

二、实验内容和原理

1、Image Binarization

图像二值化（Image Binarization）就是将图像上的像素点的灰度值设置为0或255，也就是将整个图像呈现出明显的黑白效果的过程。

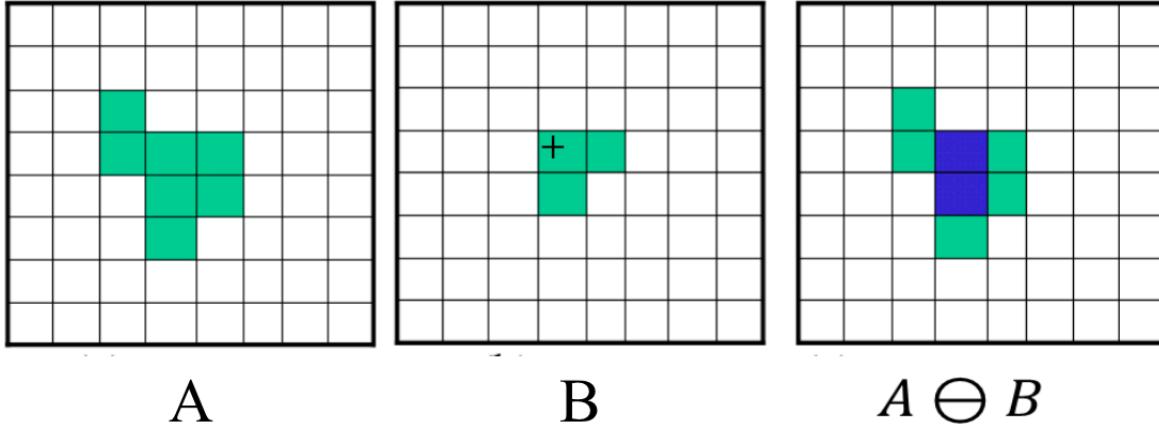
将256个亮度等级的灰度图像通过适当的阈值选取而获得仍然可以反映图像整体和局部特征的二值化图像。在数字图像处理中，二值图像占有非常重要的地位。首先，图像的二值化有利于图像的进一步处理，使图像变得简单，而且数据量减小，能凸显出感兴趣的目标的轮廓。其次，要进行二值图像的处理与分析，首先要把灰度图像二值化，得到二值化图像。所有灰度大于或等于阈值的像素被判定为属于特定物体，其灰度值为255表示，否则这些像素点被排除在物体区域以外，灰度值为0，表示背景或者例外的物体区域。

特别注意的是，如何选择合适的阈值是很关键的，阈值过高或过低都会影响图片的质量。将一张RGB色彩的BMP图像转化成二值化图像，主要可分为三步：将RGB空间转化成YUV空间；将每个像素点的YUV空间的Y值根据阈值（Threshold）划分为255或者0，255代表白色，0代表黑色；最后输出二值化图像。

2、Erosion

Erosion即图像腐蚀操作。公式定义如下。

$$A \ominus B = \{(x, y) | (B)_{xy} \subseteq A\} \quad (5)$$

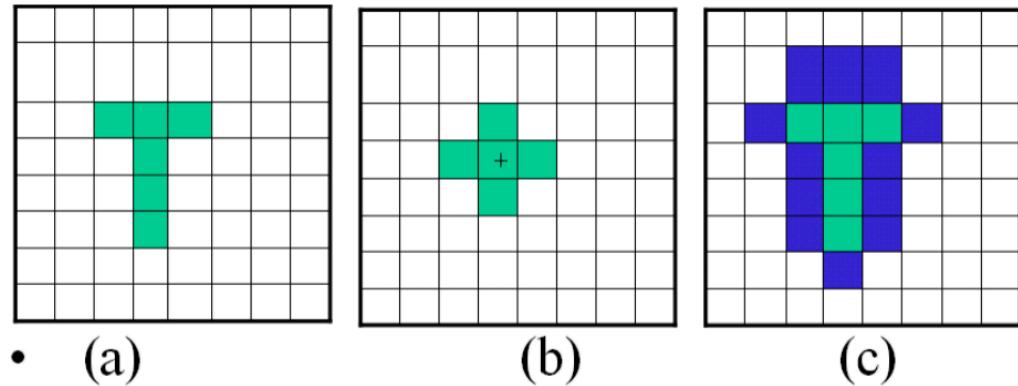


腐蚀操作即用我预设的腐蚀单元，如果某个像素群等于腐蚀单元，那么就将腐蚀单元的中心位置置为存在。腐蚀操作可以将图像“瘦身”，去除噪点，达到滤波的作用。

3、Dilation

Dilation即图像的膨胀操作。公式定义如下。

$$A \oplus B = \{z | (B)_z \cap A \neq \emptyset\} \quad (6)$$

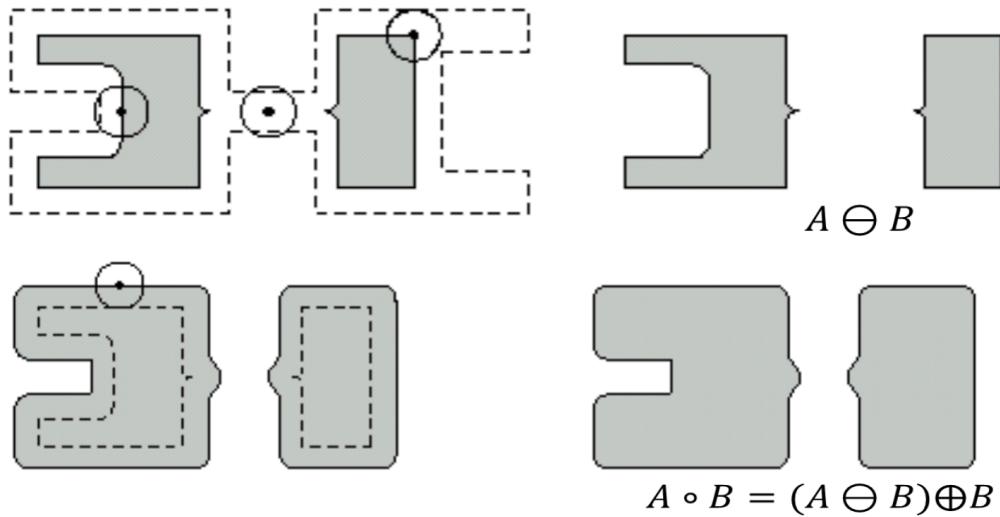


膨胀操作类似于腐蚀操作的反操作，如果我预设的膨胀单元的中心点和某个像素点重合，那么就将该像素点的周围补充成膨胀单元的形状。膨胀操作可以扩大每个像素点的边界，可以将原本相隔一定距离的像素点连接起来，将不连续的像素点转换成连续的，达到平滑边界的作用。

4、Opening

开操作即“腐蚀+膨胀”，公式定义如下。

$$A \circ B = (A \ominus B) \oplus B \quad (7)$$

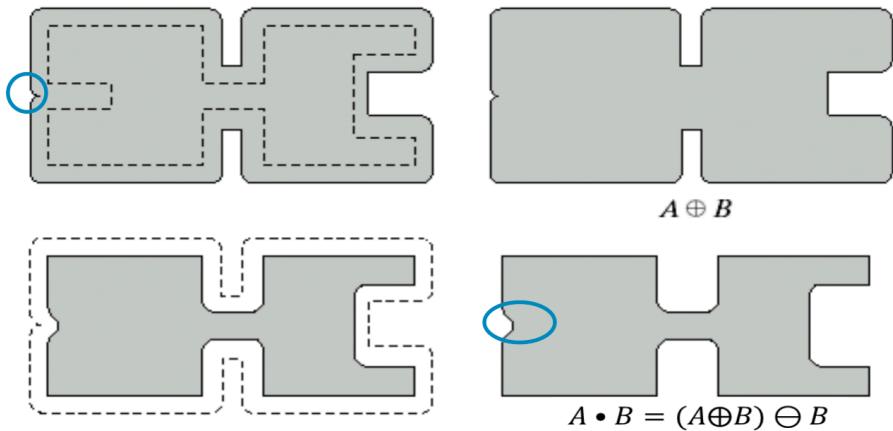


开操作将原本的一些细节省略了，再尽量恢复原始大小，先腐蚀可以达到滤波的效果。但是也不可避免地将图形变得更加分散，将原有的一些细小连接给舍弃了。

5. Closing

闭操作即“膨胀+腐蚀”，公式定义如下。

$$A \bullet B = (A \oplus B) \ominus B \quad (8)$$



闭操作则强调将原来断开的、不连续的部分连接起来，达到平滑的效果。但是有可能会填充掉一些空白处，省略细节。

三、实验步骤与分析

1、读入BMP文件、存储RGB空间、转化YUV空间

(1) 数据结构

这一步骤与HW1类似，这里仅仅多了二值化结构 BINA。

```

1  typedef struct FILEHEADER{
2      unsigned int bfSize;           // Size of Figure
3      unsigned short bfReserved1;    // Reserved1, must be 00
4      unsigned short bfReserved2;    // Reserved2, must be 00
5      unsigned int bfOffBits;        // Offsets from header to info
6 }BMPFILEHEADER;
7
8 typedef struct INFOHEADER{
9     unsigned int biSize;           // Size of this structure
10    unsigned int biWidth;          // Width
11    unsigned int biHeight;         // Height
12    unsigned short biPlanes;       // always is 1
13    unsigned short biBitCount;     // Kind of BMP, bit/pixel = 1 4 8
16 24 32
14    unsigned int biCompression;
15    unsigned int biSizeImage;
16    unsigned int biXPelsPerMeter;
17    unsigned int biYPelsPerMeter;
18    unsigned int biClrUsed;
19    unsigned int biClrImportant;

```

```

20 }BMPINFOHEADER;
21
22 typedef struct RGBSPACE{
23     unsigned char blue;
24     unsigned char green;
25     unsigned char red;
26 }RGB;
27
28 typedef struct YUVSPACE{
29     unsigned char Y;
30     unsigned char U;
31     unsigned char V;
32 }YUV;
33
34 typedef struct BINASPACE{
35     unsigned char I;
36 }BINA;

```

(2) 读入、转换

```

1 int main(){
2     FILE *infp,*binafp;
3
4     unsigned short bftype;
5     // head includes head information of BMP file.
6     BMPFILEHEADER head;
7     // info includes figure information of BMP file.
8     BMPINFOHEADER info;
9
10    // read F1.bmp
11    if((infp=fopen("F1.bmp","rb")) == NULL){
12        printf("OPEN FAILED!\n");
13        return 0;
14    }
15
16    // bftype must be 0x4d42.
17    fread(&bftype,1,sizeof(unsigned short),infp);
18    if(bftype != 0x4d42){
19        printf("This figure is not BMP!\n");
20        Sleep(1000);
21        return 0;
22    }
23
24    // read HEADER
25    fread(&head,1,sizeof(BMPFILEHEADER),infp);

```

```

26     fread(&info, 1, sizeof(BMPINFOHEADER), infp);
27
28     // We takes 24 bitcount bmp for experiment item.
29     if(info.biBitCount != 24){
30         printf("This BMP is not 24 biBitCount!\n");
31         Sleep(1000);
32         return 0;
33     }
34     /////////////////////////////////
35     /////////////////////
36     int size = info.biSizeImage / 3;
37
38     // define 3 array
39     RGB inrgb[size];
40     YUV rgb2yuv[size];
41     BINA binary[size];
42
43     // read information.
44     fread(inrgb, size, sizeof(RGB), infp);
45     fclose(infp);
46
47     int i;
48
49     // change RGB into YUV.
50     for(i=0;i<=size-1;i++){
51         rgb2yuv[i].Y = (( 66 * inrgb[i].red + 129 * inrgb[i].green +
52             25 * inrgb[i].blue + 128) >> 8) + 16;
53         rgb2yuv[i].U = (( -38 * inrgb[i].red - 74 * inrgb[i].green +
112 * inrgb[i].blue + 128) >> 8) + 128;
54         rgb2yuv[i].V = (( 112 * inrgb[i].red - 94 * inrgb[i].green -
18 * inrgb[i].blue + 128) >> 8) + 128;
55     }

```

2、二值化操作

思路上很简单，根据阈值来判断每个像素是否是1或者0。关键在于阈值的选取，实验中我采用二分查找的方法，不断缩小合适范围边界，以得到最适合的阈值。

```

1     // threshold 阈值
2     int threshold = 130;
3
4     // If Y >= threshold, we store 255 in this pixel, else store 0.
5     for(i=0;i<=size-1;i++){
6         if(rgb2yuv[i].Y >= threshold) binary[i].I = 255;
7         else binary[i].I = 0;

```

```

8     }
9
10 ///////////////////////////////////////////////////////////////////
11 // Output Binary BMP.
12 if((binafp = fopen("RGB2Binary.bmp","wb")) == NULL){
13     printf("Create Binary image FAILED!\n");
14     Sleep(1000);
15     return 0;
16 }
17
18 // Write header information and data into file.
19 fwrite(&bftype,1,sizeof(unsigned short),binafp);
20 fwrite(&head,1,sizeof(BMPFILEHEADER),binafp);
21 fwrite(&info,1,sizeof(BMPINFOHEADER),binafp);
22
23 for(i=0;i<=size-1;i++){
24     fwrite(&binary[i].I,1,sizeof(unsigned char),binafp);
25     fwrite(&binary[i].I,1,sizeof(unsigned char),binafp);
26     fwrite(&binary[i].I,1,sizeof(unsigned char),binafp);
27 }
28
29 fclose(binafp);
30 printf("Output Binary Image successfully!\n");
31 ///////////////////////////////////////////////////////////////////
32
33 }

```

3、Erosion

这里只展示关键代码。原理就是遍历，对于每一个像素点周围的情况进行判断，若符合条件，则在结果矩阵中将该腐蚀单元的中心点记为有效。

```

1   for(i=1;i<=height-2;i++){
2       for(j=1;j<=width-2;j++){
3           //if(binary[i-1][j-1] == 255) continue;
4           if(binary[i-1][j] == 255) continue;
5           //if(binary[i-1][j+1] == 255) continue;
6           if(binary[i][j-1] == 255) continue;
7           if(binary[i][j] == 255) continue;
8           if(binary[i][j+1] == 255) continue;
9           //if(binary[i+1][j-1] == 255) continue;
10          if(binary[i+1][j] == 255) continue;
11          //if(binary[i+1][j+1] == 255) continue;

```

```

12         ero[i][j] = 0;
13     }
14 }
```

以上代码选取了“十字形”的腐蚀单元。

4、Delation

膨胀操作就是遍历每一个像素点，若该像素点的中心值与膨胀单元的中心值重合相等，那么就将该单元膨胀到该像素点上。

```

1   for(i=1;i<=height-1;i++){
2       for(j=1;j<=width-2;j++){
3           if(binary[i][j] == 0){
4               // dela[i-1][j-1] = 0;
5               dela[i-1][j] = 0;
6               // dela[i-1][j+1] = 0;
7               dela[i][j-1] = 0;
8               dela[i][j] = 0;
9               dela[i][j+1] = 0;
10              // dela[i+1][j-1] = 0;
11              dela[i+1][j] = 0;
12              // dela[i+1][j+1] = 0;
13          }
14      }
15  }
```

以上代码选择了“十字形”的膨胀单元。

5、Opening

开操作与上述二者操作类似，先进行腐蚀操作，再对得到的图像进行膨胀操作。

6、Closing

闭操作与上述二者类似，先进行膨胀操作，再对得到的图像进行腐蚀操作。

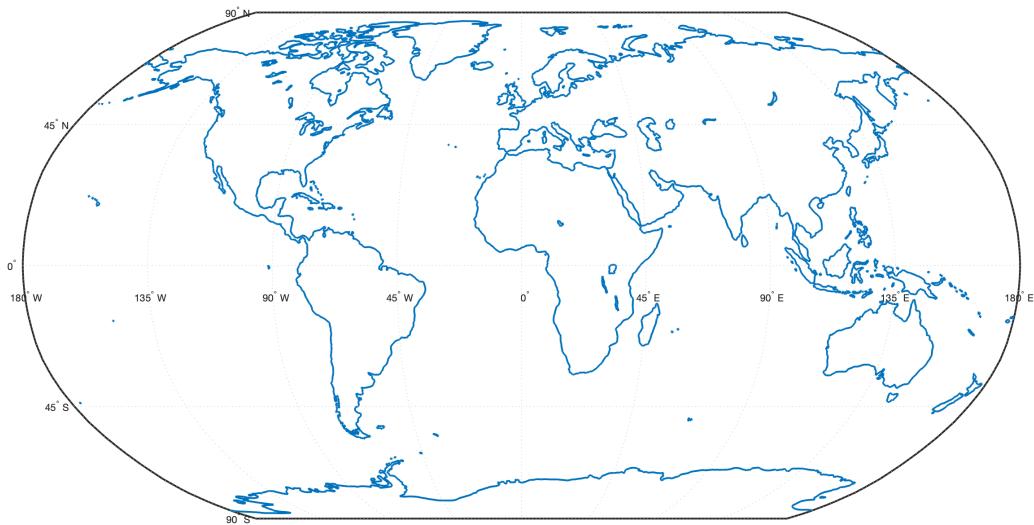
四、实验环境及运行方法

本实验运行在MACOS系统下VM Ware软件下Windows 10虚拟机的Dev C++中。源代码可编译成功。

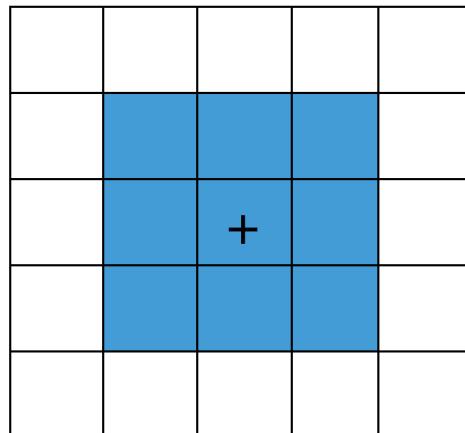
五、实验结果展示

1、Matlab中绘制的世界海岸线

Matlab绘制的世界地图海岸线曲折，细看能发现有很多毛边，影响美观，因此我尝试能否用四种操作来优化海岸线的形状。采取的单元是“9宫格”的单元，中心点是九宫格的中心点。

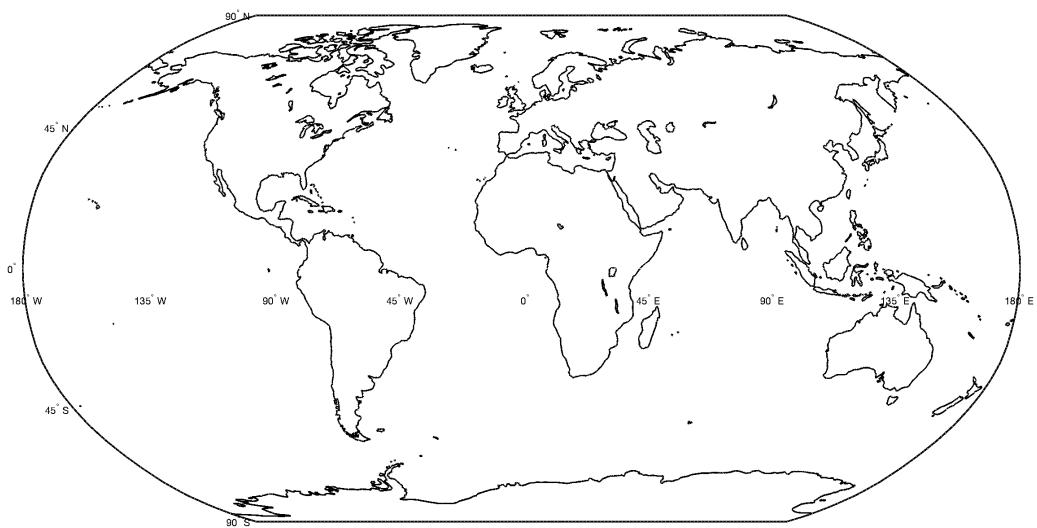


- 原始图像

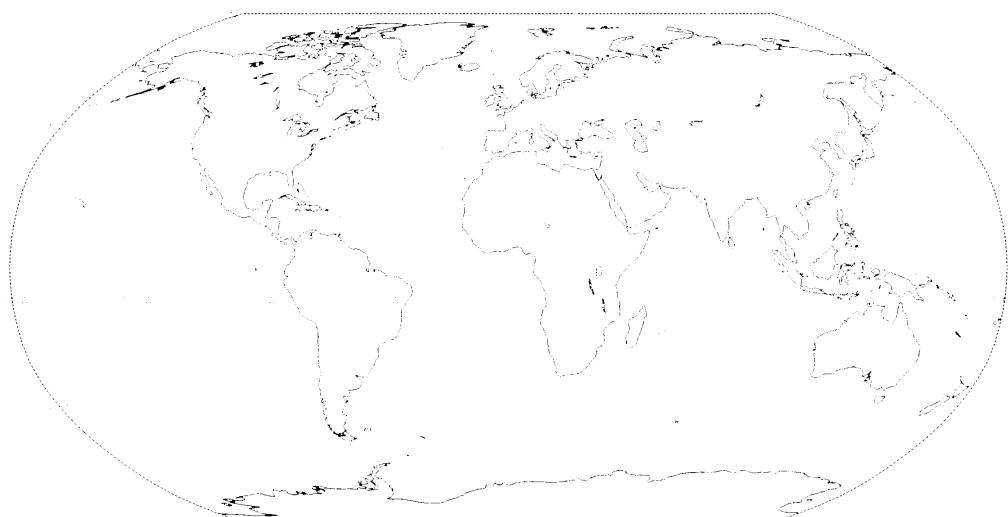


- 膨胀/腐蚀单元

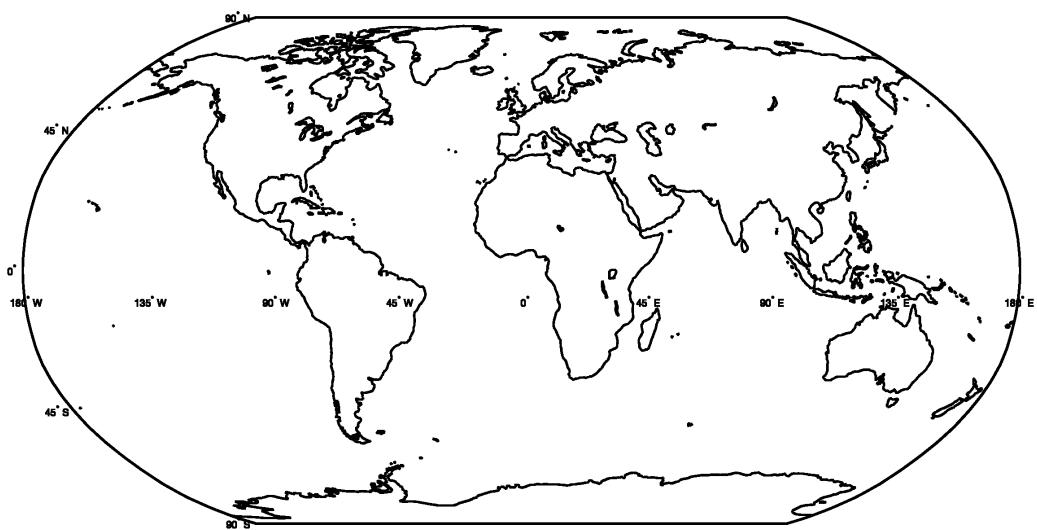
- (1) 二值化图像，阈值=125



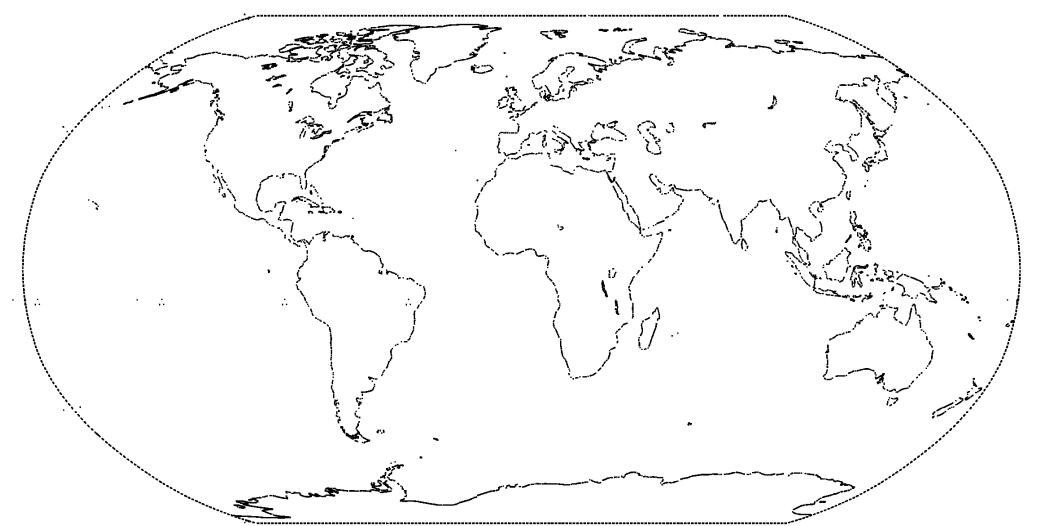
(2) 腐蚀操作图像

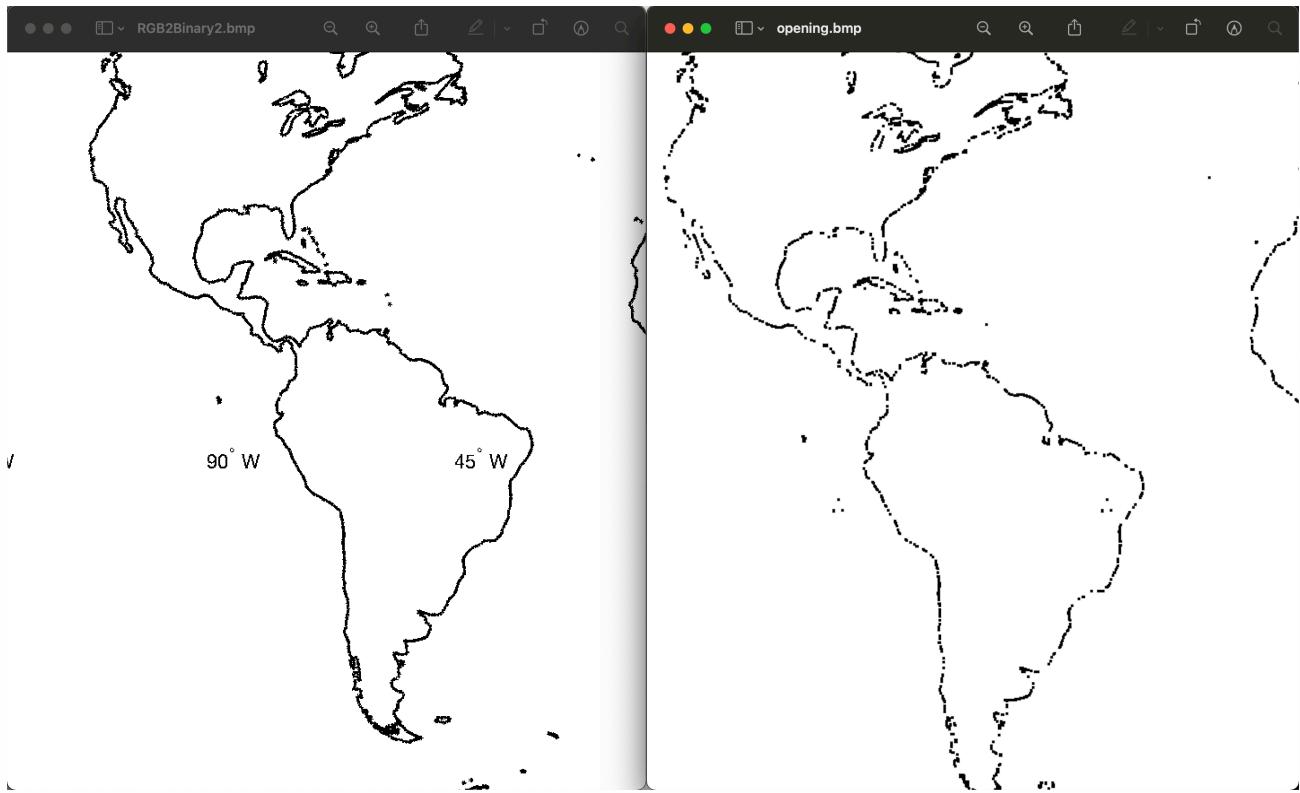


(3) 膨胀操作图像



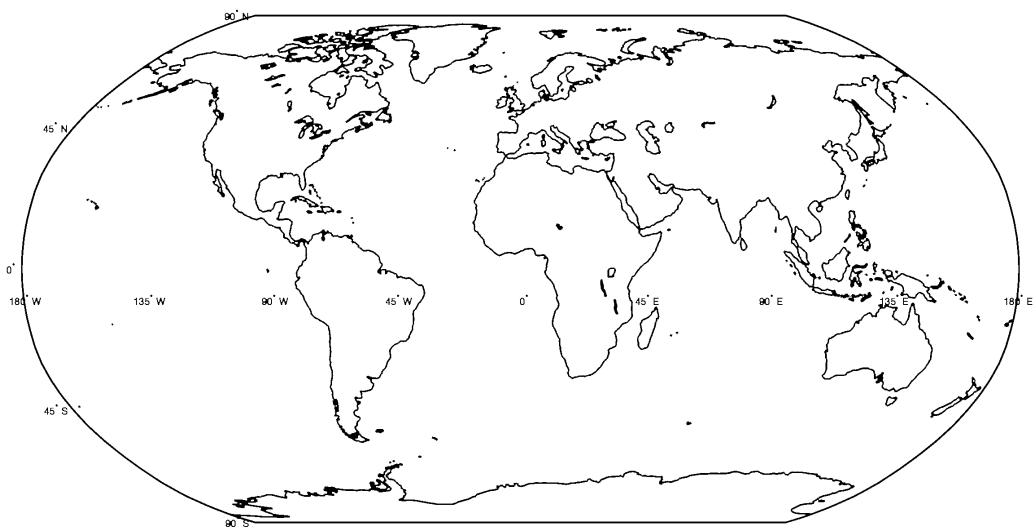
(4) 开操作图像

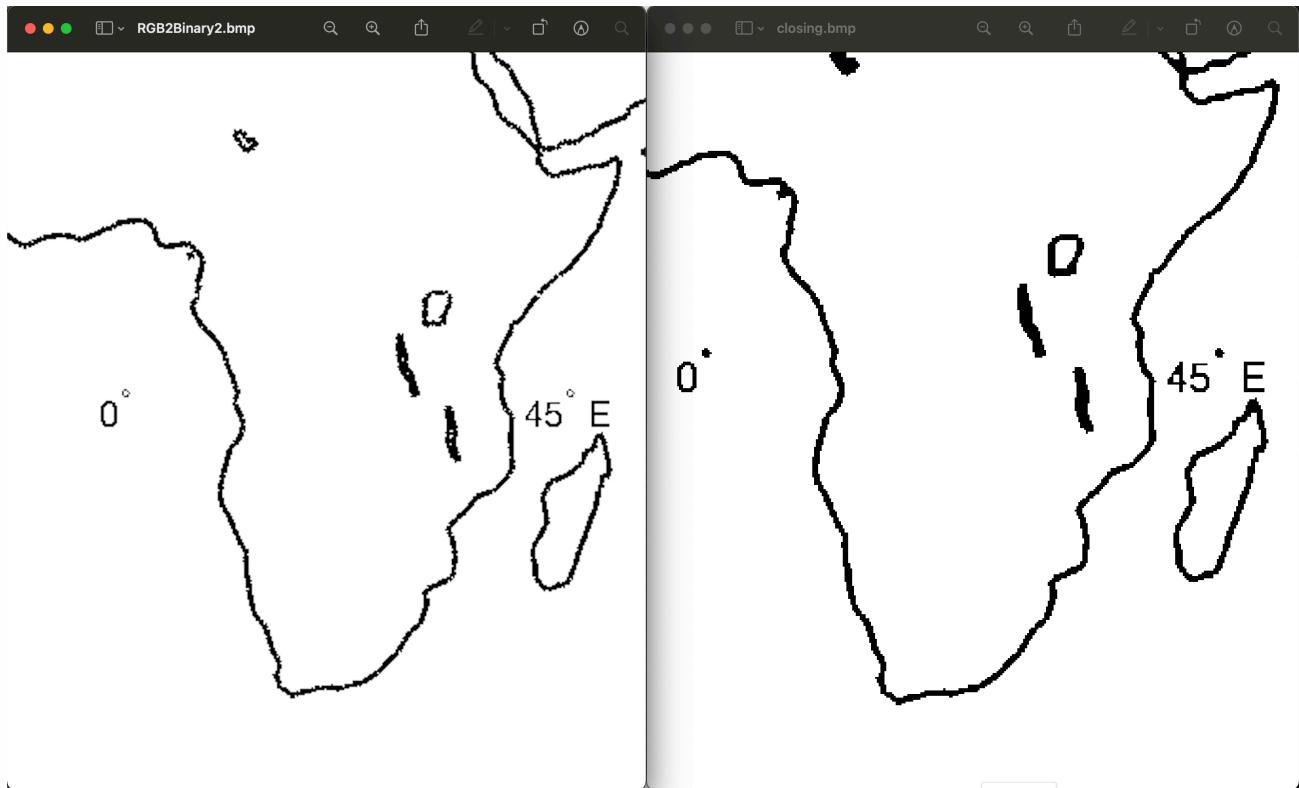




- 开操作在这张图中的特点是去掉了经纬度数据。经纬度数据的像素点较为分散，不太符合腐蚀单元的形状，因此在腐蚀操作中会被去掉。相当于进行了滤波操作。但是缺点也十分明显，原本连续的海岸线变得破碎。

(5) 闭操作图像





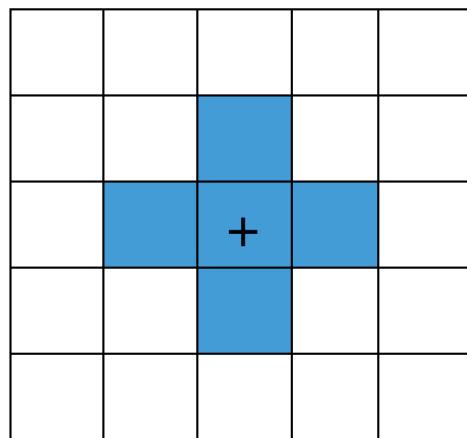
- 闭操作的最大特点是将原有的崎岖不平的线段变得平滑。比如非洲的南部海岸线，原来的二值化图像中是非常不平滑的，但是经过闭操作之后，原来的“毛躁”的海岸线变得更加平滑。

2、指纹实验

老师上课举的例子提到了指纹实验，因此我也尝试了指纹处理。膨胀单元和腐蚀单元选择的是“十字形”的像素群。



- 原始图像



- 膨胀/腐蚀单元

(1) 二值化图像，阈值=130



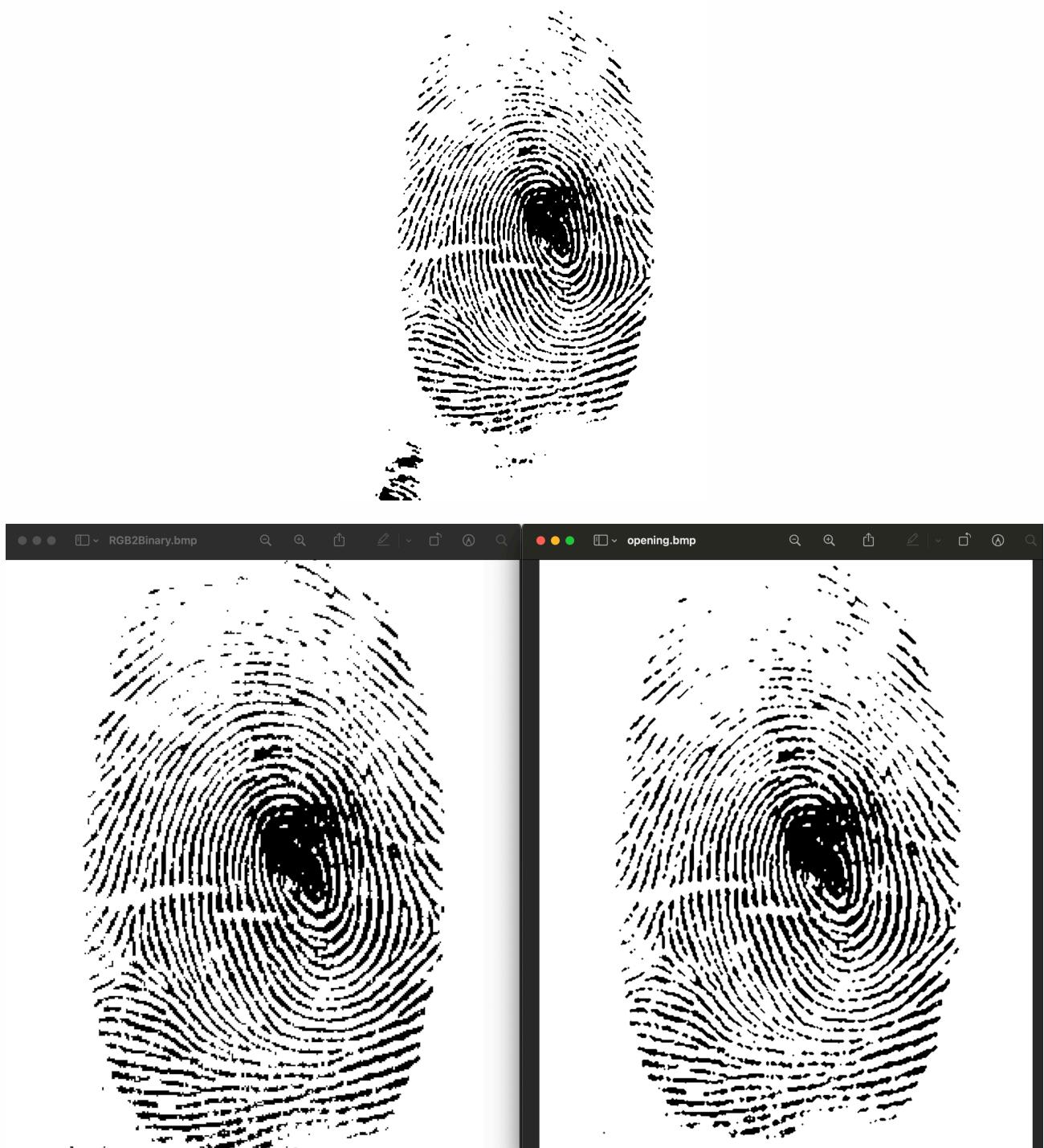
(2) 腐蚀操作图像



(3) 膨胀操作图像

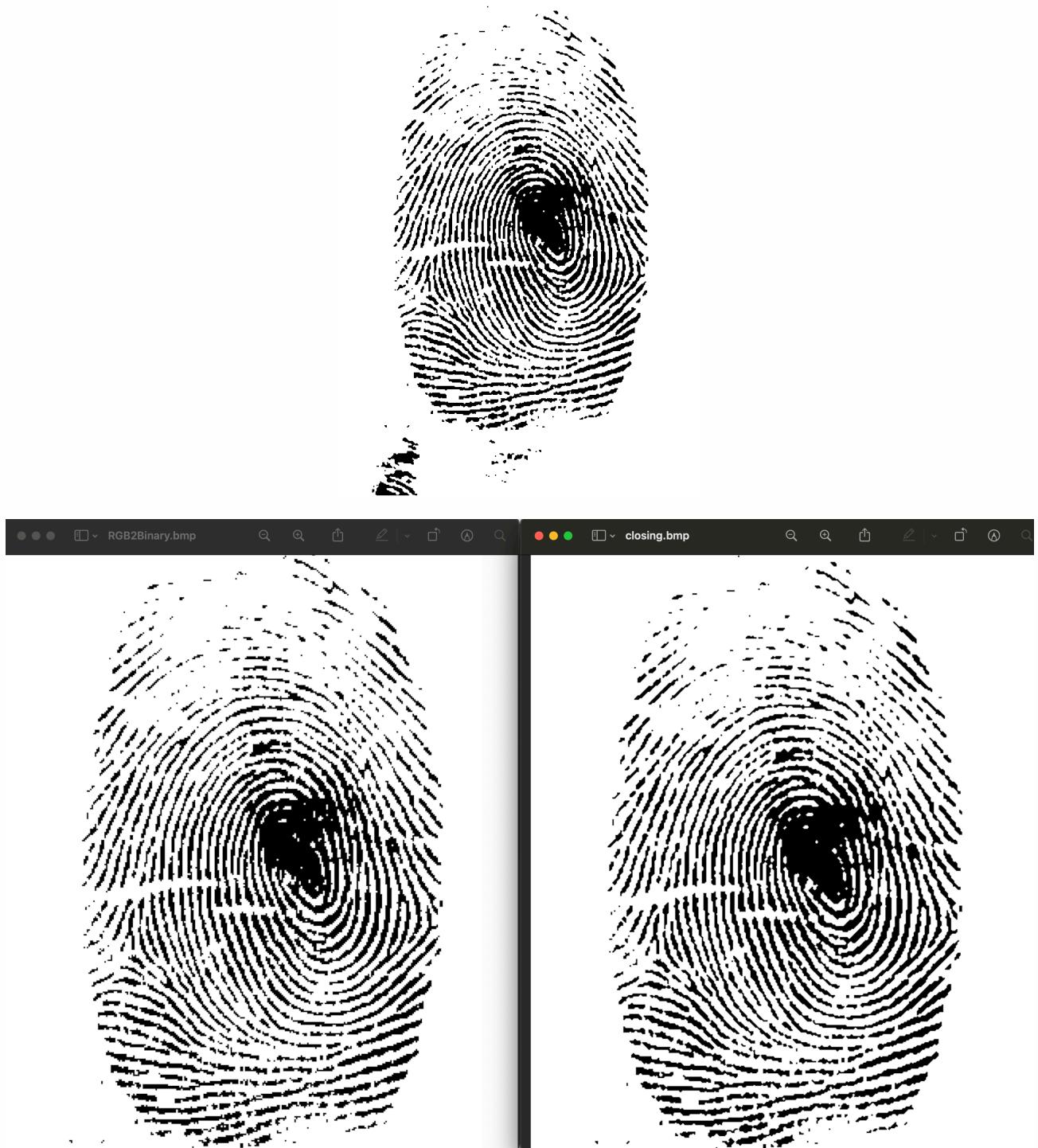


(4) 开操作图像



- 开操作去除了图像中的一些噪声点，达到了滤波的功能。使指纹的形状更加清楚

(5) 闭操作图像



- 闭操作将原来很多不连续的线条变得连续，特别是下方指纹的线条，原先是破碎、不连续，经过闭操作后变得更加均一平整。

六、心得体会

1、讨论

(1) 阈值的选取

阈值的选取好坏会直接影响到图像二值化的效果。比如世界地图中，线条较为简单，因此阈值很好确定。但是在指纹操作中，处理的图像是实际图像，存在很多的灰区域，在二值化的时候常会存在“顾此失彼”的问题。如果能使周边的指纹看清楚，那么中间的指纹就会很模糊，如果要使中间的指纹清楚，那么周边的指纹就会被省略很多。所以我尝试了很多次之后，指纹图像的灰度值选择在130左右是比较合适的。实际操作中可以使用二分查找的方法，不断缩小阈值的范围，以找到最佳呈现阈值。

(2) 膨胀、腐蚀单元的选取

为了实验不同不同腐蚀单元对于图像操作的效果，我根据不同目的选择了两种不同的单元对图像进行操作。我对海岸线主要想将其平滑处理，所以选择了九宫格式的单元，这样的海岸线的效果更加平滑，闭操作中的效果最佳。针对于指纹，其弯绕很多，我选择了十字形的单元进行操作，滤掉了很多的噪点，开操作的效果最佳，图像最为清晰，纹路最为整洁。实际过程中，针对不同需要设计不同的膨胀/腐蚀单元是值得思考的一件事。

2、心得

本次实验我们用C语言对图像进行了二值化、腐蚀、膨胀、开操作、闭操作，每种操作都有其自己的意义，针对自己的需要（平滑、滤波等）选择不同的单元及操作类型，可以得到不同的效果。实验还是很有意思的，自己动手也更加有收获。