第1章

概述

什么是OpenCV

OpenCV是一个开源的计算机视觉库，可以从*http://opencv.org*获取。1999年，Gary Bradski（加里·布拉德斯基）当时在英特尔任职，怀着通过为计算机视觉和人工智能的从业者提供稳定的基础架构并以此来推动产业发展的美好愿景，他启动了OpenCV项目。OpenCV库用C和C++语言编写，可以在Windows、Linux、Mac OS X等系统运行。同时也在积极开发Python、Java、Matlab以及其他一些语言的接口，将库导入安卓和iOS中为移动设备开发应用。OpenCV自项目成立以来获得了来自英特尔和谷歌的大力支持，尤其需要感谢Itseez[编注1]，该公司完成了早期开发的大部分工作。此后，Arraiy[编注2]加入该项目并负责维护始终开源和免费的OpenCV.org。

OpenCV设计用于进行高效的计算，十分强调实时应用的开发。它由C++语言编写并进行了深度优化，从而可以享受多线程处理的优势。如果希望得到更多在英特尔架构上的自动优化，可以购买英特尔的集成性能基元（IPP）库，该库包含了许多算法领域的底层优化程序。在库安装完毕的情况下OpenCV在运行的时候会自动调用合适的IPP库。从OpenCV 3.0开始，英特尔许可OpenCV研发团队和OpenCV社区拥有一个免费的IPP库的子库（称IPPICV），该子库默认集成在OpenCV中并在运算时发挥效用。

编注1：俄罗斯视觉公司，专门从事计算机视觉算法。2016年5月，英特尔收购该公司，以“帮助英特尔的用户打造创新型深度学习的CV应用，如果自动驾驶、数字安全监控和工业检测”（英特尔物联网总经理Doug Dacies语）。

编注2：团队成员有Ethan Rublee（CEO）、Gary Bradski博士（CTO）、Brendan Dowdle（COO）科学家Michael Tetelman博士、工程师9人（含一只小狗）。

OpenCV的一个目标是提供易于使用的计算机视觉接口，从而帮助人们快速建立精巧的视觉应用。OpenCV库包含从计算机视觉各个领域衍生出来的500多个函数，包括工业产品质量检验、医学图像处理、安保领域、交互操作、相机校正、双目视觉以及机器人学。因为计算机视觉和机器学习经常在一起使用，所以OpenCV也包含一个完备的、具有通用性的机器学习库（ML模块）。这个子库聚焦于统计模式识别以及聚类。ML模块对OpenCV的核心任务（计算机视觉）相当有用，但是这个库也足够通用，可以用于任意机器学习问题。

OpenCV怎么用

许多计算机科学家和经验丰富的程序员多多少少都了解计算机视觉的某些方面，但是很少有人熟谙计算机视觉的每一个应用。比如，很多人了解计算机视觉在安保行业的应用，一些人也知道它在网页端的图像和视频处理中的应用在逐渐增加。但很少有人知道计算机视觉在游戏交互中的应用。同时，也很少有人认识到大部分航空图像和街景图像（比如说谷歌街景）已经大量应用相机校正和图像拼接技术。有一些人略微知道一点视觉在自动监控、无人机或者生物制药分析上的应用，但很少有人知道计算机视觉早已经在制造业普遍使用。事实上，批量制造的所有东西都已经利用计算机视觉在进行某些方面的质检工作了。

OpenCV的开源许可允许任何人利用OpenCV包含的任何组件构建商业产品。你也没有义务开源自己的产品或者对该产品所涉及领域进行反馈和改进，虽然我们希望你这样做。在这种自由许可的影响下，项目有着极其庞大的用户社区，社区用户包括一些来自大公司的员工（IBM、微软、英特尔、索尼、西门子和谷歌等）以及一些研究机构（例如斯坦福大学、麻省理工学院、卡内基梅隆大学、剑桥大学以及法国国家信息与自动化研究所）。项目还有一个雅虎论坛组为用户提供提问和讨论的地方，该论坛组有超过50 000名成员。OpenCV在世界范围内都非常流行，尤其是在中国、日本、俄罗斯、欧洲和以色列有着庞大的用户社区。

自从测试版本在1999年1月发布以来，OpenCV已经广泛用于许多应用、产品以及科研工作中。这些应用包括在卫星和网络地图上拼接图像，图像扫描校准，医学图像的降噪，目标分析，安保以及工业检测系统，自动驾驶和安全系统，制造感知系统，相机校正，军事应用，无人空中、地面、水下航行器。它也被运用于声音和音乐的识别，在这些场景中，视觉识别方法被运用于声音的频谱图像。OpenCV亦是斯坦福大学的机器人斯坦

利（Stanley）[编注1]至关重要的一部分，这个机器人赢得了美国国防部高级研究计划署主持的DARPA机器人挑战赛野外机器人竞速的200万美元大奖[编注2]。

什么是计算机视觉

计算机视觉[注1]这种技术可以将静止图像或视频数据转换为一种决策或新的表示。所有这样的转换都是为了完成某种特定的目的而进行的。输入数据可能包含一些场景信息，例如“相机是搭载在一辆车上的”或者“雷达发现了一米之外有一个目标”。一个新的表示，意思是将彩色图像转换为黑白图像，或者从一个图像序列中消除相机运动所产生的影响。

因为我们是被赋予了视觉的生物，所以很容易误认为“计算机视觉也是一种很简单的任务”。计算机视觉究竟有多困难呢？请说说你是如何从一张图像中观察到一辆车的。你最开始的直觉可能具有很强的误导性。人类的大脑将视觉信号划分为许多通道，好让不同的信息流输入大脑。大脑已经被证明有一套注意力系统，在基于任务的方式上，通过图像的重要部分检验其他区域的估计。在视觉信息流中存在巨量的信息反馈，并且到现在我们对此过程也知之甚少。肌肉控制的感知器和其他所有感官都存在着广泛的相互联系，这让大脑能够利用人在世界上多年生活经验所产生的交叉联想，大脑中的反馈循环将反馈传递到每一个处理过程，包括人体的感知器官（眼睛），通过虹膜从物理上控制光线的量来调节视网膜对物体表面的感知。

然而在机器视觉系统中，计算机会从相机或者硬盘接收栅格状排列的数字，也就是说，最关键的是，机器视觉系统不存在一个预先建立的模式识别机制。没有自动控制焦距和光圈，也不能将多年的经验联系在一起。大部分的视觉系统都还处于一个非常朴素原始的阶段。图1-1展示了一辆汽车。在这张图片中，我们看到后视镜位于驾驶室旁边。但是对于计算机而言，看到的只是按照栅格状排列的数字。所有在栅格中给出的数字还有大量的噪声，所以每个数字只能给我们提供少量的信息，但是这个数字栅格就是计算机所能够“看见”的全部了。我们的任务变成将这个带有噪声的数字栅格转换为感知结果“后视镜”。图1-2给出了为什么计算机视觉如此困难的另一些解释。

编注1：2005年，这辆自动驾驶汽车成功越野行驶了212公里，在无人驾驶机器人挑战赛中第一个到达终点。

编注2：DARPPA机器人挑战赛（DRC）是机器人领域的一项重大赛事，堪称“机器人的奥林匹克”。时隔12年，雷锋网独家采访了当事人（本书作者之一）Adrian，文章标题为“重访2005 DARPA挑战赛斯坦福车队成员：Stanley夺冠背后那些不为人知的细节”。

注1：计算机视觉是一个广阔的领域。本书将帮助读者对该领域建立一个最基础的认识。同时我们也推荐Trucco[Trucco98]以获得更全面的介绍、Faugeras[Faugeras93]作为综合参考以及Hartley[Hartley06]和Faugeras[Faugeras93]关于三维视觉的专著。

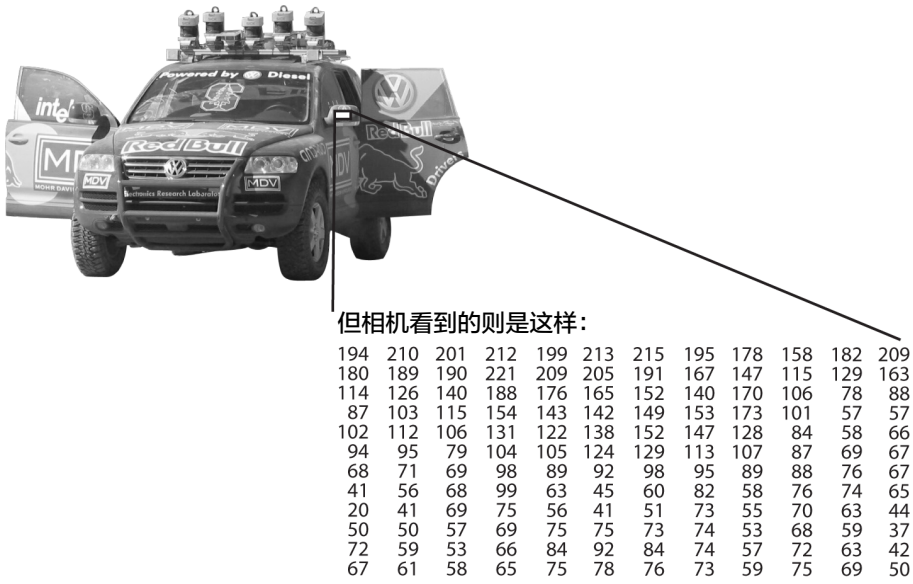


图1-1. 对于计算机来说，汽车的后视镜就是一组栅格状排列的数字

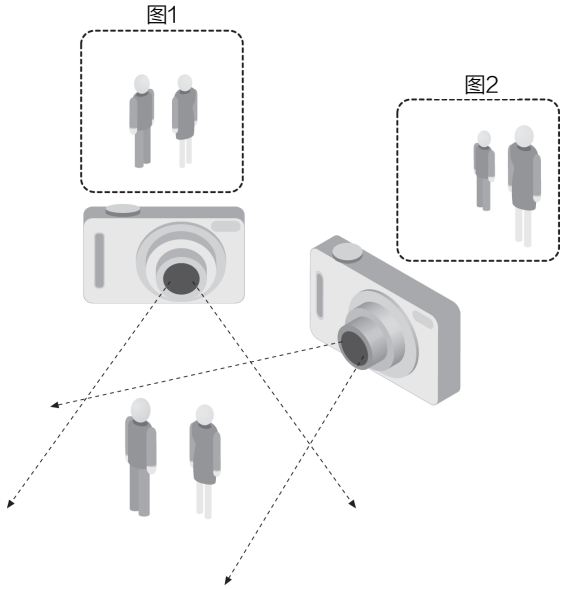


图1-2. 视觉的不适定[译注1]问题，物体的二维表示可能随着视点的不同完全改变

译注1：一个数学物理定解问题的解如果存在，唯一并且稳定的，则说明该问题是适定的（well-posed）；如果不满足，则说明该问题是不适定的（ill-posed）。

实际上，这一问题，正如我们之前所提出的，用“困难”已经不足以形容它了，它在很多情况下根本不可能解决。给定一个对于3D世界的二维（2D）观测，就不存在一个唯一的方式来重建三维信号。即使数据是完美的，相同的二维图像也可能表示一个无限的3D场景组合中的任一种情况。而且，前面也提到过，数据会被噪声和畸变所污染。这样的污染源于现实生活中的很多方面（天气、光线、折射率和运动），还有传感器中的电路噪声以及其他的一些电路系统影响，还有在采集之后对于图像压缩产生的影响。在这一系列的影响之下，我们又该如何推动事情的进展呢？

在经典的系统设计中，额外场景信息可以帮助我们从传感器的层面改善获取信息的质量。考虑这样一个例子，一个移动机器人需要在一栋建筑中找到并且拿起一个订书机。机器人就可能用到这样的事实：桌子通常放在办公室里，而订书机通常收纳在桌子里。这也同样给出了一个关于尺寸的推断：订书机的大小一定可以被桌子所收纳。更进一步，这还可以帮助减少在订书机不可能出现的地方错误识别订书机的概率（比如天花板或者窗口）。机器人可以安全忽略掉200英尺高的订书机形状的飞艇，因为飞艇没有满足被放置在木制桌面上的先验信息。相对的，在诸如图像检索等任务中，数据集中所有的订书机图像都是来自真实的订书机，这样不合常理的尺寸以及一些奇形怪状的造型都会在我们进行图片采集的时候隐式消除——因为摄影师只会去拍摄普通的正常尺寸的订书机。人们同样倾向于在拍摄的时候将拍摄目标放在图片的中间，并且倾向于在最能够展现目标特征的角度拍摄。因此，通常也有很多无意的附加信息在人们拍摄照片的时候无意加进去。

场景信息同样可以（尤其是通过机器学习技术）进行建模。隐式的变量（比如尺寸、重力的方向等不容易被直接观测到的）可以从带标记的数据集中发现关系并推测出来。或者，可以尝试使用附加的传感器测量隐式变量的值，比如利用激光雷达来测量深度，从而准确得到目标的尺寸。

计算机视觉所面临的下一个问题是噪声，我们一般使用统计的方法来对抗噪声。比如，我们很难通过单独的像素点和它的相邻像素点判断其是否是一个边缘点，但如果观察它在一个区域的统计规律，边缘检测就会变得更加简单了。一个真正的边缘应该表现为一个区域内一连串独立的点，所有点的朝向都与其最接近的点保持一致。我们也可以通过时间上的累计统计对噪声进行抑制，当然也有通过现有数据建立噪声模型来消除噪声的方法。例如，因为透镜畸变很容易建模，我们只需要学习一个简单的多项式模型来描述畸变就可以几乎完美矫正失真图像。

基于摄像机的数据，计算机视觉准备做出的动作或决定是在特定的目的或者任务的场景环境中执行的。我们也许想要移除噪声或者修复被损坏的照片，这样安全系统就可以对试图爬上栏杆等危险行为发出警报，或者对于穿过某个游乐场区域的人数进行统计。而

在大楼中漫游的机器人的视觉软件将会采取和安全系统完全不同的策略，因为两种策略处于不同的语境中。一般来说，视觉系统所处的环境约束越严格，我们就越能够依赖这些约束来简化问题，我们最终的解决方案也越可靠。

OpenCV的目标是为计算机视觉需要解决的问题提供工具。在某些情况下，函数库中的高级功能可以有效解决计算机视觉中的问题。即使遇到不能够一次性解决的问题，函数库中的基础组件也具有足够的完备性来增强解决方案的性能，以应对任意的计算机视觉难题。在后一种情况下，也存在一些使用库的可靠方法，所有的这些方法都是从尽量多使用不同的组件库开始。通常，在开发了第一个粗糙的解决方案之后，就可以发现解决方案存在哪些缺陷并且使用自己的代码与聪明才智修复那些缺陷（更为熟知的说法是“解决真正存在的问题，而不是你想象中的那些问题”）。在此之后可以使用粗糙的解决方案作为一个评判标准，评价改善水平。从这一点出发，你可以解决任意问题。

OpenCV的起源

OpenCV缘起于英特尔想要增强CPU集群性能的研究。该项目的结果是英特尔启动了许多项目，包括实时光线追踪算法以及三维墙体的显示。其中一位研究员Gary Bradski（加里·布拉德斯基），当时正在为英特尔工作，他在访问大学的时候注意到很多顶尖大学研究机构，比如MIT的媒体实验室，拥有非常完备的内部公开的计算机视觉开发接口——代码从一个学生传到另一个学生手中，并且会给每个新来的学生一个有价值的由他们自己开发的视觉应用方案。相较于从头开始设计并完成基本功能，新来的学生可以在之前的基础上进行很多新的工作。

所以，OpenCV怀着为计算机视觉提供通用性接口这一思想开始了策划。在英特尔性能实验室（Performance Library）团队的帮助下，[注2]OpenCV最初的核心代码和算法规范是英特尔俄罗斯实验室团队完成的，这就是OpenCV的缘起，从英特尔软件性能组的实验研究开始，俄罗斯的专家负责实现和优化。

俄罗斯专家团队的负责人是Vadim Pisarevsky（瓦迪姆·彼萨里夫斯基），他负责规划、编程以及大部分OpenCV的优化工作，并且到现在他仍是很多OpenCV项目的核心人物。与他一同工作的Victor Eruhimov（维克托·伊拉西莫夫）帮助构建了早期框架，Valery Kuriakin（瓦勒利·库里阿基恩）负责管理俄罗斯实验室并且为项目提供了非常大的助力。以下是OpenCV想要完成的一些目标。

注2： Shinn Horng Lee（李信弘）是主要推动者，时任IPP（集成性能基元）首席架构师。

- 为高级的视觉研究提供开源并且优化过的基础代码，不再需要重复造轮子。
- 以提供开发者可以在此基础上进行开发的通用接口为手段传播视觉相关知识，这样代码有更强的可读性和移植性。
- 以创造可移植的、优化过的免费开源代码来推动基于高级视觉的商业应用，这些代码可以自由使用，不要求商业应用程序开放或免费。

这些目标阐述了Opencv创建的目的。启用计算机视觉程序，将会增加对运算更快处理器的需求，从而使得用户购买更快的处理器，相较于售卖额外的软件，能够更快增加英特尔的收入。也许这就是这样一个开源并且免费的代码库是由一个硬件厂商而非软件厂商开发的原因。有时，硬件厂商内部拥有更多针对软件的创新空间。

对于开源项目来说，项目需要超过一个临界质量使其能够自我维持，这是开源项目非常重要的一点。目前，OpenCV已经拥有接近1100万次的下载量（2009年3月，200万次），并且这个数字还在以平均每个月160 000次下载的速度增长。OpenCV得到了来自很多用户的贡献，研发主力也很大部分转移到英特尔之外。[注3]OpenCV发展的时间线如图1-3所示。[译注1]在发展过程中，OpenCV受到互联网泡沫的影响，也受到管理层和方向变更等诸多变化的影响，在这些波动的过程中，有时候根本就没有英特尔公司任何人员参与。然而，随着多核处理器的出现以及计算机视觉的许多新应用的问世，OpenCV的价值开始上升。类似地，机器人领域的快速增长也推动了许多用户开始使用和开发这个库。在成为一个开源的函数库之后，OpenCV经过柳树车库几年的开发，现在已经得到了OpenCV基金会的支持。在今天，OpenCV由基金会以及一些上市公司和私人机构开发，更多关于OpenCV未来的信息，请参阅涉及OpenCV未来的相关章节。

OpenCV的结构

OpenCV是由层级结构组织的。处于最上层的是OpenCV和操作系统的交互。接下来是语言绑定和示例应用程序。再下一层是*opencv_contrib*模块所包含的OpenCV由其他开发人员所贡献的代码，其包含大多数高层级的函数功能。这就是OpenCV的核心。底层是基于硬件加速层（HAL）的各种硬件优化。图1-4显示了这个组织关系。

注3：在撰写本书的过程中，柳树车库（Willow Garage）这个机器人研究机构和孵化器，正在积极推动日常的OpenCV维护以及视觉在机器人领域的新进展。

译注1：2016年12月发布OpenCV 3.2；2017年8月发布OpenCV 3.3；2017年12月发布OpenCV 3.4。更详细的发展历程可以访问*https://opencv.org/releases.html*。

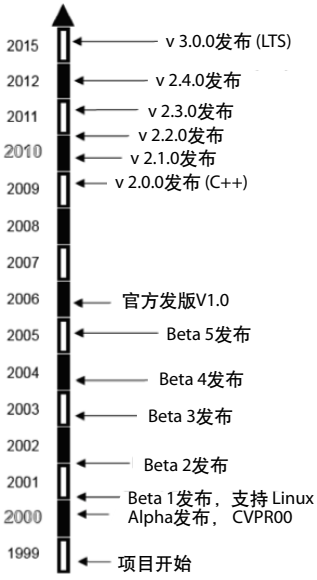


图1-3. OpenCV发展历程

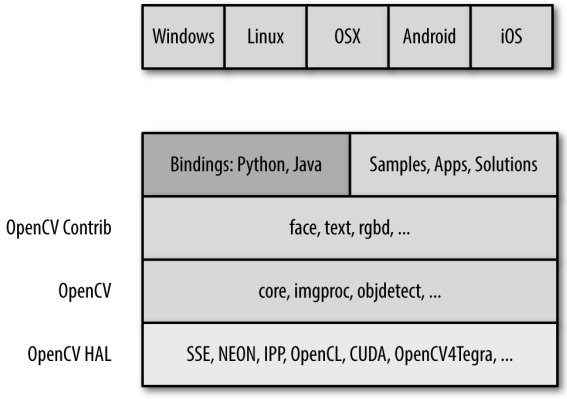


图1-4. OpenCV层级结构及其支持的操作系统

使用IPP来加速OpenCV

如果在英特尔的处理器上使用，OpenCV就会自动使用一种免费的英特尔集成性能原语库（IPP）的子集，IPP 8.x(IPPICV)。IPPICV可以在编译阶段链接到OpenCV，这样一来，会替代相应的低级优化的C语言代码（在cmake中设置`WITH_IPP=ON/OFF`开启或者关闭这一功能，默认情况为开启）。使用IPP获得的速度提升非常可观，图1-5显示了使用IPP之后得到的加速效果。

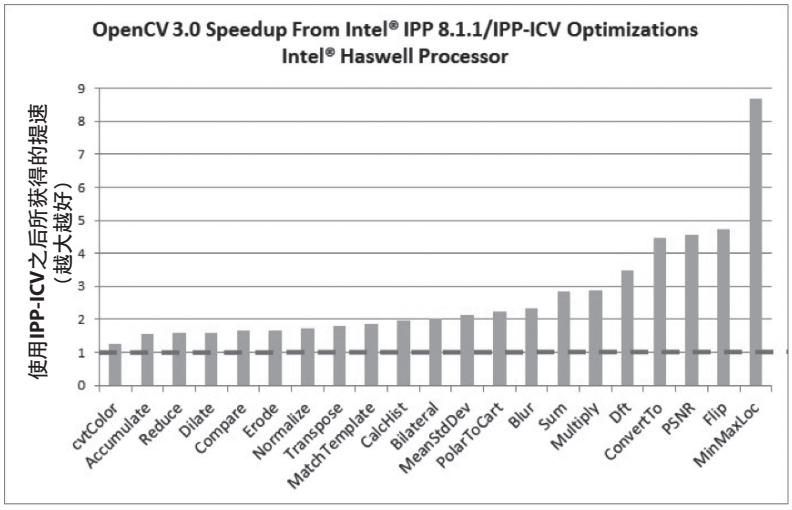


图1-5. 当OpenCV在Intel Haswell处理器上使用IPPICV时的加速效果

谁拥有OpenCV

尽管Gary Bradski在英特尔发起了OpenCV项目，但这个函数库的宗旨是促进商业和研究，这正是它的使命。因此，它是开源并且免费的，并且无论是研究还是商业目的代码都可以（全部或者部分的）使用或者被嵌入到其他程序中。它并不强制你声明基于此开发的代码必须是免费或者开源的，也不要求你将自己的改进反馈回OpenCV中，虽然我们希望可以。

下载和安装OpenCV

在OpenCV官方网站（*http://opencv.org/*）可以下载最新的且完整的源码以及大部分的release版本源码。下载链接可以通过下载页面（*http://opencv.org/downloads.html*）访问。当然，最新的代码会在github（*https://github.com/opencv/opencv*）上进行即时更新。如果想要获取最新的高级函数功能，也可以下载和编译*opencv_contrib*模块（*https://github.com/opencv/opencv_contrib*）。

安装

现在，OpenCV使用Git作为版本管理工具，使用Cmake来构建工程。[注4]当然，这些日子都一去不复返了。在许多情况下，你不必担心构建问题，因为许多环境中都有预编译好

注4：在早些时候，OpenCV开发者使用Subversion作为版本管理工具以及automake来构建工程项目。

的库。但是，一旦成为更加专业的使用者，势必需要重新编译库，并且根据具体的应用程序定制特定的选项。

Windows系统

在*http://opencv.org/downloads.html*，可以发现最新的为Windows准备的最新版本OpenCV下载链接。可以通过这个链接下载一个EXE文件，该文件会把预编译好的OpenCV解压到你的电脑上，预编译版本支持不同版本的Visual Studio环境。现在，你很快就可以开始使用OpenCV了。[注5]

另一个额外的细节是，需要再添加一个名为`OPENCV_DIR`的环境变量来告诉编译器在哪里找到OpenCV的二进制文件。可以通过命令行工具对此进行设置。[注6]

```
setx -m OPENCV_DIR D:\OpenCV\Build\x64\vc10
```

如果希望静态链接OpenCV，就只需要做到这一步。如果希望使用OpenCV的动态链接库（DLL），就需要告诉系统在哪里找到它的二进制库。为了完成这一目标，只需要在库路径中添加`%OPENCV_DIR%\bin`。（比如说，在Windows 10中，右击计算机图标，选择“属性”，然后单击“高级系统设置”，最后选择“环境变量”，把OpenCV二进制文件的路径添加到“系统变量”的`path`变量中。）

OpenCV3集成了IPP，所以如果使用最新的x86或者x64CPU，就可以获得或多或少的性能优势。

也可以按照如下操作从源码编译OpenCV。

1. 运行CMake GUI。
2. 指定OpenCV源码所在路径以及构建目标文件夹（必须和源码所在路径不同）。
3. 按两次Configure键（选择可以使用的Visual Studio编译器或者MinGW构建文件，如果正在使用MinGW的话），直到所有条目没有红色警示。
4. 使用Visual Studio打开生成的解决方案并构建。如果使用的是MinGW，则按照Linux的安装指导进行。

注5：有一点很重要的事情是，尽管Windows环境拥有预编译的release版本的库，但是它并不包含debug版本的库。所以，在开发OpenCV之前，需要打开解决方案并且自行编译那些库。

注6：当然，具体路径会根据安装的不同而有所不同，比如说在32位机器上安装，那么该路径就该是`x86`而不是`x64`。

Linux系统

由于GCC和GLIBC在不同Linux版本（SuSE，Debian，Ubuntu，等等）下拥有不同的版本，OpenCV的Linux的预编译版本不包含linux的版本号。然而，在很多情况下，你的Linux版本会提供OpenCV。如果你的版本不提供OpenCV，你将不得不从源代码中构建，与Windows安装一样，可以从*http://opencv.org/downloads.html*下载源代码，但是在这种情况下，该链接会把你转向SourceForge，在这里可以为当前的OpenCV源代码选择压缩文件。

为了编译这个库和示例程序，需要GTK+2.x或者更高的版本，还需要*gcc*以及必要的开发包*cmake*以及*libtbb*（英特尔线程构建模块）以及一些可选项目诸如*zlib*，*libpng*，libjpeg，*libtiff*以及*libjasper*的开发者版本（例如模块名称后带有*-dev*的版本）。你也需要用到Python 2.6或者更高的版本（开发者包）以及NumPy使OpenCV可以在Python环境下工作。此外，还需要来自*ffmpeg*的*libavcodec*以及其他的*libav**库（包含头文件）。

对于后者，请安装Linux发行版本所提供的的*libav*/*ffmpeg*包，或者前往*http://www.ffmpeg.org*下载。*ffmpeg*库拥有较低的通用公开（LGPL）许可证，但是它的一些组件拥有更严格的通用公开许可证（GPL）。为了结合使用非GPL的软件，你需要构建并且使用一个共享的*ffmpeg*库：

```
$> ./configure --enable-shared
$> make
$> sudo make install
```

当动态链接一个LGPL的库时，不需要为自己的代码使用GPL许可证。最终得到*/usr/local/lib/libavcodec.so.**，*/usr/local/lib/libavformat.so.**，*/usr/local/lib/libavutil.so.**以及在*/usr/local/include/libav**路径下的头文件。

为了编译这个库，需要解压*tar.gz*文件并且切换到解压过程创建的源码文件夹中，然后进行如下操作：

```
mkdir release
cd release
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..
make
sudo make install #  optional
```

第一个命令和第二个命令将创建名为*release*的子目录并且切换到*release*中，第三个命令告诉*cmake*工具如何设置你的工程。我们提供的例子可能是让你入门的正确方法，但是其他选项允许你启用更多的设置：确定哪些例子需要被构建；添加对Python的支持；添加CUDA GPU的支持，等等。默认情况下，OpenCV的*cmake*配置脚本会尝试查找和使用尽可能多的第三方库，例如如果它探测到CUDA SDK的存在，就会自动支持GPU加速的OpenCV功能。最后两个指令将编译整个库并将其安装到正确的位置。注意，如果使用

CMake来建工程，就没有必要安装OpenCV，只需要指定生成的*OpenCVConfig.cmake*文件的路径就可以了。在前面的例子中，文件将被安装在*/usr/local/share/OpenCV*。

正如Windows的例子所示，Linux建立的OpenCV将自动利用IPP进行加速（如果有安装的话）。从OpenCV 3.0版本开始，OpenCV的*cmake*设置脚本将自动下载并且链接一个IPP的免费子库（IPPICV）。如果想要禁用IPP加速，请在执行CMake的时候加上`-D WITH_IPP=OFF`指令。

Mac系统

在Mac上安装步骤和Linux上的安装步骤十分接近，不同的是，Mac拥有自己的开发环境Xcode，包含大部分在CMake过程中需要的东西。你不需要GTK+、TBB，*libjpeg*，并且：

- 在默认情况下，Cocoa会取代GTK+；
- 在默认情况下，QTKit会取代*ffmpeg*；
- GDC会取代TBB以及OpenMP。

安装步骤和Linux下安装一致。需要添加`-G Xcode`指令到CMake中来生成一个Xcode工程，从而可构建和debug工程。

从Git获取最新的OpenCV

OpenCV现在也处于积极的开发状态中，当bug报告中包含有准确的描述以及代码复现bug的时候，该bug通常会被迅速修复。然而，官方的OpenCV通常每年只发布一次到两次，如果你正在开发一个项目或产品，可能想要OpenCV能够尽快修复bug并更新。为了完成这些目的，需要在GitHub网站上访问OpenCV的Git仓库。

本节并不打算引导你如何使用Git，如果你还在使用另外的一些开源项目，也许对这套操作已经很熟悉了。如果对此并不熟悉，请查阅Jon Loeliger（乔恩·罗力格）所著的《Git版本控制管理》（*http://shop.oreilly.com/product/0636920022862.do*）。Git的命令行工具有Linux，OS X以及大部分的类UNIX系统所支持。针对Windows，我们推荐TortoiseGit（*https://tortoisegit.org/*）；针对OS X，SourceTree也许适合。

在Windows上，如果想要从Git获得OpenCV最新的版本，你需要访问*https://github.com/opencv/opencv.git*。

在Linux上，只需要输入如下指令：

```
git clone https://github.com/opencv/opencv.git
```

更多的OpenCV文档

OpenCV基础文档可以在*http://opencv.org*获取。另外，在*http://docs.opencv.org/2.4.13/doc/tutorials/tutorials.html*，也可以获得更多深入教程，OpenCV的维基网站页面现在位于*https://github.com/opencv/opencv/wiki*。

提供的文档

OpenCV 2.x自身就提供PDF格式的完整的引用手册以及丰富的教程，查看*opencv/doc*目录就可以得到。从OpenCV 3.x开始，就不再提供离线的文档了。

在线文档和维基资源

正如我们之前提到的，在*https://opencv.org*上有大量的文档和维基资源。文档分为以下几个主要部分。

参考（http://docs.opencv.org/）

这个部分包含函数、它们的参数以及如何使用它们的一些信息。

教程（http://docs.opencv.org/trunk/d9/df8/tutorial_root.html）

这个部分是许多教程的集合，这些教程会告诉你如何完成各种各样的事情。这里有一些基础课程的教程（比如如何在不同平台上安装OpenCV或者创建OpenCV项目）以及一些更高级的话题（比如目标检测的背景提取算法）。

快速指南（http://opencv.org/quickstart.html）

本部分包含一个精心编制的快速指南，只包含帮助你在特定平台上获取和运行OpenCV的内容。

速查表（http://docs.opencv.org/3.0-last-rst/opencv_cheatsheet.pdf）

只有一页篇幅的PDF文档，包含整个库的高度压缩的参考。感谢Vadim Pisarevsky制作了这个出色的速查表，这样你就可以把这漂亮的两页纸钉在小隔间的墙上随时查阅。

维基（https://github.com/opencv/opencv/wiki）

维基包含所有你可能想要的东西，甚至可能超乎你的想象。在这里，可以找到路线图、新闻、开放问题、bug追踪以及无数个更深入的主题，比如如何成为OpenCV的贡献者。

问答社区（http://answers.opencv.org/questions/）

问答社区是一个庞大的、包含数千个人遇到过的问题的档案集合。可以在这里向OpenCV社区提问，或者通过回答问题帮助其他人。

以上所有在OpenCV.org的底部的有超链接可以访问。在这些高质量的资源中，有一个更值得我们讨论——“参考”。参考划分为几个部分，每个部分都和库中一个模块有关。具体的模块列表随着时间推移而不断的发展，但模块始终是组成这个库的基本单位。每个函数都是一个模块的一部分，以下是当前OpenCV所拥有的模块。

`Core`

该模块包含OpenCV库的基础结构以及基本操作。

`Improc`

图像处理模块包含基本的图像转换，包括滤波以及类似的卷积操作。

`Highgui`（在`OpenCV 3.0`中，分割为`imcodecs`、`videoio`以及`highgui`三部分）

这个模块包含可以用来显示图像或者简单的输入的用户交互函数。这可以看作是一个非常轻量级的Windows UI工具包。

`Video`

该模块包含读取和写视频流的函数。

`Calib3d`

这个模块包括校准单个、双目以及多个相机的算法实现。

`Feature2d`

这个模块包含用于检测、描述以及匹配特征点的算法。

`Objdectect`

这个模块包含检测特定目标，比如人脸或者行人的算法。也可以训练检测器并用来检测其他物体。

`Ml`

机器学习模块本身是一个非常完备的模块，包含大量的机器学习算法实现并且这些算法都能和OpenCV的数据类型自然交互。

`Flann`

Flann的意思是“快速最邻近库”。这个库包含一些你也许不会直接使用的方法，但是其他模块中的函数会调用它在数据集中进行最邻近搜索。

`GPU`（在*OpenCV 3.x*中被分割为多个`cuda*`模块）

GPU模块主要是函数在CUDA GPU上的优化实现，此外，还有一些仅用于GPU的功能。其中一些函数能够返回很好的结果，但是需要足够好的计算资源，如果硬件没有GPU，则不会有什么提升。

`Photo`

这是一个相当新的模块，包含计算摄影学的一些函数工具。

`Stitching`

本模块是一个精巧的图像拼接流程实现。这是库中的新功能，但是，就像Photo模块一样，这个领域未来预计有很大的增长。

`Nonfree`（在*OpenCV 3.0*中，被移到*opencv_contrib/xfeatures2d*）

OpenCV包含一些受到专利保护的或者受到使用限制的（比如SIFT算法）算法。这些算法被隔离到它们自己的模块中，以表明你需要做一些特殊的工作，才可以在商业产品中使用它们。

`Contrib`（在*OpenCV 3.0*中，融合进了*opencv_contrib*）

这个模块包含一些新的、还没有被集成进OpenCV库的东西。

`Legacy`（在*OpenCV 3.0*中，被取消）

这个模块包含一些老的尚未被完全取消的东西。

`Ocl`（在*OpenCV 3.0*中，被取消，取而代之的是*T-API*）

这是一个较新的模块，可以认为它和GPU模块相似，它实现了开放并行编程的Khronos OpenCL标准。虽然现在模块的特性比GPU模块少很多，但`Ocl`模块的目标是提供可以运行在任何GPU或者是其他可以搭载Khronos的并行设备。这与GPU模块形成了鲜明的对比，后者使用Nividia CUDA工具包进行开发，因此只能在Nividia GPU设备上工作。

尽管在线文档的质量越来越高，但并没有充分解释算法的实现细节及期所要求的参数的准确含义。这本书的目的是提供这些信息以及帮助读者更深入地理解库中所有的基本模块。

OpenCV贡献库

在OpenCV 3.0中，先前单一的库分成两部分：成熟的OpenCV和*opencv_contrib*中的最新模块。前者由核心的OpenCV团队维护，并且包含(大部分)稳定的代码，而后者则不成熟，主要由社区维护和开发，可能有非OpenCV许可的部分，并且可能包括受专利保护的算法。

以下是一些可以在*opencv_contrib*中获得的模块，参阅附录B，可以查看完整的最新列表。

`Dnn`

深度神经网络。

`Face`

人脸识别。

`Text`

文本检测以及识别，基于许多开源的OCR算法。

`Rgbd`

处理由Kinect或者其他深度传感器（或者简单的由双目匹配得到的）获取的RGB+深度图像。

`Bioinspired`

一些基于生物学启发的视觉技术。

`ximgproc`和`xphoto`

先进的图像处理以及计算摄影学方法。

`Tracking`

现代目标追踪算法。

下载和编译Contributed模块

在Linux和OS X中，可以直接使用下面的指令来下载*opencv_contrib*模块。

```
git clone https://github.com/opencv/opencv_contrib.git
```

在Windows，将这个地址输入TortoiseGit或者其他客户端，接下来只需要再次配置OpenCV的Cmake：

```
cmake -D CMAKE_BUILD_TYPE=Release \
  -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules ..
```

然后和平常一样再次编译。编译好的Contributed会和普通的OpenCV模块处于同样的目录，不需要任何特殊的配置就可以使用它们。

可移植性

OpenCV基于可移植的理念设计，最初可以被任意通用的C++编译器编译。这表示它的C和C++代码都必须是相当标准的，以便跨平台支持更加容易。表1-1显示OpenCV在不同运行平台的支持，其中对于Intel和AMD 32位，64位（x86，x64）的支持是最成熟的，对ARM的支持也在迅速改善中。对于操作系统，OpenCV完全支持Windows、Linux、OS X、Android和iOS。如果一个体系结构或者操作系统没有出现在这个表中，并不意味着这个系统无法使用OpenCV。事实上，OpenCV已经被移植到几乎所有商用系统中，从亚马逊云以及40核的英特尔Xeon Phi到树莓派和机器狗中。

表1-1：OpenCV 1.0发行版本移植性引导

	x86/x64	ARM	其他：MIP，PPC
Windows	SIMD，IPP，Parallel，I/O	SIMD，Parallel (3.0)，I/O	N/A
Linux	SIMD，IPP，Parallel，[a] I/O	SIMD，Parallel，[a] I/O	Parallel，[a] I/O*
Android	SIMD，IPP (3.0)，Parallel，[b] I/O	SIMD，Parallel，[b] I/O	MIPS—基本支持
OS X/iOS	SIMD，IPP (3.0)，Parallel，I/O	SIMD，Parallel，I/O	N/A
其他：BSD，QNX等	SIMD	SIMD	

a. Linux中的并行化是通过第三方库或启用OpenMP来实现的

b. Android设备中并行是通过TBB进行的

对表1-1的说明如下。

SIMD

用于加速的向量指令：基于x86，x64，NEON以及ARM的SSE指令集。

IPP

可否使用英特尔IPP加速，从3.0版本开始，函数库自带一个IPP的免费子库（IPPICV）。

Parallel

一些用于在多核进行线程控制的标准或者第三方线程框架。

I/O

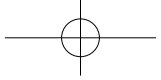
一些用于抓取或者写视频的标准或第三方API。

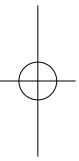
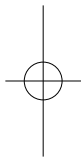
小结

在这一章中，我们对OpenCV的历史进行了研究，从1999年由Gary Bradski（加里·布拉德斯基）在英特尔公司始创到现在Arraiy的支持。我们讨论了OpenCV及其部分内容的动机。讨论了OpenCV核心库是什么样的以及*opencv_contrib*模块（参考附录B）。本章还介绍了如何下载和安装OpenCV，以及它的性能和可移植性如何。

练习

1. 下载并且安装最新的OpenCV，并分别在debug模式和release模式下进行编译。
2. 从Git下载并且编译最新的OpenCV代码。
3. 描述至少三个将3D输入转换为2D表示时引起的问题，你如何解决这些问题？

第2章

OpenCV初探

头文件

在安装OpenCV库以及设置好编程环境之后，下一个任务就是用代码来做一些有趣的事情。为了完成这个工作，我们需要先讨论一下头文件。幸运的是，头文件是按照第1章所述模块一样组织的。而我们最感兴趣的头文件莫过于*.../include/opencv2/opencv.hpp*，它包含OpenCV各个模块的头文件：

`#include "opencv2/core/core_c.h"`
　　旧式C风格的结构以及运算

`#include "opencv2/core/core.hpp"`
　　新式C++风格的结构以及数学运算

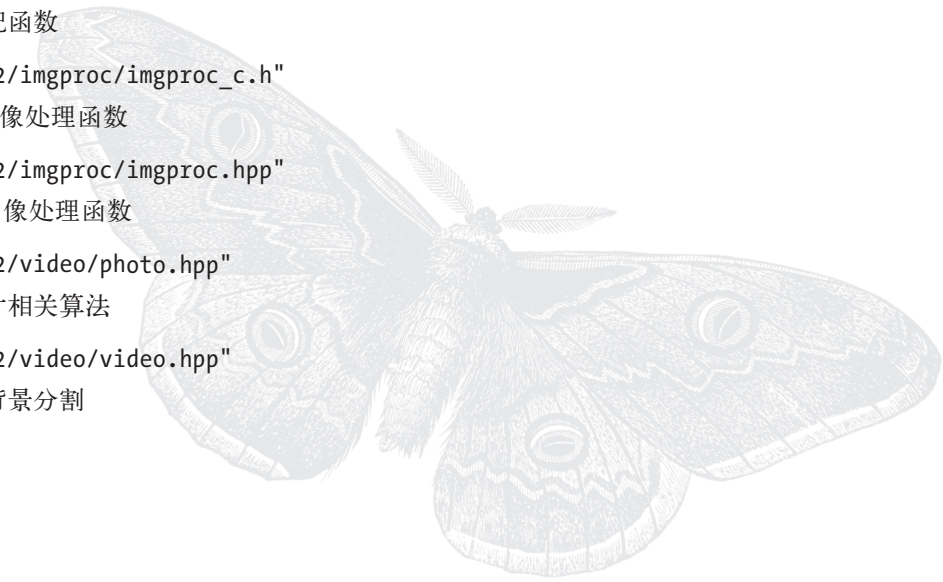
`#include "opencv2/flann/miniflann.hpp"`
　　最邻近搜索匹配函数

`#include "opencv2/imgproc/imgproc_c.h"`
　　旧式C风格的图像处理函数

`#include "opencv2/imgproc/imgproc.hpp"`
　　新式C++风格图像处理函数

`#include "opencv2/video/photo.hpp"`
　　操作和恢复照片相关算法

`#include "opencv2/video/video.hpp"`
　　视觉追踪以及背景分割



`#include "opencv2/features2d/features2d.hpp"`
　　用于追踪的二维特征

`#include "opencv2/objdetect/objdetect.hpp"`
　　级联人脸分类器、latent SVM分类器、HoG特征和平面片检测器

`#include "opencv2/calib3d/calib3d.hpp"`
　　校准以及双目视觉相关

`#include "opencv2/ml/ml.hpp"`
　　机器学习、聚类以及模式识别相关

`#include "opencv2/highgui/highgui_c.h"`
　　旧式C风格的显示、滑动条、鼠标操作以及输入输出相关

`#include "opencv2/highgui/highgui.hpp"`
　　新式C++风格的显示、滑动条、鼠标操作以及输入输出相关

`#include "opencv2/contrib/contrib.hpp"`
　　用户贡献的代码、皮肤检测、模糊Mean-Shift追踪、spin image算法及自相似特征等

你也许会使用头文件*opencv.hpp*来包含所有可能在OpenCV函数中用到的头文件，但是这会减慢编译的速度。如果只使用一个，比如说图像处理相关的函数，只包含*opencv2/imgproc/imgproc.hpp*所消耗的编译时间会比包含*opencv.hpp*消耗的编译时间少很多。这些头文件在*.../modules*文件夹中。比如，*imgproc.hpp*存在于*.../modules/imgproc/include/opencv2/imgproc/imgproc.hpp*。类似，函数对应的源文件也存在于对应的*src*文件夹中。比如，`improc`模块的`cv::Canny()`函数存在于*.../modules/improc/src/canny.cpp*。

有了前述头文件之后，就可以开始写我们的第一个C++的OpenCV程序了。

可以使用*opencv2/legacy/legacy.hpp*来调用遗留代码，比如说比较旧的斑点追踪、隐马尔科夫模型（HMM）检测、聚合追踪以及特征目标。其他位于*.../modules/legacy/include/opencv2/legacy/legacy.hpp*。

资源

网上有几个不错的OpenCV资源：

- 对整个OpenCV库进行概述：*https://www.slideshare.net/zblair/opencv-introduction*或OpenCV3概述，*http://slideplayer.com/slide/10302485/*，也可以发送邮件到*coo@netease.com*，大家共同探讨

- 关于加速的讨论：*https://www.slideshare.net/embeddedvision/making-opencv-code-run-fast-a-presentation-from-intel*
- 关于模块的讨论：*https://docs.opencv.org/3.1.0/*

第一个程序：显示图片

OpenCV提供了一些实用工具来读取从视频流到摄像机画面的各种各样的图像格式，这些工具都是HighGUI的一部分。我们将使用其中的一些工具来创建一个简单的程序，这个程序将打开一张图像并且将其显示在屏幕上（如示例2-1所示）。

示例2-1：一个简单的加载并显示图像的OpenCV程序

```
#include <opencv2/opencv.hpp> //Include file for every supported OpenCV function

int main( int argc, char** argv ) {
  cv::Mat img = cv::imread(argv[1],-1);
  if( img.empty() ) return -1;
  cv::namedWindow( "Example1", cv::WINDOW_AUTOSIZE );
  cv::imshow( "Example1", img );
  cv::waitKey( 0 );
  cv::destroyWindow( "Example1" );
  return 0;
}
```

注意，OpenCV的函数都位于cv这一命名空间下，为了调用OpenCV的函数，你需要在每个函数前加上cv::，向编译器说明你所调用的函数处于cv命名空间。为了摆脱这种繁琐的工作，我们可以像示例2-2一样用using namespace cv;指令，告诉编译器假设所有函数都位于cv命名空间下。[注1]你还需要注意示例2-1和示例2-2在头文件上的不同，在前者中，我们使用了通用的*opencv.hpp*，而在后者中，我们只使用了必须的头文件来节约编译时间。

示例2-2：与示例2-1不同的是直接使用了using namespace

```
#include "opencv2/highgui/highgui.hpp"

using namespace cv;

int main( int argc, char** argv ) {

  Mat img = imread( argv[1], -1 );
  if( img.empty() ) return -1;
```

注1：当然，如果这样做，需要冒着和其他潜在的命名空间冲突的风险。如果函数foo()已经存在于cv和std的命名空间，你必须指定自己使用的是位于cv::foo()的函数还是位于std::foo()的函数。在本书中，除了示例2-2，都会指明cv::命名空间，并以此保持一个良好的编程风格。

```
    namedWindow( "Example1", cv::WINDOW_AUTOSIZE );
    imshow( "Example1", img );
    waitKey( 0 );

    destroyWindow( "Example1" );

}
```

当使用命令行编译和运行时，[注2]示例2-1将加载一张图像到内存中并且显示到屏幕上。它会保持显示直到用户按下一个键，之后程序才会销毁窗口并退出。现在让我们来一行一行地解释代码，并花一些时间来理解每一行代码都在做什么工作。

```
    cv::Mat img = cv::imread( argv[1], -1 );
```

这一行将会载入图像。[注3]函数cv::imread()是高级的；依据文件名来决定载入图像格式的处理。这也会自动地申请图像需要的内存，注意，cv::imread()可以读取很多种图像格式，包括BMP，DIP，JPEG，JPE，PNG，PBM，PGM，PPM，SR，RAS以及TIFF。函数会返回一个cv::Mat结构，这个结构是OpenCV中你将会接触最多的自带结构。OpenCV使用这个结构来处理所有类型的图像：单通道、多通道、整型、浮点数以及各种类型。紧接着的下面这一行：

```
    if( img.empty() ) return -1;
```

检查这个图像是否真的被载入了。另一个高层级的函数cv::namedWindow()将会在屏幕打开一个窗口，其中可以包含需要显示的图片。

```
    cv::namedWindow( "Example1", cv::WINDOW_AUTOSIZE );
```

该函数由HighGUI模块提供，会将一个名称赋予窗口（在这里窗口名为"Example1"）。未来HighGUI的和这个窗口的交互函数将由这个名称来指定要与哪个窗口交互。

cv::namedWindow第二个参数说明了Windows的特性。这可以全部设置为0（默认情况下），也可以设置为cv::WINDOW_AUTOSIZE。在之前的例子中，窗口的大小将会和载入图像的大小一致，图像将会被缩放以适应窗口的大小。在之后的例子中，窗口将会在图像载入的时候被自动缩放以适应图像的真实大小，也可能由用户自行调整。

注2：当然，如果这样做的话，你需要冒着和其他潜在的命名空间冲突的风险。如果函数foo()已经存在于cv和std的命名空间，必须指定你使用的是位于cv::foo()的函数还是位于std::foo()的函数。在本书中，除了示例2-2，都会指明cv::命名空间，并以此保持一个良好的编程风格。

注3：好的程序会检查argv[1]是否存在并且给用户反馈一个错误信息，但是在这里没有。我们在本书中将会简略掉这些处理并且假设读者都有足够的的知识来理解处理错误代码的方式及其重要性。

```
cv::imshow( "Example1", img );
```

不论何时，只要在cv::Mat中拥有一个图像结构，我们都可以通过cv::imshow()进行显示。cv::imshow()将建一个窗口（如果这个窗口不存在，它会自动调用cv::namedWindow()新建一个窗口）。在调用cv::imshow()的时候，窗口将被重绘上要求的图片，并且窗口会按照要求自动调整大小（如果使用cv::WINDOW_AUTOSIZE参数）。

```
cv::waitKey(0);
```

cv::waitKey(0);函数告诉系统暂停并且等待键盘事件。如果其传入了一个大于零的参数，它将会等待等同于该参数的毫秒时间，然后继续执行程序。如果参数被设置为0或者一个负数，程序将会无限等待直到有键被按下。

因为有cv::Mat，图像将会在生命周期结束的时候自动释放，其行为类似于标准模板库（STL）中的容器类。这种自动的内存释放由内部的引用指针所控制，最重要的是，这表示我们用不着担心图像的内存申请和释放，这将程序员从OpenCV 1.0 IplImage结构繁琐的维护工作中解放了出来。

```
cv::destroyWindow( "Example1" );
```

最后，我们可以让窗口自行销毁。函数cv::destroyWindow()将会关闭窗口并且释放掉相关联的内存空间。为了更简洁的编码，我们将会在之后的例子中略过这一步。在更长、更复杂的代码中，程序员应该在窗口的生命周期自然结束之前自主销毁窗口以防止内存泄漏。

我们下一个任务是创建一个非常简单的、几乎和本例一样的程序来读取视频文件。在此之后，我们将会开始对实际图像进行更多的操作。

第二个程序：视频

用OpenCV播放视频就像显示图像一样简单。唯一不同的是，我们需要某种循环来读取视频序列中的每一帧。我们也许还需要一些方法在电影太无聊的时候来帮助跳出循环。

示例2-3：一个简单的播放视频文件的OpenCV程序

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

int main( int argc, char** argv ) {

  cv::namedWindow( "Example3", cv::WINDOW_AUTOSIZE );
  cv::VideoCapture cap;
  cap.open( string(argv[1]) );
```

```
  cv::Mat frame;
  for(;;) {
    cap >> frame;
    if( frame.empty() ) break;             // Ran out of film
    cv::imshow( "Example3", frame );
    if( cv::waitKey(33) >= 0 ) break;
  }

  return 0;
}
```

main函数从创建一个窗口开始（在本例中窗口名为"Example3"），视频读取结构cv::VideoCapture在其后被加载进来。这个结构可以打开和关闭很多类型的*ffmpeg*支持的视频文件。

```
    cap.open(string(argv[1]));
    cv::Mat frame;
```

视频读取结构通过传入字符串打开文件，这个字符串指示了想要打开的视频文件的路径。一旦视频被打开，视频读取结构将会包含所有的关于这个视频文件可以读取的属性，包括状态信息。以这样的方式创建以后，cv::VideoCapture结构将会在视频的开头被初始化。在这个程序中，cv::Mat frame声明了一个可以保存视频帧的结构。

```
    cap >> frame;
    if( frame.empty() ) break;
    cv::imshow( "Example3", frame );
```

一旦内部的while()循环开始执行，视频文件会按照帧从视频流中被读取。这个程序通过if(frame.empty())检查数据是不是真的从视频中读了出来，如果没有，程序将会退出。如果视频帧被成功读取，将通过cv::imshow()显示。

```
    if( cv::waitKey(33) >= 0 ) break;
```

一旦显示了这帧图片，我们会等待33毫秒。[注4]如果用户在这段时间在键盘有任何输入，我们将退出循环。如果没有发生，33毫秒之后我们会跳过并执行下一个循环。在退出的时候，所有数据的内存空间将会由于生命周期的结束被自动释放掉。

跳转

为了强化前面这个简单的程序以及发现更多有用的函数，现在是时候反思一下了。我们也许已经注意到示例2-3中的视频播放器不可以快速在视频中进行跳转。所以我们下一个

注4：你也许想要定义其他时间长度，在这个情况下，我们简单说明33毫秒是因为这能让视频以30FPS的速率播放，并且能够允许用户在播放的时候打断。根据以往的经验，最好检查cv::VideoCapture结构来确定视频真正的帧率（更多详情请参阅第8章）。

任务就是添加一个滑动条，使得用户可以进行视频跳转。为了执行更多的控制，我们将会允许用户通过按下S键来执行单步模式，以及按下R键来恢复连续视频播放模式，无论何时用户通过滑动条跳转到视频一个新的时间点，我们都会使用单步模式在那个点进行播放。

HighGUI工具提供了许多简单的工具用于处理视频和图像，而不仅仅是我们刚才演示的那些功能。一个特别有用的工具就是我们前面所提到的滑动条，它能够使用户轻松从视频的一个部分跳转到另一个部分。要创建一个滑动条，我们会调用cv::createTrackbar()并指明要在哪个窗口中进行显示，为了完成所需要的功能，我们还需要一个能够执行重新定位的回调函数，示例2-4给出了代码表述。

示例2-4：加入了滑动条的基本浏览窗口

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <fstream>

using namespace std;

int g_slider_position = 0;
int g_run = 1, g_dontset = 0; //start out in single step mode
cv::VideoCapture g_cap;

void onTrackbarSlide( int pos, void *) {

  g_cap.set( cv::CAP_PROP_POS_FRAMES, pos );

  if( !g_dontset )
    g_run = 1;
  g_dontset = 0;

}

int main( int argc, char** argv ) {

  cv::namedWindow( "Example2_4", cv::WINDOW_AUTOSIZE );
  g_cap.open( string(argv[1]) );
  int frames = (int) g_cap.get(cv::CAP_PROP_FRAME_COUNT);
  int tmpw   = (int) g_cap.get(cv::CAP_PROP_FRAME_WIDTH);
  int tmph   = (int) g_cap.get(cv::CAP_PROP_FRAME_HEIGHT);
  cout << "Video has " << frames << " frames of dimensions("
       << tmpw << ", " << tmph << ")." << endl;

  cv::createTrackbar("Position", "Example2_4", &g_slider_position, frames,
                onTrackbarSlide);

  cv::Mat frame;
  for(;;) {

    if( g_run != 0 ) {
```

```
      g_cap >> frame; if(frame.empty()) break;
      int current_pos = (int)g_cap.get(cv::CAP_PROP_POS_FRAMES);
      g_dontset = 1;

      cv::setTrackbarPos("Position", "Example2_4", current_pos);
      cv::imshow( "Example2_4", frame );

      g_run-=1;

    }

    char c = (char) cv::waitKey(10);
    if( c == 's' ) // single step
      {g_run = 1; cout << "Single step, run = " << g_run << endl;}
    if( c == 'r' ) // run mode
      {g_run = -1; cout << "Run mode, run = " << g_run <<endl;}
    if( c == 27 )
      break;

  }
  return(0);

}
```

本质上，代码中添加一个全局变量来表示滑动条的位置并且添加一个回调函数来更改这个变量来重新定位视频读取的位置。在创建滑动条和回调函数之后，我们就开始运行程序，[注5]让我们从全局变量开始更详细地查看整个程序。

```
    int g_slider_position = 0;
    int g_run             = 1;
    int g_dontset         = 0;      // start out in single-step mode
    VideoCapture g_cap;
```

在程序的开始处，我们定义了一个全局变量g_slider_position来存储滑动条位置。由于回调函数需要访问帧读取结构g_cap，所以我们最好将这个结构也提升为全局变量。为了保证代码的易读性。我们会在每一个全局变量前加上g_前缀以表明这是一个全局变量。同理，全局变量g_run将在新的跳转触发之后置0。当它为正的时候，指示在停止之前程序要播放多少张图片；当它为负的时候，表示系统处于连续视频播放模式。

为了防止冲突，在用户单击了滑动条跳转到一个视频的新位置之后，我们将会通过设置g_run变量为1使视频进入单步模式。即使是这样，也存在一个小问题：当视频前进的时候，我们希望滑动条也能够随着视频的当前进度前进。我们通过在程序中调用滑动条的回调函数实现这一功能。但是，我们并不希望在这个时候进入单步模式。为了避免这样的事情发生，我们引入最后一个全局变量g_dontset来避免在调整进度条位置的时候触发单步模式。

注5：　注意，有一些AVI格式和MPEG格式的文件不允许在视频中向后跳转。

```
void onTrackbarSlide(int pos, void *) {

  g_cap.set(cv::CAP_PROP_POS_FRAMES, pos);

  if( !g_dontset )
    g_run = 1;
  g_dontset = 0;

}
```

现在，定义一个用户调整滑动条的时候执行的回调程序。这个程序将会传入一个32位的整型数值以表示当前的位置。这将会是进度条存在的新的位置。在回调函数的内部，我们在新的位置通过调用g_cap.set()来真正使进度条移到我们希望的位置。if()语句设置程序是否进入单步模式。只有在用户触发滑动条事件的时候，这个设置才会生效；在系统自动调用该回调函数的时候，单步模式不会生效。

调用g_cap.set()是我们以后经常使用的一个操作，它通常配合g_mcap.get()使用。这些程序允许我们配置（或是询问，这将在稍后介绍）cv::VideoCapture结构中的许多变量。在本例中，我们输入参数CV::CAP_PROP_POS_FRAMES，这个参数指示我们想要帧集合的读取位置。[注6]

```
int frames = (int) g_cap.get(cv::CAP_PROP_FRAME_COUNT);
int tmpw   = (int) g_cap.get(cv::CAP_PROP_FRAME_WIDTH);
int tmph   = (int) g_cap.get(cv::CAP_PROP_FRAME_HEIGHT);
cout << "Video has " << frames << " frames of dimensions("
     << tmpw << ", " << tmph   << ")." << endl;
```

主程序的核心和示例2-3一致，所以我们将把重点放在我们添加了什么上面。第一个不同是在打开视频之后我们用g_cap.get()来确定总帧数以及视频的高和宽。这些数字打印出来之后，我们需要这些视频中和帧有关的信息来校准滑动条（在下一步）。

```
createTrackbar("Position", "Example2_4", &g_slider_position, frames,
               onTrackbarSlide);
```

下一步我们将创建一个滑动条，这个函数cv::createTrackbar()允许我们给滑动条一个标签[注7]（在本例中，是Position）并且指明在哪一个窗口放置我们的滑动条。我们提供一个和滑动条绑定的变量——滑动条能够达到的最大值（视频的总帧数）以及一个在滑动条移动时候的回调函数（不需要的时候也可以输入NULL）。

```
if( g_run != 0 ) {
```

注6：因为HighGUI的代码是高度自动化的，所以当一个新的视频位置被请求的时候，将自动处理一些（诸如请求的帧并非是一个关键帧等）问题。如果遇到这种情况，它会从请求位置之前的一个关键帧开始播放并且快进到请求的位置，免得我们考虑这些繁琐的问题。

```
    g_cap >> frame; if(!frame.data) break;
    int current_pos = (int)g_cap.get(cv::CAP_PROP_POS_FRAMES);
    g_dontset = 1;

    cv::setTrackbarPos("Position", "Example2_4", current_pos);
    cv::imshow( "Example2_4", frame );

    g_run-=1;

  }
```

在while循环中，为了读取和显示视频中的帧，我们还会获取在视频中的当前位置，设置g_dontset使下一个滑动条的回调函数不会让我们进入单步模式，之后我们将会委托滑动条的回调函数来更新滑动块在滑动条上的位置以便显示给用户。全局变量g_run将会减1，它的效果是让我们保持单步模式或者依据用户的设置让视频正常播放。接下来我们将会看到以下结构：

```
char c = (char) cv::waitKey(10);
if( c == 's' ) // single step
  {g_run = 1; cout << "Single step, run = " << g_run << endl;}
if( c == 'r' ) // run mode
  {g_run = -1; cout << "Run mode, run = " << g_run <<endl;}
if( c == 27 )
  break;
```

在while循环的底部，我们会等待用户的键盘输入。如果S键被按下，我们将会进入单步模式（g_run将会被设置为1，会使得程序只读取一张图片）。如果R被按下，我们将会进入连续视频模式（g_run将会被设置为-1，之后进一步递减使其对于之后任意帧都为负）。最后，如果Esc被按下，程序将会终止。再次提醒你注意，为了让代码更简洁，我们略过了回收窗口垃圾这一步cv::destroyWindow()。

简单的变换

很好，现在你已经可以用OpenCV来自己动手建立一个视频播放器了，这和现有的那些播放器差不多，但是我们的关注点在计算机视觉，所以希望做一些计算机视觉相关的工作。许多基础的计算机视觉工作都包括对视频流使用滤波器。我们将会对已有的程序进行修改以对视频中的每一帧实现一些简单的操作。

一个最简单的操作就是对图像的平滑处理，这将有通过高斯核或者其他核卷积效减小图像的信息量。OpenCV使得这样使用高斯核的卷积非常容易实现。如示例2-5所示，我们

注7：因为HighGUI是一个轻量级的、易于使用的工具，所以cv::createTrackbar()不会区分滑动条的名称和真正显示在屏幕上的滑动条旁边的标签，你也许已经注意到cv::namedWindow()也不区分窗口的名称以及真正在窗口上显示的名称。

可以新建一个名为"Example4-out"的窗口开始，并且在这个窗口显示处理结果。所以，在我们调用cv::imshow()在输入窗口并显示新获得的图像之后，我们就可以在output窗口计算并且显示处理结果。

示例2-5：加载图像并且在显示之前平滑处理图像

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

void example2_5( const cv::Mat & image ) {

  // Create some windows to show the input
  // and output images in.
  //
  cv::namedWindow( "Example2_5-in", cv::WINDOW_AUTOSIZE );
  cv::namedWindow( "Example2_5-out", cv::WINDOW_AUTOSIZE );

  // Create a window to show our input image
  //
  cv::imshow( "Example2_5-in", image );

  // Create an image to hold the smoothed output
  //
  cv::Mat out;

  // Do the smoothing
  // ( Note: Could use GaussianBlur(), blur(), medianBlur() or bilateralFilter(). )
  //
  cv::GaussianBlur( image, out, cv::Size(5,5), 3, 3);
  cv::GaussianBlur(   out, out, cv::Size(5,5), 3, 3);

  // Show the smoothed image in the output window
  //
  cv::imshow( "Example2_5-out", out );

  // Wait for the user to hit a key, windows will self destruct
  //
  cv::waitKey( 0 );

}
```

第一个cv::imshow()的调用和之前我们的使用方法没有什么差别。但是下一个调用的过程中，我们申请了另一个图像结构，接下来，C++对象cv::Mat使我们的工作变得轻松了。我们只需要实例化一个输出矩阵out，这将由程序自行管理并在合适的时候自动分配，释放内存。为了让这点更清晰，我们将其输入到两个cv::GaussianBlur()函数中。在第一次调用中，输入图像被5×5大小的高斯核模糊并且被写入out变量中。高斯核的大小必须是奇数，因为高斯卷积会在其覆盖区域的中心输出结果。在下一次调用cv::GaussianBlur()的时候，由于被分配了临时的存储空间，所以out可以同时作为输入和输出。执行了两次模糊操作的图像被作为结果输出，在终止之前等待用户进行键盘事件，然后在对象生命周期结束的时候清理申请的内存。

不那么简单的变换

接下来，我们要了解如何做更多有意思的事情。在示例2-5中，我们没有任何特殊目的地使用了高斯模糊。现在我们将使用高斯模糊来对一张图像实现基2的降采样[Rosenfeld80]。如果我们对图像进行多次降采样，就要建立一个尺度空间（也称为“图像金字塔”），这一方法是计算机视觉用于处理传感器和目标尺度变化的常用手段之一。

对于那些了解信号处理和香农-奈奎斯特采样理论[Shannon49]的读者，对信号的降采样(在本例中，我们创建一个图像并对每个像素进行采样)等效于和一系列脉冲函数进行卷积(将这些函数视为“峰值”)。这样的采样会把高频分量引入输出信号（图像）。为了避免这样的事情发生，我们希望首先通过一个高通滤波器来限制信号带宽，使其能够在采样频率之内。在OpenCV中，高斯模糊以及降采样通过`cv::pyrDown()`函数来实现，我们的实现在示例2-6中。

示例2-6：使用cv::pyrDown()来创建一个新的图像，其宽高均为原始图像的一半

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

int main( int argc, char** argv ) {

  cv::Mat img1,img2;

  cv::namedWindow( "Example1", cv::WINDOW_AUTOSIZE );
  cv::namedWindow( "Example2", cv::WINDOW_AUTOSIZE );

  img = cv::imread( argv[1] );
  cv::imshow( "Example1", img1 );

  cv::pyrDown( img1, img2);
  cv::imshow( "Example2", img2 );

  cv::waitKey(0);

  return 0;

};
```

然后，让我们来分析一段相似但更加复杂的代码，示例2-7这段代码包含有Canny边缘检测器[Canny86]`cv::Canny()`。在示例2-7中，边缘检测器通过`cv::cvtColor()`函数生成一个和原图一样大小但只有一个通道的图像，从而将图像从BGR图像转换为灰度图，这个操作在OpenCV中定义为宏`cv::COLOR_BGR2GRAY`。

示例2-7：Canny边缘检测器输出一个单通道的（灰度）图像

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

int main( int argc, char** argv ) {
```

```
    cv::Mat img_rgb, img_gry, img_cny;

    cv::namedWindow( "Example Gray",  cv::WINDOW_AUTOSIZE );
    cv::namedWindow( "Example Canny", cv::WINDOW_AUTOSIZE );

    img_rgb = cv::imread( argv[1] );

    cv::cvtColor( img_rgb, img_gry, cv::COLOR_BGR2GRAY);
    cv::imshow( "Example Gray", img_gry );

    cv::Canny( img_gry, img_cny, 10, 100, 3, true );
    cv::imshow( "Example Canny", img_cny );

    cv::waitKey(0);

}
```

这时，我们可以将很多简单的操作串联起来。比如说，如果我们想要收缩两次图像然后寻找收缩过两次的图像中的边缘，我们可以像示例2-8一样处理。

示例2-8：在一个简单图像处理流程中结合图像金字塔操作（两次）和Canny边缘检测器

```
cv::cvtColor( img_rgb, img_gry, cv::BGR2GRAY );
cv::pyrDown( img_gry, img_pyr );
cv::pyrDown( img_pyr, img_pyr2 );
cv::Canny( img_pyr2, img_cny, 10, 100, 3, true );
// do whatever with 'img_cny'
//
...
```

在示例2-9中，我们展示了一个简单的方法来读写示例2-8的像素值。

示例2-9：读写示例2-8中的像素值

```
int x = 16, y = 32;
cv::Vec3b intensity = img_rgb.at< cv::Vec3b >(y, x);

// ( Note: We could write img_rgb.at< cv::Vec3b >(x,y)[0] )
//
uchar blue  = intensity[0];
uchar green = intensity[1];
uchar red   = intensity[2];

std::cout << "At (x,y) = (" << x << ", " << y <<
          "): (blue, green, red) = (" <<
          (unsigned int)blue <<
          ", " << (unsigned int)green << ", " <<
          (unsigned int)red << ")" << std::endl;

std::cout << "Gray pixel there is: " <<
             (unsigned int)img_gry.at<uchar>(y, x) << std::endl;

x /= 4; y /= 4;
```

```
std::cout << "Pyramid2 pixel there is: " <<
             (unsigned int)img_pyr2.at<uchar>(y, x) << std::endl;

img_cny.at<uchar>(x, y) = 128; // Set the Canny pixel there to 128
```

从摄像头中读取

“视觉”在计算机世界中可以表示很多东西，在一些情况下，我们分析从任意地方加载的静止的图像。在另一些情况下，我们会分析从硬盘中读取的视频。然而在更多的情况下，我们想要和从某种摄像头中读取的实时数据流进行交互。

OpenCV——或更准确地说，OpenCV中的HighGUI模块——为我们提供了一个简单的方式来驾驭这种情况。这个方法很接近于我们使用`cv::VideoCapture`从硬盘读取视频的方式，事实上，`cv::VideoCapture`对磁盘上的文件和摄像头是有一致接口的。对于前者来说，需要给它一个指示读取文件名的路径，对于后者来说，需要给它一个相机ID号（如果只有一个摄像头连接，这个ID号通常为0），ID的默认值是`-1`，这意味着“随意选择一个”，当然，当只有一个摄像头可以选择的时候这能够很好地工作（参考第8章，查阅详细解释），示例2-10展示了从文件或者摄像头中读取视频。

示例2-10：同一个对象可以读取视频文件，也可以连接摄像头

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

int main( int argc, char** argv ) {

  cv::namedWindow( "Example2_10", cv::WINDOW_AUTOSIZE );

  cv::VideoCapture cap;
  if (argc==1) {
    cap.open(0);           // open the first camera
  } else {
    cap.open(argv[1]);
  }
  if( !cap.isOpened() ) {  // check if we succeeded
    std::cerr << "Couldn't open capture." << std::endl;
    return -1;
  }

  // The rest of program proceeds as in #simple_opencv_program_for_playing_a_vide
  ...
```

在示例2-10中，如果一个文件名被提供，OpenCV将如示例2-3所示打开指定的文件。如果没有给予任何文件名，程序将尝试打开一个摄像头。我们加入一段检查代码来确定程序是否真的开启了一些东西，如果没有，将报错。

写入AVI文件

在许多应用中，我们都希望记录一个输入流或完全不同的一些图像到视频流中。OpenCV提供一种简单的方法来实现这一点。正如我们创建一个从视频流中捕获帧的对象一样，我们可以创建一个写入对象以便将帧依次输入到一个视频文件中。允许我们进行这个工作的对象是cv::VideoWriter。

一旦将其实例化，我们就可以将每一帧图像输入到cv::VideoWriter对象中。然后在完成写入之后调用cv::VideoWriter.release()方法。为了让事情更有趣一些，示例2-11展示了一个程序，该程序会打开一个视频文件，读取它的内容后将其转换为对数极坐标（log-polar）形式（类似于人眼真正捕捉到的图像的形式，详细信息请参阅第11章），然后将对数极坐标图像写入一个新的视频文件。

示例2-11：一个完整的读取彩色视频并转换为对数极坐标视频的程序

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

int main( int argc, char* argv[] ) {

  cv::namedWindow( "Example2_11", cv::WINDOW_AUTOSIZE );
  cv::namedWindow( "Log_Polar",   cv::WINDOW_AUTOSIZE );

  // ( Note: could capture from a camera by giving a camera id as an int.)
  //
  cv::VideoCapture capture( argv[1] );

  double fps = capture.get( cv::CAP_PROP_FPS );
  cv::Size size(
    (int)capture.get( cv::CAP_PROP_FRAME_WIDTH ),
    (int)capture.get( cv::CAP_PROP_FRAME_HEIGHT )
  );

  cv::VideoWriter writer;
  writer.open( argv[2], CV_FOURCC('M','J','P','G'), fps, size );

  cv::Mat logpolar_frame, bgr_frame;
  for(;;) {

    capture >> bgr_frame;
    if( bgr_frame.empty() ) break; // end if done

    cv::imshow( "Example2_11", bgr_frame );

    cv::logPolar(
      bgr_frame,                    // Input color frame
      logpolar_frame,               // Output log-polar frame
      cv::Point2f(                  // Centerpoint for log-polar transformation
        bgr_frame.cols/2,           //  x
        bgr_frame.rows/2            //  y
      ),
```

```
      40,                              // Magnitude (scale parameter)
      cv::WARP_FILL_OUTLIERS           // Fill outliers with 'zero'
    );

    cv::imshow( "Log_Polar", logpolar_frame );
    writer << logpolar_frame;

    char c = cv::waitKey(10);
    if( c == 27 ) break;        // allow the user to break out
  }

  capture.release();
}
```

这个程序里面有很多我们非常熟悉的元素。首先是打开视频并且读取一些在cv::VideoWriter建立时用到的属性（每秒播放的帧数以及图像宽高）。在此之后，我们将会从cv::VideoReader中逐帧读取视频，并将每一帧转换为对数极坐标形式，然后将转换的对数极坐标图像写入新的视频文件，循环操作直到源文件读完或者用户按下Esc键。

cv::VideoWriter的调用有几个需要理解的参数。第一个参数是新建视频文件的文件名，第二个参数是视频编码方式，指明视频将以何种方式进行压缩。现在有非常多的编码器可以选择，但是选择的任何编码器都必须确保可以在你的机器上使用（编码器是与OpenCV分开安装的）。在我们的例子中，我们选择了通用的MJPG编码器，我们通过使用OpenCV所提供的的宏CV_FOURCC()指定它，这个宏将四个字符作为参数，所有编码器都有类似的四个字符作为其标识。比如本例中用于运动jpeg图像编码的字符为“MJPG”，所以我们指定CV_FOURCC('M', 'J', 'P', 'G')。接下来的参数是帧率以及图像的大小。在我们的例子中，这两个值和原始视频一致。

小结

在进入下一章之前，我们应该花些时间来盘点一下我们所处的阶段，并看看未来会发生什么。我们已经看到，OpenCV API为我们提供了各种易于使用的工具，用于读取和写入静态图像和视频以及从相机捕捉视频。我们还看到，库包含用于操作这些图像的基本功能。我们还没有看到库的强大元素，它允许对整个抽象数据类型进行更复杂的操作，这些数据类型对于解决实际的视觉问题非常重要。

在接下来的几章中，我们将更深入地探究基础知识，更详细地了解与接口相关的函数和图像数据类型。我们将研究原始图像处理操作符，然后再进行一些更高级的操作。此后，我们将准备探索API为各种任务提供的多种专门服务，如相机的校准、跟踪和识别。准备好了吗？赶紧上车吧!

练习

下载并安装OpenCV，如果之前还没有这么做的话。系统性地浏览它的文件夹结构。尤其是*doc*文件夹，在那里可以加载*index.html*，它可以链接到库的主文档。在此之后，浏览库的主要部分：`core`模块包含基础数据类型以及算法，`improc`模块包含图像处理和视频处理算法，`ml`模块包含机器学习和聚类算法，`highgui`模块包含输入输出功能。检查*.../samples/cpp*文件夹，那里有许多有用的例子。

1. 使用这本书或者*http://opencv.org*所提供的的安装、构建指令，在debug模式和release模式下编译这个库。这也许会花一些时间，但是你需要编译得到的库和*dll*文件。同时注意设置*cmake*编译位于*.../opencv/samples/*文件夹下的例子。
2. 切换到*.../opencv/samples/*的编译输入目录（对我们来说，位于*.../trunk/eclipse_build/bin*）然后查看*lkdemo.cpp*（这是一个运动追踪的示例）。连接上一个相机，然后运行代码，它会提示选择窗口，按下**r**来初始化追踪。你可以通过用鼠标点击视频中的位置来添加点。也可以通过按下**n**来切换到只查看追踪点的模式。再次按下**n**会在“白天”和“夜晚”模式中切换。
3. 将示例2-11中的代码和示例2-6中的代码连接起来，建立一个读取视频并存储降采样后彩色图像的程序。
4. 修改练习3中的程序，并结合示例2-2中窗口显示的代码来显示处理的图像。
5. 修改练习4中的代码，添加一个示例2-4中的滑动条，这样一来，用户可以动态控制金字塔的降采样等级（从2到8）。可以跳过存储的步骤，但要将处理结果显示出来。