

本书仅提供部分阅读，如需完整版，请联系QQ: 2404062482

提供各种IT类书籍pdf下载，如有需要，请QQ:2404062482

注：链接至淘宝，不喜者勿入！整理那么多资料也不容易，请多多见谅！非诚勿扰！

[点击购买完整版](#)



华章科技

[PACKT]
PUBLISHING

OpenCV的主要开发者和OpenCV社区的主要贡献者携手，深入解析OpenCV技术在计算机视觉项目中的应用，Amazon广泛好评

通过典型计算机视觉项目，系统讲解使用OpenCV技术构建计算机视觉相关应用的各种技术细节、方法和最佳实践，并提供全部实现源码，为读者快速实践OpenCV技术提供翔实指导

华章程序员书库

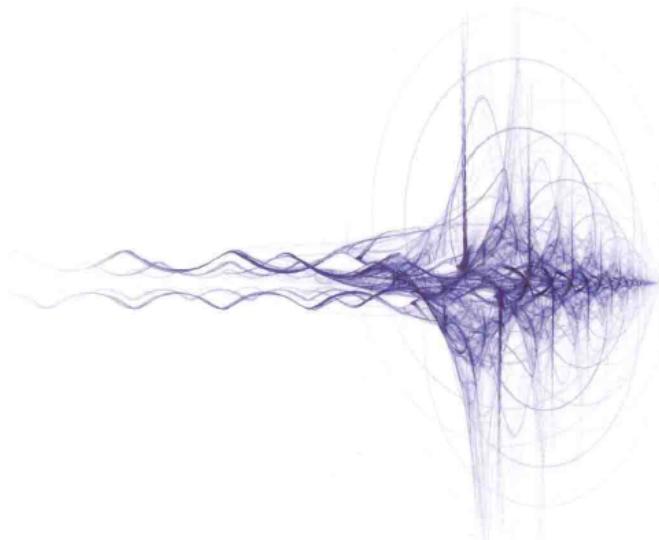
Mastering OpenCV with Practical Computer Vision Projects

深入理解OpenCV

实用计算机视觉项目解析

Daniel Lélis Baggio Shervin Emami David Millán Escrivá
Khvedchenia Ievgen Naureen Mahmood Jason Saragih Roy Shilkrot 著

刘波 译



机械工业出版社
China Machine Press

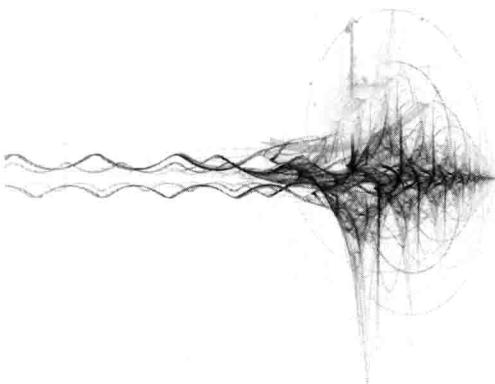
Mastering OpenCV with Practical Computer Vision Projects

深入理解OpenCV

实用计算机视觉项目解析

Daniel Lélis Baggio Shervin Emami David Millán Escrivá
Khvedchenia Ievgen Naureen Mahmood Jason Saragih Roy Shilkrot 著

刘波 译



机械工业出版社
China Machine Press

图书在版编目 (CIP) 数据

深入理解 OpenCV：实用计算机视觉项目解析 / (巴西) 博格等著；刘波译。—北京：机械工业出版社，2014.9
(华章程序员书库)

书名原文：Mastering OpenCV with Practical Computer Vision Projects

ISBN 978-7-111-47818-8

I. 深… II. ①博… ②刘… III. 图像处理软件－程序设计 IV. TP391.41

中国版本图书馆 CIP 数据核字 (2014) 第 204140 号

本书版权登记号：图字：01-2013-7571

Daniel Lélis Baggio, et al: *Mastering OpenCV with Practical Computer Vision Projects* (ISBN: 978-1-84951-782-9).

Copyright © 2012 Packt Publishing. First published in the English language under the title "Mastering OpenCV with Practical Computer Vision Projects".

All rights reserved.

Chinese simplified language edition published by China Machine Press.

Copyright © 2014 by China Machine Press.

本书中文简体字版由 Packt Publishing 授权机械工业出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

深入理解 OpenCV：实用计算机视觉项目解析

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码：100037）

责任编辑：陈佳媛

责任校对：董纪丽

印 刷：三河市宏图印务有限公司

版 次：2014 年 9 月第 1 版第 1 次印刷

开 本：186mm×240mm 1/16

印 张：15 (含彩插 0.25 印张)

书 号：ISBN 978-7-111-47818-8

定 价：59.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：(010) 88378991 88361066

投稿热线：(010) 88379604

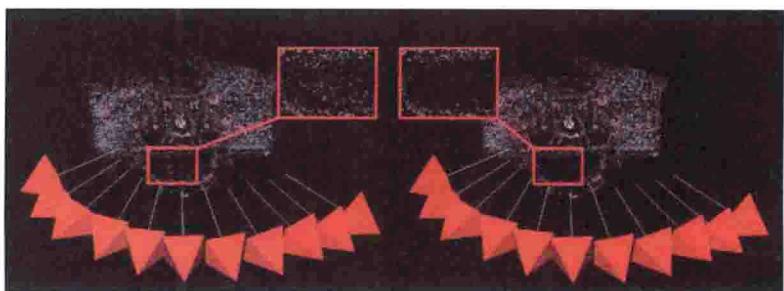
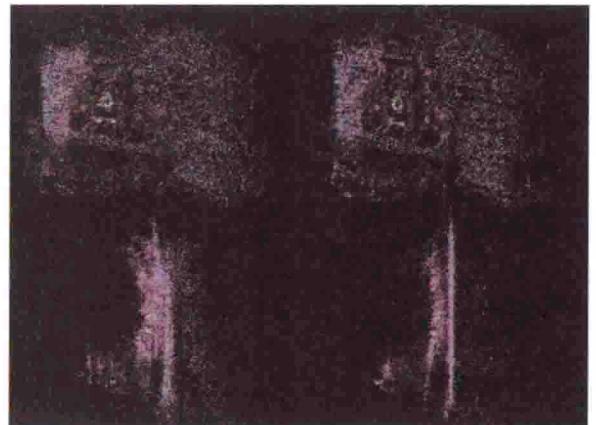
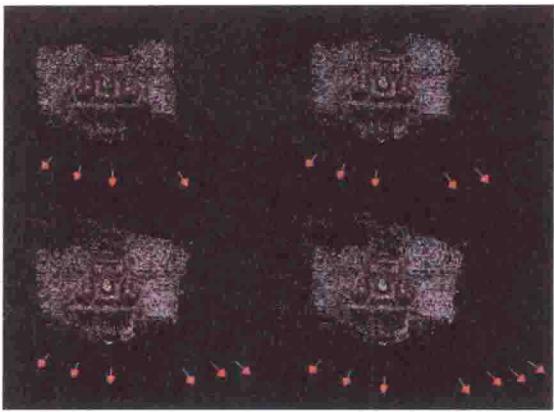
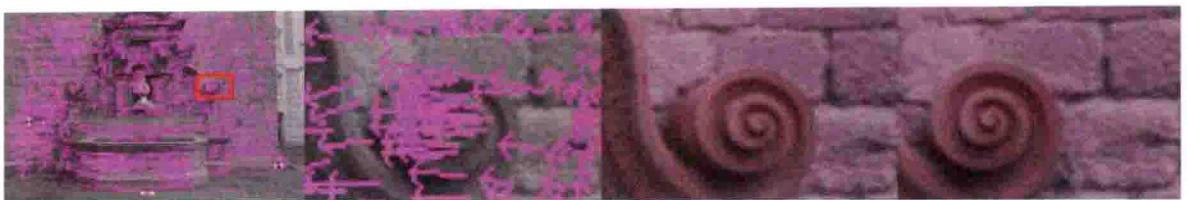
购书热线：(010) 68326294 88379649 68995259

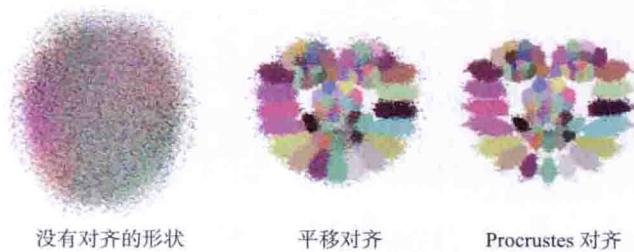
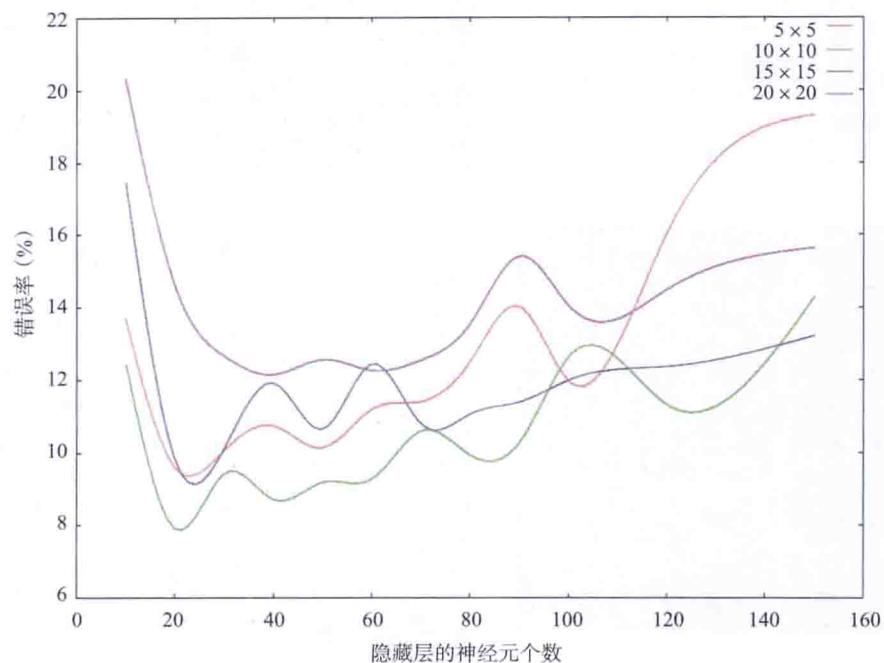
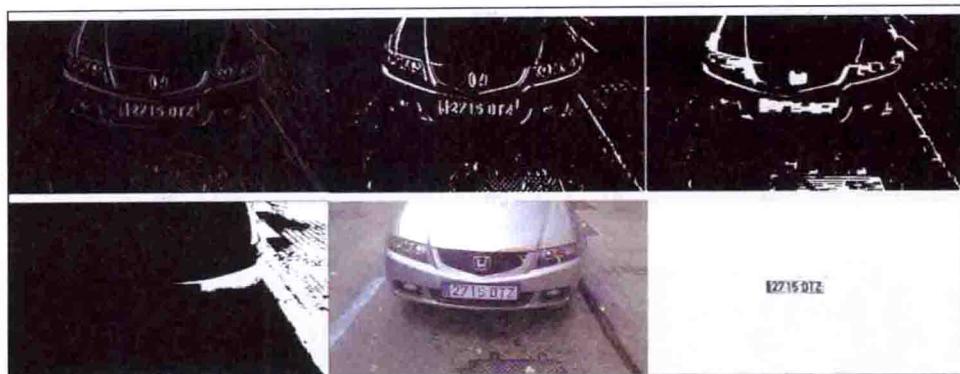
读者信箱：hzjsj@hzbook.com

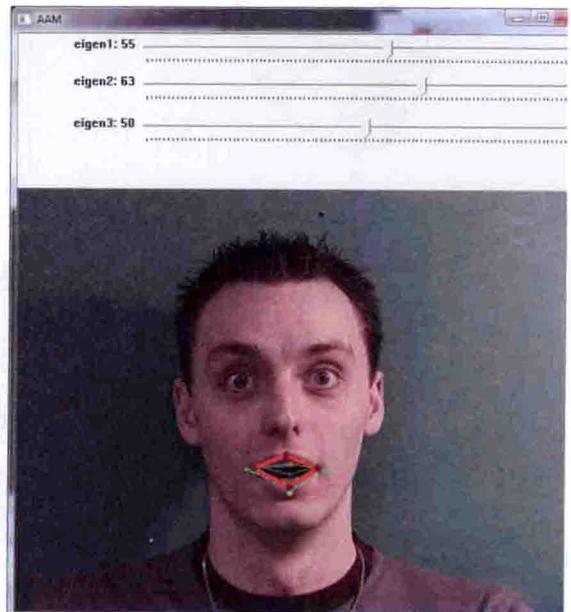
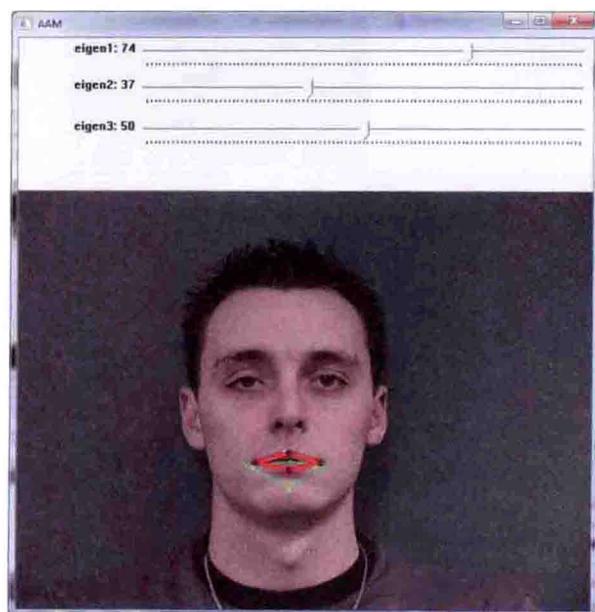
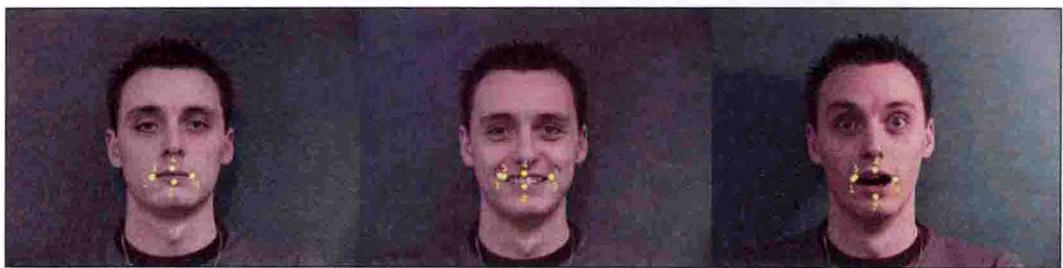
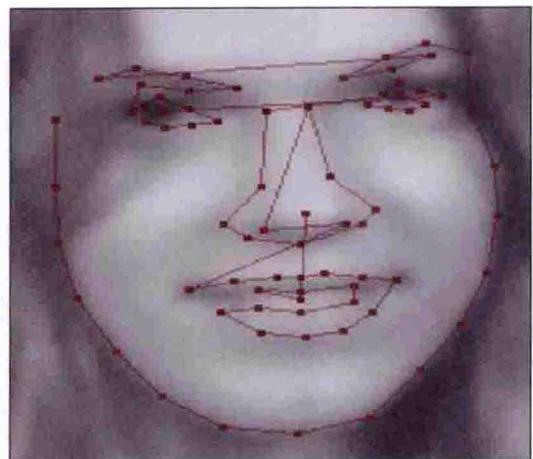
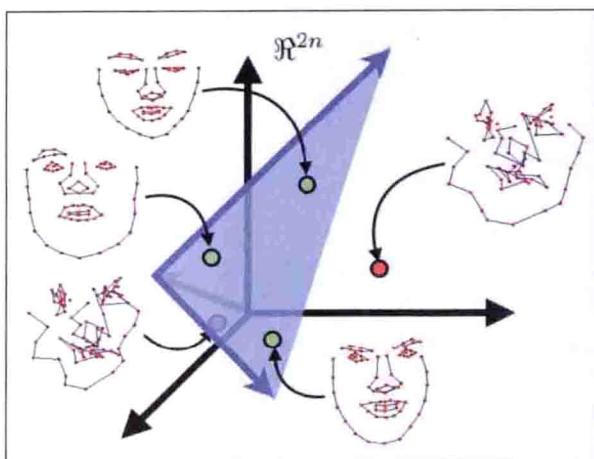
版权所有·侵权必究

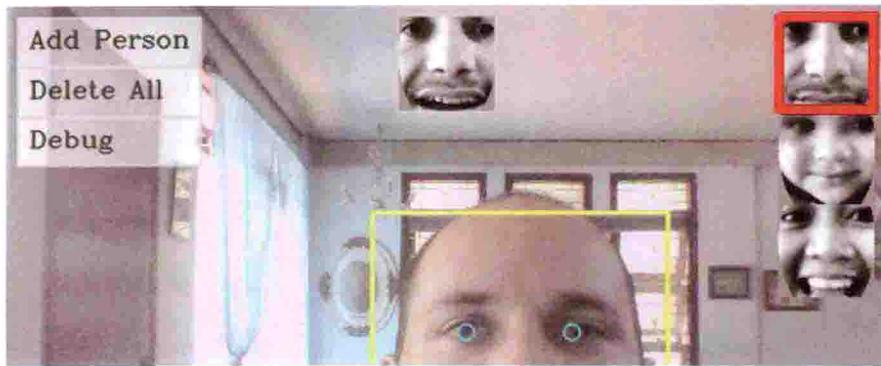
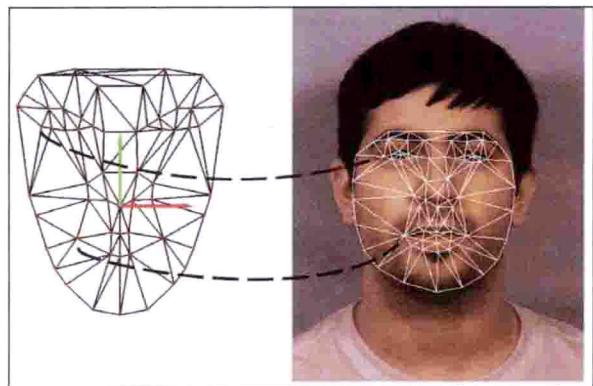
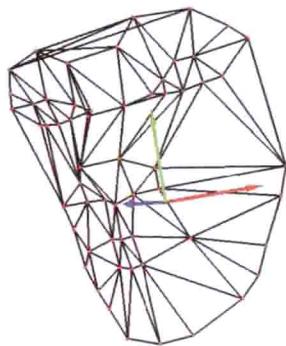
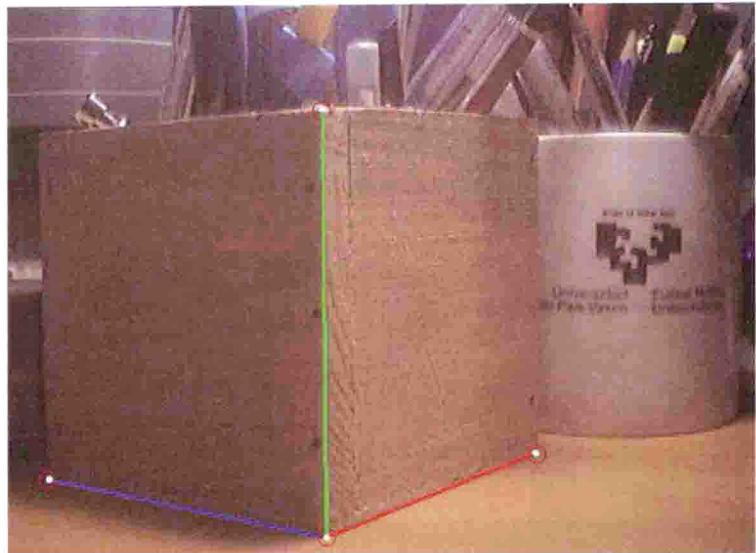
封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东









The Translator's Words 译者序

视觉是人类获取信息的主要来源。图像、视频等视觉信息载体也是当今大数据时代最大的数据源之一，在计算机工程、通信、生物学、医学、军事等领域有着广泛应用。由于计算机视觉涉及多个领域的专业知识，以及视觉对象的复杂性和视觉任务的多样，这使计算机视觉研究很困难。

OpenCV 是开源、跨平台的计算机视觉库，其全称是 Open Source Computer Vision Library。它是由英特尔公司发起并参与开发的，可在商业和研究领域中免费使用。OpenCV 能开发实时的图像处理、运动跟踪、目标检测等程序。

但目前通过实际应用项目来介绍 OpenCV 的书很少。本书通过 8 个典型的计算机视觉项目来介绍 OpenCV 强大、高效的功能。这 8 个项目涵盖了计算视觉的如下领域：基于 iPhone 或 iPad 的增强现实；从运动中得到 3D 结构；车牌识别；人脸识别与跟踪；三维头部姿态估计等。这些项目均用 C/C++ 实现，对于关键代码，作者给出非常详细的介绍。在每章中，作者不但介绍项目的应用背景、整体框架、软件设计方法，同时也深入浅出地介绍了与项目相关的机器学习理论。毫不夸张地讲，这是一本用 OpenCV 来实践计算机视觉应用难得的好书。

翻译本书的过程也是我学习的过程，虽然辛苦但也不觉得累。为了做到专业词汇准确权威，书本内容正确，意译部分既不失原著意境又无偏差，在翻译过程中查阅了大量相关资料。但由于时间和能力有限，书中内容难免存在纰漏。若有问题，读者可通过电子邮件 liubo7971@173.com 与我联系，欢迎一起探讨，共同进步。

本书翻译过程得到如下项目资助：(1) 重庆市教委研究项目“多核正则化机器学习理

论研究”，项目号：KJ130709；（2）重庆工商大学研究项目“基于多核学习的高维数据分析研究”，项目号：2013-56-09；（3）大数据稀疏表示判别字典学习及其应用技术研究项目号：KJ1400612。

感谢河南工业大学信息科学与工程学院的靳小波博士对本书翻译的支持与鼓励，也感谢我的家人，特别感谢我妻子杨雪莉和女儿刘典。虽然翻译本书占用了本应陪她们的大量时间，但她们一直包容并支持我。

Preface 前 言

本书包含 9 章，每章都用一个完整项目作为教程，并提供全部源代码，这些源代码包含了用 C++ 实现的 OpenCV 接口。每章都出自作者在 OpenCV 社区对某一主题所做出的令人瞩目的贡献，OpenCV 的主要开发者也审阅了本书。本书没有解释 OpenCV 函数的基本功能，而是第一本介绍如何使用 OpenCV 来解决整个问题的书，其中包括几个 3D 摄像机项目（增强现实、从运动中恢复 3D 结构、Kinect 交互）和几个面部表情分析项目（例如：皮肤检测、简单的面部和眼部检测、复杂的面部特征跟踪、三维头部姿势估计和人脸识别）。因此，本书能很好地与现有 OpenCV 书籍配合使用。

本书的主要内容

第 1 章包含一个针对桌面应用和 Android 应用的完整教程及相关源代码，这些应用可从真实摄像机图像中自动生成一幅卡通画或图画。在生成过程中，包括皮肤颜色变换在内的几种卡通类型可供选择。

第 2 章包含一个完整教程，该教程讲解如何针对 iPhone 或 iPad 设备来构建基于标记的增强现实 (AR) 应用，并给出每个步骤和源代码的解释。

第 3 章包含一个怎样开发无标记增强现实桌面应用的完整教程，并解释了无标记增强现实 (AR) 和其源代码。

第 4 章通过 OpenCV 实现运动中结构恢复的概念来介绍运动中的结构 (SfM)。读者将学习如何从 2D 图像重构 3D 几何结构以及如何估计摄像机位置。

第 5 章包含一个完整教程及相关源代码，该教程是通过模式识别算法（支持向量机和人工

神经网络) 而建立的自动车牌识别应用。读者将学习如何训练和预测模式识别算法来判断一幅图像是否为车牌。这对通过一组特征来识别字符也有帮助。

第 6 章包含构建一个动态人脸跟踪系统的完整教程及相关源代码，该系统能模拟和跟踪人脸的一些复杂部位。

第 7 章包含理解主动外观模型 (AAM) 和通过 OpenCV 来根据有不同脸部表情的数据帧创建 AAM 的所有背景知识。除此以外，该章解释如何根据 AAM 提供的拟合能力来匹配给定帧。然后采用 POSIT 算法来找到 3D 头部姿态。

第 8 章包含实时人脸识别应用的完整教程和源代码，该应用包括基本的脸部和眼部检测算法，能处理图像中的人脸旋转和不同光照条件。

第 9 章包含一个交互式流体模拟器 (称为流体墙) 的完整开发流程，它采用 Kinect 传感器。该章将解释怎样通过 OpenCV 的光学流方法来使用 Kinect 数据并将其集成到一个流体求解算法中。请读者通过链接 http://www.packtpub.com/sites/default/files/downloads/7829OS_Chapter9_Developing_Fluid_Wall_Using_the_Microsoft_Kinect.pdf 下载第 9 章。

阅读前的准备工作

阅读本书不需要具有计算机视觉的专业知识，但在阅读本书之前应该有良好的 C / C ++ 编程技能和 OpenCV 的基本经验。没有 OpenCV 经验的读者不妨阅读《 Learning OpenCV 》来了解 OpenCV 的特性或阅读《 *OpenCV 2 Cookbook* 》来了解如何以受推崇的 C++ 方式来使用 OpenCV，因为本书将展示如何解决现实问题，并假定读者熟悉 OpenCV 和 C/C++ 开发的基础知识。

除具有 C/C++ 和 OpenCV 的经验外，读者还需要一台计算机和相应的 IDE 环境 (例如：Visual Studio、XCode、Eclipse、QtCreator，它们可以运行在 Windows、Mac 或者 Linux 上)。有些章节有进一步的要求，特别是：

- 为了开发 Android 应用，读者需要一台 Android 设备、多种 Android 开发工具和基本的 Android 开发经验。
- 为了开发 iOS 应用，读者需要 iPhone、iPad 或 iPod Touch 设备、iOS 开发工具 (包含一台苹果电脑、XCode IDE、苹果开发者证书) 以及基本的 iOS 和 Objective-C 的开发

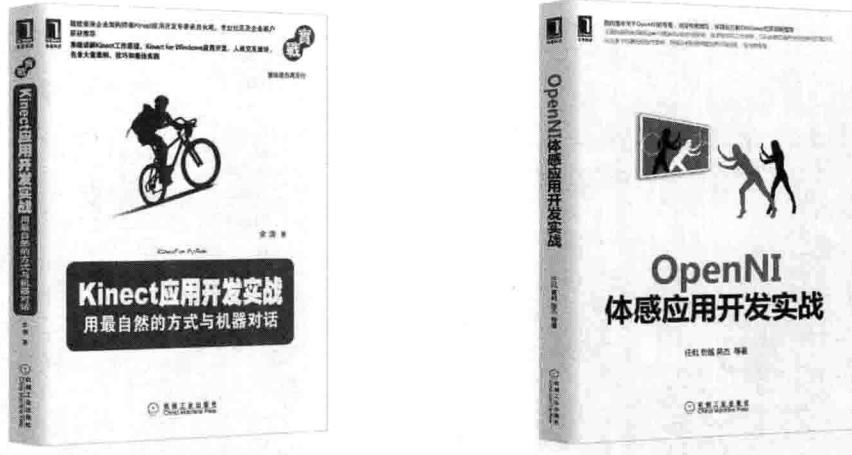
经验。

- 几个桌面项目需要一台与计算机相连的摄像机。任何普通的 USB 摄像机就足够了，但它至少是 100 万像素。
- 某些项目（包括 OpenCV 本身）会使用 CMake 来构建以支持跨平台和跨编译器。需要对构建系统有基本的理解，最好具有跨平台构建的知识。
- 需要了解线性代数方面的知识，例如：向量和矩阵的基本操作以及特征分解。

本书的读者对象

本书对想用基本的 OpenCV 知识来创建实际的计算机视觉项目的开发者来说是一本绝佳指南，此外，对于经验丰富并想获得更多计算机视觉主题的 OpenCV 专家而言也是非常好的一本书。本书向计算机科学相关专业的高年级本科生和研究生，以及研究人员和想用 OpenCV C/C++ 接口来解决实际问题的计算机视频专家提供循序渐进的实用教程。

经典体感应用开发著作



Kinect应用开发实战：用最自然的方式与机器对话

本书由微软资深企业架构师兼Kinect应用开发专家亲自执笔，既系统全面地讲解了Kinect技术的工作原理，又细致深入地讲解了Kinect交互设计、程序开发和企业应用展望。全书不仅包含大量实践指导意义极强的实战案例，而且还包括大量建议和最佳实践，是学习Kinect for Windows应用开发不可多得的参考书。

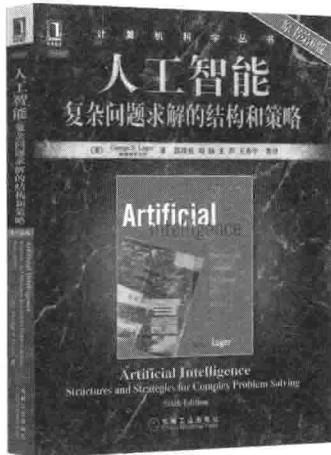
本书分为八大部分：准备篇（引言和第1章），从科幻电影的自然人机交互技术谈起，同时针对虚拟现实、增强现实、多点触摸、语音识别、眼球跟踪、人脸识别、体感操作、脑机界面等人机交互技术，结合一些生动例子来说明这些技术的最新发展动态；原理篇（第2~3章），深入剖析Kinect的硬件组成，从原理上分析Kinect的工作机制，并从计算机视觉技术角度去重点分析“体感操作”背后发生的一切；基础篇（第4~5章），对Kinect for Windows SDK进行框架性的导读，并对Kinect自然人机交互的设计提出有益的归纳和建议；开发篇（第6~9章），从Kinect的开发环境准备，到视频数据、深度数据、骨骼跟踪等开发示例，其中包括一个用Kinect测量身高的有趣示例；实例篇（第10~16章），通过一些生动有趣的应用实例（超级玛丽、水果忍者等）开发，帮助读者快速开发入门；进阶篇（第17~19章），包括姿态识别和手势识别的算法实现，Kinect技术结合3D技术的应用，同时结合Kinect在手术室的原型应用这一综合示例，将交互设计、骨骼跟踪、手势识别、语音识别等关键点“串烧”起来；展望篇（第20~22章），汇集Kinect应用的相关创意和奇思妙想，以及Kinect在医疗、教育、动作捕捉、虚拟现实、增强现实、动漫设计乃至冰川研究等诸多领域的发展前景；附录A是关于Kinect SDK命名空间Microsoft.Kinect的详细介绍，附录B是关于自然人机交互技术、计算机视觉技术的相关开源社区动态。

OpenNI体感应用开发实战

本书是国内首本关于OpenNI的实践性著作，也是首本基于Xtion设备的体感应用开发类著作。具有权威性，由国内体感应用开发领域的专家撰写，华硕官方和CNkinect社区提供支持；具有针对性，深入调研OpenNI社区开发者的需求，据此对内容进行编排；全面且系统，讲解了Xtion和OpenNI的功能使用、技术细节和工作原理，以及体感应用开发的各种知识和技巧；实战性强，包含多个有趣的综合性案例，详细分析和讲解案例的实现过程，确保读者通过本书掌握体感应用开发的技术和方法是本书的宗旨。

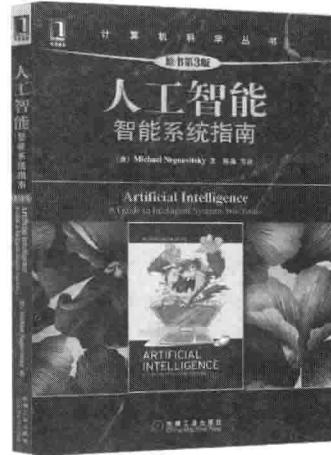
全书共19章，分为五个部分：基础篇（1~3章）介绍了自然人机交互技术、Xtion硬件设备的功能和原理、OpenNI的功能和应用；准备篇（4~6章）讲解了如何搭建OpenNI+Xtion的体感应用开发环境，以及OpenNI的一些基本功能；进阶篇（7~13章）详细讲解了人体骨骼追踪、手势识别、手部追踪、录制与重播、生产节点的建立、声音数据的获取和使用、彩色图像数据的获取和贴图等OpenNI的重要功能及其应用方法；实战篇（14~17章）详细讲解了4个有趣且具有代表性的案例，通过这部分内容读者将能掌握体感应用开发的流程与方法；高级篇（18~19章）讲解了体感应用开发中会用到的多种高级功能，如运动捕捉和OpenNI Unity工具包等。

推荐阅读



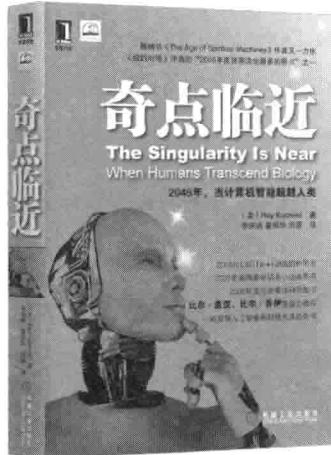
人工智能：复杂问题求解的结构和策略（原书第6版）

作者：George F. Luger ISBN：978-7-111-28345-4 定价：79.00元



人工智能：智能系统指南（原书第3版）

作者：Michael Negnevitsky ISBN：978-7-111-38455-7 定价：79.00元



奇点临近

作者：Ray Kurzweil ISBN：978-7-111-35889-3 定价：69.00元



机器学习

作者：Tom Mitchell ISBN：978-7-111-10993-7 定价：35.00元

目 录 *Contents*

译者序

前 言

第1章 Android系统上的卡通化

和皮肤变换 1

1.1 访问摄像机 2
1.2 桌面应用处理摄像机
视频的主循环 3
1.3 生成黑白素描 4
1.4 生成彩色图像和卡通 5
1.5 用边缘滤波器来生成 “怪物”模式 7
1.6 用皮肤检测来生成 “外星人”造型 8
1.6.1 皮肤检测算法 8
1.6.2 确定用户放置脸的位置 9
1.6.3 皮肤变色器的实现 10
1.7 把桌面应用移植到 Android 系统 14
1.7.1 安装使用 OpenCV 的 Android 项目 14

1.7.2 在 Android NDK 应用中 添加卡通化代码 17
1.7.3 在 Android 系统中显示 保存图像的消息 24
1.7.4 降低素描图像的随机 椒盐噪声 27
1.8 总结 31

第2章 iPhone或iPad上基于 标记的增强现实 32

2.1 使用 OpenCV 创建 iOS 项目 33
2.1.1 添加 OpenCV 框架 34
2.1.2 包含 OpenCV 头文件 35
2.2 应用程序的结构 36
2.3 标记检测 43
2.3.1 标识识别 44
2.3.2 标记编码识别 50
2.4 在三维空间放置标记 53
2.4.1 摄像机标定 53
2.4.2 标记姿态估计 54
2.5 渲染 3D 虚拟物体 56
2.5.1 创建 OpenGL 渲染层 56

2.5.2 渲染 AR 场景	59
2.6 总结	64
2.7 参考文献	64
第3章 无标记的增加现实	65
3.1 基于标记的 AR 与 无标记的 AR	65
3.2 使用特征描述符检测 视频中的任意图像	66
3.2.1 特征提取	67
3.2.2 模式对象定义	69
3.2.3 特征点匹配	69
3.2.4 删除离群值	70
3.2.5 将示例项目各部分 放在一起	76
3.3 模式姿态估计	77
3.3.1 PatternDetector.cpp	77
3.3.2 获取摄像机内矩阵	78
3.4 应用的基础架构	81
3.4.1 ARPipeline.hpp	82
3.4.2 ARPipeline.cpp	82
3.4.3 在 OpenCV 中启用 三维可视化支持	83
3.4.4 使用 OpenCV 来创建 OpenGL 窗口	84
3.4.5 使用 OpenCV 捕获视频	85
3.4.6 渲染增强现实	85
3.4.7 演示应用程序	88
3.5 总结	91
3.6 参考文献	91
第4章 使用OpenCV研究从运动 中恢复结构	92
4.1 从运动中恢复结构的概念	93
4.2 从两幅图像估计摄像机 运动	94
4.2.1 通过丰富的特征描述符 进行点匹配	94
4.2.2 通过光流进行点匹配	96
4.2.3 搜索摄像机矩阵	99
4.3 重构场景	102
4.4 从多视图中重构	105
4.5 重构的细化	108
4.6 用 PCL 来可视化 3D 点云	111
4.7 使用示例代码	113
4.8 总结	114
4.9 参考文献	115
第5章 基于SVM和神经网络 的车牌识别	116
5.1 ANPR 简介	116
5.2 ANPR 算法	118
5.3 车牌检测	119
5.3.1 图像分割	120
5.3.2 分类	125
5.4 车牌号识别	127
5.4.1 OCR 分割	127
5.4.2 特征提取	129
5.4.3 OCR 分类	130
5.4.4 评价	133
5.5 总结	136

第6章 非刚性人脸跟踪	137		
6.1 概述	138	7.2.2 三角剖分	177
6.2 实用工具	139	7.2.3 扭曲三角化结构	179
6.2.1 面向对象设计	139	7.3 模型实例化——试试主动	
6.2.2 数据收集：图像和 视频标注	140	外观模型	180
6.3 几何约束	145	7.4 主动外观模型搜索和拟合	181
6.3.1 Procrustes 分析	146	7.5 POSIT 算法	182
6.3.2 线性形状模型	148	7.5.1 深入理解 POSIT 算法	183
6.3.3 局部 – 全局相结合 的表示	150	7.5.2 POSIT 与头部模型	185
6.3.4 训练与可视化	152	7.5.3 对摄像机或视频文件 进行跟踪	185
6.4 面部特征检测器	154	7.6 总结	187
6.4.1 相关性块模型	155	7.7 参考文献	187
6.4.2 解释全局几何变换	159		
6.4.3 训练与可视化	161		
6.5 人脸检测与初始化	163		
6.6 人脸跟踪	166		
6.6.1 人脸跟踪实现	166		
6.6.2 训练与可视化	168		
6.6.3 通用与专用人脸模型	168		
6.7 总结	169		
6.8 参考文献	169		
第7章 基于AAM和POSIT的 三维头部姿态估计	170		
7.1 主动外观模型概述	171		
7.2 主动形状模型概述	172		
7.2.1 感受 PCA	174		
第8章 基于特征脸或Fisher 脸的人脸识别	189		
8.1 人脸识别与人脸检测介绍	189		
8.1.1 第一步：人脸检测	191		
8.1.2 检测人脸	194		
8.1.3 第2步：人脸预处理	196		
8.1.4 第3步：收集并 训练人脸	204		
8.1.5 第4步：人脸识别	212		
8.1.6 收尾工作：保存和 加载文件	215		
8.1.7 收尾工作：制作一个 漂亮的交互式 GUI	215		
8.2 总结	225		
8.3 参考文献	225		

Android 系统上的卡通化和皮肤变换

本章将介绍如何针对 Android 智能手机和平板电脑编写图像处理滤波器，首先在台式机上（用 C/C++）实现，然后移植到 Android 系统上（用 C/C++ 实现的代码与台式机一样，但 GUI 用 Java 来编写），这是移动设备开发所推崇的方式。本章的主要内容如下：

- 如何将现实生活中的图像转换为素描；
- 如何生成彩色图画并将素描叠加上去来生成卡通画；
- 用恐怖的“怪物”模式来创建坏人形象；
- 通过基本的皮肤检测器和皮肤变色器让人脸变成绿色“外星人”皮肤；
- 如何将桌面应用项目转换为移动设备上的应用程序。

下面的屏幕截图来自于运行在 Android 平板电脑上的卡通化应用程序。



本章希望摄像机拍摄的现实世界看起来像卡通画一样。其基本思路是用某种颜色来填充平整部分，然后用粗线来绘制图像较明显的边缘。也就是说，平整区域变得更加平，而

边缘应变得更加明显。可先检测边缘并对平整区域进行平滑处理，然后增加边缘并从顶部开始来产生一个卡通或漫画效果。

当开发移动设备上的计算机视觉应用时，一种好的方法是先创建一个完整的桌面应用版本，再移植到移动设备上，因为开发和调试桌面应用程序比移动应用程序要容易！因此，本章将以一个完整的卡通化桌面应用开始，读者可用自己喜欢的 IDE（如：Visual Studio、XCode、Eclipse、QtCreator 等）来编写该应用。当其在 PC 上正确运行后，我们将在最后一节介绍如何用 Eclipse 将其移植到 Android（或 iOS）系统中。因此会创建两个不同的项目，它们会共享绝大多数源代码，但有不同的图形用户界面。可创建两个项目都可使用的库，但为了简化起见，可将桌面项目和 Android 项目放在一起，让 Android 项目通过 desktop 文件夹来访问一些文件（cartoon.cpp 和 cartoon.h，它们包含了所有图像处理代码）。例如：

- C:\Cartoonifier_Desktop\cartoon.cpp
- C:\Cartoonifier_Desktop\cartoon.h
- C:\Cartoonifier_Desktop\main_desktop.cpp
- C:\Cartoonifier_Android\...

桌面应用有一个 OpenCV GUI 窗口，初始化摄像机，并在处理摄像机的每帧时都调用 cartoonifyImage () 函数，该函数包含了本章大多数代码。然后在 GUI 窗口显示被处理的图像。与之类似，Android 应用程序也有一个 Android GUI 窗口，会用 Java 程序来初始化摄像机，并调用前面提到的那个用 C++ 实现的 cartoonifyImage () 函数来处理摄像机的每一帧，除此之外还有 Android 菜单并支持触摸输入。本章将从头开始介绍如何创建桌面应用程序，该 Android 应用基于其中的一个 OpenCV Android 示例项目。因此，首先在读者所熟悉的 IDE 中创建桌面应用程序，其中，用于保存 GUI 代码的 main_desktop.cpp 文件将在下一节给出，该文件包含主循环、摄像机功能以及键盘输入，同时，还应创建两个项目共享的 cartoon.cpp 文件。本章大多数代码都会放在函数 cartoonifyImage () 中，该函数保存在 cartoon.cpp 文件中。

1.1 访问摄像机

可简单调用 cv::VideoCapture 对象的 open () 方法（它是访问摄像设备的 OpenCV 方法）来访问计算机的摄像头或摄像机。将默认的摄像机编号 0 传递给此函数。一些计算机有多个摄像机或将 0 作为默认摄像机编号使程序不能运行，解决这类问题的通常做法是将用户指定摄像机编号作为命令行参数，比如：若想指定摄像机编号为 1、2 或 -1，这种方法就比较恰当。

为了让程序在高分辨率摄像机上运行得更快，可用 cv::VideoCapture::set() 将摄像机的分辨率设为 640×480 。



注意：由于摄像机的模式、驱动，或操作系统不同，OpenCV 可能无法改变某些摄像机属性，这对本项目不重要，因此摄像机的属性无法修改也不要担心。

将下面的代码放到 main_desktop.cpp 的 main() 函数中：

```
int cameraNumber = 0;
if (argc > 1)
    cameraNumber = atoi(argv[1]);

// Get access to the camera.
cv::VideoCapture camera;
camera.open(cameraNumber);
if (!camera.isOpened()) {
    std::cerr << "ERROR: Could not access the camera or video!" <<
    std::endl;
    exit(1);
}

// Try to set the camera resolution.
camera.set(cv::CAP_PROP_FRAME_WIDTH, 640);
camera.set(cv::CAP_PROP_FRAME_HEIGHT, 480);
```

在摄像机被初始化后，可通过 cv::Mat 对象（它是 OpenCV 的图像容器）来获取摄像机的图像。可通过使用 C++ 流运算符将 cv::VideoCapture 对象转换成 cv::Mat 对象，由此可获取摄像机的每帧，这就像从控制台获取输入一样。



注意：使用 OpenCV 加载视频文件（例如：AVI 或 MPG 文件）非常容易，这样可代替摄像机。加载视频文件与直接从摄像机获得视频的不同之处在于创建 cv::VideoCapture 对象时，应将视频文件名（例如：camera.open (“my_video.avi”）而不是摄像机编号（例如：camera.open(0)）作为参数。这两种方法创建的 cv::VideoCapture 对象，使用方式都一样。

1.2 桌面应用处理摄像机视频的主循环

如果用 OpenCV 在屏幕上显示一个 GUI 窗口，可调用 cv::imshow() 方法，但每显示一帧图像后必须调用 cv::waitKey(0) 方法，否则 GUI 窗口的图像根本不会更新！cv::waitKey(0) 会使程序暂停除非用户按下任意一个键，若将该函数的参数设为一个正数，例如：waitKey(20) 或更大的值，则程序将会暂停一段时间，这段时间长度为该正数值个毫秒单位。

将这个主循环放到 main_desktop.cpp 中作为实时摄像机应用程序的基础：

```

while (true) {
    // Grab the next camera frame.
    cv::Mat cameraFrame;
    camera >> cameraFrame;
    if (cameraFrame.empty()) {
        std::cerr << "ERROR: Couldn't grab a camera frame." <<
        std::endl;
        exit(1);
    }
    // Create a blank output image, that we will draw onto.
    cv::Mat displayedFrame(cameraFrame.size(), cv::CV_8UC3);

    // Run the cartoonifier filter on the camera frame.
    cartoonifyImage(cameraFrame, displayedFrame);

    // Display the processed image onto the screen.
    imshow("Cartoonifier", displayedFrame);

    // IMPORTANT: Wait for at least 20 milliseconds,
    // so that the image can be displayed on the screen!
    // Also checks if a key was pressed in the GUI window.
    // Note that it should be a "char" to support Linux.
    char keypress = cv::waitKey(20); // Need this to see anything!
    if (keypress == 27) { // Escape Key

        // Quit the program!
        break;
    }
} //end while

```

1.3 生成黑白素描

为了获得视频帧的一幅素描（黑白图画）效果，可用边缘检测滤波器；若要获得一幅彩色图画，可使用边缘保持滤波器（双边滤波器）来进一步平滑平坦区域，同时保持边缘完好。将素描叠加到彩色图画上，便可得到一种卡通效果，该效果与前面最终应用程序的屏幕截图相同。

有许多边缘检测滤波器，例如：Sobel、Scharr、Laplacian 滤波器或 Canny 边缘检测。本章将使用 Laplacian 边缘滤波器，因为同 Sobel 或 Scharr 滤波器相比，它所提取的边缘最接近手工素描，并且它与 Canny 边缘检测一样，可得到清晰的素描效果，但 Canny 边缘检测更容易受视频帧中随机噪声影响，从而使得素描边缘在不同帧之间经常有剧烈变化。

尽管如此，在使用 Laplacian 边缘滤波器之前仍需对图像去噪。可使用中值滤波器来去噪，因为它有很好的去噪效果，同时还可让边缘锐化；而且比双边滤波器的效率高。由于 Laplacian 边缘滤波器只能处理灰度图像，因此必须将 OpenCV 默认的 BGR 格式转换成灰度格式。将下面的代码放在空文件 cartoon.h 的上方，这样使得在访问 OpenCV 和标准 C++

模板时不需要处处都加前缀 cv: 和 std::。

```
// Include OpenCV's C++ Interface
#include "opencv2/opencv.hpp"

using namespace cv;
using namespace std;
```

将下面的代码以及所有剩下的代码都放到 cartoonifyImage() 函数中，该 cartoonifyImage() 函数保存在 cartoon.cpp 文件中。

```
Mat gray;
cvtColor(srcColor, gray, CV_BGR2GRAY);
const int MEDIAN_BLUR_FILTER_SIZE = 7;
medianBlur(gray, gray, MEDIAN_BLUR_FILTER_SIZE);
Mat edges;
const int LAPLACIAN_FILTER_SIZE = 5;
Laplacian(gray, edges, CV_8U, LAPLACIAN_FILTER_SIZE);
```

Laplacian 边缘滤波器能生成不同亮度的边缘，为了使边缘看上去更像素描，可采用二值化阈值来使边缘只有白色或黑色。

```
Mat mask;
const int EDGES_THRESHOLD = 80;
threshold(edges, mask, EDGES_THRESHOLD, 255, THRESH_BINARY_INV);
```

下面这幅图的左边是原图，而右边那幅图是生成的边缘掩码（edge mask），它与素描很像。在生成彩色图画后（稍后介绍），将边缘掩码放到彩色图像上。

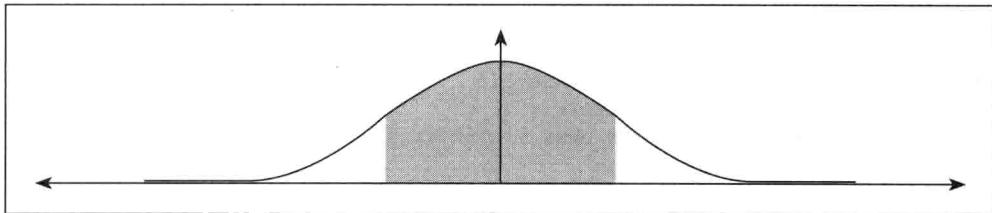


1.4 生成彩色图像和卡通

强大的双边滤波器可平滑平坦区域，同时保持边缘锐化，因此，它作为一个自动的卡通化或图画滤波器很不错，其缺点是效率低（即该滤波器运行的时间要按秒，甚至要按分钟而不是毫秒来计算）。因此，本书采用一些技巧使用户在获得一幅漂亮的卡通化图像时，也可接受其运行时间。最重要的技巧就是在低分辨率下使用双边滤波器，这会得到与高分辨率下相似的效果，但运行速度更快。可将分辨率减少为原图像的四分之一（例如：将图像的宽和高各减少二分之一）。

```
Size size = srcColor.size();
Size smallSize;
smallSize.width = size.width/2;
smallSize.height = size.height/2;
Mat smallImg = Mat(smallSize, CV_8UC3);
resize(srcColor, smallImg, smallSize, 0,0, INTER_LINEAR);
```

可通过多个小型双边滤波器来代替一个大型双边滤波器，从而在较短时间内得到很好的卡通效果。本书通过截断滤波器（见下图）来代替整个滤波器（例如：若钟形曲线有 21 个像素宽，则整个滤波器大小为 21×21 ），截断滤波器是指能达到满意效果的最小滤波器（例如：即便钟形曲线的大小为 21×21 ，但仅使用 9×9 滤波器就可以达到满意效果）。截断滤波器会使用滤波器的主要部分（下图曲线的灰色区域），而不会浪费时间在小部分（下图曲线的白色区域）上，这样会使滤波器的效率提高几倍。



控制双边滤波器可使用的 4 个参数分别是：色彩强度^Θ、位置强度、大小、重复计数。`bilateralFilter()` 函数不能覆盖它的输入值（这称为“就地处理”），因此需要一个临时的 Mat 变量，该变量在一个滤波器中作为输出变量而在另一个滤波器中作为输入变量。

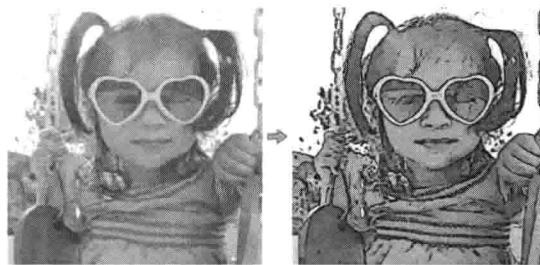
```
Mat tmp = Mat(smallSize, CV_8UC3);
int repetitions = 7; // Repetitions for strong cartoon effect.
for (int i=0; i<repetitions; i++) {
    int ksize = 9; // Filter size. Has a large effect on speed.
    double sigmaColor = 9; // Filter color strength.
    double sigmaSpace = 7; // Spatial strength. Affects speed.
    bilateralFilter(smallImg, tmp, ksize, sigmaColor, sigmaSpace);
    bilateralFilter(tmp, smallImg, ksize, sigmaColor, sigmaSpace);
}
```

注意，该处理过程使用的是缩小后的图像，因此，在处理后需将图像恢复到原来的大小。然后叠加前面得到的边缘掩码。为了将边缘掩码（即素描）叠加到由双边滤波器所产生的图画上（下页第一幅图左边那幅图像），需将目标变量 `dst` 的所有元素全置为 0（即目标变量对应的图像全置为黑色），然后将源图像（变量 `bigImg`）的像素复制到变量 `dst` 中，与“素描”掩码对应的源图像边缘的像素不会被复制。

```
Mat bigImg;
resize(smallImg, bigImg, size, 0,0, INTER_LINEAR);
dst.setTo(0);
bigImg.copyTo(dst, mask);
```

这会得到原图的卡通版，如下右图所示，这就像将素描掩码叠加到彩色图画上。

^Θ 在空间上影响像素的多少。——译者注



1.5 用边缘滤波器来生成“怪物”模式

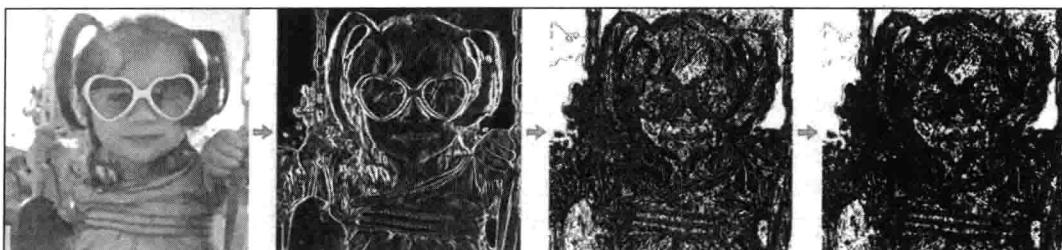
卡通和漫画总有好人物和坏人物，通过对边缘滤波器的恰当组合，可将和蔼的人物画像变成凶恶的人物画像，其技巧是通过使用小的边缘滤波器，这些滤波器能找到图像的各处边缘，然后通过中值滤波器来合并这些边缘。

在去噪后的灰度图像上执行以上操作，之前将原图像转换成灰度图像的代码和 7×7 的中值滤波器都可以用上（下面的第一张图是对灰度图像进行中值平滑滤波后得到的）。接下来不用Laplacian滤波器与二值化阈值这种组合方式，而是沿着x和y方向采用 3×3 的Scharr梯度滤波器（见下面的第二张图），然后再采用截断值很低的二值化阈值方法（见下面的第三张图），最后用 3×3 的中值平滑滤波就可得到“怪物”掩码（见下面的第四张图）。

```

Mat gray;
cvtColor(srcColor, gray, CV_BGR2GRAY);
const int MEDIAN_BLUR_FILTER_SIZE = 7;
medianBlur(gray, gray, MEDIAN_BLUR_FILTER_SIZE);
Mat edges, edges2;
Scharr(srcGray, edges, CV_8U, 1, 0);
Scharr(srcGray, edges2, CV_8U, 1, 0, -1);
edges += edges2; // Combine the x & y edges together.
const int EVIL_EDGE_THRESHOLD = 12;
threshold(edges, mask, EVIL_EDGE_THRESHOLD, 255, THRESH_BINARY_INV);
medianBlur(mask, mask, 3);

```



将现在得到的“怪物”掩码叠加到彩色卡通图像上，该图像就像是用素描掩码叠加到彩色卡通图像上一样。最终效果如下右图所示。



1.6 用皮肤检测来生成“外星人”造型

前面已经生成了素描模式、卡通模式（彩色图画 + 素描掩码）、“怪物”模式（彩色图画 + 怪物掩码），为了更有趣，再来学习一个更复杂的模式：外星人模式。可通过获取人脸的皮肤区域，并将此区域的颜色变为绿色得到该模式。

1.6.1 皮肤检测算法

有很多检测皮肤区域的方法，从简单的基于 RGB（Red-Green-Blue）颜色阈值法、HSV（Hue-Saturation-Brightness）值或颜色直方图计算和重投影，到基于混合模型的复杂机器学习算法等，这些复杂的算法需要在 CIELab 颜色空间对摄像机标定并且需要用人脸样本数据进行离线训练。即便是复杂的算法，对不同的摄像机、光照条件和皮肤类型也不一定有好的鲁棒性。由于本章所需的皮肤检测要运行在移动设备上且不需要进行标定或训练，而且这里用皮肤检测只是为了做一个“好玩”的图像滤波器，因此在这里会使用简单的皮肤检测算法。但需注意，移动设备上的微型摄像机传感器对颜色的反应往往差异很大，而且要在没有标定的情况下对不同肤色的人进行皮肤检测，这需要算法至少比简单颜色阈值法更具有鲁棒性。

例如：一个简单的 HSV 皮肤检测器就会在图像色调相当红时，饱和度较高（但不是很髙），其亮度不太黑或太亮时，会将这些区域的所有像素当成皮肤。除此以外，通常移动设备的摄像机白平衡较差，这使得人的皮肤看上去要偏蓝而不偏红，这也是使用简单 HSV 阈值法的主要问题。

对于这些情况，可采用基于 Haar 或 LBP 级联分类器的人脸识别算法（第 8 章会介绍这些算法），它们具有更好的鲁棒性。在人脸识别的过程中，可检查像素颜色的变化范围，因为人的皮肤像素事先知道，然后对有相似颜色的像素进行全图或邻域扫描，以确定人脸中心。这样做的好处是：不管人的肤色如何，甚至他们的皮肤在图像中偏蓝或偏红，都很有可能找到一些真实皮肤区域。

但是，基于 Haar 或 LBP 级联分类器的人脸识别算法在移动设备上运行缓慢，因此这类

算法可能在实时移动应用中效果不太理想。但对于移动应用来讲，可假定用户能拿着摄像机从近距离直接对准一个人的脸，因为移动设备的摄像机都可拿在用户手上，很方便移动，让用户将人脸放在指定位置并与相机保持一定距离是完全合理的，这样就不需要去检测位置和人脸的大小。这也是一些手机应用会要求用户将脸放在某个位置或用手移动屏幕上的点来确定人脸在图像中的位置的根本原因。因此，可在屏幕中间先画一个人脸轮廓，然后要求用户移动他们的脸到所示位置并通过调节远近来得到恰当的人脸大小。

1.6.2 确定用户放置脸的位置

开始外星人模式的第一步是在相机屏幕上画出人脸轮廓，以便用户知道将人脸放置在这个位置。以固定的宽高比 0.72 来画一个大椭圆，它占整个图像高度的 70%，这个比率不会使人脸看上去太瘦或太胖。

```
// Draw the color face onto a black background.
Mat faceOutline = Mat::zeros(size, CV_8UC3);
Scalar color = CV_RGB(255,255,0); // Yellow.
int thickness = 4;
// Use 70% of the screen height as the face height.
int sw = size.width;
int sh = size.height;
int faceH = sh/2 * 70/100; // "faceH" is the radius of the ellipse.
// Scale the width to be the same shape for any screen width.
int faceW = faceH * 72/100;
// Draw the face outline.
ellipse(faceOutline, Point(sw/2, sh/2), Size(faceW, faceH),
        0, 0, 360, color, thickness, CV_AA);
```

为了让人脸看上去更加明显，可再画两个眼睛的轮廓。为了让所画的眼睛看上去更加真实，不要直接用椭圆作为眼睛轮廓，而是用上椭圆作为眼睛的上半部分，用下椭圆作为眼睛的下半部分，可通过指定起始位置和角度来实现：

```
// Draw the eye outlines, as 2 arcs per eye.
int eyeW = faceW * 23/100;
int eyeH = faceH * 11/100;
int eyeX = faceW * 48/100;
int eyeY = faceH * 13/100;
Size eyeSize = Size(eyeW, eyeH);
// Set the angle and shift for the eye half ellipses.
int eyeA = 15; // angle in degrees.
int eyeYshift = 11;
// Draw the top of the right eye.
ellipse(faceOutline, Point(sw/2 - eyeX, sh/2 - eyeY),
        eyeSize, 0, 180+eyeA, 360-eyeA, color, thickness, CV_AA);
// Draw the bottom of the right eye.
ellipse(faceOutline, Point(sw/2 - eyeX, sh/2 - eyeY - eyeYshift),
        eyeSize, 0, 0+eyeA, 180-eyeA, color, thickness, CV_AA);
```

```
// Draw the top of the left eye.  
ellipse(faceOutline, Point(sw/2 + eyeX, sh/2 - eyeY),  
    eyeSize, 0, 180+eyeA, 360-eyeA, color, thickness, CV_AA);  
// Draw the bottom of the left eye.  
ellipse(faceOutline, Point(sw/2 + eyeX, sh/2 - eyeY - eyeYshift),  
    eyeSize, 0, 0+eyeA, 180-eyeA, color, thickness, CV_AA);
```

可用同样方法来画下嘴唇。

```
// Draw the bottom lip of the mouth.  
int mouthY = faceH * 48/100;  
int mouthW = faceW * 45/100;  
int mouthH = faceH * 6/100;  
ellipse(faceOutline, Point(sw/2, sh/2 + mouthY), Size(mouthW,  
    mouthH), 0, 0, 180, color, thickness, CV_AA);
```

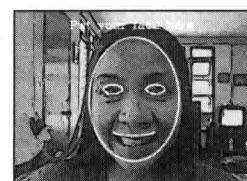
为了提示用户将脸放在指定位置上，可在屏幕上显示一条提示信息。

```
// Draw anti-aliased text.  
int fontFace = FONT_HERSHEY_COMPLEX;  
float fontScale = 1.0f;  
int fontThickness = 2;  
char *szMsg = "Put your face here";  
putText(faceOutline, szMsg, Point(sw * 23/100, sh * 10/100),  
    fontFace, fontScale, color, fontThickness, CV_AA);
```

现在可将已画好的人脸轮廓叠加到要显示的图像上，叠加过程采用 alpha 融合来将卡通化图像与该轮廓结合在一起。

```
addWeighted(dst, 1.0, faceOutline, 0.7, 0, dst, CV_8UC3);
```

最终得到的人脸轮廓如右图所示。这里并没有去检测人脸位置，因为用户可将人脸放置到这个轮廓中。



1.6.3 皮肤变色器的实现

无须去检测皮肤颜色然后看此区域是否有这样的颜色来确定皮肤区域，而是直接使用 OpenCV 的 `floodFill()` 函数。该函数类似于一些图像处理软件中的颜料桶工具。屏幕中间区域就是皮肤像素（因为在拍摄时会将人脸放在人脸轮廓里，而人脸轮廓在整幅图像中间）。为了将整个人脸变成绿色皮肤，只需对图像中心位置的像素进行绿色漫水填充。这样做至少会让人脸的某些部分变成绿色。但现实中的颜色，其饱和度和亮度可能会由于人脸不同部位而有所不同，这会使漫水填充不能覆盖面部的所有像素，除非将其阈值设为很低，但这样做又会覆盖人脸以外的像素。为了解决这个问题，可在图像中心区域不采用简单的漫水填充，而是对人脸区域中 6 个不同的皮肤像素点进行漫水填充。

OpenCV 的 `floodFill()` 函数有一个很好的性质：它会将漫水填充的效果存储到外部图像中而不修改输入图像。这一特性可得到一幅掩码图像，该图像可用来调整皮肤像素的颜色而不需要改变亮度和饱和度，也能在所有皮肤像素都为绿色（这会导致很多人脸细节丢

失)的情况下得到更真实的图像。

在 RGB 颜色空间改变皮肤颜色效果并不好。因为改变皮肤颜色需要脸部图像的亮度变化,但皮肤颜色不允许变化太大,而 RGB 无法从色彩中获取亮度。解决该问题的方法之一是采用 HSV 颜色空间,因为它能从色彩(色度)和多彩(饱和度)中获取亮度。但 HSV 的色度取值以红色开始并以红色结束,若皮肤接近红色,就需要处理小于 10% 和大于 90% 的色度值,因为在这两个范围内都是红色。可使用 Y'CrCb 颜色空间(在 OpenCV 中,它是 YUV 的变种)来解决此问题。Y'CrCb 颜色空间不仅能从颜色中获取亮度,而且对于通常的皮肤颜色其取值唯一。实际上对于大多数摄像机,图像和视频在转换成 RGB 前都使用某种 YUV 颜色空间,所以在很多情形下不需要转换就可得到 YUV 格式的图像。

可对图像进行卡通化之后再用外星人滤波器获得像卡通画一样的外星人模式;也就是说,可获取缩小彩色图像和全尺寸的边缘掩码,其中彩色图像由双边滤波器产生。皮肤检测通常在低分辨率下工作较好,因为它与分析高分辨率像素的近邻的平均值等价(也可看作是用低频信号代替高频噪声信号)。接下来的工作是在对图像进行缩放后进行的,其缩放大小与用双边滤波器处理图像时一样(即只取图像一半的宽度和高度)。下面先将彩色图像转换为 YUV 格式:

```
Mat yuv = Mat(smallSize, CV_8UC3);
cvtColor(smallImg, yuv, CV_BGR2YCrCb);
```

另外还需收缩边缘掩码,使其与彩色图像有相同的尺寸。当存储一幅单独的掩码图像时,若用 OpenCV 的 floodFill() 函数会有困难,即掩码图像应在整个图像周围用 1 个像素作边界,若输入图像大小为 $W \times H$ 个像素,则单独的掩码图像大小为 $(W+2) \times (H+2)$ 个像素。但 floodFill 函数也允许初始化边缘掩码来确保漫水填充算法不会越界,这一特性可阻止漫水填充的区域扩展到人脸外面。因此,需要两幅掩码图像:大小为 $W \times H$ 的边缘掩码图像和大小为 $(W+2) \times (H+2)$ 的边缘掩码图像,该掩码图像包含了图像的边界。可让多个 cv::Mat 对象(或头部)引用同一数据,甚至可让一个 cv::Mat 对象引用另一个 cv::Mat 图像的某一区域。因此不必分配两个单独的图像,然后复制边缘掩码像素给它们,只需分配一个包含边界的掩码图像并创建一个大小为 $W \times H$ 的 cv::Mat 头(仅用它来引用没有边界的掩码图像)。也就是说,仅有一个 $(W+2) \times (H+2)$ 大小的像素数组,但有两个 cv::Mat 对象,一个用来指向大小为 $(W+2) \times (H+2)$ 的图像,另一个用来指向图像中间大小为 $W \times H$ 的区域:

```
int sw = smallSize.width;
int sh = smallSize.height;
Mat mask, maskPlusBorder;
maskPlusBorder = Mat::zeros(sh+2, sw+2, CV_8UC1);
mask = maskPlusBorder(Rect(1,1,sw,sh)); // mask is in maskPlusBorder.
resize(edges, mask, smallSize); // Put edges in both of them.
```

整个边缘掩码(如下左图所示)既有强边缘也有弱边缘;可用二值化阈值法来得到所

需的强边缘（效果见下中图）。为了在边缘间加入一些间隙，可采用形态学算子 dilate() 和 erode() 来删除一些间隙（也会涉及“close”算子），效果见下右图：

```
const int EDGES_THRESHOLD = 80;
threshold(mask, mask, EDGES_THRESHOLD, 255, THRESH_BINARY);
dilate(mask, mask, Mat());
erode(mask, mask, Mat());
```



如前所述，需要对人脸的许多像素点使用漫水填充算法，从而保证人脸图像的各种颜色和整个阴影都能被处理。可在鼻子、脸颊和前额选择 6 个点，如下左图所示。注意，这些点的位置依赖于早期所确定的面部轮廓：

```
int const NUM_SKIN_POINTS = 6;
Point skinPts[NUM_SKIN_POINTS];
skinPts[0] = Point(sw/2, sh/2 - sh/6);
skinPts[1] = Point(sw/2 - sw/11, sh/2 - sh/6);
skinPts[2] = Point(sw/2 + sw/11, sh/2 - sh/6);
skinPts[3] = Point(sw/2, sh/2 + sh/16);
skinPts[4] = Point(sw/2 - sw/9, sh/2 + sh/16);
skinPts[5] = Point(sw/2 + sw/9, sh/2 + sh/16);
```

现在仅需为漫水填充算法找到一些好的下界和上界。漫水填充算法基于 Y'CrCb 颜色空间，因此基本上可决定亮度、红色分量和蓝色分量有多大变化。这里需要包括阴影和高亮显示以及反射在内的亮度变化较大，而颜色不需要变化。

```
const int LOWER_Y = 60;
const int UPPER_Y = 80;
const int LOWER_Cr = 25;
const int UPPER_Cr = 15;
const int LOWER_Cb = 20;
const int UPPER_Cb = 15;
Scalar lowerDiff = Scalar(LOWER_Y, LOWER_Cr, LOWER_Cb);
Scalar upperDiff = Scalar(UPPER_Y, UPPER_Cr, UPPER_Cb);
```

调用 floodFill() 时，为了存储外部掩码而必须为 flags 参数指定 FLOODFILL_MASK_ONLY 选项，该参数的其他选项均用默认值。

```
const int CONNECTED_COMPONENTS = 4; // To fill diagonally, use 8.
const int flags = CONNECTED_COMPONENTS | FLOODFILL_FIXED_RANGE \
| FLOODFILL_MASK_ONLY;
Mat edgeMask = mask.clone(); // Keep a copy of the edge mask.
// "maskPlusBorder" is initialized with edges to block floodFill().
```