## 图像的轮廓

在目标识别中我们首先要把感兴趣的目标提取出来，而一般常见的步骤都是通过颜色或纹理提取出目标的前景图（一幅黑白图像，目标以白色显示在图像中），接下来我们要对前景图进行分析进一步地把目标提取出来，而这里常常用到的就是提取目标的轮廓。

图像的轮廓在图像处理及应用中扮演着重要的角色。那么什么是轮廓（contour）？《Learning OpenCV 3》中给出的定义是这样的——轮廓即是以某种方式表示图像中的曲线的点的列表。这种表示可以根据实际的情形不同而不同。表示一条曲线的方式有很多种。OpenCV中，轮廓是由STL风格的vector< >模板对象表示的，其中vector中的每个元素都编码了曲线上，下一点的位置信息。

OpenCV里提取目标轮廓的函数是findContours，它的输入图像8位单通道图像，常用二值图像，输出的是每一个连通区域的轮廓点的集合：vector<vector<Point>>。外层vector的size代表了图像中轮廓的个数，里面vector的size代表了轮廓上点的个数

## findContours函数功能及参数

查找图像的轮廓

void findContours( InputArray image, OutputArrayOfArrays contours,

OutputArray hierarchy, int mode,

int method, Point offset = Point());

/\*\* @overload \*/ 重载

void findContours( InputArray image, OutputArrayOfArrays contours,

int mode, int method, Point offset = Point());

InputOutputArray image,//输入图像，必须是8位单通道图像，并且被变为二值图像（即图中的所有非零像素被视为1,0像素保留0）。findContour是从二值图像中来计算轮廓的，它可以使用Canny()函数处理的图像，因为这样的图像含有边缘像素；也可以使用threshold()或者 adaptiveThreshold()处理后的图像，其边缘隐含在正负区域的交界处。

OutputArrayOfArrays contours,//检测到的轮廓，每个轮廓被表示成一个point向量；contours[i]表示一个特定的轮廓，这样，contours[i][j]将表示第i个轮廓contour[i] 的一个特定的端点

OutputArray hierarchy,//可选的输出向量，包含图像的拓扑信息。其中元素的个数和检测到的轮廓的数量相等

int mode,//检索模式，说明需要的轮廓类型和希望的返回值方式；想用何种方式来对轮廓进行提取，有四个可选的值，

* RETR\_EXTERNAL：表示只提取最外面的轮廓；
* RETR\_LIST：表示提取所有轮廓并将其放入列表；
* RETR\_CCOMP:表示提取所有轮廓并将组织成一个两层结构，其中顶层轮廓是外部轮廓，第二层轮廓是“洞”的轮廓；
* RETR\_TREE：表示提取所有轮廓并组织成轮廓嵌套的完整层级结构。

enum RetrievalModes {

RETR\_EXTERNAL = 0,

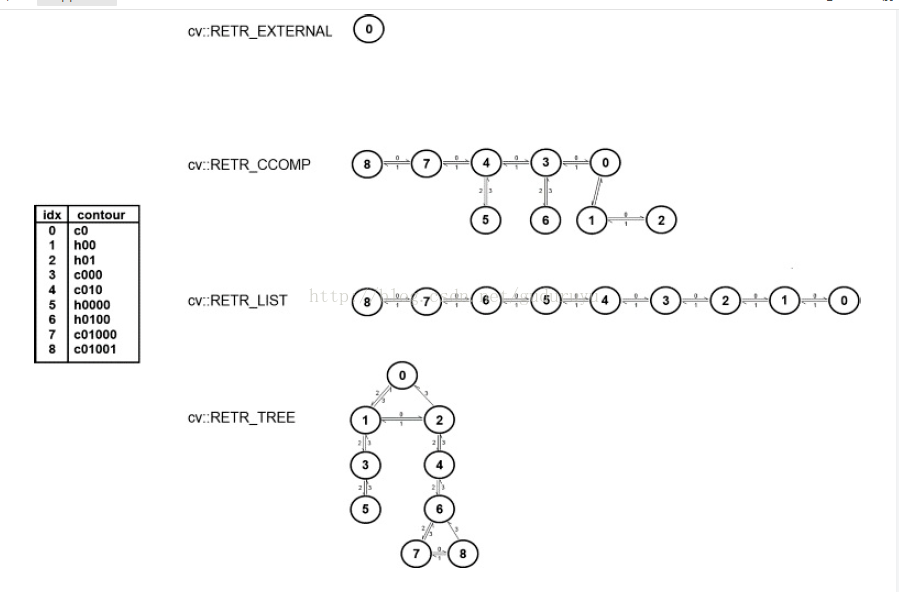
RETR\_LIST = 1,

RETR\_CCOMP = 2,

RETR\_TREE = 3,

RETR\_FLOODFILL = 4

};



int method,//轮廓近似方法轮廓如何呈现的方法，有三种可选的方法：

enum ContourApproximationModes {

CHAIN\_APPROX\_NONE = 1,将轮廓中的所有点的编码转换成点；

CHAIN\_APPROX\_SIMPLE = 2,压缩水平、垂直和对角直线段，仅保留它们的端点；

CHAIN\_APPROX\_TC89\_L1 = 3,应用Teh-Chin链近似算法中的一种风格

CHAIN\_APPROX\_TC89\_KCOS = 4 应用Teh-Chin链近似算法中的一种风格

};

Offset， 是偏移，可选，例如是Point（45,30）返回的轮廓中的所有点均作指定量的偏移。

## drawContours函数功能及参数

将轮廓绘制到图像

void drawContours(

InputOutputArray image, // 用于绘制的输入图像

InputArrayOfArrays contours, // 已经通过findContours函数找到的全部轮廓图像

int contourIdx, // 需要绘制的轮廓的指数 (-1 表示绘制所有轮廓)

Scalar& color, // 轮廓的颜色

int thickness = 1, // 轮廓线的宽度, 正数表示粗细; FILLED -1 填充；

int lineType = 8, // 轮廓线的邻域模式('4'邻域 或 '8'邻域)

cv::InputArray hierarchy = noArray(), // 可选 (从 findContours得到)

int maxLevel = INT\_MAX, // 轮廓中的最大下降

Point offset =Point() // (可选) 所有点的偏移

)

## 1代码：绘制图像的轮廓

#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

Mat src = imread("e:\\yuan.png", 1);

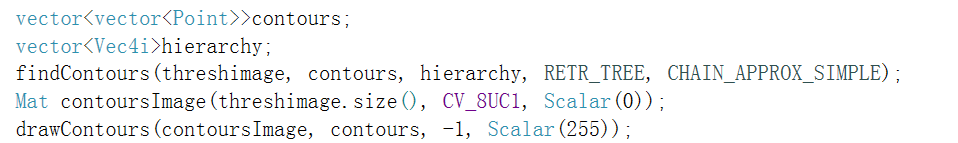
imshow("src", src);

Mat srcgray,threshimage;

cvtColor(src, srcgray, COLOR\_BGR2GRAY);

threshold(srcgray,threshimage,220,255,THRESH\_BINARY);

imshow("threshimage", threshimage);

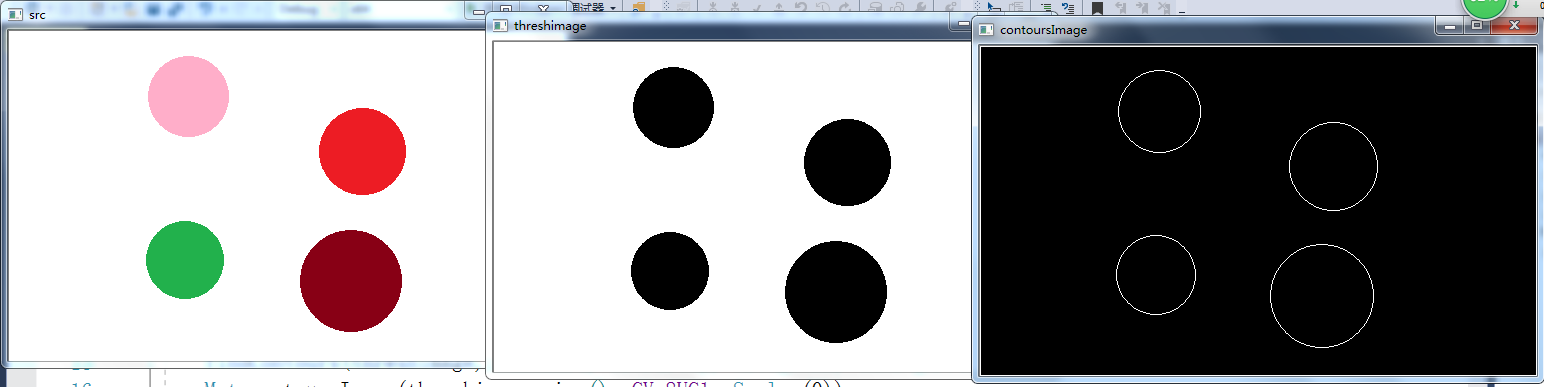


imshow("contoursImage", contoursImage);

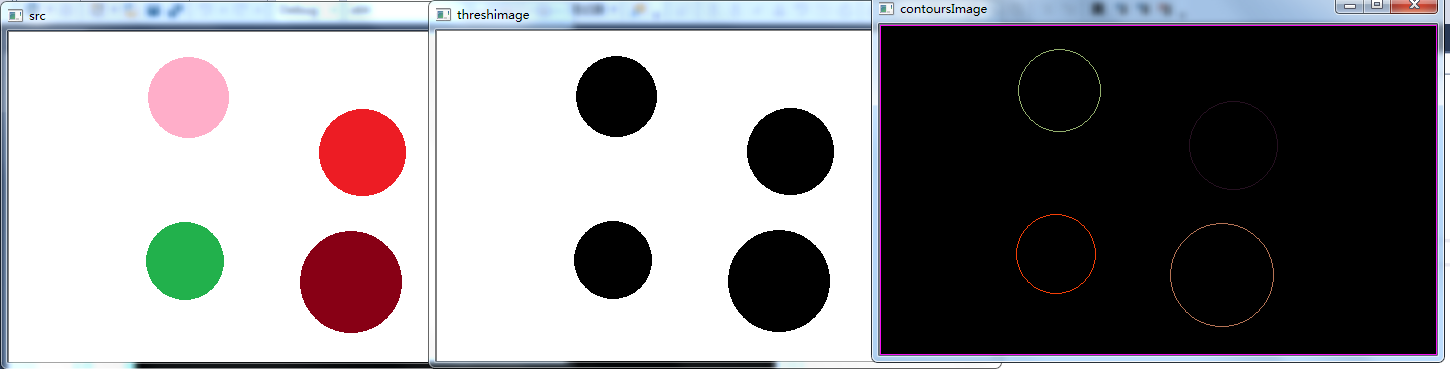
waitKey(0);

return 0;

}



## 2通过循环 绘制每个轮廓 并采用随机颜色



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

Mat src = imread("e:\\yuan.png", 1); // aodi.jpg

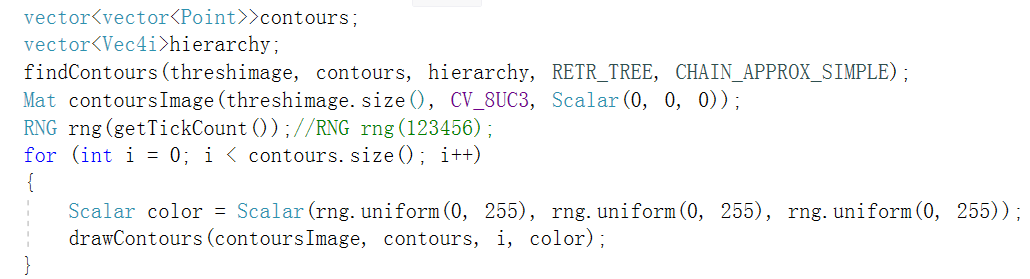
imshow("src", src);

Mat srcgray,threshimage;

cvtColor(src, srcgray, COLOR\_BGR2GRAY);

threshold(srcgray,threshimage,220,255,THRESH\_BINARY);

imshow("threshimage", threshimage);

 imshow("contoursImage", contoursImage);

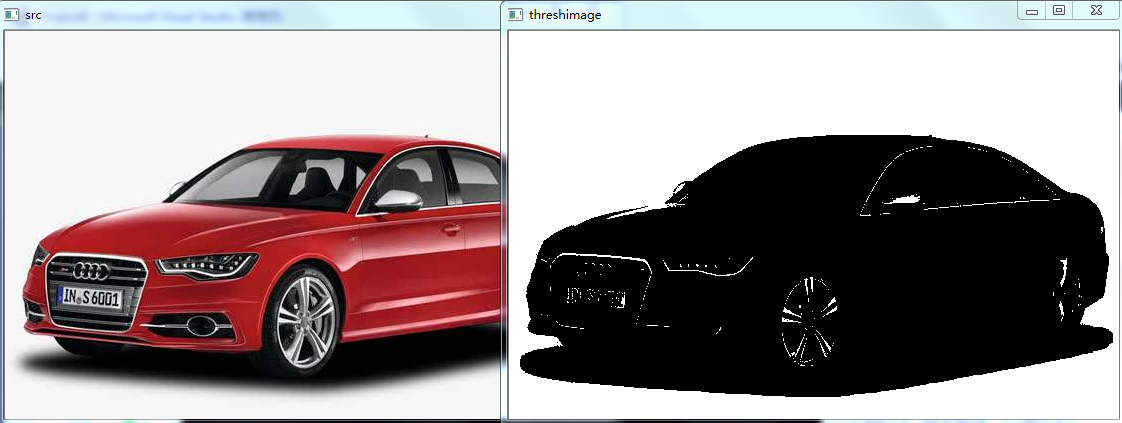
waitKey(0);

return 0;

}

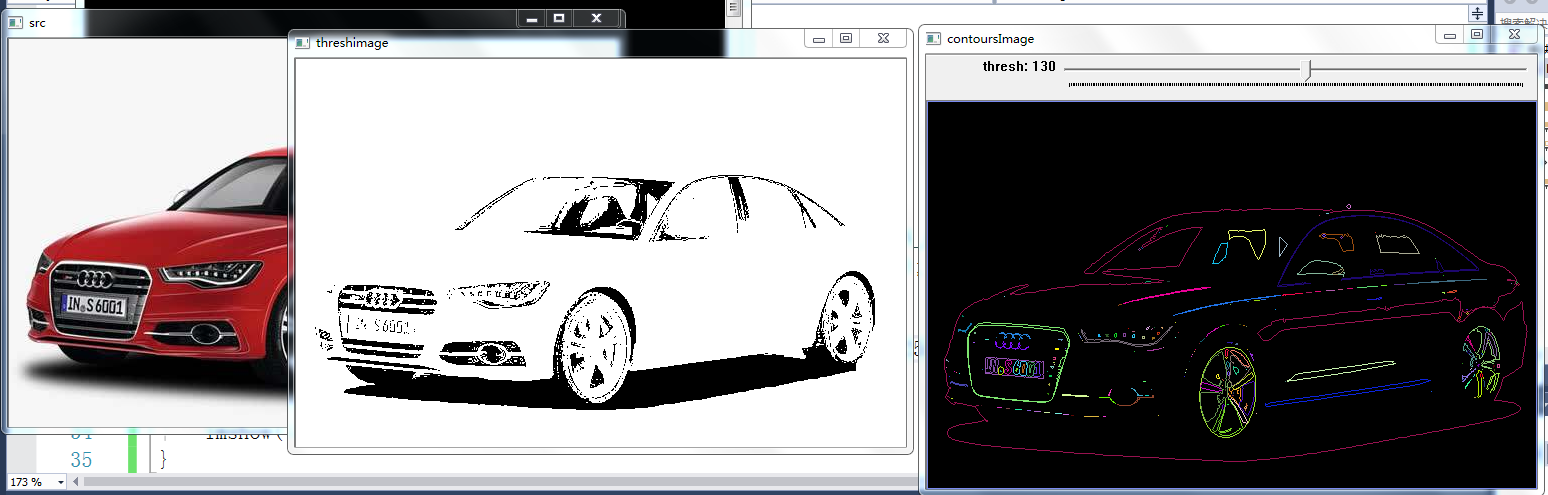
换为其他图像的效果：

Mat src = imread("e:\\aodi.jpg", 1);





## 3 滑块（ 阈值0 ~250）+findContours+ drawContours随机彩色 绘制 每个i轮廓



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int thresh = 80;

Mat src, srcgray, threshimage;

void onchange\_thresh(int, void \*);

RNG rng(getTickCount());

int main()

{

src = imread("e:\\aodi.jpg", 1);

imshow("src", src);

cvtColor(src, srcgray, COLOR\_BGR2GRAY);

threshold(srcgray, threshimage, 20, 255, THRESH\_BINARY);

imshow("threshimage", threshimage);

namedWindow("contoursImage");

createTrackbar("thresh", "contoursImage", &thresh, 250, onchange\_thresh);

onchange\_thresh(0, 0);

waitKey(0);

return 0;

}

void onchange\_thresh(int, void \*)

{

vector<vector<Point>>contours;

vector<Vec4i>hierarchy;

threshold(srcgray, threshimage, thresh, 255, THRESH\_BINARY);

findContours(threshimage, contours, hierarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE);

Mat contoursImage(threshimage.size(), CV\_8UC3, Scalar(0, 0, 0));

for (int i = 0; i < contours.size(); i++)

{

Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));

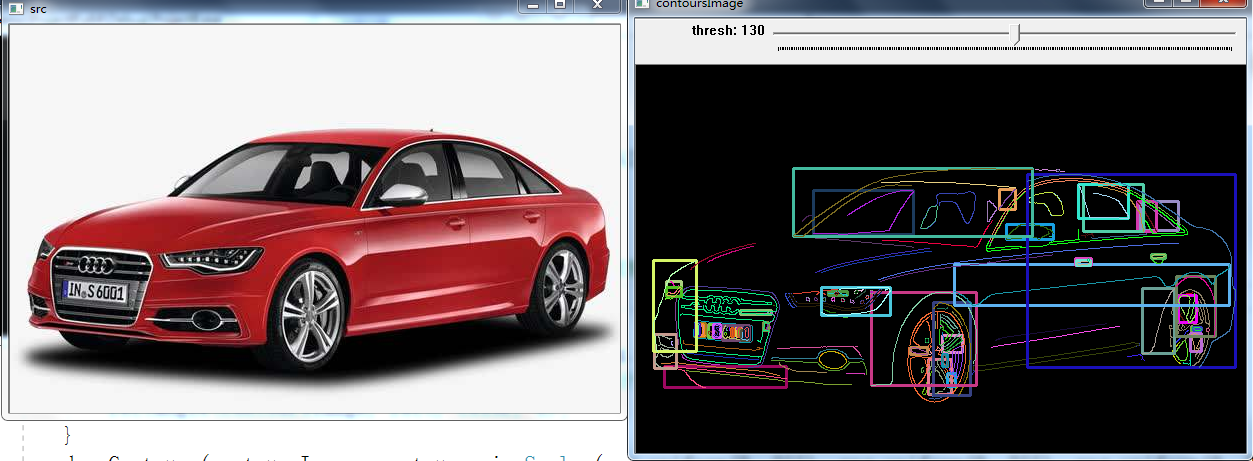
drawContours(contoursImage, contours, i, color );

}

imshow("contoursImage", contoursImage);

}

## 4 根据轮廓面积，画出矩形；滑块（canny的阈值0 ~250）+findContours+ drawContours



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int thresh = 80;

Mat src, edge;

void onchange\_thresh(int, void \*);

RNG rng(1234);

int main()

{

src = imread("e:\\aodi.jpg", 1);

imshow("src", src);

cvtColor(src, src, COLOR\_BGR2GRAY);

GaussianBlur(src, src, Size(3, 3), 0);

namedWindow("contoursImage");

createTrackbar("thresh", "contoursImage", &thresh, 250, onchange\_thresh);

onchange\_thresh(0, 0);

waitKey(0);

return 0;

}

void onchange\_thresh(int, void \*)

{

Canny(src, edge, thresh, thresh \* 2, 3);

vector<vector<Point>>contours;

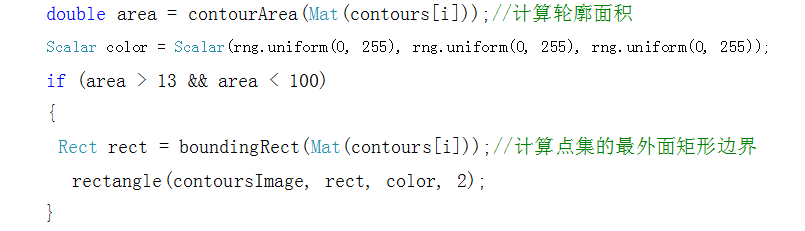
vector<Vec4i>hierarchy;

findContours(edge, contours, hierarchy, RETR\_TREE, CHAIN\_APPROX\_SIMPLE);

Mat contoursImage = Mat::zeros(edge.size(), CV\_8UC3);

for (int i = 0; i < contours.size(); i++)

{



drawContours(contoursImage, contours, i, color);

}

imshow("contoursImage", contoursImage);

}

# 2辅助学习

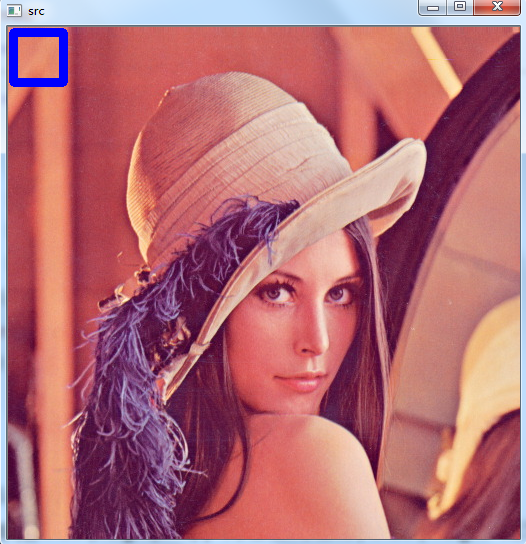
## .2.1、矩形的绘制

（1）利用对角线两点来画矩形：

void rectangle(InputOutputArray img, Point pt1 左上角, Point pt2 右下角,

const Scalar& color, int thickness = 1,

int lineType = LINE\_8, int shift = 0);



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

Mat src = imread("e:\\lena.jpg");

Point a = Point(5, 5);

Point b = Point(55, 55);

rectangle(src, a, b, Scalar(255, 0, 0),8);

imshow("src", src);

waitKey(0);

return 0;

}

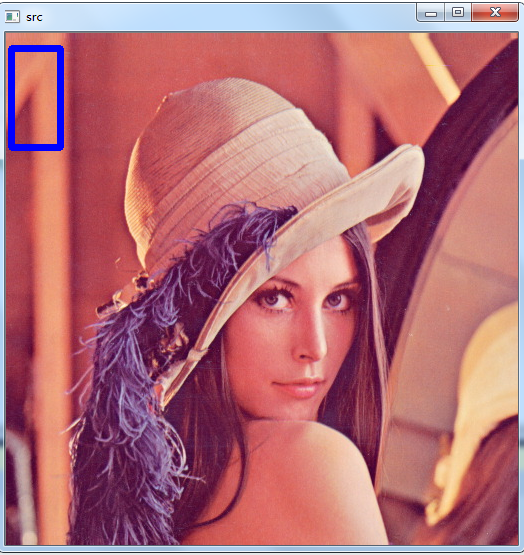
（2）传入矩形参数来画矩形：

构造函数 Rect(x,y,width,height)，x, y 为左上角坐标, width, height 则为长和宽。

void rectangle(InputOutputArray img, Rect rec,

const Scalar& color, int thickness = 1,

int lineType = LINE\_8, int shift = 0);



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

Mat src = imread("e:\\lena.jpg");

Rect rec(5, 15, 50, 100);//5 15 是矩形左上角的横纵坐标，50矩形的宽度，100矩形的高度

rectangle(src, rec, Scalar(255, 0, 0),5);

imshow("src", src);

waitKey(0);

return 0;

}

### 代码：计算若干点构成的面积

#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

vector<Point> contour;

contour.push\_back(Point(0, 0));

contour.push\_back(Point(10, 0));

contour.push\_back(Point(10, 10));

contour.push\_back(Point(0, 10));

double area = contourArea(contour);

cout <<"area =" << area << endl;//100

system("pause");

return 0;

}

## 2.2、boundingRect函数

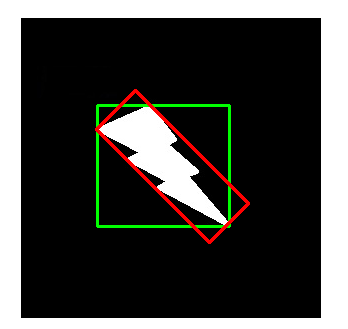
Rect boundingRect( InputArray array );

函数作用：返回包覆输入信息的最小正矩形

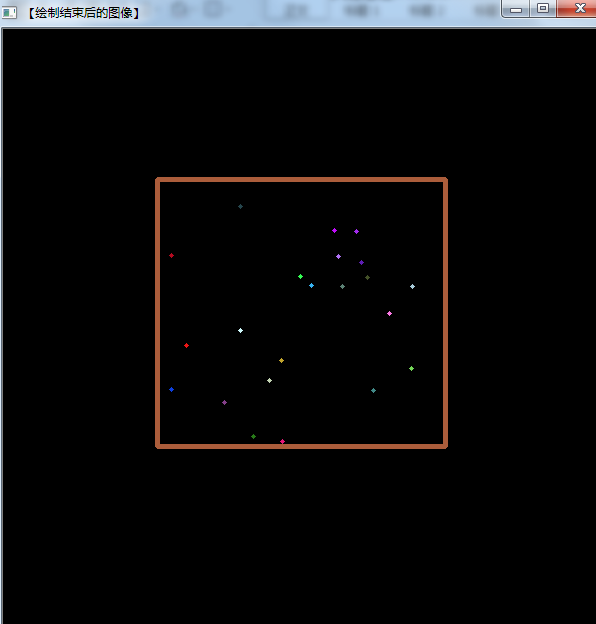
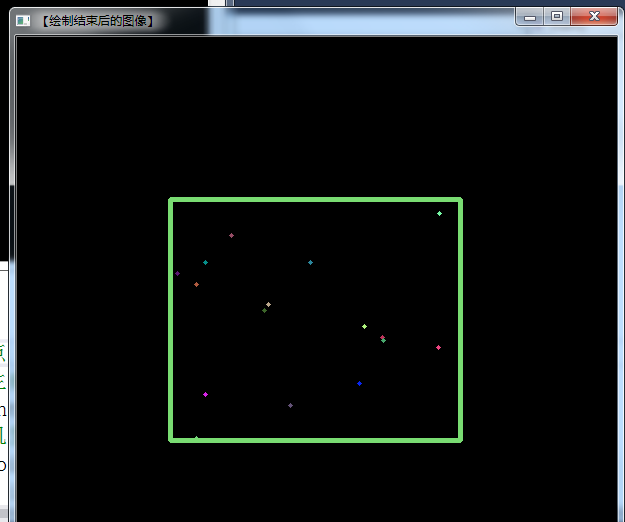
array 参数必须是vector或者Mat点集

矩形边框（Bounding Rectangle） 用一个最小的矩形，把找到的形状包起来，绿色矩形。

还有一个带旋转的矩形，面积会更小，效果见红色矩形。



### 代码：随机生成一些点，绘制最小正矩形



#include<opencv2\opencv.hpp>

#include<iostream>

using namespace std;

using namespace cv;

int main()

{

Mat srcImage(Size(600, 600), CV\_8UC3, Scalar(0));

RNG &rng = theRNG();

char key;

while (1)

{

//随机生成一些点

//首先就是随机生成点的总数量

int g\_nPointCount = rng.uniform(3, 30);

//接下来就是随机生成一些点的坐标

vector<Point> points;

for (int i = 0; i < g\_nPointCount; i++)

{

Point midPoint;

midPoint.x = rng.uniform(srcImage.cols / 4, srcImage.cols \* 3 / 4);

midPoint.y = rng.uniform(srcImage.rows / 4, srcImage.rows \* 3 / 4);

points.push\_back(midPoint);

}

//显示刚刚随机生成的那些点

for (int i = 0; i < g\_nPointCount; i++)

{

circle(srcImage, points[i], 0, Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255)), 3);

}

//在生成的那些随机点中寻找最小包围矩形，rect变量中得到了矩形

Rect rect = boundingRect(points);

rectangle(srcImage, rect, Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255)), 3); //根据得到的矩形 绘制矩形

imshow("【绘制结束后的图像】", srcImage);

key = waitKey(0);

if (key == 27)

break;

else

srcImage = Scalar::all(0);

}

waitKey(0);

return 0;

}

## 2.3、轮廓的面积contourArea

double contourArea( InputArray contour, bool oriented = false );

oriented:有默认值false，面向区域标识符，如果为true，该函数返回一个带符号的面积，其正负取决于轮廓的方向(顺时针还是逆时针)。根据这个特性可以根据面积的符号来确定轮廓的位置。如果是默认值false，则面积以绝对值的形式返回.

