目录

第一章 图像处理

一、加载图像 cv::imread()

1．第一个参数：图像文件名称,绝对或相对路径

2．第二个参数：

- IMREAD_UNCHANGED (<0) 表示加载原图
- IMREAD_GRAYSCALE (0)表示把原图作为灰度图像加载进来
- IMREAD_COLOR (>0) 表示把原图作为 RGB 图像加载进来

二、显示图像 cv::imshow()

1．第一个参数：窗口名称

2．第二参数：传入的 Mat 图像对象

三、修改图像 cv::cvtColor()

1．第一个参数：表示源图像

2．第二参数：表示色彩空间转换之后的图像

3．第三个参数：表示源和目标色彩空间如：COLOR_BGR2GRAY

四、保存图像 cv::imwrite()

1．第一个参数：需要保存到的绝对或相对路径与文件名称

2．第二参数：需要保存的 Mat 图像对象

五、拷贝特征 copyTo()

例：src.copyTo(dst,image);//把图像 image 不为 0 的像素点，用 src 的对应像素的值，赋给 dst 图像的对应像素点上。

六、保证 RBG 图片像素范围在 0-255 之间

saturate_cast<uchar>()

七、掩膜操作 filter2D()

//可用来自定义滤波，改 kernel 矩阵即可

1．定义掩膜：Mat kernel = (Mat_<char>(3,3) << 0, -1, 0, -1, 5, -1, 0, -1, 0);

2．filter2D(src, dst, src.depth(), kernel);其中 src 与 dst 是 Mat 类型变量、src.depth 表示位图深度，填-1 即可获取原图的深度

八、创建 Mat 对象

1. Mat M;//创建空白 Mat 对象

   M.create(200, 100, CV_8UC2);//高(行)200，宽(列)100，三通道

   M = Scalar(0,0,0);//像素都赋值 0

2. Mat M(200, 100, CV_8UC3, Scalar(0, 0, 255));//同上

3. int cols = dst.cols;//获取行　int rows = dst.rows;//获取列

4. int channels = image.channels();//获取通道个数

5. Mat m1; m1.create(src.size(), src.type());//创建与 src 一样大小通道的对象

九、图像像素操作

1. 单通道图像

   img.at<uchar>(y, x); //获取 img 第(y,x)的像素点

2. RGB 三通道图像

   image.at<Vec3b>(row, col)[0]; // blue

```
            image.at<Vec3b>(row, col)[1]; // green
            image.at<Vec3b>(row, col)[2]; // red
```

3. 示例(三通道取反)：

```
            int height = image.rows;//获取行
            int width = image.cols;//获取列
int channels = image.channels();//获取通道个数
for (int row = 0; row < height; row++) {
        for (int col = 0; col < width; col++) {
                    if(channels == 1){
                        int g=image.at<uchar>(row, col)；
                        image.at<uchar>(row, col)=255-g;
                    }
                else if (channels == 3) {
                            int b=image.at<Vec3b>(row, col)[0]；
                            int g=image.at<Vec3b>(row, col)[1];
                            int r=image.at<Vec3b>(row, col)[2];
                        image.at<Vec3b>(row,col)[0]=255-b; // blue
                        image.at<Vec3b>(row,col)[1]=255-g;// green
                         image.at<Vec3b>(row, col)[1]=255-r;//red
                    }
            }
}
```

4. 取反函数 bitwise_not(src,dst);

输入 src，输出 dst

十、图像像素归一化(可用来降低图像亮度)

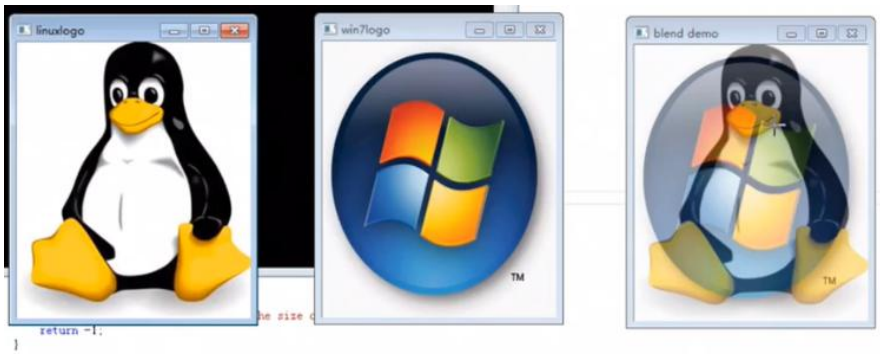
normalize(src, src, min , max, NORM_MINMAX, -1);

1. 第一个参数：输入图像
2. 第二个参数：输出图像
3. 第三四个参数：把图像像素点最大最小值缩放到 min~max
4. 第五个参数：方法，一般填 NORM_MINMAX
5. 第六个参数：填-1 即可

十一、图像混合（融合）操作 addWeighed()

addWeighted(src1, (1 -0.5), src2, 0.5, 0.0, dst);

1. 第一个参数：表示输入第一张图像
2. 第二个参数：表示第一张图像所占权重（0~1）
3. 第三个参数：表示输入第二张图像
4. 第四个参数：表示第二张图像所占权重
5. 第五个参数：在混合相加后在加上多少像素值
6. 第六个参数：最后输出的图像



十二、图像对比度和亮度

1．增加或减小对比度，对像素点乘上一个系数

2．增加或减小亮度，对像素点加减一个系数

3．利用 saturate_cast<uchar>()函数使求出的值在 0~255

十三、绘制图形

1. point(x,y);//表示坐标为 x,y 的像素点

Point p; p.x = 10; p.y = 8;

or

p = Pont(10,8);

2. Scalar(a, b, c);// blue, green, red 表示 RGB 三个通道，表示颜色

3. line(src, p1, p2,Scalar(255, 0, 0) , 1, LINE_8);//画线

   (1) 第一个参数：表示输入图像

   (2) 第二个参数：表示划线起始点

   (3) 第三个参数：表示划线结束点

   (4) 第四个参数：表示划线的颜色

   (5) 第五个参数：表示划线的粗细

   (6) 第六个参数：LINE_4/LINE_8/LINE_AA(一般用 8 位，AA 效果好，消耗大)(可以直接填 4、8)

4. rectangle(src, rect, color, 2, LINE_8);//画矩形

   (1) 第一个参数：表示输入图像

   (2) 第二个参数：Rect rect = Rect(200, 100, 300, 300);//前面两个为起始点，后面两个为长宽

(3) 第三个参数：划线的颜色

(4) 第四个参数：划线的粗细

(5) 第五个参数：LINE_4/LINE_8/LINE_AA(可以直接填 4、8)

5. ellipse(src,point(20,20), Size(10,50), 0, 0, 360, color, 2, LINE_8);//椭圆

(1) 第一个参数：表示输入图像

(2) 第二个参数：表示椭圆中心点

(3) 第三个参数：表示椭圆的长轴、短轴

(4) 第四个参数：表示椭圆图像的旋转角度

(5) 第五六个参数：表示从多少度绘制到多少度

(6) 第七个参数：划线的颜色

(7) 第八个参数：划线的粗细

(8) 第九个参数：LINE_4/LINE_8/LINE_AA(可以直接填 4、8)

6. circle(src, point(20,20), 150, color, 2, 8);//画圆

(1) 第一个参数：表示输入图像

(2) 第二个参数：表示圆心

(3) 第三个参数：表示半径

(4) 第四个参数：表示划线颜色

(5) 第五个参数：表示划线粗细

(6) 第六个参数：LINE_4/LINE_8/LINE_AA(可以直接填 4、8)

7. fillPoly(src, ppts, npt, 1, color, 8);//绘制填充图形

(1) 第一个参数：表示输入图像

(2) 第二个参数：Point pts[1][5];//例

pts[0][0] = Point(100, 100);

pts[0][1] = Point(100, 200);

pts[0][2] = Point(200, 200);

pts[0][3] = Point(200, 100);

pts[0][4] = Point(100, 100);

const Point* ppts[] = { pts[0] };//点集头指针

(3) 第三个参数：int npt[] = { 5 };//点集个数

(4) 第四个参数：划线粗细

(5) 第五个参数：划线颜色

(6) 第六个参数：LINE_4/LINE_8/LINE_AA(可以直接填 4、8)

8. putText(src,"Hello",Point(300,300),CV_FONT_HERSHEY_COMPLEX, 1.0, Scalar(12, 23, 200), 3, 8);//绘制文字

(1) 第一个参数：输入图像

(2) 第二个参数：写的字体内容

(3) 第三个参数：起始点

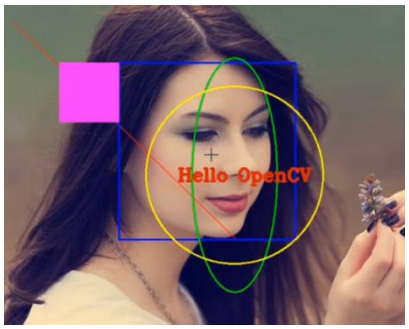
(4) 第四个参数：字体样式

(5) 第五个参数：字体大小

(6) 第六个参数：字体颜色

(7) 第七个参数：字体粗细

(8) 第八个参数：LINE_4/LINE_8/LINE_AA(可以直接填 4、8)



十四、图像模糊(卷积原理)

1. 均值模糊 blur(src, dst, Size(x,y), Point(-1,-1));
   (1) 第一个参数：输入图像
   (2) 第二个参数：输出图像
   (3) 第三个参数：用于卷积的窗口大小(必须为奇数，因为要赋值给窗口中心像素点，不是奇数没有中心像素点)
   (4) 第四个参数：一般默认 Point(-1,-1)，表示赋值给窗口中心点覆盖的像素点
2. 高斯模糊 GaussianBlur(src,dst, Size(11, 11), 5, 5);
   (1) 第一个参数：输入图像
   (2) 第二个参数：输出图像
   (3) 第三个参数：用于卷积的窗口大小(必须为奇数，因为要赋值给窗口中心像素点，不是奇数没有中心像素点)
   (4) 第四五个参数：所分配的权值
3. 中值模糊 medianBlur(src, dst, 3);//可用于去除噪点
   (1) 第一个参数：输入图像
   (2) 第二个参数：输出图像
   (3) 第三个参数：用于卷积的窗口大小

4. 高斯双边滤波 bilateralFilter(src, dst, 15, 100, 5);//可用于磨皮操作

(1) 第一个参数：输入图像

(2) 第二个参数：输出图像

(3) 第三个参数：做处理的半径

(4) 第四个参数：像素差，在差值内的像素点会进入计算

(5) 第五个参数：如果第三个参数小于 0，则根据此参数计算处理半径

十五、腐蚀与膨胀

1. 创建覆盖窗口

Mat kernel = getStructuringElement(MORPH_RECT,Size(5,5), Point(-1,-1));

(1) 第一个参数：窗口形状(MORPH_RECT \MORPH_CROSS \MORPH_ELLIPSE)

(2) 第二个参数：窗口覆盖大小，必须为奇数

(3) 第三个参数：一般为 Point(-1, -1)，为窗口中心像素点

2. 腐蚀与膨胀

erode(src, dst, kernel);//腐蚀

dilate(src, dst, kernel);//膨胀

(1) 第一个参数：输入图像

(2) 第二个参数：输出图像

(3) 第三个参数：为创建的覆盖窗口

十六、形态学操作(开闭运算，顶帽，黑帽，梯度)

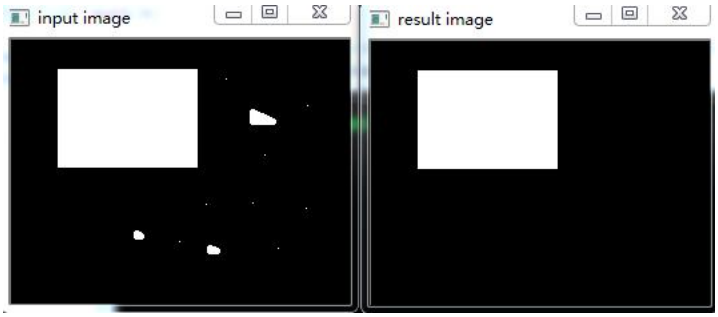
morphologyEx(src, dst, CV_MOP_BLACKHAT, kernel);

(1) 第一个参数：输入图像

(2) 第二个参数：输出图像

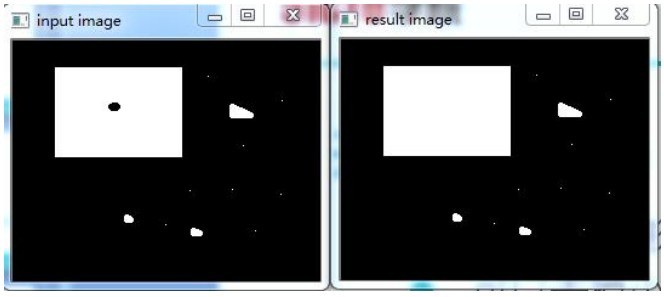
(3) 第三个参数：形态学操作

① **CV_MOP_OPEN(开运算，先腐蚀后膨胀)**



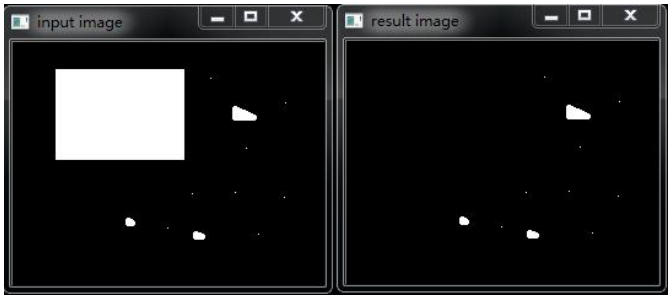
可用于去除二值化图里小的对象(黑色为背景)

② **CV_MOP_CLOSE(闭运算，先膨胀后腐蚀)**



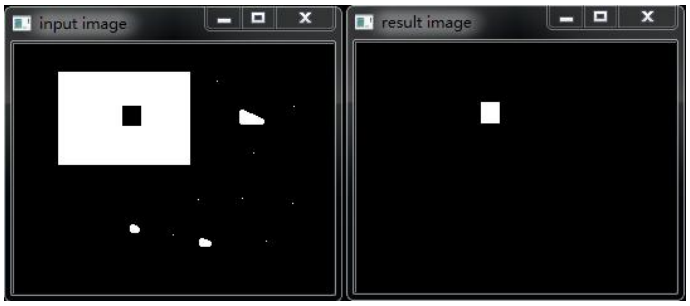
可用于在二值化图里填充小的洞(黑色为背景)

③ **CV_MOP_TOPHAT(原图像与开操作差值)**



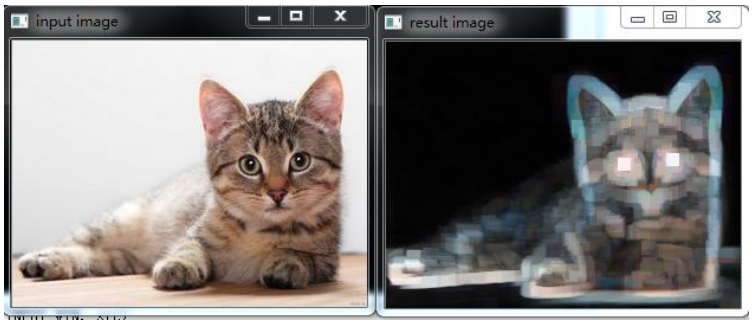
可用于获取二值化图里小的对象(黑色为背景)

④ **CV_MOP_BLACKHAT(闭操作源图像差值)**



可用于获取二值图里被填补的小洞(黑色为背景)

⑤ **CV_MOP_GRADIENT(膨胀减去腐蚀)**



(4) 第四个参数：覆盖窗口 kernel

十七、二值化

1. adaptiveThreshold(

Mat src, // 输入的灰度图像

Mat dest, // 二值图像

double maxValue, // 二值图像最大值

int adaptiveMethod // 自适应方法，两个任选一个

// (ADAPTIVE_THRESH_MEAN_C ， ADAPTIVE_THRESH_GAUSSIAN_C)

int thresholdType,//阈值类型

//(THRESH_BINARY,THRESH_BINARY_INV)

int blockSize, // 块大小

double C // 常量 C 可以是正数，0，负数

)

2. 阈值二值化 threshold(src, dst, 127,maxval, THRESH_BINARY);

(1) 第一个参数：输入图像

(2) 第二个参数：输出图像

(3) 第三个参数：像素阈值

(4) 第四个参数：所有像素点达到的最大像素值

(5) 第五个参数:

① THRESH_BINARY 小于阈值的像素点置 0，大于阈值的像素点置 maxval

② THRESH_BINARY_INV 此时小于阈值的像素点置 maxval， 大于阈值的像素点置 0；

③ THRESH_TRUNC 此时小于阈值的像素点保持原数值，大于阈值的像素点置阈值；

④ THRESH_TOZERO 此时小于阈值的像素点置 0，大于阈值的像素点保持原数值；

⑤ THRESH_TOZERO_INV 此时小于阈值的像素点保持原数值，大于阈值的像素点置 0。

注：自适应阈值：

THRESH_TRIANGLE |THRESH_BINARY

CV_THRESH_OTSU|THRESH_BINARY

后面为方法，前面为两个自适应阈值的参数任意一个即可

十八、图像金字塔

1. pyrUp(Mat src, Mat dst, Size(src.cols*2, src.rows*2))

   上采样：生成的图像是原图在宽与高各放大两倍

2. pyrDown(Mat src, Mat dst, Size(src.cols/2, src.rows/2))

   下采样：生成的图像是原图在宽与高各缩小 1/2

3. subtrcat(g1,g2,dst,Mat());//高斯不同

   g1、g2：两张不同参数进行了高斯模糊的图像

   dst：输出的图像（高斯模糊的差值）

   （高斯不同是图像的内在特征，在灰度图像增强、角点检测中经常用到）

十九、处理图像外边缘

copyMakeBorder（

- Mat src, // 输入图像
- Mat dst, // 添加边缘图像
- int top, // 增加的边缘长度，一般上下左右都取相同值，
- int bottom,
- int left,
- int right,
- int borderType // 边缘类型

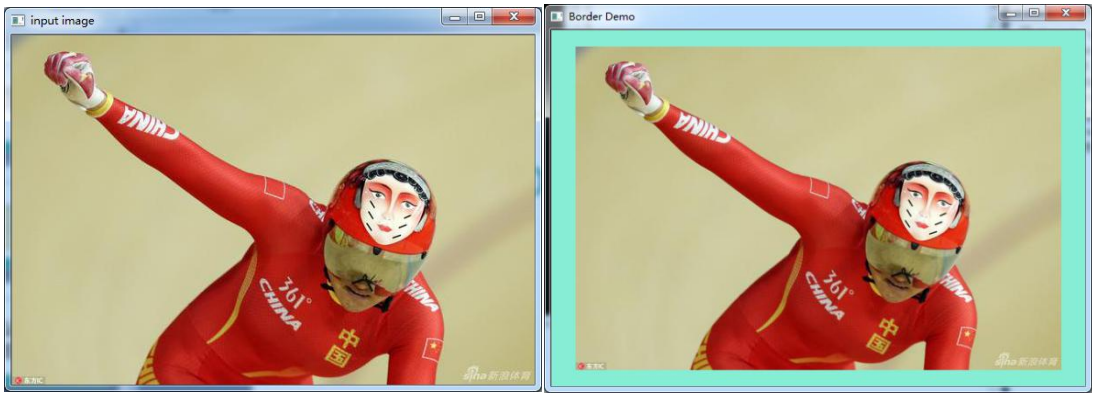
- Scalar value

）

边缘类型 borderType：

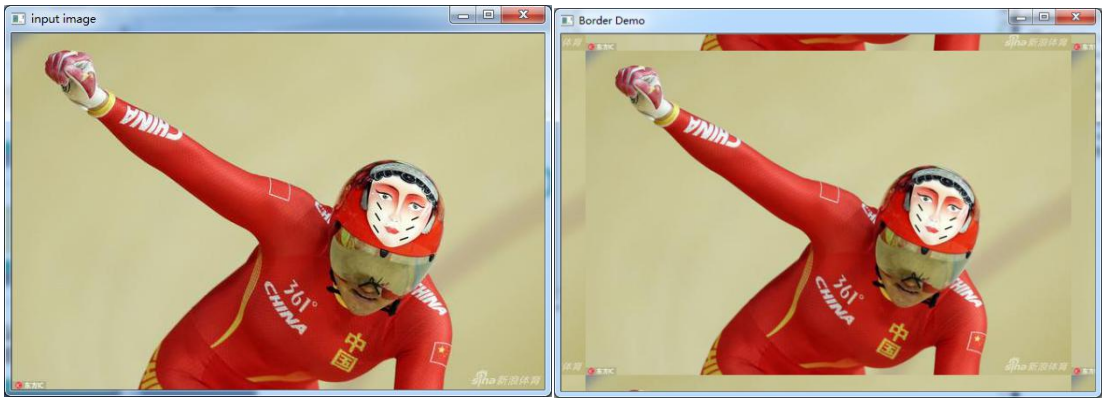
1. BORDER_DEFAULT - 周围对应像素点填充

2. BORDER_CONSTANT – 填充边缘用指定像素值



3. BORDER_REPLICATE – 都用周围的那一个像素点填充边缘

4. BORDER_WRAP – 用另外一边补偿



二十、Sobel 算子边缘检测

Sobel (

InputArray Src // 输入图像

OutputArray dst// 输出图像，大小与输入图像一致

int depth // 输出图像深度.

int dx // X 方向，几阶导数

int dy // Y 方向，几阶导数.

int ksize, SOBEL 算子 kernel 大小，必须是 1、3、5、7、

double scale = 1

double delta = 0

int borderType = BORDER_DEFAULT

);

depth 深度取值如下：

Input depth ( )	Output depth (ddepth)
CV_8U	-1/CV_16S/CV_32F/CV_64F
CV_16U/CV_16S	-1/CV_32F/CV_64F
CV_32F	-1/CV_32F/CV_64F
CV_64F	-1/CV_64F

例： GaussianBlur(src, dst, Size(3,3), 0, 0, BORDER_DEFAULT);//先高斯模糊，去除噪点干扰

cvtColor(dst, gray_src, COLOR_RGB2GRAY);//再转成灰度

Sobel(gray_src, xgrad, CV_16S, 1, 0, 3);//对 x 方向

Sobel(gray_src, ygrad, CV_16S, 0, 1, 3);//对 y 方向

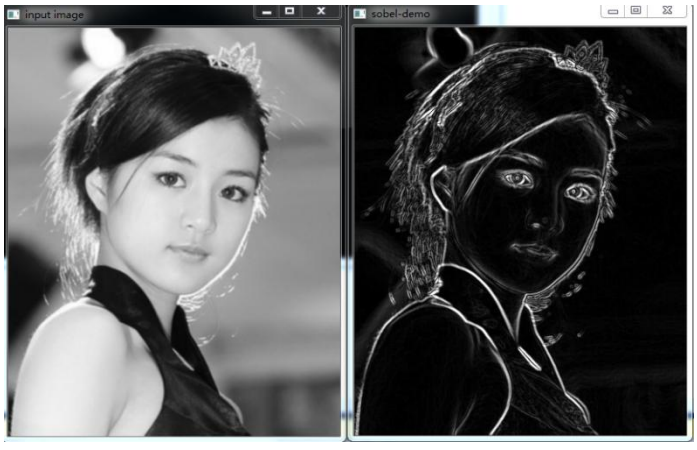
convertScaleAbs(xgrad, xgrad);

convertScaleAbs(ygrad, ygrad);

// 上面是计算图像 A 的像素绝对值，输出到图像 B

addWeighted(xgrad, 0.5, ygrad, 0.5, 0, xygrad);//x 和 y 相加

最终的到 sobel 处理得到的最终图 xygrad



二十一、Laplance 算子边缘检测

Laplacian(src,dst,CV_16S,3);

(1) 第一个参数：输入图像

(2) 第二个参数：输出图像

(3) 第三个参数：输出图像深度，一般 CV_16S 即可

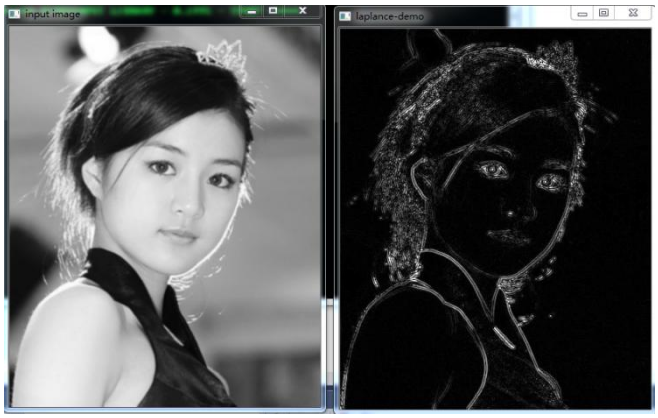
(4) 第四个参数：kernel 大小，必须是 1、3、5、7

例： GaussianBlur(src, dst, Size(3,3), 0, 0, BORDER_DEFAULT);//先高斯模糊，去除噪点干扰

cvtColor(dst, gray, COLOR_RGB2GRAY);//再转成灰度

Laplacian(gray,output,CV_16S,3);//Laplance 算子边缘检测

convertScaleAbs(output,output);//计算图像像素绝对值输出



二十二、Canny 算子边缘检测

Canny（

InputArray src, // 8-bit 的输入图像

OutputArray edges,// 输出边缘图像， 一般都是二值图像，背景是黑色

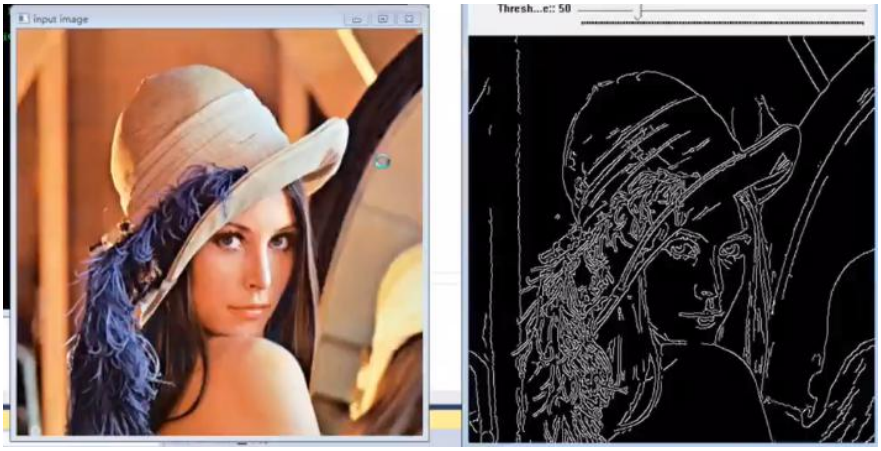
double threshold1,// 低阈值，常取高阈值的 1/2 或者 1/3

double threshold2,// 高阈值

int aptertureSize,// Soble 算子的 size，通常 3x3，取值 3

bool L2gradient // 选择 true 表示是 L2 来归一化，否则用 L1 归一化，默认的话是 false L1

）



二十三、霍夫变换(直线，圆检测)

1. 直线检测（先边缘检测，一般是先 Canny 一下）

cv::HoughLinesP(

InputArray src, // 输入图像，必须 8-bit 的灰度图像

OutputArray lines, // 输出结果，四维，两端点坐标

double rho, // 生成极坐标时候的像素扫描步长，一般取 1

double theta, //生成极坐标时候的角度步长，一般取值 CV_PI/180.0

int threshold, // 阈值，只有获得足够交点的极坐标点才被看成是直线，一般给 10

double minLineLength=0;// 最小直线长度

double maxLineGap=0;// 最大间隔，结果直线不连续，则加大

)

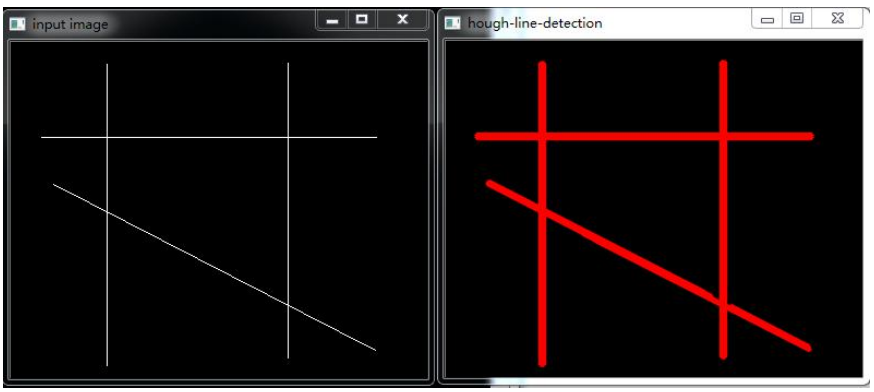
例：

1.先边缘检测，然后转为三通道图

```
imshow(INPUT_TITLE, src);
Canny(src, src_gray, 150, 200, 3);
cvtColor(src_gray, dest, CV_GRAY2BGR);
```

2.再进行霍夫变换直线检测，遍历绘制直线

```
vector<Vec4f> plines;
HoughLinesP(src_gray, plines, 1, CV_PI / 180, 10, 0, 10);
for (size_t i = 0; i < plines.size(); i++) {
    Vec4f hl = plines[i];
    line(dest, Point(hl[0], hl[1]), Point(hl[2], hl[3]), Scalar(0, 0, 255), 3, CV_AA);
}
imshow(OUTPUT_TITLE, dest);
```



2. 圆检测（对噪声比较敏感，所以首先要对图像做中值滤波）

HoughCircles(

InputArray image, // 输入图像 ,必须是 8 位的单通道灰度

图像

OutputArray circles, // 输出结果，三维，圆形坐标加半径

Int method, // 方法 - HOUGH_GRADIENT

Double dp, // dp = 1;

Double mindist, // 10 最短距离-可以分辨是两个圆的，否则认为是同心圆

Double param1, // canny edge detection low threshold 默认100

Double param2, // 中心点累加器阈值 – 候选圆心

Int minradius, // 最小半径

Int maxradius//最大半径

)

例：

```
char INPUT_TITLE[] = "input image";
char OUTPUT_TITLE[] = "hough circle demo";
namedWindow(INPUT_TITLE, CV_WINDOW_AUTOSIZE);
namedWindow(OUTPUT_TITLE, CV_WINDOW_AUTOSIZE);

// show input image
// medianBlur(src, src, 5);
cvtColor(src, src, CV_BGR2GRAY);
GaussianBlur(src, dest, Size(5, 5), 0, 0);
imshow(INPUT_TITLE, src);

// 基于灰度空间
vector<Vec3f> circles;
HoughCircles(dest, circles, HOUGH_GRADIENT, 1, 10, 100, 30, 5, 50);
// 重新转回到RGB色彩空间
cvtColor(dest, dest, CV_GRAY2BGR);
for (size_t i = 0; i < circles.size(); i++) {
    Vec3f c3 = circles[i];
    circle(dest, Point(c3[0], c3[1]), c3[2], Scalar(0, 0, 255), 3, LINE_AA);
    circle(dest, Point(c3[0], c3[1]), 2, Scalar(0, 0, 255), 3, LINE_AA);
}
imshow(OUTPUT_TITLE, dest);

waitKey(0);
return 0;
```

二十四、像素重映射（翻转，旋转等操作）

Remap(

InputArray src,// 输入图像

OutputArray dst,// 输出图像

InputArray map1,// x 映射表 需要 CV_32FC1/CV_32FC2

InputArray map2,// y 映射表需要 CV_32FC1/CV_32FC2

int interpolation,// 选择的插值方法，常见线性插值，可选择立方等，INTER_LINEAR

int borderMode,// 补充边缘方式，BORDER_CONSTANT

const Scalar borderValue// 补充边缘的颜色 color

)

例：

```
Mat map_x, map_y;
map_x.create(src.size(), CV_32FC1);//x 方向映射
map_y.create(src.size(), CV_32FC1);//y 方向映射
for (int row = 0; row < src.rows; row++) {
```

```
        for (int col = 0; col < src.cols; col++) {
            map_x.at<float>(row, col) = (src.cols - col - 1); //x 对调
            map_y.at<float>(row, col) = row; //y 不变
                }
            }
    //以上是做好了 x 和 y 方向上的映射表
    remap(src,dst,map_x,map_y, INTER_LINEAR, BORDER_CONSTANT,
Scalar(0, 255, 255));//再通过这个 API 进行处理输出 dst
```

二十五、直方图均衡化(加强单通道图片对比度)

```
    equalizeHist(
    InputArray src,//输入图像，必须是 8-bit 的单通道图像
    OutputArray dst//  输出结果
    )
```

二十六、通道分离 split()

split(// 把多通道图像分为多个单通道图像

const Mat &src, //输入图像

Mat* mvbegin）// 输出的通道图像数组

二十七、模板匹配

matchTemplate(

InputArray image,// 源图像，必须是 8-bit 或 32-bit 浮点数图像

InputArray templ,// 模板图像，类型与输入图像一致

OutputArray result,// 输出结果，必须是单通道 32 位浮点数，假设源图像 WxH,模板图像 wxh,则必须为 W-w+1, H-h+1 的大小。

int method,//使用的匹配方法

InputArray mask=noArray()//(optional)

)

- method 取 0~5，0 和 1 为结果图像中最小值的像素点为匹配点，其余为结果图像中最大值的像素点点为匹配点

```
enum cv::TemplateMatchModes {
    cv::TM_SQDIFF = 0,
    cv::TM_SQDIFF_NORMED = 1,
    cv::TM_CCORR = 2,
    cv::TM_CCORR_NORMED = 3,
    cv::TM_CCOEFF = 4,
    cv::TM_CCOEFF_NORMED = 5
}
```

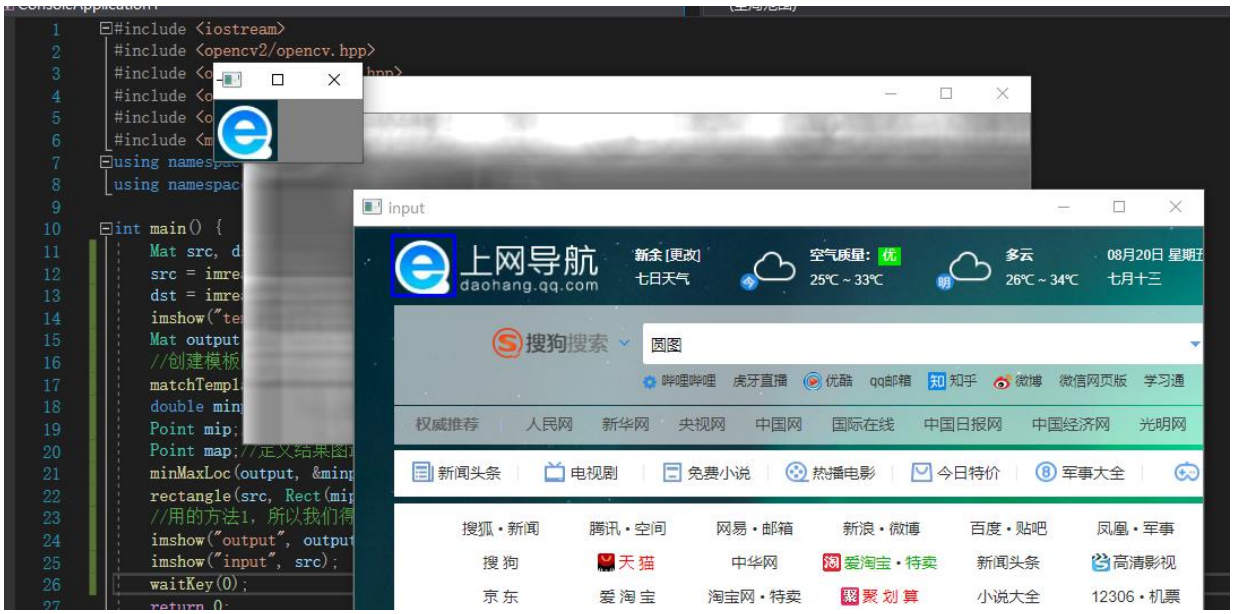
1，3，5 是做了归一化的，可直接观察输出结果图，0，2，4 没有做归一化，所以要观察输出结果图就要再加一步归一化

normalize(output, output, 0, 1, NORM_MINMAX, -1);

//将 output 的图像像素值都归成是 0-1 之间(第三和第四个参数)

例：

```
Mat src, dst;
src = imread("C://Users//hp//Desktop//opencv_src//03.png");//读取原图
dst = imread("C://Users//hp//Desktop//opencv_src//03_1.png");//读取模板图
imshow("temple", dst);
Mat output(Size(src.cols-dst.cols+1,dst.rows-dst.rows+1),CV_32FC1);
//创建模板匹配的输出结果图,大小为原图减模板图+1，为单通道32位浮点数
matchTemplate(src,dst,output,1);//调用模板匹配函数，用方法1
double minp, maxp;//定义结果图最小值像素，最大值像素
Point mip;//定义结果图最小值像素点坐标
Point map;//定义结果图最大值像素点坐标
minMaxLoc(output, &minp, &maxp, &mip, &map);//找出结果图所对应的最大最小值结果
rectangle(src, Rect(mip.x, mip.y, dst.cols, dst.rows), Scalar(255, 0, 0), 2, 8);
//用的方法1，所以我们得到的应该是结果图中像素值最小的坐标点，所以对该点在原图上画出区域
imshow("output", output);
imshow("input", src);
waitKey(0);
return 0;
```



二十八、轮廓发现与绘制轮廓

1.在二值图像上发现轮廓使用 API cv::findContours(

InputOutputArray binImg, // 输入图像，非 0 的像素被看成 1,0 的像素值保持不变，8-bit

OutputArrayOfArrays contours,// 全部发现的轮廓对象

OutputArray, hierachy// 图该的拓扑结构，可选，该轮廓发现算 法正是基于图像拓扑结构实现。

int mode, // 轮廓返回的模式

int method,// 发现方法

Point offset=Point()// 轮廓像素的位移，默认（0, 0）没有位移

)

- contours 定义：vector<vector<Point> > contours;
- hierachy 定义：vector<Vec4i> hierarchy;
- 第四个参数：int 型的 mode，定义轮廓的检索模式：
  (1) 取值一：CV_RETR_EXTERNAL 只检测最外围轮廓，包含在外围 轮廓内的内围轮廓被忽略。//常用
  (2) 取值二：CV_RETR_LIST 检测所有的轮廓，包括内围、外围 轮廓，但是检测到的轮廓不建立等级关系，彼此之间独立，没 有等级关系，这就意味着这个检索模式下不存在父轮廓或内嵌 轮廓，所以 hierarchy 向量内所有元素的第 3、第 4 个分量都会 被置为-1。
  (3) 取值三：CV_RETR_CCOMP 检测所有的轮廓，但所有

轮廓只　建立两个等级关系，外围为顶层，若外围 内的内围轮廓还包 含了其他的轮廓信息，则内围内的所有轮廓均归属于顶层。

(4) 取值四：CV_RETR_TREE， 检测所有轮廓，所有轮廓建立一个　等级树结构。外层轮廓包含内层轮廓，内层轮廓还可以继续包　含内嵌轮廓。//常用

- 第五个参数：int 型的 method，定义轮廓的近似方法：

(1) 取值一：CV_CHAIN_APPROX_NONE 保存物体边界上所有连续的轮廓点到 contours 向量内。

(2) 取值二：CV_CHAIN_APPROX_SIMPLE 仅保存轮廓的拐点信息，把所有轮廓拐点处的点保存入 contours 向量内，拐点与拐点之间直线段上的信息点不予保留。//常用

(3) 取值三和四：CV_CHAIN_APPROX_TC89_L1，CV_CHAIN_APPROX_TC89_KCOS 使用 teh-Chinl chain 近似算法。

2.在二值图像上发现轮廓使用 API cv::findContours 之后对发现的轮廓数据进行绘制显示

```
drawContours(
InputOutputArray　binImg, // 输出图像
 OutputArrayOfArrays　contours,//　全部发现的轮廓对象
Int contourIdx// 轮廓索引号
const Scalar & color,// 绘制时候颜色
int　thickness,// 绘制线宽
```

int　lineType ,// 线的类型 LINE_8

InputArray hierarchy,// 拓扑结构图

int maxlevel,// 最大层数， 0 只绘制当前的，1 表示绘制绘制当前及其内嵌的轮廓

Point offset=Point()// 轮廓位移，可选

- 通过循环遍历 contours 轮廓对象，进行一个一个画出轮廓

```
void Demo_Contours(int, void*) {
    vector<vector<Point>> contours;
    vector<Vec4i> hierarchy;
    Canny(src, dst, threshold_value, threshold_value * 2, 3, false);
    findContours(dst, contours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE, Point(0, 0));

    Mat drawImg = Mat::zeros(dst.size(), CV_8UC3);
    for (size_t i = 0; i < contours.size(); i++) {
        Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));
        drawContours(drawImg, contours, i, color, 2, LINE_8, hierarchy, 0, Point(0, 0));
    }
    imshow(output_win, drawImg);
}
```

二十九、凸包(找出轮廓最小外接多边形)

convexHull(

InputArray points,// 输入候选点 contours，来自 findContours

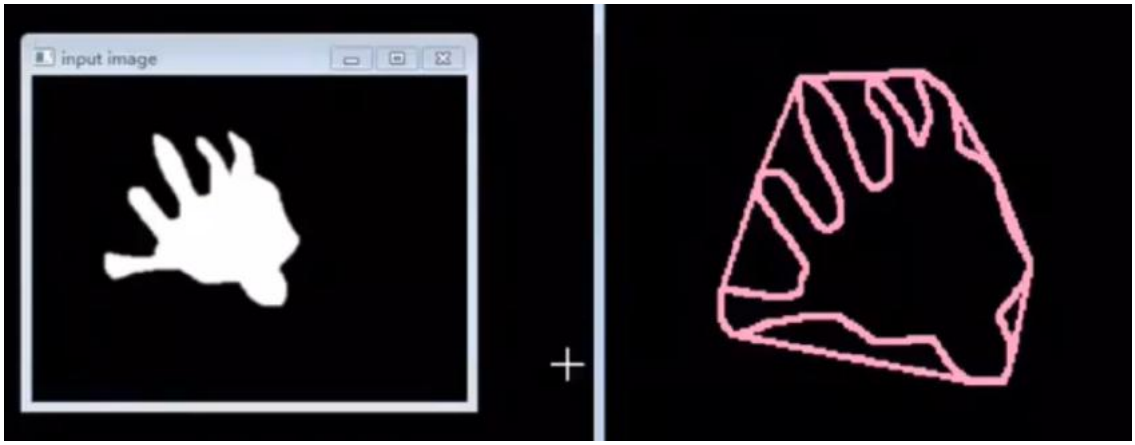
OutputArray hull,// 凸包

bool clockwise,// default true，顺时针方向寻找

bool returnPoints）// true 表示返回点个数，如果只有一个则忽略

例：

```
void Threshold_Callback(int, void*) {
    Mat threshold_output;
    vector<vector<Point>> contours;
    vector<Vec4i> hierarchy;

    threshold(src_gray, threshold_output, threshold_value, 255, THRESH_BINARY);
    findContours(threshold_output, contours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE, Point(0, 0));

    vector<vector<Point>> hull(contours.size());
    for (size_t i = 0; i < contours.size(); i++) {
        convexHull(Mat(contours[i]), hull[i], false);
    }

    Mat dst = Mat::zeros(threshold_output.size(), CV_8UC3);
    for (size_t i = 0; i < contours.size(); i++) {
        Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));
        drawContours(dst, hull, i, color, 1, LINE_8, hierarchy, 0, Point(0, 0));
        drawContours(dst, contours, i, color, 1, LINE_8, hierarchy, 0, Point(0, 0));
    }
    imshow(output_win, dst);
}
```



三十、给轮廓绘制外接矩形，圆，椭圆，旋转矩形

1.矩形和旋转矩形

cv::boundingRect(InputArray points)得到轮廓周围最小矩形左上交点坐标和右下角点坐标/长宽，绘制一个矩形

```
Rect rect;
rect = boundingRect(contours.at(i));
```

cv::minAreaRect(InputArray　points)得到一个旋转的矩形，返回旋转矩形

2.圆形和椭圆

cv::minEnclosingCircle(InputArray points, //得到最小区域圆形

Point2f& center, // 圆心位置

float& radius)// 圆的半径

cv::fitEllipse(InputArray points)得到最小椭圆

```
vector<RotatedRect> myellipse(contours.size());
myellipse[i] = fitEllipse(contours[i]);
```

例：

```
// 初始化数组大小
vector<vector<Point>> contours_poly(contours.size());
vector<Rect> boundRects(contours.size());
vector<Point2f> centers(contours.size());
vector<float> radius(contours.size());

vector<RotatedRect> minRects(contours.size());
vector<RotatedRect> minEllipses(contours.size());

// 得到最小矩形与圆形的坐标信息
for (size_t i = 0; i < contours.size(); i++) {
    // 从这些点得到闭合曲线， 3表示跟原来点的最大距离，true表示是闭合的
    // approxPolyDP(Mat(contours[i]), contours_poly[i], 3, true);
    // boundRects[i] = boundingRect(Mat(contours_poly[i]));
    // minEnclosingCircle(Mat(contours_poly[i]), centers[i], radius[i]);
    minRects[i] = minAreaRect(Mat(contours[i]));
    if (contours[i].size() > 5) {
        minEllipses[i] = fitEllipse(Mat(contours[i]));
    }
}

// 绘制
drawImg = Mat::zeros(threshold_output.size(), CV_8UC3);
for (size_t i = 0; i < contours.size(); i++) {
    Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));
    drawContours(drawImg, contours_poly, i, color, 1, LINE_8, vector<Vec4i>(), 0, Point(0, 0));

    // draw rectangle and circle
    // Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));
    // rectangle(drawImg, boundRects[i].tl(), boundRects[i].br(), color, 1, LINE_AA, 0);
    // circle(drawImg, centers[i], radius[i], color, 1, LINE_AA, 0);

    // draw rotate rectangle and ellipse
    ellipse(drawImg, minEllipses[i], color, 1, 8);
    Point2f rect_points[4];
    minRects[i].points(rect_points);
    for (int j = 0; j < 4; j++) {
        line(drawImg, rect_points[j], rect_points[(j + 1)%4], color, 1, 8, 0);
    }
}
```

三十一、图像距(轮廓面积,弧长，中心点)

1．概述：

(1) 几何矩

$M_{ji} = \sum_{x,y}(P(x,y)\cdot x^{j}\cdot y^{i})$其中(i + j)和等于几就叫做几阶距

(2) 中心距

$mu_{ji} = \sum_{x,y}(P(x,y)\cdot(x-\bar{x})^{j}\cdot(y-\bar{y})^{i})$其中x̄,ȳ表示它的中心质点。

(3) 中心归一化距

$$nu_{ji} = \frac{mu_{ji}}{m_{00}^{(i+j)/2+1}}.$$

2．计算矩

moments(// 计算矩，返回 0 阶到 3 阶所有的几何矩，2 阶到 3 阶所有的中心矩，2 阶到 3 阶所有的中心归一矩

InputArray array,//输入数据，findContours 找到的轮廓数据

bool binaryImage=false // 是否为二值图像

)

(1) array:输入数组，可以是光栅图像(单通道，8-bit 或浮点型二维　数组),或者是一个二维数组(1 X N 或 N X 1),二维数组类型为　Point 或 Point2f

(2) binaryImage:默认值是 false，如果为 true，则所有非零的像素 都会按值 1 对待，也就是说相当于对图像进行了二值化处理，　阈值为 1，此参数仅对图像有效。

生成的数据：

spatial moments
double m00
double m10
double m01
double m20
double m11
double m02
double m30
double m21
double m12
double m03

central moments
double mu20
double mu11
double mu02
double mu30
double mu21
double mu12
double mu03

central normalized moments
double nu20
double nu11
double nu02
double nu30
double nu21
double nu12
double nu03

图像中心质点 Center(x0, y0)

$$x_0 = \frac{m_{10}}{m_{00}} \quad y_0 = \frac{m_{01}}{m_{00}}$$

- 通过计算得出的数据，计算该轮廓中心质点：

```
vector<Moments>mu(contours.size()); //创建矩数组
vector<Point2f> ccs(contours.size()); //创建点集
mu[i] = moments(contours[i]); //求得每个轮廓的矩给 mu[i]
ccs[i]=Point(static_cast<float>(mu[i].m10/mu[i].m00),static_cast<
```

float>(mu[i].m01/mu[i].m00)); //计算所有轮廓中心质点给 ccs[i]

3. 计算轮廓面积

contourArea(

InputArray contour,//输入轮廓数据

bool oriented// 默认 false、返回绝对值)

4. 计算轮廓弧长

arcLength(//用于计算封闭轮廓的周长或曲线的长度

InputArray curve,//输入曲线数据

bool closed// 是否是封闭曲线)

例：

```
Canny(gray_src, canny_output, threshold_value, threshold_value * 2, 3, false);
findContours(canny_output, contours, hieracrhy, RETR_TREE, CHAIN_APPROX_SIMPLE, Point(0, 0));

// calculate center of each moments
vector<Point2f> objcenters(contours.size());
for (size_t i = 0; i < contours.size(); i++) {
    objcenters[i] = Point(static_cast<float>(mu[i].m10 / mu[i].m00), static_cast<float>(mu[i].m01 / mu[i].m00));
}

Mat drawImg = Mat::zeros(canny_output.size(), CV_8UC3);
for (size_t i = 0; i < contours.size(); i++) {
    Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));
    drawContours(drawImg, contours, i, color, 1, 8, hieracrhy);
    circle(drawImg, objcenters[i], 3, color, 2, 8, 0);
}

// calculate area and arc length
for (size_t i = 0; i < contours.size(); i++) {
    printf("[[Moments %d]] area : %.2f  arclength : %.2f \n", i, contourArea(contours[i], false), arcLength(contours[i], true));
    Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));
    drawContours(drawImg, contours, i, color, 1, 8, hieracrhy);
    circle(drawImg, objcenters[i], 3, color, 2, 8, 0);
}
```

三十二、点多边形测试(一个点与轮廓相对位置)

pointPolygonTest(

InputArray　contour,// 输入的轮廓

Point2f　pt, // 测试点

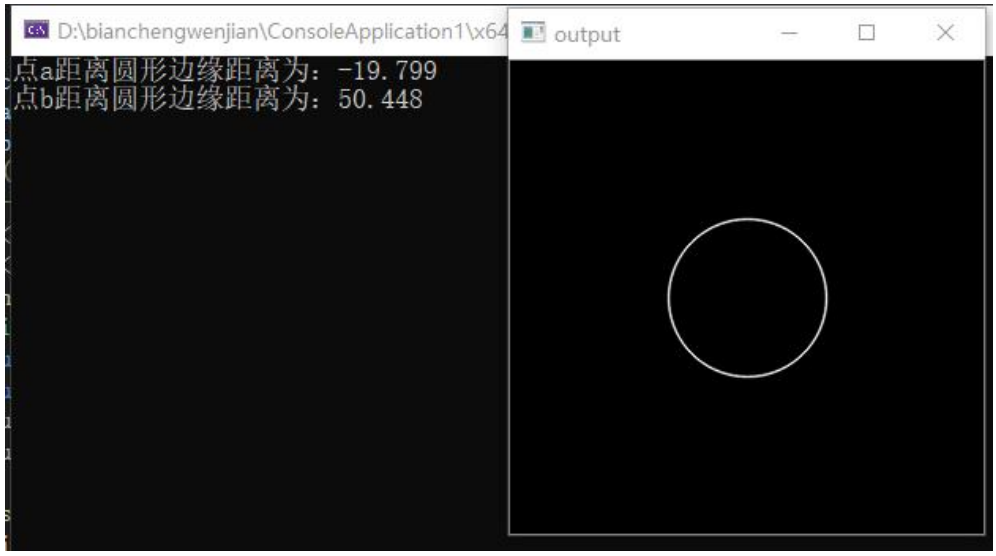
bool　measureDist // 是否返回距离值，如果是 false，1 表示在内面，0 表示在边界上，-1 表示在外部，true 返回实际距离

)

返回数据是 double 类型

例：

```
Mat src(Size(300,300),CV_8UC1,Scalar(0));
Point a(100, 100);
Point b(150, 150);
circle(src, Point(150, 150),50,Scalar(255),1,LINE_AA);
//定义一个圆心在b上，半径为50的圆
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(src,contours,hierarchy,RETR_EXTERNAL,CHAIN_APPROX_SIMPLE);
for (size_t i = 0; i < contours.size(); i++) {
    double dist1 = pointPolygonTest(contours.at(i), a, true);
    double dist2 = pointPolygonTest(contours.at(i), b, true);
    cout << "点a距离圆形边缘距离为：" << dist1 << endl;
    cout << "点b距离圆形边缘距离为：" << dist2 << endl;
}
cv::imshow("output", src);
cv::waitKey(0);
return 0;
```



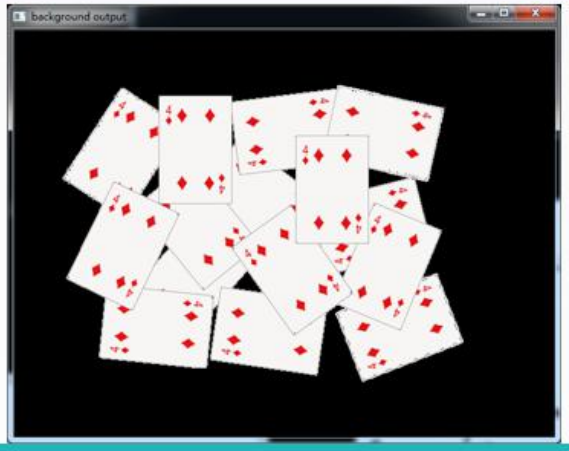
三十三、基于距离变换与分水岭的图像分割

图像分割(Image Segmentation)是图像处理最重要的处理手段之一

图像分割的目标是将图像中像素根据一定的规则分为若干(N)个 cluster 集合，每个集合包含一类像素。

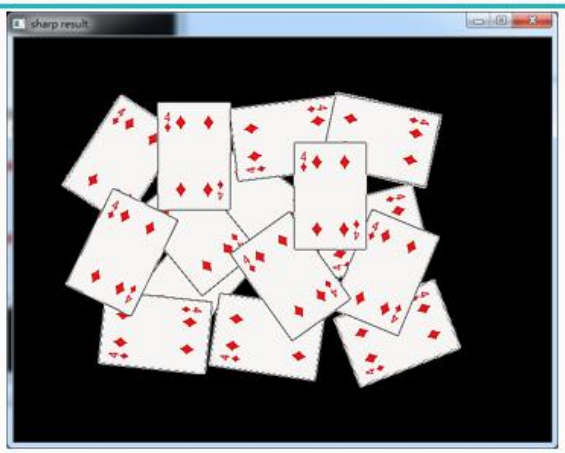
演示代码-去背景

```
// change the background
for (int row = 0; row < src.rows; row++) {
    for (int col = 0; col < src.cols; col++) {
        if (src.at<Vec3b>(row, col) == Vec3b(255, 255, 255)) {
            src.at<Vec3b>(row, col)[0] = 0;
            src.at<Vec3b>(row, col)[1] = 0;
            src.at<Vec3b>(row, col)[2] = 0;
        }
    }
}
namedWindow("background output", CV_WINDOW_AUTOSIZE);
imshow("background output", src);
```



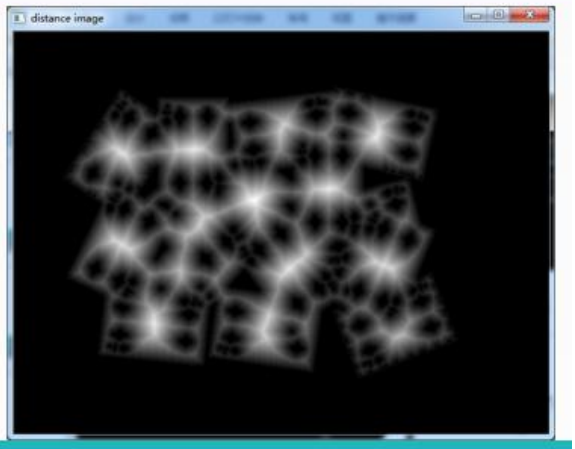
演示代码-Sharp

```
// create kernel
Mat kernel = (Mat_<float>(3, 3) << 1, 1, 1,
    1, -8, 1,
    1, 1, 1);

// make it more sharp
Mat imgLaplance;
Mat sharp = src;
filter2D(sharp, imgLaplance, CV_32F, kernel, Point(-1, -1), 0);
src.convertTo(sharp, CV_32F);
Mat imgResult = sharp - imgLaplance;
// 显示
imgResult.convertTo(imgResult, CV_8UC3);
imgLaplance.convertTo(imgLaplance, CV_8UC3);
namedWindow("sharp result", CV_WINDOW_AUTOSIZE);
imshow("sharp result", imgResult);
```



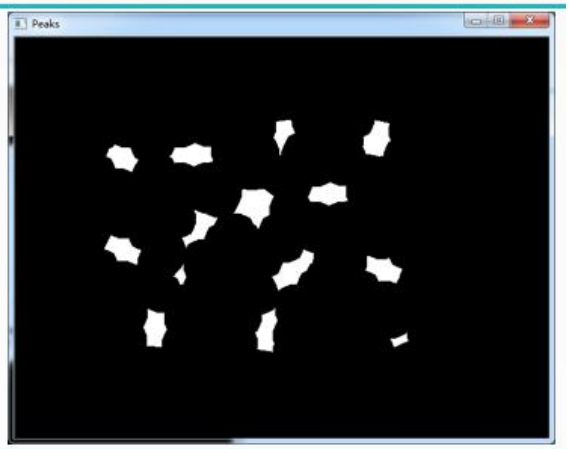
演示代码-二值距离变换

```
// 二值图像
Mat binImg;
cvtColor(src, binImg, CV_BGR2GRAY);
threshold(binImg, binImg, 40, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);
imshow("Binary Image", binImg);

// distance transformation
Mat dist;
distanceTransform(binImg, dist, CV_DIST_L2, 3);
normalize(dist, dist, 0, 1., NORM_MINMAX);
imshow("distance image", dist);

// try to get marker
threshold(dist, dist, .4, 1., CV_THRESH_BINARY);
```



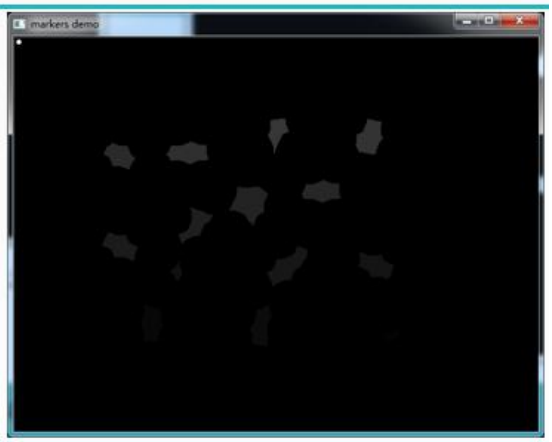
演示代码-二值腐蚀

```
// try to get marker
threshold(dist, dist, .4, 1., CV_THRESH_BINARY);

// erode the distance image
Mat kernel1 = Mat::ones(13, 13, CV_8UC1);
erode(dist, dist, kernel1);
imshow("Peaks", dist);
```



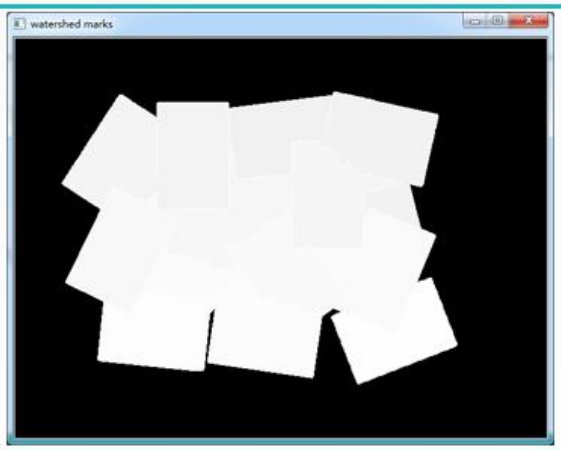
演示代码-标记

```
Mat dist_8u;
dist.convertTo(dist_8u, CV_8U);
// find contours
vector<vector<Point>> contours;
findContours(dist_8u, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE, Point(0, 0));

// create markers image
Mat markers = Mat::zeros(dist.size(), CV_32SC1);
// draw markers
for (size_t i = 0; i < contours.size(); i++) {
    drawContours(markers, contours, static_cast<int>(i), Scalar::all(static_cast<int>(i) + 1),
}
// draw background circle
circle(markers, Point(5, 5), 3, CV_RGB(255, 255, 255), -1);
imshow("markers demo", markers*1000);
```



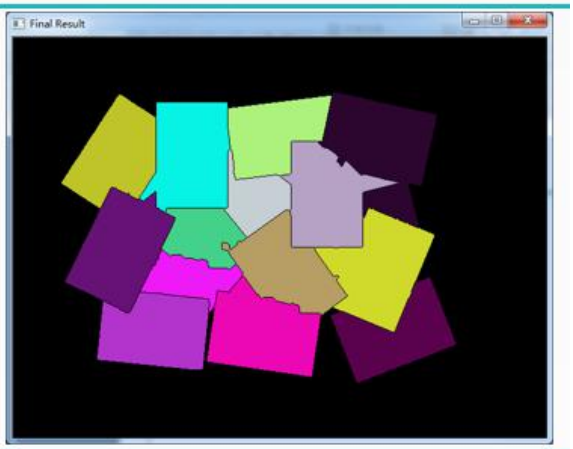
演示代码-分水岭变换

```
// perform watershed transform
watershed(src, markers);

Mat mark = Mat::zeros(markers.size(), CV_8UC1);
markers.convertTo(mark, CV_8UC1);
bitwise_not(mark, mark);
imshow("watershed marks", mark);
```



演示代码-着色效果

```
// fill with color and display final result
Mat dst = Mat::zeros(markers.size(), CV_8UC3);
for (int row = 0; row < markers.rows; row++) {
    for (int col = 0; col < markers.cols; col++) {
        int index = markers.at<int>(row, col);
        if (index > 0 && index <= static_cast<int>(contours.size())) {
            dst.at<Vec3b>(row, col) = colors[index-1];
        }
        else {
            dst.at<Vec3b>(row, col) = Vec3b(0, 0, 0);
        }
    }
}
imshow("Final Result", dst);
```



第二章 特征检测