数字图像处理第三次实验报告

作者：訾源

学号：161250220

邮箱：161250220@smail.nju.edu.cn

实验思路

本次实验中采用自己设计的启发式搜索算法来提取图片中的经纬线，过滤掉包括大陆线在内的其他线条。

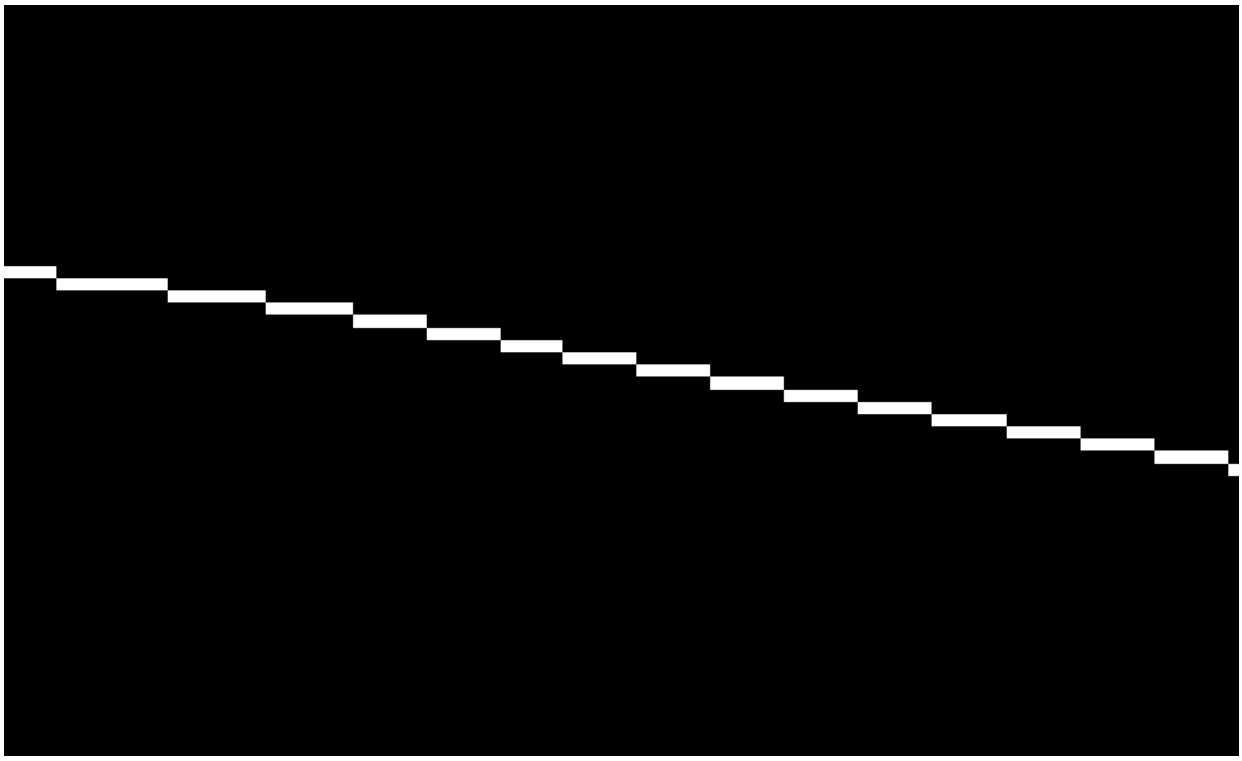
实验取得了较好的效果。

实现细节

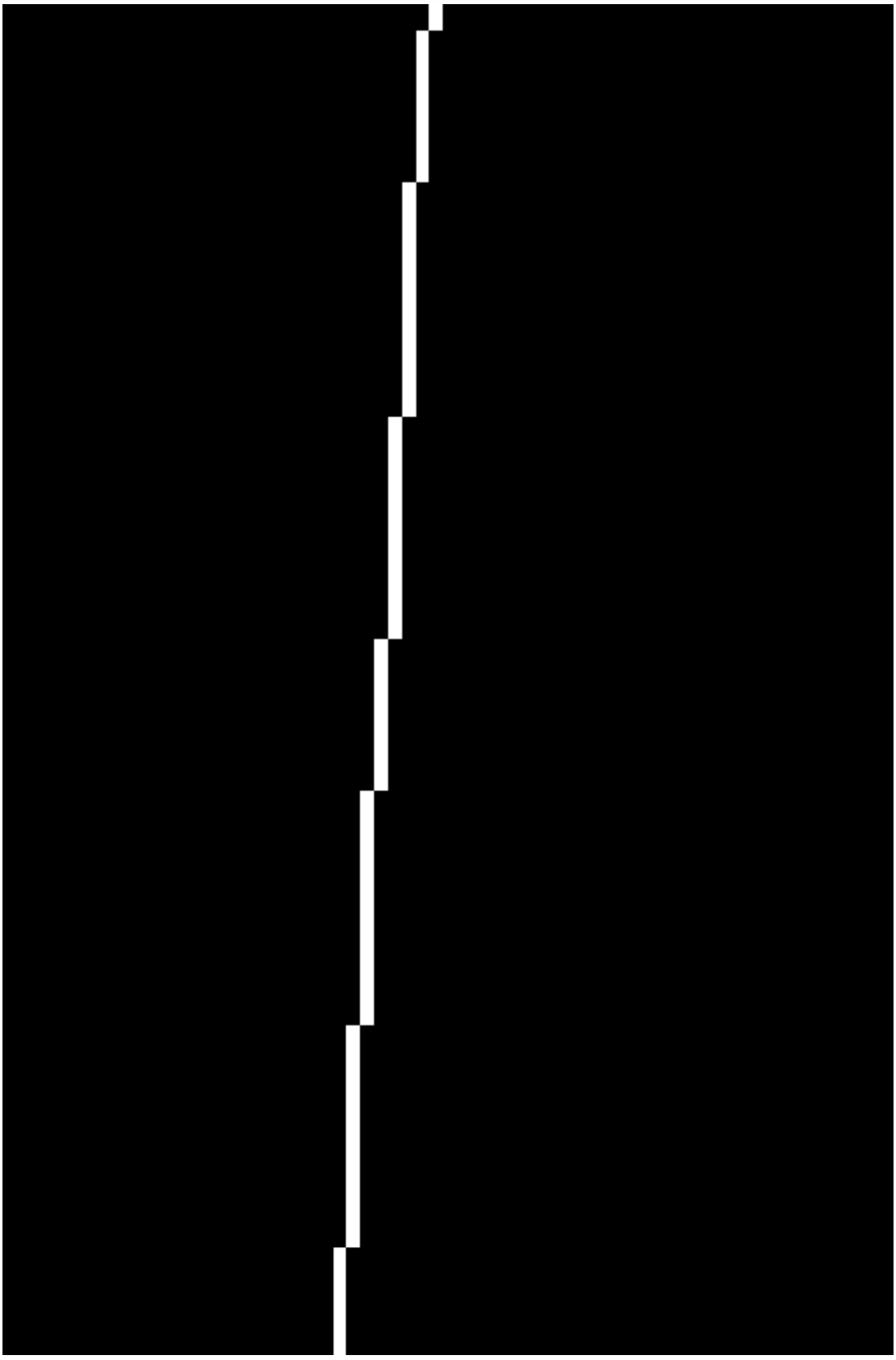
本次实验中，我设计了一个针对该问题的启发式的算法，取得了一定的效果。下面是算法的主要设计思路，以及实现上的一些说明。

设计思路

本次实验要提取的是经度线和纬度线，而其特点就是非常平滑，而且几乎都是连续的。观察其图像，可以发现，经纬线常常以以下的样式出现在图片中：



或者是这样：



除了经纬线最突出/凹陷的地方，其他地方的经纬度均是沿着两个固定的连续方向（i.e. 上图中纬线沿着西和西南方向）邻接。

而图片中的大陆线和其他的一些干扰点，则是没有该性质：曲线延展方向富有变化，十分尖锐。

因此，可以考虑从该性质出发，导出一个非常简单的算法：如果值为1的点可以被视为在一个只在2个固定方向延伸线的一部分，那么它就是经纬线上的一点，则应该被保留，否则就要被删除。

算法步骤

1. 顺序扫描原本图像中的每个点，如果该点为1，则进入步骤2，否则重复步骤1。
2. 从该点的八个方向中取出可以配对成经纬线延伸方向的两个方向（i.e. 西北和西），转入步骤3
3. 对该点以指定的两个方向进行深度优先搜索：也就是在深度优先搜索的过程中，每一步只能看到当前点以及当前点在那两个方向上的进邻的情况。如果最终存在任意一条路径，可以使得该深度优先搜索的深度达到指定数字（也就是指定某个数，如果在两个固定方向的延伸的长度大于该数字，则认为它是经纬线；可以称其为最小延伸长度），则保留该点，否则抛弃改点。之后返回步骤1。

实现说明

- 步骤1

```
for i = 1: height
    for j = 1: width
        if input_image(i, j) ~= 0
            output(i, j) = search(input_image, i, j, 55); % in face, 55,
the maxStep is a hyperparameter
        end
    end
end
```

这儿要说明的是，上文中提到的最小延伸长度，就是这儿的55，是该算法的一个超参数：对于有较平缓的大陆边界的图像，可以设置的大一点，对于大陆边界较陡的图像，可以设置的小一点；如果想要保留更多的经纬线，可以设置的小一点，如果要求对大陆边界去除彻底，可以设置的大一点。

- 步骤2

```
function out = search(image, m, n, maxStep)
    directions = [0, 1; -1, 1; -1, 0; -1, -1; 0, -1; 1, -1; 1, 0; 1, 1];
    out = 0;
    for x = 1: 8
        y = mod(x, 8) + 1;
        if dfs(image, m, n, [x, y], maxStep) == maxStep
            out = 1;
            break
        end
    end
```

`directions`是对于点的8个方向的邻接像素的位置变化量。`[x, y]`则是对于延伸方向的一个可能组合。显然，经纬线的延伸的两个固定方向一定是相邻的，所以`x, (x + 1) % + 1`是一个可能的方向组合。当搜索（搜索深度上限也设置成最大长度）最终深度等于最大长度时候，也就认为该点在一条经纬线上。

- 步骤3

```
function length = dfs(im, a, b, dirs, last)
    length = 0;
    if last ~= 1
```

```
        for index= 1: 2
            next_a = a + directions(dirs(index), 1);
            next_b = b + directions(dirs(index), 2);
            if next_a < 1 || next_b < 1 || next_a > height || next_b >
width
                continue
            elseif im(next_a, next_b) ~= 0
                length = max(dfs(im, next_a, next_b, dirs, last - 1),
length);
            end
        end
    end
    length = length + 1;
end
```

中间一小段是为了防止越界。其余部分就是最经典的深度优先搜索框架。

实验结果

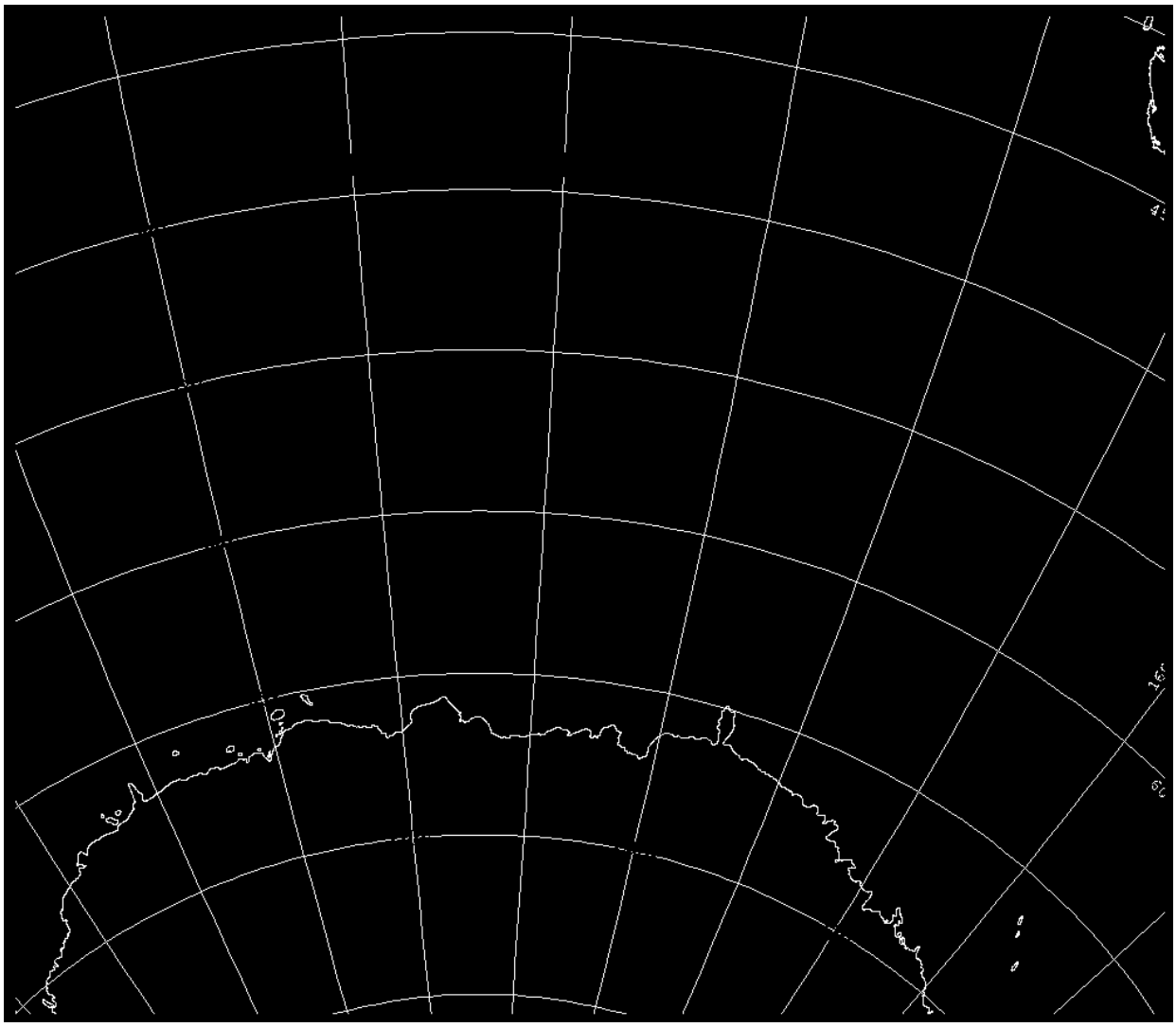
效果

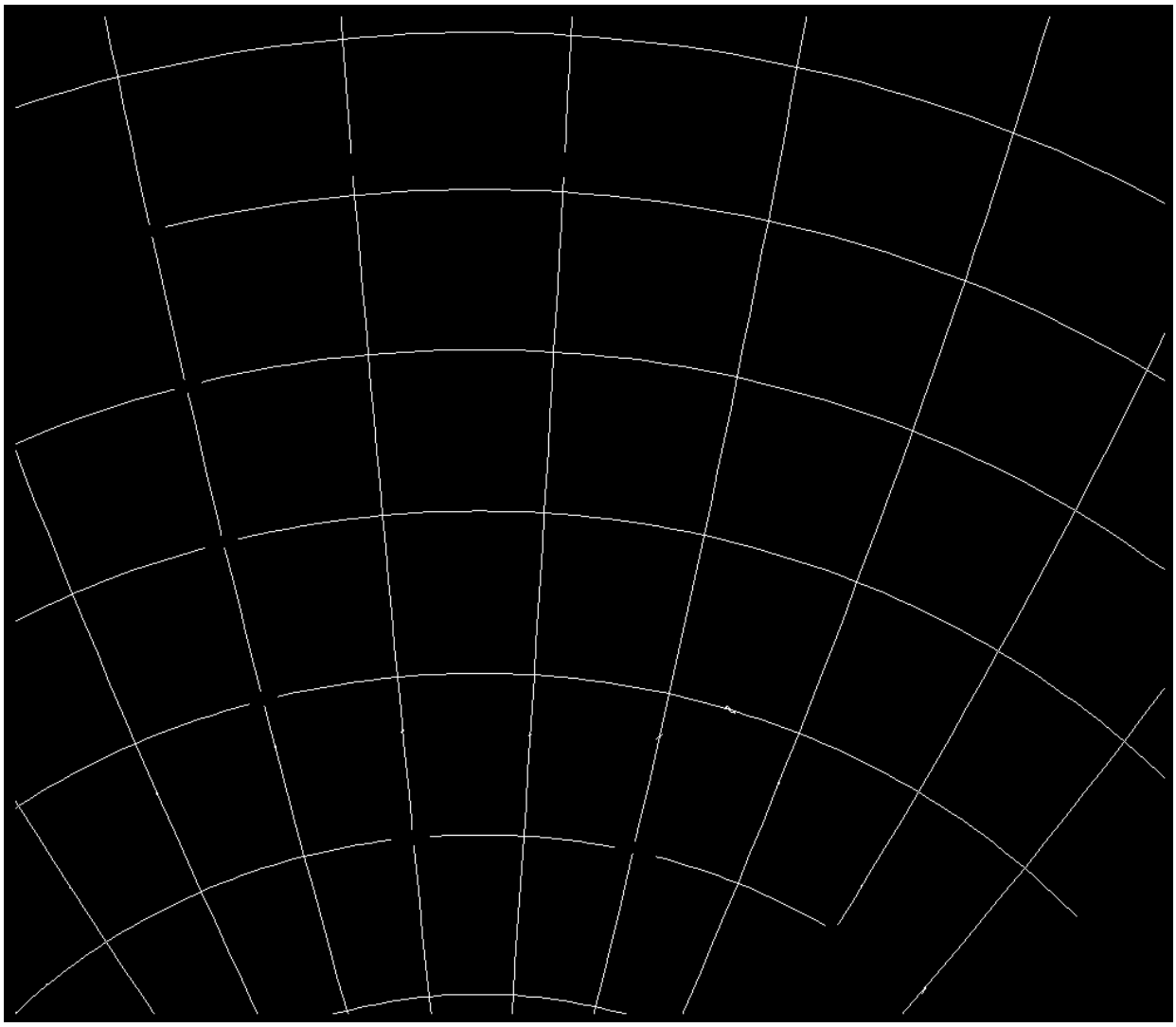
对于第一张图片，使用超参数55，用助教提供的scores函数计算结果如下：

- 总分：11278
- 经纬线召回率：97.49%
- 大陆线和小岛消除率：98.97%

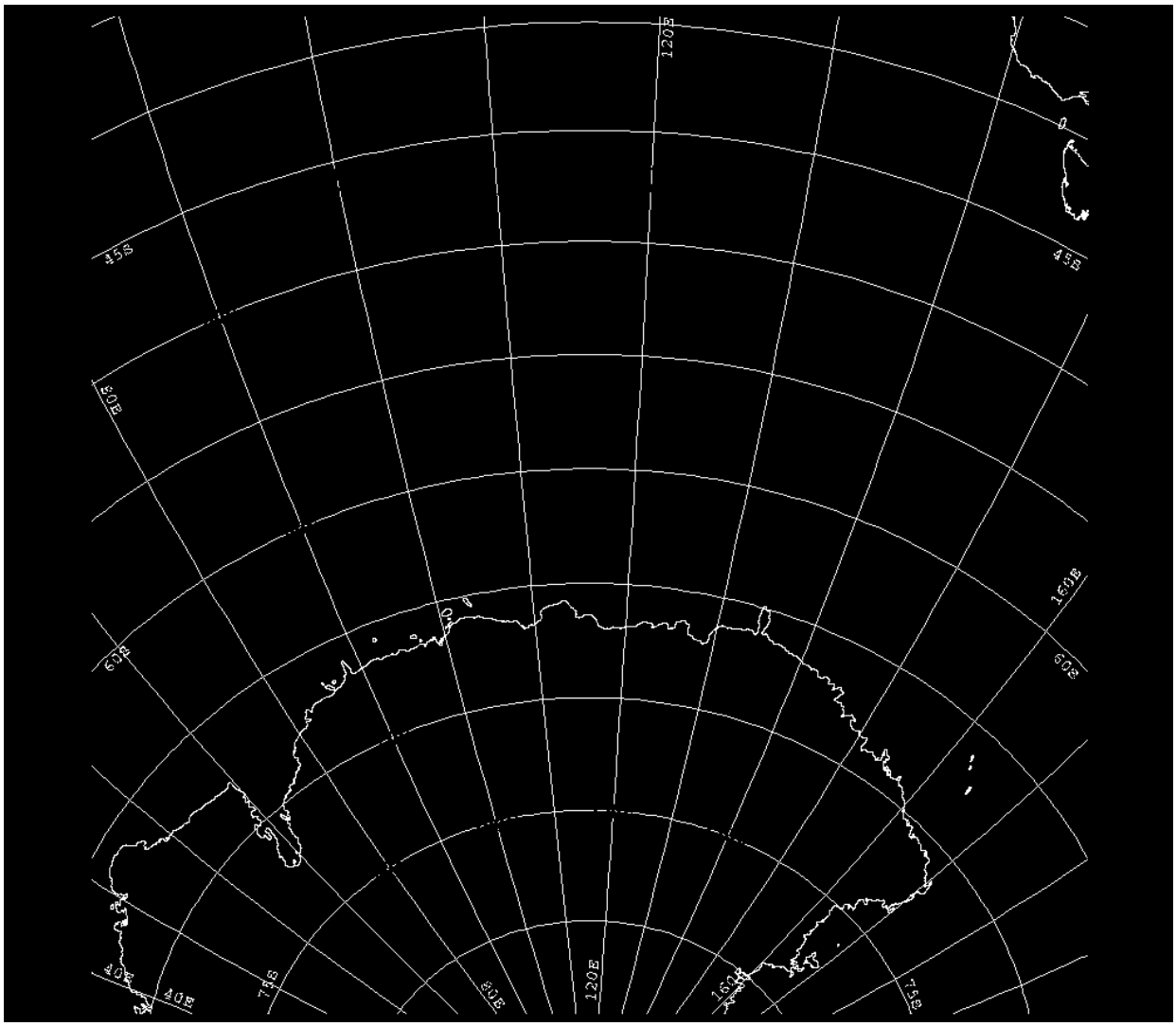
预览各张图片（均为原图+变换后图）结果如下：

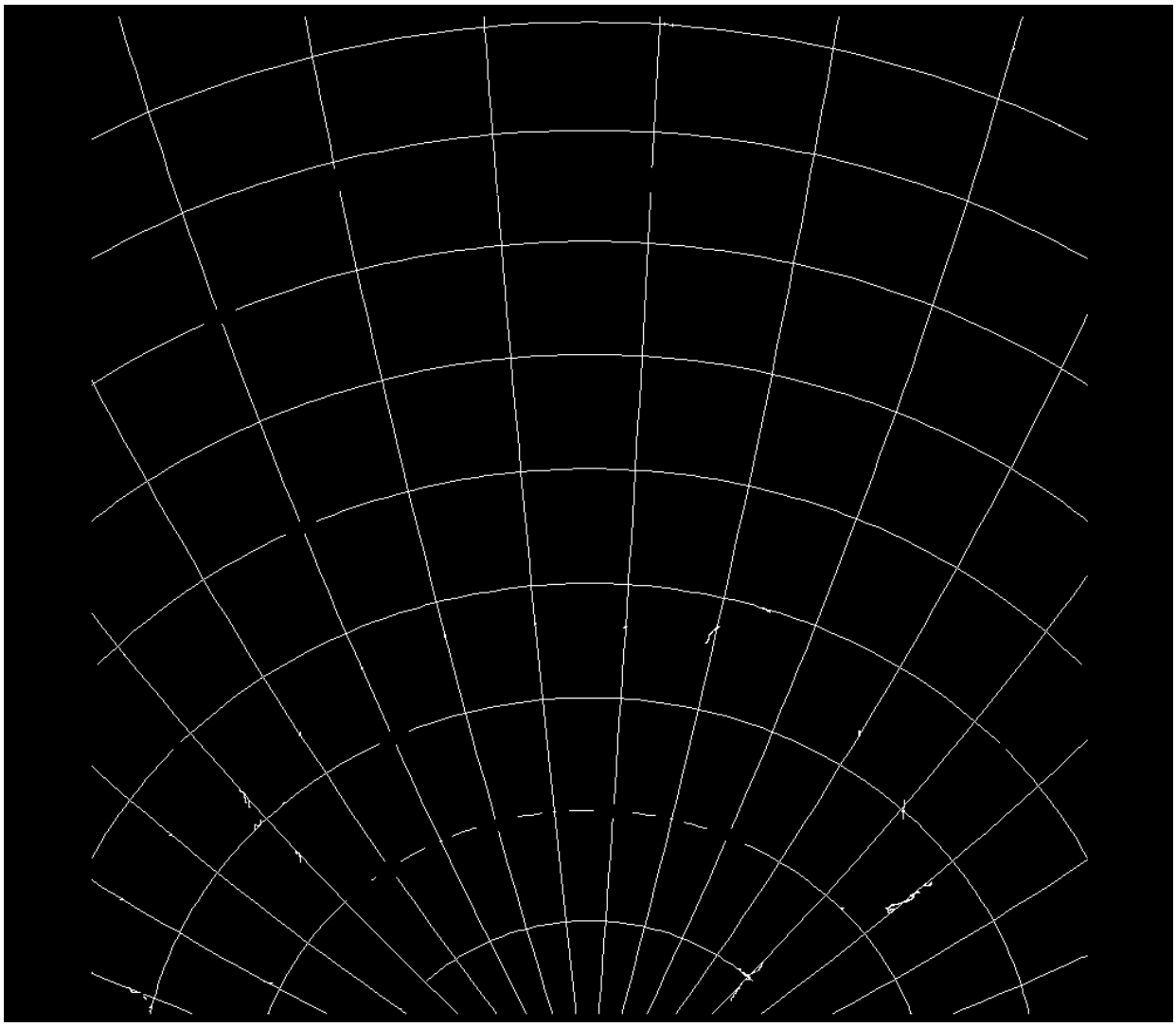
- demo图片（input.png，超参数设置为55）



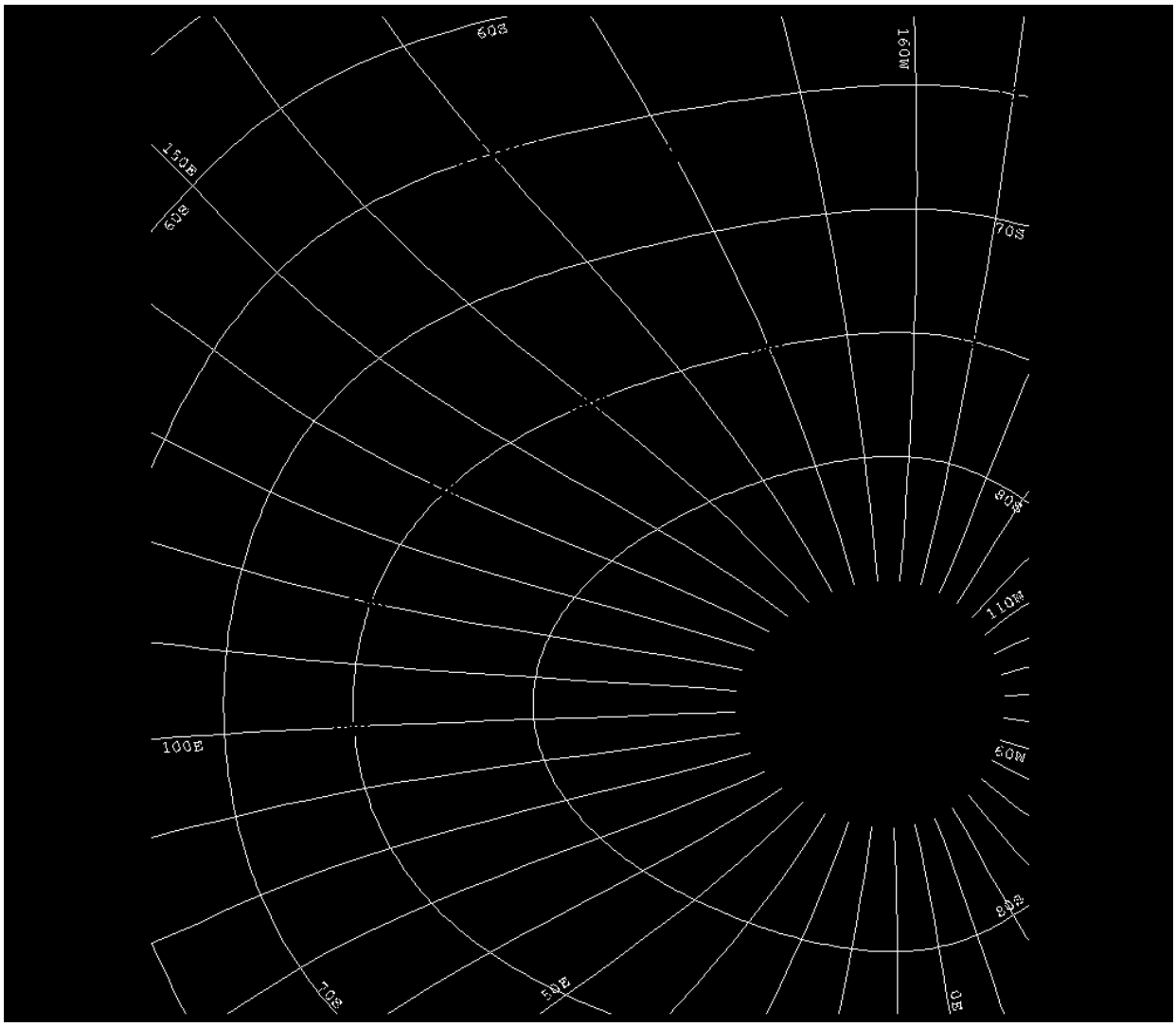


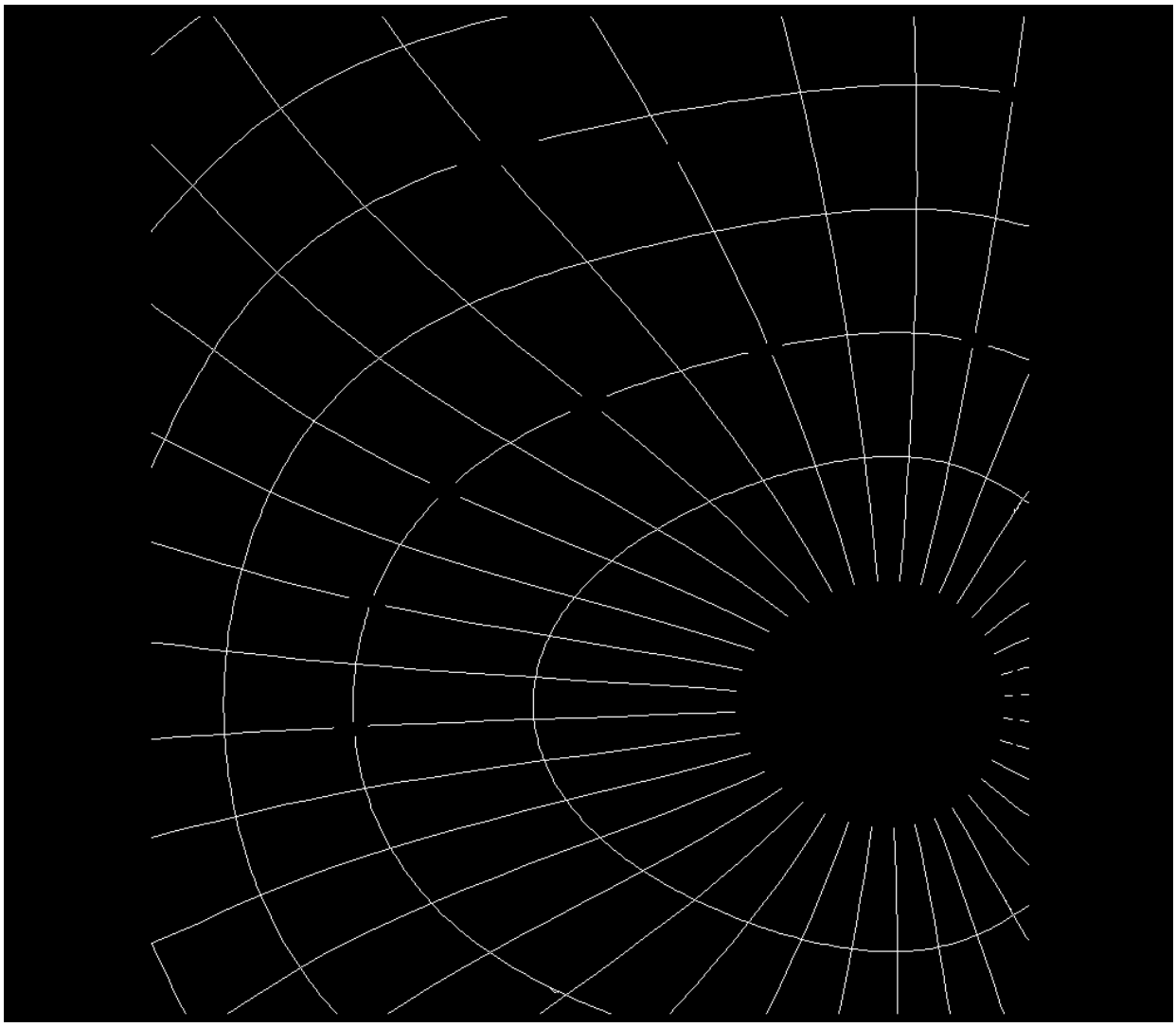
- 1.png （超参数设置为45）


120E
45E
60E
150E
60E
75E
30E
120E
150W
75E
40E
40E

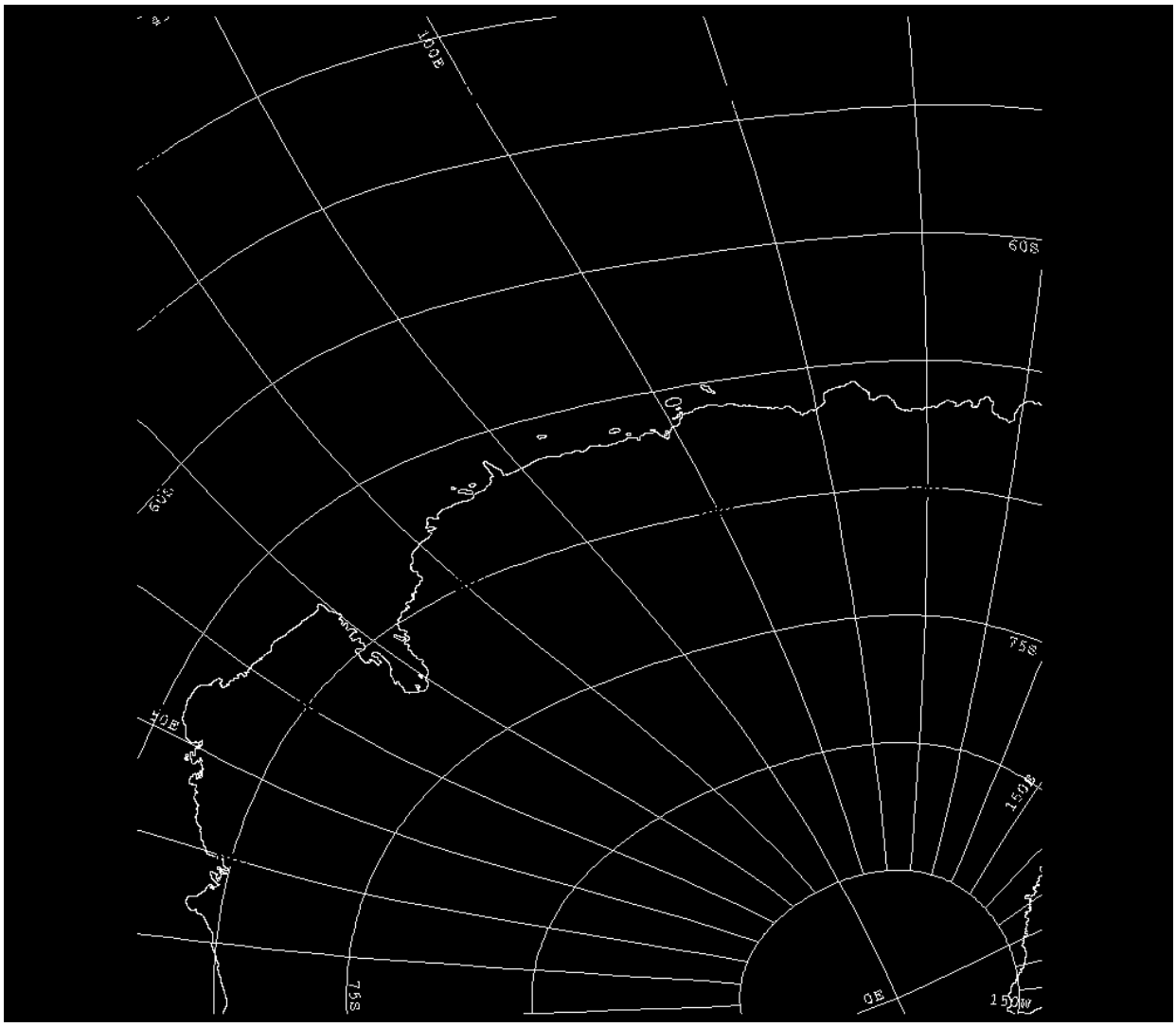


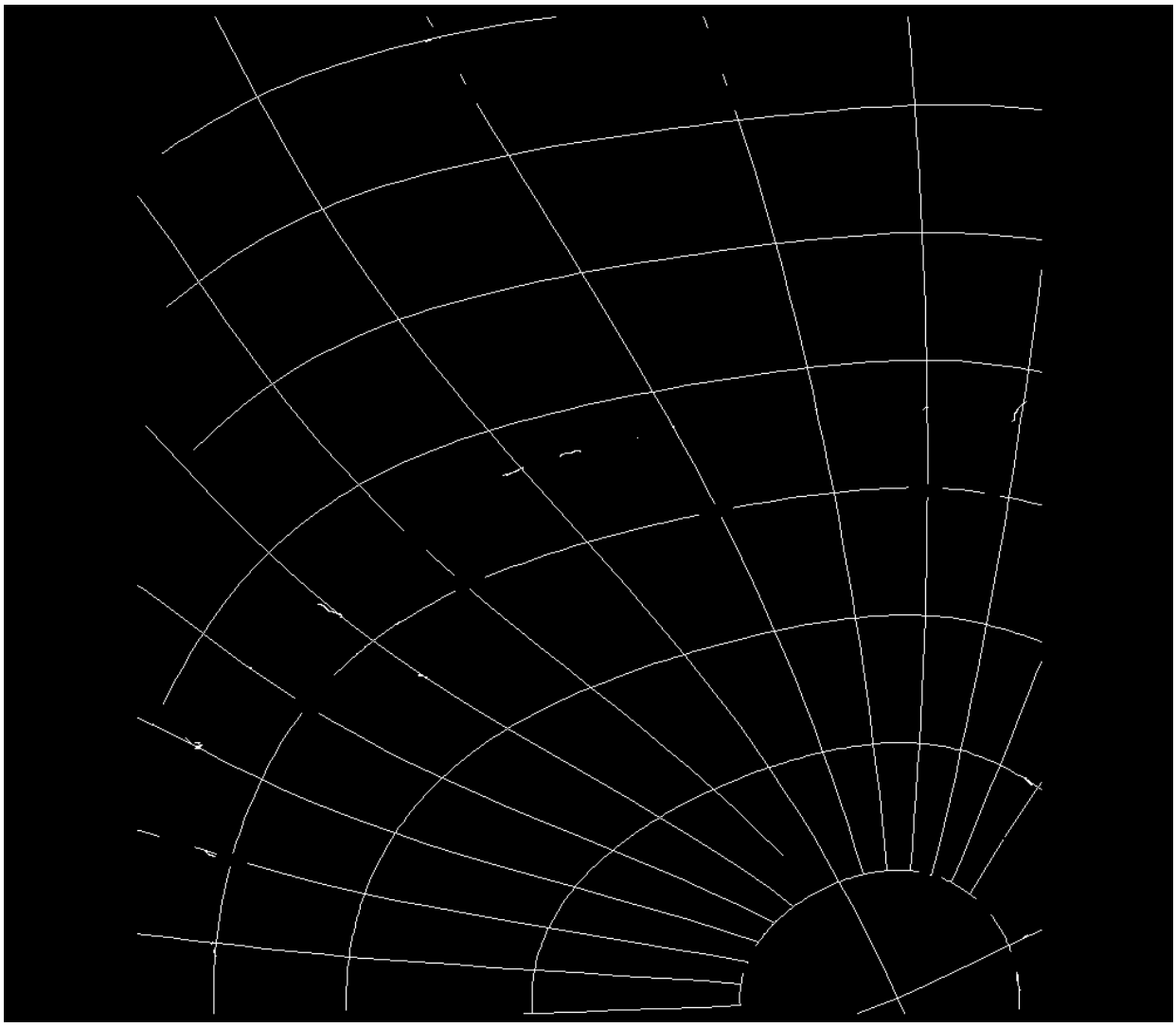
- 2.png（超参数设置为15）


60S
150E
80S
160W
70S
90W
110W
100E
90W
80S
70E
50E
0E

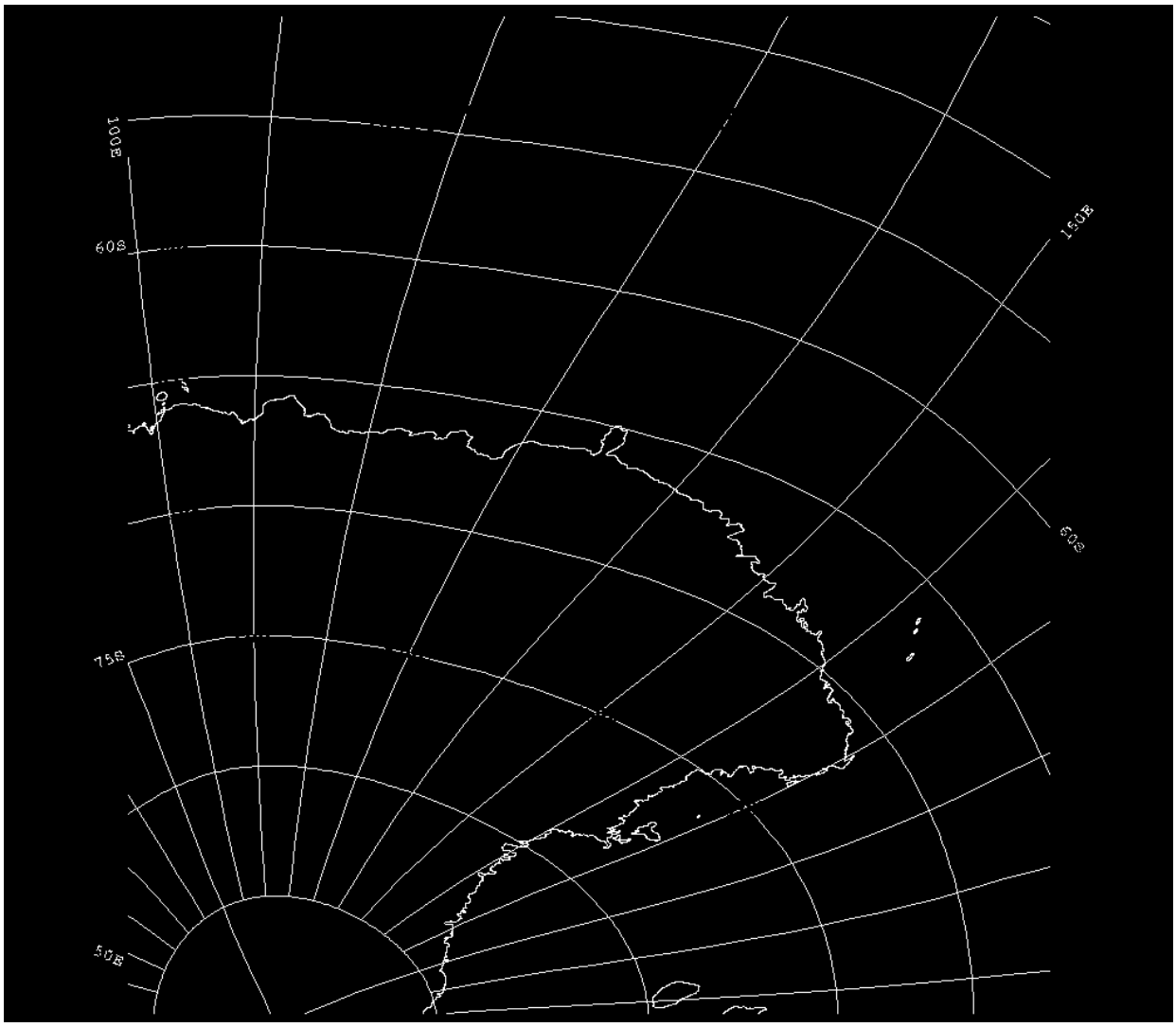


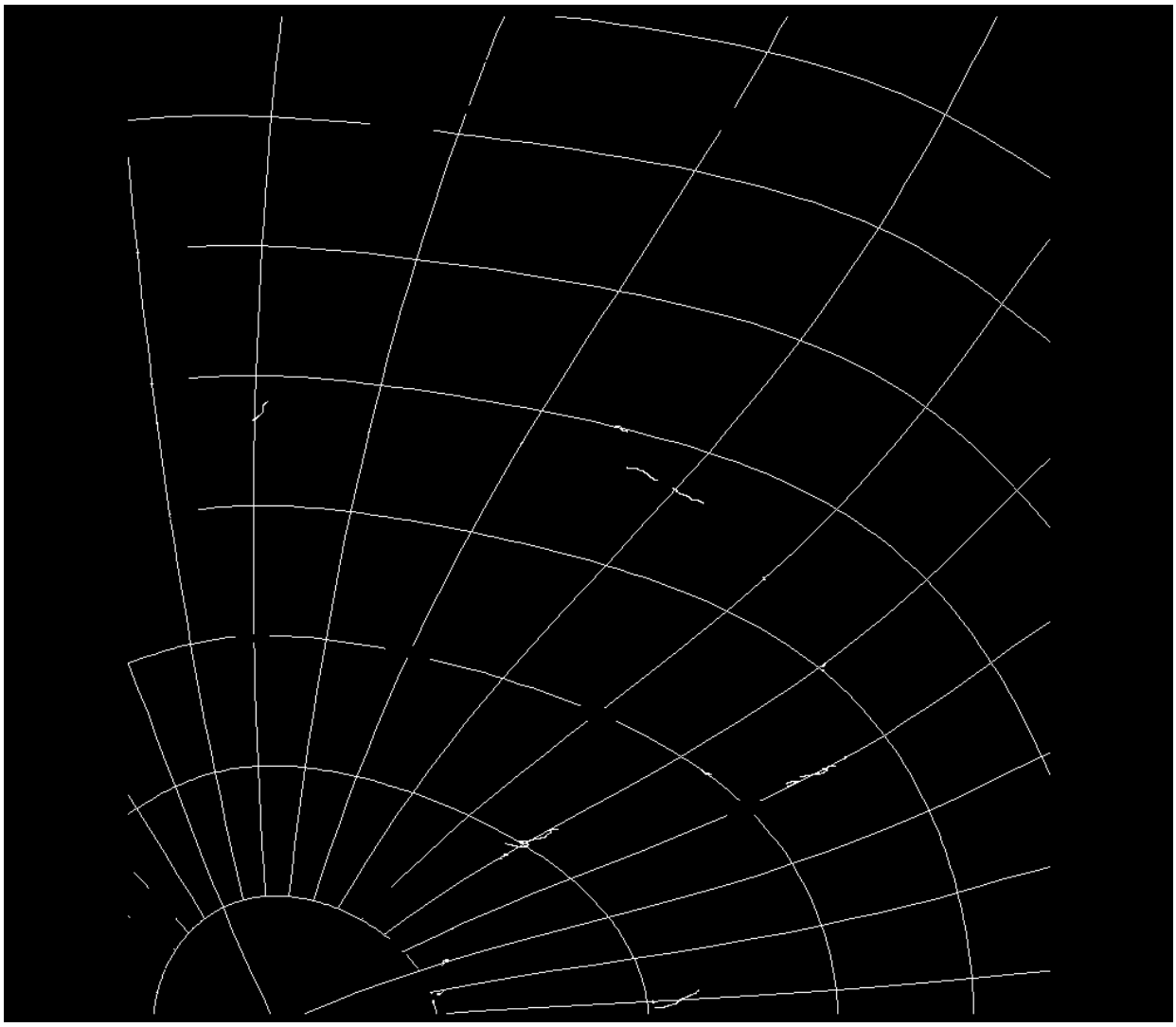
- 3.png （超参数设置为50）


120E
60S
60S
75S
30E
150E
75S
0E
150W

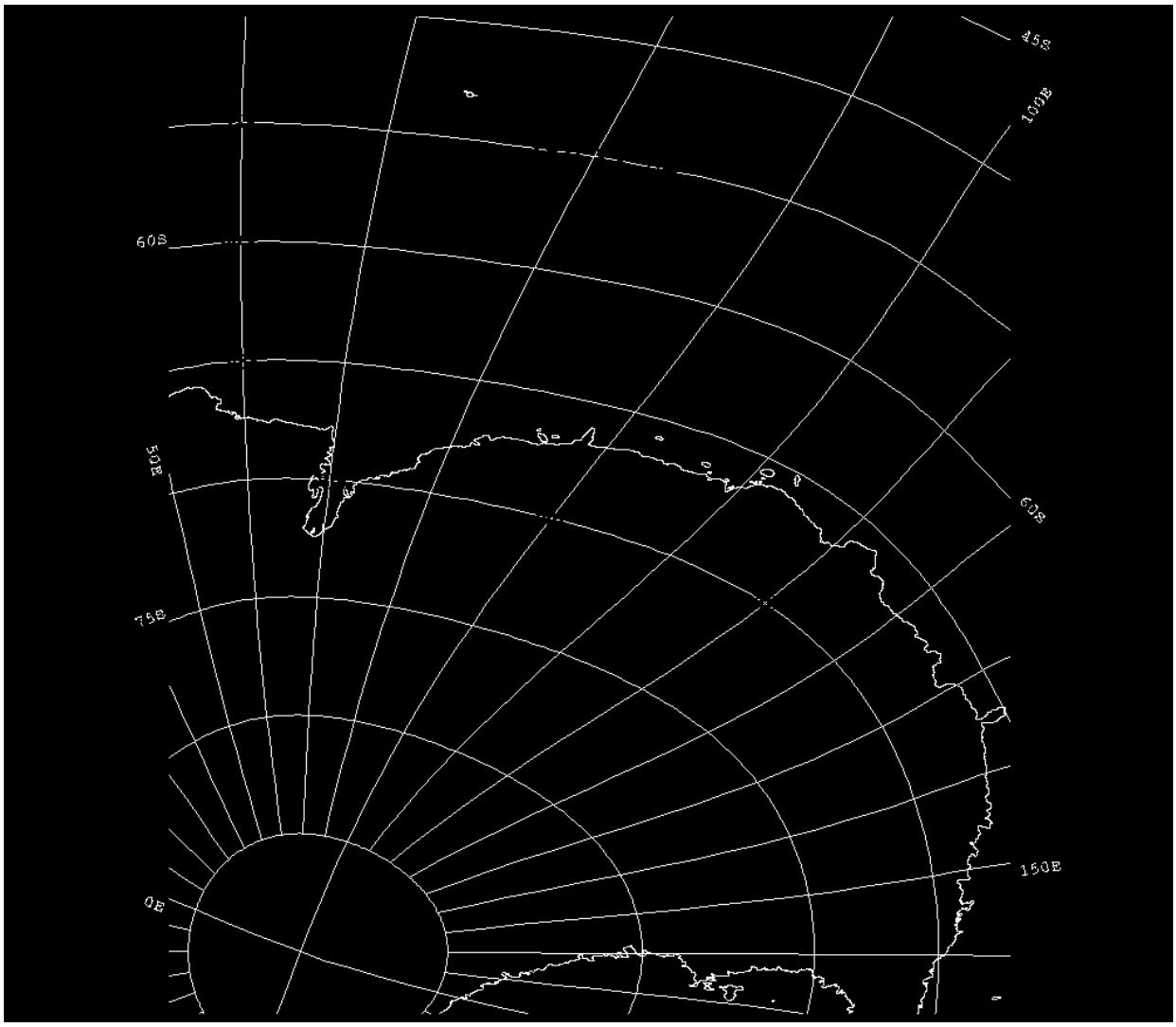


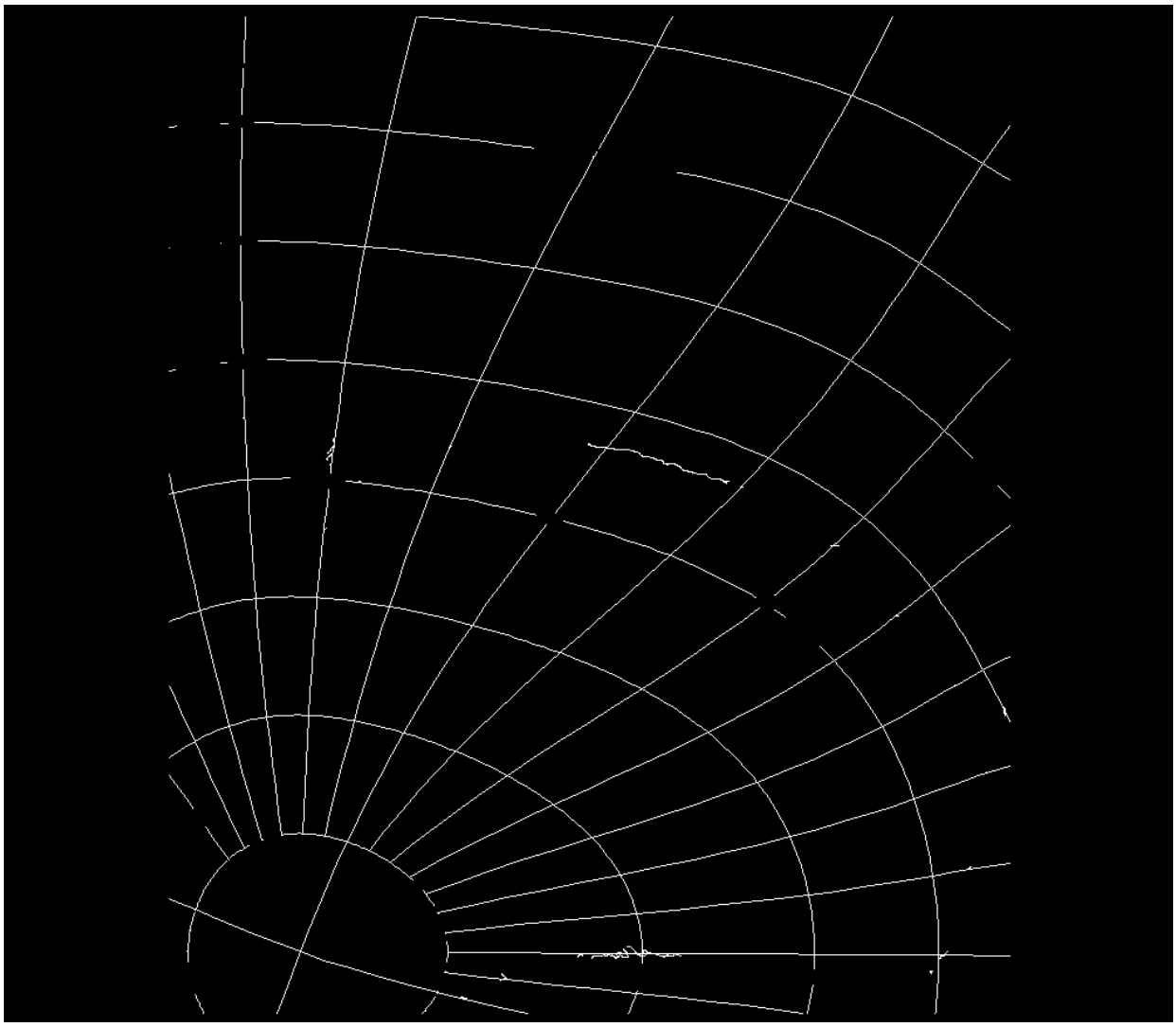
- 4.png （超参数设置为40）


100E
60S
150E
60E
75S
50S



- 5.png（超参数设置为45）


45E
105E
60S
150E
0E
75S
90E



效率

采用以下代码对效率进行评估。

```
im = im2bw(imread('input.png'));
times = [];
for i=1:100
    start = tic;
    t = my_imageprocessing(t);
    elapsed = toc(start);
    times(i) = elapsed;
end
disp(mean(times));
disp(std(times));
```

最终可以得出，处理一张图片的平均耗时为0.1774秒，标准差为0.0059秒。

可以看出，该算法有极高的运行效率。

一些尝试

原本希望通过双阈值算法，来改进性能：比如设置高阈值为55，低阈值为25，然后使用类似`canny`算法的方式将其综合起来，不过最终运算时间翻倍，而性能却几乎不见提升。可能有以下原因：

- 任务目标不同
- 任务原理不同

实验结论

优点

- 实现非常简单，非注释代码只有不到40行
- 运行效率极高
- 对于陡峭大陆线清除效果良好

缺点

- 需要进行参数选择，不能完全的自动化
- 对于平滑大陆线无能为力
- 对于小范围间断点会直接擦除