第 1 章　概述

什么是 **OpenCV**

OpenCV 是一个开源(参见 http://opensource.org)的计算机视觉库，项目主页为 http://SourceForge.net/projects/opencvlibrary。OpenCV 采用 C/C++语言编写，可以运行在 Linux/Windows/Mac 等操作系统上。OpenCV 还提供了 Python、Ruby、MATLAB 以及其他语言的接口。

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 代码编写，能够充分利用多核处理器的优势。如果是希望在 Intel 平台上得到更快的处理速度，可以购买 Intel 的高性能多媒体函数库 IPP(Integrated Performance Primitives)。IPP 库包含许多从底层优化的函数，这些函数涵盖多个应用领域。如果系统已经安装了 IPP 库，OpenCV 会在运行时自动使用相应的 IPP 库。

OpenCV 的一个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 包含的函数有 500 多个，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 还提供了 MLL(Machine Learning Library)机器学习库。该机器学习库侧重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的机器学习场合。

OpenCV 的应用领域

大多数计算机科学家和程序员已经意识到计算机视觉的重要作用。但是很少有人知道计算机视觉的所有应用。例如，大多数人或多或少地知道计算机视觉可用在监控方面，也知道视觉被越来越多地用在网络图像和视频方面。少数人也了解计算机视觉在游戏界面方面的应用。但是很少有人了解大多数航空和街道地图图像(如 Google 的 Street View)也大量使用计算机定标和图像拼接技术。一些人知道安全监控、无人飞行器或生物医学分析等方面的应用，但是很少人知道机器视觉是多么广泛地被用在工厂中：差不多所有的大规模制造的产品都在流水线上的某个环节上自动使用视觉检
测。

【1～2】

OpenCV 所有的开放源代码协议允许你使用 OpenCV 的全部代码或者 OpenCV 的部分代码生成商业产品。使用了 OpenCV 后，你不必对公众开放自己的源代码或改善后的算法，虽然我们非常希望你能够开放源代码。许多公司(IBM，Microsoft，Intel，SONY，Siemens 和 Google 等其他公司)和研究单位(例如斯坦福大学、MIT、CMU、剑桥大学和 INRIA)中的人都广泛使用 OpenCV，其部分原因是 OpenCV 采用了这个宽松的协议。Yahoo groups 里有一个 OpenCV 论坛(http://groups.yahoo.com/group/OpenCV)，用户可以在此发帖提问和讨论；该论坛大约有 20 000 个会员。OpenCV 在全世界广受欢迎，在中国、日本、俄罗斯、欧洲和以色列都有庞大的用户群。

自从OpenCV在1999年1月发布alpha版本开始，它就被广泛用在许多应用领域、产品和研究成果中。相关应用包括卫星地图和电子地图的拼接，扫描图像的对齐，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，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。将视觉识别技术用在声谱图上，OpenCV 可以进行声音和音乐识别。在斯坦福大学的 Stanley 机器人项目中，OpenCV 是其视觉系统的关键部分。Stanley 在 DARPA 机器人沙漠挑战赛中，赢得了二百万美元奖金[Thrun06]。

什么是计算机视觉

计算机视觉[①]是将来自静止图像或视频的数据转换成一个决策或者一种新的表达方式的过程，所有的这些转换都是为了达到某个目标。输入数据可以包含一些辅助信息，如“摄像机架在汽车上”或“激光扫描仪在 1 米处发现一个物体”。最终的决策可能是“场景中有一个人”或“在这个切片中有 14 个肿瘤细胞”。一种新的表达方式可以是将一张彩色照片转为灰度照片，或者从图像序列中去除摄像机晃动影响。

因为人类是视觉动物，所以会误以为可以很容易地实现计算机视觉。当你凝视图像时，从中找到一辆汽车会很困难么？你凭直觉会觉得很容易。人脑将视觉信号划分入很多个通道，将各种不同的信息输入你的大脑。你的大脑有一个关注系统，会根据任务识别出图像的重要部分，并做重点分析，而其他部分则分析得较少。在人类视觉流中存在大量的反馈，但是目前我们对之了解甚少。肌肉控制的传感器以及其他所有传感器的输入信息之间存在广泛的关联，这使得大脑可以依赖从出生以来所学到的信息。大脑中的反馈在信息处理的各个阶段都存在，在传感器硬件(眼睛)中也存在。在眼睛中通过反馈来调节通过瞳孔的进光量，以及调节视网膜表面上的接收单

元。

【2～3】

在计算机视觉系统中，计算机接接收到的是来自摄像机或者磁盘文件的一个数值矩阵。一般来说，没有内置的模式识别系统，没有自动控制的对焦和光圈，没有多年来经验的积累。视觉系统通常很低级。图 1-1 显示了一辆汽车的图像。在此图中，我们可以看到车的一侧有一个反光镜，而计算机“看”到的只是一个数值的矩阵。矩阵中的每个数值都有很大的噪声成分，所以它仅仅给出很少的信息，

这个数值矩阵就是计算机“看”到的全部。我们的任务是将这个具有噪声成分的数值矩阵变成感知：“反光镜”。图 1-2 形象地解释了为什么计算机视觉如此之难。

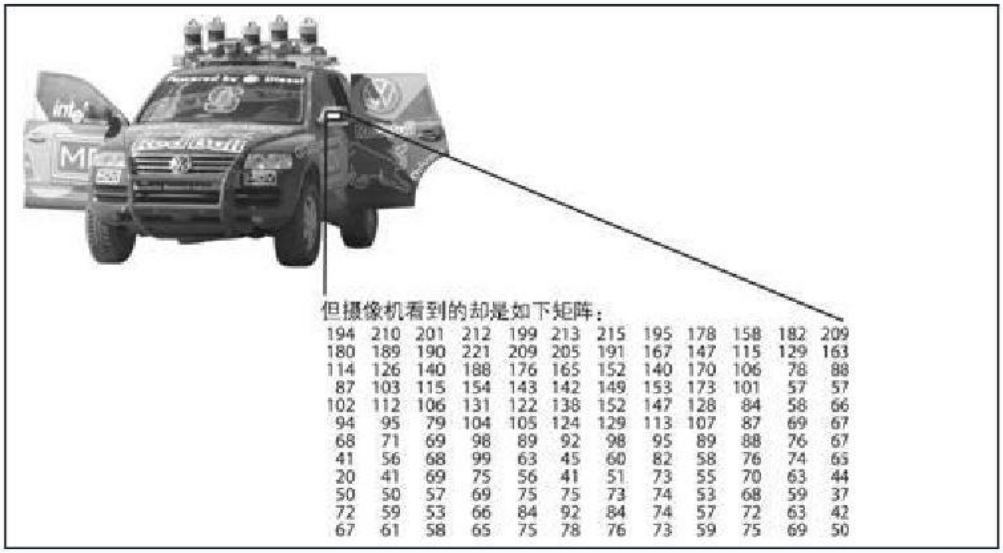

图1-1：对一个计算机来说，汽车的反光镜只是一个数值矩阵

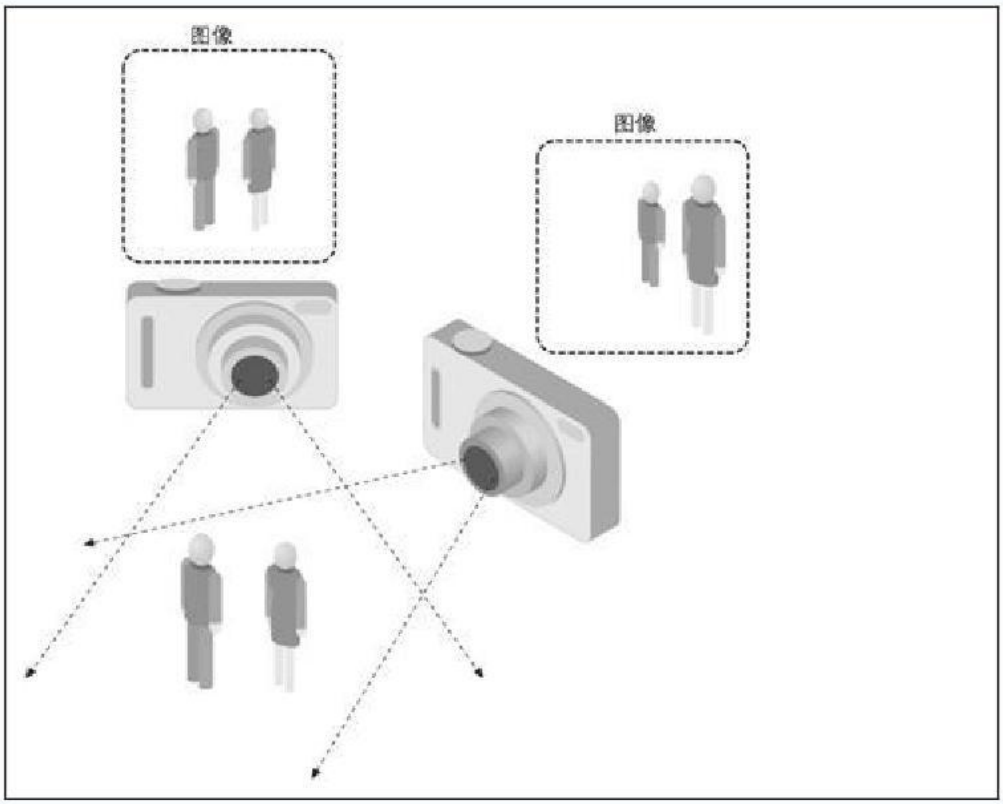

图1-2：视觉问题的病态本质：随着视点的变化，物体的二维外观会变化很大

实际上，计算机视觉问题比前面我们提到的更糟糕，它是不可解的。给出三维世界的二维视图，是没有固定方法来重建出三维信息的。在理论上，此类病态问题

没有惟一和确定的解。即使数据很完美，同一张二维图像也可以表示多种三维场景。然而，如前面提到的，数据会被噪声和形变影响。这些影响来自真实世界的变化(天气、光线、反射、运动)，镜头和机械结构的不完美，传感器上的长时间感应(运动模糊)，传感器上和其他电子器件上的电子噪声，以及图像采集后的图像压缩引入的变化。有如此多令人头疼的问题，我们如何取得进展？ 【3～4】

设计实际系统时，为了克服视觉传感器的限制，通常会使用一些其他的上下文知识。考虑这样一个例子，移动机器人在室内寻找并捡起订书机。机器人可以利用这个先验知识：可在办公室内发现桌子，订书机最可能在桌子上被找到。这给出了一个隐含的尺寸参考或参照，也就是订书机能够放在桌子上。这也可以用于消除在不可能的地方(例如在天花板或者窗户上)错误识别出订书机的可能性。机器人也完全可以忽略一个 200 英尺大小的跟订书机形状类似的广告飞艇，因为飞艇周围没有桌子的木纹背景。与之相反，在图像检索中，数据库中的所有订书机图像都是对真正的订书机拍摄的，而且尺寸很大和形状不规划的订书机图像一般不可能被拍到。也就是拍摄者一般只拍摄真正的、普通大小的订书机图像。而且人们拍照时一般会将被拍物体置于中心，且将物体放在最能表现其特征的方向上。因此在由人拍摄的图像中，具有相当多的隐含信息。

【4～5】

我们也可以使用机器学习技术对上下文信息进行显式建模。隐含的变量(例如物体大小、重力方向及其他变量)都可以通过标记好的训练数据里的数值来校正。或者，也可以通过其他的传感器来测量隐含的变量。使用激光扫描仪可以精确测量出一个物体的大小。计算机视觉面临的另一个难题是噪声问题。我们一般使用统计的方法来克服噪声。例如，一般来说不可能通过比较一个点和它紧密相邻的点来检测图像里的边缘。但是如果观察一个局部区域的统计特征，边缘检测会变得容易些。由局部区域卷积的响应连成的点串，构成边缘。另外可以通过时间维度上的统计来抑制噪声。还有一些其他的技术，可以从数据中学习显式模型，来

解决噪声和畸变问题。例如镜头畸变，可以通过学习一个简单多项式模型的参数来描述这种畸变，然后可以几乎完全校正这种畸变。

计算机视觉拟根据摄像机数据来采取行动或者做出决策，这样的行动或决策是在一个指特定目的或任务的环境中来解决。我们从图像去除噪声和损坏区域，可以让监控系统在有人爬过栅栏时给出报警，或者在一个游乐园里监控系统能够数出总共有多少人通过了某个区域。在办公室巡游的机器人的视觉软件所采用的方法与固定摄像机的不同，因为这两个系统有不同的应用环境和目标。通用的规律是：对计算机视觉应用环境的约束越多，则越能够使用这些约束来简化问题，从而使最终的解决方案越可靠。

OpenCV 的目标是为解决计算机视觉问题提供基本工具。在有些情况下，它提供的高层函数可以高效地解决计算机视觉中的一些很复杂的问题。当没有高层函数时，它提供的基本函数足够为大多数计算机视觉问题创建一个完整的解决方案。对于后者，有几个经过检验且可靠的使用 OpenCV 的方法；所有这些方法都是首先大量使用 OpenCV 函数来解决问题。一旦设计出解决方案的第一个版本，便会了解它的不足，然后可以使用自己的代码和知识来解决(更为广知的一点是“解决实际遇到的问题，而不是你想像出来的问题”)。你可以使用第一个版本的解决方案作为一个基准，用之评价解决方案的改进程度。解决方案所存在的不足可以通过系统所用的环境限制来解

决。

【5～6】

OpenCV 的起源

OpenCV 诞生于 Intel 研究中心，其目的是为了促进 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。一个在 Intel 工作的 OpenCV 作者在访问一些大学时，注意到许多顶尖大学中的研究组(如 MIT 媒体实验室)拥有很好的内部使用的开放计算机视觉库—— (在学生们之间互相传播的代码)，这会帮助一个新生从高的起点开始他/她的计算机视觉研究。这样一个新生可以在以前的基础上继续开始研究，而不用从底层写基本函数。

因此，OpenCV 的目的是开发一个普遍可用的计算机视觉库。在 Intel 的性能库团队的帮助下[②]，OpenCV 实现了一些核心代码以及算法，并发给 Intel 俄罗斯的库团队。这就是 OpenCV 的诞生之地：在与软件性能库团队的合作下，它开始于 Intel 的研究中心，并在俄罗斯得到实现和优化。

俄罗斯团队的主要负责人是 Vadim Pisarevsky，他负责管理项目、写代码并优化 OpenCV 的大部分代码，在 OpenCV 中很大一部分功劳都属于他。跟他一起，Victor Eruhimov 帮助开发了早期的架构，Valery Kuriakin 管理俄罗斯实验室并提供了很大的支持。在开始时，OpenCV 有以下三大目标。

l 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。能有效地避免“闭门造车”。

l 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的且可改写。

l 本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。

这些目标说明了 OpenCV 的缘起。计算机视觉应用的发展会增加对快速处理器的需求。与单独销售软件相比，促进处理器的升级会为 Intel 带来更多收入。这也许是为什么这个开放且免费的库出现在一家硬件生产企业中，而不是在一家软件公司中。从某种程度上说，在一家硬件公司里，在软件方面会有更多创新的空间。

【6】

任何开放源代码的努力方面，达到一定的规模使项目自己能够发展是非常重要的。目前 OpenCV 已经有大约二百万的下载量，这个数字仍然在以平均每个月 26 000 的下载量递增。OpenCV 用户组大约有 20 000 个会员。OpenCV 吸纳了许多用户的贡献，核心开发工作已经从 Intel 转移到别处[③]。OpenCV 过去的开发历程如图 1-3 所示。在发展中，OpenCV 受到网络经济泡沫破裂的影响，也受到无数次管理和发展方向变化的影响。在这些变故中，OpenCV 曾经有多次缺乏 Intel 公司人员的支持。然而，随着多核时代的到来，以及计算机视觉的更多应用的出现，OpenCV 的价值开始提升。现在 OpenCV 在几个研究所中的开发都很活跃，所以不

久应该会看到更多的功能出现，如多摄像机标定、深度信息感知、视觉与激光扫描的融合、更好的模式识别算法，同时还会支持机器人视觉的需求。关于 OpenCV 未来的发展，请参考第 14 章。

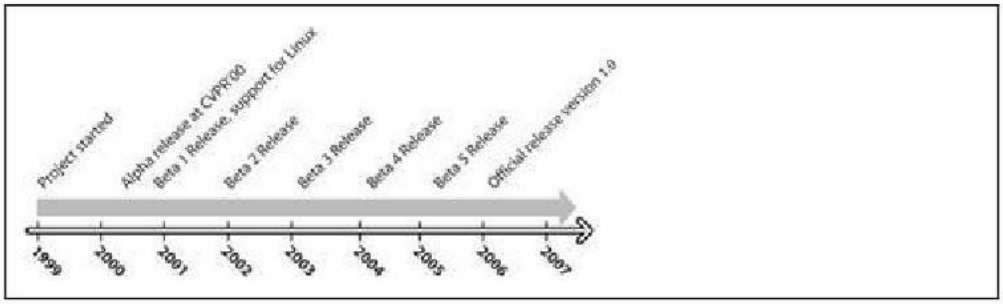

图1-3：OpenCV发展路线图

用 **IPP** 给 **OpenCV** 加速

因为 OpenCV 曾由 Intel 性能基元(IPP)团队主持，而且几个主要开发者都与 IPP 团队保持着良好的关系，所以 OpenCV 利用了 IPP 高度手工优化的代码来实现加速。使用 IPP 获得的提速是非常显著的。图 1-4 比较了另外两个视觉库 LTI[LTI]和 VXL[VXL]与 OpenCV 以及 IPP 优化的 OpenCV 的性能。请注意，性能是 OpenCV 追求的一个关键目标；它需要实时运行代码的能力。

OpenCV 使用优化了的 C 和 C++代码实现。它对 IPP 不存在任何依赖。但如果安装了 IPP，那么 OpenCV 将会通过自动载入 IPP 动态链接库来获取 IPP 的优势，来提升速

度。

【6～7】

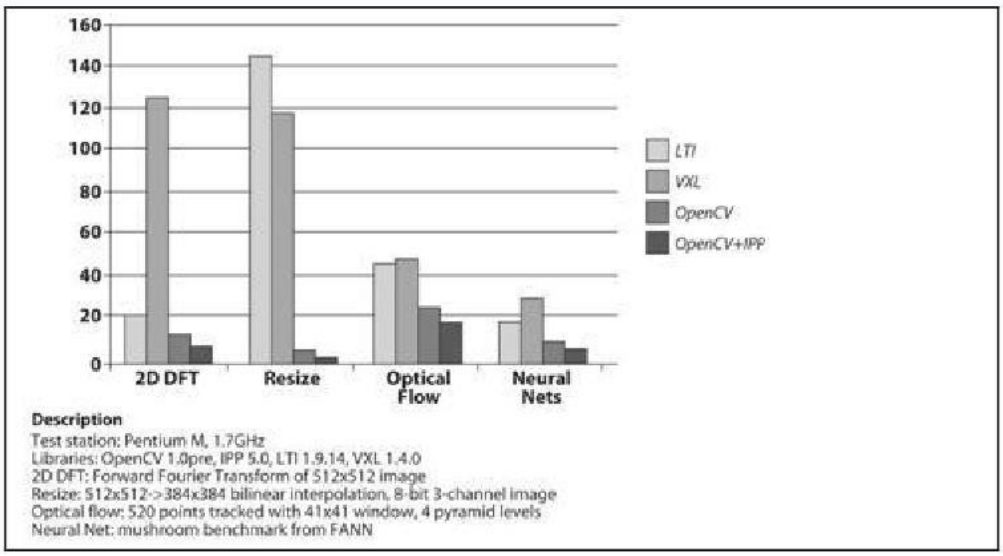

图1-4：另外两个视觉库(LTI 和 VXL)与 OpenCV(不使用和使用 IPP)的四个不同性能指标的比较：每个指标的四个柱图分别表示四个库的得分，得分与运行时间成正比；在所有指标中，OpenCV 均优于其他的两个库，且用 IPP 优化的 OpenCV 优于没有使用IPP 优化的OpenCV

OpenCV 属于谁

虽然 OpenCV 项目是 Intel 发起的，但这个库一直致力于促进商业和研究使用。它是开放源代码且免费的，无论是商业使用还是科研使用，OpenCV 的代码可用于或者嵌入(整体或部分)其他的应用程序中。它不强迫你开放或者免费发放你的源代码。它也不要求你将改进的部分提交到 OpenCV 库中——但我们希望你能够提交。

下载和安装 **OpenCV**

OpenCV 项目主页在 SourceForge 网站 http://SourceForge.net/projects/ opencvlibrary，对应的 Wiki 在 http://opencv.willowgarage.com。对于 Linux 系统，源代码发布文件为 opencv-1.0.0.tar.gz；对于 Windows 系统，则为 OpenCV_1.0.exe 安装程序。然而，最新的版本始终都在 SourceForge 的 SVN 仓库中。

安装

下载 OpenCV 库之后，就可以安装了。Linux 和 Mac OS 系统的安装细节可以查看.../opencv/目录下的 INSTALL 文本文件中的说明。INSTALL 文件中还描述了如何编译 OpenCV 和运行测试程序。对于 OpenCV 开发人员，INSTALL 还列出了所需的 autoconf、automake、libtool 和 swig 其他开发工
具。 【8～9】

Windows

从 SourceForge 网站下载 OpenCV 安装程序，然后运行安装程序。安装程序将安装 OpenCV，注册 DirectShow filter，然后进行一些安装后的处理。现在你就可以使用 OpenCV 了。你还可以进入目录.../opencv/_make，使用 MSVC++或者 MSVC.NET 2005 打开 opencv.sln，或者使用低版本的 MSVC++打开 opencv.dsw，然后生成 Debug 版的库，也可以重新生成 Release 版的库[④]。

如果需要使用 IPP 的优化功能，首先需要从 Intel 网站(http://www.intel.com/soft ware/products/ipp/index.htm)获得 IPP 并安装；请使用 5.1 或更新的版本。请确认二进制文件路径(例如 c:/program files/intel/ipp/5.1/ia32/bin)被添加到系统环境变量 PATH 中。现在 OpenCV 就能够自动探测到 IPP，并在运行时装载 IPP 了(详细信息请参考第 3 章)。

Linux

因为在 Linux 系统的各个发行版(SuSE，Debian，Ubuntu 等)的 GCC 和 GLIBC 版本并不一样，OpenCV 的 Linux 版本并不包含可直接使用的二进制库。如果发行版没有提供 OpenCV，则需要从源代码重新编译 OpenCV，具体的细节请参考.../opencv/INSTALL 文件。

如果要编译库和演示程序，需要版本为 2.x 或更高版本的 GTK+及其头文件。除此之外还要需要具有开发文件的 pkgconfig, libpng, zlib, libjpeg, libtiff 和 libjasper。同时还要安装版本为 2.3、2.4 或 2.5 的 Python 及其头文件(开发包)。同时还需要依赖 ffmpeg 0.4.9-pre1 或更高的版本中 libavcodec 和 libav*系列的库，ffmpeg 的最新版可以用以下命令获取：svn checkout svn://svn.mplayerhq.hu/ffmpeg/trunk ffmpeg。

从 http://ffmpeg.mplayerhq.hu/download.html 下载 ffmpeg 库[⑤]，ffmpeg 库的授权协议为 GNU 宽通用公共许可证(LGPL)。非 GPL 软件(如 OpenCV)使用 ffmpeg 库，需要生成和调用共享的 ffmpeg 库：

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

编译完成后会生成以下系列库文件：/usr/local/lib/libavcodec.so.*，/usr/local/lib/libavformat.so.*，/usr/local/lib/libavutil.so.*，及其对应的头文件/usr/local/include/libav

*。

下载了 OpenCV 后就可以编译 OpenCV 了[⑥]：

```
$> ./configure
$> make
$> sudo make install
$> sudo ldconfig
```

安装完成后，OpenCV 会被默认安装在以下目录：/usr/local/lib/ 和 /usr/local/include/opencv/。因此，用户需要将/usr/local/lib/添加到/etc/ld.so.conf 文件(之后需要执行 ldconfig 命令)，或者将该路径添加到 LD_LIBRARY_PATH 环境变量中。

同样在 Linux 平台也可以用 IPP 给 OpenCV 加速，IPP 的安装细节在前面已经提过。我们现在假设 IPP 安装在以下路径：/opt/intel/ipp/5.1/ia32/。修改初始化配置文件，添加<your install_path>/bin/和<your install_path>/bin/linux32 到 LD_LIBRARY_PATH 环境变量(可以直接编辑.bashrc 配置文件)：

```
LD_LIBRARY_PATH=/opt/intel/ipp/5.1/ia32/bin:/opt/intel/ipp/5.1/ia32/bin/linux32:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

另一个方法是将<your install_path>/bin 和<your install_path>/bin/linux32 添加到/etc/ld.so.conf 文件，每个文件占一行，完成后在 root 权限下(也可以使用 sudo 命令)执行 ldconfig 命令。

现在 OpenCV 就可以找到并能使用 IPP 的共享库了，具体的细节请参考/opencv/INSTALL。

Mac OS X

当写此书的时候，所有的功能都可以在 Mac OS X 下使用，但是仍然有一些限制(如 AVI 文件的写操作)；文件.../opencv/INSTALL 中详细描述了这些限制。

在 Mac OS X 下的编译需求和编译步骤跟 Linux 下类似，但是有如下不同。

- 默认情况下是使用 Carbon 而不是 GTK+。
- 默认情况下是使用 QuickTime 而不是 ffmpeg。
- pkg-config 是非必需的(它只在脚本 samplels/c/build_all.sh 中用到)
- 默认情况下不支持 RPM 和 ldconfig。使用命令 configure+make+sudo make install 来编译和安装 OpenCV；如果不是使用./configure --prefix=/usr 命令来配置的话，需要更新 DYLD_LIBRARY_PATH 变量。

如果要使用全部功能，需要使用 darwinports 来安装 libpng、libtiff、libjpeg 和 libjasper，然后使它们能够被脚本./configure 检测到(详细帮助请运行./configure --help)。对于大多数信息，可以参考 OpenCV Wiki(网址为 http://opencv.willowgarage.com/)和 Mac 相关的页面(网址为 http://opencv.willowgarage. com/Mac_OS_X_OpenCV_Port)。

通过**SVN** 获取最新的**OpenCV**代码

OpenCV 是一个相对活跃的开发项目，如果提交了 bug 的详细描述以及出错的代码，该 bug 会被很快修复。然而，OpenCV 一般一年才会发布一个或两个官方版本。如果用 OpenCV 开发比较重要的应用，你可能想获得修复了最新 bug 的最新 OpenCV 代码。如果要获取 OpenCV 的最新代码，需要通过 SourceForge 网站上的 OpenCV 库的 SVN(Subversion)获
得。 【10～11】

这里并不是一个 SVN 的完整教程。如果你参与过其他的开源项目，也许很熟悉 SVN。如果不了解 SVN，可以参考 Ben Collins-Sussman 等人所著的 Version Control with Subversion(O’Reilly 出版)。SVN 的命令行客户端一般被打包在 Linux、OS X 和大部分类 UNIX 系统中。对于 Windows 系统的用户，可以选择 TortoiseSVN(http://tortoisesvn.tigris.org/)客户端，很多命令被集成到 Windows 资源管理器的右键菜单中，使用很方便。

对于 Windows 用户，可使用 TortoiseSVN 检出最新源代码，检出地址为 https://opencvlibrary.svn.sourceforge.net/svnroot/opencvlibrary/trunk。

对于 Linux 用户，可以使用如下命令检出最新源代码：

```
svn co https://opencvlibrary.svn.sourceforge.net/svnroot/opencvlibrary/trunk
```

更多**OpenCV** 文档

OpenCV 的主要文档是随源代码一起发布的 HTML 帮助文件。除此之外，网上的参考文档还有 OpenCV Wiki 网站和以前的 HTML 帮助。

HTML 帮助文档

安装 OpenCV 后，在.../opencv/docs 子目录中有相应的 HTML 格式的帮助文件，打开 index.htm 文件，其中包含以下链接。

CXCORE

包含数据结构、矩阵运算、数据变换、对象持久(object persistence)、内存管理、错误处理、动态装载、绘图、文本和基本的数学功能等。

CV

包含图像处理、图像结构分析、运动描述和跟踪、模式识别和摄像机标定。

Machine Learning (ML)

包含许多聚类、分类和数据分析函数。

HighGUI

包含图形用户界面和图像/视频的读/写。

CVCAM

摄像机接口，在 OpenCV 1.0 以后的版本中被移除。

Haartraining

如何训练 boosted 级联物体分类器。文档在文件 .../opencv/apps/HaarTraining/ doc/haartraining.htm 中。

目录.../opencv/docs 中还有一个 IPLMAN.pdf 文件，它是 OpenCV 的早期文档。这个文档已经过时，阅读时一定要注意。但是这个文档中详细描述了一些算法以及某些算法中应该使用何种类型的图像。当然，本书是详细描述这些图像和算法的最佳参考资料。

Wiki 帮助文档

OpenCV 的文档 Wiki 所包含的内容比 OpenCV 安装文件自带的 HTML 帮助更新，并包含自带文档没有的一些内容。Wiki 网址为 http://opencv.willowgarage.com，它包含以下内容：

- 用 Eclipse 集成开发环境编译 OpenCV 的帮助
- 使用 OpenCV 进行人脸识别
- 视频监控
- 使用向导

l 摄像机支持

l 中文和韩文网站链接

另外一个 Wiki 地址为 http://opencv.willowgarage.com/wiki/CvAux，是下一节“OpenCV 架构和内容”提到的辅助函数的惟一文档。CvAux 模块包含以下信息：

l 双目匹配

l 多摄像机情况下的视点渐变

l 立体视觉中的三维跟踪

l 用于物体识别的PCA 方法

l 嵌入隐马尔可夫模型(HMM)

OpenCV 中文的 Wiki 地址为 http://www.opencv.org.cn/。

刚才提到的帮助文档并没有解释下面的问题：

l 哪些类型(浮点、整数、单字节；1-3 通道)的图像适用于某个函数？

l 哪些函数可以以in place 模式(输入和输出使用同一个图像结构)调用？

l 一些复杂函数的调用细节(如contours)？

l 目录…/opencv/samples/c/中各个例子的运行细节？

l 为什么要这样使用函数，而不仅仅是如何使用？

l 怎么样设置某些函数的参数？

本书的目的是解答上述问题。

OpenCV 的结构和内容

OpenCV 主体分为五个模块，其中四个模块如图 1-5 所示。OpenCV 的 CV 模块包含基本的图像处理函数和高级的计算机视觉算法。ML 是机器学习库，包含一些基于统计的分类和聚类工具。HighGUI 包含图像和视频输入/输出的函数。CXCore 包含 OpenCV 的一些基本数据结构和相关函数。

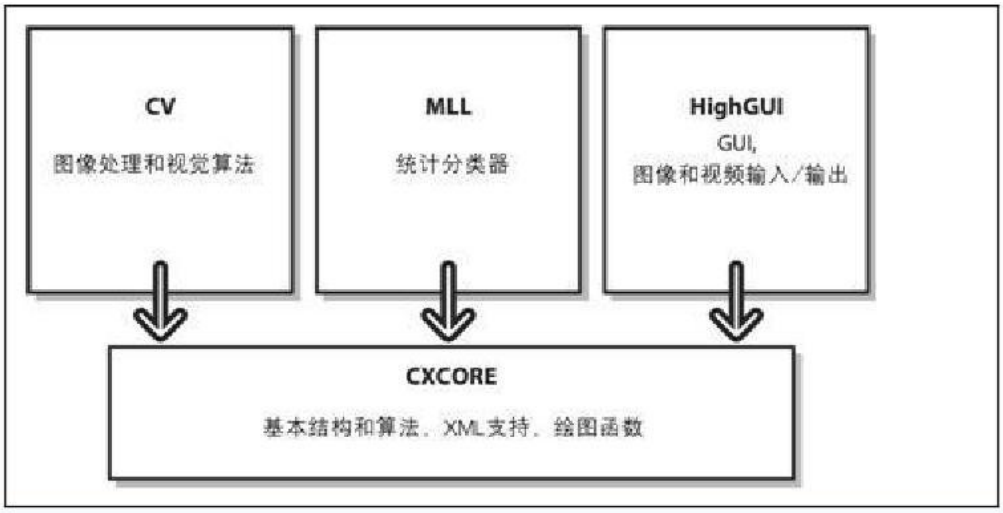

图1-5：OpenCV的基本结构

图 1-5 中并没有包含 CvAux 模块，该模块中一般存放一些即将被淘汰的算法和函数(如基于嵌入式隐马尔可夫模型的人脸识别算法)，同时还有一些新出现的实验性的算法和函数(如背景和前景的分割)。CvAux 在 Wiki 中并没有很完整的文档，而在.../opencv/docs 子目录下的 CvAux 文档也不是很完整。CvAux 包含以下一些 内容。

- 特征物体，它是一个模式识别领域里用于降低计算量的方法，本质上，依然是模板匹配。
- 一维和二维隐马尔可夫模型(HMM)，它是一个基于统计的识别方法，用动态规划来求解。
- 嵌入式 HMM(一个父 HMM 的观测量本身也符合 HMM)
- 通过立体视觉来实现的动作识别
- Delaunay 三角划分、序列等方法的扩展
- 立体视觉
- 基于轮廓线的形状匹配
- 纹理描述
- 眼睛和嘴跟踪
- 3D 跟踪
- 寻找场景中的物体的骨架(中心线)
- 通过两个不同视角的图像合成中间图像
- 前景/背景分割
- 视频监控(请参考 Wiki 的 FAQ 获得更多资料)

l 摄像机标定的C++类(C 函数和引擎已经在CV 模块中)

未来一些特性可能被合并到 CV 模块，还有一些可能永远留在 CvAux 中。

【13～14】

移植性

OpenCV 被设计为可移植的库。它的代码可以用 Borland C++, MS VC++, Intel 等编译器编译。为了使得跨平台更容易实现，C/C++代码必须按照通用的标准来编写。图 1-6 显示了目前已知的可以运行 OpenCV 在各种系统平台。基于 32 位 Intel 架构(IA32)的 Windows 系统支持最好，然后是 IA32 架构的 Linux 平台。对于 Mac OS X 平台的支持，只有在 Apple 采用 Intel 处理器后才提上议程。(在 OS X 平台上的移植目前还不像 Windows 和 Linux 平台上一样成熟，但是已在快速完善中。)成熟度次之的是在扩展内存上 64 位(EM64T)和 64 位 Intel 架构(IA64)。最不成熟的是 Sun 的硬件和其他操作系统。

	IA32	EM64T	IA64	Other (PPC, Sparc)
Windows	✓ (w. IPP; MSVC6, .NET2005+OMP, ICC, GCC, BCC)	✓ (w. IPP; MSVC6+PSDK.NET2005+OMP, PSDK)	± (w. IPP; PSDK, some tests fail)	N/A
Linux	✓ (w. IPP; GCC, BCC)	✓ (w. IPP; GCC, BCC)	✓ (GCC, ICC)	✗
MacOSX	✓ (w. IPP, GCC, native APIs)	? (not tested)	N/A	✓ (iMac G5, GCC, native APIs)
Others (BSD, Solaris...)	✗	✗	✗	Reported to build on UltraSparc Solaris

图1-6：OpenCV 1.0移植指南

如果某个 CPU 架构或操作系统没有出现在图 1-6 中，并不意味着在那上面不能使用 OpenCV。OpenCV 几乎可用于所有的商业系统，从 PowerPC Mac 到机器狗。OpenCV 同样可以很好的运行在 AMD 处理器上，IPP 也会采用 AMD 处理器里的多媒体扩展技术(MMX 等)技术进行加

速。 14～

15】

练习

1. 下载并安装最新的 OpenCV 版本，然后分别在 debug 和 release 模式下编译 OpenCV。

2. 通过 SVN 下载OpenCV的最新代码，然后编译。

3. 在三维信息转换为二维表示时，存在一些有歧义的描述，请描述至少三个歧义描述，并提供克服这些问题的方法。

第 2 章 OpenCV 入门

开始准备

安装完 OpenCV 开发包后，我们的首要任务自然是立即用它来做一些有趣的事情。为此，还需要搭建编程环境。

在 Visual Studio 开发环境中，我们需要先创建一个项目(project)，然后配置好它的各项设置，以使 OpenCV 开发包中的 highgui.lib, cxcore.lib, ml.lib 和 cv.lib 库能被正确链接[⑦]，并保证编译器的预处理器能搜索到“OpenCV „ /opencv/*/include”目录下的各个头文件。OpenCV 开发包中有许多"include"目录，它们一般都位于“C:/program files/opencv/cv/include”[⑧], “„ /opencv/cxcore/include”, “„ /opencv/ml/include”和“„ /opencv/otherlibs/ highgui”路径下。完成这些工作后，便可新建一个 C 程序文件并编写你的第一个程序。

注意： 有一些重要的头文件可以使工作变得更轻松。在头文件“…/opencv/ cxcore/include/cxtypes.h”和“cxmisc.h”中，包含许多有用的宏定义。使用这些宏，仅用一行程序便可完成结构体和数组的初始化、对链表进行排序等工作。在编译时，有几个头文件非常重要，它们分别是：机器视觉中所要用到的“…/cv/include/cv.h”和“…/cxcore/include/cxcore.h”；I/O 操作中所要用到的“…/otherlibs/highgui/highgui.h”；机器学习中所要用到的“…/ml/include/ml.h”。

初试牛刀—— 显示图像

OpenCV 开发包提供了读取各种类型的图像文件、视频内容以及摄像机输入的功能。这些功能是 OpenCV 开发包中所包含的 HighGUI 工具集的一部分。我们将使用其中的一些功能编写一段简单的程序，用以读取并在屏幕上显示一张图像。如 例 2-1 所示。

例 2-1：用于从磁盘加载并在屏幕上显示一幅图像的简单 OpenCV 程序

```
#include "highgui.h"
```

```
int main( int  argc,  char**  argv )  {
    IplImage* img  =  cvLoadImage( argv[1]  );
    cvNamedWindow( "Example1",  CV_WINDOW_AUTOSIZE  );
    cvShowImage(  "Example1", img  );
    cvWaitKey(0);
    cvReleaseImage(  &img  );
    cvDestroyWindow( "Example1"  );
}
```

当以上程序编译后，我们就可以在命令行模式下通过输入一个参数执行它。执行时，该程序将向内存加载一幅图像，并将该图像显示于屏幕上，直至按下键盘的任意一个键后它才关闭窗口并退出程序。下面我们将对以上代码做逐行分析，并仔细讲解每个函数的用法以帮助读者理解这段程序。

```
IplImage*  img  =  cvLoadImage(  argv[1] );
```

该行程序的功能是将图像文件[⑨]加载至内存。cvLoadImage()函数是一个高层调用接口，它通过文件名确定被加载文件的格式；并且该函数将自动分配图像数据结构所需的内存。需要指出的是，cvLoadImage()函数可读取绝大多数格式类型的图像文件，这些类型包括 BMP, DIB, JPEG, JPE, PNG, PBM, PGM, PPM, SR, RAS 和 TIFF。该函数执行完后将返回一个指针，此指针指向一块为描述该图像文件的数据结构(IplImage)而分配的内存块。IplImage 结构体将是我们在使用 OpenCV时会最常用到的数据结构。OpenCV 使用IplImage 结构体处理诸如单通道(single-channel)、多通道(multichannel)、整型的(integer-valued)、浮点型的(floating-point-valued)等所有类型的图像文件。

```
cvNamedWindow(  "Example1", CV_WINDOW_AUTOSIZE  );
```

cvNamedWindow()函数也是一个高层调用接口，该函数由 HighGUI 库提供。cvNamedWindow()函数用于在屏幕上创建一个窗口，将被显示的图像包含于该窗口中。函数的第一个参数指定了该窗口的窗口标题(本例中为"Example1")，如果要使用 HighGUI 库所提供的其他函数与该窗口进行交互时，我们将通过该参数值引用这个窗口。

cvNamedWindow()函数的第二个参数定义了窗口的属性。该参数可被设置为 0(默认值)或 CV_WINDOW_AUTOSIZE，设置为 0 时，窗口的大小不会因图像的大小而改变，图像只能在窗口中根据窗口的大小进行拉伸或缩放；而设置为 CV_WINDOW_AUTOSIZE 时，窗口则会根据图像的实际大小自动进行拉伸或缩放，以容纳图像。

```
cvShowImage( "Example1", img );
```

只要有一个与某个图像文件相对应的 IplImage*类型的指针，我们就可以在一个已创建好的窗口(使用 cvNamedWindow()函数创建)中使用 cvShowImage()函数显示该图像。cvShowImage()函数通过设置其第一个参数确定在哪个已存在的窗口中显示图像。cvShowImage()函数被调用时，该窗口将被重新绘制，并且图像也会显示在窗口中。如果该窗口在创建时被指定 CV_WINDOW_AUTOSIZE 标志作为 cvNamedWindow()函数的第二个参数，该窗口将根据图像的大小自动调整为与图像一致。

```
cvWaitKey(0);                                    【17～18】
```

cvWaitKey()函数的功能是使程序暂停，等待用户触发一个按键操作。但如果将该函数参数设为一个正数，则程序将暂停一段时间，时间长为该整数值个毫秒单位，然后继续执行程序，即使用户没有按下任何键。当设置该函数参数为 0 或负数时，程序将一直等待用户触发按键操作。

```
cvReleaseImage( &img );
```

一旦用完加载到内存的图像文件，我们就可以释放为该图像文件所分配的内存。我们通过为 cvReleaseImage()函数传递一个类型为 IplImage*的指针参数调用该函数，用以执行内存释放操作。对 cvReleaseImage()函数的调用执行完毕后，img 指针将被设置为 NULL。

```
cvDestroyWindow( "Example1" );
```

最后，可以销毁显示图像文件的窗口。cvDestroyWindow()函数将关闭窗口，并同时释放为该窗口所分配的所有内存(包括窗口内部的图像内存缓冲区，该缓冲区中保存了与 img 指针相关的图像文件像素信息的一个副本)。因为当应用程序的窗口被关闭时，该应用程序窗口所占用的一切资源都会由操作系统自动释放，所以对一些简单程序，不必调用 cvDestroyWindow()或 cvReleaseImage()函数显

式释放资源。但是，养成习惯每次都调用这些函数显式释放资源总是有好处的。

尽管现在已经有了这个可供我们随意把玩的简单程序，但我们并不能就此停滞不前。我们的下一个任务将要去创建一个几乎与例 2-1 一样非常简单的程序——编写程序读取并播放 AVI 视频文件。完成这个新任务后，需要开始深入思考一些问题了。

第二个程序—— 播放 **AVI** 视频

使用 OpenCV 播放视频，几乎与使用它来显示图像一样容易。播放视频时只需要处理的新问题就是如何循环地顺序读取视频中的每一帧，以及如何从枯燥的电影视频的读取中退出该循环操作。具体如例 2-2 所示。

例 2-2：一个简单的 OpenCV 程序，用于播放硬盘中的视频文件

```
#include "highgui.h"

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

    cvNamedWindow( "Example2", CV_WINDOW_AUTOSIZE );

    CvCapture* capture = cvCreateFileCapture( argv[1] );

    IplImage* frame;

    while(1) {

        frame = cvQueryFrame( capture );

        if( !frame ) break;

        cvShowImage( "Example2", frame );

        char c = cvWaitKey(33);

        if( c == 27 ) break;

    }

    cvReleaseCapture( &capture );

    cvDestroyWindow( "Example2" );

}
```


这里我们还是通过前面的方法创建一个命名窗口，在“例 2”中，事情变得更加有趣了。

```
CvCapture*  capture  = cvCreateFileCapture( argv[1] );
```

函数 cvCreateFileCapture()通过参数设置确定要读入的 AVI 文件，返回一个指向 CvCapture 结构的指针。这个结构包括了所有关于要读入 AVI 文件的信息，其中包含状态信息。在调用这个函数后，返回指针所指向的 CvCapture 结构被初始化到所对应 AVI 文件的开头。

```
frame  = cvQueryFrame(  capture  );
```

一旦进入 while(1)循环，我们便开始读入 AVI 文件，cvQueryFrame 的参数为 CvCapture 结构的指针。用来将下一帧视频文件载入内存(实际是填充或更新 CvCapture 结构中)。返回一个对应当前帧的指针。与 cvLoadImage 不同的是，cvLoadImage 为图像分配内存空间，而 cvQueryFrame 使用已经在 cvCapture 结构中分配好的内存。这样的话，就没有必要通过 cvReleaseImage()对这个返回的图像指针进行释放，当 CvCapture 结构被释放后，每一帧图像所对应的内存空间即会被释放。

```
c =  cvWaitKey(33);

if(  c  == 27  )  break;
```

当前帧被显示后，我们会等待 33 ms。[⑩][11] 如果其间用户触发了一个按键，c 会被设置成这个按键的 ASCII 码，否则，c 会被设置成-1。如果用户触发了 ESC 键(ASCII 27)，循环被退出，读入视频停止。否则 33 ms 以后继续执行循环。需要指出的是，在这个简单的例子程序中，我们没有使用任何准确的方法来控制视频帧率。我们只是简单的通过 cvWaitKey 来以固定时间间隔载入帧图像，在一个精度要求更高的程序中，通过从 CvCapture 结构体中读取实际帧率是一个更好的方法！

```
cvReleaseCapture(  &capture  );
```

退出循环体(视频文件已经读入结束或者用户触发了 Esc 键)后，我们应该释放为 CvCapture 结构开辟的内存空间，这同时也会关闭所有打开的 AVI 文件相关的文件句柄。

视频播放控制

完成了前面的程序以后，现在我们可对其加以改进，进一步探索更多可用的功能。首先会注意到，例 2-2 所实现的 AVI 播放器无法在视频播放时进行快速拖动。我们的下一个任务就是通过加入一个滚动条来实现这个功

能。 【19】

HighGUI 工具包不仅提供了我们刚刚使用的一些简单的显示函数，还包括一些图像和视频控制方法。其中一个经常使用的就是滚动条，滚动条可以使我们方便地从视频的一帧跳到另外一帧。我们通过调用 cvCreateTrackbar()来创建一个滚动条，并且通过设置参数确定滚动条所属于的窗口。为了获得所需的功能，只需要提供一个回调函数。具体如例 2-3 所示。

例 2-3：用来添加滚动条到基本浏览窗口的程序：拖动滚动条，函数 onTrackSlide()便被调用并被传入滚动条新的状态值

```
#include "cv.h"
#include "highgui.h"
int g_slider_position = 0;
CvCapture* g_capture = NULL;
void onTrackbarSlide(int pos) {
    cvSetCaptureProperty(
        g_capture,
        CV_CAP_PROP_POS_FRAMES,
        pos
    );
}
int main( int argc, char** argv ) {
    cvNamedWindow( "Example3", CV_WINDOW_AUTOSIZE );
    g_capture = cvCreateFileCapture( argv[1] );
    int frames = (int) cvGetCaptureProperty(
        g_capture,
        CV_CAP_PROP_FRAME_COUNT
    );
```

```
    if( frames!= 0 ) {
     cvCreateTrackbar(
          "Position",
          "Example3",
          &g_slider_position,
          frames,
          onTrackbarSlide
     );
    }
    IplImage* frame;
    // While loop (as in Example 2) capture & show video
    ...
    // Release memory and destroy window
    ...
    return(0);
}
```

从本质上说，这种方法是通过添加一个全局变量来表示滚动条位置并且添加一个回调函数更新变量以及重新设置视频读入位置。我们通过一个调用来创建滚动条和确定回调函数。下面让我们看看细节。

```
int g_slider_position = 0;
CvCapture* g_capture = NULL;                    【20～21】
```

首先为滚动条位置定义一个全局变量。由于回调函数需要使用 CvCapture 对象，因此我们将它定义为全局变量。为了使我们的程序可读性更强，我们在所有全局变量名称前面加上 g_。

```
void onTrackbarSlide(int pos) {
    cvSetCaptureProperty(
    g_capture,
    CV_CAP_PROP_POS_FRAMES,
```

```
    pos
);
```

现在我们定义一个回调函数，使其在滚动条被拖动时调用。滚动条的位置会被作为一个 32 位整数以参数形式传入。

后面我们会常常看到函数 cvSetCaptureProperty()被调用，同时与之配套的函数 cvGetCaptureProperty()也经常会被调用。这些函数允许我们设置(或查询)CvCapture 对象的各种属性。在本程序中我们设置参数 CV_CAP_PROP_POS_ FRAMES(这个参数表示我们以帧数来设置读入位置，如果我们想通过视频长度比例来设置读入位置，我们可以通过用 AVI_RATIO 代替 FRAMES 来实现)。最后，我们把新的滚动条位置作为参数传入。因为 HighGUI 是高度智能化的，它会自动处理一些问题，比如滚动条对应位置不是关键帧，它会从前面一个关键帧开始运行并且快进到对应帧，而不需要我们来处理这些细节问题。

```
int frames = (int) cvGetCaptureProperty(
    g_capture,
    CV_CAP_PROP_FRAME_COUNT
);
```

正如前面所说，当需要从 CvCapture 结构查询数据时，可使用 cvGetCaptureProperty 函数。在本程序中，我们希望获得视频文件的总帧数以对滚动条进行设置(具体实现在后面)。

```
if( frames!= 0 ) {
  cvCreateTrackbar(
      "Position",
      "Example3",
      &g_slider_position,
      frames,
      onTrackbarSlide
  );
}
```

前面的代码用来创建滚动条，借助函数 cvCreateTrackbar()，我们可设置滚动条的名称并确定滚动条的所属窗口。我们将一个变量绑定到这个滚动条来表示滚动条的最大值和一个回调函数(不需要回调函数时置为空，当滚动条被拖动时触发)。仔细分析，你会发现一点：cvGetCaptureProperty()返回的帧数为 0 时，滚动条不会被创建。这是因为对于有些编码方式，总的帧数获取不到，在这种情况下，我们只能直接播放视频文件而看不到滚动条。

值得注意的是，通过 HighGUI 创建的滚动条不像其他工具提供的滚动条功能这么全面。当然，也可以使用自己喜欢的其他窗口开发工具包来代替 HighGUI，但是 HighGUI 可以较快地实现一些基本功能。

最后，我们并没有将实现滚动条随着视频播放移动功能的代码包含进来，这可作为一个练习留给读者。

一个简单的变换

很好，现在你已经可以使用 OpenCV 创建自己的视频播放器了，但是我们所关心的是计算机视觉，所以下面会讨论一些计算机视觉方面的工作。很多基本的视觉任务包括对视频流的滤波。我们将通过修改前面程序，实现随着视频的播放而对其中每一帧进行一些简单的运算。

一个简单的变化就是对图像平滑处理，通过对图像数据与高斯或者其他核函数进行卷积有效的减少图像信息内容。OpenCV 使得这个卷积操作非常容易。我们首先创建一个窗口“Example4-out”用来显示处理后的图像。然后，在我们调用 cvShowImage()来显示新捕捉的图像以后，我们可以计算和在输出窗口中显示平滑处理后的图像。具体如例 2-4 所示。

例 2-4：载入一幅图像并进行平滑处理

```
#include "cv.h"
#include "highgui.h"
void example2_4( IplImage* image )
  // Create some windows to show the input
  // and output images in.
  //
```

```
    cvNamedWindow( "Example4-in" );
    cvNamedWindow( "Example4-out" );
    // Create a window to show our input image
    //
    cvShowImage( "Example4-in", image );
    // Create an image to hold the smoothed output
    //
    IplImage* out = cvCreateImage(
        cvGetSize(image),
        IPL_DEPTH_8U,
        3
    );
    // Do the smoothing
    //
    cvSmooth( image, out, CV_GAUSSIAN, 3, 3 );
    // Show the smoothed image in the output window
    //
    cvShowImage( "Example4-out", out );
    // Be tidy
    //
    cvReleaseImage( &out );
    // Wait for the user to hit a key, then clean up the windows
    //
    cvWaitKey( 0 );
    cvDestroyWindow( "Example4-in" );
    cvDestroyWindow( "Example4-out" );
}
```

【22～23】

第一个 cvShowImage()调用跟前面的例子中没有什么不同。在下面的调用，我们为另一个图像结构分配空间。前面依靠 cvCreateFileCapture()来为新的帧分配空间。事实上，cvCreateFileCapture()只分配一帧图像的空间，每次调用时覆盖前面一次的数据(这样每次调用返回的指针是一样的)。在这种情况下，我们想分配自己的图像结构空间用来存储平滑处理后的图像。第一个参数是一个 CvSize 结构，这个结构可以通过 cvGetSize(image)方便地获得；第一个参数说明了当前图像结构的大小。第二个参数告诉了我们各通道每个像素点的数据类型，最后一个参数说明了通道的总数。所以从程序中可以看出，当前图像是 3 个通道(每个通道 8 位)，图像大小同 image。

平滑处理实际上只是对 OpenCV 库函数的一个调用：我们指定输入图像，输出图像，光滑操作的方法以及平滑处理的一些参数。在本程序中，我们通过使用每个像素周围 3*3 区域进行高斯平滑处理。事实上，输入、输出图像可以是相同的，这将会使我们的程序更加有效，不过我们为了介绍 cvCreateImage()函数而没有这样使用。

现在我们可以在我们新窗口中显示处理后的图像然后释放它：cvReleaseImage()通过给定一个指向 IplImage*的指针来释放与图像对应的内存空间。 【23～24】

一个复杂一点的变换

前面的工作很棒，我们学会了做一些更有趣的事情。在例 2-4 中，我们选择了重新创建一个 IplImage 结构，并且在新建的结构中写入了一个单个变换的结果。

正如前面所提到的，我们可以以这样一种方式来应用某个变换，即用输出来覆盖输入变量，但这并非总是行得通的。具体说来，有些操作输出的图像与输入图像相比，大小、深度、通道数目都不一样。通常，我们希望对一些原始图像进行一系列操作并且产生一系列变换后的图像。

在这种情况下，介绍一些封装好的函数是很有用的，这些函数既包含输出图像内存空间的分配，同时也进行了一些我们感兴趣的变换。例如，可以考虑对图像进行缩放比例为 2 的缩放处理[Rosenfeld80]。在 OpenCV 中，我们通过函数 cvPyr

Down()来完成上述功能。这在很多重要的视觉算法中都很有用。我们在例 2-5 中执行这个函数。

例 2-5：使用 cvPyrDown() 创建一幅宽度和高度为输入图像一半尺寸的图像

```
IplImage* doPyrDown(
  IplImage*  in,
  int filter  = IPL_GAUSSIAN_5x5
) {
   // Best  to make  sure input  image is divisible  by two.
   //
   assert( in->width%2 ==  0 &&  in->height%2  ==  0  );
   IplImage* out  =  cvCreateImage(
       cvSize(  in->width/2,  in->height/2 ),
       in->depth,
       in->nChannels
   );
   cvPyrDown( in,  out  );
   return( out  );
};
```

细心的读者会发现，我们分配新的图像空间时是从旧的图像中读取所需的信息。在 OpenCV 中，所有的重要数据结构都是以结构体的形式实现，并且以结构体指针的形式传递。OpenCV 中没有私有数据！现在让我们来看一个与前类似但已加入 Canny 边界检测的例子[Canny86](见例 2-6)。在这个程序中，边缘检测器产生了一个与输入图像大小相同但只有一个通道的图像。 【24～25】

例 2-6：Canny 边缘检测将输出写入一个单通道(灰度级)图像

```
IplImage* doCanny(
   IplImage*    in,
   double    lowThresh,
```

```
    double    highThresh,
    double    aperture
) {
    If(in->nChannels != 1)
        return(0); //Canny only handles gray scale images
    IplImage* out = cvCreateImage(
        cvSize( cvGetSize( in ),
        IPL_DEPTH_8U,
        1
    );
    cvCanny( in, out, lowThresh, highThresh, aperture );
    return( out );
};
```

前面的实现使得我们更加容易进行一些连续的变换。例如，如果我们想缩放图像两次，然后在缩放后的图像中寻找边缘，我们可以很简单的进行操作组合，具体如 例 2-7 所示。

例 2-7：在一个简单的图像处理流程中进行两次缩放处理与 Canny 边缘检测

```
IplImage* img1 = doPyrDown( in, IPL_GAUSSIAN_5x5 );
IplImage* img2 = doPyrDown( img1, IPL_GAUSSIAN_5x5 );
IplImage* img3 = doCanny( img2, 10, 100, 3 );
// do whatever with 'img3'
//
...
cvReleaseImage( &img1 );
cvReleaseImage( &img2 );
cvReleaseImage( &img3 );
```

注意，对前面的函数进行嵌套调用是一个很不好的主意，因为那样，内存空间的释放将会成为一个很困难的问题。如果想偷懒儿，我们可以将下面这行代码加入我们的每个封装。

```
cvReleaseImage( &in );
```

这种“自清理”模式会很干净，但是它同样有一些缺点。比如，如果想对其中一个中间步骤的图像进行处理，便无法找到此图像的入口。为了解决前面的问题，代码可以简化(如例 2-8 所
示)。

例 2-8：通过每个独立阶段释放内存来简化例 2-7 中图像处理流程

```
IplImage* out;
out = doPyrDown( in, IPL_GAUSSIAN_5x5 );
out = doPyrDown( out, IPL_GAUSSIAN_5x5 );
out = doCanny( out, 10, 100, 3 );
// do whatever with 'out'
//
...
cvReleaseImage ( &out );
```

对于自清理方法的最后一个提醒：在 OpenCV 中，我们必须确认被释放的空间必须是我们显式分配的。参考前面从 cvQueryFrame()返回的 IplImage*指针，这个指针指向的结构是 CvCapture 结构的一部分，而 CvCapture 在初始时就已经被初始化了而且只初始化一次。通过调用 cvReleaseImage()释放 IplImage*会产生很多难以处理的问题。前面这个例子主要是想说明，虽然内存垃圾处理在 OpenCV 中很重要，但我们只需要释放自己显式分配的内存空间。

从摄像机读入数据

在计算机世界里，“视觉”这个词的含义非常丰富。在一些情况下，我们要分析从其他地方载入的固定图像。在另外一些情况下，我们要分析从磁盘中读入的视频文件。在更多的情况下，我们想处理从某些摄像设备中实时读入的视频流。

OpenCV—— 更确切地说，OpenCV 中的 HighGUI 模块—— 为我们提供了一种简单的方式来处理这种情况。这种方法类似于读取 AVI 文件，不同的是，我们调用的是 cvCreateCameraCapture()，而不是 cvCreateFileCapture()。后面一个函数参数为摄像设备 ID 而不是文件名。当然，只有存在多个摄像设备时这个参数才起作用。默认值为-1，代表“随机选择一个”；自然，它更适合当有且仅有一个摄像设备的情况(详细内容参考第 4 章)。

函数 cvCreateCameraCapture()同样返回相同的 CvCapture*指针，这使得我们后面可以使用完全类似于从视频流中获取帧的方法。当然，HighGUI 做很多工作才使得摄像机图像序列看起来像一个视频文件，但是我们不需要考虑那些问题。当我们需要处理摄像机图像序列时我们只需要简单地从摄像机获得图像，像视频文件一样处理。为了便于开发工作，大多程序实时处理的程序同样会有视频文件读入模式，CvCapture 结构的通用性使得这非常容易实现。具体如例 2-9 所示。

【26】

例 2-9：capture 结构初始化后，从视频文件或摄像设备读入图像没有区别

```
CvCapture*  capture;
if(  argc==1  )  {
    capture  = cvCreateCameraCapture(0);
}  else  {
    capture  = cvCreateFileCapture(  argv[1]  );
}
assert(  capture  !=  NULL );
// Rest  of  program  proceeds  totally  ignorant
...
```

由此可见，这种处理是很理想的。

写入 **AVI** 视频文件

在很多程序中，我们想将输入视频流或者捕获的图像序列记录到输出视频流中，OpenCV 提供了一个简洁的方法来实现这个功能。就像可以创建一个捕获设备以

便每次都能从视频流中提取一帧一样，我们同样可以创建一个写入设备以便逐帧将视频流写入视频文件。实现这个功能的函数是 cvCreateVideoWriter()。

当输出设备被创建以后，我们可以通过调用 cvWriteFrame()逐帧将视频流写入文件。写入结束后，我们调用 cvReleaseVideoWriter()来释放资源。例 2-10 描述了一个简单的程序。这个程序首先打开一个视频文件，读取文件内容，将每一帧图像转换为对数极坐标格式(就像你的眼睛真正看到的，在第 6 章会有详细描述)，最后将转化后的图像序列写入新的视频文件中。

例 2-10：一个完整的程序用来实现读入一个彩色视频文件并以灰度格式输出这个视频文件

```
// Convert a video to grayscale
// argv[1]: input video file
// argv[2]: name of new output file
//
#include "cv.h"
#include "highgui.h"
main( int argc, char* argv[] ) {
    CvCapture* capture = 0;
    capture = cvCreateFileCapture( argv[1] );
    if(!capture){
        return -1;
    }
    IplImage *bgr_frame=cvQueryFrame(capture);//Init the video read
    double fps = cvGetCaptureProperty (
        capture,
        CV_CAP_PROP_FPS
    );
    CvSize size = cvSize(
        (int)cvGetCaptureProperty( capture, CV_CAP_PROP_FRAME_WIDTH),
        (int)cvGetCaptureProperty( capture, CV_CAP_PROP_FRAME_HEIGHT)
```

```
    );
    CvVideoWriter *writer = cvCreateVideoWriter(
        argv[2],
        CV_FOURCC('M','J','P','G'),
        fps,
        size
    );
    IplImage* logpolar_frame = cvCreateImage(
        size,
        IPL_DEPTH_8U,
        3
    );
    while( (bgr_frame=cvQueryFrame(capture)) != NULL ) {
        cvLogPolar( bgr_frame, logpolar_frame,
                cvPoint2D32f(bgr_frame->width/2,
                bgr_frame->height/2),
                40,
                CV_INTER_LINEAR+CV_WARP_FILL_OUTLIERS );
        cvWriteFrame( writer, logpolar_frame );
    }
    cvReleaseVideoWriter( &writer );
    cvReleaseImage( &logpolar_frame );
    cvReleaseCapture( &capture );
    return(0);
}
```

【27～28】

仔细阅读这个程序，会发现它使用了一些我们很熟悉的函数。首先，打开一个视频文件；通过 cvQueryFrame()函数读入视频；然后，我们使用 cvGetCapture-Property()来获得视频流的各种重要属性。打开一个视频文件进行此操作，并将

各帧图像转换为对数极坐标格式，将转换后的图像逐帧写入视频文件，直到读入结束。最后释放各种资源，程序结束。

在对函数 cvCreateVideoWriter()进行调用时，有几个参数是我们必须了解的。第一个参数是用来指定新建视频文件的名称。第二个参数是视频压缩的编码格式。目前有很多流行的编解码器格式，但是无论采用哪种格式，都必须确保自己的电脑中有这种编解码器(编解码器的安装独立于 OpenCV)。在本程序中，我们选用比较流行的 MJPG 编码格式；OpenCV 用宏 CV_FOURCC()来指定编码格式，CV_FOURCC()包含 4 个字符参数。这 4 个字符构成了编解码器的“4 字标记”，每个编解码器都有一个这样的标记，Motion JPEG 编码格式的四字标记就是 MJPG，所以如此指定宏的 4 个字符参数 CV_FOURCC('M','J','P','G')。后面两个参数用来指定播放的帧率和视频图像的大小。在本程序中，我们将它们设置成原始视频文件(彩色视频文件)的帧率和图像的大小。

小结

在开始学习第 3 章前，我们需要先总结一下前面的知识，并展望后面要讲的东西。我们已经知道，OpenCV 为程序开发提供了很多简便易用的接口，可以用来载入图像、从磁盘或者从摄像设备中捕获视频。我们也知道，OpenCV 库中包含很多基本的函数用来处理这些图像。但是更多更强大的功能我们还没有看到。它可以对抽象数据类型进行更多更复杂的处理，这些对解决实际的视觉问题有很大的帮助。

在后面的几章里，我们会更深入地了解 OpenCV 的一些基础结构并对界面相关函数和图像数据类型做更详细的分析。我们会系统地讲解一些基本的图像处理操作，并在后面讲解一些高级的图像变换。最后，我们会探索更多专业的应用，如摄像机标定、跟踪、识别。OpenCV 为这些应用提供了许多 API。准备好了吗？让我们开始

吧！

【29】

练习

如果还没有在本机上安装 OpenCV，请先下载并安装。浏览目录，在 docs 目录中，可以找到 index.htm 文件，它链接到库文件的一些主要文档。下面我们进一步了解一些库文件，Cvcore 包含一些基本数据结构与算法，cv 包含图像处理和视觉的一些算法，ml 由一些机器学习和聚类算法组成，otherlics/highgui 包含一些 输入/输出函数。_make 文件夹下包含 OpenCV 的工程文件，程序实例的代码在 samples 文件夹里。

1. 进入目录…/opencv/_make。在 Windows 环境下，打开解决方案 opencv.sln，在 Linux 环境下，打开对应的 makefile，分别建立 debug 和 release 版的库文件。这会花一些时间，但是后面会用到这些库文件和动态链接库。

2. 进入目录…/opencv/samples/c/。建立一个工程(project)，导入并编译 lkdemo.c(这是一个运动跟踪程序实例)。安装上摄像机以后运行程序，选中运行窗口，键入“r”进行跟踪初始化，也可以在视频位置上单击鼠标以添加跟踪点。还可以通过键入“n”切换为只观察点(而非图像)。再次键入“n”可在“夜间”和“白天”这两个视图之间进行切换。

3. 使用例 2-10 中的视频捕捉和存储方法，结合例 2-5 中的 doPyrDown()创建一个程序，使其从摄像机读入视频数据并将缩放变换后的彩色图像存入磁盘。

4. 修改练习 3 的代码，结合例 2-1，将变换结果显示在窗口中。

5. 对练习 4 中的代码进行修改，参考例 2-3，给程序加入滚动条，使得用户可以动态调节缩放比例，缩放比例的取值为 2～8 之间。可以跳过写入磁盘操作，但是必须将变换结果显示在窗口中。

第 3 章　初探 OpenCV

OpenCV 的基本数据类型

OpenCV 提供了多种基本数据类型。虽然这些数据类型在 C 语言中不是基本类型，但结构都很简单，可将它们作为原子类型。可以在“„ /OpenCV/cxcore/include”目录下的 cxtypes.h 文件中查看其详细定义。

在这些数据类型中最简单的就是 CvPoint。CvPoint 是一个包含 integer 类型成员 x 和 y 的简单结构体。CvPoint 有两个变体类型：CvPoint2D32f 和 CvPoint3D32f。前者同样有两个成员 x，y，但它们是浮点类型；而后者却多了一个浮点类型的成员 z。

CvSize 类型与 CvPoint 非常相似，但它的数据成员是 integer 类型的 width 和 height。如果希望使用浮点类型，则选用 CvSize 的变体类型 CvSize2D32f。

CvRect 类型派生于 CvPoint 和 CvSize，它包含 4 个数据成员：x，y，width 和 height。(正如你所想的那样，该类型是一个复合类型)。

下一个(但不是最后一个)是包含 4 个整型成员的 CvScalar 类型，当内存不是问题时，CvScalar 经常用来代替 1，2 或者 3 个实数成员(在这个情况下，不需要的分量被忽略)。CvScalar 有一个单独的成员 val，它是一个指向 4 个双精度浮点数数组的指针。

所有这些数据类型具有以其名称来定义的构造函数，例如 cvSize()。(构造函数通常具有与结构类型一样的名称，只是首字母不大写)。记住，这是 C 而不是 C++，所以这些构造函数只是内联函数，它们首先提取参数列表，然后返回被赋予相关值的结

构。

【31】

各数据类型的内联构造函数被列在表 3-1 中：cvPointXXX()，cvSize()，cvRect()和 cvScalar()。这些结构都十分有用，因为它们不仅使代码更容易编写，而且也更易于阅读。假设要在(5，10)和(20，30)之间画一个白色矩形，只需简单调用：

```
cvRectangle(
  myImg,
  cvPoint(5,10),
  cvPoint(20,30),
  cvScalar(255,255,255)
);
```

表 3-1：points, size, rectangles 和 calar 三元组的结构

结构	成员	意义
CvPoint	int x, y	图像中的点
CvPoint2D32f	float x, y	二维空间中的点
CvPoint3D32f	float x, y, z	三维空间中的点
CvSize	int width, height	图像的尺寸
CvRect	int x, y, width, height	图像的部分区域
CvScalar	double val[4]	RGBA 值

cvScalar 是一个特殊的例子：它有 3 个构造函数。第一个是 cvScalar()，它需要一个、两个、三个或者四个参数并将这些参数传递给数组 val[]中的相应元素。第二个构造函数是 cvRealScalar()，它需要一个参数，它被传递给给 val[0]，而 val[]数组别的值被赋为 0。最后一个有所变化的是 cvScalarAll()，它需要一个参数并且 val[]中的 4 个元素都会设置为这个参数。

矩阵和图像类型

图 3-1 为我们展示了三种图像的类或结构层次结构。使用 OpenCV 时，会频繁遇到 IplImage 数据类型，第 2 章已经出现多次。IplImage 是我们用来为通常所说的“图像”进行编码的基本结构。这些图像可能是灰度，彩色，4 通道的(RGB+alpha)，其中每个通道可以包含任意的整数或浮点数。因此，该类型比常见的、易于理解的 3 通道 8 位 RGB 图像更通用。

OpenCV 提供了大量实用的图像操作符，包括缩放图像，单通道提取，找出特定通道最大最小值，两个图像求和，对图像进行阈值操作，等等。本章我们将仔细介绍这类操

作。

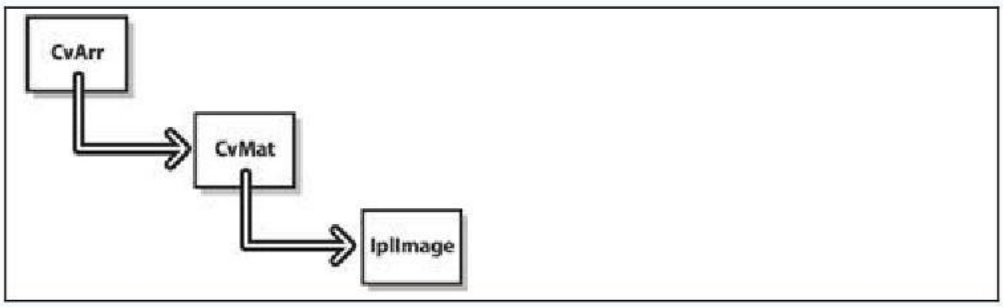

图3-1：虽然OpenCV是由C 语言实现的，但它使用的结构体也是遵循面向对象的思想设计的。实际上，IplImage由CvMat 派生，而CvMat 由CvArr 派生

在开始探讨图像细节之前，我们需要先了解另一种数据类型 CvMat，OpenCV 的矩阵结构。虽然 OpenCV 完全由 C 语言实现，但 CvMat 和 IplImage 之间的关系就如同 C++中的继承关系。实质上，IplImage 可以被视为从 CvMat 中派生的。因此，在试图了解复杂的派生类之前，最好先了解基本的类。第三个类 CvArr，可以被

视为一个抽象基类，CvMat 由它派生。在函数原型中，会经常看到 CvArr(更准确地说，CvArr*)，当它出现时，便可以将 CvMat*或 IplImage*传递到程序。

CvMat 矩阵结构

在开始学习矩阵的相关内容之前，我们需要知道两件事情。第一，在 OpenCV 中没有向量(vector)结构。任何时候需要向量，都只需要一个列矩阵(如果需要一个转置或者共轭向量，则需要一个行矩阵)。第二，OpenCV 矩阵的概念与我们在线性代数课上学习的概念相比，更抽象，尤其是矩阵的元素，并非只能取简单的数值类型。例如，一个用于新建一个二维矩阵的例程具有以下原型：

```
cvMat*  cvCreateMat  (  int  rows,  int cols,  int  type  );
```

这里 type 可以是任何预定义类型，预定义类型的结构如下：CV_<bit_depth> (S|U|F)C<number_of_channels>。于是，矩阵的元素可以是 32 位浮点型数据(CV_32FC1)，或者是无符号的 8 位三元组的整型数据(CV_8UC3)，或者是无数的其他类型的元素。一个 CvMat 的元素不一定就是个单一的数字。在矩阵中可以通过单一(简单)的输入来表示多值，这样我们可以在一个三原色图像上描绘多重色彩通道。对于一个包含 RGB 通道的简单图像，大多数的图像操作将分别应用于每一个通道(除非另有说明)。

实质上，正如例 3-1 所示，CvMat 的结构相当简单，(可以自己打开文件„ /opencv/cxcore/include/cxtypes.h 查看)。矩阵由宽度(width)、高度(height)、类型(type)、行数据长度(step，行的长度用字节表示而不是整型或者浮点型长度)和一个指向数据的指针构成(现在我们还不能讨论更多的东西)。可以通过一个指向 CvMat 的指针访问这些成员，或者对于一些普通元素，使用现成的访问方法。例如，为了获得矩阵的大小，可通过调用函数 vGetSize(CvMat*)，返回一个 CvSize 结构，便可以获取任何所需信息，或者通过独立访问高度和宽度，结构为 matrix->height 和 matrix->widt

h。

【33～34】

例 3-1：CvMat 结构：矩阵头

```
typedef struct CvMat  {
```

```
    int type;
    int step;
    int* refcount;    // for internal use only
    union {
        uchar* ptr;
        short* s;
        int*   i;
        float* fl;
        double* db;
    } data;
    union {
        int rows;
        int height;
    };
    union {
        int cols;
        int width;
};
} CvMat;
```

此类信息通常被称作矩阵头。很多程序是区分矩阵头和数据体的，后者是各个 data 成员所指向的内存位置。

矩阵有多种创建方法。最常见的方法是用 cvCreateMat()，它由多个原函数组成，如 cvCreateMatHeader()和 cvCreateData()。cvCreateMatHeader()函数创建 CvMat 结构，不为数据分配内存，而 cvCreateData()函数只负责数据的内存分配。有时，只需要函数 cvCreateMatHeader()，因为已因其他理由分配了存储空间，或因为还不准备分配存储空间。第三种方法是用函数 cvCloneMat (CvMat*)，它依据一个现有矩阵创建一个新的矩阵。当这个矩阵不再需要时，可以调用函数 c

vReleaseMat(CvMat*)释放它。

【34】

例 3-2 概述了这些函数及其密切相关的其他函数。

例3-2：矩阵的创建和释放

```
//Create a new rows by cols matrix of type 'type'.
//
CvMat* cvCreateMat( int rows, int cols, int type );
//Create only matrix header without allocating data
//
CvMat* cvCreateMatHeader( int rows, int cols, int type );
//Initialize header on existiong CvMat structure
//
CvMat* cvInitMatHeader(
  CvMat* mat,
  int    rows,
  int    cols,
  int    type,
  void*  data = NULL,
  int    step = CV_AUTOSTEP
);
//Like cvInitMatHeader() but allocates CvMat as well.
//
CvMat cvMat(
  int    rows,
  int    cols,
  int    type,
  void*  data = NULL
);
```

```
//Allocate a new matrix just like the matrix 'mat'.
//
CvMat* cvCloneMat( const cvMat* mat );
// Free the matrix 'mat', both header and data.
//
void cvReleaseMat( CvMat** mat );
```

与很多 OpenCV 结构类似，有一种构造函数叫 cvMat，它可以创建 CvMat 结构，但实际上不分配存储空间，仅创建头结构(与 cvInitMatHeader()类似)。这些方法对于存取到处散放的数据很有作用，可以将矩阵头指向这些数据，实现对这些数据的打包，并用操作矩阵的函数去处理这些数据，如例 3-3 所示。

例 3-3：用固定数据创建一个 OpenCV 矩阵

```
//Create an OpenCV Matrix containing some fixed data.
//
float vals[] = { 0.866025, -0.500000, 0.500000, 0.866025 };
CvMat rotmat;
cvInitMatHeader(
  &rotmat,
  2,
  2,
  CV_32FC1,
  vals
);
```

一旦我们创建了一个矩阵，便可用它来完成很多事情。最简单的操作就是查询数组定义和数据访问等。为查询矩阵，我们可以使用函数 cvGetElemType(const CvArr* arr)，cvGetDims(const CvArr* arr, int* sizes=NULL)和 cvGet- DimSize(const CvArr* arr，int index)。第一个返回一个整型常数，表示存储在数组里的元素类型(它可以为 CV_8UC1 和 CV_64FC4 等类型)。第二个取出数组以及一个可选择的整型指针，它返回维数(我们当前的实例是二维，但是在后面我们

将遇到的 N 维矩阵对象)。如果整型指针不为空，它将存储对应数组的高度和宽度(或者 N 维数)。最后的函数通过一个指示维数的整型数简单地返回矩阵在那个维数上矩阵的大
小。

【35～36】

矩阵数据的存取

访问矩阵中的数据有 3 种方法：简单的方法、麻烦的方法和恰当的方法。

简单的方法

从矩阵中得到一个元素的最简单的方法是利用宏 CV_MAT_ELEM()。这个宏(参见 例 3-4)传入矩阵、待提取的元素的类型、行和列数 4 个参数，返回提取出的元素 的值。

例 3-4：利用 CV_MAT_ELEM()宏存取矩阵

```
CvMat* mat = cvCreateMat( 5, 5, CV_32FC1 );
float element_3_2 = CV_MAT_ELEM( *mat, float, 3, 2 );
```

更进一步，还有一个与此宏类似的宏，叫 CV_MAT_ELEM_PTR()。CV_MAT_ELEM_ PTR()(参见例 3-5)传入矩阵、待返回元素的行和列号这 3 个参数，返回指向这个元素的指针。该宏和 CV_MAT_ELEM()宏的最重要的区别是后者在指针解引用之前将其转化成指定的类型。如果需要同时读取数据和设置数据，可以直接调用 CV_MAT_ELEM_PTR()。但在这种情况下，必须自己将指针转化成恰当
的 类型。

例 3-5：利用宏 CV_MAT_ELEM_PTR()为矩阵设置一个数值

```
CvMat* mat = cvCreateMat( 5, 5, CV_32FC1 );
float element_3_2 = 7.7;
*( (float*)CV_MAT_ELEM_PTR( *mat, 3, 2 ) ) = element_3_2;
```

【36】

遗撼的是，这些宏在每次调用的时候都重新计算指针。这意味着要查找指向矩阵基本元素数据区的指针、计算目标数据在矩阵中的相对地址，然后将相对位置与基本位置相加。所以，即使这些宏容易使用，但也不是存取矩阵的最佳方法。在计划顺序访问矩阵中的所有元素时，这种方法的缺点尤为突出。下面我们将讲述怎么运用最好的方法完成这个重要任务。

麻烦的方法

在“简单的方法”中讨论的两个宏仅仅适用于访问 1 维或 2 维的数组(回忆一下，1 维的数组，或者称为“向量”实际只是一个 n×1 维矩阵)。OpenCV 提供了处理多维数组的机制。事实上，OpenCV 可以支持普通的 N 维的数组，这个 N 值可以取值为任意大的数。

为了访问普通矩阵中的数据，我们可以利用在例 3-6 和例 3-7 中列举的 cvPtr*D 和 cvGet*D,, 等函数族。cvPtr*D 家族包括 cvPtr1D(), cvPtr2D(), cvPtr3D() 和 cvPtrND(),, 。这三个函数都可接收 CvArr*类型的矩阵指针参数，紧随其后的参数是表示索引的整数值，最后是一个可选的参数，它表示输出值的类型。函数返回一个指向所需元素的指针。对于 cvPtrND()来说，第二个参数是一个指向一个整型数组的指针，这个数组中包含索引的合适数字。后文会再次介绍此函数(在这之后的原型中，也会看到一些可选参数，必要时会有讲解)。

例 3-6：指针访问矩阵结构

```
uchar*  cvPtr1D(
  const  CvArr*    arr,
  int          idx0,
  int*             type  = NULL
);
uchar*  cvPtr2D(
  const  CvArr*    arr,
  int          idx0,
  int          idx1,
  int*             type  = NULL
```

```
);
uchar*  cvPtr3D(
  const  CvArr*    arr,
  int          idx0,
  int          idx1,
  int          idx2,
  int*             type  = NULL
);
uchar*  cvPtrND(
  const  CvArr*    arr,
  int*             idx,
  int*             type            =  NULL,
  int          create_node     = 1,
  unsigned*        precalc_hashval =  NULL
);                                                    【37～38】
```

如果仅仅是读取数据，可用另一个函数族 cvGet*D。如例 3-7 所示，该例与例 3-6 类似，但是返回矩阵元素的实际值。

例3-7：CvMat 和IPlImage元素函数

```
double cvGetReal1D(  const CvArr*  arr,  int idx0  );
double cvGetReal2D(  const CvArr*  arr, int  idx0,  int  idx1  );
double cvGetReal3D(  const CvArr*  arr, int  idx0,  int  idx1,  int idx2  );
double cvGetRealND(  const  CvArr* arr,  int* idx  );
CvScalar  cvGet1D(  const CvArr*  arr, int  idx0  );
CvScalar  cvGet2D(  const CvArr*  arr, int  idx0,  int  idx1 );
CvScalar  cvGet3D(  const CvArr*  arr, int  idx0,  int  idx1,  int idx2  );
CvScalar  cvGetND(  const CvArr* arr,  int*  idx );
```

cvGet*D 中有四个函数返回的是整型的，另外四个的返回值是 CvScalar 类型的。这意味着在使用这些函数的时候，会有很大的空间浪费。所以，只是在你认为用这些函数比较方便和高效率的时候才用它们，否则，最好用 cvPtr*D。

用 cvPtr*D()函数族还有另外一个原因，即可以用这些指针函数访问矩阵中的特定的点，然后由这个点出发，用指针的算术运算得到指向矩阵中的其他数据的指针。在多通道的矩阵中，务必记住一点：通道是连续的，例如，在一个 3 通道 2 维的表示红、绿、蓝(RGB)矩阵中。矩阵数据如下存储 rgbrgbrgb . . .。所以，要将指向该数据类型的指针移动到下一通道，我们只需要将其增加 1。如果想访问下一个“像素”或者元素集，我们只需要增加一定的偏移量，使其与通道数相等。

另一个需要知道的技巧是矩阵数组的 step 元素(参见例 3-1 和例 3-3)，step 是矩阵中行的长度，单位为字节。在那些结构中，仅靠 cols 或 width 是无法在矩阵的不同行之间移动指针的，出于效率的考虑，矩阵或图像的内存分配都是 4 字节的整数倍。所以，三个字节宽度的矩阵将被分配 4 个字节，最后一个字节被忽略。因此，如果我们得到一个字节指针，该指针指向数据元素，那么我们可以用 step 和这个指针相加以使指针指向正好在我们的点的下一行元素。如果我们有一个整型或者浮点型的矩阵，对应的有整型和浮点型的指针指向数据区域，我们将让 step/4 与指针相加来移到下一行，对双精度型的，我们让 step/8 与指针相加(这里仅仅考虑了 C 将自动地将差值与我们添加的数据类型的字节数相乘)。

【38】

例 3-8 中的 cvSet*D 和 cvGet*D 多少有些相似，它通过一次函数调用为一个矩阵或图像中的元素设置值，函数 cvSetReal*D()和函数 cvSet*D()可以用来设置矩阵或者图像中元素的数值。

例3-8：为CvMat或者IplImage元素设定值的函数

```
void cvSetReal1D( CvArr* arr, int  idx0,  double value  );
void cvSetReal2D( CvArr* arr, int  idx0,  int  idx1,  double value  );
void cvSetReal3D(
  CvArr*  arr,
  int idx0,
  int idx1,
```

```
  int idx2,
  double value
);
void cvSetRealND( CvArr* arr, int* idx, double value );
void cvSet1D( CvArr* arr, int idx0, CvScalar value );
void cvSet2D( CvArr* arr, int idx0, int idx1, CvScalar value );
void cvSet3D(
  CvArr* arr,
  int idx0,
  int idx1,
  int idx2,
  CvScalar value
);
void cvSetND( CvArr* arr, int* idx, CvScalar value );
```

为了方便，我们也可以使用 cvmSet()和 cvmGet()，这两个函数用于处理浮点型单通道矩阵，非常简单。

```
double cvmGet( const CvMat* mat, int row, int col )
void cvmSet( CvMat* mat, int row, int col, double value )
```

以下函数调用 cvmSet()：

```
cvmSet( mat, 2, 2, 0.5000 );
```

等同于 cvSetReal2D 函数调用：

```
cvSetReal2D( mat, 2, 2, 0.5000 );
```

恰当的方法

从以上所有那些访问函数来看，你可能会想，没有必要再介绍了。实际上，这些 set 和 get 函数很少派上用场。大多数时侯，计算机视觉是一种运算密集型的任务，因而你想尽量利用最有效的方法做事。毋庸置疑，通过这些函数接口是不可能做到十分高效的。相反地，应该定义自己的指针计算并且在矩阵中利用自己的方法。如果打算对数组中的每一个元素执行一些操作，使用自己的指针是尤为重要的(假设没有可以为你执行任务的 OpenCV 函数)。

要想直接访问矩阵，其实只需要知道一点，即数据是按光栅扫描顺序存储的，列(“x”)是变化最快的变量。通道是互相交错的，这意味着，对于一个多通道矩阵来说，它们变化的速度仍然比较快。例 3-9 显示了这一过程。
【39～40】

例3-9：累加一个三通道矩阵中的所有元素

```
float sum( const CvMat* mat ) {
  float s = 0.0f;
  for(int row=0; row<mat->rows; row++ ) {
    const float* ptr=(const float*)(mat->data.ptr + row * mat->step);
    for( col=0; col<mat->cols; col++ ) {
      s += *ptr++;
    }
  }
  return( s );
}
```

计算指向矩阵的指针时，记住一点：矩阵的元素 data 是一个联合体。所以，对这个指针解引用的时候，必须指明结构体中的正确的元素以便得到正确的指针类型。然后，为了使指针产生正确的偏移，必须用矩阵的行数据长度(step)元素。我们以前曾提过，行数据元素的是用字节来计算的。为了安全，指针最好用字节计算，然后分配恰当的类型，如浮点型。CvMat 结构中为了兼容 IplImage 结构，有宽度和高度的概念，这个概念已经被最新的行和列取代。最后要注意，我们为每行都重新计算了 ptr，而不是简单地从开头开始，尔后每次读的时候累加指针。这看起来好像很繁琐，但是因为CvMat 数据指针可以指向一个大型数组中的ROI，所以无法保证数据会逐行连续存取。

点的数组

有一个经常提到但又必须理解的问题是，包含多维对象的多维数组(或矩阵)和包含一维对象的高维数组之间的不同。例如，假设有 n 个三维的点，你想将这些点传递到参数类型为CvMat*的一些OpenCV 函数中。对此，有四种显而易见的方式，记住，这些方法不一定等价。一是用一个二维数组，数组的类型是 CV32FC1，有 n 行，3 列(n×3)。类似地，也可以用一个 3 行 n 列(3×n)的二维数组。也可以用一个 n 行 1 列(n×1)的数组或者 1 行 n 列(1×n)的数组，数组的类型是 CV32FC3。这些例子中，有些可以自由转换(这意味着只需传递一个，另一个便可可以计算得到)，有的则不能。要想理解原因，可以参考图 3-2 中的内存布局 情况。

从图中可以看出，在前三种方式中，点集以同样的方式被映射到内存。但最后一种方式则不同。对 N 维数组的 c 维点，情况变得更为复杂。需要记住的最关键的一点是，给定点的位置可以由以下公式计算出来。

$$\delta = (\mathrm{row}) \cdot N_{\mathrm{cols}} \cdot N_{\mathrm{channels}} + (\mathrm{col}) \cdot N_{\mathrm{channels}} + (\mathrm{channel})$$

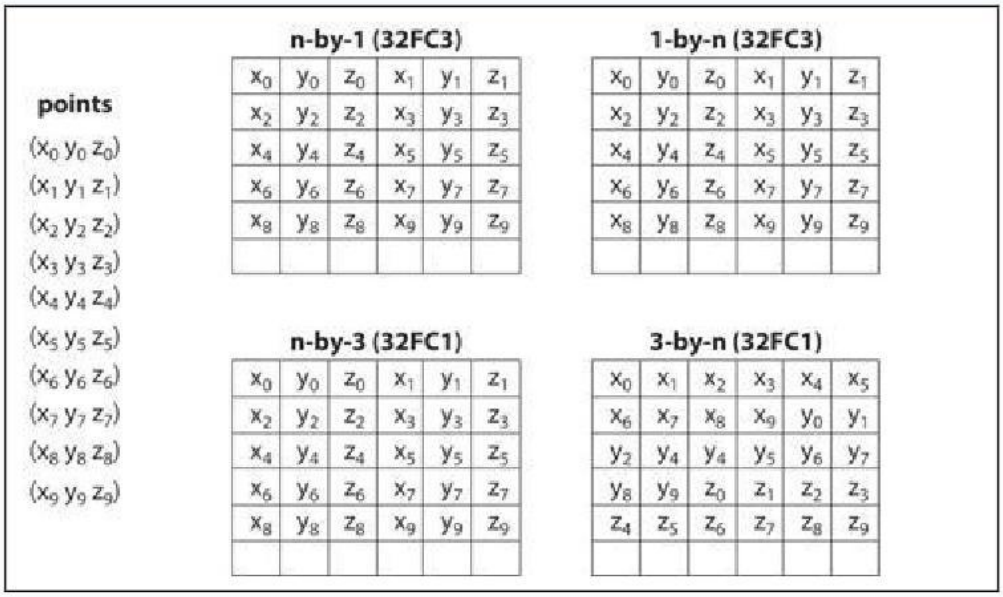

图3-2：有 10 个点，每个点由 3 个浮点数表示，这 10 个点被放在4 个结构稍有不同的数组中。前 3 种情况下，内存布局情况是相同的，但最后一种情况下，内存布局不同

其中，N_{cols} 和 $N_{channels}$ 分别表示列数和通道数。总的来说，从这个公式中可以看出一点，一个 c 维对象的N 维数组和一个一维对象的(N+c)维数组不同。至于 N=1(即把向量描绘成 n×1 或者 1×n 数组)，有一个特殊之处(即图 3-2 显示的等值)值得注意，如果考虑到性能，可以在有些情况下用到它。

关于 OpenCV 的数据类型，如 CvPoint2D 和 CvPoint2D32f，我们要说明的最后一点是：这些数据类型被定义为 C 结构，因此有严格定义的内存布局。具体说来，由整型或者浮点型组成的结构是顺序型的通道。那么，对于一维的 C 语言的对象数组来说，其数组元素都具有相同的内存布局，形如 CV32FC2 的 n×1 或者 1×n 数组。这和申请 CvPoint3D32f 类型的数组结构也是相同

的。 【41】

IplImage 数据结构

掌握了前面的知识，再来讨论 IplImage 数据结构就比较容易了。从本质上讲，它是一个 CvMat 对象，但它还有其他一些成员变量将矩阵解释为图像。这个结构

最初被定义为 Intel 图像处理库(IPL)的一部分。IplImage 结构的准确定义如例 3-10 所示。

例 3-10：IplImage 结构

```
typedef struct _IplImage  {
  int                    nSize;
  int                    ID;
  int                    nChannels;
  int                    alphaChannel;
  int                    depth;
  char                   colorModel[4];
  char                   channelSeq[4];
  int                    dataOrder;
  int                    origin;
  int                    align;
  int                    width;
  int                    height;
  struct _IplROI*          roi;
  struct _IplImage*        maskROI;
  void*                  imageId;
  struct _IplTileInfo*   tileInfo;
  int                    imageSize;
  char*                  imageData;
  int                    widthStep;
  int                    BorderMode[4];
  int                    BorderConst[4];
  char*                  imageDataOrigin;
}  IplImage;
```

我们试图讨论这些变量的某些功能。有些变量不是很重要，但是有些变量非常重要，有助于我们理解 OpenCV 解释和处理图像的方式。

width 和 height 这两个变量很重要，其次是 depth 和 nchannals。depth 变量的值取自 ipl.h 中定义的一组数据，但与在矩阵中看到的对应变量不同。因为在图像中，我们往往将深度和通道数分开处理，而在矩阵中，我们往往同时表示它们。可用的深度值如表 3-2 所

示。

表3-2：OpenCV图像类型

宏	图像像素类型
IPL_DEPTH_8U	无符号 8位整数 (8u)
IPL_DEPTH_8S	有符号 8位整数(8s)
IPL_DEPTH_16S	有符号16位 整数(16s)
IPL_DEPTH_32S	有符号32位 整数(32s)
IPL_DEPTH_32F	32 位浮 点数单 精度(32f)
IPL_DEPTH_64F	64 位浮 点数双 精度(64f)

通道数 nChannels 可取的值是 1，2，3 或 4。

随后两个重要成员是 origin 和 dataOrder。origin 变量可以有两种取值：IPL_ORIGIN_TL 或者 IPL_ORIGIN_BL，分别设置坐标原点的位置于图像的左上角或者左下角。在计算机视觉领域，一个重要的错误来源就是原点位置的定义不统一。具体而言，图像的来源、操作系统、编解码器和存储格式等因素都可以影响图像坐标原点的选取。举例来说，你或许认为自己正在从图像上面的脸部附近取样，但实际上却在图像下方的裙子附近取样。避免此类现象发生的最好办法是在最开始的时候检查一下系统，在所操作的图像块的地方画点东西试试。

dataOrder 的取值可以是 IPL_DATA_ORDER_PIXEL 或 IPL_DATA_ORDER_PLANE，前者指明数据是将像素点不同通道的值交错排在一起(这是常用的交错排列方式)，后者是把所有像素同通道值排在一起，形成通道平面，再把平面排列起来。

参数 widthStep 与前面讨论过的 CvMat 中的 step 参数类似，包括相邻行的同列点之间的字节数。仅凭变量 width 是不能计算这个值的，因为为了处理过程更高效每行都会用固定的字节数来对齐；因此在第 i 行末和第 i+1 行开始处可能会有

些冗于字节。参数 imageData 包含一个指向第一行图像数据的指针。如果图像中有些独立的平面(如当 dataOrder = IPL_DATA_ORDER_PLANE)那么把它们作为单独的图像连续摆放，总行数为 height 和 nChannels 的乘积。但通常情况下，它们是交错的，使得行数等于高度，而且每一行都有序地包含交错的通道。

最后还有一个实用的重要参数—— 感兴趣的区域(ROI)，实际上它是另一个 IPL/IPP 结构 IplROI 的实例。IplROI 包含 xOffset，yOffset，height，width 和 coi 成员变量，其中 COI 代表 channel of interest(感兴趣的通道)。ROI 的思想是：一旦设定 ROI，通常作用于整幅图像的函数便会只对 ROI 所表示的子图像进行操作。如果 IplImage 变量中设置了 ROI，则所有的 OpenCV 函数就会使用该 ROI 变量。如果 COI 被设置成非 0 值，则对该图像的操作就只作用于被指定的通道上了。不幸的是，许多 OpenCV 函数都忽略参数 COI。

访问图像数据

通常，我们需要非常迅速和高效地访问图像中的数据。这意味着我们不应受制于存取函数(如 cvSet*D 之类)。实际上，我们想要用最直接的方式访问图像内的数据。现在，应用已掌握的 IplImage 内部结构的知识，我们知道怎样做才是最佳的 方法。

虽然 OpenCV 中有很多优化函数帮助我们完成大多数的图像处理的任务，但是还有一些任务，库中没有预先包装好的函数可以帮我们解决。例如，如果我们有一个三通道 HSV 图像[Smith78]，在色度保持不变的情况下，我们要设置每个点的饱和度和高度为 255(8 位图像的最大值)，我们可以使用指针遍历图像，类似于例 3-9 中的矩阵遍历。然而，有一些细微的不同，是源于 IplImage 和 CvMat 结构的差异。例 3-11 演示了最高效的方法。

例 3-11：仅最大化 HSV 图像“S”和“V”部分

```
void saturate_sv( IplImage* img ) {
 for( int y=0; y<img->height; y++ ) {
  uchar* ptr = (uchar*) (
   img->imageData + y * img->widthStep
  );
```

```
  for( int  x=0;  x<img->width;  x++  )  {
    ptr[3*x+1]  = 255;
    ptr[3*x+2]  = 255;
  }
 }
}
```

在以上程序中，我们用指针 ptr 指向第 y 行的起始位置。接着，我们从指针中析出饱和度和高度在 x 维的值。因为这是一个三通道图像，所以 C 通道在 x 行的位置是 3*x+c。

与 CvMat 的成员 data 相比，IplImage 和 CvMat 之间的一个重要度别在于 imageData。CvMat 的 data 元素类型是联合类型，所以你必须说明需要使用的指针类型。imageData 指针是字节类型指针(uchar ＊)。我们已经知道是种类型的指针指向的数据是 uchar 类型的，这意味着，在图像上进行指针运算时，你可以简单地增加 widthStep（也以字节为单位)，而不必关心实际数据类型。在这里重新说明一下：当要处理的是矩阵时，必须对偏移并进行调整，因为数据指针可能是非字节类型；当要处理的是图像时，可以直接使用偏移，因为数据指针总是字节类型，因此当你要用到它的时候要清楚是怎么回事。

对 **ROI** 和 **widthStep** 的补充

ROI 和 widthStep 在实际工作中有很重要的作用，在很多情况下，使用它们会提高计算机视觉代码的执行速度。这是因为它们允许对图像的某一小部分进行操作，而不是对整个图像进行运算。在 OpenCV 中，普遍支持 ROI 和 widthStep，函数的操作被限于感兴趣区域。要设置或取消 ROI，就要使用 cvSetImageROI()和 cvResetImageROI()函数。如果想设置 ROI，可以使用函数 cvSetImageROI()，并为其传递一个图像指针和矩形。而取消 ROI，只需要为函数 cvResetImageROI()传递一个图像指针。

```
void  cvSetImageROI(  IplImage*  image,  CvRect rect  );
void  cvResetImageROI(  IplImage*  image  );
```

为了解释 ROI 的用法，我们假设要加载一幅图像并修改一些区域，如例 3-12 的代码，读取了一幅图像，并设置了想要的 ROI 的 x，y，width 和 height 的值，最后将 ROI 区域中像素都加上一个整数。本例程中通过内联的 cvRect()构造函数设置 ROI。通过 cvResetImageROI()函数释放 ROI 是非常重要的，否则，将忠实地只显示 ROI 区域。

例 3-12：用 imageROI 来增加某范围的像素

```
// roi_add  <image>  <x>  <y> <width>  <height> <add>
#include  <cv.h>
#include  <highgui.h>
int main(int argc,  char**  argv)
{
    IplImage* src;
    if(  argc  == 7  &&  ((src=cvLoadImage(argv[1],1))  !=  0  ))
    {
        int x  = atoi(argv[2]);
        int y  = atoi(argv[3]);
        int width  = atoi(argv[4]);
        int height  = atoi(argv[5]);
        int add  = atoi(argv[6]);
        cvSetImage  ROI(src,  cvRect(x,y,width,height));
        cvAddS(src,  cvScalar(add),src);
        cvResetImageROI(src);
        cvNamedWindow(  "Roi_Add",  1  );
        cvShowImage(  "Roi_Add", src  );
        cvWaitKey();
    }
    return 0;
}
```

使用例 3-12 中的代码把 ROI 集中于一张猫的脸部，并将其蓝色通道增加 150 后的效果如图 3-3 所

示。

【45～46】

图3-3：在猫脸上用ROI增加 150像素的效果

通过巧妙地使用 widthStep，我们可以达到同样的效果。要做到这一点，我们创建另一个图像头，让它的 width 和 height 的值等于 interest_rect 的 width 和 height 的值。我们还需要按 interest_rect 起点设置图像起点(左上角或者左下角)。下一步，我们设置子图像的 widthStep 与较大的 interest_img 相同。这样，即可在子图像中逐行地步进到大图像里子区域中下一行开始处的合适位置。最后设置子图像的 imageDate 指针指向兴趣子区域的开始，如例 3-13 所示。

例3-13：利用其他 widthStep 方法把 interest_img 的所有像素值增加 1

```
// Assuming IplImage *interest_img; and
// CvRect interest_rect;
// Use widthStep to get a region of interest
//
// (Alternate method)
//
```

```
IplImage *sub_img = cvCreateImageHeader(
  cvSize(
    interest_rect.width,
    interest_rect.height
  ),
  interest_img->depth,
  interest_img->nChannels
);
sub_img->origin = interest_img->origin;
sub_img->widthStep = interest_img->widthStep;
sub_img->imageData = interest_img->imageData +
  interest_rect.y * interest_img->widthStep +
  interest_rect.x * interest_img->nChannels;
cvAddS( sub_img, cvScalar(1), sub_img );
cvReleaseImageHeader(&sub_img);
```

看起来设置和重置 ROI 更方便一些，为什么还要使用 widthStep？原因在于有些时候在处理的过程中，想在操作过程中设置和保持一幅图像的多个子区域处于活动状态，但是 ROI 只能串行处理并且必须不断地设置和重置。

最后，我们要在此提到一个词—— 掩码或模板，在代码示例中 cvAddS()函数允许第四个参数默认值为空：const CvArr* mask=NULL。这是一个 8 位单通道数组，它允许把操作限制到任意形状的非 0 像素的掩码区，如果 ROI 随着掩码或模板变化，进程将会被限制在 ROI 和掩码的交集区域。掩码或模板只能在指定了其图像的函数中使用。

矩阵和图像操作

表 3-3 列出了一些操作矩阵图像的函数，其中的大部分对于图像处理非常有效。它们实现了图像处理中的基本操作，例如对角化、矩阵变换以及一些更复杂的诸如计算图像的统计操

作。

【47】

表3-3：矩阵和图像基本操作

函数名称	描述
cvAbs	计算数组中所有元素的绝对值
cvAbsDiff	计算两个数组差值的绝对值

续表

函数名称	描述
cvAbsDiffS	计算数组和标量差值的绝对值
cvAdd	两个数组的元素级的加运算
cvAddS	一个数组和一个标量的元素级的相加运算
cvAddWeighted	两个数组的元素级的加权相加运算 (alpha 融合)
cvAvg	计算数组中所有元素的平均值
cvAvgSdv	计算数组中所有元素的绝对值和标准差
cvCalcCovarMatrix	计算一组 n 维空间 向量的 协方差
cvCmp	对两个数组中的所有元素运用设置的比较操作
cvCmpS	对数组和标量运用设置的比较操作
cvConvertScale	用可选的缩放值转换数组元素类型
cvConvertScaleAbs	计算可选的缩放值的绝对值之后再转换数组元素的类型
cvCopy	把数组中的值复制到另一个数组中
cvCountNonZero	计算数组中非 0 值的 个数
cvCrossProduct	计算两个三维向量的向量积 (叉积)
cvCvtColor	将数组的通道从一个颜色空间转换另外一个颜色空间
cvDet	计算方阵的行列式
cvDi	用另外一个数组对一个数组进行元素级的除法运算
cvDotProduct	计算两个向量的点积
cvEigenVV	计算方阵的特征值和特征向量
cvFlip	围绕选定轴翻转
cvGEMM	矩阵乘法
cvGetCol	从一个数组的列中复制元素
cvGetCols	从数据的相邻的多列中复制元素值
cvGetDiag	复制数组中对角线上的所有元素
cvGetDims	返回数组的维数
cvGetDimSize	返回一个数组的所有维的大小
cvGetRow	从一个数组的行中复制元素值
cvGetRows	从一个数组的多个相邻的行中复制元素值
cvGetSize	得到二维的数组的尺寸，以 CvSize 返回
cvGetSubRect	从一个数组的子区域复制元素值
cvInRange	检查一个数组的元素是否在另外两个数组中的值的范围内
cvInRangeS	检查一个数组的元素的值是否在另外两个标量的范围内

续表

函数 名称	描述
cvInvert	求矩阵的转置
cvMahalonobis	计算两个向量间的马氏距离
cvMax	在两个数组中进行元素级的取最大值操作
cvMaxS	在一个数组和一个标量中进行元素级的取最大值操作
cvMerge	把几个单通道图像合并为一个多通道图像
cvMin	在两个数组中进行元素级的取最小值操作
cvMinS	在一个数组和一个标量中进行元素级的取最小值操作
cvMinMaxLoc	寻找数组中的最大最小值
cvMul	计算两个数组的元素级的乘积
cvNot	按位对数组中的每一个元素求反
cvNorm	计算两个数组的正态相关性
cvNormalize	将数组中元素进行规一化
cvOr	对两个数组进行按位或操作
cvOrS	在数组与标量之间进行按位或操作
cvReduce	通过给定的操作符将二维数组约简为向量
cvRepeat	以平铺的方式进行数组复制
cvSet	用给定值初始化数组
cvSetZero	将数组中所有元素初始化为0
cvSetIdentity	将数组中对角线上的元素设为1，其他置0
cvSol e	求出线性方程组的解
cvSplit	将多通道所组分割成多个单通道数组
cvSub	两个数组元素级的相减
cvSubS	元素级的从数组中减去标量
cvSubRS	元素级的从标量中减去数组
cvSum	对数组中的所有元素求和
cvSVD	二维矩阵的奇异值分解
cvSVBkSb	奇异值回代计算
cvTrace	计算矩阵迹
cvTranspose	矩阵的转置运算
cvXor	对两个数组进行按位异或操作
cvXorS	在数组和标量之间进行按位异或操作
cvZero	将所有数组中的元素置为0

cvAbs，cvAbsDiff 和 cvAbsDiffS

```
void  cvAbs(
   const  CvArr*    src,
   const              dst
);
void  cvAbsDiff(
  const CvArr*    src1,
```

```
    const CvArr*    src2,
    const           dst
);
void  cvAbsDiffS(
    const  CvArr*   src,
    CvScalar        value,
    const           dst
);
```

【50】

这些函数计算一个数组的绝对值或数组和其他对象的差值的绝对值，cvAbs()函数计算 src 里的值的绝对值，然后把结果写到 dst；cvAbsDiff()函数会先从 src1 减去 src2，然后将所得差的绝对值写到 dst；除了从所有 src 元素减掉的数是常标量值外，可以看到 cvAbsDiffS()函数同 cvAbsDiff()函数基本相同。

cvAdd，cvAddS，cvAddWeighted 和 alpha 融合

```
void  cvAdd(
    const  CvArr*   src1,
    const  CvArr*   src2,
    CvArr*          dst,
    const  CvArr*   mask = NULL
);
void  cvAddS(
    const  CvArr*   src,
    CvScalar        value,
    CvArr*          dst,
    const  CvArr*   mask = NULL
);
void  cvAddWeighted(
    const  CvArr*   src1,
```

```
    double          alpha,
    const CvArr*    src2,
    double          beta,
    double          gamma,
    CvArr*          dst
);
```

cvAdd()是一个简单的加法函数，它把 src1 里的所有元素同 src2 里的元素对应进行相加，然后把结果放到 dst，如果 mask 没有被设为 NULL，那么由 mask 中非零元素指定的 dst 元素值在函数执行后不变。cvAddS()与 cvAdd()非常相似，惟一不同的是被加的数量标量 value。

cvAddWeighted()函数同 cvAdd()类似，但是被写入 dst 的结果是经过下面的公式得出的：

$$dst_{x,y} = \alpha \cdot src1_{x,y} + \beta \cdot src2_{x,y} + \gamma \quad 【50】$$

这个函数可用来实现 alpha 融合 [Smith79; Porter84];也就是说，它可以用于一个图像同另一个图像的融合，函数的形式如下：

```
void  cvAddWeighted(
    const CvArr*  src1,
    double        alpha,
    const CvArr*  src2,
    double        beta,
    double        gamma,
    CvArr*        dst
);
```

在函数 cvAddWeighted()中我们有两个源图像，分别是 src1 和 src2。这些图像可以是任何类型的像素，只要它们属于同一类型即可。它们还可以有一个或三个通道(灰度或彩色)，同样也要保持类型一致。结果图像 dst，也必须同 src1 和 src2 是相同的像素类型。这些图像可能是不同尺寸，但是它们的 ROI 必须统一尺

寸，否则 OpenCV 就会产生错误，参数 alpha 是 src1 的融合强度，beta 是 src2 的融合强度，alpha 融合公式如下：

$$dst_{x,y} = \alpha \cdot src1_{x,y} + \beta \cdot src2_{x,y} + \gamma$$

可以通过设置 α 从 0 到 1 区间取值，β = 1 - α，γ 为 0，将前面公式转换为标准 alpha 融合方程。这就得出下式：

$$dst_{x,y} = \alpha \cdot src1_{x,y} + (1-\alpha) \cdot src2_{x,y}$$

但是，在加权融合图像，以及目标图像的附加偏移参数 γ 方面，cvAddWeighted()提供了更大的灵活性。一般而言，你或许想让 alpha 和 beta 不小于 0，并且两者之和不大于 1，gamma 的设置取决于像素所要调整到的平均或最大值。例 3-14 展示了 alpha 融合的用法。

例 3-14：src2 中 alpha 融合 ROI 以(0,0)开始，src1 中 ROI 以(x,y)开始

```
// alphablend <imageA> <image B> <x> <y> <width> <height>
//            <alpha> <beta>
#include <cv.h>
#include <highgui.h>
int main(int argc, char** argv)
{
   IplImage *src1, *src2;
   if( argc == 9 && ((src1=cvLoadImage(argv[1],1)) != 0
      )&&((src2=cvLoadImage(argv[2],1)) != 0 ))
   {
      int x = atoi(argv[3]);
      int y = atoi(argv[4]);
      int width = atoi(argv[5]);
      int height = atoi(argv[6]);
      double alpha = (double)atof(argv[7]);
      double beta = (double)atof(argv[8]);
      cvSetImage ROI(src1, cvRect(x,y,width,height));
      cvSetImageROI(src2, cvRect(0,0,width,height));
      cvAddWeighted(src1, alpha, src2, beta,0.0,src1);
      cvResetImageROI(src1);
      cvNamedWindow( "Alpha_blend", 1 );
      cvShowImage( "Alpha_blend", src1 );
      cvWaitKey();
   }
   return 0;
}
```

【51～52】

例 3-14 中的代码用两个源图像：初始的(src1)和待融合的(src2)。它从矩形的 ROI 中读取 src1，然后将同样大小的 ROI 应用到 src2 中，这一次设在原始位置，它从命令行读入 alpha 和 beta 的级别但是把 gamma 设为 0。Alpha 融合使用函数 cvAddWeighted()，结果被放到 src1 并显示，例子输出如图 3-4 所示，一个小孩的脸同一个猫的脸和身体被融合到了一起，值得注意的是，代码采用相同的 ROI，像图 3-3 的例子一样。这次我们使用了 ROI 作为目标融合区域。

图3-4：一个小孩的脸被alpha 融合到一只猫的脸上

cvAnd 和 cvAndS

```
void  cvAnd(
    const  CvArr*    src1,
    const  CvArr*    src2,
    CvArr*           dst,
    const  CvArr*    mask  =  NULL
);
void  cvAndS(
    const  CvArr*    src1,
```

```
    CvScalar        value,
    CvArr*          dst,
    const CvArr*    mask = NULL
);
```

这两个函数在 src1 数组上做按位与运算，在 cvAnd()中每个 dst 元素都是由相应的 src1 和 src2 两个元素进行位与运算得出的。在 cvAndS()中，位与运算由常标量 value 得出。同一般函数一样，如果 mask 是非空，就只计算非 0 mask 元素所对应的 dst 元素。

尽管支持所有的数据类型，但是对于 cvAnd()来说，src1 和 src2 要保持相同的数据类型。如果元素都是浮点型的，则使用该浮点数的按位表示。 【52】

cvAvg

```
CvScalar cvAvg(
    const CvArr*    arr,
    const CvArr*    mask = NULL
);
```

cvAvg()计算数组 arr 的平均像素值，如果 mask 为非空，那么平均值仅由那些 mask 值为非 0 的元素相对应的像素算出。

此函数还有别名 cvMean()，但不推荐使用。

cvAvgSdv

```
cvAvgSdv(
    const CvArr*    arr,
    CvScalar*       mean,
    CvScalar*       std_dev,
    const CvArr*    mask    = NULL
);
```

【53】

此函数同 cvAvg()类似，但除了求平均，还可以计算像素的标准差。

函数现在有不再使用的别名 cvMean_StdDev()。

cvCalcCovarMatrix

```
void  cvCalcCovarMatrix(
   const  CvArr**  vects,
   int            count,
   CvArr*         cov_mat,
   CvArr*         avg,
   int            flags
);
```

给定一些向量，假定这些向量表示的点是高斯分布，cvCalcCovarMatrix()将计算这些点的均值和协方差矩阵。这当然可以运用到很多方面，并且 OpenCV 有很多附加的 flags 值，在特定的环境下会起作用(参见表 3-4)。这些标志可以用标准的布尔或操作组合到一起。

表3-4：cvCalcCovarMatrix()可能用到的标志参数的值

标志参数的具体标志值	意义
CV_COVAR_NORMAL	计算均值和协方差
CV_COVAR_SCRAMBLED	快速PCA“Scrambled”协方差
CV_COVAR_USE_AVERAGE	输入均值而不是计算均值
CV_COVAR_SCALE	重新缩放输出的协方差矩阵

在所有情况下，在 vects 中是 OpenCV 指针数组(即一个指向指针数组的指针)，并有一个指示多少数组的参数 count。在所有情况下，结果将被置于 cov_mat，但是 avg 的确切含义取决于标志的值(参见表 3-4)。

标识 CV_COVAR_NORMAL 和 CV_COVAR_SCRAMBLED 是相互排斥的；只能使用其中一种，不能两者同时使用。如果为 CV_COVAR_NORMAL，函数便只计算该点的均值和协方差。

$$\sum\nolimits_{\text{normal}}^{-1} = z \begin{bmatrix} v_{0,0} - \bar{v}_0 & \cdots & v_{m,0} - \bar{v}_0 \\ \vdots & \ddots & \vdots \\ v_{0,n} - \bar{v}_n & \cdots & v_{m,n} - \bar{v}_n \end{bmatrix} \begin{bmatrix} v_{0,0} - \bar{v}_0 & \cdots & v_{m,0} - \bar{v}_0 \\ \vdots & \ddots & \vdots \\ v_{0,n} - \bar{v}_n & \cdots & v_{m,n} - \bar{v}_n \end{bmatrix}^T$$

因此，标准的协方差 $\sum_{\text{normal}}^{-1}$ 由长度为 n 的 m 个向量计算，其中 $\bar{v}_n$ 被定义为平均向量 $\bar{v}$ 的第 n 个元素，由此产生的协方差矩阵是一个 n × n 矩阵，比例 z 是一个

可选的缩放比例，除非使用 CV_COVAR_SCALE 标志，否则它将被设置为 1。

【54】

如果是 CV_COVAR_SCRAMBLED 标志，cvCalcCovarMatrix ()将如下计算：

$$\sum_{\text{scrambled}}^{2} = z\begin{bmatrix} v_{0,0}-\bar{v}_0 & \cdots & v_{m,0}-\bar{v}_0 \\ \vdots & \ddots & \vdots \\ v_{0,n}-\bar{v}_n & \cdots & v_{m,n}-\bar{v}_n \end{bmatrix}^{T}\begin{bmatrix} v_{0,0}-\bar{v}_0 & \cdots & v_{m,0}-\bar{v}_0 \\ \vdots & \ddots & \vdots \\ v_{0,n}-\bar{v}_n & \cdots & v_{m,n}-\bar{v}_n \end{bmatrix}$$

这种矩阵不是通常的协方差矩阵(注意转置运算符的位置)，这种矩阵的计算来自同样长度为 n 的 m 个向量，但由此而来的协方差矩阵是一个 m×m 矩阵。这种矩阵是用在一些特定的算法中，如针对非常大的向量的快速 PCA 分析法(人脸识别可能会用到此运算)。

如果已知平均向量，则使用标志 CV_COVAR_USE_AVG，在这种情况下，参数 avg 用来作为输入而不是输出，从而减少计算时间。

最后，标志 CV_COVAR_SCALE 用于对计算得到的协方差矩阵进行均匀缩放。这是前述方程的比例 z，同标志 CV_COVAR_NORMAL 一起使用时，应用的缩放比例将是 1.0 /m(或等效于 1.0/count)。如果不使用 CV_COVAR_SCRAMBLED，那么 z 的值将会是 1.0/n(向量长度的倒数)，cvCalcCovarMatrix()的输入输出矩阵都应该是浮点型，结果矩阵 cov_mat 的大小应当是 n×n 或者 m×m，这取决于计算的是标准协方差还是 scrambled 的协方差。应当指出的是，在 vects 中输入的“向量”并不一定要是一维的；它们也可以是两维对象(例如图像)。

cvCmp 和 cvCmpS

```
void  cvCmp(
    const  CvArr*     src1,
    const  CvArr*     src2,
    CvArr*            dst,
    int               cmp_op
);
void  cvCmpS(
    const  CvArr*     src,
    double            value,
```

```
    CvArr*          dst,
    int             cmp_op
);
```

这两个函数都是进行比较操作，比较两幅图像相应的像素值或将给定图像的像素值与某常标量值进行比较。cvCmp()和 cvCmpS()的最后一个参数的比较操作符可以是表 3-5 所列出的任意一
个。

表 3-5：cvCmp()和 cvCmpS()使用的 cmp_op 值以及由此产生的比较操作

cmp_op 的值	比较方法
CV_CMP_EQ	(src1i == src2i)
CV_CMP_GT	(src1i > src2i)
CV_CMP_GE	(src1i >= src2i)
CV_CMP_LT	(src1i < src2i)
CV_CMP_LE	(src1i <= src2i)
CV_CMP_NE	(src1i != src2i)

表 3-5 列出的比较操作都是通过相同的函数实现的，只需传递合适的参数来说明你想怎么做，这些特殊的功能操作只能应用于单通道的图像。

这些比较功能适用于这样的应用程序，当你使用某些版本的背景减法并想对结果进行掩码处理但又只从图像中提取变化区域信息时(如从安全监控摄像机看一段视 频流)。

cvConvertScale

```
void  cvConvertScale(
    const  CvArr*  src,
    CvArr*         dst,
    double         scale  = 1.0,
    double         shift =  0.0
);
```

cvConvertScale()函数实际上融多种功能于一体，它能执行几个功能中的任意之一，如果需要，也可以一起执行多个功能。第一个功能是将源图像的数据类型转

变成目标图像的数据类型。例如，如果我们有一个 8 位的 RGB 灰度图像并想把它变为 16 位有符号的图像，就可以调用函数 cvConvertScale()来做这个工作。cvConvertScale()的第二个功能是对图像数据执行线性变换。在转换成新的数据类型之后，每个像素值将乘以 scale 值，然后将 shift 值加到每个像素上。

至关重要的是要记住，尽管在函数名称中“Convert”在“Scale”之前，但执行这些操作的顺序实际上是相反的。具体来说，在数据类型转变之前，与 scale 相乘和 shift 的相加已经发生

了。

【56】

如果只是传递默认值(scale = 1.0 和 shift = 0.0)，则不必担心性能; OpenCV 足够聪明，能意识到这种情况而不会在无用的操作上浪费处理器的时间。澄清一下(如果你想添加一些)，OpenCV 还提供了宏指令 cvConvert()，该指令同 cvConvertScale()一样，但是通常只适用于 scale 和 shift 参数设为默

认 值时。

对于所有数据类型和任何数量通道 cvConvertScale ()都适用，但是源图像和目标图像的通道数量必须相同。(如果你想实现彩色图像与灰度图的相互转换，可以使用 cvCvtColor()，之后我们将会提到。)

【56～57】

cvConvertScaleAbs

```
void  cvConvertScaleAbs(
   const  CvArr*  src,
   CvArr*         dst,
   double         scale  = 1.0,
   double         shift =  0.0
);
```

cvConvertScaleAbs()与 cvConvertScale()基本相同，区别是 dst 图像元素是结果数据的绝对值。具体说来，cvConvertScaleAbs()先缩放和平移，然后算出绝对值，最后进行数据类型的转换。

cvCopy

```
void cvCopy(
  const CvArr* src,
  CvArr*       dst,
  const CvArr* mask = NULL
);
```

用于将一个图像复制到另一个图像。cvCopy()函数要求两个数组具有相同的数据类型、相同的大小和相同的维数。可以使用它来复制稀疏矩阵，但这样做时，不支持 mask。对于非稀疏矩阵和图像，mask 如果为非空，则只对与 mask 中与非 0 值相对应的 dst 中的像素赋值。

cvCountNonZero

```
int cvCountNonZero( const CvArr* arr );
```

cvCountNonZero()返回数组 arr 中非 0 像素的个数。

cvCrossProduct

```
void cvCrossProduct(
const CvArr* src1,
const CvArr* src2,
CvArr* dst
);
```

这个函数的主要功能是计算两个三维向量的叉积[Lagrange1773]。无论向量是行或者列的形式函数都支持。(实际上对于单通道矩阵，行向量和列向量的数据在内存中的排列方式完全相同)。src1 和 src2 都必须是单道数组，同时 dst 也必须是单道的，并且长度应精确为 3。所有这些阵列的数据类型都要一致。 【57】

cvCvtColor

```
void  cvCvtColor(
   const  CvArr*  src,
   CvArr*  dst,
   int  code
);
```

此前介绍的几个函数用于把一个数据类型转换成另一个数据类型，原始图像和目标图像的通道数目应保持一致。另外一个函数是 cvCvtColor()，当数据类型一致时，它将图像从一个颜色空间(通道的数值)转换到另一个[Wharton71]。具体转换操作由参数 code 来指定，表 3-6 列出了此参数可能的值。

表3-6：cvCvtColor()的转换

转换代码	解释
CV_BGR2RGB CV_RGB2BGR CV_RGBA2BGRA CV_BGRA2RGBA	在RGB 或BGR 色彩空间之间转 换(包括或者不包括 alpha 通道)
CV_RGB2RGBA CV_BGR2BGRA	在 RGB 或BGR 图像中加入alpha 通道
CV_RGBA2RGB CV_BGRA2BGR	从 RGB 或BGR 图像中删除alpha 通道
CV_RGB2BGRA CV_RGBA2BGR CV_BGRA2RGB CV_BGR2RGBA	加入或者移除alpha通 道时，转换 RGB 到 BGR 色彩空间
CV_RGB2GRAY CV_BGR2GRAY	转换 RGB 或者BGR色彩空间为灰度空间
CV_GRAY2RGB CV_GRAY2BGR CV_RGBA2GRAY CV_BGRA2GRAY	转换灰度为 RGB 或者BGR色彩空间(在进程中选择移 除alpha 通道)
CV_GRAY2RGBA CV_GRAY2BGRA	转换灰度为 RGB 或者BGR色彩空间并 且加 入alpha 通道
CV_RGB2BGR565 CV_BGR2BGR565 CV_BGR5652RGB CV_BGR5652BGR	在从RGB 或者 BGR 色彩空间转换到BGR565彩色图 画时，选择加入或者移除 alpha通道 (16位图)

转换代码	解释
CV_RGBA2BGR565	
CV_BGRA2BGR565	
CV_BGR5652RGBA	
CV_BGR5652BGRA	
CV_GRAY2BGR565	转换灰度为 BGR565 彩色图像或者反变换(16 位图)
CV_BGR5652GRAY	

续表

转换代码	解释
CV_RGB2BGR555	在从 RGB 或者 BGR 色彩空间转换到 BGR555 色彩空间时，选择加入或者移除 alpha 通道(16 位图)
CV_BGR2BGR555	
CV_BGR5552RGB	
CV_BGR5552BGR	
CV_RGBA2BGR555	
CV_BGRA2BGR555	
CV_BGR5552RGBA	
CV_BGR5552BGRA	
CV_GRAY2BGR555	转换灰度到 BGR555 色彩空间或者反变换(16 位图)
CV_BGR5552GRAY	
CV_RGB2XYZ	转换 RGB 或者 BGR 色彩空间到 CIE XYZ 色彩空间或者反变换(Rec 709 和 D65 白点)
CV_BGR2XYZ	
CV_XYZ2RGB	
CV_XYZ2BGR	
CV_RGB2YCrCb	转换 RGB 或者 BGR 色彩空间到 luma-chroma (aka YCC)色彩空间
CV_BGR2YCrCb	
CV_YCrCb2RGB	
CV_YCrCb2BGR	
CV_RGB2HSV	转换 RGB 或者 BGR 色彩空间到 HSV(hue，saturation，value)色彩空间或反变换
CV_BGR2HSV	
CV_HSV2RGB	
CV_HSV2BGR	
CV_RGB2HLS	转换 RGB 或者 BGR 色彩空间到 HLS(hue，Lightness，saturation)色彩空间或反变换
CV_BGR2HLS	
CV_HLS2RGB	
CV_HLS2BGR	
CV_RGB2Lab	转换 RGB 或者 BGR 色彩空间到 CIE LAB 色彩空间或反变换
CV_BGR2Lab	
CV_Lab2RGB	
CV_Lab2BGR	

续表

转换代码	解释
CV_RGB2Luv	转换 RGB 或者 BGR 色彩空间到 CIE Luv 色彩空间

转换代码	解释
CV_BGR2Luv	
CV_Luv2RGB	
CV_Luv2BGR	
CV_BayerBG2RGB	转换 Bayer 模式(单通道)到 RGB 或者 BGR 色彩空间
CV_BayerGB2RGB	
CV_BayerRG2RGB	
CV_BayerGR2RGB	
CV_BayerBG2BGR	
CV_BayerGB2BGR	
CV_BayerRG2BGR	
CV_BayerGR2BGR	

这里不再进一步阐述 CIE 色彩空间中 Bayer 模式的细节，但许多这样的转换是很有意义的。我们的目的是，了解 OpenCV 能够在哪些色彩空间进行转换，这对用户来说很重要。

色彩空间转换都用到以下约定：8 位图像范围是 0～255，16 位图像范围是 0～65536，浮点数的范围是 0.0～1.0。黑白图像转换为彩色图像时，最终图像的所有通道都是相同的；但是逆变换(例如 RGB 或 BGR 到灰度)，灰度值的计算使用加权公式：

$$Y=(0.299)R+(0.587)G+(0.114)B$$

就 HSV 色彩模式或者 HLS 色彩模式来说，色调通常是在 0～360 之间。在 8 位图中，这可能出现问题，因此，转换到 HSV 色彩模式，并以 8 位图的形式输出时，色调应该除以 2。

cvDet

```
double  cvDet(const CvArr*  mat);
```

cvDet()用于计算一个方阵的行列式。这个数组可以是任何数据类型，但它必须是单通道的，如果是小的矩阵，则直接用标准公式计算。然而对于大型矩阵，这样就不是很有效，行列式的计算使用高斯消去法。

值得指出的是，如果已知一个矩阵是对称正定的，也可以通过奇异值分解的策略来解决。欲了解更多信息，请参阅“cvSVD”一节。但这个策略是将 U 和 V 设置为 NULL，然后矩阵 W 的乘积就是所求正定矩阵。

cvDiv

```
void  cvDiv(
const  CvArr*  src1,
const  CvArr*  src2,
CvArr* dst,
double  scale =  1
);
```

cvDiv 是一个实现除法的简单函数；它用 src2 除以 src1 中对应元素，然后把最终的结果存到 dst 中。如果 mask 非空，那么 dst 中的任何与 mask 中 0 元素相对应的元素都不改变。如果对数组中所有元素求倒数，则可以设置 src1 为 NULL，函数将假定 src1 是一个元素全为 1 的数组。

cvDotProduct

```
double  cvDotProduct(
   const  CvArr*  src1，
   const  CvArr*  src2
);
```

【58～60】

这个函数主要计算两个 N 维向量的点积[Lagrange1773]。与叉积函数相同，点积函数也不太关注向量是行或者是列的形式。src1 和 src2 都应该是单通道的数组，并且数组的数据类型应该一致。

cvEigenVV

```
double  cvEigenVV(
   CvArr*  mat,
   CvArr*  evects,
   CvArr*  evals,
   double  eps  = 0
);
```

对对称矩阵 mat，cvEigenVV()会计算出该矩阵的特征值和相应的特征向量。函数实现的是雅可比方法[Bronshtein97]，对于小的矩阵是非常高效的，雅可比方法需要一个停止参数，它是最终矩阵中偏离对角线元素最大尺寸。可选参数 eps 用于设置这个值。在计算的过程中，所提供的矩阵 mat 的数据空间被用于计算，所以，它的值会在调用函数后改变。函数返回时，你会在 evects 中找到以行顺序保存的特征向量。对应的特征值被存储到 evals 中。特征向量的次序以对应特征值的重要性按降序排列。该 cvEigenVV()函数要求所有三个矩阵具有浮点 类型。

正如 cvDet()(前述)，如果被讨论的向量是已知的对称和正定矩阵，那么最好使用 SVD 计算 mat 的特征值和特征向量。

cvFlip

```
void cvFlip(
  const CvArr* src,
  CvArr* dst = NULL,
  int flip_mode = 0
);
```

本函数是将图像绕着在 X 轴或 Y 轴或者绕着 X 轴或 Y 轴上同时旋转。当参数 flip_mode 被设置为 0 的时候，图像只会绕 X 轴旋转。 【61】

flip_mode 被设置为正值时(例如，+1)，图像会围绕 Y 轴旋转，如果被设置成负值(例如，-1)，图像会围绕 X 轴和 Y 轴旋转。

在 Win32 运行视频处理系统时，你会发现自己经常使用此功能来进行图像格式变换，也就是坐标原点在左上角和左下角的变换。

cvGEMM

```
double cvGEMM(
  const CvArr* src1,
  const CvArr* src2,
```

```
    double alpha，
    const CvArr* src3，
    double beta，
    CvArr* dst，
    int tABC =0
);
```

广义矩阵乘法(generalized matrix multiplicatipm，GEMM)在 OpenCV 中是由 cvGEMM()来实现的，可实现矩阵乘法、转置后相乘、比例缩放等。最常见的情况下，cvGEMM()计算如下：

$$\mathbf{D}=\alpha\cdot op(\mathbf{A})\cdot op(\mathbf{B})+\beta\cdot op(\mathbf{C})$$

其中 A，B 和 C 分别是矩阵 src1，src2 和 src3，α 和 β 是数值系数，op()是附在矩阵上的可选转置。参数 src3 可以设置为空。在这种情况下，不会参与计算。转置将由可选参数 tABC 来控制，它的值可以是 0 或者(通过布尔 OR 操作)CV_GEMM_A_T、CV_GEMM_B_T 和 CV_GEMM_C_T 的任何组合(每一个标志都有一个矩阵转换相对应)。

过去的 OpenCV 包含 cvMatMul()和 cvMatMulAdd()方法，但是，它们很容易和 cvMul()混淆，其实它们的功能是完全不一样的(即两个数组的元素与元素相乘)。这个函数以宏的形式继续存在，它们直接调用 cvGEMM()。两者对应关系如表 3-7 所示。

表3-7：cvGEMM()一般用法的宏别名

cvMatMul(A，B， D)	cvGEMM(A，B，1，NULL，0，D，0)
cvMatMulAdd(A，B，C，D)	cvGEMM(A，B，1，C，1，D，0)

只有大小符合约束的矩阵才能进行乘法运算，并且所有的数据类型都应该是浮点型。cvGEMM()函数支持双通道矩阵，在这种情况下，它将双通道视为一个复数的两个部分。

cvGetCol 和 cvGetCols

```
CvMat* cvGetCol(
  const CvArr* arr，
```

```
    CvMat*  submat，
    int  col
);
CvMat* cvGetCols(
    const CvArr*  arr，
    CvMat*  submat，
    int  start_col，
    int  end_col
);
```

cvGetCol()函数被用作提取矩阵中的某一列，并把它以向量的形式返回(即只有一列的矩阵)。在这种情况下，矩阵指针submat将被修改为指向arr中的特定列，必须指出的是，该指针在修改过程中并未涉及内存的分配或数据的复制；submat 的内容仅仅是作了简单修改以使它正确地指出 arr 中所选择的列。它支持所有数据类型。

cvGetCols()函数的工作原理与 cvGetCols 完全一致，区别只在于前者将选择从 start_col 到 end_col 之间的所有列。这两个函数都返回一个与被调用的特定列或者多列(即，submat)相对应的头指针。

cvGetDiag

```
CvMat*  cvGetDiag(
    const  CvArr*  arr，
    CvMat*submat，
    int  diag=  0
);
```

cvGetDiag ()类似于 cvGetCol()；它能从一个矩阵选择某一条对角线并将其作为向量返回。submat 是一个矩阵类型的头指针。函数cvGetDiag()将填充该向量头指针中的各分量，以使用指向 arr 中的正确信息。注意，调用 cvGetDiag()会修改输入的头指针，将数据指针指向 arr 对角线上的数据，实际上，并没有复制

arr 的数据。可选参数 diag 表明 submat 指向哪一条对角线的。如果 diag 被设置为默认值 0，主对角线将被选中。如果 diag 大于 0，则始于(diag，0)的对角线将被选中，如果 diag 小于 0，则始于(0，-diag)的对角线将被选中。cvGetDiag()并不要求矩阵 arr 是方阵，但是数组 submat 长度必须与输入数组的尺寸相匹配。当该函数被调用时，最终的返回结果与输入的 submat 相同。

cvGetDims 和 cvGetDimSize

```
int  cvGetDims(
   const  CvArr*  arr，
    int* sizes=NULL
);
int  cvGetDimSize(
   const  CvArr*  arr，
    int  index
);
```

【63】

您一定还记得 OpenCV 中的矩阵维数可以远远大于 2。函数 cvGetDims()返回指定数组的维数并可返回每一个维数的大小。如果数组 sizes 非空，那么大小将被写入 sizes。如果使用了参数 sizes，它应该是一个指向 n 个整数的指针，这里的 n 指维数。如果无法事先获知维数，为了安全起见，可以把 sizes 大小指定为 CV_MAX_DIM。

函数 cvGetDimSize()返回一个由 index 参数指定的某一维大小。如果这个数组是矩阵或者图像，那么 cvGetDims()将一直返回为 2。对于矩阵和图像，由 cvGetDims()返回的 sizes 的次序将总是先是行数然后是列数。

cvGetRow 和 cvGetRows

```
CvMat*  cvGetRow(
   const  CvArr*  arr，
   CvMat*  submat，
    int  row
```

```
);
CvMat*  cvGetRows(
   const  CvArr*  arr，
   CvMat*submat，
   int  start_row，
   int  end_row
);
```

cvGetRow()获取矩阵中的一行让它作为向量(仅有一行的矩阵)返回。跟 cvGetCol()类似，矩阵头指针 submat 将被修改为指向 arr 中的某个特定行，并且对该头指针的修改不涉及内存的分配和数据的复制；submat 的内容仅是作为适当的修改以使它正确地指向 arr 中所选择的行。该指针所有数据类型。

cvGetRows()函数的工作原理与 cvGetRow()完全一致，区别只在于前者将选择从 start_row 到 end_row 之间的所有行。这两个函数都返回一个的头指针，指向特定行或者多个行。

cvGetSize

```
CvSize  cvGetSize(  const CvArr* arr  );
```

它与 cvGetDims()密切相关，cvGetDims()返回一个数组的大小。主要的不同是 cvGetSize()是专为矩阵和图像设计的，这两种对象的维数总是 2。其尺寸可以以 CvSize 结构的形式返回，例如当创建一个新的大小相同的矩阵或图像时，使用此函数就很方

便。

【64】

cvGetSubRect

```
cvGetSubRect
CvSize  cvGetSubRect(
   const  CvArr*  arr，
   CvArr*  submat，
```

```
    CvRect rect
);
```

cvGetSubRect()与 cvGetColumns()或 cvGetRows()非常类似，区别在于 cvGetSubRect()通过参数 rect 在数组中选择一个任意的子矩阵。与其他选择数组子区域的函数的函数一样，submat 仅仅是一个被 cvGetSubRect()函数填充的头，它将指向用户期望的子矩阵数据，这里并不涉及内存分配和数据的复制。

cvInRange 和 cvInRangeS

```
void  cvInRange(const CvArr*  src,
    const  CvArr*  lower,
    const  CvArr*  upper,
    CvArr*       dst
);
void  cvInRangeS(
    const  CvArr*  src,
    CvScalar     lower,
    CvScalar     upper,
    CvArr*       dst
);
```

这两个函数可用于检查图像中像素的灰度是否属于某一指定范围。cvInRange()检查，src 的每一个像素点是否落在 lower 和 upper 范围中。如果 src 的值大于或者等于 lower 值，并且小于 upper 值，那么 dst 中对应的对应值将被设置为 0xff；否则，dst 的值将被设置为 0。

cvInRangeS()的原理与之完全相同，但 src 是与 lower 和 upper 中的一组常量值(类型 CvScala)进行比较。对于这两个函数，图像 src 可以是任意类型；如果图像有多个通道，那么每一种通道都将被分别处理。注意，dst 的尺寸和通道数必须与 src 一致，且必须为 8 位的图像。

cvInvert

```
double  cvInvert(
    const  CvArr*  src，
    CvArr*  dst，
    Int  method  =  CV_LU
);
```

cvInvert()求取保存在 src 中的矩阵的逆并把结果保存在 dst 中。这个函数支持使用多种方法来计算矩阵的逆(见表 3-8)，但默认采取的是高斯消去法。该函数的返回值与所选用的方法有
关。

表3-8：cvInvert()函数中指定方法的参数值

方法的参数值	含义
CV_LU	高斯消去法 (LU 分解)
CV_SVD	奇异值分 解(SVD)
CV_SVD_SYM	对称矩阵 的SVD

就高斯消去法(method=CV_LU)来说，当函数执行完毕，src 的行列式将被返回。如果行列式是 0，那么事实上不进行求逆操作，并且数组 dst 将被设置为全 0。就 CV_SVD 或者 CV_SVD_SYM,来说，返回值是矩阵的逆条件数(最小特征值跟最大特征值的比例)。如果 src 是奇异的，那么 cvInvert()在 SVD 模式中将进行伪逆计算。

cvMahalonobis

```
CvSize  cvMahalonobis(
    const  CvArr*  vec1,
    const  CvArr*  vec2,
    CvArr*        mat
);
```

Mahalonobis 距离(Mahal)被定义为一点和高斯分布中心之间的向量距离，该距离使用给定分布的协方差矩阵的逆作为归一化标准。参见图 3-5。直观上，这是

与基础统计学中的标准分数(Z-score)类似，某一点到分布中心的距离是以该分布的方差作为单位。马氏距离则是该思路在高维空间中的推广。

cvMahalonobis()计算的公式如下：

$$r_{\text{Mahalanobis}} = \sqrt{(\mathbf{x}-\boldsymbol{\mu})^{T}\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

假设向量 vec1 对应 x 点，向量 vec2 是分布的均值。mat 是协方差矩阵的逆。实际上，这个协方差矩阵通常用 cvCalcCovarMatrix()(前面所述)来进行计算，然后用 cvInvert()来求逆。使用 SV_SVD 方法求逆是良好的程序设计习惯，因为其中一个特征值为 0 的分布这种情况在所难免!

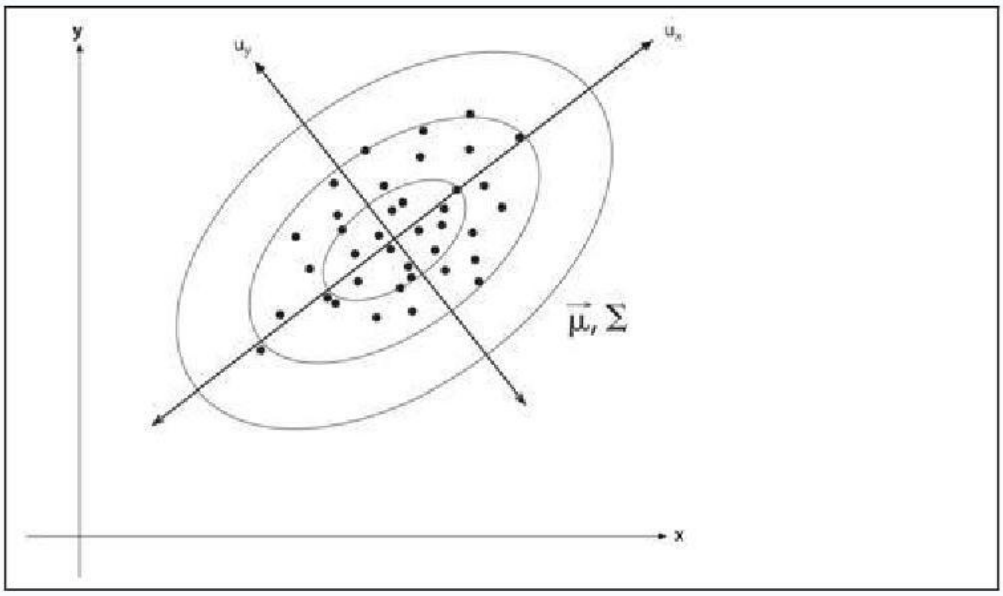

图3-5：数据在 2D 空间分布，3 个叠加在一起的椭圆分别对应到分布中心的马氏距离为1.0，2.0和3.0所有点

cvMax 和 cvMaxS

```
void  cvMax(
    const  CvArr*  src1,
    const  CvArr*  src2,
    CvArr*       dst
);
void  cvMaxS(
    const  CvArr*  src,
    double       value,
```

```
    CvArr*      dst
);                                                    【66】
```

cvMax()计算数组 src1 和 src2 中相对应的每像素一对中的最大值。而 cvMaxS()，将数组 src 与常量参数 value 进行比较。通常，如果 mask 非空，那么只有与非 0 参数相对应的 dst 中的元素参与计算。

cvMerge

```
void  cvMerge(
    const  CvArr*  src0,
    const  CvArr*  src1,
    const  CvArr*  src2,
    const  CvArr*  src3,
    CvArr*  dst
);                                                    【67】
```

cvMerge()是 cvSplit()的逆运算。数组 src0，src1，src2，和 src3 将被合并到数组 dst 中。当然，dst 应该与源数组具有相同的数据类型和尺寸，但它可以有两个，三个或四个道道。未使用的源图像参数可设置为 NULL。

cvMin 和 cvMinS

```
void  cvMin(
    const  CvArr*  src1,
    const  CvArr*  src2,
    CvArr*  dst
);
void  cvMinS(
    const  CvArr*  src,
    double  value,
    CvArr*  dst
);
```

cvMin()计算数组 src1 和 src2 中相对应的每一对像素中的最小值。而 cvMaxS()，将数组 src 与常量标量 value 进行比较。同样的，如果 mask 非空的话，那么只有与 mask 的非 0 参数相对应的 dst 中的元素进行计算。

cvMinMaxLoc

```
void  cvMinMaxLoc(
    const  CvArr*  arr,
    double*       min_val,
    double*       max_val,
    CvPoint*      min_loc  =  NULL,
    CvPoint*      max_loc  =  NULL,
    const  CvArr*  mask     = NULL
);
```

该例程找出数组 arr 中的最大值和最小值，并且(有选择性地)返回它们的地址。计算出的最大值和最小值赋值给 max_val 和 min_val。或者，如果极值的位置参数非空，那极值的位置便会写入 min_loc 和 max_loc。

通常，如果参数 mask 非空，那么只有图像 arr 中与参数 mask 中的非零的像素相对应的部分才被考虑。cvMinMaxLoc()例程仅仅处理单通道数组，如果有一个多通道的数组，则应该使用 cvSetCOI()来对某个特定通道进行设置。

cvMul

```
void  cvMul(
    const  CvArr*  src1,
    const  CvArr*  src2,
    CvArr*  dst,
    double  scale=1
);                                                    【68】
```

cvMul()是一个简单的乘法函数。它将 src1 中的元素与 src2 中相对应的元素相乘，然后把结果赋给 dst。如果 mask 是空，那么与其中 0 元素相对应的 dst 元

素都不会因此操作而改变。OpenCV 中没有函数 cvMulS()，因为该功能已经由函数 cvScale()或 cvCvtScale()提供。

除此之外，有一件事情要记住：cvMul()执行的是元素之间的乘法。有时候，在进行矩阵相乘时，可能会错误使用 cvMul()，但这无法奏效；记住，cvGEMM()才是处理矩阵乘法的函数，而不是 cvMul()。

cvNot

```
void  cvNot(
   const  CvArr*  src,
   CvArr*       dst
);
```

函数 cvNot()会将 src 中的每一个元素的每一位取反，然后把结果赋给 dst。因此，一个值为 0x00 的 8 位图像将被映射到 0xff，而值为 0x83 的图像将被映射到 0x7c。

cvNorm

```
double  cvNorm(
   const  CvArr*  arr1,
   const  CvArr*  arr2  =  NULL,
   int   norm_type  = CV_L2,
   const  CvArr*  mask  =  NULL
   );
```

这一函数可于计算一个数组的各种范数，当为该函数提供了两个数组作为参数时，可选用各种不同的公式来计算相对的距离。在前一种情况下，计算的范数如表 3-9 所示。

表 3-9：当 arr2 =NULL 时，对于不同的 norm_type 由 cvNorm()计算范数的公式

norm_type	结果
CV_C	
CV_L1	

CV_L2

如果第二个数组参数 arr2 非空，那么范数的计算将使用不同的公式，就像两个数组之间的距离。前三种情况的计算公式如表 3-10 所示，这些范数是绝对范数；在后三个情况下，将会根据第二个数组 arr2 的幅度进行重新调整。 【69～70】

表 3-10：arr2 非空，且 norm_type 不同值时函数 cvNorm()计算范数的计算公式

norm_type	结果
CV_C	
CV_L1	
CV_L2	
CV_RELATIVE_C	
CV_ RELATIVE_L1	
CV_ RELATIVE_L2	

在所有情况下，arr1 和 arr2 必须具有相同的大小和通道数。当通道数大于 1 时，将会对所有通道一起计算范数(即是说，在表 3-9 和表 3-10 中，不仅是针对 x 和 y，也针对通道数求和)。

cvNormalize

```
cvNormalize(
  const CvArr* src,
  CvArr*       dst,
  double       a        = 1.0,
  double       b        = 0.0,
  int          norm_type = CV_L2,
  const CvArr* mask     = NULL
);
```

与许多 OpenCV 函数一样，cvNormalize()的功能比表面看更多。根据 norm_type 的不同值，图像 src 将被规范化，或者以其他方式映射到 dst 的一个特定的范围内。表 3-11 列举了 norm_type 可能出现的值。

表 3-11：函数 cvNormalize()的参数 norm_type 可能的值

norm_type	结果
CV_C	
CV_L1	

续表

norm_type	结果
CV_L2	
CV_MINMAX	映射到[a, b]的范围上

计算 C 范数时，数组 src 将被进行比例变标，使其中绝对值最大的值等于 a。当计算 L1 范数成 L2 范数时，该数组也将被缩放，如使其范数为 a。如果 norm_type 的值设置为 CV_MINMAX，那么将会对数组的所有的值进行转化，使它们线性映射到 a 和 b 之间(包括 a 和 b)。

与以前一样，如果参数 mask 非空，那么只有与掩码非 0 值对应的像素会对范数的计算有贡献，并且只有那些像素会被 cvNormalize()改变。

cvOr 和 cvOrS

```
void  cvOr(
    const  CvArr*  src1,
    const  CvArr*  src2,
    CvArr*         dst,
    const  CvArr*  mask=NULL
);
void  cvOrS(
    const  CvArr*  src,
    CvScalar       value,
```

```
    CvArr*          dst,
    const  CvArr*  mask  =  NULL
);
```

这两个函数将对数组 src1 进行按位或计算。在函数 cvOr()中，dst 中的每一个元素都是是由 src1 和 src2 中相对应的元素按位做或运算的结果。在 cvOrS()函数中，将对 src 和常量 value 进行或运算。像往常一样，如果 mask 非空，则只计算 dst 中与 mask 中非 0 元素对应的元素。

该函数支持所有的数据类型，但在 cvOr()中，src1 和 src2 必须有相同的数据类型。如果数组元素是浮点类型，则使用浮点按位表示形式。

cvReduce

```
CvSize  cvReduce(
    const  CvArr*  src,
    CvArr*          dst,
    int                dim,
    int                op = CV_REDUCE_SUM
);
```

约简是指使用一些 op 所代表的组合规则，对输入的矩阵 src 的每一行(或列)进行系统的转化，使之成为成向量 dst，直到只剩一行(或列)为止(见表 3-12)。参数 op 决定如何进行约简，总结如表 3-13 所

示。 71】

表3-12：参数op在cvReduce()中所代表的转化操作

op 的值	结果
CV_REDUCE_SUM	计算所有向量的总和
CV_REDUCE_AVG	计算所有向量的平均值
CV_REDUCE_MAX	计算所有向量中的最大值
CV_REDUCE_MIN	计算所有向量中的最小值

表3-13：参数dim在 cvReduce()中控制转化的方向

dim的值	结果
+1	合并成一行
0	合并成一列

-1 转化成对应的 dst

cvReduce()支持浮点型的多通道数组。它也允许在 dst 中使用比 src 更高精度的数据类型。这关键在于要有正确的 CV_REDUCE_SUM 和 CV_REDUCE_AVG 参数，否则那里可能有溢出和累积问题。

cvRepeat

```
void  cvRepeat(
   const  CvArr*  src,
   CvArr*         dst
);
```

这一函数是将 src 的内容复制到 dst 中，重复多次，直到 dst 没有多余的空间。具体而言，dst 相对于 src 可以是任何大小。它可能比 src 大或小，它们在大小和维数之间不需要有任何的数值关系。

cvScale

```
void  cvScale(
   const  CvArr*  src,
   CvArr*         dst,
   double         scale
);
```

从宏观上讲，函数 cvScale()实际上是 cvConvertScale()的一个宏，它会将 shift 参数设置为 0.0。因此，它可以用来重新调整数组的内容，并且可以将参数从一种数据类型转换为另一种。

cvSet 和 cvSetZero

```
void  cvSet(
   CvArr*         arr,
   CvScalar       value,
   const  CvArr*  mask   =  NULL
);
```

【72】

这些函数能将数组的所有通道的所有值设置为指定的参数 value。该 cvSet()函数接受一个可选的参数：如果提供参数，那么只有那些与参数 mask 中非 0 值对应的像素将被设置为指定的值。函数 cvSetZero()仅仅是 cvSet(0.0)别名。

cvSetIdentity

```
void  cvSetIdentity(  CvArr* arr  );
```

cvSetIdentity()将会把数组中除了行数与列数相等以外的所有元素的值都设置为 0；行数与列数相等的元素的值都设置为 1。cvSetIdentity()支持所有数据类型，甚至不要求数组的行数与列数相等。

cvSolve

```
int  cvSolve(
   const  CvArr*  src1,
   const  CvArr*  src2,
   CvArr*         dst,
   int                method  =  CV_LU
);
```

基于 cvInvert()函数 cvSolve()为求解线性方程组提供了一条捷径。它的计算公式如下：

$$C=\text{argmin}_X||A \cdot X-B||$$

其中 A 是一个由 src1 指定的方阵，B 是向量 src2，然后 C 是由 cvSolve()计算的结果，目标是寻找一个最优的向量 X。最优结果向量 X 将返回给 dst。cvInvert()支持同样的方法(前述)；不过只支持浮点类型的数据。该函数将会返回一个整型值，当返回的值是一个非 0 值的话，这表明它能够找到一个解。

应当指出的是，cvSolve()可以用来解决超定的线性方程组。超定系统将使用所谓的伪逆方法进行解决，它是使用 SVD 方法找到方程组的最小二乘解的。

cvSplit

```
void  cvSplit(
   const  CvArr*  src,
```

```
    CvArr*      dst0,
    CvArr*      dst1,
    CvArr*      dst2,
    CvArr*      dst3
);
```

有些时候处理多通道图像时不是很方便。在这种情况下，可以利用 cvSplit()分别复制每个通道到多个单通道图像。如果需要，cvSplit()函数将复制 src 的各个通道到图像 dst0，dst1，dst2 和 dst3 中。目标图像必须与源图像在大小和数据类型上相匹配，当然也应该是单通道的图像。

如果源图像少于 4 个通道(这种情况经常出现)，那么传递给 cvSplit()的不必要的目标参数可设置为 NULL。

【73】

cvSub，cvSubS 和 cvSubRS

```
void cvSub(
    const CvArr* src1,
    const CvArr* src2,
    CvArr*       dst,
    const CvArr* mask = NULL
);
void cvSubS(
    const CvArr* src,
    CvScalar     value,
    CvArr*       dst,
    const CvArr* mask = NULL
);
void cvSubRS(
    const CvArr* src,
```

```
    CvScalar        value,
    CvArr*              dst,
    const  CvArr*  mask  =  NULL
);
```

cvSub()是一个简单的减法函数，它对数组 src2 和 src1 对应的元素进行减法运算，然后把结果赋给 dst。如果数组 mask 非空，那么 dst 中元素对应位置的 mask 中的 0 元素不会因此而改变。相关的函数 cvSubS()执行相类似的功能，但它会对 src 的每一个元素减去一个常量 value。函数 cvSubRS()的功能和 cvSubS()相似，但不是src 的每个元素减去一个常量，而是常量减去的src 中的每一元素。

cvSum

```
CvScalar cvSum(
    CvArr*  arr
);                                                    【75】
```

cvSum()计算数组 arr 各个通道的所有的像素的总和。注意，函数的返回类型是 CvScalar，这意味着 cvSum()提供多通道数组计算。在这种情况下，每个通道的和都会赋给类型为 CvScalar 的返回值中相应的分量。

cvSVD

```
void  cvSVD(
    CvArr*  A,
    CvArr*  W,
    CvArr*  U     = NULL,
    CvArr*  V     =  NULL,
    int     flags =  0
);
```

奇异值分解(SVD)是将一个 m×n 的矩阵 A 按如下公式分解：

$$A=U \cdot W \cdot V^{T}$$

其中，W 是一个对角矩阵，U 和 V 分别是 m×m 和 n×n 的酉矩阵。当然，矩阵 W 也是一个 m×n 的矩阵，所以在这里的“对角线”是指任何行数和列数不相等的位置的元素值一定是 0。因为 W 必须是对角矩阵，OpenCV 允许它表示为一个 m×n 阶矩阵或一个 n×1 向量(在这种情况下，该向量将只包含对角线上的“奇异”值)。

对于函数 cvSVD()来说，U 和 V 是可选参数，如果它们的值设置为 NULL，则不会返回它们的内容。最后的参数 flags 可以是表 3-14 所示三个选项中任何一个或全部(视情况进行布尔型或计算合并)。

表 3-14：cvSVD()中 flags 参数的取值

参数	结果
CV_SVD_MODIFY_A	允许改变矩阵 A
CV_SVD_U_T	A^T
CV_SVD_V_T	返回 V_T 而不是 V

cvSVBkSb

```
void  cvSVBkSb(
   const  CvArr*  W,
   const  CvArr*  U,
   const  CvArr*  V,
   const  CvArr*  B,
   CvArr*  X,
   int    flags =  0
);
```

这个函数一般不会被直接调用。与刚才所描述的 cvSVD()一起，本函数构成了基于 SVD 的方法 cvInvert()和 cvSolve()的基础。也就是说，如果你想自己实现矩阵求逆，可用这两个函数(这可以为你节省在 cvInvert()或 cvSolve()中为临时矩阵分配的一大堆内存空

间)。

函数 cvSVBkSb()对矩阵 A 进行反向替代计算，A 已分解为矩阵 U，W 和 V(即 SVD)的结构中描述出来。矩阵 X 的结果可由如下公式计算得出：

$$X=V \cdot W^{*} \cdot U^{T} \cdot B$$

矩阵 B 是可选的，如果设置为 NULL，它将会被忽略。当$\lambda_i \geq \varepsilon$ 时矩阵 W*中的对角线元素定义如下：

$$\lambda_i^{*} = \lambda_i^{-1}$$

ε 这个值是一个奇异性阈值，一个非常小的数值，通常与 W 的对角线元素的总和成正比(即$\varepsilon \propto \sum_i \lambda_i$)。

cvTrace

```
CvScalar cvTrace( const CvArr* mat );
```

矩阵的迹是对角线元素的总和。在 OpenCV 中，该功能在函数 cvGetDiag()基础上实现，因此输入的数组不需要是方阵。同样支持多通道数组，但是数组 mat 必须是浮点类型。

cvTranspose 与 cvT

```
void cvTranspose(
  const CvArr* src,
  CvArr*       dst
);
```

cvTranspose()将 src 中每一个元素的值复制到 dst 中行号与列号相调换的位置上。这个函数不支持多通道数组；然而，如果你用两通道数组表示复数，那么记住一点：cvTranspose()不执行复共轭(依靠函数 cvXorS()是实现该功能的一个快速方法，它可以直接翻转数组中虚数部分中的符号位)。宏 cvT()是函数 cvTranspose()的缩写。

cvXor 和 cvXorS

```
void cvXor(
  const CvArr* src1,
  const CvArr* src2,
```

```
    CvArr*  dst,
    const  CvArr*  mask=NULL
);
void  cvXorS(
    const  CvArr*  src,
    CvScalar  value,
    CvArr*  dst,
    const  CvArr*  mask=NULL
);
```

【76】

这两个函数在数组 src1 上按位进行异或(XOR)运算。在函数 cvXor()中，dst 的每个元素是由 src1 和 src2 中对应的元素按位进行异或运算所得到的。在函数 cvXorS()中，是与常量 value 进行按位异或运算。再次说明，如果参数 mask 非空，则只计算与 mask 中非 0 值相对应的 dst 元素。

计算支持所有的数据类型，但 src1 和 src2 在函数 cvXor()中必须是相同的数据类型。如果数组的元素是浮点类型的，那么使用浮点数的二进制表示。

cvZero

```
void  cvZero( CvArr*  arr );
```

这个函数会将数组中的所有通道的所有元素的值都设置为 0。

绘图

我们经常需要绘制图像或者在已有的图像上方绘制一些图形。为此，OpenCV 提供了一系列的函数帮助我们绘制直线、方形和圆形等。

直线

cvLine()是绘图函数中最简单的，只需用 Bresenham 算法[Bresenham65]画一条线 ：

```
void  cvLine(
  CvArr*     array,
  CvPoint    pt1,
```

```
    CvPoint     pt2,
    CvScalar    color,
    int         thickness   = 1,
    int         connectivity = 8
);
```

cvLine()函数中的第一个属性是 CvArr*。在这里，它一般为一个图像类型的指针 IplImage*。随后两个 CvPoint 是一种简单的数据结构，它只包括整型变量 x 和 y。我们可以用 CvPoint(int x, int y)函数快速地构造一个 CvPoint 类型的变量，这样可以方便地把两个整型变量值赋给 CvPoint 数据结构。

下一个属性是 CvScalar 类型的颜色变量。CvScalar 也是一种数据结构，定义如下所示：

```
typdef struct {
  double val[4];
} CvScalar;
```

可以看出，这种结构只是四个双精度浮点型变量的集合。在这里，前三个分别代表红，绿，蓝通道；没有用到第四个(它只在适当的时候用于 alpha 通道)。一个常用的便捷宏指令是 CV_RGB(r, g, b)，该指令采用三个数字作为参数并将其封装到 CvScalar。

接下来的两个属性是可选的。thickness 是线的粗细(像素)，connectivity 被设为反走样模式，默认值为“8 连通”，这种是较为平滑不会走样的线型。也可以设置为“4 连通”，这样的话，斜线会产生重叠以致看上去过于粗重，不过画起来速度要快 得多。

cvRectangle()和 cvLine()几乎同样便捷。cvRectangle()用于画矩形。除了没有 connectivity 参数，它和 cvLine()的其他参数都是一样的。因为由此产生的矩形总是平行与 X 和 Y 轴。利用 cvRectangle()，我们只需给出两个对顶点，OpenCV 便于画出一个矩形。

```
void  cvRectangle(
  CvArr*     array,
```

```
    CvPoint     pt1,
    CvPoint     pt2,
    CvScalar    color,
    int         thickness  =  1
);
```

圆形和椭圆

画圆同样简单，其参数与前相同。

```
void  cvCircle (
    CvArr*      array,
    CvPoint     center,
    int         radius,
    CvScalar    color,
    int         thickness    =  1,
    int         connectivity  =  8
);
```

对圆形和矩阵等很多封闭图形来说，thickness 参数也可以设置为 CV_FILL，其值是-1；其结果是使用与边一样的颜色填充圆内部。

椭圆函数比 cvCircle()略微复杂一些：

```
void  cvEllipse(
    CvArr*      img,
    CvPoint     center,
    CvSize      axes,
    double      angle,
    double      start_angle,
    double      end_angle,
    CvScalar    color,
    int         thickness  =  1,
    int         line_type  =  8
```

```
);                                                    【79】
```

这里，主要的新成员是 axes 属性，其类型为 CvSize。CvSize 函数与 CvPoint 和 CvScalar 非常相似；这是一个仅包含宽度和高度的简单结构。同 CvPoint 和 CvScalar 一样，CvSize 也有一个简单的构造函数 cvSize(int height, int width)，在需要的时候返回一个 CvSize 数据。在这种情况下，height 和 width 参数分别代表椭圆的长短半轴长。

angle 是指偏离主轴的角度，从 X 轴算起，逆时针方向为正。同样，start_angle 和 end_angle 表示弧线开始和结束位置的角度。因此，一个完整的椭圆必须分别将这两个值分别设为 0° 和 360° 。

使用外接矩形是描述椭圆绘制的另一种方法：

```
void  cvEllipseBox(
  CvArr*      img,
  CvBox2D     box,
  CvScalar    color,
  int         thickness  =  1,
  int         line_type  =  8,
  int         shift      = 0
);
```

这里用到 OpenCV 的另一个结构 CvBox2D：

```
typdef  struct  {
  CvPoint2D32f    center;
  CvSize2D32f     size;
  float           angle;
} CvBox2D;
```

CvPoint2D32f 是 CvPoint 的浮点形式，同时 CvSizet2D32f 也是 CvSize 的浮点形式。这些，连同倾斜角度，可以有效地描述椭圆的外接矩形。

多边形

最后，我们有一系列绘制多边形的函数。

```
void  cvFillPoly(
  CvArr*        img,
  CvPoint**     pts,
  int*          npts,
  int           contours,
  CvScalar      color,
  int           line_type  = 8
);
void  cvFillConvexPoly(
  CvArr*        img,
  CvPoint*      pts,
  int           npts,
  CvScalar      color,
  int           line_type  = 8
);
void  cvPolyLine(
  CvArr*        img,
  CvPoint**     pts,
  int*          npts,
  int           contours,
  int           is_closed,
  CvScalar      color,
  int           thickness  = 1,
  int           line_type  = 8
);
```

【79～80】

上述三种方法依据同一思路又略有不同，其主要区别是如何描述点。
在 cvFillPoly()中，点是由 CvPoint 数组提供的。它允许 cvFillPoly()在一次调用中绘制多个多边形。同样地，npts 是由记数点构成的数组，与多边形对应。如果把变量 is_closed 设为 true，那么下一个多边形的第一个线段就会从上一多边形最后一点开始。cvFillPoly()很稳定，可以处理自相交多边形，有孔的多边形等复杂问题。然而不幸的是，函数运行起来相对缓慢。
cvFillConvexPoly()和 cvFillPoly()类似。不同的是，它一次只能画一个多边形，而且只能画凸多边形。好处是，cvFillConvexPoly()运行得更快。
第三个 cvPolyLine()，其参数与 cvFillPoly()相同，但因为只需画出多边形的边，不需处理相交情况。因此，这种函数运行速度远远超过 cvFillPoly()。

字体和文字

最后一种形式的绘图是绘制文字。当然，文字创建了一套自己的复杂格式，但是，在这类事情上，OpenCV 一如既往地更关心提供一个简单的“一招解决问题”的方案，这个方案只适用于一些简单应用，而不适用于一个稳定的和完整的应用(这将降低由其他库提供的功能)。

OpenCV 有一个主要的函数，叫 cvPutText()。这个函数可以在图像上输出一些文本。参数 text 所指向的文本将打印到图像上，参数 origin 指定文本框左下角位置，参数 color 指定文本颜色。

```
void  cvPutText(
  CvArr*              img,
  const  char*        text,
  CvPoint             origin,
  const  CvFont*      font,
  CvScalar            color
);                                                    【80～81】
```

总有一些琐事使我们的工作比预期复杂，此时是 CvFont 指针表现的机会了。

概括地说，获取 CvFont*指针的方式就是调用函数 cvInitFont()。该函数采用一组参数配置一些用于屏幕输出的基本个特定字体。如果熟悉其他环境中的 GUI 编程，势必会觉得 cvInitFont 似曾相识，但只需更少的参数。

为了建立一个可以传值给 cvPutText()的 CvFont，首先必须声明一个 CvFont 变量，然后把它传递给 cvInitFont()。

```
void  cvInitFont(
  CvFont*      font,
  int          font_face,
  double       hscale,
  double       vscale,
  double       shear      = 0,
  int          thickness  =  1,
  int          line_type  =  8
);
```

观察本函数与其他相似函数的不同。正如工作在 OpenCV 环境下的 cvCreateImage()。调用 cvInitFont()时，初始化一个已经准备好的 CvFont 结构(这意味着你创建了一个变量，并传给 cvInitFont()函数一个指向新建的变量指针)，而不是像 cvCreateImage()那样创建一个结构并返回指针。

font_face 参数列在表 3-15 中(效果在图 3-6 中画出)，它可与 CV_FONT_ITALIC 组合(通过布尔或操作)。

表3-15：可用字体(全部可选变量)

标志名称	描述
CV_FONT_HERSHEY_SIMPLEX	正常尺寸sanserif字体
CV_FONT_HERSHEY_PLAIN	小尺寸 sanserif字体
CV_FONT_HERSHEY_DUPLEX	正常尺寸sanserif，比CV_FONT_HERSHEY_SIMPLEX更复杂
CV_FONT_HERSHEY_COMPLEX	正常尺寸serif，比CV_FONT_HERSHEY_DUPLEX更复杂

续表

标志名称	描述
CV_FONT_HERSHEY_TRIPLEX	正常尺寸serif，比CV_FONT_HERSHEY_COMPLEX更复杂
CV_FONT_HERSHEY_COMPLEX_SMALL	小尺寸的CV_FONT_HERSHEY_COMPLEX
CV_FONT_HERSHEY_SCRIPT_SIMPLEX	手写风格
CV_FONT_HERSHEY_SCRIPT_COMPLEX	比CV_FONT_HERSHEY_SCRIPT_SIMPLEX更复杂的风格

【81】

图3-6：表3-15 中的 8 个字体，绘制时设置 hscale = vscale = 1.0，且每行的垂直间距为 30 像素

hscale 和 vscale 只能设为 1.0 或 0.5。字体渲染时选择全高或半高(宽度同比缩放)，绘制效果与指定字体的基本定义有关。

参数 shear 创建斜体字，如果设置为 0.0 ，字体不倾斜。当设置为 1.0 时，字符倾斜范围接近 45 度。

参数 thickness 与 Line_type 的定义与其他绘图函数相同。

数据存储

OpenCV 提供了一种机制来序列化(serialize)和去序列化(de-serialize)其各种数据类型，可以从磁盘中按 YAML 或 XML 格式读/写。在第 4 章中，我们将专门介绍存储和调用常见的对象 IplImages 的函数(cvSaveImage()和 cvLoadImage())。此外，第 4 章将讨论读/写视频的特有函数：可以从文件或者摄影机中读取数据的函数 cvGrabFrame()以及写操作函数 cvCreateVideoWriter()和 cvWriteFrame()。本小节将侧重于一般对象的永久存储：读/写矩阵、OpenCV 结构、配置与日志文件。

首先，我们从有效且简便的 OpenCV 矩阵的保存和读取功能函数开始。函数是 cvSave()和 cvLoad()。例 3-15 展示了如何保存和读取一个 5×5 的单位矩阵(对角线上是 1，其余地方都是 0)。

例 3-15：存储和读取 CvMat

```
CvMat  A  =  cvMat( 5,  5,  CV_32F,  the_matrix_data  );
cvSave(  "my_matrix.xml", &A  );
...
// to  load it  then  in  some  other program  use  …
CvMat* A1  =  (CvMat*)  cvLoad(  "my_matrix.xml" );
```

CxCore 参考手册中有整节内容都在讨论数据存储。首先要知道，在 OpenCV 中，一般的数据存储要先创建一个 CvFileStorage 结构(如例 3-16)所示，该结构将内存对象存储在一个树形结构中。然后通过使用 CV_STORAGE_READ 参数的 cvOpenFileStorage()从磁盘读取数据，创建填充该结构，也可以通过使用 CV_STORAGE_WRITE 的 cvOpenFileStorage()创建并打开 CvFileStorage 写数据，而后使用适当的数据存储函数来填充它。在磁盘上，数据的存储格式为 XML 或者 YAML。

例 3-16：CvFileStorage 结构，数据通过 CxCore 数据存储函数访问

```
typedef struct CvFileStorage
{
   ...      //  hidden fields
```

```
} CvFileStorage;
```

CvFileStorage 树内部的数据是一个层次化的数据集合，包括标量、CxCore 对象(矩阵、序列和图)以及用户定义的对象。

假如有一个配置文件或日志文件。配置文件告诉我们视频有多少帧(10)，画面大小(320×240)并且将应用一个 3×3 的色彩转换矩阵。例 3-17 展示了如何从磁盘中调出 cfg.xml 文件。

例3-17：往磁盘上写一个配置文件cfg.xml

```
CvFileStorage* fs = cvOpenFileStorage(
  "cfg.xml",
  0,
  CV_STORAGE_WRITE
);
cvWriteInt( fs, "frame_count", 10 );
cvStartWriteStruct( fs, "frame_size", CV_NODE_SEQ );
cvWriteInt( fs, 0, 320 );
cvWriteInt( fs, 0, 200 );
cvEndWriteStruct(fs);
cvWrite( fs, "color_cvt_matrix", cmatrix );
cvReleaseFileStorage( &fs );
```

请留意这个例子中的一些关键函数。我们可以定义一个整型变量通过 cvWriteInt()向结构中写数据。我们也可以使用 cvStartWriteStruct()来创建任意一个可以任选一个名称(如果无名称请输入 0 或 NULL)的结构。这个结构有两个未命名的整型变量，使用 cvEndWriteStruct()结束编写结构。如果有更多的结构体，我们用相似的方法来解决；这种结构可以进行任意深度的嵌套。最后，我们使用 cvWrite()编写处色彩转换矩阵。将这个相对复杂的矩阵程序与例 3-15 中简单的 cvSave()程序进行对比。便会发现 cvSave()是 cvWrite()在只保存一个矩阵时的快捷方式。当写完数据后，使用 cvReleaseFileStorage()释放 CvFileStorage 句柄。例 3-18 显示了 XML 格式的输出内容。

例 3-18：磁盘中的 cfg.xml 文件

```
<?xml version="1.0"?>
<opencv_storage>
<frame_count>10</frame_count>
<frame_size>320 200</frame_size>
<color_cvt_matrix type_id="opencv-matrix">
  <rows>3</rows> <cols>3</cols>
  <dt>f</dt>
  <data>...</data></color_cvt_matrix>
</opencv_storage>
```

我们将会在例 3-19 中将这个配置文件读入。

例 3-19：磁盘中的 cfg.xml 文件

```
CvFileStorage* fs = cvOpenFileStorage(
  "cfg.xml",
  0,
  CV_STORAGE_READ
);
int frame_count = cvReadIntByName(
  fs,
  0,
  "frame_count",
  5 /* default value */
);
CvSeq* s = cvGetFileNodeByName(fs,0,"frame_size")->data.seq;
int frame_width = cvReadInt(
  (CvFileNode*)cvGetSeqElem(s,0)
);
int frame_height = cvReadInt(
```

```
  (CvFileNode*)cvGetSeqElem(s,1)
);
CvMat* color_cvt_matri    = (CvMat*)  cvReadByName(
  fs,
  0,
  "color_cvt_matrix"
);
cvReleaseFileStorage( &fs  );
```

在阅读时，我们像例 3-19 中那样用 cvOpenFileStorage()打开 XML 配置文件。然后用 cvReadlntByName()来读取 frame_count，如果有没有读到的数，则赋一个默认值。在这个例子中默认的值是 5。然后使用 cvGetFileNodeByName()得到结构体 frame_size。在这里我们用 cvReadlnt()读两个无名称的整型数据。随后使用 cvReadByName()读出我们已经定义的色彩转换矩阵。将本例与例 3-15 中的 cvLoad()进行对比。如果我们只有一个矩阵要读取，那么可以使用 cvLoad()，但是如果矩阵是内嵌在一个较大的结构中，必须使用 cvRead()。最后，释放 CvFileStorage 结构。

数据函数存储与 CvFileStorage 结构相关的表单列在表 3-16 中。想了解更多细节，请查看 CxCore 手册。

表 3-16：数据存储函数

函数名称	描述
打开并释放	
cvOpenFileStorage	为读/写打开存储文件
cvReleaseFileStorage	释放存储的数据
写入	
cvStartWriteStruct	开始写入新的数据结构
cvEndWriteStruct	结束写入数据结构
cvWriteInt	写入整数型
cvWriteReal	写入浮点型
cvWriteString	写入字符串
cvWriteComment	写一个 XML 或 YAML 的注释字串
cvWrite	写一个对象，例如 CvMat
cvWriteRawData	写入多个数值
cvWriteFileNode	将文件节点写入另一个文件存储器

读取

cvGetRootFileNode	获取存储器最顶层的节点
cvGetFileNodeByName	在映图或存储器中找到相应节点
cvGetHashedKey	为名称返回一个惟一的指针
cvGetFileNode	在映图或文件存储器中找到节点
cvGetFileNodeName	返回文件的节点名
cvReadInt	读取一个无名称的整数型
cvReadIntByName	读取一个有名称的整数型
cvReadReal	读取一个无名称的浮点型

续表

函数	描述
cvReadRealByName	读取一个有名称的浮点型
cvReadString	从文件节点中寻找字符串
cvReadStringByName	找到一个有名称的文件节点并返回它
cvRead	将对象解码并返回它的指针
cvReadByName	找到对象并解码
cvReadRawData	读取多个数值
cvStartReadRawData	初始化文件节点序列的读取
cvReadRawDataSlice	读取文件节点的内容

集成性能基元

Intel 公司有一个产品叫集成性能基元(Integrated Performance Primitives，IPP)库。这个库实际上是一个有着高性能内核的工具箱，它主要用于多媒体处理以及其他计算密集型应用，可发掘处理器架构的计算能力。(其他厂商的处理器也有类似的架构，只不过规模较小。)

就像第一章所探讨的，无论从软件层面还是公司内组织层面 OpenCV 都与 IPP 有着紧密的联系。最终，OpenCV 被设计成能够自动识别 IPP 库，自动将性能较低的代码切换为 IPP 同功能的高性能代码。IPP 库允许 OpenCV 依靠它获得性能提升，IPP 依靠单指令多数据(SIMD)指令以及多核架构提升性能。

学会这些基础知识，我们就可以执行各种各样的基本任务。在本书随后的内容中，我们会发现OpenCV具有许多高级功能，几乎所有这些功能都可切换到IPP运行。图像处理经常需要对大量数据做同样的重复操作，许多是可并行处理的。因此如果任何利用并行处理方式(MMX，SSE，SSE2 等)的代码获得了巨大性能提升，您不必感到惊讶。

验证安装

用来检查 IPP 库是否已经正常安装并且检验运行是否正常的方法是使用函数 cvGetModulelnfo()，如例 3-20 所示。这个函数将检查当前 OpenCV 的版本和所有附加模块。

例 3-20：使用 cvGetModulelnfo()检查 IPP

```
char*  libraries;
char*  modules;
cvGetModuleInfo(  0,  &libraries, &modules );
printf("Libraries:  %s/nModules:  %s/n",  libraries, modules );
```

例 3-20 中的代码将打印出描述已经安装的库和模块的文本。结果如下所示：

```
Libraries  cxcore:  1.0.0
Modules:  ippcv20.dll,  ippi20.dll,  ipps20.dll, ippvm20.dll
```

此处输出的模块列表就是 OpenCV 从 IPP 库中调用的模块的信息。实际上这些模块是更底层的 CPU 库函数的封装。它的运行原理将远远超出这本书的范围，但是如果在 modules 字符串中看到了 IPP 库，那么你会感到很自信，因为所有的工作都在按您预期的进行。当然你可以运用这个信息来确认 IPP 在您的系统中能正常运行。你也许会用它来检查 IPP 是否已经安装，并根据 IPP 存在与否来自动调整代码。

小结

本章介绍了我们经常遇到的一些基本数据结构。具体说来，我们了解了 OpenCV 矩阵结构和最重要的 OpenCV 图像结构 IplImage。经过对两者的仔细研究，我们得出一个结论：矩阵结构和图像结构非常相似，如果某函数可使用 CvMat，那也可使用 IplImage，反之亦然。

练习

在下面的练习中，有些函数细节在本书中未介绍，可能需要参考与 OpenCV 一起安装的或者网上 OpenCV Wiki 中的 CxCore 手册。

1. 找到并打开.../opencv/cxcore/include/cxtypes.h.通读并找到可进行如下操作的 函数。
   a. 选取一个负的浮点数，取它的绝对值，四舍五入后，取它的极值。
   b. 产生一些随机数。

c. 创建一个浮点型CvPoint2D32f，并转换成一个整数型CvPoint。

d. 将一个CvPoint 转换成一个CvPoint2D32f。

2. 下面这个练习是帮助掌握矩阵类型。创造一个三通道二维矩阵，字节类型，大小为 100×100，并设置所有数值为0。

a. 在矩阵中使用void cvCircle(CvArr* img, CvPoint center, intradius, CvScalar color, int thickness=1, int line_type=8, int shift=0)画一个圆。

b. 使用第2章所学的方法来显示这幅图像。

3. 创建一个拥有三个通道的二维字节类型矩阵，大小为 100×100，并将所有值赋为0。通过函数 cvPtr2D 将指针指向中间的通道(“绿色”)。以(20,5)与(40,20)为顶点间画一个绿色的长方形。

4. 创建一个大小为100×100的三通道 RGB 图像。将它的元素全部置0。使用指针算法以(20，5)与(40，20)为顶点绘制一个绿色平面。

5. 练习使用感兴趣区域(ROI)。创建一个210×210的单通道图像并将其归0。在图像中使用ROI和cvSet()建立一个增长如金字塔状的数组。也就是：外部边界为 0，下一个内部边界应该为20，再下一个内部边界为 40 依此类推，直到最后内部值为 200；所有的边界应该为 10 个像素的宽度。最后显示这个 图形。

6. 为一个图像创建多个图像头。读取一个大小至少为 100×100 的图像。另创建两个图像头并设置它们的origion，depth，nChannels和widthStep属性同之前读取的图像一样。在新的图像头中，设置宽度为 20，高度为30。最后，将imageData指针分别指向像素(5，10)和(50，60)像素位置。传递这两个新的图像头给 cvNot()。最后显示最初读取的图像，在那个大图像中应该有两个矩形，矩形内的值是原始值的求反值。

7. 使用 cvCmp()创建一个掩码。加载一个真实的图像。使用 cvSplit()将图像分割成红，绿，蓝三个单通道图像。

a. 找到并显示绿图。

b. 克隆这个绿图两次(分别命名为 clone1 和clone2)。

c. 求出这个绿色平面的最大值和最小值。

d. 将clone1 的所有元素赋值为 thresh=(unsigned char)((最大值-最小值)/2.0)。

e. 将clone2所有元素赋值为 0，然后调用函数 cvCmp (green_image, clone1, clone2, CV_CMP_GE)。现在 clone2 将是一个标识绿图中值超过thresh 的掩码图像。

f. 最后，使用 cvSubS(green_image,thresh/2, green_image, clone2)函数并显示结果。

8. 创建一个结构，结构中包含一个整数，一个CvPoint和一个CvRect；称结构为"my_struct"。

a. 写两个函数：void write_my_struct(CvFileStorage * fs, const char * name, my_struct *ms) 和 void read_my_struct (CvFileStorage* fs, CvFileNode* ms_node, my_struct* ms)。用它们读/写 my_struct。

b. 创建一个元素为 my_struct 结构体且长度为10 的数组，并将数组写入磁盘和从磁盘读入内存。