湖南科技大学课程教案
（章节、专题首页）

授课教师：王志喜　　职称：副教授　　单位：计算机科学与工程学院

课程名称	计算机图形图像技术
章节、专题	OpenCV的核心功能与用户接口
教学目标及基本要求	掌握基本的OpenCV程序设计技术以及OpenCV基础数据结构的相关操作。
教学重点	OpenCV基础数据结构，OpenCV矩阵的基础操作
教学难点	OpenCV基础数据结构
教学内容与时间分配	（1）OpenCV GUI命令（1课时） （2）OpenCV基础数据结构（1课时） （3）OpenCV矩阵的基础操作（1.2课时） （4）OpenCV绘图命令（0.8课时） 共计4课时。
习　　题	第11.5.1节（程序设计题）。

第11章　OpenCV的核心功能与用户接口

11.1　OpenCV GUI命令

HighGUI只是用来建立快速软件原型或试验用的。它提供了简单易用的图形用户接口，但是功能并不强大，也不是很灵活。

【注】本节暂时不对函数原型中出现的基础数据结构进行介绍，这些基础数据结构将在“OpenCV基础数据结构”一节中介绍。

11.1.1 窗口管理

1. 创建窗口

【函数原型】void namedWindow(const string &name);
【功能】创建一个可以放置图像和滑块的窗口，可以通过名字引用该窗口。
【参数】name是窗口名字，用来区分不同的窗口，并显示为窗口标题。
【说明】如果已经存在这个名字的窗口，该函数不做任何事情。

2. 显示图像

【函数原型】void imshow(const string &name, InputArray image);
【功能】在指定窗口中显示图像。
【参数】

- name：窗口的名字。
- image：待显示的图像。

【说明】

- 可显示彩色或灰度的字节图像和浮点数图像。
- 彩色图像数据按BGR顺序存储。

11.1.2 读写图像

1. 读入图像

【函数原型】Mat imread(const String &filename, int flags = IMREAD_COLOR);

【功能】从指定文件读入图像。

【参数】

- filename: 图像文件名，支持BMP、DIB、JPEG、JPG、JPE、PNG、PBM、PGM、PPM、SR、RAS、TIFF、TIF等格式。
- flag: 通常可选
  - IMREAD_UNCHANGED（等于-1，不转换载入图像）
  - IMREAD_GRAYSCALE（等于0，载入为灰度图像）
  - IMREAD_COLOR（等于1，载入为彩色图像）

【说明】如果返回对象的empty()成员的调用结果为真，则表示图像文件载入失败。

2. 存储图像

【函数原型】bool imwrite(const string &filename, InputArray image);

【功能】保存图像到指定文件。

【参数】

- filename：指定的文件名。
- image：要保存的图像。

【说明】图像格式的选择依赖于filename的扩展名。只有单通道或者3通道（通道顺序为“BGR”）字节图像才可以使用这个函数保存。如果位深度、通道数或通道顺序等不符合要求，则必须进行转换。

11.1.3 输入设备

1. 响应键盘事件

【函数原型】int waitKey(int delay = 0);

【功能】等待按键事件。

【参数】delay表示延迟的毫秒数。

【说明】

- 该函数无限等待按键事件（delay<=0）或者延迟delay毫秒。
- 返回值为按键值，如果超过指定时间则返回-1。
- 该函数是HighGUI中获取和操作键盘事件的函数，在一般的事件处理中需要循环调用。

【举例】下列程序演示了对键盘按键的检测。程序运行结果如图11-1所示。

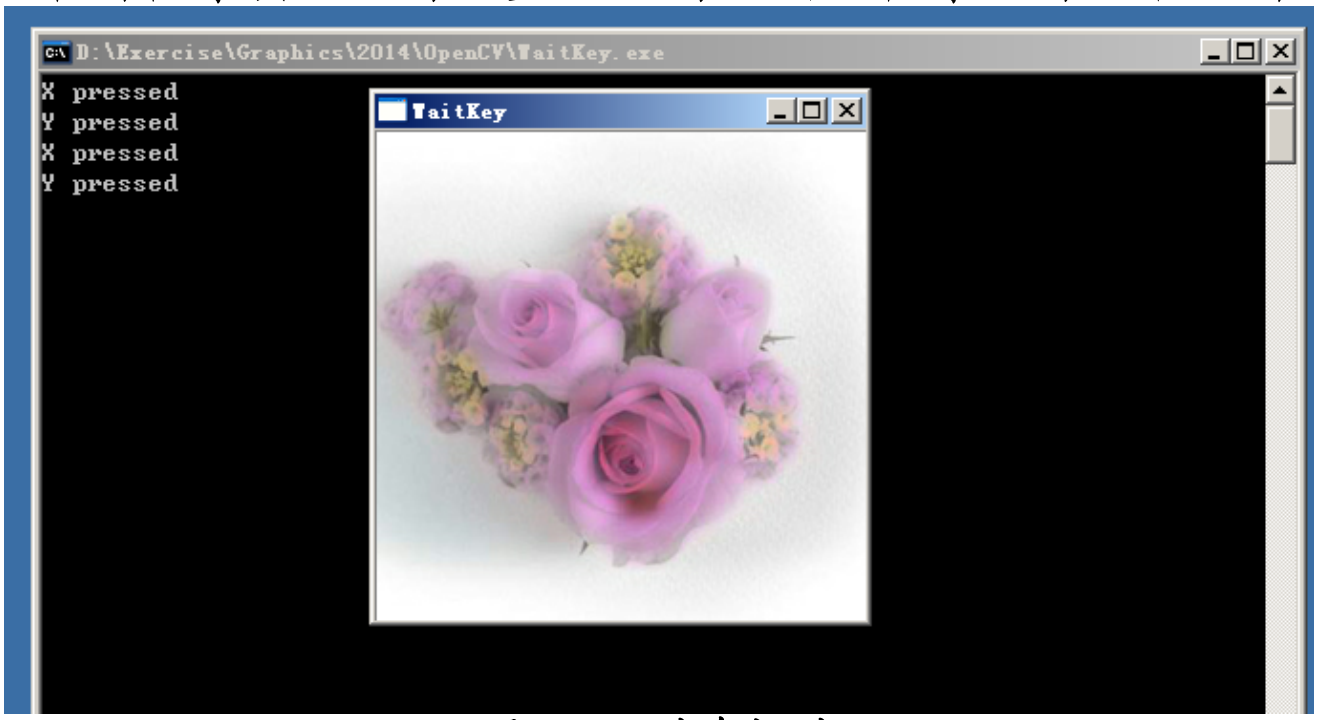


图11-1　键盘按键

```
// WaitKey.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
int main()
{   Mat im = imread("Flower.bmp"); // 载入彩色图像
    if(im.empty()) return -1; // 载入失败
    imshow("WaitKey", im); // 显示图像
    for(int c = waitKey(); c != 27; c = waitKey()) // 等待按Esc键
        cout << (char)toupper(c) << " pressed" << endl; // 显示大写字符
    imwrite("Flower.png", im); // 退出前保存为PNG文件
}
```

2. 响应鼠标事件

（1）鼠标事件响应函数的类型定义。

【类型定义】typedef void (*MouseCallback)(int event, int x, int y, int flags, void *userdata);

【参数】

- (x, y)是鼠标在图像坐标系的坐标（默认原点在上方，不是窗口坐标系）。
- userdata是用户定义的传递到setMouseCallback()函数的参数。
- event是下列值之一（见名知义，不再解释）。
  - EVENT_MOUSEMOVE
  - EVENT_LBUTTONDOWN
  - EVENT_RBUTTONDOWN
  - EVENT_MBUTTONDOWN
  - EVENT_LBUTTONUP
  - EVENT_RBUTTONUP
  - EVENT_MBUTTONUP
  - EVENT_LBUTTONDBLCLK
  - EVENT_RBUTTONDBLCLK
  - EVENT_MBUTTONDBLCLK

- flags是下列值的组合（见名知义，不再解释）。
  - EVENT_FLAG_LBUTTON
  - EVENT_FLAG_RBUTTON
  - EVENT_FLAG_MBUTTON
  - EVENT_FLAG_CTRLKEY
  - EVENT_FLAG_SHIFTKEY
  - EVENT_FLAG_ALTKEY

```
typedef void (*MouseCallback)(int event, int x, int y, int flags, void *userdata);
```

（2）注册鼠标事件响应函数。

【函数原型】void setMouseCallback(const string &winname, MouseCallback onMouse, void *userdata = 0);

【参数】

- winname：窗口的名字。
- onMouse：指定窗口里鼠标事件发生时调用的回调函数。
- userdata：用户定义的传递到回调函数的参数。

（3）举例说明。下列程序演示了对鼠标事件的响应。程序运行结果如图11-2所示。

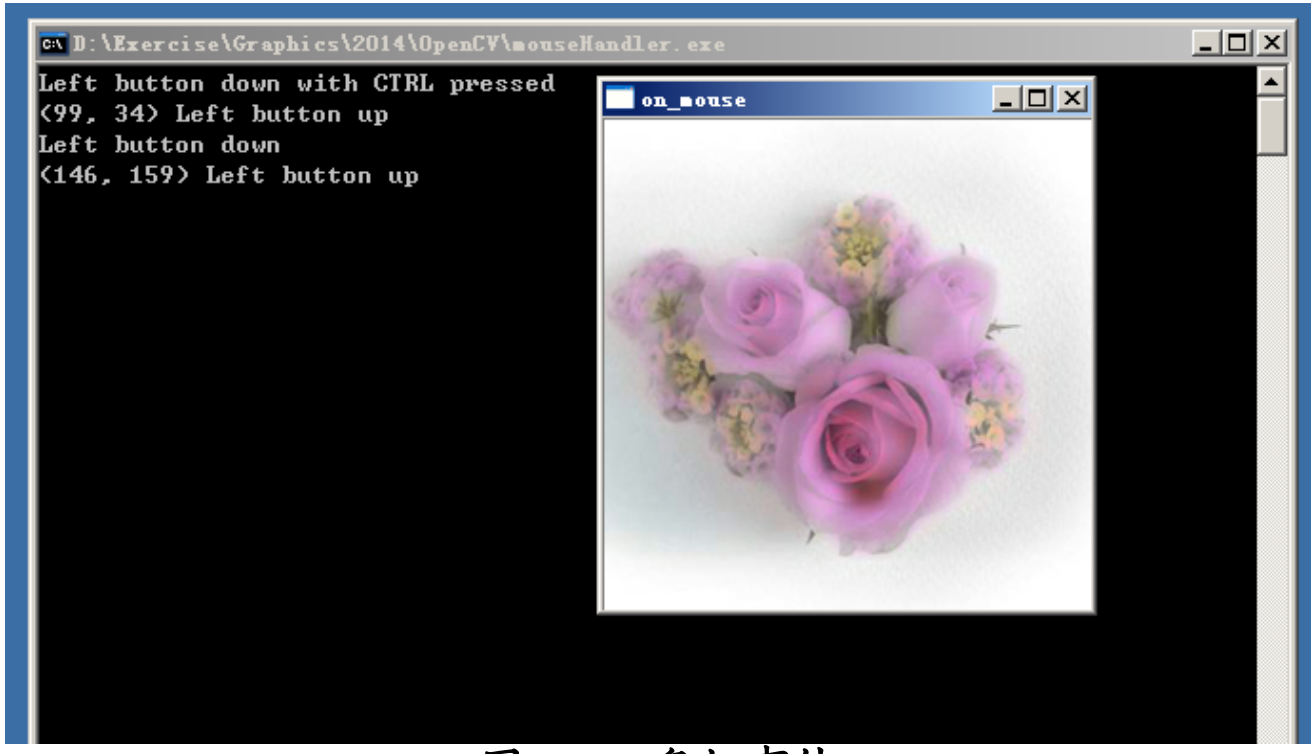


图11-2　鼠标事件

```
// MouseCallback.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
```

```
void on_mouse(int event, int x, int y, int flags, void *userdata)
{   if(event == EVENT_LBUTTONDOWN) // 鼠标左键按下
        if(flags & EVENT_FLAG_CTRLKEY) // 按下CTRL
            cout << "Left button down with CTRL pressed" << endl;
        else
            cout << "Left button down" << endl;
    else if(event == EVENT_LBUTTONUP) // 鼠标左键松开
        cout << "(" << x << ", " << y << ") Left button Up" << endl;
}
```

```
int main()
{   Mat im = imread("Flower.bmp"); // 载入彩色图像
    if(im.empty()) return -1; // 载入失败
    imshow("on_mouse", im); // 显示图像
    setMouseCallback("on_mouse", on_mouse);
    while(waitKey() != 27) {} // 等待按Esc键
}
```

3. 处理滑块事件

（1）滑块事件响应函数的类型定义。

【类型定义】typedef void (*TrackbarCallback)(int pos, void *userdata);

【参数】

- pos：滑块位置。
- userdata：用户定义的传递到createTrackbar()函数的参数。

（2）注册滑块事件响应函数。

【函数原型】int createTrackbar(const string &trackbarname, const string &winname, int *value, int count, TrackbarCallback onChange = 0, void *userdata = 0);

【功能】创建滑块，并指定滑块事件响应函数。

【参数】

- trackbarname：滑块名字。
- winname：窗口名字，这个窗口将成为滑块的父对象。
- value：可直接规定为NULL，非NULL值已弃用，不再介绍其用途。
- count：滑块位置的最大值。最小值一定是0。
- onChange：指定滑块位置改变时调用的回调函数。
- userdata：用户定义的传递到回调函数的参数。

【说明】该函数用指定的名字和范围创建滑块（显示在指定窗口的顶端），并指定当滑块位置改变时调用的回调函数。

（3）获取滑块当前位置。

【函数原型】int getTrackbarPos(const string &trackbarname, const string &winname);

【功能】返回指定滑块的当前位置。

【参数】

- trackbarname：滑块的名字。
- winname：滑块父窗口的名字。

（4）设置滑块当前位置。

【函数原型】void setTrackbarPos(const string &trackbarname, const string &winname, int pos);

【功能】设置指定滑块的位置。

【参数】

- trackbarname：滑块的名字。
- winname：滑块父窗口的名字。
- pos：新的位置。

（5）举例说明。下列程序演示了滑块的建立，滑块位置的读取和修改。程序运行结果如图11-3所示。

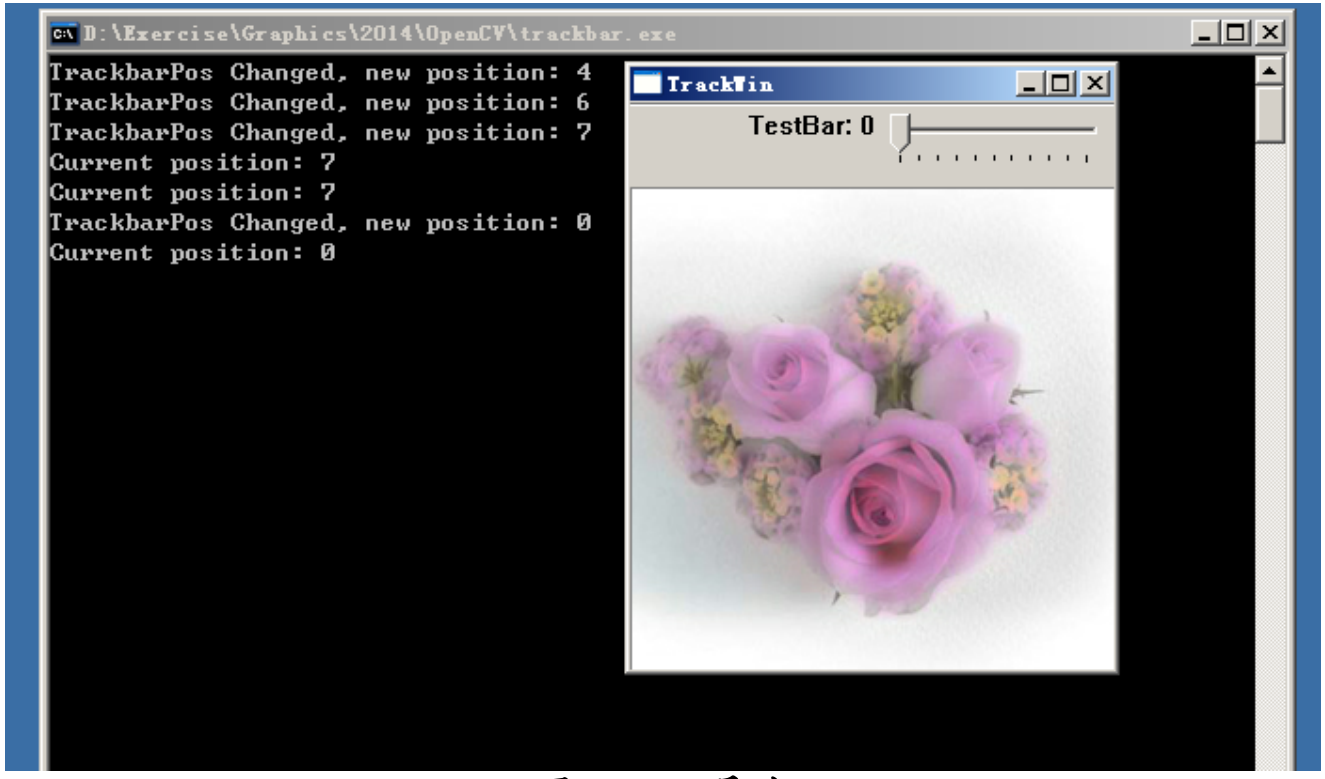


图11-3　滑块

```
// TrackBar.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
```

```
void on_change(int pos, void *prompt)
{   cout << (char *)prompt << pos << endl; // 显示提示和滑块位置
}
```

```
int main()
{   Mat im = imread("Flower.bmp"); // 载入彩色图像
    if(im.empty()) return -1; // 载入失败
    imshow("TrackWin", im); // 显示图像
```

```
    int max = 10; // 滑块位置最大值
    char prompt[] = "TrackbarPos Changed, new position: ";
    createTrackbar("TestBar", "TrackWin", NULL, max, on_change,
                   prompt);
    setTrackbarPos("TestBar", "TrackWin", 4); // 滑块初始位置
```

```
    for(int key = waitKey(); key != 27; key = waitKey())
    {   key = toupper(key);
        if(key == 'G') // 获取滑块位置
            cout << "Current position:" <<
                getTrackbarPos("TestBar", "TrackWin") << endl;
        else if(key == 'S') // 滑块位置改为0
            setTrackbarPos("TestBar", "TrackWin", 0);
    }
}
```

11.2 OpenCV基础数据结构

11.2.1 辅助数据结构

1. 点

常用的点（坐标）是Point（Point2i的简写）对象或Point2f对象，表示二维坐标系下的点，包含x和y两个数据成员，可以使用下列构造函数初始化。

```
Point(int x, int y);
Point2f(float x, float y);
```

2. 矩形大小

常用的矩形大小对象是Size（Size2i的简写）对象或Size2f对象，包含width和height两个数据成员，可以使用下列构造函数初始化。

```
Size(int width, int height);
Size2f(float width, float height);
```

3. 矩形

（1）Rect对象。最常用的矩形对象是Rect对象，包含*x*、*y*、width和height四个int数据成员，可以使用下列构造函数初始化。

```
Rect(int x, int y, int width, int height);
Rect(const Point &org, const Size &sz);
Rect(const Point &pt1, const Point &pt2);
```

（2）RotatedRect对象。倾斜的矩形对象，包含center、size和angle等三个数据成员，分别表示矩形中心、矩形大小和倾斜角度，可以使用下列构造函数初始化。

```
RotatedRect(const Point2f &center, const Size2f &size, float angle);
```

4. 小规模数值向量

常用的小规模数值向量类型的命名方式通常是“Vec<分量数><基本类型>”，其中分量数通常选用2、3或4，基本类型通常选用*b*、*s*、*i*、*f*或*d*，分别是uchar、short、int、float和double的缩写。这类对象可以使用类似下列形式的构造函数初始化。

```
Vec2b(uchar v0, uchar v1);
Vec3i(int v0, int v1, int v2);
Vec4d(double v0, double v1, double v2, double v3);
Vec4d(const double *values);
```

这类对象实现的运算包括$v_1 + v_2$、$v_1 - v_2$、v * scale、scale * v、-v、$v_1 += v_2$、$v_1 -= v_2$、v *= scale、$v_1 == v_2$、$v_1 != v_2$和norm(v)（欧氏范数）。

5. 多通道数量值

多通道数量值通常是一个Scalar类（从Vec4d类派生）的对象，包含四个double成员，通常用于描述多通道数组的像素类型，可以使用下列构造函数初始化。

```
Scalar();
Scalar(double v0); // 用v0初始化0号成员，用0初始化其他成员
Scalar(double v0, double v1, double v2 = 0, double v3 = 0);
```

6. 小规模数值矩阵

（1）命名与初始化。常用的小规模数值矩阵类型的命名方式通常是“Matx<行数><列数><基本类型>”，其中行数和列数可以是1、2、3、4或6（不能都是1），基本类型可以选用f或d，分别是float和double的缩写。这类对象可以使用类似下列形式的构造函数初始化。

```
Matx12d(double v0, double v1);
Matx31d(double v0, double v1, double v2);
Matx22f(float v0, float v1, float v2, float v3);
Matx23f(float v0, float v1, float v2, float v3, float v4, float v5);
Matx23f(const float *values);
```

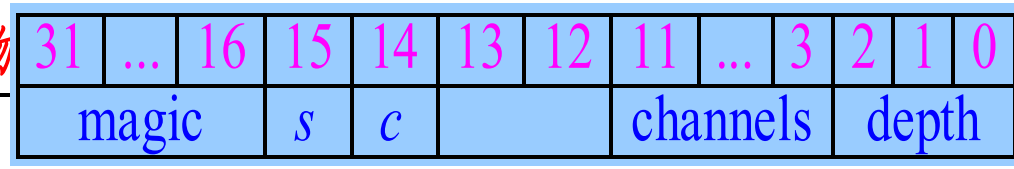
（2）用途。这类矩阵通常用于存储变换矩阵。例如，平移变换的变换矩阵

$$T(t_x, t_y) = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

通常使用下列类似的小规模数值矩阵存储。其中，尾行固定为(0, 0, 1)，不需要存储。

```
Matx23f Translation(1, 0, tx, 0, 1, ty);
```

（3）与Mat对象的转换。可以使用Mat类的构造函数将小规模数值矩阵转换为Mat对象。例如，可以使用“Mat m(Translation);”将Matx23f对象Translation转换成Mat对象m。



11.2.2 矩阵

1. 类型定义

这里只列出一些基本的数据成员。

```
class Mat
{
public:
    // 一些方法(略)
    int flags; // flags包含Mat签名等
    int dims; // 矩阵维数, >= 2
    int rows, cols; // 行数和列数, 矩阵维数>2时为(-1, -1)
    uchar *data; // 指向实际矩阵数据的指针
    // 其他成员(略)
};
```

【注】如图11-4所示，flags成员的各组成部分依次是Mat签名magic(16位）、子数组标志*s*（1位）、连续存储标志*c*（1位）、保留（2位）、通道数channels（9位）和位深度depth（3位）。

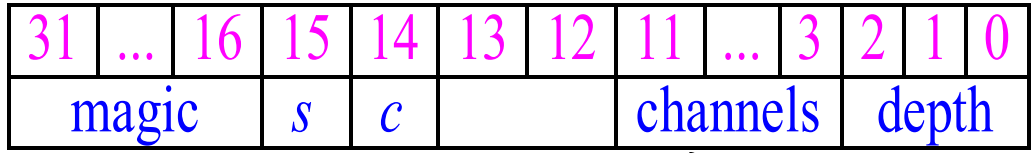


图11-4　flags的组成

2. 常用的构造函数

（1）默认构造函数。创建默认的矩阵头，不分配数据空间，data成员是nullptr。

【函数原型】Mat();

（2）拷贝构造函数。使用另一个矩阵初始化当前矩阵的矩阵头。

【函数原型】Mat(const Mat &m);

【说明】将*m*赋值给新创建的对象，该构造函数不复制矩阵数据，新对象和*m*共用矩阵数据，赋值运算也是使用这种操作方式。如图11-5所示。

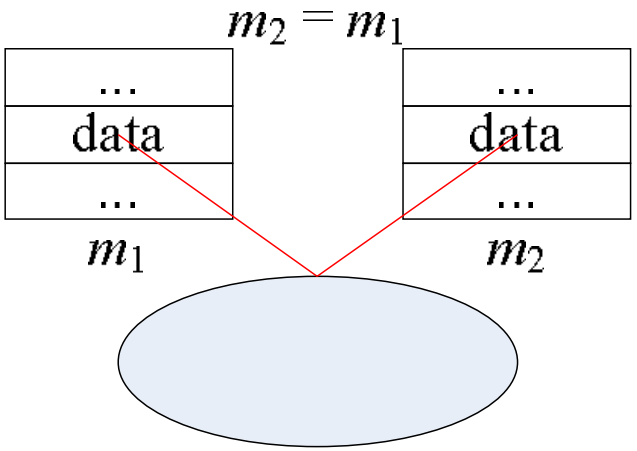


图11-5　两矩阵共用数据

（3）创建矩阵。使用指定的大小和类型创建一个2维矩阵，分配数据空间，数据空间的地址存入data成员。

【函数原型】

- Mat(int rows, int cols, int type);
- Mat(Size size, int type);

【参数】

- rows和cols：矩阵的行数和列数。
- size：矩阵的大小，包含宽度和高度。
- type：矩阵元素类型，以“CV_<位数>{SUF}C<通道数>”的形式描述。

【举例】

```
Mat X(640, 480, CV_8UC1); // 单通道字节矩阵
Mat Y(640, 480, CV_16SC2); // 双通道16位符号数矩阵
Mat W(640, 480, CV_32FC3); // 3通道单精度浮点数矩阵
```

（4）创建填充数据的矩阵。使用指定的大小和类型创建一个2维矩阵，并将所有元素都规定为指定的值。

【函数原型】

- Mat(int rows, int cols, int type, const Scalar &s);
- Mat(Size size, int type, const Scalar &s);

【参数】

- rows和cols：矩阵的行数和列数。
- size：矩阵的大小，包含宽度和高度。
- type：矩阵元素类型。
- *s*：矩阵的每个元素都用该值初始化。

【举例】

```
Mat X(640, 480, CV_8UC1, Scalar(127)); // 单通道字节矩阵
Mat Y(640, 480, CV_16SC2, Scalar(100, 100)); // 双通道16位符号数矩阵
Mat W(640, 480, CV_32FC3, Scalar(1, 0, 1)); // 3通道单精度浮点数矩阵
```

（5）用指定数据创建矩阵。使用指定的大小和类型创建一个2维矩阵头，并用指定的数组存放矩阵数据。

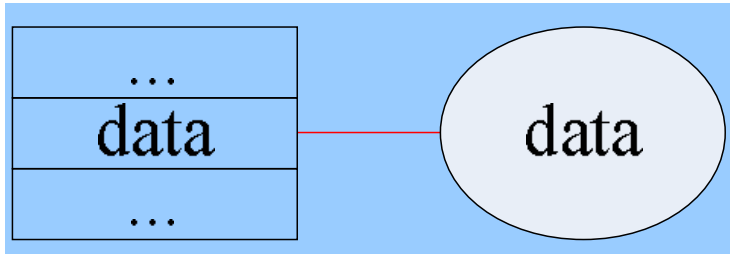
【函数原型】

- Mat(int rows, int cols, int type, void *data);
- Mat(Size size, int type, void *data);

【说明】这种构造函数不创建数据空间，直接使用用户指定的数据空间。

【参数】

- rows和cols：矩阵的行数和列数。
- size：矩阵的大小，包含宽度和高度。
- type：矩阵元素类型。
- data：指向实际存放数据的数组（类型必须与type一致）。



【举例】

```
float X[2][3] =
{   {1, 0, 2},
    {0, 1, -1}
};
Mat M(2, 3, CV_32FC1, X); // CV_32F也可
```

3. 拷贝和重建

（1）重建当前矩阵。可以使用create()成员函数在必要时重建当前矩阵。

【函数原型】

- void create(int rows, int cols, int type);
- void create(Size size, int type);

【参数】

- rows和cols：矩阵的行数和列数。
- size：矩阵的大小，包含宽度和高度。
- type：矩阵元素类型。

【说明】使用该成员函数重建当前矩阵时不能保留原有数据。

【举例】

```
Mat X(640, 480, CV_8UC1); // 单通道字节矩阵
Mat Y(640, 480, CV_16SC2); // 双通道16位符号数矩阵
```

```
X.create(640, 480, CV_16SC2); // 需要重建
Y.create(640, 480, CV_16SC2); // 不需要重建
Y.create(640, 480, CV_16UC2); // 只需修改矩阵头
```

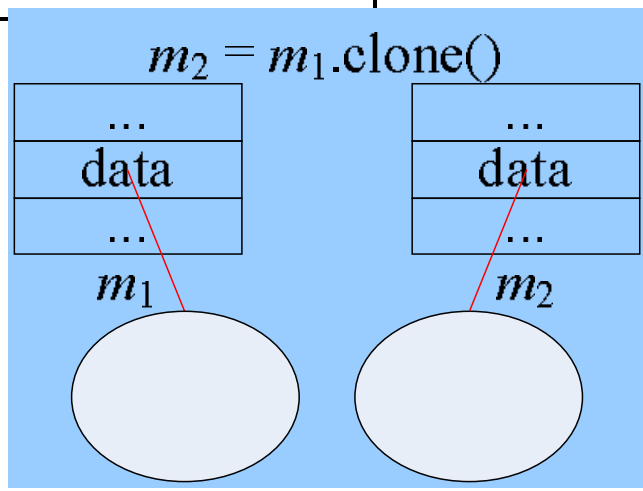
（2）创建新拷贝。使用clone()成员函数。

【函数原型】Mat clone() const;

【说明】该成员函数创建当前矩阵的一个深拷贝，就是说数据也拷贝。

【举例】

```
Mat X(4, 4, CV_32F);
Mat Y = X.clone();
```



4. Matlab风格的初始化

使用Mat的下列static成员函数根据指定的大小和类型创建一个2维全0矩阵、全1矩阵或单位矩阵。

【函数原型】

- static MatExpr zeros(int rows, int cols, int type);
- static MatExpr zeros(Size size, int type);
- static MatExpr ones(int rows, int cols, int type);
- static MatExpr ones(Size size, int type);
- static MatExpr eye(int rows, int cols, int type);
- static MatExpr eye(Size size, int type);

【参数】

- rows和cols：矩阵的行数和列数。
- size：矩阵的大小，包含宽度和高度。
- type：矩阵元素类型。

【说明】对于eye()，若$i=j$，则(i,j)元素的值为1，否则为0。

【举例】Mat m = Mat::eye(3, 4, CV_8UC3);

11.2.3 矩阵表达式

这里列出了常用的可以在任意复杂表达式中组合使用的矩阵运算，其中A和B代表矩阵，s代表多通道数量值（Scalar），v代表实数值（double）。

- 加法、减法和取负：$A+B, A+s, s+A, A-B, A-s, s-A, -A$。
- 按元素相乘和相除：A.mul(B), $A*v, v*A, A/B, A/v, v/A$。
- 矩阵相乘：$A*B$。
- 点积：A.dot(B)。将矩阵当作向量计算，结果是一个double数。
- 转置：A.t()。
- 矩阵的逆或伪逆：A.inv()。
- 比较：A op B, A op v, v op A，其中op是==, !=, >, >=, <, <=之一。结果是一个单通道字节数组，满足关系为255，否则为0。
- 按位逻辑运算：~A，A op B, A op s, s op A，其中op是&, |, ^之一。
- 按元素取小和取大：min(A, B), min(A, v), min(v, A), max(A, B), max(A, v), max(v, A)。
- 赋值：$A=B, A=s$，并且加减乘除和位逻辑运算均有相应的复合赋值，如$A+=B, A*=B$（矩阵相乘），A &= B等。

11.2.4 代理数据类型

InputArray和OutputArray两个类都是代理数据类型，用来接收Mat和vector<>等类型的对象作为输入参数。

1. InputArray

InputArray是用于将只读输入数组传递到OpenCV函数的代理类，可以由Mat、vector<>、Scalar或double等类型的对象构造，也可以由矩阵表达式构造。

该类是专门为传递参数设计的，通常不应显式地创建InputArray实例。

如果需要InputArray参数输入空矩阵，则实际参数可用noArray()或Mat()。必要时可以使用empty()成员函数检查输入是否为空矩阵。

类型InputArrayOfArrays表示函数参数可以是向量的向量，也可以是矩阵向量，当前被解读为InputArray的同义词。

2. OutputArray

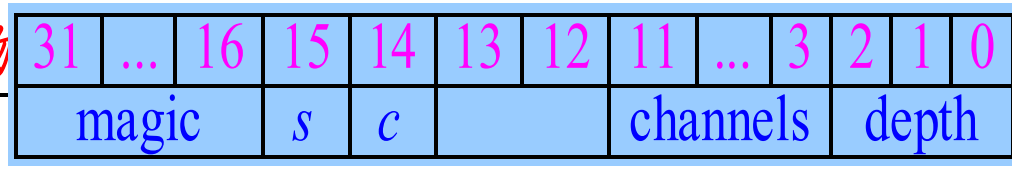
OutputArray是InputArray的派生类，是用于将可读写输入数组传递到OpenCV函数的代理类。与InputArray一样，OutputArray参数只是将Mat和vector<>等类型的对象传递给函数。同样，不要显式地创建OutputArray实例。

类型OutputArrayOfArrays、InputOutputArray和InputOutputArrayOfArrays表示函数参数可以是向量的向量，也可以是矩阵向量，当前被解读为OutputArray的同义词。

3. 在自定义函数中使用

如果需要设计自己的功能或能操作多种数组类型的方法，可以使用InputArray或OutputArray作为相应的参数。此时，在函数内部可以使用getMat()方法构造数组矩阵头（不复制数据），或使用getMatRef()方法获得数组矩阵头的引用（仅限于OutputArray），以利于使用矩阵方法操作数据。

InputArray可以使用size(), total(), type(), depth(), channels(), empty()等与Mat相似的成员函数，OutputArray还可以使用与Mat相似的成员函数create()。



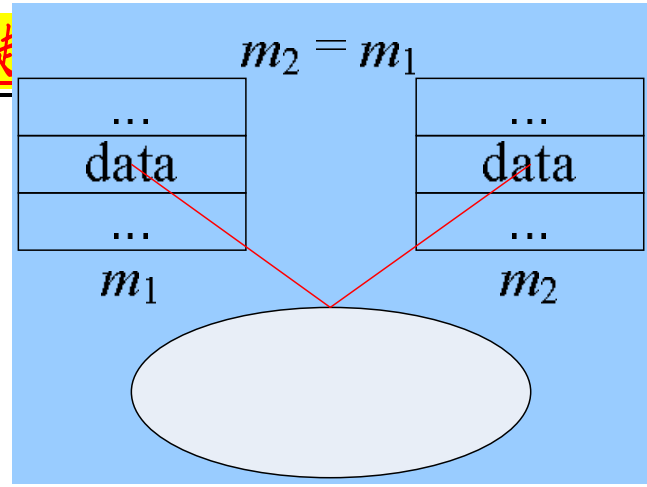
11.3 OpenCV矩阵的基础操作

11.3.1 对矩阵头的操作

1. 获得属性值

通常使用下列成员获得相应属性值。

- int rows, cols。获得该矩阵的行数和列数。
- Size size() const。返回该矩阵的大小（宽度和高度）。
- size_t total() const。返回该矩阵的元素数目。
- int type() const。返回该矩阵的元素类型（如CV_8UC3）。
- int depth() const。返回矩阵元素的位深度（如CV_8U）。
- int channels() const。返回该矩阵的通道数。
- size_t elemSize() const。返回该矩阵中每个元素占用的空间。
- bool isContinuous() const。返回矩阵元素是否连续存储。
- bool isSubmatrix() const。返回该矩阵是否是子矩阵。



2. 修改矩阵形状

使用reshape()成员函数。

【函数原型】Mat reshape(int cn, int rows = 0) const;

【功能】不拷贝数据修改数组的形状，结果是二维数组。

【参数】

- cn: 新的通道数。0表示不修改通道数。
- rows: 新的行数。0表示不修改行数，只根据新通道数修改列数，正整数表示根据新行数和新通道数修改列数。

【说明】

- 该函数不修改数组数据，只返回一个新的矩阵头。
- 只适用于1~4通道数组。

【举例】下列代码将一个4行3列2通道矩阵*X*转换成一个3行8列单通道矩阵*Y*。

```
Mat X(4, 3, CV_8UC2); // 4*3*2=24
Mat Y = X.reshape(1, 3); // 24/(1*3)=8
```

3. 改变矩阵大小

使用resize()函数。

【调用形式】void resize(InputArray src, OutputArray dst, Size dsize);

【参数】

- src：源数组。
- dst：结果数组，类型与源数组一致，必要时重建，大小由dsize指定。
- dsize：目标数组大小。

【注】因为不能保证结果数组的元素与源数组的元素一一对应，所以该函数还需要自行计算结果数组各元素的值（具体的计算方法在图像变换一章中介绍）。

11.3.2 对矩阵数据的操作

1. 填充

（1）使用矩阵表达式。mat = value，其中value可以是Scalar值或double数等。

（2）使用setTo()成员函数。

【函数原型】Mat &setTo(InputArray value, InputArray mask = noArray());

【功能】填充数组，即将数组的所有元素都改为指定值。

【参数】

- value：填充值，可以是小规模数值矩阵、Scalar值或double数等。
- mask：操作掩码，是与当前数组大小相同且通道数为1或与当前数组相同的字节数组，用于选取待修改元素（默认选取全部元素）。

【说明】若mask[i] != 0，则dst[i] = value。

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \\ A & B & C & D & E \\ F & 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 & 8 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 7 & 8 & 0 \\ 0 & B & C & D & 0 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

（3）使用随机数填充。可以使用函数randu()和randn()。

【函数原型】

- void randu(InputOutputArray dst, InputArray low, InputArray high);
- void randn(InputOutputArray dst, InputArray mean, InputArray stddev);

【功能】

- randu()使用指定范围内均匀分布的随机数填充矩阵指定矩阵。
- randu()使用正态分布（高斯分布）的随机数填充指定矩阵。

【参数】

- dst：待填充数组，必须预先分配空间。
- low和high：均匀分布随机数的下限（包含）和上界（不包含）。
- mean和stddev：正态分布随机数的平均值（期望值）和标准差。

【说明】参数low、high、mean和stddev通常是一个Scalar对象，各分量分别对应待填充数组的各通道。而且，各分量相同的Scalar对象通常使用一个double数表示。

【举例】下述示例程序演示了使用随机数填充Mat对象和通过输出流输出Mat对象。

```
// Rand.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
```

```
int main()
{   Mat im(3, 4, CV_8U); // 分配空间
    randu(im, 64, 128); // 使用均匀随机数填充(含64, 不含128)
    cout << im << endl; // 使用默认风格输出
    Mat im2(3, 4, CV_8UC3); // 分配空间
    randn(im2, Scalar(96, 0, 96), 32); // 使用高斯随机数填充
    auto fmt = Formatter::FMT_PYTHON; // Python风格
    cout << format(im2, fmt) << endl; // 使用Python风格输出
    // 有Default, Matlab, CSV, Python, Numpy, C等风格
}
```

```
[118,  92,  64,  71;
 101, 102,  66, 115;
 103,  84, 107, 102]
```

```
[[[ 82,  40,  87], [ 75,  10, 112], [ 59,  25,  95], [138,   0,  35]],
 [[ 79,   0, 157], [105,   0, 145], [ 71,   0,  46], [ 52,   0, 139]],
 [[122,   0,  79], [155,  12, 137], [ 75,   0, 130], [ 98,   0,  91]]]
```

2. 复制

使用copyTo成员函数。

【函数原型】

- void copyTo(OutputArray m) const;
- void copyTo(OutputArray m, InputArray mask) const;

【功能】复制数组，包括矩阵的头部和数据。

【参数】

- *m*：结果矩阵，必要时会重新创建。
- mask：操作掩码，是与当前数组大小相同且通道数为1或与当前数组相同的字节数组，用于选取需要复制的元素。

【说明】如果mask(i) != 0，则$m(i) = \text{src}(i)$。

【举例】作为例子，这里给出一个能够交换两个矩阵内容的函数。为了方便后续章节使用，将该函数保存在文件cvv.hpp中。

```
// 交换两个Mat的内容
void swap(Mat &X, Mat &Y)
{   Mat W = X.clone(); // W = X
    Y.copyTo(X); // X = Y
    W.copyTo(Y); // Y = W
}
```

```
// 交换两个Mat的内容, 使用Swap, 以免与swap冲突
void Swap(Mat &X, Mat &Y)
{   Mat W = X.clone(); // W = X
    Y.copyTo(X); // X = Y
    W.copyTo(Y); // Y = W
}
```

其中，文件cvv.hpp的首部如下。

```
// cvv.hpp
#pragma once
#include<opencv2/opencv.hpp>
```

3. 对角线元素赋值

【函数原型】void setIdentity(InputOutputArray mtx, const Scalar &s = Scalar(1));

【功能】对角线元素赋值为指定值s，其余元素为0。

【参数】

- mtx：待赋值的矩阵（不一定是方阵）。
- s：赋值给对角线元素的值。

【说明】若$i=j$，则(i,j)元素的值为s，否则为0。

【举例】

```
Mat M(4, 5, CV_32F);
setIdentity(M, Scalar(1)); // 对角线元素值为1
setIdentity(M, Scalar(15)); // 对角线元素值为15
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

11.3.3 对矩阵元素的操作

1. 使用Mat的at()成员

【调用形式】常用的调用形式有下列几种。

- template<class T> T &at(int i = 0);
- template<class T> T &at(int i, int j);
- template<class T> T &at(int i, int j, int k);

【功能】获得指定矩阵元素的引用。

【参数】

- *i*：元素下标的第一个成员，以0为基准。
- *j*：元素下标的第二个成员，以0为基准。
- *k*：元素下标的第三个成员，以0为基准。

【说明】必须在模板参数中指定元素的类型，通常使用基本类型和小规模数值向量。

【举例】下列程序演示了at()成员函数的使用方法。

```
// Mat_at.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
int main()
{   int rows = 2, cols = 3;
    Mat X(rows, cols, CV_8U), Y(rows, cols, CV_8UC3);
    Mat Z(rows, cols, CV_32FC3), W(rows, cols, CV_64FC2);
    for(int i = 0; i < rows; ++i)
        for(int j = 0; j < cols; ++j)
        {   auto v = i + j + i * j + 0.5;
            X.at<uchar>(i, j) = v, Y.at<Vec3b>(i, j) = v;
            Z.at<Vec3f>(i, j) = v, W.at<complex<double>>(i, j) = v;
        } // 可见，双通道Mat可用作复数Mat
    cout << X << endl; // 使用默认风格
    auto fmt = Formatter::FMT_PYTHON; // 使用Python风格
    cout << format(Y, fmt) << endl;
    cout << format(Z, fmt) << endl;
    cout << format(W, fmt) << endl;
}
```

```
[ 0, 1, 2;
  1, 3, 5]
[[[ 0, 0, 0], [ 1, 0, 0], [ 2, 0, 0]],
 [[ 1, 0, 0], [ 3, 0, 0], [ 5, 0, 0]]]
[[[0.5, 0, 0], [1.5, 0, 0], [2.5, 0, 0]],
 [[1.5, 0, 0], [3.5, 0, 0], [5.5, 0, 0]]]
[[[0.5, 0], [1.5, 0], [2.5, 0]],
 [[1.5, 0], [3.5, 0], [5.5, 0]]]
```

[0, 1, 2;   1, 3, 5]
[[[0, 0, 0], [1, 0, 0], [2, 0, 0]],   [[1, 0, 0], [3, 0, 0], [5, 0, 0]]]
[[[0.5, 0, 0], [1.5, 0, 0], [2.5, 0, 0]],   [[1.5, 0, 0], [3.5, 0, 0], [5.5, 0, 0]]]
[[[0.5, 0], [1.5, 0], [2.5, 0]],   [[1.5, 0], [3.5, 0], [5.5, 0]]]

2. 使用Mat_<>的operator()成员

Mat_<>是从Mat类派生出来的模板矩阵类，引入Mat_<>的主要目的是方便简洁地访问矩阵元素。Mat_<T>类是Mat类的模板包装器，它没有增加任何数据成员，只是提供了一些更方便更简洁的方法。Mat_<T>类和Mat类可以相互转换。

Mat在大多数情况下都是有效的。但是如果需要大量的元素访问操作，则使用Mat_<T>类会更方便更简洁。Mat_<T>::operator()(int *y*, int *x*)和Mat::at<T>(int *y*, int *x*)以相同的速度执行完全相同的操作，但前者肯定更方便，也更简洁。

```
// Mat1f.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
int main()
{   Mat1f M(3, 5); // Mat_<float>可简写为Mat1f
    for(int i = 0; i < M.rows; ++i)
        for(int j = 0; j < M.cols; ++j)
            M(i, j) = i + j; // M.at<float>(i, j)
    cout << M.at<float>(1, 2) << endl; // 3
    cout << M(1, 2) << endl; // 3, 更简洁
}
```

3. 将Mat_<>用于多通道矩阵

如果需要将Mat_<>用于多通道图像或多通道矩阵，可以使用小规模数值向量类型作为Mat_<>的模板参数。而且，1~4通道的矩阵类型Mat_<T>通常有形如“Mat<通道数><基本类型>”的简写名称，其中基本类型通常选用b、s、i、f或d，分别是uchar、short、int、float和double的缩写。例如，Mat_<uchar>可以简写为Mat1b，Mat_<Vec3d>可以简写为Mat3d。

下列程序演示了多通道Mat_<>的使用，运行结果如图11-6所示。

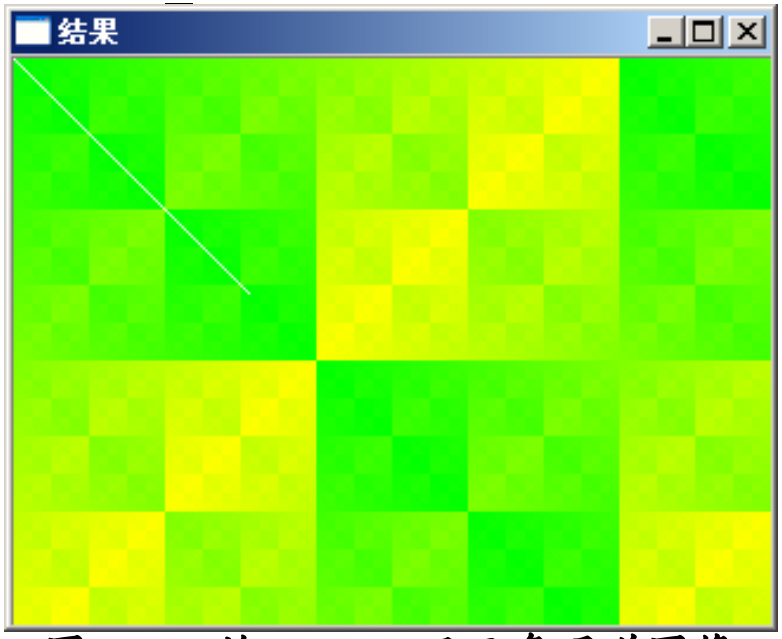
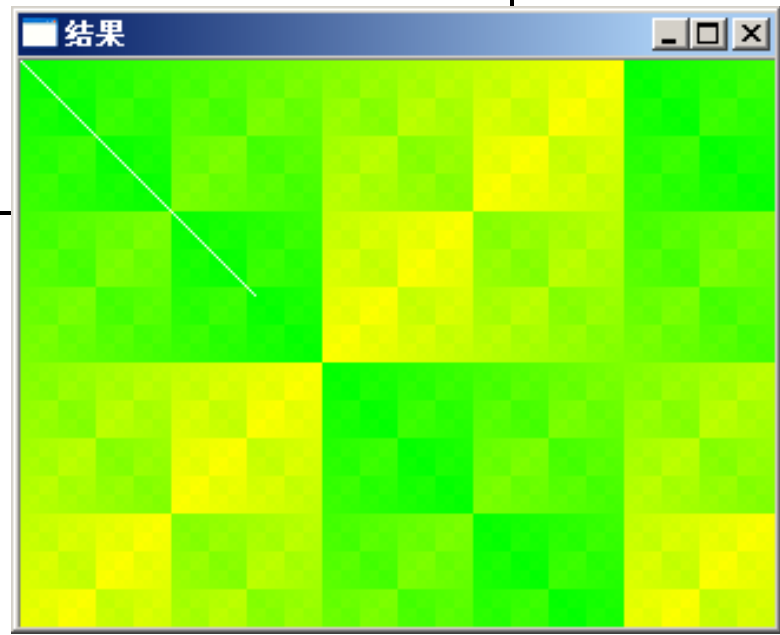


图11-6　将Mat_<>用于多通道图像

```cpp
// Mat3b.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat3b im(240, 320, {0, 255, 0}); // 绿色填充
    // 其中, Mat3b是Mat_<Vec3b>的简写
    for(int i = 0; i < 100; ++i) // 绘制一条白色对角线
        im(i, i) = {255, 255, 255};
    for(int i = 0; i < im.rows; ++i) // 对红色通道进行干扰
        for(int j = 0; j < im.cols; ++j)
            im(i, j)[2] ^= (i ^ j);
    imshow("结果", im);
    imwrite("Green.bmp", im); // 保存图像
    waitKey();
}
```



11.3.4 选取子集

1. 选取矩形子集

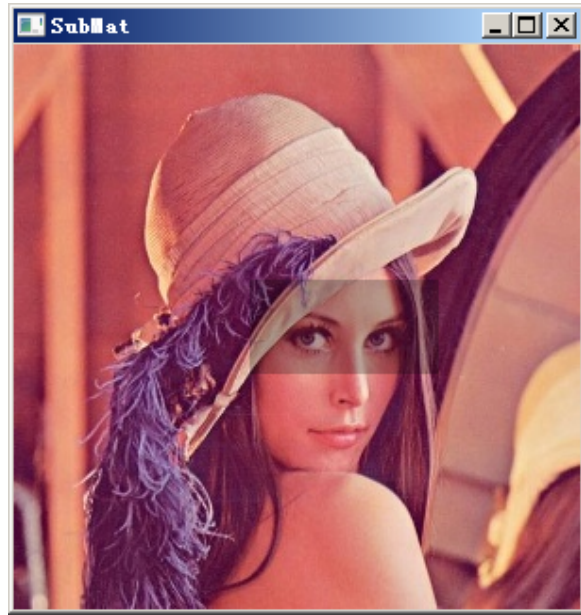
使用()运算。

【函数原型】Mat operator()(const Rect &roi) const;

【功能】返回输入矩阵的矩形子集的矩阵头。

【参数】roi是感兴趣的矩形区域。

【说明】该函数返回输入矩阵中指定矩形区域对应子集的矩阵头，从而可以将输入矩阵的一个矩形子集当作一个独立矩阵处理。



【举例】下列程序在源图像中选取一个矩形子集，并将该矩形子集乘以0.75，然后显示图像。程序运行结果如图11-7所示。

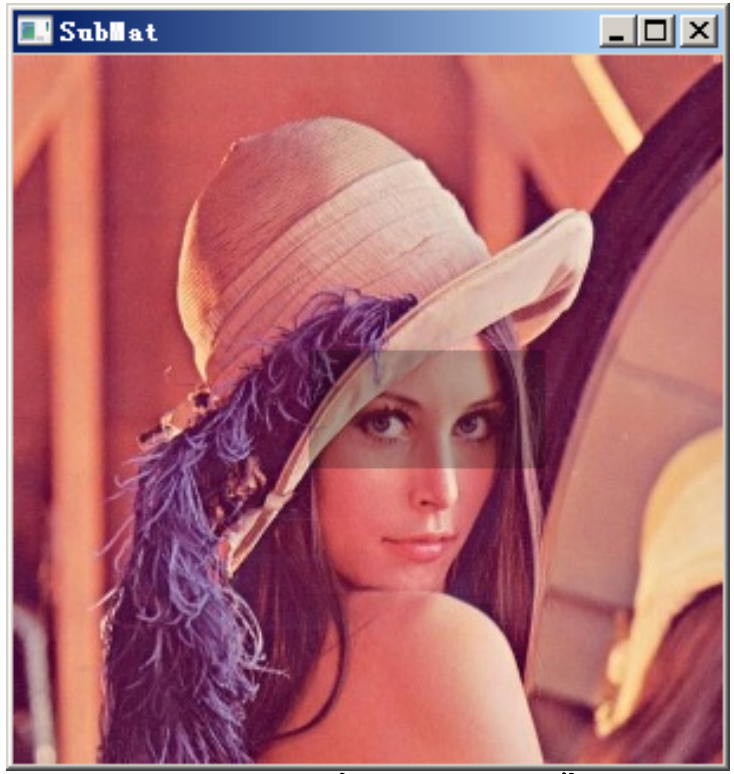


图11-7　选取矩形子集

```
// SubMat.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat im = imread("lena.jpg"); // 载入彩色图像
    if(im.empty()) return -1; // 载入失败
    Mat roi = im({ 125, 125, 100, 50 }); // 选取矩形子集
    roi *= 0.75; // 降低选取区域亮度
    imshow("SubMat", im); // 显示图像
    waitKey();
}
```

2. 通道的分割与合并

（1）分割通道。将一个多通道矩阵分割为几个单通道矩阵。

【函数原型】

- void split(const Mat &src, Mat *mv);
- void split(InputArray src, OutputArrayOfArrays mv);

【参数】

- src：原矩阵。
- mv：目标通道数组，可以是C风格的Mat数组或vector<Mat>等。

【说明】目标数组的大小和源通道个数相同，每个通道在必要时都会重建。

（2）合并通道。将若干个单通道矩阵复制到一个多通道矩阵中。

【函数原型】

- void merge(const Mat *mv, size_t count, OutputArray dst);
- void merge(InputArrayOfArrays mv, OutputArray dst);

【参数】

- mv：源通道数组，可以是C风格的Mat数组或vector<Mat>等。
- count：C风格Mat数组的大小（元素数）。
- dst：结果矩阵，必要时会重建。

（3）举例。该例子用于演示红色和蓝色通道交换后的效果。程序运行结果如图11-8所示。

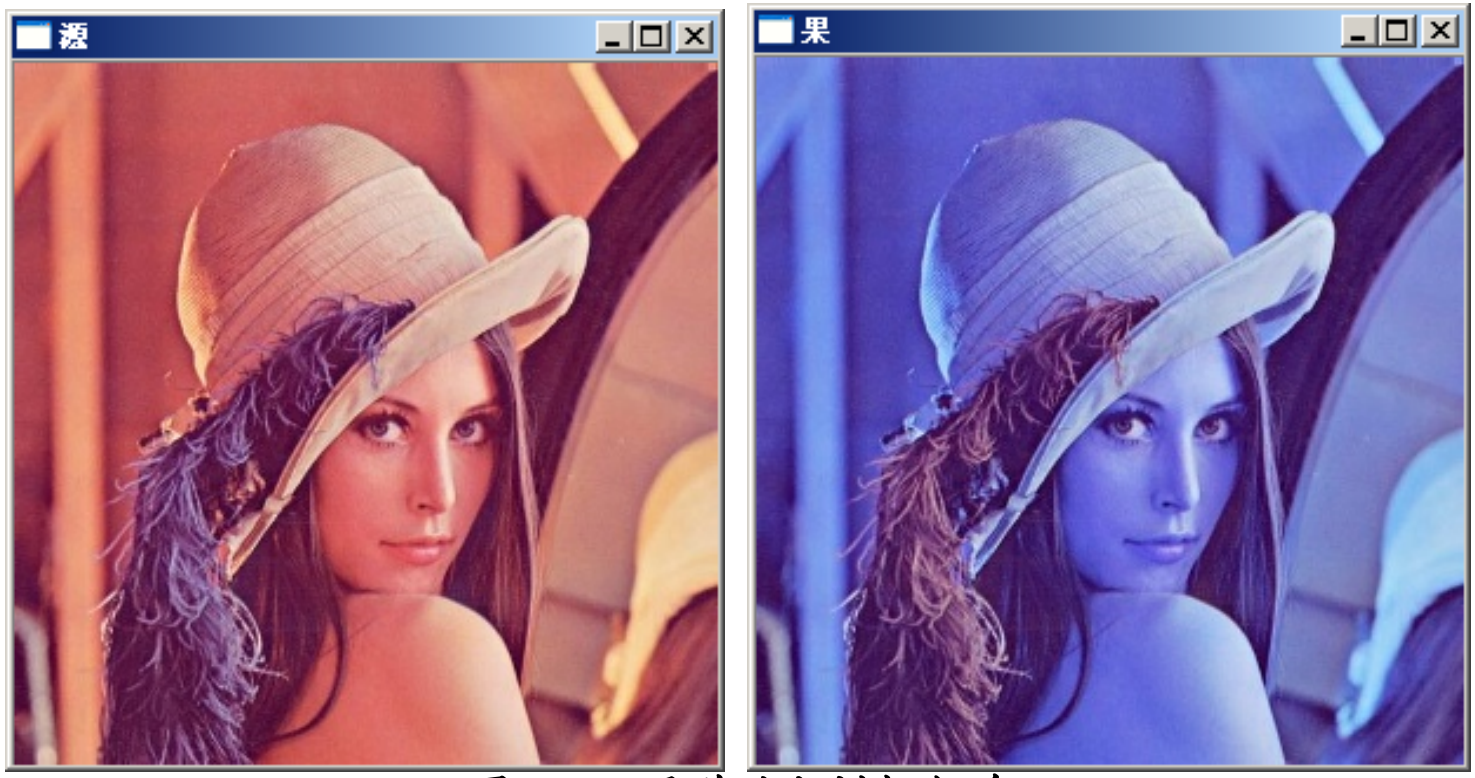


图11-8　通道的分割与合并

```
// Split.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
int main()
{   Mat X = imread("lena.jpg"); // 载入彩色图像
    if(X.empty()) return -1; // 载入失败
    imshow("源", X); // 显示源图像
    vector<Mat> mv; // 通道数组
    split(X, mv); // 提取通道
    swap(mv[0], mv[2]); // 交换RB通道(只交换了矩阵头)
    merge(mv, X); // 合并通道
    imshow("果", X); // 显示结果图像
    waitKey();
}
```

3. 通道的提取与插入

（1）提取通道。从源数组提取指定的单个通道。

【函数原型】void extractChannel(InputArray src, OutputArray dst, int coi);

【参数】

- src：原数组。
- dst：目标通道，必要时会重建。
- coi：待提取通道的索引（从0开始）。

（2）插入通道。用源数组替换目标数组的指定通道。

【函数原型】void insertChannel(InputArray src, InputOutputArray dst, int coi);

【参数】

- src：单通道的源数组。
- dst：目标数组，大小和位深度必须与源数组一致。
- coi：目标数组中待替换通道的索引（从0开始）。

【说明】目标数组的指定通道必须存在。

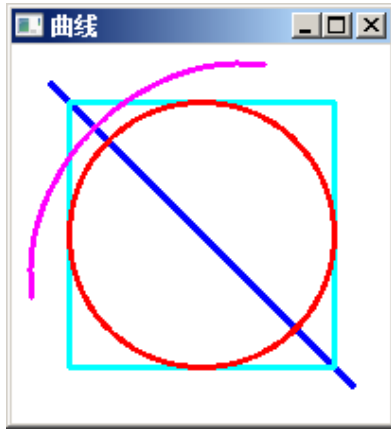
（3）举例。用提取-插入通道的方法改写前述交换红色和蓝色通道的例子。

```
// Extract.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat X = imread("lena.jpg"); // 载入彩色图像
    if(X.empty()) return -1; // 载入失败
    imshow("源", X); // 显示源图像
    // 提取蓝色通道和红色通道
    Mat B, R;
    extractChannel(X, B, 0), extractChannel(X, R, 2);
    swap(R, B); // 交换RB通道(只交换了矩阵头)
    // 插入蓝色通道和红色通道
    insertChannel(B, X, 0), insertChannel(R, X, 2);
    imshow("果", X); // 显示结果图像
    waitKey();
}
```

11.4 OpenCV绘图命令

1. 线段

【函数原型】void line(InputOutputArray img, Point pt1, Point pt2, const Scalar &color, int thickness = 1);



【功能】绘制连接两个端点的线段。

【参数】

- img：图像。
- pt1和pt2：线段的两个端点。
- color：线段的颜色。
- thickness：线段的线宽。

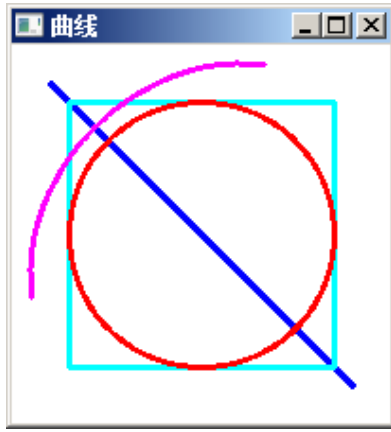
【举例】在(20, 20)与(180, 180)之间绘制一条宽度为2的蓝色线段。

```
line(im, {20, 20}, {180, 180}, {255, 0, 0}, 2); // 蓝色线段
```

2. 矩形

【函数原型】

- void rectangle(InputOutputArray img, Point pt1, Point pt2, const Scalar &color, int thickness = 1);
- void rectangle(InputOutputArray img, Rect rec, const Scalar &color, int thickness = 1);



【功能】通过对角线上的两个顶点绘制矩形。

【参数】

- img: 图像。
- pt1和pt2: 矩形对角线上的两个顶点。
- rect: 待绘制的矩形对象。
- color: 矩形的颜色。
- thickness: 矩形的线宽，负数（如FILLED）表示绘制填充矩形。

【举例】用宽度为2的青绿色线在(30, 30)与(170, 170)之间绘制一个矩形。

```
rectangle(im, {30, 30}, {170, 170}, {255, 255, 0}, 2); // 青绿色矩形
```

3. 圆和椭圆

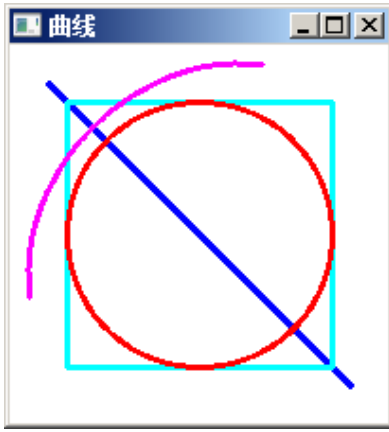
（1）圆。使用circle()函数。

【函数原型】void circle(InputOutputArray img, Point center, int radius, const Scalar &color, int thickness = 1);

【功能】绘制一个给定圆心和半径的圆。

【参数】

- img: 图像。
- center和radius: 圆心坐标和圆半径。
- color: 圆的颜色。
- thickness: 圆弧的线宽，负数表示绘制填充的圆。



【举例】用宽度为2的红色线在(100, 100)处绘制一个半径为70的圆。

```
circle(im, {100, 100}, 70, {0, 0, 255}, 2); // 红色圆
```

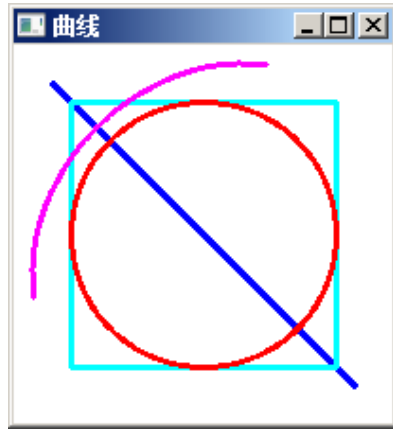
（2）椭圆。使用ellipse()函数。

【函数原型】void ellipse(InputOutputArray img, Point center, Size axes, double angle, double startAngle, double endAngle, const Scalar &color, int thickness = 1);

【功能】绘制一个指定了中心、轴半径、倾斜角度和起止范围的椭圆弧。

【参数】

- img：图像。
- center和axes：椭圆的中心、轴半径和倾斜角度。
- angle：椭圆的斜角度。
- startAngle和endAngle：椭圆弧的起始角度和终止角度。
- color：椭圆的颜色。
- thickness：椭圆的线宽，负数表示绘制填充的椭圆。



【举例】用宽度为2的紫色线绘制一个倾斜的椭圆弧，其中椭圆的中心为(100, 100)，轴半径为(100, 80)，倾斜-45度，椭圆弧的起止范围为-30度到-150度。

```
// 紫色椭圆
ellipse(im, {100, 100}, {100, 80}, -45, -30, -150, {255, 0, 255}, 2);
```

【附】完整的示例程序。

```
// Draw.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat3b im(200, 200);
    line(im, {20, 20}, {180, 180}, {255, 0, 0}, 2); // 蓝色线段
    rectangle(im, {30, 30}, {170, 170}, {255, 255, 0}, 2); // 青绿色矩形
    circle(im, {100, 100}, 70, {0, 0, 255}, 2); // 红色圆
    // 紫色椭圆
    ellipse(im, {100, 100}, {100, 80}, -45, -30, -150, {255, 0, 255}, 2);
    imshow("曲线", im);
    waitKey();
}
```

4. 折线和多边形

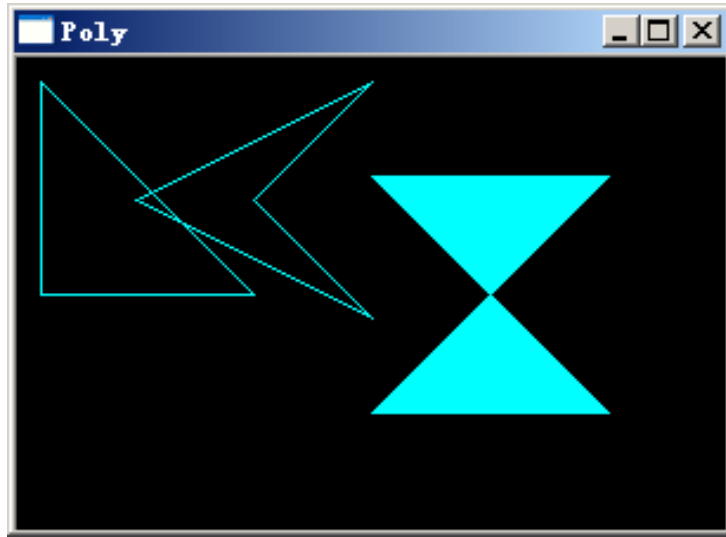
（1）折线。使用polylines()函数。

【函数原型】void polylines(InputOutputArray img, InputArrayOfArrays pts, bool isClosed, const Scalar &color, int thickness = 1);

【功能】绘制若干条折线。

【参数】

- img：图像。
- pts：每条折线的顶点集构成的数组（折线数组），是一个非均匀的二维Point数组，通常使用vector<vector<Point>>对象。
- isClosed：是否封闭。
- color：折线的颜色。
- thickness：折线的线宽。



（2）多边形。使用fillPoly()函数。

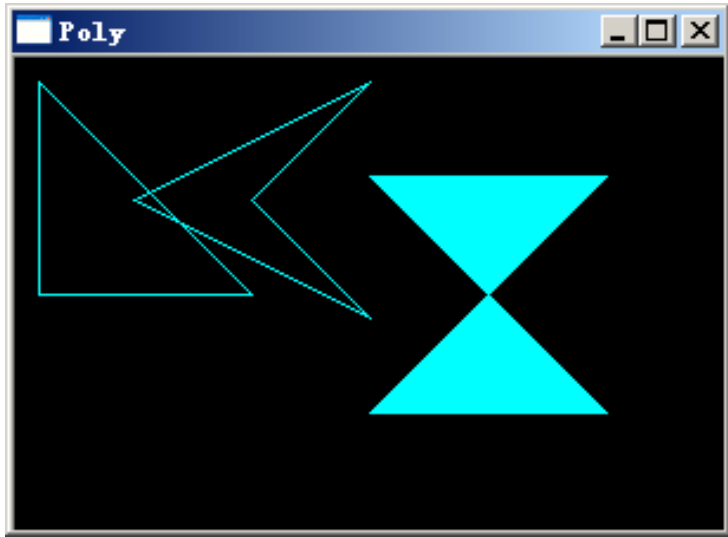
【函数原型】void fillPoly(InputOutputArray img, InputArrayOfArrays pts, const Scalar &color);

【功能】绘制若干个内部填充的多边形。

【参数】

- img：图像。
- pts：每个多边形的顶点集构成的数组（多边形数组），是一个非均匀的二维Point数组，通常使用vector<vector<Point>>对象。
- color：多边形的颜色。

【说明】该函数可以填充比较复杂的区域，例如，有漏洞的区域和有交叉点的区域等。



（3）举例。下列程序使用该函数绘制2条折线和1个有交叉的填充区域，运行结果如图11-9所示。

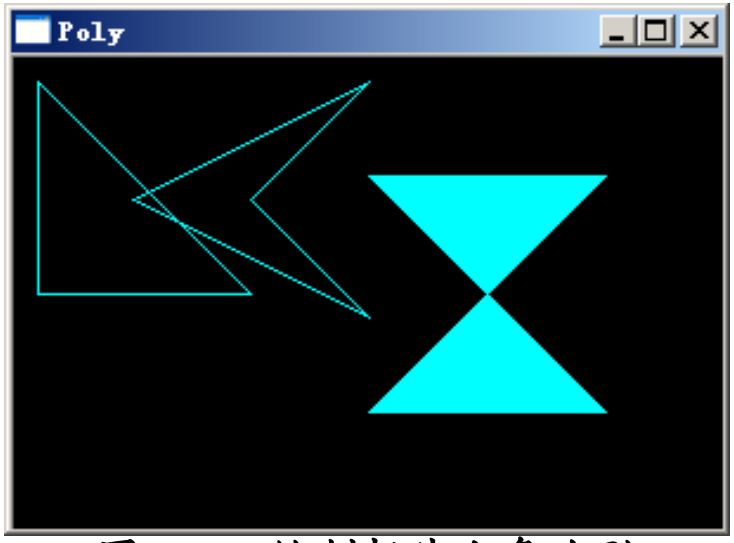


图11-9　绘制折线和多边形

```
// Poly.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
using namespace std;
int main()
{   Mat3b im(200, 300, {0, 0, 0}); // 图像, 黑色背景
    vector<Point> line1 = {{10, 10}, {10, 100}, {100, 100}}, // 折线1
    line2 = {{50, 60}, {150, 110}, {100, 60}, {150, 10}}, // 折线2
    poly = {{150, 50}, {250, 50}, {150, 150}, {250, 150}}; // 多边形
    // 绘制折线(图像, 折线数组, 封闭, 颜色)
    polylines(im, vector{line1, line2}, true, {255, 255, 0});
    // 绘制填充区域(图像, 多边形数组, 颜色)
    fillPoly(im, vector{poly}, {255, 255, 0});
    imshow("Poly", im); // 显示图像
    waitKey();
}
```

5. 文本※

（1）绘制函数。只介绍正立文本的调用形式。

【调用形式】void putText(InputOutputArray img, const String &text, Point org, int fontFace, double fontScale, Scalar color, int thickness = 1);

【功能】在图像中显示文本。

【参数】

- img：输入图像。
- text：待显示文本。
- org：待显示文本的左下角坐标（*y*坐标为基线坐标）。
- fontFace：字体名称标识符。
- fontScale：字体缩放系数。
- color：文本颜色。
- thickness：字体笔画的线宽。

（2）关于字体名称。fontFace是字体名称标识符，可选下列8种值。

- FONT_HERSHEY_SIMPLEX（无衬线字体）
- FONT_HERSHEY_PLAIN（小号无衬线字体）
- FONT_HERSHEY_DUPLEX（复杂的无衬线字体）
- FONT_HERSHEY_COMPLEX（有衬线字体）
- FONT_HERSHEY_TRIPLEX（复杂的有衬线字体）
- FONT_HERSHEY_COMPLEX_SMALL（小号有衬线字体）
- FONT_HERSHEY_SCRIPT_SIMPLEX（手写风格字体）
- FONT_HERSHEY_SCRIPT_COMPLEX（复杂的手写风格字体）

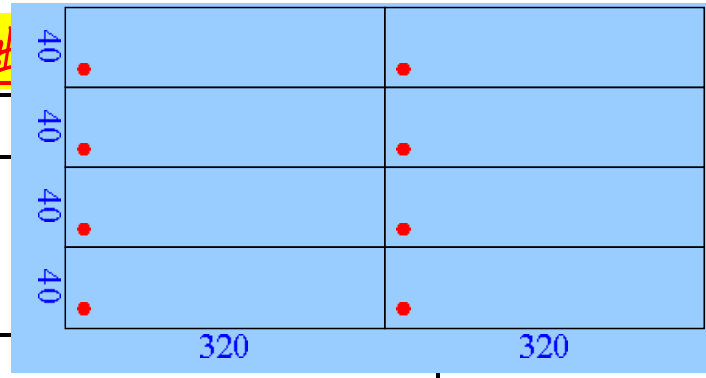
图11-10按照从上到下，从左到右的顺序展示了这8种字体的外观。

【说明】

- fontFace能够由一个选定的值和可选的FONT_ITALIC字体标记组合而成，以达到输出斜体字的目的。
- 不在指定字库中的字符用矩形字符替代。

（3）举例。下列程序调用函数putText()绘制文本，按照从上到下，从左到右的顺序给出了OpenCV支持的8种字体的外观（基线对齐），程序运行结果如图11-10所示。

图11-10　OpenCV文本外观




```
// PutText.Cpp
#include<opencv2/opencv.hpp>
using namespace cv;
int main()
{   Mat1b im(160, 640, 255); // 结果图像, 白色背景
    std::string text = "Hello, OpenCV!"; // 文本
    // 字体(名称, 风格)
    int font = FONT_HERSHEY_SIMPLEX | FONT_ITALIC;
    for(int i = 0; i < 8; ++i) // 字体名称是连续定义的
    {   int x = (i % 2 * 320) + 8; // x坐标(左侧位置, 右移8)
        int y = (i / 2 * 40 + 40) - 14; // y坐标(基线位置, 上移14)
        // 绘制文本(图像, 文本, 坐标, 字体, 倍数, 颜色, 线宽)
        putText(im, text, {x, y}, font + i, 1.2, {0}, 1);
    }
    imshow("PutText", im); // 显示图像
    waitKey();
}
```

11.5 练习题

11.5.1 程序设计题

1. 使用OpenCV在一幅新的真彩色图像中（256×256）绘制一个填充的红色矩形，显示该图像并存入图像文件（Ex11-1.jpg）。其中矩形的2个对角顶点为(64, 64)和(192, 128)。

2. 使用OpenCV装入一幅大小至少为512×512的真彩色图像，并显示该图像。然后在源图像中指定一个矩形区域（左上顶点和宽高值分别为(128, 256)和(256, 128)的矩形），并在结果图像窗口中显示源图像中被选取的部分。

3. 请使用OpenCV编写一个简单的程序，该程序首先读入一幅真彩色图像，然后将这幅彩色图像的3个通道分离出来，得到3幅灰度图像，最后显示这3幅灰度图像。

11.5.2 阶段实习题

1. 使用OpenCV装入一幅彩色图像，并显示该图像。然后在源图像窗口中使用鼠标选取一个矩形区域（可通过两次按下鼠标左键选取矩形的两个对角顶点来实现），并在结果图像窗口中显示源图像中被选取的部分。

2. 使用OpenCV编制一个简单的徒手绘图程序。该程序使用鼠标绘制图形，当鼠标左键按下时开始绘制一条曲线，鼠标左键松开时停止当前曲线的绘制。按下“S”键将当前绘制结果存入图像文件，按下“C”清除所有绘制结果。要求使用白色背景，黑色曲线。可拓展考虑绘制封闭曲线和填充区域。

3. 请根据BMP文件的格式编写一个C++函数Mat bmpread(const string &path)。该函数用于从一个真彩色BMP文件中读取图像数据，保存在Mat对象中（需要实现上下翻转）。注意，不得使用imread()和flip()之类的函数。