**OpenCV数据结构和绘图函数**

1. Mat数据结构详解
   1. Mat结构组成和适用方法

* Mat由两个数据部分组成 : 即矩阵头 + 指向矩阵的指针。

矩阵头 = 矩阵尺寸 + 存储方法 + 储存地址等

OpenCV 每个Mat对象有属于自己的信息头，但是共享同一个矩阵，它通过矩阵指针指向同一个矩阵的地址实现。

Mat A,C; //仅创建信息头部分

A = imread(“1.jpg”,CV\_LOAD\_IMAGE\_COLOR); //这里为就诊开辟内存空间

Mat B(A); //拷贝构造函数

C = A; // 复制运算符

* 创建包含边界信息的信息头的方法：

Mat D(A, Rect(10,10,100,100)); //用矩形界定

Mat E(Range:all(),Range(1,3)); //用行和列来界定

* Mat 的引用计数概念：

每当复制一个Mat对象的信息头时，增加引用计数，

每当一个信息头被释放的时候，减小引用计数。

* 通过copyTo或clone复制矩阵

Mat F = A.clone();

Mat G;

A.copyTo(G);

* 获取图像的长和宽

Ip1Image\* img = cvLoadImage(“1,jpg”,1);

Mat mtx(img);

img.width(),img.Heihgt()

* 1. 像素值的存储方法

存储像素需要指定颜色空间和数据类型。

颜色空间概念：针对一个给定的颜色，如何组合颜色元素以对其编码。

如灰度级空间，只处理黑色和白色。

彩色的颜色空间： RGB + 透明颜色 四个元素。

颜色系统种类：

1. RGB 显示设备所采用
2. HSV和HLS把颜色分解成色调、饱和度和亮度透明度
3. YCrCb在JPEG图像格式中广泛使用
4. ClE L\*a\*b 是一种感知上均匀的颜色空间，他适合用来度量两个颜色之间的距离。
   1. 创建Mat对象的方法
5. 使用Mat 构造函数

Mat M(2,2,CV\_8UC3,Scalar(0,0.255));

|  |  |  |  |
| --- | --- | --- | --- |
| 参数1 | 参数2 | 参数3 | 参数4 |
| 行 | 列 | 数据类型 | 通道数 |

数据类型结构：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CV | 位数 | 带符号与否 | 类型前缀 | C | 通道数 |

1. C\C++中通过构造函数进行初始化

Int sz[3] = {2,2,2};

Mat L(3,sz,CV\_8UC,Scalar::all(0));

1. 为已存在的Ip1Image 指针创建信息头

Ip1Image\* img = cvLoadImage(“1,jpg”,1);

Mat mtx(img);

1. 利用Create()函数

M.creat(4,4,CV\_8UC(2));

Cout <<”M=”<<” ”<<M<<endl;

* 1. OpenCV格式化输出
  2. 常用数据结构输出
* 定义输出二维点

Point2f p(6, 2);

cout << "【?2维?点Ì?】?p = " << p << ";\n" << endl;

* 定义输出三维点

Point3f p3f(8, 2, 0);

cout << "【?3维?点Ì?】?p3f = " << p3f << ";\n" << endl;

* 定义输出基于Mat的std：：vector

vector<float> v;

v.push\_back(3);

v.push\_back(5);

v.push\_back(7);

cout << "【?基¨´于®¨²Mat的Ì?vector】?shortvec = " << Mat(v) << ";\n"<<endl;

* 定义输出Mat的std：：vector

vector<Point2f> points(20);

for (size\_t i = 0; i < points.size(); ++i)

points[i] = Point2f((float)(i \* 5), (float)(i % 7));

cout << "【?二t维?点Ì?向¨°量¢?】?points = " << points<<";";

1. IplImage 数据结构详解
   1. IplImage与Mat的关系

OpenCV最初是C语言编写的一个图形库，其中IplImage是表示一个图像的结构体，也是从OpenCV1.0到目前最为重要的一个结构；

Mat是后来OpenCV封装的一个C++类，用来表示一个图像，和IplImage表示基本一致，但是Mat还添加了一些图像函数；

* 1. IplImage与Mat 创建图像数据的不同
     1. IplImage创建图像数据

IplImage ，通过以下三个函数之一创建：

/\* Allocates and initializes IplImage header \*/

CVAPI(IplImage\*) cvCreateImageHeader( CvSize size, int depth, int channels );

/\* Inializes IplImage header \*/

CVAPI(IplImage\*) cvInitImageHeader( IplImage\* image, CvSize size, int depth,

int channels, int origin CV\_DEFAULT(0),

int align CV\_DEFAULT(4));

/\* Creates IPL image (header and data) \*/

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

* + 1. Mat 创建图像数据

cv::Mat，直接可以通过构造函数Mat(int \_rows, int \_cols, int \_type, void\* \_data, size\_t \_step=AUTO\_STEP); 直接创建；

示例：

cv::Mat \* pMat = new cv::Mat( 288, 352,  CV\_8UC3, imagebufdata );  
IplImage IplImagetmp = IplImage(\*pMat);

注意：前两个参数是图像的height和width，不是width和height；

* + 1. IplImage和Mat 创建图像的比较

创建内存数据图像，直接通过 cv::Mat类比较简单，然后可以通过Mat获取IplImage，通过cvCreateImage等函数创建内存图像，比较麻烦，而且创建后，还要通过cvReleaseImage等函数释放内存，所以这里建议用cv::Mat创建；

* 1. IplImage和Mat的互相转换
* IplImage  转  Mat:

// extern  IplImage \* plpliamge; //假设 IplImage 已经创建；

 Mat \* pmatImage = new Mat( IplImage, 0  ): //第二个参数表示不进行像素数据copy；

* Mat 转 IplImage:

//extern Mat  matImage; //假设已经创建cv:Mat;

IplImage limage = IplImage ( matImage );//不进行数据copy；

通常情况对于图像的读取，IplImage 通过 cvLoadImage，   cv:Mat通过 cv::imread;

* 1. IplImage 结构体详解 – 源代码

typedef struct \_IplImage

{

int nSize; /\* sizeof(IplImage) \*/

int ID; /\* version (=0)\*/

int nChannels; /\* Most of OpenCV functions support 1,2,3 or 4 channels \*/

int alphaChannel; /\* Ignored by OpenCV \*/

int depth; /\* Pixel depth in bits: IPL\_DEPTH\_8U, IPL\_DEPTH\_8S, IPL\_DEPTH\_16S,

IPL\_DEPTH\_32S, IPL\_DEPTH\_32F and IPL\_DEPTH\_64F are supported. \*/

char colorModel[4]; /\* Ignored by OpenCV \*/

char channelSeq[4]; /\* ditto \*/

int dataOrder; /\* 0 - interleaved color channels, 1 - separate color channels.

cvCreateImage can only create interleaved images \*/

int origin; /\* 0 - top-left origin,

1 - bottom-left origin (Windows bitmaps style). \*/

int align; /\* Alignment of image rows (4 or 8).

OpenCV ignores it and uses widthStep instead. \*/

int width; /\* Image width in pixels. \*/

int height; /\* Image height in pixels. \*/

struct \_IplROI \*roi; /\* Image ROI. If NULL, the whole image is selected. \*/

struct \_IplImage \*maskROI; /\* Must be NULL. \*/

void \*imageId; /\* " " \*/

struct \_IplTileInfo \*tileInfo; /\* " " \*/

int imageSize; /\* Image data size in bytes

(==image->height\*image->widthStep

in case of interleaved data)\*/

char \*imageData; /\* Pointer to aligned image data. \*/

int widthStep; /\* Size of aligned image row in bytes. \*/

int BorderMode[4]; /\* Ignored by OpenCV. \*/

int BorderConst[4]; /\* Ditto. \*/

char \*imageDataOrigin; /\* Pointer to very origin of image data

(not necessarily aligned) -

needed for correct deallocation \*/

}

IplImage;

1. OpenCV常用类和数据结构
   1. Point类 – 点

Point point；

Point.x = 10;

Point.y = 8;

或者 Point point = point(10,8);

另外Point\_<int> ;Point2i; Point 互相等价, Point\_<float>; Point2f

* 1. Scalar类 – 颜色

Scalar(a,b,c) ; abc 即RGB; 用于传递像素值，还有第四个元素是透明，可以不写出来。

* 1. Size 类 – 尺寸

Size\_类模板提供了一些构造函数，具体可以查看D:\Program Files (x86)\opencv\sources\modules\core\include\opencv2\core.cpp;

template<typename \_Tp> class Size\_

{

public:

typedef \_Tp value\_type;

//! various constructors

Size\_();

Size\_(\_Tp \_width, \_Tp \_height);

Size\_(const Size\_& sz);

Size\_(const CvSize& sz);

Size\_(const CvSize2D32f& sz);

Size\_(const Point\_<\_Tp>& pt);

Size\_& operator = (const Size\_& sz);

//! the area (width\*height)

\_Tp area() const;

//! conversion of another data type.

template<typename \_Tp2> operator Size\_<\_Tp2>() const;

//! conversion to the old-style OpenCV types

operator CvSize() const;

operator CvSize2D32f() const;

\_Tp width, height; // the width and the height

};

最常用的是 Size\_(\_Tp \_width, \_Tp \_height);

定义模板类型的宽度和高度。

* 1. Rect 类 – 矩形

Rect\_类，成员变量x、y、width、height，分别为左上角点的坐标和矩形的宽和高。常用的成员函数有Size()返回值为一个Size，area()返回矩形的面积，contains(Point)用来判断点是否在矩形内，inside(Rect)函数判断矩形是否在该矩形内，tl()返回左上角点坐标，br()返回右下角点坐标。值得注意的是，如果想求两个矩形的交集，并集，可以用如下格式

* 求两个矩形的交集和并集

Rect rect = rect1 & rect2;

Rect rect = rect1 | rect2;

* 矩形的平移和缩放

Rect RectShift = rect1 + point;

Rect RectScale = rect1 + size;

* 返回矩形的面积

AreaRect = area();

* 判断点是否在矩形内

bPointInRect = Contains(Point);

* 判断矩形是否在矩形内

bRectInRect = Inside(Rect);

* 1. CvtColor 类 – 颜色空间

CvtColor() 是OpenCV里颜色空间转换函数，实现RGB、HSV HIS 空间的转换

也可转化为灰度图像。

函数原型：

void cv::cvtColor( InputArray \_src, OutputArray \_dst, int code, int dcn )

|  |  |
| --- | --- |
| 参数1 | 输入图像 |
| 参数2 | 输出图像 |
| 参数3 | 颜色空间转换标识符 |
| 参数4 | 目标图像通道数 |

颜色空间的转换标识符类型

enum

{

CV\_BGR2BGRA =0,

CV\_RGB2RGBA =CV\_BGR2BGRA,

CV\_BGRA2BGR =1,

CV\_RGBA2RGB =CV\_BGRA2BGR,

CV\_BGR2RGBA =2,

CV\_RGB2BGRA =CV\_BGR2RGBA,

CV\_RGBA2BGR =3,

CV\_BGRA2RGB =CV\_RGBA2BGR,

CV\_BGR2RGB =4,

CV\_RGB2BGR =CV\_BGR2RGB,

CV\_BGRA2RGBA =5,

CV\_RGBA2BGRA =CV\_BGRA2RGBA,

CV\_BGR2GRAY =6,

CV\_RGB2GRAY =7,

CV\_GRAY2BGR =8,

CV\_GRAY2RGB =CV\_GRAY2BGR,

CV\_GRAY2BGRA =9,

CV\_GRAY2RGBA =CV\_GRAY2BGRA,

CV\_BGRA2GRAY =10,

CV\_RGBA2GRAY =11,

…

CV\_RGBA2YUV\_I420 = 129,

CV\_BGRA2YUV\_I420 = 130,

CV\_RGBA2YUV\_IYUV = CV\_RGBA2YUV\_I420,

CV\_BGRA2YUV\_IYUV = CV\_BGRA2YUV\_I420,

CV\_RGB2YUV\_YV12 = 131,

CV\_BGR2YUV\_YV12 = 132,

CV\_RGBA2YUV\_YV12 = 133,

CV\_BGRA2YUV\_YV12 = 134,

CV\_COLORCVT\_MAX = 135

};

* 1. 一些常用的知识点

|  |  |  |
| --- | --- | --- |
| Matx类 | 轻量级的Mat 使用前规定好大小，如Mat23f 即是定义一个2\*3的float型的Mat | |
| Vec类 | Matx 的派生类，一维的Matx | |
| Range类 | Range::all() 是matlab符号，Range(a,b) 是Matlab中的a:b | |
| 防溢出函数 | AlignPtr(), alignSize(), allocate(), deallocate(), fastMalloc(), fastFree() | |
| 数学库函数 | FastAtan2 | 计算向量角度的函数 |
| cubeRoot | 计算立方根的函数 |
| cvCeil | 向上取整函数 |
| cvFloor | 向下取整函数 |
| cvRound | 四舍五入函数 |
| cvIsInf | MatLab函数，判断自变量是否无求大 |
| cvIsNaN | MatLab函数，判断自变量是否不是一个数 |
| 显示文字的函数 | getTextSize, cvInitFont, putText | |
| 作图函数 | Circle, clipLine,ellipse,ellipse2Poly,line,rectangle,polylines,类LineIterator | |
| 填充函数 | fillCibvexPoly, fillPoly | |
| 随机数 | RNG() | |

1. OpenCV基本图形绘制
   1. 基本的绘图函数

|  |  |
| --- | --- |
| 绘制直线 | void line( Mat& img, Point pt1, Point pt2, const Scalar& color,  int thickness, int line\_type, int shift ) |
| 绘制椭圆 | void ellipse( Mat& img, Point center, Size axes,  double angle, double start\_angle, double end\_angle,  const Scalar& color, int thickness, int line\_type, int shift ) |
| 绘制矩形 | void rectangle( Mat& img, Rect rec,  const Scalar& color, int thickness,  int lineType, int shift ) |
| 绘制圆形 | void circle( Mat& img, Point center, int radius,  const Scalar& color, int thickness, int line\_type, int shift ) |
| 绘制填充 | void fillPoly( Mat& img, const Point\*\* pts, const int\* npts, int ncontours,  const Scalar& color, int line\_type,  int shift, Point offset ) |

* 1. DrawEillpse函数的写法

OpenCV 官方例子：

绘制不同角度，尺寸相同的椭圆。

void DrawEllipse( Mat img, double angle )

{

int thickness = 2;

int lineType = 8;

ellipse( img,

Point( WINDOW\_WIDTH/2, WINDOW\_WIDTH/2 ),//中心点

Size( WINDOW\_WIDTH/4, WINDOW\_WIDTH/16 ),//位于矩形范围内

angle,//椭圆旋转角度

0, //画椭圆的起始弧度

360,// 椭圆的终止弧度

Scalar( 255, 129, 0 ),// 图形的颜色BGR，此为蓝色

thickness,// 线宽为2

lineType );// 线型为8 即8联通线型

}

* 1. DrawFilledCircle的写法

实现实心圆绘制:

void DrawFilledCircle( Mat img, Point center )

{

int thickness = -1;

int lineType = 8;

circle( img,

center, //圆心

WINDOW\_WIDTH/32, //圆的半径

Scalar( 0, 0, 255 ), //图形的颜色BGR，此为红色

thickness, //线宽为2

lineType ); //线型为8 即8联通线型

}

* 1. DrawPolygon的写法

实现凹多边形的绘制:

void DrawPolygon( Mat img )

{

int lineType = 8;

//创ä¡ä建¡§一°?些?点Ì?

Point rookPoints[1][20];

rookPoints[0][0] = Point( WINDOW\_WIDTH/4, 7\*WINDOW\_WIDTH/8 );

rookPoints[0][1] = Point( 3\*WINDOW\_WIDTH/4, 7\*WINDOW\_WIDTH/8 );

rookPoints[0][2] = Point( 3\*WINDOW\_WIDTH/4, 13\*WINDOW\_WIDTH/16 );

rookPoints[0][3] = Point( 11\*WINDOW\_WIDTH/16, 13\*WINDOW\_WIDTH/16 );

rookPoints[0][4] = Point( 19\*WINDOW\_WIDTH/32, 3\*WINDOW\_WIDTH/8 );

rookPoints[0][5] = Point( 3\*WINDOW\_WIDTH/4, 3\*WINDOW\_WIDTH/8 );

rookPoints[0][6] = Point( 3\*WINDOW\_WIDTH/4, WINDOW\_WIDTH/8 );

rookPoints[0][7] = Point( 26\*WINDOW\_WIDTH/40, WINDOW\_WIDTH/8 );

rookPoints[0][8] = Point( 26\*WINDOW\_WIDTH/40, WINDOW\_WIDTH/4 );

rookPoints[0][9] = Point( 22\*WINDOW\_WIDTH/40, WINDOW\_WIDTH/4 );

rookPoints[0][10] = Point( 22\*WINDOW\_WIDTH/40, WINDOW\_WIDTH/8 );

rookPoints[0][11] = Point( 18\*WINDOW\_WIDTH/40, WINDOW\_WIDTH/8 );

rookPoints[0][12] = Point( 18\*WINDOW\_WIDTH/40, WINDOW\_WIDTH/4 );

rookPoints[0][13] = Point( 14\*WINDOW\_WIDTH/40, WINDOW\_WIDTH/4 );

rookPoints[0][14] = Point( 14\*WINDOW\_WIDTH/40, WINDOW\_WIDTH/8 );

rookPoints[0][15] = Point( WINDOW\_WIDTH/4, WINDOW\_WIDTH/8 );

rookPoints[0][16] = Point( WINDOW\_WIDTH/4, 3\*WINDOW\_WIDTH/8 );

rookPoints[0][17] = Point( 13\*WINDOW\_WIDTH/32, 3\*WINDOW\_WIDTH/8 );

rookPoints[0][18] = Point( 5\*WINDOW\_WIDTH/16, 13\*WINDOW\_WIDTH/16 );

rookPoints[0][19] = Point( WINDOW\_WIDTH/4, 13\*WINDOW\_WIDTH/16 );

const Point\* ppt[1] = { rookPoints[0] };

int npt[] = { 20 };

fillPoly( img,

ppt, //多边形的顶点集

npt, //多边形的顶点数目

1, // 多边形的数量仅为1

Scalar( 255, 255, 255 ), //图形的颜色BGR，此为白色

lineType ); //线型为8 即8联通线型

}

* 1. DrawLine的写法

实现了线的绘制:

void DrawLine( Mat img, Point start, Point end )

{

int thickness = 2;

int lineType = 8;

line( img,

start,

end,

Scalar( 0, 0, 0 ), // 图形的颜色BGR，此为黑色

thickness, ////线宽为2

lineType ); //线型为8 即8联通线型

}