

Problem 1

In this problem, we use different transform methods to change images' properties, like position, angle, distortion, etc. All the transform methods can be generalized by the following steps:

- ① Find the inverse matrix of the transformation
- ② For each pixel of the output image, use the inverse matrix to find the corresponding pixel in the input image
- ③ If the index of the input pixel location is not an integer, then we use the bilinear interpolation to calculate the value, which should be passed to the output pixel.

For the sub problem a-c, we should find inverse matrixes separately for geometric modification, special warping and lens distortion correction.

a Geometric transformation

- Find the 4 corners pixel location for the each patch in "lighthouse1.raw", "lighthouse2.raw" and "lighthouse3.raw".
- Basically, the corner pixels have the property that the x-index and y-index should be minimum or maximum of the non-white region.
- Because the patches are rotated version of the original one, there will be some blur effects. We can see these effects clearly at the edges, like **Figure 1-1**.

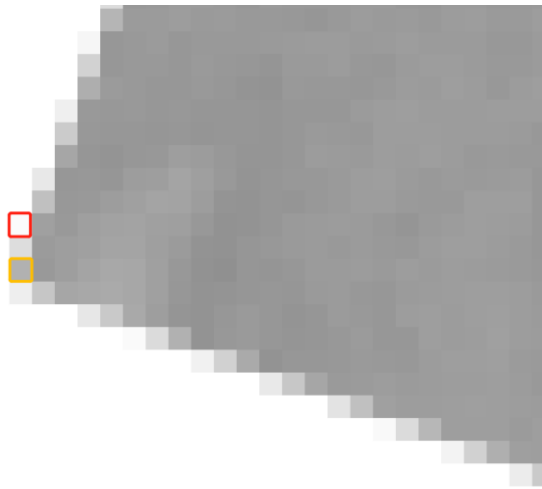


Figure 1-1: zoom in the "lighthouse1.raw"

- In the Figure above, we can know that for each corner, there are more than one candidate corner pixel because of the blur effect.
- Intuitively, we should let the program set the yellow frame pixel to be the corner pixel rather than the red frame pixel in **Figure 1-1**.
- Therefore, I choose the darkest pixel among the candidate pixels.
- Then, we need to find the 4 corners of each hole in the "lighthouse.raw"
- For each hole, we first find the up-left corner of each hole, and then find the other 3 corners left.
- In **Figure 1-2**, we can find that in order to locate the 3 up-left corner, we should find the pattern of **Figure 1-3**. (In the specific case here, we do not need to worry about that pixel outside but near the up-left corner of the hole has the value of 255.)

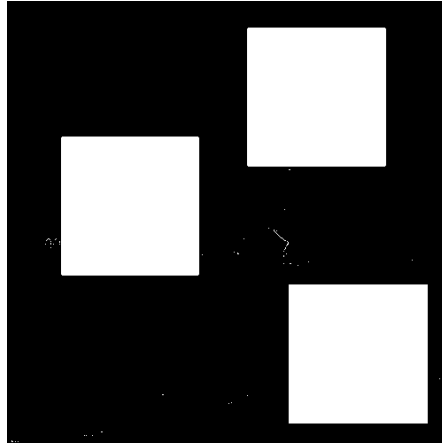


Figure 1-2: Value=255 pixels(white pixel) in "lighthouse.raw"

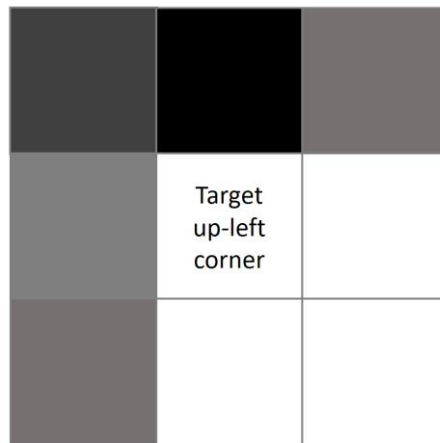
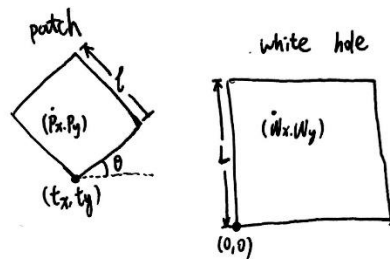


Figure 1-3: Pattern around up-left corner

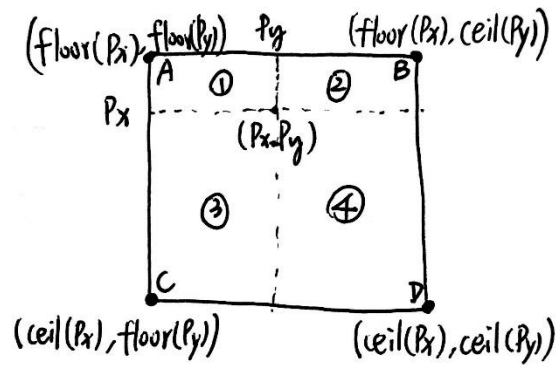
- Then, we should calculate the inverse matrix from "lighthouse.raw" to "lighthouse1(2,3).raw":
First, we decide which corner of the patch should go the Left-down side of the white hole



$$S = \frac{l}{L}$$

$$\begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = \begin{pmatrix} S \cdot \cos \theta & -S \sin \theta & t_x \\ S \sin \theta & S \cos \theta & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} W_x \\ W_y \\ 1 \end{pmatrix}$$

- The (Px,Py) in the input image may be double variable, so we should calculate the pixel value using bilinear interpolation:



Value go to the output image = (Value of A) * (Area of ④) + (Value of B) * (Area of ③) + (Value of C) * (Area of ②) + (Value of D) * (Area of ①)

- **Output image**



Figure 1-4: undesirable output image

We can find many undesirable effects:

- 1) Part "b" seems like a little bit tilted
- 2) Each part has obvious white frame

These effects are due to the inaccuracy of the corner location of the patches.

- **Improve**

Add the condition: corner pixel value must be lower than 245.

Zoom the patches: 1px larger on each side

Improved output version 1:



Figure 1-5: improved once

The output is much more desirable.

Keep improving:

Detect the white frame at the edge of each hole, and use their neighbor to correct these white lines.

Adjust the zoom parameter slightly.

Improved output version 2:



Figure 1-6: improved second time

b Spatial Warping

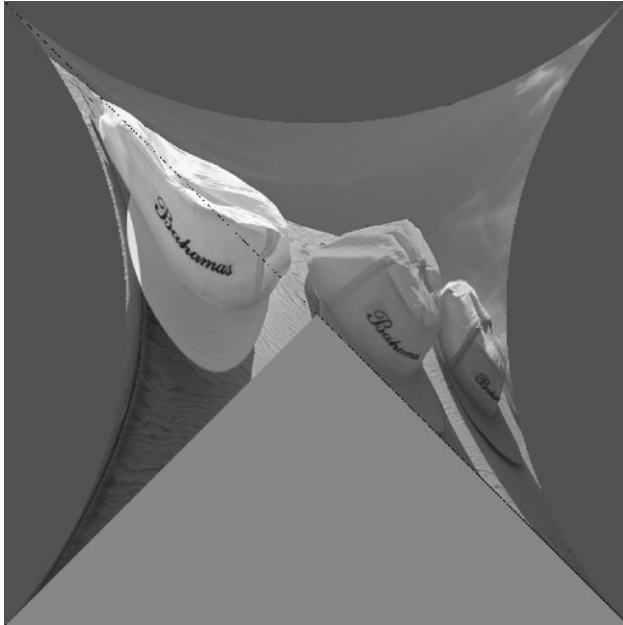
- In the picture shown in the HW description, the height of the arc could not be 128.
If we want to obtain the same effect of the wrapped image shown in the HW description, the height of each

arc should be at most 106.

Therefore, here I choose **100 as the height of the arc**.

【didn't finish】

Output:



Use the **Gaussian Elimination** method to solve the equation set:

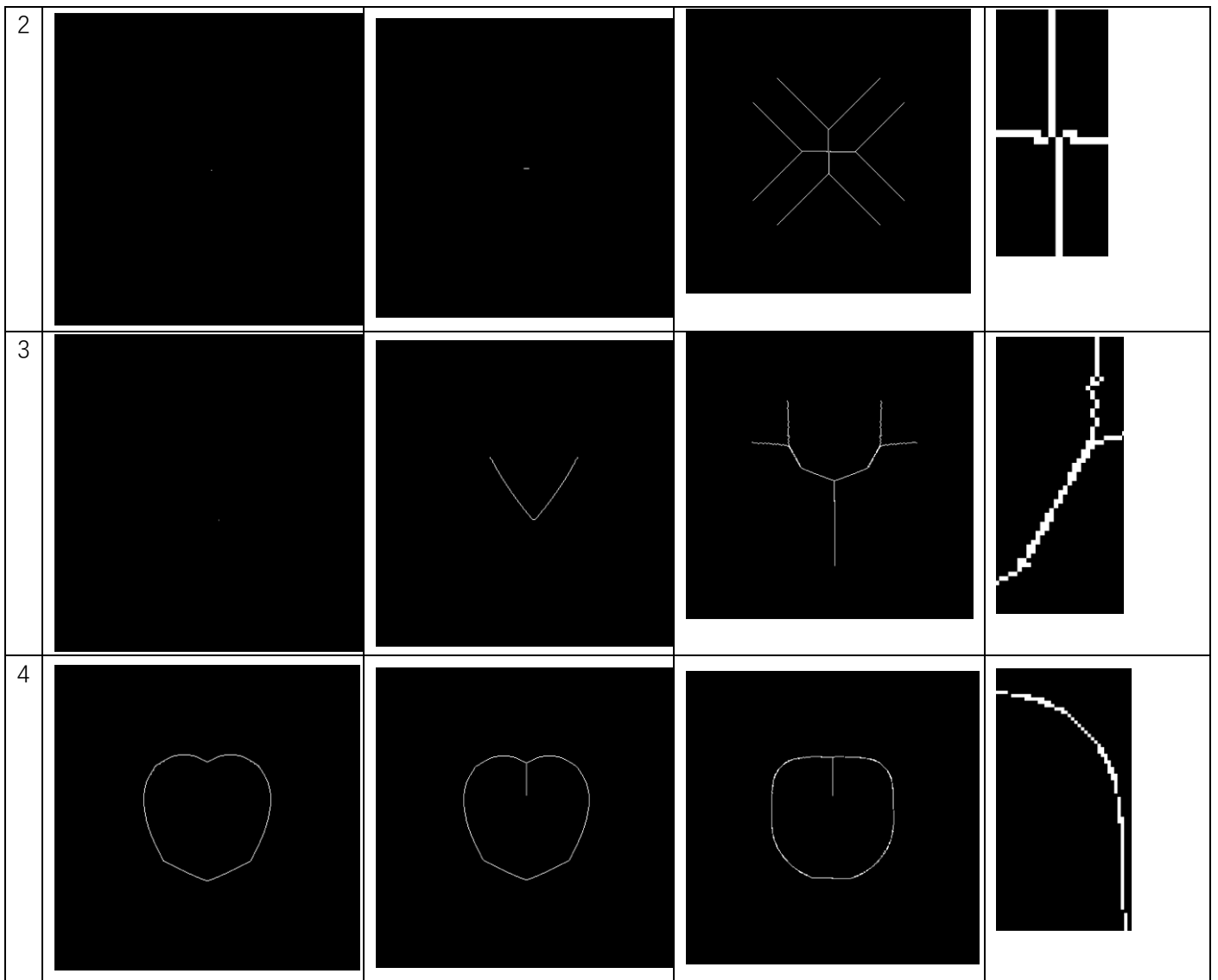
$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \\ b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \end{bmatrix} \begin{bmatrix} 1 \\ x \\ y \\ x^2 \\ xy \\ y^2 \end{bmatrix}$$

c Lens Distortion Correction

Problem 2

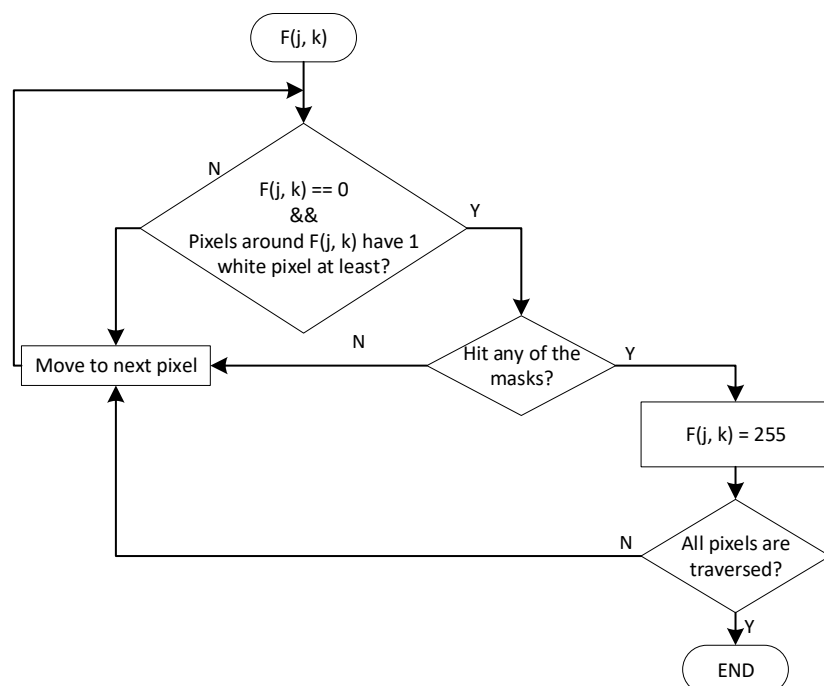
In this problem, we use morphological operations to process images in order to get different effect, and realize different process purposes like defecting defection, counting objects and so on.

Basic flow chart:

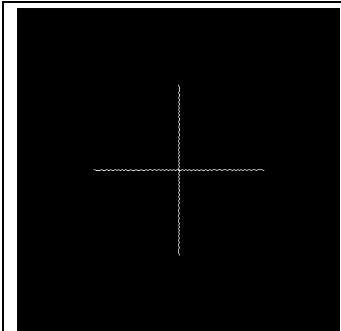
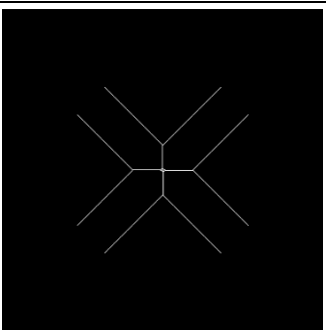
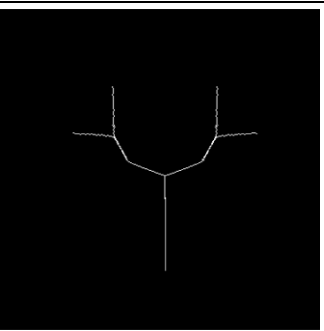
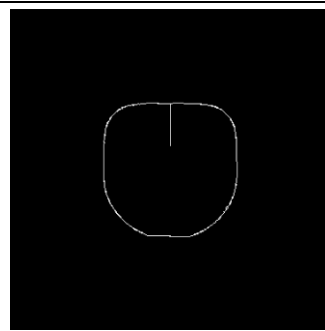
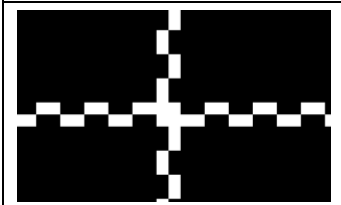
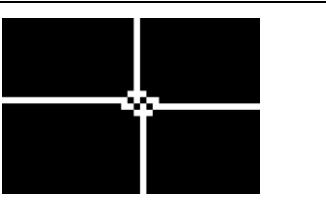
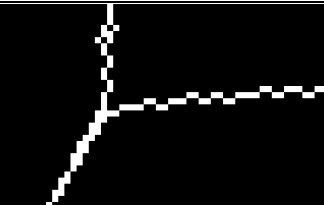
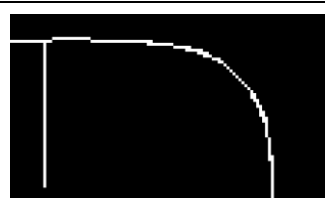


Skeletonizing can let us distinguish the different patterns very clearly, because it still contains the basic location information of the original patterns. However, if only go through the Stage one and Stage two, the skeletonizing output will have some break point along the skeleton line. This can be corrected by using the bridging process.

Bridging process flow chart:



After bridging process, the output and details are shown below:

			
			
No change	Add some points, but seems like no necessary		Connect some broken lines

Shrinking and Thinning's output have some connections: Shrinking's output can be achieved after the Thinning process.

b defect detection and correction

Apply the shrinking process to the "deer.raw", and get the output as following:

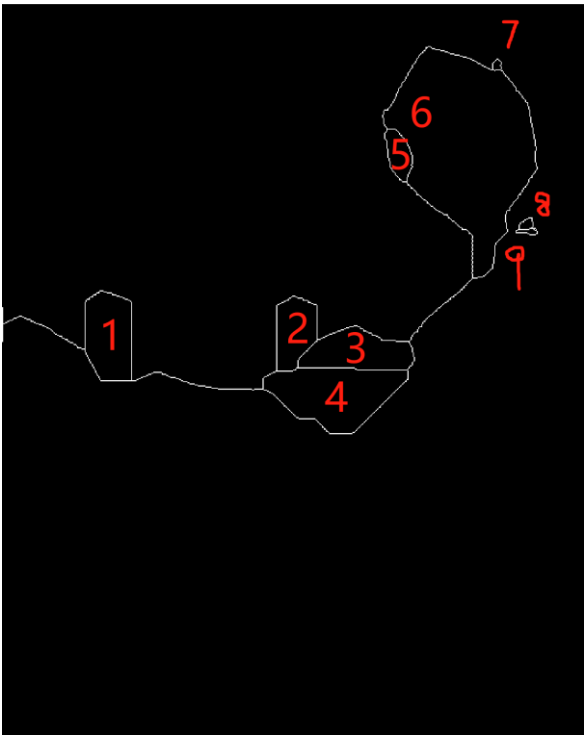


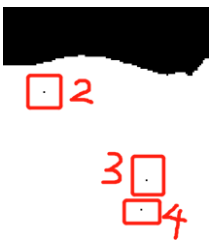
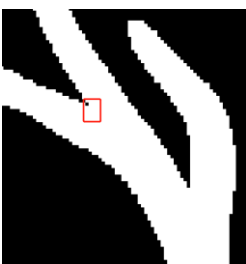

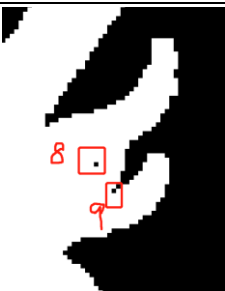


Figure 2-1: shrunked "deer.raw"

The holes in the shrinking image are defection region candidates.
The defect region has common feature: the hole's edge has vertical straight line or horizontal straight line. So, we can see that hole #1, 2, 3, 4, 7, 8, 9 are defect regions;
Go back to the original image to confirm:

#		#	
---	--	---	--

1 (defect)		6	
2 (defect) 3 (defect)		7 (defect)	
5		8 (defect) 9 (defect)	

In order to correct the defect, we can use another filter mask set:

A	255	B
255	0	255
C	255	D

(A~D: at least have three 255)

The corrected output is:

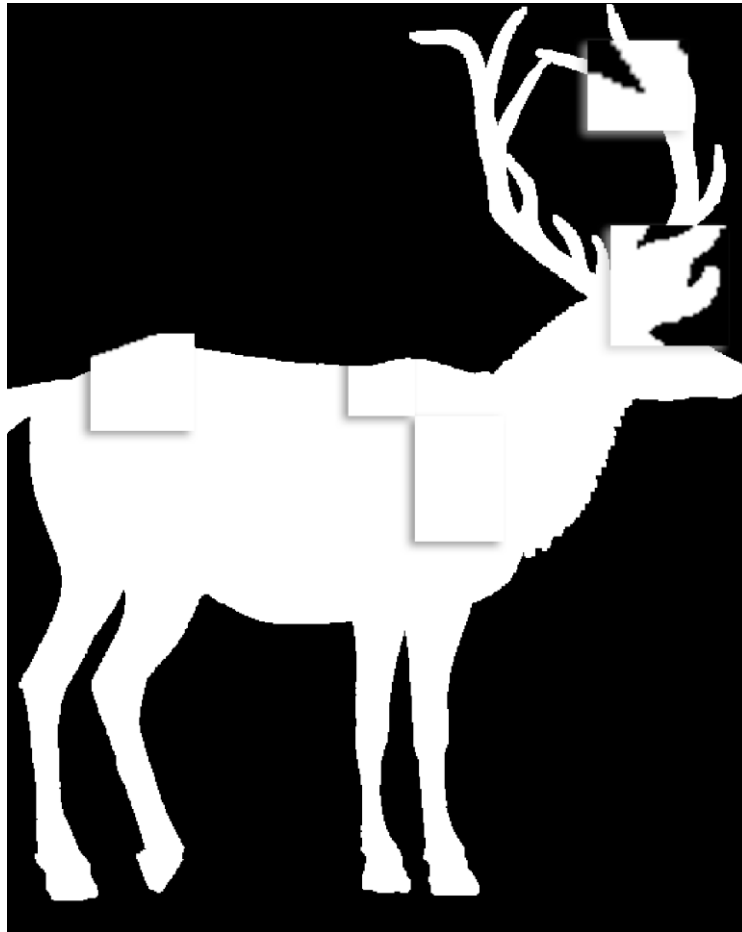


Figure 2-2: corrected “deer.raw”

Put the corrected image into the shrinking process, and we get:

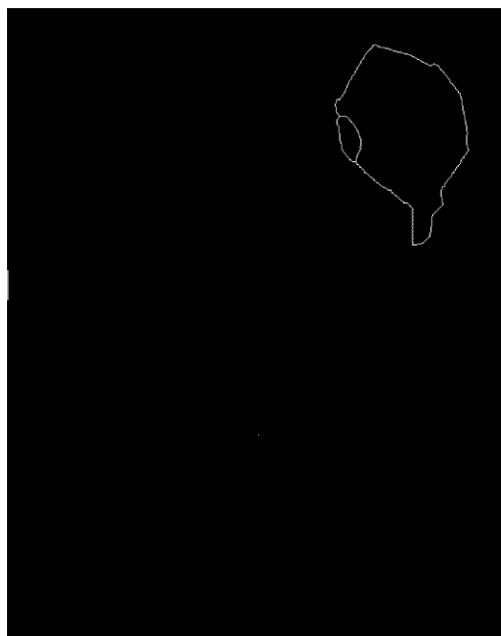


Figure 2-3: shrunk corrected “deer.raw”

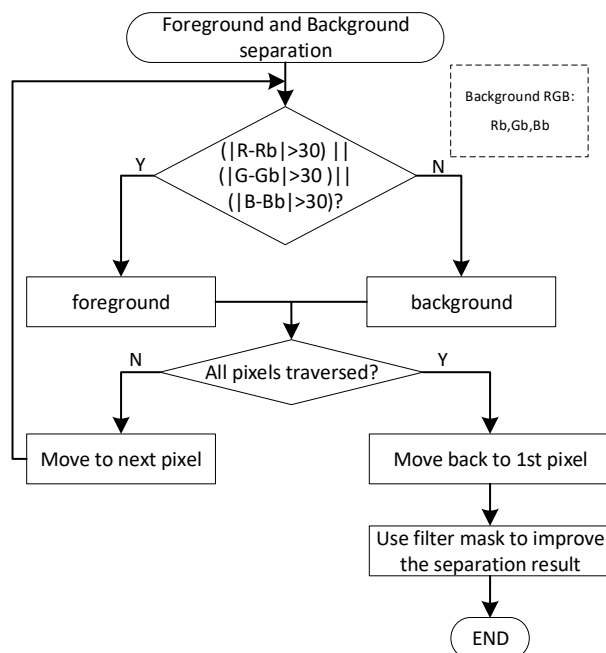
The defect holes are all gone.

c Object Analysis

Background color's RGB value:

R:	61
G:	76
B:	83

First, we should separate the foreground and the background.

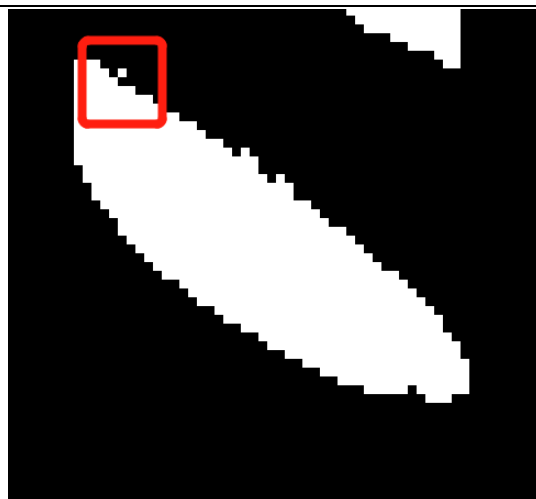
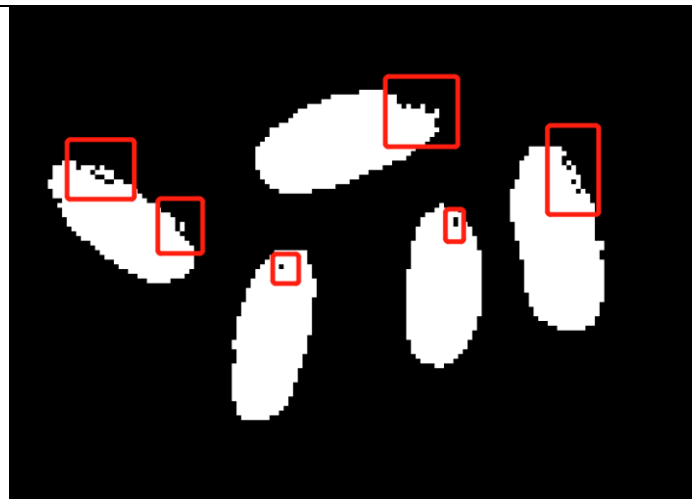


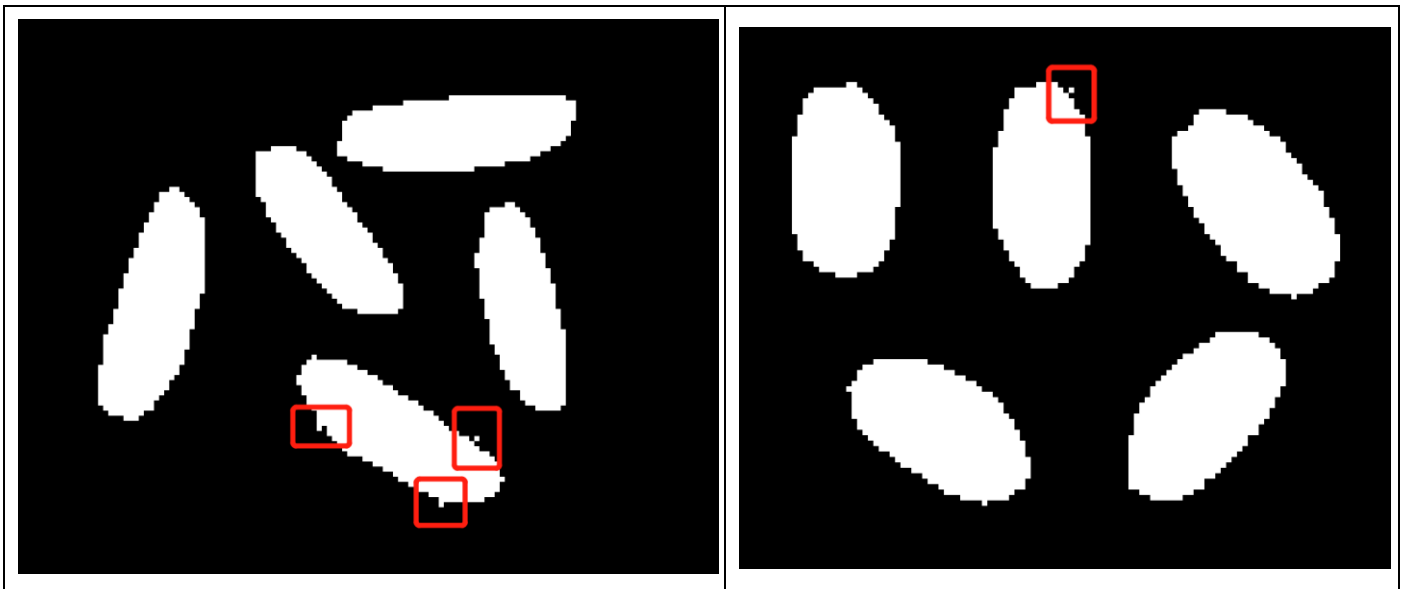
Before the edge improvement, the output is as following:



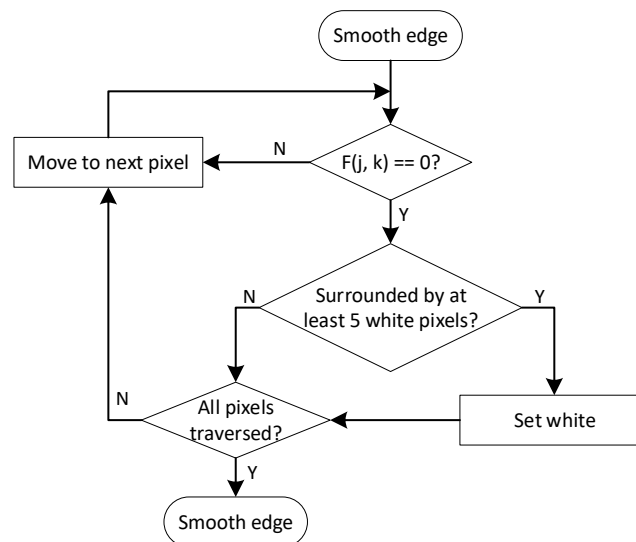
Figure 2-4: black and white “rice.raw” before smoothing the edge

Some of undesirable effects





We want each rice kernel to have smooth edge. Here are the detailed improvement process (for $0 \rightarrow 255$, $255 \rightarrow 0$ process is similar):



The smoothed edge output is shown below:



Figure 2-5: black and white “rice.raw” after smoothing the edge

After shrink process, each kernel becomes one white dot:

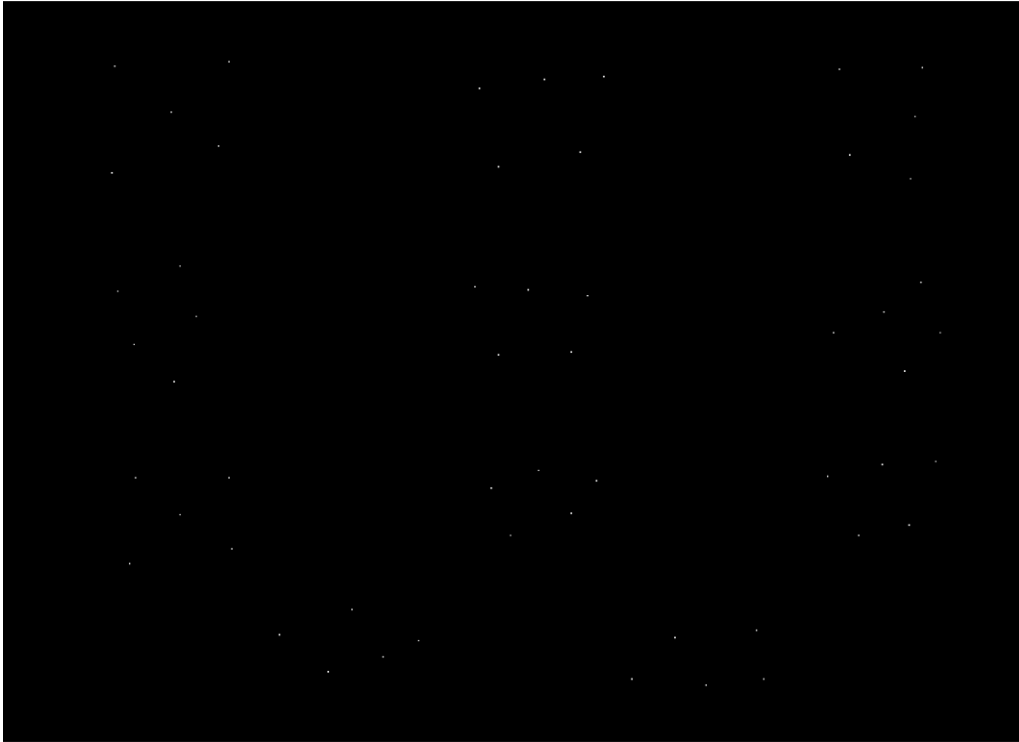


Figure 2-6: shrunk black and white “rice.raw” after smoothing the edge

We can let the program divide the Figure 2-5 into several parts, record the upper-left and lower-right corner of each region. The red line in the Figure 2-7 is a demonstration of the region edge. All these red lines are black pixel in the Figure 2-5.

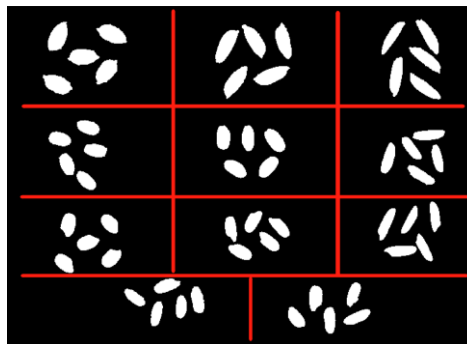


Figure 2-7: divide the rice pile

At each region, we can count the number of white pixels(total size), and the number of kernels. (total size)/# of each pile = average size of each kernel.

```
1-11, from small to large
1: 503 pixels; Pile: 10
2: 514 pixels; Pile: 8
3: 529 pixels; Pile: 4
4: 531 pixels; Pile: 11
5: 533 pixels; Pile: 9
6: 545 pixels; Pile: 7
7: 550 pixels; Pile: 6
8: 605 pixels; Pile: 5
9: 742 pixels; Pile: 3
10: 832 pixels; Pile: 1
11: 878 pixels; Pile: 2
```

Figure 2-8: one kernel's size ranking



Figure 2-8: pile #