NTU DIP 2020 Spring HW2 Report

B06902002　資工三　沈郁鈞

Problem 1: EDGE DETECTION

(a)

The original image is shown as below:

sample1.jpg

First order Edge Detection

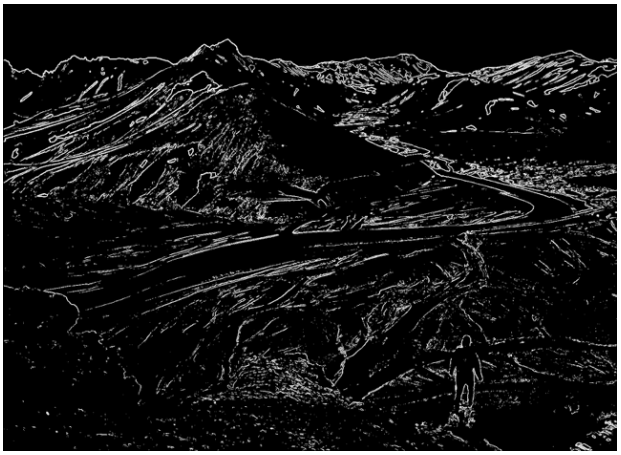
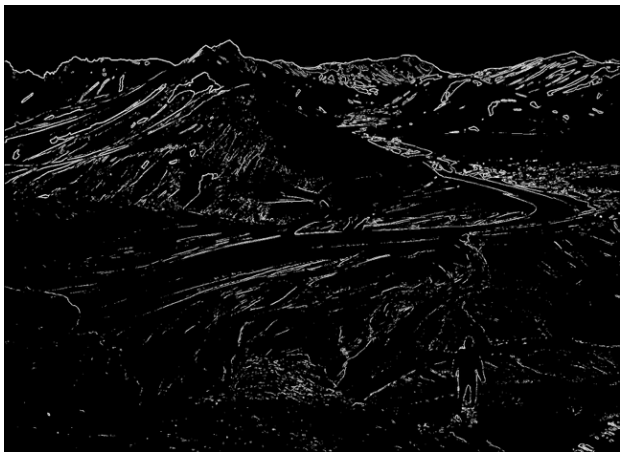
The first order edge detection is convolving the given image with the two following filters:

$$row\ filter = \begin{bmatrix} 1 & k & 1 \\ 0 & 0 & 0 \\ -1 & -k & -1 \end{bmatrix} \quad column\ filter = \begin{bmatrix} -1 & 0 & 1 \\ -k & 0 & k \\ -1 & 0 & 1 \end{bmatrix}$$

With these two filters, we will have two gradients G_r and G_c. Then, we compute the final gradient $G = \sqrt{(G_r)^2 + (G_c)^2}$. The threshold T is used to decide whether the pixel is edge or not.

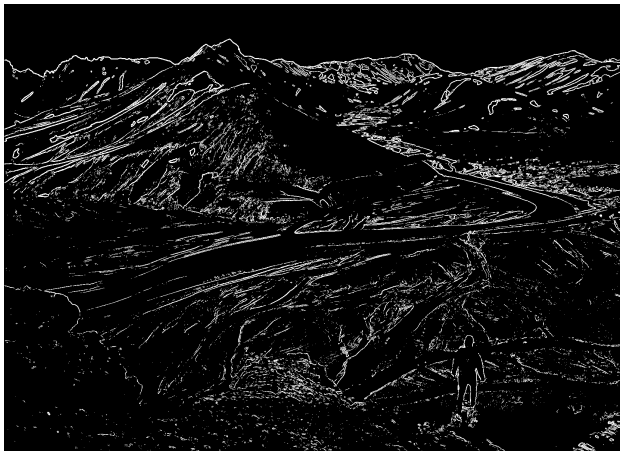
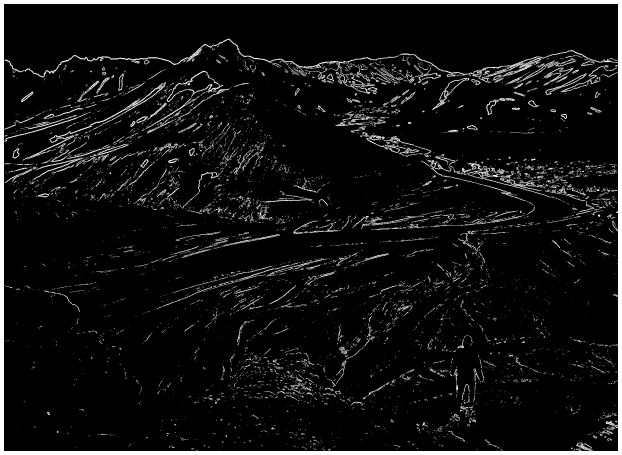
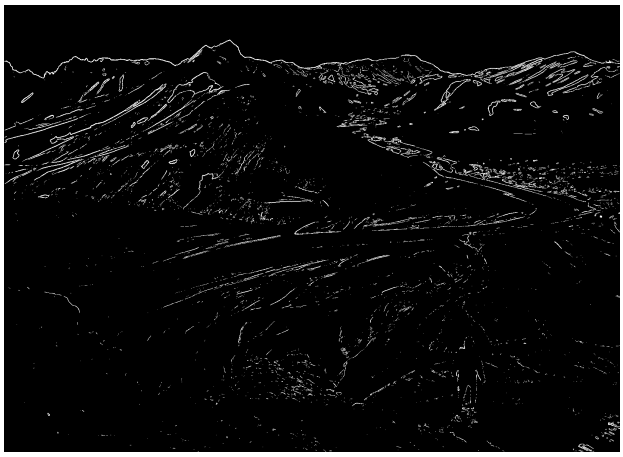
Here I discuss two cases:

1. $k = 1$ (Prewitt's edge detector)
   Below are three edge maps using Prewitt's edge detector with different threshold.

	
$T = 30$	$T = 40$
	
$T = 50$	

As threshold becomes higher, less pixels would be defined as edge pixels. Observe the bottom-half of the three images, the number of edges decreases rapidly.

2. $k = 2$ **(Sobel's edge detector)**
   Below are three edge maps using Sobel's edge detector with different threshold.

	
$T = 40$	$T = 50$
	
$T = 60$	

Same as the Prewitt's detector, as threshold becomes higher, less pixels would be defined as edge pixels. Observe the bottom-half of the three images, the number of edges decreases rapidly.

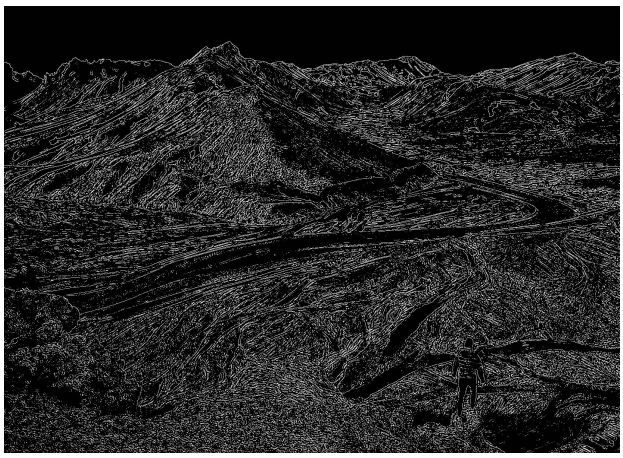
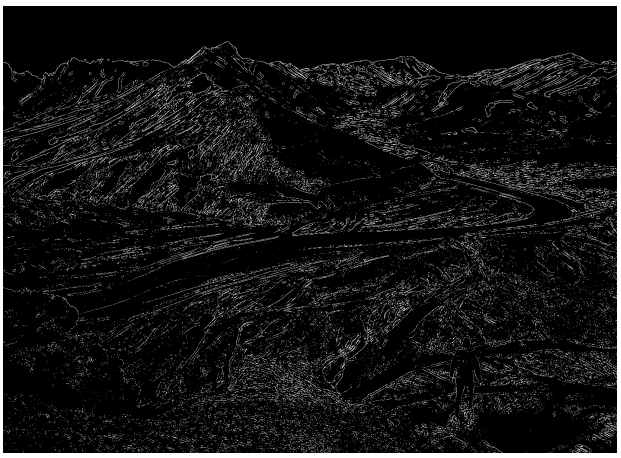
Second order Edge Detection

Here I use **Laplacian of Gaussian** to detect edges. Here I'll convolve the two images with two different filter size 9×9 and 11×11 (both of them have same $\sigma = 1.4$). The filter is shown as below:

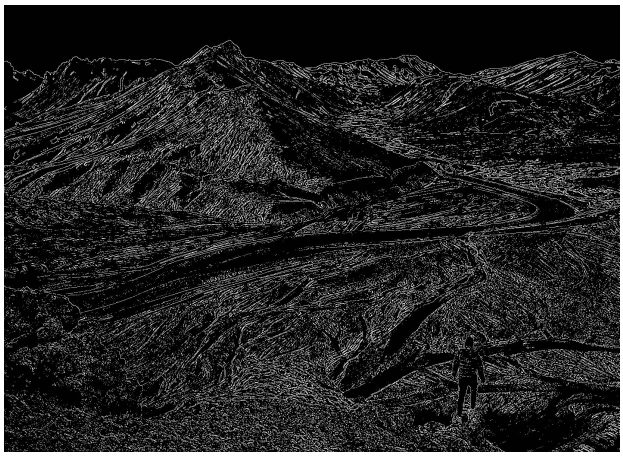
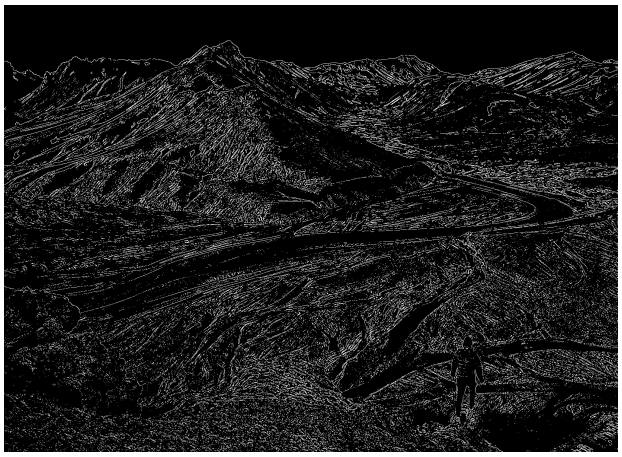
$$A_{9\times 9} = \begin{bmatrix} 0 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 0 \\ 1 & 2 & 4 & 5 & 5 & 5 & 4 & 2 & 1 \\ 1 & 4 & 5 & 3 & 0 & 3 & 5 & 4 & 1 \\ 2 & 5 & 3 & -12 & -24 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -24 & -40 & -24 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -24 & -12 & 3 & 5 & 2 \\ 1 & 4 & 5 & 3 & 0 & 3 & 5 & 4 & 1 \\ 1 & 2 & 4 & 5 & 5 & 5 & 4 & 2 & 1 \\ 0 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 0 \end{bmatrix}$$

$$B_{11\times 11} = \begin{bmatrix} 0 & 0 & 0 & -1 & -1 & -2 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -2 & -4 & -8 & -9 & -8 & -4 & -2 & 0 & 0 \\ 0 & -2 & -7 & -15 & -22 & -23 & -22 & -15 & -7 & -2 & 0 \\ -1 & -4 & -15 & -24 & -14 & -1 & -14 & -24 & -15 & -4 & -1 \\ -1 & -8 & -22 & -14 & 52 & 103 & 52 & -14 & -22 & -8 & -1 \\ -2 & -9 & -23 & -1 & 103 & 178 & 103 & -1 & -23 & -9 & -2 \\ -1 & -8 & -22 & -14 & 52 & 103 & 52 & -14 & -22 & -8 & -1 \\ -1 & -4 & -15 & -24 & -14 & -1 & -14 & -24 & -15 & -4 & -1 \\ 0 & -2 & -7 & -15 & -22 & -23 & -22 & -15 & -7 & -2 & 0 \\ 0 & 0 & -2 & -4 & -8 & -9 & -8 & -4 & -2 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -2 & -1 & -1 & 0 & 0 & 0 \end{bmatrix}$$

The two edge maps below are using 9×9 filter with different threshold:

	
$T = 500$	$T = 1000$

And the two edge maps below are using 11×11 filter with different threshold:

	
$T = 2500$	$T = 3000$

> **For these two cases, as threshold becomes higher, the number of edges decreases.**

Canny Edge Detection

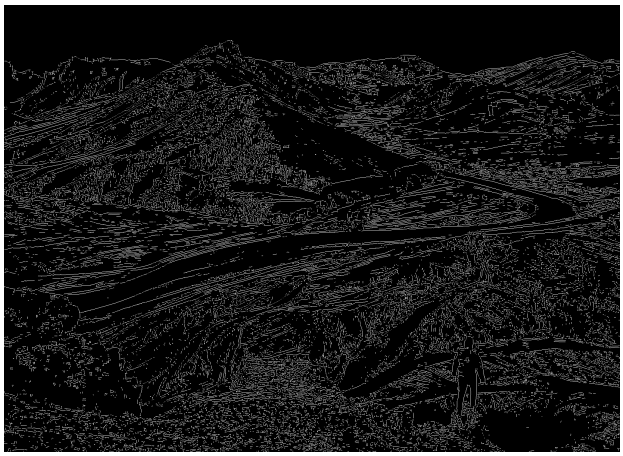
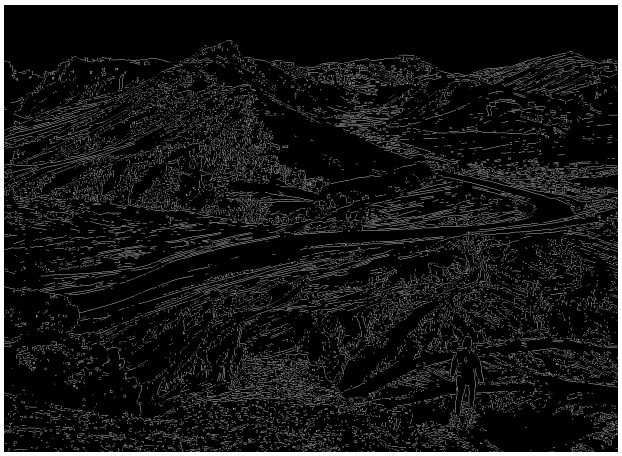
The Canny edge detection contains 5 steps:

- Noise Reduction
- Compute gradient magnitude and orientation
- Non-maximal suppression
- Hysteretic thresholding
- Connected component labeling method

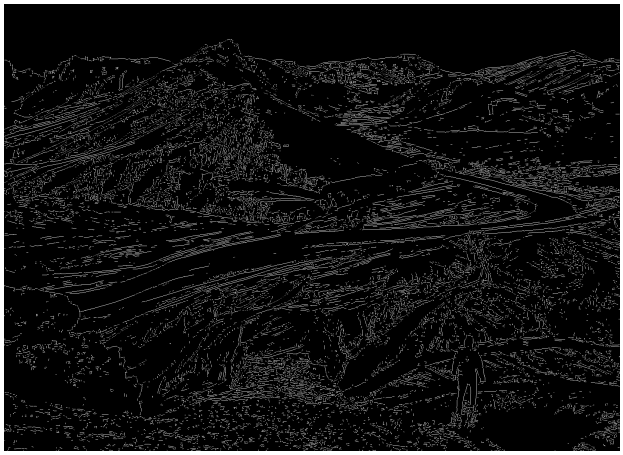
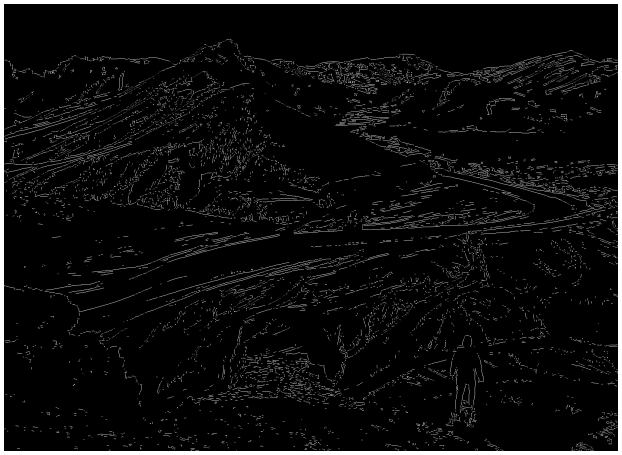
For noise reduction, I applt the following 5×5 Gaussian filter with $\sigma = 1.4$:

$$\frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

Below are two edge maps using fixed low-threshold $T_L = 10$ and different T_H:

	
$T_L = 10, T_H = 30$	$T_L = 10, T_H = 35$

Below are two edge maps using fixed low-threshold $T_L = 20$ and different T_H:

	
$T_L = 20, T_H = 40$	$T_L = 20, T_H = 60$

> From edge maps above, as threshold gets higher, number of edges decreases.

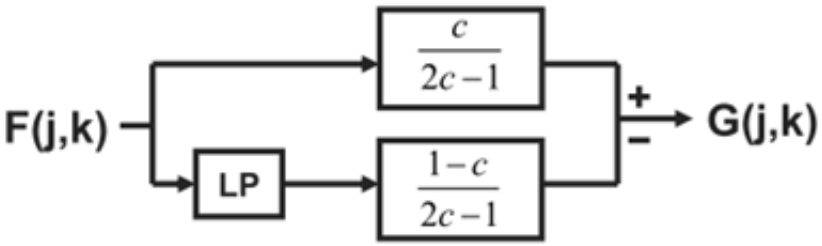
Here I list the **pros and cons** of these three edge detection method:

	First order edge detection	**Second order edge detection**	**Canny edge detection**
Pros	1. Easy to implement 2. Least time-consuming	relatively more unaffected by noise than first-order edge detection	Most accurate way to capture edge
Cons	Sensitive to noise	Time-consuming when filter size gets larger	Time-consuming (since it has 5 steps.)

(b)

Here I use **Unsharp Masking** to crispen edges.

This algorithm takes linear combination of all-pass and low-pass filters.



$$G(j,k) = \frac{c}{2c-1} F(j,k) - \frac{1-c}{2c-1} F_L(j,k), \quad where \quad \frac{3}{5} \le c \le \frac{5}{6}$$

Here I use $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ as low-pass filter, and set $c = 0.6$.

Below are images before/after unsharp masking process:

	
`sample2.jpg`	`result4.jpg`

The following two images are edge maps of the above two images
(using Prewitt's edge detector with threshold $T = 20$)

	
sample2_prewitt_20.jpg	result4_prewitt_20.jpg

Apparently, after edge crispening, more edges are detected, especially on fireworks.

However, if change c to 0.75, the effect doesn't seem to be obvious.

	
sample2.jpg	result4.jpg

Edge Maps:

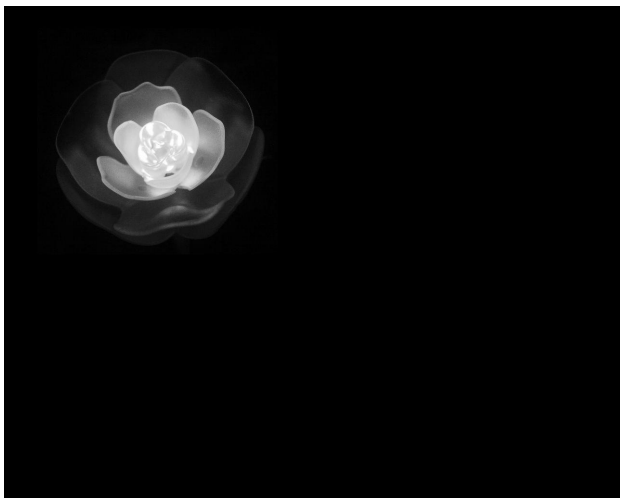
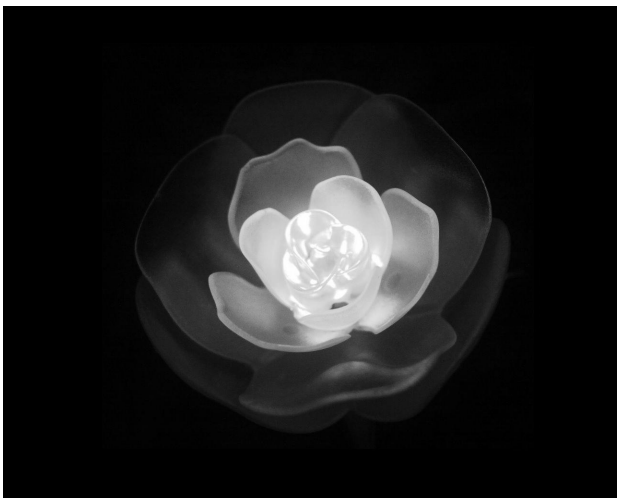
	
sample2_prewitt_20.jpg	result4_prewitt_20.jpg

In conclusion, as c becomes higher, the effect of edge crispening gets worse.

Problem 2: GEOMETRICAL MODIFICATION

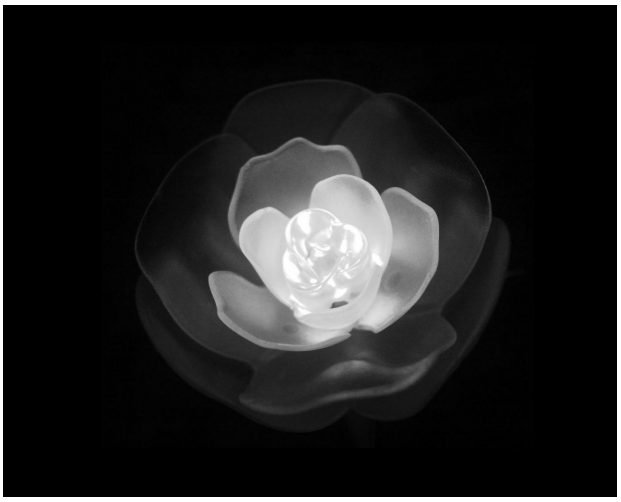
(a)

It seems that if we zoom-in the top-left corner of `sample3.jpg` , then we can get `sample4.jpg`

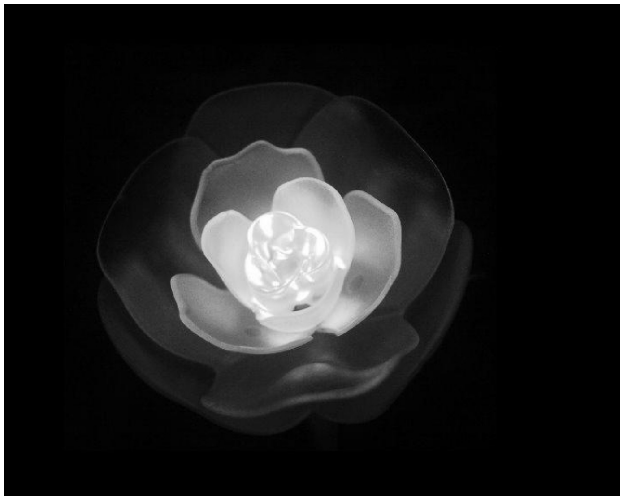
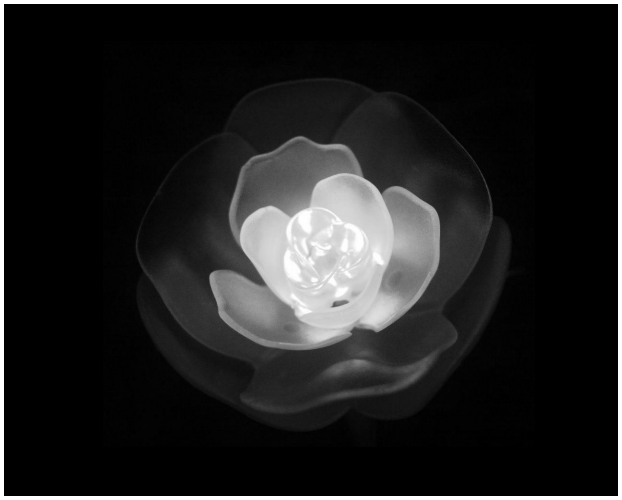
	
`sample3.jpg`	`sample4.jpg`

First, try out zoom in with rate 2.0. That is, apply transform function
$$\begin{bmatrix} k_{out} \\ j_{out} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} k_{in} \\ j_{in} \end{bmatrix}$$

The result image is shown as below:

	
	`sample4.jpg`

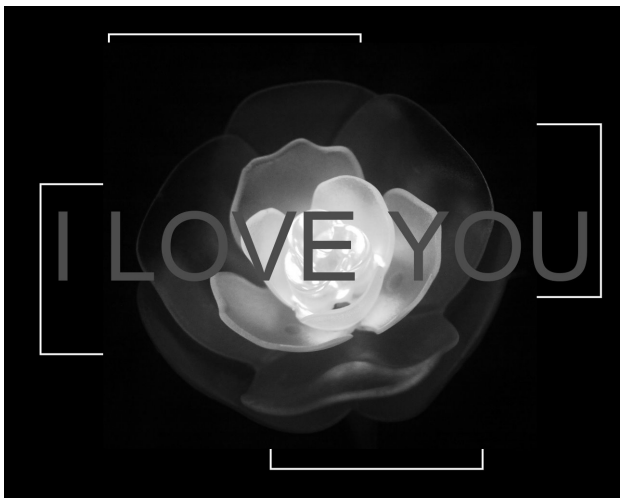
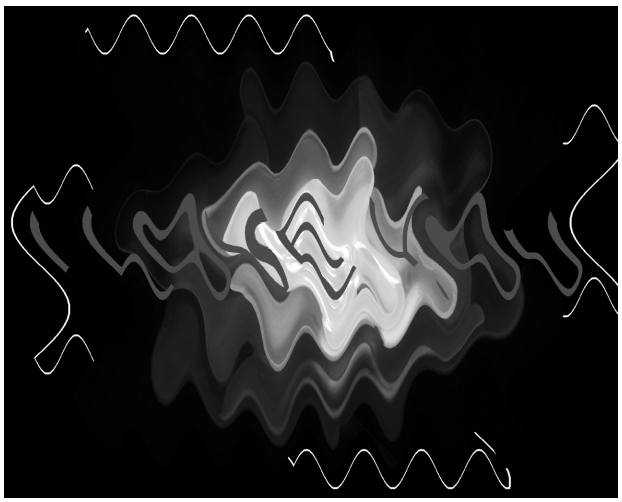
The flower seems to be bigger, so I try for smaller zoom-in rate 1.8. The transform function is $\begin{bmatrix} k_{out} \\ j_{out} \end{bmatrix} = \begin{bmatrix} 1.8 & 0 \\ 0 & 1.8 \end{bmatrix} \begin{bmatrix} k_{in} \\ j_{in} \end{bmatrix}$

	
result5.jpg	sample4.jpg

The flowers in the left seems to be as big as the right one.

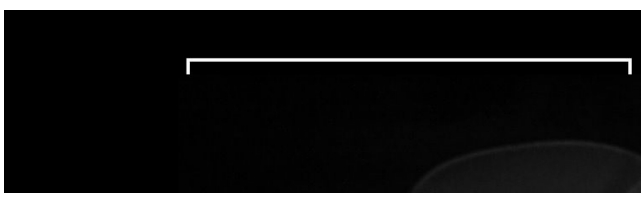
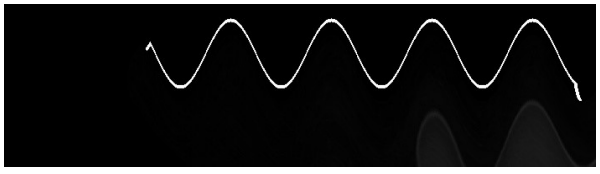
(b)

Below are `sample5.jpg` **and** `sample6.jpg` **.**

	
sample5_1.jpg	sample6_1.jpg

First, I crop `sample5.jpg` **and** `sample6.jpg` **to ramain the top and left part of the image.**

Below two images are the top-left corner of `sample5.jpg` **and** `sample6.jpg` **:**

	
sample5_1.jpg	sample6_1.jpg

Analyzing the wave of `sample6_1.jpg`, I get the wavelength $\lambda = 120$ and amplitude $h = 40$

Below two images are the most left part of `sample5.jpg` and `sample6.jpg`:

	
sample5_2.jpg	sample6_2.jpg

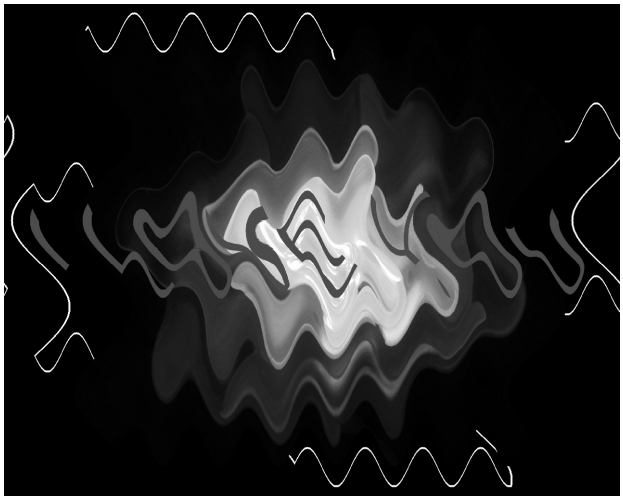
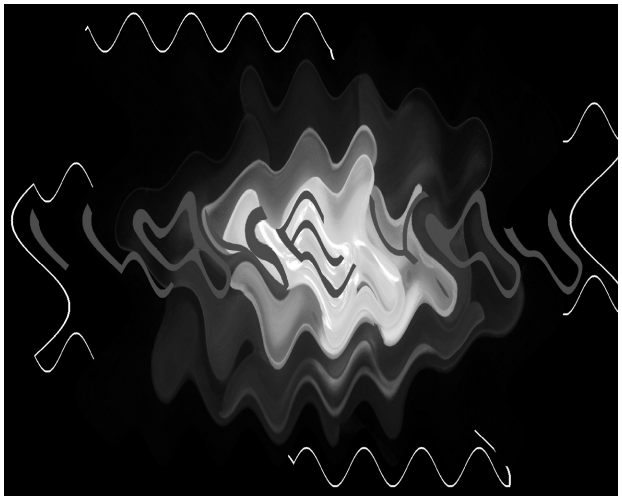
Analyzing the wave of `sample6_2.jpg`, I get the wavelength $\lambda = 360$ and amplitude $h = 60$

Thus, we can have the transform function:

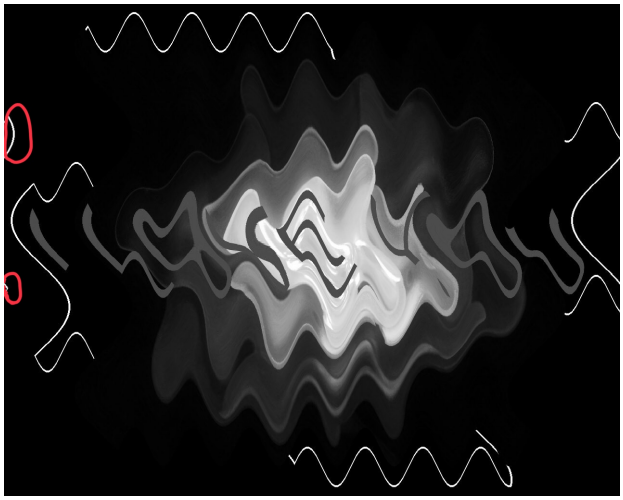
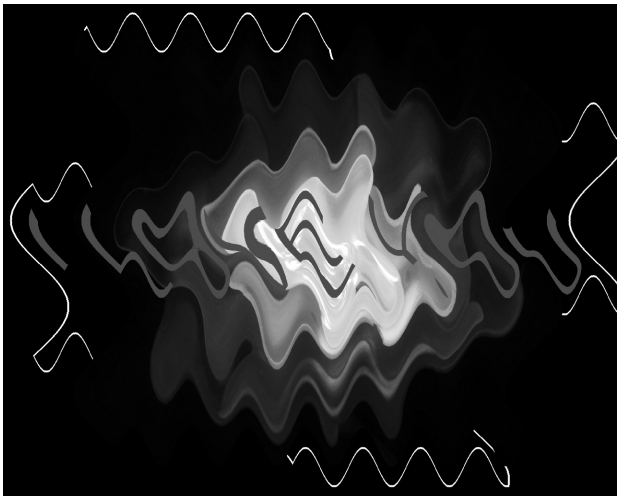
$k_{out} = k_{in} + 40 \times sin(\frac{j_{in} \times 2\pi}{120})$

$j_{out} = j_{in} + 60 \times sin(\frac{k_{in} \times 2\pi}{360})$

The result image `result6.jpg` is shown as below (compare to `sample6.jpg`):

	
result6.jpg	sample6.jpg

However, there are some differences on the left of two images. Below I circle the difference in two images.

	
`result6.jpg` **(difference)**	`sample6.jpg`