多核架构与编程技术

openCV编程入门

主要内容

- OpenCV概述
- OpenCV基本数据类型和数据结构
- OpenCV基本操作（图形界面、图像、视频）
- 基本OpenCV程序与示例

注：本章节中部分代码来自OpenCV样例，部分代码来自《OpenCV教程——基础篇》，部分讲义内容从网上收集。

OpenCV概述

Open= Open Source，CV= Computer Vision

- OpenCV（Open Source Computer Vision Library）是Intel开源计算机视觉库。由一系列 C 函数和少量 C++ 类构成，实现了图像处理和计算机视觉方面的很多通用算法。
- OpenCV 拥有包括 500 多个C函数的跨平台的中、高层 API，OpenCV 为Intel® Integrated Performance Primitives (IPP) 提供了透明接口。这意味着如果有为特定处理器优化的的 IPP 库，OpenCV 将在运行时自动加载这些库。
- 跨平台：Windows, Linux
- 免费（FREE）：无论对非商业应用和商业应用
- 速度快、使用方便

OpenCV概述

OpenCV 的重要特性：

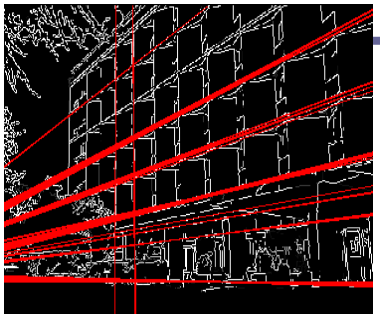
- 开源计算机视觉库采用C/C++编写；
- 使用目的是开发实时应用程序；
- 独立于操作系统、硬件和图形管理器；
- 具有通用的图像/视频载入、保存和获取模块；
- 具有底层和高层的应用开发包；
- ……

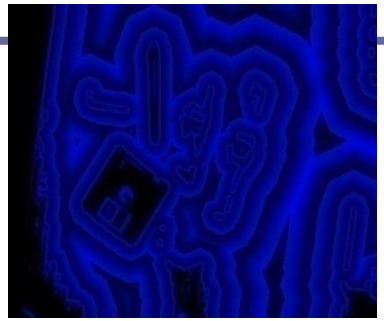
OpenCV概述

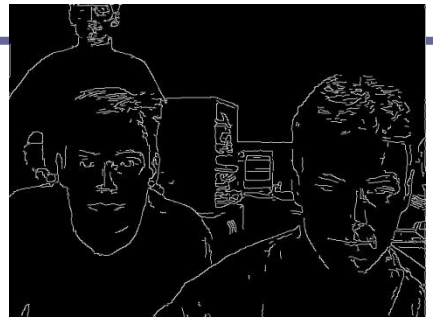
发展历史：

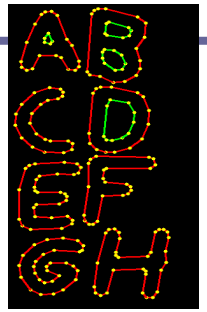
- 2000年6月，第一个开源版本OpenCV alpha 3发布。
- 2000年12月，针对linux平台的OpenCV beta 1发布。
- 2006年，支持Mac OS的OpenCV 1.0发布。
- 2009年9月，OpenCV 1.2（beta2.0）发布。
- 2009年10月1日，Version 2.0发布。
- 2010年12月6日，OpenCV 2.2发布。
- 2011年8月，OpenCV 2.3发布。
- 2018年2月，OpenCV 3.4.1发布

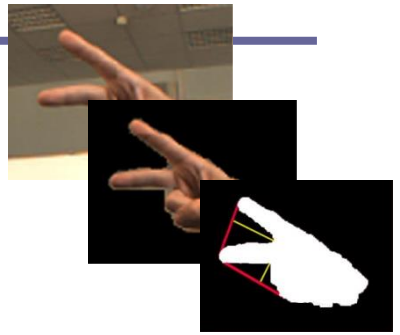
OpenCV 概述——应用领域

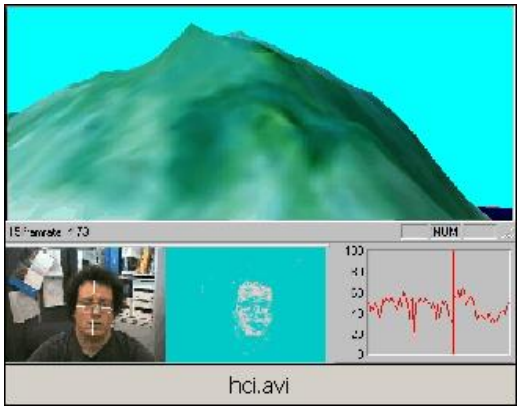


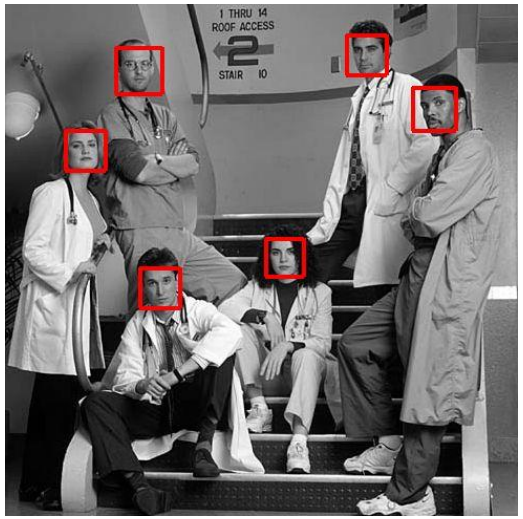


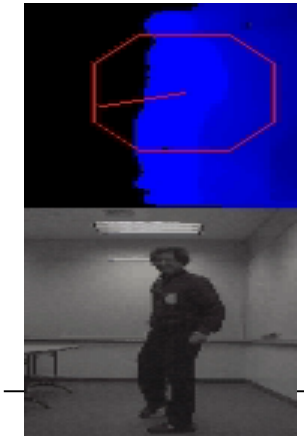


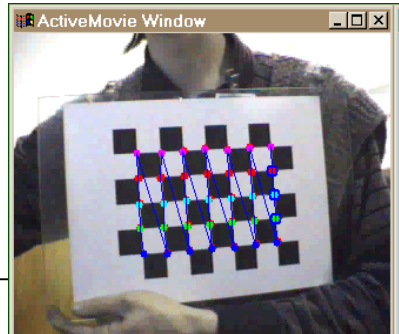


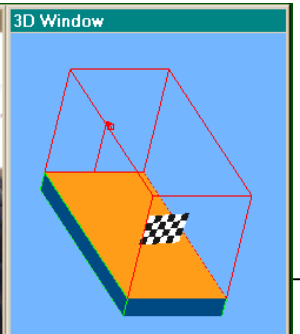


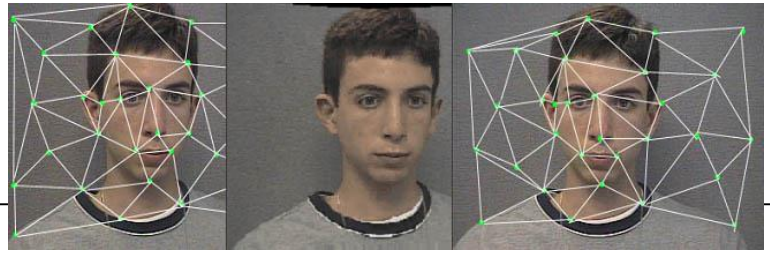












OpenCV概述

- OpenCV功能概述：
  - 矩阵和向量的操作以及线性代数算法的实现
  - 图像数据的操作及矩阵结构和图像结构的转换
  - 基本的数字图像处理能力
  - 基本的GUI功能
  - 其他功能

动态结构
摄像头定标
结构分析
运动分析
目标识别
图像标注

OpenCV概述

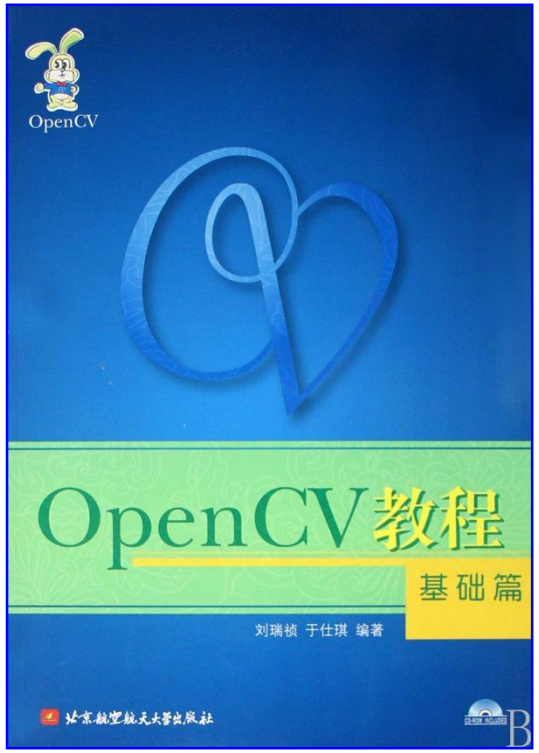
- OpenCV概述资源网站：
- OpenCV中文站
  http://www.opencv.org.cn/index.php

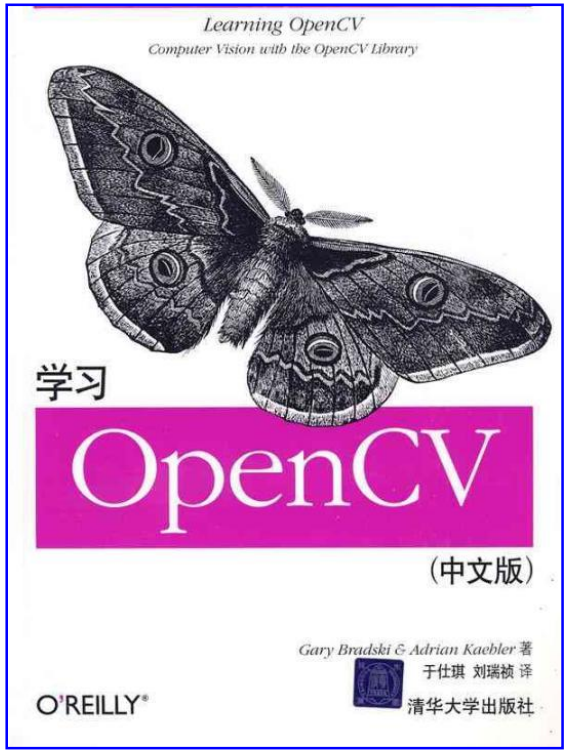
- OpenCV英文站
  http://opencv.willowgarage.com/wiki/

- OpenCV项目主页：源代码及文档下载
  http://sourceforge.net/projects/opencvlibrary/

OpenCV现有中文教材

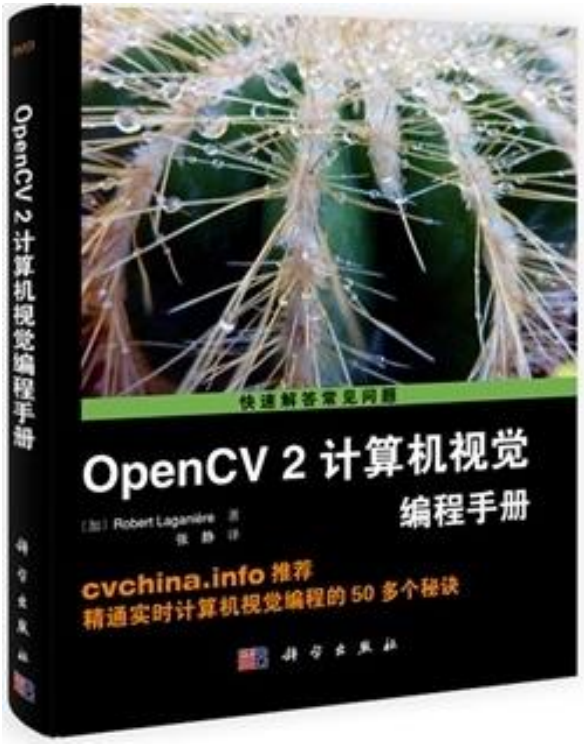
基于OpenCV1.X，建议基本掌握2.x基本环境后，再阅读。

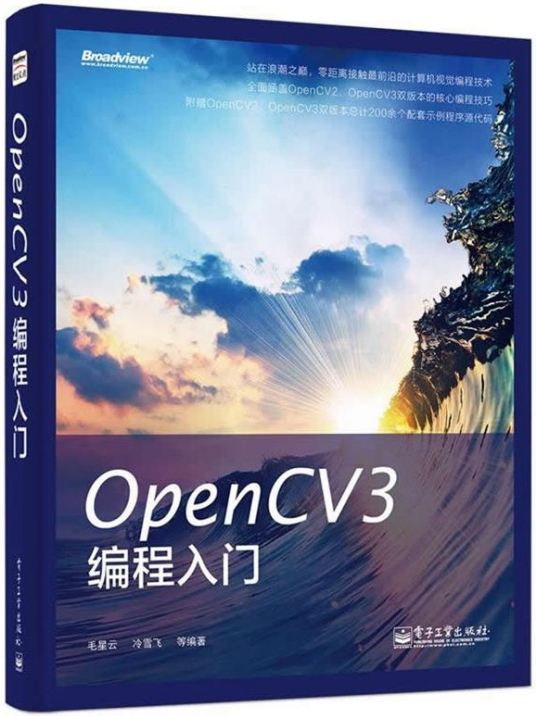


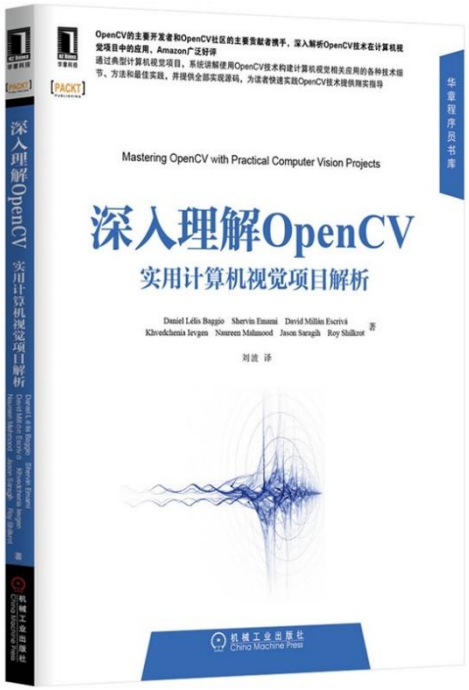


OpenCV现有中文教材

基于OpenCV2.X ~3.X

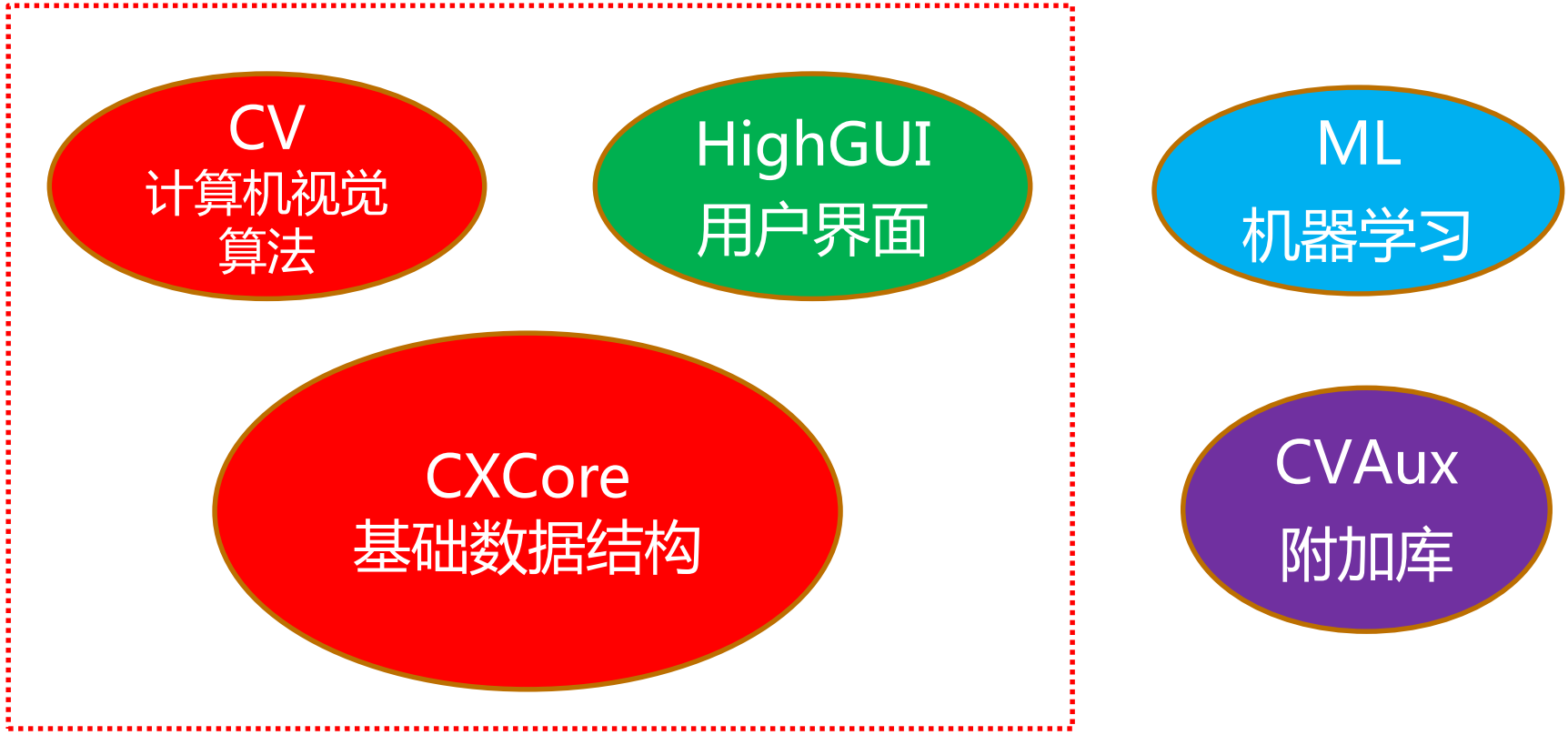






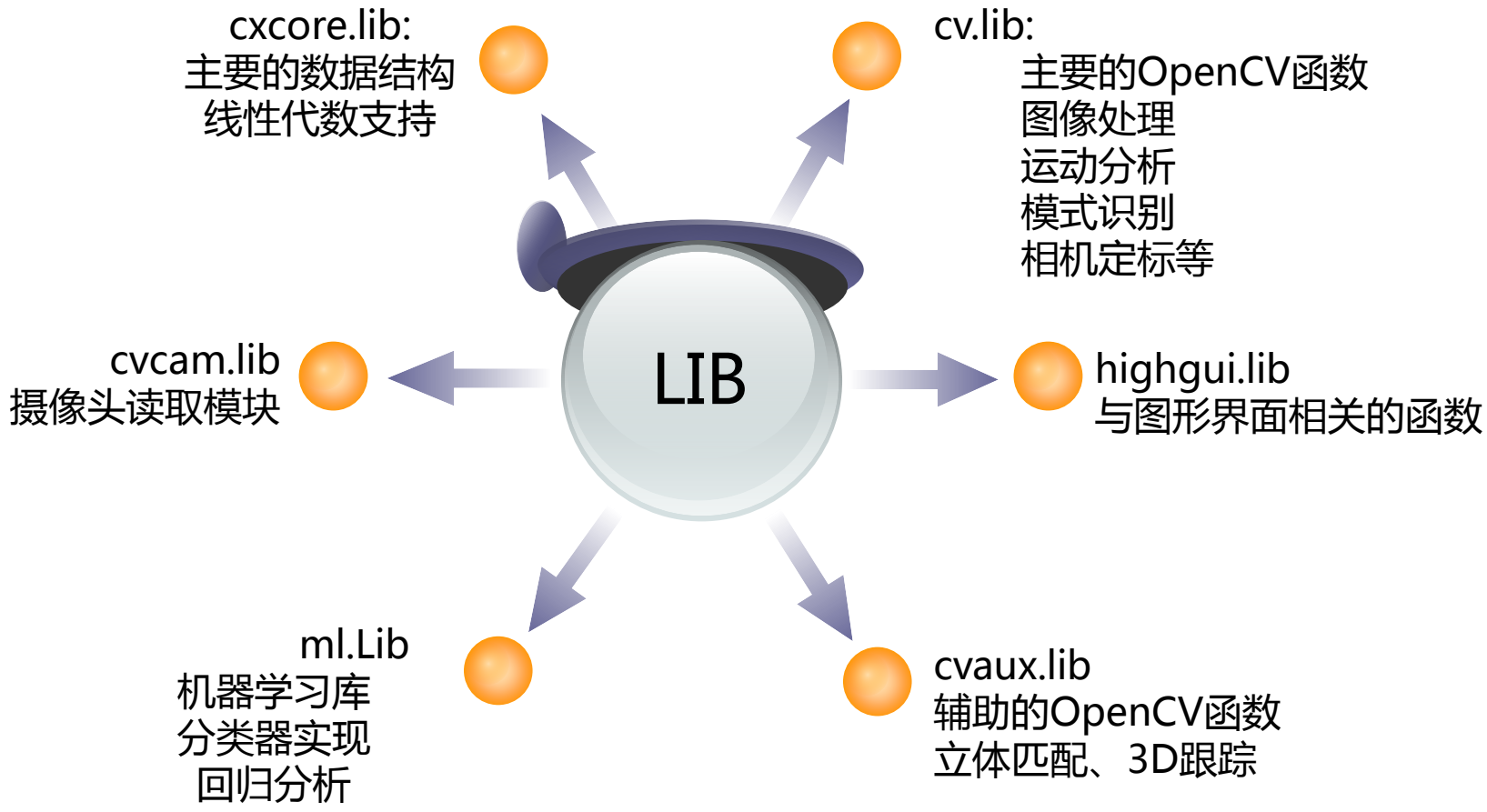
OpenCV概述

- OpenCV结构组成：



OpenCV概述

- OpenCV主要模块库：



Visual C++及其MFC版本

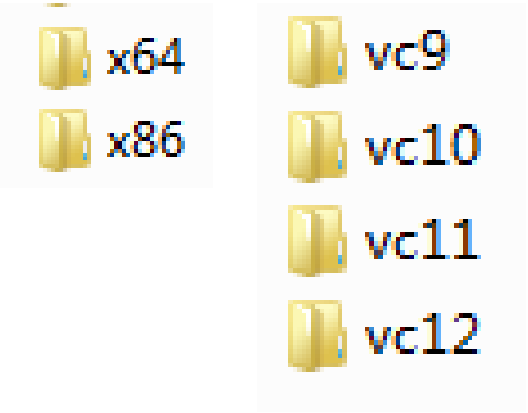
- MFC version 6.0 (still mfc42.dll)
- Visual C++ version 6.0
- MFC version 7.0 (mfc70.dll)
- Visual C++ .NET 2002
- MFC version 7.1 (mfc71.dll)
- Visual C++ .NET 2003
- MFC version 8.0 (mfc80.dll)
- Visual C++ 2005
- MFC version 9.0 (mfc90.dll)
- Visual C++ 2008
- MFC version 10.0 (mfc100.dll)
- Visual C++ 2010

MFC version 11.0

Visual C++ 2012

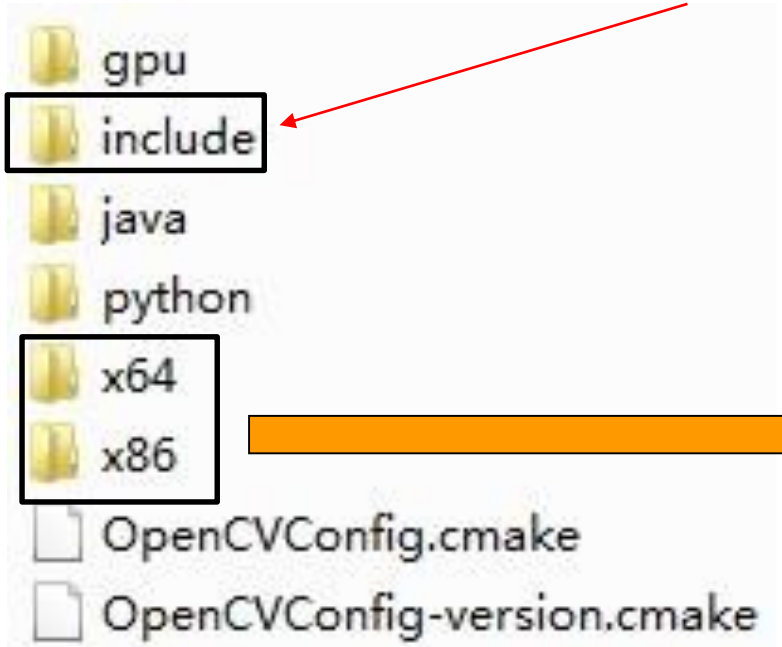
MFC version 12.0

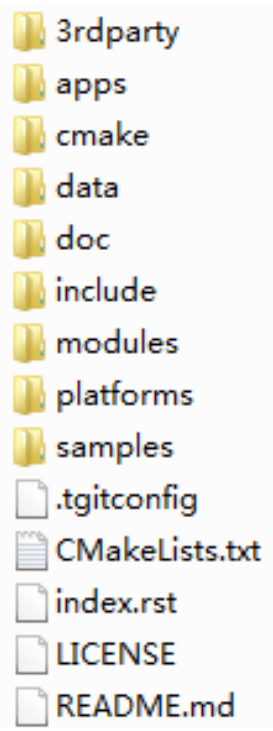
Visual C++ 2013



OpenCV的目录

- D:\Program Files\OpenCV\sources
- D:\Program Files\OpenCV\build





OpenCV3.0b+Visual Studio2013 配置1

- Vs2010以后采用属性表作为工程设置

运行库系统环境变量Path：

..\OpenCV3.0b\build\x86\vc12\bin;

编译库文件目录：

..\OpenCV3.0b\build\x86\vc12\lib

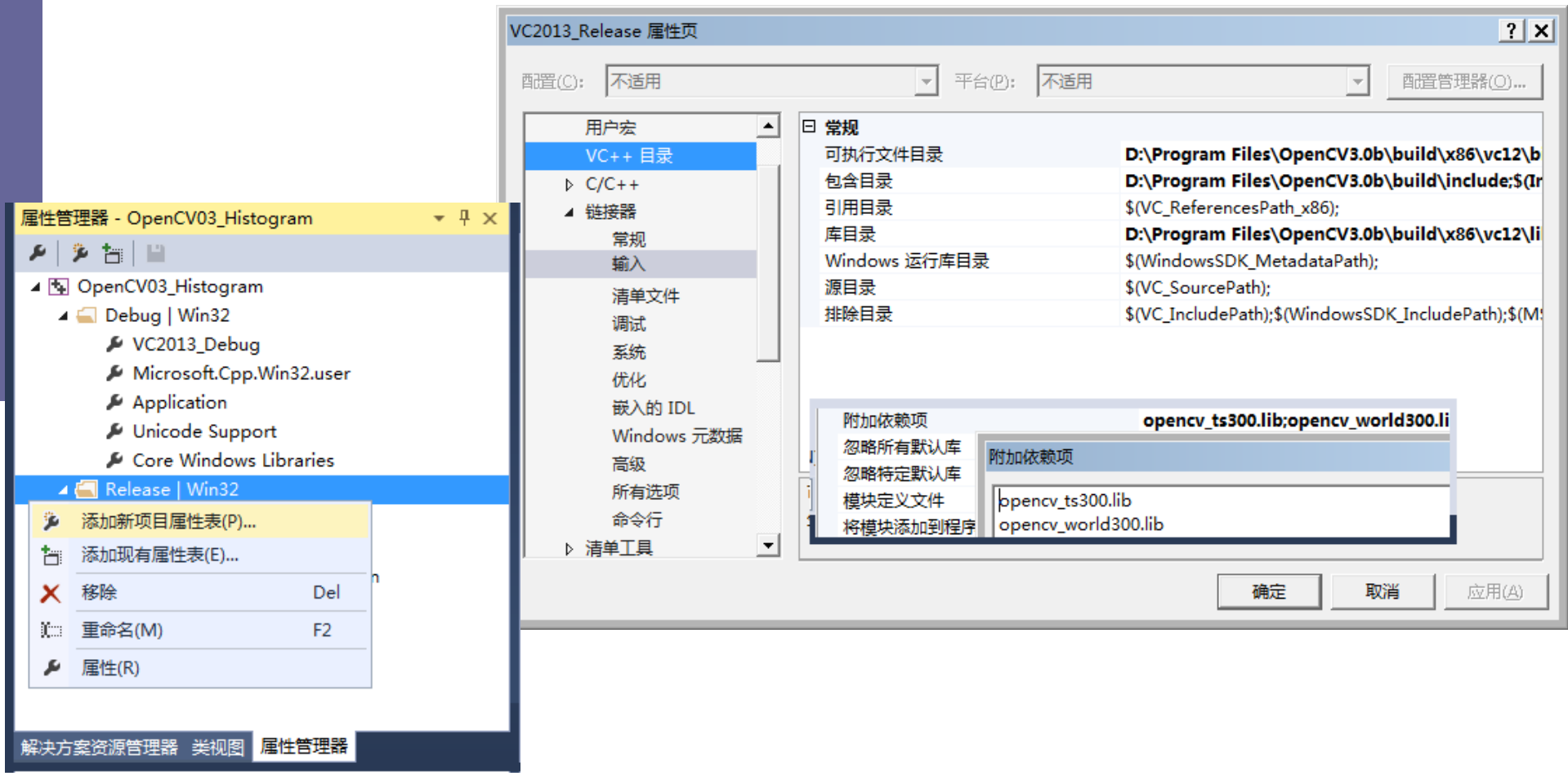
包含文件目录：

.. \OpenCV3.0b\build\include

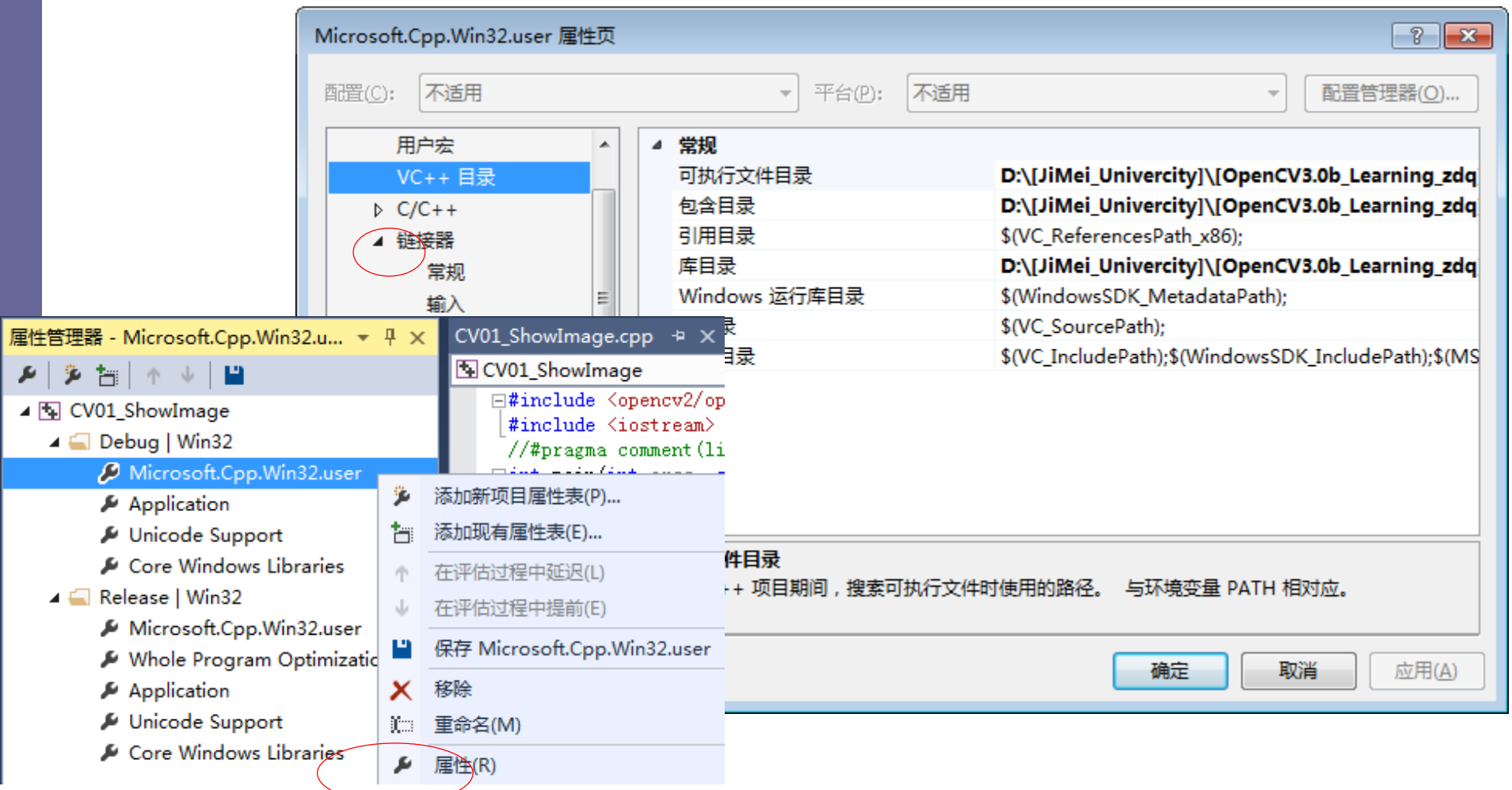
链接/输入/附加依赖项目：（带d的为Debug，不带为Release）

opencv_ts300.lib
opencv_world300.lib
opencv_ts300d.lib
opencv_world300d.lib

OpenCV3.0b+Visual Studio2013 配置2



OpenCV3.0b+Visual Studio2013 配置3



OpenCV概述

OpenCV编程样式：

1、文件命名：有cv和cvaux库文件的命名必须服从于以下规则：

- 所有的CV库文件函数名前缀为cv
- 混合的C/C++接口头文件扩展名为 .h
- 纯C++接口头文件扩展名为 .hpp
- 实现文件扩展名为 .cpp
- 为了与POSIX兼容，文件名都以小写字符组成

OpenCV概述

OpenCV编程样式：

2、文件结构

- 一行最多90个字符，不包括行结束符
- 不使用制表符
- 缩进为4个空格符，所以制表符应该用1-4个空格替换
- 头文件必须使用保护宏，防止文件被重复包含。
- 混合C/C++接口头文件用extern "C" { } 包含C语言定义。
- 为了使预编译头机制在Visual C++中工作正常，源文件必须在其它头文件前包含precomp.h头文件。
- 注意中英文字符的不同，中文易报错。

OpenCV概述

- OpenCV编程样式：

3、命名约定

- OpenCV中使用大小写混合样式来标识外部函数、数据类型和类方法。
- 宏全部使用大写字符，词间用下划线分隔。
- 所有的外部或内部名称，若在多个文件中可见，则必须含有前缀：
  外部函数使用前缀cv
  内部函数使用前缀Icv
  数据结构(C结构体、枚举、联合体、类)使用前缀Cv
  外部或某些内部宏使用前缀CV_
  内部宏使用前缀ICV_

OpenCV概述

- OpenCV编程样式：

4、函数接口设计：为了保持库的一致性，以如下方式设计接口非常重要。函数接口元素包括：

功能
名称
返回值
参数类型
参数顺序
参数默认值

- 函数功能必须定义良好并保持精简。函数应该容易镶入到使用其它OpenCV函数的不同处理过程。函数名称应该简单并能体现函数的功能。

大多数函数名形式：cv<ActionName>

OpenCV概述

- 通道的概念：独立的颜色平面

  可简单理解为表示一个像素需要几个元素

  1通道：描述一个像素点，如果是灰度，那么只需要一个数值来描述它，就是单通道，为1。

  2通道：2通道图像不常见，通常在程序处理中会用到，如傅里叶变换，可能会用到，一个通道为实数，一个通道为虚数，主要是编程方便；还有一种情况就是16位图像，本来是3通道，但是为了减少数据量，压缩为16位，刚好两个通道，常见格式有RGB555或RGB565，也就是说R占5位，G占5或6位，B占5位，也有RGBA5551格式。古老的格式，不用也罢。

  3通道：如果一个像素点，由RGB三种颜色来描述它，就是三通道，为3。

  4通道：windows的bmp有时候是一个四通道图像，R、G、B加上一个A通道，一般叫做alpha通道，表示透明度。

OpenCV概述

- ROI的概念：

  ROI在opencv中是指Region of interest，感兴趣的区域的意思。

- COI的概念：

  COI是Channel of interest的意思。在计算机表示图像的时候，是按RGBAplfa来表示一个像素的。每个R,G,B, Alfa都是一个Channel.　Region是指图像中的某一部分。

OpenCV基本数据类型和数据结构

- 基本数据类型：
- 点：CvPoint 、CvPoint2D32f、CvPoint3D32f
- 矩形框大小：CvSize 、CvSize2D32f
- 矩形框：CvRect
- 可以存放1-4个数值的数组：CvScalar
- 定义迭代算法的终止规则：CvTermCriteria
- 矩阵：CvMat 、CvMatND 、CvSparseMat
- IPL图像头部：IplImage
- 定义不确定的数组：CvArr （仅作函数参数）

OpenCV基本数据类型和数据结构

- 点数据结构：
- CvPoint 二维坐标系下的点，类型为整型

```
typedef struct CvPoint {
    int x; /* X坐标, 通常以0为基点 */
    int y; /* y坐标, 通常以0为基点 */
}CvPoint;
/* 构造函数 */
inline CvPoint cvPoint( int x, int y );
/* 从 CvPoint2D32f类型转换得来 */
inline CvPoint cvPointFrom32f( CvPoint2D32f point
)
```

- CvPoint2D32f ： 二维坐标下的点，类型为浮点
- CvPoint3D32f ： 三维坐标下的点，类型为浮点

OpenCV基本数据类型和数据结构

- 矩形数据结构：
- CvSize 矩形框大小，以像素为精度

```
typedef struct CvSize{
    int width; /* 矩形宽 */
    int height; /* 矩形高 */
}CvSize;
/* 构造函数 */
inline CvSize cvSize( int width, int height );
```

- CvSize2D32f

OpenCV基本数据类型和数据结构

- 矩形数据结构：
- CvRect 矩形框的偏移和大小

```
typedef struct CvRect{
        int x; /* 方形的最左角的x-坐标 */
        int y; /* 方形的最上或者最下角的y-坐标 */
        int width; /* 宽 */
        int height; /* 高 */
}CvRect;
/* 构造函数*/
inline CvRect cvRect(int x, int y, int width, int
height);
```

-

OpenCV基本数据类型和数据结构

- 矩阵数据结构：

```
typedef struct CvMat {
    int type;  /* CvMat 标识 (CV_MAT_MAGIC_VAL), 元素类型和标记 */
    int step;              /* 以字节为单位的行数据长度*/
    int* refcount;         /* 数据引用计数 */
    union { uchar* ptr;short* s; int* i; float* fl; double* db; } data;
/* data 指针 */
    #ifdef __cplusplus
    union { int rows; int height; };
    union { int cols; int width; };
    #else
    int rows;              /* 行数 */
    int cols;              /* 列数*/
    #endif
    } CvMat;
```

OpenCV基本数据类型和数据结构

- 矩阵数据结构：
- int type;
- 矩阵通用矩阵数据类型：
- CV_<bit_depth>(S|U|F)C<number_of_channels>
- S：带符号整数
- U：无符号整数
- F：浮点数。
- 例：CV_8UC1：8位无符号单通道矩阵；
- CV_32FC2：32位浮点数双通道矩阵。

OpenCV基本数据类型和数据结构

- 矩阵数据结构：
- int * refcount;
- 数据引用计数
- refcount==NULL：
- 矩阵的数据区为外部数据，不需释放
- refcount！=NULL：
- 需要释放矩阵头和数据区。
- int step;
- 以字节为单位的行数据长度
- 是定位元素所需要的行信息

OpenCV基本数据类型和数据结构

- 矩阵数据结构：
- union
- {uchar *ptr;
- short *s;
- int *i;
- float *fl;
- double *db;
- }data;
- 指向数据区首地址的指针。指针是公用体结构，使用时要根据矩阵的数据类型选择。

OpenCV基本数据类型和数据结构

- 图像头结构：
- IplImage：IPL 图像头

```
typedef struct _IplImage {
        int nSize; /* IplImage大小 */
        int ID; /* 版本 (=0)*/
        int nChannels; /* 大多数OPENCV函数支持1,2,3 或4 个通道 */
        int alphaChannel; /* 被OpenCV忽略 */
        int depth; /* 像素的位深度: IPL_DEPTH_8U,IPL_DEPTH_8S,
                IPL_DEPTH_16U,IPL_DEPTH_16S, IPL_DEPTH_32S,
                IPL_DEPTH_32F and IPL_DEPTH_64F 可支持 */
        char colorModel[4]; char channelSeq[4]; /* 被OpenCV忽略 */
          int dataOrder; /* 0 - 交叉存取颜色通道, 1 - 分开的颜色通道.
                                cvCreateImage只能创建交叉存取图像 */
        int origin; /* 0 - 顶—左结构,1 - 底—左结构 (BMP风格) */
        int align; /* 图像行排列 (4 or 8). OpenCV 用widthStep 代替 */
```

OpenCV基本数据类型和数据结构

- 图像头结构：
- int width; /* 图像宽像素数 */
  int height; /* 图像高像素数*/
  struct _IplROI *roi;/* 图像感兴趣区域. 当该值非空只对该区域进行处理 */
  struct _IplImage *maskROI; /* 在 OpenCV中必须置NULL */
  void *imageId; /* 同上*/
  struct _IplTileInfo *tileInfo; /*同上*/
  int imageSize; /* 图像数据大小，单位字节*/
  char *imageData; /* 指向排列的图像数据 */
  int widthStep; /* 排列的图像行大小，以字节为单位 */
  int BorderMode[4]; int BorderConst[4]; /* 边际结束模式, 被忽略*/
  char *imageDataOrigin; /* 指针指向一个不同的图像数据结构，是为了纠正图像内存分配准备的 */
  }IplImage;

OpenCV基本数据类型和数据结构

- 图像头结构：
- IplImage结构来自于 Intel Image Processing Library。OpenCV 只支持其中的一个子集：
- alphaChannel 在OpenCV中被忽略。
- colorModel 和channelSeq 被OpenCV忽略。
- dataOrder 必须是IPL_DATA_ORDER_PIXEL (颜色通道是交叉存取)，然而平面图像的被选择通道可以被处理，就像COI（感兴趣的通道）被设置过一样。
- align 是被OpenCV忽略的，而用 widthStep 去访问后继的图像行。
- 不支持maskROI 。处理MASK的函数把他当作一个分离的参数。MASK在OpenCV 里是 8-bit，然而在 IPL他是 1-bit。
- tileInfo 不支持。
- BorderMode和BorderConst是不支持的。
- OpenCV处理ROI有不同的要求。要求原图像和目标图像的尺寸或 ROI的尺寸必须精确匹配。

OpenCV基本数据类型和数据结构

- 图像头结构：
- int depth;
- 图像通用数据类型：
- IPL_DEPTH_<bit_depth>(S|U|F)
- S、U、F的意义同矩阵数据类型。
- 例：IPL_DEPTH_8U：8位无符号整数图像
- IPL_DEPTH_32F：32位浮点数图像
- int nChannels;
- 图像的通道数
- 例：灰度图为1个通道
- 复值图像为2个通道
- RGB图像为3个通道
- RGBA图像为4个通道（A通道即阿尔法通道，下去查资料了解）
- 大多数OpenCV函数支持1~4个通道

OpenCV基本数据类型和数据结构

- 图像头结构：
- int dataOrder;
- 图像数据的存储格式
- 0 :交叉存取颜色通道 1:分开存取颜色通道
- OpenCV函数只支持交叉存取的图像。
- int widthStep;
- 排列的图像行长度，以字节为单位
- 与矩阵结构中的step成员相似

OpenCV基本数据类型和数据结构

- 图像头结构：
- struct _IplROI *roi;
- ROI：Region Of Interest（感兴趣区域）
- roi==NULL：整幅图像参与运算
- roi！=NULL：ROI区域代替图像参加运算
- ROI的操作：
- cvSetImageROI()：设置ROI区域
- cvResetImageROI()：取消ROI区域
- cvGetImageROI()：得到ROI区域

OpenCV基本数据类型和数据结构

- 图像头结构：
- int origin;
- 图像像素的起始方式
- origin==0：顶-左结构
- origin==1：底-左结构（windows风格）
- char *imageData;
- 图像的数据区
- char *类型而非unsigned char *类型，进行浮点处理时可能要加到unsigned char的转换，否则会导致结果不正常。

OpenCV基本数据类型和数据结构

- 不确定数组：
- CvArr：不确定数组
- 只用作函数的参数
- 表示可接受多种类型的输入形式 （矩阵等）
- 运行时通过分析数组头的前4个字节来判断
- 大多数CvArr*做输出参数的情况下，函数是对CvArr的结构进行写入操作而不是返回指针

OpenCV基本操作

- 矩阵操作：
- 创建矩阵 CreateMat
  CvMat* cvCreateMat(int rows, int cols, int type);
- Rows 矩阵行数，cols 矩阵列数。
  Type 矩阵元素类型。 通常以 CV_<比特数>(S|U|F)C<通道数>型式描述， 例如: CV_8UC1 意思是一个8-bit 无符号单通道矩阵, CV_32SC2 意思是一个32-bit 有符号二个通道的矩阵。
- 函数 cvCreateMat 为新的矩阵分配头和下面的数据，并且返回一个指向新创建的矩阵的指针。矩阵按行存贮。所有的行以4个字节对齐。
- 删除矩阵 ReleaseMat
  void cvReleaseMat(CvMat** mat);
- 例如：CvMat* M = cvCreateMat(4, 4, CV_32FC1);
  cvReleaseMat(&M);

OpenCV基本操作

- 矩阵操作：
- 复制矩阵CloneMat：CvMat* cvCloneMat(const CvMat* mat);
- 例如：
```
CvMat* M1 = cvCreateMat( 4, 4, CV_32FC1);
CvMat* M2;
M2 = cvCloneMat(M1);
```
- 初始化矩阵
- 方法1：用cvMat初始化
```
double a[] = { 1, 2, 3, 4,5, 6, 7, 8,9, 10, 11, 12 };
CvMat Ma = cvMat( 3, 4, CV_64FC1, a);
```
- 方法2：用cvCreateMatHeader初始化
```
CvMat Ma;
cvInitMatHeader( &Ma, 3, 4, CV_64FC1, a);
```
- 初始化单位矩阵
- CvMat* M = cvCreateMat(4, 4, CV_32FC1);
cvSetIdentity(M);

OpenCV基本操作

- 矩阵操作：
- 访问矩阵元素
- (1)直接访问

```
cvmSet(M, i, j, 2); //cvmSet(CvMat* mat, int row, int col, double value);
t = cvmGet(M, i, j); //Get M(i,j)
```

- (2)已知对齐方式的直接访问

```
CvMat* M = cvCreateMat( 4, 4, CV_32FC1);
int n = M->cols;
float *data = M->data.fl;
data[i*n+j] = 3.0; //假设32位对齐
```

- (3)未知对齐方式的直接访问

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

- (4)直接访问一个已初始化的矩阵

```
Ma[i*4 + j] = 2.0;
```

OpenCV基本操作

- 矩阵操作：
- 矩阵间的操作
  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
- 矩阵元素间的操作
- cvMul(Ma, Mb, Mc);　　//Ma. * Mb -> Mc
- cvDiv(Ma, Mb, Mc); //Ma. / Mb -> Mc
- cvAddS(Ma, cvScalar(-10.0), Mc); //Ma. -10 -> Mc
- 单个矩阵的操作
- cvTranspose(Ma, Mb);　//transpose(Ma) -> Mb ，求转置
- CvScalar t=cvTrace(Ma);　//trace(Ma) ->t.val[0]，求对角线上元素的和
- double d = cvDet(Ma);　//det(Ma) ->d ，计算方阵行列式
- cvInvert(Ma, Mb);　　//inv(Ma) ->Mb，　求逆阵

OpenCV基本操作

- 矩阵操作：
- 矩阵间的操作
  向量乘法，假设Va, Vb, Vc均为n元素向量
- double res = cvDotProduct(&Va, &Vb); //Va •Vb -> res
- cvCrossProduct(&Va, &Vb , &Vc); // Va ×Vb -> Vc
- 矩阵特征值分解
  假设A, E均为n*n方阵，I为n元素向量
- cvEigenVV(&A, &E , &I);
- SVD（奇异值分解）
- 假设A, U, D, V均为n*n方阵
- cvSVD(A, D, U, V, CV_SVD_U_T|| CV_SVD_V_T); //A=UDV~T标志使U和V以转置方式返回
- 非齐次线性系统的求解
  假设A为n*n方阵，x,b均为n元素向量
- cvSolve(&A, &b, &x)

OpenCV基本操作

- 图形界面操作：
- 与windows程序设计有许多类似之处
- 创建一个有名字的窗口：
- cvNamedWindow(window_name, fixed_size_flag)
  cvNamedWindow(window_name, x, y)
- 销毁窗口：
  cvDestroyWindow(window_name)
- ……
- 操作函数很多，可根据函数名大致判断其功能
- cvNamedWindow(“ViewA”,1);　（CV_WINDOW_AUTOSIZE=1）
- cvMoveWindow("ViewA",300,100);
- cvDestroyWindow("ViewA");
- cvShowImage(window_name, image);

OpenCV基本操作

- 交互操作：
- 等待按键cvWaitKey

  int cvWaitKey(int delay=0)
- 如果delay<=0, 则无限等待，否则等待delay毫秒则返回
- 在程序循环中，有时候由于程序一直处于计算中，窗口无法重新恢复（如读出视频中的所有帧并显示），可以加入cvWaitKey，使之等待几毫秒，让窗口完成重新绘制再执行其他操作。
- 其它交互函数.....

OpenCV基本操作

- 图像操作：
- 创建头并分配数据 CreateImage

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

  size:图像宽、高.

  depth:图像元素的位深度,IPL_DEPTH_{8U|8S|16U|16S|32S|32F|64F

  channels:每个元素（像素）的颜色通道数量.可以是 1, 2, 3 或 4.
- 释放头和图像数据 ReleaseImage

  void cvReleaseImage(IplImage** image);
- 复制图像 CloneImage

  IplImage* cvCloneImage(const IplImage* image);

OpenCV基本操作

- 图像操作：
- 头分配CreateImageHeader
  IplImage* cvCreateImageHeader(CvSize size, int depth, int channels)
- 初始化被用图分配的图像头 InitImageHeader
  IplImage* cvInitImageHeader(IplImage* image, CvSize size, int depth,
  int channels, int origin=0, int align=4);
  origin　　IPL_ORIGIN_TL 或 IPL_ORIGIN_BL.
  align　　图像行排列, 典型的 4 或 8 字节.
  函数 cvInitImageHeader 初始化图像头结构, 指向用户指定的图像并且返回这个指针。
- 释放头 ReleaseImageHeader
  void cvReleaseImageHeader(IplImage** image);

OpenCV基本操作

- 图像操作：
  - 从文件读图像cvLoadImage
    IplImage* cvLoadImage(char* fileName, int flag=1)
  - OpenCV支持的图像格式：BMP、DIB、JPG、PNG、PBM、PGM、PPM、SR、RAS和TIFF
  - 写图像到文件cvSaveImage
    IplImage* cvSaveImage(char* fileName, IplImage* img)
  - 图像转换
    cvConvertImage(IplImage* src, IplImage* dst, int flags=0); //用于不同图像格式之间的转换
  - cvCvtColor(IplImage* src, IplImage* dst, int code);
    //彩色图/灰度图
    Code=CV_<X>2<Y>:<X>,<Y>=RGB,BGR,GRAY,HSV,YCrCb,XYZ,Luv,HLS

OpenCV基本操作

- 视频操作：
  - 打开摄像头
    CvCapture* cvCaptureFromCAM(camera_id=0);
  - 打开文件
    CvCapture* cvCaptureFromFile(videofile_path);
    CvCapture* cvCaptureFromAVI("inflie.avi");
  - 捕捉某一帧
    cvGrabFrame(capture) //抓住一帧，为快速遍历视频帧
    IplImage* img=cvRetrieveImage(capture);//把Grab的帧取出，或
    IplImage* cvQueryFrame(capture);
  - 释放捕捉源
    cvReleaseCapture(&capture);

OpenCV基本操作

- 视频操作：
  - 保存视频文件
  - typedef struct CvVideoWriter;
  - CvVideoWriter* cvCreateVideoWriter(const char* filename, int fourcc, double fps, CvSize frame_size, int is_color=1);
  - int cvWriteFrame(CvVideoWriter* writer, const IplImage* image);
  - void cvReleaseVideoWriter(CvVideoWriter** writer);

OpenCV基本操作

- 视频操作：
  - 获取/设置视频帧信息
    cvGetCaptureProperty(capture, property_id);
    cvSetCaptureProperty(capture, property_id, value);
  - CV_CAP_PROP_POS_MSEC - video capture timestamp
    CV_CAP_PROP_POS_FRAMES - 0-based index of the frame
    CV_CAP_PROP_POS_AVI_RATIO - relative position of video(0-start, 1-end)
    CV_CAP_PROP_FRAME_WIDTH - width of frames in the video stream
    CV_CAP_PROP_FRAME_HEIGHT - height of frames in the video stream
    CV_CAP_PROP_FPS - frame rate
    CV_CAP_PROP_FOURCC - 4-character code of codec
    CV_CAP_PROP_FRAME_COUNT - number of frames in video file

OpenCV基本操作

- 内存操作：
  - 内存相关结构体及函数
  - CvMemBlock——内存存储块结构
  - typedef struct CvMemBlock

```
{
    struct CvMemBlock* prev;
    struct CvMemBlock* next;
} CvMemBlock;
```

  - 这是OpenCV所管理的基本的内存块，代表一个单独的内存存储块结构，一个双向的循环链表。

OpenCV基本操作

- 内存操作：
- 内存相关结构体及函数
- CvMemStoragePos——内存存储块地址

```
typedef struct CvMemStorage
{
    struct CvMemBlock* bottom;    /* first allocated block */
    struct CvMemBlock* top;       /* top of the stack */
    struct CvMemStorage* parent;/* new blocks from */
    int block_size; /* block size */
    int free_space; /* free space in the top block (in bytes) */
} CvMemStorage;
```

- 一个可用来存储诸如序列，轮廓，图形,子划分等动态增长数据结构的底层结构。它是由一系列以同等大小的内存块（CvMemBlock）构成，呈列表型。存储器就如同栈，bottom指向栈底，top指向栈顶。

OpenCV基本操作

- 内存操作：
  - 内存相关结构体及函数
  - CreateMemStorage——创建内存块
  - CvMemStorage* cvCreateMemStorage(int block_size=0);
  - block_size：存储块的大小以字节表示。
  - 若大小是0byte, 则将该块设置成默认值64k。
  - 该函数创建一内存块并返回指向块首的指针。起初，存储块是空的。
  - CreateChildMemStorage——创建子内存块
  - CvMemStorage* cvCreateChildMemStorage(CvMemStorage* parent);
  - parent：父内存块
  - 该函数创建一类似于普通内存块的子内存块，除了内存分配/释放机制不同外，新的块一般是从parent那得到

OpenCV基本操作

- 内存操作：
  - 内存相关结构体及函数
  - ReleaseMemStorage——释放内存块
  - void cvReleaseMemStorage(CvMemStorage** storage);

    storage：指向被释放了的存储块的指针

    该函数释放所有的存储块，或将它们返回给各自的parent。接下来再释放header块并清除指向该块的指针。在释放parent块之前，先清除各自的child块。
  - ClearMemStorage——清空内存存储块
  - void cvClearMemStorage(CvMemStorage* storage);

    storage：存储存储块

    该函数并不释放内存（仅清空内存）。假使该内存块有一个父内存块（即：存在一内存块与其有父子关系），则函数就将所有的块返回给其parent.

OpenCV基本操作

- 内存操作：
  - 其它内存操作：
  - 除了上面介绍的直接进行内存操作的函数外，还有一些函数在使用时需要注意内存泄漏问题，在创建完一个对象后，一定要记得释放对象所占用的那部分内存。以下列出了需要注意内存释放的函数，使用时应该成对出现。
  - //创建并释放图像数据
    cvCreateImage
    cvReleaseImage
  - //创建并释放图像头
    cvCreateImageHeader
    cvReleaseImageHeader

OpenCV基本操作

- 内存操作：
  - 其它内存操作：
  - //创建并释放矩阵

    cvCreateMat

    cvReleaseMat
  - //创建并释放多维密集数组

    cvCreateMatND

    cvReleaseMatND
  - //分配并释放数组数据

    cvCreateData

    cvReleaseData
  - //创建并释放稀疏数组

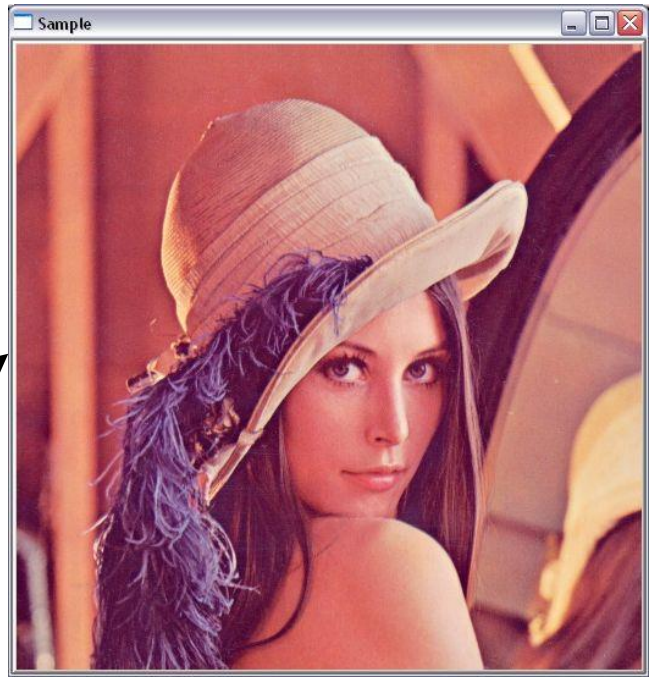
    cvCreateSparseMat

    cvReleaseSparseMat

openCV编程示例---读取和显示图像

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

int main( int argc, char** argv )
{
      IplImage* image;
      if( argc != 2 ) return -1;
      image = cvLoadImage( argv[1] );
      if( !image ) return -1;
      cvNamedWindow( “Sample”, 1 );
      cvShowImage( “Sample”, image );
      cvWaitKey(0);
      cvDestroyWindow ( “Sample” );
      cvReleaseImage (&image);
      return 0;
}
```



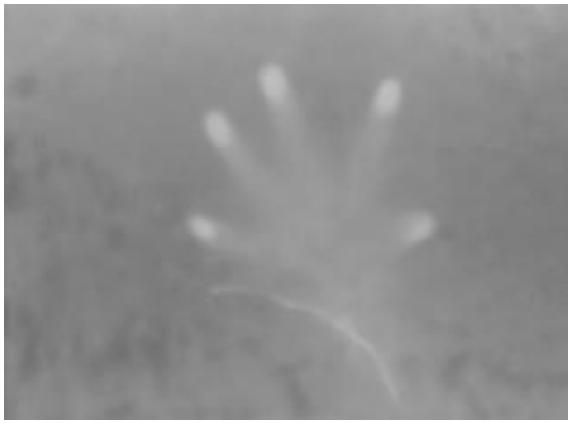
openCV编程示例—从视频中提取目标

```
#include <cv.h>
#include <highgui.h>
int main()
{
        CvCapture *capture = cvCaptureFromCAM(0);
        IplImage *firstFrame = 0;
        firstFrame = cvQueryFrame(capture);
                ……
        while((img = cvQueryFrame(capture)) && c != 27)
        {
            cvCvtColor(img, crtImg, CV_RGB2GRAY);
            cvAbsDiff(crtImg, preImg, difImg);
            cvThreshold(difImg, difImg, 30, 255.0, CV_THRESH_BINARY);
            cvShowImage("test", difImg);
            c = cvWaitKey(33);
        }
```

示例处理效果

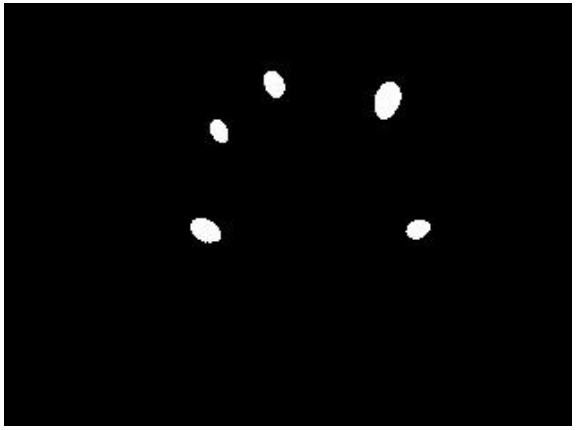
背景

新帧



背景相减

二值化



作业（最迟5月1日提交）

请用5部不同的手机拍摄同一个物品上的同一个二维码的照片，每部手机拍2张，共计10张图片。基于OpenCV，采用2种多线程编程方式同时显示这10张图片，并显示处理时间。每个同学的二维码不得相同（二维码物理尺寸不小于1厘米，不大于2.5厘米）。请提交主要源代码、release执行文件和10张原始图像。（拍摄时二维码尽量充满画面，如下图）

谢谢