Task-1:

Let an input image be:

At first we'll convert it to a gray image. Let us do it through code:

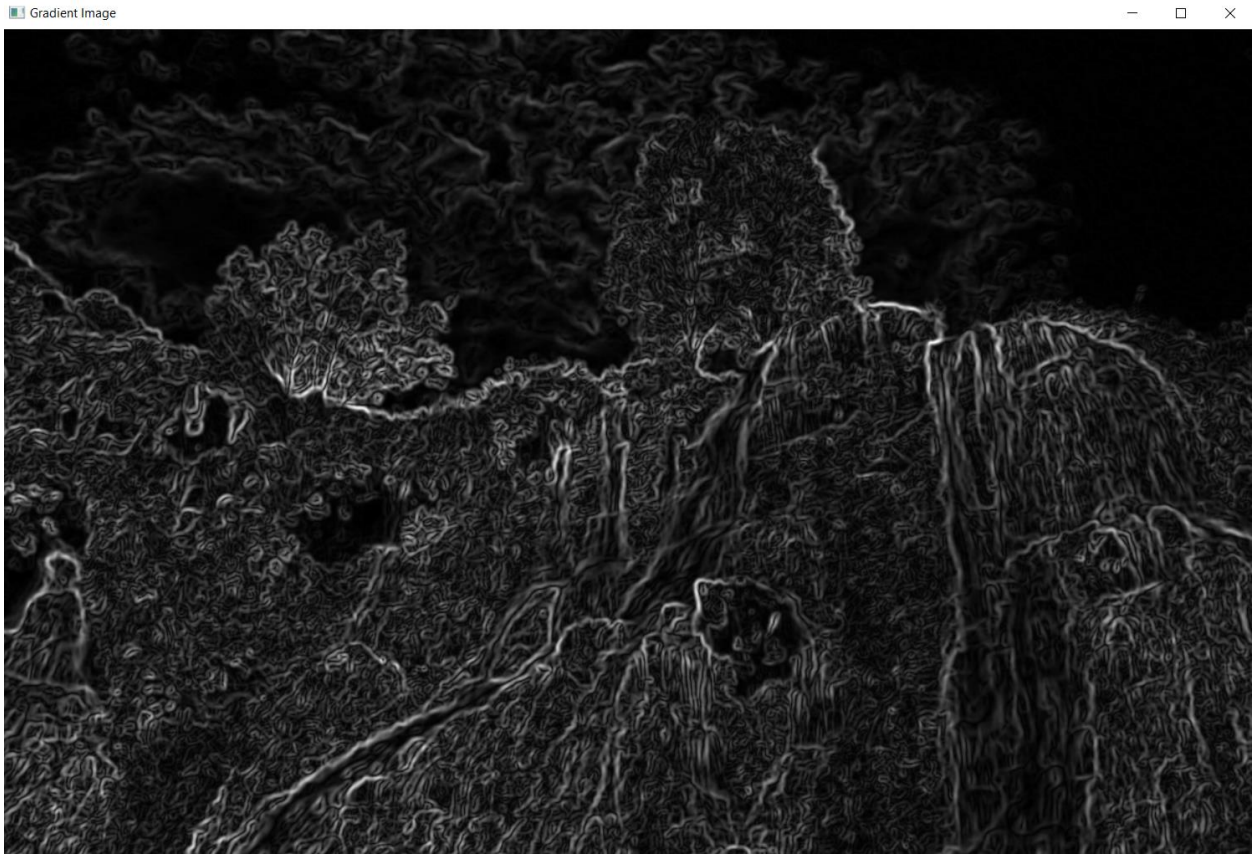
```
import cv2
import numpy as np
import math
from scipy import ndimage

img = cv2.imread('Shaqi_jrvej.jpg', 0)
gaussian = ndimage.gaussian_filter(img, sigma=1.4)

cv2.imshow('Gray Image', gaussian)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The output is as below:

Now we'll determine the intensity gradients using Sobel filters. And after that we'll find the gradient magnitude and direction.

```
sobel_x = cv2.Sobel(gaussian, cv2.CV_64F, 1, 0, ksize=5)
sobel_y = cv2.Sobel(gaussian, cv2.CV_64F, 0, 1, ksize=5)

mag = np.sqrt(sobel_x**2 + sobel_y**2)
mag = cv2.normalize(mag, None, 0, 255, cv2.NORM_MINMAX, cv2.CV_8U)

theta = np.arctan2(sobel_y, sobel_x)


cv2.imshow('Gradient Image', mag)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The output of this will be as below:

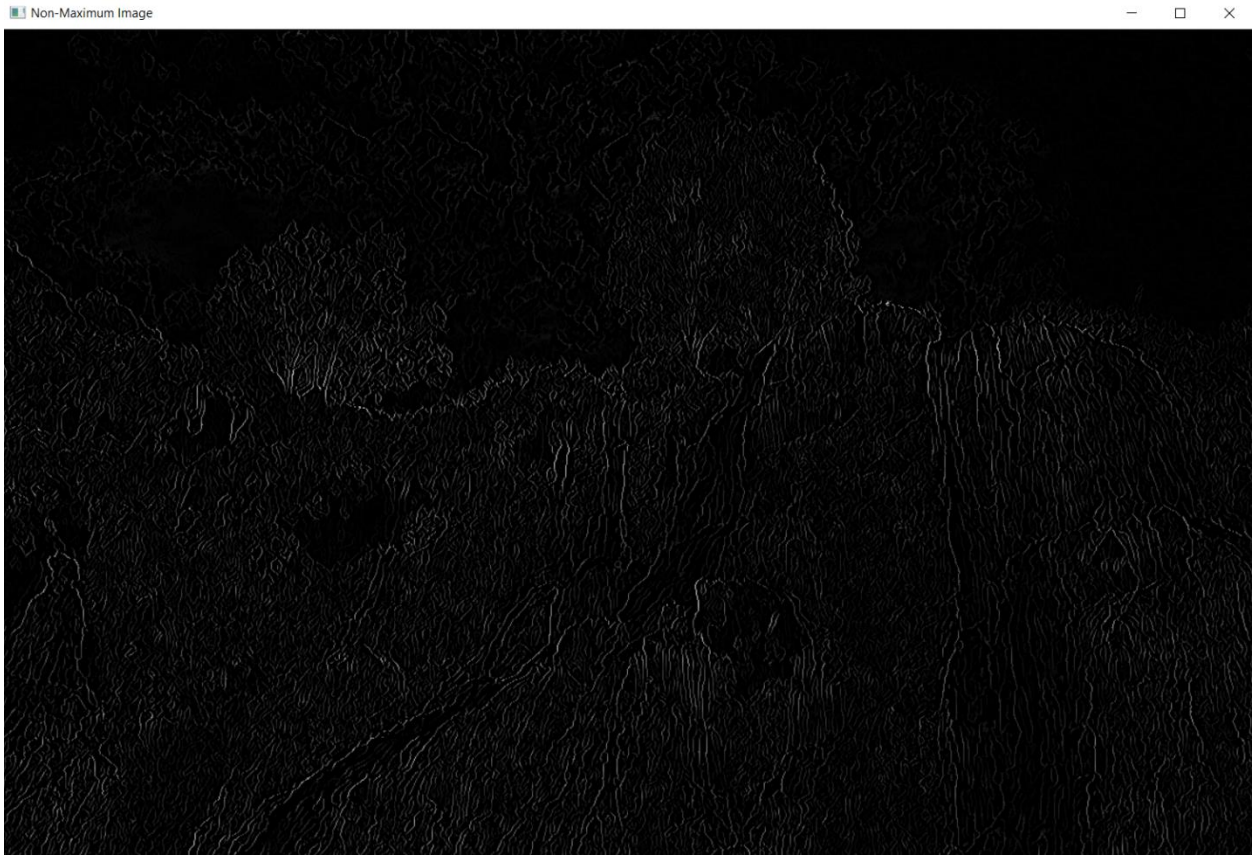


Now we'll do non-maximum suppression on the image to thin out the edges.

```
rows, cols = img.shape
edge_img = np.zeros((rows, cols))
angle = theta * 180. / np.pi
angle[angle < 0] += 180



for i in range(1, rows-1):
    for j in range(1, cols-1):
        q = 255
        r = 255
        if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):

            q = mag[i, j+1]
```

```
            r = mag[i, j-1]
        elif (22.5 <= angle[i,j] < 67.5):
            q = mag[i+1, j-1]
            r = mag[i-1, j+1]
        elif (67.5 <= angle[i,j] < 112.5):
            q = mag[i+1, j]
            r = mag[i-1, j]
        elif (112.5 <= angle[i,j] < 157.5):
            q = mag[i-1, j-1]
            r = mag[i+1, j+1]
        if (mag[i,j] >= q) and (mag[i,j] >= r):
            edge_img[i,j] = mag[i,j]
        else:
            edge_img[i,j] = 0
cv2.imshow('Non Suppressed Image', edge_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

The output of this becomes as follows:

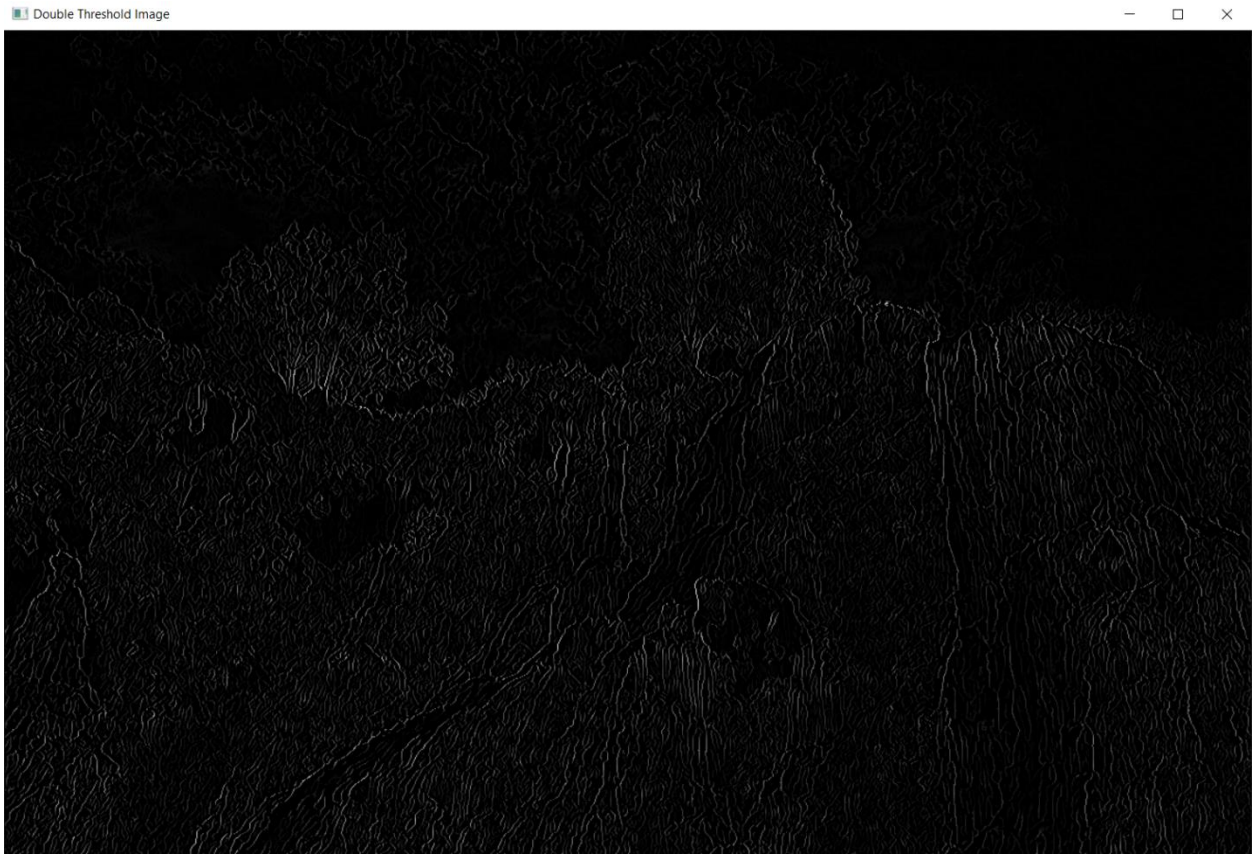


Now we do double thresholding on the image.

```
high_threshold = np.max(magnitude_suppressed)*0.0025
low_threshold = high_threshold*0.5

strong_edges = np.zeros_like(magnitude_suppressed)
weak_edges = np.zeros_like(magnitude_suppressed)
for i in range(magnitude_suppressed.shape[0]):
    for j in range(magnitude_suppressed.shape[1]):
        if magnitude_suppressed[i][j] >= high_threshold:
            strong_edges[i][j] = magnitude_suppressed[i][j]
        elif magnitude_suppressed[i][j] >= low_threshold:
            weak_edges[i][j] = magnitude_suppressed[i][j]
```

```
cv2.imshow('Double Threshold Image', strong_edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

So we get the output below:

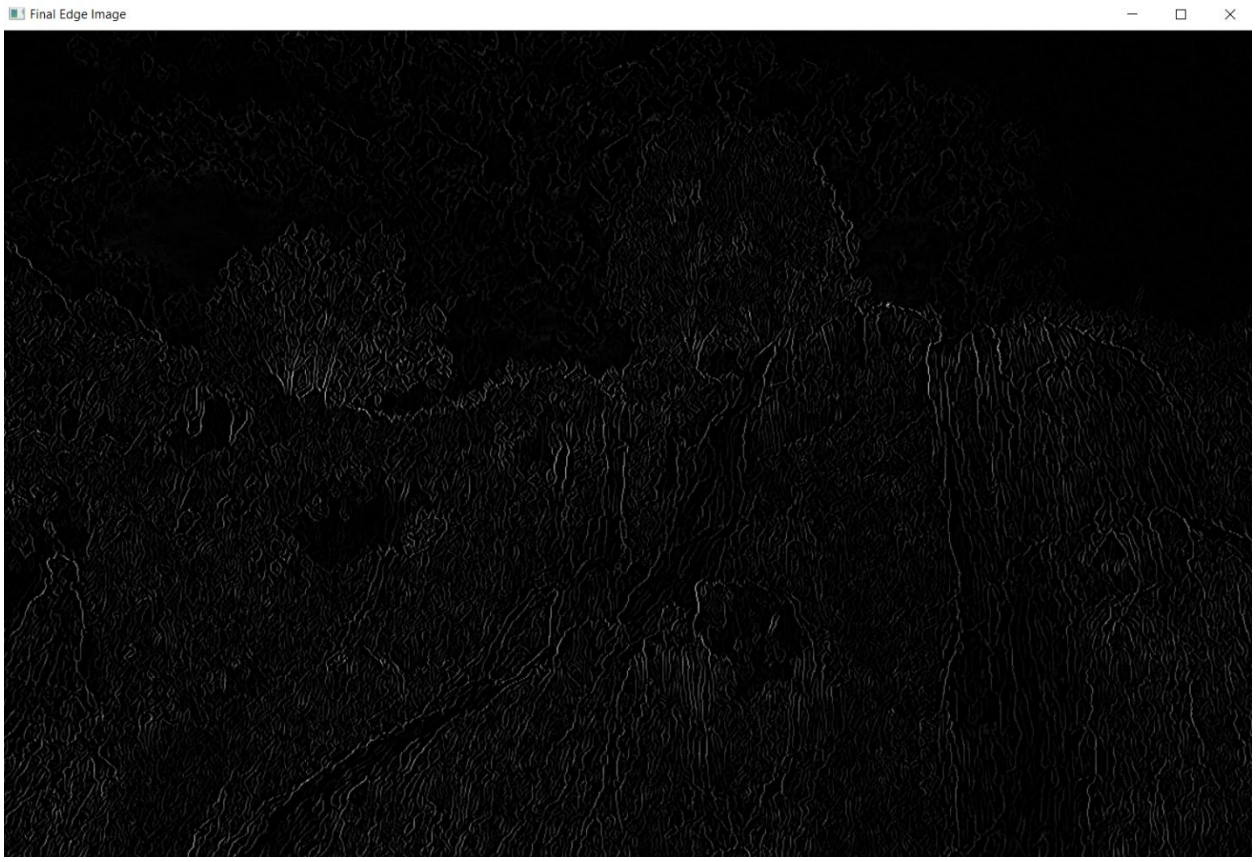


Finally we do hysteresis on the image.

```
final_edges = np.zeros_like(strong_edges)
for i in range(1, strong_edges.shape[0]-1):
    for j in range(1, strong_edges.shape[1]-1):
        if strong_edges[i][j] != 0:
            final_edges[i][j] = strong_edges[i][j]
            if weak_edges[i-1][j-1] != 0:
```

```
            final_edges[i-1][j-1] = strong_edges[i][j]
        if weak_edges[i-1][j] != 0:
            final_edges[i-1][j] = strong_edges[i][j]
        if weak_edges[i-1][j+1] != 0:
            final_edges[i-1][j+1] = strong_edges[i][j]
        if weak_edges[i][j-1] != 0:
            final_edges[i][j-1] = strong_edges[i][j]
        if weak_edges[i][j+1] != 0:
            final_edges[i][j+1] = strong_edges[i][j]
        if weak_edges[i+1][j-1] != 0:
            final_edges[i+1][j-1] = strong_edges[i][j]
        if weak_edges[i+1][j] != 0:
            final_edges[i+1][j] = strong_edges[i][j]
        if weak_edges[i+1][j+1] != 0:
            final_edges[i+1][j+1] = strong_edges[i][j]
cv2.imshow('Final Edge Image', final_edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Finally we get an output as below:



So this is the overall process of canny edge detection step by step.

Task-3:

There are several methods for increasing the Canny edge detector algorithm's speed. Here are a few examples:

- We can use parallel processing because the Canny edge detection algorithm is sequential by nature. However, using parallel processing techniques, the algorithm can be speed up. This is accomplished by segmenting the image into smaller regions and processing them in parallel.

- The Canny edge detector algorithm can be accelerated with specialized hardware like GPUs or FPGAs. These devices can perform the algorithm's

calculations in parallel, significantly increasing the algorithm's speed.

- There are several optimized algorithms for the Canny edge detector that can improve the algorithm's speed. An approximate Gaussian blur algorithm, such as the box blur algorithm, can be used instead of the Gaussian blur algorithm, which can be computationally expensive.

- One way to improve the speed of the Canny edge detector is to reduce the image size. This can be accomplished by either resizing the image or using a smaller region of interest. This, however, may come at the expense of decreased accuracy.

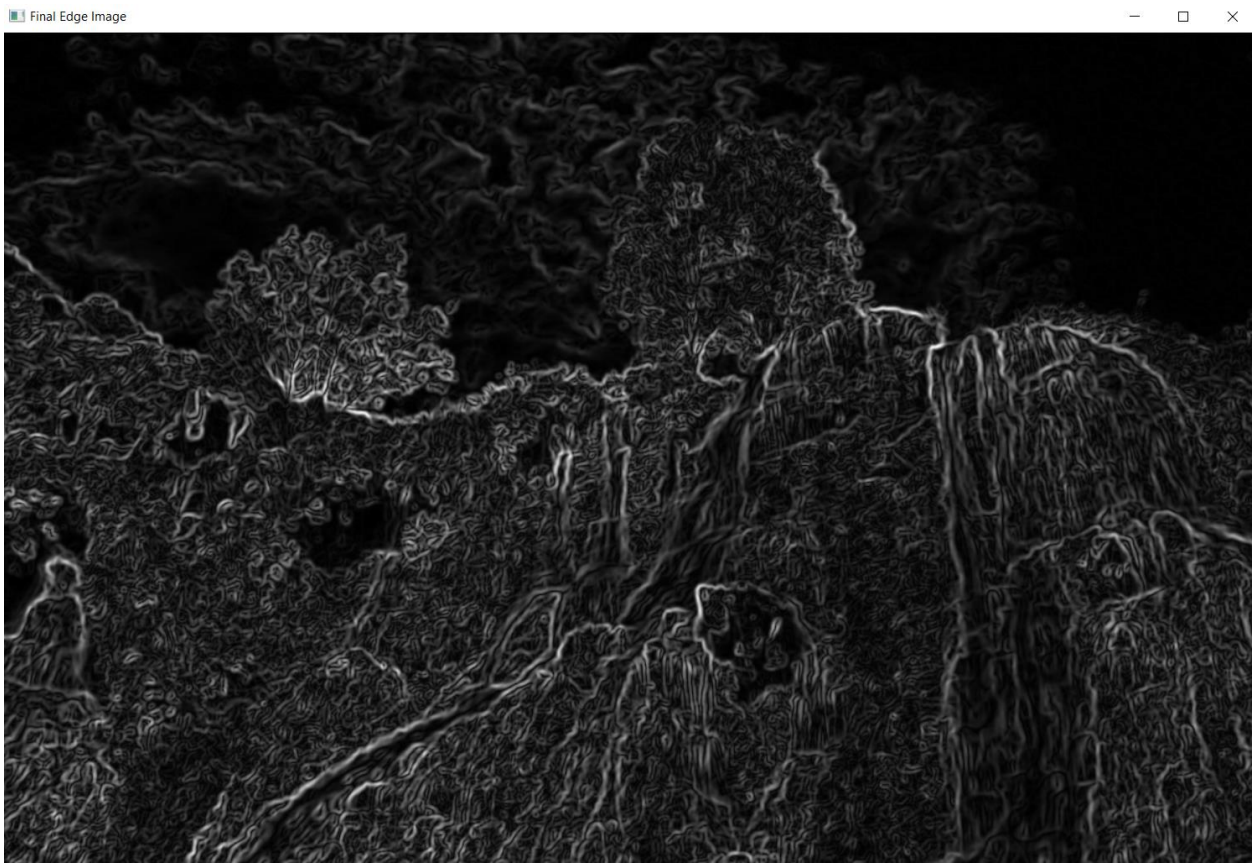
It is important to note, however, that the speed of the Canny edge detector is frequently dependent on the size of the image, the complexity of the image's edges, and the available computing resources. As a result, the best way to improve the speed of the Canny edge detector will vary depending on the application and the available resources.

Task-2:

B:

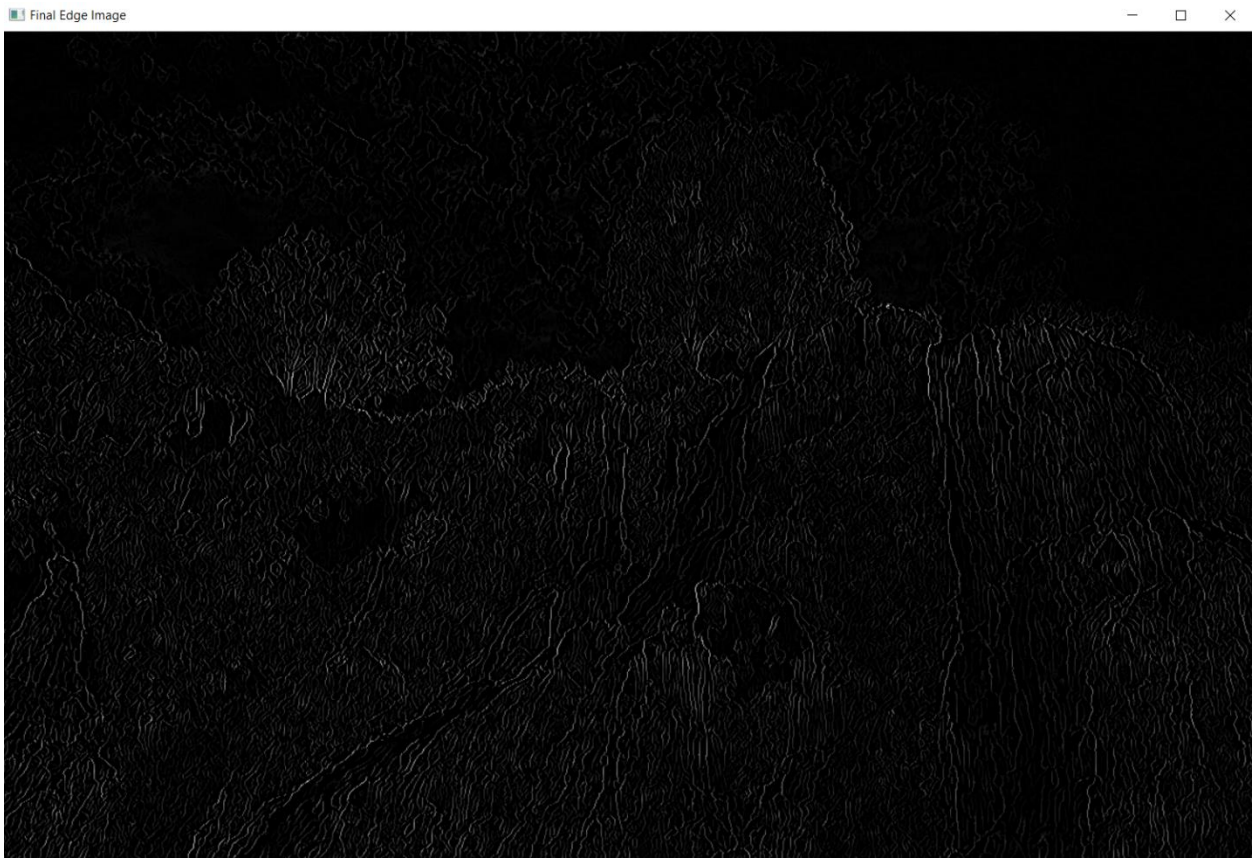
If we don't apply non-maximum suppression, then the edges remain thick after applying the sobel filter. The non-maximum suppression is done so that we get more thin edges from the processing. If it is not done then the edges remain thick for the hysteresis approach.

Without the non-maximum suppression the processed image after canny edge detection looks like:



So we can see the edges are thick after edge detection.

Whereas, doing non-maximum suppression after canny edge detection we get an output like:

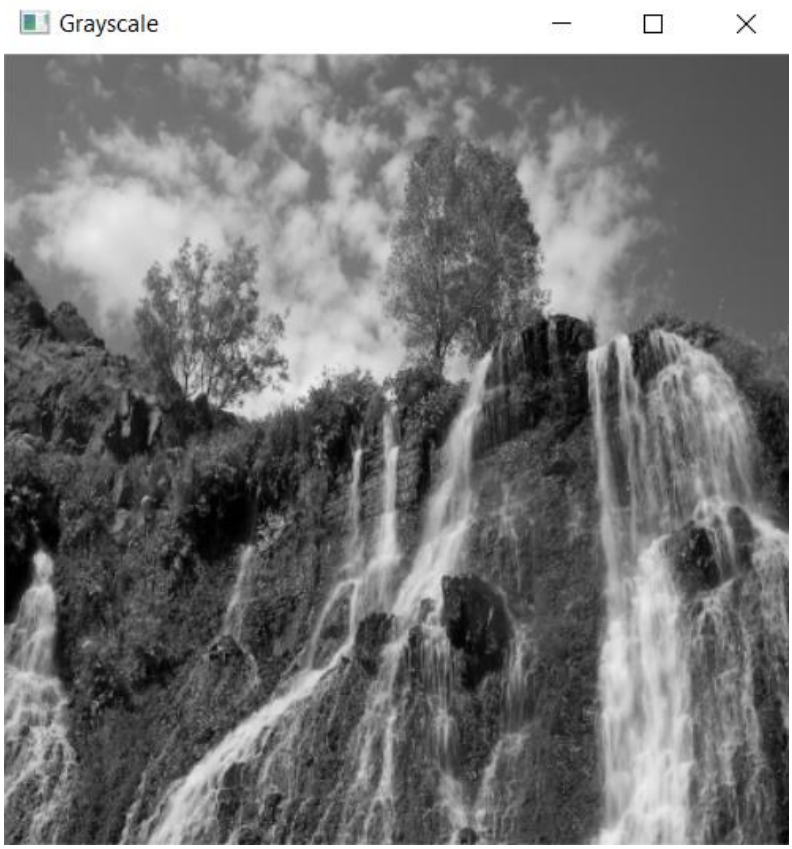


So, these are the results of not applying non-maximum suppression and applying it afterwards.

A:

For this section we're going to use the band angle 45 to 135 and compare our results of final edges with the previous ones where we took band angle from 0 to 180.

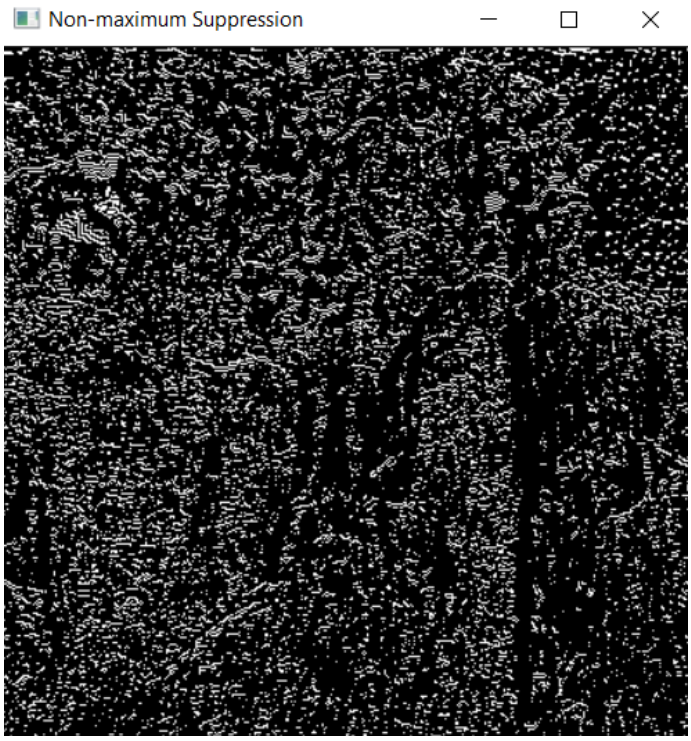
The original image or input image remains as before. The gray scale image and noise reduction image will be as similar to the previous band angle.


Grayscale

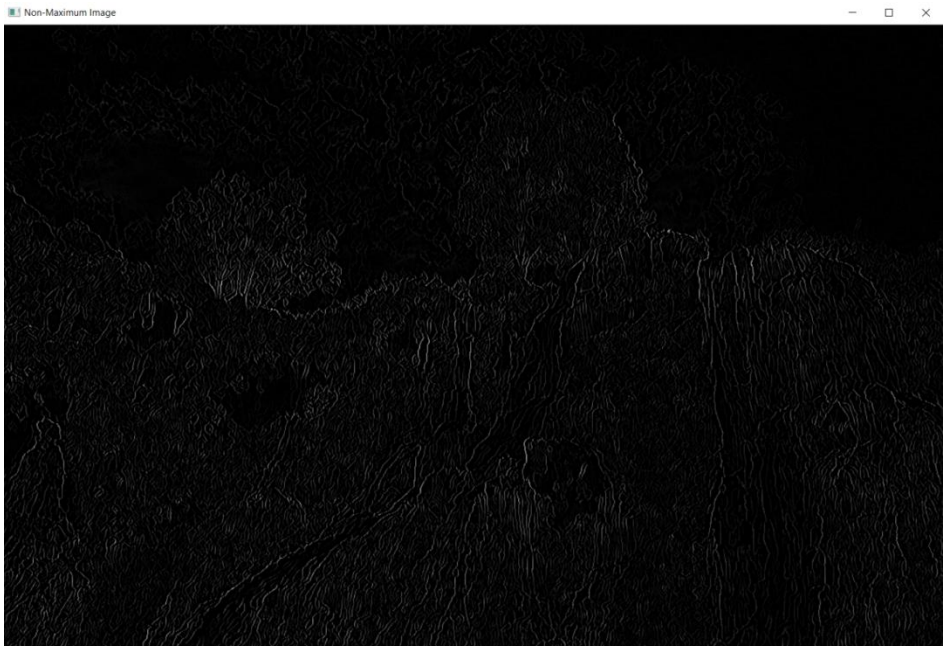

Noise Reduction

We'll see the difference in the non-maximum, double thresholding and hysteresis image for the band angle 45 to 135.

For Non-maximum:

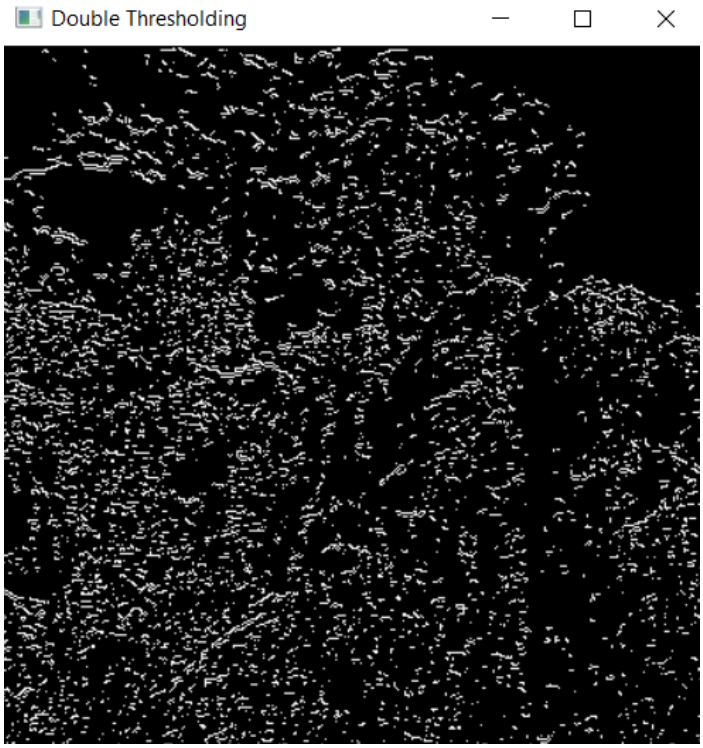


for 45 to 135

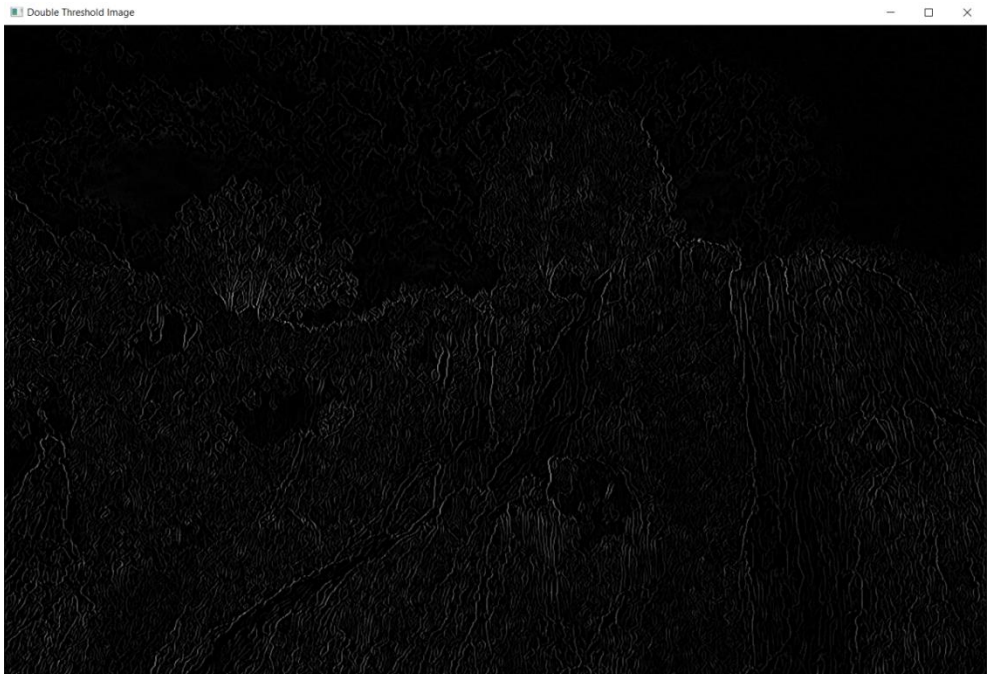


for 0 to 180

For Double thresholding:

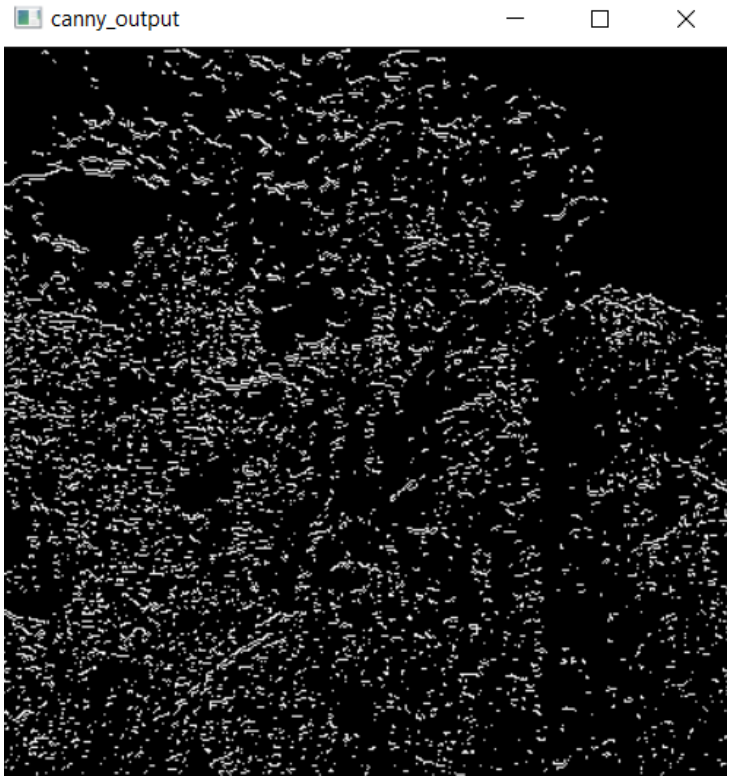


for 45 to 135

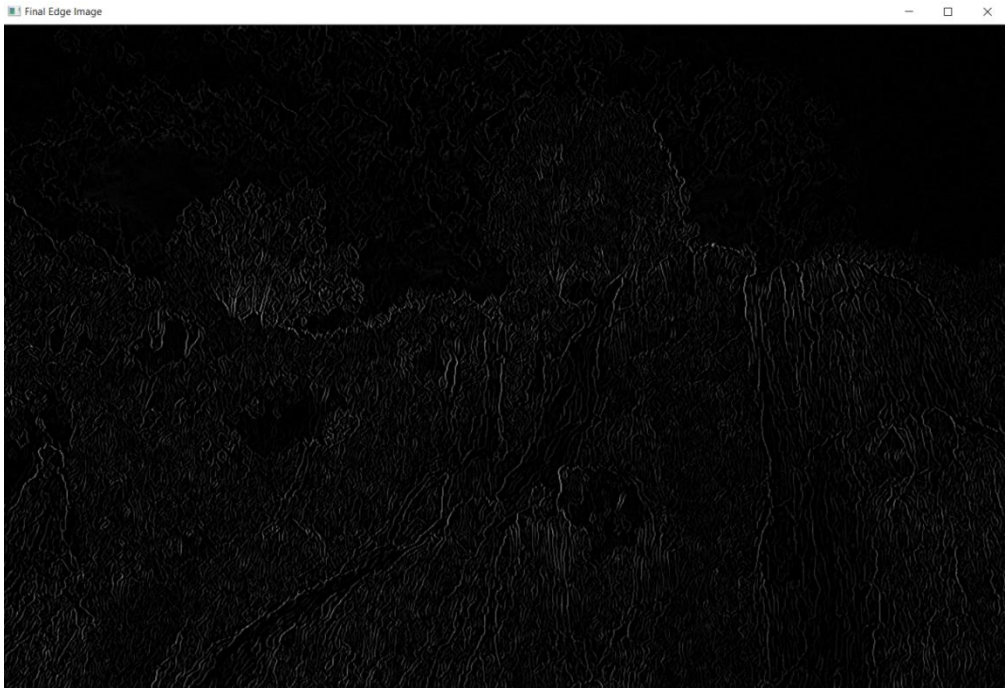


for 0 to 180

For Final edge image:



for 45 to 180



for 0 to 180