**[OpenCV2:Mat属性type，depth，step](https://www.cnblogs.com/wangguchangqing/p/4016179.html)**

在OpenCV2中Mat类无疑使占据着核心地位的

**Mat的作用**

The class Mat represents an n-dimensional dense numerical single-channel or multi-channel array. It can be used to store real or complex-valued vectors and matrices, grayscale or color images, voxel volumes, vector fields, point clouds, tensors, histograms (though, very high-dimensional histograms may be better stored in a SparseMat ).

上面的一段话引用自官方的文档，Mat类用于表示一个多维的单通道或者多通道的稠密数组。能够用来保存实数或复数的向量、矩阵，灰度或彩色图像，立体元素，点云，张量以及直方图（高维的直方图使用SparseMat保存比较好）。简而言之，Mat就是用来保存多维的矩阵的。

**Mat的常见属性**

* **data** uchar型的指针。Mat类分为了两个部分:矩阵头和指向矩阵数据部分的指针，data就是指向矩阵数据的指针。
* **dims** 矩阵的维度，例如5\*6矩阵是二维矩阵，则dims=2，三维矩阵dims=3.
* **rows**  矩阵的行数
* **cols**   矩阵的列数
* **size** 矩阵的大小，size(cols,rows),如果矩阵的维数大于2，则是size(-1,-1)
* **channels** 矩阵元素拥有的通道数，例如常见的彩色图像，每一个像素由RGB三部分组成，则channels = 3

下面的几个属性是和Mat中元素的数据类型相关的。

* **type**   
  表示了矩阵中元素的类型以及矩阵的通道个数，它是一系列的预定义的常量，其命名规则为CV\_(位数）+（数据类型）+C（通道数）。具体的有以下值：

|  |  |  |  |
| --- | --- | --- | --- |
| CV\_8UC1 | CV\_8UC2 | CV\_8UC3 | CV\_8UC4 |
| CV\_8SC1 | CV\_8SC2 | CV\_8SC3 | CV\_8SC4 |
| CV\_16UC1 | CV\_16UC2 | CV\_16UC3 | CV\_16UC4 |
| CV\_16SC1 | CV\_16SC2 | CV\_16SC3 | CV\_16SC4 |
| CV\_32SC1 | CV\_32SC2 | CV\_32SC3 | CV\_32SC4 |
| CV\_32FC1 | CV\_32FC2 | CV\_32FC3 | CV\_32FC4 |
| CV\_64FC1 | CV\_64FC2 | CV\_64FC3 | CV\_64FC4 |

* 这里U（unsigned integer）表示的是无符号整数，S（signed integer）是有符号整数，F（float）是浮点数。   
  例如：CV\_16UC2，表示的是元素类型是一个16位的无符号整数，通道为2.   
  C1，C2，C3，C4则表示通道是1,2,3,4   
  type一般是在创建Mat对象时设定，如果要取得Mat的元素类型，则无需使用**type**，使用下面的**depth**
* **depth**   
  矩阵中元素的一个通道的数据类型，这个值和type是相关的。例如 type为 CV\_16SC2，一个2通道的16位的有符号整数。那么，depth则是CV\_16S。depth也是一系列的预定义值，   
  将type的预定义值去掉通道信息就是depth值:   
  CV\_8U CV\_8S CV\_16U CV\_16S CV\_32S CV\_32F CV\_64F
* **elemSize**矩阵一个元素占用的字节数，例如：type是CV\_16SC3，那么elemSize = 3 \* 16 / 8 = 6 bytes
* **elemSize1**矩阵元素一个通道占用的字节数，例如：type是CV\_16CS3，那么elemSize1 = 16  / 8 = 2 bytes = elemSize / channels

下面是一个示例程序，具体说明Mat的各个属性：

[复制代码](javascript:void(0);)

Mat img(3, 4, CV\_16UC4, Scalar\_<uchar>(1, 2, 3, 4));

cout << img << endl;

cout << "dims:" << img.dims << endl;

cout << "rows:" << img.rows << endl;

cout << "cols:" << img.cols << endl;

cout << "channels:" << img.channels() << endl;

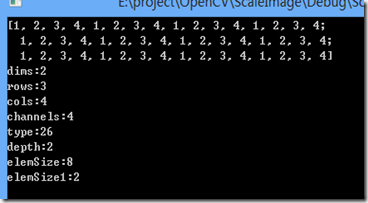
cout << "type:" << img.type() << endl;

cout << "depth:" << img.depth() << endl;

cout << "elemSize:" << img.elemSize() << endl;

cout << "elemSize1:" << img.elemSize1() << endl;

[复制代码](javascript:void(0);)

首先创建了一个3\*4的具有4个通道的矩阵，其元素类型是CV\_16U。Scalar\_是一个模板向量，用来初始化矩阵的每个像素，因为矩阵具有4个通道，Scalar\_有四个值。其运行结果：   
[](https://images0.cnblogs.com/blog/439761/201410/101612359688147.png)运行结果首先打印了Mat中的矩阵，接着是Mat的各个属性。注意其type = 26,而depth = 2。这是由于上面所说的各种预定义类型   
例如，CV\_16UC4，CV\_8U是一些预定义的常量。

**step**

Mat中的step是一个MStep的一个实例。其声明如下：

[复制代码](javascript:void(0);)

struct CV\_EXPORTS MStep

{

MStep();

MStep(size\_t s);

const size\_t& operator[](int i) const;

size\_t& operator[](int i);

operator size\_t() const;

MStep& operator = (size\_t s);

size\_t\* p;

size\_t buf[2];

protected:

MStep& operator = (const MStep&);

};

[复制代码](javascript:void(0);)

从其声明中可以看出，MStep和size\_t有比较深的关系。用size\_t作为参数的构造函数和重载的赋值运算符

MStep(size\_t s);

MStep& operator = (size\_t s);

向size\_t的类型转换以及重载的[ ]运算符返回size\_t

const size\_t& operator[](int i) const;

size\_t& operator[](int i);

size\_t的数组以及指针

size\_t\* p;

size\_t buf[2];

那么size\_t又是什么呢，看代码

typedef unsigned int size\_t;

size\_t就是无符号整数。

再看一下MStep的构造函数，就可以知道其究竟保存的是什么了。

inline Mat::MStep::MStep(size\_t s) { p = buf; p[0] = s; p[1] = 0; }

从MStep的定义可以知道，buff是一个size\_t[2]，而p是size\_t \*，也就是可以把MStep看做一个size\_t[2]。那么step中保存的这个size\_t[2]和Mat中的数据有何种关系呢。

step[0]是矩阵中一行元素的字节数。

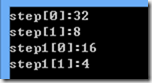
step[1]是矩阵中一个元素的自己数，也就是和上面所说的elemSize相等。

上面说到，Mat中一个uchar\* data指向矩阵数据的首地址，而现在又知道了每一行和每一个元素的数据大小，就可以快速的访问Mat中的任意元素了。下面公式：

**step1**

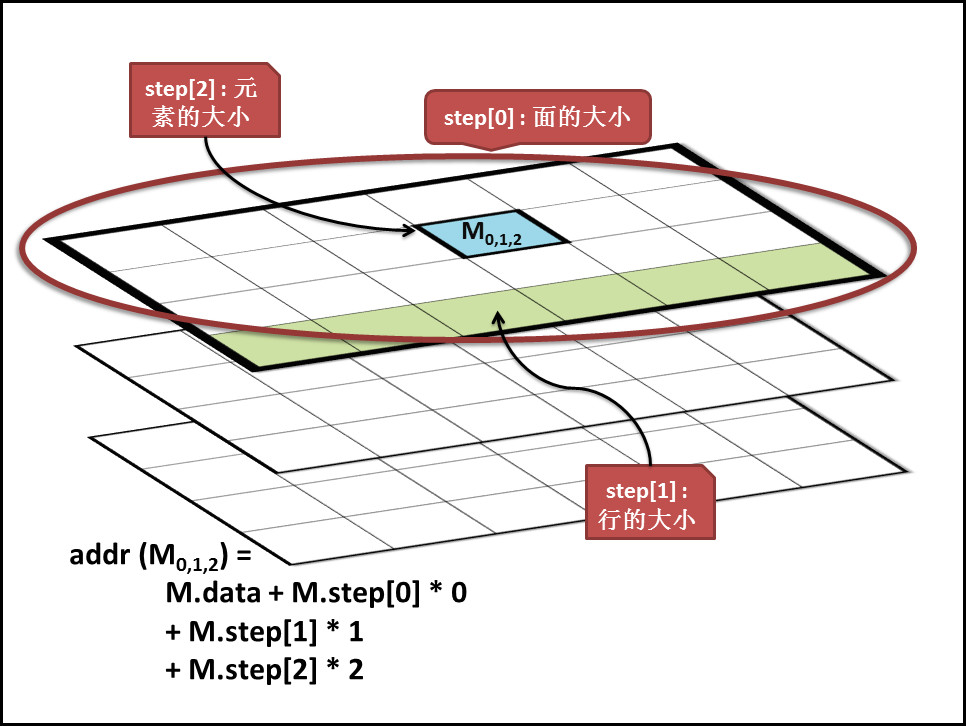
规整化的step，值为step / elemSize1。 定义如下：

inline size\_t Mat::step1(int i) const { return step.p[i]/elemSize1(); }

仍以上例代码中定义的img为例，来看下step,step1具体的值：   
[](https://images0.cnblogs.com/blog/439761/201410/101612393436479.png)img（3\*4）的type是CV\_16UC4,step[0]是其一行所占的数据字节数4 \*4 \* 16 / 8  = 32.   
step[1] 是一个元素所占的字节数，img的一个元素具有4个通道，故：4 \* 16 / 8 = 2   
step1 = step / elemSize1，elemSize1是元素的每个通道所占的字节数。

**N维的step(N > 2)**

上面分析step是一个size\_t[2]，实际不是很正确，正确的来说step应该是size\_t[dims]，dims是Mat的维度，所以对于上面的二维的Mat来说，step是size\_t[2]也是正确的。   
下面就对三维的Mat数据布局以及step（维度大于3的就算了吧）。



上图引用自[http://ggicci.blog.163.com/blog/static/210364096201261052543349/](http://ggicci.blog.163.com/blog/static/210364096201261052543349/" \o "http://ggicci.blog.163.com/blog/static/210364096201261052543349/)  搜集资料时发现了这幅图，一切就变的简单了 眨眼  感谢作者 **Ggicci**

三维的数据在Mat中是按面来存储的，上图描述的很清晰，这里不再多说。   
上面言道，step是一个size\_t[dims]，dims是维度。so，三维的step就是size\_t[3]。其余的不多说了，看图就有了。下面来创建一个三维的Mat，实际看看

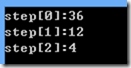
int dims[3] = { 3, 3, 3 };

Mat src(3, dims, CV\_16SC2, Scalar\_<short>(1,2));

cout << "step[0]:" << src.step[0] << endl;

cout << "step[1]:" << src.step[1] << endl;

cout << "step[2]:" << src.step[2] << endl;

首先创建一个3\*3\*3，depth为CV\_16S的两通道的Mat   
step[0]是一个数据面的大小  3 \* 3 \* (16 / 8 ) \* 2 = 36   
step[1]是一行数据的大小 3 \* (16 / 8 ) \* 2 = 12   
step[2]是一个元素的大小 2 \* (16 / 8) = 4   
[](https://images0.cnblogs.com/blog/439761/201410/101612422494967.png)  
PS: 三维的Mat 不能使用 <<运算符进行输出的。

over