{\mathbf{x}}) = \sum_{\mathbf{i} \in \mathbb{N}_0^n} \mathbf{D}^{\mathbf{i}} f(\mathbf{a}) \cdot \frac{(\dot{\mathbf{x}} - \mathbf{a})^{\mathbf{i}}}{\mathbf{i}!}. \quad (\text{B.70})$$

Note that $\mathbf{D}^{\mathbf{i}} f$ is again a n -dimensional function $\mathbb{R}^n \rightarrow \mathbb{R}$, and thus $[\mathbf{D}^{\mathbf{i}} f](\mathbf{a})$ in Eqn. (B.70) is the scalar quantity obtained by evaluating the function $[\mathbf{D}^{\mathbf{i}} f]$ at the n -dimensional point \mathbf{a} . To obtain a Taylor *approximation* of order d , the sum of the indices i_1, \dots, i_n is limited to d , i.e., the summation is constrained to index vectors \mathbf{i} with $\Sigma \mathbf{i} \leq d$. Note that the resulting formulation,

$$f(\dot{\mathbf{x}}) \approx \sum_{\substack{\mathbf{i} \in \mathbb{N}_0^n \\ \Sigma \mathbf{i} \leq d}} \mathbf{D}^{\mathbf{i}} f(\mathbf{a}) \cdot \frac{(\dot{\mathbf{x}} - \mathbf{a})^{\mathbf{i}}}{\mathbf{i}!}, \quad (\text{B.71})$$

is analogous to the one-dimensional case in Eqn. (B.63).

Example: two-variable (2D) function $f: \mathbb{R}^2 \rightarrow \mathbb{R}$

This example demonstrates the second-order ($d = 2$) Taylor expansion of a two-dimensional ($n = 2$) function $f: \mathbb{R}^2 \rightarrow \mathbb{R}$ around a point $\mathbf{a} = (x_a, y_a)$. Inserting into Eqn. (B.70), we get

$$f(\dot{\mathbf{x}}, \dot{y}) \approx \sum_{\substack{\mathbf{i} \in \mathbb{N}_0^2 \\ \Sigma \mathbf{i} \leq 2}} \mathbf{D}^{\mathbf{i}} f(x_a, y_a) \cdot \frac{1}{\mathbf{i}!} \cdot \left(\begin{matrix} \dot{x} - x_a \\ \dot{y} - y_a \end{matrix} \right)^{\mathbf{i}} \quad (\text{B.72})$$

$$= \sum_{\substack{0 \leq i, j \leq 2 \\ (i+j) \leq 2}} \frac{\partial^{i+j}}{\partial x^i \partial y^j} f(x_a, y_a) \cdot \frac{(\dot{x} - x_a)^i \cdot (\dot{y} - y_a)^j}{i! \cdot j!}. \quad (\text{B.73})$$

Since $d = 2$, the 6 permissible index vectors $\mathbf{i} = (i, j)$, with $\Sigma \mathbf{i} \leq 2$, are $(0, 0)$, $(1, 0)$, $(0, 1)$, $(1, 1)$, $(2, 0)$, and $(0, 2)$. Inserting into Eqn. (B.73), we get the corresponding Taylor approximation as

$$f(\dot{\mathbf{x}}, \dot{y}) \approx \frac{\partial^0}{\partial x^0 \partial y^0} f(x_a, y_a) \cdot \frac{(\dot{x} - x_a)^0 \cdot (\dot{y} - y_a)^0}{1 \cdot 1} \quad (\text{B.74})$$

$$\begin{aligned}
& + \frac{\partial^1}{\partial x^1 \partial y^0} f(x_a, y_a) \cdot \frac{(\dot{x}-x_a)^1 \cdot (\dot{y}-y_a)^0}{1 \cdot 1} \\
& + \frac{\partial^1}{\partial x^0 \partial y^1} f(x_a, y_a) \cdot \frac{(\dot{x}-x_a)^0 \cdot (\dot{y}-y_a)^1}{1 \cdot 1} \\
& + \frac{\partial^2}{\partial x^1 \partial y^1} f(x_a, y_a) \cdot \frac{(\dot{x}-x_a)^1 \cdot (\dot{y}-y_a)^1}{1 \cdot 1} \\
& + \frac{\partial^2}{\partial x^2 \partial y^0} f(x_a, y_a) \cdot \frac{(\dot{x}-x_a)^2 \cdot (\dot{y}-y_a)^0}{2 \cdot 1} \\
& + \frac{\partial^2}{\partial x^0 \partial y^2} f(x_a, y_a) \cdot \frac{(\dot{x}-x_a)^0 \cdot (\dot{y}-y_a)^2}{1 \cdot 2} \\
& = f(x_a, y_a) \\
& + \frac{\partial}{\partial x} f(x_a, y_a) \cdot (\dot{x}-x_a) + \frac{\partial}{\partial y} f(x_a, y_a) \cdot (\dot{y}-y_a) \\
& + \frac{\partial^2}{\partial x \partial y} f(x_a, y_a) \cdot (\dot{x}-x_a) \cdot (\dot{y}-y_a) \\
& + \frac{1}{2} \cdot \frac{\partial^2}{\partial x^2} f(x_a, y_a) \cdot (\dot{x}-x_a)^2 + \frac{1}{2} \cdot \frac{\partial^2}{\partial y^2} f(x_a, y_a) \cdot (\dot{y}-y_a)^2.
\end{aligned} \tag{B.75}$$

It is assumed that the required derivatives of f exist, i. e., that at point (x_a, y_a) f is differentiable with respect to x and y up to the second order. By slightly rearranging Eqn. (B.75) to

$$\begin{aligned}
f(\dot{x}, \dot{y}) & \approx f(x_a, y_a) + \frac{\partial}{\partial x} f(x_a, y_a) \cdot (\dot{x}-x_a) + \frac{\partial}{\partial y} f(x_a, y_a) \cdot (\dot{y}-y_a) \\
& + \frac{1}{2} \cdot \left[\frac{\partial^2 f}{\partial x^2}(x_a, y_a) \cdot (\dot{x}-x_a)^2 + 2 \cdot \frac{\partial^2 f}{\partial x \partial y}(x_a, y_a) \cdot (\dot{x}-x_a) \cdot (\dot{y}-y_a) \right. \\
& \quad \left. + \frac{\partial^2 f}{\partial y^2}(x_a, y_a) \cdot (\dot{y}-y_a)^2 \right],
\end{aligned} \tag{B.76}$$

we can now write the Taylor expansion in matrix-vector notation as

$$\begin{aligned}
f(\dot{x}, \dot{y}) & \approx \tilde{f}(\dot{x}, \dot{y}) = f(x_a, y_a) + \left(\frac{\partial}{\partial x} f(x_a, y_a) \mid \frac{\partial}{\partial y} f(x_a, y_a) \right) \cdot \begin{pmatrix} \dot{x}-x_a \\ \dot{y}-y_a \end{pmatrix} \\
& + \frac{1}{2} \cdot \left[(\dot{x}-x_a \mid \dot{y}-y_a) \cdot \begin{pmatrix} \frac{\partial^2 f}{\partial x^2}(x_a, y_a) & \frac{\partial^2 f}{\partial x \partial y}(x_a, y_a) \\ \frac{\partial^2 f}{\partial x \partial y}(x_a, y_a) & \frac{\partial^2 f}{\partial y^2}(x_a, y_a) \end{pmatrix} \cdot \begin{pmatrix} \dot{x}-x_a \\ \dot{y}-y_a \end{pmatrix} \right],
\end{aligned} \tag{B.77}$$

or, even more compactly,

$$\tilde{f}(\dot{x}) = f(\mathbf{a}) + \nabla_f^\top(\mathbf{a}) \cdot (\dot{x}-\mathbf{a}) + \frac{1}{2} \cdot (\dot{x}-\mathbf{a})^\top \cdot \mathbf{H}_f(\mathbf{a}) \cdot (\dot{x}-\mathbf{a}). \tag{B.78}$$

Here $\nabla_f^\top(\mathbf{a})$ denotes the (transposed) *gradient* vector of the function f at point \mathbf{a} (see Sec. B.6.2), and \mathbf{H}_f is the 2×2 *Hessian* matrix of f (see Sec. B.6.6),

$$\mathbf{H}_f(\mathbf{a}) = \begin{pmatrix} \frac{\partial^2}{\partial x^2} f(\mathbf{a}) & \frac{\partial^2}{\partial x \partial y} f(\mathbf{a}) \\ \frac{\partial^2}{\partial x \partial y} f(\mathbf{a}) & \frac{\partial^2}{\partial y^2} f(\mathbf{a}) \end{pmatrix}. \quad (\text{B.79})$$

If the function f is *discrete*, e.g., a scalar-valued image I , the required partial derivatives at some lattice point $\mathbf{a} = (u_a, v_a)^\top$ can be estimated from its 3×3 neighborhood, as described in Section B.7.1.

Example: three-variable (3D) function $f: \mathbb{R}^3 \rightarrow \mathbb{R}$

For a second-order Taylor expansion ($d = 2$), the situation is exactly the same as in Eqns. (B.77–B.78) for the 2D case, except that now $\dot{\mathbf{x}} = (\dot{x}, \dot{y}, \dot{z})^\top$ and $\mathbf{a} = (x_a, y_a, z_a)^\top$ are three-dimensional vectors, the corresponding gradient vector is

$$\nabla_f^\top(\mathbf{a}) = \left(\frac{\partial}{\partial x} f(\mathbf{a}) \mid \frac{\partial}{\partial y} f(\mathbf{a}) \mid \frac{\partial}{\partial z} f(\mathbf{a}) \right), \quad (\text{B.80})$$

and the Hessian is a 3×3 matrix composed of all second-order partial derivatives, that is,

$$\mathbf{H}_f(\mathbf{a}) = \begin{pmatrix} \frac{\partial^2}{\partial x^2} f(\mathbf{a}) & \frac{\partial^2}{\partial x \partial y} f(\mathbf{a}) & \frac{\partial^2}{\partial x \partial z} f(\mathbf{a}) \\ \frac{\partial^2}{\partial y \partial x} f(\mathbf{a}) & \frac{\partial^2}{\partial y^2} f(\mathbf{a}) & \frac{\partial^2}{\partial y \partial z} f(\mathbf{a}) \\ \frac{\partial^2}{\partial z \partial x} f(\mathbf{a}) & \frac{\partial^2}{\partial z \partial y} f(\mathbf{a}) & \frac{\partial^2}{\partial z^2} f(\mathbf{a}) \end{pmatrix}. \quad (\text{B.81})$$

Note that the order of differentiation is not relevant, e.g., $\frac{\partial^2}{\partial x \partial y} = \frac{\partial^2}{\partial y \partial x}$, and therefore \mathbf{H}_f is symmetric. This can be easily generalized to the n -dimensional case, though things get considerably more involved for Taylor expansions of higher orders ($d > 2$).

B.7.3 Finding the continuous extremum of a multi-variable discrete function

The aim in this section is to find the position of the minimum or maximum of the fitted (quadratic) approximation function. For the sake of simplicity, let us assume that the function $f(\mathbf{x})$ is expanded around $\mathbf{a} = \mathbf{0} = (0, 0)$. The Taylor approximation function (see Eqn. (B.78)) for this point can be written as

$$\tilde{f}(\mathbf{x}) = f(\mathbf{0}) + \nabla_f^\top(\mathbf{0}) \cdot \mathbf{x} + \frac{1}{2} \cdot \mathbf{x}^\top \cdot \mathbf{H}_f(\mathbf{0}) \cdot \mathbf{x} \quad (\text{B.82})$$

or, by omitting the $\mathbf{0}$ coordinate (i.e., substituting $f \leftarrow f(\mathbf{0})$, $\nabla_f \leftarrow \nabla_f(\mathbf{0})$, and $\mathbf{H}_f \leftarrow \mathbf{H}_f(\mathbf{0})$),

$$\tilde{f}(\mathbf{x}) = f + \nabla_f^\top \cdot \mathbf{x} + \frac{1}{2} \cdot \mathbf{x}^\top \cdot \mathbf{H}_f \cdot \mathbf{x}. \quad (\text{B.83})$$

The first derivative of this function with respect to \mathbf{x} is

$$\tilde{f}'(\mathbf{x}) = \nabla_f + \frac{1}{2} \cdot [(\mathbf{x}^\top \cdot \mathbf{H}_f)^\top + \mathbf{H}_f \cdot \mathbf{x}]. \quad (\text{B.84})$$

Since $(\mathbf{x}^\top \cdot \mathbf{H}_f)^\top = (\mathbf{H}_f^\top \cdot \mathbf{x})$ and because the Hessian matrix \mathbf{H}_f is symmetric (i.e., $\mathbf{H}_f = \mathbf{H}_f^\top$), this simplifies to

$$\tilde{f}'(\mathbf{x}) = \nabla_f + \frac{1}{2} \cdot (\mathbf{H}_f \cdot \mathbf{x} + \mathbf{H}_f \cdot \mathbf{x}) = \nabla_f + \mathbf{H}_f \cdot \mathbf{x}. \quad (\text{B.85})$$

A local maximum or minimum is found where the first derivative $\tilde{f}'(\mathbf{x})$ is zero, so we need to solve

$$\nabla_f + \mathbf{H}_f \cdot \check{\mathbf{x}} = \mathbf{0}, \quad (\text{B.86})$$

for the unknown position $\check{\mathbf{x}}$. By multiplying both sides with \mathbf{H}_f^{-1} (assuming \mathbf{H}_f is non-singular), the solution is

$$\check{\mathbf{x}} = -\mathbf{H}_f^{-1} \cdot \nabla_f, \quad (\text{B.87})$$

for the specific expansion point $\mathbf{a} = \mathbf{0}$. For an arbitrary expansion point \mathbf{a} , the extremum position is

$$\check{\mathbf{x}} = \mathbf{a} - \mathbf{H}_f^{-1}(\mathbf{a}) \cdot \nabla_f(\mathbf{a}). \quad (\text{B.88})$$

Note that the inverse Hessian matrix \mathbf{H}_f^{-1} is again symmetric.

The extremal *value* of the approximation function \tilde{f} is found by replacing \mathbf{x} in Eqn. (B.83) with the extremal position $\check{\mathbf{x}}$ (defined in Eqn. (B.87)) as

$$\begin{aligned} \tilde{f}_{\text{peak}} &= \tilde{f}(\check{\mathbf{x}}) = f + \nabla_f^\top \cdot \check{\mathbf{x}} + \frac{1}{2} \cdot \check{\mathbf{x}}^\top \cdot \mathbf{H}_f \cdot \check{\mathbf{x}} \\ &= f + \nabla_f^\top \cdot \check{\mathbf{x}} + \frac{1}{2} \cdot \check{\mathbf{x}}^\top \cdot \mathbf{H}_f \cdot (-\mathbf{H}_f^{-1}) \cdot \nabla_f \\ &= f + \nabla_f^\top \cdot \check{\mathbf{x}} - \frac{1}{2} \cdot \check{\mathbf{x}}^\top \cdot \mathbf{I} \cdot \nabla_f \\ &= f + \nabla_f^\top \cdot \check{\mathbf{x}} - \frac{1}{2} \cdot \nabla_f^\top \cdot \check{\mathbf{x}} \\ &= f + \frac{1}{2} \cdot \nabla_f^\top \cdot \check{\mathbf{x}}, \end{aligned} \quad (\text{B.89})$$

again for the expansion point $\mathbf{a} = \mathbf{0}$. For an arbitrary expansion point \mathbf{a} , the above expression changes to

$$\tilde{f}_{\text{peak}} = \tilde{f}(\check{\mathbf{x}}) = f(\mathbf{a}) + \frac{1}{2} \cdot \nabla_f^\top(\mathbf{a}) \cdot (\check{\mathbf{x}} - \mathbf{a}). \quad (\text{B.90})$$

Note that \tilde{f}_{peak} may be a local minimum or maximum, but could also be a saddle point where the first derivatives of the function are zero as well.

Local extrema in 2D

The above scheme works for n -dimensional functions f . In the specific case of a two-dimensional function $f: \mathbb{R}^2 \rightarrow \mathbb{R}$ (e.g., a 2D image), the gradient vector and the Hessian matrix can be written as

$$\nabla_f(\mathbf{a}) = \begin{pmatrix} d_x \\ d_y \end{pmatrix} \quad \text{and} \quad \mathbf{H}_f(\mathbf{a}) = \begin{pmatrix} h_{11} & h_{12} \\ h_{12} & h_{22} \end{pmatrix},$$

for a given expansion point $\mathbf{a} = (x_a, y_a)^\top$. In this case, the inverse of the Hessian matrix is

$$\mathbf{H}_f^{-1} = \frac{1}{h_{12}^2 - h_{11} \cdot h_{22}} \cdot \begin{pmatrix} -h_{22} & h_{12} \\ h_{12} & -h_{11} \end{pmatrix}, \quad (\text{B.91})$$

and the *position* of the extremal point is (see Eqn. (B.88))

$$\begin{aligned} \check{\mathbf{x}} &= \begin{pmatrix} \check{x} \\ \check{y} \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} - \frac{1}{h_{12}^2 - h_{11} \cdot h_{22}} \cdot \begin{pmatrix} -h_{22} & h_{12} \\ h_{12} & -h_{11} \end{pmatrix} \cdot \begin{pmatrix} d_x \\ d_y \end{pmatrix} \\ &= \begin{pmatrix} a \\ b \end{pmatrix} - \frac{1}{h_{12}^2 - h_{11} \cdot h_{22}} \cdot \begin{pmatrix} h_{12} \cdot d_y - h_{22} \cdot d_x \\ h_{12} \cdot d_x - h_{11} \cdot d_y \end{pmatrix}. \end{aligned} \quad (\text{B.92})$$

The extremal position is only defined if the above denominator $h_{12}^2 - h_{11} \cdot h_{22}$ (the determinant of \mathbf{H}_f) is non-zero, indicating that the Hessian matrix \mathbf{H}_f is non-singular and thus has an inverse. In this case, the *value* of \tilde{f} at the extremal position $\check{\mathbf{x}} = (\check{x}, \check{y})^\top$ (see Eqn. (B.90)) can be calculated as

$$\begin{aligned} \tilde{f}(\check{x}, \check{y}) &= f(x_a, y_a) + \frac{1}{2} \cdot (d_x, d_y) \cdot \begin{pmatrix} \check{x} - x_a \\ \check{y} - y_a \end{pmatrix} \\ &= f(x_a, y_a) + \frac{d_x \cdot (\check{x} - x_a) + d_y \cdot (\check{y} - y_a)}{2}. \end{aligned} \quad (\text{B.93})$$

A concrete example for estimating the local extremal position in a discrete 2D image can be found in “Numeric 2D example” below.

Local extrema in 3D

In the case of a three-variable function $f: \mathbb{R}^3 \rightarrow \mathbb{R}$, with a given expansion point $\mathbf{a} = (x_a, y_a, z_a)^\top$ and

$$\nabla_f(\mathbf{a}) = \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}, \quad \mathbf{H}_f(\mathbf{a}) = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{12} & h_{22} & h_{23} \\ h_{13} & h_{23} & h_{33} \end{pmatrix}$$

being the gradient vector and the Hessian matrix of f at point \mathbf{a} , respectively, the estimated extremal position is

$$\begin{aligned} \check{\mathbf{x}} = \begin{pmatrix} \check{x} \\ \check{y} \\ \check{z} \end{pmatrix} &= \begin{pmatrix} x_a \\ y_a \\ z_a \end{pmatrix} - \frac{1}{h_{13}^2 \cdot h_{22} + h_{12}^2 \cdot h_{33} + h_{11} \cdot h_{23}^2 - h_{11} \cdot h_{22} \cdot h_{33} - 2 \cdot h_{12} \cdot h_{13} \cdot h_{23}} \\ &\times \begin{pmatrix} h_{23}^2 - h_{22} \cdot h_{33} & h_{12} \cdot h_{33} - h_{13} \cdot h_{23} & h_{13} \cdot h_{22} - h_{12} \cdot h_{23} \\ h_{12} \cdot h_{33} - h_{13} \cdot h_{23} & h_{13}^2 - h_{11} \cdot h_{33} & h_{11} \cdot h_{23} - h_{12} \cdot h_{13} \\ h_{13} \cdot h_{22} - h_{12} \cdot h_{23} & h_{11} \cdot h_{23} - h_{12} \cdot h_{13} & h_{12}^2 - h_{11} \cdot h_{22} \end{pmatrix} \cdot \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix}. \end{aligned} \quad (\text{B.94})$$

Note that the inverse of the Hessian matrix \mathbf{H}_f^{-1} , expanded in Eqn. (B.94), is again symmetric and can be calculated in closed form. According to Eqn. (B.90), the estimated extremal value at $\check{\mathbf{x}}$ is

$$\begin{aligned} \tilde{f}(\check{x}, \check{y}, \check{z}) &= f(\mathbf{a}) + \frac{1}{2} \cdot \nabla_f^\top(\mathbf{a}) \cdot (\check{\mathbf{x}} - \mathbf{a}) \\ &= f(x_a, y_a, z_a) + \frac{d_x \cdot (\check{x} - x_a) + d_y \cdot (\check{y} - y_a) + d_z \cdot (\check{z} - z_a)}{2}. \end{aligned} \quad (\text{B.95})$$

Numeric 2D example

The following example shows how a local extremum can be found in a discrete 2D image with sub-pixel accuracy using a second-order Taylor approximation. Assume we are given a grayscale image I with the sample values

$$\begin{matrix} & u_a-1 & u_a & u_a+1 \\ v_a-1 & 8 & 11 & 7 \\ v_a & 15 & 16 & 9 \\ v_a+1 & 14 & 12 & 10 \end{matrix} \quad (\text{B.96})$$

in the 3×3 neighborhood of coordinate $\mathbf{a} = (u_a, v_a)^\top$. Obviously, the discrete center value $f(\mathbf{a}) = 16$ is a local maximum but (as we shall see) the maximum of the *continuous* approximation function is *not* at the center. The gradient

vector ∇_I and the Hessian Matrix \mathbf{H}_I at the expansion point \mathbf{a} are calculated from local finite differences (see Sec. B.7.1) as

$$\begin{aligned}\nabla_I(\mathbf{a}) &= \begin{pmatrix} d_x \\ d_y \end{pmatrix} = 0.5 \cdot \begin{pmatrix} 9-15 \\ 12-11 \end{pmatrix} = \begin{pmatrix} -3 \\ 0.5 \end{pmatrix} \quad \text{and} \\ \mathbf{H}_I(\mathbf{a}) &= \begin{pmatrix} h_{11} & h_{12} \\ h_{12} & h_{22} \end{pmatrix} = \begin{pmatrix} 9-2 \cdot 16+15 & 0.25 \cdot (8-14-7+10) \\ 0.25 \cdot (8-14-7+10) & 11-2 \cdot 16+12 \end{pmatrix} \\ &= \begin{pmatrix} -8.00 & -0.75 \\ -0.75 & -9.00 \end{pmatrix},\end{aligned}$$

respectively. The resulting second-order Taylor expansion about the point \mathbf{a} is the continuous function (see Eqn. (B.78))

$$\begin{aligned}\tilde{f}(\mathbf{x}) &= f(\mathbf{a}) + \nabla_I^\top(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a}) + \frac{1}{2} \cdot (\mathbf{x} - \mathbf{a})^\top \cdot \mathbf{H}_I(\mathbf{a}) \cdot (\mathbf{x} - \mathbf{a}) \\ &= 16 + (-3 \mid 0.5) \cdot \begin{pmatrix} x-u_a \\ y-v_a \end{pmatrix} + \frac{1}{2} \cdot (x-u_a \mid y-v_a) \cdot \begin{pmatrix} -8.00 & -0.75 \\ -0.75 & -9.00 \end{pmatrix} \cdot \begin{pmatrix} x-u_a \\ y-v_a \end{pmatrix}.\end{aligned}$$

We use the inverse of the Hessian matrix,

$$\mathbf{H}_I^{-1}(\mathbf{a}) = \begin{pmatrix} -8.00 & -0.75 \\ -0.75 & -9.00 \end{pmatrix}^{-1} = \begin{pmatrix} -0.125984 & 0.010499 \\ 0.010499 & -0.111986 \end{pmatrix},$$

to calculate the *position* of the local extremum $\check{\mathbf{x}}$ (see Eqn. (B.92)) as

$$\begin{aligned}\check{\mathbf{x}} &= \mathbf{a} - \mathbf{H}_I^{-1}(\mathbf{a}) \cdot \nabla_I(\mathbf{a}) \\ &= \begin{pmatrix} u_a \\ v_a \end{pmatrix} - \begin{pmatrix} -0.125984 & 0.010499 \\ 0.010499 & -0.111986 \end{pmatrix} \cdot \begin{pmatrix} -3 \\ 0.5 \end{pmatrix} = \begin{pmatrix} u_a - 0.3832 \\ v_a + 0.0875 \end{pmatrix}.\end{aligned}$$

Finally, the estimated extremal *value* (see Eqn. (B.90)) is

$$\begin{aligned}\tilde{f}(\check{\mathbf{x}}) &= f(\mathbf{a}) + \frac{1}{2} \cdot \nabla_f^\top(\mathbf{a}) \cdot (\check{\mathbf{x}} - \mathbf{a}) \\ &= 16 + \frac{1}{2} \cdot (-3 \mid 0.5) \cdot \begin{pmatrix} u_a - 0.3832 - u_a \\ v_a + 0.0875 - v_a \end{pmatrix} \\ &= 16 + \frac{1}{2} \cdot (3 \cdot 0.3832 + 0.5 \cdot 0.0875) = 16.5967.\end{aligned}$$

[Figure \(B.3\)](#) illustrates the above example, with the expansion point set to $\mathbf{a} = (u_a, v_a)^\top = (0, 0)^\top$.

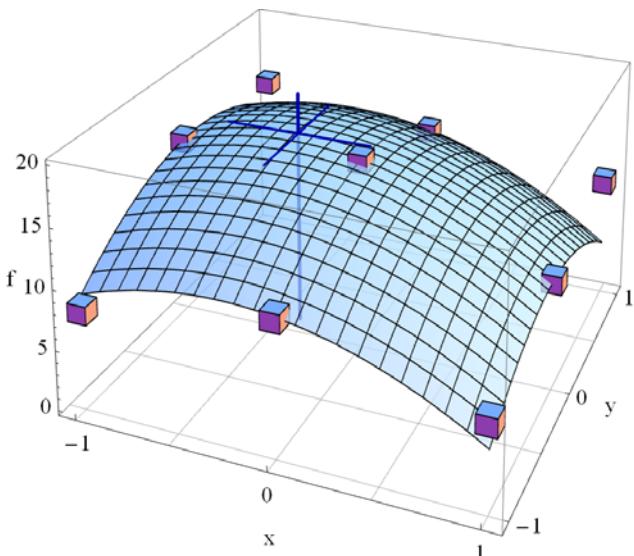


Figure B.3 Continuous Taylor approximation of a discrete image function for determining the local extremum position with sub-pixel accuracy. The cubes represent the discrete image samples in a 3×3 neighborhood around the reference coordinate $(0, 0)$, which is a local maximum of the discrete image function (see Eqn. (B.96) for the concrete values). The smooth surface shows the continuous approximation $\tilde{f}(x, y)$ obtained by second-order Taylor expansion about the center position $a = (0, 0)$. The vertical line marks the position of the local maximum $\tilde{f}(\tilde{x}) = 16.5967$ at $\tilde{x} = (-0.3832, 0.0875)$.

C

Statistical Prerequisites

C.1 Mean, variance and covariance

Given a sequence (sample) $X = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$ of n vector-valued, m -dimensional measurements, with

$$\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,m})^\top \in \mathbb{R}^m, \quad (\text{C.1})$$

the corresponding *sample mean vector* is defined as

$$\boldsymbol{\mu}(X) = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_m \end{pmatrix} = \frac{1}{n} \cdot (\mathbf{x}_1 + \mathbf{x}_2 + \dots + \mathbf{x}_n) = \frac{1}{n} \cdot \sum_{i=1}^n \mathbf{x}_i. \quad (\text{C.2})$$

The vector $\boldsymbol{\mu}(X)$ corresponds to the *centroid* of the sample vectors \mathbf{x}_i . Each scalar element μ_p is the mean of the component (also called *variante* or *dimension*) p from all n sample vectors, that is

$$\mu_p = \frac{1}{n} \cdot \sum_{i=1}^n x_{i,p}, \quad \text{for } p = 1, \dots, m. \quad (\text{C.3})$$

The *covariance* quantifies the strength of interaction between a pair of components p, q in the sample X , that is,

$$\sigma_{p,q}(X) = \frac{1}{n} \cdot \sum_{i=1}^n (x_{i,p} - \mu_p) \cdot (x_{i,q} - \mu_q). \quad (\text{C.4})$$

For efficient calculation, the above expression can be rewritten in the form

$$\sigma_{p,q}(X) = \frac{1}{n} \cdot \underbrace{\left[\sum_{i=1}^n (x_{i,p} \cdot x_{i,q}) \right]}_{S_{p,q}(X)} - \frac{1}{n} \cdot \underbrace{\left(\sum_{i=1}^n x_{i,p} \right)}_{S_p(X)} \cdot \underbrace{\left(\sum_{i=1}^n x_{i,q} \right)}_{S_q(X)}, \quad (\text{C.5})$$

which does not require the explicit calculation of μ_p and μ_q . In the special case of $p = q$, we get

$$\sigma_{p,p}(X) = \sigma_p^2(X) = \frac{1}{n} \cdot \sum_{i=1}^n (x_{i,p} - \mu_p)^2 \quad (\text{C.6})$$

$$= \frac{1}{n} \cdot \left[\sum_{i=1}^n x_{i,p}^2 - \frac{1}{n} \cdot \left(\sum_{i=1}^n x_{i,p} \right)^2 \right], \quad (\text{C.7})$$

which is the *variance within* the component p . This corresponds to the ordinary (one-dimensional) variance $\sigma_p^2(X)$ of the n scalar sample values $x_{1,p}, x_{2,p}, \dots, x_{n,p}$.

C.2 Covariance matrices

The *covariance matrix* Σ for the m -dimensional sample X is a square matrix of size $m \times m$ that is composed of the covariance values $\sigma_{p,q}$ for all pairs (p, q) of components, i. e.,

$$\Sigma(X) = \begin{pmatrix} \sigma_{1,1} & \sigma_{1,2} & \cdots & \sigma_{1,m} \\ \sigma_{2,1} & \sigma_{2,2} & \cdots & \sigma_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{m,1} & \sigma_{m,2} & \cdots & \sigma_{m,m} \end{pmatrix} = \begin{pmatrix} \sigma_1^2 & \sigma_{1,2} & \cdots & \sigma_{1,m} \\ \sigma_{2,1} & \sigma_2^2 & \cdots & \sigma_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{m,1} & \sigma_{m,2} & \cdots & \sigma_m^2 \end{pmatrix}. \quad (\text{C.8})$$

Note that an element on the diagonal of $\Sigma(X)$ is the ordinary (scalar) variance $\sigma_p^2(X)$ (Eqn. (C.6)), for $p = 1, \dots, m$, which can never be negative.

All other entries of a covariance matrix may be both positive or negative in general. However, since $\sigma_{p,q}(X) = \sigma_{q,p}(X)$, a covariance matrix is always symmetric, with up to $(m^2 + m)/2$ unique elements. Thus, any covariance matrix has the important property of being *positive semidefinite*, which implies that all of its eigenvalues (see Sec. B.4) are positive (non-negative). The covariance matrix can also be written in the form

$$\Sigma(X) = \frac{1}{n} \sum_{i=1}^n \underbrace{[\mathbf{x}_i - \boldsymbol{\mu}(X)] \cdot [\mathbf{x}_i - \boldsymbol{\mu}(X)]^\top}_{= [\mathbf{x}_i - \boldsymbol{\mu}(X)] \otimes [\mathbf{x}_i - \boldsymbol{\mu}(X)]}, \quad (\text{C.9})$$

where \otimes denotes the outer (vector) product (see also Sec. B.3.2).

The trace (sum of the diagonal elements) of the covariance matrix is called the *total variance*,

$$\sigma_{\text{total}}(X) = \text{trace}(\Sigma(X)). \quad (\text{C.10})$$

Alternatively, the (Frobenius) *norm* of the covariance matrix $\Sigma(X)$, defined as

$$\|\Sigma(X)\|_2 = \left(\sum_{i=1}^m \sum_{j=1}^m \sigma_{i,j}^2 \right)^{1/2}, \quad (\text{C.11})$$

can be used to quantify the overall variance in the sample data.

C.2.1 Example

Assume that the sample X consists of the following set of $n = 4$ three-dimensional ($m = 3$) vectors

$$\mathbf{x}_1 = \begin{pmatrix} 75 \\ 37 \\ 12 \end{pmatrix}, \quad \mathbf{x}_2 = \begin{pmatrix} 41 \\ 27 \\ 20 \end{pmatrix}, \quad \mathbf{x}_3 = \begin{pmatrix} 93 \\ 81 \\ 11 \end{pmatrix}, \quad \mathbf{x}_4 = \begin{pmatrix} 12 \\ 48 \\ 52 \end{pmatrix}.$$

with each $\mathbf{x}_i = (x_{i,R}, x_{i,G}, x_{i,B})^\top$ representing a particular RGB color tuple. The resulting *sample mean vector* (see Eqn. (C.2)) is

$$\boldsymbol{\mu}(X) = \begin{pmatrix} \mu_R \\ \mu_G \\ \mu_B \end{pmatrix} = \frac{1}{4} \cdot \begin{pmatrix} 75 + 41 + 93 + 12 \\ 37 + 27 + 81 + 48 \\ 12 + 20 + 11 + 52 \end{pmatrix} = \frac{1}{4} \cdot \begin{pmatrix} 221 \\ 193 \\ 95 \end{pmatrix} = \begin{pmatrix} 55.25 \\ 48.25 \\ 23.75 \end{pmatrix},$$

and the corresponding *covariance matrix* (Eqn. (C.8)) is

$$\Sigma(X) = \begin{pmatrix} 1296.250 & 442.583 & -627.250 \\ 442.583 & 550.250 & -70.917 \\ -627.250 & -70.917 & 370.917 \end{pmatrix}.$$

Note that this matrix is symmetric and all diagonal elements are non-negative. The *total variance* (Eqn. (C.10)) of the sample set is

$$\sigma_{\text{total}}(X) = \text{trace}(\Sigma(X)) = 1296.25 + 550.25 + 370.917 = 2217.42$$

and the *norm* of the covariance matrix (Eqn. (C.11)) is $\|\Sigma(X)\|_2 = 1764.54$.

C.3 The Gaussian distribution

The Gaussian distribution plays a major role in decision theory, pattern recognition, and statistics in general, because of its convenient analytical properties. A continuous, scalar quantity X is said to be subject to a Gaussian distribution, if the probability of observing a particular value x is

$$p(X=x) = p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \quad (\text{C.12})$$

The Gaussian distribution is completely defined by its mean μ and spread σ^2 . The fact that this is also called a “normal” distribution is commonly denoted in the form

$$p(x) \sim \mathcal{N}(X|\mu, \sigma^2) \quad \text{or simply} \quad X \sim \mathcal{N}(\mu, \sigma^2), \quad (\text{C.13})$$

saying that “ X is normally distributed with parameters μ and σ^2 .” As required for any valid probability distribution,

$$\mathcal{N}(X|\mu, \sigma^2) > 0 \quad \text{and} \quad \int_{-\infty}^{\infty} \mathcal{N}(X|\mu, \sigma^2) dx = 1. \quad (\text{C.14})$$

Thus the area under the probability distribution curve is always one, i. e., $\mathcal{N}()$ is normalized. The Gaussian function in Eqn. (C.12) has its maximum height (called “mode”) at position $x = \mu$, where its value is

$$p(x=\mu) = \frac{1}{\sqrt{2\pi\sigma^2}}. \quad (\text{C.15})$$

If a random variable X is normally distributed with mean μ and variance σ^2 , then the result of a linear mapping of the kind $X' = aX + b$ is again a random variable that is normally distributed, with parameters $\bar{\mu} = a \cdot \mu + b$ and $\bar{\sigma}^2 = a^2 \cdot \sigma^2$:

$$X \sim \mathcal{N}(\mu, \sigma^2) \Rightarrow a \cdot X + b \sim \mathcal{N}(a \cdot \mu + b, a^2 \cdot \sigma^2), \quad (\text{C.16})$$

for $a, b \in \mathbb{R}$. Moreover, if X_1, X_2 are statistically *independent*, normally distributed random variables with means μ_1, μ_2 and variances σ_1^2, σ_2^2 , respectively, then a linear combination of the form $a_1X_1 + a_2X_2$ is again normally distributed with $\mu_{12} = a_1 \cdot \mu_1 + a_2 \cdot \mu_2$ and $\sigma_{12}^2 = a_1^2 \cdot \sigma_1^2 + a_2^2 \cdot \sigma_2^2$, that is,

$$(a_1X_1 + a_2X_2) \sim \mathcal{N}(a_1 \cdot \mu_1 + a_2 \cdot \mu_2, a_1^2 \cdot \sigma_1^2 + a_2^2 \cdot \sigma_2^2). \quad (\text{C.17})$$

C.3.1 Maximum likelihood

The probability density function $p(x)$ of a statistical distribution tells us how probable it is to observe the result x for some fixed distribution parameters, such as μ and σ , in case of a normal distribution. If these parameters are *unknown* and need to be estimated,¹ it is interesting to ask the reverse question: How likely are some parameters for a given set of empirical observations? This is (in a casual sense) what the term “likelihood” stands for. In particular, a distribution’s *likelihood function* quantifies the probability that a given (fixed) set of observations was generated by some varying distribution parameters.

Note that the probability of observing the outcome x from the normal distribution,

$$p(x) = \mathcal{N}(x|\mu, \sigma^2),$$

is really a *conditional* probability, stating how probable it is to observe x from a given normal distribution with known parameters μ and σ^2 . Conversely, a likelihood function for the normal distribution could be viewed as a conditional function

$$L(\mu, \sigma^2|x),$$

whose value expresses the likelihood of (μ, σ^2) being the correct parameters for a given observation x . The maximum likelihood method tries to find optimal parameters by *maximizing* the value of a distribution’s likelihood function L .

If we draw two independent² samples x_1, x_2 that are subject to the same distribution, their *joint probability* (that is, the probability of x_1 and x_2 occurring together in the sample) is the product of their individual probabilities, that is,

$$p(x_1 \wedge x_2) = p(x_1) \cdot p(x_2). \quad (\text{C.18})$$

In general, if we are given a vector of m independent observations $X = (x_1, x_2, \dots, x_m)$ from the same distribution, the probability of observing exactly this set of values is

$$p(X) = p(x_1 \wedge x_2 \wedge \dots \wedge x_m) = p(x_1) \cdot p(x_2) \cdots p(x_m) = \prod_{i=1}^m p(x_i). \quad (\text{C.19})$$

Thus, a suitable likelihood function for the normal distribution is

$$L(\mu, \sigma^2|X) = p(X|\mu, \sigma^2) = \prod_{i=1}^m \mathcal{N}(x_i|\mu, \sigma^2) = \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}. \quad (\text{C.20})$$

¹ As required for “minimum error thresholding” in Sec. 2.1.6.

² Statistical independence is a key to keep the problem simple and tractable, although this assumption is often violated. In particular, the values of adjacent image pixels are usually not independent.

The parameters $(\hat{\mu}, \hat{\sigma}^2)$, for which $L(\mu, \sigma^2 | X)$ is a maximum, are called the maximum-likelihood estimate for X .

Note that it is not necessary for a likelihood function to be a proper (i.e., normalized) probability distribution, since it is only necessary to calculate whether a particular set of distribution parameters is more probable than another. Thus the likelihood function L may be any monotonic function of the corresponding probability p in Eqn. (C.20), in particular its *logarithm*, which is commonly used to avoid multiplying small values.

C.3.2 Gaussian mixtures

In practice, probabilistic models are often too complex to be described by a single Gaussian (or other standard) distribution. Without losing the mathematical convenience of Gaussian models, highly complex distributions can be modeled as combinations of multiple Gaussian distributions. In particular, a Gaussian *mixture model* is a linear superposition of K Gaussian distributions of the form

$$p(x) = \sum_{j=1}^K \pi_j \cdot \mathcal{N}(x | \mu_j, \sigma_j^2), \quad (\text{C.21})$$

where the weights (“mixing coefficients”) π_j express the probability that an event x was generated by the j^{th} component (with $\sum_{j=1}^K \pi_j = 1$).³ The interpretation of this mixture model is that there are K independent Gaussian “components” (each with its parameters μ_j, σ_j) that contribute to a common stream of events x_i . If a particular value x is observed, it is assumed to be the result of exactly *one* of the K components, but the identity of that component is unknown.

Assume, as a special case, that a probability distribution $p(x)$ is the superposition (mixture) of *two* Gaussian distributions, i.e.,

$$p(x) = \pi_0 \cdot \mathcal{N}(x | \mu_0, \sigma_0^2) + \pi_1 \cdot \mathcal{N}(x | \mu_1, \sigma_1^2). \quad (\text{C.22})$$

Any observed value x is assumed to be generated by either the first component (with prior probability π_0) or the second component (with prior probability π_1).

The parameters $\mu_0, \sigma_0^2, \mu_1, \sigma_1^2$ as well as the prior probabilities π_0, π_1 are unknown. Given a particular set of observations, the task is to estimate these six parameters, such that the Gaussian mixture approximates the original distribution as closely as possible. Note that, in general, the unknown parameters cannot be calculated in closed form but only with numerical methods. For further details and solution methods see [13, 37, 125], for example.

³ The weight π_j is also called the *prior* probability of the component j .

C.3.3 Creating Gaussian noise

Synthetic Gaussian noise is often used for testing in image processing, particularly for assessing the quality of smoothing filters. While the generation of pseudo-random values that follow a Gaussian distribution is not a trivial task in general,⁴ it is readily implemented in Java by the standard class `Random`. For example, the following Java method `addGaussianNoise()` adds Gaussian noise with zero mean ($\mu = 0$) and standard deviation s (σ) to a grayscale image I (of ImageJ type `FloatProcessor`):

```

1 import java.util.Random;
2
3 void addGaussianNoise (FloatProcessor I, double s) {
4     int w = I.getWidth();
5     int h = I.getHeight();
6     Random rnd = new Random();
7     for (int v=0; v<h; v++) {
8         for (int u=0; u<w; u++) {
9             float val = I.getf(u, v);
10            float noise = (float) (rnd.nextGaussian() * s);
11            I.setf(u, v, val + noise);
12        }
13    }
14 }
```

Note that (in line 10) the random values produced by successive calls to the method `nextGaussian()` follow a Gaussian distribution $\mathcal{N}(0, 1)$, with mean $\mu = 0$ and variance $\sigma^2 = 1$. As implied by Eqn. (C.16),

$$X \sim \mathcal{N}(0, 1) \Rightarrow s \cdot X \sim \mathcal{N}(0, s^2), \quad (\text{C.23})$$

and thus scaling the results from `nextGaussian()` by s makes the corresponding random variable `noise` normally distributed with $\mathcal{N}(0, s^2)$.

C.4 Image quality measures⁵

Objective error measures can be used to quantify the deviations between some corrupted data set and the original data, for example, in the context of image restoration, noise removal or compression. Common quality measures include the *mean absolute error*, the *mean square error*, the *root mean square error* and the *signal-to-noise ratio*, as listed below. In the following, \tilde{I} denotes an observed (noisy) image and I the original (noise-less) reference image, both of size $M \times N$.

⁴ Typically the so-called *polar method* is used for generating Gaussian random values [66, Sec. 3.4.1].

⁵ This summary complements Sec. 5.4, which contains additional details about the practical use of these measures.

Mean absolute error (MAE).

$$\text{MAE}(I, \tilde{I}) = \frac{1}{M \cdot N} \cdot \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} |I(u, v) - \tilde{I}(u, v)|. \quad (\text{C.24})$$

Mean square error (MSE).

$$\text{MSE}(I, \tilde{I}) = \frac{1}{M \cdot N} \cdot \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} [I(u, v) - \tilde{I}(u, v)]^2. \quad (\text{C.25})$$

Root mean square error (RMSE).

$$\begin{aligned} \text{RMSE}(I, \tilde{I}) &= \sqrt{\text{MSE}(I, \tilde{I})} \\ &= \left[\frac{1}{M \cdot N} \cdot \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} [I(u, v) - \tilde{I}(u, v)]^2 \right]^{1/2}. \end{aligned} \quad (\text{C.26})$$

Signal-to-noise ratio (SNR). This is the ratio between the average signal power P_{signal} and the average noise power P_{noise} , that is,

$$\text{SNR}(I, \tilde{I}) = \frac{P_{\text{signal}}}{P_{\text{noise}}} = \frac{\sum_{u=0}^{M-1} \sum_{v=0}^{N-1} I^2(u, v)}{\sum_{u=0}^{M-1} \sum_{v=0}^{N-1} [I(u, v) - \tilde{I}(u, v)]^2}. \quad (\text{C.27})$$

The SNR is commonly given on a logarithmic scale with the unit *decibel* (dB):

$$\text{SNR}_{[\text{dB}]}(I, \tilde{I}) = 10 \cdot \log_{10}(\text{SNR}(I, \tilde{I})). \quad (\text{C.28})$$

Peak signal-to-noise ratio (PSNR). This measure relates the maximum range of values (dynamic range) in the original image to the *mean square error* (MSE) in the form

$$\text{PSNR}(I, \tilde{I}) = \frac{[\max_{u,v} I(u, v) - \min_{u,v} I(u, v)]^2}{\frac{1}{M \cdot N} \cdot \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} [I(u, v) - \tilde{I}(u, v)]^2}, \quad (\text{C.29})$$

or $\text{PSNR}_{[\text{dB}]}(I, \tilde{I}) = 10 \cdot \log_{10}(\text{PSNR}(I, \tilde{I}))$ on a logarithmic scale.

D

Gaussian Filters

This part supplements the material presented in Chapter 7 (SIFT).

D.1 Cascading Gaussian filters

To compute a Gaussian scale space efficiently, the scale layers are usually not obtained directly from the input image by smoothing with Gaussians of increasing size. Instead, each layer can be calculated recursively from the previous layer by filtering with relatively small Gaussians. Thus, the entire scale space is implemented as a concatenation or “cascade” of smaller Gaussian filters.¹

If Gaussian filters of sizes σ_1, σ_2 are applied successively to the same image, the resulting smoothing effect is identical to using a single larger Gaussian filter H_σ^G , that is

$$(I * H_{\sigma_1}^G) * H_{\sigma_2}^G = I * (H_{\sigma_1}^G * H_{\sigma_2}^G) = I * H_\sigma^G, \quad (\text{D.1})$$

with $\sigma = \sqrt{\sigma_1^2 + \sigma_2^2}$ being the size of the resulting combined filter [61, Sec. 4.5.4]. Put in other words, the *variances* (squares of the σ values) of successive Gaussian filters add up, i. e.,

$$\sigma^2 = \sigma_1^2 + \sigma_2^2. \quad (\text{D.2})$$

In the special case of the *same* Gaussian filter being applied twice ($\sigma_1 = \sigma_2$), the effective width of the combined filter is $\sigma = \sqrt{2} \cdot \sigma_1$. Given an image that is already pre-smoothed by a Gaussian filter of width σ_1 and should be smoothed

¹ See Sec. 7.1.1 for details.

to some target scale $\sigma_2 > \sigma_1$, the required width σ_d of the additional Gaussian filter is

$$\sigma_d = \sqrt{\sigma_2^2 - \sigma_1^2}. \quad (\text{D.3})$$

In a Gaussian scale space, the scale corresponding to each level is proportional to the width (σ) of the Gaussian filter applied to derive the corresponding image. Usually the neighboring layers of the scale space differ by a constant scale factor (κ) and the transformation from one scale level to another can be accomplished by successively applying Gaussian filters. Despite the constant scale factor, however, the width of the required filters is *not* constant but depends on the image's initial scale. In particular, if we want to transform an image with scale σ_0 by a factor κ to a new scale $\kappa \cdot \sigma_0$, then (from Eqn. (D.2)) for σ_d the relation

$$(\kappa\sigma_0)^2 = \sigma_0^2 + \sigma_d^2 \quad (\text{D.4})$$

must hold. Thus, the width σ_d of the required Gaussian smoothing filter is

$$\sigma_d = \sigma_0 \cdot \sqrt{\kappa^2 - 1}. \quad (\text{D.5})$$

For example, doubling the scale ($\kappa = 2$) of an image that is pre-smoothed with σ_0 requires a Gaussian filter of width $\sigma_d = \sigma_0 \cdot \sqrt{2^2 - 1} = \sigma_0 \cdot \sqrt{3} \approx \sigma_0 \cdot 1.732$.

D.2 Effects of Gaussian filtering in the frequency domain

Doubling the σ of a Gaussian filter corresponds to cutting the bandwidth by half. For the one-dimensional Gaussian function

$$g_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{-\frac{x^2}{2\sigma^2}}, \quad (\text{D.6})$$

the continuous Fourier transform $\mathcal{F}(g_\sigma)$ is

$$G_\sigma(\omega) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{\omega^2\sigma^2}{2}}. \quad (\text{D.7})$$

If σ is doubled, the Fourier transform $\mathcal{F}(g_{2\sigma})$ becomes

$$G_{2\sigma}(\omega) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{\omega^2(2\sigma)^2}{2}} = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{4\omega^2\sigma^2}{2}} \quad (\text{D.8})$$

$$= \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{(2\omega)^2\sigma^2}{2}} = G_\sigma(2\omega), \quad (\text{D.9})$$

and, in general,

$$G_{k\sigma}(\omega) = G_\sigma(k\omega). \quad (\text{D.10})$$

That is, if σ is *increased* (or the kernel widened) by a factor k , the corresponding Fourier transform $G_\sigma(\omega)$ gets *contracted* by the same factor. In terms of linear filtering this means that widening the kernel by some factor k decimates the resulting signal bandwidth by $\frac{1}{k}$.

D.3 LoG-approximation by the difference of two Gaussians (DoG)

The Laplacian-of-Gaussian (LoG) kernel

$$\text{LoG}_\sigma(x, y) = (\nabla^2 g_\sigma)(x, y) = \frac{1}{\pi\sigma^4} \cdot \left(\frac{x^2 + y^2 - 2\sigma^2}{2\sigma^2} \right) \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}, \quad (\text{D.11})$$

has a (negative) peak at the origin with

$$\text{LoG}_\sigma(0, 0) = -\frac{1}{\pi\sigma^4}. \quad (\text{D.12})$$

Thus, the scale normalized LoG kernel, defined in Eqn. (7.9) as

$$\widehat{\text{LoG}}_\sigma(x, y) = \sigma^2 \cdot \text{LoG}_\sigma(x, y), \quad (\text{D.13})$$

has a peak value of

$$\widehat{\text{LoG}}_\sigma(0, 0) = -\frac{1}{\pi\sigma^2}. \quad (\text{D.14})$$

In comparison, the unscaled Difference-of-Gaussians (DoG) function

$$\begin{aligned} \text{DoG}_{\sigma,\kappa}(x, y) &= G_{\kappa\sigma}(x, y) - G_\sigma(x, y) \\ &= \frac{1}{2\pi\kappa^2\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\kappa^2\sigma^2}} - \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}} \end{aligned} \quad (\text{D.15})$$

has a peak value of

$$\text{DoG}_{\sigma,\kappa}(0, 0) = -\frac{\kappa^2 - 1}{2\pi\kappa^2\sigma^2}. \quad (\text{D.16})$$

By scaling the DoG function to match the LoG's center peak value, i.e., $\text{LoG}_\sigma(0, 0) = \lambda \cdot \text{DoG}_{\sigma,\kappa}(0, 0)$, the original LoG (Eqn. (D.11)) is approximated by the DoG in the form

$$\text{LoG}_\sigma(x, y) \approx \frac{2\kappa^2}{\sigma^2(\kappa^2 - 1)} \cdot \text{DoG}_{\sigma,\kappa}(x, y). \quad (\text{D.17})$$

Similarly, the scale normalized LoG (Eqn. (D.13)) is approximated by the DoG as²

$$\widehat{\text{LoG}}_\sigma(x, y) \approx \frac{2\kappa^2}{\kappa^2 - 1} \cdot \text{DoG}_{\sigma, \kappa}(x, y). \quad (\text{D.18})$$

Note that, for a constant size ratio κ , the DoG approximation is proportional to the scale normalized LoG for any scale σ .

² A different formulation, $\widehat{\text{LoG}}_\sigma(x, y) \approx \frac{1}{\kappa-1} \cdot \text{DoG}_{\sigma, \kappa}(x, y)$, is given in [80], which is the same as Eqn. (D.18) for $\kappa \rightarrow 1$, but not for $\kappa > 1$. The essence is that the leading factor is constant and independent of σ , and can thus be ignored when comparing the magnitude of the filter responses at varying scales.

E

Color Space Transformations

This part supplements the material presented in Chapter 3 (particularly Sec. 3.1.2). For additional details on color space definitions and transformations see Chapter 6 of Volume 2 [21]. Some of this material has been replicated here for easier access.

E.1 RGB/sRGB transformations

$XYZ \rightarrow sRGB$

To transform a given XYZ color to sRGB, the *linear* R, G, B values (assumed to be in the range $[0, 1]$) are first calculated by multiplying the (X, Y, Z) coordinate vector with the matrix M_{RGB} ,

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = M_{\text{RGB}} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 3.240479 & -1.537150 & -0.498535 \\ -0.969256 & 1.875992 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}. \quad (\text{E.1})$$

Subsequently, a modified gamma correction (see Vol. 1 [20, Sec. 4.7.6]) with $\gamma = 2.4$ is applied to the linear R, G, B values to obtain the non-linear sRGB components,

$$\begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} = \begin{pmatrix} f_\gamma(R) \\ f_\gamma(G) \\ f_\gamma(B) \end{pmatrix}, \quad (\text{E.2})$$

$$\text{with } f_\gamma(c) = \begin{cases} 1.055 \cdot c^{\frac{1}{2.4}} - 0.055 & \text{for } c > 0.0031308, \\ 12.92 \cdot c & \text{for } c \leq 0.0031308. \end{cases} \quad (\text{E.3})$$

The resulting R', G', B' values are limited to the $[0, 1]$ range.

sRGB \rightarrow XYZ

To calculate the transformation from sRGB back to XYZ, the non-linear R', G', B' values (in the range $[0, 1]$) are first linearized by inverting the gamma correction in Eqn. (E.3),¹

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} f_\gamma^{-1}(R') \\ f_\gamma^{-1}(G') \\ f_\gamma^{-1}(B') \end{pmatrix}, \quad (\text{E.4})$$

$$\text{with } f_\gamma^{-1}(c') = \begin{cases} \left(\frac{c'+0.055}{1.055}\right)^{2.4} & \text{for } c' > 0.03928, \\ \frac{c'}{12.92} & \text{for } c' \leq 0.03928. \end{cases} \quad (\text{E.5})$$

Finally, the linearized (R, G, B) vector is transformed to XYZ coordinates by multiplication with the inverse of the matrix M_{RGB} , i. e.,

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = M_{\text{RGB}}^{-1} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}. \quad (\text{E.6})$$

E.2 CIELAB/CIELUV transformations

This section summarizes the transformations between the CIEXYZ, CIELAB and CIELUV color spaces. The notation used here aims at visualizing the similarities of the transforms where applicable. It deviates slightly from the description of L*a*b* in Vol. 2 [21, Sec. 6.1], using precise rational constants instead of approximate floating-point values. The transformations require a reference white point $(X_{\text{ref}}, Y_{\text{ref}}, Z_{\text{ref}})$ to be specified, such as the common D65 reference (CIE 2° Standard Observer),

$$X_{\text{ref}} = 0.950456, \quad Y_{\text{ref}} = 1.000000, \quad Z_{\text{ref}} = 1.088754. \quad (\text{E.7})$$

Additional details regarding these transformations can be found in [107, 117].

¹ See Eqn. (4.35) in Vol. 1 [20, p. 86].

E.2.1 CIELAB

XYZ → **LAB**

Calculate (L^*, a^*, b^*) from given (X, Y, Z) color coordinates:

$$\begin{aligned} L^* &= 116 \cdot Y' - 16, \\ a^* &= 500 \cdot (X' - Y'), \\ b^* &= 200 \cdot (Y' - Z'), \end{aligned} \quad (\text{E.8})$$

where

$$X' = f_1\left(\frac{X}{X_{\text{ref}}}\right), \quad Y' = f_1\left(\frac{Y}{Y_{\text{ref}}}\right), \quad Z' = f_1\left(\frac{Z}{Z_{\text{ref}}}\right), \quad (\text{E.9})$$

and

$$\begin{aligned} f_1(c) &= \begin{cases} c^{1/3} & \text{for } c > \epsilon, \\ \kappa \cdot c + \frac{16}{116} & \text{for } c \leq \epsilon, \end{cases} \quad \text{with} \\ \epsilon &= \left(\frac{6}{29}\right)^3 = \frac{216}{24389} \approx 0.008856, \\ \kappa &= \frac{1}{116} \left(\frac{29}{3}\right)^3 = \frac{841}{108} \approx 7.787. \end{aligned} \quad (\text{E.10})$$

Using exact rational values for the constants ϵ and κ precludes any discontinuities in the resulting two-sectioned (linear-cubic) gamma function $f_1()$ and its inverse $f_2()$ in Eqn. (E.12). The corresponding (approximate) decimal values are the ones specified in the original standard [117, pp. 61–64]. For sRGB color images, the resulting $L^*a^*b^*$ values are in the ranges $L^* \in [0, 100]$, $a^* \in [-86.2, 98.2]$, and $b^* \in [-107.9, 94.5]$.

LAB → **XYZ**

Calculate (X, Y, Z) from given (L^*, a^*, b^*) color coordinates:

$$\begin{aligned} Y &= Y_{\text{ref}} \cdot f_2(L'), \\ X &= X_{\text{ref}} \cdot f_2\left(L' + \frac{a^*}{500}\right), \\ Z &= Z_{\text{ref}} \cdot f_2\left(L' - \frac{b^*}{200}\right), \end{aligned} \quad (\text{E.11})$$

where

$$L' = \frac{L^* + 16}{116} \quad \text{and} \quad f_2(c) = \begin{cases} c^3 & \text{for } c^3 > \epsilon, \\ \frac{c - 16/116}{\kappa} & \text{for } c^3 \leq \epsilon, \end{cases} \quad (\text{E.12})$$

with ϵ, κ as defined in Eqn. (E.10).

E.2.2 CIELUV

XYZ → **LUV**

Calculate (L^*, u^*, v^*) from given (X, Y, Z) color coordinates:

$$\begin{aligned} L^* &= 116 \cdot Y' - 16, \\ u^* &= 13 \cdot L^* \cdot (u' - u'_{\text{ref}}), \\ v^* &= 13 \cdot L^* \cdot (v' - v'_{\text{ref}}), \end{aligned} \quad (\text{E.13})$$

with Y' as defined in Eqn. (E.9) and

$$\begin{aligned} u' &= f_u(X, Y, Z), & u'_{\text{ref}} &= f_u(X_{\text{ref}}, Y_{\text{ref}}, Z_{\text{ref}}), \\ v' &= f_v(X, Y, Z), & v'_{\text{ref}} &= f_v(X_{\text{ref}}, Y_{\text{ref}}, Z_{\text{ref}}), \end{aligned} \quad (\text{E.14})$$

with

$$f_u(x, y, z) = \begin{cases} 0 & \text{for } x = 0, \\ \frac{4x}{x + 15y + 3z} & \text{for } x > 0, \end{cases} \quad (\text{E.15})$$

$$f_v(x, y, z) = \begin{cases} 0 & \text{for } y = 0, \\ \frac{9y}{x + 15y + 3z} & \text{for } y > 0. \end{cases} \quad (\text{E.16})$$

For sRGB color images, the resulting $L^*u^*v^*$ values are in the ranges $L^* \in [0, 100]$, $u^* \in [-83, 175]$, and $v^* \in [-134.1, 107.4]$. Note that the checks for zero x, y in Eqn. (E.15) and Eqn. (E.16), respectively, are not found in the original definitions but are essential in any real implementation to avoid divisions by zero.²

LUV → **XYZ**

Calculate (X, Y, Z) from given (L^*, u^*, v^*) color coordinates:

$$Y = Y_{\text{ref}} \cdot f_2\left(\frac{L^* + 16}{116}\right), \quad (\text{E.17})$$

with $f_2()$ as defined in Eqn. (E.12),

$$X = Y \cdot \frac{9u'}{4v'}, \quad Z = Y \cdot \frac{12 - 3u' - 20v'}{4v'}, \quad (\text{E.18})$$

with

$$u' = \begin{cases} u'_{\text{ref}} & \text{for } L^* = 0, \\ \frac{u^*}{13 \cdot L^*} + u'_{\text{ref}} & \text{for } L^* > 0, \end{cases} \quad v' = \begin{cases} v'_{\text{ref}} & \text{for } L^* = 0, \\ \frac{v^*}{13 \cdot L^*} + v'_{\text{ref}} & \text{for } L^* > 0, \end{cases} \quad (\text{E.19})$$

² Remember though that floating-point values (`double`, `float`) should never be tested against 0.0 but against a sufficiently small (`epsilon`) quantity.

and $u'_{\text{ref}}, v'_{\text{ref}}$ as defined in Eqn. (E.14). Notice that no explicit check for zero denominators is required in Eqn. (E.18) since v' can be assumed to be greater than zero.

Bibliography

- [1] L. ALVAREZ, P.-L. LIONS, AND J.-M. MOREL. Image selective smoothing and edge detection by nonlinear diffusion (II). *SIAM Journal on Numerical Analysis* **29**(3), 845–866 (1992).
- [2] Apache Software Foundation. “Commons Math: The Apache Commons Mathematics Library”. <http://commons.apache.org/math/index.html>.
- [3] K. ARBTER, W. E. SNYDER, H. BURKHARDT, AND G. HIRZINGER. Application of affine-invariant Fourier descriptors to recognition of 3-D objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**(7), 640–647 (1990).
- [4] G. R. ARCE, J. BACCA, AND J. L. PAREDES. Nonlinear filtering for image analysis and enhancement. In A. BOVIK, editor, “Handbook of Image and Video Processing”, pp. 109–133. Academic Press, New York, second ed. (2005).
- [5] S. ARYA, D. M. MOUNT, N. S. NETANYAHU, R. SILVERMAN, AND A. Y. WU. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM* **45**(6), 891–923 (November 1998).
- [6] J. ASTOLA, P. HAAVISTO, AND Y. NEUVO. Vector median filters. *Proceedings of the IEEE* **78**(4), 678–689 (April 1990).
- [7] J. BABAUD, A. P. WITKIN, M. BAUDIN, AND R. O. DUDA. Uniqueness of the Gaussian kernel for scale-space filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8**(1), 26–33 (January 1986).

- [8] D. BARASH. Fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(6), 844–847 (June 2002).
- [9] M. BARNI. A fast algorithm for 1-norm vector median filtering. *IEEE Transactions on Image Processing* **6**(10), 1452–1455 (October 1997).
- [10] H. BAY, A. ESS, T. TUYTELAARS, AND L. VAN GOOL. SURF: Speeded up robust features. *Computer Vision, Graphics, and Image Processing: Image Understanding* **110**(3), 346–359 (2008).
- [11] J. S. BEIS AND D. G. LOWE. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In “Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR’97)”, pp. 1000–1006, Puerto Rico (June 1997).
- [12] J. BERNSEN. Dynamic thresholding of grey-level images. In “Proceedings of the International Conference on Pattern Recognition (ICPR)”, pp. 1251–1255, Paris (October 1986). IEEE Computer Society.
- [13] C. M. BISHOP. “Pattern Recognition and Machine Learning”. Springer, New York (2006).
- [14] I. BLAYVAS, A. BRUCKSTEIN, AND R. KIMMEL. Efficient computation of adaptive threshold surfaces for image binarization. *Pattern Recognition* **39**(1), 89–101 (2006).
- [15] J. BLINN. Consider the lowly 2×2 matrix. *IEEE Computer Graphics and Applications* **16**(2), 82–88 (1996).
- [16] J. BLINN. “Jim Blinn’s Corner: Notation, Notation, Notation”. Morgan Kaufmann (2002).
- [17] A. I. BORISENKO AND I. E. TARAPOV. “Vector and Tensor Analysis with Applications”. Dover Publications, New York (1979).
- [18] M. BROWN AND D. LOWE. Invariant features from interest point groups. In “Proceedings of the British Machine Vision Conference”, pp. 656–665 (2002).
- [19] W. BURGER AND M. J. BURGE. “Digital Image Processing—An Algorithmic Introduction using Java”. Texts in Computer Science. Springer, New York (2008).
- [20] W. BURGER AND M. J. BURGE. “Principles of Digital Image Processing – Fundamental Techniques (Vol. 1)”. Undergraduate Topics in Computer Science. Springer, New York (2009).

- [21] W. BURGER AND M. J. BURGE. “Principles of Digital Image Processing – Core Algorithms (Vol. 2)”. Undergraduate Topics in Computer Science. Springer, London (2009).
- [22] P. J. BURT AND E. H. ADELSON. The Laplacian pyramid as a compact image code. *IEEE Transactions on Communications* **31**(4), 532–540 (1983).
- [23] J. F. CANNY. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8**(6), 679–698 (1986).
- [24] F. CATTÉ, P.-L. LIONS, J.-M. MOREL, AND T. COLL. Image selective smoothing and edge detection by nonlinear diffusion. *SIAM Journal on Numerical Analysis* **29**(1), 182–193 (1992).
- [25] C. I. CHANG, Y. DU, J. WANG, S. M. GUO, AND P. D. THOUIN. Survey and comparative analysis of entropy and relative entropy thresholding techniques. *IEE Proceedings—Vision, Image and Signal Processing* **153**(6), 837–850 (December 2006).
- [26] P. CHARBONNIER, L. BLANC-FERAUD, G. AUBERT, AND M. BARLAUD. Two deterministic half-quadratic regularization algorithms for computed imaging. In “Proceedings IEEE International Conference on Image Processing (ICIP-94)”, vol. 2, pp. 168–172, Austin (November 1994).
- [27] Y. CHEN AND G. LEEDHAM. Decompose algorithm for thresholding degraded historical document images. *IEE Proceedings—Vision, Image and Signal Processing* **152**(6), 702–714 (December 2005).
- [28] H. D. CHENG, X. H. JIANG, Y. SUN, AND J. WANG. Color image segmentation: advances and prospects. *Pattern Recognition* **34**(12), 2259–2281 (2001).
- [29] B. COLL, J. L. LISANI, AND C. SBERT. Color images filtering by anisotropic diffusion. In “Proceedings of the IEEE International Conference on Systems, Signals, and Image Processing (IWSSIP)”, pp. 305–308, Chalkida, Greece (2005).
- [30] D. COMANICIU AND P. MEER. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**(5), 603–619 (May 2002).
- [31] R. L. COSGRIFF. Identification of shape. Technical Report 820-11, Antenna Laboratory, Ohio State University, Department of Electrical Engineering, Columbus, Ohio (December 1960).

- [32] T. R. CRIMMINS. A complete set of Fourier descriptors for two-dimensional shapes. *IEEE Transactions on Systems, Man, and Cybernetics* **12**(6), 848–855 (November 1982).
- [33] A. CUMANI. Edge detection in multispectral images. *Computer Vision, Graphics and Image Processing* **53**(1), 40–51 (1991).
- [34] A. CUMANI. Efficient contour extraction in color images. In “Proceedings of the Third Asian Conference on Computer Vision”, ACCV, pp. 582–589, Hong Kong (January 1998). Springer.
- [35] R. DERICHE. Using Canny’s criteria to derive a recursively implemented optimal edge detector. *International Journal of Computer Vision* **1**(2), 167–187 (1987).
- [36] S. DI ZENZO. A note on the gradient of a multi-image. *Computer Vision, Graphics and Image Processing* **33**(1), 116–125 (1986).
- [37] R. O. DUDA, P. E. HART, AND D. G. STORK. “Pattern Classification”. Wiley, New York (2001).
- [38] F. DURAND AND J. DORSEY. Fast bilateral filtering for the display of high-dynamic-range images. In “Proceedings of the 29th annual conference on Computer graphics and interactive techniques (SIGGRAPH’02)”, pp. 257–266, San Antonio, Texas (July 2002).
- [39] M. ELAD. On the origin of the bilateral filter and ways to improve it. *IEEE Transactions on Image Processing* **11**(10), 1141–1151 (October 2002).
- [40] L. M. J. FLORACK, B. M. TER HAAR ROMENY, J. J. KOENDERINK, AND M. A. VIERGEVER. Scale and the differential structure of images. *Image and Vision Computing* **10**(6), 376–388 (1992).
- [41] J. H. FRIEDMAN, J. L. BENTLEY, AND R. A. FINKEL. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software* **3**(3), 209–226 (September 1977).
- [42] D. L. FRITZSCHE. A systematic method for character recognition. Technical Report 1222-4, Antenna Laboratory, Ohio State University, Department of Electrical Engineering, Columbus, Ohio (November 1961).
- [43] T. GEVERS, A. GIJSENIJ, J. VAN DE WEIJER, AND J.-M. GEUSEBROEK. “Color in Computer Vision”. Wiley (2012).
- [44] T. GEVERS AND H. STOKMAN. Classifying color edges in video into shadow-geometry, highlight, or material transitions. *IEEE Transactions on Multimedia* **5**(2), 237–243 (2003).

- [45] T. GEVERS, J. VAN DE WEIJER, AND H. STOKMAN. Color feature detection. In R. LUKAC AND K. N. PLATANIOTIS, editors, “Color Image Processing: Methods and Applications”, pp. 203–226. CRC Press (2006).
- [46] C. A. GLASBEY. An analysis of histogram-based thresholding algorithms. *Computer Vision, Graphics, and Image Processing: Graphical Models and Image Processing* **55**(6), 532–537 (1993).
- [47] R. C. GONZALEZ AND R. E. WOODS. “Digital Image Processing”. Pearson Prentice Hall, Upper Saddle River, NJ, third ed. (2008).
- [48] M. GRABNER, H. GRABNER, AND H. BISCHOF. Fast approximated SIFT. In “Proceedings of the 7th Asian Conference of Computer Vision”, pp. 918–927 (2006).
- [49] G. H. GRANLUND. Fourier preprocessing for hand print character recognition. *IEEE Transactions on Computers* **21**(2), 195–201 (February 1972).
- [50] F. GUICHARD, L. MOISAN, AND J.-M. MOREL. A review of P.D.E. models in image processing and image analysis. *J. Phys. IV France* **12**(1), 137–154 (March 2002).
- [51] J. C. HANCOCK. “An Introduction to the Principles of Communication Theory”. McGraw-Hill (1961).
- [52] I. HANNAH, D. PATEL, AND R. DAVIES. The use of variance and entropic thresholding methods for image segmentation. *Pattern Recognition* **28**(4), 1135–1143 (1995).
- [53] W. W. HARMAN. “Principles of the Statistical Theory of Communication”. McGraw-Hill (1963).
- [54] R. HESS. An open-source SIFT library. In “Proceedings of the International Conference on Multimedia, MM’10”, pp. 1493–1496, Firenze, Italy (October 2010).
- [55] V. HONG, H. PALUS, AND D. PAULUS. Edge preserving filters on color images. In “Proceedings Int’l Conf. on Computational Science, ICCS”, pp. 34–40, Kraków, Poland (2004).
- [56] A. HUERTAS AND G. MEDIONI. Detection of intensity changes with subpixel accuracy using Laplacian-Gaussian masks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8**(5), 651–664 (September 1986).
- [57] J. ILLINGWORTH AND J. KITTLER. Minimum error thresholding. *Pattern Recognition* **19**(1), 41–47 (1986).

- [58] International Telecommunications Union, ITU, Geneva. “ITU-R Recommendation BT.709-3: Basic Parameter Values for the HDTV Standard for the Studio and for International Programme Exchange” (1998).
- [59] B. JÄHNE. “Practical Handbook on Image Processing for Scientific Applications”. CRC Press, Boca Raton, FL (1997).
- [60] B. JÄHNE. “Digitale Bildverarbeitung”. Springer-Verlag, Berlin, fifth ed. (2002).
- [61] R. JAIN, R. KASTURI, AND B. G. SCHUNCK. “Machine Vision”. McGraw-Hill, Boston (1995).
- [62] Y. JIA AND T. DARRELL. Heavy-tailed distances for gradient based image descriptors. In “Proceedings of the Twenty-Fifth Annual Conference on Neural Information Processing Systems (NIPS)”, Grenada, Spain (December 2011).
- [63] L. JIN AND D. LI. A switching vector median filter based on the CIELAB color space for color image restoration. *Signal Processing* **87**(6), 1345–1354 (2007).
- [64] J. N. KAPUR, P. K. SAHOO, AND A. K. C. WONG. A new method for gray-level picture thresholding using the entropy of the histogram. *Computer Vision, Graphics, and Image Processing* **29**, 273–285 (1985).
- [65] B. KIMIA. “A Large Binary Image Database”. LEMS Vision Group, Brown University (2002). <http://www.lems.brown.edu/~dmc/>.
- [66] D. E. KNUTH. “The Art of Computer Programming, Volume 2: Seminumerical Algorithms”. Addison-Wesley, third ed. (1997).
- [67] J. J. KOENDERINK. The structure of images. *Biological Cybernetics* **50**(5), 363–370 (August 1984).
- [68] A. KOSCHAN AND M. A. ABIDI. Detection and classification of edges in color images. *IEEE Signal Processing Magazine* **22**(1), 64–73 (January 2005).
- [69] A. KOSCHAN AND M. A. ABIDI. “Digital Color Image Processing”. Wiley (2008).
- [70] F. P. KUHL AND C. R. GIARDINA. Elliptic Fourier features of a closed contour. *Computer Graphics and Image Processing* **18**(3), 236–258 (1982).
- [71] M. KUWAHARA, K. HACHIMURA, S. EIHO, AND M. KINOSHITA. Processing of RI-angiographic image. In K. PRESTON AND M. ONOE,

- editors, “Digital Processing of Biomedical Images”, pp. 187–202. Plenum, New York (1976).
- [72] V. LEPETIT AND P. FUA. Keypoint recognition using randomized trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **28**(9), 1465–1479 (September 2006).
- [73] P. E. LESTREL, editor. “Fourier Descriptors and Their Applications in Biology”. Cambridge University Press, New York (1997).
- [74] P.-S. LIAO, T.-S. CHEN, AND P.-C. CHUNG. A fast algorithm for multilevel thresholding. *Journal of Information Science and Engineering* **17**, 713–727 (2001).
- [75] C. C. LIN AND R. CHELLAPPA. Classification of partial 2-D shapes using Fourier descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **9**(5), 686–690 (1987).
- [76] B. J. LINDBLOOM. Accurate color reproduction for computer graphics applications. *SIGGRAPH Computer Graphics* **23**(3), 117–126 (1989).
- [77] T. LINDEBERG. “Scale-Space Theory in Computer Vision”. Kluwer Academic Publishers (1994).
- [78] T. LINDEBERG. Feature detection with automatic scale selection. *International Journal of Computer Vision* **30**(2), 77–116 (November 1998).
- [79] D. G. LOWE. Object recognition from local scale-invariant features. In “Proceedings of the 7th IEEE International Conference on Computer Vision”, vol. 2 of “ICCV’99”, pp. 1150–1157, Kerkyra, Corfu, Greece (1999).
- [80] D. G. LOWE. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision* **60**(2), 91–110 (November 2004).
- [81] D. G. LOWE. “Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image”. The University of British Columbia (March 2004). US Patent 6,711,293.
- [82] R. LUKAC, B. SMOLKA, AND K. N. PLATANIOTIS. Sharpening vector median filters. *Signal Processing* **87**(9), 2085–2099 (2007).
- [83] R. LUKAC, B. SMOLKA, K. N. PLATANIOTIS, AND A. N. VENETSANOPoulos. Vector sigma filters for noise detection and removal in color images. *Journal of Visual Communication and Image Representation* **17**(1), 1–26 (2006).

- [84] C. MANCAS-THILLOU AND B. GOSELIN. Color text extraction with selective metric-based clustering. *Computer Vision, Graphics, and Image Processing: Image Understanding* **107**(1-2), 97–107 (2007).
- [85] D. MARR AND E. HILDRETH. Theory of edge detection. *Proceedings of the Royal Society of London, Series B* **207**(1167), 187–217 (1980).
- [86] P. A. MLSNA AND J. J. RODRIGUEZ. Gradient and Laplacian-type edge detection. In A. BOVIK, editor, “Handbook of Image and Video Processing”, pp. 415–431. Academic Press, New York, second ed. (2005).
- [87] J. MOROVIC. “Color Gamut Mapping”. Wiley (2008).
- [88] M. MUJA AND D. G. LOWE. Fast approximate nearest neighbors with automatic algorithm configuration. In “Proceedings of the International Conference on Computer Vision Theory and Application”, VISSAPP’09, pp. 331–340, Lisboa, Portugal (February 2009).
- [89] M. NAGAO AND T. MATSUYAMA. Edge preserving smoothing. *Computer Graphics and Image Processing* **9**(4), 394–407 (1979).
- [90] S. K. NAIK AND C. A. MURTHY. Standardization of edge magnitude in color images. *IEEE Transactions on Image Processing* **15**(9), 2588–2595 (2006).
- [91] W. NIBLACK. “An Introduction to Digital Image Processing”. Prentice-Hall (1986).
- [92] M. NITZBERG AND T. SHIOTA. Nonlinear image filtering with edge and corner enhancement. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**(8), 826–833 (August 1992).
- [93] M. NIXON AND A. AGUADO. “Feature Extraction and Image Processing”. Academic Press, second ed. (2008).
- [94] W. OH AND W. B. LINDQUIST. Image thresholding by indicator kriging. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **21**(7), 590–602 (1999).
- [95] N. OTSU. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics* **9**(1), 62–66 (1979).
- [96] N. R. PAL AND S. K. PAL. A review on image segmentation techniques. *Pattern Recognition* **26**(9), 1277–1294 (1993).
- [97] S. PARIS AND F. DURAND. A fast approximation of the bilateral filter using a signal processing approach. *International Journal of Computer Vision* **81**(1), 24–52 (January 2007).

- [98] O. PELE AND M. WERMAN. A linear time histogram metric for improved SIFT matching. In “Proceedings of the 10th European Conference on Computer Vision (ECCV’08)”, pp. 495–508, Marseille, France (October 2008).
- [99] K. PERLIN. An image synthesizer. *SIGGRAPH Computer Graphics* **19**(3), 287–296 (1985).
- [100] K. PERLIN. Improving noise. In “SIGGRAPH’02: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques”, pp. 681–682, San Antonio, Texas (2002).
- [101] P. PERONA AND J. MALIK. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**(4), 629–639 (July 1990).
- [102] E. PERSOON AND K.-S. FU. Shape discrimination using Fourier descriptors. *IEEE Transactions on Systems, Man and Cybernetics* **7**(3), 170–179 (March 1977).
- [103] E. PERSOON AND K.-S. FU. Shape discrimination using Fourier descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **8**(3), 388–397 (May 1986).
- [104] T. Q. PHAM AND L. J. VAN VLIET. Separable bilateral filtering for fast video preprocessing. In “Proceedings IEEE International Conference on Multimedia and Expo”, pp. CD1–4, Los Alamitos, USA (July 2005). IEEE Computer Society.
- [105] K. N. PLATANIOTIS AND A. N. VENETSANOPoulos. “Color Image Processing and Applications”. Springer (2000).
- [106] F. PORIKLI. Constant time $O(1)$ bilateral filtering. In “Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)”, pp. 1–8, Anchorage (June 2008).
- [107] C. A. POYNTON. “Digital Video and HDTV Algorithms and Interfaces”. Morgan Kaufmann Publishers, San Francisco (2003).
- [108] S. PRAKASH AND F. V. D. HEYDEN. Normalisation of Fourier descriptors of planar shapes. *Electronics Letters* **19**(20), 828–830 (September 1983).
- [109] W. H. PRESS, S. A. TEUKOLSKY, W. T. VETTERLING, AND B. P. FLANNERY. “Numerical Recipes”. Cambridge University Press, third ed. (2007).

- [110] R. R. RAKESH, P. CHAUDHURI, AND C. A. MURTHY. Thresholding in edge detection: a statistical approach. *IEEE Transactions on Image Processing* **13**(7), 927–936 (2004).
- [111] C. W. RICHARD AND H. HEMAMI. Identification of three-dimensional objects using Fourier descriptors of the boundary curve. *IEEE Transactions on Systems, Man, and Cybernetics* **4**(4), 371–378 (July 1974).
- [112] T. W. RIDLER AND S. CALVARD. Picture thresholding using an iterative selection method. *IEEE Transactions on Systems, Man, and Cybernetics* **8**(8), 630–632 (August 1978).
- [113] P. K. SAHOO, S. SOLTANI, A. K. C. WONG, AND Y. C. CHEN. A survey of thresholding techniques. *Computer Vision, Graphics and Image Processing* **41**(2), 233–260 (1988).
- [114] G. SAPIRO. “Geometric Partial Differential Equations and Image Analysis”. Cambridge University Press (2001).
- [115] G. SAPIRO AND D. L. RINGACH. Anisotropic diffusion of multivalued images with applications to color filtering. *IEEE Transactions on Image Processing* **5**(11), 1582–1586 (November 1996).
- [116] J. SAUVOLA AND M. PIETIKÄINEN. Adaptive document image binarization. *Pattern Recognition* **33**(2), 1135–1143 (2000).
- [117] J. SCHANDA. “Colorimetry: Understanding the CIE System”. Wiley (2007).
- [118] M. SEZGIN AND B. SANKUR. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging* **13**(1), 146–165 (2004).
- [119] F. Y. SHIH AND S. CHENG. Automatic seeded region growing for color image segmentation. *Image and Vision Computing* **23**(10), 877–886 (2005).
- [120] S. N. SINHA, J.-M. FRAHM, M. POLLEFEYS, AND Y. GENC. Feature tracking and matching in video using programmable graphics hardware. *Machine Vision and Applications* **22**(1), 207–217 (2011).
- [121] B. SMOLKA, M. SZCZEPANSKI, K. N. PLATANIOTIS, AND A. N. VENET-SANOPoulos. Fast modified vector median filter. In “Proceedings of the 9th International Conference on Computer Analysis of Images and Patterns”, CAIP’01, pp. 570–580, London, UK (2001). Springer-Verlag.
- [122] M. SPIEGEL AND S. LIPSCHUTZ. “Schaum’s Outline of Vector Analysis”. McGraw-Hill, New York, second ed. (2009).

- [123] B. TANG, G. SAPIRO, AND V. CASELLES. Color image enhancement via chromaticity diffusion. *IEEE Transactions on Image Processing* **10**(5), 701–707 (May 2001).
- [124] C.-Y. TANG, Y.-L. WU, M.-K. HOR, AND W.-H. WANG. Modified SIFT descriptor for image matching under interference. In “Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC)”, pp. 3294–3300, Kunming, China (July 2008).
- [125] S. THEODORIDIS AND K. KOUTROUMBAS. “Pattern Recognition”. Academic Press, New York (1999).
- [126] C. TOMASI AND R. MANDUCHI. Bilateral filtering for gray and color images. In “Proceedings Int’l Conf. on Computer Vision”, ICCV’98, pp. 839–846, Bombay (1998).
- [127] F. TOMITA AND S. TSUJI. Extraction of multiple regions by smoothing in selected neighborhoods. *IEEE Transactions on Systems, Man, and Cybernetics* **7**, 394–407 (1977).
- [128] Ø. D. TRIER AND T. TAXT. Evaluation of binarization methods for document images. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(3), 312–315 (March 1995).
- [129] E. TRUCCO AND A. VERRI. “Introductory Techniques for 3-D Computer Vision”. Prentice Hall, Englewood Cliffs, NJ (1998).
- [130] D. TSCHUMPERLÉ. “PDEs Based Regularization of Multivalued Images and Applications”. PhD thesis, Université de Nice, Sophia Antipolis, France (2005).
- [131] D. TSCHUMPERLÉ. Fast anisotropic smoothing of multi-valued images using curvature-preserving PDEs. *International Journal of Computer Vision* **68**(1), 65–82 (June 2006).
- [132] D. TSCHUMPERLÉ AND R. DERICHE. Diffusion PDEs on vector-valued images: local approach and geometric viewpoint. *IEEE Signal Processing Magazine* **19**(5), 16–25 (September 2002).
- [133] D. TSCHUMPERLÉ AND R. DERICHE. Vector-valued image regularization with PDEs: A common framework for different applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **27**(4), 506–517 (April 2005).
- [134] J. VAN DE WEIJER. “Color Features and Local Structure in Images”. PhD thesis, University of Amsterdam (2005).

- [135] M. I. VARDAVOULIA, I. ANDREADIS, AND P. TSALIDES. A new vector median filter for colour image processing. *Pattern Recognition Letters* **22**(6-7), 675–689 (2001).
- [136] A. VEDALDI AND B. FULKERSON. VLFeat: An open and portable library of computer vision algorithms. <http://www.vlfeat.org/> (2008).
- [137] F. R. D. VELASCO. Thresholding using the ISODATA clustering algorithm. *IEEE Transactions on Systems, Man, and Cybernetics* **10**(11), 771–774 (November 1980).
- [138] D. VERNON. “Machine Vision”. Prentice Hall (1999).
- [139] T. P. WALLACE AND P. A. WINTZ. An efficient three-dimensional aircraft recognition algorithm using normalized Fourier descriptors. *Computer Vision, Graphics and Image Processing* **13**(2), 99–126 (1980).
- [140] J. WEICKERT. “Anisotropic Diffusion in Image Processing”. PhD thesis, Universität Kaiserslautern, Fachbereich Mathematik (1996).
- [141] J. WEICKERT. A review of nonlinear diffusion filtering. In B. M. TER HAAR ROMENY, L. FLORACK, J. J. KOENDERINK, AND M. A. VIERGEVER, editors, “Proceedings First International Conference on Scale-Space Theory in Computer Vision, Scale-Space’97”, Lecture Notes in Computer Science, pp. 3–28, Utrecht (July 1997). Springer.
- [142] J. WEICKERT. Coherence-enhancing diffusion filtering. *International Journal of Computer Vision* **31**(2/3), 111–127 (April 1999).
- [143] J. WEICKERT. Coherence-enhancing diffusion of colour images. *Image and Vision Computing* **17**(3/4), 201–212 (March 1999).
- [144] B. WEISS. Fast median and bilateral filtering. *ACM Transactions on Graphics* **25**(3), 519–526 (July 2006).
- [145] M. WELK, J. WEICKERT, F. BECKER, C. SCHNÖRR, C. FEDDERN, AND B. BURGETH. Median and related local filters for tensor-valued images. *Signal Processing* **87**(2), 291–308 (2007).
- [146] M.-F. WU AND H.-T. SHEU. Contour-based correspondence using Fourier descriptors. *IEE Proceedings—Vision, Image and Signal Processing* **144**(3), 150–160 (June 1997).
- [147] G. WYSZECKI AND W. S. STILES. “Color Science: Concepts and Methods, Quantitative Data and Formulae”. Wiley–Interscience, New York, second ed. (2000).

- [148] Q. YANG, K.-H. TAN, AND N. AHUJA. Real-time O(1) bilateral filtering. In “Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)”, pp. 557–564, Miami (2009).
- [149] S. D. YANOWITZ AND A. M. BRUCKSTEIN. A new method for image segmentation. *Computer Vision, Graphics, and Image Processing* **46**(1), 82–95 (1989).
- [150] G. W. ZACK, W. E. ROGERS, AND S. A. LATT. Automatic measurement of sister chromatid exchange frequency. *Journal of Histochemistry and Cytochemistry* **25**(7), 741–753 (1977).
- [151] C. T. ZAHN AND R. Z. ROSKIES. Fourier descriptors for plane closed curves. *IEEE Transactions on Computers* **21**(3), 269–281 (March 1972).
- [152] S.-Y. ZHU, K. N. PLATANIOTIS, AND A. N. VENETSANOPoulos. Comprehensive analysis of edge detection in color image processing. *Optical Engineering* **38**(4), 612–625 (1999).

Index

Symbols

\leftarrow (assignment operator) 300
 \leftarrow^+ (increm. assignment operator) 300
 \approx (approx. equal) 300
 $*$ (convolution operator) 300
 \equiv (equivalent) 300
 \cup (concatenation operator) 300
 μ (mean) 302
 ∇ (gradient operator) 84, 93, 99,
156–158, 301, 315, 316, 323, 324, 327
 ∇^2 (Laplacian operator) 144, 145, 232,
301, 317, 339
 \ominus (erosion operator) 300
 \oplus (dilation operator) 300
 \otimes (cross product operator) 300
 ∂ (partial derivative) 88, 93, 301, 315,
317
 σ (standard deviation) 235, 303
 \sim (similarity operator) 300
 $\lceil \cdot \rceil$ (ceil operator) 300
 $\lfloor \cdot \rfloor$ (floor operator) 300

A

`AdaptiveThreshold`(class) 45–47
`AdaptiveThresholdGauss`(algorithm) 44
`addGaussianNoise`(method) 335
affine combination 53
aggregate distance 68
– trimmed 74
anisotropic diffusion filter 143–161
Apache Commons Math Library 208
Apache Commons Math library 310

`applyTo`(method) 76, 164
`Arctan`(function) 301
`atan2`(method) 301
authors
– how to contact vii
AVI video 296

B

background 6
`BackgroundMode`(class) 45
`BackgroundMode`(enum type) 47
bandwidth 242, 246, 338
Bayesian decision-making 24
Bernsen thresholding 30–32
`BernsenThreshold`(algorithm) 32
`BernsenThreshold`(class) 48
Bilateral filter 126–142
– color 132
– Gaussian 131
– separable 136
`BilateralFilter`(class) 164
`BilateralFilterColor`(algorithm) 138
`BilateralFilterGray`(algorithm) 132
`BilateralFilterGraySeparable`(algorithm)
140
`BilateralFilterSeparable`(class)
165
binarization 5
blob 248
`blur`(method) 43
`blurFloat`(method) 43, 46
`blurGaussian`(method) 43
bonus chapter vii, 3

- book's website vii
 border handling 42
 boundary 170
 – pixels 39
 box filter 42
 Brent's method 208
BrentOptimizer(class) 208
 brightness 17, 59
BuildGaussianScaleSpace(algorithm)
 247
BuildSiftScaleSpace(algorithm) 254
ByteProcessor(class) 33, 48, 50, 224
- C**
 Canny edge detection 103–110
 – color 105–110, 115
 – grayscale 103–105, 115
CannyEdgeDetector(algorithm) 108
CannyEdgeDetector(class) 115–117
 card (operator) 301
 cardinality 301
 cascaded Gaussian filters 238, 337
 ceil (operator) 300
 centroid 180, 183, 329
 characteristic equation 310, 311
 chord algorithm 7
 chromaticity 62
 CIE standard observer 342
 CIELAB 55, 61, 152, 343
 CIELUV 55, 61, 152, 344–345
 CIEXYZ 56, 61, 341–345
 circle 181, 182
 collinear 312
 color
 – covariance matrix 126
 – edge detection 83–117
 – edge magnitude 95
 – edge orientation 97
 – filter 51–81, 132, 149
 – linear mixture 54
 – out-of-gamut 62
 – space 55–66, 341–345
 – thresholding 49
ColorCannyEdgeDetector(algorithm)
 111
ColorEdgeDetector(class) 115
Complex(class) 221
 concatenation 300
 conditional probability 24, 333
 conduction coefficient 144
 conductivity 144
 – function 147–149, 151, 153–155, 165
 contrast 17
 convex hull 53
 convolution 43, 52, 237, 319
 covariance 329
 – efficient calculation 330
 covariance matrix 330
 – color 126
 cross product 206
 cumulative distribution function 19
 cumulative histogram 19
- D**
 D65 342
 decibel (dB) 164, 336
Decimate(algorithm) 247
 decimated scale 264
 decimation 245
 derivative 145
 – estimation from discrete samples
 319
 – first order 95, 103, 231, 301, 314, 315
 – higher order 321
 – partial 93, 232, 301, 317
 – second order 232, 258, 301, 317, 318
 determinant 260, 261, 309, 310, 312,
 325
DFT 171–179
 – forward 173
 – inverse 173
 – periodicity 177, 187
 – spectrum 171
 – truncated 177, 179, 187
Di Zenzo/Cumani algorithm 98
 difference-of-Gaussians (DoG) 235, 339
 diffusion process 144
 dimension 329
 Dirac function 237
 direction of maximum contrast 103
 directional gradient 94, 316
 disk filter 42
 distance norm 68, 72, 76, 80, 287, 292,
 295
distanceComplex(method) 221, 222
distanceMagnitude(method) 222
DistanceNorm(class) 76
 distribution
 – normal (Gaussian) 332–334
 divergence 144, 156, 317
DiZenzoCumaniEdgeDetector(class)
 115
 dom (operator) 301
 domain 301, 302
 – filter 128
 dominant orientation 263, 267

dot product 308

E

e (unit vector) 301

edge

- direction 104
- linking 105
- localization 104
- normal 97
- orientation 84, 101
- strength 84
- suppression 260
- tangent 84, 101, 104, 159
- tracing 105

edge detection 83–117

- Canny 103–110

- monochromatic 84–88

- vector-valued (color) 88–103

edge-preserving smoothing filter
119–167

eigenpair 96, 309, 310

eigensystem 159

eigenvalue 96, 99, 116, 159, 260,
309–311, 317

- ratio 261

eigenvector 96, 159, 309–311, 317

- 2×2 matrix 310

ellipse 184, 192

- parameters 184

entropy 18, 19

error measures 335

extremum of a function 258

F

FastIsodataThreshold (algorithm) 15

FastKuwaharaFilter (algorithm) 124

FFT 173

filter

- anisotropic diffusion 143–161
- Bilateral 126–142
- box 42
- cascaded 238
- color 123–126, 132, 149
- color image 51–81
- disk 42
- domain 128
- edge-preserving smoothing 119–167
- Gaussian 42, 104, 119, 131, 159, 230,
239, 337–340
- kernel 52, 84
- Kuwahara-Hachimura 121
- Kuwahara-type 120–126
- Laplacian-of-Gaussian 231

- linear 51–66, 319
- low-pass 43, 246
- min/max 41
- multi-dimensional 68
- Nagao-Matsuyama 121
- non-linear 66–76
- Perona-Malik 146–155
- range 128
- scalar median 66, 80
- separable 43, 242
- sharpening vector median 69
- smoothing 53, 54
- successive Gaussians 238
- Tomita-Tsuji 121
- Tschumperlé-Deriche 157–161
- vector median 67, 80

filterPixel (method) 76

finite differences 145

floor (operator) 300, 302

foreground 6

Fourier descriptor 169–227

- elliptical 225

- from polygon 193

- geometric effects 195–204

- invariance 203–214, 225

- Java implementation 219

- magnitude 214

- matching 214–219, 221

- normalization 203–214, 222

- pair 183–188, 191

- phase 201

- reconstruction 173, 195

- reflection 203

- start point 200

- trigonometric 171, 193, 227

Fourier transform 171–179, 338

FourierDescriptor (class) 219

FourierDescriptorFromPolygon

(algorithm) 196

FourierDescriptorFromPolygon (class)

223

FourierDescriptorUniform (algorithm)

174, 179

FourierDescriptorUniform (class)

223

Frobenius norm 126, 331

function

- biweight 147

- complex-valued 170

- conductivity 147

- differentiable 315

- Dirac 237

- discrete 311, 313, 319

- distance 136, 214, 218
- estimating derivatives 319
- extremum 258, 313
- Gaussian 147, 231, 233, 264, 332
- gradient 93, 316
- hash 214
- Hessian matrix 317, 318
- integral 235
- Jacobian 93, 315
- logarithm 25
- multi-variable 320
- parabolic 311, 313
- partial derivatives 93, 317
- periodic 177
- principal curvature 260
- quadratic 257, 258, 311, 313
- scalar-valued 144, 315, 317, 319
- single-variable 319
- Taylor expansion 257, 319
- trigonometric 105
- vector-valued 88, 315, 316, 318

functional matrix 315

G

- gamma correction 62, 64, 341–343
- Gaussian
 - component 334
 - derivative 231
 - distribution 12, 22, 24, 25, 332, 334
 - filter 42, 131, 159, 230, 239, 337–340
 - function 231, 233
 - kernel 42, 44, 145
 - mixture 22, 334
 - noise 7, 335
 - normalized 43, 44
 - scale space 237, 338
 - successive 238, 337
 - weight 264
- GaussianBlur** (class) 43, 45, 46
- GenericFilter** (class) 76, 164
- getCoefficient** (method) 221
- getCoefficients** (method) 221
- getEdgeBinary** (method) 116
- getEdgeMagnitude** (method) 115
- getEdgeOrientation** (method) 115
- getEdgeTraces** (method) 116
- getf** (method) 335
- getHeight** (method) 335
- getHistogram** (method) 50
- getMaxCoefficientPairs** (method)
 - 221
- getMaxNegHarmonic** (method) 221
- getMaxPosHarmonic** (method) 221

- GetPartialReconstruction** (algorithm) 193
- getReconstruction** (method) 222
- getReconstructionPoint** (method) 222
- getSiftFeatures** (method) 295
- GetStartPointPhase** (algorithm) 212
- getThreshold** (method) 45, 47, 48
- getWidth** (method) 335
- GIMP 161
- GlobalThresholder** (class) 46, 47
- grad** (operator) 301, 315
- gradient 84, 144, 147, 258, 315, 317
 - directional 94, 316
 - magnitude 103, 264
 - maximum direction 316
 - multi-dimensional 88
 - orientation 103, 264
 - scalar 88, 99
 - vector 103, 104
 - vector field 316
- gradient noise 3

H

- harmonic number 177
- Harris corner detection 261
- hash function 214
- heat equation 144
- Hessian matrix 157, 158, 160, 162, 256–258, 260, 276, 278, 279, 302, 317, 318, 323
- discrete estimation 158
- Hessian of a function 318, 319
- histogram 8–9, 302
 - cumulative 19
 - multiple peaks 267
 - orientation 264, 265
 - smoothing 265
- homogeneous region 120
- hysteresis thresholding 103, 105

I

- i (imaginary unit) 302
- identity matrix 156, 302, 309
- image
 - inpainting 161
 - pyramid 242
 - quality measurement 161, 335
- image stack 166, 296
- ImageAccessor** (class) 76
- ImageJ vi
- interest point 230
- intermeans algorithm 11

invariance 203–214

– rotation 208

– scale 204

– start point 205

ISODATA

– clustering 11

– thresholding 11–14, 87

IsodataThreshold (algorithm) 13

IsodataThresholder (class) 47

isotropic 232

J

Jacobian matrix 93, 94, 302, 315–317

JAMA matrix package 310

joint probability 333

K

k-d algorithm 289

key point

– position refinement 257

– selection 252

KIMIA image dataset 195, 227

kriging 49

Kronecker product 309

Kuwahara-Hachimura filter 121

Kuwahara-type filter 120–126

KuwaharaFilter (algorithm) 122, 125

KuwaharaFilter (class) 164

L

LAB 343

Laplacian 144, 145, 157, 232, 318

Laplacian-of-Gaussian (LoG) 231

– approximation by difference of Gaussians 235, 339

– normalized 233

left multiplication (matrix/vector) 307

lightness 59

likelihood function 333

linear equation 309

list 299

– concatenation 300

local

– extremum 255, 314

– structure matrix 96, 99, 159

logarithm 25

low-pass filter 43

luma 59, 60, 152

luminance 50, 59, 60, 152

LUV color space 344–345

M

MakeDogOctave (algorithm) 254

MakeGaussianKernel2D (algorithm) 44

MakeGaussianOctave (algorithm) 247, 254

MakeInvariant (algorithm) 211

makeInvariant (method) 222, 227

MakeRotationInvariant (algorithm) 211

makeRotationInvariant (method) 222

MakeScaleInvariant (algorithm) 211

makeScaleInvariant (method) 222

MakeStartPointInvariant (algorithm)

212

makeStartPointInvariant (method)

223

makeTranslationInvariant (method)

223

map 302

MatchDescriptors (algorithm) 288

matchDescriptors (method) 295

matching 214–219

matrix 305

– Hessian 157, 158, 160, 162, 256–258, 260, 276, 278, 279, 302, 317, 318, 323

– identity 156, 302, 309

– Jacobian 93, 94, 302, 315–317

– norm 126, 331

– rank 309

– singular 309

– symmetric 311

– transpose 306

max-filter 41

MaxEntropyThreshold (class) 47

maximum entropy thresholding 18–22

maximum likelihood 333

maximum local contrast 96

MaximumEntropyThreshold (algorithm) 23

mean 9, 10, 38, 120, 329, 332, 335

– vector 329

mean absolute error (MAE) 336

mean square error (MSE) 336

median

– filter 66, 67

Mexican hat kernel 233

mid-range 10

min-filter 41

MinErrorThreshold (class) 47

minimum error thresholding 22–28

MinimumErrorThreshold (algorithm) 29

mod (operator) 302

mode 332

monochromatic edge detection 84–88

MonochromaticColorEdge (algorithm)

87

- M**
- `MonochromaticEdgeDetector` (class) 115
 - `MultiGradientColorEdge` (algorithm) 98
- N**
- `Nagao-Matsuyama` filter 121
 - negative frequency 183
 - neighborhood
 - 2D 31, 69, 71, 129, 130, 230, 323, 326, 328
 - 3D 256, 257, 259
 - square 121
 - `nextGaussian` (method) 335
 - Niblack thresholding 34–37
 - `NiblackThreshold` (algorithm) 38
 - `NiblackThresholdBox` (class) 48
 - `NiblackThresholdDisk` (class) 48
 - `NiblackThresholderGauss` (class) 45, 46, 48
 - noise 164, 335
 - Gaussian 163, 335
 - reduction 119
 - spot 163
 - synthetic 3, 161
 - non-maximum suppression 103, 105
 - norm 68, 85, 86, 89–92, 136, 138
 - Euclidean 306
 - Frobenius 126, 331
 - matrix 126, 331
 - vector 306
 - normal distribution 332
 - normalized kernel 43, 53
- O**
- OCR 37
 - octave 235, 238–240, 242–248, 254, 256, 270
 - orientation
 - dominant 267
 - histogram 264
 - Otsu’s method 14–18
 - `OtsuThreshold` (algorithm) 17
 - `OtsuThreshold` (class) 47
 - out-of-gamut colors 62
 - outer product 309
 - outlier 11
- P**
- parabolic fitting 311–314
 - partial differential equation 144
 - peak signal-to-noise ratio (PSNR) 336
 - perceptually uniform 61
 - Perlin noise 3
- R**
- `Random` (class) 335
 - random variable 332
 - range 302
 - filter 128
 - rank 309
 - rank ordering 67
 - `RankFilters` (class) 32, 33
 - region
 - homogeneous 120
 - RGB color space 55, 341–342
 - right multiplication (vector/matrix) 307
 - `rng` (operator) 302
- P**
- Perona-Malik filter 146–155
 - color 149
 - gray 146
 - `PeronaMalikColor` (algorithm) 154
 - `PeronaMalikFilter` (class) 165, 166
 - `PeronaMalikGray` (algorithm) 151
 - phase 201, 206, 208, 211
 - Photoshop 67, 85
 - `PlugInFilter` (interface) 76
 - polar method 335
 - polygon 171, 193
 - path length 194
 - uniform sampling 172, 226
 - `PolygonSampler` (class) 223
 - posterior probability 24
 - principal curvature ratio 261
 - prior probability 25, 29
 - probability 19, 332
 - conditional 24, 333
 - density function 19
 - distribution 19
 - joint 333
 - posterior 24
 - prior 19, 25, 27, 29
 - product
 - dot 308
 - inner 308
 - Kronecker 309
 - matrix-matrix 307
 - matrix-vector 307
 - outer 309
 - scalar 308
 - pyramid 242
- Q**
- quadratic function 257, 258, 267, 313
 - `QuantileThreshold` (algorithm) 11
 - `QuantileThreshold` (class) 47

- root mean square error (RMSE) 336
rotation 199
- S**
sample mean vector 329
samplePolygonUniformly(method)
 223
sampling 171
Sauvola-Pietikäinen thresholding 37
SauvolaThreshold(class) 48
scalar field 315–317, 319
scalar median filter 66, 80
ScalarMedianFilter(class) 80
scale
– absolute 238, 245
– base 245
– change 199
– decimated 264
– increment 255
– initial 238
– ratio 238
– relative 240
scale space 231
– decimation 242, 245
– discrete 237
– Gaussian 237
– hierarchical 242, 248
– implementation in SIFT 248–261
– LoG/Dog 240, 248
– octave 242
– spatial position 246
– sub-sampling 242
segmentation 5, 49
separability 43
sequence 299
set 299
setCoefficient(method) 221
setf(method) 335
shape
– reconstruction 173, 187, 192, 193,
 195, 222
– rotation 199
sharpening vector median filter 69
SharpeningVectorMedianFilter(algorithm)
 75
ShortProcessor(class) 50
SIFT 229–296
– algorithm summary 276
– descriptor 267–276
– examples 287–289
– feature matching 276–294
– implementation 259, 276, 294
– parameters 285
– scale space 248–261
SiftDescriptor(class) 295
SiftDetector(class) 295
SiftMatcher(class) 295
signal-to-noise ratio (SNR) 163, 336
size(method) 221
smoothing kernel 42
Sobel operator 84, 86
Sombrero kernel 233
source code repository vi
squared local contrast 94, 99
sRGB color space 55, 341–342
standard deviation 34
statistical independence 332, 333
step edge 54
structure matrix 160
sub-sampling 246
symmetry 203
synthetic noise 3
- T**
Taylor expansion 258, 259, 319–327
– multi-dimensional 320–327
threshold(method) 48
threshold surface 49
Threshold(class) 46, 47
thresholding 5–50
– Bernsen 30–32
– color image 49, 50
– global 6–28
– hysteresis 103
– ISODATA 11–14
– local adaptive 30–43
– maximum entropy 18–22
– minimum error 22–28
– Niblack 34–37
– Otsu 14–18
– shape-based 7
– statistical 7
Tomita-Tsuji filter 121
total variance 126, 331
trace 126, 156, 157, 162, 303, 317, 318,
 331
tracking 296
translation 197
transpose of a matrix 306
triangle algorithm 7
trigonometric coefficient 194
trimmed aggregate distance 74
truncate(method) 221, 227
truncate function 303
truncated spectrum 177, 179
TschumperleDericheFilter(algorithm)
 162

- TschumperléDericheFilter** (class)
 165
Tschumperlé-Deriche filter 157–161
tuple 299
- U**
- unit vector** 94, 97, 256, 301, 316
- V**
- variance** 9, 34, 120, 122, 329, 330, 332,
 335, 337
 – between classes 15
 – local calculation 38
 – total 126, 331
 – within class 15
variate 329
vector 299, 305–311
 – column 306
- W**
- website** vii
white point 342
- X**
- XYZ color space** 341–345

About the Authors

Wilhelm Burger received a Master's degree in Computer Science from the University of Utah (Salt Lake City) and a doctorate in Systems Science from Johannes Kepler University in Linz, Austria. As a post-graduate researcher at the Honeywell Systems & Research Center in Minneapolis and the University of California at Riverside, he worked mainly in the areas of visual motion analysis and autonomous navigation. In the Austrian research initiative on digital imaging, he led projects on generic object recognition and biometric identification. Since 1996, he has been the Director of the Digital Media degree programs at the University of Applied Sciences in Hagenberg, Upper Austria; he currently serves as Dean of the school. Personally, Wilhelm appreciates large-engine vehicles, chamber music, and (occasionally) a glass of dry "Veltliner".



Mark J. Burge is a senior scientist at MITRE in Washington, D.C. He spent seven years as a research scientist with the Swiss Federal Institute of Science (ETH) in Zürich and the Johannes Kepler University in Linz, Austria. He earned tenure as a computer science professor in the University System of Georgia (USG), and later served as a program director at the National Science Foundation (NSF). Personally, Mark is an expert on classic Italian espresso machines.



About this Book Series

The authors prepared the "camera-ready" manuscript for this book completely in LaTeX using Donald Knuth's Computer Modern fonts. Additional LaTeX packages for presenting algorithms (*algorithmicx* by János Szász), source code (*listings*, Carsten Heinz), and typesetting text in graphics (*pstrag* by Michael C. Grant and David Carlisle) were particularly helpful in this task. All illustrations were produced by the authors using ImageJ, iText, Mathematica, and Adobe Freehand. Please visit the book's support site, www.imagingbook.com, for additional features, including: complete source code for all examples, digital versions of all figures, and up-to-date page errata.