摘要：本文旨在帮助读者快速入门 OpenCV，而无需阅读冗长的参考手册。掌握了 OpenCV 的以下基础知识后，有需要的话再查阅相关的参考手册。

---

**目录**

[隐藏]

[编辑]

# 一、简介

[编辑]

## 1、OpenCV 的特点

[编辑]

### （1） 总体描述

OpenCV 是一个基于 C/C++语言的开源图像处理函数库

其代码都经过优化，可用于实时处理图像

具有良好的可移植性

可以进行图像/视频载入、保存和采集的常规操作

具有低级和高级的应用程序接口（API）

提供了面向 Intel IPP 高效多媒体函数库的接口，可针对你使用的 Intel CPU 优化代码，提高程序性能（译注：OpenCV 2.0 版的代码已显着优化，无需 IPP 来提升性能，故 2.0 版不再提供 IPP 接口）

[编辑]

### （2） 功能

图像数据操作（内存分配与释放，图像复制、设定和转换）

Image data manipulation (allocation, release, copying, setting, conversion).

图像/视频的输入输出（支持文件或摄像头的输入，图像/视频文件的输出）

Image and video I/O (file and camera based input, image/video file output).

矩阵/向量数据操作及线性代数运算（矩阵乘积、矩阵方程求解、特征值、奇异值分解）

Matrix and vector manipulation and linear algebra routines (products, solvers, eigenvalues, SVD).

支持多种动态数据结构（链表、队列、数据集、树、图）

Various dynamic data structures (lists, queues, sets, trees, graphs).

基本图像处理（去噪、边缘检测、角点检测、采样与插值、色彩变换、形态学处理、直方图、图像金字塔结构）

Basic image processing (filtering, edge detection, corner detection, sampling and interpolation, color conversion, morphological operations, histograms, image pyramids).

结构分析（连通域/分支、轮廓处理、距离转换、图像矩、模板匹配、霍夫变换、多项式逼近、曲线拟合、椭圆拟合、狄劳尼三角化）

Structural analysis (connected components, contour processing, distance transform, various moments, template matching, Hough transform, polygonal approximation, line fitting, ellipse fitting, Delaunay triangulation).

摄像头定标（寻找和跟踪定标模式、参数定标、基本矩阵估计、单应矩阵估计、立体视觉匹配）

Camera calibration (finding and tracking calibration patterns, calibration, fundamental matrix estimation, homography estimation, stereo correspondence).

运动分析（光流、动作分割、目标跟踪）

Motion analysis (optical flow, motion segmentation, tracking).

目标识别（特征方法、HMM 模型）

Object recognition (eigen-methods, HMM).

基本的 GUI（显示图像/视频、键盘/鼠标操作、滑动条）

Basic GUI (display image/video, keyboard and mouse handling, scroll-bars).

图像标注（直线、曲线、多边形、文本标注）

Image labeling (line, conic, polygon, text drawing)

[编辑]

### （3） OpenCV 模块

cv – 核心函数库

cvaux – 辅助函数库

cxcore – 数据结构与线性代数库

highgui – GUI 函数库

ml – 机器学习函数库

[编辑]

## 2、有用的学习资源

[编辑]

**（1） 参考手册：**

<opencv-root>/docs/index.htm （译注：在你的 OpenCV 安装目录<opencv-root>内）

[编辑]

**（2） 网络资源：**

官方网站: http://www.intel.com/technology/computing/opencv/

软件下载: http://sourceforge.net/projects/opencvlibrary/

[编辑]

**（3） 书籍：**

Open Source Computer Vision Library

by Gary R. Bradski, Vadim Pisarevsky, and Jean-Yves Bouguet, Springer, 1st ed. (June, 2006).

chenyusiyuan: 补充以下书籍

Learning OpenCV - Computer Vision with the OpenCV Library

by Gary Bradski & Adrian Kaehler, O'Reilly Media, 1 st ed. (September, 2008).

OpenCV 教程——基础篇

作者：刘瑞祯 于仕琪，北京航空航天大学出版社，出版日期：200706

[编辑]

**（4） 视频处理例程(在 <opencv-root>/samples/c/):**

颜色跟踪: camshiftdemo

点跟踪: lkdemo

动作分割: motempl

边缘检测: laplace

[编辑]

**（5） 图像处理例程 (在 <opencv-root>/samples/c/):**

边缘检测: edge

图像分割: pyramid_segmentation

形态学: morphology

直方图: demhist

距离变换: distrans

椭圆拟合: fitellipse

[编辑]

## 3、OpenCV 命名规则

[编辑]

**（1） 函数名:**

```
cvActionTargetMod(...)
Action = 核心功能（core functionality） (e.g. set, create)
Target = 目标图像区域（target image area） (e.g. contour, polygon)
Mod    = （可选的）调整语（optional modifiers） (e.g. argument type)
```

[编辑]

**（2） 矩阵数据类型:**

```
CV_<bit_depth>(S|U|F)C<number_of_channels>
 S = 符号整型
U = 无符号整型
F = 浮点型
E.g.: CV_8UC1 是指一个 8 位无符号整型单通道矩阵,
         CV_32FC2 是指一个 32 位浮点型双通道矩阵.
```

[编辑]

**（3） 图像数据类型:**

```
IPL_DEPTH_<bit_depth>(S|U|F)
E.g.: IPL_DEPTH_8U 图像像素数据是 8 位无符号整型.
      IPL_DEPTH_32F 图像像素数据是 32 位浮点型.
```

[编辑]

**（4） 头文件:**

```
#include <cv.h>
   #include <cvaux.h>
   #include <highgui.h>
   #include <ml.h>

   #include <cxcore.h>   // 一般不需要，cv.h 内已包含该头文件
```

[编辑]

## 4、编译建议

[编辑]

**（1） Linux:**

```
g++ hello-world.cpp -o hello-world \
    -I /usr/local/include/opencv -L /usr/local/lib  \
    -lm -lcv -lhighgui -lcvaux
```

[编辑]

**（2） Windows:**

在 Visual Studio 的'选项'和'项目'中设置好 OpenCV 相关文件的路径。

[编辑]

## 5、C 例程

```
////////////////////////////////////////////////////////////////
//////////////
//
// hello-world.cpp
//
// 该程序从文件中读入一幅图像，将之反色，然后显示出来.
//
////////////////////////////////////////////////////////////////
//////////////
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <cv.h>
#include <highgui.h>

int main(int argc, char *argv[])
{
  IplImage* img = 0;
  int height,width,step,channels;
  uchar *data;
  int i,j,k;

  if(argc<2){
    printf("Usage: main <image-file-name>\n\7");
```

```
    exit(0);
  }
```

```
  // load an image
  img=cvLoadImage(argv[1]);
  if(!img){
    printf("Could not load image file: %s\n",argv[1]);
    exit(0);
  }

  // get the image data
  height    = img->height;
  width     = img->width;
  step      = img->widthStep;
  channels  = img->nChannels;
  data      = (uchar *)img->imageData;
  printf("Processing a %dx%d image with %d
channels\n",height,width,channels);

  // create a window
  cvNamedWindow("mainWin", CV_WINDOW_AUTOSIZE);
  cvMoveWindow("mainWin", 100, 100);

  // invert the image

  // 相当于 cvNot(img);

  for(i=0;i<height;i++) for(j=0;j<width;j++)
for(k=0;k<channels;k++)
```

```
data[i*step+j*channels+k]=255-data[i*step+j*channels+k];
```

```
  // show the image
  cvShowImage("mainWin", img );

  // wait for a key
  cvWaitKey(0);

  // release the image
  cvReleaseImage(&img );
  return 0;
```

```
}
```

[编辑]

## 二、GUI 指令

[编辑]

### 1、窗口管理

[编辑]

**（1） 创建和定位一个新窗口:**

```
cvNamedWindow("win1", CV_WINDOW_AUTOSIZE);
  cvMoveWindow("win1", 100, 100); // offset from the UL corner
of the screen
```

[编辑]

**（2） 载入图像:**

```
IplImage* img=0;
  img=cvLoadImage(fileName);
  if(!img) printf("Could not load image file:
%s\n",fileName);
```

[编辑]

**（3） 显示图像:**

```
cvShowImage("win1",img);
```

该函数可以显示彩色或灰度的字节型/浮点型图像。字节型图像像素值范围为[0-255]；浮点型图像像素值范围为[0-1]。彩色图像的三色元素按 BGR（蓝-红-绿）顺序存储。

[编辑]

**（4） 关闭窗口:**

```
cvDestroyWindow("win1");
```

[编辑]

**（5） 改变窗口大小:**

```
cvResizeWindow("win1",100,100); // new width/heigh in pixels
```

[编辑]

## 2、输入处理

[编辑]

**（1） 处理鼠标事件:**

定义一个鼠标处理程序:

```
void mouseHandler(int event, int x, int y, int flags, void*
param)
  {
    switch(event){
      case CV_EVENT_LBUTTONDOWN:

        if(flags & CV_EVENT_FLAG_CTRLKEY)

          printf("Left button down with CTRL pressed\n");
        break;

      case CV_EVENT_LBUTTONUP:
        printf("Left button up\n");
        break;
    }
  }
 x,y:   相对于左上角的像素坐标
 event: CV_EVENT_LBUTTONDOWN,   CV_EVENT_RBUTTONDOWN,
CV_EVENT_MBUTTONDOWN,
        CV_EVENT_LBUTTONUP,     CV_EVENT_RBUTTONUP,
CV_EVENT_MBUTTONUP,
        CV_EVENT_LBUTTONDBLCLK, CV_EVENT_RBUTTONDBLCLK,
CV_EVENT_MBUTTONDBLCLK,
        CV_EVENT_MOUSEMOVE:
 flags: CV_EVENT_FLAG_CTRLKEY, CV_EVENT_FLAG_SHIFTKEY,
CV_EVENT_FLAG_ALTKEY,
        CV_EVENT_FLAG_LBUTTON, CV_EVENT_FLAG_RBUTTON,
CV_EVENT_FLAG_MBUTTON
```

注册该事件处理程序:

```
mouseParam=5;
```

```
  cvSetMouseCallback("win1",mouseHandler,&mouseParam);
```

[编辑]

**（2） 处理键盘事件:**

实际上对于键盘输入并没有专门的事件处理程序.

按一定间隔检测键盘输入（适用于循环体中）:

```
int key;
  key=cvWaitKey(10); // wait 10ms for input
```

中止程序等待键盘输入:

```
int key;
  key=cvWaitKey(0); // wait indefinitely for input
```

键盘输入的循环处理程序:

```
while(1){
    key=cvWaitKey(10);
    if(key==27) break;

    switch(key){
      case 'h':
        ...
        break;
      case 'i':
        ...
        break;
    }
  }
```

[编辑]

**（3） 处理滑动条事件:**

定义一个滑动条处理程序:

```
void trackbarHandler(int pos)
  {
```

```
    printf("Trackbar position: %d\n",pos);
  }
```

注册该事件处理程序:

```
int trackbarVal=25;
 int maxVal=100;
 cvCreateTrackbar("bar1", "win1", &trackbarVal ,maxVal ,
trackbarHandler);
```

获取当前的滑动条位置:

```
int pos = cvGetTrackbarPos("bar1","win1");
```

设置滑动条位置:

```
cvSetTrackbarPos("bar1", "win1", 25);
```

---

[编辑]

## 三、OpenCV 的基本数据结构

---

（译注：OpenCV 1.1、1.2 或 2.0 版本中各数据结构的结构体元素有所调整，以下仅作参考）

[编辑]

### 1、图像数据结构

[编辑]

**（1） IPL 图像:**

```
IplImage
  |-- int  nChannels;     // 颜色通道数目 (1,2,3,4)

  |-- int  depth;        // 像素的位深:
  |                    //   IPL_DEPTH_8U, IPL_DEPTH_8S,
  |                    //   IPL_DEPTH_16U,IPL_DEPTH_16S,
  |                    //   IPL_DEPTH_32S,IPL_DEPTH_32F,
  |                    //   IPL_DEPTH_64F
```

```
  |-- int  width;         // 图像宽度（像素为单位）
  |-- int  height;        // 图像高度
  |-- char* imageData;    // 图像数据指针
  |                      // 注意彩色图像按 BGR 顺序存储数据
  |-- int  dataOrder;     // 0 - 将像素点不同通道的值交错排在一起，形成单一像素平面
  |                      // 1 - 把所有像素同通道值排在一起，形成若干个通道平面，再把平面排列起来
  |                      // cvCreateImage 只能创建像素交错排列式的图像
  |-- int  origin;        // 0 – 像素原点为左上角,
  |                      // 1 – 像素原点为左下角 (Windows bitmaps style)
  |-- int  widthStep;     // 相邻行的同列点之间的字节数
  |-- int  imageSize;     // 图像的大小（字节为单位） = height*widthStep
  |-- struct _IplROI *roi;// 图像的感兴趣区域（ROI）. ROI 非空时对图像的
  |                      // 处理仅限于 ROI 区域.
  |-- char *imageDataOrigin; // 图像数据未对齐时的数据原点指针
  |                         // (需要正确地重新分配图像内存 )
  |                         // （needed for correct image deallocation）
```

```
  |-- int  align;        // 图像数据的行对齐: 4 or 8 byte
alignment
  |                    // OpenCV 中无此项，采用 widthStep 代替
  |-- char colorModel[4]; // 颜色模型 – OpenCV 中忽略此项
```

[编辑]

## 2、矩阵与向量

[编辑]

**（1） 矩阵:**

```
CvMat                    // 2D 矩阵
  |-- int   type;         // 元素类型
(uchar,short,int,float,double) 与标志
  |-- int   step;         // 整行长度字节数
  |-- int   rows, cols;    // 行、列数
  |-- int   height, width; // 矩阵高度、宽度，与 rows、cols 对应
  |-- union data;
      |-- uchar*  ptr;     // data pointer for an unsigned char
matrix
      |-- short*  s;       // data pointer for a short matrix
      |-- int*    i;       // data pointer for an integer matrix
      |-- float*  fl;      // data pointer for a float matrix
      |-- double* db;      // data pointer for a double matrix
CvMatND                  // N-维矩阵
  |-- int   type;         // 元素类型
(uchar,short,int,float,double) 与标志
  |-- int   dims;         // 矩阵维数
```

```
  |-- union data;
  |   |-- uchar*  ptr;     // data pointer for an unsigned char
matrix
  |   |-- short*  s;       // data pointer for a short matrix
  |   |-- int*    i;      // data pointer for an integer matrix
  |   |-- float*  fl;      // data pointer for a float matrix
  |   |-- double* db;      // data pointer for a double matrix
  |
```

```
  |-- struct dim[];       // 各维信息

      |-- size;           // 元素数目

      |-- step;           // 元素间距（字节为单位）

CvSparseMat // N-维稀疏矩阵
```

[编辑]

**（2） 一般矩阵:**

```
CvArr*     // 仅作为函数定义的参数使用，

           // 表明函数可以接受不同类型的矩阵作为参数，

           // 例如：IplImage*, CvMat* 甚至是 CvSeq*.

           // 矩阵的类型通过矩阵头的前 4 个字节信息来确定
```

[编辑]

**（3） 标量:**

```
CvScalar

  |-- double val[4]; //4D 向量
```

初始化函数:

```
CvScalar s = cvScalar(double val0, double val1=0, double
val2=0, double val3=0);
// Example:
```

```
CvScalar s = cvScalar(20.0);
s.val[0]=10.0;
```

注意该初始化函数的函数名与对应的结构体名称几乎同名，差别仅在于函数名第一个字母是小写的，而结构体名第一个字母是大写的。它并不是一个 C++ 构造函数。（译注：类似的还有 cvMat 与 CvMat、cvPoint 与 CvPoint 等等）

[编辑]

### 3、其它结构类型

[编辑]

**（1） 点:**

```
CvPoint     p = cvPoint(int x, int y);
CvPoint2D32f p = cvPoint2D32f(float x, float y);
CvPoint3D32f p = cvPoint3D32f(float x, float y, float z);
//E.g.:
p.x=5.0;
p.y=5.0;
```

[编辑]

**（2） 矩形框大小（以像素为精度）:**

```
CvSize      r = cvSize(int width, int height);
CvSize2D32f  r = cvSize2D32f(float width, float height);
```

[编辑]

**（3） 矩形框的偏置和大小:**

```
CvRect      r = cvRect(int x, int y, int width, int height);
```

---

[编辑]

## 四、图像处理

---

[编辑]

### 1、图像的内存分配与释放

[编辑]

**（1） 分配内存给一幅新图像:**

```
IplImage* cvCreateImage(CvSize size, int depth, int
channels);
```

size: cvSize(width,height);

depth: 像素深度: IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16U,

```
   IPL_DEPTH_16S, IPL_DEPTH_32S, IPL_DEPTH_32F, IPL_DEPTH_64F
```

channels: 像素通道数. Can be 1, 2, 3 or 4.

各通道是交错排列的. 一幅彩色图像的数据排列格式如下：

b0 g0 r0 b1 g1 r1 ...

示例:

```
// Allocate a 1-channel byte image
IplImage*
img1=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);

// Allocate a 3-channel float image
IplImage*
img2=cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);
```

[编辑]

**（2） 释放图像:**

```
IplImage*
img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
cvReleaseImage(&img);
```

[编辑]

**（3） 复制图像:**

```
IplImage*
img1=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
IplImage* img2;

img2=cvCloneImage(img1);  // 注意通过 cvCloneImage 得到的图像

                          // 也要用 cvReleaseImage 释放，否则容易产生
内存泄漏
```

[编辑]

**（4） 设置/获取感兴趣区域 ROI:**

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

大多数 OpenCV 函数都支持 ROI.

[编辑]

**（5） 设置/获取感兴趣通道 COI:**

```
void cvSetImageCOI(IplImage* image, int coi); // 0=all
int cvGetImageCOI(const IplImage* image);
```

大多数 OpenCV 函数不支持 COI.

[编辑]

## 2、图像读写

[编辑]

**（1） 从文件中读入图像:**

```
IplImage* img=0;
  img=cvLoadImage(fileName);
  if(!img) printf("Could not load image file:
%s\n",fileName);
 支持的图像格式: BMP, DIB, JPEG, JPG, JPE, PNG, PBM, PGM, PPM,
                        SR, RAS, TIFF, TIF
```

OpenCV 默认将读入的图像强制转换为一幅三通道彩色图像. 不过可以按以下方法修改读入方式:

```
img=cvLoadImage(fileName,flag);
 flag: >0 将读入的图像强制转换为一幅三通道彩色图像
       =0 将读入的图像强制转换为一幅单通道灰度图像
       <0 读入的图像通道数与所读入的文件相同.
```

[编辑]

**（2） 保存图像:**

```
if(!cvSaveImage(outFileName,img)) printf("Could not save:
%s\n", outFileName);
```

保存的图像格式由 outFileName 中的扩展名确定.

[编辑]

## 3、访问图像像素

[编辑]

**（1） 假设你要访问第 k 通道、第 i 行、第 j 列的像素。**

[编辑]

**（2） 间接访问: (通用，但效率低，可访问任意格式的图像)**

对于单通道字节型图像:

```
IplImage*
img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
CvScalar s;
s=cvGet2D(img,i,j); // get the (i,j) pixel value
printf("intensity=%f\n",s.val[0]);
s.val[0]=111;
```

```
cvSet2D(img,i,j,s); // set the (i,j) pixel value
```

对于多通道字节型/浮点型图像:

```
IplImage*
img=cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);
CvScalar s;
s=cvGet2D(img,i,j); // get the (i,j) pixel value
printf("B=%f, G=%f, R=%f\n",s.val[0],s.val[1],s.val[2]);
s.val[0]=111;
s.val[1]=111;
s.val[2]=111;
cvSet2D(img,i,j,s); // set the (i,j) pixel value
```

[编辑]

**（3） 直接访问: (效率高，但容易出错)**

对于单通道字节型图像:

```
IplImage*
img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
((uchar *)(img->imageData + i*img->widthStep))[j]=111;
```

对于多通道字节型图像:

```
IplImage*
img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,3);
((uchar *)(img->imageData +
i*img->widthStep))[j*img->nChannels + 0]=111; // B
((uchar *)(img->imageData +
i*img->widthStep))[j*img->nChannels + 1]=112; // G
((uchar *)(img->imageData +
i*img->widthStep))[j*img->nChannels + 2]=113; // R
```

对于多通道浮点型图像:

```
IplImage*
img=cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);
((float *)(img->imageData +
i*img->widthStep))[j*img->nChannels + 0]=111; // B
((float *)(img->imageData +
i*img->widthStep))[j*img->nChannels + 1]=112; // G
((float *)(img->imageData +
i*img->widthStep))[j*img->nChannels + 2]=113; // R
```

[编辑]

**（4） 基于指针的直接访问:(简单高效)**

对于单通道字节型图像:

```
IplImage* img  =
cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
int height    = img->height;
int width     = img->width;

int step      = img->widthStep/sizeof(uchar);

uchar* data    = (uchar *)img->imageData;
data[i*step+j] = 111;
```

对于多通道字节型图像:

```
IplImage* img  =
cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,3);
int height    = img->height;
int width     = img->width;
```

```
int step      = img->widthStep/sizeof(uchar);
int channels   = img->nChannels;
uchar* data    = (uchar *)img->imageData;
data[i*step+j*channels+k] = 111;
```

对于多通道浮点型图像（假设图像数据采用 4 字节（32 位）行对齐方式）：

```
IplImage* img  =
cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);
int height    = img->height;
int width     = img->width;
int step      = img->widthStep/sizeof(float);
int channels   = img->nChannels;
float * data    = (float *)img->imageData;
data[i*step+j*channels+k] = 111;
```

[编辑]

**（5） 基于 c++ wrapper 的直接访问:（更简单高效）**

首先定义一个 c++ wrapper ‘Image’，然后基于 Image 定义不同类型的图像:

```
template<class T> class Image
{
  private:
  IplImage* imgp;
  public:
  Image(IplImage* img=0) {imgp=img;}

  ~Image(){imgp=0;}

  void operator=(IplImage* img) {imgp=img;}
  inline T* operator[](const int rowIndx) {
    return ((T *)(imgp->imageData +
rowIndx*imgp->widthStep));}
};

typedef struct{
  unsigned char b,g,r;
} RgbPixel;

typedef struct{
  float b,g,r;
} RgbPixelFloat;
```

```
typedef Image<RgbPixel>      RgbImage;
typedef Image<RgbPixelFloat>  RgbImageFloat;
typedef Image<unsigned char>  BwImage;
typedef Image<float>         BwImageFloat;
```

对于单通道字节型图像:

```
IplImage*
img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,1);
BwImage imgA(img);
imgA[i][j] = 111;
```

对于多通道字节型图像:

```
IplImage*
img=cvCreateImage(cvSize(640,480),IPL_DEPTH_8U,3);
RgbImage  imgA(img);
imgA[i][j].b = 111;
imgA[i][j].g = 111;
imgA[i][j].r = 111;
```

对于多通道浮点型图像:

```
IplImage*
img=cvCreateImage(cvSize(640,480),IPL_DEPTH_32F,3);
RgbImageFloat imgA(img);
imgA[i][j].b = 111;
imgA[i][j].g = 111;
imgA[i][j].r = 111;
```

[编辑]

## 4、图像转换

[编辑]

**（1） 字节型图像的灰度-彩色转换:**

```
cvConvertImage(src, dst, flags=0);
 src = float/byte grayscale/color image
```

```
dst = byte grayscale/color image
flags = CV_CVTIMG_FLIP     (垂直翻转图像)
        CV_CVTIMG_SWAP_RB  (置换 R 和 B 通道)
```

[编辑]

**（2） 彩色图像->灰度图像:**

```
// Using the OpenCV conversion:
cvCvtColor(cimg,gimg,CV_BGR2GRAY); // cimg -> gimg
```

```
// Using a direct conversion:
```

```
for(i=0;i<cimg->height;i++) for(j=0;j<cimg->width;j++)
  gimgA[i][j]= (uchar)(cimgA[i][j].b*0.114 +
                       cimgA[i][j].g*0.587 +
                       cimgA[i][j].r*0.299);
```

[编辑]

**（3） 不同彩色空间之间的转换:**

```
cvCvtColor(src,dst,code); // src -> dst
 code    = CV_<X>2<Y>
 <X>/<Y> = RGB, BGR, GRAY, HSV, YCrCb, XYZ, Lab, Luv, HLS
```

e.g.: CV_BGR2GRAY, CV_BGR2HSV, CV_BGR2Lab

[编辑]

## 5、绘图指令

[编辑]

**（1） 绘制矩形:**

```
// 在点 (100,100) 和 (200,200) 之间绘制一矩形，边线用红色、宽度为
1
cvRectangle(img, cvPoint(100,100), cvPoint(200,200),
cvScalar(255,0,0), 1);
```

[编辑]

**（2） 绘制圆形:**

```
// 圆心为(100,100)、半径为20. 圆周绿色、宽度为1
cvCircle(img, cvPoint(100,100), 20, cvScalar(0,255,0), 1);
```

[编辑]

**（3） 绘制线段:**

```
// 在 (100,100) 和 (200,200) 之间、线宽为 1 的绿色线段
cvLine(img, cvPoint(100,100), cvPoint(200,200),
cvScalar(0,255,0), 1);
```

[编辑]

**（4） 绘制一组线段:**

```
CvPoint  curve1[]={10,10,  10,100,  100,100,  100,10};
CvPoint  curve2[]={30,30,  30,130,  130,130,  130,30,
150,10};
CvPoint* curveArr[2]={curve1, curve2};
int      nCurvePts[2]={4,5};
int      nCurves=2;
int      isCurveClosed=1;
int      lineWidth=1;

cvPolyLine(img,curveArr,nCurvePts,nCurves,isCurveClosed,c
vScalar(0,255,255),lineWidth);
```

[编辑]

**（5） 绘制一组填充颜色的多边形:**

```
cvFillPoly(img,curveArr,nCurvePts,nCurves,cvScalar(0,255,
255));
```

[编辑]

**（6） 文本标注:**

```
CvFont font;
double hScale=1.0;
double vScale=1.0;
int    lineWidth=1;
cvInitFont(&font,CV_FONT_HERSHEY_SIMPLEX|CV_FONT_ITALIC,
hScale,vScale,0,lineWidth);
```

```
cvPutText (img,"My comment",cvPoint(200,400), &font,
cvScalar(255,255,0));
```

其它可用的字体类型有: CV_FONT_HERSHEY_SIMPLEX, CV_FONT_HERSHEY_PLAIN, CV_FONT_HERSHEY_DUPLEX, CV_FONT_HERSHEY_COMPLEX, CV_FONT_HERSHEY_TRIPLEX, CV_FONT_HERSHEY_COMPLEX_SMALL, CV_FONT_HERSHEY_SCRIPT_SIMPLEX, CV_FONT_HERSHEY_SCRIPT_COMPLEX,

---

[编辑]

## 五、矩阵处理

[编辑]

### 1、矩阵的内存分配与释放

[编辑]

**（1） 总体上:**

OpenCV 使用 C 语言来进行矩阵操作。不过实际上有很多 C++语言的替代方案可以更高效地完成。

在 OpenCV 中向量被当做是有一个维数为 1 的 N 维矩阵.

矩阵按行-行方式存储，每行以 4 字节（32 位）对齐.

[编辑]

**（2） 为新矩阵分配内存:**

```
CvMat* cvCreateMat(int rows, int cols, int type);
 type: 矩阵元素类型.
 按 CV_<bit_depth>(S|U|F)C<number_of_channels> 方式指定.  例如:
CV_8UC1 、CV_32SC2.
 示例:
CvMat* M = cvCreateMat(4,4,CV_32FC1);
```

[编辑]

**（3） 释放矩阵内存:**

```
CvMat* M = cvCreateMat(4,4,CV_32FC1);
cvReleaseMat(&M);
```

[编辑]

**（4） 复制矩阵:**

```
CvMat* M1 = cvCreateMat(4,4,CV_32FC1);
CvMat* M2;
M2=cvCloneMat(M1);
```

[编辑]

**（5） 初始化矩阵:**

```
double a[] = { 1,  2,  3,  4,
               5,  6,  7,  8,

               9, 10, 11, 12 };

CvMat Ma=cvMat(3, 4, CV_64FC1, a);


//等价于:

CvMat Ma;
cvInitMatHeader(&Ma, 3, 4, CV_64FC1, a);
```

[编辑]

**（6） 初始化矩阵为单位矩阵:**

```
CvMat* M = cvCreateMat(4,4,CV_32FC1);
cvSetIdentity(M); // does not seem to be working properly
```

[编辑]

## 2、访问矩阵元素

[编辑]

**（1） 假设需要访问一个 2D 浮点型矩阵的第（i, j）个单元.**

[编辑]

**（2） 间接访问:**

```
cvmSet(M,i,j,2.0); // Set M(i,j)

t = cvmGet(M,i,j); // Get M(i,j)
```

[编辑]

**（3） 直接访问（假设矩阵数据按 4 字节行对齐）:**

```
CvMat* M    = cvCreateMat(4,4,CV_32FC1);
int n       = M->cols;
```

```
float *data = M->data.fl;
data[i*n+j] = 3.0;
```

[编辑]

**（4） 直接访问（当数据的行对齐可能存在间隙时 possible alignment gaps）：**

```
CvMat* M    = cvCreateMat(4,4,CV_32FC1);
int   step  = M->step/sizeof(float);
float *data = M->data.fl;
(data+i*step)[j] = 3.0;
```

[编辑]

**（5） 对于初始化后的矩阵进行直接访问：**

```
double a[16];

CvMat Ma = cvMat(3, 4, CV_64FC1, a);

a[i*4+j] = 2.0; // Ma(i,j)=2.0;
```

[编辑]

### 3、矩阵/向量运算

[编辑]

**（1） 矩阵之间的运算：**

```
CvMat *Ma, *Mb, *Mc;
cvAdd(Ma, Mb, Mc);      // Ma+Mb   -> Mc
cvSub(Ma, Mb, Mc);      // Ma-Mb   -> Mc
cvMatMul(Ma, Mb, Mc);   // Ma*Mb   -> Mc
```

[编辑]

**（2） 矩阵之间的元素级运算：**

```
CvMat *Ma, *Mb, *Mc;
cvMul(Ma, Mb, Mc);      // Ma.*Mb  -> Mc
cvDiv(Ma, Mb, Mc);      // Ma./Mb  -> Mc
cvAddS(Ma, cvScalar(-10.0), Mc); // Ma.-10 -> Mc
```

[编辑]

**（3） 向量乘积：**

```
double va[] = {1, 2, 3};
```

```
double vb[] = {0, 0, 1};
double vc[3];

CvMat Va=cvMat(3, 1, CV_64FC1, va);
CvMat Vb=cvMat(3, 1, CV_64FC1, vb);
CvMat Vc=cvMat(3, 1, CV_64FC1, vc);


double res=cvDotProduct(&Va,&Vb); // 向量点乘:   Va . Vb ->
res
cvCrossProduct(&Va, &Vb, &Vc);    // 向量叉乘:   Va x Vb -> Vc
```

注意在进行叉乘运算时，Va, Vb, Vc 必须是仅有 3 个元素的向量.

[编辑]

**（4） 单一矩阵的运算:**

```
CvMat *Ma, *Mb;
cvTranspose(Ma, Mb);      // 转置：transpose(Ma) -> Mb (注意
转置阵不能返回给 Ma 本身)
CvScalar t = cvTrace(Ma); // 迹：trace(Ma) -> t.val[0]
double d = cvDet(Ma);     // 行列式：det(Ma) -> d
cvInvert(Ma, Mb);         // 逆矩阵：inv(Ma) -> Mb
```

[编辑]

**（5） 非齐次线性方程求解:**

```
CvMat* A  = cvCreateMat(3,3,CV_32FC1);
CvMat* x  = cvCreateMat(3,1,CV_32FC1);
CvMat* b  = cvCreateMat(3,1,CV_32FC1);
cvSolve(&A, &b, &x);    // solve (Ax=b) for x
```

[编辑]

**（6） 特征值与特征向量 (矩阵为方阵):**

```
CvMat* A  = cvCreateMat(3,3,CV_32FC1);
CvMat* E  = cvCreateMat(3,3,CV_32FC1);
```

```
CvMat* l  = cvCreateMat(3,1,CV_32FC1);

cvEigenVV(&A, &E, &l);  // l = A 的特征值(递减顺序)

                        // E = 对应的特征向量 (行向量)
```

（7） 奇异值分解（SVD）:====

```
CvMat* A  = cvCreateMat(3,3,CV_32FC1);
CvMat* U  = cvCreateMat(3,3,CV_32FC1);
CvMat* D  = cvCreateMat(3,3,CV_32FC1);
CvMat* V  = cvCreateMat(3,3,CV_32FC1);
cvSVD(A, D, U, V, CV_SVD_U_T|CV_SVD_V_T); // A = U D V^T
```

标志位使矩阵 U 或 V 按转置形式返回 (若不转置可能运算出错).

---

[编辑]

## 六、视频处理

---

[编辑]

### 1、从视频流中捕捉一帧画面

[编辑]

**（1） OpenCV 支持从摄像头或视频文件（AVI 格式）中捕捉帧画面.**

[编辑]

**（2） 初始化一个摄像头捕捉器:**

```
CvCapture* capture = cvCaptureFromCAM(0); // capture from
video device #0
```

[编辑]

**（3） 初始化一个视频文件捕捉器:**

```
CvCapture* capture = cvCaptureFromAVI("infile.avi");
```

[编辑]

**（4） 捕捉一帧画面:**

```
IplImage* img = 0;
```

```
if(!cvGrabFrame(capture)){              // capture a frame
  printf("Could not grab a frame\n\7");
  exit(0);
}
img=cvRetrieveFrame(capture);           // retrieve the
captured frame
```

若要从多个摄像头中同步捕捉画面，则须首先从每个摄像头中抓取一帧，紧接着要将被捕捉的帧画面恢复到一个 IplImage*型图像中。（译注：这一过程其实可以用 cvQueryFrame() 函数一步完成）

[编辑]

**（5） 释放视频流捕捉器:**

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

注意由视频流捕捉器得到的图像是由捕捉器分配和释放内存的，不需要单独对图像进行释放内存的操作。

[编辑]

## 2、获取/设置视频流信息

[编辑]

**（1） 获取视频流设备信息:**

```
cvQueryFrame(capture); // 在读取视频流信息前，要先执行此操作
int frameH    = (int) cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_HEIGHT);
int frameW    = (int) cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_WIDTH);
int fps       = (int) cvGetCaptureProperty(capture,
CV_CAP_PROP_FPS);
int numFrames = (int) cvGetCaptureProperty(capture,
CV_CAP_PROP_FRAME_COUNT);
```

统计总帧数仅对视频文件有效，但似乎不太准确（译注：也许 OpenCV2.0 中此问题已解决）

[编辑]

**（2） 获取帧图信息:**

```
float posMsec   =       cvGetCaptureProperty(capture,
CV_CAP_PROP_POS_MSEC);
```

```
int posFrames   = (int) cvGetCaptureProperty(capture,
CV_CAP_PROP_POS_FRAMES);
float posRatio  =      cvGetCaptureProperty(capture,
CV_CAP_PROP_POS_AVI_RATIO);
```

所抓取的帧的位置有三种表达方式：距离第一帧画面的时间间隔（毫秒为单位），或者距离第一帧画面（序列号为 0）的序列数. 第三种方式是按相对比率，第一帧的相对比率为 0，最后一帧的相对比率为 1. 此方式仅对读取视频文件时有效.

[编辑]

**（3） 设置从视频文件抓取的第一帧画面的位置:**

```
// start capturing from a relative position of 0.9 of a video
file
cvSetCaptureProperty(capture, CV_CAP_PROP_POS_AVI_RATIO,
(double)0.9);
```

注意此方法定位并不准确。

[编辑]

## 3、保存视频文件

[编辑]

**（1） 初始化视频编写器:**

```
CvVideoWriter *writer = 0;
int isColor = 1;
int fps     = 25;  // or 30
int frameW  = 640; // 744 for firewire cameras
int frameH  = 480; // 480 for firewire cameras
writer=cvCreateVideoWriter("out.avi",CV_FOURCC('P','I','M
','1'),
                           fps,cvSize(frameW,frameH),isColor);
```

其它的编码器代号包括: CV_FOURCC('P','I','M','1') = MPEG-1 codec CV_FOURCC('M','J','P','G') = motion-jpeg codec (does not work well) CV_FOURCC('M', 'P', '4', '2') = MPEG-4.2 codec CV_FOURCC('D', 'I', 'V', '3') = MPEG-4.3 codec CV_FOURCC('D', 'I', 'V', 'X') = MPEG-4 codec CV_FOURCC('U', '2', '6', '3') = H263 codec CV_FOURCC('I', '2', '6', '3') = H263I codec CV_FOURCC('F', 'L', 'V', '1') = FLV1 codec 若编码器代号为 -1，则运行时会弹出一个编码器选择框.

[编辑]

**（2） 保持视频文件:**

```
IplImage* img = 0;
int nFrames = 50;
for(i=0;i<nFrames;i++){
  cvGrabFrame(capture);        // capture a frame
  img=cvRetrieveFrame(capture);  // retrieve the captured
frame
  // img = cvQueryFrame(capture);

  cvWriteFrame(writer,img);     // add the frame to the file

}
```

要查看所抓取到的帧画面，可以在循环中加入以下语句:

```
cvShowImage("mainWin", img);
key=cvWaitKey(20);          // wait 20 ms
```

注意 cvWaitKey 参数应该不小于 20 ms，否则画面的显示可能出错.

[编辑]

**（3） 释放视频编写器:**

```
cvReleaseVideoWriter(&writer);
```

---

By Gady Agam 2006-03-31 翻译：chenyusiyuan 2010-1-26