OpenCV--基本数据类型

基本数据类型

openCv提供很多基本的数据类型，这些数据类型并非是C原生的数据类型，但是结构相对简单。可以视作是OpenCV的原子数据类型。可以在/include/cxtypes.h中查看每个数据类型的基本格式。

基本上概括可以分为以下几个数据类型：

名称	结构	意义
CvPoint	Integer x; Integer y;	图像中的点
CvPoint2D32f	Float x; Float y;	二维图像中的点，是CvPoint的派生
CVPoint3D32f	float x,y,z;	三维图像中的点
cvsize	Int width,height	图像的尺寸
Cvrect	Int x,y,width,height	图像中的区域
cvscalar	Double val[4]	对应的RGBA值

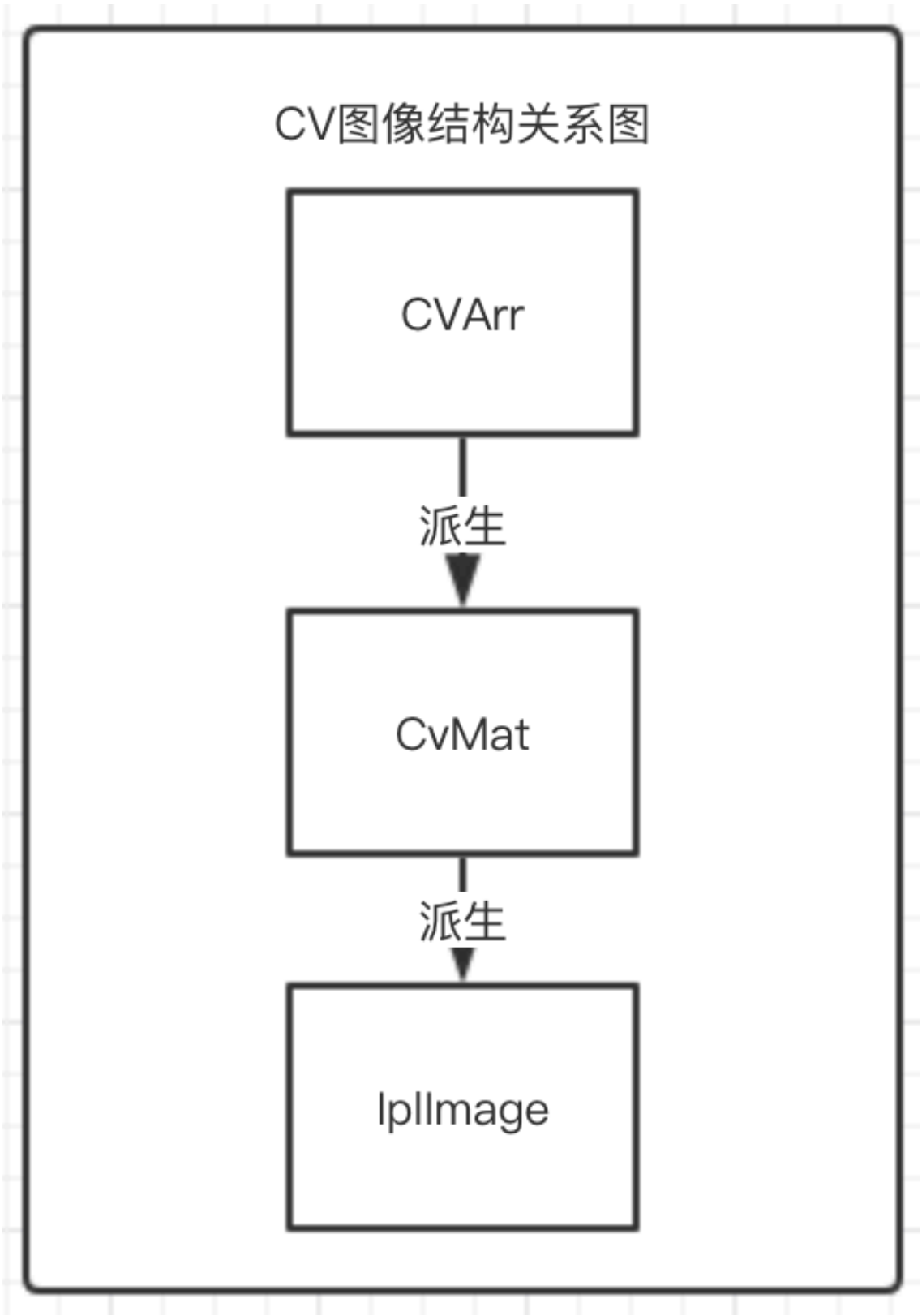
cvsize和cvPoint一致，都具有派生的cvsize2D32f和cvsize3D32f。值得注意的是CVScalar,如果是内存不是瓶颈的情况下，cvScalar经常用来替代其余几种参数类型。他的val指针可以用来指向4个双精度浮点数数组指针。

cvScalar是一个特殊的数据结构，它具有3个构造函数，第一个是 cvScalar（）他需要将一个、两个、三个、四个参数并将数据传递到val对应的数组指针相应的元素。第二个构造函数是cvRealScalar（），他需要一个参数，用于传递到val[0],最后一个是cvScalarAll，他需要一个参数，并且把val4]全部都设置为这个参数。

矩阵和图像类型

使用OpenCV时，会经常使用到IplImage数据类型，IplImage是我们用来为通常所说的图像进行编码的基本数据结构，这些图像可能是灰度图，彩色图，4通道图，其中每个通道可以包含然后整数或者浮点数。因此该类型是最常见的，比3通道RGB图更通用。

OpenCV提供了大量的图片处理运算方法，比如缩放图片、单通道提取、特定通道最大值最小值。两个图形求和等基本函数方法。图像数据结构CvArr，CvMat,IplImage之间的继承关系如下：



OpenCV是由C语言编写，但是使用了面向对象的思想。这里主要讲一下CvMat数据结构，CvMat和IPlImage之间的关系就好比是C++的继承关系一样，实质上IplImage可以看做是从CvMat中派生而来的。首先理解这三种数据结构的基本关系，CvArr是一个基本的抽象

类，CVMat由它派生，在很多函数都是由最基础的抽象类作为传递成员参数，因此，可以直接传递CvMat和IplImage数据类型进行处理。

- ## CVMat矩阵结构

在学习之前，首先要明白，在OpenCV中，是没有向量（vector）这个结构的。如果要使用vector，都会转做是一个列矩阵。另外一个，是openCV的矩阵概念比起线性代数的矩阵概念更加抽象，尤其是矩阵元素，并非只能取简单的数值类型。比如用于建立一个二维矩阵的历程如下：

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

这里的type可以是任何定义类型，预定义结构比如：CV_<bit_depth> 于是，矩阵可以是CV_32FC1 32位浮点数类型，或者是无符号整型 （CV_8UC3。CVMat可以通过vGetSize来获取Mat的Size信息。在types.h，Mat的定义如下：

```
typedef struct CvMat
{
    int type;
    int step;

    /* for internal use only */
    int* refcount;
    int hdr_refcount;

    union
    {
        uchar* ptr;
        short* s;
        int* i;
        float* fl;
        double* db;
    } data;

#ifdef __cplusplus
    union
    {
        int rows;
        int height;
    };

    union
    {
        int cols;
        int width;
    };
#else
    int rows;
    int cols;
#endif
```

```
#ifdef __cplusplus
    CvMat() {}
    CvMat(const CvMat& m) { memcpy(this, &m, sizeof(CvMat));}
    CvMat(const cv::Mat& m);
#endif

}
```

CvMat;

矩阵数据的存取

访问矩阵中的数据有3种方法，简单的方式，麻烦的方式和合理的方式。

- 简单的方式

从矩阵中得到一个元素的最简单方法是利用宏定义 CV_MAT_ELEM().这个宏传入矩阵，待提取的元素的类型，行和列数四个参数，返回则是提取的元素值。例如：

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

其次，还有一个此类的宏定义，CV_MAT_PTR(). CV_MAT_PTR()传入矩阵，待返回的行列号这三个参数，返回指向这个元素的指针。和前者对比最大的区别在于，候着在引用的时候直接转换成了对应的类型。如果需要同事读取和设置数据，可以直接调用CV_MAT_ELEM_PTR().但是在这种情况下，必须要讲指针转换成指定的类型。例如：

```
        CVMat* mat = cvCreateMat(5,5,CV_32FC1);
        Float elem_3_2 = 7.7;
        *((float*)CV_MAT_ELEM_PTR(*mat,3,2)) = elem_3_2;
```

- 麻烦的方式

简单的方式只适合访问1维或者2维数组，openCV也提供了处理多维数组的方法。事实上，OpenCV可以支持N维的数组，这个N可以是任意值。

在实际的操作中，有不少cvPtr*D和cvPtrND的函数方法。cvPtr*D的方法包括了，cvPtr1D、cvPtr2D、cvPtr3D、cvPtrND等。这几个函数可以接受CVArr*指针数据。

- 恰当的方法

实际上，通过这类隐射方法去修改效率并不是很高，特别是在计算机图形学的运算中，大部分伴随着成万次的运算量。如果要提升效率则需要使用自己的指针。

如果需要对一个数组进行操作，只需要知道数据是按照光栅扫描顺序存储的，列是最

快变化的量。通道是相互交错的，这意味着，对于一个多个通道而言，他们变化的速度任然比较快。

测试用例：

```
#include <iostream>
#include "opencv/highgui.h"
#include "opencv/cv.h"
using namespace std;
using namespace cv;
#define IMAGE_LOAD_PATH  "/Users/genesis/Pictures/IMG_3393.JPG"
float sum(const CvMat* mat)
{
    float sum = 0;
    for (int i = 0; i<mat->rows; i++) {
        const float *ptr = (const float* )(mat->data.ptr+i*mat->step);
        for (int col= 0; col<mat->cols; col++) {
            sum += *ptr++;
        }
    }
    return sum;
}
int main(int argc, const char * argv[]) {
    CvMat * mat = cvCreateMat(3, 3, CV_32FC1);
    for (int i=0; i<3; i++) {
        for (int j=0; j<3; j++) {
            *((float*)CV_MAT_ELEM_PTR(*mat, i, j))= j;
        }
    }
    std::cout<<"the mat count sum :"<<sum(mat)<<endl;
    return 0;
}
```

也可以精简一下上述的代码，使用上述指针合理的方式去做。

```

    CvMat * mat = cvCreateMat(3, 3, CV_32FC1);
    for (int i=0; i<3; i++) {
          float *ptr = ( float* )(mat->data.ptr+i*mat->step);
        for (int j=0; j<3; j++) {
//            *((float*)CV_MAT_ELEM_PTR(*mat, i, j))= j;
            *ptr++= j;
        }
    }
```

基本上CVMat的方式和Android/MFC等ARGB数据格式类似的读写操作。

IplImage数据结构

实际上，它是一个cvMat对象，但是他有一些其它的成员变量可以解释为图像。这个结构最初被定义为Intel图像处理库（IPL）的一部分。IPLImage结构的准确定义如下：

```
typedef struct
#ifdef __cplusplus
  CV_EXPORTS
#endif
_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 */
#ifdef __cplusplus
    _IplImage() {}
    _IplImage(const cv::Mat& m);
#endif
```

```
}
IplImage;
```

其中，最重要的是width、height这两个变量。其次，deptch和nchannals。deptch的变量取值于ipl.h定义中的一组数据。在图像中，我们往往将图像的深度和通道数分开处理。而在矩阵中，我们往往同时用矩阵数据表达他们。深度值如下所示：

宏定义	图像类型
#define IPL_DEPTH_1U 1	无符号1位整数
#define IPL_DEPTH_8U 8	无符号8位整数
#define IPL_DEPTH_16U 16	无符号16位整数
#define IPL_DEPTH_32F 32	无符号32位整数
#define IPL_DEPTH_8S (IPL_DEPTH_SIGN | 8)	有符号8位
#define IPL_DEPTH_16S (IPL_DEPTH_SIGN| 16)	有符号16位
#define IPL_DEPTH_32S (IPL_DEPTH_SIGN|32)	有符号32位

通道数nChannels可取的值包括了1，2，3，4.

其次是origin和dataOrder。origin可以是两个取值，IPL_ORIGIN_TL或者是IPL_ORIGIN_BL，分别设置了坐标原点位于图像的左下角或者是左上角。在机械视觉的领域中，常常因为坐标原点的不统一，导致了一些识别算法的异常。

dataorder的取值是IPL_DATA_ORDER_PIXEL或者IPL_DATA_ORDER_PLANE,所指向的是数据是讲像素点不同通道交错排到一起，候着是讲所有像素同通道值排在一起。形成通道平面，在把平面排到一起。

参数widthstep表示了相对同列之间的字节数。

与CVMat的成员data比起来，IplImage和CVMat之间最重要的区别在于imageData.cvMat的data元素类型是联合类型，所以你必须要说明使用的数据类型。imageData指针是字节类型（uchar*）我们是已知他是指向了uchar类型的数据。在计算图像算法的时候，你可以用widthstep，而不是记录多少字节的方式去做。计算偏移量。

ROI和widthstep在实际的工程中有很多有用的效果，提高了机器视觉的执行速度，这是由于，这方法是对局部的图像进行算法处理。要设置或者取消就需要使用cvsetImageROI和cvResetImageROI函数。具体测试代码如下：

```
int main(int argc, const char * argv[]) {
    IplImage *input = cvLoadImage(IMAGE_LOAD_PATH);
    int x= 100;
    int y =100;
    int width = 150;
    int height = 150;
    int add =100;
    cvSetImageROI(input, cvRect(x, y, width, height));
    cvAddS(input, cvScalar(add), input);
    cvResetImageROI(input);

    cvShowImage("roi ",input);

    //fixme 清空Input Image
    cvReleaseImage(&input);
    cvWaitKey();
    return 0;
}
```

上述代码是对图片局部做处理，对一个通道分量加了100。