

Broadview

站在浪潮之巅，零距离接触最前沿的计算机视觉编程技术  
Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

全面涵盖OpenCV2、OpenCV3双版本的核心编程技巧

附赠OpenCV2、OpenCV3双版本总计200余个配套示例程序源代码

# OpenCV3 编程入门

毛星云 冷雪飞 等编著

 電子工業出版社  
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY  
<http://www.phei.com.cn>

# OpenCV3 编程入门

毛星云 冷雪飞 王碧辉 吴松森 编著

電子工業出版社  
Publishing House of Electronics Industry

## 内 容 简 介

OpenCV 在计算机视觉领域扮演着重要的角色。作为一个基于开源发行的跨平台计算机视觉库，OpenCV 实现了图像处理和计算机视觉方面的很多通用算法。本书以当前最新版本的 OpenCV 最常用最核心的组件模块为索引，深入浅出地介绍了 OpenCV2 和 OpenCV3 中的强大功能、性能，以及新特性。书本配套的 OpenCV2 和 OpenCV3 双版本的示例代码包中，含有总计两百多个详细注释的程序源代码与思路说明。读者可以按图索骥，按技术方向进行快速上手和深入学习。

本书要求读者具有基础的 C/C++ 知识，适合研究计算机视觉以及相关领域的在校学生和老师、初次接触 OpenCV 但有一定 C/C++ 编程基础的研究人员，以及已有过 OpenCV 1.0 编程经验，想快速了解并上手 OpenCV2、OpenCV3 编程的计算机视觉领域的专业人员。本书也适合于图像处理、计算机视觉领域的业余爱好者、开源项目爱好者做为通向新版 OpenCV 的参考手册之用。

本书配套的【示例程序】、【.exe 可执行文件】、【书内彩图】的下载链接可通过扫描本书封底后勒口的二维码获取。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

## 图书在版编目（CIP）数据

OpenCV3 编程入门 / 毛星云等编著. —北京：电子工业出版社，2015.2

ISBN 978-7-121-25331-7

I. ①O… II. ①毛… III. ①图象处理软件—程序设计 IV. ①TP391.41

中国版本图书馆 CIP 数据核字（2014）第 310454 号

责任编辑：陈晓猛

印 刷：北京京科印刷有限公司

装 订：三河市皇庄路通装订厂

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：787×1092 1/16 印张：29.25 字数：653 千字

版 次：2015 年 2 月第 1 版

印 次：2015 年 2 月第 1 次印刷

定 价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 [zlts@phei.com.cn](mailto:zlts@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

服务热线：（010）88258888。

# 前　　言

计算机视觉是一个近几年来日臻成熟的领域。随着运算性能强劲而又价格实惠的计算设备的不断问世，创建复杂的图像应用从未像今天这般容易。OpenCV 在计算机视觉领域扮演着重要的角色，它是一个基于开源发行的跨平台计算机视觉库，实现了图像处理和计算机视觉方面的很多通用算法。自 1999 年问世以来，OpenCV 已经被计算机视觉领域的学者和开发者视为首选工具，并成为了计算机视觉领域最有力的研究工具之一。

OpenCV 最初由 Intel 的一个小组进行开发。在一系列的 beta 版本后，OpenCV 1.0 正式版本终于在 2006 年 10 月 19 日发布。

2009 年 10 月 1 日，OpenCV 2.0 问世，它带来了全新的 C++ 接口，将 OpenCV 的能力无限放大。在 2.0 的时代，OpenCV 增加了全新的平台支持，包括 iOS 和 Andriod，通过 CUDA 和 OpenCL 实现了 GPU 加速，为 Python 和 Java 用户提供了接口，基于 Github 和 Buildbot 构建了充满艺术感的持续集成的系统，所以才有了被全世界的很多公司和学校所采用的稳定易用的 OpenCV 2.4.x。

2014 年 8 月 21 日，OpenCV 3.0 Alpha 发布，它带来了全新的项目架构的改变，宣告了计算机视觉新时代的来临。和其他大型项目一样，OpenCV3 抛弃了整体统一架构，使用内核+插件的架构形式，让自身主体更加稳定，而附加的库则可以更加灵活多变，以保持高速的发展与迭代。

本书源自于笔者在 CSDN 上连载的名为“OpenCV 入门教程”的系列博客文章，自 2014 年 2 月 24 日发表第一篇以来，得到了广大 OpenCV 爱好者的广泛关注与支持，累计阅读量突破了 40 多万人次。不少读者强烈希望将这些内容集结成书，并加入更多新的内容。于是，经过笔者半年的笔耕不辍，便有了现在这本书的诞生。

作为一本入门级的 OpenCV 编程教材，本书以详细注释的程序代码为主线，以新版 OpenCV 最核心的 core、HighGUI、improc 和 feature2d 这 4 个组件的相关函数、类和数据结构为出发点，详细讲解了学习新版本 OpenCV 中会遇到了各种问题，并提供了详尽的实战代码作为参考。本书的写作初衷是让更多的使用者能熟练使用采用新版 C++ 接口的 OpenCV2 或 OpenCV3，了解 OpenCV2 和 OpenCV3 的诸多细节上的区别，以推动新版 OpenCV 在世界范围内的普及。

## 本书的内容安排

本书分为 4 个部分、11 个章节，现将内容梗概列举如下。

第 1 章 邂逅 OpenCV：介绍 OpenCV 的周边概念，分析 OpenCV 的基本架

构，讲解 OpenCV3 的新特性。本章重点讲解了 OpenCV 的下载、安装与配置过程；在配置完成后，带领大家正式领略 OpenCV 的魅力，讲解了 4 个 OpenCV 图像处理小程序，并指导大家学习如何使用 OpenCV 操作视频和调用摄像头。

**第 2 章 启程前的认知准备：**进行 OpenCV 官方例程的引导学习与赏析，讲解如何编译 OpenCV 的源代码，并引入了对一些周边概念的认知。

**第 3 章 HighGUI 图形用户界面初步：**对图像的载入、显示和输出到文件进行详细地分析，讲解 OpenCV 中滑动条的创建和使用，以及如何用鼠标进行交互操作。

**第 4 章 OpenCV 数据结构与基本绘图：**讲解 OpenCV 中常用的数据结构以及基本的绘图操作。

**第 5 章 core 组件进阶：**讲解 core 模块的一些进阶知识点，如操作图像中的像素、图像混合、分离颜色通道、调节图像的对比度和亮度、进行离散傅里叶变换，以及输入输出 XML 和 YAML 文件。

**第 6 章 图像处理：**学习各种利用 OpenCV 进行图像处理的方法，包括属于线性滤波的方框滤波、均值滤波与高斯滤波，属于非线性滤波的中值滤波、双边滤波；两种基本形态学操作——膨胀与腐蚀；5 种高级形态学滤波操作——开运算、闭运算、形态学梯度、顶帽以及黑帽；此外，还有漫水填充算法、图像金字塔、图像缩放、阈值化。

**第 7 章 图像变换：**讲解多种类型的图像变换方法。包括利用 OpenCV 进行边缘检测所用到的 canny 算子、sobel 算子，Laplace 算子以及 scharr 滤波器；进行图像特征提取的霍夫线变换、霍夫圆变换，重映射和仿射变换以及直方图均衡化。

**第 8 章 图像轮廓与图像分割修复：**讲解如何查找轮廓并绘制轮廓，如何寻找物体的凸包，使用多边形来包围轮廓，以及计算一个图像的矩。此外还介绍了分水岭算法和图像修补操作的实现方法。

**第 9 章 直方图与匹配：**讲解图像直方图相关的编程技巧，以及直方图对比、反向投影和模板匹配技术。

**第 10 章 角点检测：**讲解 Harris 角点检测和 Shi-Tomasi 角点检测，以及一种亚像素角点检测方法。

**第 11 章 特征检测与匹配：**使用 OpenCV2 讲解并实现了 SURF、SIFT 和 ORB 特征检测方法，并在 FLANN 特征匹配的基础上，进一步实现了利用 Homography 映射来找出已知物体。

## 适合阅读本书的读者

- 研究计算机视觉以及相关领域的在校学生和老师

本书拥有详实的内容，注释详尽的代码，会是助你通过 OpenCV 来研习计算

机视觉理论、撰写论文、通过毕业设计、完成科研项目的得力工具。同时，本书适合作为大学计算机视觉课程的教学用书。

- 初次接触 OpenCV、有一定 C/C++ 编程基础的研究人员

作为一本定位为快速入门新版 OpenCV 标准的编程教程，本书需要的仅仅是一些简单的 C/C++ 编程语言基础。如果你已经具备了这些基础，并对计算机视觉感兴趣，那么本书正是为你所准备的。

- 已经有过 OpenCV 1.0 编程经验，想快速了解并上手 OpenCV2、OpenCV3 编程的计算机视觉领域的专业人员

如果你曾经使用过 OpenCV 1.0，或者研读过 OpenCV 1.0 时代的经典著作《Learning OpenCV》，本书会让你倍感亲切。你会发现新版 OpenCV 带了更多强大和便利的特性，让你事半功倍，如虎添翼。

- 想拥有一本新版 OpenCV 接口工具书的计算机视觉爱好者

本书中将自 OpenCV2 以来（包括 OpenCV3）的常用类和函数进行了详细地讲解，并在附录中提供了“书本核心函数清单”以便检索。你会在书中快速查找到你需要用到的函数、数据结构和类的用法。

- 想拥有海量的详细注释的 OpenCV2、OpenCV3 示例程序代码的 OpenCV 爱好者

本书包含 OpenCV2 版的 95 个书本主线示例程序源代码、21 个附赠示例程序源代码，OpenCV3 版的 95 个书本主线示例程序源代码。OpenCV2、OpenCV3 两版代码提供分开下载。这些程序代码都经过详细而有条理的注释，并提供可以独立运行的.exe 文件供快速查看程序效果，方便查看和检索。你会在海量的示例程序中找到你需要的参考代码，从而加速你的研究和学习。

- 图像处理、计算机视觉领域的业余爱好者

海阔凭鱼跃，天高任鸟飞，计算机视觉领域的宝库任你探索。

- 开源项目爱好者

OpenCV 作为一个完全免费并开源代码开发的计算机视觉代码库，有总计上百万行的源代码供你研究学习，本书将是引导你学习它们的良师益友。

## 本书的示例程序说明

本书的示例程序最初都在 OpenCV 2.4.9（2014 年 4 月 15 日面世）版本下开发，书稿初版也是基于 OpenCV 2.4.9 而写。在书稿写作和修订过程中，恰逢 OpenCV 3.0 Alpha（2014 年 8 月 21 日）和 OpenCV3 Beta（2014 年 11 月 11 日）的发布，所以本书在审校和修订过程中（2014 年 12 月 1 日），决定站在浪潮之巅，以 OpenCV2 为主，加入 OpenCV3 的诸多特性，让这本书可以同时胜任 OpenCV2 和 OpenCV3 两个版本教材的角色。这也是为什么本书会有 OpenCV2 和 OpenCV3

两个独立版本的示例程序的原因。

两个版本、详细注释的 100 多个示例程序源代码是本书的灵魂，现将示例程序的相关情况概括如下。

- 本书包含 OpenCV2 版的 95 个书本主线示例程序源代码、21 个附赠示例程序源代码，以及 OpenCV3 版的 95 个书本主线示例程序源代码。
- OpenCV2、OpenCV3 两版代码提供分开下载。
- OpenCV2 版的示例程序在 Windows7 64 位旗舰版、Visual Studio 2010 、 OpenCV 2.4.9 的环境下开发与测试，理论上支持 OpenCV2 系列的所有版本的编译运行。
- OpenCV3 版的示例程序在 Windows7 64 位旗舰版、Visual Studio 2010 、 OpenCV 3.0 beta 的环境下开发与测试，理论上支持目前已经发布的 OpenCV3 全版本。
- 程序源代码都经过详细而有条理的注释。
- 额外提供可以独立运行的.exe 文件供快速查看程序效果，并方便检索。

本书配套示例代码的下载方式有以下几种。

- 扫描本书封面后勒口的二维码，得到下载地址。
- 在作者博客 ([http://blog.csdn.net/poem\\_qianmo](http://blog.csdn.net/poem_qianmo)) 中单击相应的书本维护博文里贴出的下载链接。
- 在电子工业出版社的官方网站 (<http://www.phei.com.cn>) 中进行下载。
- 直接用搜索引擎搜索“《OpenCV3 编程入门》书本配套源代码”，找到对应的下载地址进行下载。

## 致谢

首先需要感谢我的导师，南京航空航天大学冷雪飞教授的知遇之恩，她也亲自参与撰写了本书的部分章节。在攻读硕士学位阶段，如果没有导师的谆谆教诲，我不会和 OpenCV 相遇，也就不会有此书的出版。

感谢我的同门师兄王碧辉与吴松森参与撰写本书的部分章节，他们为本书的完善做出了卓越的贡献。

感谢 OpenCV 开发团队为世界研发出如此强大且稳定、易用的计算机开源视觉库，并持续不断地对其进行维护与更新。

感谢父母将我养育成人，感谢家人们的嘘寒问暖，你们是我最坚强的后盾。

感谢母校南京航空航天大学赐予我一颗不甘平庸、上下求索的心。

感谢南京航空航天大学的戴泉晨老师对本书出版所做出的帮助与支持。

感谢国家自然科学基金青年科学基金项目“新型单定子二自由度超声电机及其驱动的智能云台系统的关键技术研究”(项目批准号：51205193)对本书理论研究方面提供的经费支持。

感谢电子工业出版社博文视点的陈晓猛和丁一琼编辑为本书的出版所做的大量的工作，他们对出版物的专业和严谨的态度给我留下了深刻的印象。

最后，需要感谢我博客上的众多读者们，是你们对这本书的期待和热情的留言让我有了完成这本书的动力和勇气。

## 交流与勘误

由于编者水平有限，书籍即使经过了多次的校对，也难免会有疏漏之处。希望书本前的你，能够热心地指出书本中错误，以便在这本书下一版印刷的时候，能以一个更完美更严谨的样子，呈现在大家的面前。另外，你要相信你不是一个人在战斗，在作者的博客中，可以找到与自己志同道合的众多喜欢计算机视觉编程技术的爱好者们。我们可以一同交流，共同学习进步。

最后，愿大家在本书的帮助下，都能很好地入门和掌握新版 OpenCV。

愿本书能为新版 OpenCV 在国内的普及以及在世界范围内的发展，献上绵薄之力。

作者博客地址：[http://blog.csdn.net/poem\\_qianmo](http://blog.csdn.net/poem_qianmo)

作者联系邮箱：[happylifemxy@163.com](mailto:happylifemxy@163.com)

作者新浪微博：@浅墨\_毛星云

浅墨

2014 年 12 月于南京

# 目 录

第一部分 快速上手 OpenCV ..... 1

第 1 章 邂逅 OpenCV ..... 3

1.1	OpenCV 周边概念认知 ..... 4
1.1.1	图像处理、计算机视觉与 OpenCV ..... 4
1.1.2	OpenCV 概述 ..... 4
1.1.3	起源及发展 ..... 5
1.1.4	应用概述 ..... 6
1.2	OpenCV 基本架构分析 ..... 7
1.3	OpenCV3 带来了什么 ..... 11
1.3.1	项目架构的改变 ..... 11
1.3.2	将 OpenCV2 代码升级到 OpenCV3 报错时的一些策略 ..... 12
1.4	OpenCV 的下载、安装与配置 ..... 14
1.4.1	预准备：下载和安装集成开发环境 ..... 14
1.4.2	第一步：下载和安装 OpenCV SDK ..... 15
1.4.3	第二步：配置环境变量 ..... 16
1.4.4	第三步：工程包含（include）目录的配置 ..... 17
1.4.5	第四步：工程库（lib）目录的配置 ..... 21
1.4.6	第五步：链接库的配置 ..... 22
1.4.7	第六步：在 Windows 文件夹下加入 OpenCV 动态链接库 ..... 25
1.4.8	第七步：最终测试 ..... 26
1.4.9	可能遇到的问题和解决方案 ..... 27
1.5	快速上手 OpenCV 图像处理 ..... 28
1.5.1	第一个程序：图像显示 ..... 29
1.5.2	第二个程序：图像腐蚀 ..... 30
1.5.3	第三个程序：图像模糊 ..... 31
1.5.4	第四个程序：canny 边缘检测 ..... 32
1.6	OpenCV 视频操作基础 ..... 34
1.6.1	读取并播放视频 ..... 34
1.6.2	调用摄像头采集图像 ..... 35
1.7	本章小结 ..... 38

第 2 章 启程前的认知准备	39
2.1 OpenCV 官方例程引导与赏析	40
2.1.1 彩色目标跟踪: Camshift	41
2.1.2 光流: optical flow	42
2.1.3 点追踪: lkdemo	43
2.1.4 人脸识别: objectDetection	43
2.1.5 支持向量机引导	44
2.2 开源的魅力: 编译 OpenCV 源代码	45
2.2.1 下载安装 CMake	45
2.2.2 使用 CMake 生成 OpenCV 源代码工程的解决方案	46
2.2.3 编译 OpenCV 源代码	50
2.3 “opencv.hpp” 头文件认知	53
2.4 命名规范约定	54
2.5 argc 与 argv 参数解惑	56
2.5.1 初识 main 函数中的 argc 和 argv	56
2.5.2 argc、argv 的具体含义	57
2.5.3 Visual Studio 中 main 函数的几种写法说明	58
2.5.4 总结	59
2.6 格式输出函数 printf()简析	59
2.6.1 格式输出: printf()函数	59
2.6.2 示例程序: printf 函数的用法示例	60
2.7 智能显示当前使用的 OpenCV 版本	61
2.8 本章小结	61
第 3 章 HighGUI 图形用户界面初步	63
3.1 图像的载入、显示和输出到文件	64
3.1.1 OpenCV 的命名空间	64
3.1.2 Mat 类简析	64
3.1.3 图像的载入与显示概述	65
3.1.4 图像的载入: imread()函数	65
3.1.5 图像的显示: imshow()函数	66
3.1.6 关于 InputArray 类型	67
3.1.7 创建窗口: namedWindow()函数	67
3.1.8 输出图像到文件: imwrite()函数	68
3.1.9 综合示例程序: 图像的载入、显示与输出	70
3.2 滑动条的创建和使用	73
3.2.1 创建滑动条: createTrackbar()函数	73
3.2.2 获取当前轨迹条的位置: getTrackbarPos()函数	76
3.3 鼠标操作	76
3.4 本章小结	80

第二部分 初探 core 组件.....	83
第 4 章 OpenCV 数据结构与基本绘图.....	85
4.1 基础图像容器 Mat.....	86
4.1.1 数字图像存储概述 .....	86
4.1.2 Mat 结构的使用 .....	86
4.1.3 像素值的存储方法 .....	88
4.1.4 显式创建 Mat 对象的七种方法 .....	89
4.1.5 OpenCV 中的格式化输出方法 .....	91
4.1.6 输出其他常用数据结构.....	94
4.1.7 示例程序：基础图像容器 Mat 类的使用 .....	95
4.2 常用数据结构和函数 .....	95
4.2.1 点的表示： Point 类 .....	96
4.2.2 颜色的表示： Scalar 类.....	96
4.2.3 尺寸的表示： Size 类 .....	96
4.2.4 矩形的表示： Rect 类 .....	97
4.2.5 颜色空间转换： cvtColor()函数 .....	98
4.2.6 其他常用的知识点 .....	100
4.3 基本图形的绘制 .....	100
4.3.1 DrawEllipse()函数的写法 .....	101
4.3.2 DrawFilledCircle()函数的写法 .....	102
4.3.3 DrawPolygon()函数的写法.....	102
4.3.4 DrawLine()函数的写法.....	103
4.3.5 main 函数的写法 .....	104
4.4 本章小结 .....	106
第 5 章 core 组件进阶 .....	107
5.1 访问图像中的像素 .....	108
5.1.1 图像在内存之中的存储方式 .....	108
5.1.2 颜色空间缩减 .....	108
5.1.3 LUT 函数： Look up table 操作 .....	109
5.1.4 计时函数 .....	110
5.1.5 访问图像中像素的三类方法 .....	110
5.1.6 示例程序 .....	114
5.2 ROI 区域图像叠加&图像混合 .....	114
5.2.1 感兴趣区域： ROI .....	115
5.2.2 线性混合操作 .....	116
5.2.3 计算数组加权和： addWeighted()函数 .....	117
5.2.4 综合示例： 初级图像混合 .....	120

5.3 分离颜色通道、多通道图像混合.....	125
5.3.1 通道分离: split()函数.....	125
5.3.2 通道合并: merge()函数.....	126
5.3.3 示例程序: 多通道图像混合.....	127
5.4 图像对比度、亮度值调整.....	131
5.4.1 理论依据.....	131
5.4.2 访问图片中的像素.....	131
5.4.3 示例程序: 图像对比度、亮度值调整.....	132
5.5 离散傅里叶变换.....	135
5.5.1 离散傅里叶变换的原理.....	135
5.5.2 dft()函数详解.....	136
5.5.3 返回 DFT 最优尺寸大小: getOptimalDFTSize()函数.....	137
5.5.4 扩充图像边界: copyMakeBorder()函数.....	137
5.5.5 计算二维矢量的幅值: magnitude()函数.....	138
5.5.6 计算自然对数: log()函数.....	138
5.5.7 矩阵归一化: normalize()函数.....	138
5.5.8 示例程序: 离散傅里叶变换.....	139
5.6 输入输出 XML 和 YAML 文件.....	144
5.6.1 XML 和 YAML 文件简介.....	144
5.6.2 FileStorage 类操作文件的使用引导.....	144
5.6.3 示例程序: XML 和 YAML 文件的写入.....	147
5.6.4 示例程序: XML 和 YAML 文件的读取.....	148
5.7 本章小结.....	150
<b>第三部分 掌握 imgproc 组件 .....</b>	<b>151</b>
<b>第 6 章 图像处理 .....</b>	<b>153</b>
6.1 线性滤波: 方框滤波、均值滤波、高斯滤波.....	154
6.1.1 平滑处理.....	154
6.1.2 图像滤波与滤波器.....	154
6.1.3 线性滤波器的简介.....	155
6.1.4 滤波和模糊.....	155
6.1.5 邻域算子与线性邻域滤波.....	155
6.1.6 方框滤波 (box Filter) .....	156
6.1.7 均值滤波 .....	157
6.1.8 高斯滤波 .....	159
6.1.9 线性滤波相关 OpenCV 源码剖析.....	160
6.1.10 OpenCV 中 GaussianBlur 函数源码剖析.....	164
6.1.11 线性滤波核心 API 函数 .....	165
6.1.12 图像线性滤波综合示例.....	170

6.2 非线性滤波：中值滤波、双边滤波.....	175
6.2.1 非线性滤波概述 .....	175
6.2.2 中值滤波 .....	175
6.2.3 双边滤波 .....	177
6.2.4 非线性滤波相关核心 API 函数 .....	178
6.2.5 OpenCV 中的 5 种图像滤波综合示例.....	181
6.3 形态学滤波（1）：腐蚀与膨胀.....	187
6.3.1 形态学概述 .....	187
6.3.2 膨胀 .....	188
6.3.3 腐蚀 .....	189
6.3.4 相关 OpenCV 源码分析溯源 .....	190
6.3.5 相关核心 API 函数讲解 .....	191
6.3.6 综合示例：腐蚀与膨胀.....	195
6.4 形态学滤波（2）：开运算、闭运算、形态学梯度、顶帽、黑帽.....	198
6.4.1 开运算 .....	199
6.4.2 闭运算 .....	200
6.4.3 形态学梯度 .....	200
6.4.4 顶帽 .....	201
6.4.5 黑帽 .....	202
6.4.6 形态学滤波 OpenCV 源码分析溯源.....	203
6.4.7 核心 API 函数：morphologyEx().....	205
6.4.8 各形态学操作使用范例一览 .....	206
6.4.9 综合示例：形态学滤波.....	208
6.5 漫水填充 .....	214
6.5.1 漫水填充的定义 .....	214
6.5.2 漫水填充法的基本思想.....	214
6.5.3 实现漫水填充算法：floodFill 函数.....	214
6.5.4 综合示例：漫水填充.....	216
6.6 图像金字塔与图片尺寸缩放 .....	223
6.6.1 引言 .....	223
6.6.2 关于图像金字塔 .....	223
6.6.3 高斯金字塔 .....	225
6.6.4 拉普拉斯金字塔 .....	226
6.6.5 尺寸调整：resize()函数 .....	227
6.6.6 图像金字塔相关 API 函数 .....	230
6.6.7 综合示例：图像金字塔与图片尺寸缩放 .....	234
6.7 阈值化 .....	237
6.7.1 固定阈值操作：Threshold()函数 .....	238
6.7.2 自适应阈值操作：adaptiveThreshold()函数 .....	239

6.7.3	示例程序：基本阈值操作	240
6.8	本章小结	244
<b>第 7 章 图像变换</b>		<b>247</b>
7.1	基于 OpenCV 的边缘检测	248
7.1.1	边缘检测的一般步骤	248
7.1.2	canny 算子	248
7.1.3	sobel 算子	253
7.1.4	Laplacian 算子	256
7.1.5	scharr 滤波器	259
7.1.6	综合示例：边缘检测	262
7.2	霍夫变换	267
7.2.1	霍夫变换概述	267
7.2.2	OpenCV 中的霍夫线变换	268
7.2.3	霍夫线变换的原理	268
7.2.4	标准霍夫变换：HoughLines()函数	270
7.2.5	累计概率霍夫变换：HoughLinesP()函数	272
7.2.6	霍夫圆变换	274
7.2.7	霍夫梯度法的原理	275
7.2.8	霍夫梯度法的缺点	276
7.2.9	霍夫圆变换：HoughCircles()函数	276
7.2.10	综合示例：霍夫变换	278
7.3	重映射	281
7.3.1	重映射的概念	281
7.3.2	实现重映射：remap()函数	282
7.3.3	基础示例程序：基本重映射	283
7.3.4	综合示例程序：实现多种重映射	285
7.4	仿射变换	289
7.4.1	认识仿射变换	289
7.4.2	仿射变换的求法	290
7.4.3	进行仿射变换：warpAffine()函数	291
7.4.4	计算二维旋转变换矩阵：getRotationMatrix2D()函数	292
7.4.5	示例程序：仿射变换	292
7.5	直方图均衡化	295
7.5.1	直方图均衡化的概念和特点	296
7.5.2	实现直方图均衡化：equalizeHist()函数	297
7.5.3	示例程序：直方图均衡化	298
7.6	本章小结	300

<b>第 8 章 图像轮廓与图像分割修复</b>	303
8.1 查找并绘制轮廓	304
8.1.1 寻找轮廓: findContours()函数	304
8.1.2 绘制轮廓: drawContours()函数	305
8.1.3 基础示例程序: 轮廓查找	306
8.1.4 综合示例程序: 查找并绘制轮廓	308
8.2 寻找物体的凸包	312
8.2.1 凸包	312
8.2.2 寻找凸包: convexHull()函数	313
8.2.3 基础示例程序: 凸包检测基础	313
8.2.4 综合示例程序: 寻找和绘制物体的凸包	315
8.3 使用多边形将轮廓包围	318
8.3.1 返回外部矩形边界: boundingRect()函数	318
8.3.2 寻找最小包围矩形: minAreaRect()函数	318
8.3.3 寻找最小包围圆形: minEnclosingCircle()函数	318
8.3.4 用椭圆拟合二维点集: fitEllipse()函数	319
8.3.5 逼近多边形曲线: approxPolyDP()函数	319
8.3.6 基础示例程序: 创建包围轮廓的矩形边界	319
8.3.7 基础示例程序: 创建包围轮廓的圆形边界	321
8.3.8 综合示例程序: 使用多边形包围轮廓	324
8.4 图像的矩	327
8.4.1 矩的计算: moments()函数	328
8.4.2 计算轮廓面积: contourArea()函数	328
8.4.3 计算轮廓长度: arcLength()函数	328
8.4.4 综合示例程序: 查找和绘制图像轮廓矩	329
8.5 分水岭算法	333
8.5.1 实现分水岭算法: watershed()函数	334
8.5.2 综合示例程序: 分水岭算法	334
8.6 图像修补	338
8.6.1 实现图像修补: inpaint()函数	340
8.6.2 综合示例程序: 图像修补	341
8.7 本章小结	343
<b>第 9 章 直方图与匹配</b>	345
9.1 图像直方图概述	346
9.2 直方图的计算与绘制	347
9.2.1 计算直方图: calcHist()函数	347
9.2.2 找寻最值: minMaxLoc()函数	348
9.2.3 示例程序: 绘制 H—S 直方图	348

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

9.2.4	示例程序：计算并绘制图像一维直方图.....	350
9.2.5	示例程序：绘制 RGB 三色直方图.....	352
9.3	直方图对比 .....	355
9.3.1	对比直方图：compareHist()函数.....	355
9.3.2	示例程序：直方图对比.....	356
9.4	反向投影 .....	360
9.4.1	引言 .....	360
9.4.2	反向投影的工作原理.....	360
9.4.3	反向投影的作用 .....	361
9.4.4	反向投影的结果 .....	361
9.4.5	计算反向投影：calcBackProject()函数 .....	361
9.4.6	通道复制：mixChannels()函数 .....	362
9.4.7	综合程序：反向投影.....	363
9.5	模板匹配 .....	367
9.5.1	模板匹配的概念与原理.....	367
9.5.2	实现模板匹配：matchTemplate()函数.....	367
9.5.3	综合示例：模板匹配.....	369
9.6	本章小结 .....	373
<b>第四部分 深入 feature2d 组件 .....</b>		<b>375</b>
<b>第 10 章 角点检测 .....</b>		<b>377</b>
10.1	Harris 角点检测 .....	378
10.1.1	兴趣点与角点 .....	378
10.1.2	角点检测 .....	378
10.1.3	harris 角点检测 .....	379
10.1.4	实现 Harris 角点检测：cornerHarris()函数 .....	379
10.1.5	综合示例：harris 角点检测与绘制.....	381
10.2	Shi-Tomasi 角点检测 .....	384
10.2.1	Shi-Tomasi 角点检测概述 .....	384
10.2.2	确定图像强角点：goodFeaturesToTrack()函数.....	384
10.2.3	综合示例：Shi-Tomasi 角点检测 .....	385
10.3	亚像素级角点检测 .....	388
10.3.1	背景概述 .....	388
10.3.2	寻找亚像素角点：cornerSubPix()函数 .....	389
10.3.3	综合示例：亚像素级角点检测.....	389
10.4	本章小结 .....	392
<b>第 11 章 特征检测与匹配 .....</b>		<b>395</b>
11.1	SURF 特征点检测 .....	396

11.1.1 SURF 算法概览.....	396
11.1.2 SURF 算法原理.....	396
11.1.3 SURF 类相关 OpenCV 源码剖析.....	400
11.1.4 绘制关键点: drawKeypoints()函数.....	401
11.1.5 KeyPoint 类 .....	402
11.1.6 示例程序: SURF 特征点检测.....	402
11.2 SURF 特征提取 .....	405
11.2.1 绘制匹配点: drawMatches()函数.....	405
11.2.2 BruteForceMatcher 类源码分析.....	407
11.2.3 示例程序: SURF 特征提取.....	408
11.3 使用 FLANN 进行特征点匹配 .....	410
11.3.1 FlannBasedMatcher 类的简单分析.....	410
11.3.2 找到最佳匹配: DescriptorMatcher::match 方法.....	411
11.3.3 示例程序: 使用 FLANN 进行特征点匹配.....	411
11.3.4 综合示例程序: FLANN 结合 SURF 进行关键点的描述和匹配 .....	413
11.3.5 综合示例程序: SIFT 配合暴力匹配进行关键点描述和提取 .....	417
11.4 寻找已知物体 .....	420
11.4.1 寻找透视变换: findHomography()函数 .....	421
11.4.2 进行透视矩阵变换: perspectiveTransform()函数 .....	421
11.4.3 示例程序: 寻找已知物体.....	422
11.5 ORB 特征提取 .....	425
11.5.1 ORB 算法概述 .....	425
11.5.2 相关概念认知.....	425
11.5.3 ORB 类相关源码简单分析.....	426
11.5.4 示例程序: ORB 算法描述与匹配 .....	426
11.6 本章小结 .....	430
附录 .....	433
A1 配套示例程序清单 .....	433
A2 随书额外附赠的程序一览.....	436
A3 书本核心函数清单 .....	439
A4 Mat 类函数一览 .....	442
A4.1 构造函数: Mat::Mat .....	442
A4.2 析构函数 Mat::~Mat .....	444
A4.3 Mat 类成员函数 .....	444
主要参考文献 .....	447

# 第一部分

## 快速上手 OpenCV

欢迎来到奇妙的 OpenCV 编程世界，作为本书的作者，非常荣幸能与你共同站在浪潮之巅，一起踏上精彩绝伦的图像处理编程之旅，探索和学习这座近几年来蓬勃发展的图像处理与计算机视觉的大宝藏。

正所谓，授之以鱼，不如授之以渔。作为 OpenCV 入门的开篇，此部分旨在让你快速认识和了解新版本 OpenCV，并提供了一些指导大家跳出书本桎梏，依靠官方文档自学的思路，比如对 OpenCV 官方例程的引导与赏析、手把手教你阅读和编译 OpenCV 总计 44 万多行的源代码。这让你不仅能快速入门新版 OpenCV，更能学到在书本之外找到新的、进阶的学习 OpenCV 资源和代码的方法。

通过这部分的学习，相信可以对新版本 OpenCV 有一个全面而立体的认识。

**第一部分包含如下三个章节：**

---

- 第 1 章 邂逅 OpenCV
- 第 2 章 启程前的认知准备
- 第 3 章 HighGUI 图形用户界面初步

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

# 第 1 章

## 邂逅 OpenCV

### 导读

---

本章中，你将学到：

- 什么是计算机视觉
- 什么是 OpenCV
- 计算机视觉与 OpenCV 的联系
- OpenCV 的起源发展
- OpenCV 的应用领域
- 对 OpenCV、OpenAL、OpenGL 的辨析
- OpenCV 的基本架构分析
- OpenCV 的下载、安装与配置
- 通过几个程序快速上手 OpenCV
- 如何进行 OpenCV 视频操作
- 如何用 OpenCV 调用摄像头

## 1.1 OpenCV 周边概念认知

### 1.1.1 图像处理、计算机视觉与 OpenCV

图像处理（Image Processing）是用计算机对图像进行分析，以达到所需结果的技术，又称影像处理。图像处理技术一般包括图像压缩，增强和复原，匹配、描述和识别 3 个部分。图像处理一般指数字图像处理（Digital Image Processing）。其中，数字图像是指用工业相机、摄像机、扫描仪等设备经过拍摄得到的一个大的二维数组。该数组的元素称为像素，其值称为灰度值。而数字图像处理是通过计算机对图像进行去除噪声、增强、复原、分割、提取特征等处理的方法和技术。

计算机视觉（Computer Vision）是一门研究如何使机器“看”的科学，具体地说，就是是指用摄影机和电脑代替人眼对目标进行识别、跟踪和测量等机器视觉，并进一步做图形处理，用电脑处理使之成为更适合人眼观察或传送给仪器检测的图像的一门学科。作为一门科学学科，计算机视觉研究相关的理论和技术，试图建立能够从图像或者多维数据中获取“信息”的人工智能系统。因为感知可以看做是从感官信号中提取信息，所以计算机视觉也可以看做是研究如何使人工系统从图像或多维数据中“感知”的科学。

图像处理和计算机视觉的区别在于：图像处理侧重于“处理”图像——如增强，还原，去噪，分割，等等；而计算机视觉重点在于使用计算机（也许是可移动式的）来模拟人的视觉，因此模拟才是计算机视觉领域的最终目标。

而 OpenCV（Open Source Computer Vision Library），是一个基于开源发行的跨平台计算机视觉库，它实现了图像处理和计算机视觉方面的很多通用算法，已经成为了计算机视觉领域最有力的研究工具之一。

### 1.1.2 OpenCV 概述

OpenCV 的全称是 Open Source Computer Vision Library，直译就是“开源计算机视觉库”。取代表开源的单词“Open”、“Computer”的首字母“C”以及“Vision”的首字母“V”，组合命名为“OpenCV”。

OpenCV 于 1999 年由 Intel 建立，如今由 Willow Garage 提供支持。它是一个基于开源发行的跨平台计算机视觉库，可以运行在 Linux、Windows、Mac OS、Android、iOS、Maemo、FreeBSD、OpenBSD 等操作系统上。OpenCV 由一系列 C 函数和 C++类构成，轻量且高效。强大的 OpenCV 除了用 C/C++语言进行开发和使用之外，还支持使用 C#、Ch、Ruby 等编程语言，同时提供了对 Python、Ruby、MATLAB 等语言的接口，实现了图像处理和计算机视觉方面的很多通用算法。图 1.1 所示为 OpenCV 的 Logo 图形。

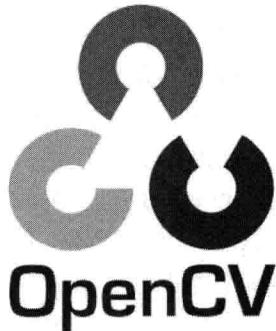


图 1.1 OpenCV Logo

OpenCV 的设计目标是执行速度尽量快，主要关注实时应用。它采用优化的 C/C++ 代码编写，能够充分利用多核处理器的优势，其主要目标是构建一个简单易用的计算机视觉框架，以帮助开发人员更便捷地设计更复杂的计算机视觉相关应用程序。

OpenCV 由一系列 C 函数和 C++ 类构成，拥有包括 500 多个 C 函数的跨平台的中高层 API。它不依赖于其他的外部库——但如果有需要，也可以使用某些外部库。OpenCV 覆盖了计算机视觉的许多应用领域，如工厂产品检测、医学成像、信息安全、用户界面、摄像机标定、立体视觉和机器人等。因为计算机视觉和机器学习密切相关，所以 OpenCV 还提供 MLL（Machine Learning Library）机器学习库。该机器学习库主要用于统计方面的模式识别和聚类（clustering）。MLL 除了用在视觉相关的任务中，还可以方便地应用于其他机器学习场合。

OpenCV 官方主页：<http://opencv.org>

OpenCV Github 主页：<https://github.com/Itseez/opencv>

OpenCV 开发版 Wiki 主页：<http://code.opencv.org>

### 1.1.3 起源及发展

OpenCV 项目最早由 Intel 公司于 1999 年启动，旨在促进 CPU 密集型应用。为了达到这一目的，Intel 启动了多个项目，包括实时光线追踪和三维显示墙。

在 Intel 的性能库团队的帮助下，OpenCV 实现了一些核心代码和算法，并发给 Intel 俄罗斯的库团队。因此，OpenCV 的诞生，是在与软件性能库团队的合作下，发源于 Intel 的研究中心，并在俄罗斯得到实现和优化。

在开始时，OpenCV 有以下三大目标，这三大目标也说明了 OpenCV 的初衷。

- 为基本的视觉应用提供开放且优化的源代码，以促进视觉研究的发展，从而有效地避免“闭门造车”。
- 通过提供一个通用的架构来传播视觉知识，开发者可以在这个架构上继续开展工作，所以代码应该是非常易读且可改写的。
- OpenCV 库采用的协议不要求商业产品继续开放代码，这使得可移植的、

性能被优化的代码可以自由获取，可以促进基于视觉的商业应用的发展。

而如今看来，OpenCV 出现的目的是提供一个可普遍适用的计算机视觉库。OpenCV 的第一个预览版本于 2000 年在 IEEE Conference on Computer Vision and Pattern Recognition 公开，并且后续提供了 5 个测试版本。1.0 版本于 2006 年发布。OpenCV 的第二个主要版本是 2009 年 10 月的 OpenCV 2.0。该版本的主要更新包括 C++ 接口，更容易、更安全的模式，新的函数，以及对现有实现代码的优化（特别是多核心方面）。现在，约定每 6 个月就会有一个官方 OpenCV 版本发布，并且是由某商业公司赞助的独立小组进行开发。在 2012 年 8 月，对 OpenCV 的支持由一个非营利性组织（OpenCV.org）来提供，并保留了一个开发者网站和用户网站。值得一提的是，现在它也集成了对 CUDA 的支持。

2014 年 8 月 21 日，随着 OpenCV3 时代的第一个版本 OpenCV3.0 Alpha 在官网首页现身并提供下载，这个强大的计算机视觉库正式迎来了全新的纪元。



本书的示例程序最初都是以 OpenCV 2.4.9（2014 年 4 月 15 日面世）版本为开发环境的，书稿初版也是基于 OpenCV2.4.9 而写。在书稿写作和修订过程中，恰逢 OpenCV3.0 Alpha（2014 年 8 月 21 日）和 OpenCV3 Beta（2014 年 11 月 11 日）的发布，所以本书在审校和修订过程中（2014 年 11 月 29 日），决定站在浪潮之巅，以 OpenCV3 为主，加入 OpenCV3 的诸多特性，让这本书可以同时胜任 OpenCV2 和 OpenCV3 两个版本教材的角色。这也是为什么本书会有 OpenCV2 和 OpenCV3 两个独立版本的示例程序的原因。其中，OpenCV2 版的示例程序原则上支持 OpenCV 2 系列的所有版本的编译运行；OpenCV3 版的示例程序原则上支持目前已经发布的 OpenCV3 全版本。在书本正文贴出的代码中，以 OpenCV3 代码为主角，且将 OpenCV2 和 OpenCV3 代码有明显区别的地方进行了对比书写，方便大家学习。

#### 1.1.4 应用概述

自从 OpenCV 在 1999 年 1 月发布 beta 版本开始，它就被广泛应用于许多领域、产品和研究成果，具体包括卫星地图和电子地图的拼接、扫描图像对齐、医学图像去噪（消噪或滤波）、图像中的物体分析、安全和入侵检测系统、自动监视和安全系统，以及制造业中的产品质量检测系统、摄像机标定、军事应用、无人飞行器、无人汽车和无人水下机器人。此外，还可以将视觉识别技术用在声谱图上，用 OpenCV 进行声音和音乐识别。

OpenCV 提供的视觉处理算法非常丰富。由于它部分以高效的 C 语言编写，加上其开源的特性，处理得当，不需要添加新的外部支持也可以完整地编译链接生成执行程序，所以很多研究者用它来做算法的移植。OpenCV 的代码经过适当改写可以正常地运行在 DSP 系统和单片机系统中，这种移植在高等院校中经常作为相关专业本科生的毕业设计或者研究生、博士生的课题选题。

OpenCV 可用于解决如下领域的问题：

- 人机交互
- 物体识别
- 图像分区
- 人脸识别
- 动作识别
- 运动跟踪
- 机器人

## 1.2 OpenCV 基本架构分析

初学 OpenCV 时，先了解一下 OpenCV 的整体模块架构，再重点学习和突破自己感兴趣的部分，就会有得心应手，一览众山小的学习体验。基于这个理念，笔者决定将此节的内容放在本书的首章，让大家高屋建瓴，在学习之初就能把握住新版 OpenCV 的脉络。

在此需要提示一下，若想和笔者一起在计算机中进入到 OpenCV 的文件夹目录，可以先跳到 1.3 节，先学习如何进行 OpenCV 的安装和配置，然后再反过来本节的内容。至于 OpenCV 组件结构的研究方法，我们不妨管中窥豹，通过 OpenCV 安装路径下 include 目录里面头文件的分类存放，来一窥 OpenCV 这些年迅猛发展起来的庞杂组件架构。

进入到...\\opencv\\build\\include 目录，可以看到有 opencv 和 opencv2 这两个文件夹。显然，opencv 这个文件夹里面包含着旧版的头文件，而 opencv2 这个文件夹里面包含着具有时代意义的新版 OpenCV2 系列的头文件。

在 opencv 这个文件夹里面，也就是...\\opencv\\build\\include\\opencv 目录下，可以看到如下各种头文件。这里面大概就是 OpenCV 1.0 最核心的，而且保留下来的内容的头文件，如图 1.2 所示。我们可以把它们整体理解为一个大的组件。

cv.h	2013/12/20 星期五 ...	C/C++ Header	4 KB
cv.hpp	2013/12/20 星期五 ...	C/C++ Header	3 KB
cvaux.h	2013/12/20 星期五 ...	C/C++ Header	3 KB
cvaux.hpp	2013/12/20 星期五 ...	C/C++ Header	3 KB
cwimage.h	2013/12/20 星期五 ...	C/C++ Header	3 KB
cxcore.h	2013/12/20 星期五 ...	C/C++ Header	3 KB
cxcore.hpp	2013/12/20 星期五 ...	C/C++ Header	3 KB
cxeigen.hpp	2013/12/20 星期五 ...	C/C++ Header	3 KB
cxmisc.h	2013/12/20 星期五 ...	C/C++ Header	1 KB
highgui.h	2013/12/20 星期五 ...	C/C++ Header	3 KB
ml.h	2013/12/20 星期五 ...	C/C++ Header	3 KB

图 1.2 OpenCV 1.0 头文件

下面再来看看我们重点关注的 opencv2 文件夹，在...\\opencv\\build\\include\\opencv2 目录下，可以看到类似如图 1.3 所示的文件夹。

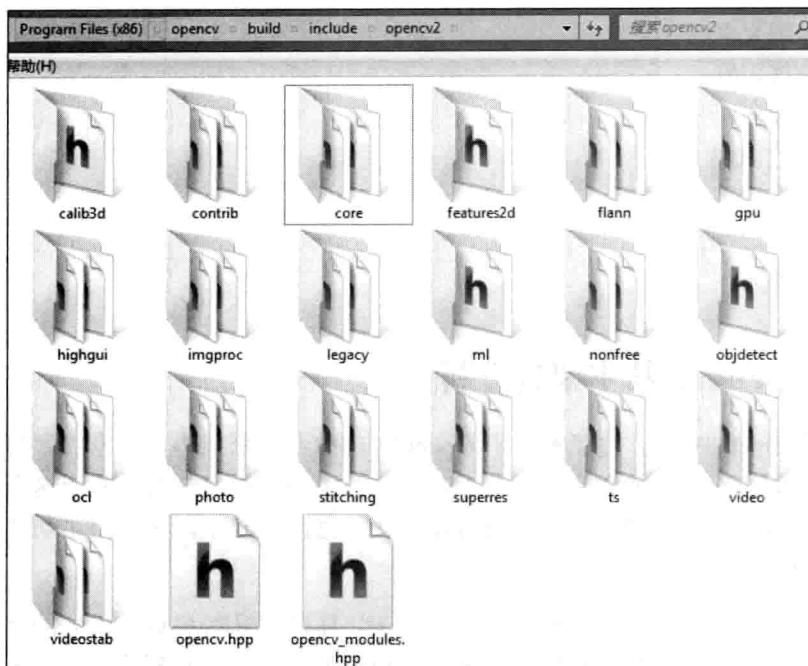


图 1.3 新版 OpenCV 的头文件（以 2.4.9 为例）

若仔细观察，会发现上述文件夹中有个名为 `opencv_modules.hpp` 的 `hpp` 文件，里面存放的是 OpenCV2 中与新模块构造相关的说明代码，打开可以发现其定义的是 OpenCV2 所有组件的宏，具体如下。

```
#define HAVE_OPENCV_CALIB3D
#define HAVE_OPENCV CONTRIB
#define HAVE_OPENCV_CORE
#define HAVE_OPENCV_FEATURES2D
#define HAVE_OPENCV_FLANN
#define HAVE_OPENCV_GPU
#define HAVE_OPENCV_HIGHLGI
#define HAVE_OPENCV_IMGPROC
#define HAVE_OPENCV_LEGACY
#define HAVE_OPENCV_ML
#define HAVE_OPENCV_NONFREE
#define HAVE_OPENCV_OBJDETECT
#define HAVE_OPENCV_OCL
#define HAVE_OPENCV_PHOTO
#define HAVE_OPENCV_STITCHING
#define HAVE_OPENCV_SUPERRES
#define HAVE_OPENCV_TS
#define HAVE_OPENCV_VIDEO
#define HAVE_OPENCV_VIDEOSTAB
```

下面就是 OpenCV 的所有模块，按照宏定义的顺序依次介绍。

(1) 【calib3d】——Calibration（校准）和 3D 这两个词的组合缩写。这个模块主要是相机校准和三维重建相关的内容，包括基本的多视角几何算法、单个立体摄像头标定、物体姿态估计、立体相似性算法、3D 信息的重建等。

(2) 【contrib】——Contributed/Experimental Stuf 的缩写。该模块包含了一些最近添加的不太稳定的可选功能，不用去多管。新增了新型人脸识别、立体匹配、人工视网膜模型等技术。

(3) 【core】——核心功能模块，包含如下内容：

- OpenCV 基本数据结构
- 动态数据结构
- 绘图函数
- 数组操作相关函数
- 辅助功能与系统函数和宏
- 与 OpenGL 的互操作

(4) 【imgproc】——Image 和 Process 这两个单词的缩写组合，图像处理模块。包含如下内容：

- 线性和非线性的图像滤波
- 图像的几何变换
- 其他（Miscellaneous）图像转换
- 直方图相关
- 结构分析和形状描述
- 运动分析和对象跟踪
- 特征检测
- 目标检测等内容

(5) 【features2d】——也就是 Features2D，即 2D 功能框架，包含如下内容：

- 特征检测和描述
- 特征检测器（Feature Detectors）通用接口
- 描述符提取器（Descriptor Extractors）通用接口
- 描述符匹配器（Descriptor Matchers）通用接口
- 通用描述符（Generic Descriptor）匹配器通用接口
- 关键点绘制函数和匹配功能绘制函数

(6) 【flann】——Fast Library for Approximate Nearest Neighbors，高维的近似近邻快速搜索算法库，包含以下两个部分：

- 快速近似最近邻搜索
- 聚类

(7) 【gpu】——运用 GPU 加速的计算机视觉模块。

(8) 【highgui】——高层 GUI 图形用户界面，包含媒体的输入输出、视频捕捉、图像和视频的编码解码、图形交互界面的接口等内容。

(9) 【legacy】——一些已经废弃的代码库，保留下作为向下兼容，包含如下内容：

- 运动分析
- 期望最大化
- 直方图
- 平面细分 (C API)
- 特征检测和描述 (Feature Detection and Description)
- 描述符提取器 (Descriptor Extractors) 的通用接口
- 通用描述符 (Generic Descriptor Matchers) 的常用接口
- 匹配器

(10) 【ml】——Machine Learning, 机器学习模块, 基本上是统计模型和分类算法, 包含如下内容:

- 统计模型 (Statistical Models)
- 一般贝叶斯分类器 (Normal Bayes Classifier)
- K-近邻 (K-Nearest Neighbors)
- 支持向量机 (Support Vector Machines)
- 决策树 (Decision Trees)
- 提升 (Boosting)
- 梯度提高树 (Gradient Boosted Trees)
- 随机树 (Random Trees)
- 超随机树 (Extremely randomized trees)
- 期望最大化 (Expectation Maximization)
- 神经网络 (Neural Networks)
- MLData

(11) 【nonfree】——一些具有专利的算法模块, 包含特征检测和 GPU 相关的内容。最好不要商用。

(12) 【objdetect】——目标检测模块, 包含 Cascade Classification (级联分类) 和 Latent SVM 这两个部分。

(13) 【ocl】——OpenCL-accelerated Computer Vision, 运用 OpenCL 加速的计算机视觉组件模块。

(14) 【photo】——Computational Photography, 包含图像修复和图像去噪两部分

(15) 【stitching】——images stitching, 图像拼接模块, 包含如下部分:

- 拼接流水线
- 特点寻找和匹配图像
- 估计旋转
- 自动校准
- 图片歪斜
- 接缝估测
- 曝光补偿

- 图片混合

(16) 【superres】——SuperResolution，超分辨率技术的相关功能模块。

(17) 【ts】——OpenCV 测试相关代码，不用去管。

(18) 【video】——视频分析组件，该模块包括运动估计、背景分离、对象跟踪等视频处理相关内容。

(19) 【Videostab】——Video stabilization，视频稳定相关的组件，官方文档中没有多做介绍，不用管它。

看到这里，相信大家已经对 OpenCV 的模块架构设计有了一定的认识。OpenCV 其实就是这么多模块作为代码容器组合起来的一个 SDK（Software Development Kit，软件开发工具包）而已，并不繁杂，也不稀奇。

## 1.3 OpenCV3 带来了什么

2009 年 10 月 01 日，OpenCV 2.0 发布，这标志着革命性的 OpenCV2 时代的来临。OpenCV 2 带来了全新的 C++ 接口，将 OpenCV 的能力无限放大。在 2.0 的时代，OpenCV 增加了新的平台支持，包括 iOS 和 Andriod，通过 CUDA 和 OpenCL 实现了 GPU 加速，为 Python 和 Java 用户提供了接口，基于 Github 和 Buildbot 构建了充满艺术感的持续集成系统，所以才有了被全世界的很多公司和学校所采用的稳定易用的 OpenCV 2.4.x。

2014 年 8 月 21 日，OpenCV 3.0 Alpha 发布，宣告着 OpenCV3 时代的登场。官方更新日志中提到，OpenCV 在 3.0 中改变了项目架构的方式，所以 3.0 时代不会有像 2.0 时代一样激进的尝试，只会有足够稳定的改进。且不说更多眼花缭乱的新特性，项目架构的改变是 OpenCV3 最为重大的革新之处。让我们在 1.3.1 节中一起进行了解。

### 1.3.1 项目架构的改变

通过 1.2 节《OpenCV 基本架构分析》的介绍，我们可以发现 OpenCV 是一个相对于整体的项目，各个模块都是以整体的形式构建然后组合在一起的。然而，随着功能的增加，OpenCV 主体集成了各种各样的功能模块，变得越来越臃肿。

而 3.0 的出现，就是为了给日益发福的 OpenCV 减肥，因为 OpenCV3 决定像其他大项目一样，抛弃整体架构，使用内核+插件的架构形式。

在 GitHub 中，除了存放着正式版 OpenCV 的主仓库和新增加的“opencv\_extra”仓库以外，OpenCV 3.0 中还添加了一个名为 opencv\_contrib 的全新仓库，这个新仓库中有很多让人兴奋的功能：包括脸部识别和文本探测，以及文本识别、新的边缘检测器、充满艺术感的图像修复、深度地图处理、新的光流和追踪算法等。



OpenCV 主仓库的地址: <https://github.com/itseez/opencv>

opencv\_extra 仓库地址: [https://github.com/itseez/opencv\\_extra](https://github.com/itseez/opencv_extra)

opencv\_contrib 仓库地址: [http://github.com/itseez/opencv\\_contrib](http://github.com/itseez/opencv_contrib)

正式版 opencv 与 opencv\_contrib 之间的区别如下:

- 两者都由 OpenCV 官方开发团队持续集成系统维护，虽然目前 opencv\_contrib 仓库中的代码测试并没有完成，很多功能不稳定。
- 主体的 opencv 在 GitHub 中由 Itseez 提供，其有着非常稳定的 API 以及少部分的创新。
- opencv\_contrib 仓库是大多数实验性代码放置的地方，一些 API 可能会有改变，一直会欢迎广大开发者们贡献新的精彩算法。
- opencv\_contrib 中的这些额外模块可以在 CMake 中用 OPENCV\_EXTRA\_MODULES\_PATH=/modules 传递给 CMake 文件，和 OpenCV3 主体中的代码一起编译和运行。
- opencv\_contrib 的文档是自动生成的，可以在 <http://docs.opencv.org/master/> 中找到，并会在随后的版本中更加完善。

还有一些 OpenCV3 的细节由于比较小众，在此不再赘述，大家可以在 OpenCV3 的官网中查看到更多的更新信息。

### 1.3.2 将 OpenCV2 代码升级到 OpenCV3 报错时的一些策略

由于 OpenCV3 的主体部分只是在 OpenCV2 的基础上进行小幅度的更改，所以我们会发现不少 OpenCV2 下开发的程序仍然可以在 OpenCV3 中正常编译运行。

然而，有一些版本升级导致的改变则会让我们在 OpenCV2 下开发的程序在 OpenCV3 中报错。本节，让我们将这些知识点聚在一起做一个列举，供大家在把基于 OpenCV2 的代码升级到 OpenCV3 时查阅。

#### 1.【问题一】由于宏名称的变更照成的“未声明的标识符”系列问题



有时候，遇到此类问题加入一句 “#include<cv.h>”便可以让 OpenCV3 也能认识一些 “CV\_” 前缀的宏，将问题解决掉。因为有不少 OpenCV1 的宏依然在 OpenCV3 的 cv.h 头文件中有定义。

**症状：**在 OpenCV3 的环境下运行 OpenCV2 中写的程序，报 “error C2065: ‘CV\_WINDOW\_AUTOSIZE’：未声明的标识符” 系列错误。

**分析：**OpenCV3 取消了 OpenCV1 中残留的 “CV\_” 式的宏前缀命名规范，对这些 CV\_ 前缀的宏使用了新的命名规范。这里有几种情况要分类讨论。

(1) 情况 1：直接去掉 “CV\_” 前缀

在这种情况下，比较典型的有如下一些函数。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

1) namedWindow 函数中, 例如将 CV\_WINDOW\_AUTOSIZE 改为 WINDOW\_AUTOSIZE。

2) threshold 函数中, 例如将 CV\_THRESH\_BINARY 改为 THRESH\_BINARY。

3) line 函数等一系列绘图函数中, CV\_FILLED 改为 FILLED。

4) remap 函数中, CV\_INTER\_LINEAR 改为 INTER\_LINEAR。

5) 在鼠标操作函数 SetMouseCallback 中, 将 CV\_EVENT\_LBUTTONDOWN 改为 EVENT\_LBUTTONDOWN、CV\_EVENT\_LBUTTONDOWN 改为 EVENT\_LBUTTONDOWN、CV\_EVENT\_FLAG\_LBUTTON 改为 EVENT\_FLAG\_LBUTTON、CV\_EVENT\_MOUSEMOVE 改为 EVENT\_MOUSEMOVE。

7) HoughCircles 函数中, 将 CV\_HOUGH\_GRADIENT 改成 HOUGH\_GRADIENT。

8) inpaint 函数中, 将 CV\_INPAINT\_TELEA 改为 INPAINT\_TELEA、CV\_INPAINT\_NS 写作 INPAINT\_NS。

9) matchTemplate 函数中, 将 CV\_TM\_SQDIFF 改为 TM\_SQDIFF、CV\_TM\_SQDIFF\_NORMED 改为 TM\_SQDIFF\_NORMED。

10) 在 imwrite 函数相关使用中, 将 CV\_IMWRITE\_PNG\_COMPRESSION 改成 IMWRITE\_PNG\_COMPRESSION。

11) 设置摄像头尺寸的时候, 将 CV\_CAP\_PROP\_FRAME\_WIDTH 改为 CAP\_PROP\_FRAME\_WIDTH、CV\_CAP\_PROP\_FRAME\_HEIGHT 改为 CAP\_PROP\_FRAME\_HEIGHT。

(2) 情况 2: 需要用新的前缀替换

在这种情况下, 比较典型的有如下一些函数。

1) line 函数等一系列绘图函数中, CV\_AA 改为 LINE\_AA

2) cvtColor 函数中颜色空间转换系的宏, 全替换为 “COLOR\_” 前缀, 如 CV\_BGR2HSV 改为 COLOR\_BGR2HSV

3) normalize 函数中, 将 CV\_MINMAX 改为 NORM\_MINMAX。

4) morphologyEx 函数中的宏, 全部替换为 “MORPH\_” 前缀, 如 CV\_MOP\_OPEN 改为 MORPH\_OPEN。

5) threshold 函数中的宏, 全部替换为 “THRESH\_” 前缀, 如将 CV\_THRESH\_BINARY 改 THRESH\_BINARY

(3) 情况 3: 需要在新的命名空间中使用宏

在这种情况下, 比较典型的有如下一些函数。

1) TermCriteria 函数中, CV\_TERMCRIT\_EPS 改为 TermCriteria::EPS、CV\_TERMCRIT\_ITER 改为 TermCriteria::MAX\_ITER

2) CascadeClassifier::detectMultiScale 函数中, CV\_HAAR\_SCALE\_IMAGE 改为 CASCADE\_SCALE\_IMAGE。

**2.【问题二】使用 vector 容器之时提示“error C2065: “vector”: 未声明的标识符”系列错误。**

症状: 在 OpenCV3 的环境下运行 OpenCV2 中写的程序, 使用了 vector 容器, 而未包含 STD 命名空间, 于是便会报“error C2065: “vector”: 未声明的标识符”系列错误。

分析: OpenCV3 中并没有在头文件中使用标准程序库 std 的命名空间。所以遇到这个错误, 在我们写的程序开头加上一句“using namespace std;”使用 C++ 的标准命名空间即可解决问题。

上述两个就是 OpenCV2 代码升级 OpenCV3 时最常会遇到的问题。接下来, 列举一些 OpenCV3 升级时一些其他的改动细节, 以结束此节。

### 3.其他一些细节问题的解决方案

1) features2d.hpp 头文件路径的更改。将#include <opencv2/nonfree/features2d.hpp>

改为#include <opencv2/features2d.hpp>

2) core.hpp 头文件路径更改。将#include <opencv2/core/core.hpp>改为#include <opencv2/core.hpp>

3) 用 format 进行格式化输出时, 将 format(r,"python") 改成 format(r, Formatter::FMT\_PYTHON)。

4) 定义尺寸时, 将 cvSize(-1,-1) 改为 Size(-1,-1)。

5) 在表示颜色时, 将 CV\_RGB 改为 Scalar。



在 OpenCV 的官方开发网站的更新日志页面, 可以看到历代 OpenCV 官方详细的改动日志: <http://code.opencv.org/projects/opencv/wiki/ChangeLog>

## 1.4 OpenCV 的下载、安装与配置

在本节开头需要说明的是, OpenCV 开发环境的搭建方法从最初的 1.0 版到当前的最新版本, 基本上没有大的改变。在笔者初次编写本章内容的时候, OpenCV 的最新版本是 2014 年 4 月 25 日发行的 2.4.9 版, 并且此版本的配置过程比较典型, 因此本书的开发环境搭建就将这个版本作为主要示例。这套方法可以看作 OpenCV 的万金油配置方法, 无论以后 OpenCV 更新到什么版本的, 或者是需要配置以前的任何一版, 原则上都可以使用本节介绍的配置方法来进行开发环境的搭建。

### 1.4.1 预准备: 下载和安装集成开发环境

既然是进行 OpenCV 编程开发的学习, 一个顺手的集成开发环境自然少不了。

本书的内容以及示例程序都是默认以微软的 Visual Studio 作为开发环境的。关于 Visual Studio，目前有众多的版本，从最经典的 Visual Studio 6.0，历经 Visual Studio 2002、Visual Studio 2005、Visual Studio 2010、Visual Studio 2012，直到当前的最新版本 Visual Studio 2015。目前业界使用最多，用起来最顺手也支持率最高的，当属 Visual Studio 2010。本书所有的示例程序都是在 Visual Studio 2010 旗舰版的环境下开发编写的，虽然使用其他版本 Visual Studio 也同样可行，但大家在学习此书时，最好也同样使用 Visual Studio 2010，因为各个版本之间还是有些细微差别的。需要注意，不少旧版本 Visual Studio 可能和新版本的 OpenCV 不兼容，若实在不想使用 Visual Studio 2010，推荐大家使用 Visual Studio 2010 以上的版本，如 Visual Studio 2012、Visual Studio 2013、Visual Studio 2015 等，因为选用低版本的 Visual Studio 可能在开发过程中出现一些难以解决的问题。

#### 1.4.2 第一步：下载和安装 OpenCV SDK

本书配置 OpenCV 时，将以最为典型的 OpenCV2.4.9 作为参考，只要弄懂该配置过程，以后可以轻松完成当前发行的任意一个 OpenCV 版本的配置任务，而区别仅仅是下文中的第五步——链接库的配置，把对应的 249 改成自己对应的 OpenCV 版本号即可。

 OpenCV3 的配置方法。OpenCV3 官方对配置过程已经简化了，将 OpenCV2 中的近 20 个 lib 文件浓缩成了 opencv\_ts300.lib、opencv\_world300.lib (Release 版本) 这两个。所以区别也是在第五步链接库的配置处，将 OpenCV2 中的那一大串 lib 对应地改为 opencv\_ts300.lib、opencv\_world300.lib (Release 版本) 或 opencv\_ts300d.lib、opencv\_world300d.lib (Debug 版本) 即可

在官网 <http://opencv.org/> 上找到 OpenCV Windows 版下载下来，如图 1.4 所示。

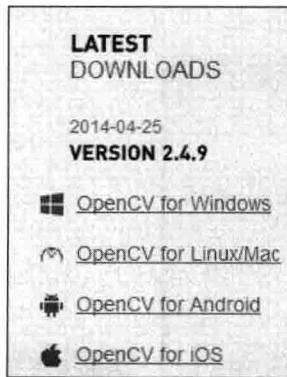


图 1.4 OpenCV 安装包下载区域

下载完后得到文件“opencv 2.x.x.exe”，如图 1.5 所示。下载完成后，便可以开始进行 OpenCV 的安装和配置。与其说是安装，不如叫解压更加合适，因为我们下载的.exe 安装文件就是一个自解压程序而已。双击这个文件后程序会提示我们解压到某个地方，推荐放到…\Program Files\下，比如 D:\Program Files (因为

OpenCV 项目文件打包的时候，根目录就是 opencv，所以我们不需要额外新建一个名为 opencv 的文件夹），然后在弹出的对话框中点击【Extract】按钮。如图 1.6 所示。

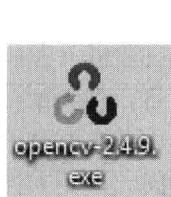


图 1.5 OpenCV 2.4.9 安装程序

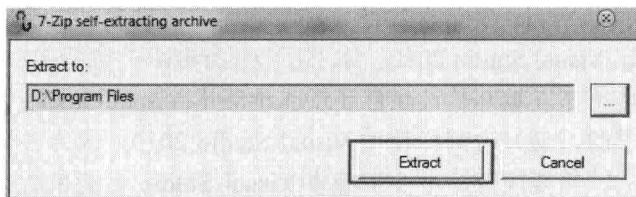


图 1.6 提取 OpenCV 文件

稍后，OpenCV 约几个 GB 的文件就解压到了我们上文指定的目录下，本步骤到此结束。在此进行简单的文件内容的说明：解压完成后，会在指定的路径下生成一个名为 opencv 的文件夹，它包含了两个子文件夹，分别名为 build 和 sources。其中，build 文件夹中是支持 OpenCV 使用的相关文件，而 sources 中为 OpenCV 的源代码及相关文件。如果仅仅是希望将 OpenCV 写的程序在电脑中跑起来并想要尽量节省硬盘空间，就只需要 build 里面的文件内容，sources 文件夹完全可以删掉。但是需要注意的是，OpenCV 官方示例集以及说明文档，也就是 samples 文件夹里面的示例程序和 doc 文件夹中的说明文档，都是存放在 sources 文件夹里面的。所以当然是推荐保留 sources 文件夹，若真要删除，还请三思而后行。

### 1.4.3 第二步：配置环境变量

配置方法如下。

【计算机】→【(右键) 属性】→【高级系统设置】→【高级(标签)】→【环境变量】→(双击) 系统变量中的 PATH→在变量值里面添加相应的路径。注意：是“添加”相应的值，和之前已有的值用分号“;”来分隔，而不是删掉之前已有的变量值。如图 1.7 和图 1.8 所示。

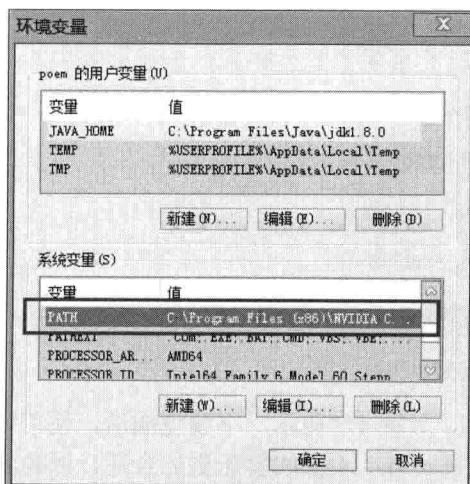


图 1.7 选取 PATH 环境变量

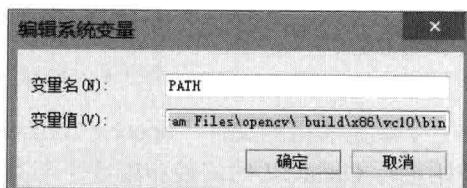


图 1.8 修改 PATH 环境变量的值

## SimpPDF Merge and Split Unregistered Version - http://www.simpopdf.com

对于 32 位系统，就添加“... opencv\build\x86\vc10\bin”，和之前的就有的环境变量用英文的分号“;”进行分隔。而对于 64 位系统，可以两个都添加上——“... opencv\build\x86\vc10\bin” 和 “... opencv\build\x64\vc10\bin”。这样，到时候才可以在编译器 Win32 和 X64 中来回切换。

例如，笔者的就是 D:\Program Files\opencv\build\x64\vc10\bin;D:\Program Files\opencv\build\x86\vc10\bin 这两个路径。

也有朋友在笔者博客留言道“亲测 64 位系统也只需添加 ..... opencv\build\x86\vc10\bin 即可”，这是只针对 32 位编译器版本的做法，也是可行的。



变量值实际为 bin 文件夹的路径。D 表示 OpenCV 安装于 D 盘；X64 表示运行系统环境位 64 位系统，若安装于 32 位系统，应为 X86；vc10 表示编译环境为 Microsoft Visual Studio 2010。变量添加完成后最好注销系统，然后才会生效。

### 1.4.4 第三步：工程包含（include）目录的配置

笔者在写作此节内容之前，发现互联网上相当多的 OpenCV 配置博文都写道“每次新建工程都要重新配置”，其实是不用这样麻烦的。下面且看如何经过一次的配置就可以“一劳永逸”。

首先是在 Visual Studio 里面新建一个控制台应用程序，最好是勾选空项目。考虑到不少看此节的读者们很少接触 Visual Studio，那么在这里将过程详细截取出来，方便大家快速掌握配置方法。

1. 打开 Visual Studio，新建一个项目。可以单击【起始页】中的【新建项目】，如图 1.9 所示。或者依次单击菜单栏中的【文件】→【新建】→【项目】，如图 1.10 所示。

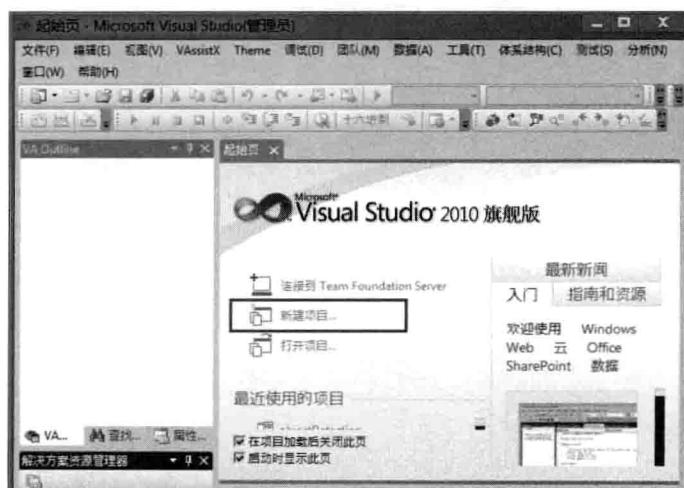


图 1.9 打开新建项目对话框的方法一

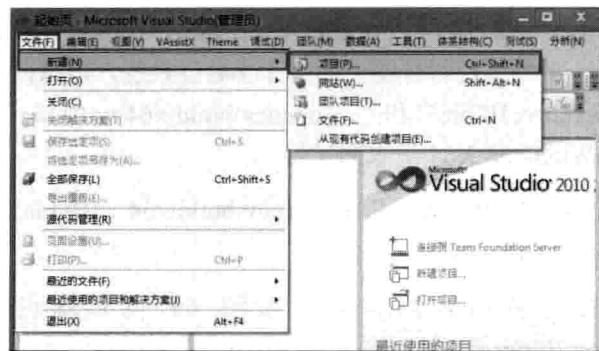


图 1.10 打开新建项目对话框的方法二

然后选择新建【Win32 控制台应用程序】，进行命名，比如 test1，然后选好路径，单击【确定】，如图 1.11 所示。



图 1.11 新建 Win32 控制台应用程序

2. 进入【Win32 应用程序向导】页面后，单击“下一步”，或者“应用程序设置”，会跳转到同样的应用程序设置页面。如图 1.12 所示。

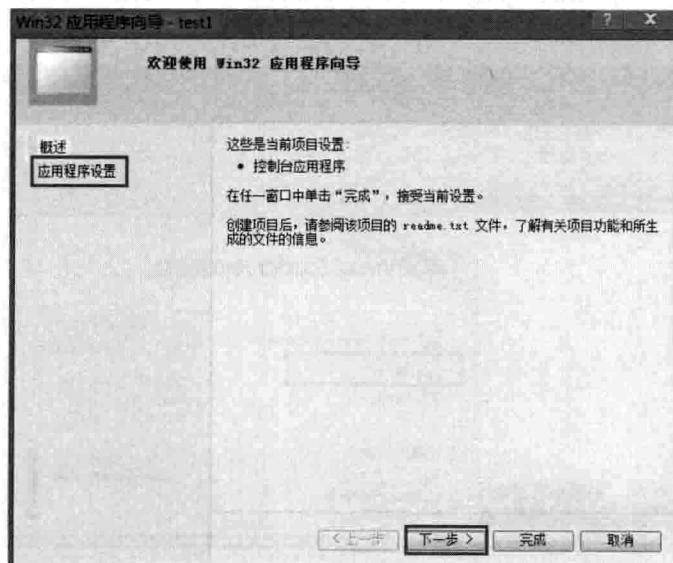


图 1.12 Win32 应用程序向导页面

3. 在应用程序设置页面，勾选【空项目】。如图 1.13 所示。

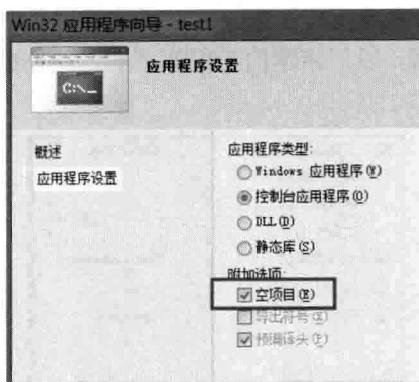


图 1.13 勾选空项目

4. 在解决方案资源管理器的【源文件】处右键单击→添加→新建项，准备在工程中新建一个 cpp 源文件。如图 1.14 所示。

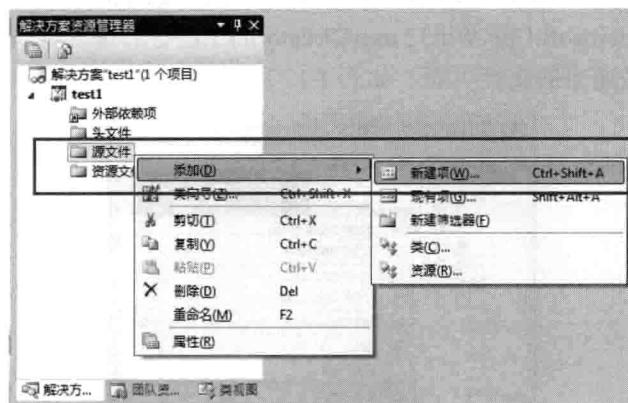


图 1.14 在源文件中添加新建项

5. 选定【C++文件 (.cpp)】，进行命名，比如“main”，然后单击【添加】，一个新的.cpp 文件就添加到了工程中。如图 1.15 所示。



图 1.15 在源文件中添加新建项

6. 在菜单栏里单击【视图】→【属性管理器】(如图 1.16 所示), 就会在 Visual Studio 中多出一个属性管理器工作区来。而在属性管理器中进行一次配置, 就相当于进行了通用的配置过程, 以后新建的工程就不用再额外进行配置。这就是我们这种配置方法的核心思想。

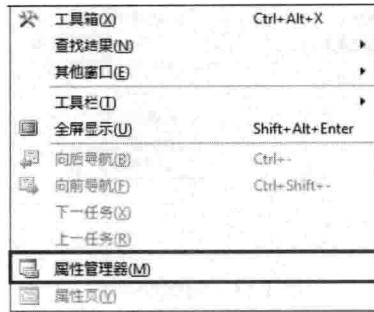


图 1.16 属性管理器

7. 在新出现的“属性管理器”工作区中, 展开【Debug|Win32】文件夹, 对文件夹中的【Microsoft.Cpp.Win32.userDirectories】进行右键属性操作, 或者双击, 即可打开工程最通用的属性页面。如图 1.17 所示。



图 1.17 打开工程最通用的属性页面

8. 打开属性页面后, 首先在【通用属性】→【VC++目录】→【包含目录】中添加以下三个目录。如图 1.18、1.19 所示。

- D:\Program Files\opencv\build\include
- D:\Program Files\opencv\build\include\opencv
- D:\Program Files\opencv\build\include\opencv2



图 1.18 添加包含目录 (1)



图 1.19 添加包含目录 (2)

当然，这是之前把 OpenCV 解压到 D:\Program Files\下的情况。实际的路径还要看读者自己把 OpenCV 解压到了哪个目录下，根据具体情况来调节。

#### 1.4.5 第四步：工程库（lib）目录的配置

第四步和第三步差不多，在“属性管理器”工作区中，单击【项目】→Debug|Win32→Microsoft.Cpp.Win32.userDirectories（右键属性，或者双击）打开属性页面。然后在【通用属性】→【VC++目录】→【库目录】中，添加D:\Program Files\opencv\build\x86\vc10\lib这个路径。如图1.20所示。

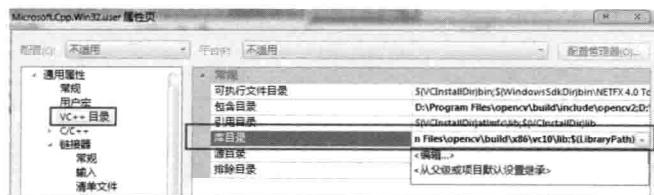


图 1.20 添加库目录

最好不要手动用键盘输入这里给出的路径，而是去预览里面指定出来，这样会准确得多。因为路径不小心输入出错，可能在编译时出现“LINK : fatal error LNK1104: 无法打开文件“opencv\_calib3d249.lib”之类的错误提示。

这里选择 x86 还是 x64 是一个常常令人困惑的问题。当然，对于 32 位操作系统，铁定就是选 x86 了。如果是 64 位操作系统，不要直接想当然地选择 x64，正确的理解是这样的：

不管是 32 位还是 64 位操作系统，只需要考虑用 Win32 编译器还是 x64 编译器。其实配置选择什么跟 64 位还是 32 位系统没有直接的关系，而是在于在编译程序时是使用哪个编译器。编译器选的是 Win32，就用 x86；编译器选的是 x64，就用 x64。不过一般情况下，都是用的 Win32 的 x86 编译器。所以，无论 32 还是 64 位操作系统，配置文件最好都选择 x86 版的。

## SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

另外，这里的 vc10 表示 vs2010，如果是其他版本的 visual studio，稍微要微调一下。其中：vc8 等同于 Visual Studio 2005，vc9 等同于 Visual Studio 2008，vc10 等同于 Visual Studio 2010，vc11 等同于 Visual Studio 2012，vc12 等同于 Visual Studio 2013，后续版本以此类推。

### 1.4.6 第五步：链接库的配置

同样在“属性管理器”工作区中单击【项目】→【Debug|Win32】→【Microsoft.Cpp.Win32.userDirectories】(右键单击然后单击属性，或者直接双击)，即可打开属性页面。然后依次单击【通用属性】→【链接器】→【输入】→【附加的依赖项】，如图 1.21 所示。

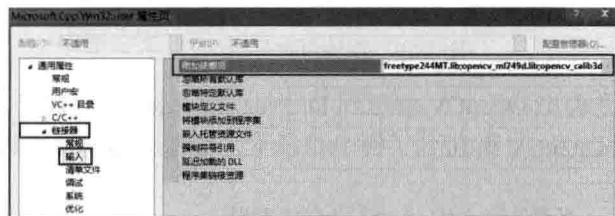


图 1.21 添加附加依赖项

这里给出两个版本的例子，对于其他版本读者可以自行变通，将 248 或 249 改成对应的 OpenCV 版本即可。

对于【OpenCV 2.4.8】，添加以下 248 版本的 lib(lib 顺序是：19 个带 d 的 debug 版 lib 写在前面，19 个不带 d 的 release 版 lib 写在后面，即优先支持 debug 模式的编译运行)。

```
opencv_m1248d.lib
opencv_calib3d248d.lib
opencv_contrib248d.lib
opencv_core248d.lib
opencv_features2d248d.lib
opencv_flann248d.lib
opencv_gpu248d.lib
opencv_highgui248d.lib
opencv_imgproc248d.lib
opencv_legacy248d.lib
opencv_objdetect248d.lib
opencv_ts248d.lib
opencv_video248d.lib
opencv_nonfree248d.lib
opencv_ocl248d.lib
opencv_photo248d.lib
opencv_stitching248d.lib
opencv_superres248d.lib
opencv_videostab248d.lib

opencv_objdetect248.lib
opencv_ts248.lib
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
opencv_video248.lib  
opencv_nonfree248.lib  
opencv_ocl248.lib  
opencv_photo248.lib  
opencv_stitching248.lib  
opencv_superres248.lib  
opencv_videostab248.lib  
opencv_calib3d248.lib  
opencv_contrib248.lib  
opencv_core248.lib  
opencv_features2d248.lib  
opencv_flann248.lib  
opencv_gpu248.lib  
opencv_highgui248.lib  
opencv_imgproc248.lib  
opencv_legacy248.lib  
opencv_ml248.lib
```

对于【OpenCV 2.4.9】，添加如下 249 版本的 lib（这样的 lib 顺序是：19 个带 d 的 debug 版的 lib 写在前面，19 个不带 d 的 release 版的 lib 写在后面），如图 1.22 所示。

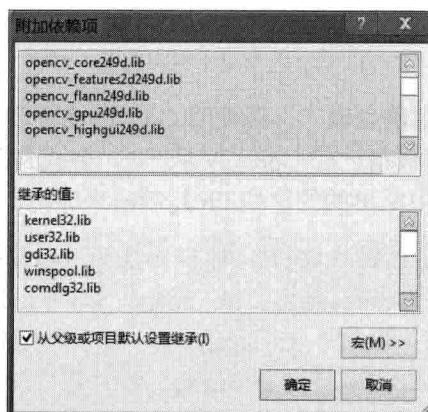


图 1.22 添加附加依赖项

```
opencv_ml249d.lib  
opencv_calib3d249d.lib  
opencv_contrib249d.lib  
opencv_core249d.lib  
opencv_features2d249d.lib  
opencv_flann249d.lib  
opencv_gpu249d.lib  
opencv_highgui249d.lib  
opencv_imgproc249d.lib  
opencv_legacy249d.lib  
opencv_objdetect249d.lib  
opencv_ts249d.lib  
opencv_video249d.lib  
opencv_nonfree249d.lib  
opencv_ocl249d.lib
```

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
opencv_photo249d.lib  
opencv_stitching249d.lib  
opencv_superres249d.lib  
opencv_videostab249d.lib  
  
opencv_objdetect249.lib  
opencv_ts249.lib  
opencv_video249.lib  
opencv_nonfree249.lib  
opencv_ocl249.lib  
opencv_photo249.lib  
opencv_stitching249.lib  
opencv_superres249.lib  
opencv_videostab249.lib  
opencv_calib3d249.lib  
opencv_contrib249.lib  
opencv_core249.lib  
opencv_features2d249.lib  
opencv_flann249.lib  
opencv_gpu249.lib  
opencv_highgui249.lib  
opencv_imgproc249.lib  
opencv_legacy249.lib  
opencv_ml249.lib
```

上述内容无须手动用键盘输入，请使用 Google 或百度搜索“OpenCV2.4.9 配置”关键字（将 2.4.9 改成自己的 OpenCV 版本），会找到海量的 OpenCV 配置博文，然后将博主在文章中贴出的这些 lib 复制出来即可。



对于 OpenCV3，将这些 lib 对应地改为 opencv\_ts300d.lib、opencv\_world300d.lib（Debug 版本）或 opencv\_ts300.lib、opencv\_world300.lib（Release 版本）即可。且 OpenCV3 似乎终于修复了下文中介绍的 OpenCV2 时代让众多使用者苦恼的，由于带 d 和不带 d 的 lib 的区别而导致报错的问题。测试使用将带 d 的 Debug 版本的 lib 作为【属性】→【链接器】→【输入】→【附加的依赖项】中的链接库，程序在 OpenCV3 中可以同时支持 Debug 和 Release 版本代码的正确运行，不会再像 OpenCV2 一样报字符串异常照成的内存错误。

需要注意的是，所粘贴内容即为之前我们解压的 OpenCV 目录 D:\opencv\build\x86\vc10\lib 下所有 lib 库文件的名字。其中的 249 代表 OpenCV 版本为 2.4.9，若是其他版本则在这里要进行相应的更改。比如说 2.4.6 版的 OpenCV，那么这里的 opencv\_calib3d249d.lib 就要改成 opencv\_calib3d246d.lib。

Debug 文件库名有 d 结尾，Release 没有，如 opencv\_ts248d.lib（debug 版本的 lib）和 opencv\_ts248.lib（release 版本的 lib）。

按照以上方式进行配置，也许会出现 debug 下可以运行但是 release 下不能运行的情况（因为字符串读取问题引起的诸如图片载入不了、报指针越界、内存错

## SimpPDF Merge and Split Unregistered Version - http://www.simpopdf.com

误等等), 这算是 OpenCV 自 2.4.1 以来的一个 bug。这里会优先选择在依赖项中, 写在前面的那一类 lib 作为默认可以支持的调试方式。若带 d 的 debug 系的十几个 lib 写在前面, 那么就默认支持 debug 模式下可以载入图片、识别字符串等操作。反之, 若不带 d 的 release 系的 lib 写在前面, 那么就默认支持 release 模式下可以载入图片、识别字符串等操作。

解决方案: 上述方法在 debug 下可以正常载入图片 (因为写在前面的是众多带 d 的 lib), 若想在 release 模式下也支持图片载入等操作, 可以在工程的 release 模式下, 将不带 d 的 lib 添加到【项目】→【属性】→【配置属性】→【链接器】→【输入】→【附加的依赖项】下即可。注意这样打开的是当前工程的属性页, 在这里将 release 版 (即不带 d) 的相关 lib 添加进去, 就表示让当前的这个工程的 release 模式, 有了这些 lib 作为依赖。类似错误如图 1.23 所示。

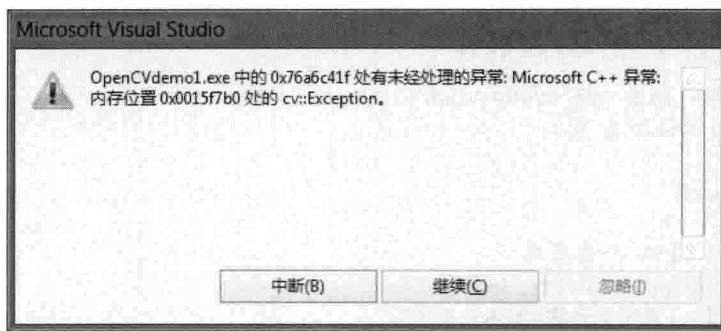


图 1.23 “未经处理的异常”系列错误报错图示

### 1.4.7 第六步: 在 Windows 文件夹下加入 OpenCV 动态链接库

需要注意的是: 如果环境变量配置得准确, 且配置之后经过重启, 就没有进行这步配置的必要了。即做完上面第五步的配置, 重启一次, 就可以直接跳到第七步进行测试, 看是否配置成功。当然, 若想不重启而马上查看是否配置成功, 就需要进行这步的配置。

在运行基于 OpenCV 的程序的时候, 往往会得到与图 1.24 所示界面类似的结果。

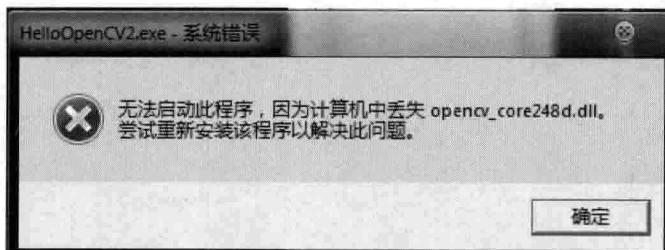


图 1.24 “计算机中丢失 XX.dll”系列错误报错图示

这种问题最简单的方法, 是把相关的 dll 都复制到 Windows 操作系统的目录下。如果 Windows 安装在 C 盘, 那么 32 位系统的放置路径就是 C:\Windows\System32, 而 64 位系统, 放置路径为 C:\Windows\SysWOW64。

按照之前我们的 OpenCV 的存放环境, 这些 dll 存放在 D:\Program Files\opencv\

build\x86\vc10\bin 目录下。找到这个目录，用【Alt+A】全选，【Alt+C】复制，然后转到 C:\Windows\SysWOW64 下，用【Alt+V】粘贴，这步就完成了。

OpenCV 的配置过程就是如上 6 步，接下来我们进行最终的实例测试来检测是否配置无误。

#### 1.4.8 第七步：最终测试



此次配置的工程文件已在本书配套代码包中给出，是第一个示例程序，名为“HelloOpenCV”。大家若不想手动打出上述代码，或者配置过程中遇到问题想查看配置细节，都可以进行参考。

测试过程为用 OpenCV 载入并显示一张图片到窗口中。我们用 Visual Studio 新建一个空项目控制台应用程序，随意命名，比如叫做“HelloOpenCV”，然后新建一个 cpp 文件，输入如下代码。

```
#include <opencv2/opencv.hpp>
using namespace cv;

int main()
{
    // 【1】读入一张图片
    Mat img=imread("1.jpg");
    // 【2】在窗口中显示载入的图片
    imshow("【载入的图片】",img);
    // 【3】等待 6000 ms 后窗口自动关闭
    waitKey(6000);
}
```

当然，上述代码中带双斜杠 “//” 的文字是注释，输入不影响测试结果。我们放置一张名为 1.jpg 的图片到工程目录中（和 cpp 源文件同一路经），然后单击 Visual Studio 中的【运行】按钮。如果配置成功，就不会报错，从而得到一个控制台窗口和一张图片窗口，分别如图 1.25 和图 1.26 所示。

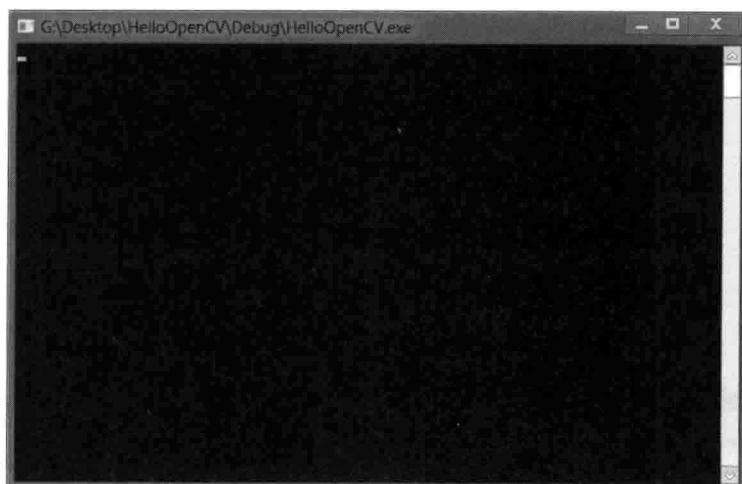


图 1.25 程序运行后得到的控制台窗口



图 1.26 程序运行后得到的图片窗口

### 1.4.9 可能遇到的问题和解决方案

生活不可能是一帆风顺的，我们的配置过程也是如此。笔者在几次的配置过程中，遇到了如下几种典型问题，相信各位读者也可能会遇到，本小节就在这里集中列举这些问题，希望能对大家有所帮助。

#### 1. 【问题 1】找不到 core.h

出现这个问题也许是因为 include 的时候粗心大意了。比如你的版本是 2.4.6，在这个版本下，opencv 根文件夹下面会有个 include，但配置的时候如果包含的是它就错了，正确的应该填 build 文件夹中的那个 include。

#### 2. 【问题 2】无法解析的外部命令

这个问题其实上文已经有过解释：不管是 32 位还是 64 位操作系统，只需要考虑用 win32 编译器还是 X64 编译器。

其实配置选择什么跟 64 位还是 32 位系统没有直接的关系，而是在于你在编译程序时使用的是哪个编译器：编译器是 win32，就用 x86；编译器是 X64，就用 X64。不过一般情况下，都是用的 win32 的 X86 编译器。所以，无论 32 还是 64 位操作系统，配置文件最好都选择 x86 版的。

#### 3. 【问题 3】形如--error LNK2005:xxx 已经在 msrvtd.lib(MSVCR90D.dll)中定义

出现这个问题，把静态库不包含就行了。

#### 4. 【问题 4】应用程序无法正常启动 0xc000007b

这是 Lib 包含的问题。可能你同时包含了 X86 和 X64 的，可能包含出错了。而且对于 windows7/ 8 64 位，dll 要放在和 System32 文件夹同级的 SysWOW64 文

件夹中。

### 5.【问题5】明明图片路径是对的，却载入不进图片，提示指针越界，有未经处理的异常

类似错误如图1.22，这表示在内存中图片没有读取成功，导致指针越界异常。这时要再次检查以确保图片的后缀名和路径与代码中的一致。一般情况下出现此问题算是OpenCV的一个bug，是工程属性里面关于带d和不带d的lib文件的附加依赖项的问题。

就算配置成功，若想在debug和release模式下同时可以运行，还需手动在工程属性里面加上一些lib。当得到这样的错误时，可以把调试方式改一改，将debug和release互换，如图1.27所示。

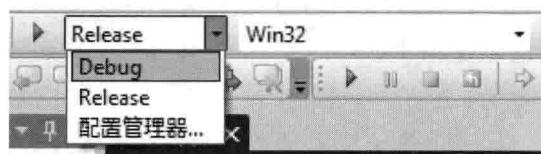


图1.27 更改调试方式

或者打开当前工程的属性页（注意不是通用属性页），debug或者release哪个报错，就把对应的带d或不带d的lib添加到【（当前）工程属性】→【链接器】→【输入】→【附加的依赖项】下即可。

### 6.【问题6】无法打开文件“opencv\_ml249d.lib”系列的错误

fatal error LNK1104：无法打开文件“opencv\_ml249d.lib”是一个常见的错误。这个错误主要是因为包含的库目录中，和包含的附加依赖项不能相互对应照成的。也许是“opencv\_ml249d.lib”多加了一个空格，成了“opencv\_ml249d.lib”，就会报错。遇到这个问题，检查以下三个方面。

- (1) 检查第四步“4.工程库（lib）目录的配置”库目录中的路径是否准确。
- (2) 检查第五步“5.链接库的配置”中“附加依赖项”的格式有没有问题，有没有多空格，版本号248、249是否正确，有没有多一个空格或少一个点。
- (3) 第二步环境变量的配置是否准确。

另外的解决方案是：依次进入【项目】→【属性管理器】→【Debug|Win32->Microsoft.Cpp.Win32.userDirectories】中的【属性页面】→【链接器】→【常规】，在里面的【附加库目录】中加入相应的lib文件目录。

## 1.5 快速上手OpenCV图像处理

上一节中，我们已经配置完成OpenCV，而本节将带领大家接触几个OpenCV图像处理相关的程序，看看OpenCV用简洁的代码能实现哪些有趣的图像效果，让大家快速地爱上OpenCV，提高学习兴趣。

### 1.5.1 第一个程序：图像显示

本小节介绍的程序即为上一节 OpenCV 配置中的测试用例，为了演示的连贯性而再次在此处展开讲解。

在新版本 OpenCV 中，图像显示过程非常简单，只需用 `imread` 函数载入到新版本的图像存储数据结构 `Mat` 类中，然后用 `imshow` 函数显示即可。这两个函数在第 3 章会有详细讲解。打开 Visual Studio，新建一个控制台项目，然后新建一个 `cpp` 文件，在其中添加如下代码：

```
#include <opencv2/opencv.hpp>
using namespace cv; //包含 cv 命名空间

void main()
{
    Mat srcImage = imread("1.jpg"); //载入图像
    imshow("【原始图】", srcImage); //显示图像
    waitKey(0); //等待任意按键按下
}
```

其中，`#include <opencv2/opencv.hpp>` 一句为 OpenCV 头文件的包含；而 `using namespace cv;` 一句为命名空间的包含，这在第 3 章的第 1 节会有详细的讲解。之后，先载入图像，然后显示图像，并且调用 `waitKey` 函数等待按键按下，以便让图片窗口一直显示，直到有按键按下。

放置一张以网络上下载的图片，并改名为程序代码中指定的“`1.jpg`”，然后运行程序，可以得到如图 1.28 所示效果。

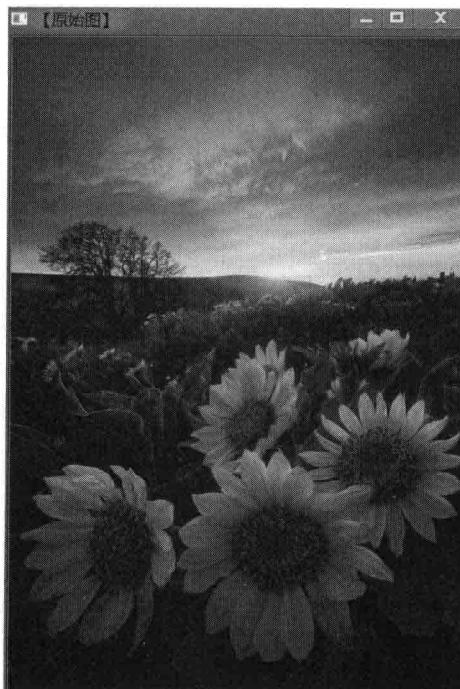


图 1.28 运行效果图

### 1.5.2 第二个程序：图像腐蚀

我们再来看如何用 OpenCV 实现最基本的形态学运算之一——腐蚀，即用图像中的暗色部分“腐蚀”掉图像中的高亮部分。图像形态学操作在后文有详细讲解，这里我们先一起了解一下：同样是新建控制台项目，新建 cpp 源文件，放置名为“1.jpg”的图片到工程目录下（和 cpp 源文件同一目录下）。



之后的示例程序，都默认采取类似的操作——在 Visual Studio 中新建项目，新建源文件，放置名为“1.jpg”的图片到 cpp 源文件同一目录下。特在此说明。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
-----
#include <opencv2/highgui/highgui.hpp> //OpenCV  highgui 模块头文件
#include <opencv2/imgproc/imgproc.hpp> //OpenCV 图像处理头文件
using namespace cv; //包含 cv 命名空间

int main() // 控制台应用程序的入口函数，我们的程序从这里开始
{
    //载入原图
    Mat srcImage = imread("1.jpg");
    //显示原图
    imshow("【原图】腐蚀操作", srcImage);
    //进行腐蚀操作
    Mat element = getStructuringElement(MORPH_RECT, Size(15, 15));
    Mat dstImage;
    erode(srcImage, dstImage, element);
    //显示效果图
    imshow("【效果图】腐蚀操作", dstImage);
    waitKey(0);

    return 0;
}
```

程序首先依然是载入和显示一幅图像，然后定义一个 Mat 类型的变量来获得 getStructuringElement 函数的返回值，而 getStructuringElement 函数的返回值为指定形状和尺寸的结构元素（内核矩阵）。参数准备完毕，接着便可以调用 erode 函数进行图像腐蚀操作，最后调用 imshow 函数进行显示，用 waitKey 函数等待按键按下，以便能让窗口一直显示。

原图和程序运行效果图如图 1.29 和图 1.30 所示。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

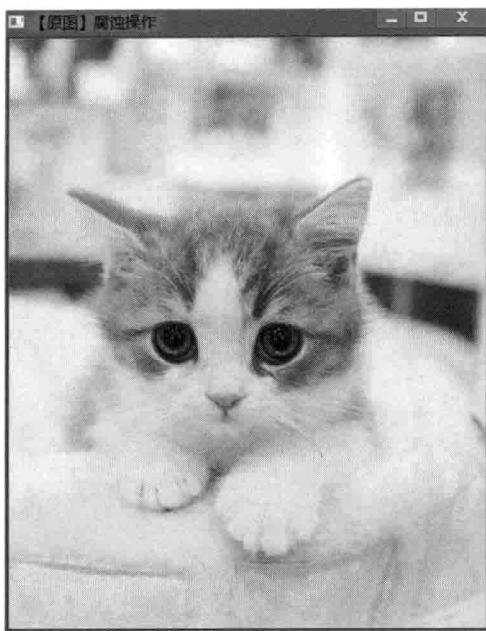


图 1.29 原始图

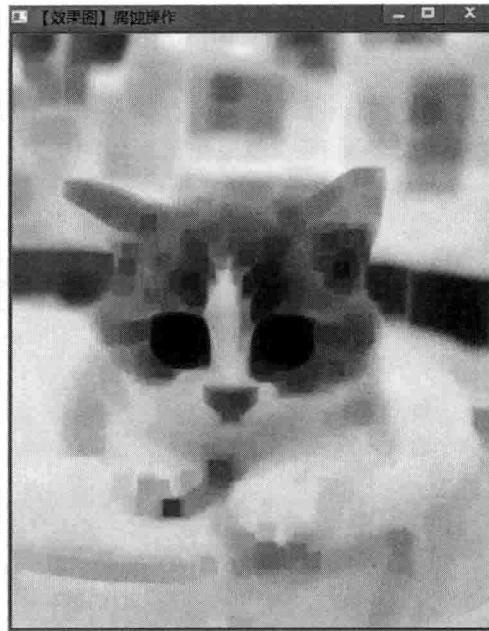


图 1.30 图像腐蚀效果图

### 1.5.3 第三个程序：图像模糊

接着让我们看看用 OpenCV 对图像进行均值滤波操作，模糊一幅图像的代码如何书写。主要使用进行均值滤波操作的 blur 函数，代码如下：

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
//-----  

#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;  

-----【main()函数】-----
// 描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----  

int main()
{
    //【1】载入原始图
    Mat srcImage=imread("1.jpg");

    //【2】显示原始图
    imshow( "均值滤波【原图】", srcImage );

    //【3】进行均值滤波操作
    Mat dstImage;
    blur( srcImage, dstImage, Size(7, 7));

    //【4】显示效果图
    imshow( "均值滤波【效果图】" ,dstImage );
```

```
    waitKey( 0 );
}
```

程序代码非常好理解，载入并显示原始图后，调用一次 blur 函数，最后显示效果图。原始图和效果图分别如图 1.31 和 1.32 所示。



图 1.31 原始图

图 1.32 均值滤波效果图

#### 1.5.4 第四个程序：canny 边缘检测

接着我们来看看如何用 OpenCV 进行 canny 边缘检测。载入图像，并将其转成灰度图，再用 blur 函数进行图像模糊以降噪，然后用 canny 函数进行边缘检测，最后进行显示。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
-----
#include <opencv2/opencv.hpp>
#include<opencv2/imgproc/imgproc.hpp>
using namespace cv;

-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
-----
int main()
{
    //【0】载入原始图
    Mat srcImage = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的素材
```

```
图
imshow("【原始图】Canny 边缘检测", srcImage); //显示原始图
Mat dstImage,edge,grayImage; //参数定义

//【1】创建与 src 同类型和大小的矩阵(dst)
dstImage.create( srcImage.size(), srcImage.type() );

//【2】将原图像转换为灰度图像
//此句代码的 OpenCV2 版为:
//cvtColor( srcImage, grayImage, CV_BGR2GRAY );
//此句代码的 OpenCV3 版为:
cvtColor( srcImage, grayImage, COLOR_BGR2GRAY );

//【3】先使用 3x3 内核来降噪
blur( grayImage, edge, Size(3,3) );

//【4】运行 Canny 算子
Canny( edge, edge, 3, 9,3 );

//【5】显示效果图
imshow("【效果图】Canny 边缘检测", edge);

waitKey(0);

return 0;
}
```

原始图和效果图分别如图 1.33 和 1.34 所示。

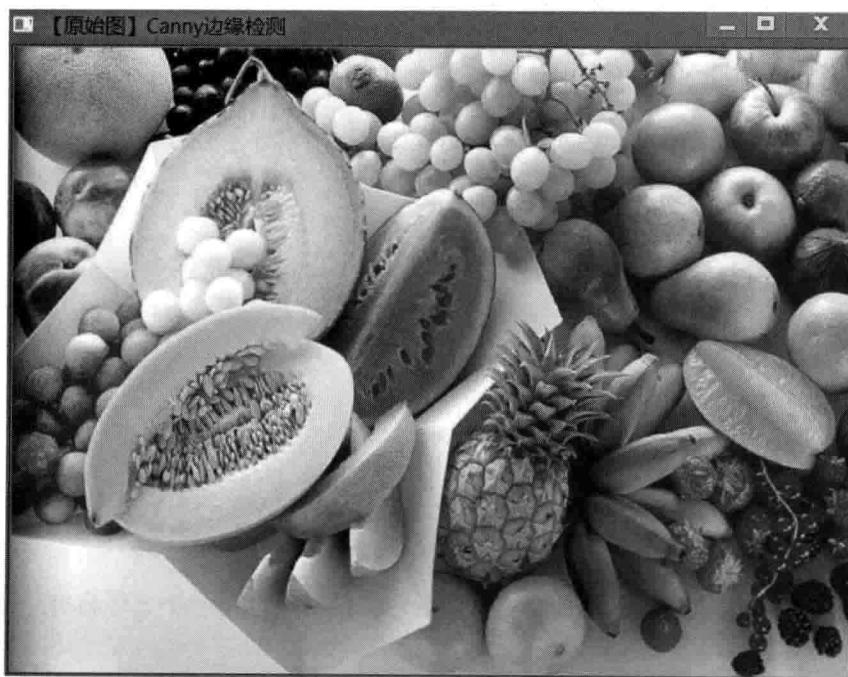


图 1.33 原始图

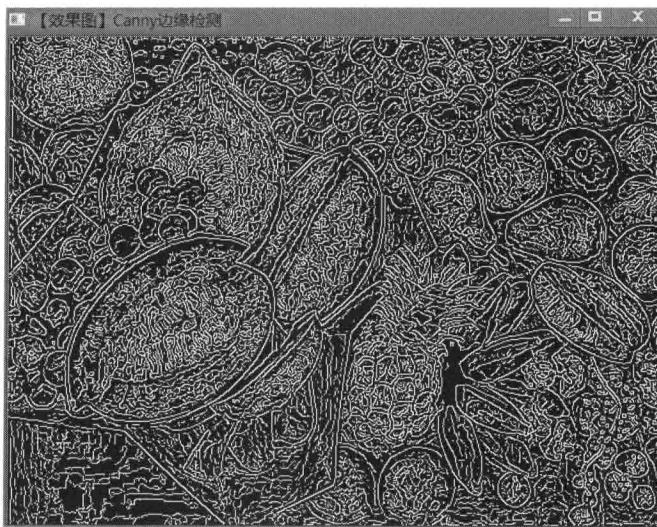


图 1.34 边缘检测效果图

## 1.6 OpenCV 视频操作基础

本节我们将学习如何利用 OpenCV 中的 VideoCapture 类，来对视频进行读取显示，以及调用摄像头。

VideoCapture 是 OpenCV 2.X 中新增的一个类，对应于之前 C 语言版本的 CvCapture 结构体。它提供了从摄像机或视频文件捕获视频的 C++ 接口，作用是从视频文件或从摄像头捕获视频并显示出来。

### 1.6.1 读取并播放视频

通过对 VideoCapture 类的分析，可以发现利用它读入视频的方法一般有如下两种。比如读入的视频为工程路径下名为“1.avi”的视频文件，那么这两种写法分别如下。

(1) 先实例化再初始化：

```
VideoCapture capture;  
capture.open("1.avi");
```

(2) 在实例化的同时进行初始化：

```
VideoCapture capture("1.avi");
```

这两种写法的区别就如我们定义一个 int 类型的变量一样：“int a;a=1;”为先定义再初始化；“int a=1;”为在定义时初始化。

视频读入到 VideoCapture 类对象之后，紧接着可以用一个循环将每一帧显示出来，相关代码如下：

```
//循环显示每一帧  
while(1)
```

```
{  
    Mat frame; //定义一个Mat 变量，用于存储每一帧的图像  
    capture>>frame; //读取当前帧  
    imshow("读取视频",frame); //显示当前帧  
    waitKey(30); //延时 30ms  
}
```

这段代码中，首先定义了一个 Mat 变量，用于存储每一帧的图像，接着读取当前帧到 Mat 变量中，然后调用 imshow 显示当前的这一帧图像，并用 waitKey 延时 30 毫秒，开始下一次循环。

将本节讲解的内容串联起来，就成了这一节颇具代表性且代码相当精简的视频的载入示例程序。在这里贴出其源码给大家参考：

```
#include <opencv2\opencv.hpp>  
using namespace cv;  
  
int main()  
{  
    //【1】读入视频  
    VideoCapture capture("1.avi");  
  
    //【2】循环显示每一帧  
    while(1)  
    {  
        Mat frame; //定义一个Mat 变量，用于存储每一帧的图像  
        capture>>frame; //读取当前帧  
        imshow("读取视频",frame); //显示当前帧  
        waitKey(30); //延时 30ms  
    }  
    return 0;  
}
```

在工程目录下，笔者准备了一个名为“1.avi”的视频文件放置于工程目录之下，程序运行截图如图 1.35 所示。

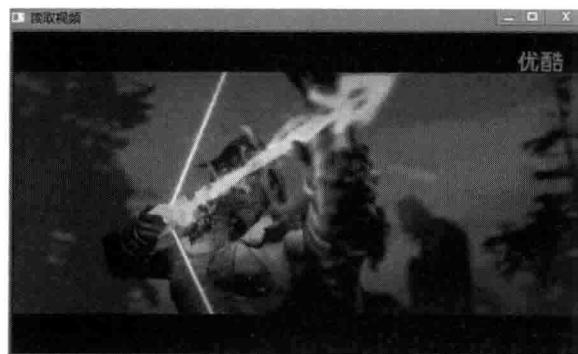


图 1.35 播放视频

## 1.6.2 调用摄像头采集图像

上一小节讲到了视频文件的读入，而如果我们要调用摄像头进行视频采集的

话，将代码 `VideoCapture capture ("1.avi");` 中的“1.avi”换为0就可以了，表示调用摄像头而不是从文件中读取视频。对应于上文讲到的两种写法，即：

(1) 先实例化再初始化

```
VideoCapture capture;  
capture.open(0);
```

(2) 在实例化的同时进行初始化

```
VideoCapture capture(0);
```

通过两种方式中的一种读入视频之后，循环将每一帧显示出来代码和之前1.5.1节讲到的内容完全相同，可以发现，利用`VideoCapture`类调用摄像头采集视频和从文件中读入视频的区别仅仅是在`VideoCapture`类对象初始化时指定的内容的区别，即，是指定文件名如“1.avi”，还是填一个0表示调用摄像头而已。接下来让我们一起看看用OpenCV调用摄像头的精简示例程序，并在代码中熟悉整个调用过程。

```
#include <opencv2/opencv.hpp>  
using namespace cv;  
  
int main()  
{  
    //【1】从摄像头读入视频  
    VideoCapture capture(0);  
  
    //【2】循环显示每一帧  
    while(1)  
    {  
        Mat frame; //定义一个Mat变量，用于存储每一帧的图像  
        capture>>frame; //读取当前帧  
        imshow("读取视频",frame); //显示当前帧  
        waitKey(30); //延时30ms  
    }  
    return 0;  
}
```

摄像头采集到的视频如图1.36所示。

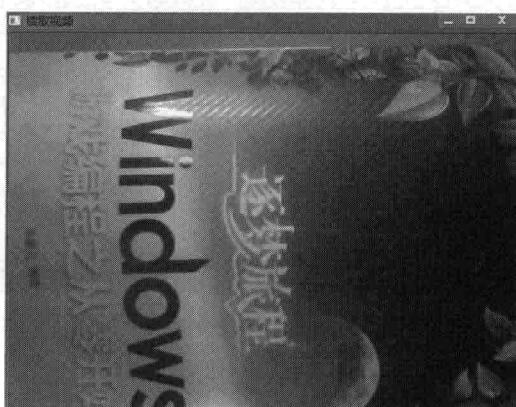


图1.36 摄像头采集到的视频

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

另外，我们还可以用上面的摄像头调用示例程序来配合 canny 边缘检测，得到 canny 边缘检测并高斯模糊后的摄像头采集视频，源码如下：

```
#include "opencv2/opencv.hpp"
using namespace cv;

int main()
{
    //从摄像头读入视频
    VideoCapture capture(0);
    Mat edges;

    //循环显示每一帧
    while(1)
    {
        //【1】读入图像
        Mat frame;//定义一个 Mat 变量，用于存储每一帧的图像
        capture >> frame; //读取当前帧

        //【2】将原图像转换为灰度图像
        cvtColor(frame, edges, CV_BGR2GRAY);//转化 BGR 彩色图为灰度图

        //【3】使用 3x3 内核来降噪 (2x3+1=7)
        blur(edges, edges, Size(7,7));//进行模糊

        //【4】进行 canny 边缘检测并显示
        Canny(edges, edges, 0, 30, 3);
        imshow("被 canny 后的视频", edges); //显示经过处理后的当前帧
        if(waitKey(30) >= 0) break; //延时 30ms
    }
    return 0;
}
```

运行效果图如图 1.37 所示。

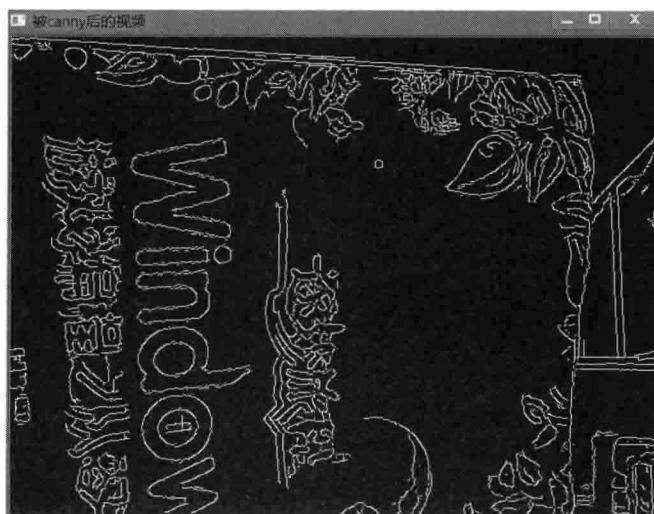


图 1.37 经过 canny 边缘检测后的视频

## 1.7 本章小结

相信通过本章的学习，大家已经对 OpenCV 有了一些初步的认知。本章中，我们不仅介绍了 OpenCV 的周边概念，还分析了其基本架构。然后重点讲解了 OpenCV 的下载、安装与配置过程。配置完成后，我们带领大家正式开始领略 OpenCV 的魅力，接触了四个 OpenCV 图像处理小程序。而章节最后，我们还学习了如何使用 OpenCV 操作视频和调用摄像头。内容可谓非常丰富，相信认真学习完此章，会对接下来 OpenCV 的深入学习打下很好的基础。

### 本章示例程序清单

示例程序序号	程序说明	对应章节
1	OpenCV 环境配置的测试用例	1.3.8
2	快速上手 OpenCV 的第一个程序：图像显示	1.4.1
3	快速上手 OpenCV 的第二个程序：图像腐蚀	1.4.2
4	快速上手 OpenCV 的第三个程序：blur 图像模糊	1.4.3
5	快速上手 OpenCV 的第四个程序：canny 边缘检测	1.4.4
6	读取并播放视频	1.5.1
7	调用摄像头采集图像	1.5.2



可以根据每章小结或者书中提供的示例程序的序号，在本书配套示例程序包中快速查询到需要查看的工程源代码或者可运行.exe程序。

# 第 2 章

## 启程前的认知准备

### 导读

---

本章中，你将学到：

- OpenCV 官方例程的引导学习与赏析
- 如何编译 OpenCV 的源代码
- 对 “opencv.hpp” 头文件的认知
- 本书的命名规范约定
- argc 与 argv 参数解惑
- 格式输出函数 printf() 用法小议

本章内容可以进行跳跃性的阅读，阅读顺序由读者自行选择。其中 2.5、2.6 节内容为 C 语言基础薄弱的读者所准备，有一定的 C 语言基础的读者对这两节可以略读。

## 2.1 OpenCV 官方例程引导与赏析

OpenCV 作为一个在全球使用人数众多的计算机视觉库，其实官方已经准备了大量的示例程序，供广大初学者学习。而市面上绝大多数的 OpenCV 书籍教程和网络博文的第一手知识来源，就是这些官方提供的技术文档和示例程序。

本节旨在带领大家粗略了解 OpenCV 官方提供的示例程序，看看学好 OpenCV 之后，可以写出哪些炫酷的程序来。

经过第 1 章的学习，我们已经安装、下载并配置好了 OpenCV。在 OpenCV 安装目录下，可以找到 OpenCV 官方提供的示例代码。具体位于…\opencv\sources\samples\cpp 目录下。

如图 2.1 所示，若将 OpenCV 安装于 D:\Program Files(x86)\路径，则源码的存放目录应为 D:\Program Files (x86)\opencv\sources\samples\cpp。

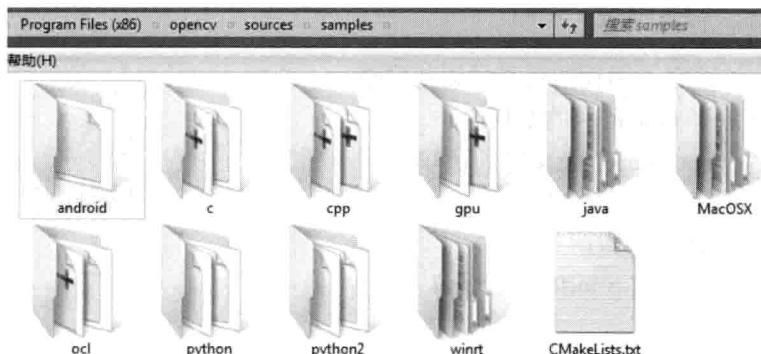


图 2.1 OpenCV 官方示例程序文件夹

通过观察文件名可以发现，OpenCV 官方提供了 Android、C、CPP、GPU、Java、Mac OS X、OCL、Python、Python2、Win RT 等众多版本的示例程序。不难理解，名为 c 的文件夹中存放着 OpenCV1.0 等旧版本的示例程序，而名为 cpp 的文件夹中存放着 OpenCV 2.X 等新版本的示例程序。

本书旨在对新版本 OpenCV 的讲解，自然主要专注于 cpp 文件夹内的众多示例程序。打开该文件夹，可以发现 100 余个 C++ 版本的 OpenCV 官方示例程序。在…\opencv\sources\samples\cpp\tutorial\_code 路径下，存放着和官方教程配套的示例程序。其内容按 OpenCV 各组件模块而分类，非常适合学习，大家可以按需查询，分类学习，各个击破。

接下来，笔者将带着大家在此 100 余个示例程序中选择比较有特点的几个，得出运行效果。为了方便大家学习，以下的几个示例都和本书的其他主线示例程序一样，在配套代码包中提供了可以直接运行的执行文件与源工程。请在配套示例程序包中的第 2 章中找到第 8 到第 13 这 6 个配套示例程序（它们分别对应了 2.1.1 到 2.1.6 这 5 节的内容，2.1.5 节分为了两个工程），对其进行编译运行并查看结果。

### 2.1.1 彩色目标跟踪：Camshift

本小节讲解的例程为彩色目标跟踪，程序的用法是根据鼠标框选区域的色度光谱来进行摄像头读入的视频目标的跟踪。其主要采用 CamShift 算法，全称是“Continuously Adaptive Mean-SHIFT”，是对 MeanShift 算法的改进，被称为连续自适应的 MeanShift 算法。打开本书配套的第 8 个示例程序（或者是在…\opencv\sources\samples\cpp 目录下找到名为 camshiftdemo.cpp 的文件，自行修改部分代码。实际路径会因为 OpenCV 版本的不同略有差异），编译并运行，用鼠标在窗口中框选要跟踪的区域，便可以得到如图 2.2、2.3、2.4 所示运行截图。



图 2.2 彩色目标跟踪运行截图（1）

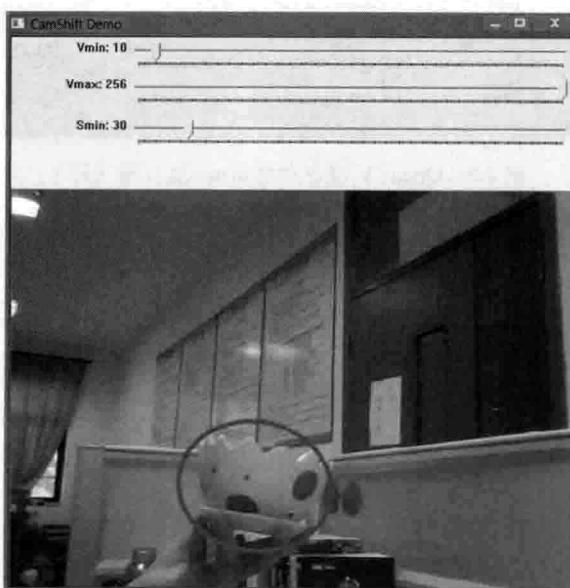


图 2.3 彩色目标跟踪运行截图（2）

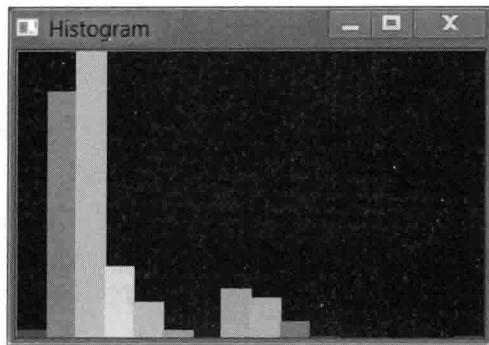


图 2.4 对应的直方图

### 2.1.2 光流：optical flow

光流（optical flow）法是目前运动图像分析的重要方法，由 Gibson 于 1950 年首先提出。光流用来指定时变图像中模式的运动速度，因为当物体在运动时，在图像上对应点的亮度模式也在运动。这种图像亮度模式的表观运动（apparent motion）就是光流。光流表达了图像的变化，由于它包含了目标运动的信息，因此可被观察者用来确定目标的运动情况。图 2.5 和 2.6 即为 OpenCV 官方为我们准备的示例程序的运行效果图。

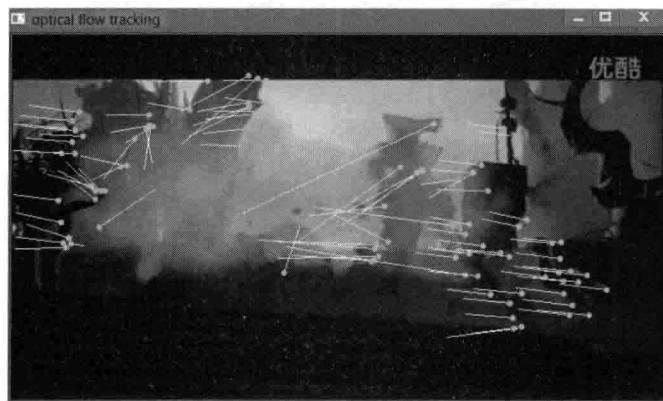


图 2.5 OpenCV 官方光流 demo 运行图（1）

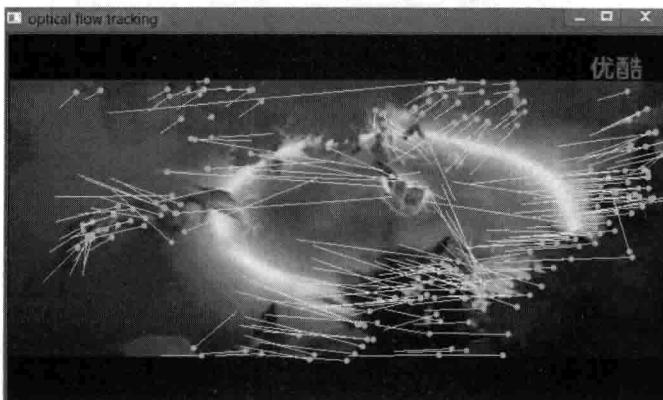


图 2.6 OpenCV 官方光流 demo 运行图（2）

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

### 2.1.3 点追踪: lkdemo

在…\opencv\sources\samples\cpp 目录下（实际路径会因为 OpenCV 版本的不同略有差异）的 lkdemo.cpp 文件中，存放着这样一个精彩的例程。程序运行后，会自动启用摄像头，这时按键盘上的“r”键来启动自动点追踪，便可以看到如图 2.7 所示的效果图 1。而我们在摄像头中移动物体，可以看到物体上的点随着物体一同移动（如图 2.8 所示），富含科技感且妙趣横生。



图 2.7 点追踪程序运行图（1）

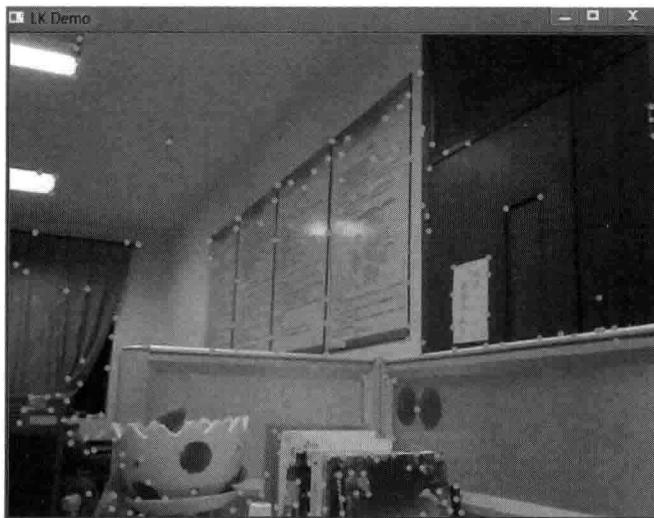


图 2.8 点追踪程序运行图（2）

### 2.1.4 人脸识别: objectDetection

人脸识别是图像处理与 OpenCV 非常重要的应用之一，OpenCV 官方专门有教程和代码讲解其实现方法。此示例程序就是使用 objdetect 模块检测摄像头视频流中的人脸，位于…\opencv\sources\samples\cpp\tutorial\_code\objectDetection 路径

之下。需要额外注意的是，需要将“…\opencv\sources\data\haarcascades”路径下的“haarcascade\_eye\_tree\_eyeglasses.xml”和“haarcascade\_frontalface\_alt.xml”文件复制到和源文件同一目录中，才能正确运行。运行程序，将自己的脸对准摄像头，或者放置一张照片对准摄像头任其捕获，便可以发现程序准确地识别出了人脸，并用彩色的圆将脸圈出。如图 2.9 所示。

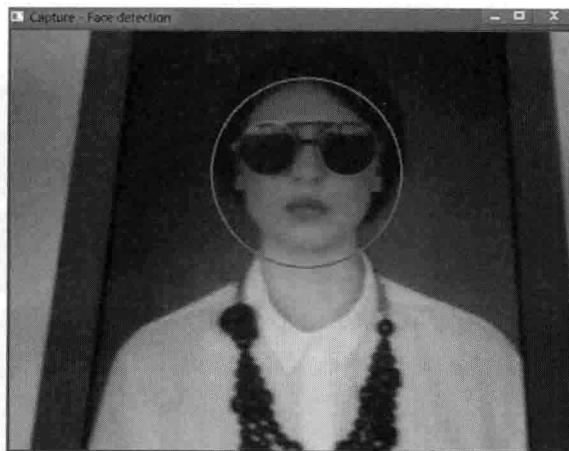


图 2.9 人脸识别效果图

### 2.1.5 支持向量机引导

在 OpenCV 的机器学习模块中，官方为我们准备了两个示例程序。第一个程序是使用 CvSVM::train 函数训练一个 SVM 分类器，如图 2.10 所示。

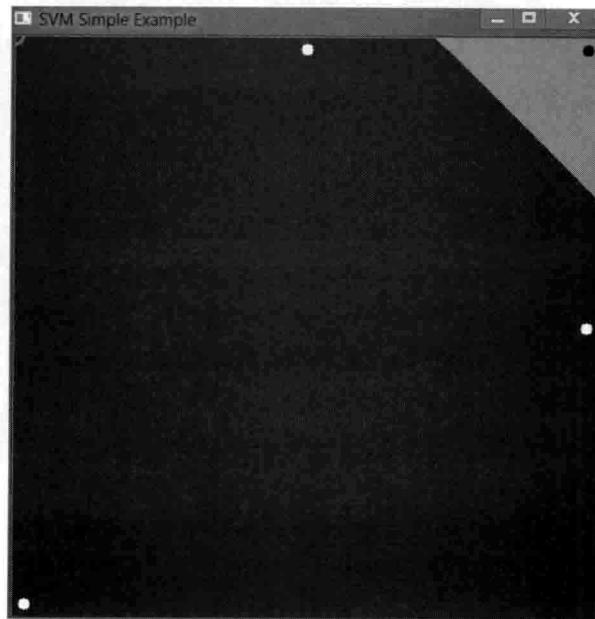


图 2.10 训练 SVM 分类器程序截图

第二个示例程序主要用于讲解在训练数据线性不可分时，如何定义支持向量

机的最优化问题。如图 2.11 所示。

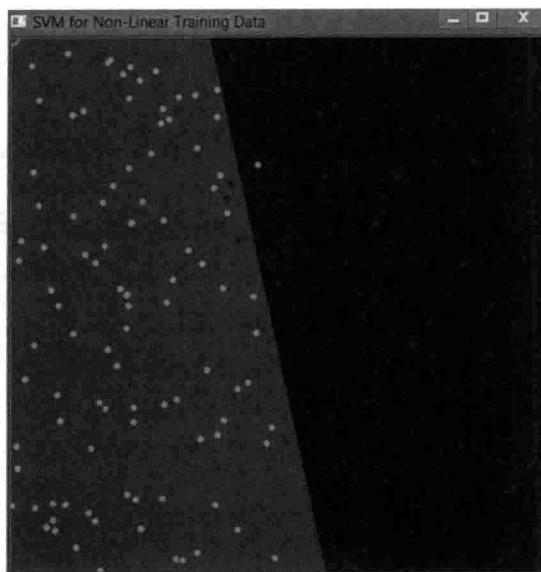


图 2.11 SVM 对线性不可分数据的处理程序截图

以上就是对 OpenCV 官方提供的示例程序的引导和概览。

## 2.2 开源的魅力：编译 OpenCV 源代码

本节，我们将一起探讨如何通过已经安装的 OpenCV，选择不同的编译器类型，生成高度还原的 OpenCV 开发时的解决方案工程文件，并欣赏 OpenCV 新版本中总计六十六多万行的精妙源代码。我们可以对源代码进行再次编译，得到二进制文件，或者修改原版官方的 OpenCV 代码，并编译后为自己所用，从而为深入理解 OpenCV 的开源魅力迈出坚实的一步。

### 2.2.1 下载安装 CMake

想要在 Windows 平台下生成 OpenCV 的解决方案，需要一个名为 CMake 的开源软件。note: CMake，是“crossplatform make”的缩写，它是一个跨平台的安装（编译）工具，可以用简单的语句来描述所有平台的安装（编译过程）。他能够输出各种各样的 makefile 或者 project 文件，能测试编译器所支持的 C++ 特性，类似 UNIX 下的 automake。只是 CMake 的组态档取名为 CmakeLists.txt。Cmake 并不直接建构出最终的软件，而是产生标准的建构档（如 Unix 的 Makefile 或 Windows Visual C++ 的 projects/workspaces），然后再依一般的建构方式使用。这使得熟悉某个集成开发环境（IDE）的开发者可以用标准的方式建构他的软件，这种可以使用各平台的原生建构系统的能力是 CMake 和 SCons 等其他类似系统的区别之处。

CMake 可以在官网：<http://www.cmake.org/> 上下载到。打开此链接，首先转到其下载页面，如图 2.12 所示。下载页面的 Source distributions 处可以下载到 CMake 软件的源码，对这款开源软件感兴趣的读者不妨研究一下。

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>



图 2.12 CMake 源码下载页面

在 Binary distributions 处可以下载到 CMake 的执行文件（如图 2.13），我们选择 Windows (Win32 Installer) 版的进行下载。

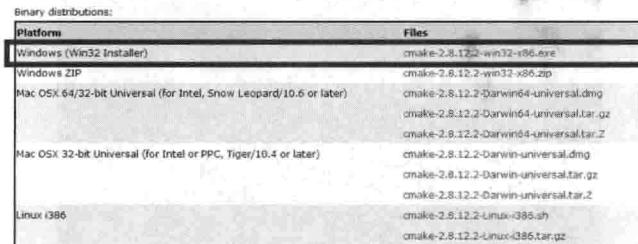


图 2.13 CMake 软件下载页面

下载完成之后便是安装 CMake，这一步很简单，在此不再赘言。安装完成后，如果没有生成桌面快捷方式，则在安装路径下（如 D:\Program Files(x86)\CMake 2.8\bin）处找到【△cmake-gui.exe】运行。

### 2.2.2 使用 CMake 生成 OpenCV 源代码工程的解决方案

这一步是全文的核心内容，为了使讲解条理清晰，我们进行分步介绍。

#### 1. 【第一步】运行 cmake-gui

上文已经讲到，如果没有生成桌面快捷方式，在安装路径下（如 D:\Program Files(x86)\CMake 2.8\bin 处）找到【△cmake-gui.exe】运行。

#### 2. 【第二步】指定 OpenCV 的安装路径

如图，单击方框内的“Browse Source”按钮，在弹出的对话框中指定 OpenCV 安装时源代码的存储路径。图中以 OpenCV 安装在 D:\Program Files 下为例，在此选择路径：D:\Program Files\opencv\sources。

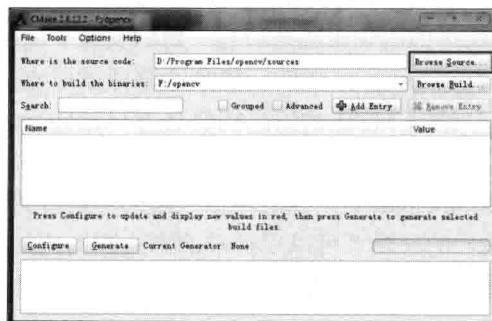


图 2.14 指定 OpenCV 的安装路径

我们可以发现，此路径下必须会有一个名为 CMakeLists.txt 的文件，这就是

给 CMake 留下的配置文件。CMake 可以根据这个配置文件，通过选择不同的编译器，来生成不同的解决方案——VisualStudio 的编译器对应的就是生成 Visual Studio 版的 sln 解决方案。

### 3. 【第三步】指定解决方案的存放路径

单击对话框中的“Browse Build”按钮，在弹出的对话框中指定存放生成的 opencv 解决方案的路径。比如 F:/opencv。

### 4. 【第四步】第一次 Configure

路径都设置好后，单击如图 2.15 所示的【Configure】按钮，进行第一次配置过程。

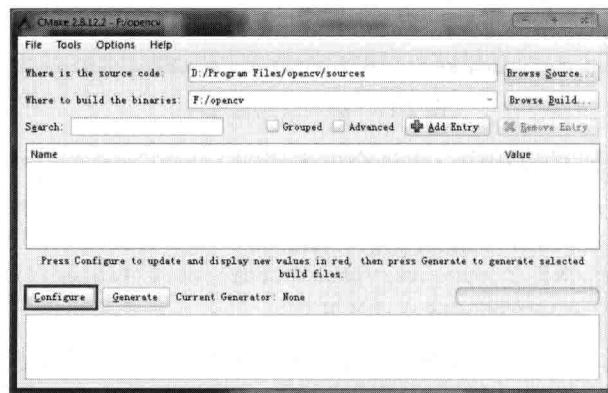


图 2.15 第一次 Configure

然后会弹出如图 2.16 所示进行编译器选择的对话框。

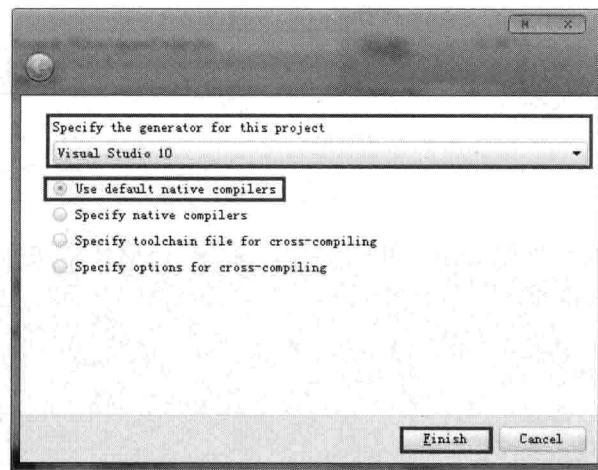


图 2.16 编译器选择

先选定“Use default native compilers”，然后可以发现下拉列表中提供了几十种编译器供选择。因为我们安装了 Visual Studio，这里会默认选择对应版本的 Visual Studio 编译器，比如 Visual Studio 10（即之后会生成对应 VS2010 的 sln 解决方案）。

确认无误后，单击“finish”按钮。于是，CMake 开始第一次源代码配置过程，如图 2.17 所示。

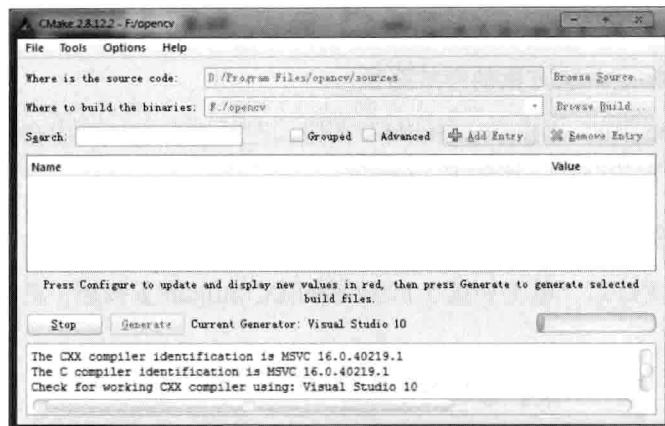


图 2.17 第一次配置过程

在 CMake 处理的过程中，也许会出现诸如

```
"Could not copy from: D:/Program Files(x86)/CMake 2.8/share/cmake
2.8/Templates/CMakeVSMacros2.vsmacros
to: C:/Users/浅墨/Documents/VisualStudio 2010/Projects/VSMacros80/
CMakeMacros/CMakeVSMacros2.vsmacros"
```

的红色字样警告，因为这是系统用户的路径名有中文字符“浅墨”。CMake 不识别中文路径，只要我们在上面的第三步中设置生成的路径中没有中文就行了。就算有中文而造成了这个错误，也对我们这次的生成无碍，不用去管它。

看到进度条读到尽头，出现 Configuring done 字样，第一次的源码配置就完成了，如图 2.18 所示。

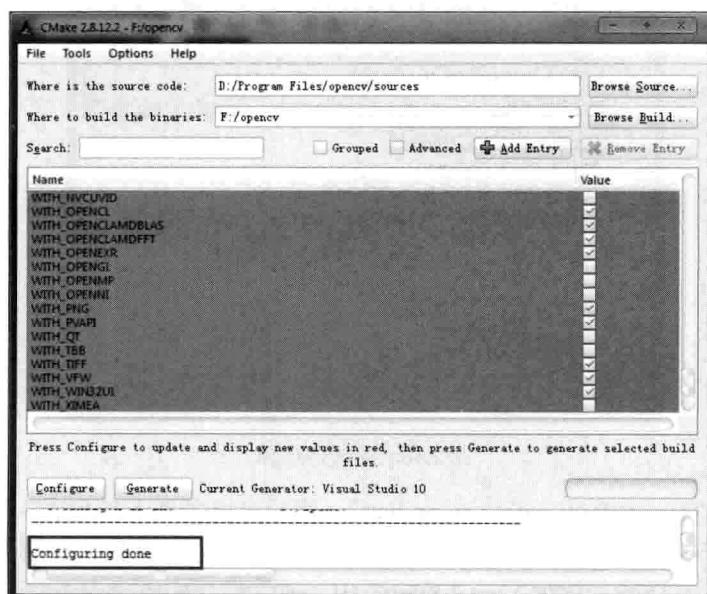


图 2.18 第一次配置完成

## 5. 【第五步】第二次 Configure

第一次配置完成之后，还需要进行第二次配置，于是再次单击“Configure”按钮。这次的配置很快，几秒钟就会再次出现“Configuring done”字样，并且选中部分也都正常了。如图 2.19 所示。

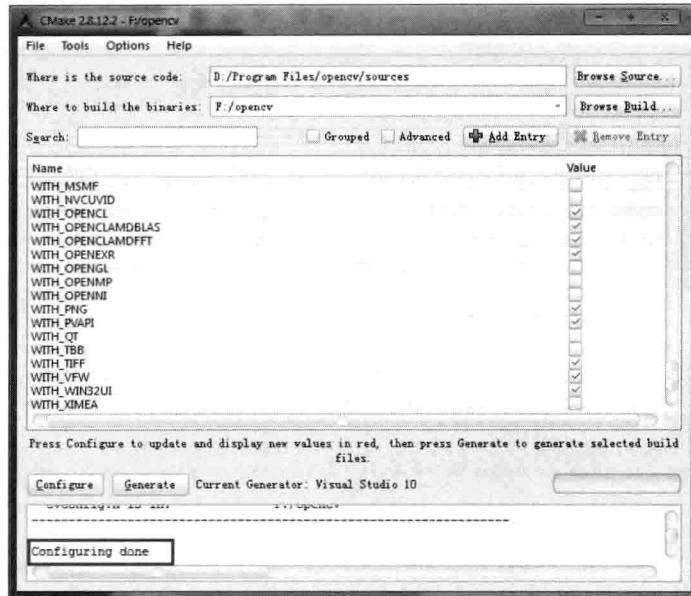


图 2.19 第二次配置完成

## 6. 【第六步】单击 Generate，成功生成项目

这样，就只需要单击 Generate 按钮，来生成最终的解决方案了。如图 2.20 所示。

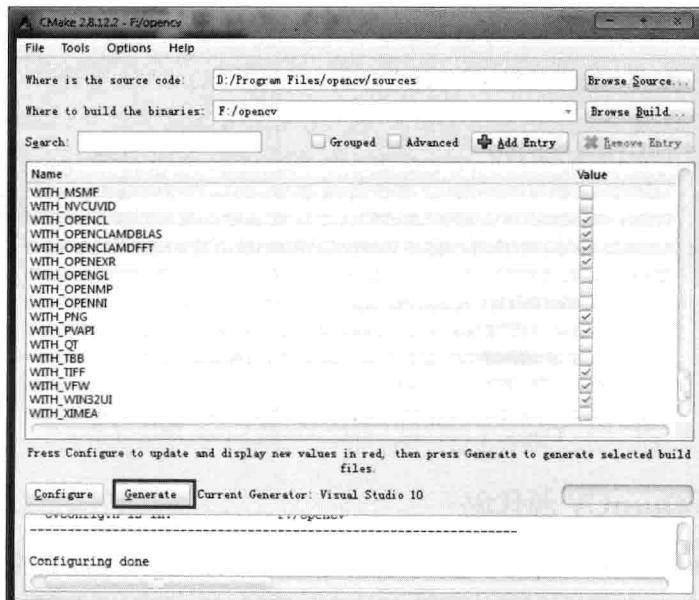


图 2.20 单击生成【Generate】按钮

因为之前已经有过两次的 configure 过程，所以生成解决方案也将会非常快。如图 2.21 所示。

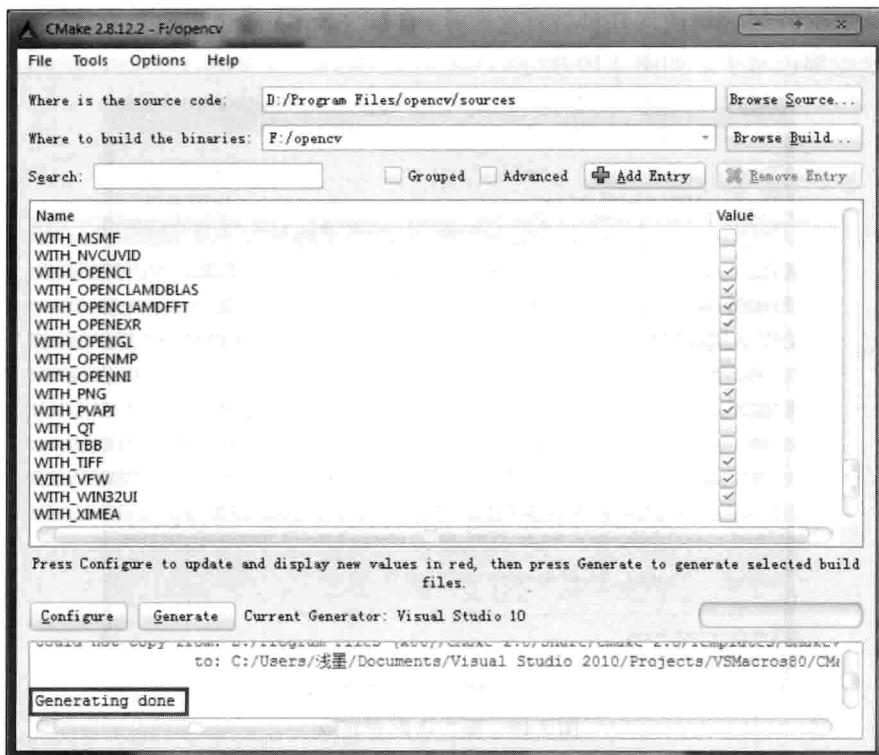


图 2.21 成功生成解决方案

出现 Generating done 字样，就表示大功告成，可以去之前指定的路径（F:\opencv）下找寻生成的解决方案了。

**Note**

注意：2.4.8、2.4.9 的 OpenCV 用 CMake 生成的工程只有 3M 多，相比之前测试 2.4.6 版本的 OpenCV 有 3 个多 G 的工程，笔者一开始都以为生成出错了。但是点开 sln 工程，发现里面依然可以看到源代码。这是因为从 2.4.7 版本起，OpenCV 源代码就直接包含在了 opencv 的安装路径下，我们生成的 sln 工程，也只是链接到了 opencv 安装路径下的源文件而已，工程本身并不大。工程文件如图 2.22 所示。



图 2.22 OpenCV 源代码工程的 Visual Studio 解决方案文件

### 2.2.3 编译 OpenCV 源代码

打开刚刚生成的“OpenCV.sln”解决方案，可以看到一个庞大的工程——这是一个包含了 67 个项目的解决方案（对 OpenCV 2.4.9 而言），如图 2.23 所示。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

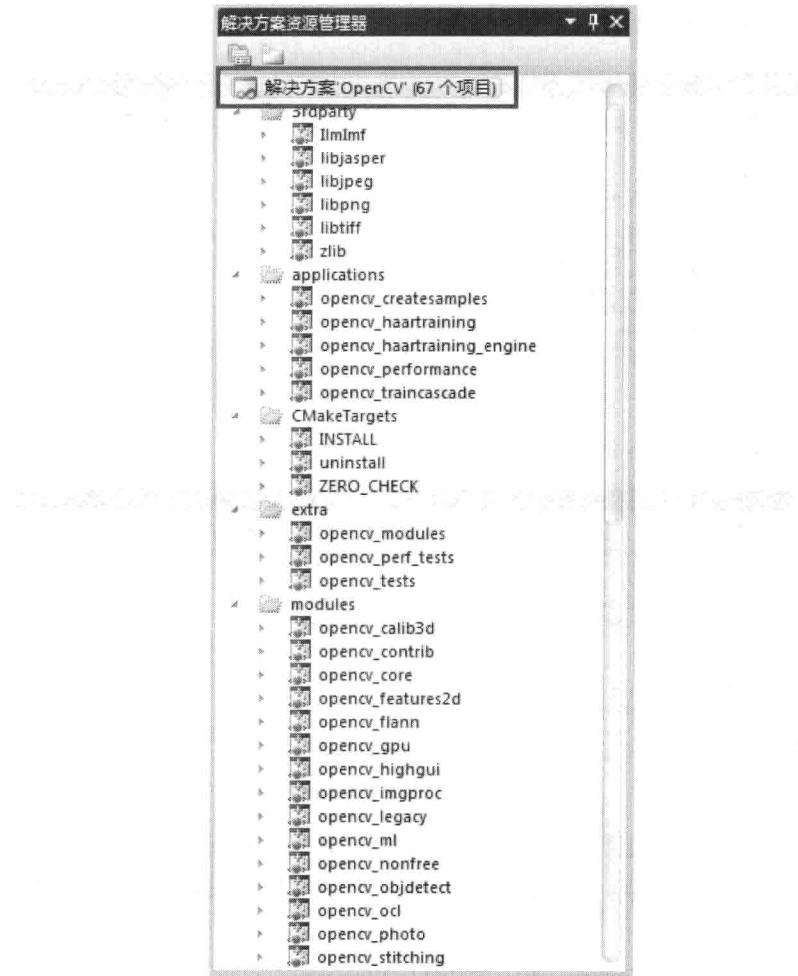


图 2.23 OpenCV 源代码的资源管理器视图

大家这时在解决方案资源管理界面，可以随意点进去一个感兴趣的项目，比如 `opencv_core`，再在 `opencv_core/Src/matrix.cpp` 查看其中某个文件的源代码。笔者截的这张图（图 2.24）是大家以后都会很熟悉的 Mat 类型的某个构造函数的源代码。

```

matrix.cpp x jconfig.h
cv::Mat
265 Mat::Mat(const Mat& m, const Range & _rowRange, const Range & _colRange) : size(&m.size)
266 {
267     initEmpty();
268     CV_Assert(m.dims >= 2);
269     if(m.dims > 2)
270     {
271         if(_rowRange < Range)
272             r[0] = _rowRange;
273         r[1] = _rowRange;
274         r[2] = _colRange;
275         for(int i = 2; i < m.dims; ++i)
276             r[i] = 0;
277         *this = m(r);
278     }
279 }

```

图 2.24 Mat 类型某构造函数的源代码

对源代码进行欣赏之后，我们可以按【F5】或者使用其他操作来启动调试，

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

编译过程如图 2.25 所示。

```
输出
显示输出来源(S): 生成
8> half.cpp
6> 正在生成代码...
8> IexBaseExc.cpp
8> IexThrowErrnoExc.cpp
8> IlmThread.cpp
8> IlmThreadMutex.cpp
8> IlmThreadMutexWin32.cpp
8> IlmThreadPool.cpp
8> IlmThreadSemaphore.cpp
8> IlmThreadSemaphoreWin32.cpp
8> IlmThreadWin32.cpp
```

图 2.25 OpenCV 源代码编译过程

编译结果如图 2.26 所示。

```
输出
显示输出来源(S): 生成
62> CMake does not need to re-run because F:\openCV2.4.8\CMakeFiles\generate.stamp is up-to-date.
62> Build all projects
62>FinalizeBuildStatus:
62> 正在删除文件“Win32\Debug\ALL_BUILD\ALL_BUILD.unsuccessfulbuild”。
62> 正在对“Win32\Debug\ALL_BUILD\ALL_BUILD.lastbuildstate”执行 Touch 任务。
62>
62>生成成功。
62>
62>已用时间 00:00:01.84
===== 生成: 成功 62 个, 失败 0 个, 最新 0 个, 跳过 0 个 ======
```

图 2.26 OpenCV 源代码编译结果

在实验过程中，编译总共用时约 5 分钟，而具体时间和机器配置息息相关。从截图 2.26 中可以发现，这次编译成功了 62 个项目，失败了 0 个，即全部编译生成成功。

另外，编译完成会得到如图 2.27 所示的警告。

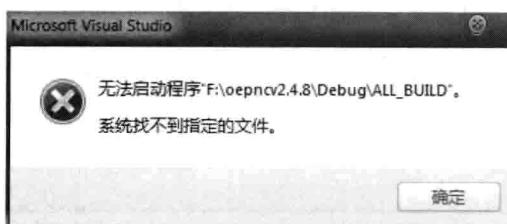


图 2.27 “系统找不到指定文件”错误提示对话框

若编译完成时出现这样的对话框，无须紧张，因为这是正常的。原因是 OpenCV 的源代码工程默认将“ALL\_BUILD”这个项目设为了启动项，编译成功后，就会默认运行它。



图 2.28 被默认设置为启动项的“ALL\_BUILD”项目

## Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

而“ALL\_BUILD”是一个项目生成周边的杂项，不是exe执行文件，本身不可以运行，所以自然就会弹出这样的错误提示。

若想让此报错的对话框不显示，指定一个另外的启动项即可。我们可以在解决方案资源管理器里，对需要设为启动项的那个项目右键点击，在弹出的菜单项中单击【设为启动项(J)】进行设定。如下图：



图 2.29 设置新的启动项目



编译 OpenCV，其实就是得到了一些二进制的生成文件，如 dll、lib 和 exe。因为是在 debug 下编译的，所以在工程目录的 bin\debug 下会生成 OpenCV 对应版本的依赖库（700 多 MB，有点大），可以供以后调用此次编译的 OpenCV 时使用。

## 2.3 “opencv.hpp” 头文件认知

在任意一个 OpenCV 程序中，通过转到定义，我们可以发现“#include <opencv2/opencv.hpp>”一句中的头文件定义类似如下：

```
#include <opencv2/opencv.hpp>
#ifndef __OPENCV_ALL_HPP__
#define __OPENCV_ALL_HPP__

#include "opencv2/core/core_c.h"
#include "opencv2/core/core.hpp"
#include "opencv2/flann/miniflann.hpp"
#include "opencv2/imgproc/imgproc_c.h"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/photo/photo.hpp"
#include "opencv2/video/video.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/calib3d/calib3d.hpp"
```

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
#include "opencv2/ml/ml.hpp"
#include "opencv2/highgui/highgui_c.h"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/contrib/contrib.hpp"

#endif
```

通过观察代码可知，`opencv.hpp` 中已经包含了 OpenCV 各模块的头文件，如高层 GUI 图形用户界面模块头文件“`highgui.hpp`”、图像处理模块头文件“`imgproc.hpp`”、2D 特征模块头文件“`features2d.hpp`”等。

所以，我们在编写 `core`、`objdetect`、`imgproc`、`photo`、`video`、`features2d`、`objdetect`、`calib3d`、`ml`、`highgui`、`contrib` 模块的应用程序时，原则上仅写上一句“`#include <opencv2/opencv.hpp>`”即可，这样可以精简优化代码。

而本书为了便于演示和帮助大家理解，在编写特定模块的程序时，默认情况下还是包含了相应模块的头文件。

## 2.4 命名规范约定

使用一套成熟的命名规则，不仅可以让规范行事，还可以让别人在阅读我们书写的代码时，可以更好更快地理解我们的思路，从而增强了代码的可读性，方便各程序员之间相互交流代码。

《代码大全（第二版）》书中第 277 页，表 11-3 展示的一套成熟的命名规则，笔者对其细节部分进行细微地改进，得到如表 2.1 所示命名约定。

表 2.1 命名规则约定

描述	实例
类名混合使用大小写，首字母大写	ClassName
类型定义，包括枚举和 <code>typedef</code> ，混合使用大小写，首字母大写	TypeName
枚举类型除了混合使用大小写外，总以复数形式表示	EnumeratedTypes
局部变量混合使用大小写，且首字母小写，其名字应该与底层数据类型无关，而且应该反映该变量所代表的事物	localVariable
子程序参数的格式混合使用大小写，且每个单词首字母大写，其名字应该与底层数据类型无关，而且应该反映该变量所代表的事物	RoutineParameter
对类的多个子程序可见（只对该类可见）的成员变量名用 <code>m_</code> 前缀	<code>m_ClassVariable</code>
局部变量名用 <code>g_</code> 前缀	<code>g_GlobalVariable</code>
具名常量全部大写	CONSTANT
宏全部大写，单词间用分隔符“ <code>_</code> ”隔开	SCREEN_WIDTH
枚举类型成员名用能反映其基础类型的、单数形式的前缀。例如， <code>Color_Red</code> , <code>Color_Blue</code>	Base_EnumeratedType

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

说到命名规则，当然少不了匈牙利命名法。匈牙利命名法是编程时的一种命名规范，其基本原则是：变量名=属性+类型+对象描述，其中每一对象的名称都要求有明确含义，可以取对象名字全称或名字的一部分。命名要基于容易记忆容易理解的原则，并要求保证名字的连贯性。据说这种命名法是一位叫 Charles Simonyi 的匈牙利程序员发明的，后来他在微软工作了几年，于是这种命名法就通过微软的各种产品和文档资料向世界传播开了。现在，大部分程序员不管使用什么软件进行开发，或多或少都沿用了这种命名法。匈牙利命名法的出发点是把变量名按“属性+类型+对象描述”的顺序组合起来，以便程序员命名变量时对变量的类型和其他属性有直观的了解。下面是匈牙利变量命名法的一些规范。

匈牙利命名法中常用的小写字母的缀如表 2.2 所示。

表 2.2 变量命名规范

前缀写法	类型	描述	实例
ch	char	8 位字符	chGrade
ch	TCHAR	如果_UNICODE 定义，则为 16 位字符	chName
b	BOOL	布尔值	bEnable
n	int	整型（其大小依赖于操作系统）	nLength
n	UINT	无符号值（其大小依赖于操作系统）	nHeight
w	WORD	16 位无符号值	wPos
l	LONG	32 位有符号整型	lOffset
dw	DWORD	32 位无符号整型	dwRange
p	*	指针	pDoc
lp	FAR*	远指针	lpszName
lpsz	LPSTR	32 位字符串指针	lpszName
lpsz	LPCSTR	32 位常量字符串指针	lpszName
lpsz	LPCTSTR	如果_UNICODE 定义，则为 32 位常量字符串指针	lpszName
h	handle	Windows 对象句柄	hWnd
lpfn	callback	指向 CALLBACK 函数的远指针	LpfnName

关键字字母组合如表 2.3 所示。

表 2.3 关键字字母组合

描述内容	使用的关键字字母组合
最大值	Max
最小值	Min
初始化	Init
临时变量	T (或 Temp)
源对象	Src
目的对象	Dst

虽然命名规范摆在这里，但我们在实际应用中切记不要迂腐，不要墨守成规，该变通的时候可以做适当的变通。且一旦确定了一套合适且成熟命名规范后，最好不要中途更改。

## 2.5 argc 与 argv 参数解惑

### 2.5.1 初识 main 函数中的 argc 和 argv

在与 OpenCV 打交道时，我们常会在相关的示例程序中见到 argc 和 argv 这两个参数，如 OpenCV 的官方示例程序 Samples 中、OpenCV 经典教材《Learning OpenCV》中，等等。这常常会让众多初学者疑惑，不清楚它们是何用途。本节内容正为解此惑而写。

argc 和 argv 中的 arg 指的是“参数”（例如：arguments, argument counter 和 argument vector）。其中，argc 为整数，用来统计运行程序时送给 main 函数的命令行参数的个数；而\* argv[ ]：为字符串数组，用来存放指向字符串参数的指针数组，每一个元素指向一个参数。

Argc, argc 这两个参数一般在用命令行编译程序时有用。在初学 C++ 时，往往要弱化 argc 和 argv 的用法，main 函数常常不带参数，如下。

```
int main()
{
}
```

而在 opencv 的官方示例程序中，main 函数的写法常常会带上两个形参，一般为 argc 和 argv，并且在函数体内部会使用到这两个形参，如下：

```
int main( int argc, char** argv )
{
const char* imagename = argc > 1 ? argv[1] : "lena.jpg";
.....
}
```

其实，带形参的 main 函数，如 main (int argc, char \*argv[ ], char \*\*env),

## Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

是 UNIX、Linux 以及 Mac OS 操作系统中 C/C++ 的 main 函数的标准写法，并且是血统最纯正的 main 函数的写法。可能是由于外国的专家们更习惯使用 UNIX、Linux 以及 Mac OS 等操作系统，所以我们接触到由他们开发和维护的 OpenCV 这款开源视觉库的时候，自然会发现代码中常有 argc 和 argv 的出现。

### 2.5.2 argc、argv 的具体含义

argc 和 argv 这两个参数一般在用命令行编译程序时有用。

主函数 main 中变量（int argc, char \*argv[ ]）的含义

有些编译器允许将 main() 的返回类型声明为 void，这就已不再是合法的 C++ 了。

其实，main (int argc, char \*argv[ ], char \*\*env) 才是 UNIX 和 Linux 中的标准写法。其中，第一个参数，int 类型的 argc，为整型，用来统计程序运行时发送给 main 函数的命令行参数的个数，在 Vsual Studio 中默认值为 1。第二个参数，char\* 类型的 argv[ ]：为字符串数组，用来存放指向的字符串参数的指针数组，每一个元素指向一个参数。各成员含义如下：

- argv[0]指向程序运行的全路径名
- argv[1]指向在 DOS 命令行中执行程序名后的第一个字符串
- argv[2]指向执行程序名后的第二个字符串
- argv[3]指向执行程序名后的第三个字符串
- argv[argc]为 NULL

需要指出，argv[1] 对应于【项目属性】→【配置属性】→【调试】→【命令参数】中的值。记住双引号也要带上，比如读取名为 1.jpg 的图片，如图 2.30 所示，就要在命令参数中填字符串“1.jpg”。

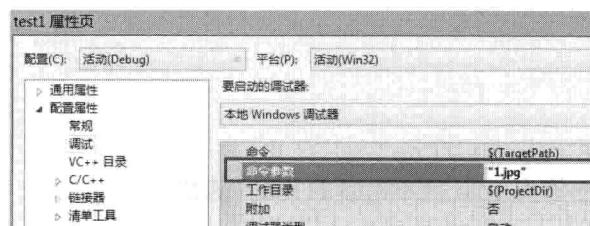


图 2.30 属性页中的命令参数（1）

而如果有多个字符串，则用空格隔开。比如要读入两张名称分别为 1.jpg 和 2.jpg 的图片，在命令参数中填“1.jpg”“2.jpg”（“1.jpg”和“2.jpg”之间用空格隔开）即可，如图 2.31 所示。

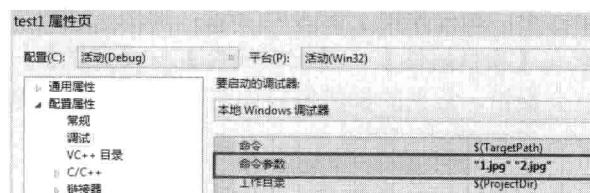


图 2.31 属性页中的命令参数（2）

第三个参数，char \*\*类型的env，为字符串数组。env[ ]的每一个元素都包含ENVVAR=value形式的字符串。其中ENVVAR为环境变量，value为ENVVAR的对应值。在OpenCV中很少使用它。



argc、argv和env是在main()函数之前被赋值的。其实，main()函数严格意义上并不是真正的程序入口点函数，往往入口点还与操作系统有关。而在Windows的控制台应用程序中，将main()函数作为程序入口点，并且很少使用argc、argv等命令行参数。

### 2.5.3 Visual Studio中main函数的几种写法说明

需要注意的是，在如今各版本的Visual Studio编译器中，main()函数带参数argc和argv或不带，也就是说，无论我们是否在函数体中使用argc和argv，返回值为void或不为void，都是合法的。

即至少有如下3种写法合法。

#### 1.【写法一】返回值为整型带参的main函数

```
int main( int argc, char** argv )
{
    //函数体内使用或不使用argc和argv都可行
    .....
    return 1;
}
```

#### 2.【写法二】返回值为整型不带参的main函数

```
int main( int argc, char** argv )
{
    //函数体内使用了argc或argv
    .....
    return 1;
}
```

#### 3.【写法三】返回值为void且不带参的main函数

```
int main()
{
    .....
    return 1;
}
```

在Visual Studio中，如果使用了argv或argc，即上文代码中的第一种写法，且在使用之前没有在【项目属性】→【配置属性】→【调试】→【命令参数】中指定参数的值，就会报错，常见的报错窗口如图2.32所示，这是研究OpenCV官方提供的示例程序时经常碰到的错误。

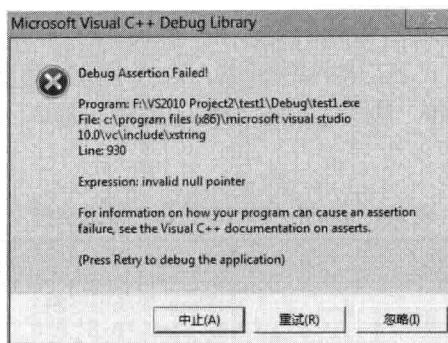


图 2.32 命令参数缺失时常见的报错窗口

想要解决此问题并编译通过，除了上文讲过的在项目属性页中填上命令参数外，最简单的做法就是在不影响原始程序的基础上，将和 argv 或 argc 有关的代码进行替换或注释。

比如将“Mat srcImage=imread (argv[1], 1); //读取字符串名为 argv[1]的图片”替换为“Mat srcImage=imread (“1.jpg”,1); //工程目录下有一张名为 “1.jpg”的图片”。

#### 2.5.4 总结

讲解至此，读者应该对 argc 和 argv 有了比较透彻的认识。简单来说

- int argc 表示命令行字符串的个数
- char \*argv[]表示命令行参数的字符串

由于我们使用的开发环境为 Visual Studio，往后若在 OpenCV 相关代码中遇到这两个参数，可以在代码中将其用路径字符串替换，或者在项目属性页中给其赋值，而对程序整体影响不大的部分就将其注释掉。

## 2.6 格式输出函数 printf()简析

### 2.6.1 格式输出：printf()函数

相信学习过 C 语言的读者们都知道，printf 函数并非 OpenCV 中的函数，而是标准的 C 语言函数，包含在 stdio.h 之中。只不过 OpenCV 对其也有包含，我们只需包含头文件如 opencv.hpp，就可以使用它。

printf 函数是我们经常会用到的格式输出函数，其关键字最末一个字母 f 即为“格式”(format)之意。其功能是按用户指定的格式，把指定的数据显示到窗口中。printf 函数调用的一般形式如下：

```
int printf(const char *format, ...);
```

即：

```
int printf("格式控制字符串", 输出表列);
```

`printf` 函数的一个比较有特殊的用法是“格式字符串”，其用于指定输出格式，可由格式字符串和非格式字符串两种组成。格式字符串是以%开头的字符串，在%后面跟有各种格式字符，以说明输出数据的类型、形式、长度、小数位数等。如表 2.4 所示。

表 2.4 `printf` “格式字符串”候选字符

格式字符串	作用
%d	将整数转成十进制
%f	将整数转成浮点数
%u	十进制无符号整数
%o	将整数转成八进制
%c	将整数转成对应的 ASCII 字符
%s	将整数转成字符串
%x	整数转成小写十六进制
%X	整数转成大写十六进制
%p	输出地址符
%%	输出百分比符号，不进行转换

而上述表中没有，或不带%的“非格式字符”，则以原样输出，且格式字符串和各输出项在数量和类型上应该一一对应。

除了格式字符串，`printf` 还有一些特殊规定的字符，用法如表 2.5 所示。

表 2.5 `printf` 中特殊规定的字符

规定字符	作用
\n	换行操作
\f	清屏并换页
\r	回车
\t	Tab 符
\xhh	用 16 进表示的 ASCII 码，其中每个 h 可以用 0~f 中的一个代替

## 2.6.2 示例程序：`printf` 函数的用法示例

以上讲解的内容已经足够应付本书将要讲解的代码里关于 `printf` 的使用了，现在，让我们看一个示例，以结束 `printf` 函数的讲解，向下一节进发。

```
#include <opencv2/opencv.hpp>
using namespace cv;

void main()
{
    int a=66,b=68;
    printf("\n\t%d %d\n",a,b); //输出十进制整型
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

printf("\n\t%06d,%06d\n",a,b); //输出 6 位十进制整型
printf("\n\t%c,%c\n",a,b); //按字符型输出
printf("\n\t结果为: a=%d,b=%d",a,b); //可以配合其他内容一同输出
getchar(); //等待读入任意字符而结束, 在此用于保持窗口显示, 直到任意按键按下
}

```

运行结果如图 2.33 所示。

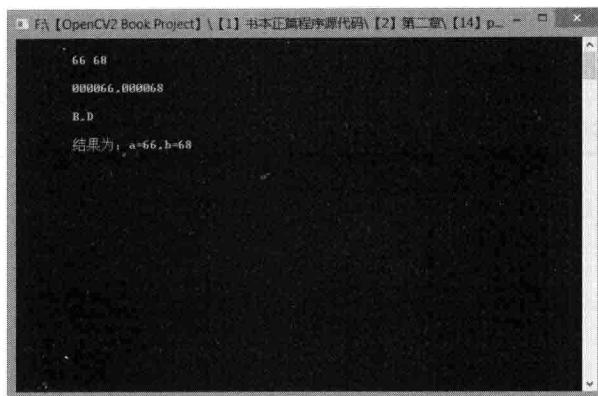


图 2.33 printf 输出结果

## 2.7 智能显示当前使用的 OpenCV 版本

如果大家已经查看了书本配套示例代码中的第 8 到第 13 的程序源码或者是后面章节中稍微复杂一点的示例程序，就会发现它们能根据你当前使用的 OpenCV 版本智能检测并输出相应的版本号。其实现非常简单，原理是 OpenCV 为我们提供了一个名为“CV\_VERSION”的用于标识当前 OpenCV 版本的宏。于是，我们使用 cout 或者 printf 等文字输出函数以此宏作为输出对象，可以智能地显示当前的 OpenCV 版本号。

可以书写如下代码进行 OpenCV 版本的智能检测，并将版本号输出到控制台窗口中。

```
printf("\t当前使用的 OpenCV 版本为 OpenCV " CV_VERSION );
```

## 2.8 本章小结

在本章中，我们学习了很多非常实用的知识，使大家可以对 OpenCV 有一个宏观的认知，也给大家提供了一些脱离书本依靠官方文档自学的思路，比如对 OpenCV 官方例程的引导与赏析，和如何编译 OpenCV 总计四十四万多行的源代码。此外，还有一些命名规范的约定和相关 C 语言基础函数的复习。本章内容授之以鱼，更授之以渔，相信通过本章内容的学习，可以对 OpenCV 有一个更加全面的认识。

### 本章示例程序清单

示例程序序号	程序说明	对应章节
8	官方例程引导、赏析之彩色目标跟踪：Camshift	2.1.1
9	官方例程引导、赏析之光流：optical flow	2.1.2

续表

示例程序序号	程序说明	对应章节
10	官方例程引导、赏析之点追踪：lkdemo	2.1.3
11	官方例程引导、赏析之人脸识别：objectDetection	2.1.4
12	官方例程引导、赏析之支持向量机：支持向量机引导	2.1.5
13	官方例程引导、赏析之支持向量机：处理线性不可分数据	2.1.5
14	printf 函数的用法示例	2.6.2

# 第 3 章

# HighGUI 图形用户界面初步

## 导读

---

在 OpenCV 架构分析一节中我们讲过，HighGUI 模块为高层 GUI 图形用户界面模块，包含媒体的输入输出、视频捕捉、图像和视频的编码解码、图形交互界面的接口等内容。而在 1.5 节中讲到的 VideoCapture 视频类，就是出自此 HighGUI 模块。本章旨在为大家展开讲解 OpenCV 中最常用到的一些交互操作，包括图像的载入、显示和输出，为程序添加滑动条，以及鼠标操作等常用内容。

本章中，你将学到：

- 图像的载入、显示和输出到文件的详细分析
- 滑动条（Trackbar）的创建和使用
- OpenCV 中的鼠标操作

## 3.1 图像的载入、显示和输出到文件

学习过以往版本 OpenCV 的读者应该都清楚,对于 OpenCV1.0 时代的基于 C 语言接口而建的图像存储格式 `IplImage*`, 如果在退出前忘记 `release` 掉的话, 会造成内存泄露, 而且用起来十分繁琐。我们在 `debug` 程序的时候, 往往很大部分时间会去纠结手动释放内存相关的问题。虽然对于小型的程序来说, 手动管理内存不是什么难题, 但一旦开发的项目日益庞大, 代码量达到一定的规模, 我们便会开始越来越多地纠缠于内存管理的问题, 而不能把全部精力用于解决核心开发目标。因为不合适的图像存储数据结构而疲于维护日益庞大的项目, 就有些舍本逐末的感觉了。

自踏入 2.0 版本的时代以来, OpenCV 采用了 `Mat` 类作为数据结构进行图像存取。这一改进使 OpenCV 变得和几乎零门槛入门的 Matlab 一样, 很容易上手和用于实际开发。新版 OpenCV 中甚至有些函数名称都和 Matlab 中的一样, 比如大家所熟知的 `imread`、`imwrite`、`imshow` 等函数。这对于广大图像处理和计算机视觉领域的研究者们来说, 的确是一件可喜可贺的事情。而这一小节中, 我们主要来详细讲解 OpenCV2、OpenCV3 入门最基本的问题, 即图像的载入、显示和输出。

### 3.1.1 OpenCV 的命名空间

OpenCV 中的 C++类和函数都是定义在命名空间 `cv` 之内的, 有两种方法可以访问: 第一种, 是在代码开头的适当位置加上 `usingnamespace cv;`; 这句代码, 规定程序位于此命名空间之内; 另外一种, 是在使用 OpenCV 的每一个类和函数时, 都加入 `cv::`命名空间。不过这种情况会很繁琐, 每用一个 OpenCV 的类或者函数, 都要多敲四下键盘写出 `cv::`。所以, 推荐大家在代码开头的适当位置, 加上 `using namespace cv;`这句。

比如在写简单的 OpenCV 程序的时候, 以下三句可以作为标配:

```
#include <opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
using namespace cv;
```

### 3.1.2 Mat 类简析

`Mat` 类是用于保存图像以及其他矩阵数据的数据结构, 默认情况下其尺寸为 0。我们也可以指定其初始尺寸, 比如定义一个 `Mat` 类对象, 就要写 `cv::Mat pic(320,640,cv::Scalar(100));`

`Mat` 类型作为 OpenCV2、OpenCV3 新纪元的重要代表, 在稍后的章节中, 笔者会花长篇幅详细讲解它, 现在我们只要理解它是对应于 OpenCV1.0 时代的 `IplImage`, 主要用来存放图像的数据结构就行了。对于本节, 我们需要用到关于 `Mat` 的其实就简单的这样一句代码:

```
Mat srcImage= imread("dota.jpg");
```

这表示从工程目录下把一幅名为 dota.jpg 的 jpg 类型的图像载入到 Mat 类型的 srcImage 变量中。对于这里的 imread 函数，用于将图片读入 Mat 类型中，会在下文进行详细剖析。而 Mat 类，也会在第 4 章中进行更加全面的讲解。

### 3.1.3 图像的载入与显示概述

在新版本的 OpenCV2 中，最简单的图像载入和显示只需要两句代码，非常便捷。这两句代码分别对应了两个函数，它们分别是 imread() 以及 imshow()。

### 3.1.4 图像的载入：imread() 函数

首先来看 imread 函数，其用于读取文件中的图片到 OpenCV 中。可以在 OpenCV 官方文档中查到它的原型，如下。

```
Mat imread(const string& filename, int flags=1 );
```

(1) 第一个参数，const string&类型的 filename，填我们需要载入的图片路径名。在 Windows 操作系统下，OpenCV 的 imread 函数支持如下类型的图像载入。

- Windows 位图：\*.bmp, \*.dib
- JPEG 文件：\*.jpeg, \*.jpg, \*.jpe
- JPEG 2000 文件：\*.jp2
- PNG 图片：\*.png
- 便携文件格式：\*.pbm, \*.pgm, \*.ppm
- Sun rasters 光栅文件：\*.sr, \*.ras
- TIFF 文件：\*.tiff, \*.tif

(2) 第二个参数，int 类型的 flags，为载入标识，它指定一个加载图像的颜色类型。可以看到它自带默认值 1，所以有时候这个参数在调用时可以忽略。在看了下面的讲解之后，我们就会发现，如果在调用时忽略这个参数，就表示载入三通道的彩色图像。这个参数可以在 OpenCV 中标识图像格式的枚举体中取值。通过转到定义，我们可以在 higui\_c.h 中发现这个枚举的定义是这样的：

```
enum
{
    /* 8bit, color or not */
    CV_LOAD_IMAGE_UNCHANGED = -1,
    /* 8bit, gray */
    CV_LOAD_IMAGE_GRAYSCALE = 0,
    /* ?, color */
    CV_LOAD_IMAGE_COLOR      = 1,
    /* any depth, ? */
    CV_LOAD_IMAGE_ANYDEPTH   = 2,
    /* ?, any color */
    CV_LOAD_IMAGEANYCOLOR    = 4
};
```

对常用标识符相应的解释：

- CV\_LOAD\_IMAGE\_UNCHANGED——等价取值为-1，这个标识在新版本中已被废置，忽略。
- CV\_LOAD\_IMAGE\_GRAYSCALE——等价取值为0，如果取这个标识的话，始终将图像转换成灰度再返回。
- CV\_LOAD\_IMAGE\_COLOR——等价取值为1，如果取这个标识，总是转换图像到彩色再返回。
- CV\_LOAD\_IMAGE\_ANYDEPTH——等价取值为2，如果取这个标识，且载入的图像的深度为16位或者32位，就返回对应深度的图像，否则，就转换为8位图像再返回。

需要说明的是，如果输入有冲突的标志，将采用较小的数字值。比如CV\_LOAD\_IMAGE\_COLOR | CV\_LOAD\_IMAGE\_ANYCOLOR 将载入三通道图。而如果想要载入最真实无损的源图像，可以选择 CV\_LOAD\_IMAGE\_ANYDEPTH | CV\_LOAD\_IMAGE\_ANYCOLOR。



OpenCV3 中此处列举的 CV\_LOAD\_IMAGE\_UNCHANGED、CV\_LOAD\_IMAGE\_GRAYSCALE 、 CV\_LOAD\_IMAGE\_COLOR 、 CV\_LOAD\_IMAGE\_ANYDEPTH、 CV\_LOAD\_IMAGE\_ANYCOLOR 等宏已经失效，但在官方文档和源代码中暂时没有发现新的可以替换它们的宏，所以请暂时用与其等价的-1、0、1、2、4 代替。相信后续 OpenCV 版本中一定会进行修复。

因为 flags 是 int 型的变量，若我们不在这个枚举体中取固定的值，可以这样进行：

- flags >0 返回一个3通道的彩色图像；
- flags =0 返回灰度图像；
- flags <0 返回包含 Alpha 通道的加载图像。



输出的图像默认情况下不返回 Alpha 通道。如果想要载入 Alpha 通道，这里就需要取负值。比如，将 flags 取 1999 也是可以的，和取 1 的效果一样，它们同样表示返回一个3通道的彩色图像。

另外，需要额外注意，若以彩色模式载入图像，解码后的图像会以 BGR 的通道顺序进行存储，即蓝、绿、红的顺序，而不是通常的 RGB 的顺序。

经过上面详细的讲解，我们一起看几个载入示例，以便多方位地掌握 imread 函数的用法。

```
Mat image0=imread("1.jpg",2 | 4); //载入无损的源图像
Mat image1=imread("1.jpg",0); //载入灰度图
Mat image2=imread("1.jpg",199); //载入3通道的彩色图像
```

### 3.1.5 图像的显示：imshow()函数

imshow()函数用于在指定的窗口中显示一幅图像，函数原型如下。

```
void imshow(const string& winname, InputArray mat);
```

- 第一个参数: const string&类型的 winname, 填需要显示的窗口标识名称。
- 第二个参数: InputArray 类型的 mat, 填需要显示的图像。

imshow 函数用于在指定的窗口中显示图像。如果窗口是用 CV\_WINDOW\_AUTOSIZE (默认值) 标志创建的, 那么显示图像原始大小。否则, 将图像进行缩放以适合窗口。而 imshow 函数缩放图像, 取决于图像的深度, 具体如下。

- 如果载入的图像是 8 位无符号类型(8-bit unsigned), 就显示图像本来的样子。
- 如果图像是 16 位无符号类型(16-bit unsigned)或 32 位整型(32-bit integer), 便用像素值除以 256。也就是说, 值的范围是[0,255 x 256]映射到[0, 255]。
- 如果图像是 32 位浮点型 (32-bit floating-point), 像素值便要乘以 255。也就是说, 该值的范围是[0, 1]映射到[0, 255]。

还有一点, 在窗口创建的时候, 如果设定了支持 OpenGL(WINDOW\_OPENGL), 那么 imshow 还支持 ogl::Buffer、ogl::Texture2D 以及 gpu::GpuMat 作为输入。

关于 imwrite 和 imshow 函数最精简的示例程序, 可以参考 1.3.8 节“最终的测试”或 1.4.1 节“第一个程序: 图像显示”中的代码。

### 3.1.6 关于 InputArray 类型

对于这里的 InputArray 类型, 通过对 InputArray 转到定义, 我们可以在 core.hpp 中查到一个 typedef 声明, 如下:

```
typedef const _InputArray& InputArray;
```

这其实一个类型声明引用, 就是说 \_InputArray 和 InputArray 是一个意思, 因此我们就来做最后一步: 对 \_InputArray 进行转到定义, 可以在 core.hpp 头文件中发现 InputArray 的真身。InputArray 的源代码略显冗长, 不在这里贴出, 若读者自己去实践并通过转到定义的方法找到 InputArray 的真身, 便可以看到 \_InputArray 类的里面首先定义了一个枚举, 然后是各类的模板类型和一些方法。而很多时候, 遇到函数原型中的 InputArray/OutputArray 类型, 我们把它简单地当做 Mat 类型即可。

### 3.1.7 创建窗口: namedWindow()函数

namedWindow 函数用于创建一个窗口。若是简单地进行图片显示, 可以略去 namedWindow 函数的调用, 即先调用 imread 读入图片, 然后用 imshow 直接指定出窗口名进行显示即可。但需要在显示窗口之前就用到窗口名时, 比如我们后面会马上讲到滑动条的使用, 要指定滑动条依附到某个窗口上, 就需要 namedWindow 函数先创建出窗口, 显式地规定窗口名称了。

namedWindow 的函数原型如下:

```
void namedWindow(const string& winname,int flags=WINDOW_AUTOSIZE );
```

- (1) 第一个参数, const string&型的 name, 填写被用作窗口的标识符的窗口名称。

(2) 第二个参数, int 类型的 flags, 窗口的标识, 可以填如下几种值。

- WINDOW\_NORMAL, 设置这个值, 用户可以改变窗口的大小(没有限制)。OpenCV2 中它还可以写为 CV\_WINDOW\_NORMAL。
- WINDOW\_AUTOSIZE, 设置这个值, 窗口大小会自动调整以适应所显示的图像, 并且用户不能手动改变窗口大小。OpenCV2 中它还可以写为 CV\_WINDOW\_AUTOSIZE。
- WINDOW\_OPENGL, 设置这个值, 窗口创建的时候会支持 OpenGL。OpenCV2 中它还可以写为 CV\_WINDOW\_OPENGL。

首先需要注意的是, namedWindow 函数有默认值 WINDOW\_AUTOSIZE, 所以, 一般情况下, 这个函数我们填一个变量就行了。namedWindow 函数的作用是通过指定的名字, 创建一个可以作为图像和进度条的容器窗口。如果具有相同名称的窗口已经存在, 则函数不做任何事情。我们可以调用 destroyWindow()或者 destroyAllWindows()函数来关闭窗口, 并取消之前分配的与窗口相关的所有内存空间。

但是事实上, 对于代码量不大的简单程序来说, 我们完全没有必要手动调用上述的 destroyWindow()或者 destroyAllWindows()函数, 因为在退出时, 所有的资源和应用程序的窗口会被操作系统自动关闭。

### 3.1.8 输出图像到文件: imwrite()函数

在 OpenCV 中, 输出图像到文件一般采用 imwrite 函数, 它的声明如下。

```
bool imwrite(const string& filename, InputArray img, const vector<int>& params=vector<int>());
```

(1) 第一个参数, const string&类型的 filename, 填需要写入的文件名。注意要带上后缀, 如“123.jpg”。

(2) 第二个参数, InputArray 类型的 img, 一般填一个 Mat 类型的图像数据。

(3) 第三个参数, const vector<int>&类型的 params, 表示为特定格式保存的参数编码。它有默认值 vector<int>(), 所以一般情况下不需要填写。而如果要填写的话, 有下面这些需要了解的地方:

- 对于 JPEG 格式的图片, 这个参数表示从 0 到 100 的图片质量 (CV\_IMWRITE\_JPEG\_QUALITY), 默认值是 95。
- 对于 PNG 格式的图片, 这个参数表示压缩级别 (CV\_IMWRITE\_PNG\_COMPRESSION) 从 0 到 9。较高的值意味着更小的尺寸和更长的压缩时间, 默认值是 3。
- 对于 PPM, PGM, 或 PBM 格式的图片, 这个参数表示一个二进制格式标志 (CV\_IMWRITE\_PXM\_BINARY), 取值为 0 或 1, 默认值是 1。

imwrite 函数用于将图像保存到指定的文件。图像格式是基于文件扩展名的, 可保存的扩展名和 imread 中可以读取的图像扩展名一致。

下面是一个示例程序，讲解 imwrite 函数的用法——在 OpenCV 中生成一幅 png 图片，并写入到当前工程目录下。

```
#include<opencv2/opencv.hpp>
#include <vector>

using namespace cv;
using namespace std;

void createAlphaMat(Mat &mat)
{
    for(int i = 0; i < mat.rows; ++i) {
        for(int j = 0; j < mat.cols; ++j) {
            Vec4b&rgba = mat.at<Vec4b>(i, j);
            rgba[0] = UCHAR_MAX;
            rgba[1] = saturate_cast<uchar>((float (mat.cols - j)) /
                ((float)mat.cols) *UCHAR_MAX);
            rgba[2] = saturate_cast<uchar>((float (mat.rows - i)) /
                ((float)mat.rows) *UCHAR_MAX);
            rgba[3] = saturate_cast<uchar>(0.5 * (rgba[1] + rgba[2]));
        }
    }
}

int main()
{
    // 创建带 Alpha 通道的 Mat
    Mat mat(480, 640, CV_8UC4);
    createAlphaMat(mat);

    vector<int>compression_params;
    // 此句代码的 OpenCV2 版为：
    // compression_params.push_back(CV_IMWRITE_PNG_COMPRESSION);
    // 此句代码的 OpenCV3 版为：
    compression_params.push_back(IMWRITE_PNG_COMPRESSION);
    compression_params.push_back(9);

    try{
        imwrite("透明 Alpha 值图.png", mat, compression_params);
        imshow("生成的 PNG 图", mat);
        fprintf(stdout, "PNG 图片文件的 alpha 数据保存完毕~\n可以在工程目录下
查看由 imwrite 函数生成的图片\n");
        waitKey(0);
    }
    catch(runtime_error& ex) {
        fprintf(stderr, "图像转换成 PNG 格式发生错误: %s\n", ex.what());
        return 1;
    }

    return 0;
}
```

程序运行截图如图 3.1 所示。运行完毕，我们可以在工程目录下发现一张生成的名为“透明 Alpha 值图.png”的图片文件。

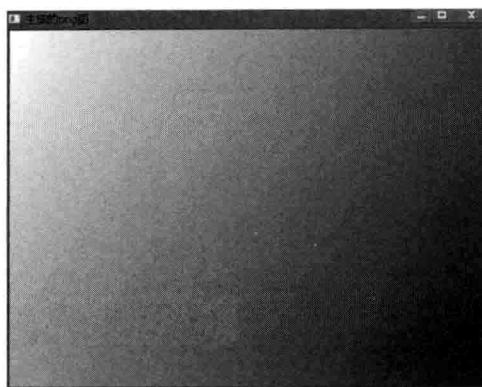


图 3.1 由 imwrite 生成的图

### 3.1.9 综合示例程序：图像的载入、显示与输出

本小节将给出一个综合示例，演示如何载入图像，进行简单的图像混合，显示图像，并且输出混合后的图像到 jpg 格式的文件中。

出于注重演示效果的原因，程序中图像混合的具体细节我们放到稍后的篇幅中再讲，现在先给大家看看混合的效果和源码。以下就是本节的综合示例程序，经过详细注释的代码非常简单明了。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
-----
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;

int main()
{
    -----【一、图像的载入和显示】-----
    //  描述：以下三行代码用于完成图像的载入和显示
    //

    Mat girl=imread("girl.jpg"); //载入图像到Mat
    namedWindow("【1】动漫图"); //创建一个名为 "【1】动漫图"的窗口
    imshow("【1】动漫图",girl); //显示名为 "【1】动漫图"的窗口

    -----【二、初级图像混合】-----
    //  描述：二、初级图像混合
    //
    //载入图片
    Mat image= imread("dota.jpg",199);
    Mat logo= imread("dota_logo.jpg");
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
//载入后先显示
namedWindow("【2】原画图");
imshow("【2】原画图",image);

namedWindow("【3】logo图");
imshow("【3】logo图",logo);

//定义一个Mat类型，用于存放，图像的ROI
Mat imageROI;
//方法一
imageROI= image(Rect(800,350,logo.cols,logo.rows));
//方法二
//imageROI=
image(Range(350,350+logo.rows),Range(800,800+logo.cols));

//将logo加到原图上
addWeighted(imageROI,0.5,logo,0.3,0.,imageROI);

//显示结果
namedWindow("【4】原画+logo图");
imshow("【4】原画+logo图",image);

-----【三、图像的输出】-----
//描述：将一个Mat图像输出到图像文件
//-----输出一张jpg图片到工程目录下
imwrite("由imwrite生成的图片.jpg",image);

waitKey();

return 0;
}
```

运行这个程序，会弹出4个在OpenCV中创建的窗口。

下面是运行截图。首先是图像载入和显示的演示，我们载入了一张动漫人物图，如图3.2所示。



图3.2 动漫显示图

接着是载入一张 dota2 原画（图 3.3）和 dota2 的 logo（图 3.4），为图像融合做准备。



图 3.3 dota2 原画图窗口



图 3.4 logo 图

最终，经过处理，得到 dota2 原画+logo 的融合，并输出一张名为“由 imwrite 生成的图片.jpg”的图片到工程目录下。如图 3.5 所示。



图 3.5 融合后的效果图

## 3.2 滑动条的创建和使用

滑动条（Trackbar）是 OpenCV 动态调节参数特别好用的一种工具，它依附于窗口而存在。

由于 OpenCV 中并没有实现按钮的功能，所以很多时候，我们还可以用仅含 0-1 的滑动条来实现按钮的按下、弹起效果。

### 3.2.1 创建滑动条：createTrackbar()函数

createTrackbar 函数用于创建一个可以调整数值的滑动条（常常也被称作轨迹条），并将滑动条附加到指定的窗口上，使用起来很方便。需要记住，它往往会和一个回调函数配合起来使用。先看下它的函数原型，如下。

```
C++: int createTrackbar(const string& trackbarname, const string& winname,
```

```
int* value, int count, TrackbarCallback onChange=0, void* userdata=0);
```

- 第一个参数，const string&类型的 trackbarname，轨迹条的名字，用来代表我们创建的轨迹条。
- 第二个参数，const string&类型的 winname，窗口的名字，表示这个轨迹条会依附到哪个窗口上，即对应 namedWindow()创建窗口时填的某一个窗口名。
- 第三个参数，int\* 类型的 value，一个指向整型的指针，表示滑块的位置。在创建时，滑块的初始位置就是该变量当前的值。
- 第四个参数，int 类型的 count，表示滑块可以达到的最大位置的值。滑块最小位置的值始终为 0。
- 第五个参数，TrackbarCallback 类型的 onChange，它有默认值 0。这是一个指向回调函数的指针，每次滑块位置改变时，这个函数都会进行回调。并且这个函数的原型必须为 void XXXX(int, void\*);，其中第一个参数是轨迹条的位置，第二个参数是用户数据（看下面的第六个参数）。如果回调是 NULL 指针，则表示没有回调函数的调用，仅第三个参数 value 有变化。
- 第六个参数，void\*类型的 userdata，也有默认值 0。这个参数是用户传给回调函数的数据，用来处理轨迹条事件。如果使用的第三个参数 value 实参是全局变量的话，完全可以不去管这个 userdata 参数。

createTrackbar 函数为我们创建了一个具有特定名称和范围的轨迹条（Trackbar，或者说是滑块范围控制工具），指定一个和轨迹条位置同步的变量，而且要指定回调函数 onChange（第五个参数），在轨迹条位置改变的时候来调用这个回调函数，并且，创建的轨迹条显示在指定的 winname（第二个参数）所代表的窗口上。

至于回调函数，就是一个通过函数指针调用的函数。如果我们把函数的指针（地址）作为参数传递给另一个函数，当这个指针被用来调用其所指向的函数时，就称其为回调函数。回调函数不由该函数的实现方直接调用，而是在特定的事件

或条件发生时由另外的一方调用，用于对该事件或条件进行响应。

在函数讲解之后，给大家一个 createTrackbar 函数使用的小例子作为参照。

```
//创建轨迹条
createTrackbar("对比度：", "【效果图窗口】", &g_nContrastValue,
300, on_Change); // g_nContrastValue 为全局的整型变量, on_Change 为
回调函数的函数名(在 C/C++ 中, 函数名为指向函数地址的指针)
```

接着，我们一起来欣赏一个完整的使用示例，它演示了如何用轨迹条来控制两幅图像的 Alpha 混合。

```
#include <opencv2/opencv.hpp>
#include "opencv2/highgui/highgui.hpp"
using namespace cv;

#define WINDOW_NAME "【线性混合示例】"           //为窗口标题定义的宏

//-----【全局变量声明部分】-----
//      描述：全局变量声明
//-----[ on_Trackbar() 函数 ]-----
//      描述：响应滑动条的回调函数
//-----
void on_Trackbar( int, void* )
{
    //求出当前 alpha 值相对于最大值的比例
    g_dAlphaValue = (double) g_nAlphaValueSlider/g_nMaxAlphaValue ;
    //则 beta 值为 1 减去 alpha 值
    g_dBetaValue = ( 1.0 - g_dAlphaValue );

    //根据 alpha 和 beta 值进行线性混合
    addWeighted( g_srcImage1, g_dAlphaValue, g_srcImage2, g_dBetaValue,
0.0, g_dstImage );

    //显示效果图
    imshow( WINDOW_NAME, g_dstImage );
}

//-----【 main() 函数 】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
int main( int argc, char** argv )
{
    //加载图像（两图像的尺寸需相同）
    g_srcImage1 = imread("1.jpg");
    g_srcImage2 = imread("2.jpg");
    if( !g_srcImage1.data ) { printf("读取第一幅图片错误，请确定目录下是否有imread函数指定图片存在~!\n"); return -1; }
    if( !g_srcImage2.data ) { printf("读取第二幅图片错误，请确定目录下是否有imread函数指定图片存在~!\n"); return -1; }

    //设置滑动条初值为 70
    g_nAlphaValueSlider = 70;

    //创建窗体
    namedWindow(WINDOW_NAME, 1);

    //在创建的窗体中创建一个滑动条控件
    char TrackbarName[50];
    sprintf( TrackbarName, "透明值 %d", g_nMaxAlphaValue );

    createTrackbar( TrackbarName, WINDOW_NAME, &g_nAlphaValueSlider,
    g_nMaxAlphaValue, on_Trackbar );

    //结果在回调函数中显示
    on_Trackbar( g_nAlphaValueSlider, 0 );

    //按任意键退出
    waitKey(0);

    return 0;
}
```

运行此程序，我们可以通过调节滑动条的位置，来得到不同的混合效果。如图 3.6、图 3.7、图 3.8 所示。



图 3.6 透明值为 70 时的效果图



图 3.7 透明值为 100 时的效果图



图 3.8 透明值为 0 时的效果图

### 3.2.2 获取当前轨迹条的位置：getTrackbarPos()函数

本小节介绍一个配合 createTrackbar 使用的函数——getTrackbarPos(), 它用于获取当前轨迹条的位置。

下面这个函数用于获取当前轨迹条的位置并返回。

```
C++: int getTrackbarPos(const string& trackbarname, const string& winname);
```

- 第一个参数，const string&类型的 trackbarname，表示轨迹条的名字。
- 第二个参数，const string&类型的 winname，表示轨迹条的父窗口的名称。

## 3.3 鼠标操作

OpenCV 中的鼠标操作和滑动条的消息映射方式很类似，都是通过一个中介函数配合一个回调函数来实现的。创建和指定滑动条回调函数的函数为 createTrackbar，而指定鼠标操作消息回调函数的函数为 SetMouseCallback。下面一起来了解一下它。

SetMouseCallback 函数的作用是为指定的窗口设置鼠标回调函数，原型如下。

```
C++: void setMouseCallback(const string& winname, MouseCallback onMouse, void* userdata=0)
```

- 第一个参数，const string&类型的 winname，窗口的名字。
- 第二个参数，MouseCallback 类型的 onMouse，指定窗口里每次鼠标时间发

## Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

生的时候，被调用的函数指针。这个函数的原型的大概形式为 void Foo(int event, int x, int y, int flags, void\* param)。其中 event 是 EVENT\_+ 变量之一，x 和 y 是鼠标指针在图像坐标系（需要注意，不是窗口坐标系）中的坐标值，flags 是 EVENT\_FLAG 的组合，param 是用户定义的传递到 SetMouseCallback 函数调用的参数。如 EVENT\_MOUSEMOVE 为鼠标移动消息、EVENT\_LBUTTONDOWN 为鼠标左键按下消息等。



在 OpenCV2 中，上述 “EVENT\_” 之前可以加上 “CV\_” 前缀。

- 第三个参数，void\*类型的 userdata，用户定义的传递到回调函数的参数，有默认值 0。

下面看一个详细注释的示例程序，在实战中了解此函数的用法以及如何在 OpenCV 中使用鼠标进行交互。

```
-----【头文件、命名空间包含部分】-----  
//      描述：包含程序所使用的头文件和命名空间  
-----  
#include <opencv2/opencv.hpp>  
using namespace cv;  
  
#define WINDOW_NAME "【程序窗口】"          //为窗口标题定义的宏  
  
-----【全局函数声明部分】-----  
//      描述：全局函数的声明  
-----  
void on_MouseHandle(int event, int x, int y, int flags, void* param);  
void DrawRectangle( cv::Mat& img, cv::Rect box );  
void ShowHelpText();  
  
-----【全局变量声明部分】-----  
//      描述：全局变量的声明  
-----  
Rect g_rectangle;  
bool g_bDrawingBox = false; //是否进行绘制  
RNG g_rng(12345);  
  
-----【main() 函数】-----  
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行  
-----  
int main( int argc, char** argv )  
{  
  
    //【1】准备参数  
    g_rectangle = Rect(-1,-1,0,0);  
    Mat srcImage(600, 800,CV_8UC3), tempImage;  
    srcImage.copyTo(tempImage);
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
g_rectangle = Rect(-1,-1,0,0);
srcImage = Scalar::all(0);

//【2】设置鼠标操作回调函数
namedWindow( WINDOW_NAME );
setMouseCallback(WINDOW_NAME, on_MouseHandle, (void*)&srcImage);

//【3】程序主循环，当进行绘制的标识符为真时，进行绘制
while(1)
{
    srcImage.copyTo(tempImage); //复制源图到临时变量
    if( g_bDrawingBox ) DrawRectangle( tempImage, g_rectangle ); //当进行绘制的标识符为真，则进行绘制
    imshow( WINDOW_NAME, tempImage );
    if( waitKey( 10 ) == 27 ) break; //按下 ESC 键，程序退出
}
return 0;
}

//-----【on_MouseHandle() 函数】-----
//      描述：鼠标回调函数，根据不同的鼠标事件进行不同的操作
//-----
void on_MouseHandle(int event, int x, int y, int flags, void* param)
{

    Mat& image = *(cv::Mat*) param;
    switch( event )
    {
        //鼠标移动消息
        case EVENT_MOUSEMOVE:
        {
            if( g_bDrawingBox ) //如果是否进行绘制的标识符为真，则记录下长和宽到 RECT 型变量中
            {
                g_rectangle.width = x-g_rectangle.x;
                g_rectangle.height = y-g_rectangle.y;
            }
        }
        break;

        //左键按下消息
        case EVENT_LBUTTONDOWN:
        {
            g_bDrawingBox = true;
            g_rectangle = Rect( x, y, 0, 0 ); //记录起始点
        }
        break;

        //左键抬起消息
        case EVENT_LBUTTONUP:
        {

```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
g_bDrawingBox = false; //置标识符为 false
//对宽和高小于 0 的处理
if( g_rectangle.width < 0 )
{
    g_rectangle.x += g_rectangle.width;
    g_rectangle.width *= -1;
}

if( g_rectangle.height < 0 )
{
    g_rectangle.y += g_rectangle.height;
    g_rectangle.height *= -1;
}
//调用函数进行绘制
DrawRectangle( image, g_rectangle );
}

break;

}

}

-----【 DrawRectangle() 函数】-----
//      描述：自定义的矩形绘制函数
-----
void DrawRectangle( cv::Mat& img, cv::Rect box )
{
    rectangle(img,box.tl(),box.br(),Scalar(g_rng.uniform(0,255),
g_rng.uniform(0,255), g_rng.uniform(0,255))); //随机颜色
}
```

首先一起看看程序运行效果。我们可以通过鼠标左键的按下和松开来在黑色的窗口中绘制出一个一个彩色的矩形。如图 3.9、图 3.10 所示。

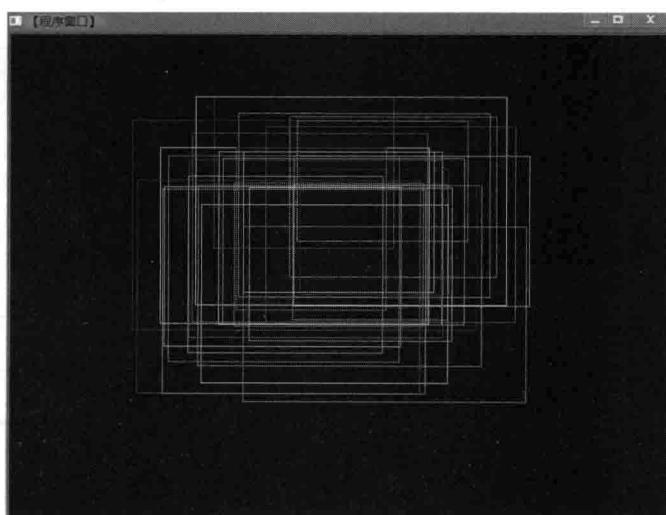


图 3.9 运行截图（1）

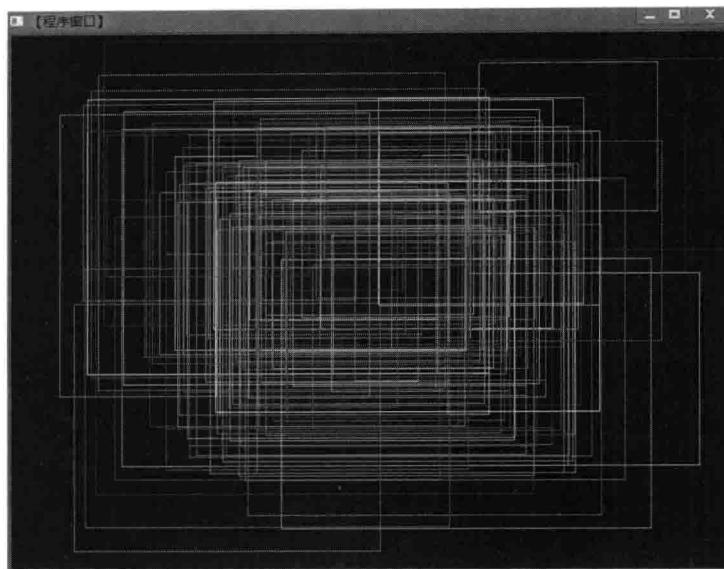


图 3.10 运行截图 (2)

上述示例程序中的 `on_MouseHandle` 就是我们的鼠标消息回调函数。其中用一个 `switch` 语句指出了各种类型的鼠标消息，如鼠标移动消息 `EVENT_MOUSEMOVE`、左键按下消息 `EVENT_LBUTTONDOWN`、左键抬起消息 `EVENT_LBUTTONUP`，并对其进行处理，以得到随机颜色的矩形绘制的功能。

### 3.4 本章小结

本章中，我们学习了 OpenCV 的高层 GUI 图形用户界面模块 `highgui` 中最重要的几个方面，分别是图像的载入、显示与输出图像到文件，以及如何使用滑动条如何进行鼠标操作。

#### 本章核心函数清单

函数名称	用途	讲解章节
<code>imread</code>	用于读取文件中的图片到 OpenCV 中	3.1.4
<code>imshow</code>	在指定的窗口中显示一幅图像	3.1.5
<code>namedWindow</code>	用于创建一个窗口	3.1.7
<code>imwrite</code>	输出图像到文件	3.1.8
<code>createTrackbar</code>	用于创建一个可以调整数值的轨迹条	3.2.1
<code>getTrackbarPos</code>	用于获取轨迹条的当前位置	3.2.2
<code>SetMouseCallback</code>	为指定的窗口设置鼠标回调函数	3.3

**本章示例程序清单**

示例程序序号	程序说明	对应章节
15	用 imwrite 函数生成 png 透明图	3.1.8
16	综合示例程序：图像的载入、显示与输出	3.1.9
17	为程序界面添加滑动条	3.2.1
18	鼠标操作	3.3

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

## 第二部分 初探 core 组件

OpenCV 中的 core 组件是核心功能模块，第二部分包含如下内容：

---

- 基本数据结构
- 动态数据结构
- 绘图函数
- 数组操作相关函数
- 辅助功能与系统函数和宏
- 与 OpenGL 的互操作

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

# 第 4 章

# OpenCV 数据结构与基本绘图

## 导读

---

本章中，你将学到：

- 基础图像容器 Mat 的用法
- OpenCV 中的多种格式化输出方法
- 常用的数据结构
- 基本绘图操作

## 4.1 基础图像容器 Mat

### 4.1.1 数字图像存储概述

我们可以通过各种各样的方法从现实世界获取到数字图像，如借助相机、扫描仪、计算机摄像头或磁共振成像等。通常由显示屏上看到的都是真实而漂亮的图像，但是这些图像在转化到我们的数字设备中时，记录的却是图像中的每个点的数值。

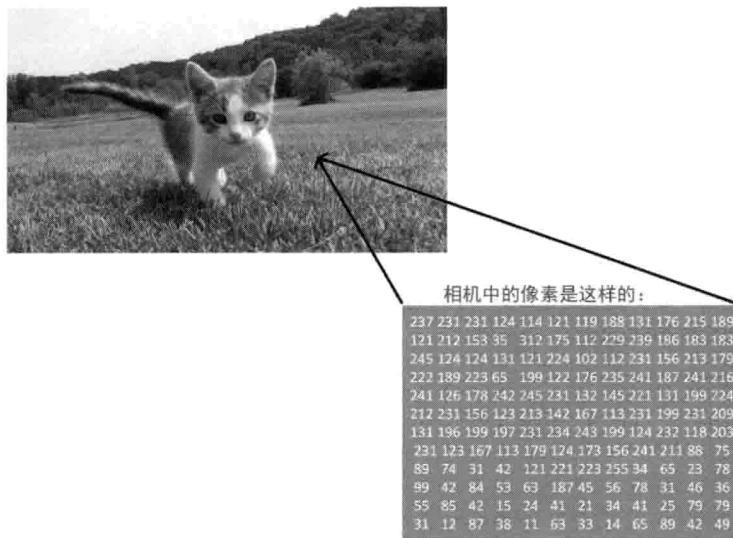


图 4.1 图像的存储图示

比如在图 4.1 中你可以看到草坪的颜色是一个包含众多强度值的像素点矩阵。可以说，矩阵就是图像在数码设备中的表现形式。OpenCV 作为一个计算机视觉库，其主要的工作是处理和操作并进一步了解这些形式和信息。因此，理解 OpenCV 是如何存储和处理图像是非常有必要的。那么，就让我们来进一步了解和学习。

### 4.1.2 Mat 结构的使用

自 2001 年以来，OpenCV 的函数库一直是基于 C 接口构建的，因此在最初的几个 OpenCV 版本中，一般使用名为 `IplImage` 的 C 语言结构体在内存中存储图像。时至今日，这仍出现在大多数的旧版教程和教学材料中，如最经典的 OpenCV 教程《Learning OpenCV》。

对于 OpenCV1.X 时代的基于 C 语言接口而建的图像存储格式 `IplImage*`，如果在退出前忘记 `release` 掉的话，就会照成内存泄露，而且用起来有些不便，我们在调试的时候，往往要花费很多时间在手动释放内存的问题上。虽然对于小型的程序来说，手动管理内存不是问题，但一旦需要书写和维护的代码越来越庞大，

我们便会开始越来越多地纠缠于内存管理的问题，而不是着力解决最终的开发目标。这，就有些舍本逐末的感觉了。

幸运的是，C++出现了，并且带来了类的概念，这使我们有了另外一个选择：自动的内存管理（非严格意义上的）。这对于广大图像处理领域的研究者来说，的确是一件可喜可贺的事情。也就是说，OpenCV 在 2.0 版本中引入了一个新的 C++ 接口，利用自动内存管理给出了解决问题的新方法。使用此方法，我们不需要再纠结在管理内存的问题，而且代码会变得干净而简洁。

但 C++ 接口唯一的不足是：当前许多嵌入式开发系统只支持 C 语言。所以，当开发目标不是仅能使用 C 语言作为开发语言时，便没有必要使用旧的 C 语言接口了，除非你真的很有自信。

从 OpenCV 踏入 2.0 时代，使用 Mat 类数据结构作为主打之后，OpenCV 变得越发像需要很少编程涵养的 Matlab 那样，上手很方便。甚至有些函数名称都和 Matlab 一样，比如大家所熟知的 imread、imwrite、imshow 等函数。

关于 Mat 类，首先我们要知道的是：

- (1) 不必再手动为其开辟空间。
- (2) 不必再在不需要时立即将空间释放。

这里指的是手动开辟空间并非必须，但它依旧是存在的——大多数 OpenCV 函数仍会手动地为输出数据开辟空间。当传递一个已经存在的 Mat 对象时，开辟好的矩阵空间会被重用。也就是说，我们每次都使用大小正好内存来完成任务。

总而言之，Mat 是一个类，由两个数据部分组成：矩阵头（包含矩阵尺寸、存储方法、存储地址等信息）和一个指向存储所有像素值的矩阵（根据所选存储方法的不同，矩阵可以是不同的维数）的指针。矩阵头的尺寸是常数值，但矩阵本身的尺寸会依图像的不同而不同，通常比矩阵头的尺寸大数个数量级。因此，当在程序中传递图像并创建副本时，大的开销是由矩阵造成的，而不是信息头。OpenCV 是一个图像处理库，囊括了大量的图像处理函数，为了解决问题通常要使用库中的多个函数，因此在函数中传递图像是常有的事。同时不要忘了我们正在讨论的是计算量很大的图像处理算法，因此，除非万不得已，不应该进行大图像的复制，因为这会降低程序的运行速度。

为了解决此问题，OpenCV 使用了引用计数机制。其思路是让每个 Mat 对象有自己的信息头，但共享同一个矩阵。这通过让矩阵指针指向同一地址而实现。而拷贝构造函数则只复制信息头和矩阵指针，而不复制矩阵。

来看下面这段代码。

```
Mat A, C; // 仅创建信息头部分
A = imread("1.jpg", CV_LOAD_IMAGE_COLOR); // 这里为矩阵开辟内存
Mat B(A); // 使用拷贝构造函数
C = A; // 赋值运算符
```

以上代码中的所有 Mat 对象最终都指向同一个也是唯一一个数据矩阵。虽然它们的信息头不同，但通过任何一个对象所做的改变也会影响其他对象。实际上，不同的对象只是访问相同数据的不同途径而已。这里还要提及一个比较棒的功能：我们可以创建只引用部分数据的信息头。比如想要创建一个感兴趣区域（ROI），只需要创建包含边界信息的信息头：

```
Mat D (A, Rect(10, 10, 100, 100)); // 使用矩形界定  
Mat E = A(Range::all(), Range(1,3)); // 用行和列来界定
```

现在你也许会问：如果矩阵属于多个 Mat 对象，那么当不再需要它时，谁来负责清理呢？

简单的回答是：最后一个使用它的对象。通过引用计数机制来实现。我们无论什么时候复制一个 Mat 对象的信息头，都会增加矩阵的引用次数。反之，当一个头被释放之后，这个计数被减一；当计数值为零，矩阵会被清理。但某些时候你仍会想复制矩阵本身（不只是信息头和矩阵指针），这时可以使用函数 clone() 或者 copyTo()。

```
Mat F = A.clone();  
Mat G;  
A.copyTo(G);
```

现在改变 F 或者 G 就不会影响 Mat 信息头所指向的矩阵。本小节可总结为如下 4 个要点。

- OpenCV 函数中输出图像的内存分配是自动完成的（如果不特别指定的话）。
- 使用 OpenCV 的 C++ 接口时不需要考虑内存释放问题。
- 赋值运算符和拷贝构造函数（构造函数）只复制信息头。
- 使用函数 clone() 或者 copyTo() 来复制一幅图像的矩阵。

### 4.1.3 像素值的存储方法

本节我们将讲解如何存储像素值。存储像素值需要指定颜色空间和数据类型。其中，颜色空间是指针对一个给定的颜色，如何组合颜色元素以对其编码。最简单的颜色空间要属灰度级空间，只处理黑色和白色，对它们进行组合便可以产生不同程度的灰色。

对于彩色方式则有更多种类的颜色空间，但不论哪种方式都是把颜色分成三个或者四个基元素，通过组合基元素可以产生所有的颜色。RGB 颜色空间是最常用的一种颜色空间，这归功于它也是人眼内部构成颜色的方式。它的基色是红色、绿色和蓝色，有时为了表示透明颜色也会加入第四个元素 alpha (A)。

颜色系统有很多，它们各有优势，具体如下。

- RGB 是最常见的，这是因为人眼采用相似的工作机制，它也被显示设备所采用
- HSV 和 HLS 把颜色分解成色调、饱和度和亮度/明度。这是描述颜色更自然的方式，比如可以通过抛弃最后一个元素，使算法对输入图像的光照条

件不敏感

- YCrCb 在 JPEG 图像格式中广泛使用
- CIE L\*a\*b\*是一种在感知上均匀的颜色空间，它适合用来度量两个颜色之间的距离

每个组成元素都有其自己的定义域，而定义域取决于其数据类型，如何存储一个元素决定了我们在其定义域上能够控制的精度。最小的数据类型是 `char`，占一个字节或者 8 位，可以是有符号型（0 到 255 之间）或无符号型（-127 到+127 之间）。尽管使用三个 `char` 型元素已经可以表示 1600 万种可能的颜色（使用 RGB 颜色空间），但若使用 `float`（4 字节，32 位）或 `double`（8 字节，64 位）则能给出更加精细的颜色分辨能力。但同时也要切记，增加元素的尺寸也会增加图像所占的内存空间。

#### 4.1.4 显式创建 Mat 对象的七种方法

在之前的章节中，我们已经讲解了如何使用函数 `imwrite()` 函数将一个矩阵写入图像文件中。但是作为 `debug`，更加方便的方式是看实际值，我们可以通过 `Mat` 的运算符“`<<`”来实现。但要记住，`Mat` 的运算符“`<<`”只对二维矩阵有效。

`Mat` 不但是一个非常有用的图像容器类，同时也是个通用的矩阵类，我们也可以用它来创建和操作多维矩阵。

创建一个 `Mat` 对象有多种方法，列举如下。

##### 1. 【方法一】使用 `Mat()` 构造函数

最常用的方法是直接使用 `Mat()` 构造函数，这种方法简单明了，示范代码如下。

```
Mat M(2, 2, CV_8UC3, Scalar(0, 0, 255));
cout << "M = " << endl << " " << M << endl << endl;
```

上述代码的运行结果如图 4.2 所示。

```
M =
[0, 0, 255, 0, 0, 255;
 0, 0, 255, 0, 0, 255]
```

图 4.2 方法一示例代码运行结果

对于二维多通道图像，首先要定义其尺寸，即行数和列数。然后，需要指定存储元素的数据类型以及每个矩阵点的通道数。为此，依据下面的规则有多种定义：

`CV_[The number of bits per item] [Signed or Unsigned] [Type Prefix]C [The channel number]`

即：

`CV_[位数][带符号与否][类型前缀]C[通道数]`

比如 `CV_8UC3` 表示使用 8 位的 `unsigned char` 型，每个像素由三个元素组成三通道。而预先定义的通道数可以多达四个。另外，`Scalar` 是个 `short` 型的向量，

能使用指定的定制化值来初始化矩阵，它还可以用于表示颜色，后文有详细讲解。当然，若需要更多通道数，可以使用大写的宏并把通道数放在小括号中，如方法二中的代码所示。

## 2. 【方法二】在 C\C++ 中通过构造函数进行初始化

这种方法为在 C\C++ 中通过构造函数进行初始化，示范代码如下。

```
int sz[3] = {2, 2, 2};
Mat L(3, sz, CV_8UC, Scalar::all(0));
```

上面的例子演示了如何创建一个超过两维的矩阵：指定维数，然后传递一个指向一个数组的指针，这个数组包含每个维度的尺寸；后续的两个参数与方法一中的相同。

## 3. 【方法三】为已存在的 IplImage 指针创建信息头

方法三是为已存在的 IplImage 指针创建信息头，示范代码如下。

```
IplImage* img = cvLoadImage("1.jpg", 1);
Mat mtx(img); // 转换 IplImage*-> Mat
```

## 4. 【方法四】利用 Create() 函数

方法四是利用 Mat 类中的 Create() 成员函数进行 Mat 类的初始化操作，示范代码如下。

```
M.create(4, 4, CV_8UC(2));
cout << "M = " << endl << " " << M << endl << endl;
```

上述代码的运行结果如图 4.3 所示。

```
M =
[205, 205, 205, 205, 205, 205, 205, 205;
 205, 205, 205, 205, 205, 205, 205, 205;
 205, 205, 205, 205, 205, 205, 205, 205;
 205, 205, 205, 205, 205, 205, 205, 205]
```

图 4.3 方法四示例代码运行结果

需要注意的是，此创建方法不能为矩阵设初值，只是在改变尺寸时重新为矩阵数据开辟内存而已。

## 5. 【方法五】采用 Matlab 式的初始化方式

方法五采用 Matlab 形式的初始化方式：zeros()，ones()，eyes()。使用以下方式指定尺寸和数据类型：

```
Mat E = Mat::eye(4, 4, CV_64F);
cout << "E = " << endl << " " << E << endl << endl;

Mat O = Mat::ones(2, 2, CV_32F);
cout << "O = " << endl << " " << O << endl << endl;

Mat Z = Mat::zeros(3, 3, CV_8UC1);
cout << "Z = " << endl << " " << Z << endl << endl;
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

上述代码的运行结果如图 4.4 所示。

```

E =
[1, 0, 0, 0;
 0, 1, 0, 0;
 0, 0, 1, 0;
 0, 0, 0, 1]

Z =
[0, 0, 0;
 0, 0, 0;
 0, 0, 0]

O =
[1, 0;
 0, 1]

```

图 4.4 方法五示例代码运行结果

## 6. 【方法六】对小矩阵使用逗号分隔式初始化函数

方法六为对小矩阵使用逗号分隔式初始化函数，示范代码如下。

```
Mat C = (Mat<double>(3, 3) << 0, -1, 0, -1, 5, -1, 0, -1, 0);
cout << "C = " << endl << " " << C << endl << endl;
```

上述代码的运行结果如图 4.5 所示。

```

C =
[0, -1, 0;
 -1, 5, -1;
 0, -1, 0]

```

图 4.5 方法六示例代码运行结果

## 7. 【方法七】为已存在的对象创建新信息头

方法七为使用成员函数 clone() 或者 copyTo() 为一个已存在的 Mat 对象创建一个新的信息头，示范代码如下。

```
Mat RowClone = C.row(1).clone();
cout << "RowClone = " << endl << " " << RowClone << endl << endl;
```

上述代码的运行结果如图 4.6 所示。

```

RowClone =
[-1, 5, -1]

```

图 4.6 方法七示例代码运行结果

### 4.1.5 OpenCV 中的格式化输出方法

在上一个例子中我们可以看到有默认的格式选项，同时，OpenCV 也提供了风格各异的格式化输出方法，本小节将对这些方法一一进行演示和列举。

首先是下面代码中将要使用的 r 矩阵的定义。需要注意，我们可以通过用 randu() 函数产生的随机值来填充矩阵，需要给定一个上限和下限来确保随机值在期望的范围内。

```
Mat r = Mat(10, 3, CV_8UC3);
randu(r, Scalar::all(0), Scalar::all(255));
```

初始化完 r 矩阵，下面便开始对输出风格的讲解。

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

### 1. 【风格一】OpenCV 默认风格

风格一为 OpenCV 默认风格的输出方法，如下。

```
cout << "r (OpenCV 默认风格) = " << r << ";" << endl << endl;
```

上述代码的运行结果如图 4.7 所示。

```
r <OpenCV默认风格> = [91, 2, 79, 179, 52, 205, 236, 8, 181;
239, 26, 248, 207, 218, 45, 183, 158, 101;
102, 18, 118, 68, 210, 139, 198, 207, 211;
181, 162, 197, 191, 196, 40, 7, 243, 230;
45, 6, 48, 173, 242, 125, 175, 90, 63;
90, 22, 112, 221, 167, 224, 113, 208, 123;
214, 35, 229, 6, 143, 138, 98, 81, 118;
187, 167, 140, 218, 178, 23, 43, 133, 154;
150, 76, 101, 8, 38, 238, 84, 47, 7;
```

图 4.7 OpenCV 默认风格运行结果

### 2. 【风格二】Python 风格

风格二为 Python 风格的输出方法，如下。

```
//此句代码的 OpenCV2 版为：
cout << "r (Python 风格) = " << format(r,"python") << ";" << endl << endl;

//此句代码的 OpenCV3 版为：
cout << "r (Python 风格) = " << format(r, Formatter::FMT_PYTHON) << ";" << endl << endl;
```

上述代码的运行结果如图 4.8 所示。

```
r <Python风格> = [[[91, 2, 79], [179, 52, 205], [236, 8, 181]],
[[239, 26, 248], [207, 218, 45], [183, 158, 101]],
[[102, 18, 118], [68, 210, 139], [198, 207, 211]],
[[181, 162, 197], [191, 196, 40], [7, 243, 230]],
[[45, 6, 48], [173, 242, 125], [175, 90, 63]],
[[90, 22, 112], [221, 167, 224], [113, 208, 123]],
[[214, 35, 229], [6, 143, 138], [98, 81, 118]],
[[187, 167, 140], [218, 178, 23], [43, 133, 154]],
[[150, 76, 101], [8, 38, 238], [84, 47, 7]],
[[117, 246, 163], [237, 69, 129], [60, 101, 41]]];
```

图 4.8 Python 风格运行结果

### 3. 【风格三】逗号分隔风格 (Comma separated values, CSV)

风格三为逗号分隔风格的输出方法，如下。

```
//此句代码的 OpenCV2 版为：
cout << "r(逗号分隔风格) = " << format(r,"csv") << ";" << endl << endl;

//此句代码的 OpenCV3 版为：
cout << "r (逗号分隔风格) = " << format(r, Formatter::FMT_CSV) << ";" << endl << endl;
```

上述代码的运行结果如图 4.9 所示。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
r <逗号分隔风格> = 91, 2, 79, 179, 52, 205, 236, 8, 181
239, 26, 248, 207, 218, 45, 183, 158, 101
102, 18, 118, 68, 210, 139, 198, 207, 211
181, 162, 197, 191, 196, 40, 7, 243, 230
45, 6, 48, 173, 242, 125, 175, 90, 63
90, 22, 112, 221, 167, 224, 113, 208, 123
214, 35, 229, 6, 143, 138, 98, 81, 118
187, 167, 140, 218, 178, 23, 43, 133, 154
150, 76, 101, 8, 38, 238, 84, 47, 7
117, 246, 163, 237, 69, 129, 60, 101, 41
;
```

图 4.9 逗号分隔风格运行结果

#### 4. 【风格四】Numpy 风格

风格四为 Numpy 风格的输出方法，如下。

```
//此句代码的 OpenCV2 版为：
cout << "r (Numpy 风格) = " << format(r,"numpy") << ";" << endl << endl;

//此句代码的 OpenCV3 版为：
cout << "r (Numpy 风格) = " << format(r, Formatter::FMT_NUMPY )<< ";" 
<< endl << endl;
```

上述代码的运行结果如图 4.10 所示。

```
r <Numpy风格> = array([[91, 2, 79], [179, 52, 205], [236, 8, 181],
[239, 26, 248], [207, 218, 45], [183, 158, 101],
[102, 18, 118], [68, 210, 139], [198, 207, 211],
[181, 162, 197], [191, 196, 40], [7, 243, 230],
[45, 6, 48], [173, 242, 125], [175, 90, 63]),
[90, 22, 112], [221, 167, 224], [113, 208, 123]),
[214, 35, 229], [6, 143, 138], [98, 81, 118],
[187, 167, 140], [218, 178, 23], [43, 133, 154]),
[150, 76, 101], [8, 38, 238], [84, 47, 7]),
[117, 246, 163], [237, 69, 129], [60, 101, 41]]), type='uint8');
```

图 4.10 Numpy 风格运行结果

#### 5. 【风格五】C 语言风格

风格五为 C 语言风格的输出方法，如下。

```
//此句代码的 OpenCV2 版为：
cout << "r (C 语言风格) = " << format(r,"C") << ";" << endl << endl;

//此句代码的 OpenCV3 版为：
cout << "r (C 语言风格) = " << format(r, Formatter::FMT_C      ) << ";" 
<< endl << endl;
```

上述代码的运行结果如图 4.11 所示。

```
r <C语言风格> = {91, 2, 79, 179, 52, 205, 236, 8, 181,
239, 26, 248, 207, 218, 45, 183, 158, 101,
102, 18, 118, 68, 210, 139, 198, 207, 211,
181, 162, 197, 191, 196, 40, 7, 243, 230,
45, 6, 48, 173, 242, 125, 175, 90, 63,
90, 22, 112, 221, 167, 224, 113, 208, 123,
214, 35, 229, 6, 143, 138, 98, 81, 118,
187, 167, 140, 218, 178, 23, 43, 133, 154,
150, 76, 101, 8, 38, 238, 84, 47, 7,
117, 246, 163, 237, 69, 129, 60, 101, 41};
```

图 4.11 C 语言风格运行结果

## 4.1.6 输出其他常用数据结构

之前我们讲解了如何输出 Mat 类型，其实，OpenCV 同样支持使用运算符“`<<`”来打印其他常用的 OpenCV 数据结构，本小节会通过代码对其中典型的几种进行讲解。

### 1. 定义和输出二维点

首先看看二维点的定义和输出方法：

```
Point2f p(6, 2);
cout << "【二维点】p = " << p << ";"\n" << endl;
```

上述代码的运行结果如图 4.12 所示。

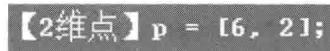


图 4.12 输出二维点

### 2. 定义和输出三维点

以下是三维点的定义和输出方法：

```
Point3f p3f(8, 2, 0);
cout << "【三维点】p3f = " << p3f << ";"\n" << endl;
```

上述代码的运行结果如图 4.13 所示。



图 4.13 输出三维点

### 3. 定义和输出基于 Mat 的 std::vector

接着是基于 Mat 类的 std::vector 的定义和输出方法：

```
vector<float> v;
v.push_back(3);
v.push_back(5);
v.push_back(7);

cout << "【基于Mat的vector】shortvec = " << Mat(v) << ";"\n" << endl;
```

上述代码的运行结果如图 4.14 所示。

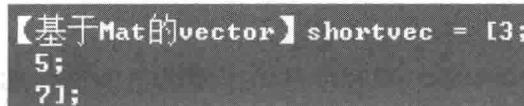


图 4.14 输出基于 Mat 的 vector

### 4. 定义和输出 std::vector 点

最后看看如何定义和输出存放着点的 vector 容器，以存放二维点 Point2f 为例：

```
vector<Point2f> points(20);
for (size_t i = 0; i < points.size(); ++i)
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
points[i] = Point2f((float)(i * 5), (float)(i % 7));  
  
cout << 【二维点向量】points = " << points<<";";
```

上述代码的运行结果如图 4.15 所示。

```
【二维点向量】points = [0, 0;  
5, 1;  
10, 2;  
15, 3;  
20, 4;  
25, 5;  
30, 6;  
35, 0;  
40, 1;  
45, 2;  
50, 3;  
55, 4;  
60, 5;  
65, 6;  
70, 0;  
75, 1;  
80, 2;  
85, 3;  
90, 4;  
95, 5];
```

图 4.15 输出 std::vector 点

#### 4.1.7 示例程序：基础图像容器 Mat 类的使用

本小节介绍的这些代码片段都被整理放到了一个短小的示例程序中，大家可以在书本配套示例程序包中找到此程序并运行。

运行截图之一如图 4.16 所示。

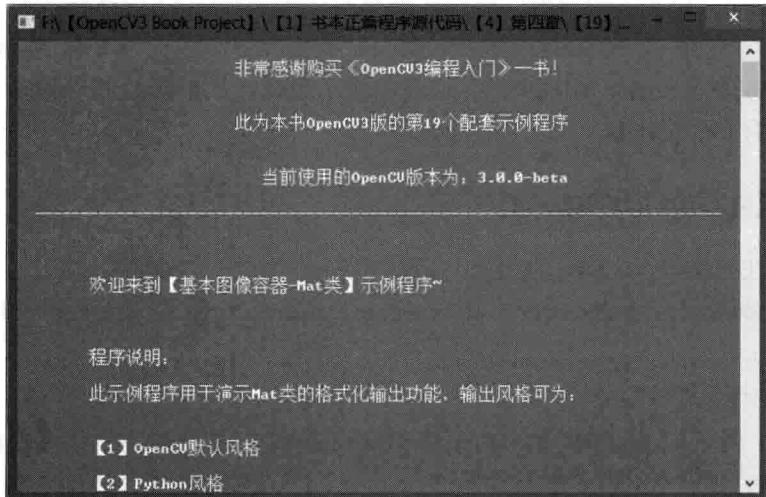


图 4.16 示例程序运行截图

## 4.2 常用数据结构和函数

本节我们将简单讲解 OpenCV 中常用的数据结构与函数。

### 4.2.1 点的表示: Point 类

Point 类数据结构表示了二维坐标系下的点, 即由其图像坐标  $x$  和  $y$  指定的 2D 点。用法如下:

```
Point point;
point.x = 10;
point.y = 8;
```

或者

```
Point point = Point(10, 8);
```

另外, 在 OpenCV 中有如下定义:

```
typedef Point<int> Point2i;
typedef Point2i Point;
typedef Point<float> Point2f;
```

所以, `Point<int>`、`Point2i`、`Point` 互相等价, `Point<float>`、`Point2f` 互相等价。

### 4.2.2 颜色的表示: Scalar 类

Scalar() 表示具有 4 个元素的数组, 在 OpenCV 中被大量用于传递像素值, 如 RGB 颜色值。而 RGB 颜色值为三个参数, 其实对于 Scalar 函数来说, 如果用不到第四个参数, 则不需要写出来; 若只写三个参数, OpenCV 会认为我们就想表示三个参数。

来看个例子。如果给出以下颜色参数表达式:

```
Scalar( a, b, c )
```

那么定义的 RGB 颜色值: 红色分量为  $c$ , 绿色分量为  $b$ , 蓝色分量为  $a$ 。

Scalar 类的源头为 Scalar\_类, 而 Scalar\_类是 Vec4x 的一个变种, 我们常用的 Scalar 其实就是 `Scalar<double>`。这就解释了为什么很多函数的参数输入可以是 Mat, 也可以是 Scalar。

### 4.2.3 尺寸的表示: Size 类

通过在代码中对 Size 类进行“转到定义”操作, 我们可以在……\opencv\sources\modules\core\include\opencv2\core\core.hpp 路径下, 找到 Size 类相关的源代码:

```
typedef Size<int> Size2i;
typedef Size2i Size;
```

其中, `Size_` 是个模板类, 在这里 `Size<int>` 表示其类体内部的模板所代表的类型为 int。那这两句代码的意思, 就是首先给已知的数据类型 `Size<int>` 起个新名字, 叫 `Size2i`。然后又给已知的数据类型 `Size2i` 起个新名字, 叫 `Size`。所以, 连起来就是, `Size<int>`、`Size2i`、`Size` 这三个类型名等价。

然后我们追根溯源, 找到 `Size_` 模板类的定义:

```
template<typename _Tp> class Size_
{
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

public:
    typedef _Tp value_type;

    //不同的构造函数定义
    Size_();
    Size_(_Tp _width, _Tp _height);
    Size_(const Size_& sz);
    Size_(const CvSize& sz);
    Size_(const CvSize2D32f& sz);
    Size_(const Point_<_Tp>& pt);

    Size_& operator = (const Size_& sz);
    //区域(width*height)
    _Tp area() const;

    //转化为另一种数据类型
    template<typename _Tp2> operator Size_<_Tp2>() const;

    //转换为旧式的 OpenCV 类型
    operator CvSize() const;
    operator CvSize2D32f() const;

    _Tp width, height; //宽度和高度, 常用属性
};

```

可以看到 `Size_` 模板类的内部又是重载了一些构造函数，其中，我们使用频率最高的是下面这个构造函数：

```
Size_(_Tp _width, _Tp _height);
```

另外，代码末尾定义了模板类型的宽度和高度：

```
_Tp width, height; //宽度和高度
```

于是我们可以用 `XXX.width` 和 `XXX.height` 来分别表示其宽度和高度。

下面给出一个示例，方便大家理解。

```
Size(5, 5); //构造出的 Size 宽度和高度都为 5，即 XXX.width 和 XXX.height 都为 5
```

#### 4.2.4 矩形的表示：Rect 类

`Rect` 类的成员变量有 `x`、`y`、`width`、`height`，分别为左上角点的坐标和矩形的宽和高。常用的成员函数有：`Size()`返回值为 `Size`；`area()`返回矩形的面积；`contains(Point)`判断点是否在矩形内；`inside(Rect)`函数判断矩形是否在该矩形内；`tl()`返回左上角点坐标；`br()`返回右下角点坐标。值得注意的是，如果想求两个矩形的交集和并集，可以用如下格式：

```
Rect rect = rect1 & rect2;
Rect rect = rect1 | rect2;
```

如果想让矩形进行平移操作和缩放操作，甚至可以这样写：

```
Rect rectShift = rect + point;
```

```
Rect rectScale = rect + size;
```

#### 4.2.5 颜色空间转换: cvtColor()函数

cvtColor()函数是 OpenCV 里的颜色空间转换函数，可以实现 RGB 颜色向 HSV、HSI 等颜色空间的转换，也可以转换为灰度图像。

原型如下：

```
C++: void cvtColor(InputArray src, OutputArray dst, int code, int dstCn=0)
```

第一个参数为输入图像，第二个参数为输出图像，第三个参数为颜色空间转换的标识符（具体见表 4.1），第四个参数为目标图像的通道数，若该参数是 0，表示目标图像取源图像的通道数。下面是一个调用示例：

```
//此句代码的 OpenCV2 版为：  
cvtColor(srcImage,dstImage, CV_GRAY2BGR); //转换原始图为灰度图  
//此句代码的 OpenCV3 版为：  
cvtColor(srcImage,dstImage, COLOR_GRAY2BGR); //转换原始图为灰度图
```

而随着 OpenCV 版本的升级，cvtColor()函数对于颜色空间种类的支持也是越来越多。其标识符列举如表 4.1 所示。

表 4.1 cvtColor()函数标识符（OpenCV2 版）

转换类型	标识符列举
RGB <-> BGR	CV_BGR2BGRA、CV_RGB2BGRA、CV_BGRA2RGBA、CV_BGR2BGRA、CV_BGRA2BGR
RGB <-> 5X5	CV_BGR5652RGBA、CV_BGR2RGB555 等
RGB <-> Gray	CV_RGB2GRAY、CV_GRAY2RGB、CV_RGBA2GRAY、CV_GRAY2RGBA
RGB <-> CIE XYZ	CV_BGR2XYZ、CV_RGB2XYZ、CV_XYZ2BGR、CV_XYZ2RGB
RGB <-> YCrCb (YUV) JPEG	CV_RGB2YCrCb、CV_RGB2YCrCb、CV_YCrCb2BGR、CV_YCrCb2RGB、CV_RGB2YUV (可将 YCrCb 用 YUV 替代)
RGB <-> HSV	CV_BGR2HSV、CV_RGB2HSV、CV_HSV2BGR、CV_HSV2RGB
RGB <-> HLS	CV_BGR2HLS、CV_RGB2HLS、CV_HLS2BGR、CV_HLS2RGB
RGB <-> CIE L*a*b*	CV_BGR2Lab、CV_RGB2Lab、CV_Lab2BGR、CV_Lab2RGB
RGB <-> CIE L*u*v*	CV_BGR2Luv、CV_RGB2Luv、CV_Luv2BGR、CV_Luv2RGB
RGB <-> Bayer	CV_BayerBG2BGR、CV_BayerGB2BGR、CV_BayerRG2BGR、CV_BayerGR2BGR、CV_BayerBG2RGB、CV_BayerGB2RGB、CV_BayerRG2RGB、CV_BayerGR2RGB (在 CCD 和 CMOS 上常用的 Bayer 模式)
YUV420 <-> RGB	CV_YUV420sp2BGR、CV_YUV420sp2RGB、CV_YUV420i2BGR、CV_YUV420i2RGB

表 4.2 cvtColor()函数标识符 (OpenCV3 版)

转换类型	标识符列举
RGB <--> BGR	COLOR_BGR2BGRA 、 COLOR_RGB2BGRA 、 COLOR_BGRA2RGBA 、 COLOR_BGR2BGRA 、 COLOR_BGRA2BGR
RGB <--> 5X5	COLOR_BGR5652RGBA、 COLOR_BGR2RGB55 等
RGB <--> Gray	COLOR_RGB2GRAY 、 COLOR_GRAY2RGB 、 COLOR_RGBA2GRAY、 COLOR_GRAY2RGBA
RGB <--> CIE XYZ	COLOR_BGR2XYZ、COLOR_RGB2XYZ、COLOR_XYZ2BGR、 COLOR_XYZ2RGB
RGB <--> YCrCb (YUV) JPEG	COLOR_RGB2YCrCb 、 COLOR_RGB2YCrCb 、 COLOR_YCrCb2BGR 、 COLOR_YCrCb2RGB 、 COLOR_RGB2YUV (可将 YCrCb 用 YUV 替代)
RGB <--> HSV	COLOR_BGR2HSV、COLOR_RGB2HSV、COLOR_HSV2BGR、 COLOR_HSV2RGB
RGB <--> HLS	COLOR_BGR2HLS、COLOR_RGB2HLS、COLOR_HLS2BGR、 COLOR_HLS2RGB
RGB <--> CIE L*a*b*	COLOR_BGR2Lab、COLOR_RGB2Lab、COLOR_Lab2BGR、 COLOR_Lab2RGB
RGB <--> CIE L*u*v*	COLOR_BGR2Luv、COLOR_RGB2Luv、COLOR_Luv2BGR、 COLOR_Luv2RGB
RGB <--> Bayer	COLOR_BayerBG2BGR 、 COLOR_BayerGB2BGR 、 COLOR_BayerRG2BGR 、 COLOR_BayerGR2BGR 、 COLOR_BayerBG2RGB 、 COLOR_BayerGB2RGB 、 COLOR_BayerRG2RGB、 COLOR_BayerGR2RGB (在 CCD 和 CMOS 上常用的 Bayer 模式)
YUV420 <--> RGB	COLOR_YUV420sp2BGR 、 COLOR_YUV420sp2RGB 、 COLOR_YUV420i2BGR、 COLOR_YUV420i2RGB

即对于颜色空间转换，OpenCV2 的 CV\_ 前缀的宏命名规范，被 OpenCV3 中 COLOR\_ 式的宏命名前缀所取代。另外，在这里需要再次提醒大家的是，OpenCV 默认的图片通道存储顺序是 BGR，即蓝绿红，而不是 RGB。

本节最后，附上进行颜色空间转换的最简化版代码，大家可以选择上表中列举的宏替换掉 cvtColor 函数中名为 COLOR\_BGR2Lab 的宏，进行 cvtColor 函数的测试。

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
using namespace cv;
```

```
void main( )
{
    //【1】载入图片
    Mat srcImage=imread("1.jpg",1),dstImage;
    //【2】转换颜色空间
    cvtColor(srcImage,dstImage, COLOR_BGR2Lab);
    //【3】显示效果图
    imshow("效果图",dstImage);
    //【4】保持窗口显示
    waitKey();
}
```

## 4.2.6 其他常用的知识点

本小节我们列举一下 OpenCV 的 Core 模块中其他常用的知识点，如下。

- Matx 是个轻量级的 Mat，必须在使用前规定好大小，比如一个 2\*3 的 float 型的 Matx，可以声明为 Matx23f。
- Vec 是 Matx 的一个派生类，是一个一维的 Matx，跟 vector 很相似。在 OpenCV 源码中有如下定义。

```
template<typename _Tp, int n> class Vec : public Matx<_Tp, n, 1> {...};
typedef Vec<uchar, 2> Vec2b;
```

- Range 类其实就是为了使 OpenCV 的使用更像 MATLAB 而产生的。比如 Range::all() 其实就是 MATLAB 里的符号。而 Range(a, b) 其实就是 MATLAB 中的 a: b，注意这里的 a 和 b 都应为整型。
- OpenCV 中防止内存溢出的函数有 alignPtr、alignSize、allocate、deallocate、fastMalloc、fastFree 等。
- <math.h> 里的一些函数使用起来很方便，有计算向量角度的函数 fastAtan2、计算立方根的函数 cubeRoot、向上取整函数 cvCeil、向下取整函数 cvFloor、四舍五入函数 cvRound 等。还有一些类似 MATLAB 里面的函数，比如 cvIsInf 判断自变量是否无穷大，cvIsNaN 判断自变量是否不是一个数。
- 显示文字相关的函数有 getTextSize、cvInitFont、putText。
- 作图相关的函数有 circle、clipLine、ellipse、ellipse2Poly、line、rectangle、polylines、类 LineIterator。
- 填充相关的函数有 fillConvexPoly、fillPoly。
- OpenCV 中 RNG() 函数的作用为初始化随机数状态的生成器。

其他数据结构相关的知识由于使用较少在此不再过多说明，在需要用到的时候读者可以配合 OpenCV 文档，并查阅 OpenCV 的源码进行学习。

## 4.3 基本图形的绘制

本节我们将学习如何用 Point 在图像中定义 2D 点、如何使用 Scalar 表示颜色值。涉及到的绘制函数如下：

- 用于绘制直线的 line 函数；
- 用于绘制椭圆的 ellipse 函数；
- 用于绘制矩形的 rectangle 函数；
- 用于绘制圆的 circle 函数；
- 用于绘制填充的多边形的 fillPoly 函数。

让我们通过一个程序实例的学习来掌握 OpenCV 中各种绘制函数的用法。此程序的原型为 OpenCV 官方的示例程序，主要的脉络是定义了几个自定义的绘制函数，然后调用这些自定义的函数绘制出了两幅图——一幅化学原子示例图和一幅组合图。

在此，我们主要分析一下程序中的 4 个自定义函数的写法和用意。

需要注意，程序的源文件开头有如下的宏定义：

```
#define WINDOW_WIDTH 600//定义窗口大小的宏
```

### 4.3.1 DrawEllipse()函数的写法

如下为 DrawEllipse() 函数的代码。

```
-----【DrawEllipse() 函数】-----  
//      描述：自定义的绘制函数，实现了绘制不同角度、相同尺寸的椭圆  
-----  
void DrawEllipse( Mat img, double angle )  
{  
    int thickness = 2;  
    int lineType = 8;  
  
    ellipse( img,  
             Point( WINDOW_WIDTH/2, WINDOW_WIDTH/2 ),  
             Size( WINDOW_WIDTH/4, WINDOW_WIDTH/16 ),  
             angle,  
             0,  
             360,  
             Scalar( 255, 129, 0 ),  
             thickness,  
             lineType );  
}
```

此函数的写法解析如下。

函数 DrawEllipse 调用了 OpenCV 中的 ellipse 函数，将椭圆画到图像 img 上，椭圆中心为点(WINDOW\_WIDTH/2.0, WINDOW\_WIDTH/2.0)，并且大小位于矩形(WINDOW\_WIDTH/4.0, WINDOW\_WIDTH/16.0)内。椭圆旋转角度为 angle，扩展的弧度从 0 度到 360 度。图形颜色为 Scalar(255,129,0) 代表的蓝色，线宽(thickness)为 2，线型(lineType)为 8 (8 联通线型)。



线型的含义详见第 8 章 8.1.2 节的表 8.3。

### 4.3.2 DrawFilledCircle()函数的写法

```
-----【DrawFilledCircle() 函数】-----
//      描述：自定义的绘制函数，实现了实心圆的绘制
-----
void DrawFilledCircle( Mat img, Point center )
{
    int thickness = -1;
    int lineType = 8;

    circle( img,
            center,
            WINDOW_WIDTH/32,
            Scalar( 0, 0, 255 ),
            thickness,
            lineType );
}
```

此函数的写法解析如下。

函数 DrawFilledCircle() 调用了 OpenCV 中的 circle 函数，将圆画到图像 img 上，圆心由点 center 定义，圆的半径为 WINDOW\_WIDTH/32，圆的颜色为 Scalar(0,0,255)，按 BGR 的格式为红色，线粗定义为 thickness = -1，因此绘制的圆是实心的。

### 4.3.3 DrawPolygon()函数的写法

```
-----【DrawPolygon() 函数】-----
//      描述：自定义的绘制函数，实现了凹多边形的绘制
-----
void DrawPolygon( Mat img )
{
    int lineType = 8;

    // 创建一些点
    Point rookPoints[1][20];
    rookPoints[0][0] = Point(    WINDOW_WIDTH/4,    7*WINDOW_WIDTH/8 );
    rookPoints[0][1] = Point( 3*WINDOW_WIDTH/4,    7*WINDOW_WIDTH/8 );
    rookPoints[0][2] = Point( 3*WINDOW_WIDTH/4,
13*WINDOW_WIDTH/16 );
    rookPoints[0][3] = Point( 11*WINDOW_WIDTH/16,
13*WINDOW_WIDTH/16 );
    rookPoints[0][4] = Point( 19*WINDOW_WIDTH/32, 3*WINDOW_WIDTH/8 );
    rookPoints[0][5] = Point( 3*WINDOW_WIDTH/4, 3*WINDOW_WIDTH/8 );
    rookPoints[0][6] = Point( 3*WINDOW_WIDTH/4,    WINDOW_WIDTH/8 );
    rookPoints[0][7] = Point( 26*WINDOW_WIDTH/40,  WINDOW_WIDTH/8 );
    rookPoints[0][8] = Point( 26*WINDOW_WIDTH/40,  WINDOW_WIDTH/4 );
    rookPoints[0][9] = Point( 22*WINDOW_WIDTH/40,  WINDOW_WIDTH/4 );
    rookPoints[0][10] = Point( 22*WINDOW_WIDTH/40,  WINDOW_WIDTH/8 );
    rookPoints[0][11] = Point( 18*WINDOW_WIDTH/40,  WINDOW_WIDTH/8 );
    rookPoints[0][12] = Point( 18*WINDOW_WIDTH/40,  WINDOW_WIDTH/4 );
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

rookPoints[0][13] = Point( 14*WINDOW_WIDTH/40,      WINDOW_WIDTH/4 );
rookPoints[0][14] = Point( 14*WINDOW_WIDTH/40,      WINDOW_WIDTH/8 );
rookPoints[0][15] = Point(      WINDOW_WIDTH/4,      WINDOW_WIDTH/8 );
rookPoints[0][16] = Point(      WINDOW_WIDTH/4,      3*WINDOW_WIDTH/8 );
rookPoints[0][17] = Point( 13*WINDOW_WIDTH/32,      3*WINDOW_WIDTH/8 );
rookPoints[0][18] = Point(  5*WINDOW_WIDTH/16,
13*WINDOW_WIDTH/16 );
rookPoints[0][19] = Point(      WINDOW_WIDTH/4,
13*WINDOW_WIDTH/16 );

const Point* ppt[1] = { rookPoints[0] };
int npt[] = { 20 };

fillPoly( img,
ppt,
npt,
1,
Scalar( 255, 255, 255 ),
lineType );
}

```

此函数的写法解析如下。

函数 DrawPolygon() 调用了 OpenCV 中的 fillPoly 函数，用于将多边形画到图像 img 上，其中多边形的顶点集为 ppt，要绘制的多边形顶点数目为 npt，要绘制的多边形数量仅为 1，多边形的颜色定义为白色 Scalar(255,255,255)。

#### 4.3.4 DrawLine()函数的写法

```

-----【 DrawLine() 函数 】-----
//      描述：自定义的绘制函数，实现了线的绘制
-----
void DrawLine( Mat img, Point start, Point end )
{
    int thickness = 2;
    int lineType = 8;
    line( img,
          start,
          end,
          Scalar( 0, 0, 0 ),
          thickness,
          lineType );
}

```

此函数的写法解析如下。

DrawLin() 函数调用了 OpenCV 中的 line 函数，用于在图像 img 上画一条从点 start 到点 end 的直线段，线的颜色为 Scalar(0,0,0) 代表的黑色，线的粗细 thickness 为 2，且此线为 8 联通 (lineType = 8)。

以上为 4 个自定义函数的写法和分析。接下来，让我们一起看看 main 函数的写法。

### 4.3.5 main 函数的写法

main 函数的写法非常简单，先创建空白的 Mat 图像，然后调用函数绘制化学中的原子示例图，接着绘制组合图，最后显示绘制出的图像。

```
//-----【头文件、命名空间包含部分】-----  
  
//      描述：包含程序所使用的头文件和命名空间  
//-----  
#include <opencv2/core/core.hpp>  
#include <opencv2/highgui/highgui.hpp>  
#include <opencv2/imgproc/imgproc.hpp>  
using namespace cv;  
  
//-----【宏定义部分】-----  
//  描述：定义一些辅助宏  
//-----  
#define WINDOW_NAME1 "【绘制图 1】"          //为窗口标题定义的宏  
#define WINDOW_NAME2 "【绘制图 2】"          //为窗口标题定义的宏  
#define WINDOW_WIDTH 600//定义窗口大小的宏  
//-----【main() 函数】-----  
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行  
//-----  
int main( void )  
{  
  
    // 创建空白的 Mat 图像  
    Mat atomImage = Mat::zeros( WINDOW_WIDTH, WINDOW_WIDTH, CV_8UC3 );  
    Mat rookImage = Mat::zeros( WINDOW_WIDTH, WINDOW_WIDTH, CV_8UC3 );  
  
    // -----<1>绘制化学中的原子示例图-----  
  
    //【1.1】先绘制出椭圆  
    DrawEllipse( atomImage, 90 );  
    DrawEllipse( atomImage, 0 );  
    DrawEllipse( atomImage, 45 );  
    DrawEllipse( atomImage, -45 );  
  
    //【1.2】再绘制圆心  
    DrawFilledCircle( atomImage, Point( WINDOW_WIDTH/2,  
    WINDOW_WIDTH/2 ) );  
  
    // -----<2>绘制组合图-----  
    //【2.1】先绘制出椭圆  
    DrawPolygon( rookImage );  
  
    //【2.2】绘制矩形  
    rectangle( rookImage,  
    Point( 0, 7*WINDOW_WIDTH/8 ),  
    Point( WINDOW_WIDTH, WINDOW_WIDTH ),  
    Scalar( 0, 255, 255 ),  
    -1,
```

```
    8 );  
  
    // 【2.3】绘制一些线段  
    DrawLine( rookImage, Point( 0, 15*WINDOW_WIDTH/16 ),  
    Point( WINDOW_WIDTH, 15*WINDOW_WIDTH/16 ) );  
    DrawLine( rookImage, Point( WINDOW_WIDTH/4, 7*WINDOW_WIDTH/8 ),  
    Point( WINDOW_WIDTH/4, WINDOW_WIDTH ) );  
    DrawLine( rookImage, Point( WINDOW_WIDTH/2, 7*WINDOW_WIDTH/8 ),  
    Point( WINDOW_WIDTH/2, WINDOW_WIDTH ) );  
    DrawLine( rookImage, Point( 3*WINDOW_WIDTH/4, 7*WINDOW_WIDTH/8 ),  
    Point( 3*WINDOW_WIDTH/4, WINDOW_WIDTH ) );  
  
    // -----<3>显示绘制出的图像-----  
    imshow( WINDOW_NAME1, atomImage );  
    moveWindow( WINDOW_NAME1, 0, 200 );  
    imshow( WINDOW_NAME2, rookImage );  
    moveWindow( WINDOW_NAME2, WINDOW_WIDTH, 200 );  
  
    waitKey( 0 );  
    return(0);  
}
```

运行上述代码组合起来的程序，便可以绘制出如图 4.17 和图 4.18 所示的图片。

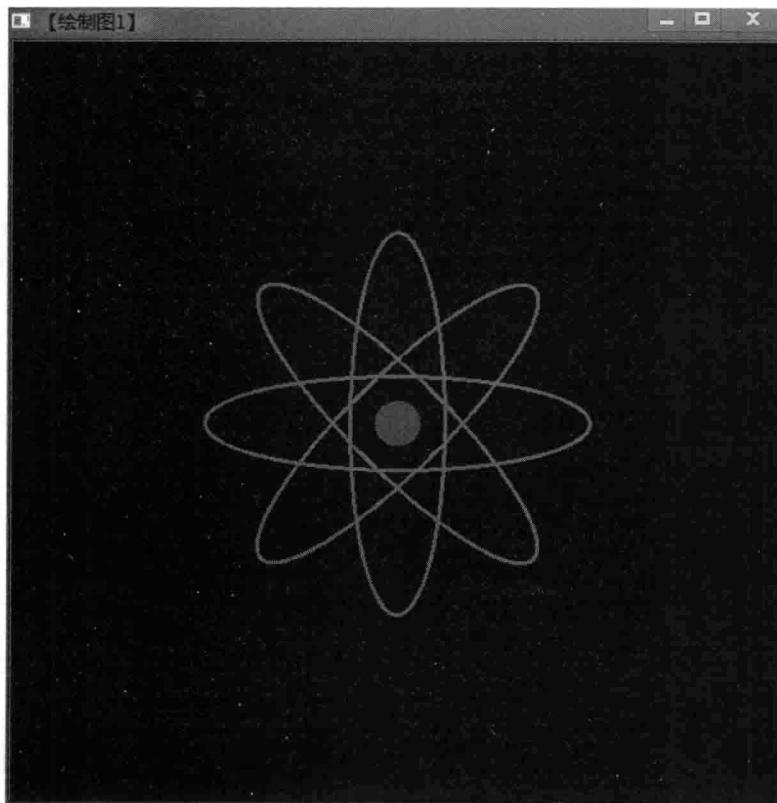


图 4.17 绘制出的化学原子示例图

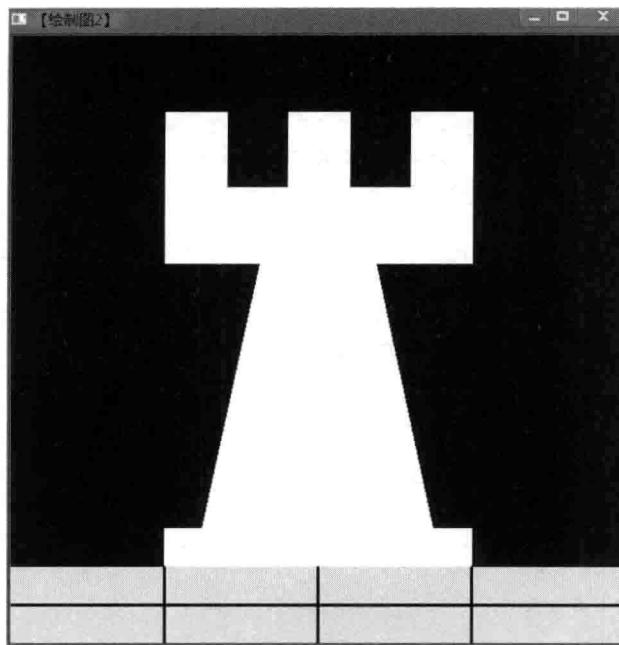


图 4.18 绘制出的组合图

## 4.4 本章小结

本章我们主要学习了经常会遇到的各种数据结构，主要是基础图像容器 Mat 的用法。

### 本章核心函数/类清单

函数名称	说明	对应讲解章节
Mat::Mat()	Mat 类的构造函数	4.1.4
Mat::Create()	Mat 类的成员函数，可用于 Mat 类的初始化操作	4.1.4
Point 类	用于表示点的数据结构	4.2.1
Scalar 类	用于表示颜色的数据结构	4.2.2
Size 类	用于表示尺寸的数据结构	4.2.3
Rect 类	用于表示矩形的数据结构	4.2.4
CvtColor()	用于颜色空间转换	4.2.5

### 本章示例程序清单

示例程序序号	程序说明	对应章节
19	基础图像容器 Mat 类的使用	4.1.7
20	基本图形的绘制	4.3

# 第 5 章

## core 组件进阶

### 导读

---

本章中，你将学到：

- 如何操作图像中的像素
- 设置感兴趣区域（ROI）
- 如何进行图像混合
- 如何分离颜色通道
- 如何进行多通道图像混合
- 如何调整图像的对比度和亮度值
- 如何对图像进行离散傅里叶变换
- 如何输入输出 XML 和 YAML 文件

## 5.1 访问图像中的像素

### 5.1.1 图像在内存之中的存储方式

在之前的章节中，我们已经了解到图像矩阵的大小取决于所用的颜色模型，确切地说，取决于所用通道数。如果是灰度图像，矩阵就会如图 5.1 所示。

	Column 0	Column 1	Column ...	Column m
Row 0	0,0	0,1	...	0, m
Row 1	1,0	1,1	...	1, m
Row ...	...,0	...,1	...	..., m
Row n	n,0	n,1	n,...	n, m

图 5.1 灰度图像模型的矩阵排列

而对多通道图像来说，矩阵中的列会包含多个子列，其子列个数与通道数相等。例如，如图 5.2 所示 RGB 颜色模型的矩阵。

	Column 0	Column 1	Column ...	Column m
Row 0	0,0 1,0 ...,0 n,0	0,1 1,1 ...,1 n,1	0,1 1,1 ...,1 n,1	0, m 1, m ..., m n, m
Row 1	0,0 1,0 ...,0 n,0	0,1 1,1 ...,1 n,1	0,1 1,1 ...,1 n,1	0, m 1, m ..., m n, m
Row ...	0,0 1,0 ...,0 n,0	0,1 1,1 ...,1 n,1	0,1 1,1 ...,1 n,1	0, m 1, m ..., m n, m
Row n	0,0 1,0 ...,0 n,0	0,1 1,1 ...,1 n,1	0,1 1,1 ...,1 n,1	0, m 1, m ..., m n, m

图 5.2 RGB 颜色模型的矩阵排列

可以看到，OpenCV 中子列的通道顺序是反过来的——BGR 而不是 RGB。很多情况下，因为内存足够大，可实现连续存储，因此，图像中的各行就能一行一行地连接起来，形成一个长行。连续存储有助于提升图像扫描速度，我们可以使用 `isContinuous()` 来判断矩阵是否是连续存储的。相关示例会在接下来的内容中提供。

### 5.1.2 颜色空间缩减

我们知道，若矩阵元素存储的是单通道像素，使用 C 或 C++ 的无符号字符类型，那么像素可有 256 个不同值。但若是三通道图像，这种存储格式的颜色数就太多了（确切地说，有一千六百多万种）。用如此之多的颜色来进行处理，可能会对我们的算法性能造成严重影响。

其实，仅用这些颜色中具有代表性的很小的部分，就足以达到同样的效果。

如此，颜色空间缩减（color space reduction）便可以派上用场了，它在很多应用中可以大大降低运算复杂度。颜色空间缩减的做法是：将现有颜色空间值除以某个输入值，以获得较少的颜色数。也就是“做减法”，比如颜色值 0 到 9 可取为新值 0, 10 到 19 可取为 10，以此类推。

如 `uchar` 类型的三通道图像，每个通道取值可以是 0~255，于是就有  $256 \times 256$  个不同的值。我们可以定义：

- 0~9 范围的像素值为 0;
- 10~19 范围的像素值为 10;
- 20~29 范围的像素值为 20。

这样的操作将颜色取值降低为  $26 \times 26 \times 26$  种情况。这个操作可以用一个简单的公式来实现。因为 C++ 中 int 类型除法操作会自动截余。例如  $I_{old}=14$ ;  
 $I_{new}=(I_{old}/10)*10=(14/10)*10=1*10=10$ ;

`uchar` (无符号字符, 即 0 到 255 之间取值的数) 类型的值除以 int 值, 结果仍是 `char`。因为结果是 `char` 类型的, 所以求出来小数也要向下取整。利用这一点, 刚才提到在 `uchar` 定义域中进行的颜色缩减运算就可以表达为下面的形式:

$$I_{new} = \left( \frac{I_{old}}{10} \right) \times 10$$

因为 C++ 中 int 类型除法操作会自动截余。比如:

```
Iold=14; Inew=(Iold/10)*10=(14/10)*10=1*10=10;
```

在处理图像像素时, 每个像素需要进行一遍上述计算, 也需要一定的时间花销。但我们注意到其实只有 0~255 种像素, 即只有 256 种情况。进一步可以把 256 种计算好的结果提前存在表中 `table` 中, 这样每种情况不需计算, 直接从 `table` 中取结果即可。

```
int divideWith=10;
uchar table[256];
for (int i = 0; i < 256; ++i)
    table[i] = divideWith * (i/divideWith);
```

于是 `table[i]` 存放的是值为 `i` 的像素减小颜色空间的结果, 这样也就可以理解上述方法中的操作:

```
p[j] = table[p[j]];
```

这样, 简单的颜色空间缩减算法就可由下面两步组成:

- (1) 遍历图像矩阵的每一个像素;
- (2) 对像素应用上述公式。

值得注意的是, 我们这里用到了除法和乘法运算, 而这两种运算又特别费时, 所以, 应尽可能用代价较低的加、减、赋值等运算来替换它们。此外, 还应注意, 上述运算的输入仅能在某个有限范围内取值, 如 `uchar` 类型可取 256 个值。

由此可知, 对于较大的图像, 有效的方法是预先计算所有可能的值, 然后需要这些值的时候, 利用查找表直接赋值即可。查找表是一维或多维数组, 存储了不同输入值所对应的输出值, 其优势在于只需读取、无须计算。

### 5.1.3 LUT 函数: Look up table 操作

对于上文提到的 Look up table 操作, OpenCV 官方文档中强烈推荐我们使用

## SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

一个原型为 operationsOnArrays:LUT()<lut>的函数来进行。它用于批量进行图像元素查找、扫描与操作图像。其使用方法如下：

```
//首先我们建立一个mat型用于查表
Mat lookUpTable(1, 256, CV_8U);
uchar* p = lookUpTable.data;
for( int i = 0; i < 256; ++i)
    p[i] = table[i];

//然后我们调用函数(I是输入 J是输出):
for( int i = 0; i < times; ++i)
    LUT(I, lookUpTable, J);
```

### 5.1.4 计时函数

另外有个问题是是如何计时。可以利用这两个简便的计时函数——getTickCount()和getTickFrequency()。

- getTickCount()函数返回CPU自某个事件（如启动电脑）以来走过的时钟周期数
- getTickFrequency()函数返回CPU一秒钟所走的时钟周期数。这样，我们就能轻松地以秒为单位对某运算计时。

这两个函数组合起来使用的示例如下。

```
double time0 = static_cast<double>(getTickCount()); //记录起始时间
//进行图像处理操作……
time0 = ((double)getTickCount() - time0)/getTickFrequency();
cout<<"此方法运行时间为: "<<time0<<"秒"<<endl; //输出运行时间
```

### 5.1.5 访问图像中像素的三类方法

任何图像处理算法，都是从操作每个像素开始的。即使我们不会使用OpenCV提供的各种图像处理函数，只要了解了图像处理算法的基本原理，也可以写出具有相同功能的程序。在OpenCV中，提供了三种访问每个像素的方法。

- 方法一 指针访问：C操作符[]；
- 方法二 迭代器 iterator；
- 方法三 动态地址计算。

这三种方法在访问速度上略有差异。debug模式下，这种差异非常明显，不过在release模式下，这种差异就不太明显了。我们通过一组程序来说明这几种方法。程序的目的是减少图像中颜色的数量，比如原来的图像是256种颜色，我们希望将它变成64种颜色，那只需要将原来的颜色除以4（整除）以后再乘以4就可以了。

将要使用的主程序代码如下。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
-----  
#include <opencv2/core/core.hpp>  
#include <opencv2/highgui/highgui.hpp>  
#include <iostream>  
using namespace std;  
using namespace cv;  
  
-----【全局函数声明部分】-----  
//      描述：全局函数声明  
-----  
void colorReduce(Mat& inputImage, Mat& outputImage, int div);  
  
-----【main()函数】-----  
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行  
-----  
int main()  
{  
    //【1】创建原始图并显示  
    Mat srcImage = imread("1.jpg");  
    imshow("原始图像", srcImage);  
  
    //【2】按原始图的参数规格来创建效果图  
    Mat dstImage;  
    dstImage.create(srcImage.rows, srcImage.cols, srcImage.type()); //效果图的大小、类型与原图片相同  
  
    //【3】记录起始时间  
    double time0 = static_cast<double>(getTickCount());  
  
    //【4】调用颜色空间缩减函数  
    colorReduce(srcImage, dstImage, 32);  
  
    //【5】计算运行时间并输出  
    time0 = ((double)getTickCount() - time0)/getTickFrequency();  
    cout<<"此方法运行时间为："<<time0<<"秒"<<endl; //输出运行时间  
  
    //【6】显示效果图  
    imshow("效果图", dstImage);  
    waitKey(0);  
}
```

主程序中调用 colorReduce 函数来完成减少颜色的工作，而我们根据访问每个像素三类方法，写出了三个版本的 colorReduce 函数。下面，我们将分别对其进行简单地讲解。

## 1. 【方法一】用指针访问像素

用指针访问像素的这种方法利用的是 C 语言中的操作符 [ ]。这种方法最快，但是略有点抽象。实验条件下单次运行时间为 0.00665378。范例代码如下。

```
-----【colorReduce()函数】-----  
//      描述：使用【指针访问：C 操作符 [ ]】方法版的颜色空间缩减函数
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

//-----
void colorReduce(Mat& inputImage, Mat& outputImage, int div)
{
    //参数准备
    outputImage = inputImage.clone(); //复制实参到临时变量
    int rowNumber = outputImage.rows; //行数
    int colNumber = outputImage.cols*outputImage.channels(); //列数×
    //通道数=每一行元素的个数

    //双重循环，遍历所有的像素值
    for(int i = 0;i < rowNumber;i++) //行循环
    {
        uchar* data = outputImage.ptr<uchar>(i); //获取第 i 行的首地址
        for(int j = 0;j < colNumber;j++) //列循环
        {
            // -----【开始处理每个像素】-----
            data[j] = data[j]/div*div + div/2;
            // -----【处理结束】-----
        } //行处理结束
    }
}

```

下面对上述代码进行讲解。

Mat 类有若干成员函数可以获取图像的属性。公有成员变量 cols 和 rows 给出了图像的宽和高，而成员函数 channels() 用于返回图像的通道数。灰度图的通道数为 1，彩色图的通道数为 3。

每行的像素值由以下语句得到：

```
int colNumber = outputImage.cols*outputImage.channels(); //列数×通道数=
//每一行元素的个数
```

为了简化指针运算，Mat 类提供了 ptr 函数可以得到图像任意行的首地址。ptr 是一个模板函数，它返回第 *i* 行的首地址：

```
uchar* data = outputImage.ptr<uchar>(i); //获取第 i 行的首地址
```

而双层循环内部的那句处理像素的代码，我们可以等效地使用指针运算从一列移动到下一列。所以，也可以这样来写：

```
*data++=*data/div*div+div/2;
```

## 2. 【方法二】用迭代器操作像素

第二种方法为用迭代器操作像素，这种方法与 STL 库的用法类似。

在迭代法中，我们所要做的仅仅是获得图像矩阵的 begin 和 end，然后增加迭代直至从 begin 到 end。将\*操作符添加在迭代指针前，即可访问当前指向的内容。

相比用指针直接访问可能出现越界问题，迭代器绝对是非常安全的方法。

```

//-----【colorReduce() 函数】-----
//      描述：使用【迭代器】方法版本的颜色空间缩减函数
//-----
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

void colorReduce(Mat& inputImage, Mat& outputImage, int div)
{
    //参数准备
    outputImage = inputImage.clone(); //复制实参到临时变量
    //获取迭代器
    Mat_<Vec3b>::iterator it = outputImage.begin<Vec3b>(); //初始位置
    的迭代器
    Mat_<Vec3b>::iterator itend = outputImage.end<Vec3b>(); //终止位置
    的迭代器

    //存取彩色图像像素
    for(;it != itend;++it)
    {
        // -----【开始处理每个像素】-----
        (*it)[0] = (*it)[0]/div*div + div/2;
        (*it)[1] = (*it)[1]/div*div + div/2;
        (*it)[2] = (*it)[2]/div*div + div/2;
        // -----【处理结束】-----
    }
}

```

实验条件下单次运行时间为 0.242588。

不熟悉面向对象编程中迭代器的概念的读者，可以阅读与 STL 中迭代器相关的入门书籍和文字。用关键字“STL 迭代器”进行搜索可以找到各种相关的博文和资料。

### 3. 【方法三】动态地址计算

第三种方法为用动态地址计算来操作像素。下面是使用动态地址运算配合 at 方法的 colorReduce 函数的代码。这种方法简洁明了，符合大家对像素的直观认识。实验条件下单次运行时间约为 0.334131。

```

//-----【colorReduce() 函数】-----
//      描述：使用【动态地址运算配合 at】方法版本的颜色空间缩减函数
//-----
void colorReduce(Mat& inputImage, Mat& outputImage, int div)
{
    //参数准备
    outputImage = inputImage.clone(); //复制实参到临时变量
    int rowNumber = outputImage.rows; //行数
    int colNumber = outputImage.cols; //列数

    //存取彩色图像像素
    for(int i = 0;i < rowNumber;i++)
    {
        for(int j = 0;j < colNumber;j++)
        {
            // -----【开始处理每个像素】-----
            outputImage.at<Vec3b>(i,j)[0] =
            outputImage.at<Vec3b>(i,j)[0]/div*div + div/2; //蓝色通道
            outputImage.at<Vec3b>(i,j)[1] =

```

## SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

        outputImage.at<Vec3b>(i,j)[1]/div*div + div/2; //绿色通道
        outputImage.at<Vec3b>(i,j)[2] =
    outputImage.at<Vec3b>(i,j)[2]/div*div + div/2; //红色通道
    // -----【处理结束】-----
} // 行处理结束
}
}

```

让我们讲解一下上述的代码。

Mat类中的cols和rows给出了图像的宽和高。而成员函数at(int y, int x)可以用来存取图像元素，但是必须在编译期知道图像的数据类型。需要注意的是，我们一定要确保指定的数据类型要和矩阵中的数据类型相符合，因为at方法本身不会对任何数据类型进行转换。

对于彩色图像，每个像素由三个部分构成：蓝色通道、绿色通道和红色通道(BGR)。因此，对于一个包含彩色图像的Mat，会返回一个由三个8位数组成的向量。OpenCV将此类型的向量定义为Vec3b，即由三个unsigned char组成的向量。这也解释了为什么存取彩色图像像素的代码可以写出如下形式：

```
image.at<Vec3b>(j,i)[channel]=value;
```

其中，索引值channel标明了颜色通道号。

另外需要再次提醒大家的是，OpenCV中的彩色图像不是以RGB的顺序存放的，而是BGR，所以程序中的outputImage.at<Vec3b>(i,j)[0]代表的是该点的B分量。同理还有(\*it)[0]。

### 5.1.6示例程序

本节至此结束。在这里说明一下本节相关的配套示例程序。本节共有四个配套的示例程序，按示例程序序号来排列为：

- 【21】用指针访问像素
- 【22】用迭代器访问像素
- 【23】用动态地址计算配合at访问像素
- 【24】遍历图像像素的14种方法

其中，第21~23分别为5.1.5节中讲到三种方法的示例程序，而第24个示例程序“遍历图像像素的14种方法”为来自一本国外OpenCV2书籍的配套示例程序，其用14个函数分别封装好了14种像素存取方法。比较典型，于是将其注释后提供给大家学习。

## 5.2 ROI区域图像叠加&图像混合

在本节中，我们将一起学习在OpenCV中如何定义感兴趣区域ROI，如何使用addWeighted函数进行图像混合操作，以及如何将ROI和addWeighted函数结

合起来使用，对指定区域进行图像混合操作。

### 5.2.1 感兴趣区域：ROI

在图像处理领域，我们常常需要设置感兴趣区域（ROI, region of interest），来专注或者简化工作过程。也就是从图像中选择的一个图像区域，这个区域是图像分析所关注的重点。我们圈定这个区域，以便进行进一步处理。而且，使用 ROI 指定想读入的目标，可以减少处理时间，增加精度，给图像处理带来不小的便利。

定义 ROI 区域有两种方法：第一种是使用表示矩形区域的 Rect。它指定矩形的左上角坐标（构造函数的前两个参数）和矩形的长宽（构造函数的后两个参数）以定义一个矩形区域。

其中，image 为已经载入好的图片。

```
//定义一个 Mat 类型并给其设定 ROI 区域
Mat imageROI;
//方法一
imageROI=image(Rect(500,250,logo.cols,logo.rows));
```

另一种定义 ROI 的方式是指定感兴趣行或列的范围（Range）。Range 是指从起始索引到终止索引（不包括终止索引）的一段连续序列。cRange 可以用来定义 Range。如果使用 Range 来定义 ROI，那么前例中定义 ROI 的代码可以重写为：

```
//方法二
imageROI=image(Range(250,250+logoImage.rows),Range(200,200+logoImage
.cols));
```

下面我们来看一个实例，显示如何利用 ROI 将一幅图加到另一幅图的指定位置。大家如果需要复制以下函数中的代码直接运行，可以自己建一个基于 console 的程序，然后把函数体中的内容复制到 main 函数中，然后找两幅大小合适的图片，加入到工程目录下，并和代码中读取的文件名一致即可。

在下面的代码中，我们通过一个图像掩膜（mask），直接将插入处的像素设置为 logo 图像的像素值，这样效果会很逼真。

```
-----【 ROI_AddImage() 函数】-----
// 函数名: ROI_AddImage()
// 描述: 利用感兴趣区域 ROI 实现图像叠加
-----
bool ROI_AddImage()
{
    //【1】读入图像
    Mat srcImage1= imread("dota_pa.jpg");
    Mat logoImage= imread("dota_logo.jpg");
    if(!srcImage1.data) { printf("读取 srcImage1 错误~! \n"); return
false; }
    if(!logoImage.data) { printf("读取 logoImage 错误~! \n"); return
false; }
```

```

//【2】定义一个 Mat 类型并给其设定 ROI 区域
Mat imageROI=
srcImage1(Rect(200,250,logoImage.cols,logoImage.rows));

//【3】加载掩模（必须是灰度图）
Mat mask= imread("dota_logo.jpg",0);

//【4】将掩膜复制到 ROI
logoImage.copyTo(imageROI,mask);

//【5】显示结果
namedWindow("<1>利用 ROI 实现图像叠加示例窗口");
imshow("<1>利用 ROI 实现图像叠加示例窗口",srcImage1);

return true;
}

```

这个函数首先是载入了两张 jpg 图片到 srcImage1 和 logoImage 中，然后定义了一个 Mat 类型的 imageROI，并使用 Rect 设置其感兴趣区域为 srcImage1 中的一块区域，将 imageROI 和 srcImage1 关联起来。接着定义了一个 Mat 类型的的 mask 并读入 dota\_logo.jpg，顺势使用 Mat:: copyTo 把 mask 中的内容复制到 imageROI 中，于是就得到了最终的效果图。namedWindow 和 imshow 配合使用，显示出最终的结果。

运行结果如图 5.3 所示。



图 5.3 运行结果

图 5.3 中白色的 dota2 logo，就是通过操作之后加上去的图像。

## 5.2.2 线性混合操作

线性混合操作是一种典型的二元（两个输入）的像素操作，它的理论公式如下：

$$g(x) = (1-a)f_a(x) + af_3(x)$$

我们通过在范围 0 到 1 之间改变 alpha 值，来对两幅图像 ( $f_0(x)$  和  $f_1(x)$ ) 或

两段视频（同样为  $f_0(x)$  和  $f_1(x)$ ）产生时间上的画面叠化（cross-dissolve）效果，就像幻灯片放映和电影制作中的那样，也就是在幻灯片翻页时设置的前后页缓慢过渡叠加效果，以及电影情节过渡时经常出现的画面叠加效果。

实现方面，主要运用了 OpenCV 中 `addWeighted` 函数，下面来一起全面地了解它。

### 5.2.3 计算数组加权和：`addWeighted()` 函数

这个函数的作用是计算两个数组（图像阵列）的加权和。原型如下：

```
void (InputArray src1, double alpha, InputArray src2, double beta, double gamma, OutputArray dst, int dtype=-1);
```

- 第一个参数，`InputArray` 类型的 `src1`，表示需要加权的第一个数组，常常填一个 `Mat`；
- 第二个参数，`double` 类型的 `alpha`，表示第一个数组的权重；
- 第三个参数，`InputArray` 类型的 `src2`，表示第二个数组，它需要和第一个数组拥有相同的尺寸和通道数；
- 第四个参数，`double` 类型的 `beta`，表示第二个数组的权重值；
- 第五个参数，`double` 类型的 `gamma`，一个加到权重总和上的标量值。其含义通过接下来列出的式子自然会理解；
- 第六个参数，`OutputArray` 类型的 `dst`，输出的数组，它和输入的两个数组拥有相同的尺寸和通道数；
- 第七个参数，`int` 类型的 `dtype`，输出阵列的可选深度，有默认值-1。当两个输入数组具有相同的深度时，这个参数设置为-1（默认值），即等同于 `src1.depth()`。

下面的数学公式表示：用 `addWeighted` 函数计算以下两个数组（`src1` 和 `src2`）的加权和，得到结果输出给第四个参数，也就是 `addWeighted` 函数的作用的矩阵表达式。

```
dst = src1[I]*alpha+ src2[I]*beta + gamma;
```

其中 `I` 是多维数组元素的索引值。而且，在遇到多通道数组的时候，每个通道都需要独立地进行处理。另外需要注意的是，当输出数组的深度为 `CV_32S` 时，这个函数就不适用了，这时候就会内存溢出或者算出的结果压根不对。

理论和函数的讲解就是上面这些，接着我们来看代码实例，以融会贯通。

```
-----【LinearBlending() 函数】-----  
// 函数名: LinearBlending()  
// 描述: 利用 cv::addWeighted() 函数实现图像线性混合  
-----  
bool LinearBlending()  
{  
    //【0】定义一些局部变量  
    double alphaValue = 0.5;  
    double betaValue;
```

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

    Mat srcImage2, srcImage3, dstImage;

    //【1】读取图像（两幅图片需为同样的类型和尺寸）
    srcImage2= imread("mogu.jpg");
    srcImage3= imread("rain.jpg");

    if(!srcImage2.data) { printf("读取 srcImage2 错误~! \n"); return
false; }
    if(!srcImage3.data) { printf("读取 srcImage3 错误~! \n"); return
false; }

    //【2】做图像混合加权操作
    betaValue= ( 1.0 - alphaValue );
    addWeighted(srcImage2, alphaValue, srcImage3, betaValue, 0.0,
dstImage);

    //【3】创建并显示原图窗口
    namedWindow("<2>线性混合示例窗口【原图】", 1);
    imshow("<2>线性混合示例窗口【原图】", srcImage2 );

    namedWindow("<3>线性混合示例窗口【效果图】", 1);
    imshow("<3>线性混合示例窗口【效果图】", dstImage );

    return true;
}

```

下面对以上代码进行解析。

(0) 首先当然是定义一些局部变量、alpha 值、beta 值，以及三个 Mat 类型的变量

```

//【0】定义一些局部变量
double alphaValue = 0.5;
double betaValue;
Mat srcImage2, srcImage3, dstImage;

```

在这里我们设置 alpha 值为 0.5。

(1) 读取两幅图像并作错误处理

这步很简单，直接看代码。

```

//读取图像（两幅图片需为同样的类型和尺寸）
srcImage2= imread("mogu.jpg");
srcImage3= imread("rain.jpg");
if(!srcImage2.data) { printf("读取 srcImage2 错误~! \n"); return false; }
if(!srcImage3.data) { printf("读取 srcImage3 错误~! \n"); return
false; }

```

在这里需要注意的是，因为我们是对 srcImage1 和 srcImage2 求和，所以它们必须要有相同的尺寸（宽度和高度）和类型，不然多余的部分没有对应的“伴”，肯定会出现问题。

## (2) 进行图像混合加权操作

载入图像后，我们就可以来生成混合图像，也就是之前公式中的  $g(x)$ 。为此目的，使用函数 `addWeighted` 可以很方便地实现。代码其实很简单，就是这样：

```
//【2】进行图像混合加权操作  
betaValue = ( 1.0 - alphaValue );  
addWeighted( srcImage2, alphaValue, srcImage3, betaValue, 0.0,  
dstImage );  
其中对应于 addWeighted 的参数中的 beta 值为 1-alpha, gamma 值为 0。
```

## (3) 创建显示窗口，显示图像

```
//【3】创建并显示原图窗口  
namedWindow ("<2>线性混合示例窗口【原图】", 1);  
imshow ("<2>线性混合示例窗口【原图】", srcImage2 );  
  
namedWindow ("<3>线性混合示例窗口【效果图】", 1);  
imshow ("<3>线性混合示例窗口【效果图】", dstImage );
```

接着来看一下运行效果图，首先是原图（图 5.4）。



图 5.4 原始图

然后是效果图，如图 5.5 所示。



图 5.5 线性混合效果图

### 5.2.4 综合示例：初级图像混合

在前文介绍的设定感兴趣区域 ROI 和使用 addWeighted 函数进行图像线性混合的基础上，我们还可以将二者结合起来使用，也就是先指定 ROI，再用 addWeighted 函数对指定的 ROI 区域的图像进行混合操作。我们将其封装在了一个名为 ROI\_LinearBlending 的函数中，方便大家分块学习。代码如下。

```
-----【ROI_LinearBlending()】-----
// 函数名: ROI_LinearBlending()
// 描述:线性混合实现函数,指定区域线性图像混合.利用 cv::addWeighted() 函数结合定义
//          感兴趣区域 ROI, 实现自定义区域的线性混合
-----
bool ROI_LinearBlending()
{
    //【1】读取图像
    Mat srcImage4= imread("data_pa.jpg",1);
    Mat logoImage= imread("data_logo.jpg");

    if(!srcImage4.data) { printf("读取 srcImage4 错误~! \n"); return
false; }
    if(!logoImage.data) { printf("读取 logoImage 错误~! \n"); return
false; }

    //【2】定义一个 Mat 类型并给其设定 ROI 区域
    Mat imageROI;
    //方法一

    imageROI=srcImage4(Rect(200,250,logoImage.cols,logoImage.rows));
    //方法二

    //imageROI=srcImage4(Range(250,250+logoImage.rows),Range(200,200+log
oImage.cols));

    //【3】将 logo 加到原图上
    addWeighted(imageROI,0.5,logoImage,0.3,0,imageROI);

    //【4】显示结果
    namedWindow("<4>区域线性图像混合示例窗口");
    imshow("<4>区域线性图像混合示例窗口",srcImage4);

    return true;
}
```

下面放出详细注释的本节示例源代码，示例代码都将封装在了不同的函数中，具体如下。

```
-----【头文件、命名空间包含部分】-----
//      描述: 包含程序所依赖的头文件和命名空间
-----
#include <opencv2/opencv.hpp>
```

```
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
using namespace cv;
using namespace std;

//-----【全局函数声明部分】-----
//    描述：全局函数声明
//-----
bool ROI_AddImage();
bool LinearBlending();
bool ROI_LinearBlending();

//-----【main()函数】-----
//    描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----
int main()
{
    system("color 5E");

    if(ROI_AddImage() && LinearBlending() && ROI_LinearBlending())
    {
        cout<<endl<<"运行成功，得出了你需要的图像~！：)";
    }

    waitKey(0);
    return 0;
}

//-----【ROI_AddImage() 函数】-----
// 函数名：ROI_AddImage()
//    描述：利用感兴趣区域 ROI 实现图像叠加
//-----
bool ROI_AddImage()
{
    //【1】读入图像
    Mat srcImage1= imread("dota_pa.jpg");
    Mat logoImage= imread("dota_logo.jpg");
    if(!srcImage1.data) { printf("读取 srcImage1 错误~！\n"); return
false; }
    if(!logoImage.data) { printf("读取 logoImage 错误~！\n"); return
false; }

    //【2】定义一个 Mat 类型并给其设定 ROI 区域
    Mat imageROI=
srcImage1(Rect(200,250,logoImage.cols,logoImage.rows));

    //【3】加载掩模（必须是灰度图）
    Mat mask= imread("dota_logo.jpg",0);

    //【4】将掩膜复制到 ROI
```

```
logoImage.copyTo(imageROI,mask);

//【5】显示结果
namedWindow("<1>利用 ROI 实现图像叠加示例窗口");
imshow("<1>利用 ROI 实现图像叠加示例窗口",srcImage1);

return true;
}

-----【LinearBlending() 函数】-----
// 函数名: LinearBlending()
// 描述: 利用 cv::addWeighted() 函数实现图像线性混合
-----
bool LinearBlending()
{
    //【0】定义一些局部变量
    double alphaValue = 0.5;
    double betaValue;
    Mat srcImage2, srcImage3, dstImage;

    //【1】读取图像 ( 两幅图片需为同样的类型和尺寸 )
    srcImage2= imread("mogu.jpg");
    srcImage3= imread("rain.jpg");

    if(!srcImage2.data) { printf("读取 srcImage2 错误~! \n"); return false; }
    if(!srcImage3.data) { printf("读取 srcImage3 错误~! \n"); return false; }

    //【2】进行图像混合加权操作
    betaValue= ( 1.0 - alphaValue );
    addWeighted(srcImage2, alphaValue, srcImage3, betaValue, 0.0,
dstImage);

    //【3】创建并显示原图窗口
    namedWindow("<2>线性混合示例窗口【原图】", 1);
    imshow("<2>线性混合示例窗口【原图】", srcImage2 );

    namedWindow("<3>线性混合示例窗口【效果图】", 1);
    imshow("<3>线性混合示例窗口【效果图】", dstImage );

    return true;
}

-----【ROI_LinearBlending()】-----
// 函数名: ROI_LinearBlending()
// 描述: 线性混合实现函数, 指定区域线性图像混合. 利用 cv::addWeighted() 函数结合定义
//          感兴趣区域 ROI, 实现自定义区域的线性混合
//-----
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
bool ROI_LinearBlending()
{
    //【1】读取图像
    Mat srcImage4= imread("dota_pa.jpg",1);
    Mat logoImage= imread("dota_logo.jpg");

    if(!srcImage4.data ) { printf("读取 srcImage4 错误~! \n"); return
false; }
    if(!logoImage.data ) { printf("读取 logoImage 错误~! \n"); return
false; }

    //【2】定义一个 Mat 类型并给其设定 ROI 区域
    Mat imageROI;
    //方法一

    imageROI=srcImage4(Rect(200,250,logoImage.cols,logoImage.rows));
    //方法二

    //imageROI=srcImage4(Range(250,250+logoImage.rows),Range(200,200+log
oImage.cols));

    //【3】将 logo 加到原图上
    addWeighted(imageROI,0.5,logoImage,0.3,0.,imageROI);

    //【4】显示结果
    namedWindow("<4>区域线性图像混合示例窗口");
    imshow("<4>区域线性图像混合示例窗口",srcImage4);

    return true;
}
```

最后看一下整体的运行效果，如图 5.6、图 5.7、图 5.8、图 5.9 所示。



图 5.6 图像叠加效果图

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>



图 5.7 区域线性混合效果图



图 5.8 线性混合原始图



图 5.9 线性混合效果图

## 5.3 分离颜色通道、多通道图像混合

上节中我们讲解了如何使用 `addWeighted` 函数进行图像混合操作，以及如何将 ROI 和 `addWeighted` 函数结合起来，对指定区域进行图像混合操作。

而为了更好地观察一些图像材料的特征，有时需要对 RGB 三个颜色通道的分量进行分别显示和调整。通过 OpenCV 的 `split` 和 `merge` 方法可以很方便地达到目的。

这一节，我们会详细介绍这两个互为“冤家”的函数。首先来看看进行通道分离的 `split` 函数。

### 5.3.1 通道分离：`split()` 函数

`split` 函数用于将一个多通道数组分离成几个单通道数组。这里的 `array` 按语境翻译为数组或者阵列。

这个 `split` 函数的 C++ 版本有两个原型，分别是：

- C++: `void split(const Mat& src, Mat* mvbegin);`
- C++: `void split(InputArray m, OutputArrayOfArrays mv);`

变量介绍如下：

- 第一个参数，`InputArray` 类型的 `m` 或者 `const Mat&` 类型的 `src`，填我们需要进行分离的多通道数组。
- 第二个参数，`OutputArrayOfArrays` 类型的 `mv`，填函数的输出数组或者输出的 `vector` 容器。

`split` 函数分割多通道数组转换成独立的单通道数组，公式如下：

$$mv[c](I) = src(I)_c$$

最后我们一起看一个示例。

```
Mat srcImage;
Mat imageROI;
vector<Mat> channels;
srcImage= cv::imread("dota.jpg");

// 把一个 3 通道图像转换成 3 个单通道图像
split(srcImage,channels); // 分离色彩通道
    imageROI=channels.at(0);

addWeighted(imageROI(Rect(385, 250, logoImage.cols, logoImage.rows)), 1.0,
            logoImage, 0.5, 0., imageROI(Rect(385, 250, logoImage.cols, logoImage.rows)));
}

merge(channels, srcImage4);
```

```

namedWindow("sample");
imshow("sample",srcImage);

```

将一个多通道数组分离成几个单通道数组的 `split()` 函数的内容大概就是以上这些了，下面我们来看一下和它关系密切的 `merge()` 函数。

### 5.3.2 通道合并：`merge()` 函数

`merge()` 函数是 `split()` 函数的逆向操作——将多个数组合并成一个多通道的数组。它通过组合一些给定的单通道数组，将这些孤立的单通道数组合并成一个多通道的数组，从而创建出一个由多个单通道阵列组成的多通道阵列。它有两个基于 C++ 的函数原型如下。

- C++: `void merge(const Mat* mv, size_t count, OutputArray dst)`
- C++: `void merge(InputArrayOfArrays mv,OutputArray dst)`

变量介绍如下。

- 第一个参数，`mv`。填需要被合并的输入矩阵或 `vector` 容器的阵列，这个 `mv` 参数中所有的矩阵必须有着一样的尺寸和深度。
- 第二个参数，`count`。当 `mv` 为一个空白的 C 数组时，代表输入矩阵的个数，这个参数显然必须大于 1。
- 第三个参数，`dst`。即输出矩阵，和 `mv[0]` 拥有一样的尺寸和深度，并且通道的数量是矩阵阵列中的通道的总数。

函数解析如下。

`merge` 函数的功能是将一些数组合并成一个多通道的数组。关于组合的细节，输出矩阵中的每个元素都将是输出数组的串接。其中，第  $i$  个输入数组的元素被视为 `mv[i]`。C 一般用其中的 `Mat::at()` 方法对某个通道进行存取，也就是这样用：`channels.at(0)`。

这里的 `Mat::at()` 方法返回一个引用到指定的数组元素。注意是引用，相当于两者等价，也就是修改其中一个，另一个也会随之改变。

依然是一个示例，如下。

```

vector<Mat> channels;
Mat imageBlueChannel;
Mat imageGreenChannel;
Mat imageRedChannel;
srcImage4= imread("dota.jpg");
// 把一个3通道图像转换成3个单通道图像
split(srcImage4,channels); //分离色彩通道
imageBlueChannel = channels.at(0);
imageGreenChannel = channels.at(1);
imageRedChannel = channels.at(2);

```

上面的代码先做了相关的类型声明，然后把载入的 3 通道图像转换成 3 个单

通道图像，放到 vector<Mat>类型的 channels 中，接着进行引用赋值。

根据 OpenCV 的 BGR 色彩空间(Bule、Green、Red, 蓝绿红)，其中 channels.at(0) 就表示引用取出 channels 中的蓝色分量，channels.at(1)就表示引用取出 channels 中的绿色分量，channels.at(2)就表示引用取出 channels 中的红色分量。

一对做相反操作的 split()函数和 merge()函数的用法就是这些。另外提一点，如果我们需要从多通道数组中提取出特定的单通道数组，或者说实现一些复杂的通道组合，可以使用 mixChannels()函数。

### 5.3.3 示例程序：多通道图像混合

在本小节展示的示例程序中，我们把多通道图像混合的实现代码封装在了名为 MultiChannelBlending()的函数中。详细注释的代码如下。

```
-----【头文件、命名空间包含部分】-----
// 描述：包含程序所依赖的头文件和命名空间
//-----  
#include <opencv2/opencv.hpp>
#include <iostream>
using namespace cv;
using namespace std;  
  
-----【全局函数声明部分】-----
// 描述：全局函数声明
//-----  
bool ROI_AddImage();
bool LinearBlending();
bool ROI_LinearBlending();  
  
-----【main() 函数】-----
// 描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----  
int main()
{
    system("color 5E");

    if(ROI_AddImage() && LinearBlending() && ROI_LinearBlending())
    {
        cout<<endl<<"运行成功，得出了你需要的图像~！：)" ;
    }

    waitKey(0);
    return 0;
}  
  
-----【ROI_AddImage() 函数】-----
// 函数名：ROI_AddImage()
// 描述：利用感兴趣区域 ROI 实现图像叠加
//-----
```

```
bool ROI_AddImage()
{
    // 【1】读入图像
    Mat srcImage1= imread("dota_pa.jpg");
    Mat logoImage= imread("dota_logo.jpg");
    if( !srcImage1.data ) { printf("读取图片错误, 请确定目录下是否有 imread\n函数指定图片存在~! \n"); return false; }
    if( !logoImage.data ) { printf("读取图片错误, 请确定目录下是否有 imread\n函数指定图片存在~! \n"); return false; }

    // 【2】定义一个 Mat 类型并给其设定 ROI 区域
    Mat imageROI=
srcImage1(Rect(200,250,logoImage.cols,logoImage.rows));

    // 【3】加载掩膜（必须是灰度图）
    Mat mask= imread("dota_logo.jpg",0);

    //【4】将掩膜复制到 ROI
    logoImage.copyTo(imageROI,mask);

    // 【5】显示结果
    namedWindow("<1>利用 ROI 实现图像叠加示例窗口");
    imshow("<1>利用 ROI 实现图像叠加示例窗口",srcImage1);

    return true;
}

//-----【LinearBlending() 函数】-----
// 函数名: LinearBlending()
// 描述: 利用 cv::addWeighted() 函数实现图像线性混合
//-----
bool LinearBlending()
{
    //【0】定义一些局部变量
    double alphaValue = 0.5;
    double betaValue;
    Mat srcImage2, srcImage3, dstImage;

    // 【1】读取图像（两幅图片需为同样的类型和尺寸）
    srcImage2 = imread("mogu.jpg");
    srcImage3 = imread("rain.jpg");

    if( !srcImage2.data ) { printf("读取图片错误, 请确定目录下是否有 imread\n函数指定图片存在~! \n"); return false; }
    if( !srcImage3.data ) { printf("读取图片错误, 请确定目录下是否有 imread\n函数指定图片存在~! \n"); return false; }

    // 【2】进行图像混合加权操作
    betaValue = ( 1.0 - alphaValue );
    addWeighted( srcImage2, alphaValue, srcImage3, betaValue, 0.0,
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
dstImage);

// 【3】创建并显示原图窗口
namedWindow("＜2＞线性混合示例窗口【原图】", 1);
imshow( "＜2＞线性混合示例窗口【原图】", srcImage2 );

namedWindow("＜3＞线性混合示例窗口【效果图】", 1);
imshow( "＜3＞线性混合示例窗口【效果图】", dstImage );

return true;

}

-----【ROI_LinearBlending()】-----
// 函数名: ROI_LinearBlending()
// 描述:线性混合实现函数,指定区域线性图像混合.利用 cv::addWeighted() 函数结合定义
//          感兴趣区域 ROI, 实现自定义区域的线性混合
-----

bool ROI_LinearBlending()
{

    //【1】读取图像
    Mat srcImage4= imread("dota_pa.jpg",1);
    Mat logoImage= imread("dota_logo.jpg");

    if( !srcImage4.data ) { printf("读取图片错误, 请确定目录下是否有 imread\n函数指定图片存在~! \n"); return false; }
    if( !logoImage.data ) { printf("读取图片错误, 请确定目录下是否有 imread\n函数指定图片存在~! \n"); return false; }

    //【2】定义一个 Mat 类型并给其设定 ROI 区域
    Mat imageROI;
    //方法一
    imageROI= srcImage4(Rect(200,250,logoImage.cols,logoImage.rows));
    //方法二
    //imageROI=
    srcImage4(Range(250,250+logoImage.rows),Range(200,200+logoImage.cols));
}

    //【3】将 logo 加到原图上
    addWeighted(imageROI,0.5,logoImage,0.3,0.,imageROI);

    //【4】显示结果
    namedWindow("＜4＞区域线性图像混合示例窗口");
    imshow("＜4＞区域线性图像混合示例窗口",srcImage4);

    return true;
}
```

可以发现, 其实多通道混合的实现函数中的代码大体可分成三部分, 分别对蓝绿红三个通道进行处理, 唯一不同的地方在于取通道分量时取的是 channels.at(0)、

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

channels.at(1)还是 channels.at(2)。

运行截图如图 5.10、图 5.11、图 5.12 所示，具体区别表现在箭头指向处。



图 5.10 游戏原画+logo 蓝色通道



图 5.11 游戏原画+logo 绿色通道



图 5.12 游戏原画+logo 红色通道

## 5.4 图像对比度、亮度值调整

本节我们将学习如何用 OpenCV 进行图像对比度和亮度值的动态调整。

### 5.4.1 理论依据

首先了解一下算子的概念。一般的图像处理算子都是一个函数，它接受一个或多个输入图像，并产生输出图像。下面是算子的一般形式。

$$g(x) = h(f(x)) \text{ 或者 } g(x) = h(f_0(x), \dots, f_n(x))$$

本节所讲解的图像亮度和对比度的调整操作，其实属于图像处理变换中比较简单的一种——点操作（point operators）。点操作有一个特点：仅仅根据输入像素值（有时可加上某些全局信息或参数），来计算相应的输出像素值。这类算子包括亮度（brightness）和对比度（contrast）调整、颜色校正（color correction）和变换（transformations）。

两种最常用的点操作（或者说点算子）是乘上一个常数（对对比度的调节）以及加上一个常数（对应亮度值的调节）。公式如下：

$$g(x) = a * f(x) + b$$

看到这个式子，我们关于图像亮度和对比度调整的策略就比较好理解了。

其中：

- 参数  $f(x)$  表示源图像像素。
- 参数  $g(x)$  表示输出图像像素。
- 参数  $a$ （需要满足  $a > 0$ ）被称为增益（gain），常常被用来控制图像的对比度。
- 参数  $b$  通常被称为偏置（bias），常常被用来控制图像的亮度。

而更进一步，我们这样改写这个式子：

$$g(i,j) = a * f(i,j) + b$$

其中， $i$  和  $j$  表示像素位于第  $i$  行和第  $j$  列，这个式子可以用来作为我们在 OpenCV 中控制图像的亮度和对比度的理论公式。

### 5.4.2 访问图片中的像素

访问图片中的像素有很多种方式，在本书 5.1 节“访问图像中的像素”中已有过比较系统的讲解。

而为了执行如下运算：

$$g(i,j) = a * f(i,j) + b$$

我们需要访问图像的每一个像素。因为是对 GBR 图像进行运算，每个像素有三个值（G、B、R），所以我们必须分别访问它们（OpenCV 中的图像存储模式为

GBR)。以下是访问像素的代码片段，使用了三个 for 循环。

```
//三个for循环，执行运算 new_image(i,j) = a*image(i,j) + b
for(int y = 0; y < image.rows; y++ )
{
    for(int x = 0; x < image.cols; x++ )
    {
        for(int c = 0; c < 3; c++ )
        {
            new_image.at<Vec3b>(y,x)[c] =
saturate_cast<uchar>( (g_nContrastValue*0.01)*(image.at<Vec3b>(y,x)[
c] ) + g_nBrightValue );
        }
    }
}
```

让我们分三个方面进行讲解。

- 为了访问图像的每一个像素，使用这样的语法：image.at<Vec3b>(y,x)[c]。

其中， $y$  是像素所在的行， $x$  是像素所在的列， $c$  是 R、G、B（对应 0、1、2）其中之一。

- 因为运算结果可能会超出像素取值范围（溢出），还可能是非整数（如果是浮点数的话），所以要用 `saturate_cast` 对结果进行转换，以确保它为有效值。
- 这里的  $a$  也就是对比度，一般为了观察的效果，它的取值为 0.0 到 3.0 的浮点值，但是轨迹条一般取值都会取整数，因此在这里我们可以将其代表对比度值的 `nContrastValue` 参数设为 0 到 300 之间的整型，在最后的式子中乘以一个 0.01，这样就完成了轨迹条中 300 个不同取值的变化。这就是为什么在式子中，会有 `saturate_cast<uchar>((g_nContrastValue*0.01)*(image.at<Vec3b>(y,x)[c]) + g_nBrightValue)` 中的 `g_nContrastValue*0.01`。

### 5.4.3 示例程序：图像对比度、亮度值调整

本小节依然是一个详细注释的配套示例程序，把本节前文介绍的知识点以代码为载体，展现给大家。

此示例程序用两个轨迹条分别控制对比度和亮度值，详细注释的示例程序代码如下。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
-----
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
using namespace std;
using namespace cv;

-----【全局函数声明部分】-----
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
//      描述：全局函数声明
//-----
static void on_ContrastAndBright(int, void *);
static void ShowHelpText();

//-----【全局变量声明部分】-----
//      描述：全局变量声明
//-----
int g_nContrastValue; //对比度值
int g_nBrightValue; //亮度值
Mat g_srcImage,g_dstImage;
//-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----
int main()
{
    // 【1】读取输入图像
    g_srcImage = imread( "1.jpg");
    if( !g_srcImage.data ) { printf("读取图片错误，请确定目录下是否有
imread函数指定图片存在~!"); return false; }
    g_dstImage = Mat::zeros( g_srcImage.size(), g_srcImage.type() );

    // 【2】设定对比度和亮度的初值
    g_nContrastValue=80;
    g_nBrightValue=80;

    // 【3】创建效果图窗口
    namedWindow("【效果图窗口】", 1);

    // 【4】创建轨迹条
    createTrackbar("对比度: ", "【效果图窗口】",&g_nContrastValue,
300,on_ContrastAndBright );
    createTrackbar("亮度: ", "【效果图窗口】",&g_nBrightValue,
200,on_ContrastAndBright );

    // 【5】进行回调函数初始化
    on_ContrastAndBright(g_nContrastValue,0);
    on_ContrastAndBright(g_nBrightValue,0);

    // 【6】按下"q"键时，程序退出
    while(char(waitKey(1)) != 'q') {}
    return 0;
}

//-----【on_ContrastAndBright()函数】-----
//      描述：改变图像对比度和亮度值的回调函数
//-----
static void on_ContrastAndBright(int, void *)
{
    // 创建窗口
    namedWindow("【原始图窗口】", 1);
```

```

// 三个 for 循环，执行运算 g_dstImage(i,j) = a*g_srcImage(i,j) + b
for( int y = 0; y < g_srcImage.rows; y++ )
{
    for( int x = 0; x < g_srcImage.cols; x++ )
    {
        for( int c = 0; c < 3; c++ )
        {
            g_dstImage.at<Vec3b>(y,x)[c] =
saturate_cast<uchar>( (g_nContrastValue*0.01)*( g_srcImage.at<Vec3b>
(y,x)[c] ) + g_nBrightValue );
        }
    }
}

// 显示图像
imshow("【原始图窗口】", g_srcImage);
imshow("【效果图窗口】", g_dstImage);
}

```

讲解一下上述代码中的 `saturate_cast` 模板函数，其用于溢出保护，大致原理如下。

```

if(data<0)
    data=0;
else if(data>255)
    data=255;

```

最后看一下程序的运行截图。运行这个程序会得到两个图片显示窗口：第一个为原图窗口（图 5.13），第二个为效果图窗口（图 5.14）。在效果图窗口中可以调节两个轨迹条，来改变当前图片的对比度和亮度。



图 5.13 原始图



图 5.14 可调节对比度和亮度的效果图

## 5.5 离散傅里叶变换

离散傅里叶变换（Discrete Fourier Transform，缩写为 DFT），是指傅里叶变换在时域和频域上都呈现离散的形式，将时域信号的采样变换为在离散时间傅里叶变换（DTFT）频域的采样。在形式上，变换两端（时域和频域上）的序列是有限长的，而实际上这两组序列都应当被认为是离散周期信号的主值序列。即使对有限长的离散信号做 DFT，也应当对其经过周期延拓成为周期信号再进行变换。在实际应用中，通常采用快速傅里叶变换来高效计算 DFT。

### 5.5.1 离散傅里叶变换的原理

简单来说，对一张图像使用傅里叶变换就是将它分解成正弦和余弦两部分，也就是将图像从空间域（spatial domain）转换到频域（frequency domain）。

这一转换的理论基础为：任一函数都可以表示成无数个正弦和余弦函数的和的形式。傅里叶变换就是一个用来将函数分解的工具。

二维图像的傅里叶变换可以用以下数学公式表达。

$$F(k, l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i, j) e^{-i2\pi(\frac{ki}{N} + \frac{l}{N})}$$
$$e^{ix} = \cos x + i \sin x$$

式中  $f$  是空间域（spatial domain）值， $F$  是频域（frequency domain）值。转换之后的频域值是复数，因此，显示傅里叶变换之后的结果需要使用实数图像（real image）加虚数图像（complex image），或者幅度图像（magnitude image）加相位图像（phase image）的形式。在实际的图像处理过程中，仅仅使用了幅度图像，因为幅度图像包含了原图像的几乎所有我们需要的几何信息。然而，如果想通过修改幅度图像或者相位图像的方法来间接修改原空间图像，需要使用逆傅里叶变换得到修改后的空间图像，这样就必须同时保留幅度图像和相位图像了。

在此示例中，我们将展示如何计算以及显示傅里叶变换后的幅度图像。由于数字图像的离散性，像素值的取值范围也是有限的。比如在一张灰度图像中，像素灰度值一般在 0 到 255 之间。因此，我们这里讨论的也仅仅是离散傅里叶变换（DFT）。如果需要得到图像中的几何结构信息，那你就要用到它了。

在频域里面，对于一幅图像，高频部分代表了图像的细节、纹理信息；低频部分代表了图像的轮廓信息。如果对一幅精细的图像使用低通滤波器，那么滤波后的结果就只剩下轮廓了。这与信号处理的基本思想是相通的。如果图像受到的噪声恰好位于某个特定的“频率”范围内，则可以通过滤波器来恢复原来的图像。傅里叶变换在图像处理中可以做到图像增强与图像去噪、图像分割之边缘检测、图像特征提取、图像压缩等。

### 5.5.2 dft()函数详解

dft 函数的作用是对一维或二维浮点数数组进行正向或反向离散傅里叶变换。

```
C++: void dft(InputArray src, OutputArray dst, int flags=0, int nonzeroRows=0)
```

- 第一个参数，InputArray 类型的 src。输入矩阵，可以为实数或者虚数。
- 第二个参数，OutputArray 类型的 dst。函数调用后的运算结果存在这里，其尺寸和类型取决于标识符，也就是第三个参数 flags。
- 第三个参数，int 类型的 flags。转换的标识符，有默认值 0，取值可以为表 6.1 中标识符的结合。

表 6.1 dft 标识符取值列表

标识符名称	意义
DFT_INVERSE	用一维或二维逆变换代替默认的正向变换
DFT_SCALE	缩放比例标识符，输出的结果都会以 $1/N$ 进行缩放，通常会结合 DFT_INVERSE 一起使用
DFT_ROWS	对输入矩阵的每行进行正向或反向的变换，此标识符可以在处理多种矢量的时候用于减小资源开销，这些处理常常是三维或高维变换等复杂操作
DFT_COMPLEX_OUTPUT	进行一维或二维实数数组正变换。这样的结果虽然是复数阵列，但拥有复数的共轭对称性 (CCS)，所以可以被写成一个拥有同样尺寸的实数阵列
DFT_REAL_OUTPUT	进行一维或二维复数数组反变换。这样的结果通常是一个大小相同的复矩阵。如果输入的矩阵有复数的共轭对称性（比如是一个带有 DFT_COMPLEX_OUTPUT 标识符的正变换结果），便会输出实矩阵

- 第四个参数，int 类型的 nonzeroRows，有默认值 0。当此参数设为非零时（最好是取值为想要处理的那一行的值，比如 C.rows），函数会假设只有输入矩阵的第一个非零行包含非零元素（没有设置 DFT\_INVERSE 标识符），或只有输出矩阵的第一个非零行包含非零元素（设置了 DFT\_INVERSE 标识符）。这样的话，函数就可对其他行进行更高效的处理，以节省时间开销。这项技术尤其是在采用 DFT 计算矩阵卷积时非常有效。

讲解完函数的参数含义，下面我们看一个用 dft 函数计算两个二维实矩阵卷积的示例核心片段。

```
void convolveDFT(InputArray A, InputArray B, OutputArray C)
{
    //【1】初始化输出矩阵
    C.create(abs(A.rows - B.rows)+1, abs(A.cols - B.cols)+1, A.type());
    Size dftSize;

    //【2】计算 DFT 变换的尺寸
    dftSize.width = getOptimalDFTSize(A.cols + B.cols - 1);
```

```

dftSize.height = getOptimalDFTSize(A.rows + B.rows - 1);

//【3】分配临时缓冲区并初始化置零
Mat tempA(dftSize, A.type(), Scalar::all(0));
Mat tempB(dftSize, B.type(), Scalar::all(0));

//【4】分别复制 A 和 B 到 tempA 和 tempB 的左上角
Mat roiA(tempA, Rect(0, 0, A.cols, A.rows));
A.copyTo(roiA);
Mat roiB(tempB, Rect(0, 0, B.cols, B.rows));
B.copyTo(roiB);

//【5】就地操作 (in-place), 进行快速傅里叶变换, 并将 nonzeroRows 参数置为非零,
以进行更加快速地处理
dft(tempA, tempA, 0, A.rows);
dft(tempB, tempB, 0, B.rows);

//【6】将得到的频谱相乘, 结果存放于 tempA 中
mulSpectrums(tempA, tempB, tempA);

//【7】将结果变换为频域, 且尽管结果行 (result rows) 都为非零, 我们只需要其中的
C.rows 的第一行, 所以采用 nonzeroRows == C.rows
dft(tempA, tempA, DFT_INVERSE + DFT_SCALE, C.rows);

//【8】将结果复制到 C 中
tempA(Rect(0, 0, C.cols, C.rows)).copyTo(C);

//所有的临时缓冲区将被自动释放, 所以无须收尾操作
}

```

此示例中的注释已经十分详尽。其中出现了新的函数 `MulSpectrums`, 它的作用是计算两个傅里叶频谱的每个元素的乘法, 前两个参数为输入的参加乘法运算的两个矩阵, 第三个参数为得到的乘法结果矩阵。

由于 5.5.8 节会放出一个使用了不少新函数的示例程序, 下面我们先将这些“新面孔”各个击破。

### 5.5.3 返回 DFT 最优尺寸大小: `getOptimalDFTSize()` 函数

`getOptimalDFTSize` 函数返回给定向量尺寸的傅里叶最优尺寸大小。为了提高离散傅里叶变换的运行速度, 需要扩充图像, 而具体扩充多少, 就由这个函数来计算得到。

```
C++: int getOptimalDFTSize(int vecsize)
```

此函数的唯一一个参数为 `int` 类型的 `vecsize`, 向量尺寸, 即图片的 `rows`、`cols`。

### 5.5.4 扩充图像边界: `copyMakeBorder()` 函数

`copyMakeBorder` 函数的作用是扩充图像边界。

```
C++: void copyMakeBorder(InputArray src, OutputArray dst, int top, int
```

```
bottom, int left, int right, int borderType, const Scalar&
value=Scalar() )
```

- 第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可。
- 第二个参数，OutputArray 类型的 dst，函数调用后的运算结果存在这里，即这个参数用于存放函数调用后的输出结果，需和源图片有一样的尺寸和类型，且 size 应该为 Size (src.cols+left+right, src.rows+top+bottom)。
- 接下来的 4 个参数分别为 int 类型的 top、bottom、left、right，分别表示在源图像的四个方向上扩充多少像素，例如 top=2, bottom=2, left=2, right=2 就意味着在源图像的上下左右各扩充两个像素宽度的边界。
- 第七个参数，borderType 类型的，边界类型，常见取值为 BORDER\_CONSTANT，可参考 borderInterpolate() 得到更多的细节。
- 第八个参数，const Scalar&类型的 value，有默认值 Scalar()，可以理解为默认值为 0。当 borderType 取值为 BORDER\_CONSTANT 时，这个参数表示边界值。

### 5.5.5 计算二维矢量的幅值：magnitude()函数

magnitude() 函数用于计算二维矢量的幅值。

```
C++: void magnitude(InputArray x, InputArray y, OutputArray magnitude)
```

- 第一个参数，InputArray 类型的 x，表示矢量的浮点型 X 坐标值，也就是实部。
- 第一个参数，InputArray 类型的 y，表示矢量的浮点型 Y 坐标值，也就是虚部。
- 第三次参数，OutputArray 类型的 magnitude，输出的幅值，它和第一个参数 x 有着同样的尺寸和类型。

下式可以表示 magnitude() 函数的原理：

$$dst(I) = \sqrt{x(I)^2 + y(I)^2}$$

### 5.5.6 计算自然对数：log()函数

log() 函数的功能是计算每个数组元素绝对值的自然对数。

```
C++: void log(InputArray src, OutputArray dst)
```

第一个参数为输入图像，第二个参数为得到的对数值。其原理如下。

$$dst(I) = \begin{cases} \log|src(I)| & \text{if } src(I) \neq 0 \\ C & \text{otherwise} \end{cases}$$

### 5.5.7 矩阵归一化：normalize()函数

normalize() 的作用是进行矩阵归一化。

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
C++: void normalize(InputArray src, OutputArray dst, double alpha=1,
double beta=0, int norm_type=NORM_L2, int dtype=-1, InputArray
mask=noArray() )
```

- 第一个参数, InputArray 类型的 src。输入图像, 即源图像, 填 Mat 类的对象即可。
- 第二个参数, OutputArray 类型的 dst。函数调用后的运算结果存在这里, 和源图片有一样的尺寸和类型。
- 第三个参数, double 类型的 alpha。归一化后的最大值, 有默认值 1。
- 第四个参数, double 类型的 beta。归一化后的最大值, 有默认值 0。
- 第五个参数, int 类型的 norm\_type。归一化类型, 有 NORM\_INF、NORM\_L1、NORM\_L2 和 NORM\_MINMAX 等参数可选, 有默认值 NORM\_L2。
- 第六个参数, int 类型的 dtype, 有默认值-1。当此参数取负值时, 输出矩阵和 src 有同样的类型, 否则, 它和 src 有同样的通道数, 且此时图像深度为 CV\_MAT\_DEPTH (dtype)。
- 第七个参数, InputArray 类型的 mask, 可选的操作掩膜, 有默认值 noArray()。

### 5.5.8 示例程序：离散傅里叶变换

这节中我们将学习一个以 dft() 函数为核心, 对图像求傅里叶变换的, 有详细注释的示例程序。

在此示例中, 将展示如何计算以及显示傅里叶变换后的幅度图像。由于数字图像的离散性, 像素值的取值范围也是有限的。比如在一张灰度图像中, 像素灰度值一般在 0 到 255 之间。因此, 我们这里讨论的也仅仅是离散傅里叶变换(DFT)。如果需要得到图像中的几何结构信息, 那么就要用到离散傅里叶变换。下面的步骤将以输入图像为单通道的灰度图像 I 为例, 进行分步说明。

#### 1. 【第一步】载入原始图像

我们在这一步以灰度模式读取原始图像, 进行是否读取成功的检测, 并显示出读取到的图像。代码如下。

```
//【1】以灰度模式读取原始图像并显示
Mat srcImage = imread("1.jpg", 0);
if(!srcImage.data) { printf("读取图片错误, 请确定目录下是否有 imread 函
数指定图片存在~! \n"); return false; }
imshow("原始图像", srcImage);
```

#### 2. 【第二步】将图像扩大到合适的尺寸

离散傅里叶变换的运行速度与图片的尺寸有很大关系。当图像的尺寸是 2、3、5 的整数倍时, 计算速度最快。因此, 为了达到快速计算的目的, 经常通过添凑新的边缘像素的方法获取最佳图像尺寸。函数 getOptimalDFTSize() 用于返回最佳尺寸, 而函数 copyMakeBorder() 用于填充边缘像素, 这一步代码如下。

```
//【2】将输入图像延扩到最佳的尺寸, 边界用 0 补充
int m = getOptimalDFTSize( I.rows );
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

int n = getOptimalDFTSize( I.cols );
//将添加的像素初始化为0。
Mat padded;
copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT,
Scalar::all(0));

```

### 3. 【第三步】为傅里叶变换的结果（实部和虚部）分配存储空间

傅里叶变换的结果是复数，这就是说对于每个原图像值，结果会有两个图像值。此外，频域值范围远远超过空间值范围，因此至少要将频域储存在 float 格式中。所以我们将输入图像转换成浮点类型，并多加一个额外通道来储存复数部分。

```

//【3】为傅里叶变换的结果(实部和虚部)分配存储空间
//将planes数组组合合并成一个多通道的数组complexI
Mat planes[] = {Mat<float>(padded), Mat::zeros(padded.size(),
CV_32F)};
Mat complexI;
merge(planes, 2, complexI);

```

### 4. 【第四步】进行离散傅里叶变换

这里的离散傅里叶变换为图像就地计算模式 (in-place，输入输出为同一图像)。

```

//【4】进行就地离散傅里叶变换
dft(complexI, complexI);

```

### 5. 【第五步】将复数转换为幅值

复数包含实数部分 (Re) 和虚数部分 (imaginary – Im)。离散傅里叶变换的结果是复数，对应的幅度可以表示为：

$$M = \sqrt{\operatorname{Re}(DFT(I))^2 + \operatorname{Im}(DFT(I))^2}$$

那么，转化为 OpenCV 代码，就是下面这样的。

```

//【5】将复数转换为幅值，即=> log(1 + sqrt(Re(DFT(I))^2 + Im(DFT(I))^2))
split(complexI, planes); // 将多通道数组complexI分离成几个单通道数组,
planes[0] = Re(DFT(I)), planes[1] = Im(DFT(I))
magnitude(planes[0], planes[1], planes[0]); // planes[0] = magnitude
Mat magnitudeImage = planes[0];

```

### 6. 【第六步】进行对数尺度 (logarithmic scale) 缩放。

傅里叶变换的幅度值范围大到不适合在屏幕上显示。高值在屏幕上显示为白点，而低值为黑点，高低值的变化无法有效分辨。为了在屏幕上凸显出高低变化的连续性，我们可以用对数尺度来替换线性尺度，公式如下。

$$M_1 = \log(1+M)$$

而写成 OpenCV 代码，就是下面这样的。

```

//【6】进行对数尺度(logarithmic scale)缩放
magnitudeImage += Scalar::all(1);

```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
log(magnitudeImage, magnitudeImage); //求自然对数
```

## 7. 【第七步】剪切和重分布幅度图象限

因为在第二步中延扩了图像，那现在是时候将新添加的像素剔除了。为了方便显示，也可以重新分布幅度图象限位置（注：将第五步得到的幅度图从中间划开，得到4张1/4子图像，将每张子图像看成幅度图的一个象限，重新分布，即刻4个角点重叠到图片中心）。这样的话原点(0,0)就位移到图像中心了。OpenCV代码如下。

```
//【7】剪切和重分布幅度图象限
//若有奇数行或奇数列，进行频谱裁剪
magnitudeImage = magnitudeImage(Rect(0, 0, magnitudeImage.cols & -2,
magnitudeImage.rows & -2));
//重新排列傅里叶图像中的象限，使得原点位于图像中心
int cx = magnitudeImage.cols/2;
int cy = magnitudeImage.rows/2;
Mat q0(magnitudeImage, Rect(0, 0, cx, cy)); // ROI 区域的左上
Mat q1(magnitudeImage, Rect(cx, 0, cx, cy)); // ROI 区域的右上
Mat q2(magnitudeImage, Rect(0, cy, cx, cy)); // ROI 区域的左下
Mat q3(magnitudeImage, Rect(cx, cy, cx, cy)); // ROI 区域的右下
//交换象限（左上与右下进行交换）
Mat tmp;
q0.copyTo(tmp);
q3.copyTo(q0);
tmp.copyTo(q3);
//交换象限（右上与左下进行交换）
q1.copyTo(tmp);
q2.copyTo(q1);
tmp.copyTo(q2);
```

## 8. 【第八步】归一化

这一步仍然是为了显示。现在有了重分布后的幅度图，但是幅度值仍然超过可显示范围[0,1]。我们使用 normalize()函数将幅度归一化到可显示范围。

```
//【8】归一化，用 0 到 1 之间的浮点值将矩阵变换为可视的图像格式
//此句代码的 OpenCV2 版为：
//normalize(magnitudeImage, magnitudeImage, 0, 1, CV_MINMAX);
//此句代码的 OpenCV3 版为：
normalize(magnitudeImage, magnitudeImage, 0, 1, NORM_MINMAX);
```

## 9. 【第九步】显示效果图

处理完成，最后进行显示即可。

```
//【9】显示效果图
imshow("频谱幅值", magnitudeImage);
```

将上述代码组合到一起，便得到了程序的完整源代码。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所使用的头文件和命名空间
-----
```

**SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>**

```
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
using namespace cv;

//-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始执行
//-----
int main( )
{
    //【1】以灰度模式读取原始图像并显示
    Mat srcImage = imread("l.jpg", 0);
    if(!srcImage.data) { printf("读取图片错误，请确定目录下是否有 imread 函数指定图片存在~! \n"); return false; }
    imshow("原始图像", srcImage);

    ShowHelpText();

    //【2】将输入图像延伸到最佳的尺寸，边界用0补充
    int m = getOptimalDFTSize( srcImage.rows );
    int n = getOptimalDFTSize( srcImage.cols );
    //将添加的像素初始化为0。
    Mat padded;
    copyMakeBorder(srcImage, padded, 0, m - srcImage.rows, 0, n - srcImage.cols, BORDER_CONSTANT, Scalar::all(0));

    //【3】为傅里叶变换的结果(实部和虚部)分配存储空间。
    //将 planes 数组组合合并成一个多通道的数组 complexI
    Mat planes[] = {Mat<float>(padded), Mat::zeros(padded.size(), CV_32F)};
    Mat complexI;
    merge(planes, 2, complexI);

    //【4】进行就地离散傅里叶变换
    dft(complexI, complexI);

    //【5】将复数转换为幅值，即=> log(1 + sqrt(Re(DFT(I))^2 + Im(DFT(I))^2))
    split(complexI, planes); // 将多通道数组 complexI 分离成几个单通道数组,
    planes[0] = Re(DFT(I)), planes[1] = Im(DFT(I))
    magnitude(planes[0], planes[1], planes[0]); // planes[0] = magnitude
    Mat magnitudeImage = planes[0];

    //【6】进行对数尺度(logarithmic scale)缩放
    magnitudeImage += Scalar::all(1);
    log(magnitudeImage, magnitudeImage); //求自然对数

    //【7】剪切和重分布幅度图象限
    //若有奇数行或奇数列，进行频谱裁剪
    magnitudeImage = magnitudeImage(Rect(0, 0, magnitudeImage.cols & -2,
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
magnitudeImage.rows & -2));
    //重新排列傅里叶图像中的象限，使得原点位于图像中心
    int cx = magnitudeImage.cols/2;
    int cy = magnitudeImage.rows/2;
    Mat q0(magnitudeImage, Rect(0, 0, cx, cy));    // ROI 区域的左上
    Mat q1(magnitudeImage, Rect(cx, 0, cx, cy));   // ROI 区域的右上
    Mat q2(magnitudeImage, Rect(0, cy, cx, cy));   // ROI 区域的左下
    Mat q3(magnitudeImage, Rect(cx, cy, cx, cy)); // ROI 区域的右下
    //交换象限（左上与右下进行交换）
    Mat tmp;
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    //交换象限（右上与左下进行交换）
    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);

    //【8】归一化，用 0 到 1 之间的浮点值将矩阵变换为可视的图像格式
    //此句代码的 OpenCV2 版为：
    //normalize(magnitudeImage, magnitudeImage, 0, 1, CV_MINMAX);
    //此句代码的 OpenCV3 版为：
    normalize(magnitudeImage, magnitudeImage, 0, 1, NORM_MINMAX);

    //【9】显示效果图
    imshow("频谱幅值", magnitudeImage);
    waitKey();

    return 0;
}
```

程序注释已经足够详尽，其中新出现的函数都在上文中有过介绍，所以在这里就不花篇幅进一步展开讲解了。让我们一起看一下运行截图（图 5.15、图 5.16），以结束本节的学习。

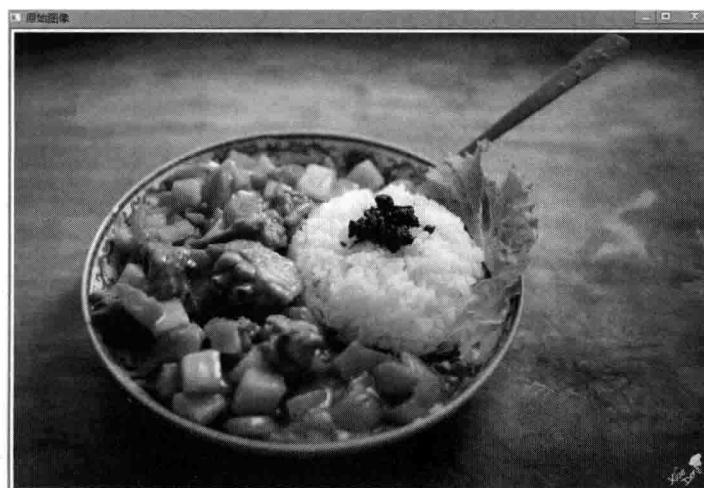


图 5.15 原始图

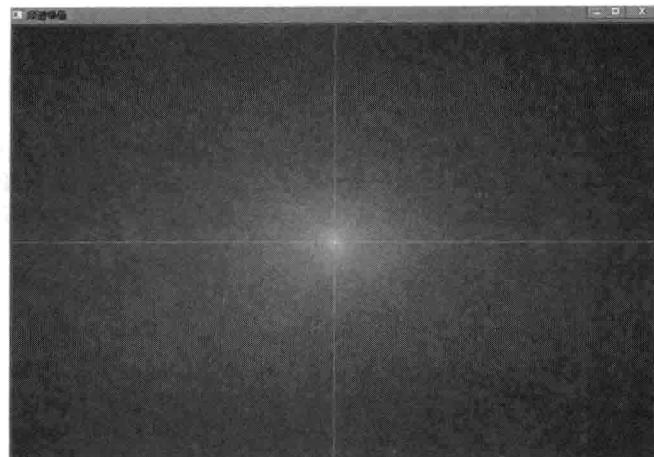


图 5.16 离散傅里叶变换效果图

## 5.6 输入输出 XML 和 YAML 文件

### 5.6.1 XML 和 YAML 文件简介

本节我们将一起认识 XML 和 YAML 这两种文件类型。

所谓 XML，即 eXtensible Markup Language，翻译成中文为“可扩展标识语言”。首先，XML 是一种元标记语言。所谓“元标记”，就是开发者可以根据自身需要定义自己的标记，比如可以定义标记<book>、<name>。任何满足 XML 命名规则的名称都可以标记，这就向不同的应用程序打开了的大门。此外，XML 是一种语义/结构化语言，它描述了文档的结构和语义。

YAML 是“YAML Ain't a Markup Language”（译为“YAML 不是一种置标语言”）的递回缩写。在开发的这种语言时，YAML 的原意是：“Yet Another Markup Language”（仍是一种置标语言），但为了强调这种语言以数据为中心，而不是以置标语言为重点，而用返璞词进行重新命名。YAML 是一个可读性高，用来表达资料序列的格式。它参考了其他多种语言，包括：XML、C 语言、Python、Perl，以及电子邮件格式 RFC2822。



.yml 和.yaml 同为 YAML 格式的后缀名

总之，YAML 试图用一种比 XML 更敏捷的方式，来完成 XML 所完成的任务。

### 5.6.2 FileStorage 类操作文件的使用引导

XML 和 YAML 是使用非常广泛的文件格式，可以利用 XML 或者 YAML 格式的文件存储和还原各式各样的数据结构。当然，它们还可以存储和载入任意复杂的数据结构，其中就包括了 OpenCV 相关周边的数据结构，以及各种原始数据类型，如整数和浮点数字和文本字符串。

我们一般使用如下过程来写入或者读取数据到 XML 或 YAML 文件中。

(1) 实例化一个 `FileStorage` 类的对象，用默认带参数的构造函数完成初始化，或者用 `FileStorage::open()` 成员函数辅助初始化。

(2) 使用流操作符<<进行文件写入操作，或者>>进行文件读取操作，类似 C++ 中的文件输入输出流。

(3) 使用 `FileStorage::release()` 函数析构掉 `FileStorage` 类对象，同时关闭文件。

下面分别对这三个步骤进行实例讲解。

## 1. 【第一步】XML、YAML 文件的打开

### (1) 准备文件写操作

`FileStorage` 是 OpenCV 中 XML 和 YAML 文件的存储类，封装了所有相关的信息。它是 OpenCV 从文件中读数据或向文件中写数据时必须要使用的一个类。

此类的构造函数为 `FileStorage::FileStorage`，有两个重载，如下。

```
C++: FileStorage::FileStorage()
C++: FileStorage::FileStorage(const string& source, int flags, const
string& encoding=string())
```

构造函数在实际使用中，方法一般有两种。

1) 对于第二种带参数的构造函数，进行写操作范例如下。

```
FileStorage fs("abc.xml", FileStorage::WRITE);
```

2) 对于第一种不带参数的构造函数，可以使用其成员函数 `FileStorage::open` 进行数据的写操作，范例如下。

```
FileStorage fs;
fs.open("abc.xml", FileStorage:: WRITE);
```

### (2) 准备文件读操作

上面讲到的都是以 `FileStorage:: WRITE` 为标识符的写操作，而读操作，采用 `FileStorage::READ` 标识符即可，相关示例代码如下。

1) 第一种方式

```
FileStorage fs("abc.xml", FileStorage:: READ);
```

2) 第二种方式

```
FileStorage fs;
fs.open("abc.xml", FileStorage:: READ);
```

另外需要注意的是，上面的这些操作示例是对 XML 文件为例子作演示的，而对 YAML 文件，操作方法是类似的，就是将 XML 文件换为 YAML 文件即可。

## 2. 【第二步】进行文件读写操作

### (1) 文本和数字的输入和输出

定义好 `FileStorage` 类对象之后，写入文件可以使用“`<<`”运算符，例如：

```
fs << "iterationNr" << 100;
```

而读取文件，使用“`>>`”运算符，例如：

```
int itNr;
fs["iterationNr"] >> itNr;
itNr = (int) fs["iterationNr"];
```

## (2) OpenCV 数据结构的输入和输出

关于 OpenCV 数据结构的输入和输出，和基本的 C++形式相同，范例如下。

```
//数据结构的初始化
Mat R = Mat_<uchar>::eye(3, 3),
Mat T = Mat_<double>::zeros(3, 1);
// 向 Mat 中写入数据
fs << "R" << R;
fs << "T" << T;
// 从 Mat 中读取数据
fs["R"] >> R;
fs["T"] >> T;
```

## 3. 【第三步】vector (arrays) 和 maps 的输入和输出

对于 `vector` 结构的输入和输出，要注意在第一个元素前加上“[”，在最后一个元素前加上“]”。例如：

```
fs << "strings" << "["; // 开始读入 string 文本序列
fs << "image1.jpg" << "Awesomeness" << "baboon.jpg";
fs << "]"; // 关闭序列
```

而对于 `map` 结构的操作，使用的符号是“{”和“}”，例如：

```
fs << "Mapping"; // 开始读入 Mapping 文本
fs << "{" << "One" << 1;
fs << "Two" << 2 << "}";
fs << "}" << endl;
```

读取这些结构的时候，会用到 `FileNode` 和 `FileNodeIterator` 数据结构。对 `FileStorage` 类的“[”、“]”操作符会返回 `FileNode` 数据类型；对于一连串的 node，可以使用 `FileNodeIterator` 结构，例如：

```
FileNode n = fs["strings"]; // 读取字符串序列以得到节点
if (n.type() != FileNode::SEQ)
{
    cerr << "发生错误！字符串不是一个序列！" << endl;
    return 1;
}

FileNodeIterator it = n.begin(), it_end = n.end(); // 遍历节点
for (; it != it_end; ++it)
    cout << (*it) << endl;
```

## 4. 【第四步】文件关闭

需要注意的是，文件关闭操作会在 `FileStorage` 类销毁时自动进行，但我们也

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

可显式调用其析构函数 FileStorage::release() 实现。FileStorage::release() 函数会析构掉 FileStorage 类对象，同时关闭文件。

调用过程非常简单，如下。

```
fs.release();
```

下面将通过实例来帮助大家将以上知识融会贯通。

### 5.6.3 示例程序：XML 和 YAML 文件的写入

让我们先看一个关于 XML 或 YAML 文件的写入实例，示例代码如下：

```
#include "opencv2/opencv.hpp"
#include <time.h>

using namespace cv;

int main()
{
    // 初始化
    FileStorage fs("test.yaml", FileStorage::WRITE);

    // 开始文件写入
    fs << "frameCount" << 5;
    time_t rawtime; time(&rawtime);
    fs << "calibrationDate" << asctime(localtime(&rawtime));
    Mat cameraMatrix = (Mat<double>(3,3) << 1000, 0, 320, 0, 1000, 240,
0, 0, 1);
    Mat distCoeffs = (Mat<double>(5,1) << 0.1, 0.01, -0.001, 0, 0);
    fs << "cameraMatrix" << cameraMatrix << "distCoeffs" << distCoeffs;
    fs << "features" << "[";
    for( int i = 0; i < 3; i++ )
    {
        int x = rand() % 640;
        int y = rand() % 480;
        uchar lbp = rand() % 256;

        fs << ":" << "x" << x << "y" << y << "lbp" << ":";
        for( int j = 0; j < 8; j++ )
            fs << ((lbp >> j) & 1);
        fs << "]" << "}";
    }
    fs << "]";
    fs.release();

    printf("文件读写完毕，请在工程目录下查看生成的文件~");
    getchar();

    return 0;
}
```

运行此程序，会在工程目录下程序一个名为“test.yaml”的文件，用记事本或

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

者其他文本编辑器比如 notepad++打开它，可以发现写入的 YAML 文档内容如下。

```
%YAML:1.0
frameCount: 5
calibrationDate: "Wed Jun 25 10:20:17 2014\n"
cameraMatrix: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ 1000., 0., 320., 0., 1000., 240., 0., 0., 1. ]
distCoeffs: !!opencv-matrix
  rows: 5
  cols: 1
  dt: d
  data: [ 1.0000000000000001e-001, 1.000000000000000e-002,
    -1.000000000000000e-003, 0., 0. ]
features:
  - { x:41, y:227, lbp:[ 0, 1, 1, 1, 1, 1, 0, 1 ] }
  - { x:260, y:449, lbp:[ 0, 0, 1, 1, 0, 1, 1, 0 ] }
  - { x:598, y:78, lbp:[ 0, 1, 0, 0, 1, 0, 1, 0 ] }
```

如果我们修改上面代码的 FileStorage fs (“test.yaml”, FileStorage::WRITE) 这一句代码，将其中的 “test.yaml” 的后缀换为 xml、yml、txt 甚至 doc，都是可以得到运行结果的，大家不妨尝试一下。

### 5.6.4 示例程序：XML 和 YAML 文件的读取

接着，我们一起看读操作应该如何进行，完整的源代码如下。

```
#include "opencv2/opencv.hpp"
#include <time.h>

using namespace cv;
using namespace std;

int main()
{
    //改变 console 字体颜色
    system("color 6F");

    //初始化
    FileStorage fs2("test.yaml", FileStorage::READ);

    // 第一种方法，对 FileNode 操作
    int frameCount = (int)fs2["frameCount"];

    std::string date;
    // 第二种方法，使用 FileNode 运算符>>
    fs2["calibrationDate"] >> date;

    Mat cameraMatrix2, distCoeffs2;
    fs2["cameraMatrix"] >> cameraMatrix2;
```

## Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

fs2["distCoeffs"] >> distCoeffs2;

cout << "frameCount: " << frameCount << endl
<< "calibration date: " << date << endl
<< "camera matrix: " << cameraMatrix2 << endl
<< "distortion coeffs: " << distCoeffs2 << endl;

FileNode features = fs2["features"];
FileNodeIterator it = features.begin(), it_end = features.end();
int idx = 0;
std::vector<uchar> lbpval;

//使用 FileNodeIterator 遍历序列
for( ; it != it_end; ++it, idx++ )
{
    cout << "feature #" << idx << ":" ;
    cout << "x=" << (int)(*it)[ "x" ] << ", y=" << (int)(*it)[ "y" ] <<
", lbp: (";
    // 我们也可以使用 filenode >> std::vector 操作符来很容易地读数值阵列
    (*it)[ "lbp" ] >> lbpval;
    for( int i = 0; i < (int)lbpval.size(); i++ )
        cout << " " << (int)lbpval[i];
    cout << ")" << endl;
}
fs2.release();

//程序结束，输出一些帮助文字
printf("\n文件读取完毕，请输入任意键结束程序~");
getchar();

return 0;
}

```

此程序需要工程目录下有指定的文件存在，如我们将前文刚刚生成的 test.yaml 复制到工程目录下，运行此程序，便可以得出正确的结果。运行截图如图 5.17 所示。

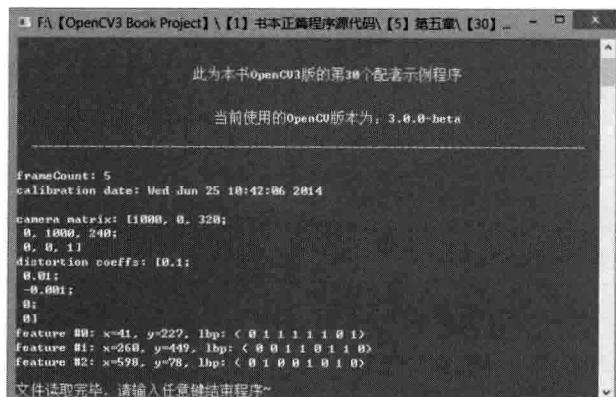


图 5.17 运行输出截图

## 5.7 本章小结

本节中，我们学习了 core 模块的一些进阶知识点——操作图像中的像素、图像混合、分离颜色通道、调节图像的对比度和亮度、进行离散傅里叶变换，以及输入输出 XML 和 YAML 文件。

### 本章核心函数清单

函数名称	说明	对应讲解章节
addWeighted	计算两个数组（图像阵列）的加权和	5.2.3
split	将一个多通道数组分离成几个单通道数组	5.3.1
merge	将多个数组组合合并成一个多通道的数组	5.3.2
dft	对一维或二维浮点数数组进行正向或反向离散傅里叶变换	5.5.2
getOptimalDFTSize	返回给定向量尺寸的傅里叶最优尺寸大小	5.5.3
copyMakeBorder	扩充图像边界	5.5.4
magnitude	计算二维矢量的幅值	5.5.5
log	计算每个数组元素绝对值的自然对数	5.5.6
normalize	进行矩阵归一化	5.5.7
FileStorage 类	进行文件操作的类	5.6.2

### 本章示例程序清单

示例程序序号	程序说明	对应章节
21	操作图像中像素的方法一：用指针访问像素	5.1.5、5.1.6
22	操作图像中像素的方法二：用迭代器操作像素	5.1.5、5.1.6
23	操作图像中像素的方法三：动态地址计算	5.1.5、5.1.6
24	遍历图像中像素的 14 种方法	5.1.6
25	初级图像混合	5.2.4
26	多通道图像混合	5.3.3
27	图像对比度、亮度值调整	5.4.3
28	离散傅里叶变换	5.5.8
29	XML 和 YAML 文件的写入	5.6.3
30	XML 和 YAML 文件的读取	5.6.4

# 第三部分

## 掌握 imgproc 组件

imgproc 组件是 Image 和 Process 这两个单词的缩写组合，即图像处理模块，这个模块包含了如下内容：

- 线性和非线性的图像滤波
- 图像的几何变换
- 其他（Miscellaneous）图像转换
- 直方图相关
- 结构分析和形状描述
- 运动分析和对象跟踪
- 特征检测
- 目标检测等内容

第三部分包含如下四个章节：

---

- 第 6 章 图像处理
- 第 7 章 图像变换
- 第 8 章 图像轮廓与图像分割修复
- 第 9 章 直方图与匹配

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

# 第 6 章

## 图像处理

### 导读

---

到这里，我们已经了解了 OpenCV 库的结构，知道了用来表示图像的基本数据结构，也熟悉了运行程序并将结果显示到屏幕上的 Highgui 接口。现在，我们可以利用这些控制图像结构的基本方法，来学习更加复杂的图像处理方法。

所谓的学习高级的处理方法，即把图像以“图像”的方式来处理，而不是作为只由颜色值或灰度图组成的数组。而我们在提到“图像处理”时，就是表示使用图像结构中所定义的高层处理方法来完成特殊任务，而这些任务，就是图形和视觉范畴的任务。

### 本章中，你将学到：

---

- 三种线性滤波：方框滤波、均值滤波、高斯滤波
- 两种非线性滤波：中值滤波、双边滤波
- 7 种图像处理形态学：腐蚀、膨胀，开运算、闭运算、形态学梯度、顶帽、黑帽
- 漫水填充
- 图像缩放
- 图像金字塔
- 阈值化

## 6.1 线性滤波：方框滤波、均值滤波、高斯滤波

### 6.1.1 平滑处理

平滑处理（smoothing）也称模糊处理（blurring），是一种简单且使用频率很高的图像处理方法。平滑处理的用途有很多，最常见的是用来减少图像上的噪点或者失真。在涉及到降低图像分辨率时，平滑处理是非常好用的方法。

### 6.1.2 图像滤波与滤波器

图像滤波，指在尽量保留图像细节特征的条件下对目标图像的噪声进行抑制，是图像预处理中不可缺少的操作，其处理效果的好坏将直接影响到后续图像处理和分析的有效性和可靠性。

消除图像中的噪声成分叫作图像的平滑化或滤波操作。信号或图像的能量大部分集中在幅度谱的低频和中频段，而在较高频段，有用的信息经常被噪声淹没。因此一个能降低高频成分幅度的滤波器就能够减弱噪声的影响。

图像滤波的目的有两个：一个是抽出对象的特征作为图像识别的特征模式；另一个是为适应图像处理的要求，消除图像数字化时所混入的噪声。

而对滤波处理的要求也有两条：一是不能损坏图像的轮廓及边缘等重要信息；二是使图像清晰视觉效果好。

平滑滤波是低频增强的空间域滤波技术。它的目的有两类：一类是模糊；另一类是消除噪音。

空间域的平滑滤波一般采用简单平均法进行，就是求邻近像元点的平均亮度值。邻域的大小与平滑的效果直接相关，邻域越大平滑的效果越好，但邻域过大，平滑也会使边缘信息损失的越大，从而使输出的图像变得模糊，因此需合理选择邻域的大小。

关于滤波器，一种形象的比喻是：可以把滤波器想象成一个包含加权系数的窗口，当使用这个滤波器平滑处理图像时，就把这个窗口放到图像之上，透过这个窗口来看我们得到的图像。

滤波器的种类有很多，在新版本的 OpenCV 中，提供了如下 5 种常用的图像平滑处理操作方法，它们分别被封装在单独的函数中，使用起来非常方便。

- 方框滤波——BoxBlur 函数
- 均值滤波（邻域平均滤波）——Blur 函数
- 高斯滤波——GaussianBlur 函数
- 中值滤波——medianBlur 函数
- 双边滤波——bilateralFilter 函数

本节要讲解的是作为线性滤波的方框滤波、均值滤波和高斯滤波。其他两种

非线性滤波操作——中值滤波和双边滤波，我们留待下节讲解。

### 6.1.3 线性滤波器的简介

线性滤波器：线性滤波器经常用于剔除输入信号中不想要的频率或者从许多频率中选择一个想要的频率。

几种常见的线性滤波器如下。

- 低通滤波器：允许低频率通过；
- 高通滤波器：允许高频率通过；
- 带通滤波器：允许一定范围频率通过；
- 带阻滤波器：阻止一定范围频率通过并且允许其他频率通过；
- 全通滤波器：允许所有频率通过，仅仅改变相位关系；
- 陷波滤波器（Band-Stop Filter）：阻止一个狭窄频率范围通过，是一种特殊带阻滤波器。

### 6.1.4 滤波和模糊

关于滤波和模糊，大家往往在初次接触的时候会弄混淆：“一会儿说滤波，一会儿又说模糊，似乎不太清楚。”

没关系，在这里，我们就来分析一下，为大家扫清障碍。

上文已经提到过，滤波是将信号中特定波段频率滤除的操作，是抑制和防止干扰的一项重要措施。

为了方便说明，就拿我们经常用的高斯滤波来作例子吧。滤波可分低通滤波和高通滤波两种：高斯滤波是指用高斯函数作为滤波函数的滤波操作，至于是不是模糊，要看是高斯低通还是高斯高通，低通就是模糊，高通就是锐化。

其实说白了是很简单的：

- 高斯滤波是指用高斯函数作为滤波函数的滤波操作；
- 高斯模糊就是高斯低通滤波。

### 6.1.5 邻域算子与线性邻域滤波

邻域算子（局部算子）是利用给定像素周围的像素值的决定此像素的最终输出值的一种算子。而线性邻域滤波就是一种常用的邻域算子，像素的输出值取决于输入像素的加权和，具体过程如图 6.1 所示。

邻域算子除了用于局部色调调整以外，还可以用于图像滤波，以实现图像的平滑和锐化，图像边缘增强或者图像噪声的去除。本节我们介绍的主角是线性邻域滤波算子，即用不同的权重去结合一个小邻域内的像素，来得到应有的处理效果。

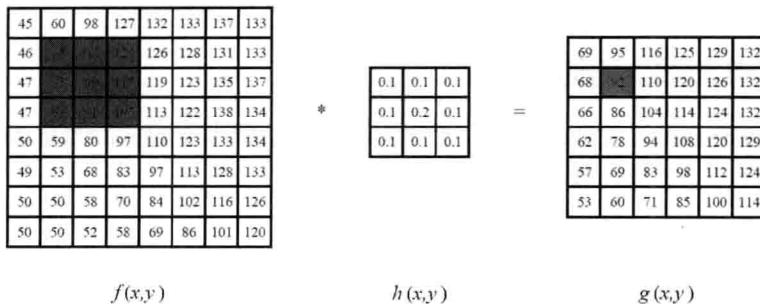


图 6.1 邻域滤波（卷积）

图注：邻域滤波（卷积）——左边图像与中间图像的卷积产生右边图像。目标图像中蓝色标记的像素是利用原图像中红色标记的像素计算得到的。

线性滤波处理的输出像素值  $g(i,j)$  是输入像素值  $f(i+k, j+I)$  的加权和，如下

$$g(i,j) = \sum_{k,I} f(i+k, j+I)h(k,I)$$

其中的  $h(k,I)$ ，我们称其为“核”，是滤波器的加权系数，即滤波器的“滤波系数”。

上面的式子可以简单写作：

$$g = f \otimes h$$

其中  $f$  表示输入像素值， $h$  表示加权系数“核”， $g$  表示输出像素值。

在新版本的 OpenCV 中，提供了如下三种常用的线性滤波操作，它们分别被封装在单独的函数中，使用起来非常方便。

- 方框滤波——boxblur 函数
- 均值滤波——blur 函数
- 高斯滤波——GaussianBlur 函数

下面我们来对它们进行一一介绍。

### 6.1.6 方框滤波 (box Filter)

方框滤波 (box Filter) 被封装在一个名为 boxblur 的函数中，即 boxblur 函数的作用是使用方框滤波器 (box filter) 来模糊一张图片，从 src 输入，从 dst 输出。

函数原型如下。

```
C++: void boxFilter(InputArray src, OutputArray dst, int ddepth, Size
ksize, Point anchor=Point(-1,-1), bool normalize=true, int
borderType=BORDER_DEFAULT )
```

参数详解如下。

- 第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对

象即可。该函数对通道是独立处理的，且可以处理任意通道数的图片。但需要注意，待处理的图片深度应该为 CV\_8U、CV\_16U、CV\_16S、CV\_32F 以及 CV\_64F 之一。

- 第二个参数，OutputArray 类型的 dst，即目标图像，需要和源图片有一样的尺寸和类型。
- 第三个参数，int 类型的 ddepth，输出图像的深度，-1 代表使用原图深度，即 src.depth()。
- 第四个参数，Size 类型（对 Size 类型稍后有讲解）的 ksize，内核的大小。一般用 Size(w,h) 来表示内核的大小，其中 w 为像素宽度，h 为像素高度。Size(3,3) 就表示 3x3 的核大小，Size(5,5) 就表示 5x5 的核大小。
- 第五个参数，Point 类型的 anchor，表示锚点（即被平滑的那个点）。注意它有默认值 Point(-1,-1)。如果这个点坐标是负值的话，就表示取核的中心为锚点，所以默认值 Point(-1,-1) 表示这个锚点在核的中心。
- 第六个参数，bool 类型的 normalize，默认值为 true，一个标识符，表示内核是否被其区域归一化（normalized）了。
- 第七个参数，int 类型的 borderType，用于推断图像外部像素的某种边界模式。有默认值 BORDER\_DEFAULT，我们一般不去管它。

BoxFilter() 函数方框滤波所用的核表示如下。

$$K = a \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

其中：

$$a \begin{cases} \frac{1}{ksize.width*ksize.height} & \text{当normalize = true时} \\ 1 & \text{当normalize = false时} \end{cases}$$

上式中 f 表示原图，h 表示核，g 表示目标图，当 normalize=true 的时候，方框滤波就变成了我们熟悉的均值滤波。也就是说，均值滤波是方框滤波归一化（normalized）后的特殊情况。其中，归一化就是把要处理的量都缩放到一个范围内，比如 (0,1)，以便统一处理和直观量化。而非归一化（Unnormalized）的方框滤波用于计算每个像素邻域内的积分特性，比如密集光流算法（dense optical flow algorithms）中用到的图像倒数的协方差矩阵（covariance matrices of image derivatives）。

如果我们要在可变的窗口中计算像素总和，可以使用 integral() 函数。

### 6.1.7 均值滤波

均值滤波，是最简单的一种滤波操作，输出图像的每一个像素是核窗口内输入图像对应像素的平均值（所有像素加权系数相等），其实说白了它就是归一化后

的方框滤波。我们在下文进行源码剖析时会发现，blur 函数内部中其实就是在调用了一下 boxFilter。

下面开始讲均值滤波的内容吧。

### 1. 均值滤波的理论简析

均值滤波是典型的线性滤波算法，主要方法为邻域平均法，即用一片图像区域的各个像素的均值来代替原图像中的各个像素值。一般需要在图像上对目标像素给出一个模板（内核），该模板包括了其周围的临近像素（比如以目标像素为中心的周围 8 (3x3-1) 个像素，构成一个滤波模板，即去掉目标像素本身）。再用模板中的全体像素的平均值来代替原来像素值。即对待处理的当前像素点  $(x,y)$ ，选择一个模板，该模板由其近邻的若干像素组成，求模板中所有像素的均值，再把该均值赋予当前像素点  $(x,y)$ ，作为处理后图像在该点上的灰度点  $g(x,y)$ ，即  $g(x,y) = \frac{1}{m} \sum f(x,y)$ ，其中  $m$  为该模板中包含当前像素在内的像素总个数。

### 2. 均值滤波的缺陷

均值滤波本身存在着固有的缺陷，即它不能很好地保护图像细节，在图像去噪的同时也破坏了图像的细节部分，从而使图像变得模糊，不能很好地去除噪声点。

### 3. 在 OpenCV 中使用均值滤波——blur 函数

blur 函数的作用是：对输入的图像 src 进行均值滤波后用 dst 输出。

blur 函数在 OpenCV 官方文档中，给出的其核是这样的：

$$K = \frac{1}{\text{ksize.width} * \text{ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

这个内核一看就明了，就是在求均值，即 blur 函数封装的就是均值滤波。

blur 函数的原型如下。

```
C++: void blur(InputArray src, OutputArray dst, Size ksize, Point anchor=Point(-1,-1), int borderType=BORDER_DEFAULT )
```

- 第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可。该函数对通道是独立处理的，且可以处理任意通道数的图片。但需要注意的是，待处理的图片深度应该为 CV\_8U、CV\_16U、CV\_16S、CV\_32F 以及 CV\_64F 之一。
- 第二个参数，OutputArray 类型的 dst，即目标图像，需要和源图片有一样的尺寸和类型。比如可以用 Mat::Clone，以源图片为模板，来初始化得到如假包换的目标图。
- 第三个参数，Size 类型（对 Size 类型稍后有讲解）的 ksize，内核的大小。

一般写作 `Size(w,h)` 来表示内核的大小（其中，`w` 为像素宽度，`h` 为像素高度）。`Size(3,3)` 就表示  $3 \times 3$  的核大小，`Size(5,5)` 就表示  $5 \times 5$  的核大小。

- 第四个参数，`Point` 类型的 `anchor`，表示锚点（即被平滑的那个点），注意它有默认值 `Point(-1,-1)`。如果这个点坐标是负值，就表示取核的中心为锚点，所以默认值 `Point(-1,-1)` 表示这个锚点在核的中心。
- 第五个参数，`int` 类型的 `borderType`，用于推断图像外部像素的某种边界模式。有默认值 `BORDER_DEFAULT`，我们一般不去管它。

## 6.1.8 高斯滤波

### 1. 高斯滤波的理论简析

高斯滤波是一种线性平滑滤波，可以消除高斯噪声，广泛应用于图像处理的减噪过程。通俗地讲，高斯滤波就是对整幅图像进行加权平均的过程，每一个像素点的值，都由其本身和邻域内的其他像素值经过加权平均后得到。高斯滤波的具体操作是：用一个模板（或称卷积、掩模）扫描图像中的每一个像素，用模板确定的邻域内像素的加权平均灰度值去替代模板中心像素点的值。

大家常说高斯滤波是最有用的滤波操作，虽然它用起来效率往往不是最高的。高斯模糊技术生成的图像，其视觉效果就像是经过一个半透明屏幕在观察图像，这与镜头焦外成像效果散景以及普通照明阴影中的效果都明显不同。高斯平滑也用于计算机视觉算法中的预先处理阶段，以增强图像在不同比例大小下的图像效果（参见尺度空间表示以及尺度空间实现）。从数学的角度来看，图像的高斯模糊过程就是图像与正态分布做卷积。由于正态分布又叫作高斯分布，所以这项技术就叫作高斯模糊。

图像与圆形方框模糊做卷积将会生成更加精确的焦外成像效果。由于高斯函数的傅里叶变换是另外一个高斯函数，所以高斯模糊对于图像来说就是一个低通滤波操作。

高斯滤波器是一类根据高斯函数的形状来选择权值的线性平滑滤波器。高斯平滑滤波器对于抑制服从正态分布的噪声非常有效。一维零均值高斯函数如下。

$$G(x) = \exp(-x^2/(2\sigma^2))$$

其中，高斯分布参数 `Sigma` 决定了高斯函数的宽度。对于图像处理来说，常用二维零均值离散高斯函数作平滑滤波器。

二维高斯函数如下。

$$G_0(x,y) = Ae^{-\frac{(x-u_x)^2}{2\sigma_x^2} - \frac{(y-u_y)^2}{2\sigma_y^2}}$$

### 2. 高斯滤波：`GaussianBlur` 函数

`GaussianBlur` 函数的作用是用高斯滤波器来模糊一张图片，对输入的图像 `src`

进行高斯滤波后用 dst 输出。它将源图像和指定的高斯核函数做卷积运算，并且支持就地过滤（In-placefiltering）。

```
C++: void GaussianBlur(InputArray src,OutputArray dst, Size ksize,
double sigmaX, double sigmaY=0, int borderType=BORDER_DEFAULT )
```

- 第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可。它可以是单独的任意通道数的图片，但需要注意的是，其图片深度应该为 CV\_8U、CV\_16U、CV\_16S、CV\_32F 以及 CV\_64F 之一。
- 第二个参数，OutputArray 类型的 dst，即目标图像，需要和源图片有一样的尺寸和类型。比如可以用 Mat::Clone，以源图片为模板，来初始化得到如假包换的目标图。
- 第三个参数，Size 类型的 ksize 高斯内核的大小。其中 ksize.width 和 ksize.height 可以不同，但它们都必须为正数和奇数，或者是零，这都由 sigma 计算而来。
- 第四个参数，double 类型的 sigmaX，表示高斯核函数在 X 方向的标准偏差。
- 第五个参数，double 类型的 sigmaY，表示高斯核函数在 Y 方向的标准偏差。若 sigmaY 为零，就将它设为 sigmaX；如果 sigmaX 和 sigmaY 都是 0，那么就由 ksize.width 和 ksize.height 计算出来。

为了结果的正确性着想，最好是把第三个参数 Size、第四个参数 sigmaX 和第五个参数 sigmaY 全部指定到。

- 第六个参数，int 类型的 borderType，用于推断图像外部像素的某种边界模式。有默认值 BORDER\_DEFAULT，我们一般不去管它。

### 6.1.9 线性滤波相关 OpenCV 源码剖析

在这一部分中，笔者将带领大家领略 OpenCV 的开源魅力，对 OpenCV 中节讲解到的线性滤波函数——boxFilter、blur 和 GaussianBlur 函数以及周边所涉及到的源码进行适当的剖析。

通过本章的学习，我们可以对 OpenCV 有更加深刻的理解，成为一个高端大气的 OpenCV 使用者。

#### 1. OpenCV 中 boxFilter 函数源码解析

可以在 OpenCV 的安装路径\sources\modules\imgproc\src 下的 smooth.cpp 源文件中找到 boxFilter 函数的源代码，如下。

```
void cv::boxFilter( InputArray _src,OutputArray _dst, int ddepth,
                   Size ksize, Point anchor,
                   bool normalize, int borderType)
{
    Mat src = _src.getMat(); // 复制源图的形参 Mat 数据到临时变量，用于稍后的操作
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

int sdepth =src.depth(), cn = src.channels(); //定义 int 型临时变量，代表
源图深度的 sdepth，源图通道的引用 cn

//处理 ddepth 小于零的情况
if( ddepth < 0 )
    ddepth = sdepth;
_dst.create( src.size(), CV_MAKETYPE(ddepth, cn) ); //初始化目标图
Mat dst =_dst.getMat(); //复制目标图的形参 Mat 数据到临时变量，用于稍后的操作

//处理 borderType 不为 BORDER_CONSTANT 且 normalize 为真的情况
if( borderType != BORDER_CONSTANT && normalize ) {
    if( src.rows == 1 )
        ksize.height = 1;
    if( src.cols == 1 )
        ksize.width = 1;
}

//若之前有过 HAVE_TEGRA_OPTIMIZATION 优化选项的定义，则执行宏体中的 tegra 优化
版函数并返回
#ifdef HAVE_TEGRA_OPTIMIZATION
    if ( tegra::box(src, dst, ksize, anchor, normalize, borderType) )
        return;
#endif

//调用 FilterEngine 滤波引擎，正式开始滤波操作
Ptr<FilterEngine> f = createBoxFilter( src.type(), dst.type(),
                                         ksize, anchor, normalize, borderType );
f->apply( src, dst );
}

```

其中，`Ptr` 是用来动态分配的对象的智能指针模板类。可以发现，函数的内部代码思路是很清晰的：先复制源图的形参 `Mat` 数据到临时变量，定义一些临时变量，再处理 `ddepth` 小于零的情况，接着处理 `borderType` 不为 `BORDER_CONSTANT` 且 `normalize` 为真的情况，最终调用 `FilterEngine` 滤波引擎创建一个 `BoxFilter`，正式开始滤波操作。

这里的 `FilterEngine` 是 OpenCV 图像滤波功能的核心引擎，我们有必要详细剖析看其源代码。

## 2. FilterEngine 类解析：OpenCV 图像滤波核心引擎

`FilterEngine` 类是 OpenCV 关于图像滤波的主力军类，是 OpenCV 图像滤波功能的核心引擎。各种滤波函数如 `blur`、`GaussianBlur`，其实就是在函数末尾处定义了一个 `Ptr<FilterEngine>` 类型的 `f`，然后 `f->apply(src, dst)` 了一下而已。

这个类可以把几乎所有的滤波操作施加到图像上，它包含了所有必要的中间缓存器。有很多和滤波相关的 `create` 系函数的返回值直接就是 `Ptr<FilterEngine>`。比如

- `cv::createSeparableLinearFilter()`

- cv::createLinearFilter(),cv::createGaussianFilter(), cv::createDerivFilter()
- cv::createBoxFilter()
- cv::createMorphologyFilter()

下面给出其中一个函数的原型。

```
Ptr<FilterEngine>createLinearFilter(int srcType, int dstType,
InputArray kernel, Point_anchor=Point(-1,-1), double delta=0, int
rowBorderType=BORDER_DEFAULT, int columnBorderType=-1, const Scalar&
borderValue=Scalar() )
```

上面提到过，其中的 `Ptr` 是用来动态分配的对象的智能指针模板类，而尖括号里的模板参数就是 `FilterEngine`。

使用 `FilterEngine` 类可以分块处理大量的图像，构建复杂的管线，其中就包含一些进行滤波阶段。如果我们需要使用预先定义好的的滤波操作，有 `cv::filter2D()`、`cv::erode()` 和 `cv::dilate()` 可以选择，它们不依赖于 `FilterEngine`，在自己函数体内部就实现了 `FilterEngine` 提供的功能；不像其他诸如我们今天讲的 `blur` 系列函数，依赖于 `FilterEngine` 引擎。

我们看下其类声明经过详细注释的源码，如下。

```
class CV_EXPORTS FilterEngine
{
public:
    //默认构造函数
    FilterEngine();
    //完整的构造函数。_filter2D、_rowFilter 和 _columnFilter 之一，必须为非空
    FilterEngine(const Ptr<BaseFilter>& _filter2D,
                 const Ptr<BaseRowFilter>& _rowFilter,
                 const Ptr<BaseColumnFilter>& _columnFilter,
                 int srcType, int dstType, int bufType,
                 int _rowBorderType=BORDER_REPLICATE,
                 int _columnBorderType=-1,
                 const Scalar& _borderValue=Scalar());
    //默认析构函数
    virtual ~FilterEngine();
    //重新初始化引擎。释放之前滤波器申请的内存
    void init(const Ptr<BaseFilter>& _filter2D,
              const Ptr<BaseRowFilter>& _rowFilter,
              const Ptr<BaseColumnFilter>& _columnFilter,
              int srcType, int dstType, int bufType,
              int _rowBorderType=BORDER_REPLICATE, int
              _columnBorderType=-1,
              const Scalar& _borderValue=Scalar());
    //开始对指定了 ROI 区域和尺寸的图片进行滤波操作
    virtual int start(Size wholeSize, Rect roi, int maxBufRows=-1);
    //开始对指定了 ROI 区域的图片进行滤波操作
    virtual int start(const Mat& src, const Rect& srcRoi=Rect(0,0,-1,-1),
                     bool isolated=false, int maxBufRows=-1);
    //处理图像的下一个 srcCount 行（函数的第三个参数）
```

```

virtual int proceed(const uchar* src, int srcStep, int srcCount,
                    uchar* dst, int dstStep);
//对图像指定的 ROI 区域进行滤波操作，若 srcRoi=(0,0,-1,-1)，则对整个图像进行
滤波操作

virtual void apply( const Mat& src, Mat& dst,
                     const Rect& srcRoi=Rect(0,0,-1,-1),
                     Point dstOfs=Point(0,0),
                     bool isolated=false);

//如果滤波器可分离，则返回 true
bool isSeparable() const { return (const BaseFilter*)filter2D == 0; }

//返回输入和输出行数
int remainingInputRows() const;
int remainingOutputRows() const;

//一些成员参数定义
int srcType, dstType, bufType;
Size ksize;
Point anchor;
int maxWidth;
Size wholeSize;
Rect roi;
int dx1, dx2;
int rowBorderType, columnBorderType;
vector<int> borderTab;
int borderElemSize;
vector<uchar> ringBuf;
vector<uchar> srcRow;
vector<uchar> constBorderValue;
vector<uchar> constBorderRow;
int bufStep, startY, startY0, endY, rowCount, dstY;
vector<uchar*> rows;

Ptr<BaseFilter> filter2D;
Ptr<BaseRowFilter> rowFilter;
Ptr<BaseColumnFilter> columnFilter;
};


```

### 3. OpenCV 中 blur 函数源码剖析

我们可以在 OpenCV 的安装路径 \sources\modules\imgproc\src 下的 smooth.cpp 源文件中找到 blur 的源代码，下面一起来看 OpenCV 中 blur 函数定义的真面目。

```

void cv::blur(InputArray src, OutputArray dst,
              Size ksize, Point anchor, int borderType )
{
//调用 boxFilter 函数进行处理
    boxFilter( src, dst, -1, ksize, anchor, true, borderType );
}

```

可以发现，在 blur 函数内部调用了一个 boxFilter 函数，且第 6 个参数为 true，也就是我们上文所说的 normalize=true，即均值滤波是均一化后的方框滤波。

### 6.1.10 OpenCV 中 GaussianBlur 函数源码剖析

下面我们看一下 OpenCV 中 GaussianBlur 函数源代码：

```
void cv::GaussianBlur( InputArray _src, OutputArray _dst, Size ksize,
                      double sigma1, double sigma2,
                      int borderType )
{
    //复制形参 Mat 数据到临时变量，用于稍后的操作
    Mat src = _src.getMat();
    _dst.create( src.size(), src.type() );
    Mat dst = _dst.getMat();

    //处理边界选项不为 BORDER_CONSTANT 时的情况
    if( borderType != BORDER_CONSTANT )
    {
        if( src.rows == 1 )
            ksize.height = 1;
        if( src.cols == 1 )
            ksize.width = 1;
    }

    //若 ksize 长宽都为 1，将源图复制给目标图
    if( ksize.width == 1 && ksize.height == 1 )
    {
        src.copyTo(dst);
        return;
    }

    //若之前有过 HAVE_TEGRA_OPTIMIZATION 优化选项的定义，则执行宏体中的 tegra 优化
    //版函数并返回
#ifdef HAVE_TEGRA_OPTIMIZATION
    if(sigma1 == 0 && sigma2 == 0 && tegra::gaussian(src,dst, ksize,
    borderType))
        return;
#endif

    //如果 HAVE_IPP&& (IPP_VERSION_MAJOR >= 7 为真，则执行宏体中语句
#ifndef HAVE_IPP
    if(defined HAVE_IPP &&(IPP_VERSION_MAJOR >= 7)
        if(src.type() == CV_32FC1 && sigma1 == sigma2 &&ksize.width ==
        ksize.height && sigma1 != 0.0 )
    {
        IppiSize roi = {src.cols, src.rows};
        int bufSize = 0;
       ippiFilterGaussGetBufferSize_32f_C1R(roi, ksize.width,
&bufSize);
        AutoBuffer<uchar> buf(bufSize+128);
        if( ippifilterGaussBorder_32f_C1R((const Ipp32f
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

*) src.data, (int)src.step,
    (Ipp32f *)dst.data, (int)dst.step,
    roi,ksize.width, (Ipp32f)sigma1,
    (IppiBorderType)borderType, 0.0,
    alignPtr(&buf[0],32)) >= 0 )

return;
}

#endif

//调用滤波引擎，正式进行高斯滤波操作
Ptr<FilterEngine> f = createGaussianFilter( src.type(), ksize,sigma1,
sigma2, borderType );
f->apply( src, dst );
}

```

通过本节的源码解析，相信大家应该对 OpenCV 中的线性滤波有了比较详细的认识，已经跃跃欲试，想看这个几个函数用起来可以得出什么效果了。

### 6.1.11 线性滤波核心 API 函数

本节的内容就是为了大家能快速上手 boxFilter、blur 和 GaussianBlur 这三个函数所准备的。下面进行详细讲解。

#### 1. 方框滤波：boxFilter 函数

boxFilter 的函数作用是使用方框滤波（box filter）来模糊一张图片，由 src 输入，dst 输出。

函数原型如下。

```
C++: void boxFilter(InputArray src,OutputArray dst, int ddepth, Size
ksize, Point anchor=Point(-1,-1), boolnormalize=true, int
borderType=BORDER_DEFAULT )
```

参数详解如下。

- 第一个参数：InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可。该函数对通道是独立处理的，且可以处理任意通道数的图片。但需要注意，待处理的图片深度应该是 CV\_8U、CV\_16U、CV\_16S、CV\_32F、CV\_64F 之一。
- 第二个参数：OutputArray 类型的 dst，即目标图像，需要和源图片有一样的尺寸和类型。
- 第三个参数：int 类型的 ddepth，输出图像的深度。“-1”代表使用原图深度，即 src.depth()。
- 第四个参数：Size 类型的 ksize，内核的大小。一般用 Size(w,h)的写法来表示内核的大小（其中，w 为像素宽度，h 为像素高度）。例如，Size(3,3)表示 3x3 的核大小；Size(5,5)就表示 5x5 的核大小。
- 第五个参数：Point 类型的 anchor，表示锚点（即被平滑的那个点），注意它有默认值 Point(-1,-1)。如果这个点坐标是负值的话，就表示取核的中心

为锚点，所以默认值 Point(-1,-1) 表示这个锚点在核的中心。

- 第六个参数：bool 类型的 normalize，默认值为 true，一个标识符，表示内核是否被其区域归一化（normalized）了。
- 第七个参数：int 类型的 borderType，用于推断图像外部像素的某种边界模式。有默认值 BORDER\_DEFAULT，我们一般不去管它。

调用代码示范如下。

```
//载入原图  
Mat image=imread("2.jpg");  
//进行均值滤波操作  
Mat out;  
boxFilter(image, out, -1,Size(5, 5));
```

用上面三句核心代码架起来的完整程序代码如下。

```
-----【头文件、命名空间包含部分】-----  
//      描述：包含程序所依赖的头文件和命名空间  
-----  
#include "opencv2/core/core.hpp"  
#include "opencv2/highgui/highgui.hpp"  
#include "opencv2/imgproc/imgproc.hpp"  
  
using namespace cv;  
  
-----【main()函数】-----  
//      描述：控制台应用程序的入口函数，我们的程序从这里开始  
-----  
int main()  
{  
    //载入原图  
    Mat image=imread("2.jpg");  
  
    //创建窗口  
    namedWindow("均值滤波【原图】");  
    namedWindow("均值滤波【效果图】");  
  
    //显示原图  
    imshow("均值滤波【原图】", image );  
  
    //进行滤波操作  
    Mat out;  
    boxFilter(image, out, -1,Size(5, 5));  
  
    //显示效果图  
    imshow("均值滤波【效果图】", out );  
  
    waitKey(0 );  
}
```

此程序运行效果图（内核大小为 Size(5,5)）如图 6.2 所示。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

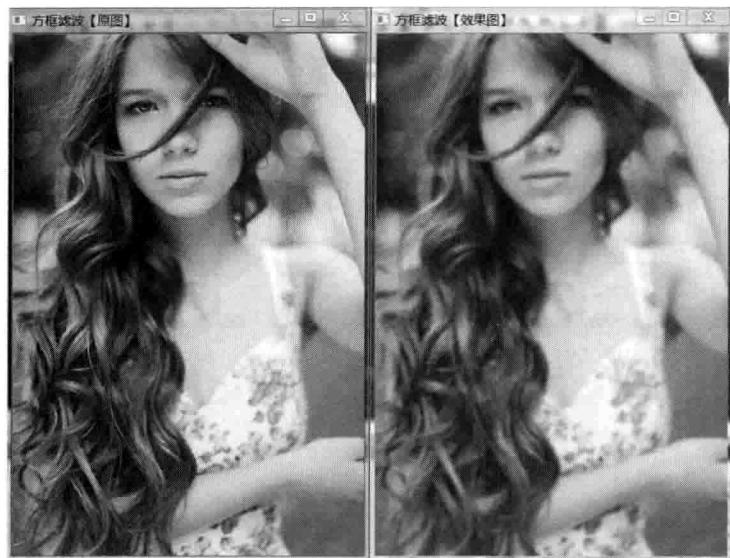


图 6.2 运行效果图（内核大小 Size(5, 5)）

## 2. 均值滤波：blur 函数

blur 的作用是对输入的图像 src 进行均值滤波后用 dst 输出。

函数原型如下。

```
C++: void blur(InputArray src, OutputArray dst, Size ksize, Point anchor=Point(-1,-1), int borderType=BORDER_DEFAULT )
```

参数详解如下。

- 第一个参数：InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可。该函数对通道是独立处理的，且可以处理任意通道数的图片。但需要注意，待处理的图片深度应该为 CV\_8U、CV\_16U、CV\_16S、CV\_32F、CV\_64F 之一。
- 第二个参数：OutputArray 类型的 dst，即目标图像，需要和源图片有一样的尺寸和类型。比如可以用 Mat::Clone，以源图片为模板，来初始化得到如假包换的目标图。
- 第三个参数：Size 类型（对 Size 类型稍后有讲解）的 ksize，内核的大小。一般用 Size(w,h)的写法来表示内核的大小（其中 w 为像素宽度，h 为像素高度）。例如，Size(3,3)表示 3x3 的核大小，Size(5,5)就表示 5x5 的核大小。
- 第四个参数：Point 类型的 anchor，表示锚点（即被平滑的那个点），注意它有默认值 Point(-1,-1)。如果这个点坐标是负值的话，就表示取核的中心为锚点，所以默认值 Point(-1,-1)表示这个锚点在核的中心。
- 第五个参数：int 类型的 borderType，用于推断图像外部像素的某种边界模式。有默认值 BORDER\_DEFAULT，我们一般不去管它。

调用代码示范如下。

```
//载入原图  
Mat image=imread("1.jpg");  
//进行均值滤波操作  
Mat out;  
blur(image, out, Size(7, 7));
```

用上面三句核心代码架起来的完整程序代码如下。

```
#include "opencv2/core/core.hpp"  
#include "opencv2/highgui/highgui.hpp"  
#include "opencv2/imgproc/imgproc.hpp"  
using namespace cv;  
  
int main()  
{  
    //载入原图  
    Mat image=imread("1.jpg");  
  
    //创建窗口  
    namedWindow("均值滤波【原图】");  
    namedWindow("均值滤波【效果图】");  
  
    //显示原图  
    imshow("均值滤波【原图】", image);  
  
    //进行滤波操作  
    Mat out;  
    blur(image, out, Size(7, 7));  
  
    //显示效果图  
    imshow("均值滤波【效果图】", out);  
  
    waitKey(0);  
}
```

此程序运行效果（内核大小 Size(7,7)）如图 6.3 所示。



图 6.3 运行效果图（内核大小 Size(7,7)）

### 3. 高斯滤波：GaussianBlur 函数

GaussianBlur 函数的作用是用高斯滤波器来模糊一张图片，对输入的图像 src 进行高斯滤波后用 dst 输出。

函数原型如下。

```
C++: void GaussianBlur(InputArray src, OutputArray dst, Size ksize,  
double sigmaX, double sigmaY=0, int borderType=BORDER_DEFAULT )
```

参数详解如下。

- 第一个参数：InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可。它可以是单独的任意通道数的图片。但需要注意，图片深度应该为 CV\_8U、CV\_16U、CV\_16S、CV\_32F、CV\_64F 之一。
- 第二个参数：OutputArray 类型的 dst，即目标图像，需要和源图片有一样的尺寸和类型。比如可以用 Mat::Clone，以源图片为模板，来初始化得到如假包换的目标图。
- 第三个参数：Size 类型的 ksize 高斯内核的大小。其中 ksize.width 和 ksize.height 可以不同，但它们都必须为正数和奇数，也可以为零。它们都是由 sigma 计算而来的。
- 第四个参数：double 类型的 sigmaX，表示高斯核函数在 X 方向的标准偏差。
- 第五个参数：double 类型的 sigmaY，表示高斯核函数在 Y 方向的标准偏差。若 sigmaY 为零，就将它设为 sigmaX，如果 sigmaX 和 sigmaY 都是 0，那么就由 ksize.width 和 ksize.height 计算出来。为了结果的正确性着想，最好是把第三个参数 Size、第四个参数 sigmaX 和第五个参数 sigmaY 全部指定到。
- 第六个参数：int 类型的 borderType，用于推断图像外部像素的某种边界模式。注意它有默认值 BORDER\_DEFAULT。

调用示例如下。

```
//载入原图  
Mat image=imread("1.jpg");  
//进行滤波操作  
Mat out;  
GaussianBlur( image, out, Size( 5, 5 ), 0, 0 );
```

用上面三句核心代码架起来的完整程序代码如下。

```
-----【头文件、命名空间包含部分】-----  
//      描述：包含程序所依赖的头文件和命名空间  
-----  
#include "opencv2/core/core.hpp"  
#include "opencv2/highgui/highgui.hpp"  
#include "opencv2/imgproc/imgproc.hpp"  
using namespace cv;  
  
-----【main()函数】-----
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----
int main()
{
    //载入原图
    Mat image=imread("1.jpg");

    //创建窗口
    namedWindow("高斯滤波【原图】");
    namedWindow("高斯滤波【效果图】");

    //显示原图
    imshow("高斯滤波【原图】", image);

    //进行均值滤波操作
    Mat out;
    GaussianBlur(image, out, Size( 3, 3 ), 0, 0 );

    //显示效果图
    imshow("均值滤波【效果图】", out);

    waitKey(0);
}
```

此程序运行效果（内核大小 Size(5,5)）如图 6.4 所示。



图 6.4 运行效果图（内核大小 Size(5,5)）

### 6.1.12 图像线性滤波综合示例

本节我们将学习一个综合性示例程序，把前文介绍的知识点以代码为载体，展现给大家。

这个示例程序中可以用滑动条来控制三种线性滤波的核参数值。通过滑动滚动条，就可以控制图像在三种线性滤波下的模糊度，有一定的可玩性。详细注释

的代码如下。

```
-----【头文件、命名空间包含部分】-----
//    描述：包含程序所依赖的头文件和命名空间
-----
#include <opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
using namespace std;
using namespace cv;

-----【全局变量声明部分】-----
//    描述：全局变量声明
-----
Mat g_srcImage,g_dstImage1,g_dstImage2,g_dstImage3;//存储图片的 Mat 类型
int g_nBoxFilterValue=3; //方框滤波参数值
int g_nMeanBlurValue=3; //均值滤波参数值
int g_nGaussianBlurValue=3; //高斯滤波参数值

-----【全局函数声明部分】-----
//    描述：全局函数声明
-----
//轨迹条的回调函数
static void on_BoxFilter(int, void *);           //方框滤波
static void on_MeanBlur(int, void *);             //均值滤波
static void on_GaussianBlur(int, void *);          //高斯滤波

-----【main() 函数】-----
//    描述：控制台应用程序的入口函数，我们的程序从这里开始
-----
int main()
{
    //改变 console 字体颜色
    system("color5E");

    //载入原图
    g_srcImage= imread( "1.jpg", 1 );
    if(!g_srcImage.data ) { printf("读取 srcImage 错误~! \n"); return
false; }

    //复制原图到三个 Mat 类型中
    g_dstImage1= g_srcImage.clone( );
    g_dstImage2= g_srcImage.clone( );
    g_dstImage3= g_srcImage.clone( );

    //显示原图
    namedWindow("【<0>原图窗口】", 1);
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
imshow("【<0>原图窗口】", g_srcImage);

//=====【<1>方框滤波】=====
//创建窗口
namedWindow("【<1>方框滤波】", 1);
//创建轨迹条
createTrackbar("内核值: ", "【<1>方框滤波】", &g_nBoxFilterValue,
40, on_BoxFilter );
on_MeanBlur(g_nBoxFilterValue, 0);
imshow("【<1>方框滤波】", g_dstImage1);
//=====

//=====【<2>均值滤波】=====
//创建窗口
namedWindow("【<2>均值滤波】", 1);
//创建轨迹条
createTrackbar("内核值: ", "【<2>均值滤波】", &g_nMeanBlurValue,
40, on_MeanBlur );
on_MeanBlur(g_nMeanBlurValue, 0);
//=====

//=====【<3>高斯滤波】=====
//创建窗口
namedWindow("【<3>高斯滤波】", 1);
//创建轨迹条
createTrackbar("内核值: ", "【<3>高斯滤波】", &g_nGaussianBlurValue,
40, on_GaussianBlur );
on_GaussianBlur(g_nGaussianBlurValue, 0);
//=====

//输出一些帮助信息
cout<<endl<<"\t, 请调整滚动条观察图像效果~\n\n"
<<"\t按下\"q\"键时, 程序退出~!\n" ;

//按下"q"键时, 程序退出
while(char(waitKey(1))!= 'q') {}

return 0;
}

//-----【on_BoxFilter() 函数】-----
//    描述: 方框滤波操作的回调函数
//-----
static void on_BoxFilter(int, void *
{
    //方框滤波操作
    boxFilter(g_srcImage, g_dstImage1, -1, Size(g_nBoxFilterValue+1,
g_nBoxFilterValue+1));
    //显示窗口
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
imshow("【<1>方框滤波】", g_dstImage1);
}

-----【on_MeanBlur() 函数】-----
//    描述：均值滤波操作的回调函数
-----
static void on_MeanBlur(int, void *)
{
    //均值滤波操作
    blur(g_srcImage, g_dstImage2, Size( g_nMeanBlurValue+1,
g_nMeanBlurValue+1), Point(-1,-1));
    //显示窗口
    imshow("【<2>均值滤波】", g_dstImage2);
}

-----【on_GaussianBlur() 函数】-----
//    描述：高斯滤波操作的回调函数
-----
static void on_GaussianBlur(int, void *)
{
    //高斯滤波操作
    GaussianBlur(g_srcImage, g_dstImage3,
Size( g_nGaussianBlurValue*2+1,g_nGaussianBlurValue*2+1 ), 0, 0);
    //显示窗口
    imshow("【<3>高斯滤波】", g_dstImage3);
}
```

代码注释已经十分详细了，最后一起看一下程序运行得到的效果，如图 6.6、6.7、6.8 所示。原图如图 6.5 所示。



图 6.5 原始图



图 6.6 方框滤波效果图



图 6.7 均值滤波效果图

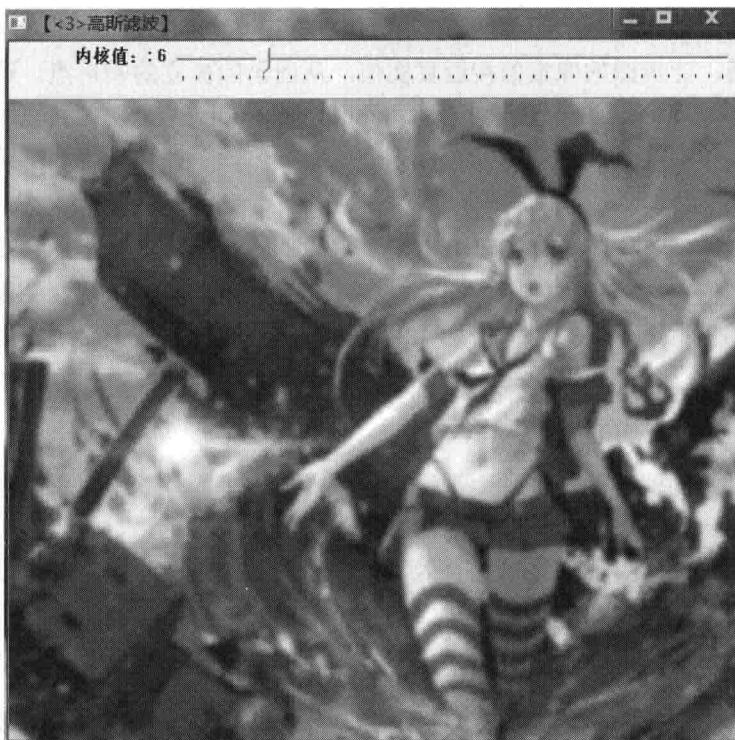


图 6.8 高斯滤波效果图

## 6.2 非线性滤波：中值滤波、双边滤波

正如我们在 6.1 节中讲到的，线性滤波可以实现很多种不同的图像变换。而非线性滤波，如中值滤波器和双边滤波器，有时可以达到更好的实现效果。

### 6.2.1 非线性滤波概述

在 6.1 节中，我们所考虑的滤波器都是线性的，即两个信号之和的响应和它们各自响应之和相等。换句话说，每个像素的输出值是一些输入像素的加权和。线性滤波器易于构造，并且易于从频率响应角度来进行分析。

然而，在很多情况下，使用邻域像素的非线性滤波会得到更好的效果。比如在噪声是散粒噪声而不是高斯噪声，即图像偶尔会出现很大的值的时候，用高斯滤波器对图像进行模糊的话，噪声像素是不会被去除的，它们只是转换为更为柔和但仍然可见的散粒。这就到了中值滤波登场的时候了。

### 6.2.2 中值滤波

中值滤波（Median filter）是一种典型的非线性滤波技术，基本思想是用像素点邻域灰度值的中值来代替该像素点的灰度值，该方法在去除脉冲噪声、椒盐噪声的同时又能保留图像的边缘细节。

中值滤波是基于排序统计理论的一种能有效抑制噪声的非线性信号处理技

术，其基本原理是把数字图像或数字序列中一点的值用该点的一个邻域中各点值的中值代替，让周围的像素值接近真实值，从而消除孤立的噪声点。这对于斑点噪声（speckle noise）和椒盐噪声（salt-and-pepper noise）来说尤其有用，因为它不依赖于邻域内那些与典型值差别很大的值。中值滤波器在处理连续图像窗函数时与线性滤波器的工作方式类似，但滤波过程却不再是加权运算。

中值滤波在一定的条件下可以克服常见线性滤波器，如最小均方滤波、方框滤波器、均值滤波等带来的图像细节模糊，而且对滤除脉冲干扰及图像扫描噪声非常有效，也常用于保护边缘信息。保存边缘的特性使它在不希望出现边缘模糊的场合也很有用，是非常经典的平滑噪声处理方法。

- 中值滤波与均值滤波器比较

**优势：**在均值滤波器中，由于噪声成分被放入平均计算中，所以输出受到了噪声的影响。但是在中值滤波器中，由于噪声成分很难选上，所以几乎不会影响到输出。因此同样用  $3 \times 3$  区域进行处理，中值滤波消除的噪声能力更胜一筹。中值滤波无论是在消除噪声还是保存边缘方面都是一个不错的方法。

**劣势：**中值滤波花费的时间是均值滤波的 5 倍以上。

顾名思义，中值滤波选择每个像素的邻域像素中的中值作为输出，或者说中值滤波将每一像素点的灰度值设置为该点某邻域窗口内的所有像素点灰度值的中值。

例如，取  $3 \times 3$  的函数窗，计算以点  $[i,j]$  为中心的函数窗像素中值，具体步骤如下。

- (1) 按强度值大小排列像素点。
- (2) 选择排序像素集的中间值作为点  $[i,j]$  的新值。

一般采用奇数点的邻域来计算中值，但像素点数为偶数时，中值就取排序像素中间两点的平均值。采用大小不同邻域的中值滤波器的结果如图 6.9 所示。



图 6.9 中值滤波效果图

中值滤波在一定条件下，可以克服线性滤波器（如均值滤波等）所带来的图像细节模糊，对滤除脉冲干扰即图像扫描噪声最为有效，而且在实际运算过程中并不需要图像的统计特性，也给计算带来不少方便。但是对一些细节（特别是细、尖顶等）多的图像不太适合。

### 6.2.3 双边滤波

双边滤波（Bilateral filter）是一种非线性的滤波方法，是结合图像的空间邻近度和像素值相似度的一种折中处理，同时考虑空域信息和灰度相似性，达到保边去噪的目的，具有简单、非迭代、局部的特点。

双边滤波器的好处是可以做边缘保存（edge preserving）。以往常用维纳滤波或者高斯滤波去降噪，但二者都会较明显地模糊边缘，对于高频细节的保护效果并不明显。双边滤波器顾名思义，比高斯滤波多了一个高斯方差  $\sigma_d$ ，它是基于空间分布的高斯滤波函数，所以在边缘附近，离得较远的像素不会对边缘上的像素值影响太多，这样就保证了边缘附近像素值的保存。但是，由于保存了过多的高频信息，对于彩色图像里的高频噪声，双边滤波器不能够干净地滤掉，只能对于低频信息进行较好地滤波。

在双边滤波器中，输出像素的值依赖于邻域像素值的加权值组合，公式如下。

$$g(i, j) = \frac{\sum_{k,l} f(k, l) \omega(i, j, k, l)}{\sum_{k,l} \omega(i, j, k, l)}$$

而加权系数  $\omega(i, j, k, l)$  取决于定义域核和值域核的乘积。

其中定义域核表示如下。

$$d(i, j, k, l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2}\right)$$

值域核表示如下。

$$r(i, j, k, l) = \exp\left(-\frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2}\right)$$

定义域滤波和值域滤波如图 6.10 所示。

	2	1	0	1	2		0.0	0.0	0.0	0.0	0.2
2	0.1	0.3	0.4	0.3	0.1		0.0	0.0	0.0	0.0	0.2
1	0.3	0.6	0.8	0.6	0.3		0.0	0.0	0.0	0.4	0.8
0	0.4	0.8	1.0	0.8	0.4		0.0	0.0	1.0	0.8	0.4
1	0.3	0.6	0.8	0.6	0.3		0.0	0.2	0.8	0.8	1.0
2	0.1	0.3	0.4	0.3	0.1		0.2	0.4	1.0	0.8	0.4

定义域滤波

值域滤波

图 6.10 定义域滤波和值域滤波

两者相乘后，就会产生依赖于数据的双边滤波权重函数，如下。

$$\omega(i, j, k, l) = \exp\left(-\frac{(i-k)^2 + (j-l)^2}{2\sigma_d^2} - \frac{\|f(i, j) - f(k, l)\|^2}{2\sigma_r^2}\right)$$

## 6.2.4 非线性滤波相关核心 API 函数

### 1. 中值滤波：medianBlur 函数

medianBlur 函数使用中值滤波器来平滑（模糊）处理一张图片，从 src 输入，结果从 dst 输出。对于多通道图片，它对每一个通道都单独进行处理，并且支持就地操作（In-place operation）。函数原型如下。

```
C++: void medianBlur(InputArray src, OutputArray dst, int ksize)
```

参数详解如下。

- 第一个参数，InputArray 类型的 src，函数的输入参数，填 1、3 或者 4 通道的 Mat 类型的图像。当 ksize 为 3 或者 5 的时候，图像深度需为 CV\_8U、CV\_16U、CV\_32F 其中之一，而对于较大孔径尺寸的图片，它只能是 CV\_8U。
- 第二个参数：OutputArray 类型的 dst，即目标图像，函数的输出参数，需要和源图片有一样的尺寸和类型。我们可以用 Mat::Clone，以源图片为模板，来初始化得到如假包换的目标图。
- 第三个参数：int 类型的 ksize，孔径的线性尺寸（aperture linear size），注意这个参数必须是大于 1 的奇数，比如：3、5、7、9……

调用范例如下。

```
//载入原图
Mat image=imread("1.jpg");
//进行中值滤波操作
Mat out;
medianBlur( image, out, 7 );
```

用上面三句核心代码架起来的完整程序代码如下。

```
-----【头文件、命名空间包含部分】-----
//    描述：包含程序所依赖的头文件和命名空间
//-----[main() 函数]-----
//    描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----[main()]
{
    //载入原图
```

```
Mat image=imread("1.jpg");

//创建窗口
namedWindow("中值滤波【原图】");
namedWindow("中值滤波【效果图】");

//显示原图
imshow("中值滤波【原图】", image);

//进行中值滤波操作
Mat out;
medianBlur( image, out, 7);

//显示效果图
imshow("中值滤波【效果图】", out);

waitKey(0);
}
```

运行效果图（孔径的线性尺寸为 7）如图 6.11 所示。

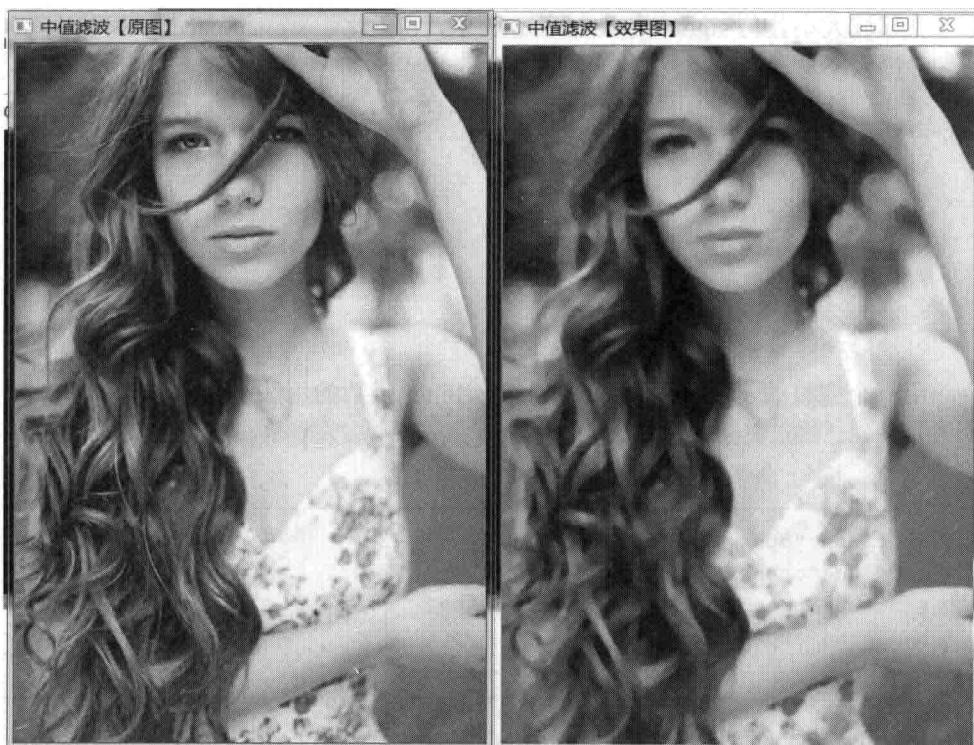


图 6.11 运行效果图（孔径线性尺寸为 7）

## 2. 双边滤波：bilateralFilter 函数

此函数的作用是用双边滤波器来模糊处理一张图片，由 src 输入图片，结果于 dst 输出。函数原型如下。

C++: void bilateralFilter(InputArray src, OutputArray dst, int d, double

```
sigmaColor, double sigmaSpace, int borderType=BORDER_DEFAULT)
```

参数详解如下。

- 第一个参数：InputArray 类型的 src，输入图像，即源图像，需要为 8 位或者浮点型单通道、三通道的图像。
- 第二个参数：OutputArray 类型的 dst，即目标图像，需要和源图片有一样的尺寸和类型。
- 第三个参数：int 类型的 d，表示在过滤过程中每个像素邻域的直径。如果这个值被设为非正数，那么 OpenCV 会从第五个参数 sigmaSpace 来计算出它。
- 第四个参数：double 类型的 sigmaColor，颜色空间滤波器的 sigma 值。这个参数的值越大，就表明该像素邻域内有越宽广的颜色会被混合到一起，产生较大的半相等颜色区域。
- 第五个参数：double 类型的 sigmaSpace，坐标空间中滤波器的 sigma 值，坐标空间的标注方差。它的数值越大，意味着越远的像素会相互影响，从而使更大的区域中足够相似的颜色获取相同颜色。当  $d > 0$  时，d 指定了邻域大小且与 sigmaSpace 无关。否则，d 正比于 sigmaSpace。
- 第六个参数：int 类型的 borderType，用于推断图像外部像素的某种边界模式。注意它有默认值 BORDER\_DEFAULT。

调用代码示范如下。

```
//载入原图  
Mat image=imread("1.jpg");  
//进行双边滤波操作  
Mat out;  
bilateralFilter( image, out, 25, 25*2, 25/2 );
```

用一个完整的示例程序来说明 bilateralFilter 函数的用法，具体如下。

```
-----【头文件、命名空间包含部分】-----  
//    描述：包含程序所依赖的头文件和命名空间  
-----  
#include "opencv2/core/core.hpp"  
#include "opencv2/highgui/highgui.hpp"  
#include "opencv2/imgproc/imgproc.hpp"  
using namespace cv;  
  
-----【main()函数】-----  
//    描述：控制台应用程序的入口函数，我们的程序从这里开始  
-----  
int main()  
{  
    //载入原图  
    Mat image=imread("1.jpg");  
  
    //创建窗口  
    namedWindow("双边滤波【原图】" );
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
namedWindow("双边滤波【效果图】");

//显示原图
imshow("双边滤波【原图】", image);

//进行双边滤波操作
Mat out;
bilateralFilter( image, out, 25, 25*2, 25/2 );

//显示效果图
imshow("双边滤波【效果图】", out);

waitKey(0);
}
```

运行效果图如图 6.12 所示。



图 6.12 双边滤波运行效果图

### 6.2.5 OpenCV 中的 5 种图像滤波综合示例

这一小节会将前文介绍的知识点以代码为载体，展现给大家。

在下面这个示例程序中，可以用滑动条来控制我们学习到的各种滤波（方框滤波、均值滤波、高斯滤波、中值滤波、双边滤波）的参数值，通过滑动滚动条，就可以控制图像在各种平滑处理下的模糊度，有一定的可玩性。详细注释的完整代码如下。

```
//-----【头文件、命名空间包含部分】-----
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
//      描述：包含程序所依赖的头文件和命名空间包含
//-----
#include <opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>
#include <iostream>
using namespace std;
using namespace cv;

//-----【全局变量声明部分】-----
//      描述：全局变量声明
//-----
Mat
g_srcImage,g_dstImage1,g_dstImage2,g_dstImage3,g_dstImage4,g_dstImage5;
int g_nBoxFilterValue=6; //方框滤波内核值
int g_nMeanBlurValue=10; //均值滤波内核值
int g_nGaussianBlurValue=6; //高斯滤波内核值
int g_nMedianBlurValue=10; //中值滤波参数值
int g_nBilateralFilterValue=10; //双边滤波参数值

//-----【全局函数声明部分】-----
//      描述：全局函数声明
//-----
//轨迹条回调函数
static void on_BoxFilter(int, void *);           //方框滤波
static void on_MeanBlur(int, void *);             //均值块滤波器
static void on_GaussianBlur(int, void *);          //高斯滤波器
static void on_MedianBlur(int, void *);            //中值滤波器
static void on_BilateralFilter(int, void*);        //双边滤波器

//-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----
int main( )
{
    system("color 5E");

    //载入原图
    g_srcImage= imread( "1.jpg", 1 );
    if(!g_srcImage.data ) { printf("读取 srcImage 错误~! \n"); return false; }

    //复制原图到 4 个 Mat 类型中
    g_dstImage1= g_srcImage.clone( );
    g_dstImage2= g_srcImage.clone( );
    g_dstImage3= g_srcImage.clone( );
    g_dstImage4= g_srcImage.clone( );
```

```
g_dstImage5= g_srcImage.clone( );  
  
//显示原图  
namedWindow("【<0>原图窗口】", 1);  
imshow("【<0>原图窗口】", g_srcImage);  
  
//=====【<1>方框滤波】=====  
//创建窗口  
namedWindow("【<1>方框滤波】", 1);  
//创建轨迹条  
createTrackbar("内核值: ", "【<1>方框滤波】", &g_nBoxFilterValue,  
50, on_BoxFilter );  
on_MeanBlur(g_nBoxFilterValue, 0);  
imshow("【<1>方框滤波】", g_dstImage1);  
//=====  
  
//=====【<2>均值滤波】=====  
//创建窗口  
namedWindow("【<2>均值滤波】", 1);  
//创建轨迹条  
createTrackbar("内核值: ", "【<2>均值滤波】", &g_nMeanBlurValue,  
50, on_MeanBlur );  
on_MeanBlur(g_nMeanBlurValue, 0);  
//=====  
  
//=====【<3>高斯滤波】=====  
//创建窗口  
namedWindow("【<3>高斯滤波】", 1);  
//创建轨迹条  
createTrackbar("内核值: ", "【<3>高斯滤波】", &g_nGaussianBlurValue,  
50, on_GaussianBlur );  
on_GaussianBlur(g_nGaussianBlurValue, 0);  
//=====  
  
//=====【<4>中值滤波】=====  
//创建窗口  
namedWindow("【<4>中值滤波】", 1);  
//创建轨迹条  
createTrackbar("参数值: ", "【<4>中值滤波】", &g_nMedianBlurValue,  
50, on_MedianBlur );  
on_MedianBlur(g_nMedianBlurValue, 0);  
//=====  
  
//=====【<5>双边滤波】=====  
//创建窗口  
namedWindow("【<5>双边滤波】", 1);  
//创建轨迹条
```

```
createTrackbar("参数值: ", "【<5>双边滤波】"
", &g_nBilateralFilterValue, 50, on_BilateralFilter);
on_BilateralFilter(g_nBilateralFilterValue, 0);
//=====
while(char(waitKey(1))!= 'q') {}

return 0;
}

-----【 on_BoxFilter() 函数】-----
//      描述: 方框滤波操作的回调函数
//-----
static void on_BoxFilter(int, void *)
{
    //方框滤波操作
    boxFilter(g_srcImage, g_dstImage1, -1, Size( g_nBoxFilterValue+1,
g_nBoxFilterValue+1));
    //显示窗口
    imshow("【<1>方框滤波】", g_dstImage1);
}

-----【 on_MeanBlur() 函数】-----
//      描述: 均值滤波操作的回调函数
//-----
static void on_MeanBlur(int, void *)
{
    blur(g_srcImage, g_dstImage2, Size( g_nMeanBlurValue+1,
g_nMeanBlurValue+1), Point(-1,-1));
    imshow("【<2>均值滤波】", g_dstImage2);
}

-----【 on_GaussianBlur() 函数】-----
//      描述: 高斯滤波操作的回调函数
//-----
static void on_GaussianBlur(int, void *)
{
    GaussianBlur(g_srcImage, g_dstImage3,
Size( g_nGaussianBlurValue*2+1,g_nGaussianBlurValue*2+1 ), 0, 0);
    imshow("【<3>高斯滤波】", g_dstImage3);
}

-----【 on_MedianBlur() 函数】-----
//      描述: 中值滤波操作的回调函数
//-----
static void on_MedianBlur(int, void *)
{
    medianBlur( g_srcImage, g_dstImage4, g_nMedianBlurValue*2+1 );
    imshow("【<4>中值滤波】", g_dstImage4);
}
```

```
-----【on_BilateralFilter() 函数】-----  
//      描述：双边滤波操作的回调函数  
-----  
static void on_BilateralFilter(int, void *)  
{  
    bilateralFilter( g_srcImage, g_dstImage5,  
g_nBilateralFilterValue,  
g_nBilateralFilterValue*2,g_nBilateralFilterValue/2 );  
    imshow("【<5>双边滤波】", g_dstImage5);  
}
```

下面是程序运行的原图和效果图示例，如图 6.13 至 6.18 所示。



图 6.13 原始图



图 6.14 方框滤波效果图



图 6.15 均值滤波效果图



图 6.16 高斯滤波效果图



图 6.17 中值滤波效果图

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>



图 6.18 双边滤波效果图

观察各种截图可以发现，方框滤波和均值滤波效果很相似，中值滤波对原图颠覆很大，而双边滤波和原图差别不大，要仔细看才看得出效果。

## 6.3 形态学滤波 (1): 腐蚀与膨胀

本小节中，我们将一起探究图像处理中最基本的形态学运算——膨胀与腐蚀。

### 6.3.1 形态学概述

形态学 (morphology) 一词通常表示生物学的一个分支，该分支主要研究动植物的形态和结构。而我们图像处理中的形态学，往往指的是数学形态学。下面一起来了解数学形态学的概念。

数学形态学 (Mathematical morphology) 是一门建立在格论和拓扑学基础之上的图像分析学科，是数学形态学图像处理的基本理论。其基本的运算包括：二值腐蚀和膨胀、二值开闭运算、骨架抽取、极限腐蚀、击中击不中变换、形态学梯度、Top-hat 变换、颗粒分析、流域变换、灰值腐蚀和膨胀、灰值开闭运算、灰值形态学梯度等。

简单来讲，形态学操作就是基于形状的一系列图像处理操作。OpenCV 为进行图像的形态学变换提供了快捷、方便的函数。最基本的形态学操作有两种，分别是：膨胀 (dilate) 与腐蚀 (erode)。

膨胀与腐蚀能实现多种多样的功能，主要如下。

- 消除噪声；
- 分割 (isolate) 出独立的图像元素，在图像中连接 (join) 相邻的元素；
- 寻找图像中的明显的极大值区域或极小值区域；
- 求出图像的梯度。

在这里给出下文会用于对比膨胀与腐蚀运算的“浅墨”字样毛笔字原图，如

图 6.19 所示。



图 6.19 “浅墨”字样毛笔字原图

在进行腐蚀和膨胀的讲解之前，首先提醒大家注意，腐蚀和膨胀是对白色部分（高亮部分）而言的，不是黑色部分。膨胀是图像中的高亮部分进行膨胀，类似于“领域扩张”，效果图拥有比原图更大的高亮区域；腐蚀是原图中的高亮部分被腐蚀，类似于“领域被蚕食”，效果图拥有比原图更小的高亮区域。

### 6.3.2 膨胀

膨胀（dilate）就是求局部最大值的操作。从数学角度来说，膨胀或者腐蚀操作就是将图像（或图像的一部分区域，称之为 A）与核（称之为 B）进行卷积。

核可以是任何形状和大小，它拥有一个单独定义出来的参考点，我们称其为锚点（anchorpoint）。多数情况下，核是一个小的，中间带有参考点和实心正方形或者圆盘。其实，可以把核视为模板或者掩码。

而膨胀就是求局部最大值的操作。核 B 与图形卷积，即计算核 B 覆盖的区域的像素点的最大值，并把这个最大值赋值给参考点指定的像素。这样就会使图像中的高亮区域逐渐增长，如图 6.20 所示。这就是膨胀操作的初衷。

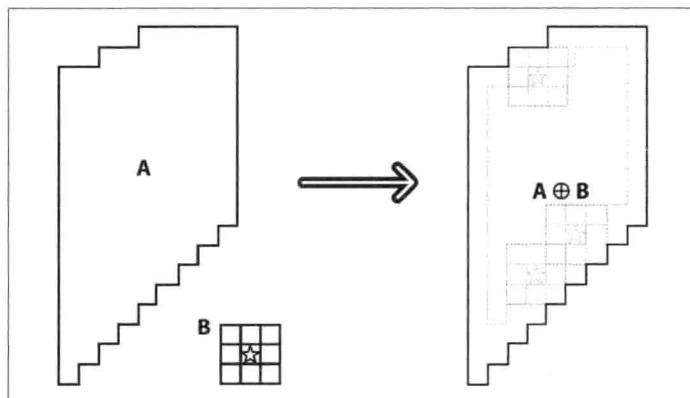


图 6.20 膨胀操作示例图

膨胀的数学表达式如下。

$$dst(x, y) = \max_{(x', y'): \text{ element}(x', y') \neq 0} \text{src}(x+x', y+y')$$

图 6.21 和图 6.22 分别是毛笔字和照片膨胀的效果示例。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>



图 6.21 膨胀效果图（毛笔字）

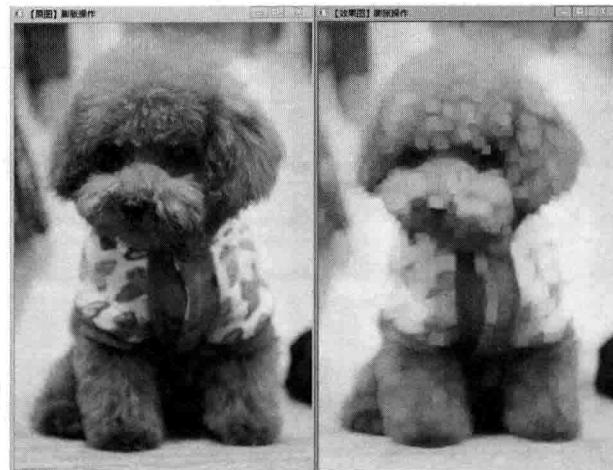


图 6.22 膨胀效果图（照片）

### 6.3.3 腐蚀

大家应该知道，膨胀和腐蚀（erode）是相反的一对操作，所以腐蚀就是求局部最小值的操作。

我们一般都会把腐蚀和膨胀进行对比理解和学习。下文就可以看到，两者的函数原型也是基本一样的。腐蚀操作示例如图 6.23 所示。

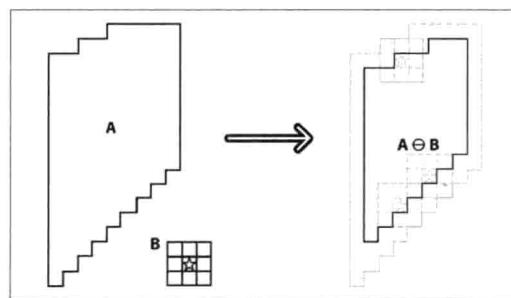


图 6.23 腐蚀操作示例图

腐蚀的数学表达式如下。

$$dst(x, y) = \min_{(x', y') : \text{element}(x', y') \neq 0} \text{src}(x+x', y+y')$$

图 6.24、图 6.25 分别给出了毛笔字和照片的腐蚀效果示例。

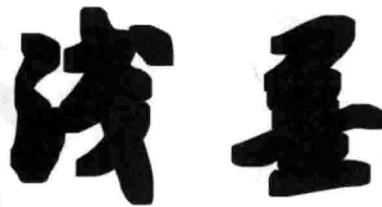


图 6.24 腐蚀效果图（毛笔字）



图 6.25 腐蚀效果图（照片）

### 6.3.4 相关 OpenCV 源码分析溯源

在…\opencv\sources\modules\imgproc\src\morph.cpp 路径中，我们可以发现 erode（腐蚀）函数和 dilate（膨胀）函数的源码，如下。

```
void cv::erode( InputArray src, OutputArray dst, InputArray kernel,
               Point anchor, int iterations,
               int borderType, constScalar& borderColor )
{
    //调用 morphOp 函数，并设定标识符为 MORPH_ERODE
    morphOp( MORPH_ERODE, src, dst, kernel, anchor, iterations,
             borderType, borderColor );
}

void cv::dilate( InputArray src, OutputArray dst, InputArray kernel,
                Point anchor, int iterations,
                int borderType, constScalar& borderColor )
{
    //调用 morphOp 函数，并设定标识符为 MORPH_DILATE
    morphOp( MORPH_DILATE, src, dst, kernel, anchor, iterations,
             borderType, borderColor );
}
```

可以发现，erode 和 dilate 这两个函数内部就是调用了一下 morphOp，只是它

们调用 morphOp 时, 第一个参数标识符不同: 一个为 MORPH\_ERODE (腐蚀), 一个为 MORPH\_DILATE (膨胀)。

morphOp 函数的源码在...\\opencv\\sources\\modules\\imgproc\\src\\morph.cpp 中的第 1286 行, 有兴趣的朋友们可以自行研究, 这里就不再展开分析了。

### 6.3.5 相关核心 API 函数讲解

#### 1. 膨胀: dilate 函数

dilate 函数使用像素邻域内的局部极大运算符来膨胀一张图片, 从 src 输入, 由 dst 输出。支持就地 (in-place) 操作。

函数原型如下。

```
C++: void dilate(
    InputArray src,
    OutputArray dst,
    InputArray kernel,
    Point anchor=Point(-1,-1),
    int iterations=1,
    int borderType=BORDER_CONSTANT,
    const Scalar& borderColor=morphologyDefaultBorderValue()
);
```

参数详解如下。

- 第一个参数, InputArray 类型的 src, 输入图像, 即源图像, 填 Mat 类的对象即可。图像通道的数量可以是任意的, 但图像深度应为 CV\_8U、CV\_16U、CV\_16S、CV\_32F 或 CV\_64F 其中之一。
- 第二个参数, OutputArray 类型的 dst, 即目标图像, 需要和源图片有一样的尺寸和类型。
- 第三个参数, InputArray 类型的 kernel, 膨胀操作的核。当为 NULL 时, 表示的是使用参考点位于中心  $3 \times 3$  的核。

我们一般使用函数 getStructuringElement 配合这个参数的使用。getStructuringElement 函数会返回指定形状和尺寸的结构元素 (内核矩阵)。其中, getStructuringElement 函数的第一个参数表示内核的形状, 有如下三种形状可以选择。

- 矩形: MORPH\_RECT;
- 交叉形: MORPH\_CROSS;
- 椭圆形: MORPH\_ELLIPSE。

而 getStructuringElement 函数的第二和第三个参数分别是内核的尺寸以及锚点的位置。

一般在调用 erode 以及 dilate 函数之前, 先定义一个 Mat 类型的变量来获得 getStructuringElement 函数的返回值。对于锚点的位置, 有默认值 Point(-1,-1), 表

示锚点位于中心。此外，需要注意，十字形的 element 形状唯一依赖于锚点的位置，而在其他情况下，锚点只是影响了形态学运算结果的偏移。

`getStructuringElement` 函数相关的调用示例代码如下。

```
int g_nStructElementSize = 3; //结构元素(内核矩阵)的尺寸

//获取自定义核
Mat element = getStructuringElement(MORPH_RECT,
    Size(2*g_nStructElementSize+1,2*g_nStructElementSize+1),
    Point(g_nStructElementSize, g_nStructElementSize));
```

调用之后，我们可以在接下来调用 `erode` 或 `dilate` 函数时，在第三个参数填保存了 `getStructuringElement` 返回值的 `Mat` 类型变量。对应于上面的示例，就是 `element` 变量。

- 第四个参数，`Point` 类型的 `anchor`，锚的位置，其有默认值(-1,-1)，表示锚位于中心。
- 第五个参数，`int` 类型的 `iterations`，迭代使用 `erode()` 函数的次数，默认值为 1。
- 第六个参数，`int` 类型的 `borderType`，用于推断图像外部像素的某种边界模式。注意它有默认值 `BORDER_DEFAULT`。
- 第七个参数，`const Scalar&`类型的 `borderValue`，当边界为常数时的边界值，有默认值 `morphologyDefaultBorderValue()`，一般不用去管它。需要用到它时，可以看官方文档中的 `createMorphologyFilter()` 函数，以得到更详细的解释。

使用 `erode` 函数，一般只需要填前面的三个参数，后面的四个参数都有默认值，而且往往会结合 `getStructuringElement` 一起使用。

下面给出一个调用范例。

```
//载入原图
Mat image = imread("1.jpg");
//获取自定义核
    Mat element = getStructuringElement(MORPH_RECT, Size(15, 15));
    Mat out;
    //进行膨胀操作
    dilate(image, out, element);
```

用上面核心代码架起来的完整程序代码如下。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
-----
#include <opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>
#include <iostream>

-----【命名空间声明部分】-----
```

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
//      描述: 包含程序所使用的命名空间
//-----
using namespace std;
using namespace cv;

//-----【 main() 函数 】-----
//      描述: 控制台应用程序的入口函数, 我们的程序从这里开始
//-----
int main( )
{
    //载入原图
    Mat image = imread("1.jpg");

    //创建窗口
    namedWindow("【 原图 】膨胀操作");
    namedWindow("【 效果图 】膨胀操作");

    //显示原图
    imshow("【 原图 】膨胀操作", image);

    //获取自定义核
    Mat element = getStructuringElement(MORPH_RECT, Size(15, 15));
    Mat out;
    //进行膨胀操作
    dilate(image, out, element);

    //显示效果图
    imshow("【 效果图 】膨胀操作", out);

    waitKey(0);

    return 0;
}
```

此程序运行截图见前文中图 6.22。

## 2. 腐蚀: erode 函数

erode 函数使用像素邻域内的局部极小运算符来腐蚀一张图片, 从 src 输入, 由 dst 输出。支持就地 (in-place) 操作。

看一下函数原型, 如下。

```
C++: void erode(
    InputArray src,
    OutputArray dst,
    InputArray kernel,
    Point anchor=Point(-1,-1),
    int iterations=1,
    int borderType=BORDER_CONSTANT,
    const Scalar& borderColor=morphologyDefaultBorderValue()
);
```

参数详解如下。

- 第一个参数，`InputArray`类型的`src`，输入图像，即源图像，填`Mat`类的对象即可。图像通道的数量可以是任意的，但图像深度应为`CV_8U`、`CV_16U`、`CV_16S`、`CV_32F`或`CV_64F`其中之一。
- 第二个参数，`OutputArray`类型的`dst`，即目标图像，需要和源图片有一样的尺寸和类型。
- 第三个参数，`InputArray`类型的`kernel`，腐蚀操作的内核。为`NULL`时，表示的是使用参考点位于中心 $3\times 3$ 的核。一般使用函数`getStructuringElement`配合这个参数的使用。`getStructuringElement`函数会返回指定形状和尺寸的结构元素（内核矩阵，具体看上文中`dilate`函数的第三个参数讲解部分）。
- 第四个参数，`Point`类型的`anchor`，锚的位置。其有默认值 $(-1, -1)$ ，表示锚位于单位`(element)`的中心，一般不用管它。
- 第五个参数，`int`类型的`iterations`，迭代使用`erode()`函数的次数，默认值为1。
- 第六个参数，`int`类型的`borderType`，用于推断图像外部像素的某种边界模式。注意它有默认值`BORDER_DEFAULT`。
- 第七个参数，`const Scalar&`类型的`borderValue`，当边界为常数时的边界值，有默认值`morphologyDefaultBorderValue()`，一般不用去管它。需要用到它时，可以看官方文档中的`createMorphologyFilter()`函数以得到更详细的解释。

同样的，使用`erode`函数，一般只需要填前面的三个参数，后面的四个参数都有默认值。而且往往结合`getStructuringElement`一起使用。

调用范例如下。

```
//载入原图
Mat image = imread("1.jpg");
//获取自定义核
Mat element = getStructuringElement(MORPH_RECT, Size(15, 15));
Mat out;
//进行腐蚀操作
erode(image,out, element);
```

用上面核心代码架起来的完整程序代码如下。

```
-----【头文件、命名空间包含部分】-----
//    描述：包含程序所依赖的头文件和命名空间
-----
#include <opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>
#include <iostream>
using namespace std;
using namespace cv;
```

```
-----【 main() 函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
-----
int main()
{
    //载入原图
    Matimage = imread("1.jpg");

    //创建窗口
    namedWindow("【原图】腐蚀操作");
    namedWindow("【效果图】腐蚀操作");

    //显示原图
    imshow("【原图】腐蚀操作", image);

    //获取自定义核
    Mat element = getStructuringElement(MORPH_RECT, Size(15, 15));
    Mat out;

    //进行腐蚀操作
    erode(image, out, element);

    //显示效果图
    imshow("【效果图】腐蚀操作", out);

    waitKey(0);

    return 0;
}
```

此程序运行截图见前文中图 6.25。

### 6.3.6 综合示例：腐蚀与膨胀

此示例程序中的效果图窗口中有两个滑动条第一个滑动条“腐蚀/膨胀”用于在腐蚀/膨胀之间进行切换；第二个滚动条“内核尺寸”用于调节形态学操作时的内核尺寸，以得到效果不同的图像，有一定的可玩性。详细注释的示例程序代码如下。

```
-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
-----
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>
#include <iostream>
using namespace std;
using namespace cv;
```

```
-----【全局变量声明部分】-----
//      描述：全局变量声明
-----

Mat g_srcImage, g_dstImage;//原始图和效果图
int g_nTrackbarNumer = 0;//0表示腐蚀 erode, 1 表示膨胀 dilate
int g_nStructElementSize = 3; //结构元素(内核矩阵)的尺寸


-----【全局函数声明部分】-----
//      描述：全局函数声明
-----

void Process(); //膨胀和腐蚀的处理函数
void on_TrackbarNumChange(int, void *); //回调函数
void on_ElementSizeChange(int, void *); //回调函数


-----【main()函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----

int main()
{
    //改变 console 字体颜色
    system("color5E");

    //载入原图
    g_srcImage= imread("1.jpg");
    if(!g_srcImage.data ) { printf("读取 srcImage 错误~! \n"); return
false; }

    //显示原始图
    namedWindow("【原始图】");
    imshow("【原始图】", g_srcImage);

    //进行初次腐蚀操作并显示效果图
    namedWindow("【效果图】");
    //获取自定义核
    Mat element = getStructuringElement(MORPH_RECT,
Size(2*g_nStructElementSize+1,2*g_nStructElementSize+1),Point( g_nSt
ructElementSize, g_nStructElementSize ));
    erode(g_srcImage,g_dstImage, element);
    imshow("【效果图】", g_dstImage);

    //创建轨迹条
    createTrackbar("腐蚀/膨胀", "【效果图】", &g_nTrackbarNumer, 1,
on_TrackbarNumChange);
    createTrackbar("内核尺寸", "【效果图】",&g_nStructElementSize, 21,
on_ElementSizeChange);

    //轮询获取按键信息，若下 q 键，程序退出
    while(char(waitKey(1))!= 'q') { }
```

Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

```
        return 0;
    }

//-----【 Process() 函数】-----
//      描述：进行自定义的腐蚀和膨胀操作
//-----
void Process()
{
    //获取自定义核
    Mat element = getStructuringElement(MORPH_RECT,
Size(2*g_nStructElementSize+1,2*g_nStructElementSize+1),Point( g_nSt
ructElementSize, g_nStructElementSize ));

    //进行腐蚀或膨胀操作
    if(g_nTrackbarNumer== 0) {
        erode(g_srcImage,g_dstImage, element);
    }
    else{
        dilate(g_srcImage,g_dstImage, element);
    }

    //显示效果图
    imshow("【效果图】", g_dstImage);
}

//-----【 on_TrackbarNumChange() 函数】-----
//      描述：腐蚀和膨胀之间切换开关的回调函数
//-----
void on_TrackbarNumChange(int, void *)
{
    //腐蚀和膨胀之间效果已经切换，回调函数体内需调用一次 Process 函数，使改变
    //后的效果立即生效并显示出来
    Process();
}

//-----【 on_ElementSizeChange() 函数】-----
//      描述：腐蚀和膨胀操作内核改变时的回调函数
//-----
void on_ElementSizeChange(int, void *)
{
    //内核尺寸已改变，回调函数体内需调用一次 Process 函数，使改变后的效果立即
    //生效并显示出来
    Process();
}
```

运行截图分别如图 6.26、图 6.27、图 6.28 所示。



图 6.26 原始图



图 6.27 膨胀效果图



图 6.28 腐蚀效果图

由于篇幅所限，截图就先展示这些，更多的运行效果请大家自行运行书本配套示例程序进行学习和观赏。

## 6.4 形态学滤波(2): 开运算、闭运算、形态学梯度、顶帽、黑帽

上一节中，我们重点了解了腐蚀和膨胀这两种最基本的形态学操作，而运用这两个基本操作，可以实现更高级的形态学变换。

所以，本节的主角是 OpenCV 中的 morphologyEx 函数，它利用基本的膨胀和腐蚀技术，来执行更加高级的形态学变换，如开闭运算、形态学梯度、“顶帽”、“黑帽”等。

首先，我们需要知道，形态学的高级形态，往往都是建立在腐蚀和膨胀这两个基本操作之上的。而关于腐蚀和膨胀，概念和细节以及相关代码请参考上一小节。对膨胀和腐蚀心中有数了，接下来的高级形态学操作，应该就不难理解。

#### 6.4.1 开运算

开运算（Opening Operation），其实就是先腐蚀后膨胀的过程。其数学表达式如下：

```
dst=open(src,element)=dilate(erode(src,element))
```

开运算可以用来消除小物体，在纤细点处分离物体，并且在平滑较大物体的边界的同时不明显改变其面积。原始图请参考图 6.19，而效果图见图 6.30 和 6.31。



图 6.30 开运算效果图

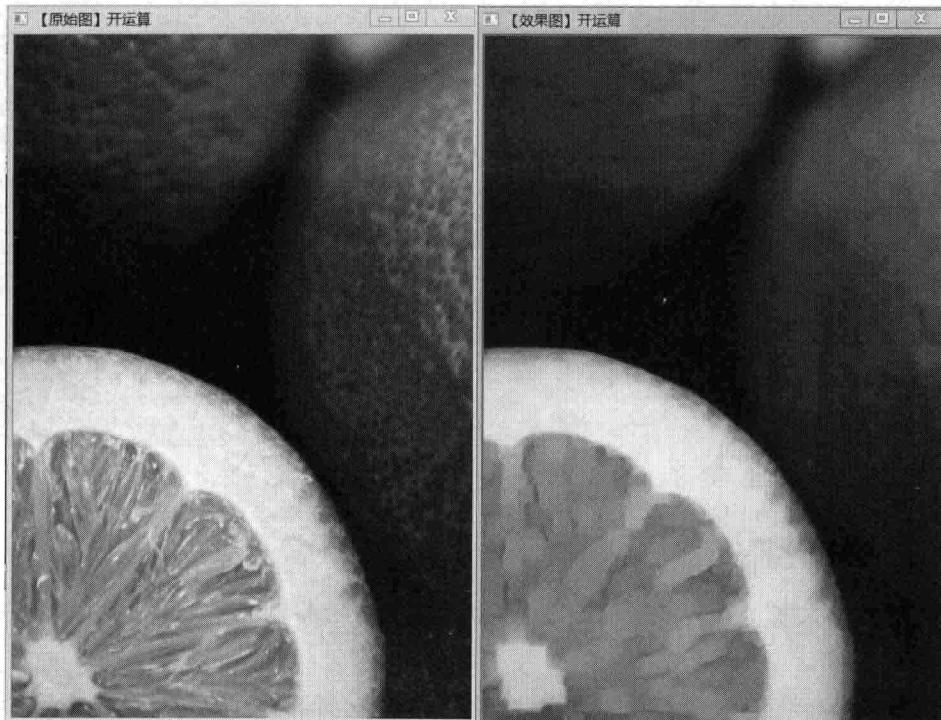


图 6.31 图片开运算效果图

### 6.4.2 闭运算

先膨胀后腐蚀的过程称为闭运算 (Closing Operation)，其数学表达式如下：

```
dst=close(src,element)= erode(dilate(src,element))
```

闭运算能够排除小型黑洞 (黑色区域)。效果图如图 6.32 和图 6.33 所示。



图 6.32 闭运算效果图



图 6.33 图片闭运算效果图

### 6.4.3 形态学梯度

形态学梯度 (Morphological Gradient) 是膨胀图与腐蚀图之差，数学表达式如下：

```
dst=morph-grad(src,element)= dilate(src,element)- erode(src,element)
```

对二值图像进行这一操作可以将团块 (blob) 的边缘突出出来。我们可以用

形态学梯度来保留物体的边缘轮廓，如图 6.34 和图 6.35 所示。



6.34 形态学梯度效果图



图 6.35 图片形态学梯度效果图

#### 6.4.4 顶帽

顶帽运算 (Top Hat) 又常常被译为“礼帽”运算，是原图像与上文刚刚介绍的“开运算”的结果图之差，数学表达式如下：

```
dst=tophat(src,element)=src-open(src,element)
```

因为开运算带来的结果是放大了裂缝或者局部低亮度的区域。因此，从原图中减去开运算后的图，得到的效果图突出了比原图轮廓周围的区域更明亮的区域，且这一操作与选择的核的大小相关。

顶帽运算往往用来分离比邻近点亮一些的斑块。在一幅图像具有大幅的背景，

而微小物品比较有规律的情况下，可以使用顶帽运算进行背景提取。

如图 6.36、图 6.37 所示。

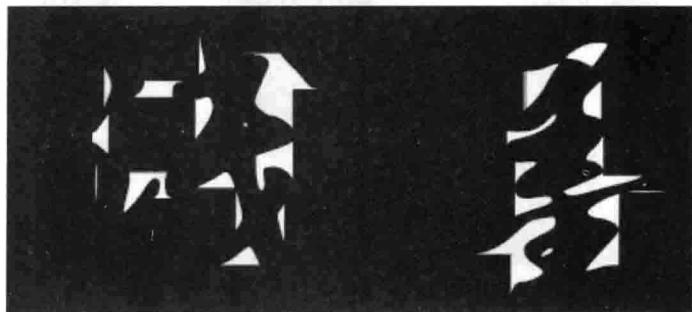


图 6.36 顶帽效果图

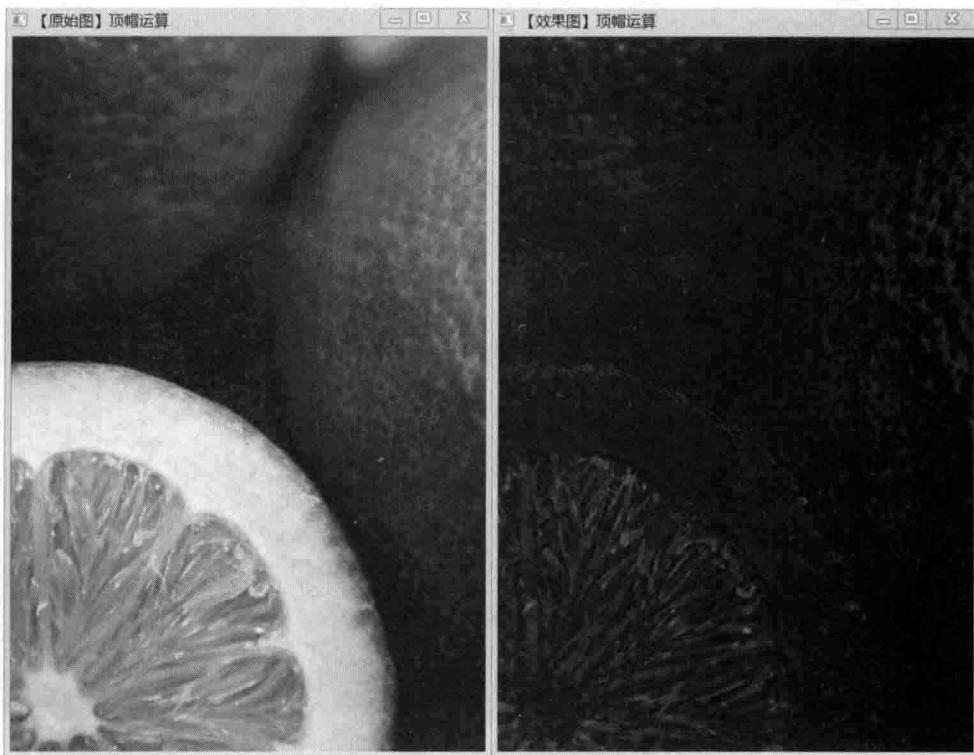


图 6.37 图片顶帽效果图

#### 6.4.5 黑帽

黑帽（Black Hat）运算是闭运算的结果图与原图像之差。数学表达式为：

```
dst=blackhat (src,element)=close(src,element)- src
```

黑帽运算后的效果图突出了比原图轮廓周围的区域更暗的区域，且这一操作和选择的核的大小相关。

所以，黑帽运算用来分离比邻近点暗一些的斑块，效果图有着非常完美的轮廓。示例如图 6.38 和图 6.39 所示。

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>



图 6.38 黑帽效果图

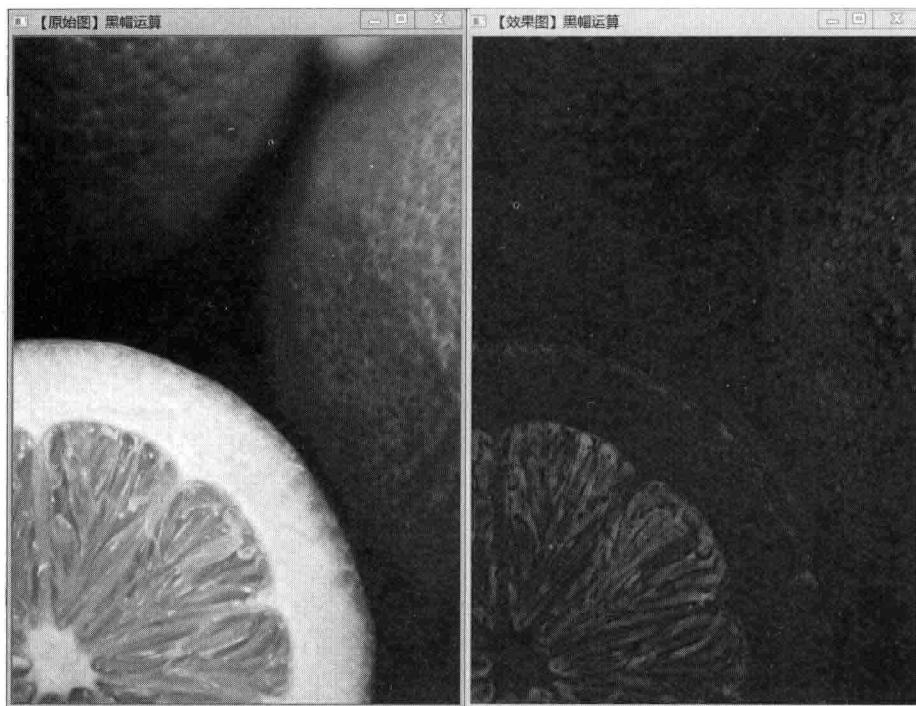


图 6.39 照片黑帽效果图

#### 6.4.6 形态学滤波 OpenCV 源码分析溯源

本节的主角是 OpenCV 中的 `morphologyEx` 函数, 它利用基本的膨胀和腐蚀技术, 来执行更加高级的形态学变换, 如开闭运算、形态学梯度、“顶帽”、“黑帽”等。这一节我们来一起看一下 `morphologyEx` 函数的源代码。

```
void cv::morphologyEx( InputArray _src, OutputArray _dst, int op,
                      InputArray kernel, Point anchor, int iterations,
                      int borderType, const Scalar& borderColor )
{
    //复制 Mat 数据到临时变量
    Mat src = _src.getMat(), temp;
    _dst.create(src.size(), src.type());
    Mat dst = _dst.getMat();

    //一个大 switch, 根据不同的标识符取不同的操作
    switch( op )
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
{  
    case MORPH_ERODE:  
        erode( src, dst, kernel, anchor, iterations, borderType,  
borderValue );  
        break;  
    case MORPH_DILATE:  
        dilate( src, dst, kernel, anchor, iterations, borderType,  
borderValue );  
        break;  
    case MORPH_OPEN:  
        erode( src, dst, kernel, anchor, iterations, borderType,  
borderValue );  
        dilate( dst, dst, kernel, anchor, iterations, borderType,  
borderValue );  
        break;  
    case CV_MOP_CLOSE:  
        dilate( src, dst, kernel, anchor, iterations, borderType,  
borderValue );  
        erode( dst, dst, kernel, anchor, iterations, borderType,  
borderValue );  
        break;  
    case CV_MOP_GRADIENT:  
        erode( src, temp, kernel, anchor, iterations, borderType,  
borderValue );  
        dilate( src, dst, kernel, anchor, iterations, borderType,  
borderValue );  
        dst -= temp;  
        break;  
    case CV_MOP_TOPHAT:  
        if( src.data != dst.data )  
            temp = dst;  
        erode( src, temp, kernel, anchor, iterations, borderType,  
borderValue );  
        dilate( temp, temp, kernel, anchor, iterations, borderType,  
borderValue );  
        dst = src - temp;  
        break;  
    case CV_MOP_BLACKHAT:  
        if( src.data != dst.data )  
            temp = dst;  
        dilate( src, temp, kernel, anchor, iterations, borderType,  
borderValue );  
        erode( temp, temp, kernel, anchor, iterations, borderType,  
borderValue );  
        dst = temp - src;  
        break;  
    default:  
        CV_Error( CV_StsBadArg, "unknown morphological operation" );  
}
```

看上面的源码可以发现，morphologyEx 函数其实就是一个大 switch 而

已，根据不同的标识符取不同的操作。比如开运算 MORPH\_OPEN，按我们上文中讲解的数学表达式，就是先腐蚀后膨胀，即依次调用 erode 和 dilate 函数，代码非常简明干净。

#### 6.4.7 核心 API 函数：morphologyEx()

上面已经讲到，morphologyEx 函数利用基本的膨胀和腐蚀技术，来执行更加高级形态学变换，如开闭运算、形态学梯度、“顶帽”、“黑帽”等。下面我们来详细地讲解它的参数意义和使用方法。

```
C++: void morphologyEx(
    InputArray src,
    OutputArray dst,
    int op,
    InputArray kernel,
    Point anchor=Point(-1,-1),
    int iterations=1,
    int borderType=BORDER_CONSTANT,
    const Scalar& borderColor=morphologyDefaultBorderValue() );
```

- 第一个参数，InputArray 类型的 src，输入图像，即源图像，填 Mat 类的对象即可。图像位深应该为以下 5 种之一：CV\_8U、CV\_16U、CV\_16S、CV\_32F 和 CV\_64F。
- 第二个参数，OutputArray 类型的 dst，即目标图像，函数的输出参数，需要和源图片有一样的尺寸和类型。
- 第三个参数，int 类型的 op，表示形态学运算的类型，可以是如表 6.2 中任意之一的标识符。

表 6.2 morphologyEx 函数可取标识符列举

标识符	含义
MORPH_OPEN	开运算
MORPH_CLOSE	闭运算
MORPH_GRADIENT	形态学梯度
MORPH_TOPHAT	顶帽
MORPH_BLACKHAT	黑帽
MORPH_ERODE	腐蚀
MORPH_DILATE	膨胀

对于 OpenCV2，另有 CV 版本的标识符也可选择，如 CV\_MOP\_CLOSE、CV\_MOP\_GRADIENT、CV\_MOP\_TOPHAT、CV\_MOP\_BLACKHAT，这是 OpenCV1.0 系列版本遗留下来的标识符，在 OpenCV2 中和上面的“MORPH\_OPEN”具有同样的效果，但已经在 OpenCV3 中废止。

- 第四个参数，InputArray 类型的 kernel，形态学运算的内核。若为 NULL，

SimpPDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

表示的是使用参考点位于中心  $3 \times 3$  的核。一般使用函数 `getStructuringElement` 配合这个参数的使用。`getStructuringElement` 函数会返回指定形状和尺寸的结构元素（内核矩阵）。关于 `getStructuringElement` 我们之前有讲到过，这里为了大家参阅方便，再写一遍。

`getStructuringElement` 函数的第一个参数表示内核的形状，我们可以选择如下三种形状之一：

- 矩形——`MORPH_RECT`
- 交叉形——`MORPH_CROSS`
- 椭圆形——`MORPH_ELLIPSE`

而 `getStructuringElement` 函数的第二和第三个参数分别是内核的尺寸以及锚点的位置。

一般在调用 `erode` 以及 `dilate` 函数之前，要先定义一个 `Mat` 类型的变量来获得 `getStructuringElement` 函数的返回值。对于锚点的位置，有默认值 `Point(-1,-1)`，表示锚点位于中心。另外需要注意：十字形的 `element` 形状唯一依赖于锚点的位置。而在其他情况下，锚点只是影响形态学运算结果的偏移。

`getStructuringElement` 函数相关的调用示例代码如下。

```
int g_nStructElementSize = 3; // 结构元素(内核矩阵)的尺寸

// 获取自定义核
Mat element = getStructuringElement(MORPH_RECT,
    Size(2*g_nStructElementSize+1, 2*g_nStructElementSize+1),
    Point(g_nStructElementSize, g_nStructElementSize));
```

之后，便可以在调用 `erode`、`dilate` 或 `morphologyEx` 函数时，由 `kernel` 参数填保存 `getStructuringElement` 返回值的 `Mat` 类型变量。对应于上面的示例，就是填 `element` 变量。

- 第五个参数，`Point` 类型的 `anchor`，锚的位置，其有默认值 `(-1,-1)`，表示锚位于中心。
- 第六个参数，`int` 类型的 `iterations`，迭代使用函数的次数，默认值为 1。
- 第七个参数，`int` 类型的 `borderType`，用于推断图像外部像素的某种边界模式。注意它有默认值 `BORDER_CONSTANT`。
- 第八个参数，`const Scalar&` 类型的 `borderValue`，当边界为常数时的边界值，有默认值 `morphologyDefaultBorderValue()`，一般不用去管它。需要用到它时，可以看官方文档中的 `createMorphologyFilter()` 函数得到更详细的解释。其中的这些操作都可以进行就地 (`in-place`) 操作，且对于多通道图像，每一个通道都单独进行操作。

#### 6.4.8 各形态学操作使用范例一览

核心函数讲解完毕，下面便开始讲解使用范例。为了方便大家需要的时候随

## Simpo PDF Merge and Split Unregistered Version - http://www.simpopdf.com

时取用，这里提供了利用 morphologyEx 函数实现的几乎全部的形态学操作：开运算、闭运算、形态学梯度、顶帽、黑帽、腐蚀、膨胀的效果实现简化版完整代码。其实说白了，这些代码基本上内容一致，就是改一下 morphologyEx 里面的第三个标识符参数而已。核都是选的 MORPH\_RECT（矩形元素结构）。另外，通过观察源代码发现，最基本的腐蚀和膨胀操作也可以用 morphologyEx 函数来实现，它们由 morphologyEx 函数源码中 switch 的前两个 case 来实现（虽然在 case 体内就是简单地各自调用了一下 erode 和 dilation 函数，但还是有写出来的必要）。所以在里面，我们也用 morphologyEx 再重新来实现一遍它们。以下便为实现代码（以形态学梯度为例）。

```
//-----【头文件、命名空间包含部分】-----  
//      描述：包含程序所依赖的头文件和命名空间  
//-----  
#include <opencv2/opencv.hpp>  
#include<opencv2/highgui/highgui.hpp>  
#include<opencv2/imgproc/imgproc.hpp>  
using namespace cv;  
  
//-----【main()函数】-----  
//      描述：控制台应用程序的入口函数，我们的程序从这里开始  
//-----  
int main()  
{  
    //载入原始图  
    Mat image = imread("1.jpg"); //工程目录下应该有一张名为 1.jpg 的素材  
    图  
    //创建窗口  
    namedWindow("【原始图】");  
    namedWindow("【效果图】");  
    //显示原始图  
    imshow("【原始图】", image);  
    //定义核  
    Mat element = getStructuringElement(MORPH_RECT, Size(15, 15));  
    //进行形态学操作  
    morphologyEx(image,image, MORPH_GRADIENT , element);  
    //显示效果图  
    imshow("【效果图】", image);  
  
    waitKey(0);  
  
    return 0;  
}
```

程序运行截图见图 6.35。

上述代码是以形态学梯度为例的程序，而若要选取其他形态学的操作，只需将 morphologyEx 函数中的第三个参数 MORPH\_GRADIENT 替换成表 6.2 中相应的标识符即可。但是方面大家方便查看和使用，在配套示例程序代码包中，还是为大家分别准备了开运算、闭运算、形态学梯度、顶帽、黑帽、腐蚀、膨胀的效

果的实现程序。具体程序在本书的配套代码包中的序号为 41 到 47，读者可以进行快速查阅。

### 6.4.9 综合示例：形态学滤波

这个综合示例程序中，一共会出现 4 个显示图像的窗口，包括原始图一个，开/闭运算为一个，腐蚀/膨胀为一个，顶帽/黑帽运算为一个。它们分别使用滚动条，来控制得到的形态学效果，且迭代值为 10 的时候为中间点。另外，还可以通过键盘按键 1、2、3 以及空格键来调节成不同的元素结构（矩形、椭圆、十字形）。说明页面如图 6.40 所示。

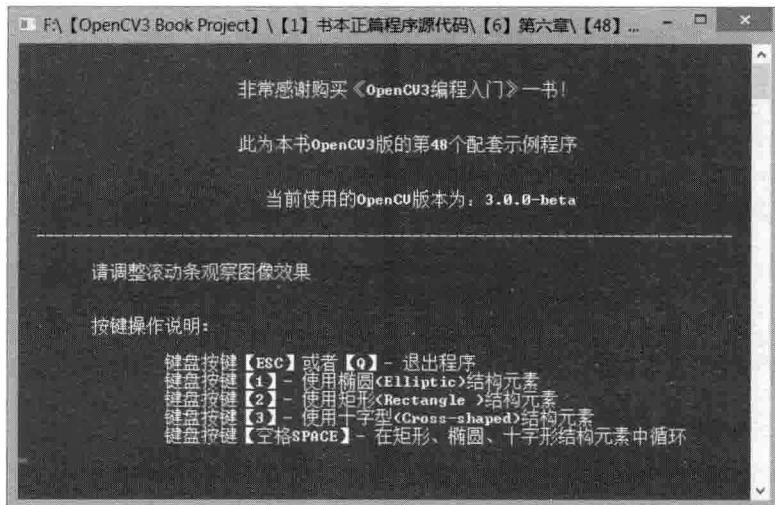


图 6.40 程序说明文字页面

详细注释的程序代码如下。

```

-----【头文件、命名空间包含部分】-----
//      描述：包含程序所依赖的头文件和命名空间
-----

#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace std;
using namespace cv;

-----【全局变量声明部分】-----
//      描述：全局变量声明
Mat g_srcImage, g_dstImage;//原始图和效果图
int g_nElementShape = MORPH_RECT;//元素结构的形状

//变量接收的 TrackBar 位置参数
int g_nMaxIterationNum = 10;
int g_nOpenCloseNum = 0;
int g_nErodeDilateNum = 0;

```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```
int g_nTopBlackHatNum = 0;

//-----【全局函数声明部分】-----
//      描述：全局函数声明
//-----

static void on_OpenClose(int, void*); //回调函数
static void on_ErodeDilate(int, void*); //回调函数
static void on_TopBlackHat(int, void*); //回调函数
static void ShowHelpText(); //帮助文字显示

//-----【main() 函数】-----
//      描述：控制台应用程序的入口函数，我们的程序从这里开始
//-----

int main()
{
    //载入原图
    g_srcImage = imread("1.jpg"); //工程目录下需要有一张名为 1.jpg 的素材图
    if( !g_srcImage.data ) { printf("读取 srcImage 错误~! \n"); return false; }

    //显示原始图
    namedWindow("【原始图】");
    imshow("【原始图】", g_srcImage);

    //创建三个窗口
    namedWindow("【开运算/闭运算】", 1);
    namedWindow("【腐蚀/膨胀】", 1);
    namedWindow("【顶帽/黑帽】", 1);

    //参数赋值
    g_nOpenCloseNum=9;
    g_nErodeDilateNum=9;
    g_nTopBlackHatNum=2;

    //分别为三个窗口创建滚动条
    createTrackbar("迭代值", "【开运算/闭运算】",
        &g_nOpenCloseNum, g_nMaxIterationNum*2+1, on_OpenClose);
    createTrackbar("迭代值", "【腐蚀/膨胀】",
        &g_nErodeDilateNum, g_nMaxIterationNum*2+1, on_ErodeDilate);
    createTrackbar("迭代值", "【顶帽/黑帽】",
        &g_nTopBlackHatNum, g_nMaxIterationNum*2+1, on_TopBlackHat);

    //轮询获取按键信息
    while(1)
    {
        int c;

        //执行回调函数
        on_OpenClose(g_nOpenCloseNum, 0);
    }
}
```

Simpo PDF Merge and Split Unregistered Version - <http://www.simpopdf.com>

```

        on_ErodeDilate(g_nErodeDilateNum, 0);
        on_TopBlackHat(g_nTopBlackHatNum, 0);

        //获取按键
        c = waitKey(0);

        //按下键盘按键 Q 或者 ESC, 程序退出
        if( (char)c == 'q'||(char)c == 27 )
            break;
        //按下键盘按键 1, 使用椭圆(Elliptic)结构元素结构元素 MORPH_ELLIPSE
        if( (char)c == 49 )//键盘按键 1 的 ASII 码为 49
            g_nElementShape = MORPH_ELLIPSE;
        //按下键盘按键 2, 使用矩形(Rectangle)结构元素 MORPH_RECT
        else if( (char)c == 50 )//键盘按键 2 的 ASII 码为 50
            g_nElementShape = MORPH_RECT;
        //按下键盘按键 3, 使用十字形(Cross-shaped)结构元素 MORPH_CROSS
        else if( (char)c == 51 )//键盘按键 3 的 ASII 码为 51
            g_nElementShape = MORPH_CROSS;
        //按下键盘按键 space, 在矩形、椭圆、十字形结构元素中循环
        else if( (char)c == ' ' )
            g_nElementShape = (g_nElementShape + 1) % 3;
    }

    return 0;
}

//-----【on_OpenClose() 函数】-----
//      描述:【开运算/闭运算】窗口的回调函数
//-----
static void on_OpenClose(int, void*)
{
    //偏移量的定义
    int offset = g_nOpenCloseNum - g_nMaxIterationNum;//偏移量
    int Absolute_offset = offset > 0 ? offset : -offset;//偏移量绝对值
    //自定义核
    Mat element = getStructuringElement(g_nElementShape,
Size(Absolute_offset*2+1, Absolute_offset*2+1), Point(Absolute_offset,
Absolute_offset));
    //进行操作
    if( offset < 0 )
        //此句代码的 OpenCV2 版为:
        //morphologyEx(g_srcImage, g_dstImage, CV_MOP_OPEN, element);
        //此句代码的 OpenCV3 版为:
        morphologyEx(g_srcImage, g_dstImage, MORPH_OPEN, element);
    else
        //此句代码的 OpenCV2 版为:
        //morphologyEx(g_srcImage, g_dstImage, CV_MOP_CLOSE, element);
        //此句代码的 OpenCV3 版为:
        morphologyEx(g_srcImage, g_dstImage, MORPH_CLOSE, element);
    //显示图像
    imshow("【开运算/闭运算】", g_dstImage);
}

```

```
}

//-----【on_ErodeDilate() 函数】-----
//    描述:【腐蚀/膨胀】窗口的回调函数
//-----
static void on_ErodeDilate(int, void*)
{
    //偏移量的定义
    int offset = g_nErodeDilateNum - g_nMaxIterationNum;    //偏移量
    int Absolute_offset = offset > 0 ? offset : -offset;//偏移量绝对值
    //自定义核
    Mat element = getStructuringElement(g_nElementShape,
Size(Absolute_offset*2+1, Absolute_offset*2+1), Point(Absolute_offset,
Absolute_offset) );
    //进行操作
    if( offset < 0 )
        erode(g_srcImage, g_dstImage, element);
    else
        dilate(g_srcImage, g_dstImage, element);
    //显示图像
    imshow("【腐蚀/膨胀】",g_dstImage);
}

//-----【on_TopBlackHat() 函数】-----
//    描述:【顶帽运算/黑帽运算】窗口的回调函数
//-----
static void on_TopBlackHat(int, void*)
{
    //偏移量的定义
    int offset = g_nTopBlackHatNum - g_nMaxIterationNum;//偏移量
    int Absolute_offset = offset > 0 ? offset : -offset;//偏移量绝对值
    //自定义核
    Mat element = getStructuringElement(g_nElementShape,
Size(Absolute_offset*2+1, Absolute_offset*2+1), Point(Absolute_offset,
Absolute_offset) );
    //进行操作
    if( offset < 0 )
        morphologyEx(g_srcImage, g_dstImage, MORPH_TOPHAT , element);
    else
        morphologyEx(g_srcImage, g_dstImage, MORPH_BLACKHAT, element);
    //显示图像
    imshow("【顶帽/黑帽】",g_dstImage);
}
```

以上便是经过详细注释的程序代码，相信大家通过上文的学习，可以很好地掌握这些代码的含义和用法。下面放出此程序的运行效果图（图片素材为电影《美国队长 2》的海报）。原图为 6.41，效果图为 6.42、6.43、6.44、6.45、6.46、6.47。



图 6.41 原始图



图 6.42 腐蚀效果图



图 6.43 膨胀效果图



图 6.44 开运算效果图

图 6.45 闭运算效果图

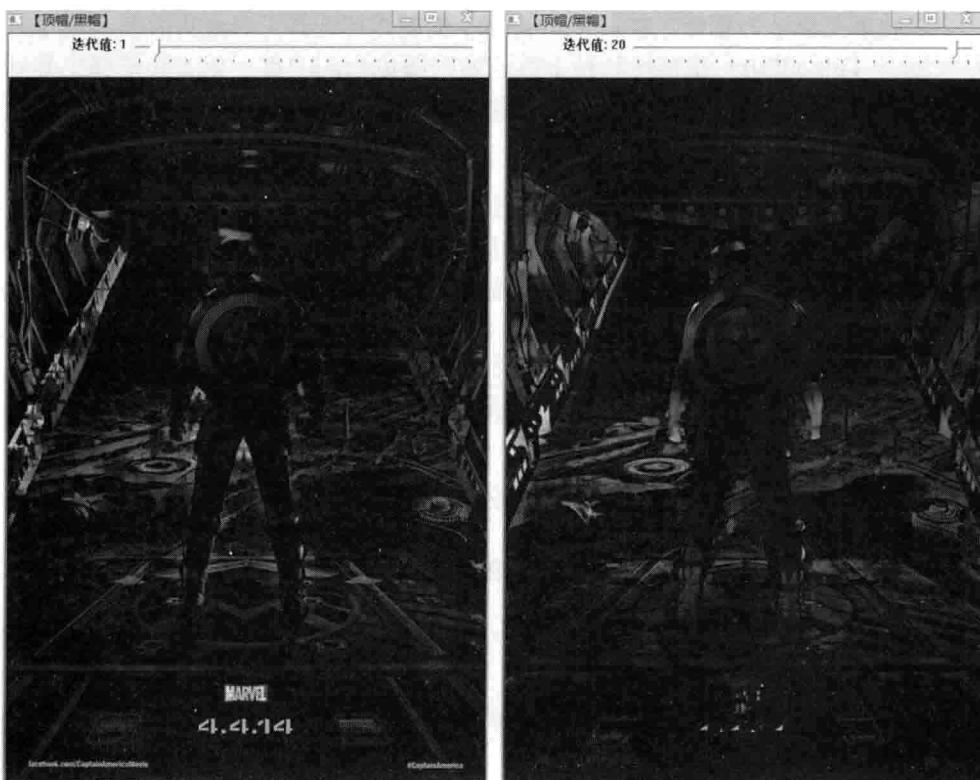


图 6.46 顶帽运算效果图

图 6.47 黑帽运算效果图

## 6.5 漫水填充

本节我们将一起探讨 OpenCV 填充算法中漫水填充算法相关的知识点，并了解 OpenCV 中实现漫水填充算法的两个版本的 `floodFill` 函数的使用方法。

### 6.5.1 漫水填充的定义

漫水填充法是一种用特定的颜色填充连通区域，通过设置可连通像素的上下限以及连通方式来达到不同的填充效果的方法。漫水填充经常被用来标记或分离图像的一部分，以便对其进行进一步处理或分析，也可以用来从输入图像获取掩码区域，掩码会加速处理过程，或只处理掩码指定的像素点，操作的结果总是某个连续的区域。

另外，`floodfill` 官方译作“漫水填充”，但是很多朋友们总是喜欢说成“水漫填充”，因为受从小到大深入骨髓的“水漫金山”这个成语的影响。

### 6.5.2 漫水填充法的基本思想

所谓漫水填充，简单来说，就是自动选中了和种子点相连的区域，接着将该区域替换成指定的颜色，这是个非常有用的功能，经常用来标记或者分离图像的一部分进行处理或分析。漫水填充也可以用来从输入图像获取掩码区域，掩码会加速处理过程，或者只处理掩码指定的像素点。

以此填充算法为基础，类似 PhotoShop 的魔术棒选择工具就很容易实现了。漫水填充（FloodFill）是查找和种子点连通的颜色相同的点，魔术棒选择工具则是查找和种子点连通的颜色相近的点，把和初始种子像素颜色相近的点压进栈做为新种子。

在 OpenCV 中，漫水填充是填充算法中最通用的方法。且在 OpenCV 2.X 中，使用 C++ 重写过的 `FloodFill` 函数有两个版本：一个不带掩膜 `mask` 的版本，和一个带 `mask` 的版本。这个掩膜 `mask`，就是用于进一步控制哪些区域将被填充颜色（比如说当对同一图像进行多次填充时）。这两个版本的 `FloodFill`，都必须在图像中选择一个种子点，然后把临近区域所有相似点填充上同样的颜色，不同的是，不一定将所有的邻近像素点都染上同一颜色，漫水填充操作的结果总是某个连续的区域。当邻近像素点位于给定的范围（从 `loDiff` 到 `upDiff`）内或在原始 `seedPoint` 像素值范围内时，`FloodFill` 函数就会为这个点涂上颜色。

### 6.5.3 实现漫水填充算法：`floodFill` 函数

在 OpenCV 中，漫水填充算法由 `floodFill` 函数实现，其作用是用我们指定的颜色从种子点开始填充一个连接域。连通性由像素值的接近程度来衡量。OpenCV2.X 有两个 C++ 重写版本的 `floodFill`，具体如下。

第一个版本的 `floodFill`:

```
int floodFill(InputOutputArray image, Point seedPoint, Scalar newVal,
```

```
Rect* rect=0, Scalar loDiff=Scalar(), Scalar upDiff=Scalar(), int  
flags=4 )
```

第二个版本的 floodFill:

```
int floodFill(InputOutputArray image, InputOutputArray mask, Point  
seedPoint, Scalar newVal, Rect* rect=0, Scalar loDiff=Scalar(), Scalar  
upDiff=Scalar(), int flags=4 )
```

下面是参数详解。这两个版本除了第二个参数外，其他的参数都是共用的。

(1) 第一个参数，InputOutputArray 类型的 image，输入/输出 1 通道或 3 通道，8 位或浮点图像，具体参数由之后的参数指明。

(2) 第二个参数，InputOutputArray 类型的 mask，这是第二个版本的 floodFill 独享的参数，表示操作掩模。它应该为单通道，8 位，长和宽上都比输入图像 image 大两个像素点的图像。第二个版本的 floodFill 需要使用以及更新掩膜，所以对于这个 mask 参数，我们一定要将其准备好并填在此处。需要注意的是，漫水填充不会填充掩膜 mask 的非零像素区域。例如，一个边缘检测算子的输出可以用来作为掩膜，以防止填充到边缘。同样的，也可以在多次的函数调用中使用同一个掩膜，以保证填充的区域不会重叠。另外需要注意的是，掩膜 mask 会比需填充的图像大，所以 mask 中与输入图像(x,y)像素点相对应的点的坐标为(x+1,y+1)。

(3) 第三个参数，Point 类型的 seedPoint，漫水填充算法的起始点。

(4) 第四个参数，Scalar 类型的 newVal，像素点被染色的值，即在重绘区域像素的新值。

(5) 第五个参数，Rect\*类型的 rect，有默认值 0，一个可选的参数，用于设置 floodFill 函数将要重绘区域的最小边界矩形区域。

(6) 第六个参数，Scalar 类型的 loDiff，有默认值 Scalar()，表示当前观察像素值与其部件邻域像素值或者待加入该部件的种子像素之间的亮度或颜色之负差(lower brightness/color difference) 的最大值。

(7) 第七个参数，Scalar 类型的 upDiff，有默认值 Scalar()，表示当前观察像素值与其部件邻域像素值或者待加入该部件的种子像素之间的亮度或颜色之正差(lower brightness/color difference) 的最大值。

(8) 第八个参数，int 类型的 flags，操作标志符，此参数包含三个部分，比较复杂，我们一起详细看看。

- 低八位(第 0~7 位)用于控制算法的连通性，可取 4(4 为默认值)或者 8。如果设为 4，表示填充算法只考虑当前像素水平方向和垂直方向的相邻点；如果设为 8，除上述相邻点外，还会包含对角线方向的相邻点。
- 高八位部分(16~23 位)可以为 0 或者如下两种选项标识符的组合。
  - ◆ FLOODFILL\_FIXED\_RANGE：如果设置为这个标识符，就会考虑当前像素与种子像素之间的差，否则就考虑当前像素与其相邻像素的差。也就是说，这个范围是浮动的。

- ◆ FLOODFILL\_MASK\_ONLY - 如果设置为这个标识符, 函数不会去填充改变原始图像(也就是忽略第三个参数 newVal), 而是去填充掩模图像(mask)。这个标识符只对第二个版本的 floodFill 有用, 因第一个版本里面压根就没有 mask 参数。
- 中间八位部分, 上面关于高八位 FLOODFILL\_MASK\_ONLY 标识符中已经说得很明显, 需要输入符合要求的掩码。Floodfill 的 flags 参数的中间八位的值就是用于指定填充掩码图像的值的。但如果 flags 中间八位的值为 0, 则掩码会用 1 来填充。

而所有 flags 可以用 or 操作符连接起来, 即 “|”。例如, 如果想用 8 邻域填充, 并填充固定像素值范围, 填充掩码而不是填充源图像, 以及设填充值为 38, 那么输入的参数是下面这样:

```
flags=8 | FLOODFILL_MASK_ONLY | FLOODFILL_FIXED_RANGE | (38<<8)
```

接着, 来看一个关于 Floodfill 的简单的调用范例。

```
-----【头文件、命名空间包含部分】-----
//      描述: 包含程序所依赖的头文件和命名空间
-----
#include <opencv2/opencv.hpp>
#include <opencv2/imgproc/imgproc.hpp>
using namespace cv;

-----【main()函数】-----
//      描述: 控制台应用程序的入口函数, 我们的程序从这里开始
-----

int main()
{
    Mat src = imread("1.jpg");
    imshow("【原始图】", src);
    Rect ccomp;
    floodFill(src, Point(50,300), Scalar(155, 255, 55), &ccomp,
              Scalar(20, 20, 20), Scalar(20, 20, 20));
    imshow("【效果图】", src);
    waitKey(0);
    return 0;
}
```

接着我们看看此程序的运行截图。原图如图 6.48 所示, 效果图如图 6.49 所示。

#### 6.5.4 综合示例: 漫水填充

本次的综合示例为 OpenCV 文档中自带的一个程序。作者对其做了适当的修改并详细注释, 供大家消化理解。

操作说明如图 6.50 所示。