山东大学 计算机科学与技术 学院

计算机视觉 课程实验报告

学号：201822130233	姓名： 李云龙	
实验题目：图像结构 1		

实验过程中遇到和解决的问题：

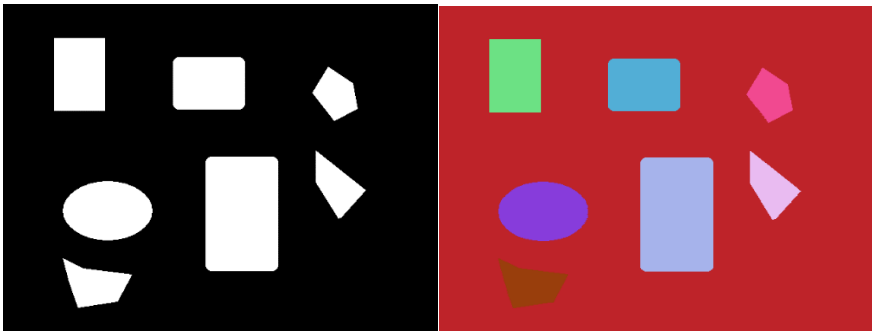
（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

实验 7.1 连通域

对一幅输入二值图像进行如下操作（连通域计算可以调用 cv::connectedComponent 函数）：

对每个连通区域进行随机上色，如下图所示；

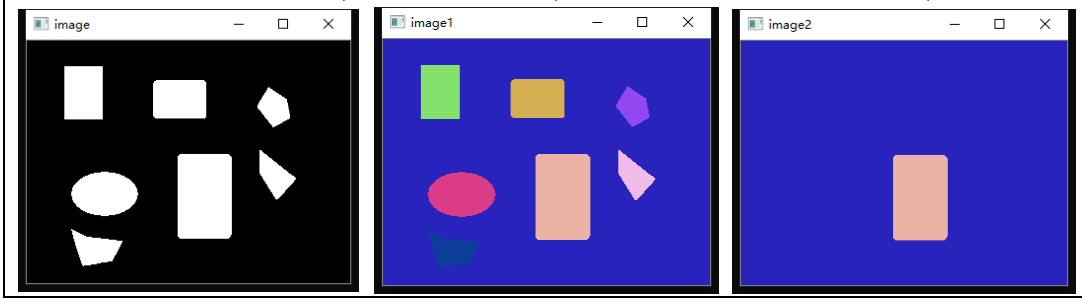
删除较小的前景区域，只保留最大的一个；



使用 OpenCV 的 connectedComponentsWithStats 函数，返回值为连通域个数，函数声明如下：

```
int cv::connectedComponentsWithStats (
    cv::InputArrayn image, // input 8-bit single-channel (binary)
    cv::OutputArray labels, // output label map
    cv::OutputArray stats, // Nx5 matrix (CV_32S) of statistics: x, y, width, height, area
    cv::OutputArray centroids, // Nx2 CV_64F matrix of centroids: cx, cy
    int connectivity = 8, // 4- or 8-connected components
    int ltype = CV_32S // Output label type (CV_32S or CV_16U)
);
```

调用 connectedComponentsWithStats 函数，获取到每个连通域的信息，随机初始化每个连通域的颜色，遍历每个位置，根据其 label 值进行上色，得到 image1。



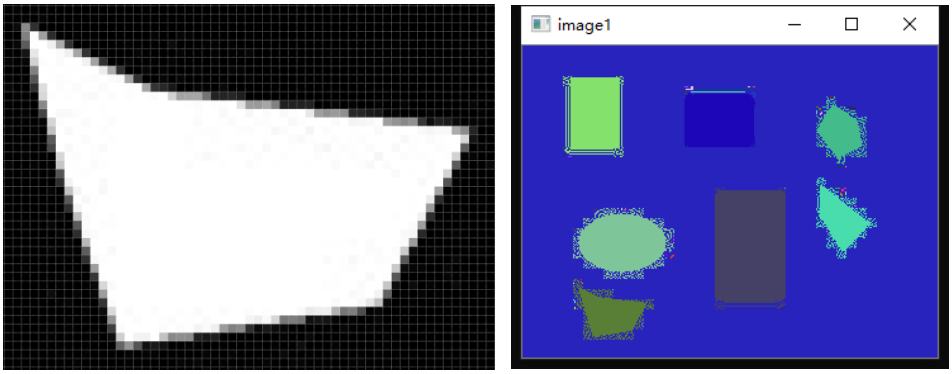
stats 中保存了每个连通域的大小，遍历每个连通域，排除掉 label 为 0 的（背景）找出面积最大的前景，遍历每个像素位置，对于面积不是最大的前景都按背景处理以删除较小的前景只保留最大的一个得到 image2。

尝试自己写一个计算连通域的代码，扫描一边图像，将每个像素所属连通域保存在 image_label 数组中。对第一行单独处理，之后的每一行中的像素都要和左边上边的像素进行比较以确定属于哪个连通域，当要合并两个连通域时，采取比较简单的做法（但是可能违背了减少运算的初衷），为避免错误，从头开始遍历到左边元素，把每个属于左边元素标签的都重新标记为上方元素的标签，连通域个数减一。

```
for (int k = (y - 1)*col + x; k < y*col + x; ++k) {
    if (image_label[k] == left_label) {
        image_label[k] = up_label;
    }
}
image_label[y*col + x] = up_label;
--cnt;
```

遍历 image_label 数组，根据其 label 值进行上色。由于 label 不一定是成规律的（如 12345），不好依据 label 存储在数组中（其实可以存在 map 里），因此颜色用 label 运算得出而不是用随机数。遍历一次 image_label 数组，用 map 记录即可找到数量最多的 label 即最大前景（不算背景 label-1）。

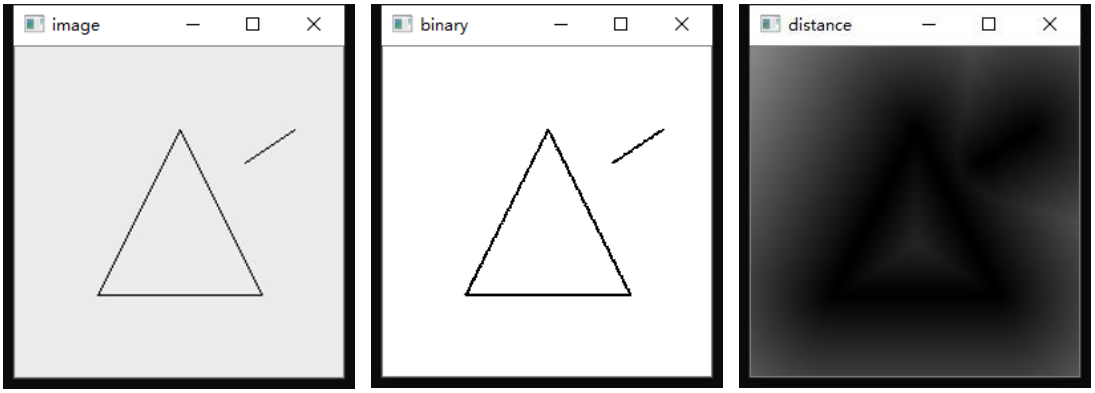
原图像与后面的操作之间还有一步二值化处理，遍历时认为像素仅 0 和 255 两种作为背景与前景，而最开始的图像是由中间值的，所以会有问题。这种情况下需要选取一个合适的阈值。



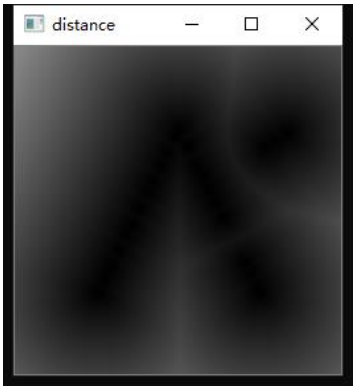
实验 7.2 距离变换

了解 OpenCV 的距离变换函数 distanceTransform，使用合适的测试图像进行测试，并将距离场可视化输出。

使用 OpenCV 的 distanceTransform 函数计算图像中每一个非零点距离离自己最近的零点的距离，distanceTransform 的第二个 Mat 矩阵参数保存了每一个点与最近的零点的距离信息，图像上越亮的点，代表了离零点的距离越远。选取一个合适的阈值对输入图像做二值化处理，使用 distanceTransform 函数得到 dis_image，将其每个元素转化为 uchar 类型得到最终可视化输出 show_image。



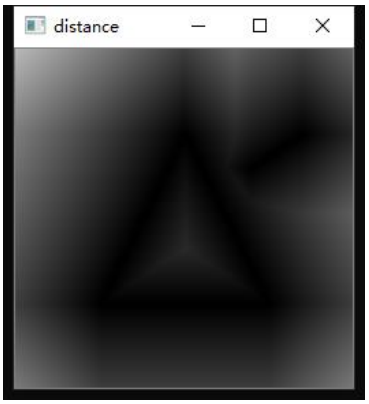
若不进行二值化处理，得到的结果是不符合预期的，原因是黑色并不是灰度值为 0 的纯黑，在计算距离时没有纳入计算。



尝试自己依据二维的距离变换实现，完成初始化后，正向从左上到右下，反向从右下到左上各遍历一遍。效果与传入参数 DIST_L1 时的 distanceTransform 一致

```
int row = image.rows, col = image.cols;
Mat dis_image(image.size(), CV_32SC1);
for (int y = 0; y < row; ++y) {
    const uchar *p = image.ptr<uchar>(y);
    int *dp = dis_image.ptr<int>(y);
    for (int x = 0; x < col; ++x) {
        if (p[x] == 0) dp[x] = 0;
        else if (p[x] == 255) dp[x] = UCHAR_MAX;
    }
}
```

```
for (int x = 1; x < col; ++x) {
    dis_image.at<int>(0, x) = min(dis_image.at<int>(0, x), dis_image.at<int>(0, x - 1) + 1);
}
for (int y = 1; y < row; ++y) {
    dis_image.at<int>(y, 0) = min(dis_image.at<int>(y, 0), dis_image.at<int>(y - 1, 0) + 1);
}
for (int y = 1; y < row; ++y) {
    for (int x = 1; x < col; ++x) {
        dis_image.at<int>(y, x) = min(dis_image.at<int>(y, x), dis_image.at<int>(y - 1, x) + 1);
        dis_image.at<int>(y, x) = min(dis_image.at<int>(y, x), dis_image.at<int>(y, x - 1) + 1);
    }
}
```



结果分析与体会：

通过本次实验，使用了 OpenCV 的函数 connectedComponentsWithStats 计算连通域，distanceTransform 计算并可视化距离场，并且自己也尝试用课上所学实现了相关代码。