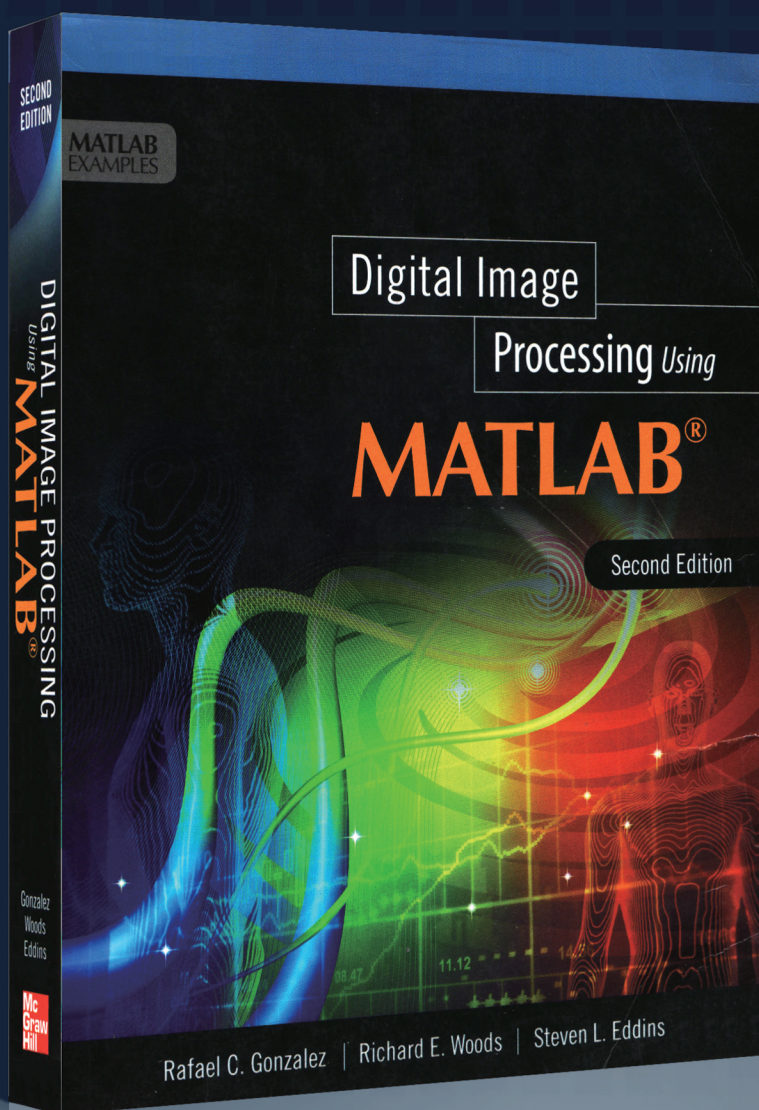


国外计算机科学经典教材

# 数字图像处理的 MATLAB实现(第2版)

[美] Rafael C. Gonzalez Richard E. Woods Steven L. Eddins 著  
阮秋琦 译



Mc  
Graw  
Hill Education

Sale or distribution of this edition is illegal outside the People's Republic of China, excluding Hong Kong, Macao SAR and Taiwan.

清华大学出版社

国外计算机科学经典教材

# 数字图像处理的 MATLAB 实现

## (第 2 版)

Rafael C. Gonzalez

[美] Richard E. Woods 著

Steven L. Eddins

阮秋琦 译

清华大学出版社

北 京

Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins  
Digital Image Processing Using MATLAB, Second Edition  
EISBN: 978-0-071-08478-9

Copyright © 2011 by The McGraw-Hill Companies, Inc.

All Rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including without limitation photocopying, recording, taping, or any database, information or retrieval system, without the prior written permission of the publisher.

This authorized Chinese translation is jointly published by McGraw-Hill Education (Asia) and Tsinghua University Press. This edition is authorized for sale in the People's Republic of China only, excluding Hong Kong, Macao SAR and Taiwan.

Copyright © 2013 by McGraw-Hill Education (Asia), a division of the Singapore Branch of The McGraw-Hill Companies, Inc. and Tsinghua University Press.

版权所有。未经出版人事先书面许可，对本出版物的任何部分不得以任何方式或途径复制或传播，包括但不限于复印、录制、录音，或通过任何数据库、信息或可检索的系统。

本授权中文简体字翻译版由麦格劳-希尔(亚洲)教育出版公司和清华大学出版社合作出版。此版本经授权仅限在中华人民共和国境内(不包括香港特别行政区、澳门特别行政区和台湾)销售。

版权©2013 由麦格劳-希尔(亚洲)教育出版公司与清华大学出版社所有。

北京市版权局著作权合同登记号 图字：01-2011-6441

本书封面贴有 McGraw-Hill 公司防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13701121933

#### 图书在版编目(CIP)数据

数字图像处理的 MATLAB 实现：第 2 版/(美)冈萨雷斯(Gonzalez, R.), (美)伍兹(Woods, R.), (美)艾丁斯(Eddins, S.) 著；阮秋琦 译. —北京：清华大学出版社，2013.4

(国外计算机科学经典教材)

书名原文：Digital Image Processing Using MATLAB, Second Edition

ISBN 978-7-302-30745-7

I. ①数… II. ①冈… ②伍… ③艾… ④阮… III. ①数字图像处理—Matlab 软件—教材  
IV. ①TN911.73

中国版本图书馆 CIP 数据核字(2012)第 284173 号

责任编辑：王 军 李维杰

装帧设计：牛静敏

责任校对：邱晓玉

责任印制：

出版发行：清华大学出版社

网 址：<http://www.tup.com.cn>, <http://www.wqbook.com>

地 址：北京清华大学学研大厦 A 座 邮 编：100084

社 总 机：010-62770175

邮 购：010-62786544

投稿与读者服务：010-62776969, [c-service@tup.tsinghua.edu.cn](mailto:c-service@tup.tsinghua.edu.cn)

质 量 反 馈：010-62772015, [zhiliang@tup.tsinghua.edu.cn](mailto:zhiliang@tup.tsinghua.edu.cn)

印 刷 者：

装 订 者：

经 销：全国新华书店

开 本：185mm×260mm

印 张：37.5 字 数：959 千字

版 次：2013 年 4 月第 1 版

印 次：2013 年 4 月第 1 次印刷

印 数：1~4000

定 价：69.80 元

---

产品编号：

# 出版说明

近年来，我国的高等教育特别是计算机学科教育，进行了一系列大的调整 and 改革，亟需一批门类齐全、具有国际先进水平的计算机经典教材，以适应我国当前计算机科学的教學需要。通过使用国外优秀的计算机科学经典教材，可以了解并吸收国际先进的教學思想和教學方法，使我国的计算机科学教育能够跟上国际计算机教育发展的步伐，从而培养出更多具有国际水准的计算机专业人才，增强我国计算机产业的核心竞争力。为此，我们从国外多家知名的出版机构 Pearson、McGraw-Hill、John Wiley & Sons、Springer、Cengage Learning 等精选、引进了这套“国外计算机科学经典教材”。

作为世界级的图书出版机构，Pearson、McGraw-Hill、John Wiley & Sons、Springer、Cengage Learning 通过与世界级的计算机教育大师携手，每年都为全球的计算机高等教育奉献大量的优秀教材。清华大学出版社和这些世界知名的出版机构长期保持着紧密友好的合作关系，这次引进的“国外计算机科学经典教材”便全是出自上述这些出版机构。同时，为了组织该套教材的出版，我们在国内聘请了一批知名的专家和教授，成立了专门的教材编审委员会。

教材编审委员会的运作从教材的选题阶段即开始启动，各位委员根据国内外高等院校计算机科学及相关专业的现有课程体系，并结合各个专业的培养方向，从上述这些出版机构出版的计算机系列教材中精心挑选针对性强的题材，以保证该套教材的优秀性和领先性，避免出现“低质重复引进”或“高质消化不良”的现象。

为了保证出版质量，我们为这套教材配备了一批经验丰富的编辑、排版、校对人员，制定了更加严格的出版流程。本套教材的译者，全部由对应专业的高校教师或拥有相关经验的 IT 专家担任。每本教材的责编在翻译伊始，就定期不间断地与该书的译者进行交流与反馈。为了尽可能地保留与发扬教材原著的精华，在经过翻译、排版和传统的三审三校之后，我们还请编审委员或相关的专家教授对文稿进行审读，以最大程度地弥补和修正在前面一系列加工过程中对教材造成的误差和瑕疵。

由于时间紧迫和受全体制作人员自身能力所限，该套教材在出版过程中很可能还存在一些遗憾，欢迎广大师生来电来信批评指正。同时，也欢迎读者朋友积极向我们推荐各类优秀的国外计算机教材，共同为我国高等院校计算机教育事业贡献力量。

清华大学出版社



# 国外计算机科学经典教材

## 编审委员会

主任委员：

孙家广      清华大学教授

副主任委员：

周立柱      清华大学教授

委员(按姓氏笔画排序)：

王成山	天津大学教授
王 珊	中国人民大学教授
冯少荣	厦门大学教授
冯全源	西南交通大学教授
刘乐善	华中科技大学教授
刘腾红	中南财经政法大学教授
吉根林	南京师范大学教授
孙吉贵	吉林大学教授
阮秋琦	北京交通大学教授
何 晨	上海交通大学教授
吴百锋	复旦大学教授
李 彤	云南大学教授
沈钧毅	西安交通大学教授
邵志清	华东理工大学教授
陈 纯	浙江大学教授
陈 钟	北京大学教授
陈道蓄	南京大学教授
周伯生	北京航空航天大学教授
孟祥旭	山东大学教授
姚淑珍	北京航空航天大学教授
徐佩霞	中国科学技术大学教授
徐晓飞	哈尔滨工业大学教授
秦小麟	南京航空航天大学教授
钱培德	苏州大学教授
曹元大	北京理工大学教授
龚声蓉	苏州大学教授
谢希仁	中国人民解放军理工大学教授

## 作者简介

### Rafael C. Gonzalez

Rafael C. Gonzalez 于 1965 年从美国迈阿密大学获得电子工程学士学位，并于 1967 年和 1970 年在美国佛罗里达大学分别获得电子工程硕士和博士学位。1970 年加盟田纳西大学(UTK)电子工程和计算机科学系。1973 年晋升为副教授，1978 年晋升为教授，1984 年成为杰出贡献教授。他从 1994 年到 1997 年任系主任。现已退休，担任田纳西大学的电子和计算机科学名誉教授。

他是田纳西大学图像和模式分析实验室、机器人和计算机视觉实验室的创始人。1982 年他还创建了 Perceptics 公司，直至 1992 年一直任董事长；1989 年 Westinghouse 股份有限公司收购了这家公司。在他的指导下，Perceptics 公司在图像处理、计算机视觉、光盘存储技术方面取得了极大成功。在刚开始的 10 年中，Perceptics 公司推出一系列创新产品，包括全球首款商用计算机视觉系统，该系统可自动读取行进中车辆的车牌；在遍布全美 6 个不同制造地点生产供美国海军使用的一系列大规模图像处理和归档系统，这种系统用于检测 Trident II 潜艇项目中导弹的火箭发动机；为先进的 Macintosh 计算机设计市场领先的图像板；拥有万亿字节的光盘生产线。

他还是模式识别、图像处理和机器学习领域企业和政府的常任顾问。他在这些领域获得的学术荣誉包括：1977 年 UTK 工学院职员成就奖、1978 年 UTK Chancellor 的研究学者奖、1980 年 Magnavox 工程教授奖以及 1980 年 M.E.Brooks 杰出教授奖。1981 年他成为田纳西大学的 IBM 教授，并于 1984 年被评为杰出贡献教授。他于 1985 年获得迈阿密大学授予的著名校友奖，1986 年获得 Phi Kappa Phi 学者奖，1992 年获得田纳西大学的 Nathan W. Dougherty 工程优秀奖。工业领域的荣誉包括：1987 年获得 IEEE 田纳西商业发展杰出工程师奖、1988 年获得 Albert Rose National 商业图像处理优秀奖、1989 年获得 B.Otto Wheeley 优秀技术传播奖、1989 年获得 Coopers 和 Lybrand 企业家年度奖、1992 年获得 IEEE 第 3 区杰出工程师奖以及 1993 年 Technology Development 的自动成像协会国家奖。

Gonzalez 博士在模式识别、图像处理和机器人领域单独撰写或与他人合作撰写了 100 多篇技术文章、两本技术书籍和 5 本教科书。他的书在遍布全球的 1000 多所大学和研究机构使用。他列入全美名人传、工程名人传、世界名人传和 10 个其他国家的国际名人传。他是两个美国专利持有者或合有者，并担任 *IEEE Transaction on Systems, Man and Cybernetics* 和《国际计算机和信息科学》杂志的副主编。他是多个专业和名誉学会的会员，包括 Tau Beta Pi、Phi Kappa Phi、Eta Kappa Nu 和 Sigma Xi。他还是 IEEE 的会士。

## Richard E. Woods

Richard E. Woods 在田纳西大学获得电子工程学士、硕士和博士学位。他有广泛的专业经历,做过企业家、传统的学术工作者、政府顾问和工业管理者。最近他创立了 MedData 交互公司,这是一家专门开发医用手持计算机系统的高科技公司。他还是 Perceptics 公司的奠基人和副总裁,在该公司,负责许多公司的定量图像分析和自动决策产品的开发。

在加盟 Perceptics 和 MedData 之前, Woods 博士担任田纳西大学电子工程和计算机科学系的助理教授,还曾任 Union Carbide 公司的计算机应用工程师。作为顾问,他参与为多个空间和军事机关(包括 NASA、弹道导弹系统指挥和 Oak Ridge 国家实验室)开发各种专用数字处理器。

Woods 博士发表或合作发表了大量有关数字信号处理方面的文章,并且是本领域引领性教科书《数字图像处理》的合著者。他是多个专业学会(包括 Tau Beta Pi、Phi Kappa Phi 和 IEEE)的会员。1986 年他被评为田纳西大学杰出工程校友。

## Steven L. Eddins

Steven L. Eddins 是 MathWorks 公司图像处理开发组的项目经理。他领导开发了该公司多个版本的图像处理工具箱。他的专业兴趣包括构建基于最新图像处理算法且广泛用于科学和工程领域的软件工具。在 1993 年加盟 MathWorks 公司之前, Eddins 博士是芝加哥伊利诺依大学电子工程和计算机科学系的教师。在那里,他为研究生和高年级学生讲授数字图像处理、计算机视觉、模式识别和滤波器设计课程,并从事图像压缩方面的研究。Eddins 博士于 1986 年在芝加哥工学院电子工程系获得学士学位,于 1990 年在该校获得博士学位,他是 IEEE 高级会员。

## 致 谢

在此衷心感谢学术界、工业界和政府部门中为本书作出贡献的多位人士，对于他们以各种方式为本书所做的重大贡献，感激之情无以言表。他们是 Mongi A. Abidi、Peter J. Acklam、Serge Beucher、Ernesto Bribiesca、Michael W. Davidson、Courtney Esposito、Naomi Fernandes、Susan L. Forsburg、Thomas R. Gest、Chris Griffin、Daniel A. Hammer、Roger Heady、Brian Johnson、Mike Karr、Lisa Kempler、Roy Lurie、Jeff Mather、Eugene McGoldrick、Ashley Mohamed、Joseph E. Pascente、David R. Pickens、Edgardo Felipe Riveron、Michael Robinson、Brett Shoelson、Loren Shure、Inpakala Simon、Jack Sklanski、Sally Stowe、Craig Watson、Greg Wolodkin 和 Mara Yale(按这些人士姓氏的首字母排序)。我们还要感谢允许在本书中使用与之相关的材料的组织。

# 前言

本书在上一版的基础上做了全面更新。像上一版一样，本书重点关注这样一个事实：在数字图像处理领域，问题求解通常需要完成大量实验工作，包括软件模拟和对大量样本图像的测试。虽然典型算法的开发是以理论知识为基础的，但这些算法的实际实现几乎总是要求参数评估，并常常做算法的修正和候选解决方案的比较。这样一来，灵活的、全面的选择和文档资料齐全的软件开发环境往往成为关键因素，软件开发环境在成本、开发时间和图像处理解决方法的可移植性上都具有重要的影响。

尽管如此重要，但意外的是很少有以教材形式编写的涉及数字图像处理的理论原理和软件实现方面的材料。2004 年撰写的本书第 1 版正好满足了这一需要。这个新版本秉承了这一宗旨，它的主要目标是为用现代软件工具实现图像处理算法提供基础。额外目标是使本书自成系统，通俗易懂，便于具有数字图像处理、数学分析及计算机编程基础知识背景的人理解和学习，所有这些基础知识在技术学科初级或高级课程中都可以找到。同时也希望读者具备 MATLAB 的初级知识。

为达到这一目的，我们觉得需要两个关键因素。首先是选择图像处理素材，也就是在数字图像处理领域、涵盖在正规课程中的有代表性的素材；其次是选择已经得到充分支持和证明，并在现实世界中得到广泛应用的软件工具。

为了满足本书的主要目的，后续各章中的多数理论概念选自 Gonzalez 和 Woods 合著的《数字图像处理》一书，该书在 30 多年中被全世界教师选为引领性的教材。所选的软件工具来自 MATLAB 数字图像处理工具箱，该工具箱在教育 and 工业应用中同样占有优势。撰写本书的基本策略是继续在成熟的理论概念和使用最新软件工具的实现技巧之间提供无缝集成。

本书内容沿用了《数字图像处理》一书的组织方式。采用这种方法，读者很容易理解本书讨论的所有数字图像处理概念，并将它们作为进一步阅读的最新参考。

遵循这种方法，使得我们能以简明扼要的方法提供理论材料，从而集中精力解决图像处理问题的软件实现。因为图像处理工作在 MATLAB 计算环境下，所以图像处理工具箱具有极大的优势，这不仅体现在计算工具的宽泛性上，而且还体现在它支持今天所用的的大多数操作系统上。这本书的鲜明特点是强调如何开发新代码以增强已有的 MATLAB 和工具箱功能。这在图像处理领域是重要特性，正如前面提到的那样，这是大量的算法开发和实验工作所需要的特性。

在介绍了 MATLAB 函数和编程基础知识后，本书接着讨论图像处理的主要方面，涵盖的内容包括灰度变换、模糊图像处理、线性和非线性空间滤波、频域滤波、图像复原和重建、几何变换和图像配准、彩色图像处理、小波、图像数据压缩、数学形态学图像处理、

图像分割、区域和边界表示及描述，还包括如何用 MATLAB 和工具箱函数解决图像处理问题的大量说明。在没有所需函数的情况下，编写新的函数和文本也作为本书教学中强调的内容。后续章节包括了120多个新函数。这些函数使图像处理工具箱的范围增加了近40%，也进一步说明了如何实现新的图像处理软件解决方案。

本书是一本教科书，并非软件手册。虽然本书自成系统，但我们还是建立了与本书配套的学习资源网站，该网站被设计用于支持很多领域(见 1.5 节)。对于学生来说，为便于跟踪正常课程学习，或者便于个别从事编程的人员自学，该网站包括背景材料的辅导和综述，以及方案和本书中所有图像的图像库。对于教师来说，该网站包含课堂上讲授的材料和书中使用的所有图像、图形的 PPT。已很熟悉图像处理和工具箱基础知识的人员可以发现该网站包含最新参考、最新实现技术以及在其他地方不容易找到的热点支持材料。所有新书选购者都有资格免费下载本书开发的所有新函数的可执行文件。

正像大多数此类书籍那样，在手稿完成以后，我们一直在努力修改。因此，我们在内容取舍方面已尽了很大努力，这些内容都是基本内容。虽然数字图像处理领域的知识体系在快速更新和完善，但本书介绍的知识不会过时，将会历久弥新。我们相信，本书的读者将从中受益，并因此发现本书在他们的工作中是适时且有用的。



# 本书配套学习资源网站

本书完全自成体系，配套学习资源网站 [www.ImageProcessingPlace.com](http://www.ImageProcessingPlace.com) 为大量重要领域提供了有力支持。

对于学生或各位读者来说，网站包括：

- 回顾 MATLAB、概率、统计、向量和矩阵等方面的知识
- 计算机项目示例
- 用于指导完成本书讨论的大多数话题的辅导章节
- 包含本书全部图像的数据库

对于教师来说，网站包括：

- PPT 形式的课堂教学材料
- 指向其他培训资源的多个链接

对于从业者来说，网站包含了其他一些专题，例如：

- 与商业网站的链接
- 挑选出的最新参考资料
- 与商业图像数据库的链接

网站包含新话题、数字图像以及本书出版后出现的其他相关材料，可使读者不断了解到最前沿、最新的内容。

本书虽然经过千锤百炼，但个别错误仍在所难免，读者可以通过浏览本书配套学习资源网站来了解勘误信息。

# 目 录

第 1 章 绪言	1
1.1 背景知识	1
1.2 什么是数字图像处理	2
1.3 MATLAB 和图像处理工具箱的背景知识	3
1.4 本书涵盖的图像处理范围	3
1.5 本书配套学习资源网站	4
1.6 符号	5
1.7 MATLAB 基础	5
1.7.1 MATLAB 桌面	5
1.7.2 使用 MATLAB 编辑器和调试器	6
1.7.3 获得帮助	6
1.7.4 保存和检索工作数据	7
1.7.5 数字图像的表示	7
1.7.6 图像的输入/输出和显示	9
1.7.7 类和图像类型	10
1.7.8 M-函数编程	12
1.8 关于本书的参考文献	24
1.9 小结	24
第 2 章 灰度变换与空间滤波	25
2.1 背景知识	25
2.2 灰度变换函数	26
2.2.1 imadjust 和 stretchlim 函数	26
2.2.2 对数及对比度扩展变换	28
2.2.3 指定任意灰度变换	29
2.2.4 针对灰度变换的某些公用 M-函数	30
2.3 直方图处理与函数绘图	35

2.3.1 生成并绘制图像的直方图	35
2.3.2 直方图均衡化	39
2.3.3 直方图匹配法(规定化)	42
2.3.4 函数 adapthisteq	45
2.4 空间滤波	46
2.4.1 线性空间滤波	47
2.4.2 非线性空间滤波	52
2.5 图像处理工具箱中标准的空间滤波器	54
2.5.1 线性空间滤波器	54
2.5.2 非线性空间滤波	58
2.6 将模糊技术用于灰度变换和空间滤波	59
2.6.1 背景知识	60
2.6.2 模糊集合介绍	60
2.6.3 使用模糊集合	63
2.6.4 一组自定义的模糊 M-函数	68
2.6.5 将模糊集合用于灰度变换	81
2.6.6 将模糊集合用于空间滤波	83
2.7 小结	87
第 3 章 频域处理	89
3.1 二维离散傅立叶变换	89
3.2 在 MATLAB 中计算及观察二维 DFT	92
3.3 频域滤波	95
3.3.1 基础知识	95
3.3.2 DFT 滤波的基本步骤	99
3.3.3 频域滤波的 M-函数	100
3.4 从空域滤波器获得频域滤波器	101

3.5	在频域中直接生成滤波器	105	4.11.2	平行射束投影和 雷登变换	156
3.5.1	建立网格数组以实现 频域滤波器	105	4.11.3	傅立叶切片定理与 滤波反投影	158
3.5.2	频域低通(平滑)滤波器	106	4.11.4	滤波器的实现	160
3.5.3	线框及表面绘制	108	4.11.5	利用扇形射束的滤波 反投影重建	161
3.6	高通(锐化)频域滤波器	111	4.11.6	函数 <code>radon</code>	161
3.6.1	高通滤波函数	112	4.11.7	函数 <code>iradon</code>	163
3.6.2	高频强调滤波	113	4.11.8	扇形射束的数据处理	166
3.7	选择性滤波	115	4.12	小结	173
3.7.1	带阻和带通滤波器	115			
3.7.2	陷波带阻和陷波带通 滤波器	117			
3.8	小结	122			
第 4 章	图像复原	123	第 5 章	几何变换与图像配准	175
4.1	图像退化/复原处理的模型	123	5.1	点变换	175
4.2	噪声模型	124	5.2	仿射变换	179
4.2.1	用 <code>imnoise</code> 函数为图像 添加噪声	124	5.3	投影变换	181
4.2.2	用给定分布产生空间 随机噪声	125	5.4	应用于图像的几何变换	182
4.2.3	周期噪声	132	5.5	MATLAB 中的图像 坐标系统	184
4.2.4	估计噪声参数	135	5.5.1	输出图像位置	186
4.3	仅有噪声的复原—— 空间滤波	139	5.5.2	控制输出网格	188
4.3.1	空间噪声滤波器	139	5.6	图像内插	190
4.3.2	自适应空间滤波器	142	5.6.1	二维内插	192
4.4	通过频域滤波减少周期噪声	144	5.6.2	内插方法的比较	193
4.5	退化函数建模	144	5.7	图像配准	194
4.6	直接逆滤波	146	5.7.1	配准处理	195
4.7	维纳滤波	147	5.7.2	使用 <code>cpselect</code> 的手工 特征选择和匹配	195
4.8	约束的最小二乘法(规则化) 滤波	149	5.7.3	使用 <code>cp2tform</code> 推断 变换参数	196
4.9	利用露西-理查德森算法的 迭代非线性复原	151	5.7.4	观察对准的图像	197
4.10	盲去卷积	154	5.7.5	基于区域的配准	199
4.11	来自投影的图像重建	155	5.7.6	基于特征的自动配准	202
4.11.1	背景	155	5.8	小结	203
			第 6 章	彩色图像处理	205
			6.1	在 MATLAB 中彩色图像的 表示	205
			6.1.1	RGB 图像	205

6.1.2 索引图像 .....	207	8.2.1 霍夫曼码 .....	289
6.1.3 处理 RGB 图像和索引 图像的函数 .....	210	8.2.2 霍夫曼编码 .....	293
6.2 彩色空间之间的转换 .....	213	8.2.3 霍夫曼译码 .....	298
6.2.1 NTSC 彩色空间 .....	213	8.3 空间冗余 .....	305
6.2.2 YCbCr 彩色空间 .....	214	8.4 不相关的信息 .....	309
6.2.3 HSV 彩色空间 .....	214	8.5 JPEG 压缩 .....	311
6.2.4 CMY 和 CMYK 彩色空间 .....	215	8.5.1 JPEG .....	312
6.2.5 HSI 彩色空间 .....	216	8.5.2 JPEG 2000 .....	317
6.2.6 独立于设备的彩色空间 .....	222	8.6 视频压缩 .....	324
6.3 彩色图像处理的基础知识 .....	229	8.6.1 MATLAB 图像序列和 电影 .....	325
6.4 彩色变换 .....	230	8.6.2 时间冗余和运动补偿 .....	327
6.5 彩色图像的空间滤波 .....	237	8.7 小结 .....	334
6.5.1 彩色图像的平滑处理 .....	237	<b>第 9 章 形态学图像处理 .....</b>	<b>335</b>
6.5.2 彩色图像的锐化处理 .....	240	9.1 预备知识 .....	335
6.6 直接在 RGB 矢量空间中 处理 .....	241	9.1.1 集合论中的基本概念 .....	335
6.6.1 使用梯度的彩色边缘检测 .....	241	9.1.2 二值图像、集合及 逻辑算子 .....	337
6.6.2 在 RGB 向量空间中 分割图像 .....	244	9.2 膨胀和腐蚀 .....	338
6.7 小结 .....	247	9.2.1 膨胀 .....	338
<b>第 7 章 小波 .....</b>	<b>249</b>	9.2.2 结构元的分解 .....	340
7.1 背景 .....	249	9.2.3 strel 函数 .....	341
7.2 快速小波变换 .....	251	9.2.4 腐蚀 .....	343
7.2.1 使用小波工具箱的 FWT .....	252	9.3 膨胀与腐蚀的结合 .....	345
7.2.2 不使用小波工具箱的 FWT .....	257	9.3.1 开操作和闭操作 .....	345
7.3 小波分解结构的处理 .....	264	9.3.2 击中或击中不中变换 .....	347
7.3.1 不使用小波工具箱编辑 小波分解系数 .....	266	9.3.3 运用查询表 .....	349
7.3.2 显示小波分解系数 .....	270	9.3.4 bwmorph 函数 .....	353
7.4 快速小波反变换 .....	274	9.4 标记连通分量 .....	355
7.5 图像处理中的小波 .....	278	9.5 形态学重建 .....	358
7.6 小结 .....	282	9.5.1 通过重建进行开操作 .....	359
<b>第 8 章 图像压缩 .....</b>	<b>283</b>	9.5.2 填充孔洞 .....	359
8.1 背景 .....	283	9.5.3 清除边界物体 .....	360
8.2 编码冗余 .....	286	9.6 灰度级形态学 .....	360
		9.6.1 膨胀和腐蚀 .....	361
		9.6.2 开操作和闭操作 .....	362
		9.6.3 重建 .....	366
		9.7 小结 .....	369

<b>第 10 章 图像分割</b> .....	<b>371</b>	<b>第 11 章 表示与描述</b> .....	<b>415</b>
10.1 点、线和边缘检测 .....	371	11.1 背景知识 .....	415
10.1.1 点检测 .....	372	11.1.1 用于提取区域及其 边界的函数 .....	416
10.1.2 线检测 .....	373	11.1.2 本章使用的 MATLAB 和 IPT 附加函数 .....	419
10.1.3 使用函数 edge 的 边缘检测 .....	374	11.1.3 一些基本的实用 M-函数 .....	420
10.2 使用霍夫变换的线检测 .....	381	11.2 表示 .....	422
10.2.1 背景 .....	381	11.2.1 链码 .....	422
10.2.2 与霍夫变换有关的 工具箱函数 .....	383	11.2.2 使用最小周长多边形的 多边形近似 .....	424
10.3 阈值处理 .....	386	11.2.3 标记 .....	430
10.3.1 基础知识 .....	386	11.2.4 边界片段 .....	431
10.3.2 基本全局阈值处理 .....	387	11.2.5 骨骼 .....	432
10.3.3 使用 Otsu's 方法的最佳 全局阈值处理 .....	388	11.3 边界描述子 .....	433
10.3.4 使用图像平滑改进全局 阈值处理 .....	391	11.3.1 一些简单的描述子 .....	433
10.3.5 使用边缘改进全局阈值 处理 .....	392	11.3.2 形状数 .....	434
10.3.6 基于局部统计的可变 阈值处理 .....	396	11.3.3 傅立叶描述子 .....	435
10.3.7 使用移动平均的图像 阈值处理 .....	398	11.3.4 统计矩 .....	438
10.4 基于区域的分割 .....	400	11.3.5 拐角 .....	439
10.4.1 基本表达式 .....	401	11.4 区域描述子 .....	445
10.4.2 区域生长 .....	401	11.4.1 函数 regionprops .....	445
10.4.3 区域分离和聚合 .....	404	11.4.2 纹理 .....	447
10.5 使用分水岭变换的分割 .....	408	11.4.3 不变矩 .....	456
10.5.1 使用距离变换的分水岭 分割 .....	409	11.5 主分量描述 .....	458
10.5.2 使用梯度的分水岭 分割 .....	410	11.6 小结 .....	466
10.5.3 控制标记符的分水岭 分割 .....	411	附录 A M-函数汇总 .....	467
10.6 小结 .....	413	附录 B ICE 和 MATLAB 的图形 用户界面 .....	485
		附录 C 附加的自定义 M-函数 .....	507
		参考文献 .....	557
		索引 .....	561

## 彩色图像处理

在这一章，我们将讨论利用图像处理工具箱的彩色图像处理的基本原理，并用开发的彩色生成和变换函数对工具箱的功能加以扩展。这一章的讨论主要面向那些已经对彩色图像处理的基本原理和术语比较熟悉的那部分读者。

### 6.1 在 MATLAB 中彩色图像的表达

正如 1.7.8 节中解释过的那样，图像处理工具箱将彩色图像当作索引图或 RGB 图像来处理。在本节我们将详细地讨论这两种类型的图像。

#### 6.1.1 RGB 图像

一幅 RGB 图像就是  $M \times N \times 3$  大小的彩色像素的数组，其中的每个彩色像素点都是在特定空间位置的彩色图像所对应的红、绿、蓝三个分量(见图 6-1)。RGB 图像也可以看做由三个灰度图像形成的“堆栈”，当发送到彩色监视器的红、绿、蓝输入端时，就在屏幕上产生彩色图像。按照惯例，形成一幅 RGB 彩色图像的三幅图像通常被称作红、绿、蓝分量图像。分量图像的数据类决定了它们的取值范围。如果一幅 RGB 图像的数据类是 double，那么取值范围就是 [0,1]。类似的，对于 uint8 类或 uint16 类的 RGB 图像，取值范围分别是 [0,255] 或 [0,65535]。用来表示这些分量图像像素值的比特数决定了一幅 RGB 图像的比特深度。例如，如果每个分量图像都是 8 比特的图像，那么对应的 RGB 图像的深度就是 24 比特。通常，所有分量图像的比特数都是相同的。在这种情况下，一幅 RGB 图像可能有的色彩数就是  $(2^b)^3$ ，其中的  $b$  就是每个分量图像的比特数。对于 8 比特图像，颜色数为 16 777 216。

令  $f_R$ 、 $f_G$  和  $f_B$  分别表示三幅 RGB 分量图像。RGB 图像就是利用 cat(连接)操作将这些分量图像组合而成的彩色图像：

```
rgb_image = cat(3, fR, fG, fB)
```

在运算中，图像按顺序放置。通常，cat(dim,A1,A2...) 沿着由 dim 指定的方式连接数组(它们必须是相同尺寸)。例如，如果 dim=1，数组就垂直安排；如果 dim=2，数组就水平安排；如果 dim=3，数组就按照三维方式堆叠，如图 6-1 所示。

如果所有的分量图像都是一样的，那么结果是一幅灰度图像。令 rgb\_image 表示一幅 RGB



图像，下面这些命令可以提取出三个分量图像：

```
>> fR = rgb_image(: , : , 1);
>> fG = rgb_image(: , : , 2);
>> fB = rgb_image(: , : , 3);
```

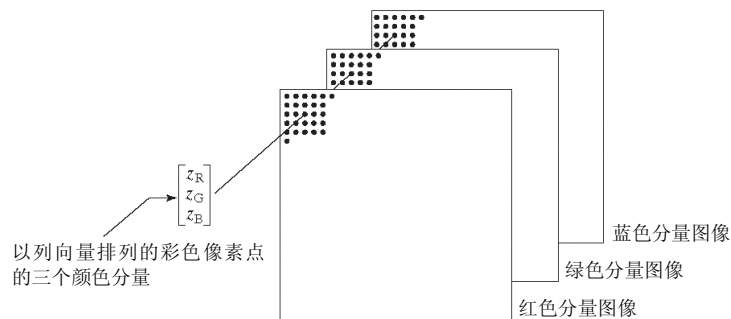


图 6-1 以相应的三个颜色分量图像形成 RGB 彩色图像的像素点

RGB 彩色空间常常用 RGB 彩色立方体加以显示，如图 6-2 所示。这个立方体的顶点是光的原色(红、绿、蓝)和二次色(青色、紫红色、黄色)。

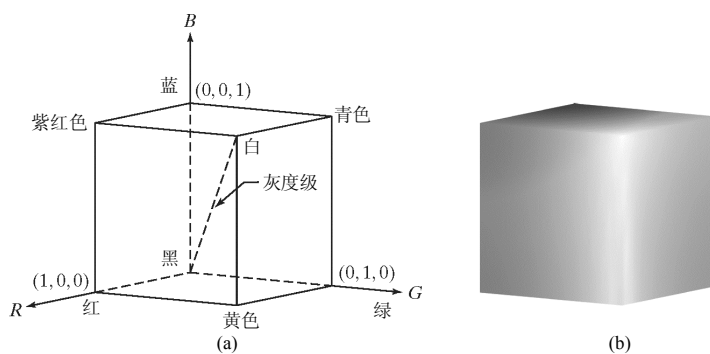


图 6-2 (a) 在顶点处显示光的原色和二次色的 RGB 彩色立方体的示意图，沿主对角线是原点的黑色到点(1,1,1)的白色的亮度值；(b) RGB 彩色立方体

为了能从任何透视方向观测这个彩色立方体，可使用自定义函数 `rgbcube`：

```
rgbcube(vx,vy,vz)
```

在提示符后键入 `rgbcube(vx,vy,vz)`，便会在 MATLAB 桌面上生成从点(vx,vy,vz)处观察到的 RGB 立方体。结果图像可用函数 `print` 存储到磁盘上。`rgbcube` 函数的代码如下：

```
function rgbcube(vx, vy, vz)
%RGBCUBE Displays an RGB cube on the MATLAB desktop.
%   RGBCUBE(VX, VY, VZ) displays an RGB color cube, viewed from point
%   (VX, VY, VZ). With no input arguments, RGBCUBE uses (10,10,4) as
%   the default viewing coordinates. To view individual color
%   planes, use the following viewing coordinates, where the first
%   color in the sequence is the closest to the viewing axis, and the
%   other colors are as seen from that axis, proceeding to the right
%   right (or above), and then moving clockwise.
%
%   -----
```

```

%          COLOR PLANE          ( vx, vy, vz)
%          -----
%          Blue-Magenta-White-Cyan      ( 0, 0, 10)
%          Red-Yellow-White-Magenta     ( 10, 0, 0)
%          Green-Cyan-White-Yellow      ( 0, 10, 0)
%          Black-Red-Magenta-Blue       ( 0, -10, 0)
%          Black-Blue-Cyan-Green        (-10, 0, 0)
%          Black-Red-Yellow-Green       ( 0, 0, -10)

% Set up parameters for function patch.
vertices_matrix = [0 0 0;0 0 1;0 1 0;0 1 1;1 0 0;1 0 1;1 1 0;1 1 1];
faces_matrix = [1 5 6 2;1 3 7 5;1 2 4 3;2 4 8 6;3 7 8 4;5 6 8 7];
colors = vertices_matrix;
% The order of the cube vertices was selected to be the same as
% the order of the (R,G,B) colors (e.g., (0,0,0) corresponds to
% black, (1, 1, 1) corresponds to white, and so on.)

% Generate RGB cube using function patch.
patch('Vertices', vertices_matrix, 'Faces', faces_matrix, ...
      'FaceVertexCData', colors, 'FaceColor', 'interp', ...
      'EdgeAlpha', 0)

% Set up viewing point.
if nargin == 0
    vx = 10; vy = 10; vz = 4;
elseif nargin ~= 3
    error('Wrong number of inputs.')
end
axis off
view([vx, vy, vz])
axis square

```

### 6.1.2 索引图像

索引图像有两个分量：整数数据矩阵  $X$  和彩色映射矩阵  $\text{map}$ 。矩阵  $\text{map}$  是  $m \times 3$  大小、由 `double` 类型且范围在  $[0,1]$  之间的浮点数构成的数组。 $\text{map}$  的长度  $m$  等于定义的颜色数。 $\text{map}$  的每一行都定义有单色的红、绿、蓝分量。索引图像将像素的亮度值“直接映射”到彩色值。每个像素的颜色由对应的整数矩阵  $X$  的值作为指向  $\text{map}$  的索引决定。如果  $X$  是 `double` 类型，那么值 1 指向  $\text{map}$  的第一行，值 2 指向第二行，等等。如果  $X$  是 `uint8` 或 `uint16` 类型，那么值 0 指向  $\text{map}$  的第一行。这些概念都会在图 6-3 中给予说明。

为显示一幅索引图像，可写为：

```
>> imshow (X ,map)
```

或者写为：

```
>> image (x)
>> colormap(map)
```

彩色映射用索引图像来存储，当使用函数 `imread` 加载图像时，索引图像将自动和图像一起被载入。

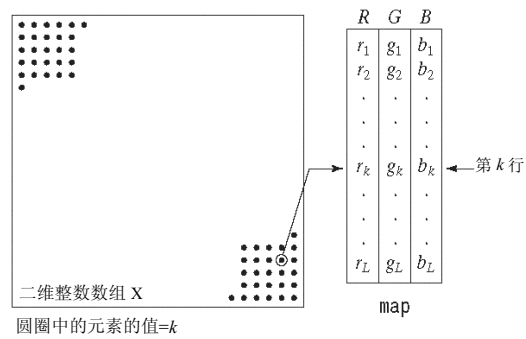


图 6-3 索引图像的元素。整数数组 X 中元素的值决定彩色映射的行数。每一行包含 RGB 三元组，L 是总行数

有时候，用较少的颜色去近似表达索引图像是有必要的。为此，我们使用函数 `imapprox`，语法如下：

```
[Y, newmap] = imapprox (X, map,n)
```

这个函数利用彩色映射 `newmap` 来返回数组 `Y`，最多有 `n` 种颜色。输入数组 `X` 的类型可以是 `uint8`、`uint16` 或 `double`。如果 `n` 小于等于 256，那么输出 `Y` 是 `uint8` 类；如果 `n` 大于 256，那么 `Y` 是 `double` 类。

当 `map` 中的行数比 `X` 中的整数值数目少时，`X` 中的多重值将在 `map` 中赋以相同的颜色。例如，假设 `X` 由 4 个等宽的垂直带组成，它们的值分别为 1、64、128 和 256。如果我们指定彩色映射 `map=[0 0 0;1 1 1]`，那么 `X` 中所有值为 1 的元素就会指向 `map` 的第一行(黑色)，其他所有的元素都将指向第二行(白色)。因而，指令 `imshow(X, map)` 的执行会显示出由一条黑色带、后面紧跟三条白色带的图像。事实上，只要 `map` 的长度是 65，这都是正确的。当是 65 时，显示一条黑色带，后面紧跟着一条灰色带，然后是两条白色带。如果 `map` 的长度超过了 `X` 中元素允许的值范围，就会得出无意义的结果图像。

指定彩色映射的办法有很多，一种方法就是利用如下语句：

```
>> map(k, :) = [r(k) g(k) b(k)];
```

其中，`[r(k) g(k) b(k)]` 是 RGB 值，指定彩色映射的一行。变化的 `k` 值可将 `map` 填满。

表 6-1 列出了一些基本颜色的 RGB 值。表中三种格式的任何一种都可以用来指定颜色。例如，用下面三条语句中的任何一条都可以把图像的背景色改成绿色：

```
>> whitebg('g');  
>> whitebg('green');  
>> whitebg([0 1 0]);
```

表 6-1 一些基本颜色的 RGB 值，可以用长名或短名(用单引号括起来)代替数字三元组，进而指定一套 RGB 颜色

长 名	短 名	RGB 值
Black	k	[0 0 0]
Blue	b	[0 0 1]
Green	g	[0 1 0]
Cyan	c	[0 1 1]

(续表)

长 名	短 名	RGB 值
Red	r	[1 0 0]
Magenta	m	[1 0 1]
Yellow	y	[1 1 0]
White	w	[1 1 1]

除了表 6-1 中的颜色外，其他颜色还包含一些小数值。例如[.5 .5 .5]是灰色、[.5 0 0]是暗红色、[.49 1 .83]是碧绿色。

MATLAB 提供了一些预定义的彩色映射，可用下面的指令来访问：

```
>> colormap(map_name)
```

上面将彩色映射设定为矩阵 map\_name。一个例子是：

```
>> colormap(copper)
```

其中，copper 是 MATLAB 彩色映射函数。在这个映射中，颜色从黑色到明亮的紫铜色平稳变化。如果显示的最后一张图是索引图像，这个指令就会将映射转成紫铜色。作为另一种选择，这个图像也可以直接用希望的彩色映射加以显示：

```
>>imshow(X, copper)
```

表 6-2 列出了 MATLAB 中可用的一些彩色映射。这些彩色映射的长度(颜色的数目)可以用加了圆括号的数字来说明，例如 gray(8)将产生 8 阶灰色的彩色映射。

表 6-2 MATLAB 中预先定义的一些彩色映射

函 数	描 述
autumn	从红色到橙色、再到黄色平缓变化
bone	对蓝色分量用较高的值进行灰度级的彩色映射 当添加“电子学方法”以观看灰度图像时，这个彩色映射很有用
colorcube	在 RGB 彩色空间中包含许多有规律放置的颜色，试图提供更多的灰度级、纯红、纯绿和纯蓝
cool	由从青到深红色调平滑变化的颜色分量组成
copper	从黑到浅铜色平缓变化
flag	由红、白、蓝和黑颜色分量组成。这个彩色映射随着每个索引增量完全改变颜色
gray	返回线性灰度级的彩色映射
hot	从黑通过红、橙、黄，再到白色平缓变化
hsv	色调-饱和度-亮度彩色模型的色调分量变化。彩色由红开始，通过黄、绿、青、蓝、深红，再回到红。这个彩色映射对于显示周期函数特别合适
jet	范围从蓝到红，并经过青、黄和橙

(续表)

函 数	描 述
lines	产生的彩色映射由 ColorOrder 属性和灰度色调决定。关于该函数的细节，可查看函数 ColorOrder 的帮助页
pink	包含粉红的大青色调。粉红彩色映射提供灰度照片的棕色色调
prism	重复 6 种颜色：红、橙、黄、绿、蓝和紫
spring	由深红和黄色色调组成
summer	由绿色和黄色色调组成
winter	由蓝色和绿色色调组成
white	这是全白单色颜色映射

6.1.3 处理 RGB 图像和索引图像的函数

表 6-3 列出了在 RGB 图像、索引图像和灰度图像之间进行转换的工具箱函数。为了明确本节中使用的符号，我们使用 rgb\_image 来表示 RGB 图像，用 gray\_image 来表示灰度图像，用 bw 来表示黑白(二值)图像，用 X 来表示索引图像的数据矩阵分量。回想前面，索引图像由整数数据矩阵和彩色映射矩阵组成。

函数 dither 可以用于灰度图像和彩色图像。“抖动”是印刷和出版行业常用的一种处理手段，在由点组成的印刷页上给出色调变化的直观印象。对于灰度图像，抖动调色试图使用在白色背景上产生黑点的二值图像来得到灰色色调(反之亦然)。

点的大小可变，从明亮区域的小点到黑暗区域中逐渐增大的较大点。执行“抖动”算法的关键问题是要折中考虑视觉感受的精确性和计算的复杂度。在工具箱中应用的“抖动”方法基于佛罗伊德-斯坦伯格(Floyd-Steinberg)算法(Floyd 和 Steinberg[1975])和阿里坎尼(Ulichney)算法(Ulichney[1987])。用于灰度图像的函数 dither 的语法是：

```
bw = dither(gray_image)
```

其中，正如前面注释过的那样，gray\_image 是灰度图像，bw 是抖动处理过的二值图像(逻辑类)。

表 6-3 用于在 RGB 图像、索引图像和灰度图像之间进行转换的工具箱函数

函 数	描 述
Dither	采用“抖动”方法从 RGB 图像创建索引图像
grayslice	从灰度图像通过阈值处理创建索引图像
gray2ind	从灰度图像创建索引图像
ind2gray	从索引图像创建灰度图像
rgb2ind	从 RGB 图像创建索引图像
ind2rgb	从索引图像创建 RGB 图像
rgb2gray	从 RGB 图像创建灰度图像

当处理彩色图像时,“抖动”主要是与函数 `rgb2ind` 联合起来以减少图像中的颜色数目。这个函数将在本节的后面讨论。

函数 `grayslice` 有如下语法:

```
X = grayslice(gray_image, n)
```

这个函数使用阈值对灰度图像进行阈值处理以产生索引图像:

$$\frac{1}{n}, \frac{2}{n}, \dots, \frac{n-1}{n}$$

正如前面注释过的那样,索引图像可以看做用指令 `imshow(X,map)` 通过长度适当的映射(例如 `jet(16)`)得到的。另一种语法是:

```
X = grayslice(gray_image, v)
```

其中, `v` 是矢量(值的范围为[0,1]),用来给 `gray_image` 赋阈值。函数 `grayslice` 是伪彩色图像处理的基本工具,伪彩色为指定的灰度区域赋予不同的颜色。输入图像的类型可以是 `uint8`、`uint16` 或 `double`。即使输入图像的类型是 `uint8` 或 `uint16`, `v` 的阈值也必须在 [0,1] 之间。函数执行了必要的缩放操作。

函数 `gray2ind` 采用如下语法:

```
[X, map] = gray2ind(gray_image, n)
```

执行缩放后,围绕图像 `gray_image`,使用彩色映射 `gray(n)` 生成索引图像 `X`。如果 `n` 省略,那么默认值为 64。输入图像的类型可以是 `uint8`、`uint16` 或 `double`。如果 `n` 的值小于等于 256,那么输出图像 `X` 的类型是 `uint8`;如果 `n` 的值大于 256,那么类型是 `uint16`。

函数 `ind2gray` 的语法为:

```
gray_image = ind2gray(X,map)
```

这个函数把由 `X` 和 `map` 构成的索引图像转换成灰度图像。数组 `X` 的类型可以是 `uint8`、`uint16` 或 `double`。输出图像是 `double` 类。

本章比较感兴趣的函数是 `rgb2ind`,语法如下:

```
[X, map] = rgb2ind(rgb_image, n, dither_option)
```

在这里,`n` 决定 `map` 的颜色数目,`dither_option` 可以是如下两个值之一: 'dither'(默认值),如有必要,以损失空间分辨率为代价,从而达到更好的颜色分辨率;相反, 'nodither' 将原图上的每个颜色用与之最接近的颜色映射到新图上(取决于 `n` 的值),不执行抖动。输入图像可以是 `uint8`、`uint16` 或 `double` 类。如果 `n` 的值小于等于 256,输出数组 `X` 是 `uint8` 类,否则便是 `uint16` 类。例 6.1 显示了没有彩色减少的抖动效果。

函数 `ind2rgb` 的语法为:

```
rgb_image = ind2rgb(X, map)
```

这个函数将矩阵 `X` 和对应的彩色映射 `map` 转换成 RGB 格式。`X` 可以是 `uint8`、`uint16` 或 `double` 类。输出的 RGB 图像是大小为 `M×N×3` 的 `double` 类的数组。



最后，函数 `rgb2gray` 的语法为：

```
gray_image = rgb2gray (rgb_image)
```

这个函数将 RGB 图像转换成灰度图像。输入的 RGB 图像可以是 `uint8`、`uint16` 或 `double` 类，输出图像与输入图像的类相同。

#### 例 6.1 对表 6-3 中某些函数的用法展示

函数 `rgb2ind` 对于减少 RGB 图像中的色彩数目很有用，正如这个函数说明的那样，关于使用抖动选项的优点，可考虑图 6-4(a)，这是一幅 24 比特的 RGB 图像 `f`。

图 6-4(b)和图 6-4(c)显示了应用如下指令后的结果：

```
>> [X1, map1] = rgb2ind(f, 8, 'nodither');
>> imshow(X1, map1)
```

和

```
>> [X2, map2] = rgb2ind(f, 8, 'dither');
>> figure, imshow(X2, map2)
```

两幅图像均只有 8 种颜色，这相对 `uint8` 图像 `f` 可能拥有的 1600 万种颜色来说有显著减少。图 6-4(b)显示了明显的伪轮廓，特别是在大花朵的中心部位。抖动处理之后的图像显示出了较好的色调，而且伪轮廓明显减少，这就是用抖动引入的“随机性”的结果。图像有一点点模糊，但视觉上的确优于图 6-4(b)。

抖动处理的效果用灰度图像来说明往往更好。图 6-4(d)和图 6-4(e)是由下列指令获得的：

```
>> g = rgb2gray(f)
>> g1 = dither(g);
>> figure, imshow(g); figure, imshow(g1)
```

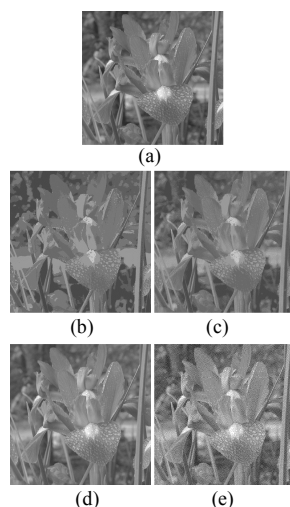


图 6-4 (a) RGB 图像；(b) 没有进行抖动处理的彩色数目减少到 8 后的图像；(c) 经抖动处理的彩色数目减少到 8 后的图像；(d) 用函数 `rgb2gray` 得到(a)的灰度图像；(e) 经抖动处理后的灰度图像(这是二值图像)

图 6-4(e)是二值图像，这再一次说明了数据显著减少的程度。图 6-4(c)和图 6-4(e)展示了“抖

动”之所以成为印刷和出版行业主要方法的原因，特别是在纸张质量和印刷分辨率不高的情况下(比如报纸的印刷)。

## 6.2 彩色空间之间的转换

正如在前面说明过的那样，在一幅 RGB 图像中，工具箱直接把颜色描述成 RGB 值，或者在索引图像中间接地用 RGB 格式来存储彩色映射。

然而，还有其他的彩色空间(又被称作彩色模型)，它们在实际中的应用可能比 RGB 更方便或更恰当。这些模型是 RGB 模型的变换，包括 NTSC、YCbCr、HSV、CMY、CMYK 和 HSI 彩色空间。工具箱提供了从 RGB 向 NTSC、YCbCr、HSV、CMY 彩色空间转换或转换回来的函数。用于转换成 HSI 彩色空间并转换回来的函数在本节后面讨论。

### 6.2.1 NTSC 彩色空间

NTSC 彩色空间用于模拟电视。这种格式的主要优势是灰度信息和彩色数据是分离开来的，所以同一信号可以用于彩色电视机和黑白电视机。在 NTSC 格式中，图像数据由三部分组成：亮度(Y)、色调(I)和饱和度(Q)。在这里，字母 YIQ 的选择常常是按照惯例进行的。亮度分量描述灰度信息，其他两个分量携带电视信号的彩色信息。YIQ 分量都是用线性变换从一幅图像的 RGB 分量得到的：

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.229 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

注意，第一行的各元素之和为 1，而下两行元素的和为 0。这和预想的一样，因为对于一幅灰度图像，所有的 RGB 分量都是相等的；所以对于这样的图像来说，I 和 Q 分量应该是 0。函数 `rgb2ntsc` 可执行前边的变换：

```
yiql_image = rgb2ntsc(rgb_image)
```

其中，输入的 RGB 图像可以是 `uint8`、`uint16` 或 `double` 类。输出图像是大小为  $M \times N \times 3$  的 `double` 类数组。分量图像 `yiql_image(:, :, 1)` 是亮度、`yiql_image(:, :, 2)` 是色调、`yiql_image(:, :, 3)` 代表饱和度。

类似的，RGB 分量可以利用下面的线性变换从 YIQ 分量得到：

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

工具箱函数 `ntsc2rgb` 执行这个变换，语法是：

```
rgb_image = ntsc2rgb(yiql_image)
```

输入和输出图像都是 `double` 类。

## 6.2.2 YCbCr 彩色空间

YCbCr 彩色空间广泛用于数字视频。在这种格式中,亮度信息用单独的分量  $Y$  来表示,彩色信息是用两个色差分量  $Cb$  和  $Cr$  来存储的。分量  $Cb$  是蓝色分量与参考值的差,分量  $Cr$  是红色分量与参考值的差。工具箱采用的从 RGB 转换为 YcbCr 的变换是:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 65.481 & 128.553 & 24.966 \\ -37.797 & -74.203 & 112.000 \\ 112.000 & -93.786 & -18.214 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

转换函数是:

```
ycbcr_image = rgb2ycbcr(rgb_image)
```

输入的 RGB 图像可以是 uint8、uint16 或 double 类。输出图像和输入图像的类型相同。使用类似的变换可以从 YCbCr 转换回 RGB:

```
rgb_image = ycbcr2rgb(ycbcr_image)
```

输入的 YCbCr 图像可以是 uint8、uint16 或 double 类。输出图像和输入图像的类型相同。

## 6.2.3 HSV 彩色空间

HSV(色调、饱和度、值)是人们用来从颜色轮或调色板中挑选颜色(例如颜料或墨水)时使用的彩色系统之一,值得考虑的是,这个颜色系统比 RGB 系统更接近人们的经验和对彩色的感知。在画家的术语里,色调、饱和度和数值被称作色调、明暗和色值。

HSV 彩色空间可以通过从 RGB 彩色立方体沿灰度轴(连接黑色顶点和白色顶点的轴)用公式来表达,从而得出图 6-5(a)所示的六边形表示的彩色调色板。当我们沿着图 6-5(b)中的垂直轴(灰)轴移动时,这个与轴垂直的六边形平面的大小是变化的,并产生了图中描述的量。

色调分量是用围绕彩色六边形的角度来描述的,典型的是将红轴作为参考( $0^\circ$ )轴。值分量是沿着这个圆锥体的轴度量的。

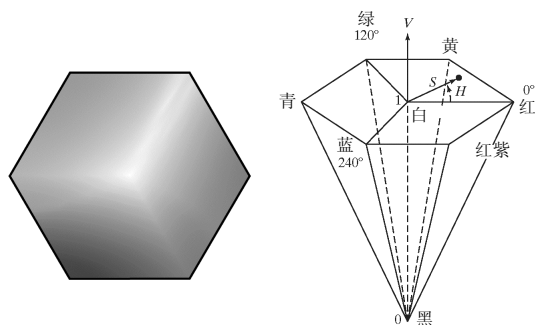


图 6-5 (a) HSV 彩色六边形; (b) HSV 六面锥体

$V=0$ , 轴的末端为黑色;  $V=1$ , 轴的末端为白色。轴位于图 6-5(a)中全彩色六边形的中心。这样,这个轴就描绘了所有灰度级的深浅。饱和度(颜色的纯净度)分量是由距  $V$  轴的距离来度量的。

HSV 彩色系统是以圆柱坐标系为基础的。将 RGB 转换为 HSV 需要将公式展开，将 RGB 值(笛卡尔坐标系)映射至圆柱坐标系。大多数的计算机图形学文献都对这一问题给出了详细论述，我们在这里就不再展开这个公式了。

将 RGB 转换为 HSV 的 MATLAB 函数是 `rgb2hsv`，语法如下：

```
hsv_image = rgb2hsv(rgb_image)
```

输入的 RGB 图像可以是 `uint8`、`uint16` 或 `double` 类，输出图像是 `double` 类。将 HSV 转换回 RGB 的函数为 `hsv2rgb`，语法如下：

```
rgb_image = hsv2rgb(hsv_image)
```

输入图像的类必须是 `double` 类，输出也是 `double` 类。

### 6.2.4 CMY 和 CMYK 彩色空间

青色、紫红色和黄色是光的二次色，或者换一种说法，它们是颜料的原色。例如，当在表面涂上青色颜料，再用白光照射时，没有红光从表面反射。也就是说，青色颜料从表面反射的光中减去了红光。

大多数将颜料堆积于纸上的设备，比如彩色打印机和复印机，都需要 CMY 数据输入，或在内部将 RGB 转换为 CMY，近似的转换可用下面的公式实现：

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

其中，假想所有的颜色值都已经归一化在[0,1]之间。这个公式证明了刚才的描述，从涂满纯青色的表面反射的光不包含红色(公式中的  $C = 1 - R$ )。同样，纯净的紫红色不反射绿色，纯净的黄色不反射蓝色。上述公式还证明，从 1 减去个别的 CMY 值，可以从一组 CMY 值很容易地获得 RGB 值。

在理论上，等量的颜料原色，将青色、紫红色和黄色混合会产生黑色。在实践中，将这些颜色混合印刷会生成模糊不清的黑色。所以，为了生成纯正的黑色(打印中主要的颜色)，第 4 种颜色——黑色便添加进来了，从而给出提升的 CMYK 彩色模型。由此，当出版人谈论“4 色印刷”时，他们指的是 CMY 彩色模型再加黑色。

在 2.2.1 节介绍的 `imcomplement` 函数可近似地把 RGB 模型转换为 CMY 模型：

```
cmy_image = imcomplement(rgb_image)
```

也可以用这个函数将 CMY 图像转换为 RGB 图像：

```
rgb_image = imcomplement(cmy_image)
```

高质量的 CMY 或 CMYK 转换需要掌握打印机墨水和介质的知识,以及用于决定什么地方用黑墨水(K)代替其他三种墨水的启发式方法。这种转换可使用为特殊打印机创建的 ICC 彩色剖面完成(关于 ICC, 见 6.2.6 节)。

### 6.2.5 HSI 彩色空间

除了 HSV 之外,至今为止,我们讨论过的彩色空间还没有根据人的解释适合以术语描述彩色的方式。例如,没有人会用给出构成颜色的每一种颜料原色的百分比这种方法来提供汽车的颜色。

当人们观察彩色物体时,往往倾向于使用色调、饱和度和亮度来描述。色调是描述纯色的颜色属性,而饱和度给出了纯色被白光冲淡程度的度量。亮度是主观描述符,事实上几乎无法度量。这使得无色的亮度概念得以具体化,并且是描述颜色感觉的关键因素之一。我们已经知道,亮度(灰度)是描述单色图像的最有用描述子。这个明确的分量可以被度量,也可以被容易地描述出来。

我们将要讨论的彩色空间叫做 HSI(色彩、饱和度、强度),该模型将强度分量与从一幅彩色图像中承载的彩色信息分开。正如结果那样,HSI 模型是开发基于彩色描述的图像处理算法的理想工具。对于人来说,这种描述自然而直观,毕竟人是这些算法的开发者和使用者。HSV 彩色空间有些类似,当按照画家的调色板来解释时,HSV 关注的是所能呈现的有意义的色彩。

正如在 6.1.1 节中讨论过的那样,RGB 彩色图像是由三个单色的亮度图像构成的,所以,我们可以从一幅 RGB 图像中提取出亮度并不奇怪。如果采用图 6-2 中的彩色立方体,假设我们站在黑色顶点(0,0,0)处,那么正上方正对是白色顶点(1,1,1),如图 6-6(a)所示。再同图 6-2 联系起来看,亮度是沿着连接两个顶点的连线分布的。在图 6-6 中,这条连接黑色和白色顶点的线(亮度轴)是垂直的。因此,如果想确定图 6-6 中任意彩色点的亮度分量,就需要经过包含彩色点且垂直于亮度轴的平面。这个平面和亮度轴的交点将给出范围在[0,1]之间的亮度值。我们还注意到,饱和度随着与亮度轴之间的距离函数而增加。事实上,在亮度轴上的点的饱和度为 0,事实很明显,沿这个轴上的所有点都是灰度色调。

为了弄清从已经给出的 RGB 点怎样决定色调,考虑图 6-6(b),其中显示了由三个点(分别为黑色、白色和青色)定义的平面。在这个平面上,含有黑色和白色顶点的这个事实告诉我们:亮度轴同样在这个平面上。此外我们看到,由亮度轴和立方体边界共同定义的、这个平面上包含的所有点都有相同的色调(在此例中为青色)。这是因为在彩色三角形内,颜色是这三个顶点颜色的各种组合或者是由它们混合而成的。如果这些顶点中的两个是黑色和白色,第三个顶点是彩色的点,那么这个三角形中所有点的色调都是相同的,因为白色和黑色分量对于色彩的变化没有影响(当然在这个三角形中,点的亮度和饱和度会有变化)。以垂直的亮度轴旋转这个深浅平面,我们可以获得不同的色调。从这些概念中,我们得到下面的结论:形成 HSI 空间所需的色调、饱和度和亮度值,可以通过 RGB 彩色立方体得到。也就是说,通过算出刚刚在前边推出的几何推理公式,就可以将任意的 RGB 点转换成 HSI 模型中对应的点。

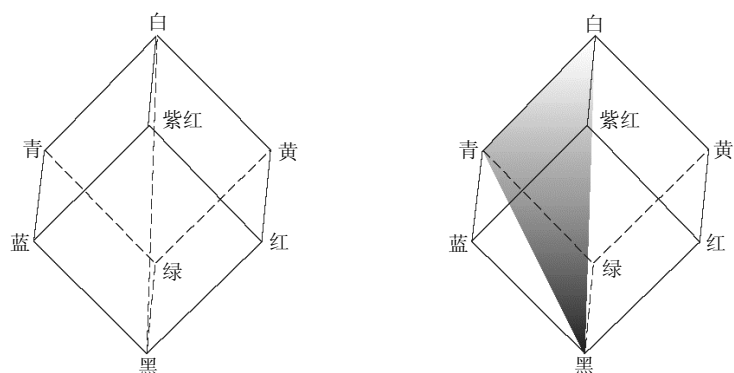


图 6-6 RGB 与 HSI 彩色模型之间的关系

基于前面的讨论,我们认识到,HSI 空间由垂直的亮度轴以及垂直于此轴的某个平面上彩色点的轨迹组成。当平面沿着垂直轴上下移动时,由平面和立方体表面相交定义的边界为三角形或六边形。如果从立方体的灰度轴向下看去,如图6-7(a)所示,这可能变得更加直观。在这个平面上,我们看到各原色之间都相隔了  $120^\circ$ ,各二次色和各原色之间相隔了  $60^\circ$ ,这意味着各二次色之间也相隔了  $120^\circ$ 。

图6-7(b)显示了六边形和某个任意的彩色点(用点的形式显示)。这个点的色调由某个参考点的夹角决定。通常(但也并不总是),距红轴  $0^\circ$  的角,表示色调为 0,并且色调从此点逆时针增长。饱和度(到垂直轴的距离)就是从原点到此点的矢量的长度。注意,这个原点由色彩平面和垂直亮度轴的交点决定。HSI 色彩空间的重要组成部分是:垂直亮度轴到彩色点的矢量长度,以及该矢量与红轴的夹角。因此,当用刚才讨论过的六边形,甚至如图 6-7(c)和图 6-7(d)所示的三角形或圆形定义 HSI 平面是不足为奇的。选择哪个形状并不重要,因为任意形状都可以通过几何变换变换成另外两个。

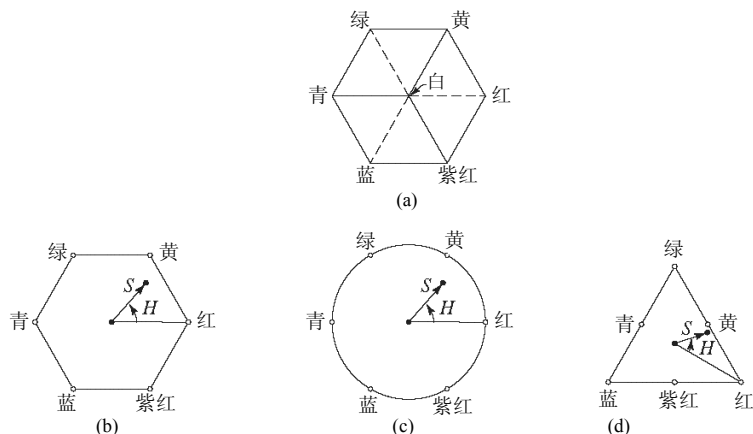


图 6-7 HIS 颜色模型中的色调和饱和度。点是任意的彩色点,与红轴的夹角给出了色调,向量的长度是饱和度。在这些平面上,所有彩色的亮度都由垂直亮度轴上平面的位置给出

图 6-8 显示了基于彩色三角形和圆形的 HSI 模型。



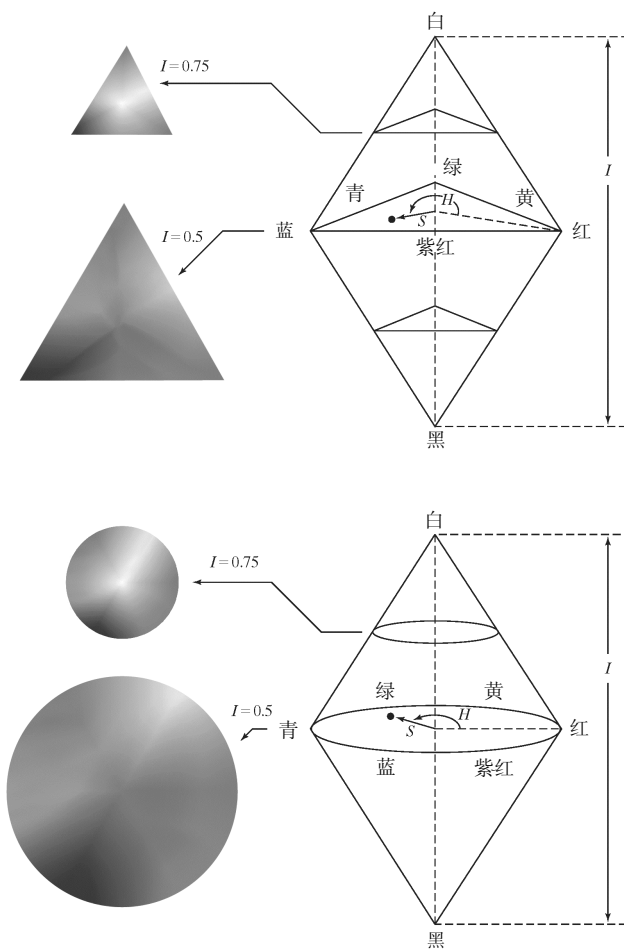


图 6-8 基于三角形和圆形平面的 HSI 模型，三角形和圆形都垂直于亮度轴

### 1. 将颜色从 RGB 转换为 HSI

在下面的讨论中，我们给出从 RGB 到 HSI 必要的转换公式，这里不予推导。对公式的详细推导可参考本书网站(1.5 节列出了地址)。给出一幅 RGB 彩色格式的图像，那么每个 RGB 像素的  $H$  分量可用下面的公式得到：

$$H = \begin{cases} \theta & B \leq G \\ 360 - \theta & B > G \end{cases}$$

其中：

$$\theta = \cos^{-1} \left\{ \frac{0.5[(R-G) + (R-B)]}{[(R-G)^2 + (R-B)(G-B)]^{1/2}} \right\}$$

饱和度由下面的式子给出：

$$S = 1 - \frac{3}{(R+G+B)} [\min(R, G, B)]$$

最后，亮度由下面的式子给出：

$$I = \frac{1}{3}(R + G + B)$$

假定 RGB 值已经归一化在[0,1]之间，角度  $\theta$  使用关于 HSI 空间的红轴来度量，正如图 6-7 中指出的那样。将从  $H$  的公式中得出的所有结果除以  $360^\circ$ ，即可将色调归一化在[0,1]之间。如果给出的 RGB 值在[0,1]之间，那么其他的两个 HSI 分量就已经在[0,1]之间了。

## 2. 将颜色从 HSI 转换为 RGB

给定在[0,1]之间的 HSI 值，我们现在希望找出同一范围内相应的 RGB 值。可用的公式依赖于  $H$  的值。有三个感兴趣的部分，正如早些时候提到的那样，分别对应原色之间相隔  $120^\circ$  的范围。我们用  $360^\circ$  乘以  $H$ ，这样就将色调的值还原成了原来的范围—— $[0^\circ, 360^\circ]$ 。

**RG 区域** ( $0^\circ \leq H < 120^\circ$ ) 如果  $H$  在这个区域内，那么 RGB 分量由下式给出：

$$R = I \left[ 1 + \frac{S \cos H}{\cos(60^\circ - H)} \right]$$

$$G = 3I - (R + B)$$

和

$$B = I(1 - S)$$

**GB 区域** ( $120^\circ \leq H < 240^\circ$ ) 如果给出的  $H$  值在这个区域内，我们就先从中减去  $120^\circ$ ：

$$H = H - 120^\circ$$

那么，这时 RGB 分量是：

$$R = I(1 - S)$$

$$G = I \left[ 1 + \frac{S \cos H}{\cos(60^\circ - H)} \right]$$

并且

$$B = 3I - (R + G)$$

**BR 区域** ( $240^\circ \leq H \leq 360^\circ$ ) 最后，如果  $H$  在这个区域内，我们就从中减去  $240^\circ$ ：

$$H = H - 240^\circ$$

RGB 分量分别是：

$$R = 3I - (G + B)$$

其中：

$$G = I(1 - S)$$

和

$$B = I \left[ 1 + \frac{S \cos H}{\cos(60^\circ - H)} \right]$$

我们将在 6.5.1 节说明这些公式在图像处理中的应用。

### 3. 用于从 RGB 到 HIS 进行转换的 M-函数

下面是惯用的函数：

```
hsi = rgb2hsi(rgb)
```

该函数执行刚刚讨论过的把 RGB 转换成 HSI 格式的公式。其中, rgb 和 hsi 分别表示 RGB 和 HSI 图像。下面的代码演示了这个函数的使用：

```
function hsi = rgb2hsi(rgb)
%RGB2HSI Converts an RGB image to HSI.
%   HSI = RGB2HSI(RGB) converts an RGB image to HSI. The input image
%   is assumed to be of size M-by-N-by-3, where the third dimension
%   accounts for three image planes: red, green, and blue, in that
%   order. If all RGB component images are equal, the HSI conversion
%   is undefined. The input image can be of class double (with
%   values in the range [0, 1]), uint8, or uint16.
%
%   The output image, HSI, is of class double, where:
%       HSI(:, :, 1) = hue image normalized to the range [0,1] by
%                       dividing all angle values by 2*pi.
%       HSI(:, :, 2) = saturation image, in the range [0, 1].
%       HSI(:, :, 3) = intensity image, in the range [0, 1].
%   Extract the individual component images.
rgb = im2double(rgb);
r = rgb(:, :, 1);
g = rgb(:, :, 2);
b = rgb(:, :, 3);

% Implement the conversion equations.
num = 0.5*((r - g) + (r - b));
den = sqrt((r - g).^2 + (r - b).*(g - b));
theta = acos(num./(den + eps));

H = theta;
H(b > g) = 2*pi - H(b > g);
H = H/(2*pi);

num = min(min(r, g), b);
den = r + g + b;
den(den == 0) = eps;
S = 1 - 3.* num./den;
H(S == 0) = 0;
I = (r + g + b)/3;

% Combine all three results into an hsi image.
hsi = cat(3, H, S, I);
```

#### 4. 用于从 HSI 到 RGB 进行转换的 M-函数

下边的函数：

```
rgb = hsi2rgb(hsi)
```

执行把 HSI 转换成 RGB 格式的公式。下面的代码演示了这个函数的使用：

```
function rgb = hsi2rgb(hsi)
%HSI2RGB Converts an HSI image to RGB.
%   RGB = HSI2RGB(HSI) converts an HSI image RGB, where HSI is
%   assumed to be of class double with:
%       HSI(:, :, 1) = hue image, assumed to be in the range
%       [0, 1] by having been divided by 2*pi.
%       HSI(:, :, 2) = saturation image, in the range [0, 1];
%       HSI(:, :, 3) = intensity image, in the range [0, 1].
%
%   The components of the output image are:
%       RGB(:, :, 1) = red.
%       RGB(:, :, 2) = green.
%       RGB(:, :, 3) = blue.

% Extract the individual HSI component images.
H = hsi(:, :, 1) * 2 * pi;
S = hsi(:, :, 2);
I = hsi(:, :, 3);

% Implement the conversion equations.
R = zeros(size(hsi, 1), size(hsi, 2));
G = zeros(size(hsi, 1), size(hsi, 2));
B = zeros(size(hsi, 1), size(hsi, 2));

% RG sector (0 <= H < 2*pi/3).
idx = find( (0 <= H) & (H < 2*pi/3));
B(idx) = I(idx) .* (1 - S(idx));
R(idx) = I(idx) .* (1 + S(idx) .* cos(H(idx))./ ...
                    cos(pi/3 - H(idx)));
G(idx) = 3*I(idx) - (R(idx) + B(idx));

% BG sector (2*pi/3 <= H < 4*pi/3).
idx = find( (2*pi/3 <= H) & (H < 4*pi/3) );
R(idx) = I(idx) .* (1 - S(idx));
G(idx) = I(idx) .* (1 + S(idx) .* cos(H(idx) - 2*pi/3) ./ ...
                    cos(pi - H(idx)));
B(idx) = 3*I(idx) - (R(idx) + G(idx));

% BR sector.
idx = find( (4*pi/3 <= H) & (H <= 2*pi));
G(idx) = I(idx) .* (1 - S(idx));
B(idx) = I(idx) .* (1 + S(idx) .* cos(H(idx) - 4*pi/3) ./ ...
                    cos(5*pi/3 - H(idx)));
R(idx) = 3*I(idx) - (G(idx) + B(idx));

% Combine all three results into an RGB image. Clip to [0, 1] to
% compensate for floating-point arithmetic rounding effects.
rgb = cat(3, R, G, B);
rgb = max(min(rgb, 1), 0);
```

### 例 6.2 从 RGB 转换为 HSI

图 6-9 显示了在白色背景下一幅 RGB 立方体图像的色调、饱和度和亮度分量,这类似于图 6-2(b)中的图像。图 6-9(a)是色调图像,它的最大特征是:在立方体的前(红)平面中沿  $45^\circ$  线是不连续的。为理解不连续的原因,可参考图 6-2(b),从立方体的红顶点到白顶点画一条线,并在这条线的中间选择一个点。从这个点开始向右画一条路径,沿着立方体周围直到返回起始点。在这一路径上遇到的彩色是黄、绿、青、蓝、深红、红。根据图 6-7,沿着这条路径的色调的值应该从  $0^\circ$  到  $360^\circ$  (也就是说,从色调可能的最低值到最高值)增加。这正好是图 6-9 中显示的,因为在图中,最低值表示黑,最高值表示白。

图 6-9(b)中的饱和度图像显示了较暗的值逐渐逼近 RGB 立方体的白色顶点,指示彩色饱和度越来越弱,直至白色。最后,在图 6-9(c)中显示了图像中的每个像素都是相应于图 6-2(b)中 RGB 像素值的平均值。注意,这幅图像的背景是白色,因为在彩色图像中,背景的亮度是白色。其他两幅图的背景是黑色的,因为白色的色调和饱和度是 0。

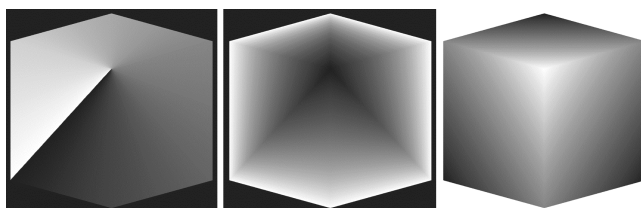


图 6-9 RGB 彩色立方体图像的 HSI 分量图像: (a) 色调图像; (b) 饱和度图像; (c) 亮度图像

## 6.2.6 独立于设备的彩色空间

从 6.2.1 到 6.2.5 节,我们关注的是彩色空间,彩色空间以使计算更方便的方法描述了彩色信息,或以对特殊应用更直观或更合适的方法描述彩色。迄今为止讨论的所有空间都与设备相关。例如,RGB 彩色的外观随着显示器和扫描仪的特性而变化,CMYK 彩色随打印墨水和纸张特性而变化。

在这一节,我们集中讨论独立于设备的彩色空间。在彩色图像系统中,为达到一致的、高质量的彩色重现,需要对系统中的每个彩色设备进行理解和描述。在可控的环境中,有可能调整系统的各种部件以达到满意的结果。例如,在图片打印商店中,手工优化颜料、洗印以及洗印子系统,进而达到一致结果是可能的。另一方面,这个方法在开放的数字成像系统中却不实用,数字成像系统由许多设备组成,其中被处理和观察的图像是不可控的(例如互联网)。

### 1. 背景

通常,在辨别不同颜色时使用的特征是亮度、色调和饱和度。正如在本节早些时候指出的那样,亮度具体表达无色的强度的概念,色调是在光波混合中与主要波长相关的属性,色调描述了观察者感知的主要彩色。这样一来,当我们称物体为红色、橘黄色和黄色时,指的是色调。饱和度指的是相对纯度或与色调相混合的白光数量。纯色是全饱和的彩色,比如粉红色(红和白)和淡紫色(紫和白),有较小的饱和度,饱和度与所加白光的数量成反比。色调和饱和度合起来称为颜色,因此,彩色是由颜色和亮度表现的特性。为形成一种特殊颜色而需要的红、绿、蓝颜色叫做三色值,分别用 X、Y、Z 来表示。颜色是由颜色自身的三色系数来指定的,定义为:

$$x = \frac{X}{X+Y+Z}$$

$$y = \frac{Y}{X+Y+Z}$$

$$z = \frac{Z}{X+Y+Z} = 1 - x - y$$

并且:

$$x + y + z = 1$$

其中,  $x$ 、 $y$  和  $z$  分别表示红、绿、蓝分量。在可见光谱中, 光的任何波长均可产生与那个波长相对应的三色值, 它们可直接从曲线或表中得到, 曲线和表已从广泛的实验结果编制出来了(Poynton[1996])。

用得最广泛的独立于设备的三色值彩色空间之一是 1931 年提出的 CIE XYZ 彩色空间。在 CIE XYZ 彩色空间中,  $Y$  被选为特定亮度的度量。由  $Y$  以及颜色值  $x$  和  $y$  定义的彩色空间被称为  $xyY$  彩色空间。 $X$  和  $Z$  三色值可通过  $x$ 、 $y$  和  $Y$  值用下式计算出来:

$$X = \frac{Y}{y}x$$

和

$$Z = \frac{Y}{y}(1 - x - y)$$

可以从前边的公式看出, 在 XYZ 和  $xyY$  CIE 彩色空间之间存在直接对应关系。以  $x$  和  $y$  为参数的函数显示人能感觉到的颜色范围的图(图 6-10)被称为色度图。在色度图中, 对于  $x$  和  $y$  的任何值, 相应的  $z$  值是  $z=1-(x+y)$ 。例如在图 6-10 中, 标注为绿色的点拥有近似 62% 的绿和近似 25% 的红。因此, 针对那种颜色的光的蓝分量是 13%。

各种单(纯谱)彩色的位置——从 380nm 的紫色到 780nm 的红色, 均围绕着色度图的舌形线的边界指示出来了。边界的直线部分称为紫色线, 这些颜色没有单一的等价值。边界上的任何点都不是实际的颜色, 但是, 在图的内部表示某些谱色的混合。在图 6-10 中, 等能量点对应三原色的等量比例, 表示白光的 CIE 标准。位于色度图边界上的任何点都是全饱和的。当某个点离开边界并接近等能量点时, 就会有更多的白光加入, 饱和度变小。等能量点处的饱和度为 0。

在色度图中, 连接任意两个点的直线段说明了所有不同的颜色变化, 可以由相加的两种颜色混合得到。例如, 考虑连接图 6-10 中红和绿的直线。如果在颜色中有比绿光更多的红光, 那么描述颜色的点将在线段上, 并且比绿点更接近红点。类似地, 从等能量点到色度图边界上任意一点而画的一条线将定义所有的特殊谱色的浓淡色调。

把这个过程扩展到三种颜色是很简单的。为了决定从色度图中给定的任何三种颜色均可得到的颜色范围, 我们为三个色点中的每一个画连接线。结果是三角形, 并且在三角形的边界和内部会产生由三种原始颜色混合而成的颜色。由任何三个固定颜色处的顶点形成的三角形不能包括图 6-10 中的全部颜色区域。此观察使得我们经常所做的解释变得很清楚, 任何颜色都可从确定的三原色产生是误解。

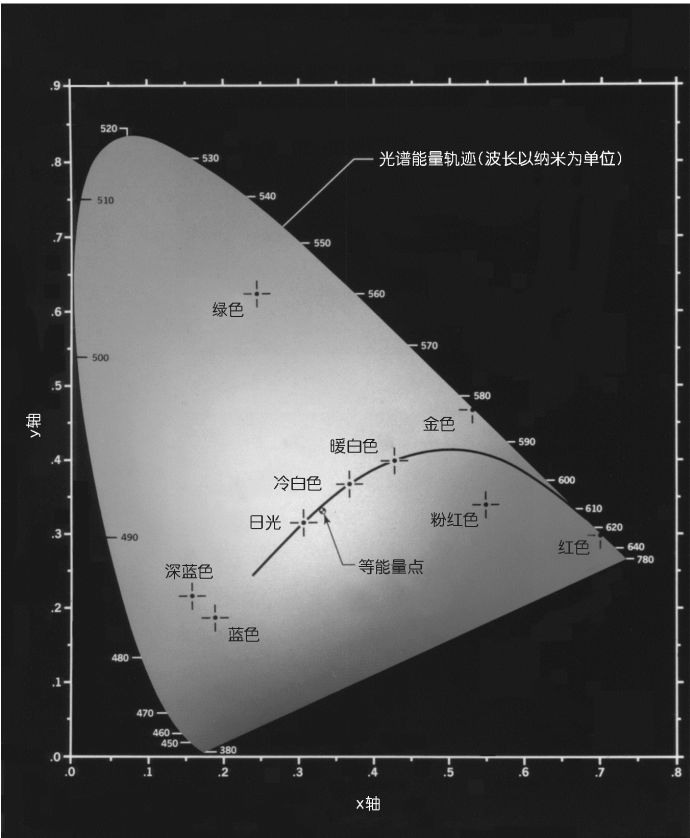


图 6-10 CIE 色度图

2. CIE 家族中独立于设备的彩色空间

10 年间，由于 XYZ 彩色空间的引入，CIE 还开发了一些其他的彩色空间规范，试图提供可供选择的比 XYZ 更好的能适应某些目的的彩色表示。例如，1976 年 CIE 引入的  $L^*a^*b^*$  彩色空间，这是在色彩科学、创新艺术和诸如打印机、摄像机、扫描仪等彩色设备的设计中广泛应用的彩色空间。作为工作空间，相比 XYZ， $L^*a^*b^*$  具有两个重要优点。首先， $L^*a^*b^*$  更清楚地分离了彩色信息(用  $a^*$  和  $b^*$  值表示)和灰度信息(完全用  $L^*$  值表示)。其次，在这个空间中， $L^*a^*b^*$  颜色被设计为欧氏距离，这很好地对应了彩色之间的感觉差别。因为这个性质， $L^*a^*b^*$  彩色空间被认为是感知一致的。作为必然结果， $L^*$  值对人类的亮度感知是线性的，也就是说，如果一种颜色的  $L^*$  值是另一种颜色  $L^*$  值的两倍，那么第一种颜色感觉亮两倍。注意，由于人类视觉系统的复杂性，感知的一致性仅仅是近似把握。表 6-4 列出了 CIE 家族中设备独立的受图像处理工具箱支持的彩色空间。对于 CIE 彩色模型的技术细节，请参考 Sharma 编写的书籍。

表 6-4 图像处理工具箱支持的独立于设备的彩色空间

彩 色 空 间	描 述
XYZ	最初于 1931 年引入的 CIE 彩色空间规范
xyY	提供了归一化颜色值的 CIE 规范，大写的 Y 值表示亮度，与 XYZ 中的一样
uvL	试图使颜色平面在视觉上更一致的 CIE 规范，L 是亮度，与 XYZ 中的一样

(续表)

彩 色 空 间	描 述
u' v' L	为了改进一致性, u 和 v 被重新标定的 CIE 规范
L*a*b*	试图使亮度在视觉上更一致的 CIE 规范, L*是 L 的非线性尺度, 它对白色基准点是归一化的
L*ch	这里的 c 是浓度, h 是色调, 这些值是 L*a*b*中 a*和 b*的极坐标变换

3. sRGB 彩色空间

正如本节早些时候提到的那样, RGB 彩色模型依赖于设备。这意味着对于给定的一组 R、G、B 值, 不存在单一、明确的彩色解释。另外, 图像文件常常不包含获取图像时所用设备的彩色特性信息。就像结果那样, 相同的图像文件可能(经常就是)在不同的计算机系统中看上去明显不同。比如 20 世纪 90 年代, 互联网应用激增, Web 设计人员常常发现, 他们不能准确地预知在用户浏览器上显示时图像的颜色看上去如何。

为解决这个问题, 微软和惠普提出了新的默认彩色空间标准, 称作 sRGB。sRGB 彩色空间被设计为与计算机的 CRT 监视器标准特性相一致, 并且与 PC 机在家庭和办公室观察环境相一致。sRGB 彩色空间独立于设备。因此, sRGB 颜色值很容易被改变为另一个独立于设备的彩色空间。

sRGB 标准已变成计算机界广泛接受的标准, 特别是面向消费者的设备。数字摄像机、扫描仪、计算机显示器和打印机等, 都例行地被设计为假定图像的 RGB 值与 sRGB 的彩色空间是一致的, 除非图像文件包含更多的指定设备的彩色信息。

4. CIE 和 sRGB 彩色空间之间的转换

工具箱函数 makecform 和 applycform 可用于独立于设备的彩色空间之间的转换。表 6-5 列出了它们支持的转换。函数 makecform 用于创建 cform 结构, 方式类似于使用 maketform 创建 tform 结构(见第 5 章)。相应的 makecform 语法是:

```
cform = makecform(type)
```

其中, type 是表 6-5 所示的字符串之一。函数 applycform 使用 cform 结构转换颜色。applycform 的语法是:

```
g = applycform(f, cform)
```

表 6-5 图像处理工具箱支持的独立于设备的彩色空间转换

makecform 中使用的类型	彩 色 空 间
'lab2lch'、'lch2lab'	L*a*b*和 L*ch
'lab2srgb'、'srgb2lab'	L*a*b*和 sRGB
'lab2xyz'、'xyz2lab'	L*a*b*和 XYZ
'srgb2xyz'、'xyz2srgb'	sRGB 和 XYZ
'upvpl2xyz'、'xyz2upvpl'	u' v' L 和 XYZ
'uvl2xyz'、'xyz2uvl'	uvL 和 XYZ
'xyl2xyz'、'xyz2xyl'	xyY 和 XYZ



**例 6.3** 以  $L^*a^*b^*$  彩色空间为基础创建在感觉上一致的彩色空间

在这个例子中,我们构建一个彩色标尺,它可用于彩色和黑白出版物。McNames 列出了这样一些彩色标尺的设计原则:

- 两个彩色标尺间感知颜色的不同应该与沿着标尺之间的距离成正比。
- 亮度应该单调地递增,因此标尺工作于黑白出版物。
- 贯穿标尺的相邻彩色应该尽可能明显。
- 标尺应包含较宽的彩色范围。
- 彩色标尺应该直观。

我们用创建  $L^*a^*b^*$  彩色空间的路径设计彩色标尺以满足前 4 条原则。第 1 条原则,感觉尺度的一致性,可用  $L^*a^*b^*$  中的彩色等距离间隔来满足。第 2 条原则,单调递增的亮度,可用构建  $L^*$  ( $L^*$  的值在 0(黑)和 100(完美的漫射亮度)间变化)值的线性斜坡来满足。这里设计在 40 和 80 之间间隔相等的估值为 1024 的斜坡:

```
>> L = linspace(40, 80, 1024);
```

对于不同的相邻彩色,第 3 条原则可用色调上的不同彩色来满足,对应于  $a^*b^*$  平面中彩色的极角:

```
>> radius = 70;
>> theta = linspace(0, pi, 1024);
>> a = radius * cos(theta);
>> b = radius * sin(theta);
```

第 4 条原则要求使用宽范围的颜色。我们的一组  $a^*$  和  $b^*$  值可延伸得尽可能远(以极角计)。在标尺的开始,得到接近的第一个颜色,没有最后颜色。

下一步,我们制作一幅  $L^*a^*b^*$  彩色标尺的  $100 \times 104 \times 3$  大小的图像:

```
>> L = repmat(L, 100, 1);
>> a = repmat(a, 100, 1);
>> b = repmat(b, 100, 1);
>> lab_scale = cat(3, L, a, b);
```

为了在 MATLAB 中显示彩色标尺图像,首先必须转换为 RGB 图像。使用 makecform 制作 cform 结构,然后使用 applycform:

```
>> cform = makecform('lab2srgb');
>> rgb_scale = applycform(lab_scale, cform);
>> imshow(rgb_scale)
```

图 6-11 显示了结果。



图 6-11 基于  $L^*a^*b^*$  彩色空间的在感觉上一致的标尺

第 5 条原则,在直觉上依赖于应用,评估起来要困难得多。不同的颜色标尺可用相同过程来构建,但却使用在  $a^*b^*$  平面上  $L^*$  中不同的开始值和结束值。得到的新彩色标尺可能对一定的应用更直观。

## 5. ICC 彩色剖面

文本彩色在计算机监视器上可以有外观，并且在打印时，外观会有很大不同。或者在不同的打印机上打印时，文本的颜色可能看起来也不同。为了在不同的输入、输出和显示设备上得到高质量的彩色重现，创建变换以把颜色从某个设备映射到另一个设备是必要的。通常，在每一对设备之间需要单独的彩色变换，对于不同的打印条件需要附加变换，设置设备质量等。这些变换中的每一个都必须用谨慎的控制和校准实验条件来开发。很明显，这样一种方法会被证明对所有这些高花费的高端系统并不适用。

国际彩色协会(ICC)在 1993 年建立的行业组织已经标准化了另一种不同方法。每一个设备正好有两个与之关联的变换，而不管在系统中可能存在的其他设备的数量。其中的一个变换把设备彩色转换为标准，独立于设备的彩色空间叫做剖面连接空间(*Profile Connection Space*, PCS)。另一个变换是第一个变换的反变换，用来把 PCS 彩色转换回设备彩色(PCS 不是 XYZ，就是  $L^*a^*b^*$ )。这两个变换合起来构成设备的 ICC 彩色剖面。

ICC 的主要目的之一是创建标准化的、维护和提升 ICC 彩色剖面的标准。图像处理工具箱函数 `iccread` 读取剖面文件，语法是：

```
p = iccread(filename)
```

输出 `p` 是包含文件头信息和数字参数，以及计算设备和 PCS 间彩色空间转换所必需的表格。使用 ICC 剖面转换彩色是通过使用 `makecform` 和 `applycform` 实现的。

`makecform` 的语法是：

```
cform = makecform('icc', src_profile, dest_profile)
```

其中，`src_profile` 是源设备剖面的文件名，`dest_profile` 是目的设备剖面的文件名。

ICC 彩色剖面标准包括用于处理被称为全域映射(*gamut mapping*)的临界彩色转换的机制。彩色全域是彩色空间中的量，用于定义设备可复现的彩色范围。彩色全域随设备的不同而不同。例如，典型监视器显示的某些颜色用打印机就不能复现。因此，颜色从一种设备映射到另一种设备时，考虑不同的全域是有必要的。源和目的全域间差别的补偿处理被称为全域映射。有许多不同的方法用于全域映射。有些方法对于特定的目的要好于其他方法。ICC 彩色剖面标准为全域映射定义了 4 个“目的”(称作渲染意图)。表 6-6 中描述了这些渲染目的。指定渲染意图的 `makecform` 语法是：

```
cform = makecform('icc', src_profile, dest_profile, ...
    'SourceRenderingIntent', src_intent, ...
    'DestRenderingIntent', dest_intent)
```

其中，`src_intent` 和 `dest_intent` 可从 'Perceptual'(默认)、'AbsoluteColorimetric'、'RelativeColorimetric' 和 'Saturation' 中选择。

表 6-6 ICC 剖面的渲染意图

渲染意图	描 述
感知的	优化全域映射以达到在美学意义上最合意的结果, 可能没有维持全域彩色
绝对色度的	把全域之外的彩色映射到最近的全域表面。保持全域内的彩色关系, 根据最完美的扩散器渲染彩色
相对色度的	把全域之外的彩色映射到最近的全域表面。保持全域内的彩色关系, 根据设备的白点或输出介质渲染彩色
饱和的	在牺牲移动色调的可能代价下, 最大化设备彩色的饱和度。打算使用简单的图形和图标, 而不是图像

#### 例 6.4 ICC 彩色剖面的软实验

在这个例子中, 我们使用 ICC 彩色剖面 `makecform` 和 `applycform` 执行被称为软实验的处理。软实验在计算机监视器上模拟打印一幅彩色图像时将呈现的外观。概念上, 软实验分两步进行:

- (1) 把监视器颜色(常假定为 sRGB)转换为输出设备颜色, 通常用感知的渲染意图。
- (2) 把计算过的输出设备颜色转换回监视器颜色, 用绝对色度的渲染意图。

对于输入剖面, 我们将使用 `sRGB.icm`——描述 sRGB 彩色空间的剖面, 使用工具箱载运。我们的输出是 `SNAP2007.icc`——新闻纸剖面, 包含在 ICC 的注册剖面中([www.color.org/registry](http://www.color.org/registry))。我们的样品图像与图 6-4(a)一样。首先用环绕图像的拥有粗白边和细灰边的边界对图像做预处理。这些边界将使得更容易观察对新闻用纸“白”度的模拟:

```
>> f = imread('Fig0604(a).tif');
>> fp = padarray(f, [40 40], 255, 'both');
>> fp = padarray(fp, [4 4], 230, 'both');
>> imshow(fp)
```

图 6-12(a)显示了填充的图像。

下一步, 我们读进两个剖面并用它们把虹膜图像从 sRGB 转换为新闻纸颜色:

```
>> p_srgb = iccread('sRGB.icm');
>> p_snap = iccread('SNAP2007.icc');
>> cform1 = makecform('icc', p_srgb, p_snap);
>> fp_newsprint = applycform(fp, cform1);
```

最后, 我们用绝对色度的渲染意图创建 `cform` 结构, 以便为了显示而将之转换回 sRGB:

```
>> cform2 = makecform('icc', p_snap, p_srgb, ...
    'SourceRenderingIntent', 'AbsoluteColorimetric', ...
    'DestRenderingIntent', 'AbsoluteColorimetric');
>> fp_proof = applycform(fp_newsprint, cform2);
>> imshow(fp_proof)
```

图 6-12(b)显示了结果。这幅图像本身仅是在监视器上看到的实际结果的近似, 因为印刷书籍的彩色全域与监视器的彩色全域不同。

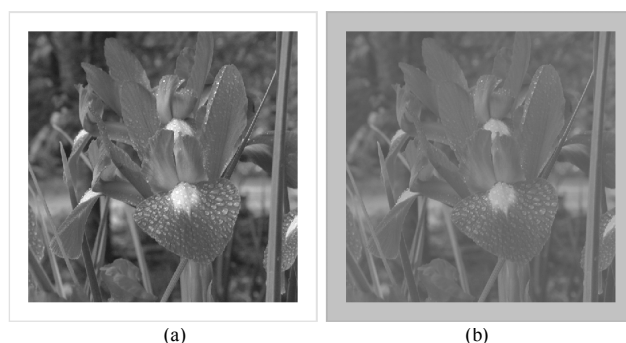


图 6-12 软实验示例：(a) 具有白边的原始图像；(b) 当在新闻纸上打印时对图像外观的模拟

## 6.3 彩色图像处理的基础知识

在这一节，我们开始研究用于彩色图像的处理技术。虽然它们远没有穷尽，但是在本节开发的技术说明了针对各种图像处理任务怎样处理彩色图像。针对下面的讨论目的，我们把彩色图像处理细分成 3 个主要领域：

- 1) 颜色变换(也叫彩色映射)
- 2) 单独彩色平面的空间处理
- 3) 颜色向量的处理

第 1 类处理每个彩色平面的像素，这类处理严格地以像素值为基础，而不是它们的空间坐标。这类处理类似于 2.2 节的灰度变换处理。第 2 类处理涉及各个彩色平面的空间(邻域)滤波，类似于 2.4 节和 2.5 节讨论的空间滤波。第 3 类处理涉及以同时处理彩色图像的所有分量为为基础的处理技术。因为全彩图像至少有三个分量，彩色像素可以用向量来处理。例如在 RGB 系统中，每个彩色点都可以在 RGB 系统中作为从原点延伸到那一点的向量来描述(见图 6-2)。

令  $c$  代表 RGB 彩色空间中的任意向量：

$$c = \begin{bmatrix} c_R \\ c_G \\ c_B \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

这个公式指出， $c$  分量是一副彩色图像在某个点上的 RGB 分量。考虑彩色分量是坐标的函数这样的事实，用下边的符号表示：

$$c(x, y) = \begin{bmatrix} c_R(x, y) \\ c_G(x, y) \\ c_B(x, y) \end{bmatrix} = \begin{bmatrix} R(x, y) \\ G(x, y) \\ B(x, y) \end{bmatrix}$$

对于一幅大小为  $M \times N$  的图像，有  $MN$  个这样的向量  $c(x, y)$ ，其中， $x=0, 1, 2, \dots, M-1$  且  $y=0, 1, 2, \dots, N-1$ 。

在某些情况下，无论彩色图像每次处理一个平面，还是作为向量处理，都会得到相等的结果。然而，正如在 6.6 节中更详细的描述那样，不会总是这样的情况。为了使两种方法都相同，两个条件必须满足：首先，处理必须对向量和标量都可用。其次，针对向量的每个分量的操作

必须与其他分量无关。正如说明的那样,图 6-13 显示了灰度图像及全彩色图像的空间邻域处理。假设处理是求邻域平均,在图 6-13(a)中,平均值可以通过对邻域内所有像素的灰度级求和,并且用邻域内的像素总数去除来完成。在图 6-13(b)中,平均值通过对邻域内全部向量求和,并用邻域内向量总数去除每个分量来完成。但是,平均向量的每个分量是图像中相应分量的像素的和。如果平均是分别在图像的每个分量上计算,然后形成彩色向量,那么将得到相同的结果。

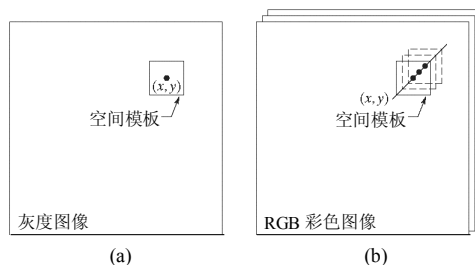


图 6-13 灰度图像及 RGB 彩色图像的空间模板

## 6.4 彩色变换

这里描述的是单一彩色模型情况下的技术,以处理彩色图像的彩色分量或单色图像的亮度分量为基础。对于彩色图像,我们限定如下形式的变换:

$$s_i = T_i(r_i) \quad i = 1, 2, \dots, n$$

这里,  $r_i$  和  $s_i$  是输入和输出图像的彩色分量,  $n$  是  $r_i$  是彩色空间的维数(或是彩色分量的数量),并且  $T_i$  是全彩色变换(或叫映射)函数。如果输入图像是单色的,公式将如下所示:

$$s_i = T_i(r) \quad i = 1, 2, \dots, n$$

这里,  $r$  表示灰度级的值,  $s_i$  和  $T_i$  正如上文谈到的那样,并且  $n$  是在  $S_i$  中彩色分量的数量。这个公式描述了灰度级对任意颜色的映射,这一处理在伪彩色变换或伪彩色映射中经常提到。注意,如果我们让  $r_1=r_2=r_3=r$ , 第一个公式可用来处理 RGB 中的单色图像。在其他情况下,这里给出的公式可直接延伸到 2.2 节介绍的灰度变换公式。在那一节,如果变换是适用的,那么所有的  $n$  个伪彩色或真彩色变换函数  $\{T_1, T_2, \dots, T_n\}$  是与空间图像坐标  $(x, y)$  无关的。

在第 2 章介绍过一些灰度变换,比如 `imcomplement`, 用于计算一幅图像的负片,与被变换图像的灰度内容无关。另一方面, `histeq` 是自适应的,它与灰度的分布有关,但是一旦必要的参数被估计到了,变换就固定了。还有另外一种情况,比如 `imadjust`, 要求使用者选择合适的曲线形状参数,最好是交互地指定。当用伪彩色和全彩色映射时,尤其是涉及人类观察和说明(比如彩色平衡)的时候,存在类似的情况。在这些应用中,用直接产生候选函数的图形描述,并在被处理图像上观察综合效果(实时)的方法,是选择合适映射函数的最好方法。

图 6-14 说明了用图示法指定映射函数的一种简单而有力的方法。图 6-14(a)显示了一种变换,这是通过线性地插入三个控制点(图中用圆圈住的坐标)来形成的。图 6-14(b)显示了另一种变换,由相同的三个控制点通过三次样条内插得到。图 6-14(c)和(d)分别提供了更复杂的线性和三次样条内插方法。而这两种内插方法在 MATLAB 中均已得到支持,线性内插使用下面指令实现:

```
z = interpq(x,y,xi)
```

上述指令返回列向量，包括在  $xi$  点处线性内插的一维函数  $z$ 。列向量  $x$  和  $y$  指定如下控制点的坐标。 $x$  的元素一定是单调增长的。 $z$  的长度等于  $xi$  的长度。例如：

```
>> z = interpq([0 255],[0 255],[0:255],')
```

该式产生含有 256 个元素的一点接一点的与控制点(0,0)及(255,255)相连接的映射，也就是  $z=[0,1,2,...255]'$ 。

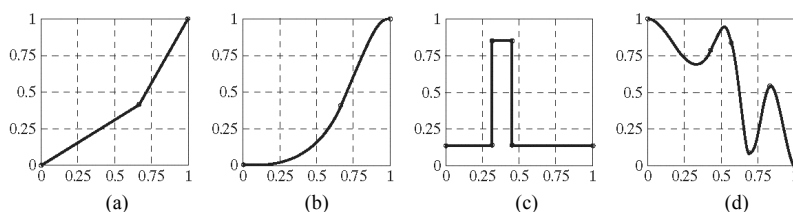


图 6-14 指定使用了控制点的映射函数：(a)和(c)是线性内插；(b)和(d)是三次样条内插

采用类似的方法，三次样条内插用 `spline` 函数来实现：

```
z = spline(x,y,xi)
```

其中，变量  $x$ 、 $y$ 、 $z$  和  $xi$  与在前一段中对 `interpq` 的描述一样。然而， $xi$  在 `spline` 函数中使用时必须是不同的。此外，如果  $y$  比  $x$  多包含两个或更多个元素，那么  $y$  的第一项和最后一项假定引入三次样条的滚降。例如，在图 6-14(b)中描述过的函数一般使用“0”滚降产生该函数。变换函数的说明可以用图形法操作控制点的方式交互地产生，那些控制点输入到 `interpq` 和 `spline` 函数并实时地显示将被处理的图像的结果。语法是：

```
g = ice('property name', 'property value',...)
```

在这里，'property name'和'property value'必须成对出现，并且点指出由相应的输入对组成的模式的重复。表 6-7 列出了 `ice` 函数中使用的有效对，一些例子在本节稍后给出。关于'wait'参数，当选择'on'选项时，选项不是显式的，就是默认的，输出  $g$  是处理后的图像。在这种情况下，`ice` 作为处理的控制，包括光标，在命令窗口不必键入任何命令，直到函数关闭。这时，最后的结果是带有句柄的图像  $g$  (通常是图形目标)。

表 6-7 函数 `ice` 的有效输入

属 性 名	属 性 值
'image'	一幅 RGB 或单色输入图像 $f$ ，由交互指定的映射来变换
'space'	被修改的分量彩色空间。可能的值是 'rgb'、'cmv'、'hsi'、'hsv'、'ntsc' (或 'yiq') 和 'ycbcr'，默认值是 'rgb'
'wait'	如果是 'on' (默认)， $g$ 是被映射的图像；如果是 'off'， $g$ 是映射输入图像的句柄

当选择 'off' 时， $g$  为处理后图像的句柄，并且控制立即返回到命令窗口；因此，还可以和 `ice` 函数一起键入新的指令来执行。为了得到图形目标的属性，我们用函数：

```
h = get(g)
```

这个函数返回全部特性以及由句柄  $g$  识别的图形目标的当前可用值。这些特性保存在结构  $h$  内,所以在提示符处键入  $h$ ,就会列出所有的处理后图像的特性(见 1.7.8 节关于结构的解释)。为了提取特殊的特性,我们可键入  $h.properName$ <sup>①</sup>。

令  $f$  表示一幅 RGB 或单色图像,下面是  $ice$  函数语法的使用示例:

```
>> ice                                % Only the ice
                                     % graphical
                                     % interface is
                                     % displayed.

>> g = ice('image', f);              % Shows and returns
                                     % the mapped image g.

>> g = ice('image', f, 'wait', 'off') % Shows g and returns
                                     % the handle.

>> g = ice('image', f, 'space', 'hsi') % Maps RGB image f in
                                     % HSI space.
```

注意,当指定的彩色空间不同于 RGB 时,输入图像(不管是单色还是 RGB)在执行任何映射之前,需要被变换到指定的空间。对于输出,映射后的图像转换为 RGB。 $ice$  的输出总是 RGB,输入总是单色或 RGB。如果键入  $g=ice('image',f)$ ,图像和图形用户界面(GUI)就像图 6-15 显示的那样在 MATLAB 桌面上出现。最初,变换曲线是在每个端点带有控制点的直线,控制点由鼠标操作,如表 6-8 中总结的那样。表 6-9 列出了  $ice$  函数的典型应用。

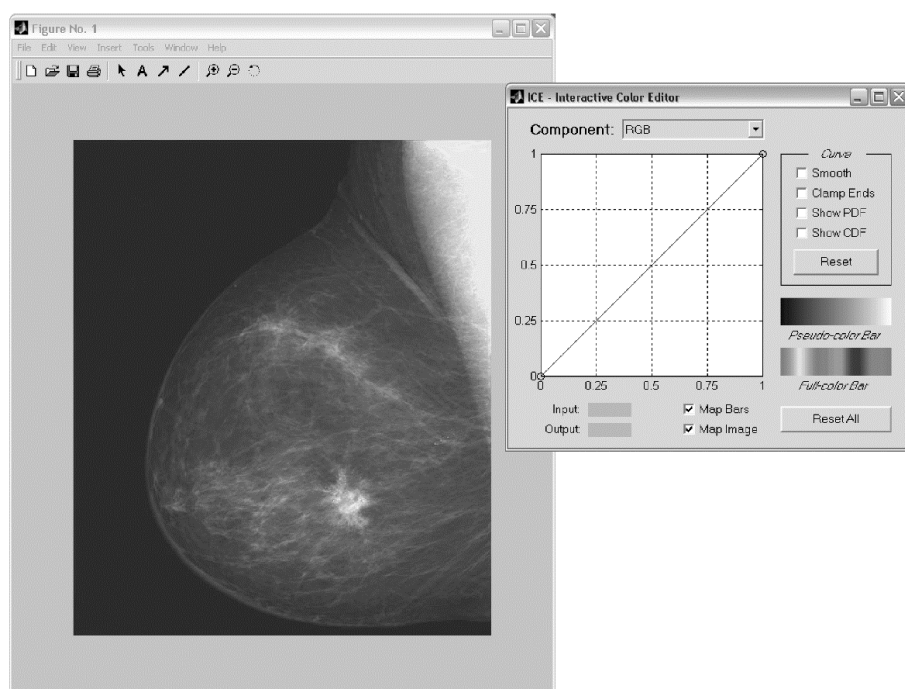


图 6-15  $ice$  函数典型的打开窗口(原图像由 G.E.医学系统提供)

① 无论 MATLAB 创建什么图形目标,都会分配标志符(叫句柄)给目标,作为访问目标的属性。当用 M-文件创建和直接对目标改进图形外观或创建客户绘图命令时,图形句柄非常有用。

表 6-8 用鼠标操作控制点

鼠 标 动 作	结 果
左键	通过拖按鼠标移动控制点
左键+Shift 键	添加控制点, 通过拖拉可以改变控制点的位置(当一直按着 Shift 键时)
左键+Control 键	删除控制点

\* 对于三键鼠标来说, 左键、中键和右键配合移动, 可添加和删除表 6-8 中的操作。

表 6-9 复选框的功能和 GUI 按钮

GUI 元 素	描 述
Smooth	检查三次样条(平滑曲线)内插。如果没有检查, 就使用分段线性内插
Clamp Ends	在三次样条内插中, 检查强制开始和结束的曲线坡度为 0, 分段线性内插不受影响
Show PDF	显示由映射函数引起的图像分量的概率密度函数(也就是直方图)
Show CDF	代替 PDF 显示累积分布函数(注意, PDF 和 CDF 不能同时显示)
Map Images	如果检查过, 图像映射被激活, 否则不激活
Map Bars	如果检查过, 伪彩色和全彩色条带映射被激活, 否则不映射条带(分别为灰度楔形和色度楔形)
Reset	初始化当前显示的映射函数, 并不检查所有的曲线参数
Reset All	初始化所有映射函数
Input/Output	显示变换曲线上已选择控制点的坐标, 输入指的是水平轴, 输出指的是垂直轴
Component	为交互操作选择映射函数, 在 RGB 空间中, 可能的选择包括 R、G、B 和 RGB(映射全部三颜色分量)。在 HSI 空间中, 可选择 H、S、I 和 HS, 等等

例 6.5 单色负片和彩色分量的反映射

图 6-16(a)显示了在图 6-15 默认的 RGB 曲线被修改之后, 产生的相反的或负的映射函数的 ice 界面。为了建立新的映射函数, 控制点(0,0)被移动(通过单击和拖拉到右上角)到(0,1), 并且控制点(1,1)被移动到点(1,0)。请注意, 光标的坐标如何在输入输出框中显示成红色。仅修改 RGB 映射, 单独的 R、G、B 映射分别保留默认的 1:1 状态(见表 6-6 所列分量)。

对于单色输入, 这可以保证单色输出。图 6-16(b)显示了从反映射得到的单色负片。注意, 与图 2-3(b)完全相同, 它是由 imcomplement 函数得到的。在图 6-16(a)中, 伪彩色条带是图 6-15 中灰度条的“照片底片”。

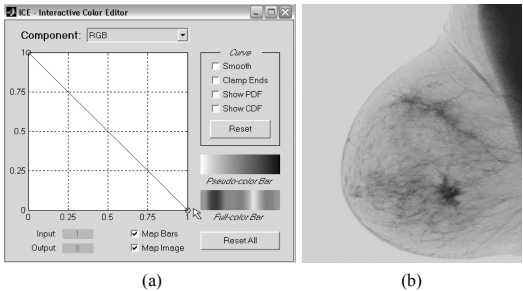


图 6-16 (a) 负映射函数; (b) 图 6-15 中单色图像的效果



反的或负的映射函数在彩色处理中也很有用。正如在图 6-17(a)和(b)中显示的那样,映射的结果使人回想起普通彩色胶片的底片。例如在图 6-17(a)中最底行的红色粉笔,被变换为图 6-17(b)中的青色,即红色的补色。原色的补色是其他两种原色的混合(例如,青色就是蓝色加绿色)。在灰度情况下,补色对于在彩色暗区增强细节很有用。尤其是当这块区域的大小占支配地位时。注意在图 6.16(a)中,全彩色条带包含图 6-15 中全彩条色调的补色。

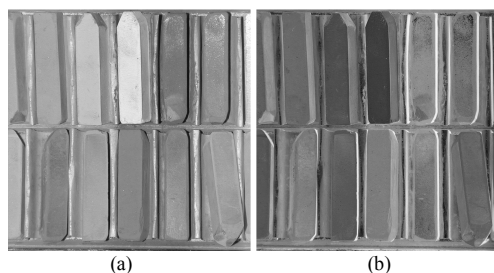


图 6-17 (a) 全彩色图像; (b) 相应的负片(彩色补色)

### 例 6.6 单色和彩色对比度增强

考虑对于单色和彩色对比度操作的 ice 函数的下一个应用。图 6-18(a)到(c)展示了在单色图像处理中 ice 函数的效果。图 6-18(d)到(f)显示了对彩色输入类似的效果。正如前边的例子那样,映射函数没有保持在默认值或 1:1 状态下。在这两种处理顺序中,显示 PDF 复选框是激活的。这样一来,图 6-18(a)中航空照片的直方图被显示在图 6-18(c)中 gamma 形状的映射函数之下(见 2.2.1 节)。并且在图 6-18(f)中,针对图 6-18(c)中的彩色图像提供了 3 个直方图——为三个彩色分量中的每个均提供直方图。虽然在图 6-18(f)中, s 形的映射函数增加了图 6-18(d)中图像的对比度(与图 6-18(e)相比),但是这对于色度仍有一点影响。

彩色的小变化(见图 6-18(e))实际上是觉察不到的,但是映射的结果是明显的,比如在图 6-18(f)中,从映射的全彩色参考条带中可以看到。回忆前边的例子,一幅 RGB 图像的 3 个分量的等量变化在彩色方面就会有戏剧性的效果(见图 6-17 中彩色补色的映射)。

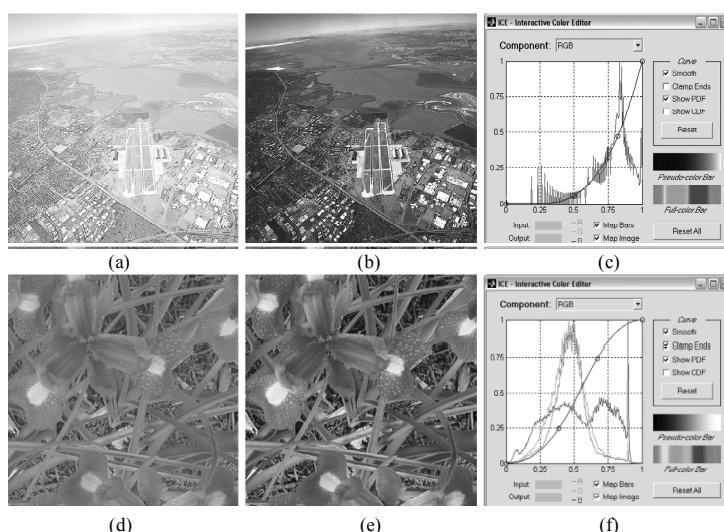


图 6-18 使用 ice 函数的单色和全彩色对比度增强: (a)和(b)是输入图像,这两者都有被冲淡的现象; (b)和(e)显示了处理后的结果; (c)和(f)是 ice 显示。(原始的黑白图像由 NASA 提供)

对例 6.5 和例 6.6 中输入图像的红、绿、蓝分量施以相同的映射，也就是采用相同的变换函数。为避免定义相同的函数，ice 函数提供了“全分量”函数(当在 RGB 彩色空间内操作时的 RGB 曲线)。函数用来映射所有的输入分量。本章中其他示例展示的变换会对 3 个分量进行不同的处理。

### 例 6.7 伪彩色映射

正如早些时候提到的那样，当一幅黑白图像在 RGB 彩色空间中描述，并且产生的分量进行独立映射时，变换的结果就是一幅伪彩色图像，其中输入图像的灰度级被任意彩色代替。进行这些变换是很有用的，因为人眼可以辨认出上百万种颜色，但是相对来说，只能辨认不多的灰度。这样，伪彩色映射常常被用于变换较小的灰度变化，以使人眼看得见，或者突出重要的灰度区域。事实上，伪彩色的主要应用是人类的可视化，也就是在一幅灰度图像或经过灰度到彩色赋值的序列图像中对灰度事件的判读。

图 6-19(a)是一幅包含几个裂缝和孔(通过图像中间的明亮白条纹)的焊接部分(水平黑色区域)的 X 光图像。伪彩色图像显示在图 6-19(b)中，是使用图 6-19(c)和(d)中的映射函数对输入中 RGB 的绿蓝分量进行映射而产生的。注意，伪彩色映射会产生令人惊异的视觉上的不同。GUI 伪彩色参考条带提供了方便的视觉引导以合成映射。正像你在图 6-19(c)和(d)中看到的那样，交互地指定映射函数，把黑到白灰度变换为蓝色和红色之间的色调，黄色保留为白色。当然，黄色也可对应焊接的裂缝和孔，它们在本例中是相当重要的特征。

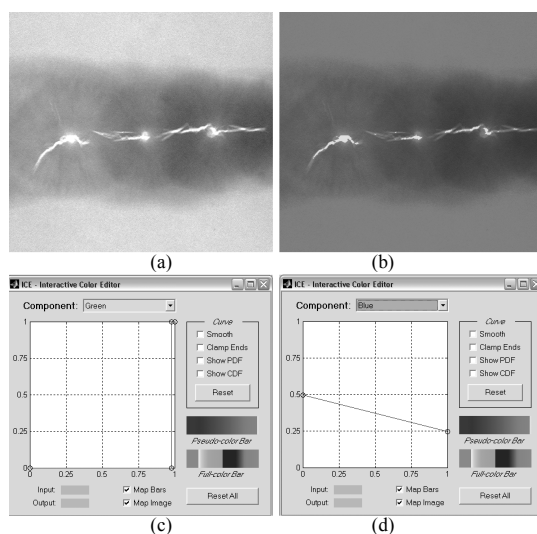


图 6-19 (a) 一幅有缺陷的焊接部位的 X 光图像; (b) 焊接的伪彩色图像; (c)和(d) 绿、蓝分量的映射函数。(原图像由 X-TEK Systems 公司提供)

### 例 6.8 色彩平衡

图 6-20 显示了包括全彩图像的应用，这对独立地映射一幅图像的彩色分量是很有利的。通常叫做色彩平衡或色彩校正，这种类型的映射主要支持高端彩色重现系统，但是现在，在大多数桌面计算机上都可以运行。其中一种重要的应用是照片增强。虽然色彩的不平衡可以通过颜色分光计对已知图像的色彩进行分析来决定，但当有白色区域时，这里的 RGB 和 CMY 分量在

相等的情况下,得到正确的视觉估计是有可能的。在图 6-20 中我们可以看到,肤色对视觉估计来说是优秀的样本,因为人类对正常肤色的反应很敏锐。

图 6-20(a)显示了一幅用过量的紫红色的 CMY 扫描后的母亲和孩子的图像(记住,只有 RGB 版本的图像能由 MATLAB 显示)。为了保持 MATLAB 的简单和兼容性,ice 函数仅接受 RGB(和单色)输入,但是可在各种彩色空间中处理输入图像,表 6-7 对此有详述。为了交互地修改 RGB 图像 f1 的 CMY 分量,可以使用合适的 ice 调用:

```
>> f2 = ice('image',f1,'space','cmy');
```

如图 6-20 所示,紫红色的略微减少对图像颜色有显著的影响。

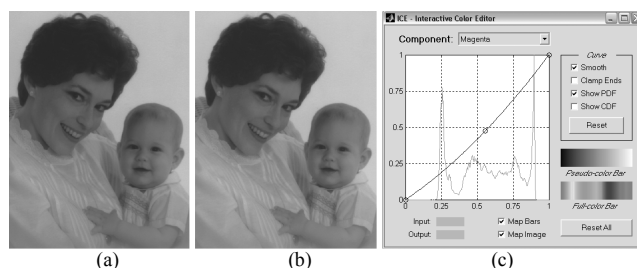


图 6-20 用于色彩平衡的 ice 函数: (a) 紫色很重的图像; (b) 校正过的图像; (c) 用于校正不平衡色彩的映射函数

### 例 6.9 基于直方图的映射

直方图均衡是一种灰度映射处理,用于寻求产生具有均匀灰度直方图的单色图像。正如在 2.3.2 节讨论过的那样,所需要的映射函数是输入图像中灰度的累积分布函数(CDF)。因为彩色图像有多个分量,所以灰度技术必须被改成处理多个分量以及关联的直方图。正如可以预期的那样,直方图分别均衡彩色图像的分量是不明智的,结果经常产生错误的颜色。更合逻辑的方法是均匀地延伸彩色的亮度,保留彩色本身(例如色调)不变。

图 6-21(a)显示了调味瓶架子上放着调味瓶和混合容器的彩色图像。图 6-21(b)中变换后的图像明显要亮一些,这幅图像是用图 6-21(c)和(d)中的变换产生的。现在,木桌的造型和纹理以及放置在木桌上的几个调味瓶都看得见了。使用图 6-21(c)中的函数映射亮度分量,使之大致接近 CDF 的那个分量(也显示在图中)。选择图 6-21(d)中的色调映射函数,旨在改进亮度均衡结果的全部彩色感受。注意,输入图像的直方图和输出图像的色彩度、饱和度和亮度分量都分别显示在图 6-21(e)和(f)中。当亮度和饱和度分量被更改时,色度分量实际上还是相同的(这正是希望的)。最后注意,为了在 HSI 彩色空间中处理 RGB 图像,在调用 ice 时,我们输入了特定的名称/数值对 'space'/'hsi'。

在本节中,前边例子产生的输出图像是 RGB 类型和 unit8 类。对于单色结果,比如在例 6.5 中,RGB 输出的所有 3 个分量是完全相同的。通过表 6-3 中的 rgb2gray 函数或者使用下列指令,可以得到更简洁的表示:

```
>> f3 = f2(:,:,1);
```

其中, f2 是一幅由 ice 产生的 RGB 图像,并且 f3 是一幅黑白图像。

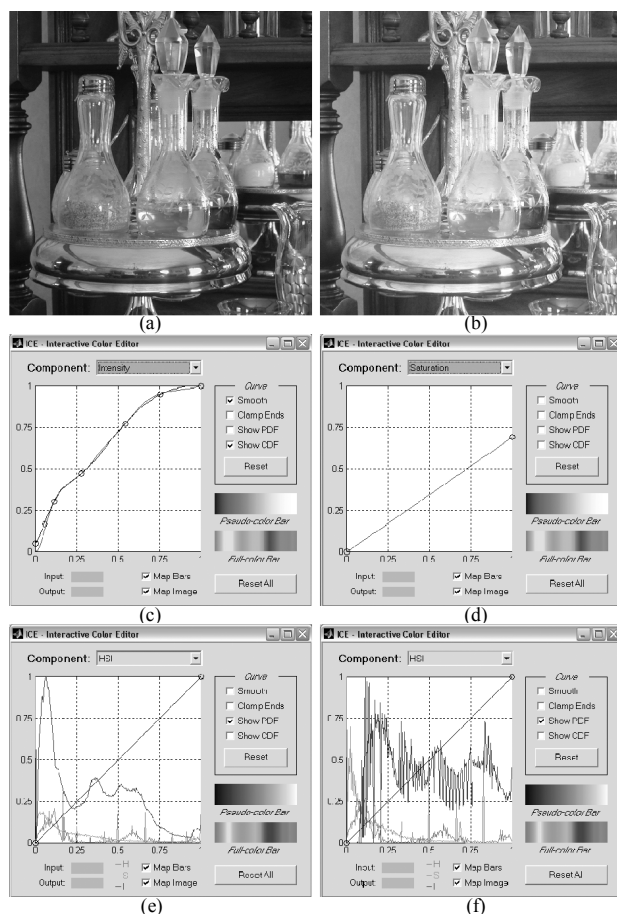


图 6-21 在 HSI 彩色空间中, 用饱和度调整的直方图均衡: (a) 输入图像; (b) 映射结果; (c) 亮度分量映射函数和累积分布函数; (d) 饱和度分量映射函数; (e) 输入图像的分量直方图; (f) 映射结果的分量直方图

## 6.5 彩色图像的空间滤波

6.4 节的材料在单一彩色分量平面的单一图像像素上执行色彩变换。更复杂级别的处理包括执行空间邻域处理, 这也在单一图像平面上进行。这一分析与 2.2 节针对灰度变换的讨论, 以及 2.4 节和 2.5 节有关空间滤波的讨论是类似的。我们介绍的彩色图像的空间滤波集中在 RGB 图像上, 但是对于其他彩色模型, 基本概念也是可用的。我们用两个线性滤波的例子说明彩色图像的空间处理: 图像平滑和图像锐化。

### 6.5.1 彩色图像的平滑处理

参考图 6-13(a)以及 2.4 节和 2.5 节中的讨论, 平滑单色图像的一种方法是定义相应的系数是 1 的模板, 用空间模板的系数去乘所有像素的值, 并用模板中元素的总数去除。用空间模板平滑全彩色图像的处理见图 6-13(b)。平滑处理(以 RGB 空间为例)以处理灰度图像时采用的方式并以公式来表达, 除了代替单像素外, 我们现在还处理 6.3 节中的向量值。

令  $S_{xy}$  表示彩色图像中以  $(x,y)$  为中心的邻域的一组坐标。在该邻域中, RGB 向量的平均值是:

$$\bar{c}(x, y) = \frac{1}{K} \sum_{(s,t) \in S_{xy}} c(s, t)$$

其中,  $K$  是邻域中像素点的数量。根据 6.3 节中的讨论, 附加向量的特性是:

$$\bar{c}(x, y) = \begin{bmatrix} \frac{1}{K} \sum_{(s,t) \in S_{xy}} R(s, t) \\ \frac{1}{K} \sum_{(s,t) \in S_{xy}} G(s, t) \\ \frac{1}{K} \sum_{(s,t) \in S_{xy}} B(s, t) \end{bmatrix}$$

我们意识到, 这个向量的每个分量都作为我们将要得到的结果, 结果是用每个分量图像执行邻域平均获得的, 这里使用的是上边提到的滤波器模板。因此, 我们得出这样的结论: 用邻域平均的平滑可以在每个图像平面的基础上执行。如果邻域平均直接在彩色向量空间执行, 那么结果是相同的。正如在 2.5.1 节中讨论的那样, 前面讨论的空间平滑滤波器类型是用带选项 'average' 的 `fspecial` 函数产生的。一旦滤波器产生, 滤波就用 2.4.1 节中介绍过的函数 `imfilter` 来执行。概念上, 平滑 RGB 彩色图像 `fc` 时, 线性空间滤波由下面的步骤组成:

(1) 抽取 3 个分量图像:

```
>> fr = fc(:,:,1);
>> fg = fc(:,:,2);
>> fb = fc(:,:,3);
```

(2) 分别过滤每个分量图像。例如, 令 `w` 表示用 `fspecial` 产生的平滑滤波器, 平滑红色分量图像:

```
>> fR_filtered = imfilter(fR, w, 'replicate');
```

其他两个分量图像与此类似。

(3) 重建滤波过的 RGB 图像:

```
>> fc_filtered = cat(3, fR_filtered, fG_filtered, fB_filtered);
```

然而, 因为可以在 MATLAB 中使用与单色图像相同的语法来执行 RGB 图像的线性滤波, 所以可以把前三步合并为一步:

```
>> fc_filtered = imfilter(fc, w, 'replicate');
```

#### 例 6.10 彩色图像的平滑处理

图 6-22(a)显示了尺寸为  $1197 \times 1197$  像素的 RGB 图像, 并且图 6-22(b)到(d)是 RGB 分量图像, 它们是用前面描述过的步骤提取出来的。我们从前边讨论的结果知道, 平滑单独的分量图像和用前一段末尾给出的指令平滑原始 RGB 图像是相同的。图 6-24(a)显示了使用大小为  $25 \times 25$  像素的平均滤波器得到的结果。

下边我们研究仅对图 6-22(a)中 HSI 版本的亮度分量进行平滑的效果。图 6-23(a)到(c)显示了用函数 `rgb2hsl` 得到的三幅 HSI 分量图像。其中, `fc` 是图 6-22(a)。

```
>> h = rgb2hsl(fc);
```

```
>> H = h(:, :, 1);
>> S = h(:, :, 2);
>> I = h(:, :, 3);
```

接下来,我们用尺寸为  $25 \times 25$  像素的相同滤波器滤波亮度分量。平均滤波器已足够大,可以产生有意义的模糊度。选择这个尺寸的滤波器,是为了演示在 RGB 空间中进行平滑处理的效果,与在 RGB 空间被变换到 HSI 空间后,只使用图像的亮度分量达到类似结果之间的不同之处。图 6-24(b)通过如下指令获得:

```
>> w = fspecial('average', 25);
>> I_filtered = imfilter(I, w, 'replicate');
>> h = cat(3, H, S, I_filtered);
>> f = hsi2rgb(h); % Back to RGB for comparison.
>> imshow(f);
```

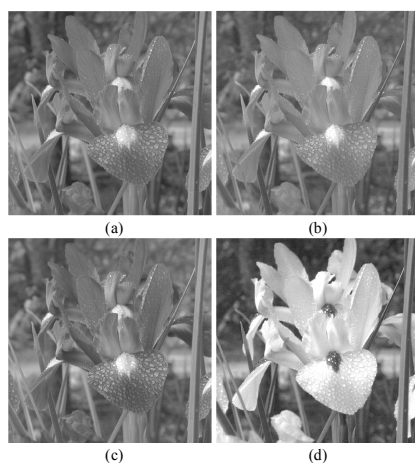


图 6-22 (a) RGB 图像; (b)到(d)分别是红、绿、蓝分量图像

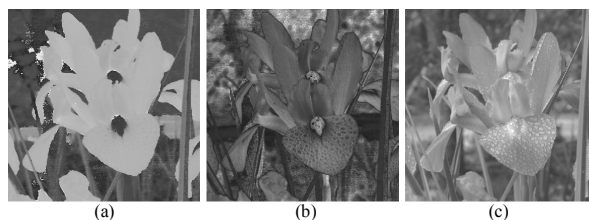


图 6-23 从左到右: 图 6-22(a)的色调、饱和度和亮度分量

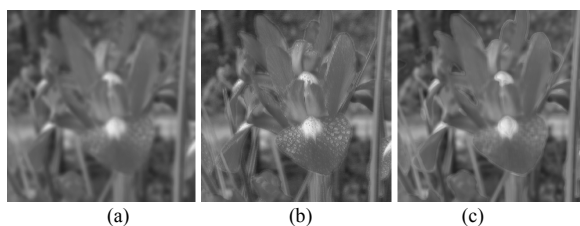


图 6-24 (a) 分别通过平滑  $R$ 、 $G$ 、 $B$  图像平面得到的平滑后的 RGB 图像; (b) 仅对 HIS 相等图像的亮度分量进行平滑的结果; (c) 平滑所有三个 HIS 分量的结果

显然,滤波后的两种结果完全不同。例如,除图像有点模糊以外,请注意图 6-24(b)中花朵

顶部模糊的绿色边缘。原因是通过平滑处理,亮度分量值的变化减少了,但色调和饱和度分量没有变化。合乎逻辑的情况是用相同的滤波器去平滑所有的三个 HSI 分量,然而,这将改变色调和饱和度值之间的相对关系,这样会产生无意义的结果,如图 6-24(c)所示。特别是在这幅图像中,观察图像有多少围绕着花朵的较亮绿色边缘。还有围绕着中心黄色区域的边界,这个效果也是十分明显的。

一般来说,当模板的尺寸减小时,对 RGB 分量图像进行滤波和对同一图像的 HSI 亮度分量进行滤波时,得到的差别也减少了。

## 6.5.2 彩色图像的锐化处理

用线性空间滤波锐化一幅 RGB 图像遵循与前面相同的步骤,但是应使用锐化滤波器。在本节考虑使用拉普拉斯(见 2.5.1 节)使图像锐化。

从向量分析中,我们知道向量的拉普拉斯被定义为矢量,它们的分量等于输入向量的分量的拉普拉斯。在 RGB 彩色系统中,在 6.3 节引入的矢量  $\mathbf{c}$  的拉普拉斯是:

$$\nabla^2[\mathbf{c}(x,y)] = \begin{bmatrix} \nabla^2 R(x,y) \\ \nabla^2 G(x,y) \\ \nabla^2 B(x,y) \end{bmatrix}$$

正如在前面介绍的那样,这告诉我们,可以通过分别计算每个分量图像的拉普拉斯来计算全彩图像的拉普拉斯。

### 例 6.11 彩色图像的锐化处理

图 6-25(a)显示了稍微有点模糊的图 6-22(a)的版本 fb,这是用 5×5 的均值滤波器得到的。为了锐化这幅图像,我们使用拉普拉斯滤波模板:

```
>> lapmask = [1 1 1;1 -8 1;1 1 1];
```

然后,用如下命令计算增强后的图像并显示:

```
>> fb = tofloat(fb);
>> fen = fb - imfilter(fb, lapmask, 'replicate');
>> imshow(fen)
```

注意,和前面一样,RGB 图像直接用 imfilter 滤波,图 6-25(b)显示了结果。注意,图像在锐度特性上的显著加强,比如水滴、叶子上的纹路、花朵黄色的中心和前景中明显的绿色植物。

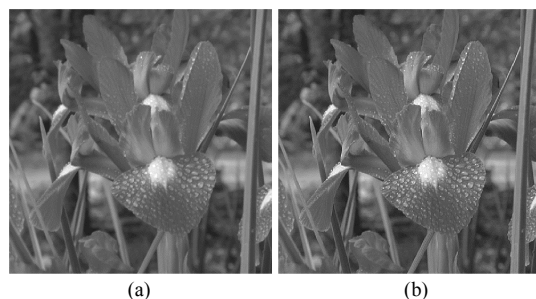


图 6-25 (a)模糊图像; (b) 用拉普拉斯增强图像

## 6.6 直接在 RGB 矢量空间中处理

正如在 6.3 节中提到的那样, 存在这样一些情况, 基于单独彩色平面的处理不等于直接在 RGB 矢量空间中进行的计算。在本节, 我们将通过考虑彩色图像处理的两个重要应用来说明矢量处理: 彩色边缘检测和区域分割。

### 6.6.1 使用梯度的彩色边缘检测

2D 函数  $f(x,y)$  的梯度定义为如下矢量:

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

这个矢量的大小是:

$$\begin{aligned} \nabla f = \text{mag}(\nabla f) &= [g_x^2 + g_y^2]^{1/2} \\ &= [(\partial f / \partial x)^2 + (\partial f / \partial y)^2]^{1/2} \end{aligned}$$

通常, 这个数量用绝对值来近似:

$$\nabla f \approx |g_x| + |g_y|$$

这个近似值避免了平方和开方计算, 但是仍然具有推导过的特性(例如在常数区域为 0, 在像素值变化的区域, 幅度与变化程度成比例)。在通常的应用中, 把梯度的幅值简单地作为梯度。

梯度向量的基本特性是在  $f$  坐标  $(x,y)$  处指向最大变化率的方向。最大变化率发生的角度是:

$$\alpha(x,y) = \tan^{-1} \left[ \frac{g_y}{g_x} \right]$$

按照惯例, 这个导数用在一幅图像中某个小邻域内像素值的差来近似。图 6-26(a)显示了  $3 \times 3$  大小的邻域, 这里的  $z$  代表亮度值。在区域中心点  $x$  方向上(垂直)的偏微分的近似由下边的差给出:

$$g_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

$z_1$	$z_2$	$z_3$	-1	-2	-1	-1	0	1
$z_4$	$z_5$	$z_6$	0	0	0	-2	0	2
$z_7$	$z_8$	$z_9$	1	2	1	-1	0	1
(a)			(b)			(c)		

图 6-26 (a) 为小的邻域; (b)和(c) 用于计算关于邻域中心点的  $x$  方向(垂直)和  $y$  方向(水平)的梯度的 Sobel 模板

类似的,  $y$  方向上的微分由下面的差近似:

$$g_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

在一幅图像上, 所有点的这两个数量很容易计算, 可分别使用图 6-26(b)和(c)中的两个模板



单独对图像进行滤波(使用函数 `imfilter`)。然后,相应梯度图像的近似可以通过对两个滤波图像的绝对值的求和获得。刚刚讨论的模板是在表 2-5 中提到的 Sobel 模板。它们可用 `fspecial` 函数产生。

用刚刚讨论的这一方法计算梯度是在灰度图像中进行边缘检测时最常用的方法之一,在第 10 章将更详细地进行讨论。此刻,我们的兴趣是在 RGB 彩色空间中计算梯度。然而,刚刚推导的方法可应用于 2D 空间,但是不能扩展到高维空间。将之运用到 RGB 图像的唯一方法是计算每个彩色图像分量的梯度,然后合并结果。遗憾的是,正如我们在本节稍后说明的那样,这与直接在 RGB 向量空间中计算边缘是不同的。

问题是在 6.3 节定义过的向量  $c$  的梯度。下面是可选的各种方法之一,在这里,梯度的概念可延伸到向量函数。

令  $r$ 、 $g$  和  $b$  是 RGB 彩色空间沿  $R$ 、 $G$ 、 $B$  轴的单位矢量(见图 6-2),定义矢量:

$$u = \frac{\partial R}{\partial x} r + \frac{\partial G}{\partial x} g + \frac{\partial B}{\partial x} b$$

和

$$v = \frac{\partial R}{\partial y} r + \frac{\partial G}{\partial y} g + \frac{\partial B}{\partial y} b$$

根据这些矢量的点积,定义  $g_{xx}$ 、 $g_{yy}$  和  $g_{xy}$ :

$$g_{xx} = u \cdot u = u^T u = \left| \frac{\partial R}{\partial x} \right|^2 + \left| \frac{\partial G}{\partial x} \right|^2 + \left| \frac{\partial B}{\partial x} \right|^2$$

$$g_{yy} = v \cdot v = v^T v = \left| \frac{\partial R}{\partial y} \right|^2 + \left| \frac{\partial G}{\partial y} \right|^2 + \left| \frac{\partial B}{\partial y} \right|^2$$

和

$$g_{xy} = u \cdot v = u^T v = \frac{\partial R}{\partial x} \frac{\partial R}{\partial y} + \frac{\partial G}{\partial x} \frac{\partial G}{\partial y} + \frac{\partial B}{\partial x} \frac{\partial B}{\partial y}$$

记住  $R$ 、 $G$  和  $B$ , 因而  $g$  是  $x$  和  $y$  的函数。使用这种符号,可以说明—— $c(x,y)$  的最大变化率方向将作为  $(x,y)$  函数由角度给出:

$$\theta(x,y) = \frac{1}{2} \tan^{-1} \left[ \frac{2g_{xy}}{(g_{xx} - g_{yy})} \right]$$

并且在该方向上,由  $\theta(x,y)$  给出的变化率的值(例如梯度值)由下式给出:

$$F_\theta(x,y) = \left\{ \frac{1}{2} \left[ (g_{xx} + g_{yy}) + (g_{xx} - g_{yy}) \cos 2\theta(x,y) + 2g_{xy} \sin 2\theta(x,y) \right] \right\}^{1/2}$$

数组  $\theta(x,y)$  和  $F_\theta(x,y)$  是与输入图像尺寸相同的图像。 $\theta(x,y)$  的元素是用于计算梯度的每个点的角度,并且  $F_\theta(x,y)$  是梯度图像。因为  $\tan(\alpha) = \tan(\alpha \pm \pi)$ , 所以如果  $\theta_0$  是上述公式方程的解,

那么  $\theta_0 \pm \pi/2$  也是。此外,  $F_\theta(x,y)=F_{\theta+\pi}(x,y)$ ,  $F$  仅需要在半开区间  $[0, \pi)$  上计算  $\theta$  的值。

上述方程提供了两个相隔  $90^\circ$  的值, 这一事实意味着这个方程涉及每个点  $(x,y)$  的一对正交方向。沿着这些方向  $F$  是最大的, 并且沿着另一个方向是最小的。所以, 最后结果由选择的每个点上的最大值产生。这些结果的推导是很冗长的, 并且作为我们当前讨论的基本目标来说, 在这里详细讨论获益不大。可以在 Di Zenzo 的论文中找到相关细节。为了执行上述公式, 要求偏微分计算可以用在本节早先讨论的 Sobel 算子来实现。

下面的函数用来计算彩色图像的梯度(代码可见附录 C):

```
[VG,A,PPG] = colorgrad(f,T)
```

其中,  $f$  是 RGB 图像,  $T$  是  $[0,1]$  范围内的阈值选项(默认为 0);  $VG$  是 RGB 向量梯度  $F_\theta(x,y)$ ;  $A$  是以弧度计的角度  $\theta(x,y)$ , 并且  $PPG$  是由单独彩色平面的 2D 梯度之和形成的梯度图像。计算上述方程时, 要求全部微分都可用函数 `colorgrad` 中的 Sobel 算子来实现。输出  $VG$  和  $PPG$  通过 `colorgrad` 被归一化到  $[0, 1]$  范围内, 并且把它们作为阈值进行处理。所以, 它们的值小于或等于  $T$ ,  $VG(x,y)=0$ ; 对于其他的情况,  $VG(x,y)=VG(x,y)$ 。类似的解释可用于  $PPG$ 。

#### 例 6.12 使用函数 `colorgrad` 检测 RGB 边缘

图 6-27(a)到(c)显示了 3 幅黑白图像, 当使用 RGB 平面时, 产生图 6-27(d)所示的彩色图像。这个例子的目的是: (1) 说明函数 `colorgrad` 的使用; (2) 说明通过合并单独彩色平面的梯度来计算彩色图像的梯度, 与直接在 RGB 向量空间中用刚刚说明的方法计算梯度是不同的。

令  $f$  表示图 6-27(d)中的 RGB 图像。命令:

```
>> [VG,A,PPG] = colorgrad(f)
```

产生的图像  $VG$  和  $PPG$  显示在图 6-27(e)和(f)中。在这两个结果中, 最重要的不同是图 6-27(f)的水平边缘比图 6-27(e)的对应边缘更弱。原因很简单: 当蓝色平面的梯度产生单一水平边缘时, 红色和绿色平面的梯度(图 6-27(a)和(b))产生两个垂直边缘。为形成  $PPG$ , 相加这三个梯度将产生两倍于水平边缘亮度的垂直边缘。

另一方面, 当彩色图像的梯度在向量空间中(图 6-27(e))直接计算时, 垂直和水平边缘的比值是  $\sqrt{2}$  而不是 2。原因也很简单: 参考图 6-2(a)中的彩色立方体和图 6-27(d)中的图像, 我们看到, 彩色图像的垂直边缘是在蓝白方块和黑黄方块之间。这些颜色在彩色立方体之间的距离是  $\sqrt{2}$ , 但是在黑蓝和黄白(水平边缘)之间的距离仅是 1。这样, 垂直和水平差别的比率是  $\sqrt{2}$ 。如果边缘准确度是问题, 尤其是当使用阈值时, 那么这两个方法之间的差别是很重要的。例如, 如果我们使用 0.6 的阈值, 图 6-27(f)中的水平线将会消失。

当我们的兴趣是边缘检测, 而不是准确度时, 一般来说, 刚刚讨论过的两种方法拥有可比较的结果。例如, 图 6-28(b)和(c)类似于图 6-27(e)和(f)。它们通过对图 6-28(a)运用 `colorgrad` 函数而获得。图 6-28(d)是标度在范围  $[0,1]$  内的两幅图像的梯度差。两幅图像的绝对最大差别是 0.2, 相对 8 bit 范围  $[0,255]$  来说, 相当于将之转换为 51 灰度级。然而, 这两幅梯度图像在视觉外观上却十分接近, 图 6-28(b)在某些地方有一点亮(与前一段解释的理由相似)。因而, 对于这种类型的分析, 每个独立分量梯度的计算方法比较简单, 一般来说可以接受。在其他情况下, 比如准确性很重要的情况, 向量方法可能是必需的。

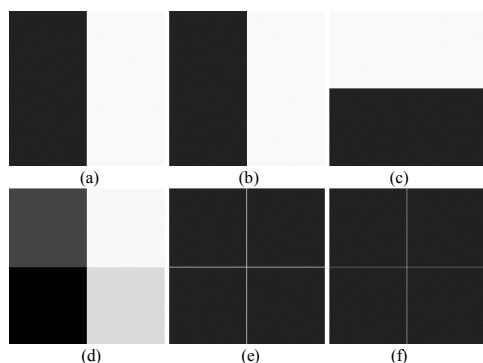


图 6-27 (a)到(c) RGB 分量图像; (d) 相应的彩色图像; (e) 在 RGB 向量空间中直接计算的梯度; (f) 通过分别计算每个 RGB 分量图像的 2D 梯度并相加结果而获得的梯度

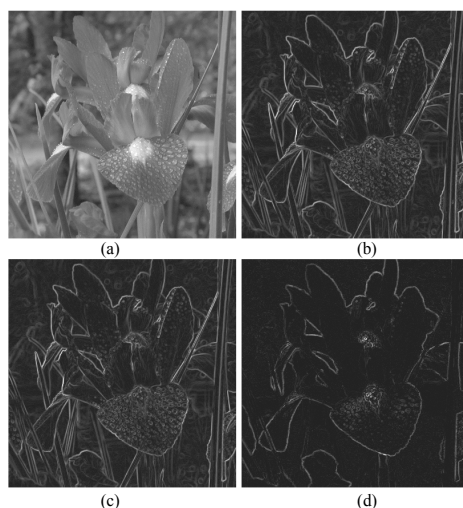


图 6-28 (a) RGB 图像; (b) 在 RGB 向量空间计算的梯度; (c) 在图 6-27(f)中计算的梯度; (d) (b)和(c)之间的绝对差, 标度为[0,1]范围内

## 6.6.2 在 RGB 向量空间中分割图像

分割是把一幅图像分成一些区域。虽然分割是第 10 章的主题, 但出于连续性考虑, 我们在这里概要地考虑彩色区域分割, 下面的讨论你不会感到有困难。

使用 RGB 彩色向量进行彩色区域分割是很简单的。假设目的是在 RGB 图像中分割某个特定彩色区域内的物体。给定一组感兴趣的有代表性的彩色(或彩色范围)样点, 我们获得“平均”或期望的颜色估计, 这是希望的分割。让这个平均色用 RGB 向量  $m$  来定义。分割的目的是对图像中的每个 RGB 像素进行分类, 使之在指定的范围内有或没有一种颜色。为了执行这一比较, 拥有相似性度量是必要的。最简单的度量之一是欧几里德距离。令  $z$  表示 RGB 空间的任意点。如果它们之间的距离小于指定的阈值  $T$ , 那么  $z$  相似于  $m$ 。 $z$  和  $m$  之间的欧几里德距离由下式给出:

$$\begin{aligned} D(z, m) &= \|z - m\| \\ &= \left[ (z - m)^T (z - m) \right]^{1/2} \\ &= \left[ (z_R - m_R)^2 + (z_G - m_G)^2 + (z_B - m_B)^2 \right]^{1/2} \end{aligned}$$

在这里,  $\|\bullet\|$  是参量的范数, 并且下标  $R$ 、 $G$ 、 $B$  表示向量  $m$  和  $z$  的 RGB 分量。点的轨迹  $D(z, m) \leq T$  是半径为  $T$  的球, 正如图 6-29(a)说明的那样。根据定义, 包括在球的内部或表面的点满足特定的彩色准则, 球外部的点则不满足。在图像中对这两组点编码, 比如黑的和白的, 将会产生分割的二值图像。上述方程的有用归纳是距离度量形式:

$$D(z, m) = \left[ (z - m)^T C^{-1} (z - m) \right]^{1/2}$$

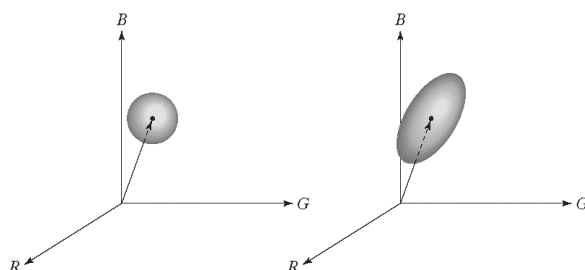


图 6-29 在以分割为目的的 RGB 向量空间中封装数据的两种方法

在这里,  $C$  是我们要分割的有代表性彩色样值的协方差矩阵。一般来说, 这个距离以 Mahalanobis 距离为参考。 $D(z, m) \leq T$  的轨迹描述了实的 3D 椭球体(见图 6-29(b)), 它的重要特性是: 主轴取向是在最大的数据分离方向上。当  $C=I$  时, 同样的矩阵, Mahalanobis 距离降至 Euclidean 距离。除了现在数据包含在椭球内之外, 分割与在这段之前已经描述过的一样。

在刚才描述的方法中, 分割通过函数 `colorseg`(代码见附录 C)实现, 语法如下:

```
S = colorseg(method, f, T, parameters)
```

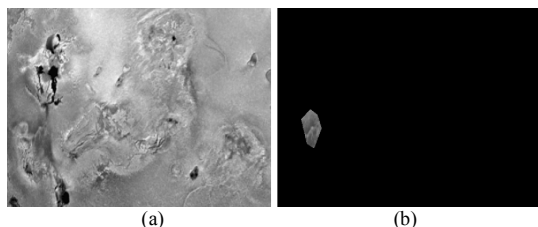
在这里, `method` 不是 'euclidean' 就是 'mahalanobis', `f` 是待分割的 RGB 彩色图像, `T` 是前边描述的阈值。如果选择 'euclidean', 输入参量将是 `m`, 如果选择 'mahalanobis', 将是 `m` 和 `c`。参数 `m` 是均值  $m$ , `c` 是协方差矩阵  $C$ 。输出 `S` 是二值图像(和原始图像的大小相同), 包括 0s。阈值测试失败的点为 0, 通过阈值测试的点为 1。1 表示基于彩色内容从 `f` 分割出的区域。

### 例 6.13 RGB 彩色图像分割

图 6-30(a)显示了一幅木星的卫星表面区域的伪彩色图像。在这幅图像中, 淡红色描述的是新的从活火山喷射出来的熔岩, 并且在黄色熔岩周围是老的硫磺沉淀物。为了比较, 这个例子演示了如何使用 `colorseg` 函数的两个选项对淡红色区域进行分割。

首先, 我们获得待分割彩色区域的样本。一种简单地获得感兴趣区域的方法是使用在 4.2.4 节描述的 `roipoly` 函数, 这个函数可产生能交互选择的区域的二值模板。这样, 令 `f` 表示图 6-30(a)中的彩色图像, 图 6-30(b)中的区域便可用下面命令得到:

```
>> mask = roipoly(f); % Select region interactively.
>> red = immultiply(mask, f(:, :, 1));
>> green = immultiply(mask, f(:, :, 2));
>> blue = immultiply(mask, f(:, :, 3));
>> g = cat(3, red, green, blue);
>> figure, imshow(g);
```

图 6-30 (a) 木星的卫星表面区域的伪彩色图像; (b) 用函数 `roipoly` 交互式地提取感兴趣的区域。(原图像由 NASA 提供)

其中,模板是用 `roipoly` 产生的二值图像(和 `f` 的大小相同)。下一步,我们计算感兴趣区域内点的平均矢量和协方差矩阵。但是首先,ROI 区域内点的坐标必须提取出来:

```
>> [M, N, K] = size(g);
>> I = reshape(g, M * N, 3);
>> idx = find(mask);
>> I = double(I(idx, 1:3));
>> [C, m] = covmatrix(I);
```

第 2 条语句重新排列 `g` 中的彩色像素,就像 `I` 的行一样,并且第 3 条语句找出彩色像素的行索引,它们不是黑的。这些都不是图 6-30(b)中模板图像的背景像素。

最后,预先要做的计算是决定 `T` 值。好的开始是让 `T` 变为彩色分量标准差的倍数。`C` 的主对角线包括 RGB 分量的方差,所以必须抽取这些元素并计算它们的平方根:

```
>> d = diag(c);
>> sd = sqrt(d);
    22.0643    24.2442    16.2806
```

`sd` 的第 1 个元素是 ROI 中彩色像素的红色分量的标准差,并且对另外两个分量也是相似的。现在进行图像分割,以 `T` 的 25 倍值作为阈值,这个值是最大标准差的近似: `T=25`, `T=50`, `T=75`, `T=100`。针对函数的 `'euclidean'` 选项,采用 `T=25`:

```
>> E25 = colorseg('euclidean',f,25,m);
```

图 6-31(a)显示了结果,并且图 6-31(b)到(d)显示了分别用 `T=50`、`75` 和 `100` 进行分割的结果。类似的,图 6-32(a)到(d)显示了以同样的阈值顺序,使用 `'mahalanobis'` 选项获得的结果。

有意义的结果(依赖图 6-30(a)中的红色)可用 `'Euclidean'` 选项获得,使用 `T=25` 和 `50`。但是,当 `T=75` 和 `100` 时会产生明显的过分分割。另一方面,使用 `'mahalanobis'` 选项, `T` 采用相同的值,结果明显更为准确,如图 6-32 所示。原因是在 ROI 中,3D 彩色数据的分离在使用椭球的情况下比圆球匹配得更好。注意,在这两种增大 `T` 的方法中,允许包括分割区域的红色更淡些,这正像我们期望的那样。

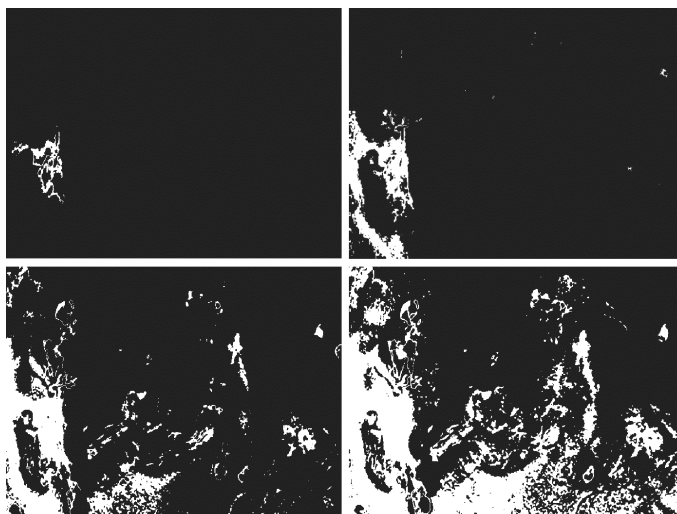


图 6-31 在函数 `colorseg` 中使用 `'euclidean'` 选项, `T=25`、`50`、`75`、`100`, 图 6-30(a)的分割效果

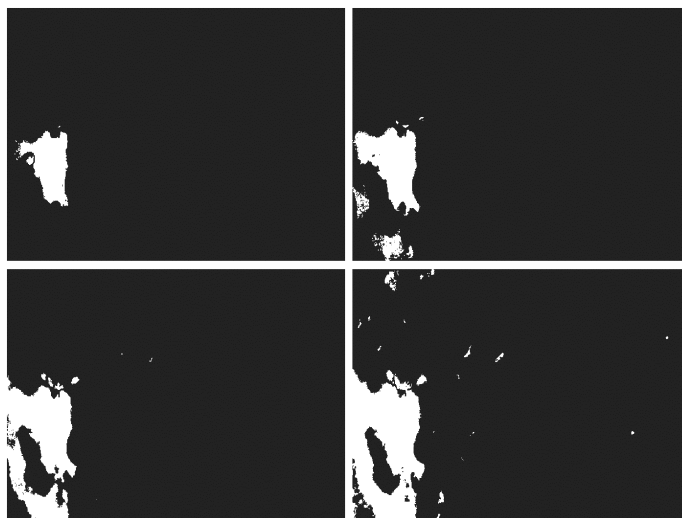


图 6-32 在函数 `colorseg` 中使用 'mahalanobis' 选项,  $T=25, 50, 75, 100$ , 图 6-30(a) 的分割效果, 请与图 6-31 进行比较

## 6.7 小结

本章针对图像处理中的彩色应用这一基本话题, 并对如何使用 MATLAB、图像处理工具箱以及在前面开发的新函数实现这些概念做了介绍。彩色模型这一领域的范围足够广, 以至于全书都将讨论这一话题。为了在图像处理中更有效, 这里讨论的模型都是精选的, 并且也因为它们而为在这个领域做进一步研究提供了良好的基础。

关于伪彩色、单独彩色平面的全彩处理与在前一章针对单色图像开发的图像处理技术具有紧密联系。关于彩色向量空间的讨论与前些章讨论的方法相悖, 这里强调灰度图像处理和全彩色图像处理之间的一些重要不同。本章开始时讨论的彩色向量处理技术是基于向量处理技术的代表, 包括中值和其他排序滤波、自适应和形态滤波、图像复原、图像压缩和其他技术。

## 图 像 分 割

前一章开始将输入和输出都是图像的图像处理方法转变为输入是图像，但输出是从图像中抽取出来的属性的处理方法。分割是此方向的另一个重要步骤。

分割把图像细分为它的组成要素或物体，细分的水平取决于要解决的问题。也就是说，当感兴趣的物体已经被隔离出来时，就应该停止分割。例如，自动的电子装配线中的自动检测，兴趣在于分析产品图像以判断物体是否存在特殊的异常，例如元件缺失或连接线断裂等。这里没有必要使用超过识别这些元件所需要的细节水平的分割。

特殊图像的分割是图像处理中最困难的技术之一。分割的精确度决定了最终计算机分析的成功与否。因为这个原因，应该仔细考虑稳定分割的可能性。在某些情况下，比如工业检测方面的应用，有时至少在控制环境的某些测量上是有可能的。其他方面，比如在遥感中，用户控制图像获取的主要限制是传感器的选择。

单色图像的分割算法通常是基于图像亮度值的两个基本特性：不连续性和相似性。在第一类中，方法是基于亮度的突变来分割一幅图像，比如边缘，在第二类中，主要方法是根据事先定义的准则把图像分割成相似的区域。

本章我们讨论一些刚才提到的两类方法，它们主要应用于单色图像(彩色图像分割已在 6.6 节中讨论)。我们从适合检测不连续亮度的方法开始，比如点、线、边缘。

边缘检测已经是多年来分割算法的主要成果。除了边缘检测本身外，我们还讨论使用霍夫变换检测线性边缘线段的分割。边缘检测的讨论紧接阈值处理技术之后。阈值处理也是占有重要地位的基本分割方法，特别是在速度成为重要因素的应用场合。完成对阈值处理的讨论后，开始讨论面向区域的分割方法。我们以被称为“分水岭分割”的形态学分割方法来结束这一章。这种方法特别有吸引力，因为它能产生闭合的、有良好定义的区域边界以及全局形式的表现，并提供了一个框架，其中关于图像的先验知识在特殊运用下可以用来改善分割结果。正如前边的章节那样，我们开发了一些新的可补充图像处理工具箱的函数。

### 10.1 点、线和边缘检测

在这一节，我们讨论在数字图像中检测亮度不连续的三种基本类型：点、线和边缘。寻找不连续的最常见方法是在 2.4 节和 2.5 节中描述的对整幅图像运用模板。对于  $3 \times 3$  的模板来说，该过程包括计算系数与模板覆盖区域包含的亮度的乘积之和。图像中的模板在任何一点的响应

$R$  由下式给出:

$$R = w_1 z_1 + w_2 z_2 + \cdots + w_9 z_9 = \sum_{i=1}^9 w_i z_i$$

其中,  $z_i$  是与模板系数  $w_i$  相关的像素的亮度。和前面一样, 模板的响应以它自身的中心来定义。

### 10.1.1 点检测

嵌在一幅图像的恒定区域或亮度几乎不变的区域里的孤立点的检测, 在原理上都是比较简单的。

使用图 10-1 中的模板, 如果在模板中心位置  $|R| \geq T$ , 我们就说孤立的点已经被检测出来了。

其中,  $T$  是非负的阈值。这种检测点的方法在工具箱中使用 `imfilter` 函数, 并用图 10-1 中的模板来实现。重要的要求是当模板中心是孤立点时, 模板的响应最强, 而在恒定亮度区域中, 响应为零。

-1	-1	-1
-1	8	-1
-1	-1	-1

图 10-1 点检测的模板

如果  $T$  已给出, 下面的指令执行刚才讨论的点检测:

```
>> g = abs(imfilter(tofloat(f), w)) >= T;
```

其中,  $f$  是输入图像,  $w$  是适合点检测的模板(也就是图 10-1 中的模板),  $g$  是包含检测点的图像。回忆 2.4.1 节中的讨论, `imfilter` 把输出转换为输入所属的类, 如果输入是整数类, 并且 `abs` 操作不接受整数数据, 那么在滤波操作中用 `tofloat(f)` 来防止对数值的过早截取。输出图像  $g$  是 `logical` 类; 值是 0 和 1。如果  $T$  值没有给出, 那么通常基于滤波结果来选取。在那种情况下, 先前的一串指令分成三个基本步骤: 1) 计算滤波后的图像 `abs(imfilter(tofloat(f), w))`; 2) 从滤波后的图像的数据中找出  $T$  的值; 3) 把滤波后的图像与  $T$  做比较。

#### 例 10.1 点检测

图 10-2(a)显示了一幅图像  $f$ , 在球体的东北象限有一个几乎不可见的黑点, 我们用下列程序检测该点:

```
>> w = [-1 -1 -1; -1 8 -1; -1 -1 -1];
>> g = abs(imfilter(tofloat(f), w));
>> T = max(g(:));
>> g = g >= T;
>> imshow(g)
```

在滤波后的图像  $g$  中选择最大值作为  $T$  值。然后在  $g$  中寻找所有的  $g \geq T$  的点。假设所有的点是孤立的镶嵌在恒定或是近似恒定的背景上。我们可识别能给出最大响应的点。因为在选择  $g$  中的最大值作为  $T$  值的情况下, 在  $g$  中不存在比  $T$  值大的点。我们使用 `>=` 算子(代替 `=`)定义一致性。如图 10-2(b)所示, 其中有一个孤立点, 该点使用  $T$  值置为 `max(g(:))` 且满足  $g \geq T$  的条件。

点检测的另一种方法是在大小为  $m \times n$  的所有邻点中寻找一些点, 最大值和最小值的差超出了  $T$  的值。这种方法可以用 2.5.2 节中介绍的 `ordfilt2` 命令来完成:



```
>> g = ordfilt2(f, m*n, ones(m, n)) - ordfilt2(f, 1, ones(m, n));
>> g = g >= T;
```

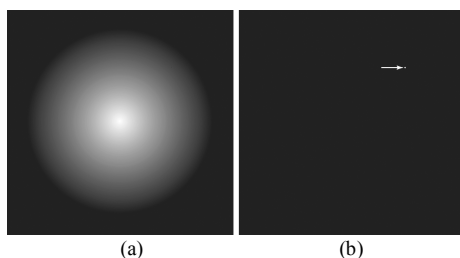


图 10-2 (a) 在球体的东北象限中带有几乎不可见的孤立黑点的灰度图像; (b) 已检测到该点的图像(为了便于观看, 黑点放大了)

很容易证实, 选择  $m = n = 5$  和  $T = \max(g(:))$  将产生与图 10-2(b) 中同样的结果。前面的公式比图 10-1 中的模板更灵活。例如, 如果我们想要计算邻域中最高和下一个最高像素值之间的差, 那么可以用前面由  $m*n-1$  表示的最右边的值代替 1。这一基本理论的其他方案可用相似的方法明确表达。

### 10.1.2 线检测

更复杂一点的是线检测。如果图 10-3 中的模板在图像上移动, 就会对水平线(一个像素宽)的响应更强烈。对于恒定的背景, 当线通过模板的中间一行时可能产生更大的响应。同样, 图 10-3 中的第 2 个模板对  $+45^\circ$  线响应最好, 第 3 个模板对垂直线响应最好, 第 4 个模板对  $-45^\circ$  线响应最好。注意, 每个模板的优先方向都用比其他可能方向要大的系数加权。每个模板的系数之和为 0, 这表明在恒定亮度区域中, 模板的响应为 0。

-1	-1	-1	2	-1	-1	-1	2	-1	-1	-1	2	-1	-1	2	-1	-1	2
2	2	2	-1	2	-1	-1	2	-1	-1	2	-1	-1	2	-1	2	-1	-1
-1	-1	-1	-1	-1	2	-1	2	-1	2	-1	2	-1	2	-1	2	-1	-1
水平			$45^\circ$			垂直			$-45^\circ$								

图 10-3 线检测模板

令  $R_1$ 、 $R_2$ 、 $R_3$  和  $R_4$  代表图 10-3 中模板的响应, 从左到右, 这里的  $R$  已经在 10.1.1 节的等式中给出。假定这 4 个模板分别用于图像, 如果在图像的中心点, 满足  $|R_i| > |R_j|, j \neq i$ , 我们就说那个点与模板  $i$  方向的线更相关。如果我们对图像中所有由给定模板定义的方向的线感兴趣, 可以简单地通过图像运行这些模板, 并对结果的绝对值取阈值, 留下来的点便是响应最强烈的那些点, 这些点与模板定义的方向最接近, 并且线只有一个像素宽。

#### 例 10.2 检测指定方向的线

图 10-4(a) 显示了一幅数字化(二值化)的电子线路连线模板的一部分。图像大小是  $486 \times 486$  像素。假如我们要寻找所有的一个像素宽的  $+45^\circ$  线。为了这一目的, 我们使用图 10-3 中的第 2 个模板。图 10-4(b) 到图 10-4(f) 是使用以下指令产生的, 其中的  $f$  是图 10-4(a) 中的图像:

```
>> w = [2 -1 -1; -1 2 -1; -1 -1 2];
>> g = imfilter(tofloat(f), w);
```

```

>> imshow(g, [ ]) % Fig. 10.4(b)
>> gtop = g(1:120, 1:120); % Top, left section.
>> gtop = pixeldup(gtop, 4); % Enlarge by pixel duplication.
>> figure, imshow(gtop, [ ]) % Fig. 10.4(c)
>> gbot = g(end - 119:end, end - 119:end);
>> gbot = pixeldup(gbot, 4);
>> figure, imshow(gbot, [ ]) % Fig. 10.4(d)
>> g = abs(g);
>> figure, imshow(g, [ ]) % Fig. 10.4(e)
>> T = max(g(:));
>> g = g >= T;
>> figure, imshow(g) % Fig. 10.4(f)

```

在图 10-4(b)中, 比灰色背景暗一点的阴影与负值相对应。在 $+45^\circ$  方向上有两个主要线段, 一个在左上方, 另一个在右下方(图10-4(c)和图10-4(d)显示了这两个区域的放大的片段)。注意, 在图 10-4(d)中, 直线部分比图 10-4(c)中的线段要亮得多。原因是: 图 10-4(a)中右下方的部分只有一个像素宽, 但左上方的部分不是。模板的响应对一个像素宽的部分比较强烈。

图 10-4(e)显示了图 10-4(b)的绝对值。因为我们对最强的响应感兴趣, 所以令  $T$  等于图像中的最大值。图 10-4(f)以白色显示了满足条件  $g \geq T$  的点, 其中的  $g$  是图 10-4(e)中的图像。图中的孤立点也是对模板同样响应强烈的点。在原始图像中, 这些点以及它们紧邻的点都是以这样的方法进行导向的。在那些孤立的位置, 模板将产生最大响应。这些孤立的点可以由图 10-1 中的模板来检测, 然后删除, 也可以使用前一章讨论的数学形态学算子来删除。

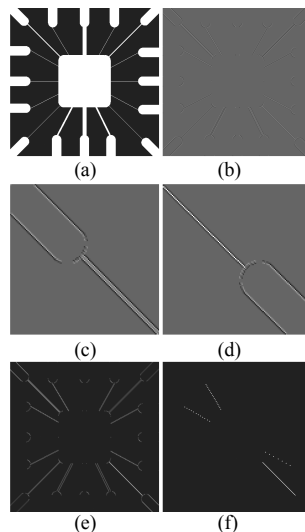


图 10-4 (a) 连线模板图像; (b) 用图 10-3 中的  $+45^\circ$  线处理后的结果; (c) (b) 中左上角的放大效果; (d) (b) 中右下角的放大效果; (e) (b) 的绝对值; (f) 所有的点(白色), 这些点的值满足  $g \geq T$ , 在这里,  $g$  是(e)中的图像((f)中的点是为了便于观看而被放大了)

### 10.1.3 使用函数 edge 的边缘检测

虽然点检测和线检测在图像分割的任何讨论中的确很重要, 但是到目前为止, 边缘检测最通用的方法是检测亮度的不连续。这样的不连续是用一阶和二阶导数来检测的。图像处理中选择的一阶导数是在 6.6.1 节中定义过的梯度。为方便起见, 我们在这里适当地重写相关公式。二维函数  $f(x, y)$  的梯度定义为一个向量:

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

这个向量的幅值是:

$$|\nabla f| = \text{mag}(\nabla f) = [g_x^2 + g_y^2]^{1/2} = [(\partial f / \partial x)^2 + (\partial f / \partial y)^2]^{1/2}$$

为了简化计算，有时候这个数值采取省略平方根计算的方法：

$$\nabla f \approx g_x^2 + g_y^2$$

或者取绝对值：

$$\nabla f \approx |g_x| + |g_y|$$

这些近似值仍然具有微分性质；也就是说，它们在恒定亮度区域中的值为零，而且它们的值与可变亮度区域中的亮度变化程度有关。

在实践中，通常用梯度的幅值或近似值来简单地作为“梯度”。

梯度向量的基本性质是：梯度向量指向 $(x,y)$ 坐标处 $f$ 的最大变化率方向。最大变化率处发生的角度是：

$$\alpha(x,y) = \tan^{-1} \left( \frac{g_y}{g_x} \right)$$

用函数 `edge` 估计  $g_x$  和  $g_y$  的方法在本节稍后讨论。图像处理中的二阶导数通常用 2.5.1 节中介绍的拉普拉斯来计算。回忆一下，二维函数  $f(x,y)$  的拉普拉斯由二阶微分构成：

$$\nabla^2 f(x,y) = \frac{\partial^2 f(x,y)}{\partial x^2} + \frac{\partial^2 f(x,y)}{\partial y^2}$$

拉普拉斯自身很少被直接用作边缘检测，因为作为二阶导数，拉普拉斯对噪声的敏感性无法接受，它的幅度会产生双边缘，而且无法检测边缘方向。然而，正像本节稍后讨论的那样，当与其他边缘检测技术组合使用时，拉普拉斯是强有力的补充方法。例如，虽然双边缘不适合直接做边缘检测，但是这个性质可通过寻找双边缘间的零交叉来为边缘定位。

以前边的讨论为背景，边缘检测的基本概念是用以下两个基本准则之一，在图像中寻找亮度发生快速变化的位置：

- 寻找亮度的一阶导数的幅度比指定阈值大的地方。
- 寻找亮度的二阶导数中有零交叉的位置。

图像处理工具箱函数 `edge` 提供了一些以刚才讨论过的准则为基础的边缘估计器。对于一些估计器来说，能否作为边缘检测器取决于对水平、垂直或两者是否敏感。这个函数的一般语法是：

```
[g, t]=edge(f, 'method', parameters)
```

其中， $f$  是输入图像，`method` 是表 10-1 中列出方法中的一种，`parameters` 是下边说明的附加参数。输出  $g$  是在  $f$  中被检测到的边缘点的位置为 1，而在其他地方为 0 的逻辑数组。参数  $t$  是可选择的；由 `edge` 给出阈值，以决定哪个梯度值足够大到可以被称作边缘点。

### 1. Sobel 边缘检测算子

一阶导数被近似为数字的差。Sobel 边缘检测算子用下边的  $3 \times 3$  邻域的行和列之间的离散差计算梯度(见图 10-5(a))，其中，每一行和每一列的中心像素用 2 来加权以提供平滑(Gonzalez 和 Woods[2008])：

$$\nabla f = [g_x^2 + g_y^2]^{1/2} = \{[(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)]^2 + [(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)]^2\}^{1/2}$$

其中， $z$  是亮度。如果在位置  $(x,y)$  处， $\nabla f \geq T$ ，那么在该位置的像素是边缘像素，其中的  $T$  是指定的阈值。

从 2.5.1 节的讨论中我们知道，Sobel 边缘检测可以通过图 10-5(b)左边的模板滤波一幅图像  $f$  来实现(用 `imfilter` 函数)，然后，用另一个模板再对图像  $f$  滤波，对每一幅滤波后的图像的像素值再平方，把两个值加起来，计算平方根值。

类似的解释可用于表 10-1 中的第 2 和第 3 项。另外，`edge` 函数只是将前边的操作简单打包为函数来调用，并且增加其他的特征，例如接受一个阈值或自动地决定阈值。除此之外，`edge` 包含不可能直接用 `imfilter` 实现的边缘检测技术。

Sobel 检测算子的一般调用语法是：

```
[g,t] = edge(f, 'sobel', T, dir)
```

其中  $f$  是输入图像， $T$  是指定的阈值，`dir` 是指定的检测边缘的首选方向：'horizontal'、'vertical'或'both'(默认值)。

正如先前表明的那样， $g$  是在被检测到的边缘位置处为 1、而在其他位置为 0 的逻辑图像。

输出的参数  $t$  是可选择的，是 `edge` 使用的阈值。如果  $T$  被指定了值，那么  $t=T$ ；否则，如果  $T$  没有被赋值(或是空的[])，`edge` 令  $t$  等于自动决定的阈值，并用于边缘检测。

在输出参量列表中要包括  $t$  的主要原因之一是：为了得到阈值的初始值，它可以改进并且在后续调用中传递给函数。如果使用语法 `g=edge(f)` 或 `[g,t]=egde(f)`，`edge` 函数将使用 Sobel 检测算子作为默认值。

表 10-1 函数 `edge` 中可用的边缘检测算子

边缘检测算子	描 述
Sobel	用图 10-5(b)所示的 Sobel 近似导数寻找边缘
Prewitt	用图 10-5(c)所示的 Prewitt 近似导数寻找边缘
Roberts	用图 10-5(d)所示的 Roberts 近似导数寻找边缘
LoG	在使用高斯滤波器的拉普拉斯滤波 $f(x,y)$ 后，通过寻找零交叉来发现边缘
零交叉	用指定的滤波器滤波 $f(x,y)$ 之后，寻找零交叉以发现边缘
Canny	通过寻找 $f(x,y)$ 的梯度的局部最大值来发现边缘。梯度由高斯滤波器的微分来计算。该方法使用两个阈值检测强的和弱的边缘，如果它们被连接到强边缘，那么在输出中只包含弱边缘。因此，这种方法更适合用于检测真实的弱边缘

2. Prewitt 边缘检测算子

Prewitt 边缘检测算子使用图 10-5(c)中的模板数字地近似一阶导数  $g_x$  和  $g_y$ 。一般调用语法是：

```
[g,t] = edge(f, 'prewitt', T, dir)
```

这个函数的参数和 Sobel 参数相同。Prewitt 检测算子相比 Sobel 检测算子在计算上要简单一点，但是比较容易产生噪声。

3. Roberts 边缘检测算子

Roberts 边缘检测算子使用图10-5(d)中的模板以相邻像素的差数字地近似一阶引导数。一般调用语法是：

```
[g,t] = edge(f, 'roberts', T, dir)
```

这个函数的参数和 Sobel 参数相同。Roberts 检测算子是数字图像处理中最古老的边缘检测算子中的一种，如图 10-5(d)所示，并且也是最简单的一种。因为在部分功能上有限制，所以这种检测算子的使用明显少于其他几种算子(比如，Roberts 检测算子是非对称的，而且不能检测诸如 45° 倍数的边缘)。然而，在简单和速度为主导因素的情况下，Roberts 检测算子还是经常用在硬件实现方面。

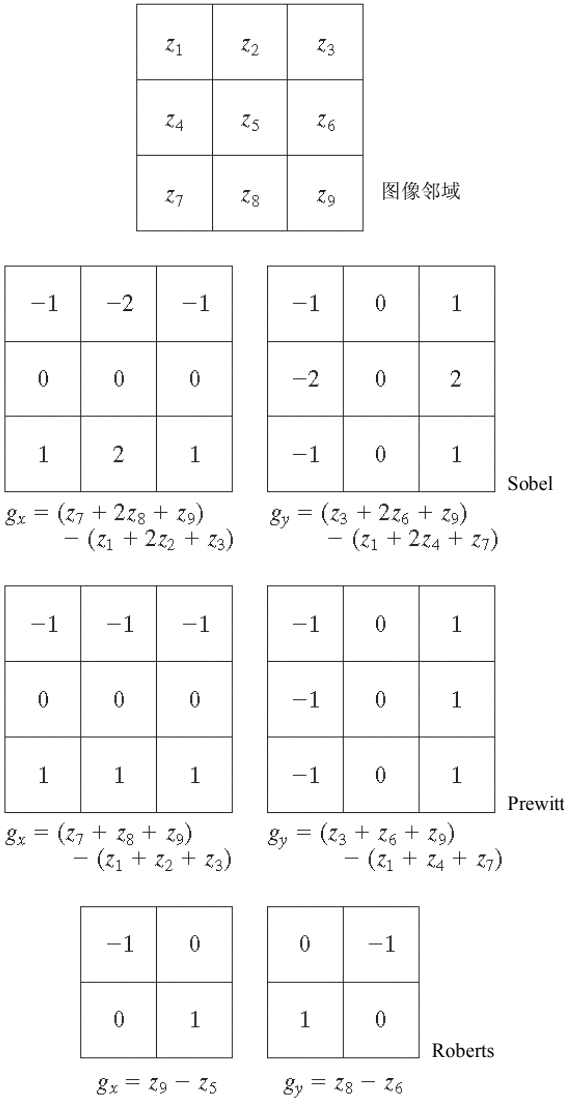


图 10-5 边缘检测算子模板以及它们实现的一阶导数

#### 4. LoG 检测算子

考虑高斯函数：

$$G(x, y) = e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

其中的  $\sigma$  是标准差。这是平滑函数，如果和图像卷积，将会使图像变模糊。模糊的程度由  $\sigma$  的值决定。这个函数的拉普拉斯算法是：

$$\nabla^2 G(x, y) = \frac{\partial^2 G(x, y)}{\partial x^2} + \frac{\partial^2 G(x, y)}{\partial y^2} = \left[ \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right] e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

出于明显的原因，这个函数被称为 LoG。因为求二阶导数是线性操作，所以用  $\nabla^2 G(x, y)$  卷积(滤波)这幅图像与先用平滑函数对图像卷积，再对结果进行拉普拉斯计算的结果是一样的。这是 LoG 检测算子的关键概念。

我们用  $\nabla^2 G(x, y)$  卷积图像，这么做得得到两个效果：平滑图像(因而减少了噪声)；计算拉普拉斯，从而产生双边缘图像。然后在双边缘之间定位由发现的零交叉组成的边缘。

LoG 检测算子的一般调用语法是：

```
[g,t] = edge(f, 'log', T, sigma)
```

其中的 sigma 是标准差，其他参数和前边解释过的一样。sigma 的默认值是 2。和以前一样，edge 忽略一切不比 T 强的边缘。如果 T 值没有给出或为空[]，edge 会自动选值。将 T 设置为 0，将产生封闭的轮廓，这是我们熟知的 LoG 方法的典型特征。

#### 5. 零交叉检测算子

这种检测算子基于的概念与 LoG 方法相同，但是卷积使用特殊的滤波函数 H 来完成，调用语法为：

```
[g,t] = edge(f, 'zerocross', T, H)
```

其他参数和 LoG 解释的一样。

#### 6. Canny 检测算子

Canny 检测算子(Canny[1986])是 edge 函数中最强的边缘检测算子。总结如下：

- 1) 图像用指定了标准差  $\sigma$  的高斯滤波器来平滑，用以减少噪声。
- 2) 局部梯度  $[g_x^2 + g_y^2]^{\frac{1}{2}}$  和边缘方向  $\tan^{-1}(g_x / g_y)$  在每一点都计算。表 10-1 中前三项技术的任意一项都可以用来计算导数。边缘点被定义为梯度方向上局部强度最大的点。
- 3) 在 2) 中决定的边缘点在梯度幅度图像上给出脊。然后，算法则追踪所有脊的顶部，设置所有的不在脊的顶部的像素为零。因此，在输出中给出一条细线，这是众所周知的非最大值抑制处理。脊像素使用称为“滞后阈值”的技术进行阈值处理，这种技术以使用两个阈值为基础，即  $T_1$  和  $T_2$ ，且  $T_1 < T_2$ 。值大于  $T_2$  的脊像素称作强边缘像素， $T_1$  和  $T_2$  之间的脊像素称作弱边缘

像素。

4) 最后，算法用合并 8 连接的弱像素点到强像素点的方法执行边缘连接。

Canny 检测算子的语法是：

```
[g,t] = edge(f, 'canny', T, sigma)
```

在这里， $T$  是向量。 $T=[T_1, T_2]$ ，包含在前边步骤 3) 的两个阈值， $\sigma$  是平滑滤波器的标准差。如果  $t$  包括在输出参量中， $t$  就是二元矢量，其中包含该算法用到的两个阈值。语法中的其余参数和其他方法中解释的一样，包括：如果  $T$  没有指定，就自动计算阈值。 $\sigma$  的默认值是 1。

### 例 10.3 Sobel 边缘检测算子的使用

我们可以用下列指令提取并显示图 10-6(a) 中图像  $f$  的垂直边缘：

```
>> [gv, t] = edge(f, 'sobel' , 'vertical');
>> imshow(gv)
>> t
t =
    0.0516
```

如图 10-6(b) 所示，结果中的主要边缘是垂直边缘(倾斜的边缘有垂直和水平分量，所以也能被检测到)。可以指定较高的阈值，从而把较弱的边缘去掉。

例如，图 10-6(c) 是用下边的指令产生的：

```
>> gv = edge(f, 'sobel' ,0.15, 'vertical');
```

在指令中，使用相同的  $T$  值：

```
>> gboth = edge(f, 'sobel' ,0.15)
```

这会产生图 10-6(d) 中的结果，突出显示了垂直和水平边缘。 $\text{edge}$  函数不能在  $\pm 45^\circ$  上计算 Sobel 边缘。为了计算这些边缘，我们需要特定的模板并使用  $\text{imfilter}$  函数。例如，图 10-6(e) 由下面的指令产生：

```
>> wneg45 = [-2 -1 0; -1 0 1; 0 1 2]
wneg45 =
    -2    -1     0
    -1     0     1
     0     1     2

>> gneg45 = imfilter(tofloat(f), wneg45, 'replicate');
>> T = 0.3*max(abs(gneg45(:)));
>> gneg45 = gneg45 >= T;
>> figure, imshow(gneg45);
```

图 10-6(e) 中最强的边缘是面向  $-45^\circ$  角的边缘。同样，通过模板  $wpos45 = [0 \ 1 \ 2; -1 \ 0 \ 1; -2 \ -1 \ 0]$ ，并使用同样的命令序列，就能得到面向  $+45^\circ$  角的边缘，如图 10-6(f) 所示。

在  $\text{edge}$  函数中使用 'prewitt' 和 'roberts' 选项并遵循与刚刚说明过的 Sobel 边缘检测算子相同的过程。

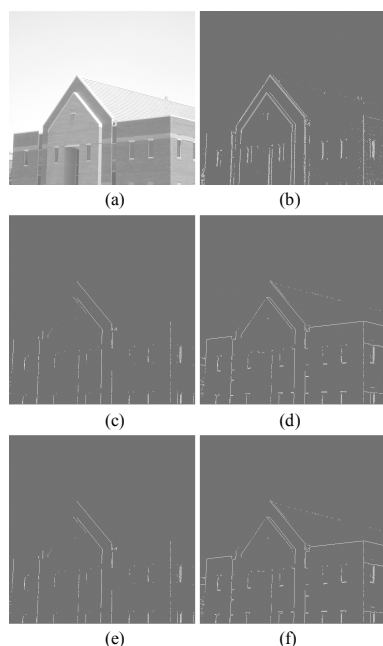


图 10-6 (a) 原始图像; (b) 使用自动决定阈值的垂直 Sobel 模板并经 edge 函数处理后的结果; (c) 使用指定阈值的结果; (d) 使用指定阈值决定垂直和水平边缘的结果; (e) 使用 imfilter 函数并用指定的模板及阈值计算 $-45^\circ$ 边缘的结果; (f) 使用函数 imfilter 并用指定的模板及阈值计算 $+45^\circ$ 边缘的结果

#### 例 10.4 Sobel、LoG 和 Canny 边缘检测算子的比较

在这个例子中, 我们比较 Sobel、LoG 和 Canny 边缘检测算子的相关性能。目的是通过提取图 10-6(a)中建筑图像  $f$  的主要边缘特征, 产生干净清晰的边缘图, 并去掉不相关的细节, 比如砖墙和屋顶的纹理。在本次讨论中, 我们感兴趣的主要边缘是建筑的角落、窗户、由亮的砖结构形成的门口和门本身、屋顶线以及围绕建筑且距离地面三分之二的混凝土带。图 10-7 左列显示了使用默认的 'sobel'、'log' 和 'canny' 选项得到的边缘图像。

```
>> f = tofloat(f);
>> [gSobel_default, ts] = edge(f, 'sobel'); % Fig. 10.7(a)
>> [gLoG_default, tlog] = edge(f, 'log'); % Fig. 10.7(c)
>> [gCanny_default, tc] = edge(f, 'canny'); % Fig. 10.7(e)
```

在输出参量中, 由前面计算得到的阈值是  $ts=0.074$ 、 $tlog=0.0025$  和  $tc=[0.019, 0.047]$ 。对于 'log' 和 'canny' 选项, sigma 的默认值分别是 2.0 和 1.0。除了 Sobel 图像之外, 通过默认值计算得出的图像与想要得到的清晰边缘图相差较远。由默认值开始, 每个选项的参数分别随早些时候提到的显示主要特性的目标而变化, 并尽可能减少不必要的细节。图 10-7 右列的图像是由以下指令得到的:

```
>> gSobel_best = edge(f, 'sobel', 0.05); % Fig. 10.7(b)
>> gLoG_best = edge(f, 'log', 0.003, 2.25); % Fig. 10.7(d)
>> gCanny_best = edge(f, 'canny', [0.04 0.10], 1.5); % Fig. 10.7(f)
```

如图 10-7(b)所示, Sobel 得出的结果与我们想要的水泥带和门口的左边缘相差还较远。在图 10-7(d)中, LoG 的结果相比 Sobel 的结果要好一些, 比 LoG 的默认值得出的结果要好得多。



但是，门口的左边缘没有检测出来，建筑物周围水泥带的两个边也还是不够清晰。

Canny 得出的结果(图10-7(f))到目前为止要远远好于前两种。特别注意：门口的左边缘已被清晰检测到，还有混凝土带和其他细节，比如门口上方屋顶的通风栅格。除了检测到要求的特征之外，Canny 检测算子还产生了最清晰的边缘图。

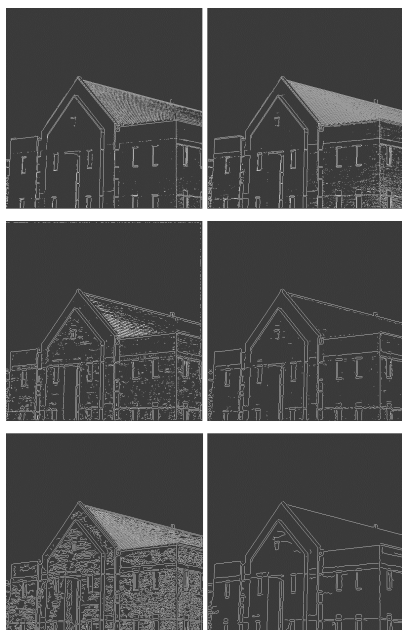


图 10-7 左列，使用 Sobel、LoG、Canny 边缘检测算子默认选项的结果；右列，当需要减少不相关的细节时，显示图 10-6(a)中原始图像的主要特征，Canny 边缘检测算子能产生最好的结果

## 10.2 使用霍夫变换的线检测

在理想情况下，10.1 节讨论的方法应该只产生位于边缘上的像素。实际上，得到的像素因为噪声，以及不均匀照明引起的边缘断裂和杂散的亮度不连续而难以得到完全的边缘特性。

因而，典型的边缘检测算法紧接着用连接过程把像素组装成有意义的边缘。一种寻找并连接图像中线段的方法是霍夫变换(Hough[1962])。

### 10.2.1 背景

给定图像(典型的二值图像)中的  $n$  个点，假如我们想要寻找位于直线上所有点的子集，可能的解决方法就是先找到由每一对点决定的所有线，然后寻找接近特殊线的所有点的子集。在此过程中，需要找到  $n(n-1)/2 \sim n^2$  条线，然后进行  $n(n(n-1))/2 \sim n^3$  次每一点与所有线的比较。这种方法运算很繁琐，一般是不用的。

另一种解决方法是使用霍夫变换，我们考虑点  $(x_i, y_i)$  以及所有通过这个点的线，有无穷多的线通过点  $(x_i, y_i)$ 。其中，针对  $a$  和  $b$  的一些值，满足斜截式  $y_i = ax_i + b$  的所有线都通过该点。该公式可以写为  $b = -ax_i + y_i$ ，并且考虑  $ab$  平面(也称为参数空间)对固定点  $(x_i, y_i)$  得到一条线的方程。此外，第二个点  $(x_j, y_j)$  也有这样一条在参数空间中与之相关的线，这条线和与  $(x_i, y_i)$  相关的线相交于点  $(a', b')$ ，其中， $a'$  是斜率， $b'$  是在  $xy$  平面上包含点  $(x_i, y_i)$  和  $(x_j, y_j)$  的线的截距。事实上，

在参数空间中，这条线包含的所有点都有相交于 $(a', b')$ 点的直线。图 10-8 说明了这些概念。

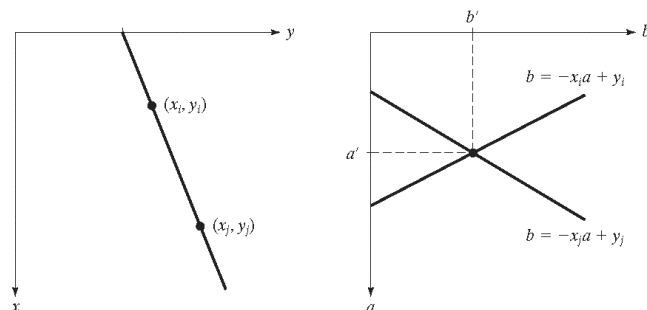


图 10-8 (a)  $xy$  平面; (b) 参数空间

原则上，与图像上的点 $(x_i, y_i)$ 相对应的参数空间中的所有线都可以绘制出来，而图像上的线可以通过很多参数空间的线的交叉来识别。然而，用这种方法的实际困难是： $a$ (线的斜率)接近无限大，也就是接近垂直方向。这个困难的解决办法使用线的法线表示法：

$$x \cos \theta + y \sin \theta = \rho$$

图 10-9(a)说明了参数  $\rho$  和  $\theta$  的几何解释。水平线的  $\theta=0$ ， $\rho$  等于正的  $x$  截距。相似的，垂直线的  $\theta=90^\circ$ ， $\rho$  等于正的  $y$  截距。或者， $\theta=-90^\circ$ ， $\rho$  等于负的  $y$  截距。

图 10-9(b)中的每一条正弦曲线表示通过特定点 $(x_i, y_i)$ 的一族直线。交叉点对应于通过 $(x_i, y_i)$ 和 $(x_j, y_j)$ 的线。

霍夫变换在计算上的诱人之处是把  $\rho\theta$  空间再细分为所谓的累加单元，正如图 10-9(c)中说明的那样，其中的  $(\rho_{\min}, \rho_{\max})$  和  $(\theta_{\min}, \theta_{\max})$  是预期的参数值范围。一般来说，值的最大范围是  $-90^\circ \leq \theta \leq 90^\circ$  和  $-D \leq \rho \leq D$ ，其中的  $D$  是图像中两个角之间的最远距离。

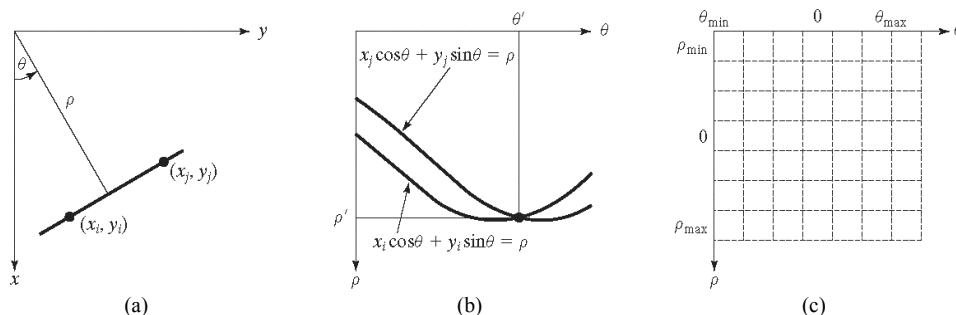


图 10-9 (a)  $xy$  平面上线的参数化; (b)  $\rho\theta$  平面上的正弦曲线; 交叉点 $(\rho', \theta')$ 对应连接 $(x_i, y_i)$ 和 $(x_j, y_j)$ 的线的参数;  
(c) 把  $\rho\theta$  平面划分为累加单元

在坐标 $(i, j)$ 的单元位置，累加器的值是  $A(i, j)$ ，对应于参数空间坐标 $(\rho_i, \theta_j)$ 的正方形。最初，这些单元位置为零。然后，对于每个图像平面上的非背景点 $(x_k, y_k)$ (就是  $xy$  平面)，我们令  $\theta$  等于在  $\theta$  轴上允许的细分值，并通过公式  $\rho = x_k \cos \theta + y_k \sin \theta$  解出相应的  $\rho$  值。然后，得到的  $\rho$  值四舍五入为最接近的  $\rho$  轴上允许的单元值。相应的累加器单元增加一个增量。在这个过程的最后，累加单元  $A(i, j)$  中的值  $Q$  就意味着  $xy$  平面上位于线  $x \cos \theta_j + y \sin \theta_j = \rho_i$  上的点有  $Q$  个。在  $\rho\theta$  平面上，细分的数目决定了这些点的共线的精确度。累加器数组在工具箱中叫做霍夫变换矩阵，简称霍夫变换。

## 10.2.2 与霍夫变换有关的工具箱函数

图像处理工具箱提供了三个与霍夫变换有关的函数。函数 `hough` 实现了前面讨论的概念，函数 `houghpeaks` 寻找霍夫变换的峰值(累加单元的高计数)，函数 `houghlines` 以来自其他两个函数的结果为基础在原始图像中提取线段。

### 1. 函数 `hough`

函数 `hough` 支持任意的默认语法：

```
[H, theta, rho] = hough(f)
```

还支持完整的语法形式：

```
[H, theta, rho] = hough(f, 'ThetaRes', val1, 'RhoRes', val2)
```

其中， $H$  是霍夫变换矩阵， $\theta$  (以度计)和  $\rho$  是  $\rho$  和  $\theta$  值向量，在这些值上产生霍夫变换。输入  $f$  是二值图像， $val1$  是 0 到 90 的标量，指定了沿  $\theta$  轴霍夫变换的间距(默认是 1)， $val2$  是  $0 < val2 < \text{hypot}(\text{size}(f,1), \text{size}(f,2))$  的实标量，指定了沿  $\rho$  轴的霍夫变换的间隔(默认是 1)。

#### 例 10.5 霍夫变换的说明

在这个例子中，我们用简单的合成图像来说明 `hough` 函数的机理：

```
>> f = zeros(101, 101);
>> f(1, 1) = 1; f(101, 1) = 1; f(1, 101) = 1;
>> f(101, 101) = 1; f(51, 51) = 1;
```

图 10-10(a)显示了我们的测试图像，下面使用默认值计算并显示霍夫变换的结果：

```
>> H = hough(f)
>> imshow(H, [])
```

图 10-10(b)显示了结果，以熟悉的方法使用 `imshow` 函数来显示。在带有标度轴的较大图中显现霍夫变换常常更有用。

在接下来的代码片段中，我们调用带有三个参数的 `hough` 函数。然后把向量  $\theta$  和  $\rho$  作为附加输入参量传递给 `imshow`，从而控制水平轴和垂直轴的标度。我们还要把 'InitialMagnification' 选项传递给带有值 'fit' 的 `imshow` 函数，因此，整个图像将被强迫在图形窗口中进行装配。`axis` 函数被用来打开轴标记，并使其显示填充图的矩形框。最后，`xlabel` 和 `ylabel` 函数(见 2.3.1 节)用希腊字母 LaTeX 字体符号在轴上标值：

```
>> [H, theta, rho] = hough(f);
>> imshow(H, [], 'XData', theta, 'YData', rho, 'InitialMagnification', 'fit')
>> axis on, axis normal
>> xlabel('\theta'), ylabel('\rho')
```

图 10-10(c)显示了标上值之后的结果。三条曲线(直线也可考虑为曲线)在  $\pm 45^\circ$  处的交点指出： $f$  中有两组三个共线的点。两条曲线在  $(\rho, \theta) = (0, -90)$ 、 $(-100, -90)$ 、 $(0, 0)$  和  $(100, 0)$  处的交点指出：有 4 组位于垂直线和水平线上的共线点。

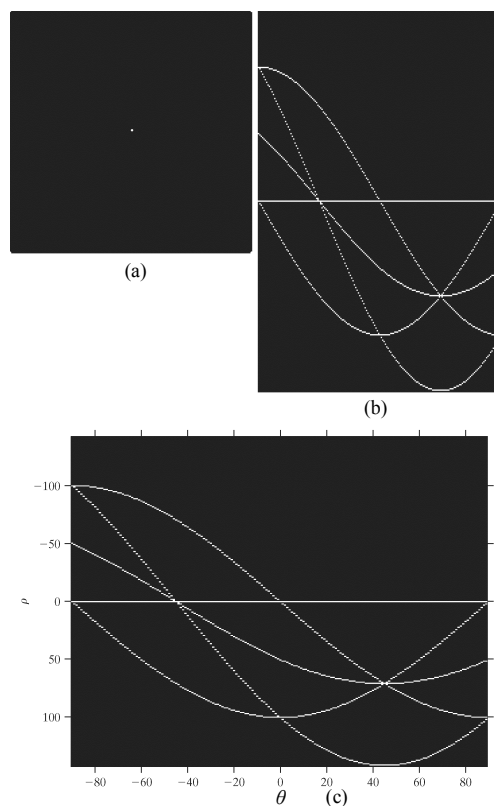


图 10-10 (a) 有 5 个点的二值图像(4 个点在角上); (b) 使用 `imshow` 函数显示的霍夫变换; (c) 另一种带有标度轴的霍夫变换((a)中的点已被放大以便于观看)

## 2. 函数 `houghpeaks`

线检测和连接用的霍夫变换的第一步是用高的计数寻找累加单元(工具箱文本把高的计数单元作为峰值)。因为存在霍夫变换参数空间中的量化和典型图像的边缘并不是很完美的直线这样的事实,霍夫变换的峰值倾向于相比霍夫变换单元更多。函数 `houghpeaks` 用任意默认语法来寻找指定的峰值数:

```
peaks = houghpeaks(H, NumPeaks)
```

或者使用完整的语法形式:

```
peaks = houghpeaks(..., 'Threshold', val1, 'NHoodSize', val2)
```

其中,“...”指来自默认语法和 `peaks` 的输入是持有峰值行和列坐标的  $Q \times 2$  大小的矩阵。 $Q$  的范围是 0 到 `NumPeaks`,  $H$  是霍夫变换矩阵。参数 `val1` 是非负的标量,指定了  $H$  中的什么值被考虑为峰值; `val1` 可以从 0 到 `Inf` 变化,默认值是  $0.5 * \max(H(:))$ 。参数 `val2` 是奇整数的两元素矢量,指定量围绕峰值的邻域大小。当鉴别出峰值之后,邻域中的元素被置为 0。默认是由最小奇数值组成的两元素矢量大于或等于  $\text{size}(H)/50$ 。这个过程的基本思想是:通过把发现峰值的直接邻域中的霍夫变换单元置 0 来清理峰值。我们在例 10.6 中说明函数 `houghpeaks`。

### 3. 函数 houghlines

一旦一组候选的峰值在霍夫变换中被识别出来,如果存在与这些峰值相关的有意义的线段,剩下的就是决定线的起始点和终点。函数 `houghlines` 用默认的语法执行这个任务:

```
lines = houghlines(f, theta, rho, peaks)
```

或者使用完整的语法形式:

```
lines = houghlines(..., 'FillGap', val1, 'MinLength', val2)
```

其中, `theta` 和 `rho` 是来自函数 `hough` 的输出, `peaks` 是函数 `houghpeaks` 的输出。输出 `lines` 是结构数组,长度等于找到的线段数。结构中的每个元素可以看成一条线,并含有下列字段:

- `point1`: 两元素向量 `[r1, c1]`, 指定了线段终点的行列坐标。
- `point2`: 两元素向量 `[r2, c2]`, 指定了线段其他终点的行列坐标。
- `theta`: 与线相关的霍夫变换的以度计量的角度。
- `rho`: 与线相关的霍夫变换的  $\rho$  轴位置。

其他参数如下:

`val1` 是正的标量,指定了与相同的霍夫变换相关的两条线段的距离。当两条线段之间的距离小于指定的值时,函数 `houghlines` 把线段合并为一条线段(默认的距离是 20 个像素)。参数 `val2` 是正的标量,指定合并的线是保留还是丢弃。如果合并的线比 `val2` 指定的值短,就丢弃(默认值是 40)。

#### 例 10.6 用霍夫变换检测和连接线

在这个例子中,我们用函数 `hough`、`houghpeaks` 和 `houghlines` 寻找图 10-7(f)所示二值图像 `f` 的一组线段。首先,我们用比默认值更好的角间距(用 0.2 代替 1.0)计算和显示霍夫变换:

```
>> [H, theta, rho] = hough(f, 'ThetaResolution', 0.2);
>> imshow(H, [], 'XData', theta, 'YData', rho, 'InitialMagnification', 'fit')
>> axis on, axis normal
>> xlabel('\theta'), ylabel('\rho')
```

下一步,我们用函数 `houghpeaks` 寻找 5 个有意义的霍夫变换的峰值:

```
>> peaks = houghpeaks(H, 5);
>> hold on
>> plot(theta(peaks(:, 2)), rho(peaks(:, 1)), ...
        'linestyle', 'none', 'marker', 's', 'color', 'w')
```

前边的操作计算和显示霍夫变换,并添加使用函数 `houghpeaks` 的默认设置寻找到的 5 个峰值位置。图 10-11(a)显示了结果。例如,最左边较小的方形确定与房顶相关的累加单元,以工具箱的角度作为参考倾向于近似  $-74^\circ$ , 在图 10-9(a)中是  $-16^\circ$ 。最后,我们使用函数 `houghlines` 寻找和连接线段,用函数 `imshow`、`hold on` 和 `plot` 在原始的二值图像上添加线段:

```
>> lines = houghlines(f, theta, rho, peaks);
>> figure, imshow(f), hold on
>> for k = 1:length(lines)
xy = [lines(k).point1 ; lines(k).point2];
plot(xy(:,1), xy(:,2), 'LineWidth', 4, 'Color', [.8 .8 .8]);
end
```

图 10-11(b)显示了使用检测到的叠加了较粗灰线的线段得到的结果。

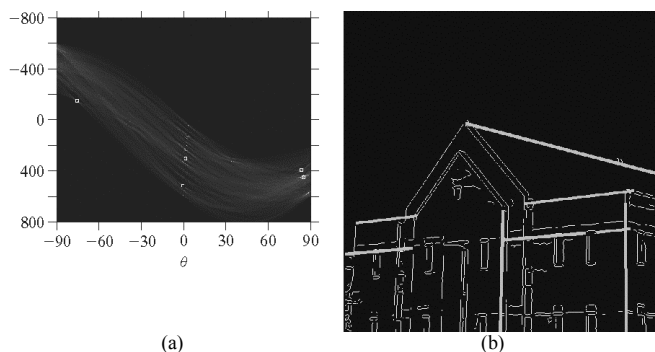


图 10-11 (a) 带有已选择的 5 个峰值位置的霍夫变换; (b) 与霍夫变换峰值对应的线段

## 10.3 阈值处理

由于直观性和实现的简单性,图像的阈值处理在图像分割应用中享有核心地位。我们在前面章节的讨论中已经用过阈值处理。在本节,我们讨论自动选择阈值的方法,此外还将考虑一种根据局部图像的性质来改变阈值的方法。

### 10.3.1 基础知识

假设图 10-12(a)中的灰度直方图对应于暗背景上由亮物体组成的图像  $f(x,y)$ , 这样, 目标和背景像素拥有的灰度级可分为两个占主导地位的模式。从背景提取目标的很显然方法是选取阈值  $T$  来分离这两个模式。然后, 满足  $f(x,y) \geq T$  的任何图像点被称为物体点, 其他的点称为背景点(反过来, 在亮背景上的暗物体也是一样)。阈值处理后的图像  $g(x,y)$  被定义为:

$$g(x,y) = \begin{cases} a & f(x,y) > T \\ b & f(x,y) \leq T \end{cases}$$

标注为  $a$  的像素对应目标, 标注为  $b$  的像素对应背景。通常为了方便, 令  $a=1$ (白),  $b=0$ (黑)。  $T$  是常数时, 适用于整幅图像, 上述公式用于全局阈值处理。当阈值  $T$  在一幅图像上变化时, 我们用“可变阈值”这一术语。术语“局部阈值处理”或“区域阈值处理”还用于表示一幅图像中在任何点  $(x,y)$  处,  $T$  值的变化依靠  $(x,y)$  邻域的特性(例如邻域像素的平均灰度)。如果  $T$  依赖空间坐标本身, 可变阈值常常叫做动态的或自适应的阈值处理。这些术语的使用并不普遍, 并且在有关图像处理的文献中你可能会看到在交替地使用它们。

图 10-12(b)显示了一个更加困难的阈值处理问题, 其中包含 3 个相应的直方图模式, 例如暗背景中亮物体的两种类型。在这里, 如果  $f(x,y) \leq T_1$ , 多(双)阈值处理将把  $(x,y)$  处的像素分属为背景; 如果  $T_1 < f(x,y) \leq T_2$ , 就分属于物体类; 如果  $f(x,y) > T_2$ , 就分属于另一个物体。也就是说, 分割的图像由下式决定:

$$g(x,y) = \begin{cases} a & f(x,y) > T_2 \\ b & T_1 < f(x,y) \leq T_2 \\ c & f(x,y) \leq T_1 \end{cases}$$

其中,  $a$ 、 $b$ 、 $c$  是三个不同的灰度值。要求多于两个阈值的分割是很难解决的(常常是不可

能的), 并且好的结果通常使用其他方法得到。例如在 10.3.6 和 10.3.7 节讨论的可变阈值处理, 或在 10.4 节讨论的区域生长方法。

基于前边的讨论, 我们得出如下结论: 灰度阈值处理的成功直接关系到直方图模式下可分离的谷的宽度和深度。由此, 影响谷的特性的关键因素是: 1) 峰间的分离度(峰分开得越远, 分离模式的可能性越好); 2) 图像中的噪声内容(模式随噪声增加而加宽); 3) 物体和背景的相对大小; 4) 光源的均匀性; 5) 图像的反射特性的均匀性。至于这些因素怎样影响阈值处理方法的详细讨论, 见 Gonzalez 和 Woods 的著作[2008]。

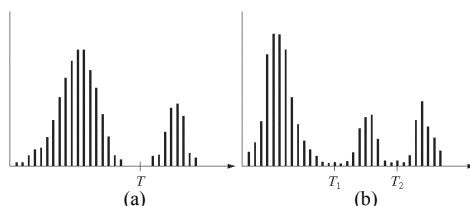


图 10-12 可以分割的灰度直方图: (a) 使用单阈值; (b) 使用双阈值。它们分别是单峰和双峰直方图

### 10.3.2 基本全局阈值处理

选取阈值的一种方法就是图像直方图的视觉检测。例如, 图10-12(a)中的直方图有两个截然不同的模式; 很容易选择阈值  $T$  来分开它们。选择  $T$  的另一个方法是反复实验, 选取不同的阈值, 直到观测者觉得产生了较好的结果为止, 这在交互环境下特别有效。例如, 这种方法允许使用者通过 widget(图形控制)改变阈值, 就像游标一样, 可以立即看见结果。

通常, 在图像处理中首选的方法是使用一种能基于图像数据自动地选择阈值的算法, 为了自动选阈值, 下列迭代过程采用的就是这样的方法:

- (1) 针对全局阈值选择初始估计值  $T$ 。
- (2) 用  $T$  分割图像。这会产生两组像素:  $G_1$  由所有灰度值大于  $T$  的像素组成,  $G_2$  由所有灰度值小于等于  $T$  的像素组成。
- (3) 分别计算  $G_1$ 、 $G_2$  区域内的平均灰度值  $m_1$  和  $m_2$ 。
- (4) 计算出新的阈值:

$$T = \frac{1}{2}(m_1 + m_2)$$

- (5) 重复步骤(2)~(4), 直到在连续的重复中,  $T$  的差异比预先设定的参数  $\Delta T$  小为止。
- (6) 使用函数 `im2bw` 分割图像:

```
g = im2bw(f, T/den)
```

其中, `den` 是整数(例如一幅 8 比特图像的 255), 是  $T/\text{den}$  比率为 1 的数值范围内的最大值, 正如函数 `im2bw` 要求的那样。

在速度成为重要问题时, 参数  $\Delta T$  用于控制迭代次数。通常,  $\Delta T$  越大, 算法执行的迭代次数越少。这可以得到证明, 假如初始阈值在图像中的最大和最小灰度值之间选择(平均图像灰度对  $T$  来说是不错的选择), 那么算法在有限的步数内收敛。根据分割, 在涉及物体和背景的直方图模式之间有相当清楚的谷的情况下, 算法会工作得很好。我们以下边的例子来说明在 MATLAB 中如何执行这个过程。

**例 10.7 计算全局阈值**

刚才讨论的基本迭代方法可按如下方式来执行, 其中的  $f$  是图 10-13(a)中的图像:

```
>> count = 0;
>> T = mean2(f);
>> done = false;
>> while ~done
    count = count + 1;
    g = f > T;
    Tnext = 0.5*(mean(f(g)) + mean(f(~g)));
    done = abs(T - Tnext) < 0.5;
    T = Tnext;
end
>> count
count =
    2
>> T
T =
    125.3860
>> g = im2bw(f, T/255);
>> imshow(f) % Fig. 10.13(a).
>> figure, imhist(f) % Fig. 10.13(b).
>> figure, imshow(g) % Fig. 10.13(c).
```

算法仅需两步迭代就可收敛, 并且得到的阈值接近灰度级的中点。可以期待清晰的分割, 因为直方图中的模式之间有很宽的分开度。

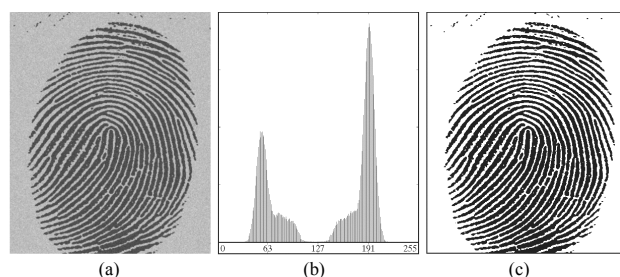


图 10-13 (a) 带噪声的指纹; (b) 直方图; (c) 用全局阈值分割的结果(为清楚起见, 边界是手工加上去的)。(原图像由美国国家标准和技术研究所提供)

**10.3.3 使用 Otsu's 方法的最佳全局阈值处理**

令一幅图像的直方图成分由下式表示:

$$p_q = \frac{n_q}{n} \quad q = 0, 1, 2, \dots, L-1$$

其中,  $n$  是图像中像素的总数,  $n_q$  是灰度级为  $q$  的像素数目,  $L$  是图像中所有可能的灰度级数(记住, 灰度级是整数)。现在, 假设阈值  $k$  已经选定了,  $C_1$  是一组灰度级为  $[0, 1, 2, \dots, k]$  的像素,  $C_2$  是一组灰度级为  $[k+1, \dots, L-1]$  的像素。Otsu's 为最佳方法, 在某种意义上, 选择阈值  $k$ , 最大类间方差  $\sigma_b^2(k)$  定义为:

$$\sigma_b^2(k) = P_1(k)[m_1(k) - m_g]^2 + P_2(k)[m_2(k) - m_g]^2$$

这里,  $P_1(k)$  是集合  $C_1$  发生的概率:



$$P_1(k) = \sum_{i=0}^k p_i$$

例如, 如果设置  $k=0$ , 那么拥有为  $k$  赋值的任何像素  $C_1$  集合的概率为 0, 类似的, 集合  $C_2$  发生的概率是:

$$P_2(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_1(k)$$

$m_1(k)$  和  $m_2(k)$  分别是集合  $C_1$  和  $C_2$  中像素的平均灰度。 $m_g$  是全局均值(整个图像的平均灰度):

$$m_g = \sum_{i=0}^{L-1} i p_i$$

此外, 直到灰度级  $k$  的平均灰度由下式给出:

$$m(k) = \sum_{i=0}^k i p_i$$

展开  $\sigma_b^2(k)$  的表达式, 并使用  $P_2(k) = 1 - P_1(k)$  这一事实, 我们可以把类间方差写成:

$$\sigma_b^2(k) = \frac{[m_g P_1(k) - m(k)]^2}{P_1(k)[1 - P_1(k)]}$$

这个式子在计算上稍微有效一些, 因为对于所有的  $k$  值来说, 只有两个参数  $m$  和  $P_1$  必须计算( $m_g$  只计算一次)。将类间方差最大化的想法是: 方差较大, 完全分割一幅图像的阈值将会更接近。注意, 这个最佳度量完全基于参数, 它可以直接从图像的直方图得到。另外, 因为  $k$  是处于范围  $[0, L-1]$  内的整数, 所以寻找  $\sigma_b^2(k)$  的最大值是很简单的事情。我们一步步通过  $k$  的  $L$  个可能值, 并在每一步计算方差。然后, 选择给出  $\sigma_b^2(k)$  最大值的  $k$ 。这个  $k$  就是最佳阈值。如果最大值不唯一, 那么所用的阈值是找到的所有最佳  $k$  值的平均。类间方差对总的图像灰度方差的比率, 是把图像灰度分为两类(也就是物体和背景)的可分性度量:

$$\eta(k) = \frac{\sigma_b^2(k)}{\sigma_g^2}$$

可用如下范围来显示:

$$0 \leq \eta(k^*) \leq 1$$

其中,  $k$  是最佳阈值。该度量可用于使恒定图像(像素完全不可以分成两类)达到最小值, 还可用于使二值图像(像素完全可分)达到最大值。

工具箱函数 `graythresh` 用于计算 Otsu's 阈值。语法是:

```
[T, SM] = graythresh(f)
```

在这里,  $f$  是输入图像,  $T$  是产生的阈值并被归一化到  $[0,1]$  中,  $SM$  是可分性度量。正如前面说明的那样, 函数 `im2bw` 用来分割图像。

**例 10.8** 对使用 Otsu's 方法和 10.3.2 节的基本全局阈值处理方法分割图像的比较

我们使用图 10-13(a)中的图像  $f$  来比较 Otsu's 方法和上一节的基本全局阈值处理方法:

```
>> [T, SM] = graythresh(f)
T =
    0.4902
SM =
```

```

0.9437
>> T*255
ans =
125

```

这个阈值与通过上一节的基本全局阈值处理算法得到的阈值几乎相同，因此可以期待相同的分割结果。注意，高的 SM 值说明灰度分成两类的可能性高。

图 10-14(a)(一幅聚合物细胞的图像，称为  $f_2$ ) 中存在更困难的分割任务。目标是从背景中分割出细胞的边界(图像中最亮的区域)。图像的直方图(见图 10-14(b))完全不是双峰的。因此，期望通过简单的算法达到适当的分割有一些困难。图 10-14(c) 中图像的获得过程与图 10-13(c) 相同。算法经一次迭代就收敛了，并得到了等于 169.4 的阈值  $T$ 。使用这个阈值：

```

>> g = im2bw(f2, T/255);
>> imshow(g)

```

可以得到图 10-14(c) 中的结果。

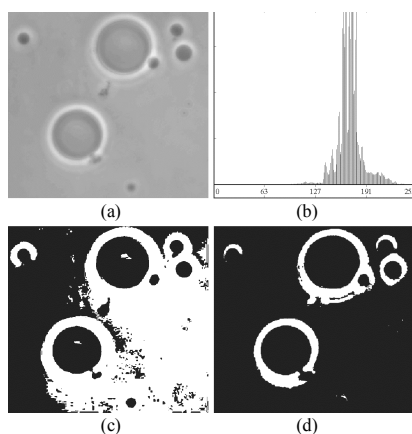


图 10-14 (a) 原始图像; (b) 直方图(高值被修剪为低值中的高亮细节); (c) 使用基本全局算法的分割结果; (d) 使用 Otsu's 算法得到的结果。(原图像由宾夕法尼亚大学的 Daniel A. Hammer 教授提供)

正如你看到的那样，分割并不成功。现在，我们用 Otsu's 方法分割图像：

```

>> [T, SM] = graythresh(f2);
>> SM
SM =
0.4662
>> T*255
ans =
181
>> g = im2bw(f2, T);
>> figure, imshow(g) % Fig. 10.14(d).

```

正如图 10-14(d) 中显示的那样，采用 Otsu's 方法的分割是有效的。尽管分离度的度量值相对较低，但聚合物细胞的边界以合理的准确性从背景中提取出来了。

类间方差的所有参数都以图像的直方图为基础。正如不久你将要看到的那样，存在一种应用，其中可以用直方图而不是图像来计算 Otsu's 阈值，和函数 `graythresh` 一样。下面的自定义函数计算给定图像直方图的  $T$  和  $SM$ ：

```

function [T, SM] = otsuthresh(h)
%OTSUTHRESH Otsu's optimum threshold given a histogram.
% [T, SM] = OTSUTHRESH(H) computes an optimum threshold, T, in the
% range [0 1] using Otsu's method for a given a histogram, H.

% Normalize the histogram to unit area. If h is already normalized,
% the following operation has no effect.
h = h/sum(h);
h = h(:); % h must be a column vector for processing below.

% All the possible intensities represented in the histogram (256 for
% 8 bits). (i must be a column vector for processing below.)
i = (1: numel(h))';

% Values of P1 for all values of k.
P1 = cumsum(h);

% Values of the mean for all values of k.
m = cumsum(i.*h);

% The image mean.
mG = m(end);

% The between-class variance.
sigSquared = ((mG*P1 - m).^2)./(P1.*(1 - P1) + eps);

% Find the maximum of sigSquared. The index where the max occurs is
% the optimum threshold. There may be several contiguous max values.
% Average them to obtain the final threshold.
maxSigsq = max(sigSquared);
T = mean(find(sigSquared == maxSigsq));

% Normalized to range [0 1]. 1 is subtracted because MATLAB indexing
% starts at 1, but image intensities start at 0.
T = (T - 1)/(numel(h) - 1);

% Separability measure.
SM = maxSigsq / (sum(((i - mG).^2) .* h) + eps);

```

很容易验证，这个函数给出了与函数 `graythresh` 一样的结果。

### 10.3.4 使用图像平滑改进全局阈值处理

噪声可以把简单的阈值处理问题转变为不能解决的问题。当噪声不能在源头减少，并且阈值处理是选择的分割方法时，提高性能的一种常用技术是在阈值处理之前先对图像进行平滑。我们用一个例子介绍这种方法。

在没有噪声时，图10-15(a)中的原始图像是双值的，可以使用处在两幅图像灰度值之间的任何阈值做完美的阈值处理。图 10-15(a)中的图像是在原始的二值图像上加入均值为 0、标准差为 50 个灰度级的高斯噪声的结果。带噪图像的直方图(见图 10-15(b))清楚地指出，阈值处理在不加改变的图像上可能会失败。图 10-15(c)中的结果是用 Otsu's 方法得到的，这证实了这一点(物体上的每个暗点和背景上的每个亮点是阈值处理误差，所以分割稍微有点不成功)。

图 10-15(d)显示了用  $5 \times 5$  的均值模板平滑带噪图像(图像的尺寸是  $651 \times 814$  像素)的结果,图 10-15(e)是图像的直方图。由于平滑,直方图的形状改进是明显的,并且可以期望平滑后的图像的阈值处理近于完美。如图 10-15(f)所示,的确是这种情况。在分割后的图像中,物体和背景间的边界稍微有点失真,这是由平滑图像的边界模糊导致的。事实上,平滑一幅图像越强烈,在分割结果中就应该预见到边界误差越大。图 10-15 中的图像是用下列指令产生的:

```
>> f = imread('septagon.tif');
```

为了得到图 10-15(a),使用函数 `imnoise` 在这幅图像上加入均值为 0、标准差为 50 个灰度级的高斯噪声。

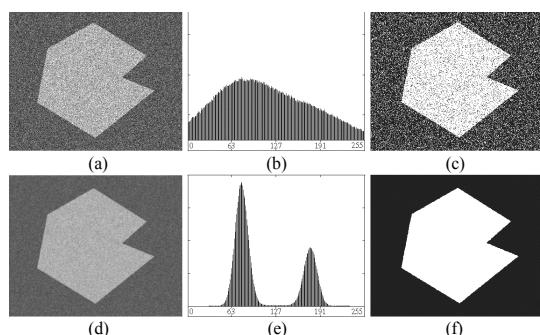


图 10-15 (a) 带噪图像; (b) 直方图; (c) 用 Otsu's 方法得到的结果; (d) 用  $5 \times 5$  均值模板平滑后的图像; (f) 使用 Otsu's 方法的阈值处理后的结果

工具箱使用方差作为输入,并且假定灰度范围是  $[0, 1]$ 。因为采用 255 灰度级,所以输入到 `imnoise` 函数的方差是  $50^2/255^2=0.038$ :

```
>> fn = imnoise(f,'gaussian', 0, 0.038);
>> imshow(fn) % Fig. 10.15(a).
```

The rest of the images in Fig. 10.15 were generated as follows:

```
>> figure, imhist(fn) % Fig. 10.15(b);
>> Tn = graythresh(fn);
>> gn = im2bw(fn, Tn);
>> figure, imshow(gn)
>> % Smooth the image and repeat.
>> w = fspecial('average', 5);
>> fa = imfilter(fn, w, 'replicate');
>> figure, imshow(fa) % Fig. 10.15(d).
>> figure, imhist(fa) % Fig. 10.15(e).
>> Ta = graythresh(fa);
>> ga = im2bw(fa, Ta);
>> figure, imshow(ga) % Fig. 10.15(f).
```

### 10.3.5 使用边缘改进全局阈值处理

基于前面 4 节的讨论,我们得出以下结论:如果直方图的峰是高的、窄的、对称的,并且由深的谷分开,那么选到好的阈值的机会就会增大。改进直方图的一种方法就是仅考虑那些位于或接近物体和背景间边缘的像素。比较直接和明显的改进是直方图不依赖物体和背景的相对

大小。另外，位于物体上的任何像素的概率将近似等于位于背景上的像素的概率，这样就改进了直方图峰值的对称性。最后，正如下面内容指出的那样，满足某些基于梯度度量的像素在直方图的峰值之间有较深的谷。刚才讨论的方法假设物体和背景之间的边缘已知。显然，这一信息在分割期间并不可用，正如在物体和背景间寻找分界线正是分割所要做的一切那样。然而，像素是否处在边缘上的暗示可能可以由计算梯度或拉普拉斯的绝对值来获得(记住，一副图像的拉普拉斯同时拥有正值和负值)。典型的、可比较的结果可以用两种方法的任意一种得到。前边的讨论是下边算法的总结，其中， $f(x,y)$ 是输入图像：

(1) 用 10.1 节讨论的任何方法计算来自  $f(x,y)$  的边缘图像。边缘图像可以是梯度或拉普拉斯的绝对值。

(2) 指定阈值  $T$ 。

(3) 用来自步骤(2)的阈值对来自步骤(1)的图像进行阈值处理，产生一副二值图像  $g_T(x,y)$ 。这幅图像在步骤(4)中选择来自  $f(x,y)$  的对应于强边缘的像素并作为标记图像使用。

(4) 仅用  $f(x,y)$  中的像素计算直方图，对应  $g_T(x,y)$  中 1 值像素的位置。

(5) 用来自步骤(4)中的直方图，通过全局阈值方法(如 Otsu's 方法)来分割  $f(x,y)$ 。

通常，通过指定  $T$  值( $T$  值与某个百分比对应)来典型地设置高值(例如 90%)，因此，在阈值计算中使用图像的边缘上没有多少像素。自定义函数 `percentile2i`(见附录 C)可用于这个目的。该函数计算灰度值  $I$ ， $I$  对应指定的百分比。语法是：

```
I = percentile2i(h, P)
```

其中， $h$  是图像的直方图， $p$  是在  $[0, 1]$  范围内的百分比值。输出  $I$  是灰度级(也在  $[0, 1]$  范围内)，对应第  $p$  个百分点。

### 例 10.9 使用基于梯度的边缘信息改进全局阈值处理

图 10-16(a)显示了在尺寸上缩小为几个像素的 septagon 图像。这幅图像被具有 0 均值和标准差为 10 个灰度级的高斯噪声污染了。从图 10-16(b)中的直方图可以看出，直方图是单峰的，并且从我们的关于物体大得多的负面经验，可以得出如下结论：在这种情况下，全局阈值处理将会失败。当物体比背景小得多时，它们对直方图的贡献可以忽略不计。使用边缘信息可以改进这种情况，图 10-16(c)是用下列指令得到的梯度图像：

```
>> f = tofloat(imread('Fig1016(a).tif'));
>> sx = fspecial('sobel');
>> sy = sx';
>> gx = imfilter(f,sx,'replicate');
>> gy = imfilter(f,sy,'replicate');
>> grad = sqrt(gx.*gx + gy.*gy);
>> grad = grad/max(grad(:));
```

其中，对于浮点图像，最后的指令把 `grad` 的值归一化到正确的  $[0, 1]$  范围内。下一步，我们得到 `grad` 的直方图，并使用高的百分比(99.9%)估计梯度的阈值，记住，我们只想保留梯度图像中较大的值，这个值应该发生在接近物体和背景的边界处：

```
>> h = imhist(grad);
>> Q = percentile2i(h, 0.999);
```

其中， $Q$  在  $[0, 1]$  范围内。下一步，用  $Q$  对梯度做阈值处理，形成标记图像，并且从  $f$  中提

取梯度值比  $Q$  大的点，得到结果的直方图：

```
>> markerImage = grad > Q;
>> figure, imshow(markerImage) % Fig. 10.16(c).
>> fp = f.*markerImage;
>> figure, imshow(fp) % Fig. 10.16(d).
>> hp = imhist(fp);
```

图像  $fp$  包含围绕背景和物体边界的  $f$  的像素，所以， $fp$  的直方图受 0 的控制。因为我们的兴趣在于物体边界周围的分割值，所以需要消除 0 对直方图的贡献。因此，把  $fp$  的第一个元素排除在外；然后，用结果的直方图得到 Otsu 阈值：

```
>> hp(1) = 0;
>> bar(hp, 0) % Fig. 10.16(e).
>> T = otsuthresh(hp);
>> T*(numel(hp) - 1)
ans =
    133.5000
```

直方图  $hp$  显示于图 10-16(e)中。我们观察到，现在有了明显的相对较窄的、用一个深谷分开的峰。就像期望的那样，最佳阈值接近模式的中点，从而可以期待近于完美的分割。

```
>> g = im2bw(f, T);
>> figure, imshow(g) % Fig. 10.16(f).
```

如图 10-16(f)所示，图像的确被完全地分割了。

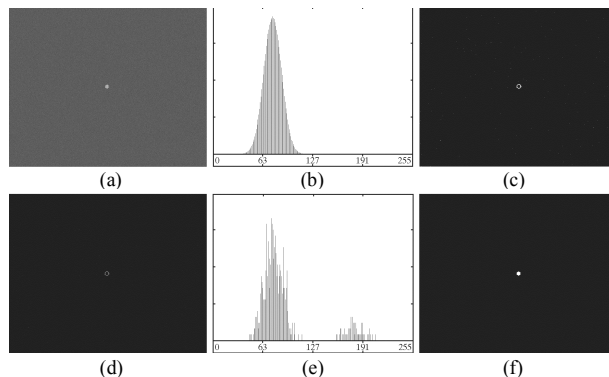


图 10-16 (a) 带噪声的小的 septagon 图像；(b) 直方图；(c) 以 99.9% 进行阈值处理后的梯度幅值图像；(d) 由(a)和(c)的乘积形成的图像；(e) 在(d)图像中非 0 像素的直方图；(f) 使用(e)中的直方图找到的通过 Otsu 阈值分割图(a)的结果(找到的阈值是 133.5，近似于直方图峰值的中点)。(原图像由南加州大学的 Susan L. Forsburg 教授提供)

#### 例 10.10 用拉普拉斯边缘信息改进全局阈值处理

在这个例子中，我们考虑更复杂的阈值处理问题，并且说明如何用拉普拉斯得到导致改善分割的边缘信息。图 10-17(a)是一幅酵母细胞的 8 比特图像，其中，我们希望用全局阈值处理得到亮点对应的区域。作为开始，图 10-17(b)显示了图像的直方图。

图 10-17(c)是直接把 Otsu's 方法用于图像的结果：

```
>> f = tofloat(imread('Fig1017(a).tif'));
>> imhist(f) % Fig. 10.17(b).
```

```
>> hf = imhist(f);
>> [Tf SMf] = graythresh(f);
>> gf = im2bw(f, Tf);
>> figure, imshow(gf) % Fig. 10.17(c).
```

我们看到, Otsu's 方法没有达到检测亮点的原始目标。当该方法能够孤立区域本身的某些细胞时,在右边的一些区域没有脱开。由 Otsu's 方法计算的阈值是 42, 并且可分性度量是 0.636。下边的步骤除了使用拉普拉斯绝对值边缘信息之外, 其他与例 10.9 类似。这里使用稍低的百分比, 因为拉普拉斯阈值比前边的例子更稀疏:

```
>> w = [-1 -1 -1; -1 8 -1; -1 -1 -1];
>> lap = abs(imfilter(f, w, 'replicate'));
>> lap = lap/max(lap(:));
>> h = imhist(lap);
>> Q = percentile2i(h, 0.995);
>> markerImage = lap > Q;
>> fp = f.*markerImage;
>> figure, imshow(fp) % Fig. 10.17(d).
>> hp = imhist(fp);
>> hp(1) = 0;
>> figure, bar(hp, 0) % Fig. 10.17(e).
>> T = otsuthresh(hp);
>> g = im2bw(f, T);
>> figure, imshow(g) % Fig. 10.17(f).
```

图 10-17(d)显示了  $f$  和  $\text{markerImage}$  的乘积。注意在这幅图像中, 点如何聚集在亮点的边缘附近, 正如在刚才讨论中期望的那样。图 10-17(e)是(d)中非 0 像素的直方图。最后, 10.17(f)显示了以图 10-17(e)的直方图为基础, 使用 Otsu's 方法全局分割原始图像的结果。这个结果适合图像中亮点的位置。用 Otsu's 方法计算的阈值是 115, 并且分离度量是 0.762, 这两个值比直接从图像得到的值高。

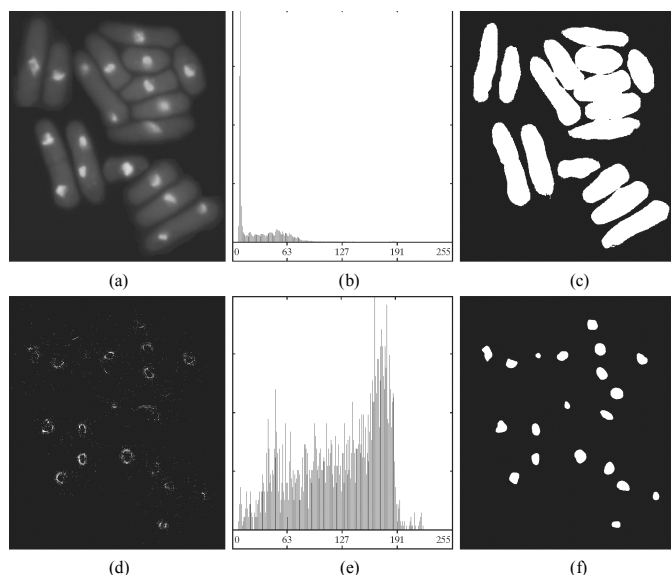


图 10-17 (a) 酵母细胞的图像; (b) (a) 的直方图; (c) 用函数 `graythresh` 对(a)进行分割; (d) 标记图像与原始图像的乘积; (e) (d) 中非 0 像素的直方图; (f) 以(e)中的直方图为基础, 用 Otsu's 方法进行阈值处理后的图像。(原图像由南加利福尼亚大学的 Susan L. Forsburg 教授提供)

### 10.3.6 基于局部统计的可变阈值处理

当背景照明高度不均匀时,有代表性的全局阈值处理就会失败。针对这个问题的一种解决办法是试图估计明暗函数,用于补偿不均匀的灰度模式;然后,用上边讨论的方法之一对图像做全局阈值处理。在 9.6.2 节,你已经见过这种方法的一个例子。针对不规则光照的补偿,或存在多于一个支配物体灰度的情况下(在这种情况下,全局阈值处理也有困难),采用的另一种方法是进行可变阈值处理。这种方法在  $(x, y)$  的邻域中以一个或多个指定像素的特性在图像的每一点  $(x, y)$  计算阈值。

我们用一幅图像中每一点的邻域中像素的标准差和均值来说明局部阈值处理的基本方法。这两个量对决定局部阈值十分有用,因为它们是局部对比度和平均灰度的描述子。令  $\sigma_{xy}$  和  $m_{xy}$  代表包含在一幅图像中以坐标  $(x, y)$  为中心的邻域中的一组像素的标准差和均值。为计算局部标准差,可以使用函数 `stdfilt`, 语法如下:

```
g = stdfilt(f, nhood)
```

其中,  $f$  是输入图像,  $nhood$  是由 0 和 1 组成的数组。在这个数组中,非 0 元素指定了用于计算局部标准差的邻域。 $nhood$  的尺寸在每个维度上必须是奇数,默认值是 3。为了计算局部均值,可以使用下列通用函数:

```
function mean = localmean(f, nhood)
%LOCALMEAN Computes an array of local means.
% MEAN = LOCALMEAN(F, NHOOD) computes the mean at the center of
% every neighborhood of F defined by NHOOD, an array of zeros and
% ones where the nonzero elements specify the neighbors used in the
% computation of the local means. The size of NHOOD must be odd in
% each dimension; the default is ones(3). Output MEAN is an array
% the same size as F containing the local mean at each point.

if nargin == 1
    nhood = ones(3) / 9;
else
    nhood = nhood / sum(nhood(:));
end
mean = imfilter(tofloat(f), nhood, 'replicate');
```

下面是变量、基于局部均值和标准差的阈值的普通形式:

$$T_{xy} = a\sigma_{xy} + bm_{xy}$$

其中,  $a$  和  $b$  是非负的常数。另一种有用的形式是:

$$T_{xy} = a\sigma_{xy} + bm_G$$

这里,  $m_G$  是全局图像均值。分割图像的计算如下:

$$g(x, y) = \begin{cases} 1 & f(x, y) > T_{xy} \\ 0 & f(x, y) \leq T_{xy} \end{cases}$$

其中,  $f(x, y)$  是输入图像。这个式子可在所有的像素位置进行评估和应用。

有意义的加权可用结合逻辑的局部特性代替算术特性加到局部阈值处理上,就像上边那样。



例如，可以借助逻辑“与”定义局部阈值处理，如下所示：

$$g(x,y)=\begin{cases} 1 & f(x,y) > a\sigma_{xy} \text{ AND } f(x,y) > bm \\ 0 & \text{其他} \end{cases}$$

其中， $m$  不是局部均值就是全局均值  $m_{xy}$ ，正如上边定义的那样。

下边的函数用这个公式执行局部阈值处理。这个函数的基本结构可很容易地与其他逻辑和/或局部操作相结合。

```
function g = localthresh(f, nhood, a, b, meantype)
%LOCALTHRESH Local thresholding.
%   G = LOCALTHRESH(F, NHOOD, A, B, MEANTYPE) thresholds image F by
%   computing a local threshold at the center, (x, y), of every
%   neighborhood in F. The size of the neighborhoods is defined by
%   NHOOD, an array of zeros and ones in which the nonzero elements
%   specify the neighbors used in the computation of the local mean
%   and standard deviation. The size of NHOOD must be odd in both
%   dimensions.
%
%   The segmented image is given by
%
%       1 if (F > A*SIG) AND (F > B*MEAN)
%   G =
%       0 otherwise
%
%   where SIG is an array of the same size as F containing the local
%   standard deviations. If MEANTYPE = 'local' (the default), then
%   MEAN is an array of local means. If MEANTYPE = 'global', then
%   MEAN is the global (image) mean, a scalar. Constants A and B
%   are nonnegative scalars.

% Intialize.
f = tofloat(f);

% Compute the local standard deviations.
SIG = stdfilt(f, nhood);
% Compute MEAN.
if nargin == 5 && strcmp(meantype, 'global')
    MEAN = mean2(f);
else
    MEAN = localmean(f, nhood); % This is a custom function.
end

% Obtain the segmented image.
g = (f > a*SIG) & (f > b*MEAN);
```

### 例 10.11 对全局和局部阈值处理的比较

图 10-18(a)显示了来自例 10.10 的图像。我们想要从背景中分割出细胞来，并且从细胞的主  
体分出细胞核(内部的亮区域)。这幅图像中有三个主要的灰度级，因此有理由期待这样的分割  
是可能的。然而，使用单一阈值做这个工作非常不可靠，这在图 10-18(b)中已验证过了，图 10-18(b)  
显示了用 Otsu's 方法得到的结果。

```
>> [TGlobal] = graythresh(f);
>> gGlobal = im2bw(f, TGlobal);
>> imshow(gGlobal) % Fig. 10.18(b).
```

其中,  $f$  是图 10-18(a)中的图像。正如图中所示, 可以部分地从背景中分割出细胞(一些分割过的细胞连在了一起), 但是这种分割方法不能提取细胞核。

因为细胞核比细胞本身明显较亮, 所以预期围绕细胞核边界的标准差相对较大, 而围绕细胞边界的标准差稍微小一些。如图 10-18(c)所示, 的确是这种情况。由此得出以下结论: 在基于局部标准差的函数 `localthresh` 中, 这应该是很有帮助的。

```
>> g = localthresh(f, ones(3), 30, 1.5, 'global');
>> SIG = stdfilt(f, ones(3));
>> figure, imshow(SIG, [ ]) % Fig. 10.18(c).
>> figure, imshow(g) % Fig. 10.18(d).
```

如图 10-18(d)所示, 运用了属性的分割是相当有效的。个别细胞已经从背景中分割出来了, 并且细胞核也被完全分割出来了。函数使用的值是由实验决定的, 这样的做法在应用中很常见。当背景接近于常数, 并且所有物体的灰度高于或低于背景灰度时, 选择全局均值一般会得到较好的结果。

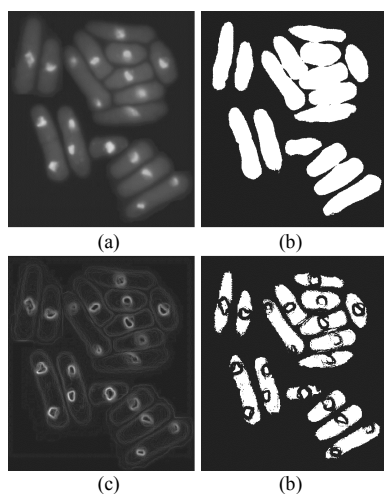


图 10-18 (a) 酵母细胞图像; (b) 用 Otsu's 方法分割的图像; (c) 局部标准差图像; (d) 用局部阈值处理分割的图像

### 10.3.7 使用移动平均的图像阈值处理

前面讨论的局部阈值处理方法的一种特殊情况是: 沿着一幅图像的扫描线计算移动平均。当速度是基本要求时, 这个实现在文本处理中十分有用。典型的扫描是以 `zigzag` 模式逐线执行, 进而减少照明偏差。令  $Z_{k+1}$  表示在扫描顺序中, 在第  $k+1$  步遇到的一个点。在新点处的移动平均(平均灰度)由下式给出:

$$m(k+1) = \frac{1}{n} \sum_{i=k+2-n}^{k+1} z_i = m(k) + \frac{1}{n}(z_{k+1} - z_{k-n})$$

其中,  $n$  代表计算平均时使用的点数,  $m(1) = z_1/n$ 。这个初始值并不严格正确, 因为单点的平均值是该点自身。然而, 我们使用  $m(1) = z_1/n$ , 在前边的平均等式第一次启动时, 并不要求

特殊的计算。另一方法是使用一个值，如果图像的边界用  $n-1$  个 0 填充，我们将得到这个值。算法只初始化一次，不必在每一行初始化。因为移动平均在图像中对每一点计算，所以分割用下式执行：

$$f(x,y)=\begin{cases} 1 & f(x,y) > Km_{xy} \\ 0 & \text{其他} \end{cases}$$

其中， $K$  是  $[0, 1]$  范围内的常数， $m_{xy}$  是输入图像在点  $(x,y)$  处的移动平均。下面的通用函数实现了刚刚讨论的概念，函数使用 MATLAB 函数 `filter`，这个一维滤波函数的基本语法如下：

```
Y = filter(c, d, X)
```

这个函数采用由分子系数向量  $c$  和分母系数向量  $d$  离散后的滤波器，对向量  $X$  中的数据进行滤波。如果  $d=1$ (标量)，那么  $c$  中的系数定义为完全滤波：

```
function g = movingthresh(f, n, K)
%MOVINGTHRESH Image segmentation using a moving average threshold.
% G = MOVINGTHRESH(F, n, K) segments image F by thresholding its
% intensities based on the moving average of the intensities along
% individual rows of the image. The average at pixel k is formed
% by averaging the intensities of that pixel and its n - 1
% preceding neighbors. To reduce shading bias, the scanning is
% done in a zig-zag manner, treating the pixels as if they were a
% 1-D, continuous stream. If the value of the image at a point
% exceeds K percent of the value of the running average at that
% point, a 1 is output in that location in G. Otherwise a 0 is
% output. At the end of the procedure, G is thus the thresholded
% (segmented) image. K must be a scalar in the range [0, 1].

% Preliminaries.
f = tofloat(f);
[M, N] = size(f);
if (n < 1) || (rem(n, 1) ~= 0)
    error('n must be an integer >= 1.')
end
if K < 0 || K > 1
    error('K must be a fraction in the range [0, 1].')
end

% Flip every other row of f to produce the equivalent of a zig-zag
% scanning pattern. Convert image to a vector.
f(2:2:end, :) = fliplr(f(2:2:end, :));
f = f'; % Still a matrix.
f = f(:)'; % Convert to row vector for use in function filter.

% Compute the moving average.
maf = ones(1, n)/n; % The 1-D moving average filter.
ma = filter(maf, 1, f); % Computation of moving average.

% Perform thresholding.
g = f > K * ma;

% Go back to image format (indexed subscripts).
```

```
g = reshape(g, N, M)';
% Flip alternate rows back.
g(2:2:end, :) = fliplr(g(2:2:end, :));
```

### 例 10.12 利用移动平均的图像阈值处理

图10-19(a)显示了一幅由斑点灰度模式遮蔽了的手写文本图像,这种灰度阴影可以发生,例如,由照相闪光灯可得到这样的图像。图10-19(b)是用 Otsu's 全局阈值处理方法分割的结果。

```
>> f = imread('Fig1119(a).tif');
>> T = graythresh(f);
>> g1 = im2bw(f, T); % Fig. 10.19(b).
```

全局阈值处理能克服灰度变化并不意外。图10-19(c)显示了使用移动平均进行局部阈值处理的成功分割:

```
>> g2 = movingthresh(f, 20, 0.5);
>> figure, imshow(g2) % Fig. 10.19(c).
```

一种经验方法是令平均窗口的宽度为平均笔画宽度的 5 倍。在这种情况下,平均宽度是 4 个像素,因此我们令  $n=20$ , 并且令  $K=0.5$ (算法对这些参数值并不特别敏感)。为了演示这种分割方法的效率,使用与分割图10-19(d)中图像时相同的参数,这幅图像的典型变化是被正弦灰度变量污染了,当文本扫描器不好的时候会发生这种情况。如图10-19(e)和(f)所示,分割的结果类似于图10-19 的顶部图像。很明显,在这两种情况下,使用相同的  $n$  和  $K$  值得到了成功的分割结果,表明了这个方法的相对稳定性。通常,当感兴趣的物体对于图像尺寸来说较小(或较细)时,基于移动平均的阈值处理工作得很好。通常,打字或手写文本都满足这个条件。

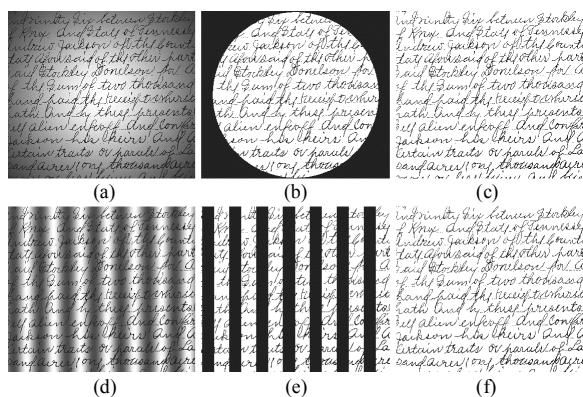


图 10-19 (a) 由斑点阴影污染了的文本图像; (b) 用 Otsu's 的全局阈值处理方法分割的结果; (c) 用移动平均进行局部阈值处理的结果; (d)~(f) 以相同操作顺序用于由正弦阴影污染了的图像的结果

## 10.4 基于区域的分割

分割的目的是把图像分成区域。在 10.1 节和 10.2 节中,我们基于灰度级的不连续性来寻找区域间边界的方法,解决了这一问题。但是在 10.3 节中,分割是基于像素特性(如灰度值)的分布,通过阈值处理得以完成的。这一节讨论直接寻找区域的分割技术。

### 10.4.1 基本表达式

令  $R$  表示整个图像区域。可以认为分割是把  $R$  分成了  $n$  个子区域—— $R_1, R_2, R_3, \dots, R_n$  的处理，满足：

- (a)  $\bigcup_{i=1}^n R_i = R$
- (b)  $R_i$  是连接区域,  $i=1, 2, \dots, n$
- (c)  $R_i \cap R_j = \Phi$ , 针对所有的  $i$  和  $j, i \neq j$
- (d)  $P(R_i) = \text{TRUE}$ ,  $i=1, 2, \dots, n$
- (e)  $P(R_i \cap R_j) = \text{FALSE}$ , 针对任何邻接的区域  $R_i$  和  $R_j$

在这里,  $P(R_i)$  是集合  $R_i$  中定义的点的逻辑谓词,  $\Phi$  是空集。条件(a)指出分割必须是完全的, 即每个点都必须在某个区域中。条件(b)要求区域中的点应该被连接(如 4 连接或 8 连接)。条件(c)说明区域间必须是不相交的。条件(d)说明分割区域中的像素点必须满足的性质, 比如, 如果  $R_i$  中的所有像素拥有相同的灰度级, 那么  $P(R_i) = \text{TURE}$ 。最后, 条件(e)指出, 邻接区域  $R_i$  和  $R_j$  在属性  $P$  上的意义是不一样的。

### 10.4.2 区域生长

顾名思义, 区域生长是根据预先定义的生长准则, 把像素或子区域集成较大区域的处理方法。基本处理方法是 以一组“种子”点开始, 形成这些生长区域, 把预先定义好的与这些种子性质相近的邻域像素附加到这些种子上(比如指定的灰度级或颜色范围)。

通常, 为了选择由一个或多个种子点组成的集合, 可以以问题的性质作为基础, 就像例 10.14 中描述的那样。当没有先验的信息可用时, 一种方法是在每个像素上计算一组相同的特性, 最后在生长处理期间分配像素到区域中。如果这个计算的结果显示一簇值, 就把拥有这些特性的像素放在可以作为种子的这些簇的中心附近。

相似性准则的选择不但依赖于所考虑的问题, 而且也依赖于图像可用数据的类型。例如, 对卫星图像的分析强烈依赖于颜色。在彩色图像中, 在没有固有信息可用的情况下, 这个问题可能明显地会更困难, 甚至是不可能的。图像是单色时, 图像分析应该用一组基于灰度级(例如运动或纹理)和空间特性(例如连通性)的描述符来进行。我们在第 11 章中将讨论对区域特性很有用的描述符。

如果在区域生长处理中没有连通信息(或连接性)而只有描述符的话, 可能会产生错误结果。例如, 仅仅用三个不同的灰度值检验像素的随机排列。把相同灰度的像素分组, 进而形成区域, 但不注意连通性, 将有可能产生无意义的分割结果。

区域生长中的另一个问题是停止规则的明确表达。基本上, 当没有更多像素的满足包含在区域中的准则时, 区域生长的过程就应该停止。灰度值、纹理、色彩这样的准则实际上是局部的, 不考虑区域生长的历史。增加区域生长算法能力的附加准则利用了大小的概念, 候选像素和迄今为止被生长像素之间的相似性(比如对候选灰度和生长区域中平均灰度的比较), 以及被生长区域的形状等。这些类型的描述符的应用基于如下假设: 期待结果的模型至少部分可用。

为了说明区域分割是如何在 MATLAB 中操作的, 我们开发了下面的 M-函数来完成基本的区域生长, 这个函数名为 `regiongrow`, 语法是:

```
[g,NR,SI,TI] = regiongrow(f,S,T)
```

其中,  $f$  是被分割的图像, 参数  $S$  可以是数组(与  $f$  大小相同)或标量。如果  $S$  是数组, 那么在所有种子点的坐标处必须为 1, 而在其他地方为 0。这样的数组可以通过观察决定, 或者通过外部的用于寻找种子的函数决定。如果  $S$  是标量, 就将  $S$  定义为灰度值; 在  $f$  中, 具有该灰度值的所有点都可以是种子。类似的,  $T$  也可以是数组(与  $f$  大小相同)或标量。如果  $T$  是数组, 那么对于  $f$  中的每个位置都应该包含阈值。如果  $T$  是标量, 就将之定义为全局阈值。阈值用来测试图像中的像素与种子是否足够相似, 或者是否是 8 连接的。 $S$  和  $T$  的所有值必须标定在  $[0,1]$  范围内, 并独立于输入图像的类。

例如, 如果  $S=a$  且  $T=b$ , 就比较灰度; 如果像素灰度和  $a$  之间差的绝对值小于或等于  $b$ , 像素将被认为和  $a$  相似(在通过阈值进行测试的场景下)。另外, 如果问题中的像素是 8 连接到一个或多个种子值, 那么这个像素就认为是一个或多个区域的成员。类似地, 如果  $S$  或  $T$  是数组, 也有类似的结论, 只不过是在  $S$  和  $T$  的相应元素间进行比较。

在输出中,  $g$  是分割后的图像, 每个区域的成员都用不同的整数标出。参数  $NR$  是要寻找的区域的数目。参数  $SI$  是包含种子点的图像, 参数  $TI$  是包含经过连通性处理前通过阈值测试的像素的一幅图像。 $SI$  和  $TI$  与  $f$  的大小一样。

函数 `regiongrow` 的代码如下。注意, 第 9 章中函数 `bwmorph` 的用法是为了把  $S$ (当  $S$  是数组时)中的每个区域与种子点连接的数目减少为 1, 而 `imreconstruct` 函数用于寻找连接每个种子点的像素。

```
function [g, NR, SI, TI] = regiongrow(f, S, T)
%REGIONGROW Perform segmentation by region growing.
% [G, NR, SI, TI] = REGIONGROW(F, S, T). S can be an array (the
% same size as F) with a 1 at the coordinates of every seed point
% and 0s elsewhere. S can also be a single seed value. Similarly,
% T can be an array (the same size as F) containing a threshold
% value for each pixel in F. T can also be a scalar, in which case
% it becomes a global threshold. All values in S and T must be in
% the range [0, 1]
%
% G is the result of region growing, with each region labeled by a
% different integer, NR is the number of regions, SI is the final
% seed image used by the algorithm, and TI is the image consisting
% of the pixels in F that satisfied the threshold test, but before
% they were processed for connectivity.

f = tofloat(f);
% If S is a scalar, obtain the seed image.
if numel(S) == 1
    SI = f == S;
    S1 = S;
else
    % S is an array. Eliminate duplicate, connected seed locations
    % to reduce the number of loop executions in the following
    % sections of code.
    SI = bwmorph(S, 'shrink', Inf);
    S1 = f(SI); % Array of seed values.
end
```

```

TI = false(size(f));
for K = 1:length(SI)
    seedvalue = SI(K);
    S = abs(f - seedvalue) <= T; % Re-use variable S.
    TI = TI | S;
end
% Use function imreconstruct with SI as the marker image to
% obtain the regions corresponding to each seed in S. Function
% bwlabel assigns a different integer to each connected region.
[g, NR] = bwlabel(imreconstruct(SI, TI));

```

### 例 10.13 使用区域生长检测焊接空隙

图10-20(a)显示了一幅包含几个裂缝(水平的暗区域)和空隙(穿过图像中部的、亮的、白色的水平方向条纹)的X射线焊接图像。我们希望使用 `regiongrow` 函数来分割相应的焊接缺陷区域。这些被分割的区域可以用于自动检测这一任务,可以用于包含历史研究的数据库,也可以用于控制自动焊接系统。

第一步是确定初始种子点。在这一应用中,已知焊缝缺损区域中的一些像素趋向于有最大的数字值(在这种情况下是255)。基于这个信息,我们令  $S=1$  (所有的  $S$  值必须标定为  $[0, 1]$  范围内)。下一步是选择阈值或阈值数组。在这个例子中,我们令阈值等于65(当标定为  $[0, 1]$  范围内时,值是0.26)。这个值来自对图10-21中直方图的分析,并表示225和左边第一个主谷的位置之间的差(190),主谷的位置表示暗的焊接区域中的最高灰度值。图10-20所示的结果是通过调用下列函数产生的:

```
>> [g, NR, SI, TI] = regiongrow(f, 1, 0.26);
```

图10-20(b)显示了种子点(图像  $SI$ )。在这种情况下,种子点很多,因为种子被指定为在图像中具有数值225的所有点(标定后是1)。图10-20(c)是图像  $TI$ , 显示了所有通过阈值测试的点;也就是说,具有灰度  $Z_i$  且满足  $|Z_i - S| \leq T$  的点。图10-20(d)显示了提取图10-20(c)中所有连接到种子点的像素的结果。这是分割后的图像  $g$ 。通过将这幅图像与原始图像进行比较,区域生长过程确实以合理的精确度分割了焊接的缺陷这一点是很明显的。最后,通过观察图10-21中的直方图,注意到不可能通过10.3节中讨论的任何阈值处理方法获得相同或等价的解决方案。在这种情况下,连通性是基本要求。

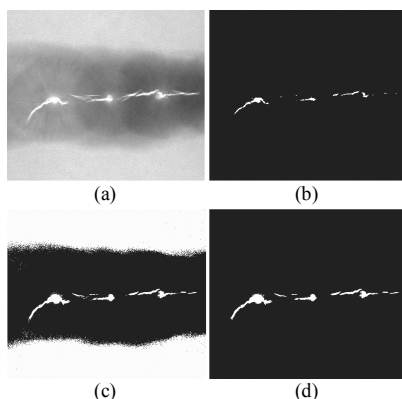


图 10-20 (a) 显示有焊接缺陷的图像; (b) 种子点; (c) 显示所有通过了阈值测试的像素的二值图像(白色); (d) (c)中的所有像素在对种子点进行8连通性分析后的结果。(原图像由 XTEK Systems 公司提供)

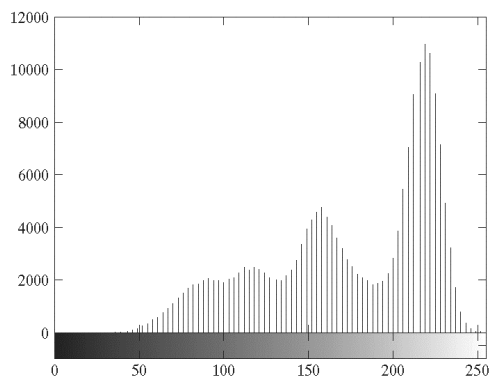


图 10-21 图 10-20(a)的直方图

### 10.4.3 区域分离和聚合

刚刚讨论的过程是从一组种子点生长区域。还有一种可供选择的方法是再次把图像细分为一组任意的、互不连接的区域，然后在试图满足 10.4.1 节规定的条件下合并或分离这些区域。

令  $R$  代表整个图像区域，选择属性  $P$ 。分割  $R$  的一种方法是把  $R$  连续地细分成越来越小的象限区域，以便对任何区域  $R_i$  都有  $P(R_i)=\text{TRUE}$ 。我们从整个区域开始：如果  $P(R)=\text{TRUE}$ ，就把图像分成 4 象限；如果对每个 4 象限来说， $P$  都是 FALSE，就再细分象限为子象限，这样继续下去。这种特别的分离技术有一种方便的表现方法，叫做四叉树；这是一棵树，树中的每个节点都恰好有 4 个后代，如图 10-22 所示。子图像对应四叉树的节点，有时称为四叉区域或四叉图像。注意，树的根对应整幅图像，每个节点对应被再分为 4 个后代节点的节点。在这种情况下，只有  $R_4$  被进一步细分了。

如果只使用分离，最终的部分通常包括具有相同属性的邻近区域，这个缺点可以通过下边的合并及分离来解决。如果满足 10.4.1 节的限制，将要求只合并邻近的区域，合并像素满足属性  $P$ 。也就是说，两个邻近的区域  $R_j$  和  $R_k$ ，只有在满足  $P(R_j \cup R_k)=\text{TURE}$  的时候才能合并。

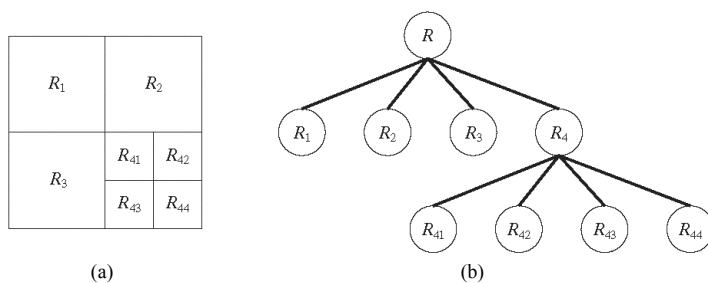


图 10-22 (a) 分割图像；(b) 相应的四叉树

前面的讨论可以用下面的过程加以总结：

- (1) 分离任意区域  $R_i$  为 4 个不相连的象限，满足  $P(R_i)=\text{FALSE}$ 。
- (2) 当无法进一步分离时，合并任何满足  $P(R_j \cup R_k)=\text{TURE}$  的区域  $R_j$  和  $R_k$ 。
- (3) 在无法进一步合并的时候停止。

前面介绍的基本主题可能会有多种变化。例如，如果允许合并两个邻近的区域  $R_i$  和  $R_j$ ，而它们中的每一个都各自满足属性，就可得到有意义的简化。这可产生更简单的算法(更快)，因



为属性的测试被限制在四叉区域。正如在例 10.14 中那样，这种简化还是有可能在实践中产生出好的分割结果的。在上述过程的步骤(2)中使用这一方法，满足属性的所有四叉区域都用 1 填充，并且它们的连通性可以很容易被检查，比如用 `imreconstruct` 函数。在效果上，这个函数能达到希望的邻近象限区域的合并。不满足属性的四叉区域都用 0 填充，从而产生分割的图像。

在工具箱中执行四叉树分解的函数是 `qtdecomp`，我们感兴趣的语法如下：

```
Z = qtdecomp(f, @split_test, parameters)
```

这里，`f` 是输入图像，`Z` 是包括四叉树结构的稀疏矩阵。如果 `Z(k, m)` 非零，那么 `(k, m)` 是分解块的左上角，而且块的大小是 `Z(k, m)`。

`split_test` 函数(见下面例子中的 `splitmerge` 函数)用来决定某个区域是否进行分离，`parameters`(用逗号分开)是 `split_test` 函数要求的附加参数。这个结构和 2.4.2 节中讨论的 `coltfilt` 函数相似。

为了在四叉树分解中得到实际的四叉区域像素值，使用 `qtgetblk` 函数，语法是：

```
[vals, r, c] = qtgetblk(f, Z, m)
```

其中，`vals` 是数组，`vals` 包含 `f` 中四叉树分解的尺寸为 `m×m` 的块的值，`Z` 是由 `qtdecomp` 返回的稀疏矩阵。参数 `r` 和 `c` 是包含块的左上角行和列坐标的向量。

下面通过编写基本的分离和合并用的 `M`-函数来简化早些时候讨论的方法来说明 `qtdecomp` 函数的使用，如果其中的两个区域分别满足属性，它们将被合并。我们可以调用的 `splitmerge` 函数拥有如下调用语法：

```
g = splitmerge(f, mindim, @predicate)
```

在这里，`f` 是输入图像，`g` 是输出图像，其中的每个连接区域都用不同的整数标注。参数 `mindim` 定义在分解中允许的最小块；该参数必须是 2 的正整数次幂，允许分解下至 `1×1` 像素大小的区域，虽然这么精细的细节在实际中并不常用。函数 `predicate` 是用户自定义函数，语法为：

```
flag = predicate(region)
```

如果 `region`(区域)中的像素满足函数中由代码定义的属性，函数就必须被写成返回 `true`(逻辑 1)；否则，`flag` 的值就必须是 `false`(逻辑 0)。例 10.14 说明了该函数的应用。

函数 `splitmerge` 的结构很简单。首先，图像被函数 `qtdecomp` 分块。函数 `split_test` 使用 `predicate` 来决定区域是否应该被分离。因为当区域被分成 4 个时，我们并不知道产生的 4 个区域中的哪一个将通过属性测试。在了解了在分离的图像中哪一个区域通过测试之后，考查一下区域是有必要的。函数 `predicate` 也用于这个目的。任何通过测试的四叉区域都用 1 填充，任何没有通过测试的用 0 填充。标识数组可在每个被填充了 1 的区域通过选择元素来创建。标识数组连同分割的图像一起被用于决定区域的连通性(邻接性)；函数 `imreconstruct` 也用于这一目的。

如果有必要，紧接着 `splitmerge` 函数，程序把输入图像填补为方形的，它的维数是包围图像的 2 的最小整数次幂。这就允许函数 `qtdecomp` 对区域一直分离下去，直至 `1×1` 大小(单个像素)，正如早些时候提到的那样。

```

function g = splitmerge(f, mindim, fun)
%SPLITMERGE Segment an image using a split-and-merge algorithm.
%   G = SPLITMERGE(F, MINDIM, @PREDICATE) segments image F by using
%   a split-and-merge approach based on quadtree decomposition.
%   MINDIM (a nonnegative integer power of 2) specifies the minimum
%   dimension of the quadtree regions (subimages) allowed. If
%   necessary, the program pads the input image with zeros to the
%   nearest square size that is an integer power of 2. This
%   guarantees that the algorithm used in the quadtree decomposition
%   will be able to split the image down to blocks of size 1-by-1.
%   The result is cropped back to the original size of the input
%   image. In the output, G, each connected region is labeled with a
%   different integer.
%
%   Note that in the function call we use @PREDICATE for the value
%   of fun. PREDICATE is a user-defined function. Its syntax is
%
%       FLAG = PREDICATE(REGION) Must return TRUE if the pixels in
%       REGION satisfy the predicate defined in the body of the
%       function; otherwise, the value of FLAG must be FALSE.
%
%   The following simple example of function PREDICATE is used in
%   Example 10.14 of the book. It sets FLAG to TRUE if the
%   intensities of the pixels in REGION have a standard deviation
%   that exceeds 10, and their mean intensity is between 0 and 125.
%   Otherwise FLAG is set to false.
%
%       function flag = predicate(region)
%       sd = std2(region);
%       m = mean2(region);
%       flag = (sd > 10) & (m > 0) & (m < 125);

% Pad the image with zeros to the nearest square size that is an
% integer power of 2. This allows decomposition down to regions of
% size 1-by-1.
Q = 2^nextpow2(max(size(f)));
[M, N] = size(f);
f = padarray(f, [Q - M, Q - N], 'post');

% Perform splitting first.
Z = qtdecomp(f, @split_test, mindim, fun);
% Then, perform merging by looking at each quadregion and setting
% all its elements to 1 if the block satisfies the predicate defined
% in function PREDICATE.

% First, get the size of the largest block. Use full because Z is
% sparse.
Lmax = full(max(Z(:)));
% Next, set the output image initially to all zeros. The MARKER
% array is used later to establish connectivity.
g = zeros(size(f));
MARKER = zeros(size(f));
% Begin the merging stage.
for K = 1:Lmax
    [vals, r, c] = qtgetblk(f, Z, K);

```

```

if ~isempty(vals)
    % Check the predicate for each of the regions of size K-by-K
    % with coordinates given by vectors r and c.
    for I = 1:length(r)
        xlow = r(I); ylow = c(I);
        xhigh = xlow + K - 1; yhigh = ylow + K - 1;
        region = f(xlow:xhigh, ylow:yhigh);
        flag = fun(region);
        if flag
            g(xlow:xhigh, ylow:yhigh) = 1;
            MARKER(xlow, ylow) = 1;
        end
    end
end
end
end
% Finally, obtain each connected region and label it with a
% different integer value using function bwlablel.
g = bwlablel(imreconstruct(MARKER, g));

% Crop and exit.
g = g(1:M, 1:N);

%-----%
function v = split_test(B, mindim, fun)
% THIS FUNCTION IS PART OF FUNCTION SPLIT-MERGE. IT DETERMINES
% WHETHER QUADREGIONS ARE SPLIT. The function returns in v
% logical 1s (TRUE) for the blocks that should be split and
% logical 0s (FALSE) for those that should not.

% Quadregion B, passed by qtdecomp, is the current decomposition of
% the image into k blocks of size m-by-m.
% k is the number of regions in B at this point in the procedure.
k = size(B, 3);

% Perform the split test on each block. If the predicate function
% (fun) returns TRUE, the region is split, so we set the appropriate
% element of v to TRUE. Else, the appropriate element of v is set to
% FALSE.
v(1:k) = false;
for I = 1:k
    quadregion = B(:, :, I);
    if size(quadregion, 1) <= mindim
        v(I) = false;
        continue
    end
    flag = fun(quadregion);
    if flag
        v(I) = true;
    end
end
end

```

#### 例 10.14 使用了区域分离和合并的图像分割

图 10-23(a)显示了一幅天鹅星座环的 X 射线频段图像。图像的大小为 256×256 像素。该例的目的是分割出环绕致密中心的稀疏环。我们感兴趣的区域有某些明显的特征，这些特征在分

割中会有帮助。首先,我们注意到数据具有随机性,这表明稀疏环的标准差比背景(值为 0,因为背景是恒定的)和较大中心区域的标准差大。相似的,包含外环区域数据的均值(平均亮度)应该比背景(值为 0)的均值大,而比较大且较亮的中心区域的均值小。这样,我们应该能够利用这两个参数来分割感兴趣的区域。事实上,函数 `splitmerge` 的文档中指出,属性函数包含该问题的一些知识。属性函数 `predicate` 中显示的参数由计算图 10-23(a)中各个子区域的均值和标准差来决定。

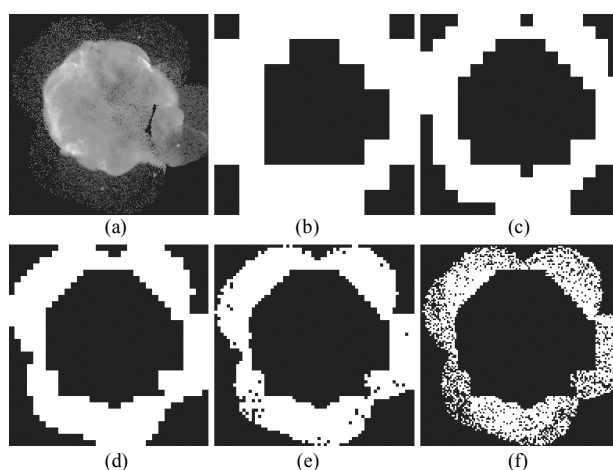


图 10-23 使用分离和合并算法分割图像:(a) 原始图像;(b)到(f) 使用函数 `splitmerge` 且 `mindim` 的值分别等于 32、16、8、4、2 时进行分割的结果。(原图像由 NASA 提供)

图 10-23(b)到(f)显示了使用函数 `splitmerge` 且 `mindim` 的值分别等于 32、16、8、4、2 时分割图 10-23(a)的结果。所有图像均显示了边界的细节水平与 `mindim` 的值成反比的分割结果。图 10-23 的所有结果都是合理的分割。如果以除原始图像之外的这些图像之一作为模板提取感兴趣区域,图 10-23(d)的结果将是最好的选择,因为它是具有最多细节的实心区域。刚刚说明的方法的一个重要方面,是在函数 `predicate` 中“捕获”信息的能力,这种能力对特定任务的分割有帮助。

## 10.5 使用分水岭变换的分割

在地理学中,分水岭是指山脊,通过不同的水系排水来分离区域。汇水盆地是把水排入河流或水库的地理区域。分水岭变换把这些概念用于灰度图像处理,在某种程度上可用于解决各种图像分割问题。

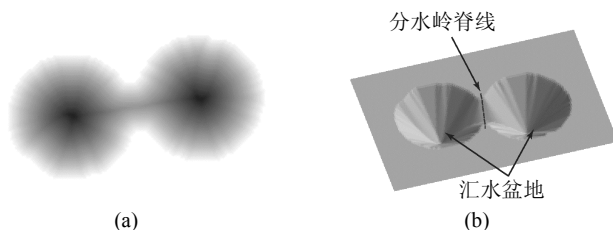


图 10-24 (a) 灰度图像;(b) 显示分水岭脊线和汇水盆地且被看做表面的图像

为了理解分水岭变换,要求我们把灰度图像看做拓扑表面。在这里, $f(x, y)$ 的值被解释为高度。例如,可以把图 10-24(a)中的简单图像形象化为图 10-24(b)中的三维表面。如果想象雨水降到这个表面上,很清楚,水将被收集到标为汇水盆地的两个区域,分水岭脊线上的降水将准确地、很可能是相等地收集到两个汇水盆地中的一个,分水岭变换将找到灰度图像中的汇水盆地和脊线。在解决图像分割问题方面,关键概念是把开始图像变为另外一幅图像,那些汇水盆地是我们想要辨别的目标或区域。

### 10.5.1 使用距离变换的分水岭分割

针对分割,与分水岭变换相配合的常用工具是距离变换。二值图像的距离变换是相对简单的概念:是指从每个像素到最接近零值的像素的距离。例如,图10-25(a)显示了一个小的二值图像矩阵。图10-25(b)显示了相应的距离变换。注意,每个值为 1 的像素的距离变换为 0,因为最靠近的非 0 像素是它本身。距离变换可以用工具箱函数 `bwdist` 来计算,调用语法为:

```
D = bwdist(f)
```

1	1	0	0	0	0.00	0.00	1.00	2.00	3.00
1	1	0	0	0	0.00	0.00	1.00	2.00	3.00
0	0	0	0	0	1.00	1.00	1.41	2.00	2.24
0	0	0	0	0	1.41	1.00	1.00	1.00	1.41
0	1	1	1	0	1.00	0.00	0.00	0.00	1.00

(a)
(b)

图 10-25 (a) 二值图像; (b) 距离变换

#### 例 10.15 使用距离变换和分水岭变换分割二值图像

在这个例子中,我们说明如何与工具箱的分水岭变换一起,使用距离变换分割彼此有些接触的圆形水滴。特别是,我们想要分割图 9-29(b)中处理过的销钉图像。首先,正如 10.3.1 节中描述的那样,使用 `im2bw` 和 `graythresh` 把图像变换为二值图像:

```
>> g = im2bw(f, graythresh(f));
```

图 10-26(a)显示了结果。下一步是对图像求补,计算距离变换。然后,用函数 `watershed` 计算距离变换的负分水岭变换。该函数的调用语法是:

```
L = watershed(A,conn)
```

其中, $L$  是在 9.4 节讨论和定义过的标记矩阵。 $A$  是输入数组(一般可以是任何维数,但在本章是二维),并且 `conn` 指定了连通性(对于二维数组是 4 或 8(默认值))。在  $L$  中,正整数与汇水盆地相对应,零值指出分水岭的脊线像素:

```
>> gc = ~g;
>> D = bwdist(gc);
>> L = watershed(-D);
>> w = L == 0;
```

图 10-26(b)和(c)显示了求补后的图像及其距离变换。因为  $L$  的 0 值像素是分水岭的脊线像素,前面代码的最后一行计算二值图像  $w$ ,图中仅显示这些像素。分水岭的脊线图像显示于图

10-26(d)中。最后，使用原始的二值图像和图像  $w$  的“补”，通过逻辑 AND 操作完成分割，如图 10-26(e)所示：

```
>> g2 = g & ~w;
```

注意，图 10-20(e)中的某些物体没有很好地分开。这被称为过分割，这是使用基于分水岭的分割方法时常会出现的问题。下边将讨论克服这一问题的不同技术。

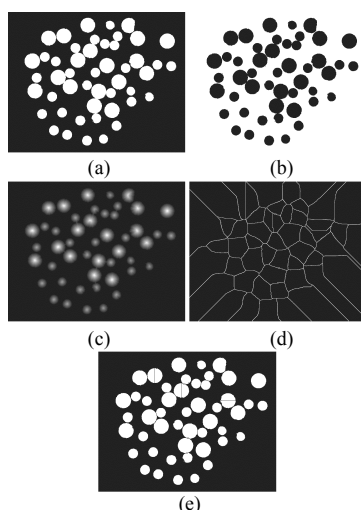


图 10-26 (a) 二值图像；(b) 图像(a)的补；(c) 距离变换；(d) 距离变换的负分水岭脊线；(e) 以黑色叠加在原始二值图像上的分水岭脊线。存在一些过分割，这很明显

## 10.5.2 使用梯度的分水岭分割

在使用针对分割的分水岭变换之前，常常使用梯度幅度对图像进行预处理。梯度幅度图像沿着物体的边缘有较高的像素值，而在其他地方则有较低的像素值。在理想的情况下，分水岭变换可得到沿物体边缘的分水岭脊线。下边的例子说明了这一概念。

### 例 10.16 使用梯度和分水岭变换分割灰度图像

图 10-27(a)显示了一幅包含若干暗斑点的图像  $f$ 。从计算梯度幅度开始，不是采用 10.1 节中讨论的线性滤波方法，就是采用 9.6.1 节中讨论的形态学梯度方法：

```
>> h = fspecial('sobel');
>> fd = tofloat(f);
>> g = sqrt(imfilter(fd, h, 'replicate') .^ 2 + ...
            imfilter(fd, h', 'replicate') .^ 2);
```

图 10-27(b)显示了梯度幅度图像  $g$ 。下一步，计算梯度的分水岭变换并寻找分水岭脊线：

```
>> L = watershed(g);
>> wr = L == 0;
```

正如图 10-27(c)所示，这是不太好的分割结果；有太多的与我们感兴趣的物体边界不对应的分水岭脊线。这是过分割的另一个例子。针对这一问题的解决方法是在计算分水岭变换之前先平滑梯度图像。在这里，我们采用第 9 章描述的闭-开操作：

```
>> g2 = imclose(imopen(g, ones(3,3)), ones(3,3));
>> L2 = watershed(g2);
>> wr2 = L2 == 0;
>> f2 = f;
>> f2(wr2) = 255;
```

前边代码的最后两行以 `wr` 用白线在原始图像上叠加分水岭脊线。图 10-27(d)显示了叠加结果。虽然增强图 10-27(c)的目的达到了,但仍然存在一些附加的脊线,对于决定哪些汇水盆地真正与感兴趣物体有关还是很困难。下面将进一步细致描述基于分水岭的解决这些困难的分割方法。

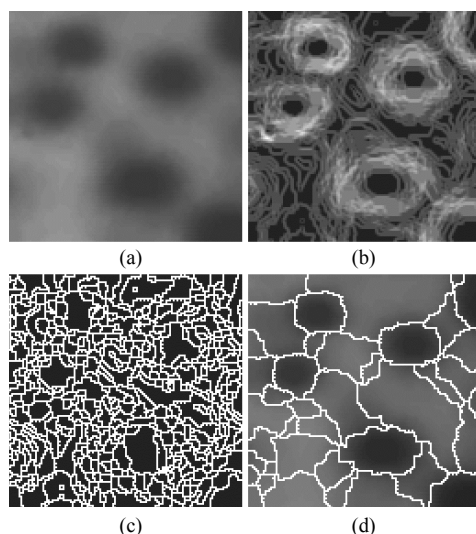


图 10-27 (a) 小斑点的灰度图像; (b) 梯度幅度图像; (c) 显示有些严重过分割的(b)的分水岭变换; (d) 平滑梯度图像的分水岭变换, 一些过分割仍然比较明显。(原图像由巴黎 CMM/Ecole 的 S. Beucher 博士提供)

### 10.5.3 控制标记符的分水岭分割

正如你在 10.5.2 节中看到的那样, 将分水岭变换直接用于梯度图像时, 噪声和梯度的其他局部不规则性常常会导致过分割。由这些因素导致的问题可能会非常严重, 以至于实际结果不可用。按照当前讨论的思路, 这将意味着产生大量的分割区域。实际解决这一问题的一种方法是把附加知识加进分割过程的预处理步骤, 从而限制允许的区域数目。

用于控制过分割的一种方法是基于标记符的概念。标记符是属于一幅图像的连通分量。我们希望有一个内部标记符集合, 它们处在每个感兴趣物体的内部, 而外部标记符集合包含在背景中。这些标记符使用例 10.17 中描述的过程来改进梯度图像。在有关图像处理的文献中, 有各种用于计算内部和外部标记符的建议, 其中包括前面章节描述的线性滤波、非线性滤波以及形态学处理。对于特定的应用, 我们选择的方法依赖于与应用相关的图像的特性。

#### 例 10.17 标记符控制的分水岭分割示例

这个例子把标记符控制的分水岭分割用于图 10-28(a)中的电泳凝胶图像。考虑从计算梯度图像的分水岭变换得到的结果开始, 并且不做任何其他处理。

```
>> h = fspecial('sobel');
>> fd = tofloat(f);
```

```
>> g = sqrt(imfilter(fd, h, 'replicate') .^ 2 + ...
            imfilter(fd, h, 'replicate') .^ 2);
>> L = watershed(g);
>> wr = L == 0;
```

在图10-28(b)中可以看到,结果严重地过分割了。一部分是由于存在大量的小区域,工具箱函数 `imregionalmin` 计算图像中所有的局部小区域的位置,调用语法为:

```
rm = imregionalmin(f)
```

其中, `f` 是灰度图像, `rm` 是二值图像, `rm` 的前景像素标记出局部小区域的位置。可以把 `imregionalmin` 函数用于梯度图像,并查看为什么分水岭函数会产生许多小的汇水盆地:

```
>> rm = imregionalmin(g);
```

图 10-28(c)显示的多数局部小区域的位置非常浅,并且表现的细节与我们的分割问题不相干。为了消除这些无关的小区域,可以使用工具箱函数 `imextendedmin`,计算比近邻点更暗(用某个高度阈值)的图像中“低点”的集合。这个函数的调用语法为:

```
im = imextendedmin(f,h)
```

其中, `f` 是灰度图像, `h` 是高度阈值, `im` 是一幅二值图像, `im` 的前景像素标记了深的局部小区域的位置。在这里,我们用函数 `imextendedmin` 来得到内部标记符的集合:

```
>> im = imextendedmin(f,2);
>> fim = f;
>> fim(im) = 175;
```

最后两行将灰斑点的最小区域位置叠加在原始图像上,如图10-28(d)所示。可以看到,得到的斑点的确很合理地标记了我们想要分割的物体。

接下来,我们必须寻找外部标记符,或者我们确信的属于背景的像素。此处遵循的方法是用已找到的像素来标记背景,这些像素恰好位于内部标记符的中间。令人惊讶的是,通过解决其他分水岭问题,可以做到这些;特别是计算内部标记符图像 `im` 的距离变换的分水岭变换:

```
>> Lim = watershed(bwdist(im));
>> em = Lim == 0;
```

图 10-28(e)显示了二值图像 `em` 中的分水岭脊线。因为这些脊线位于由 `im` 标记的暗斑点之间,所以它们应该是很好的外部标记符。

我们用内部和外部标记符可以采用称为最小覆盖的过程改进梯度图像。最小覆盖技术可以改进灰度图像,因此,局部最小区域仅发生在标记过的位置。如果需要移动所有的局部最小区域,其他像素值就需要上推。工具箱函数 `imimposition` 可实现这种技术,调用语法如下:

```
mp = imimposemin(f, mask)
```

其中, `f` 是灰度图像, `mask` 是二值图像, `mask` 的前景像素标记了输出图像 `mp` 中局部最小区域的期望位置。通过在内部和外部标记符的位置覆盖局部最小区域,可以改进梯度图像:

```
>> g2 = imimposemin(g, im | em);
```



图10-28(f)显示了结果。最后，我们准备计算改进了标记符的梯度图像的分水岭变换，并研究 `ridgelines` 函数的调用结果：

```
>> L2 = watershed(g2);  
>> f2 = f;  
>> f2(L2 == 0) = 255;
```

最后两行是在原始图像上叠加分水岭脊线。一个有很大改进的分割结果如图 10-28(g)所示。

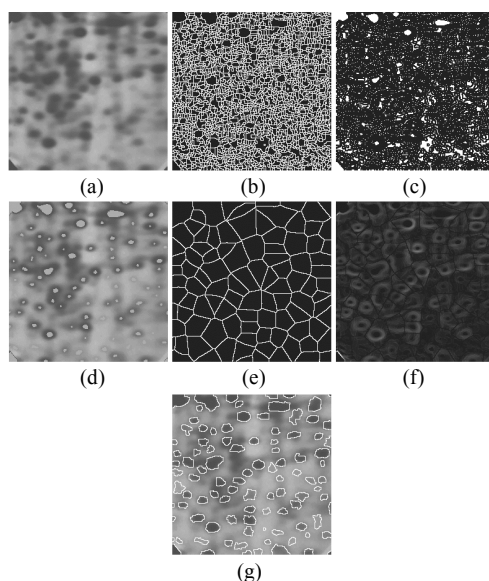


图 10-28 (a) 凝胶图像；(b) 对梯度幅值图像进行分水岭变换的过分割结果；(c) 梯度幅值的局部小区域；(d) 内部标记符；(e) 外部标记符；(f) 改进的梯度幅值；(g) 分割结果。(原图像由巴黎 CMM/Ecole 的 S. Beucher 博士提供)

标记符的选择范围可以从刚才描述的简单过程到更复杂的方法，涉及尺寸、形状、位置、相对距离、纹理内容等等(见第 11 章中关于描述子的内容)。指针是携带对分割有影响的先验知识的标记符。人们常常使用先验知识在每天的视觉中帮助解决分割和高级任务。最为熟悉的便是使用文本。因此，分水岭分割提供可以有效利用这些类型的知识的框架这一事实，是这一方法的突出优点。

## 10.6 小结

图像分割是大多数自动图像模式识别和场景分析问题中基本的预备步骤。正如本章提供的方法和示例中指出的那样，选择一种分割技术而不是其他技术大多由将要考虑的问题的特性决定。本章讨论的方法虽然远没有穷尽，但在实践中却是有代表性的常用技术。