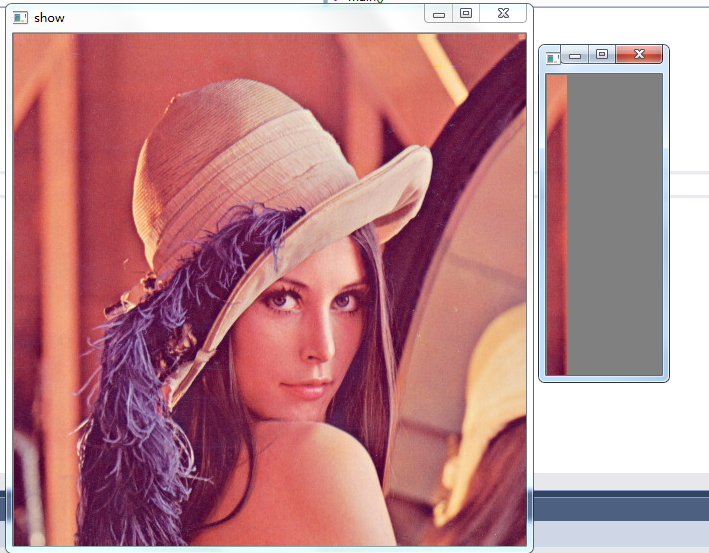
## 提取感兴趣的区域ROI

OpenCV中利用Rect来设置感兴趣矩形区域，可将整个图像的某一部分分离，并利用copyTo函数进行复制。另外可以通过鼠标事件操作获取矩形的上顶点和下顶点，然后求出对应的矩形区域，通过Rect操作进而实现矩形区域提取。

### 1获取感兴趣区域代码



#include<opencv2\opencv.hpp>

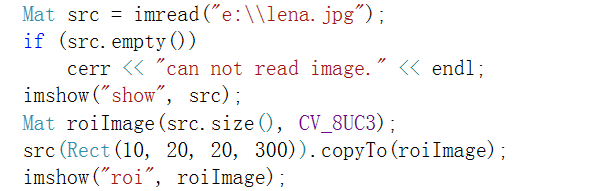
#include<iostream>

using namespace std;

using namespace cv;

int main()

{



waitKey(0);

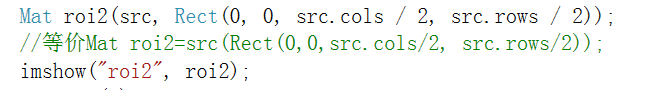
system("pause");

return 0;

}

第二种方法：

在waitKey(0);之前加入下面语句：



#### 1.1cerr的学习

cerr不经过缓冲而直接输出，一般用于迅速输出出错信息，是标准错误，默认情况下被关联到标准输出流，但它不被缓冲，也就说错误消息可以直接发送到显示器，而无需等到缓冲区或者新的换行符时，才被显示。

#### 1.2copyto函数

void copyTo( OutputArray m ) const;

void copyTo( OutputArray m, InputArray mask ) const;

src.copyTo(dst, mask);

将 src 的位于 mask 中的部分，拷贝到 dst 中。mask是一个“掩膜”， 其中非零的位置既是指定了 src 中的那些需要拷贝的部分。

https://blog.csdn.net/u013105205/article/details/78835969

#### 1.3copyto函数,代码学习

#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

#define IMG\_PATH "e:\\lena.jpg"

#define CAT\_PATH "e:\\cat.jpg"

void testroi(void) {

Mat img = imread(IMG\_PATH);

Mat cat = imread(CAT\_PATH);

if (img.empty() || cat.empty())

cerr << "can not read image." << endl;

// 指定感兴趣区域，两种方法

Mat ROI = img(Rect(0, 0, cat.cols, cat.rows));

Mat ROI2(img, Rect(40, 40, cat.cols, cat.rows));

// 展示 roi 区域

imshow("roi", ROI);

cout << endl<< "将猫放到感兴趣区域，两种方法" << endl;

//cat.copyTo(ROI);

cat.copyTo(ROI, cat);

imshow("lena with cat", img);

// 在图像中画出 矩形方法1

rectangle(img, Rect(240, 240, cat.cols, cat.rows), Scalar(0, 0, 255));

imshow("with rectangle box", img);

// 在图像中画出 矩形方法2

cout << endl<< "利用 Rect 保存方框，然后使用" << endl;

Rect r1 = Rect(100, 0, 200, 200);

rectangle(img, r1, Scalar(255, 0, 0));

imshow("with rectangle box 2", img);

}

void contour\_roi(void) {

Mat img = imread(IMG\_PATH);

Mat dst;

Mat roi = Mat::zeros(img.size(), CV\_8U);

// 利用 边界设置roi区域

vector<vector<Point>> contour;

vector<Point> pts;

pts.push\_back(Point(30, 45));

pts.push\_back(Point(100, 15));

pts.push\_back(Point(300, 145));

pts.push\_back(Point(330, 240));

pts.push\_back(Point(50, 250));

contour.push\_back(pts);

drawContours(roi, contour, 0, Scalar::all(255), -1); // 画出

img.copyTo(dst, roi);

imshow("roi", roi);

imshow("img", img);

imshow("dst", dst);

}

void circle\_roi(void) {

Mat image = imread(IMG\_PATH);

Mat dst = Mat::zeros(image.size(), image.type());

Mat mask = Mat::zeros(image.size(), CV\_8U);

Point circleCenter(mask.cols / 2, mask.rows / 2);

int radius = min(mask.cols, mask.rows) / 2;

//circle(mask, circleCenter, radius, Scalar(255),-1); // 画圆

// 画椭圆

ellipse(mask, circleCenter, Size(240, 146), 10, -180, 180, Scalar(255), -1);

image.copyTo(dst, mask);

imshow("mask", mask);

imshow("image", image);

imshow("dst", dst);

}

int main()

{

testroi();

//contour\_roi();

//circle\_roi();

waitKey(0);

system("pause");

return 0;

}

### 2鼠标操作：void setMouseCallback(const String& winname, MouseCallback onMouse, void\* userdata = 0);

鼠标操作的消息映射方式：中介函数（SetMouseCallback）+回调函数。

第一个参数为窗口的名字，对名为winname的视窗进行鼠标监控；

第二个参数用来指定窗口每次鼠标时候发生的时候，被调用函数指针

第三个参数则为用户定义的传递到回调函数的参数。

函数指针原型：

void Foo(int event ,int x ,int y ,int flags ,void \* param)

#### 参数event是鼠标事件

enum MouseEventTypes {

EVENT\_MOUSEMOVE = 0, //!< 鼠标滑动indicates that the mouse pointer has moved over the window.

EVENT\_LBUTTONDOWN = 1, //!<左击indicates that the left mouse button is pressed.

EVENT\_RBUTTONDOWN = 2, //!<右击 indicates that the right mouse button is pressed.

EVENT\_MBUTTONDOWN = 3, //!<中键点击 indicates that the middle mouse button is pressed.

EVENT\_LBUTTONUP = 4, //!<左键放开indicates that left mouse button is released.

EVENT\_RBUTTONUP = 5, //!< 右键放开 indicates that right mouse button is released.

EVENT\_MBUTTONUP = 6, //!<中键放开indicates that middle mouse button is released.

EVENT\_LBUTTONDBLCLK = 7, //!< 左键双击indicates that left mouse button is double clicked.

EVENT\_RBUTTONDBLCLK = 8, //!<右键双击 indicates that right mouse button is double clicked.

EVENT\_MBUTTONDBLCLK = 9, //!<中键双击 indicates that middle mouse button is double clicked.

EVENT\_MOUSEWHEEL = 10,//!< positive and negative values mean forward and backward scrolling, respectively.

EVENT\_MOUSEHWHEEL = 11 //!< positive and negative values mean right and left scrolling, respectively.

};

#### flags,拖拽和键盘操作的代号是CV\_EVENT\_FLAG的组合，flag的状态有：

EVENT\_FLAG\_LBUTTON 左键拖拽

EVENT\_FLAG\_RBUTTON 右键拖拽

EVENT\_FLAG\_MBUTTON 中键拖拽

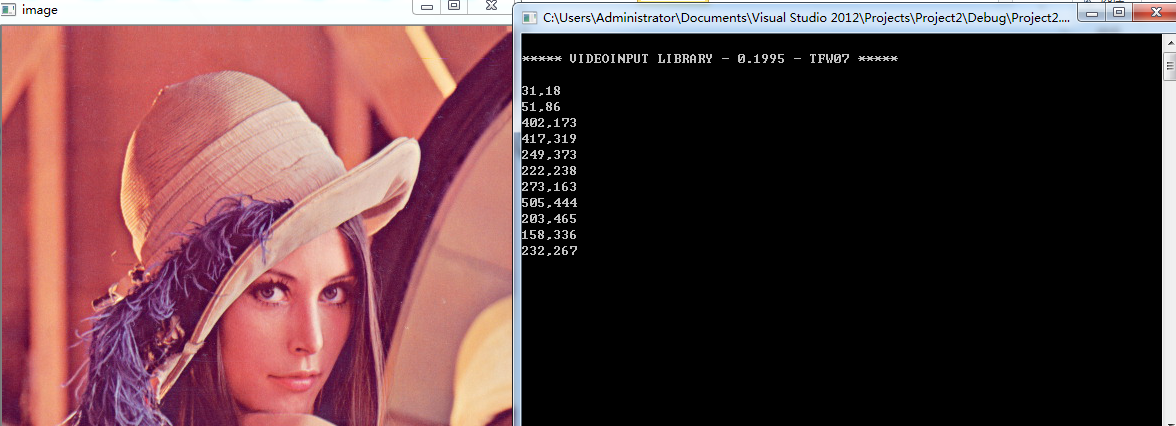
EVENT\_FLAG\_CTRLKEY 按住Ctrl不放

EVENT\_FLAG\_SHIFTKEY 按住Shift不放

EVENT\_FLAG\_ALTKEY 按住Alt不放

param是用户定义的传递到setMouseCallback函数调用的参数，有默认值0

### 2.1代码：鼠标操作，左键点击图像，输出对应的点击位置

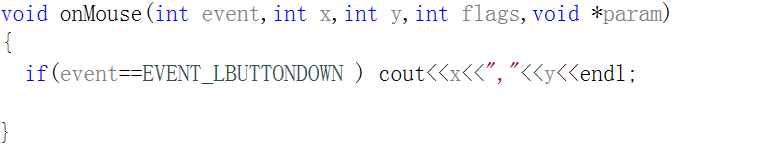


#include<opencv2\opencv.hpp>

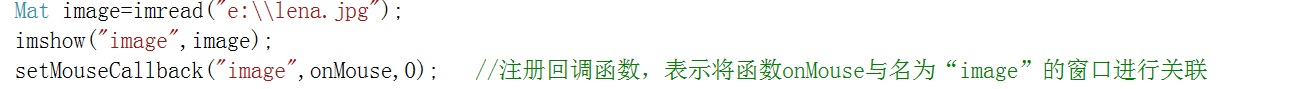
#include<iostream>

using namespace std;

using namespace cv;

 int main( )

{

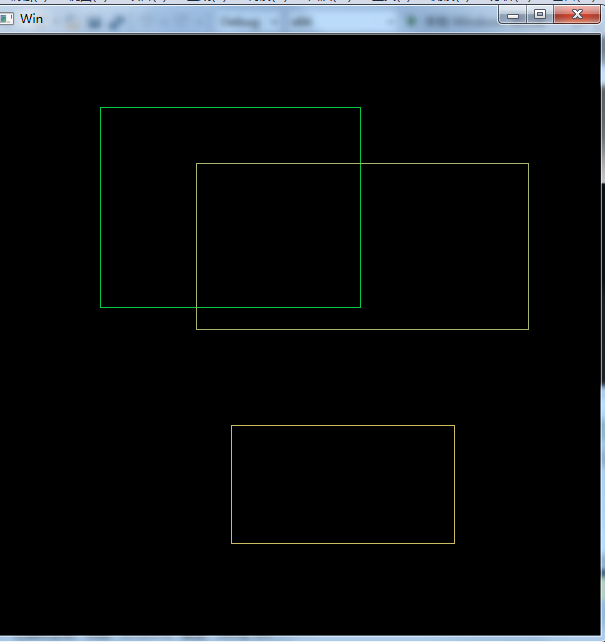
 waitKey(0);

return 0;

}

//鼠标函数还可能收到的事件有：EVENT\_MOUSEMOVE,EVENT\_LBUTTONUP,EVENT\_RBUTTONDOWN,EVENT\_RBUTTONUP

### 2.2代码：在画布上，通过拖动鼠标画矩形



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

Rect g\_rect;

bool g\_DrawFlag = false;

RNG g\_rng(12345);

//矩形绘制函数

void on\_MouseHandle(int event, int x, int y, int flag, void\* param);

void MouseEvent();

void DrawRectangle(Mat& img, Rect box);

int main()

{

MouseEvent();

waitKey(0);

return 0;

}

void MouseEvent()

{

//准备参数

g\_rect = Rect(-1, -1, 0, 0);

Mat srcImage(600, 600, CV\_8UC3,Scalar::all(0)), tempImage;

srcImage.copyTo(tempImage);

//设置鼠标操作回调函数

namedWindow("Win");

setMouseCallback("Win", on\_MouseHandle, (void\*)&srcImage);

//绘画

while (1)

{

srcImage.copyTo(tempImage);

if (g\_DrawFlag) DrawRectangle(tempImage, g\_rect);

imshow("Win", tempImage);

if (waitKey(10) == 27) break;//ESC 退出

}

}

//鼠标回调事件

void on\_MouseHandle(int event, int x, int y, int flag, void\* param)

{

Mat& image = \*(Mat\*)param;

switch (event)

{

case EVENT\_LBUTTONDOWN://左键按下

g\_DrawFlag = true;

g\_rect = Rect(x, y, 0, 0);//设置g\_rect的初始值在同一个点

break;

case EVENT\_MOUSEMOVE://移动

if (g\_DrawFlag) {

//计算，g\_rect宽高=鼠标当前位置坐标-g\_rect左上角的坐标

g\_rect.width = x - g\_rect.x;

g\_rect.height = y - g\_rect.y;

}

break;

case EVENT\_LBUTTONUP://左键抬起

g\_DrawFlag = false;

//当g\_rect宽高小于0,起始点xy坐标置为较小靠左上角的点,宽高取绝对值

if (g\_rect.width < 0) {

g\_rect.x += g\_rect.width;

g\_rect.width \*= -1;

}

if (g\_rect.height < 0) {

g\_rect.y += g\_rect.height;

g\_rect.height \*= -1;

}

//画矩形

DrawRectangle(image, g\_rect);

break;

}

}

void DrawRectangle(Mat& img, Rect box)

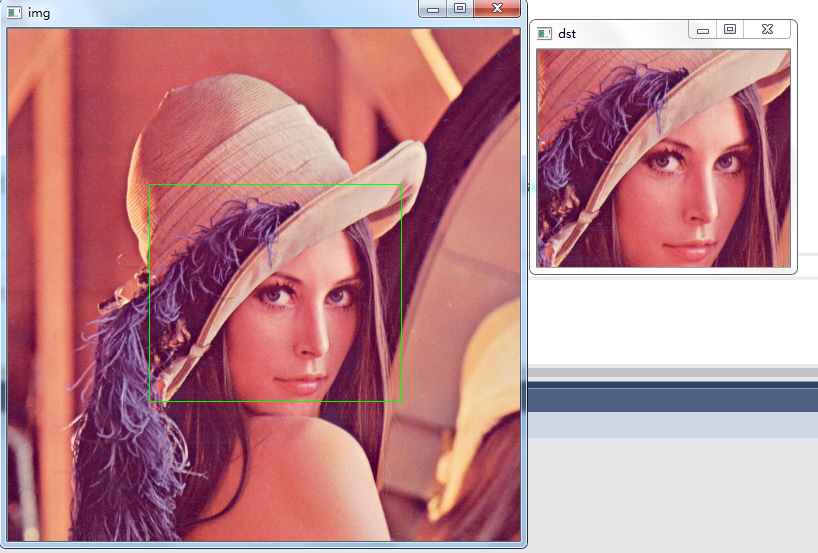
{

//rectangle画矩形,tl左上角的点，br右下角的点,Scalar设置颜色，设置为3通道,g\_rng.uniform(0, 255)随机颜色

rectangle(img, box.tl(), box.br(), Scalar(g\_rng.uniform(0, 255), g\_rng.uniform(0, 255), g\_rng.uniform(0, 255)));

}

### 2.3鼠标操作图像：通过点击鼠标，选择矩形ROI区域，为后续的ROI区域处理提供方便



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

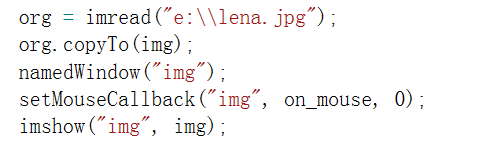
using namespace cv;

Mat org, dst, img, tmp;

void on\_mouse(int event, int x, int y, int flags, void \*ustc);

int main()

{



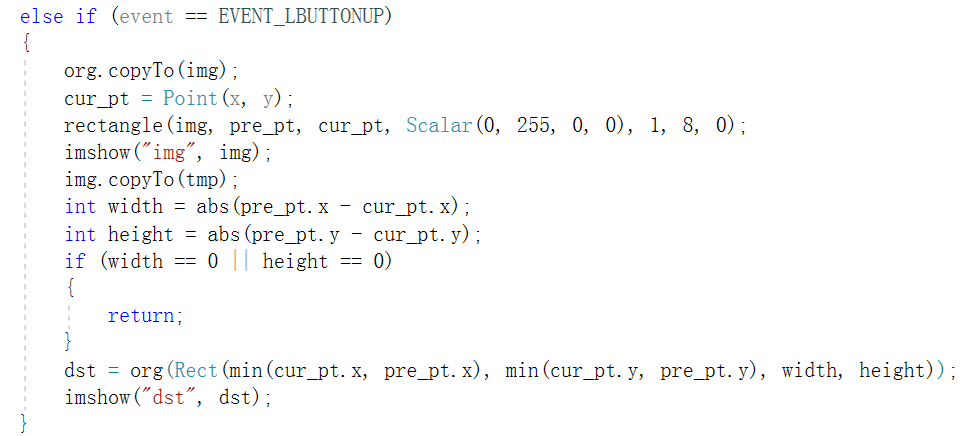
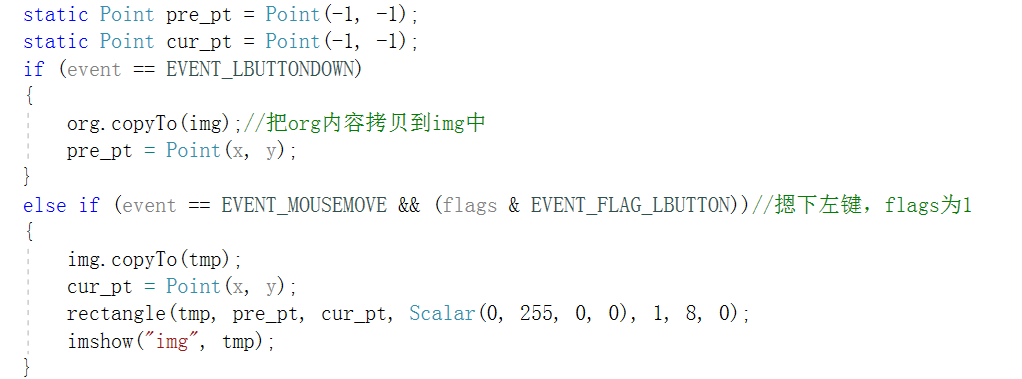
waitKey(0);

return 0;

}

void on\_mouse(int event, int x, int y, int flags, void \*ustc)

{



}