Fingerprint Extraction and Restoration

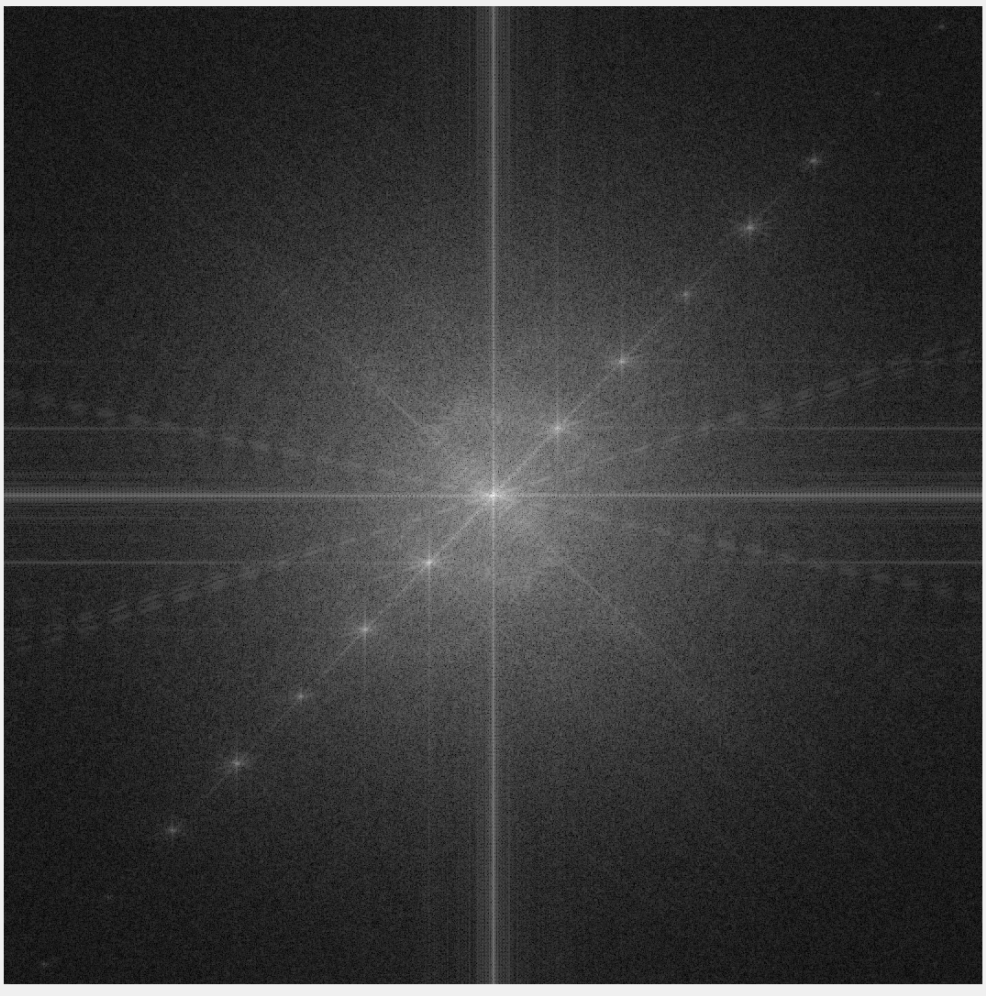
Yin Liu **--Lab Report**

Background：

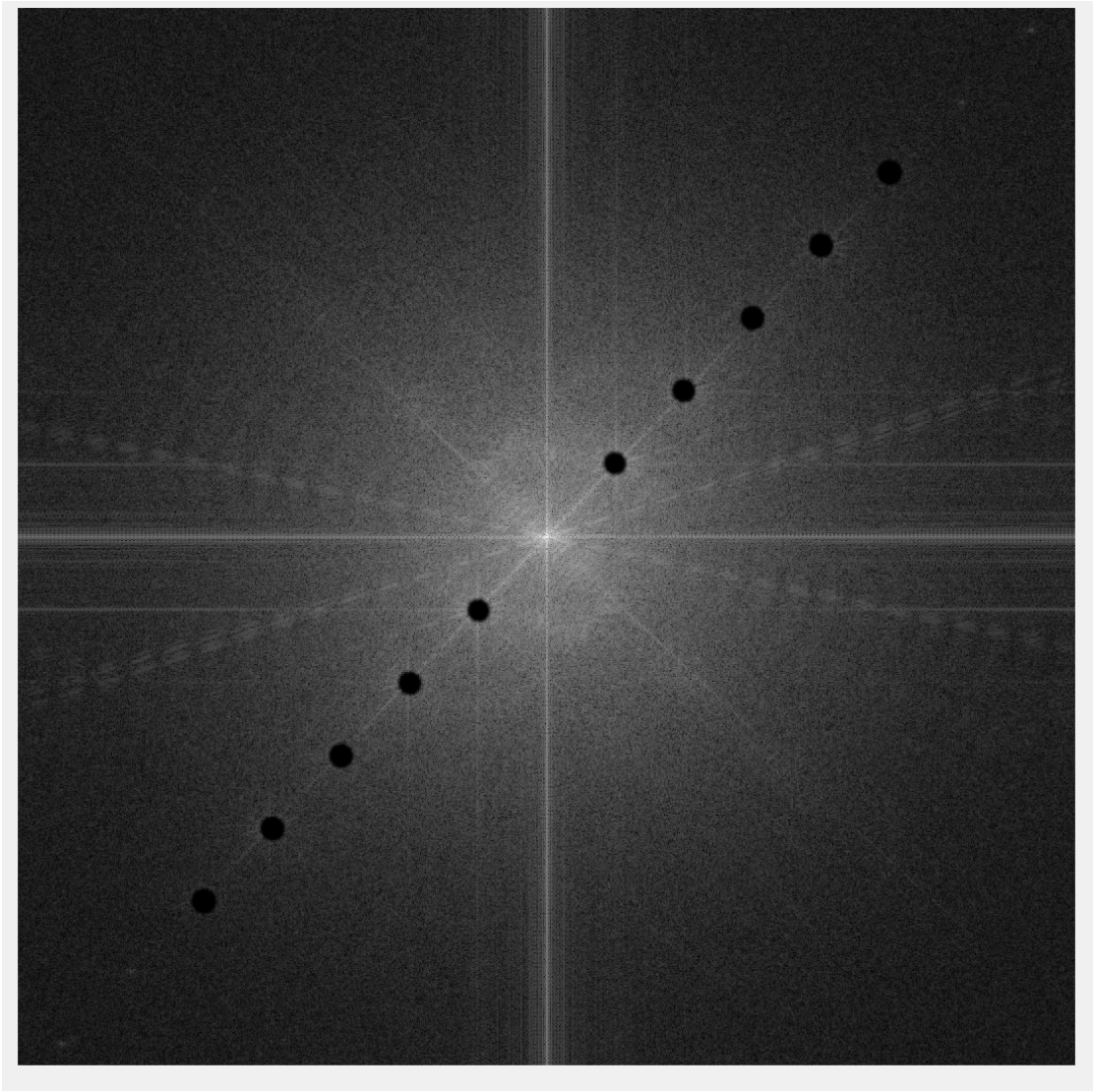
Fingerprint image quality is usually poor. The background texture interference is serious and the ridge line is fuzzy.

Use a notch filter to remove the background texture：

Since there exist 10 high-frequency points on the spectrum from the bottom left to the top right which represent the noise that is due to the regular stripes in original time-domain image (named "fingerprint.bmp"). Below is how they look like:



The highlighted pointed are corresponding to the background of original fingerprint – the series of oblique stripes extending from upper left to bottom right. In order to remove it, I manually find the center point and blacken the neighborhood. Finally, I get the following picture:



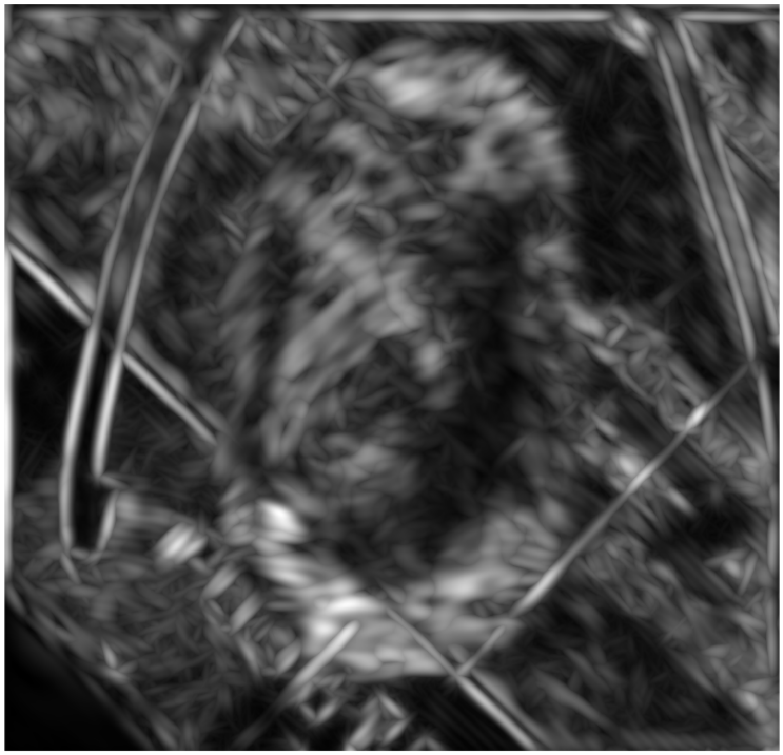
Inverse-Fourier transform is then implemented, and the real part is extracted. After that, on the upper left corner, original size of the results is taken out. And the image with less background noise texture comes out:

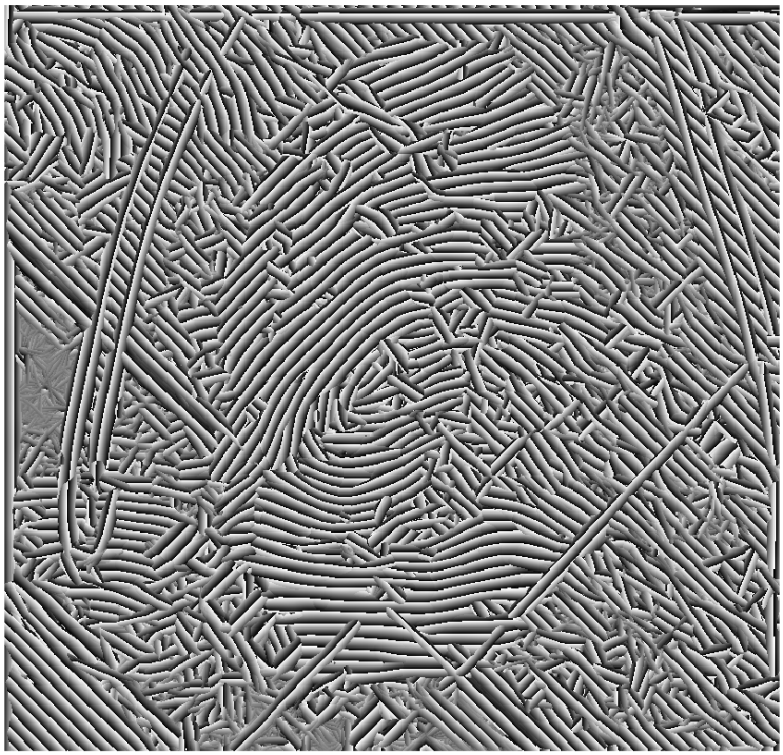
As for the codes, the script I have written is 'notch0'. When n = 10, (actually the 'black filled circles' created when n > 3 are all similar and could be all fed to later process), run the script and I get a mid product called 'halfway.bmp'.

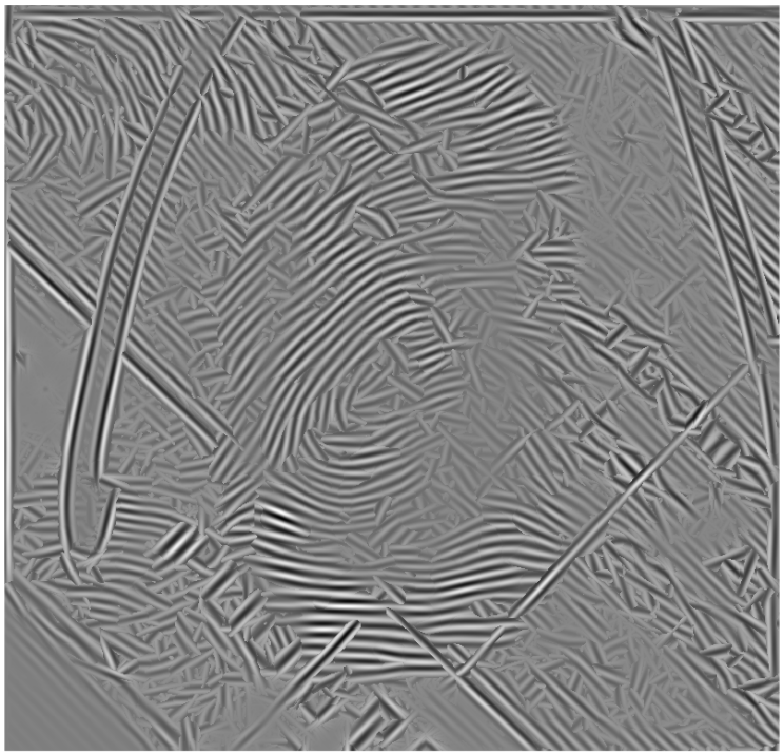
Implement adaptive enhancement to the fingerprint ridges with Gabor filters.

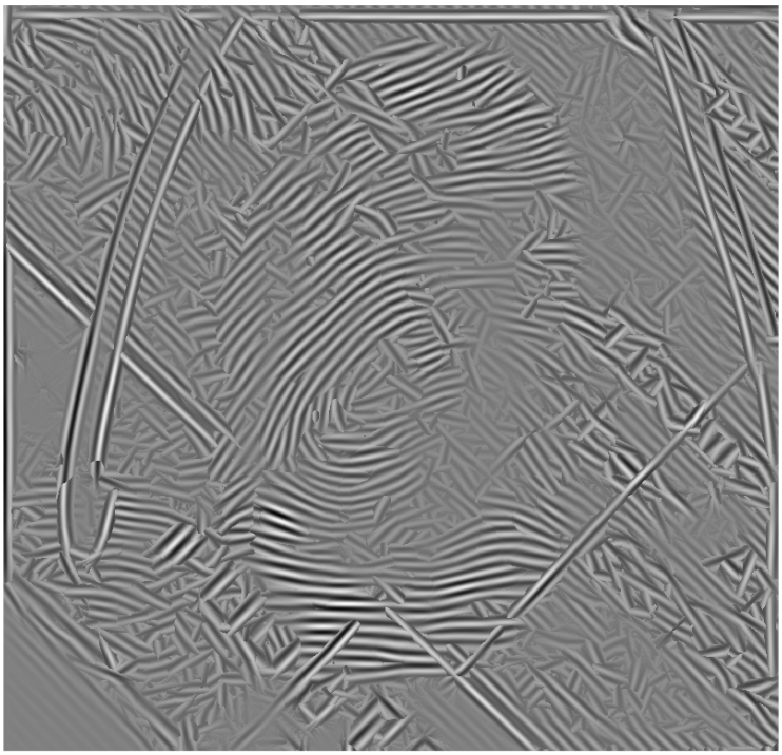
In my code, I have chosen 16 normally distributed directions. Then to each pixel, I use the 16 Gabor filters to process original image and find which one get the max magnitude of this pixel. After

that, to this specific pixel, no matter to its magnitude or its phase, it will be processed by its according filter. Different pixels may be processed by different filters. Thus the enhanced fingerprint ridges are obtained. The images for magnitude, phase, real part and imaginary part are displayed as below:

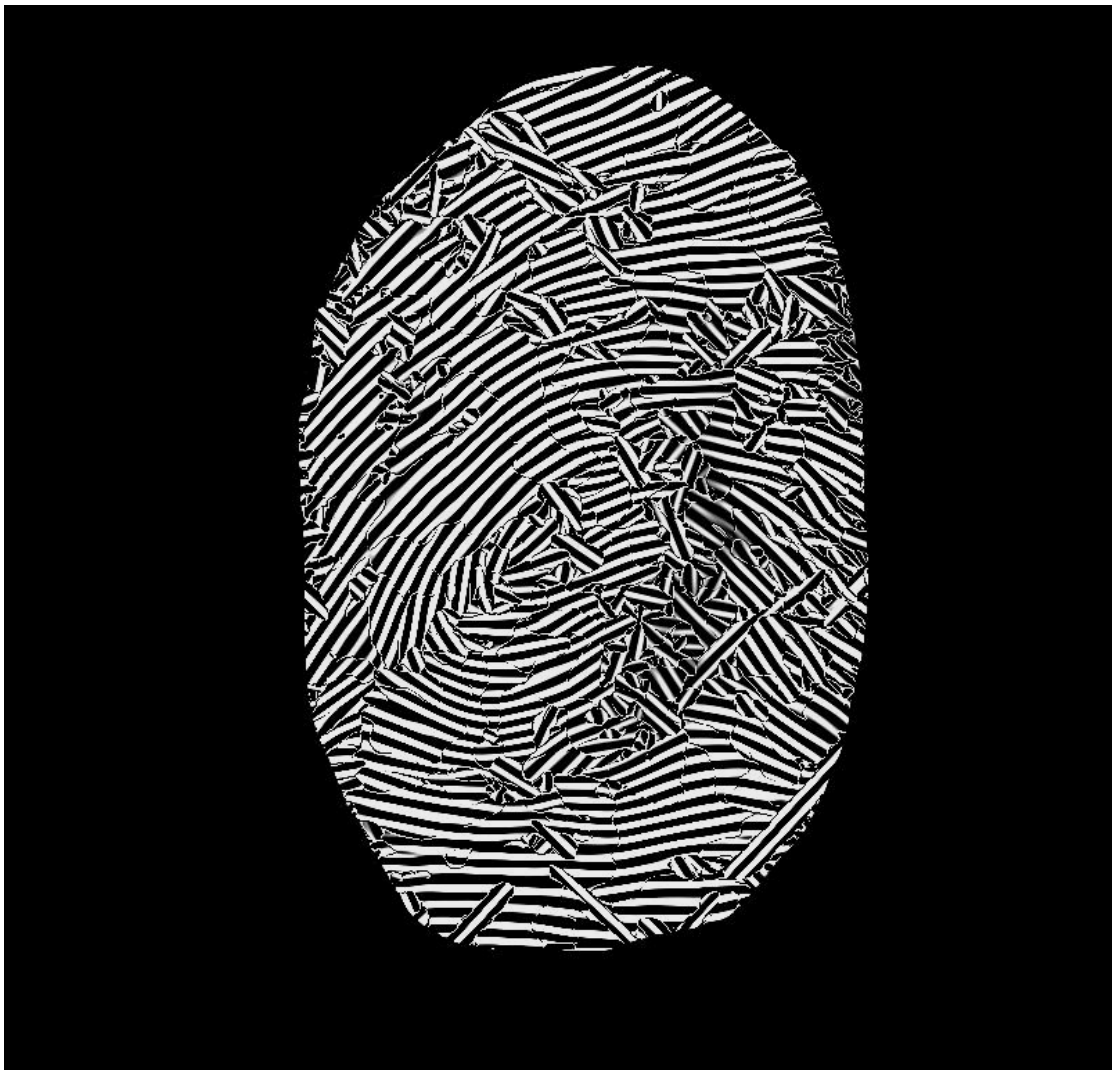








After some discussion with my partner, I get to know that better performance can be achieved by using Laplace Filter to sharpen the images and by extracting the fingerprints via binary image. The result is shown below:



Below is the second part of fingerprint process. (The original image has been replaced by my supervisor, that is we only need to deal with a new fingerprint image)

Improved Gabor filter

The code is gabor666.m。

This time, instead of using the specific filter that make the pixel's magnitude max, I have adopted a new idea of weighted operation, considering the former method may cause the fingerprint distortion and since I also don't want to dismiss the 'small-magnitude' directions. Specifically, I calculate the weights (sum is 1) after I do some process of the magnitude of all 16 directions of each pixel. Then I use the weights to do '.*' operation to the real parts of all 16 directions and get the weighted real-part image, which is the image after pre-process. In this way, it can filter the 'bridge connections' on the fingerprint more thoroughly than doing the same thing without Gabor filters.

'Some process' above, refers to power of 5.It doesn't matter what the power is. There are only 2 things to be careful: not ignore any one direction; but to enhance the 'high-magnitude' direction and dampen 'small-magnitude'

direction. As a result, high power is a good choice。

With the Gabor filter process, results are as below:

Ridges segmentation:

Ridges segmentation is set as two parts of Binarizing and Morphological Process respectively. The Binarizing algorithm can be realized using 'im2bw' function in the MATLAB Image Processing Toolbox. Below I have designed appropriate morphological algorithm to process the binary image, where I have tried to eliminate the deficiencies of the binary image (e.g. holes on ridges, irregular ***concave-convexes on the sketches of ridges, isolated islands on the white background, etc.****).*

Codes are direction_line.m and division2.m。

The former one is used to make the 'line-type' structure

elements of more than one direction.

The latter one binarizes the original image at first. Then it erodes the processed image and dilates it. The lengths and direction quantity of the selected structure elements are gained through experiments. I have chosen the structure elements of 10 directions each with length of 7 (pixel). The fingerprint after being ridge segmentation is shown below:

Ridges thinning:

It is set as two parts of thinning and post-processing. We can use the ‘bwmorph’ function in MATLAB. In the

following I have implement suitable morphological algorithm to do post-process upon thinning image (short line elimination, rag elimination and bridge elimination).

Codes are f_thin_pre.m, f_thin_after, and Pruning.m.

F_thin_pre is the prosess of thinning using toolbox function ‘bwmorph’. It can be done ‘Inf’ (infinite) times, which means it will not stop iterate until the result doesn’t change. Afterwards, I have created a binary image manually, which is aimed at wiping off the unneeded background outside the core fingerprints. Result is as below:

Pruning has used 12 end-point structure elements, making it more effectively to find those singular points with ‘hit-

or-miss' algorithm and more efficiently to conduct the pruning.

F_thin_after mainly uses pruning to process the fingerprint that has been thinned. After that, I have tried many algorithms in 'bwmorph'. With a lot of comparisons, finally I have used the method 'clean' to further optimize the image (short lines, rags and bridges haven't existed much after the thinning pre-process, therefore the operations here couldn't bring about a large improvement).

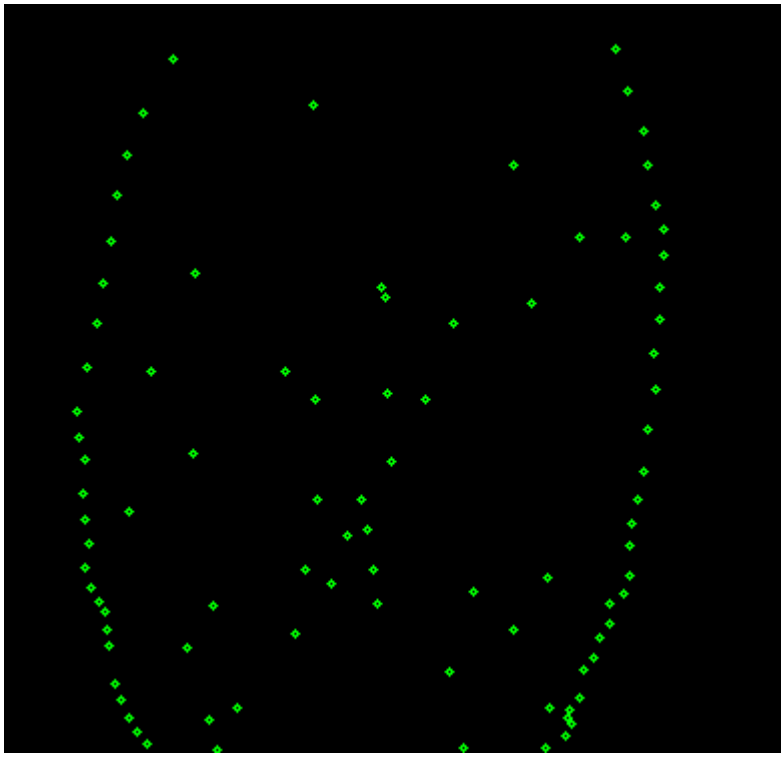
In the end, the image after the whole process of ridges thinning is as below:

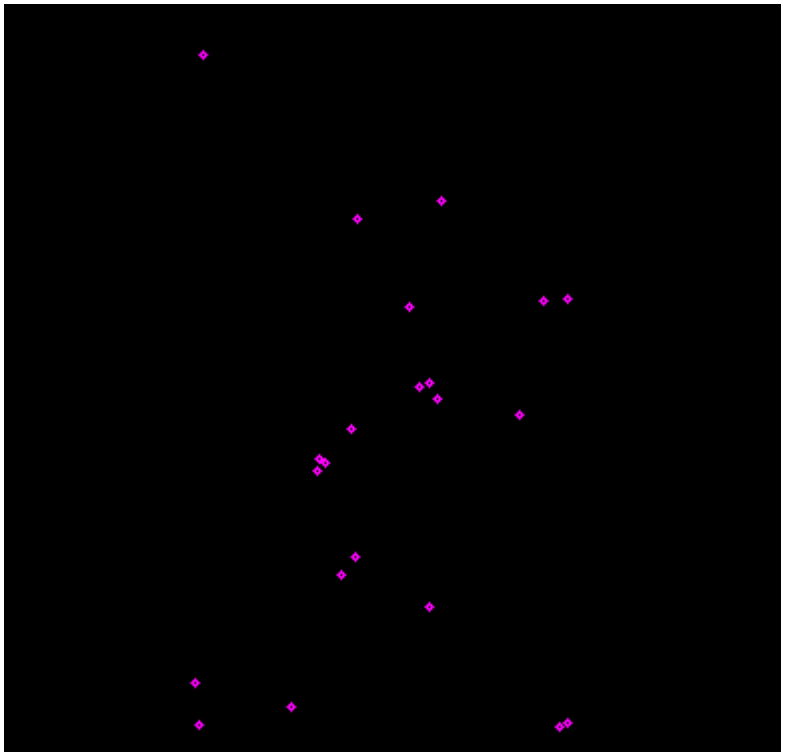
Detailed points detection

The algorithm for detailed points detection needs to discriminate different types of detailed points (end points and bifurcations. The types of pixel P is defined by its cross number ‘cn’.

This task is not very hard based on ridges thinning. The key point is that it is necessary to do the calculation of cross numbers upon non-peripheral pixels and decide whether each pixel is a detailed point accordingly. It is worthwhile to notice that the detailed points should be the points in the core fingerprint in the first place, otherwise the endpoints of each piece of line will count, which would be ridiculous. Then I implement the dilation onto the images of 2 kinds of detailed points respectively and ‘cut out’ the middle pixel, making each detailed point become a ‘circle’. At this time I transform the original image and 2 detailed images to the 3-channel rgb images: original rgb image can be gained by concatenating 3 identical grew-scale ones while the 2 images of detaied points have been processed differently. In order to achieve the better discrimination visual effect, I want the

detailed pictures to be displayed in color. So when being concatenating, only one or two is original image and another two or one is 0-filled image. 程序中分别用了g分量表示端点，用了r和b分量表示交叉点，因此呈现出绿色和紫色的圈。In the program, ‘g’ component is used to represent endpoints while ‘r’ and ‘b’ components are used to represent bifurcations. Hence the 2 kinds of points are shown as green and purple *circles*.





By superpose the original image and 2 detailed images, the final beautiful picture is obtained: