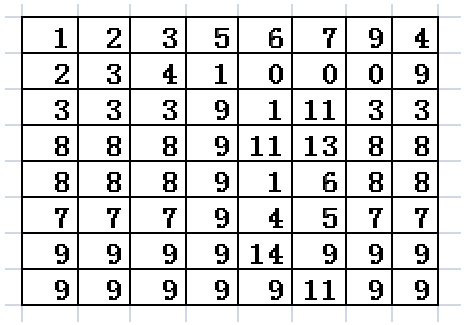
24.直方图计算

- 直方图概念
- API学习
- 代码演示

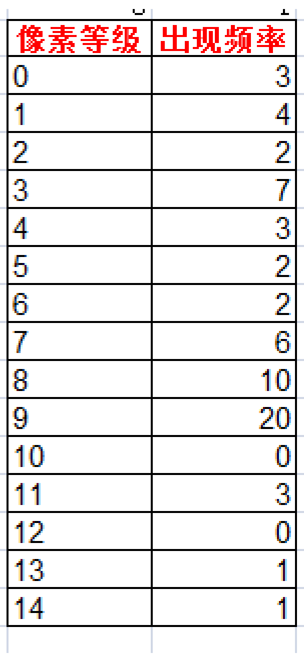
为梦想增值！

直方图概念

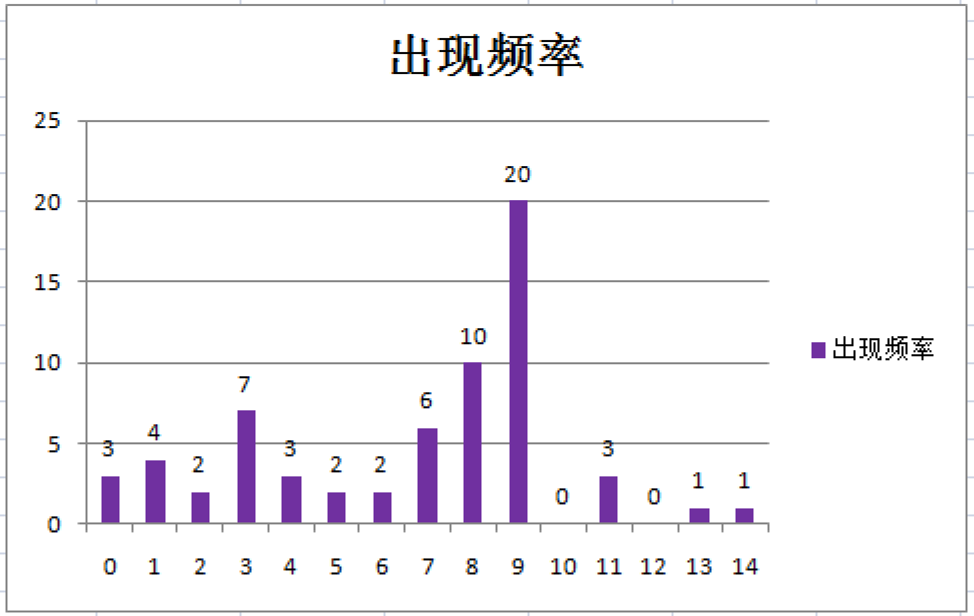


1	2	3	5	6	7	9	4
2	3	4	1	0	0	0	9
3	3	3	9	1	11	3	3
8	8	8	9	11	13	8	8
8	8	8	9	1	6	8	8
7	7	7	9	4	5	7	7
9	9	9	9	14	9	9	9
9	9	9	9	9	11	9	9

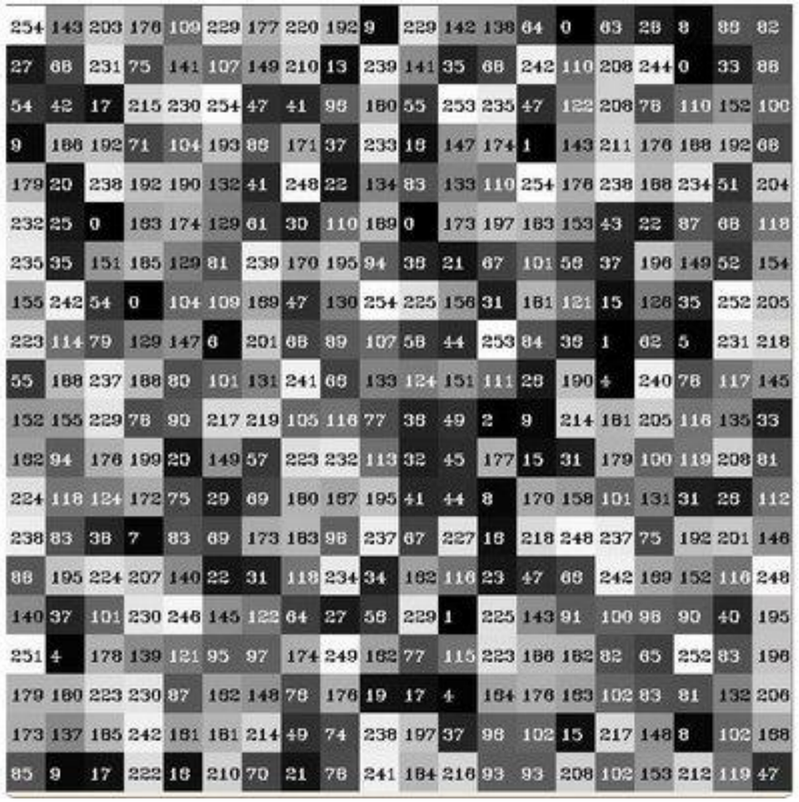
假设有图像数据8x8，像素值范围0~14共15个灰度等级，统计得到各个等级出现次数及直方图如右侧所示，每个紫色的长条叫BIN



像素等级	出现频率
0	3
1	4
2	2
3	7
4	3
5	2
6	2
7	6
8	10
9	20
10	0
11	3
12	0
13	1
14	1

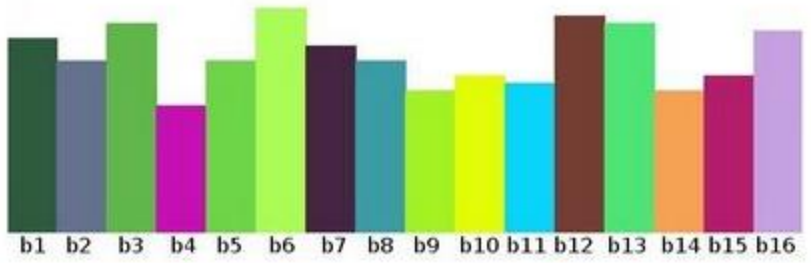


直方图概念



254	143	203	176	109	229	177	220	192	9	229	142	138	64	0	63	28	8	88	82
27	68	231	75	141	107	149	210	13	239	141	35	68	242	110	208	244	0	33	88
54	42	17	215	230	254	47	41	98	180	55	253	235	47	122	208	78	110	152	100
9	186	192	71	104	193	88	171	37	233	18	147	174	1	143	211	176	188	192	68
179	20	238	192	190	132	41	248	22	134	83	133	110	254	176	238	188	234	51	204
232	25	0	183	174	129	61	30	110	189	0	173	197	183	153	43	22	87	68	118
235	35	151	185	129	81	239	170	195	94	38	21	67	101	58	37	196	149	52	154
155	242	54	0	104	109	169	47	130	254	225	156	31	161	121	15	126	35	252	205
223	114	79	129	147	6	201	68	89	107	58	44	253	84	38	1	62	5	231	218
55	188	237	188	80	101	131	241	68	133	124	151	111	28	190	4	240	78	117	145
152	155	229	78	90	217	219	105	116	77	38	49	2	9	214	161	205	116	135	33
182	94	176	199	20	149	57	223	232	113	32	45	177	15	31	179	100	119	208	81
224	118	124	172	75	29	69	180	187	195	41	44	8	170	158	101	131	31	28	112
238	83	38	7	83	69	173	183	98	237	67	227	18	218	248	237	75	192	201	146
88	195	224	207	140	22	31	118	234	34	182	116	23	47	68	242	169	152	116	248
140	37	101	230	246	145	122	64	27	58	229	1	225	143	91	100	98	90	40	195
251	4	178	139	121	95	97	174	249	182	77	115	223	186	182	82	65	252	83	196
179	180	223	230	87	182	148	78	176	19	17	4	184	176	183	102	83	81	132	206
173	137	185	242	181	181	214	49	74	238	197	37	98	102	15	217	148	8	102	168
85	9	17	222	18	210	70	21	78	241	184	216	93	93	208	102	153	212	119	47

$$[0,255] = [0,15] \cup [16,31] \cup \ldots \cup [240,255]$$
$$range = bin_1 \cup bin_2 \cup \ldots \cup bin_{n=15}$$



为梦想增值!

直方图概念

- 上述直方图概念是基于图像像素值，其实对图像梯度、每个像素的角度、等一切图像的属性值，我们都可以建立直方图。这个才是直方图的概念真正意义，不过是基于图像像素灰度直方图是最常见的。
- 直方图最常见的几个属性：

- dims 表示维度，对灰度图像来说只有一个通道值dims=1
- bins 表示在维度中子区域大小划分，bins=256，划分为256个级别
- range 表示值得范围，灰度值范围为[0~255]之间

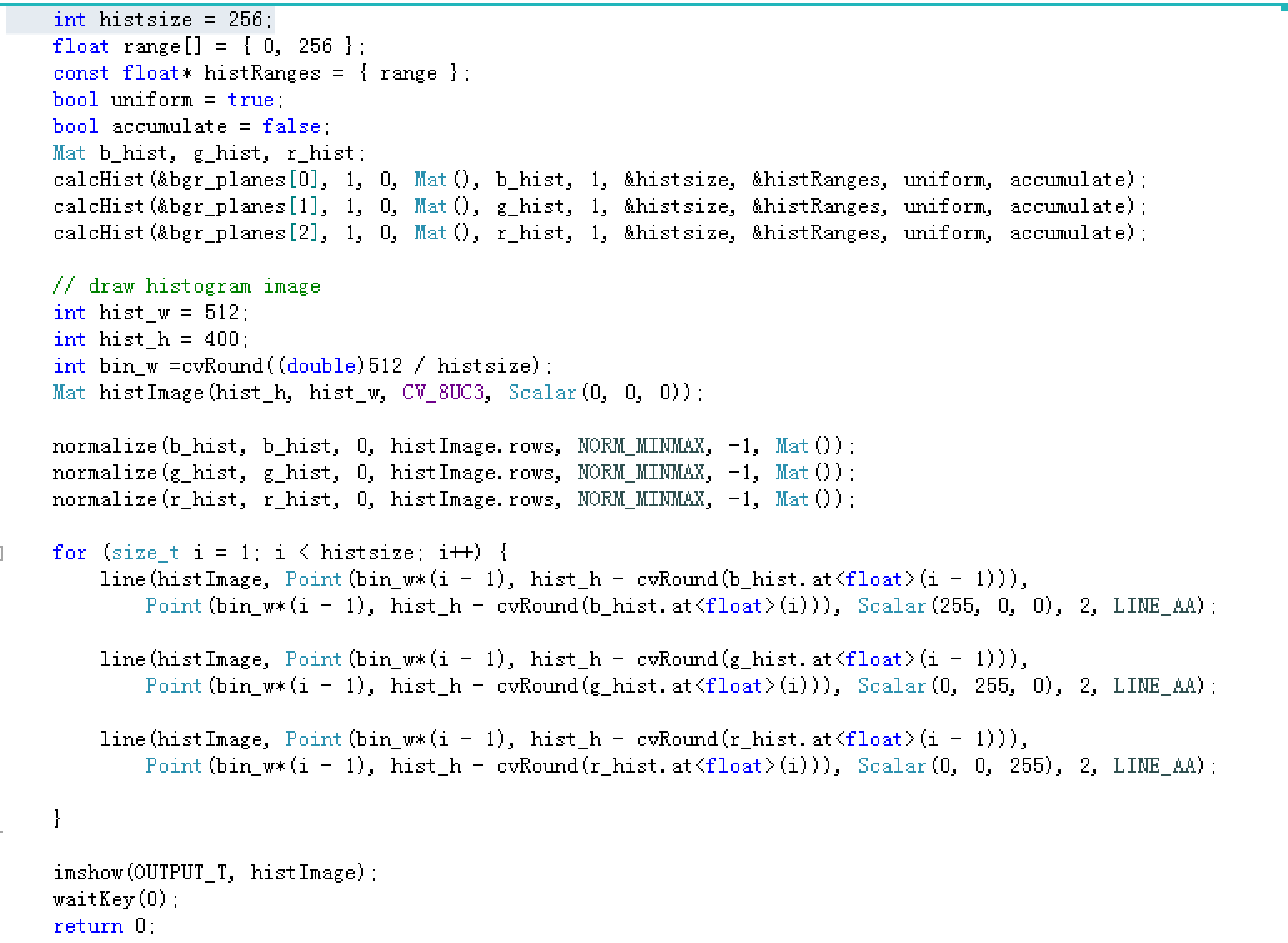
为梦想增值！

API学习

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

calcHist(
 const Mat* images,//输入图像指针
int images,// 图像数目
const int* channels,// 通道数
InputArray mask,// 输入mask，可选，不用
OutputArray hist,//输出的直方图数据
int dims,// 维数
const int* histsize,// 直方图级数
const float* ranges,// 值域范围
bool uniform,// true by default
bool accumulate// false by defaut
)

演示代码



```
int histsize = 256;
float range[] = { 0, 256 };
const float* histRanges = { range };
bool uniform = true;
bool accumulate = false;
Mat b_hist, g_hist, r_hist;
calcHist(&bgr_planes[0], 1, 0, Mat(), b_hist, 1, &histsize, &histRanges, uniform, accumulate);
calcHist(&bgr_planes[1], 1, 0, Mat(), g_hist, 1, &histsize, &histRanges, uniform, accumulate);
calcHist(&bgr_planes[2], 1, 0, Mat(), r_hist, 1, &histsize, &histRanges, uniform, accumulate);

// draw histogram image
int hist_w = 512;
int hist_h = 400;
int bin_w =cvRound((double)512 / histsize);
Mat histImage(hist_h, hist_w, CV_8UC3, Scalar(0, 0, 0));

normalize(b_hist, b_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());
normalize(g_hist, g_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());
normalize(r_hist, r_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat());

for (size_t i = 1; i < histsize; i++) {
    line(histImage, Point(bin_w*(i - 1), hist_h - cvRound(b_hist.at<float>(i - 1))),
        Point(bin_w*(i - 1), hist_h - cvRound(b_hist.at<float>(i))), Scalar(255, 0, 0), 2, LINE_AA);

    line(histImage, Point(bin_w*(i - 1), hist_h - cvRound(g_hist.at<float>(i - 1))),
        Point(bin_w*(i - 1), hist_h - cvRound(g_hist.at<float>(i))), Scalar(0, 255, 0), 2, LINE_AA);

    line(histImage, Point(bin_w*(i - 1), hist_h - cvRound(r_hist.at<float>(i - 1))),
        Point(bin_w*(i - 1), hist_h - cvRound(r_hist.at<float>(i))), Scalar(0, 0, 255), 2, LINE_AA);

}

imshow(OUTPUT_T, histImage);
waitKey(0);
return 0;
```

为梦想增值！

25.直方图比较

- 直方图比较方法
- 相关API
- 代码演示

为梦想增值！

直方图比较方法-概述

对输入的两张图像计算得到直方图H1与H2，归一化到相同的尺度空间然后可以通过计算H1与H2的之间的距离得到两个直方图的相似程度进而比较图像本身的相似程度。Opencv提供的比较方法有四种:

-Correlation 相关性比较
-Chi-Square 卡方比较
-Intersection 十字交叉性
-Bhattacharyya distance 巴氏距离

直方图比较方法-相关性计算 (CV_COMP_CORREL)

$$d(H_1,H_2)=\frac{\sum_I (H_1(I)-\bar{H}_1)(H_2(I)-\bar{H}_2)}{\sqrt{\sum_I (H_1(I)-\bar{H}_1)^2 \sum_I (H_2(I)-\bar{H}_2)^2}}$$

其中

$$\bar{H}_k=\frac{1}{N}\sum_J H_k(J)$$

其中N是直方图的BIN个数，

$\overline{H}$是均值

直方图比较方法-卡方计算 (CV_COMP_CHISQR)

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

H1,H2分别表示两个图像的直方图数据

为梦想增值！

直方图比较方法-十字计算 (CV_COMP_INTERSECT)

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

H1,H2分别表示两个图像的直方图数据

直方图比较方法-巴氏距离计算(CV_COMP_BHATTACHARYYA)

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H}_1 \bar{H}_2 N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}}$$

$\overline{H}$是均值

H1,H2分别表示两个图像的直方图数据

为梦想增值！

相关API

- 首先把图像从RGB色彩空间转换到HSV色彩空间cvtColor
- 计算图像的直方图，然后归一化到[0~1]之间calcHist和normalize;
- 使用上述四种比较方法之一进行比较compareHist

相关API cv::compareHist

```
compareHist(
InputArray h1, // 直方图数据， 下同
InputArray H2,
int method// 比较方法， 上述四种方法之一
)
```

演示代码

加载图像数据

```
Mat base, test1, test2;
base = imread(“D:/vcprojects/images/lena.jpg”);
if (!base.data) {
    printf(“could not load image...\n”);
    return -1;
}
test1 = imread(“D:/vcprojects/images/lena.png”);
test2 = imread(“D:/vcprojects/images/lenanoise.png”);
```

从RGB空间转换到HSV空间

```
cvtColor(base, base, CV_BGR2HSV);
cvtColor(test1, test1, CV_BGR2HSV);
cvtColor(test2, test2, CV_BGR2HSV);
```

计算直方图并归一化

```
int h_bins = 50; int s_bins = 60;
int histSize[] = { h_bins, s_bins };
// hue varies from 0 to 179, saturation from 0 to 255
float h_ranges[] = { 0, 180 };
float s_ranges[] = { 0, 256 };
const float* ranges[] = { h_ranges, s_ranges };
// Use the o-th and 1-st channels
int channels[] = { 0, 1 };
MatND hist_base;
MatND hist_test1;
MatND hist_test2;

calcHist(&base, 1,  channels, Mat(), hist_base, 2, histSize, ranges, true, false);
normalize(hist_base, hist_base, 0, 1, NORM_MINMAX, -1, Mat());

calcHist(&test1, 1, channels, Mat(), hist_test1, 2, histSize, ranges, true, false);
normalize(hist_test1, hist_test1, 0, 1, NORM_MINMAX, -1, Mat());

calcHist(&test2, 1, channels, Mat(), hist_test2, 2, histSize, ranges, true, false);
normalize(hist_test2, hist_test2, 0, 1, NORM_MINMAX, -1, Mat());
```

比较直方图，并返回值

```
double basebase = compareHist(hist_base, hist_base, CV_COMP_CORREL);
double basetest1 = compareHist(hist_base, hist_test1, CV_COMP_CORREL);
double basetest2 = compareHist(hist_base, hist_test2, CV_COMP_CORREL);
double tesltest2 = compareHist(hist_test1, hist_test2, CV_COMP_CORREL);
printf(“test1 compare with test2 correlation value :%f”, tesltest2);
```

为梦想增值!

26.直方图反向投影(Back Projection)

- 反向投影
- 实现步骤与相关API
- 代码演示

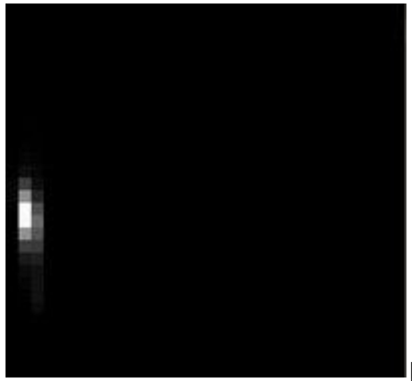
为梦想增值！

反向投影

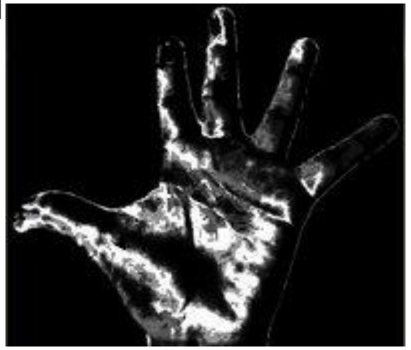
- 反向投影是反映直方图模型在目标图像中的分布情况
- 简单点说就是用直方图模型去目标图像中寻找是否有相似的对象。通常用HSV色彩空间的HS两个通道直方图模型

反向投影

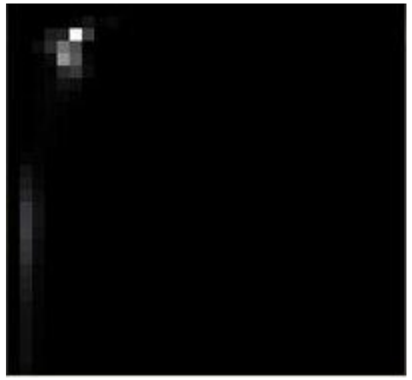
直方图模型建立



反向投影结果



待检测图像



为梦想增值!

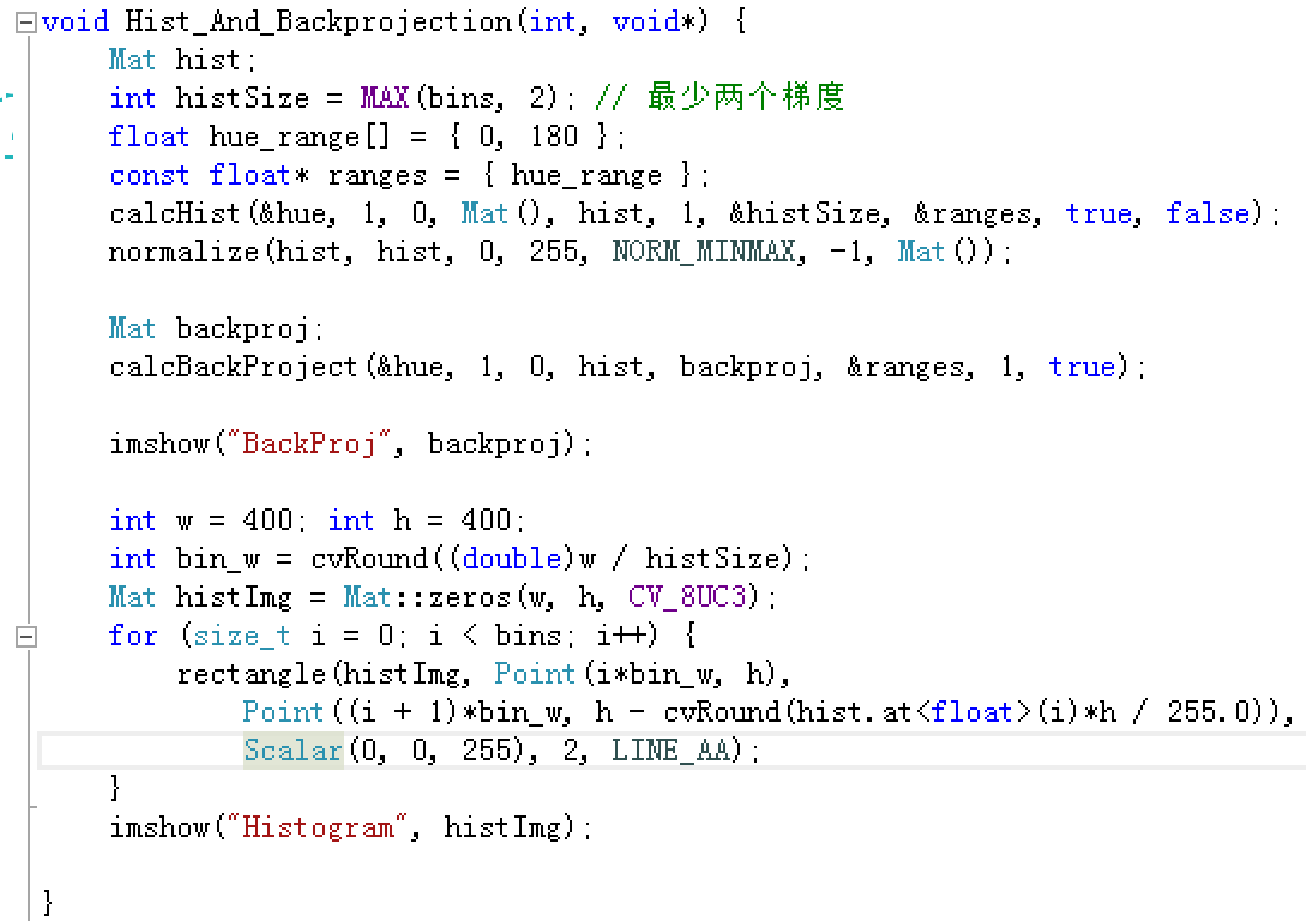
反向投影 – 步骤

1.建立直方图模型
2.计算待测图像直方图并映射到模型中
3.从模型反向计算生成图像

实现步骤与相关API

- 加载图片imread
- 将图像从RGB色彩空间转换到HSV色彩空间cvtColor
- 计算直方图和归一化calcHist与normalize
- Mat与MatND其中Mat表示二维数组，MatND表示三维或者多维数据，此处均可以用Mat表示。
- 计算反向投影图像 - calcBackProject

演示代



```
void Hist_And_Backprojection(int, void*) {
    Mat hist;
    int histSize = MAX(bins, 2); // 最少两个梯度
    float hue_range[] = { 0, 180 };
    const float* ranges = { hue_range };
    calcHist(&hue, 1, 0, Mat(), hist, 1, &histSize, &ranges, true, false);
    normalize(hist, hist, 0, 255, NORM_MINMAX, -1, Mat());

    Mat backproj;
    calcBackProject(&hue, 1, 0, hist, backproj, &ranges, 1, true);

    imshow("BackProj", backproj);

    int w = 400; int h = 400;
    int bin_w = cvRound((double)w / histSize);
    Mat histImg = Mat::zeros(w, h, CV_8UC3);
    for (size_t i = 0; i < bins; i++) {
        rectangle(histImg, Point(i*bin_w, h),
            Point((i + 1)*bin_w, h - cvRound(hist.at<float>(i)*h / 255.0)),
            Scalar(0, 0, 255), 2, LINE_AA);
    }
    imshow("Histogram", histImg);

}
```

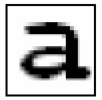
为梦想增值!

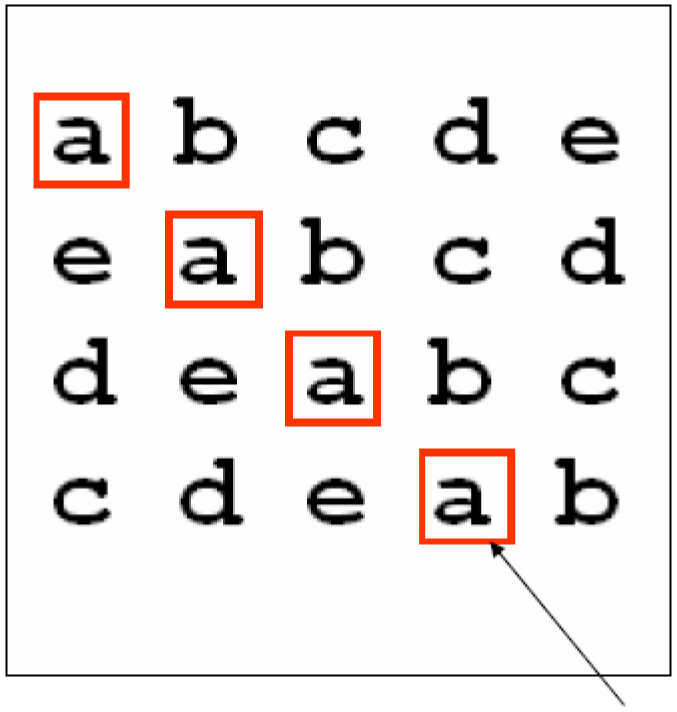
27.模板匹配(Template Match)

- 模板匹配介绍
- 相关API介绍
- 代码演示

为梦想增值！

模板匹配介绍


模板



为梦想增值!

模板匹配介绍

- 模板匹配就是在整个图像区域发现与给定子图像匹配的小块区域。
- 所以模板匹配首先需要一个模板图像T（给定的子图像）
- 另外需要一个待检测的图像-源图像S
- 工作方法，在带检测图像上，从左到右，从上向下计算模板图像与重叠子图像的匹配度，匹配程度越大，两者相同的可能性越大。

模板匹配介绍 – 匹配算法介绍

OpenCV中提供了六种常见的匹配算法如下:

1. 计算平方不同

$$R(x,y)=\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2$$

2. 计算相关性

3. 计算相关系数

$$R(x,y)=\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y'))$$

$$R(x,y)=\sum_{x',y'}(T'(x',y')\cdot I'(x+x',y+y'))$$

$$T'(x',y')=T(x',y')-1/(w\cdot h)\cdot\textstyle\sum_{x'',y''}T(x'',y'')$$
$$I'(x+x',y+y')=I(x+x',y+y')-1/(w\cdot h)\cdot\textstyle\sum_{x'',y''}I(x+x'',y+y'')$$

为梦想增值!

模板匹配介绍 – 匹配算法介绍

计算归一化平方不同

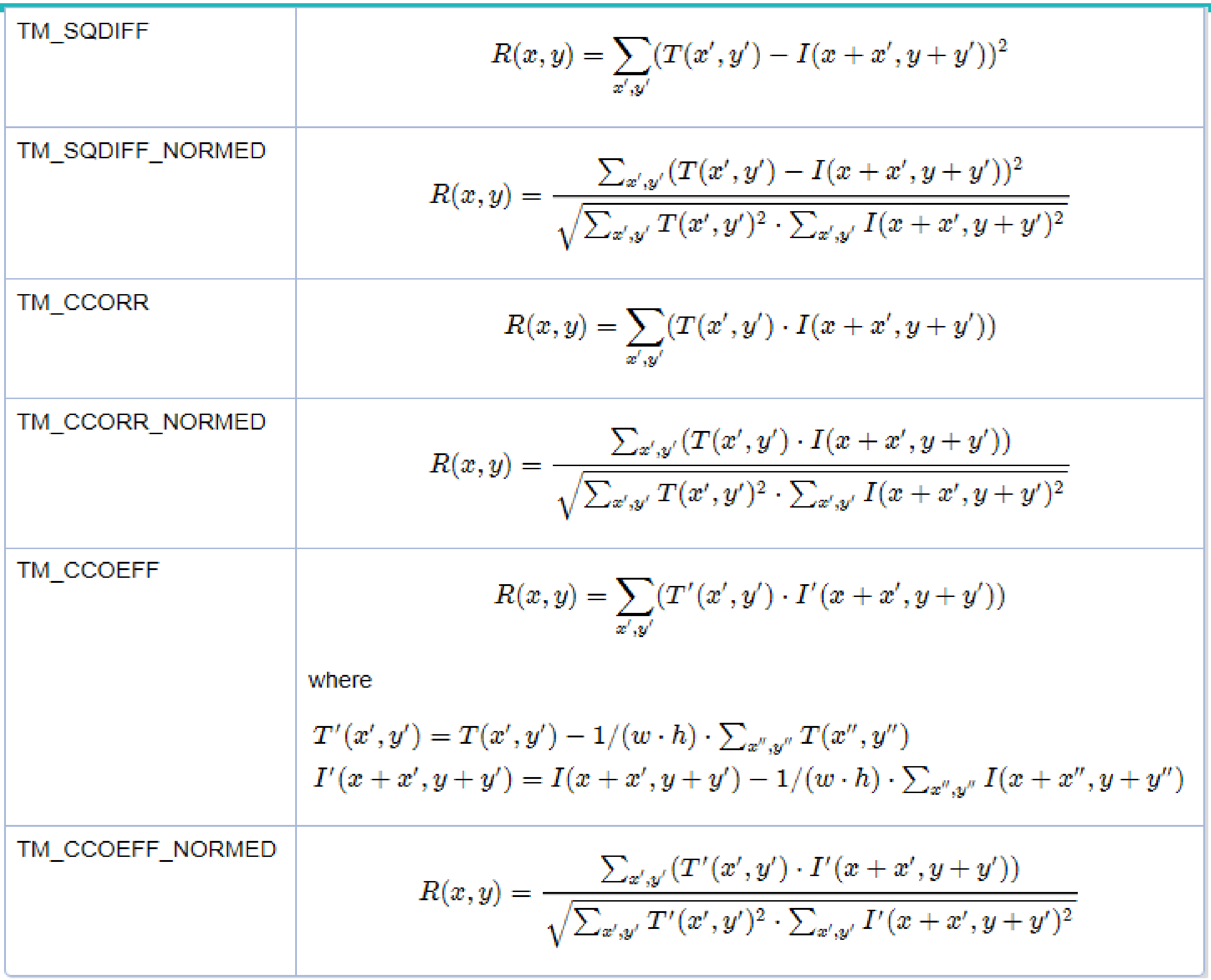
$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$$

[illegible]

$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$$

$$R(x,y)=\frac{\sum_{x',y'}(T'(x',y')\cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'}T'(x',y')^2\cdot\sum_{x',y'}I'(x+x',y+y')^2}}$$

为梦想增值!



TM_SQDIFF	$$R(x,y)=\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2$$
TM_SQDIFF_NORMED	$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$$
TM_CCORR	$$R(x,y)=\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y'))$$
TM_CCORR_NORMED	$$R(x,y)=\frac{\sum_{x',y'}(T(x',y')\cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$$
TM_CCOEFF	$$R(x,y)=\sum_{x',y'}(T'(x',y')\cdot I'(x+x',y+y'))$$ where $T'(x',y')=T(x',y')-1/(w\cdot h)\cdot\sum_{x'',y''}T(x'',y'')$ $I'(x+x',y+y')=I(x+x',y+y')-1/(w\cdot h)\cdot\sum_{x'',y''}I(x+x'',y+y'')$
TM_CCOEFF_NORMED	$$R(x,y)=\frac{\sum_{x',y'}(T'(x',y')\cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'}T'(x',y')^2\cdot\sum_{x',y'}I'(x+x',y+y')^2}}$$

为梦想增值!

相关API介绍cv::matchTemplate

matchTemplate(

InputArray image,// 源图像，必须是8-bit或者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,**

则结果必须为W-w+1, H-h+1的大小。

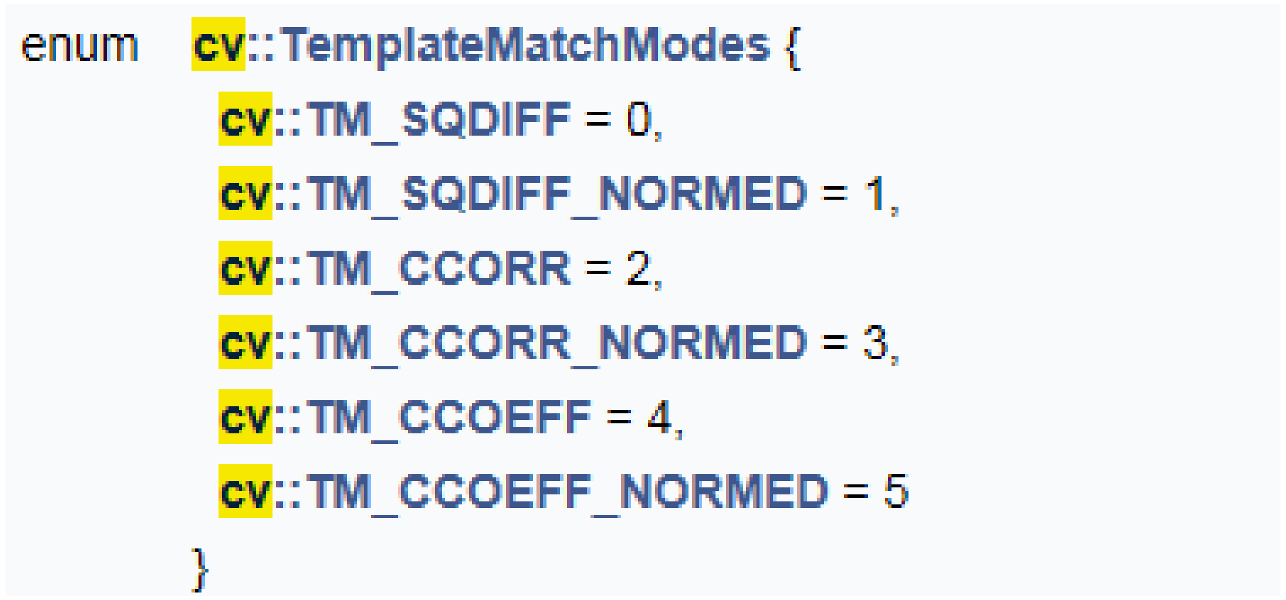
int method,//使用的匹配方法

InputArray mask=noArray()//(optional)

)

为梦想增值！

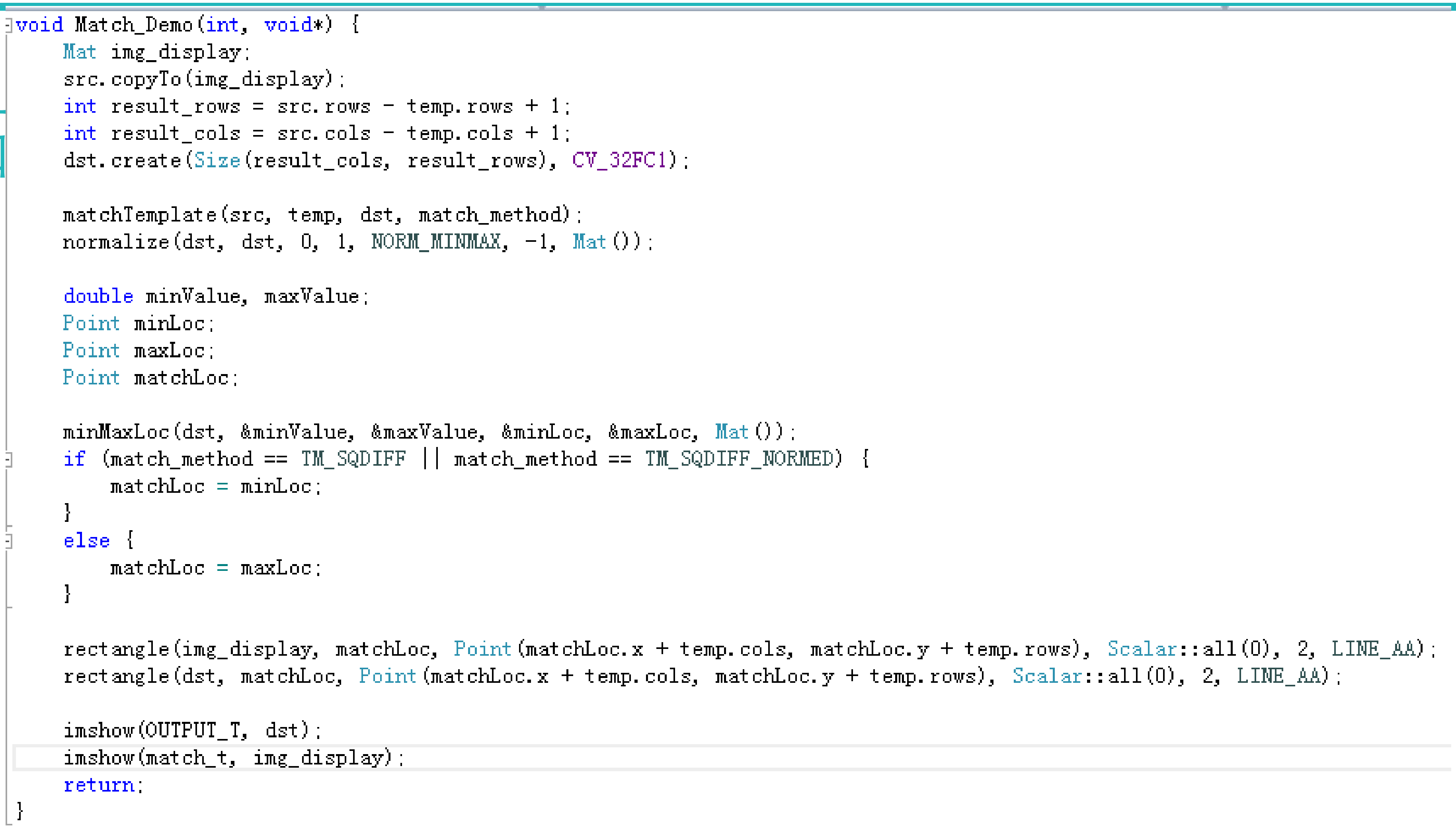
相关API介绍cv::matchTemplate



```
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
        }
```

为梦想增值！

演示代码



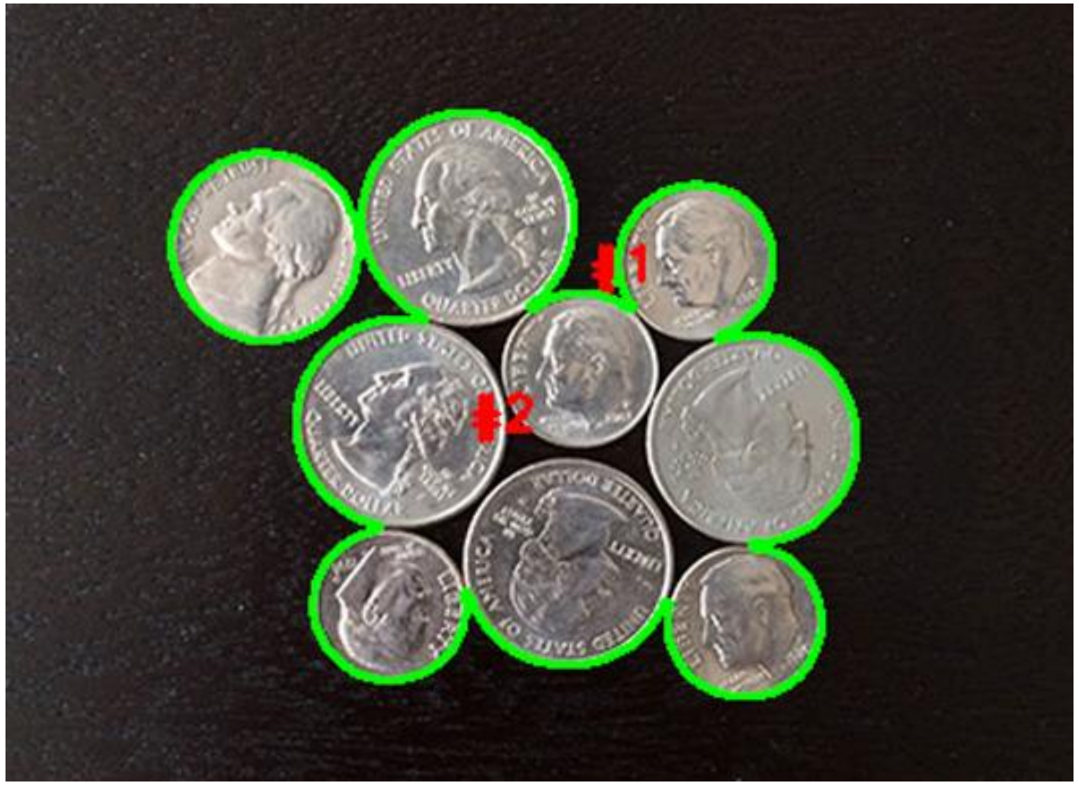
```
void Match_Demo(int, void*) {
    Mat img_display;
    src.copyTo(img_display);
    int result_rows = src.rows - temp.rows + 1;
    int result_cols = src.cols - temp.cols + 1;
    dst.create(Size(result_cols, result_rows), CV_32FC1);

    matchTemplate(src, temp, dst, match_method);
    normalize(dst, dst, 0, 1, NORM_MINMAX, -1, Mat());

    double minValue, maxValue;
    Point minLoc;
    Point maxLoc;
    Point matchLoc;

    minMaxLoc(dst, &minValue, &maxValue, &minLoc, &maxLoc, Mat());
    if (match_method == TM_SQDIFF || match_method == TM_SQDIFF_NORMED) {
        matchLoc = minLoc;
    }
    else {
        matchLoc = maxLoc;
    }

    rectangle(img_display, matchLoc, Point(matchLoc.x + temp.cols, matchLoc.y + temp.rows), Scalar::all(0), 2, LINE_AA);
    rectangle(dst, matchLoc, Point(matchLoc.x + temp.cols, matchLoc.y + temp.rows), Scalar::all(0), 2, LINE_AA);

    imshow(OUTPUT_T, dst);
    imshow(match_t, img_display);
    return;
}
```

为梦想增值！

28.轮廓发现(find contour in your image)

- 轮廓发现(find contour)
- 代码演示

为梦想增值!


#1
#2

轮廓发现(find contour)

- **轮廓发现**是基于图像边缘提取的基础寻找对象轮廓的方法。所以边缘提取的阈值选定会影响最终轮廓发现结果
- **API介绍**
  - findContours发现轮廓
  - drawContours绘制轮廓

轮廓发现(find contour)

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
)

为梦想增值！

轮廓绘制(draw contour)

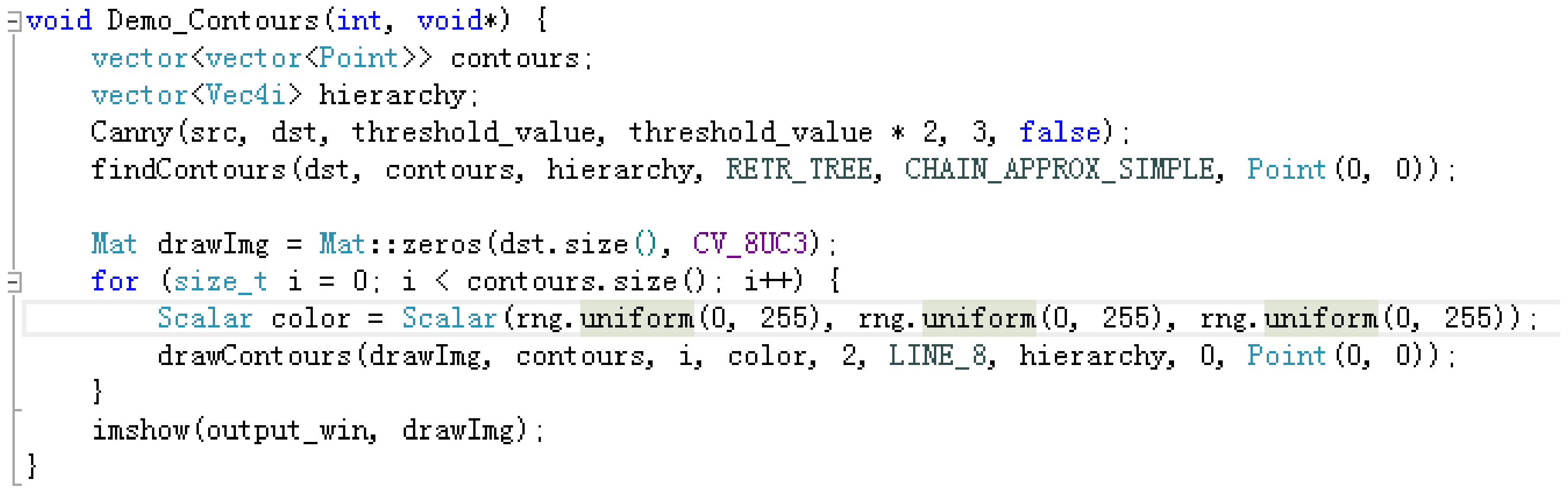
在二值图像上发现轮廓使用API cv::findContours之后对发现的轮廓数据进行绘制显示

```
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
```

演示代码

- 输入图像转为灰度图像cvtColor
- 使用Canny进行边缘提取，得到二值图像
- 使用findContours寻找轮廓
- 使用drawContours绘制轮廓

演示代码



```
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);
}
```

为梦想增值！

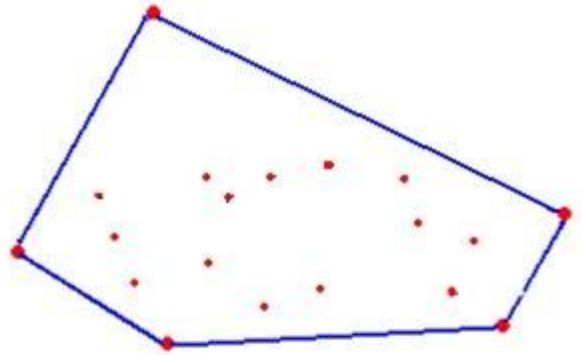
29.凸包-Convex Hull

- 概念介绍
- API说明
- 代码演示

为梦想增值！

概念介绍

- **什么是凸包(Convex Hull)，**在一个多变形边缘或者内部任意两个点的连线都包含在多边形边界或者内部。

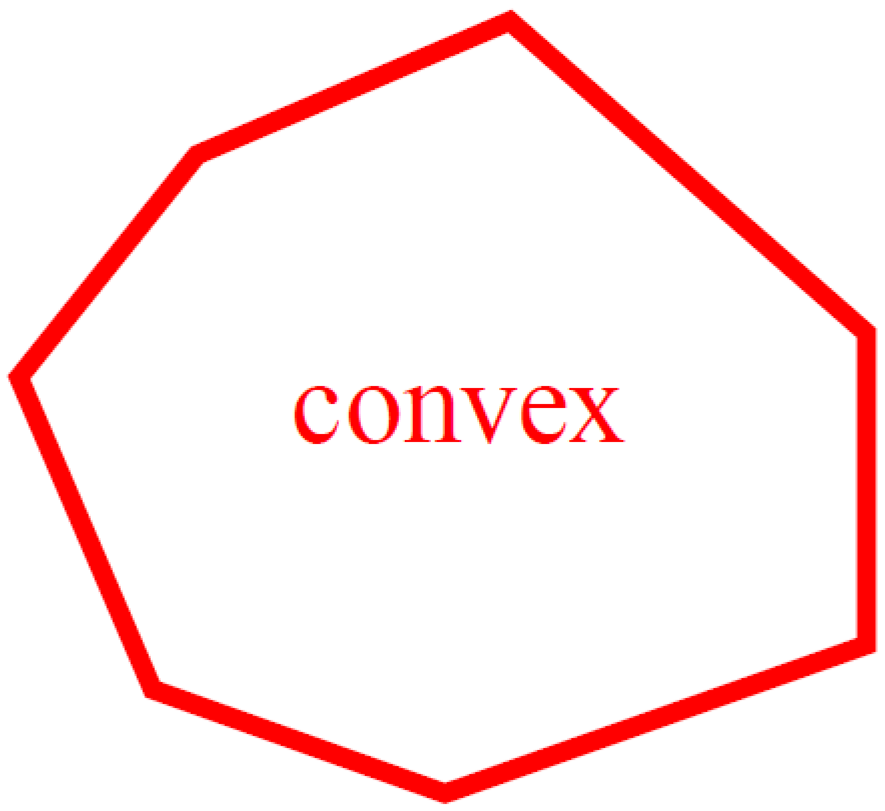


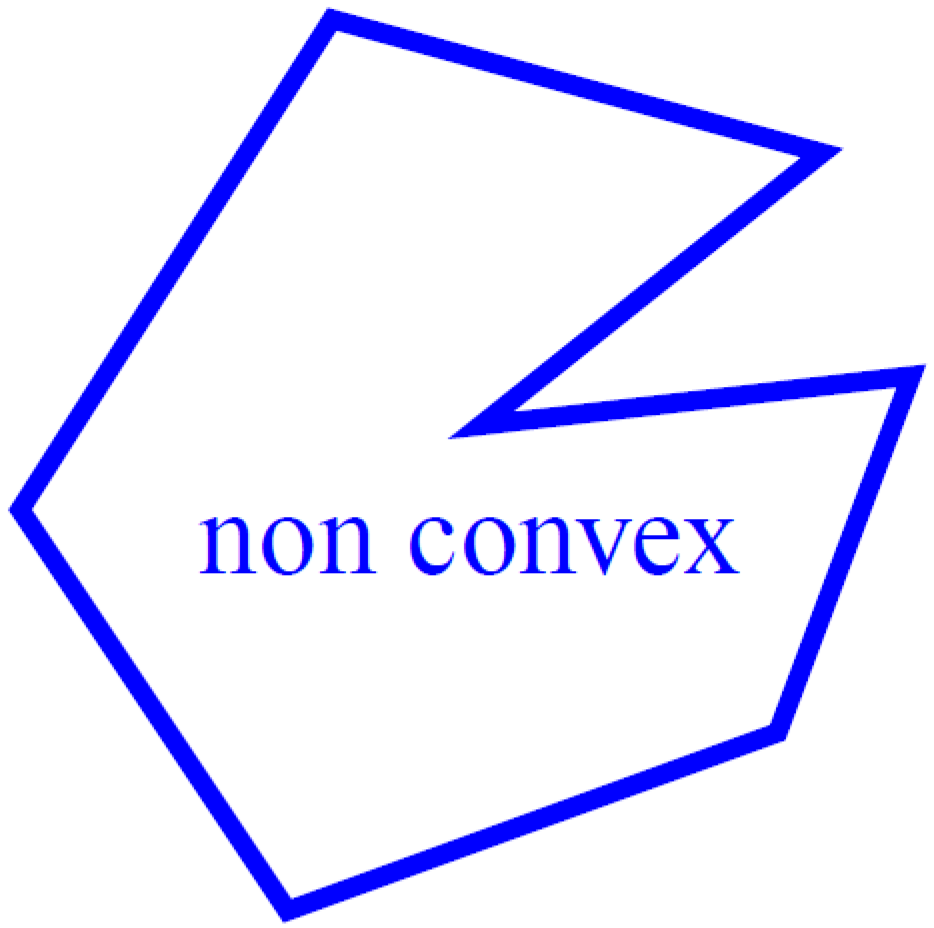
正式定义：
包含点集合S中所有点的最小凸多边形称为凸包

- **检测算法**

- Graham扫描法

为梦想增值！


convex

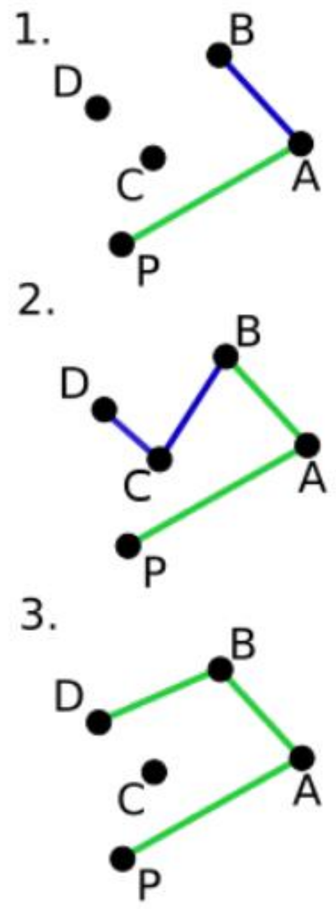

non convex

概念介绍-Graham扫描算法

- 首先选择Y方向最低的点作为起始点p0
- 从p0开始极坐标扫描，依次添加p1….pn（排序顺序是根据极坐标的角度大小，逆时针方向）
- 对每个点pi来说，如果添加pi点到凸包中导致一个左转向（逆时针方法）则添加该点到凸包， 反之如果导致一个右转向（顺时针方向）删除该点从凸包中

概念介绍-Graham扫描算法

- No worry,我们只是需要了解，OpenCV已经实现了凸包发现算法和API提供我们使用。



API说明cv::convexHull

- convexHull(

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

OutputArray hull,// 凸包

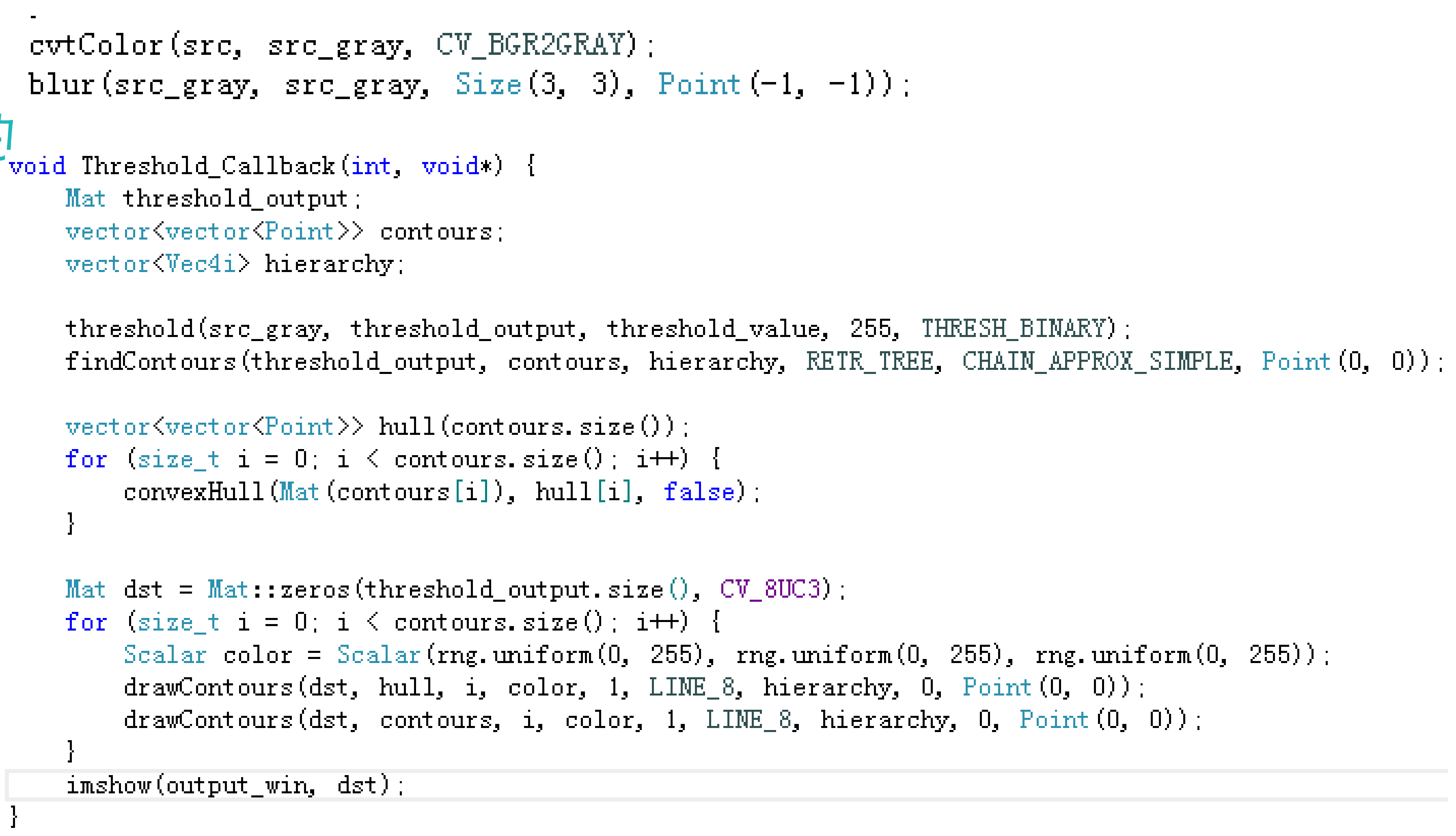
bool clockwise,// default true, 顺时针方向

bool returnPoints）// true 表示返回点个数，如果第二个参数是 vector<Point>则自动忽略

代码演示

- 首先把图像从RGB转为灰度
- 然后再转为二值图像
- 在通过发现轮廓得到候选点
- 凸包API调用
- 绘制显示。

演示代码



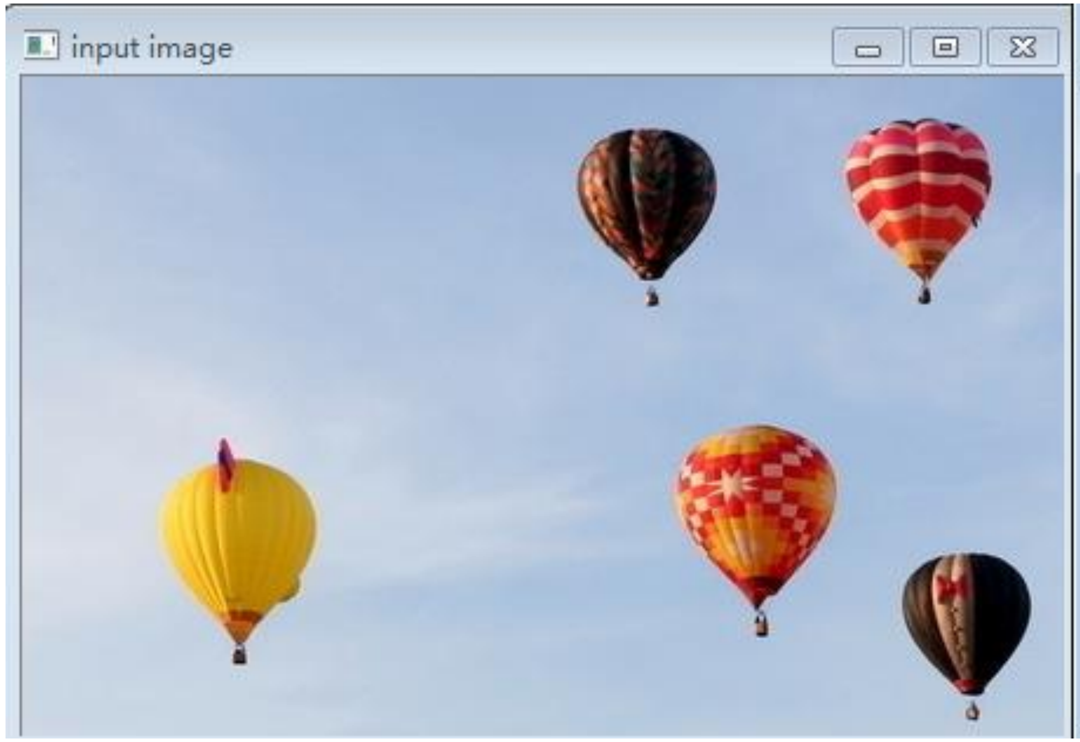
```
cvtColor(src, src_gray, CV_BGR2GRAY);
blur(src_gray, src_gray, Size(3, 3), Point(-1, -1));

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);
}
```

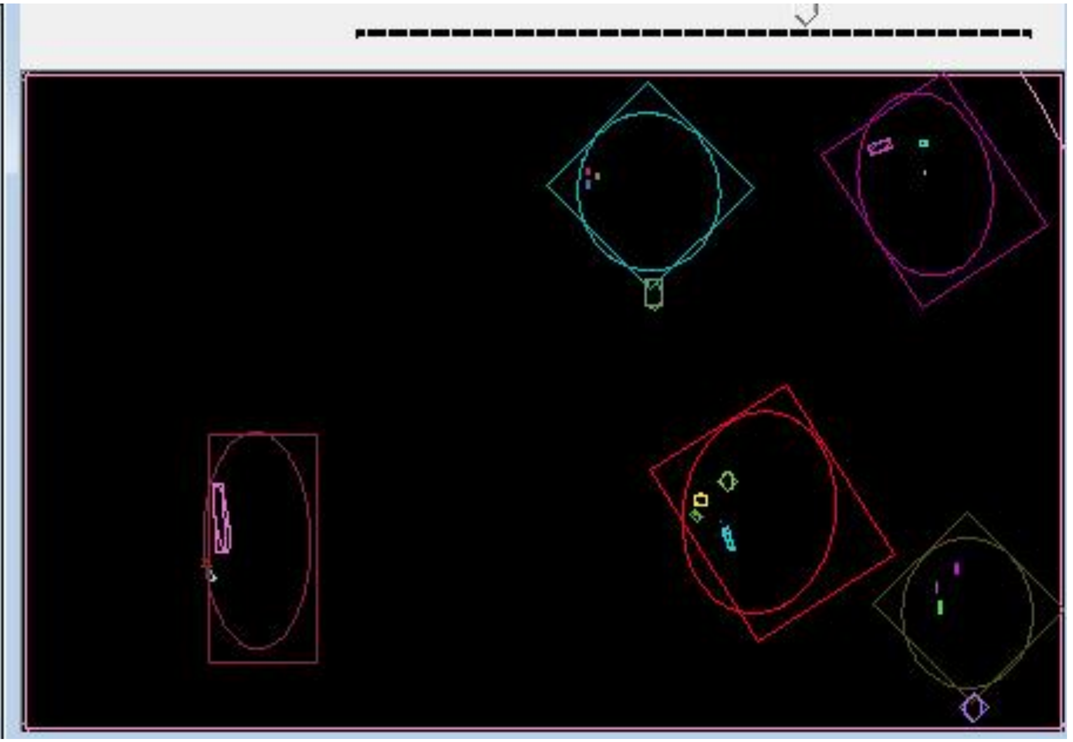
为梦想增值!

30.轮廓周围绘制矩形框和圆形框

- API介绍
- 代码演示

为梦想增值！



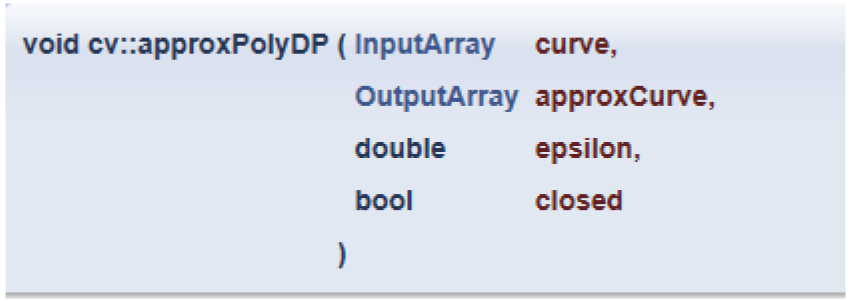


为梦想增值!

轮廓周围绘制矩形 -API

- approxPolyDP(InputArray curve, OutputArray approxCurve, double epsilon, bool closed)

基于RDP算法实现,目的是减少多边形轮廓点数



```
void cv::approxPolyDP ( InputArray   curve,
                        OutputArray  approxCurve,
                        double       epsilon,
                        bool         closed
                      )
```

为梦想增值!

轮廓周围绘制矩形-API

- 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，绘制一个矩形
- cv::minAreaRect(InputArray points)得到一个旋转的矩形，返回旋转矩形

轮廓周围绘制圆和椭圆-API

- 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

演示代码-步骤

- 首先将图像变为二值图像
- 发现轮廓，找到图像轮廓
- 通过相关API在轮廓点上找到最小包含矩形和圆，旋转矩形与椭圆。
- 绘制它们。

演示代码

```cpp
Mat threshold_output;
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;

threshold(gray_src, threshold_output, threshold_v, 255, THRESH_BINARY);
findContours(threshold_output, contours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE, Point(0, 0));
```

```cpp
// 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]));
    }
}
```

```cpp
// 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);
    }
}
```

为梦想增值！

31.图像矩(Image Moments)

- 矩的概念介绍
- API介绍与使用
- 代码演示

为梦想增值！

矩的概念介绍

- 几何矩
  - 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
  - 中心距 $mu_{ji} = \sum_{x,y}(P(x,y)\cdot (x-\bar{x})^j \cdot (y-\bar{y})^i)$其中$\bar{x}, \bar{y}$表示它的中心质点。
  - 中心归一化距 $\mathrm{nu}_{ji} = \frac{\mathrm{mu}_{ji}}{\mathrm{m}_{00}^{(i+j)/2+1}}.$

矩的概念介绍

- 图像中心Center(x0, y0)

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

API介绍与使用 – cv::moments 计算生成数据

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

API介绍与使用-计算矩cv::moments

moments(
InputArray array,//输入数据
bool binaryImage=false // 是否为二值图像
)

contourArea(
InputArray contour,//输入轮廓数据
bool oriented// 默认false、返回绝对值)

arcLength(
InputArray curve,//输入曲线数据
bool closed// 是否是封闭曲线)

演示代码-步骤

- 提取图像边缘
- 发现轮廓
- 计算每个轮廓对象的矩
- 计算每个对象的中心、弧长、面积

演示代码

```
cvtColor(src, gray_src, CV_BGR2GRAY);
GaussianBlur(gray_src, gray_src, Size(3, 3), 0, 0);
```

```
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);
}
```

为梦想增值!

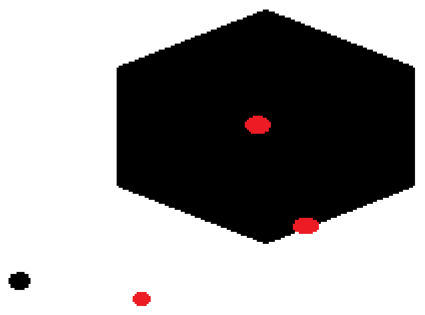
32.点多边形测试

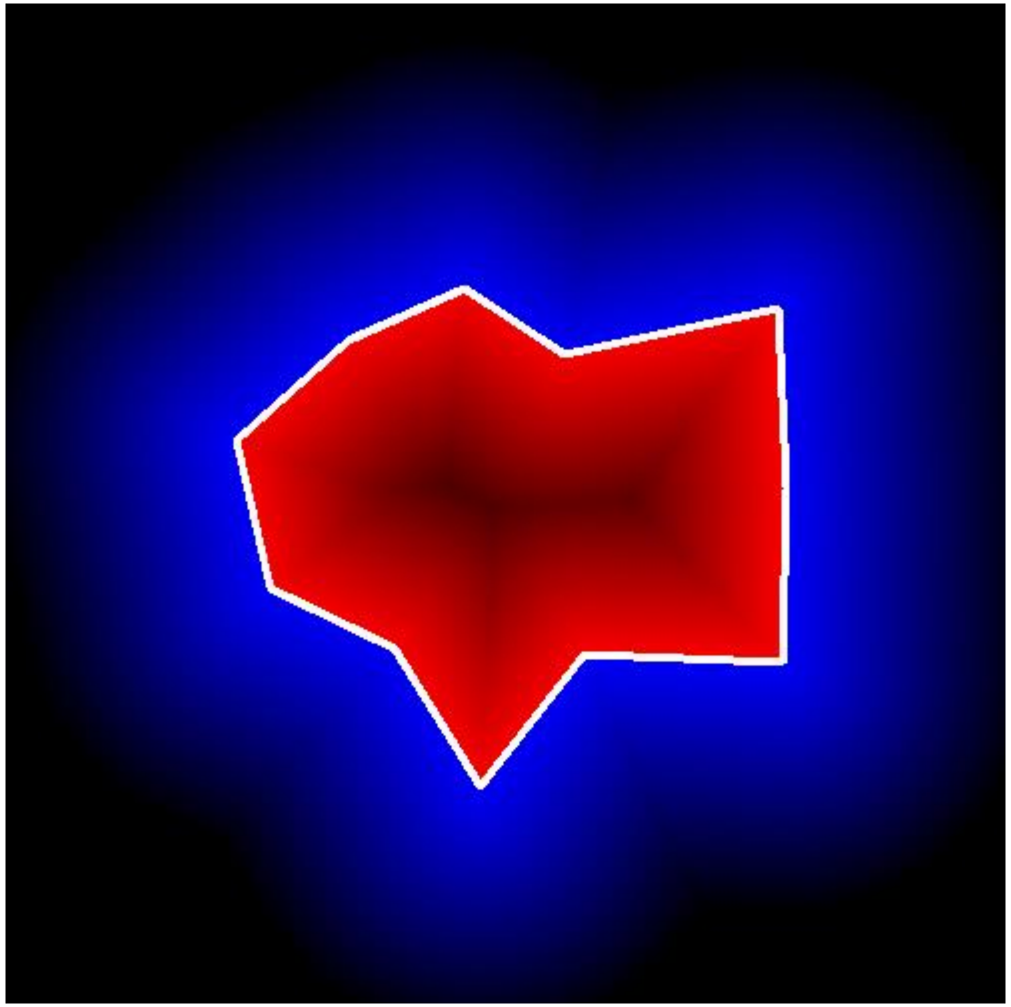
- 概念介绍
- API介绍
- 代码演示

为梦想增值！

概念介绍 - 点多边形测试

- 测试一个点是否在给定的多边形内部，边缘或者外部





为梦想增值！

API介绍 cv::pointPolygonTest

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

演示代码-步骤

- 构建一张400x400大小的图片， Mat::Zero(400, 400, CV_8UC1)
- 画上一个六边形的闭合区域line
- 发现轮廓
- 对图像中所有像素点做点 多边形测试， 得到距离， 归一化后显示。

演示代码-细节

```
double minValue, maxValue;
minMaxLoc(raw_dist, &minValue, &maxValue, 0, 0, Mat());
```

- 内部

```
drawImg.at<Vec3b>(row, col)[0] = (uchar)((abs(distance)/maxValue)*255);
```

- 外部

```
drawImg.at<Vec3b>(row, col)[1] = (uchar)(255*(abs(distance)/minValue));
```

- 边缘线

```
drawImg.at<Vec3b>(row, col)[0] = 255;
drawImg.at<Vec3b>(row, col)[1] = saturate_cast<uchar>(minValue);
drawImg.at<Vec3b>(row, col)[2] = saturate_cast<uchar>(minValue);
```

为梦想增值！

演示代码

```cpp
const int r = 100;
Mat src = Mat::zeros(r * 4, r * 4, CV_8UC1);

vector<Point2f> vert(6);
vert[0] = Point(3 * r / 2, static_cast<int>(1.34*r));
vert[1] = Point(1 * r, 2 * r);
vert[2] = Point(3 * r / 2, static_cast<int>(2.866*r));
vert[3] = Point(5 * r / 2, static_cast<int>(2.866*r));
vert[4] = Point(3 * r, 2 * r);
vert[5] = Point(5 * r / 2, static_cast<int>(1.34*r));

for (int i = 0; i < 6; i++) {
    line(src, vert[i], vert[(i + 1) % 6], Scalar(255), 3, 8, 0);
}
```

```cpp
vector<vector<Point>> contours;
vector<Vec4i> hieracrhy;

// full data copy by clone
Mat src_copy = src.clone();
// find contours
findContours(src_copy, contours, hieracrhy, RETR_TREE, CHAIN_APPROX_SIMPLE, Point(0, 0));

Mat raw_dist = Mat(src_copy.size(), CV_32FC1);
for (int row = 0; row < src_copy.rows; row++) {
    for (int col = 0; col < src_copy.cols; col++) {
        raw_dist.at<float>(row, col) = (float)pointPolygonTest(contours[0], Point2f((float)col, (float)row), true);
    }
}
```

为梦想增值!

演示代码

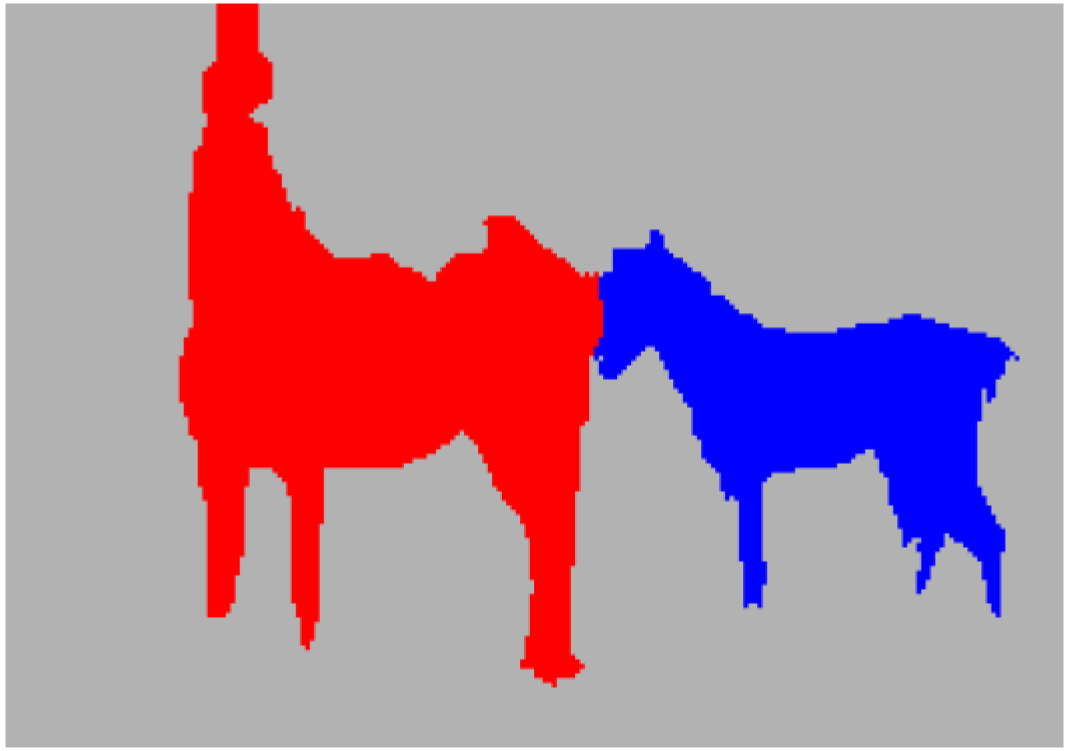
```
double minValue, maxValue;
minMaxLoc(raw_dist, &minValue, &maxValue, 0, 0, Mat());
minValue = abs(minValue);
maxValue = abs(maxValue);

Mat drawImg = Mat::zeros(raw_dist.size(), CV_8UC3);
for (int row = 0; row < src_copy.rows; row++) {
    for (int col = 0; col < src_copy.cols; col++) {
        float distance = raw_dist.at<float>(row, col);
        if (distance > 0) {
            drawImg.at<Vec3b>(row, col)[0] = (uchar)((abs(distance)/maxValue)*255);
        }
        else if (distance < 0) {
            drawImg.at<Vec3b>(row, col)[1] = (uchar)(255*(abs(distance)/minValue));
        } else {
            drawImg.at<Vec3b>(row, col)[0] = 255;
            drawImg.at<Vec3b>(row, col)[1] = saturate_cast<uchar>(minValue);
            drawImg.at<Vec3b>(row, col)[2] = saturate_cast<uchar>(minValue);
        }
    }
}
```

33.基于距离变换与分水岭的图像分割

- 什么是图像分割
- 距离变换与分水岭介绍
- 相关API
- 代码演示

为梦想增值！

什么是图像分割(Image Segmentation)

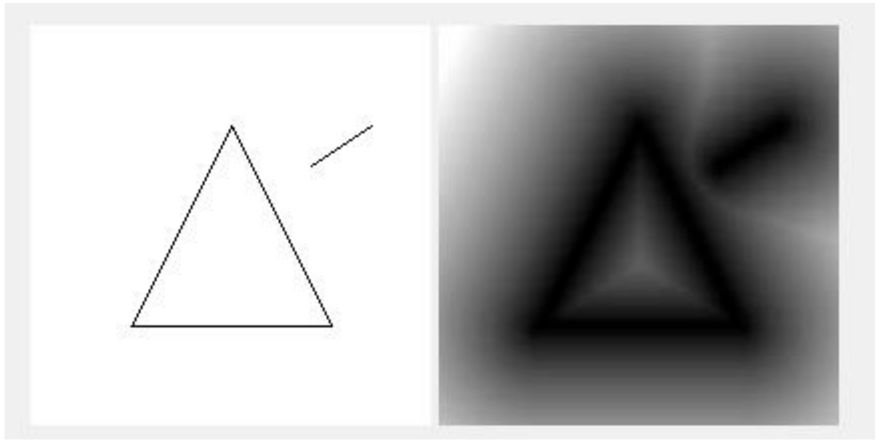


为梦想增值!

什么是图像分割

- 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。
- 根据算法分为监督学习方法和无监督学习方法，图像分割的算法多数都是无监督学习方法 - KMeans

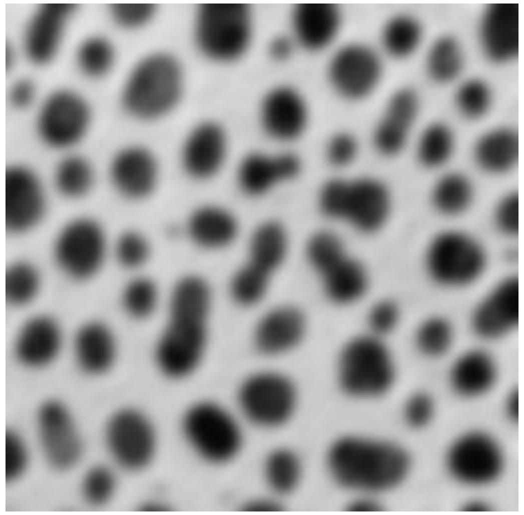
距离变换与分水岭介绍

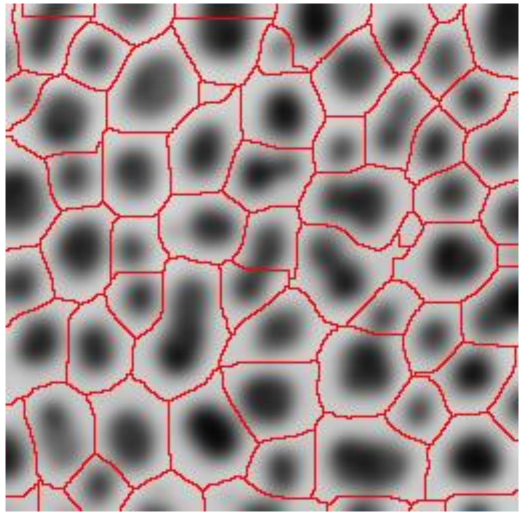


还记得上节课的内容，测试点多边形得到结果跟距离变换相似

为梦想增值！

距离变换与分水岭介绍





距离变换与分水岭介绍

- 距离变换常见算法有两种

  - 不断膨胀/ 腐蚀得到

  - 基于倒角距离

- 分水岭变换常见的算法

  - 基于浸泡理论实现

相关API

- cv::distanceTransform(InputArray src, OutputArray dst, OutputArray labels, int distanceType, int maskSize, int labelType=DIST_LABEL_CCOMP)

distanceType = DIST_L1/DIST_L2,

maskSize = 3x3,最新的支持5x5，推荐3x3、

labels离散维诺图输出

dst输出8位或者32位的浮点数，单一通道，大小与输入图像一致

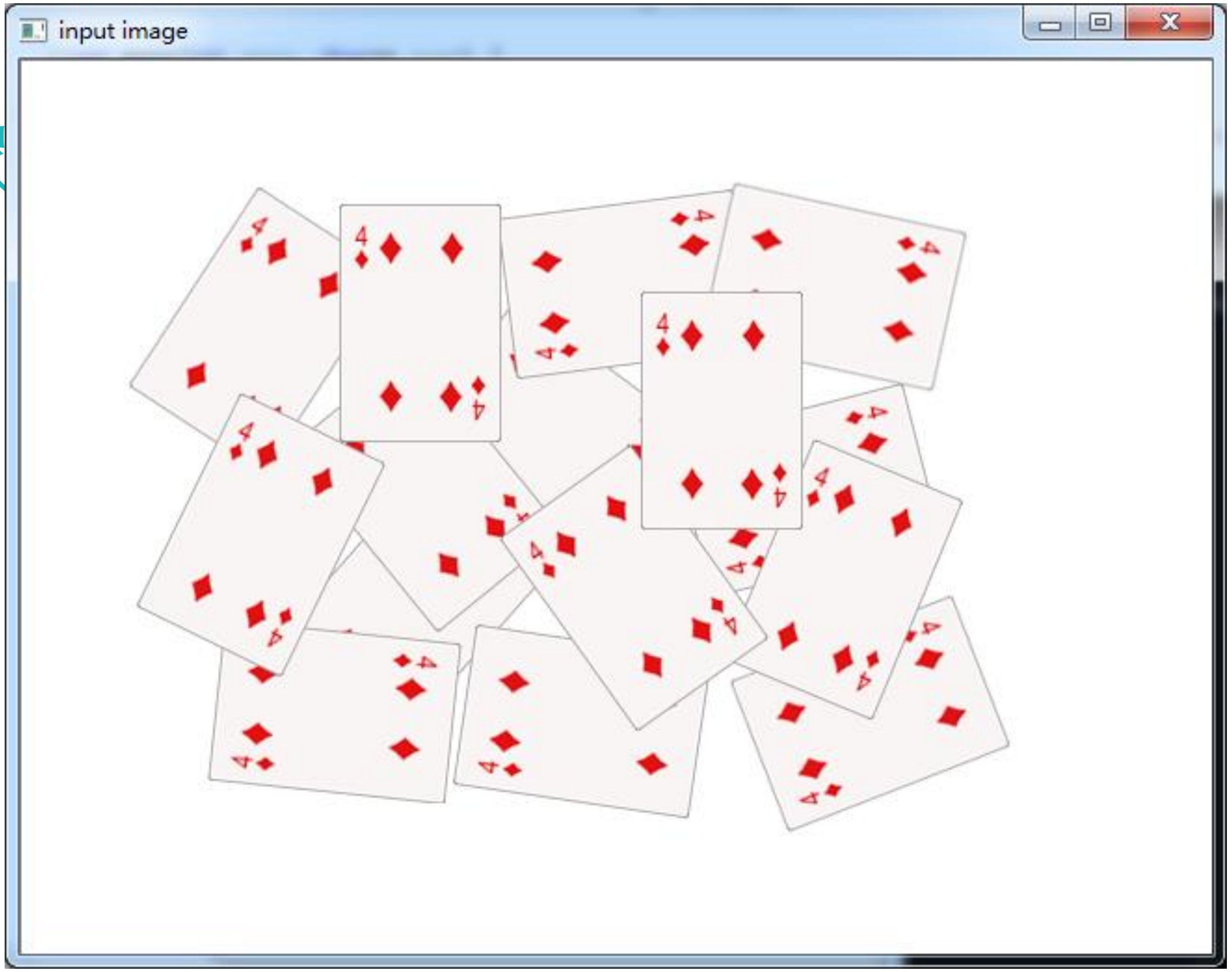
- cv::watershed(InputArray image, InputOutputArray markers)

处理流程

1.将白色背景变成黑色-目的是为后面的变换做准备
2. 使用filter2D与拉普拉斯算子实现图像对比度提高，sharp
3. 转为二值图像通过threshold
4. 距离变换
5. 对距离变换结果进行归一化到[0~1]之间
6. 使用阈值，再次二值化，得到标记
7. 腐蚀得到每个Peak - erode
8.发现轮廓 – findContours
9. 绘制轮廓- drawContours
10.分水岭变换 watershed
11. 对每个分割区域着色输出结果

演示代码-加载图像

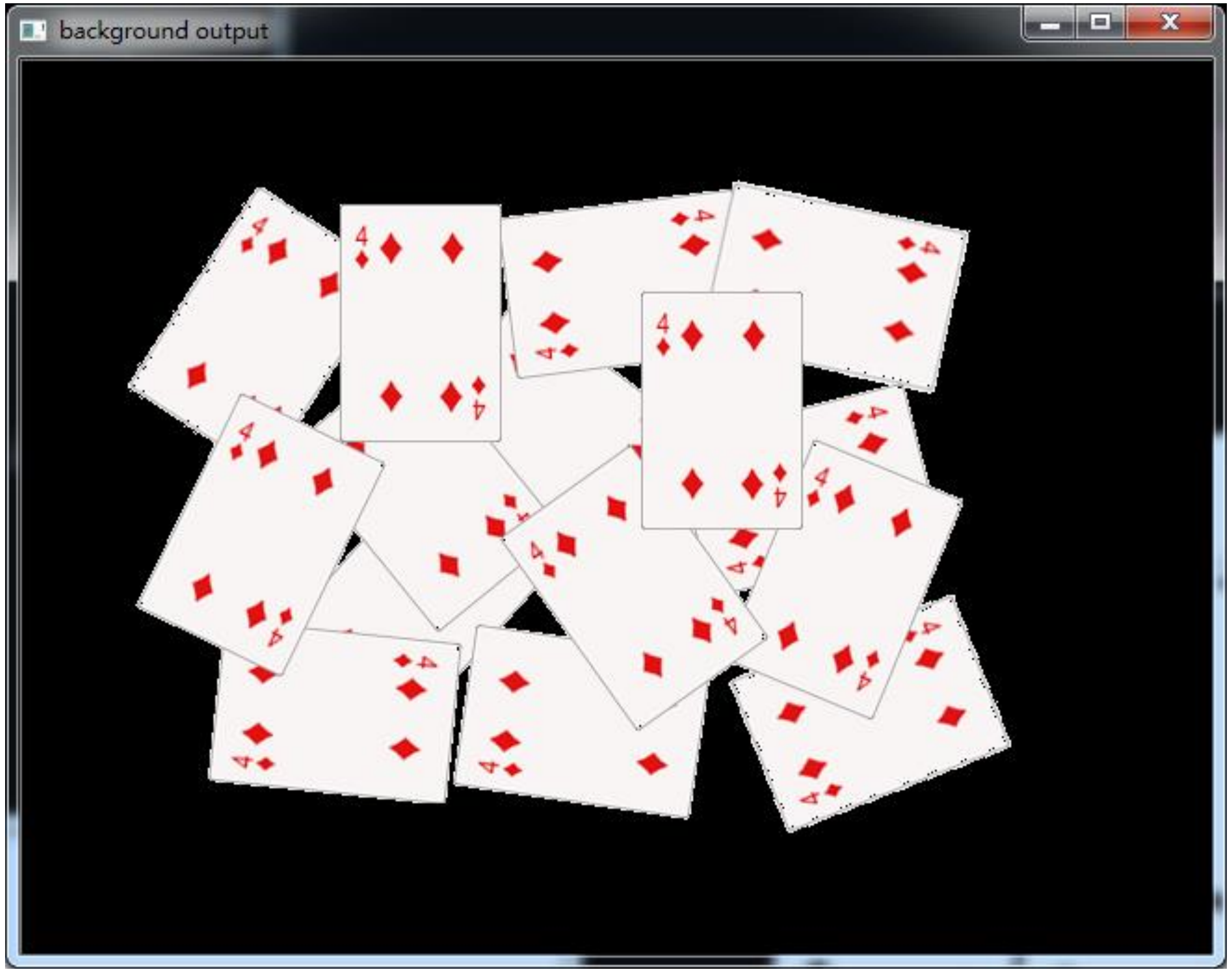
```
char input_win[] = "input image";
char watershed_win[] = "watershed segmentation demo";
Mat src = imread("D:/vcprojects/images/cards.png");
// Mat src = imread("D:/kuaidi.jpg");
if (src.empty()) {
    printf("could not load image...\n");
    return -1;
}
namedWindow(input_win, CV_WINDOW_AUTOSIZE);
//namedWindow(watershed_win, CV_WINDOW_AUTOSIZE);
imshow(input_win, src);
```



为梦想增值!

演示代码-去背景

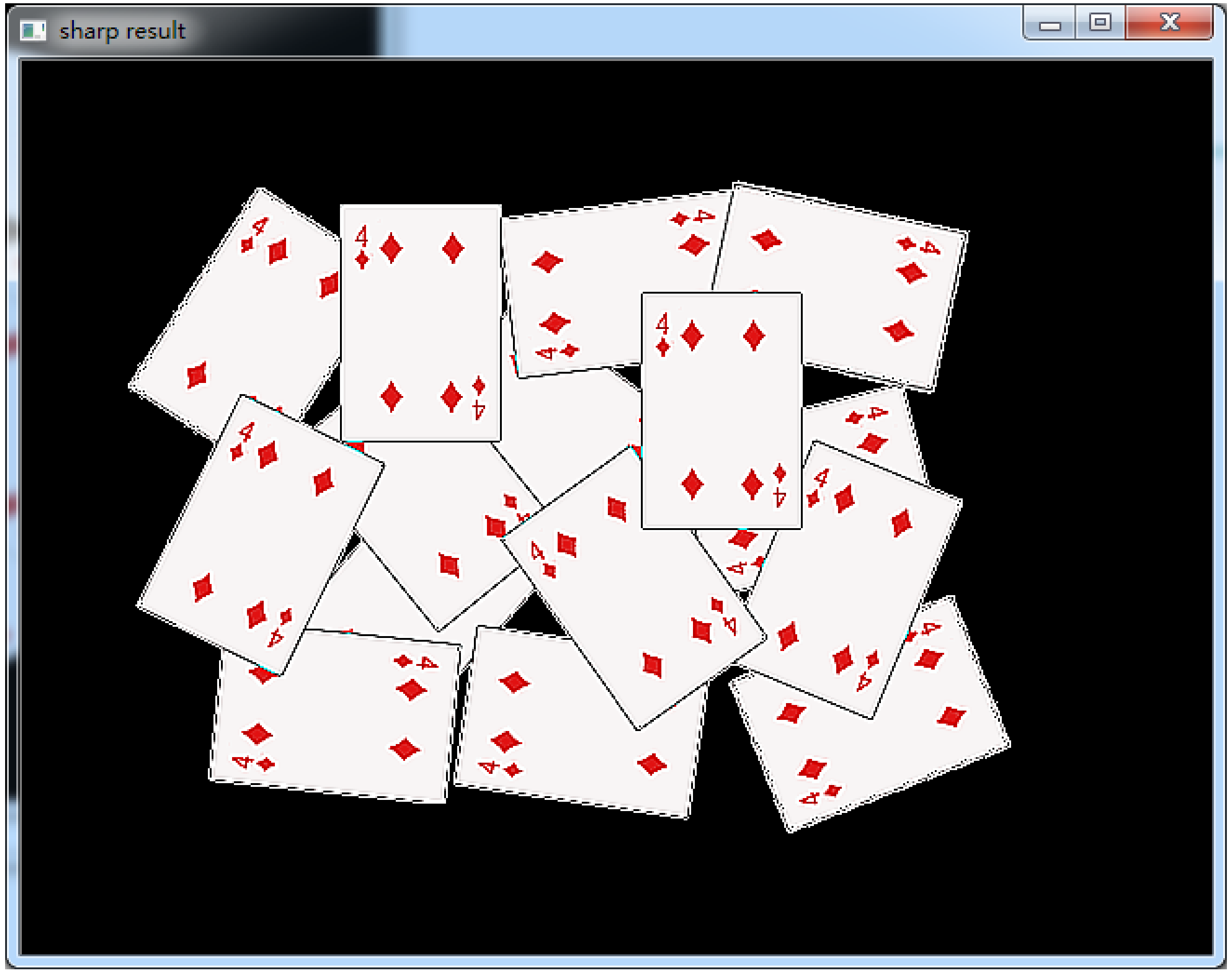
```
// change the background
for (int row = 0; row < src.rows; row++) {
    for (int col = 0; col < src.cols; col++) {
        if (src.at<Vec3b>(row, col) == Vec3b(255, 255, 255)) {
            src.at<Vec3b>(row, col)[0] = 0;
            src.at<Vec3b>(row, col)[1] = 0;
            src.at<Vec3b>(row,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



为梦想增值!

演示代码-Sharp

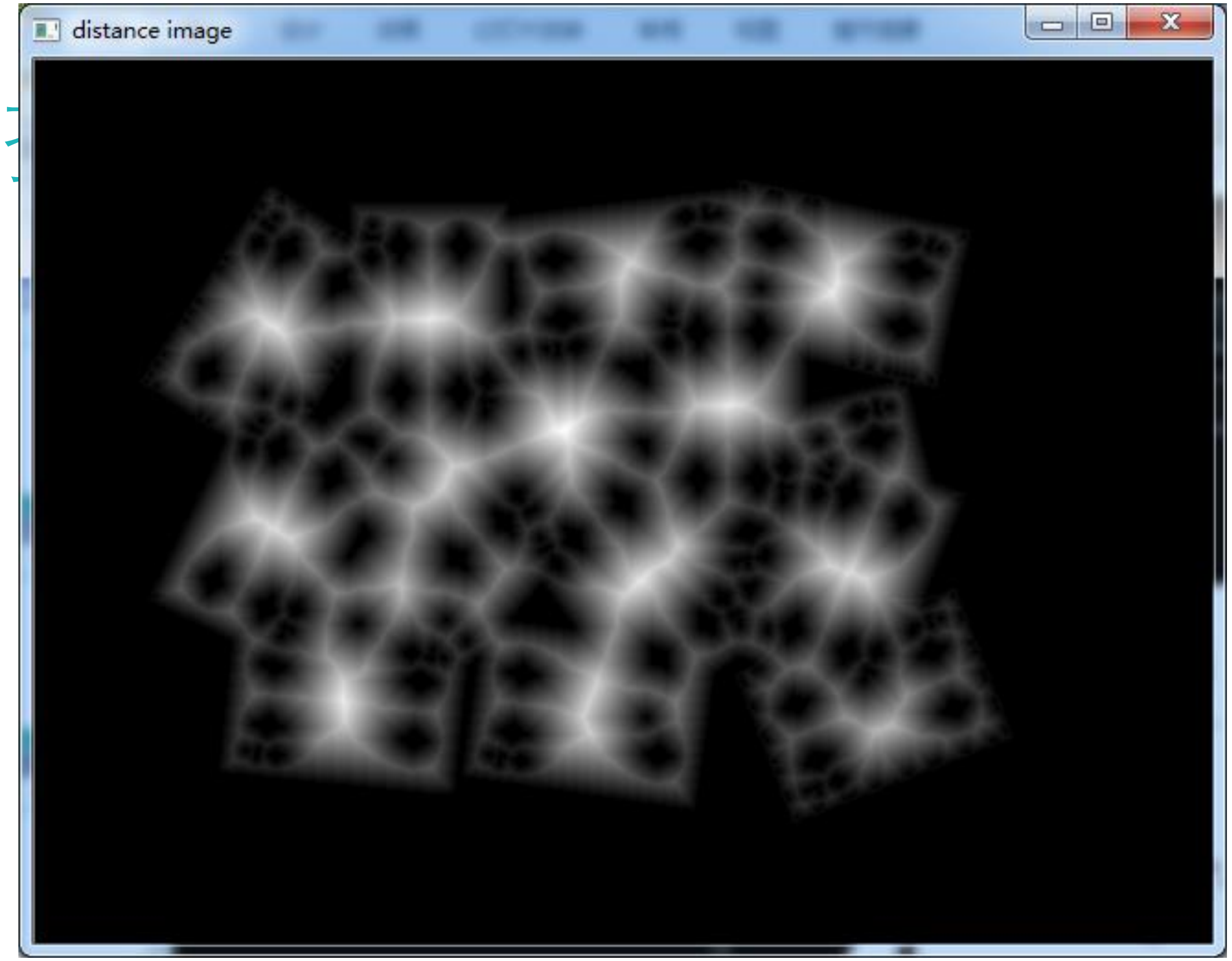
```
// create kernel
Mat kernel = (Mat_<float>(3, 3) << 1, 1, 1,
    1, -8, 1,
    1, 1, 1);

// 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);
```



为梦想增值！

演示代码-二值距离变

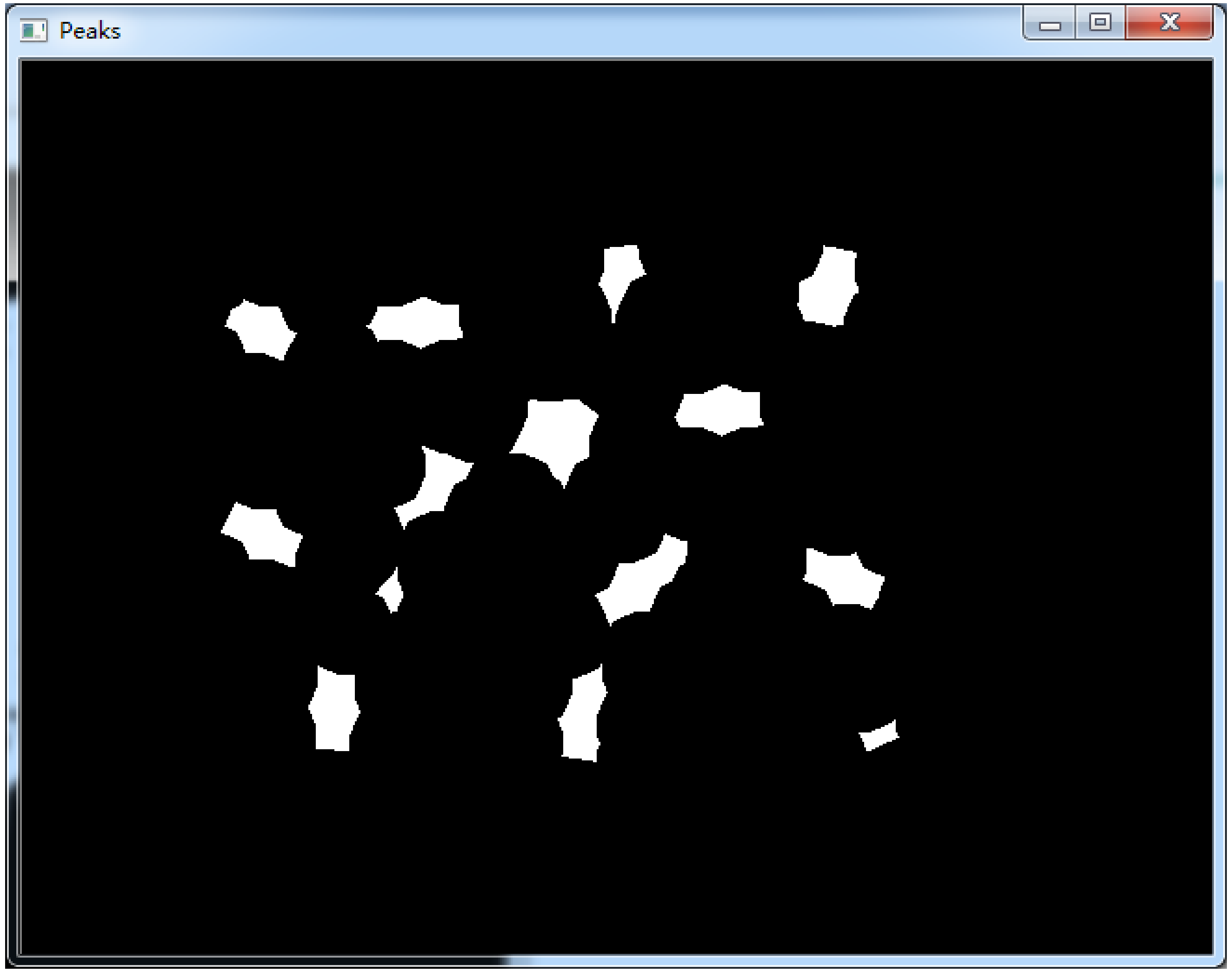
```
// 二值图像
Mat binImg;
cvtColor(src, binImg, CV_BGR2GRAY);
threshold(binImg, binImg, 40, 255, CV_THRESH_BINARY | CV_THRESH_OTSU);
imshow("Binary Image", binImg);

// 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);
```



为梦想增值!

演示代码-二值腐蚀

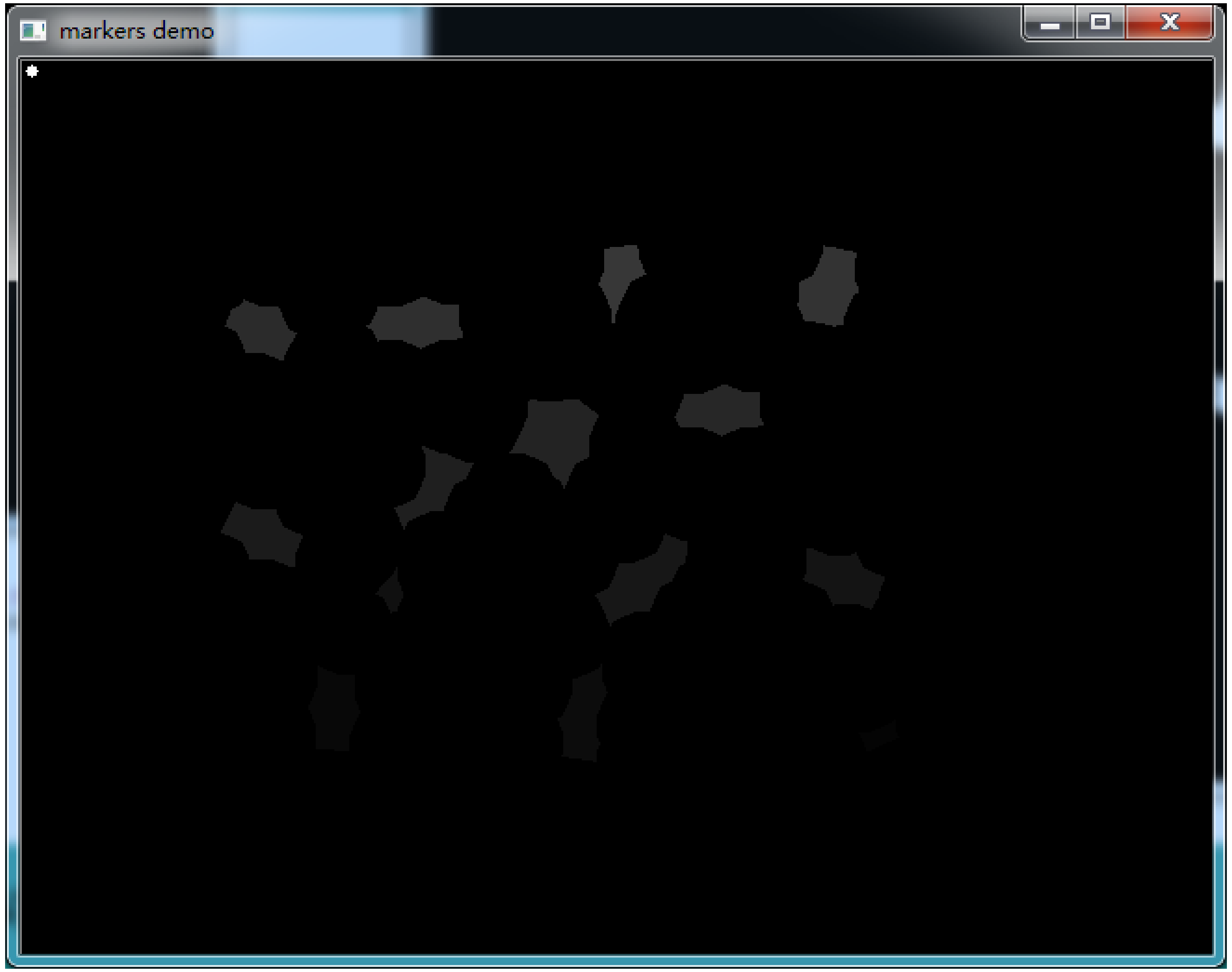
```
// try to get marker
threshold(dist, dist, .4, 1., CV_THRESH_BINARY);

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



为梦想增值!

演示代码-标记

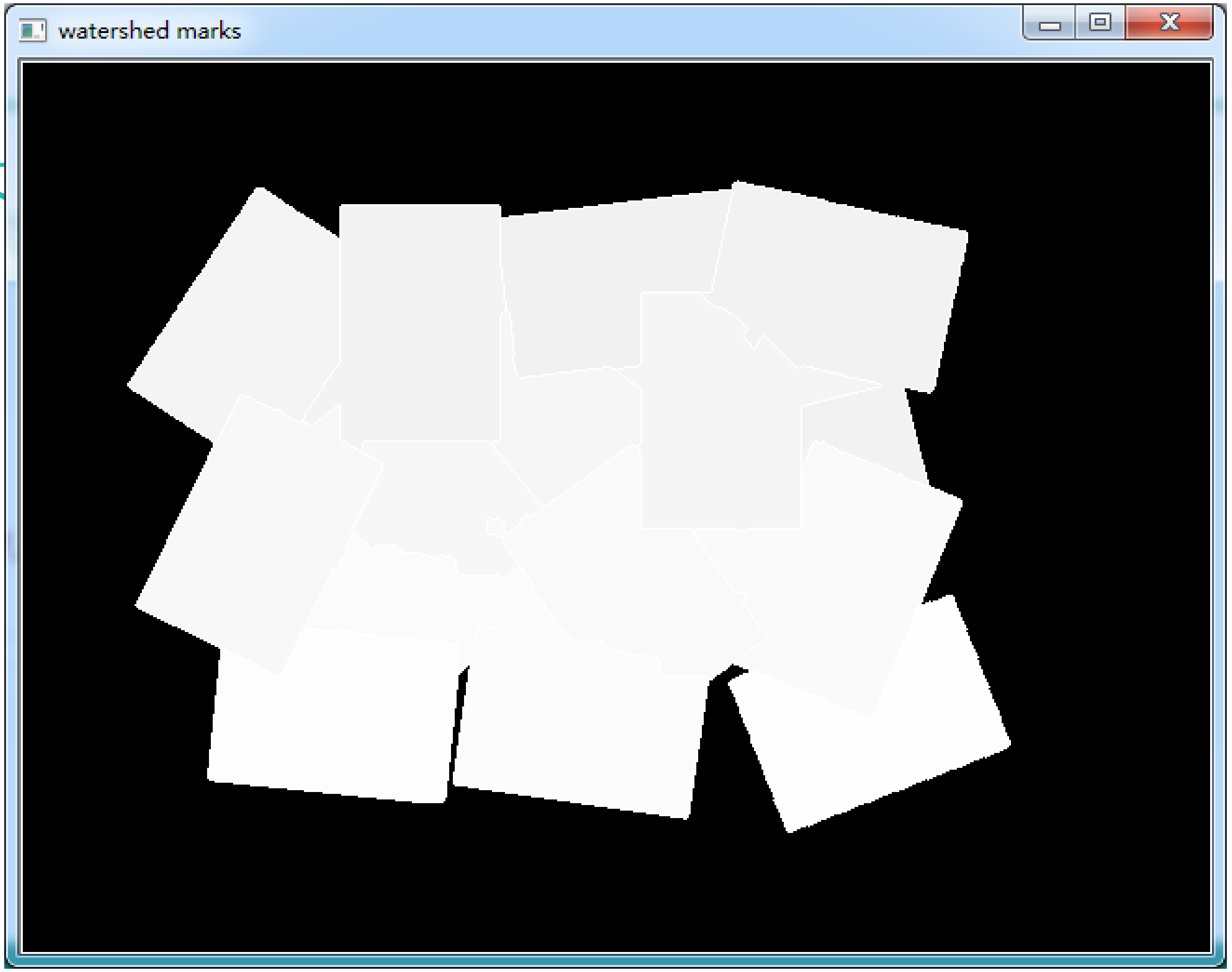
```cpp
Mat dist_8u;
dist.convertTo(dist_8u, CV_8U);
// find contours
vector<vector<Point>> contours;
findContours(dist_8u, contours, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE, Point(0, 0));

// 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);
```



为梦想增值!

演示代码-分水岭变换

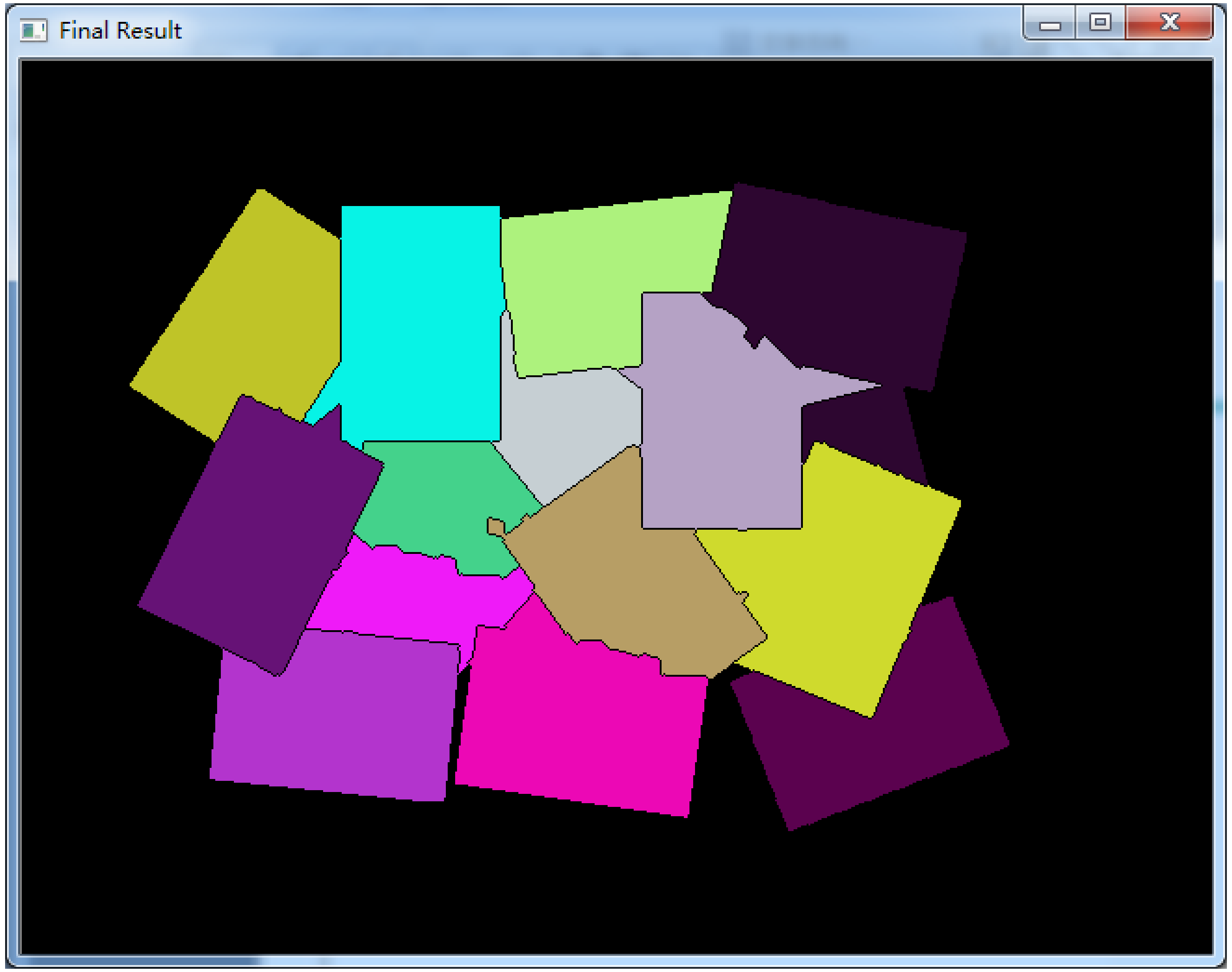
```
// perform watershed transform
watershed(src, markers);

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



为梦想增值！

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);
```



为梦想增值!