Visión Computacional para imágenes y video (Gpo 10)

Alumnos:

- Armando Bringas Corpus - A01200230
- Guillermo Alfonso Muñiz Hermosillo - A01793101
- Jorge Luis Arroyo Chavelas - A01793023
- Samantha R Mancias Carrillo - A01196762
- Sofia E Mancias Carrillo - A01196563

Profesores:

- Dr. Gilberto Ochoa Ruiz
- Mtra. Yetnalezi Quintas Ruiz

3. Sobel and Canny Edge Detection

Table of Contents

Importing Libraries

In [1]:
```
import cv2
import skimage
import numpy as np
from scipy import ndimage
from skimage import exposure
from PIL import Image, ImageOps
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from scipy.ndimage import convolve
from scipy.ndimage import gaussian_filter as gauss
from scipy.ndimage import median_filter as med
```

Sobel Edge Detection

As a first step in extracting features, you will apply the Sobel edge detection algorithm. This finds regions of the image with large gradient values in multiple directions. Regions with high omnidirectional gradient are likely to be edges or transitions in the pixel values.

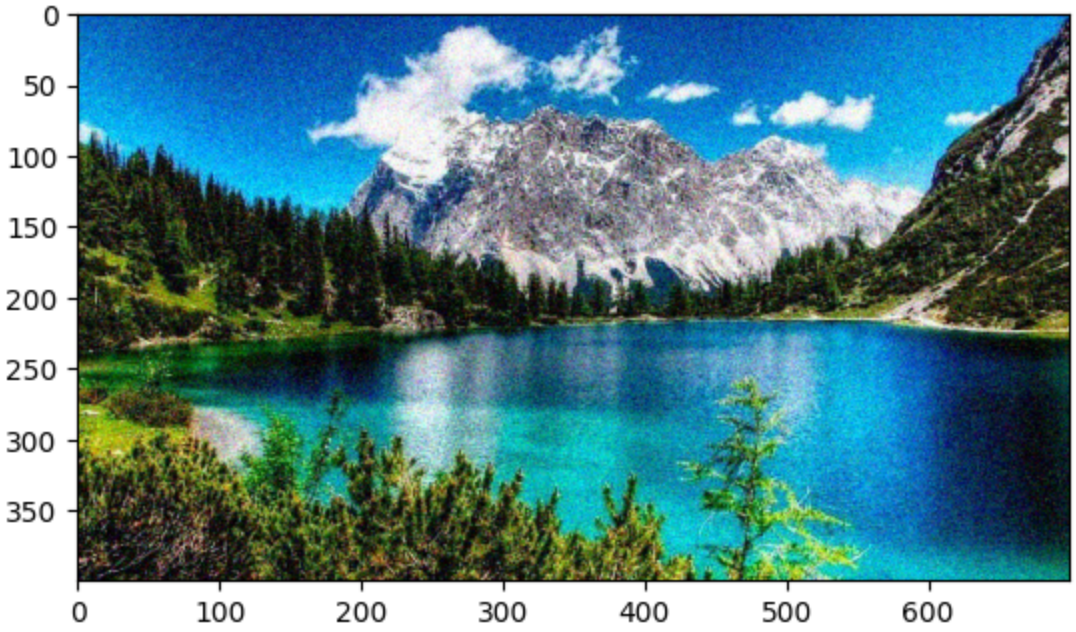
The code in the cell below applies the Sobel algorithm to the median filtered image, using these steps:

1. Convert the color image to grayscale for the gradient calculation since it is two dimensional.
2. Compute the gradient in the x and y (horizontal and vertical) directions.
3. Compute the magnitude of the gradient.
4. Normalize the gradient values.

In [2]:
```
def edge_sobel(image):
    from scipy import ndimage
    import skimage.color as sc
    import numpy as np
    image = sc.rgb2gray(image) # Convert color image to gray scale
    dx = ndimage.sobel(image, 1)  # horizontal derivative
    dy = ndimage.sobel(image, 0)  # vertical derivative
    mag = np.hypot(dx, dy)  # magnitude
    mag *= 255.0 / np.amax(mag)  # normalize (Q&D)
    mag = mag.astype(np.uint8)
    return mag
```

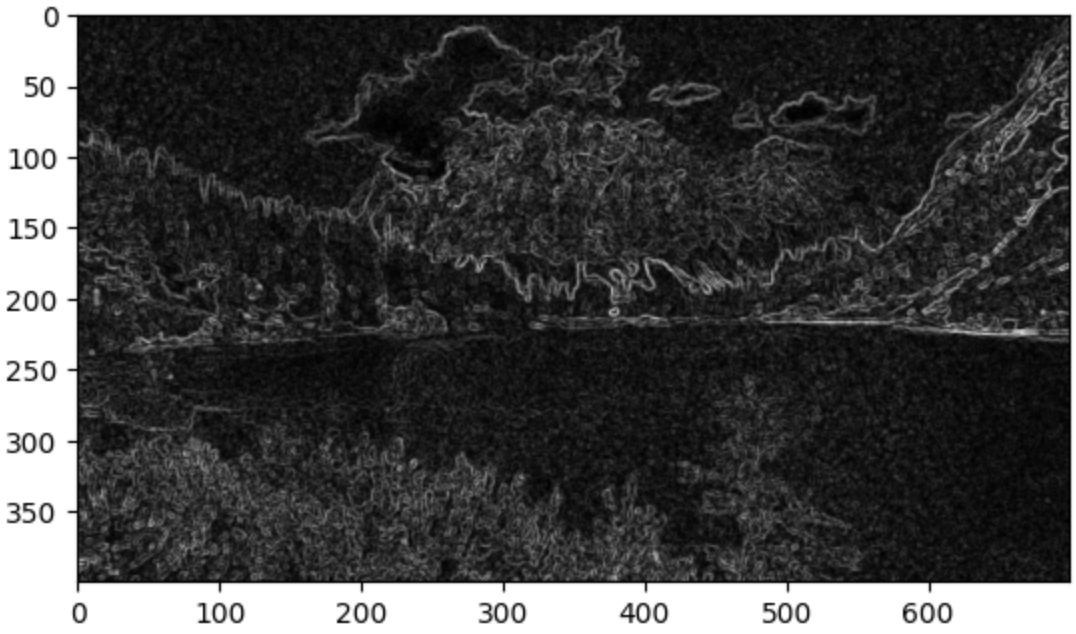
In [3]:
```
original_image = np.load('data/img.npy')

img = skimage.util.random_noise(original_image)
plt.imshow(img)
```

Out[3]:
```
<matplotlib.image.AxesImage at 0x7fa27a285f40>
```



In [4]:
```
img_med = med(img, size=2)
img_edge = edge_sobel(img_med)
plt.imshow(img_edge, cmap="gray")
```

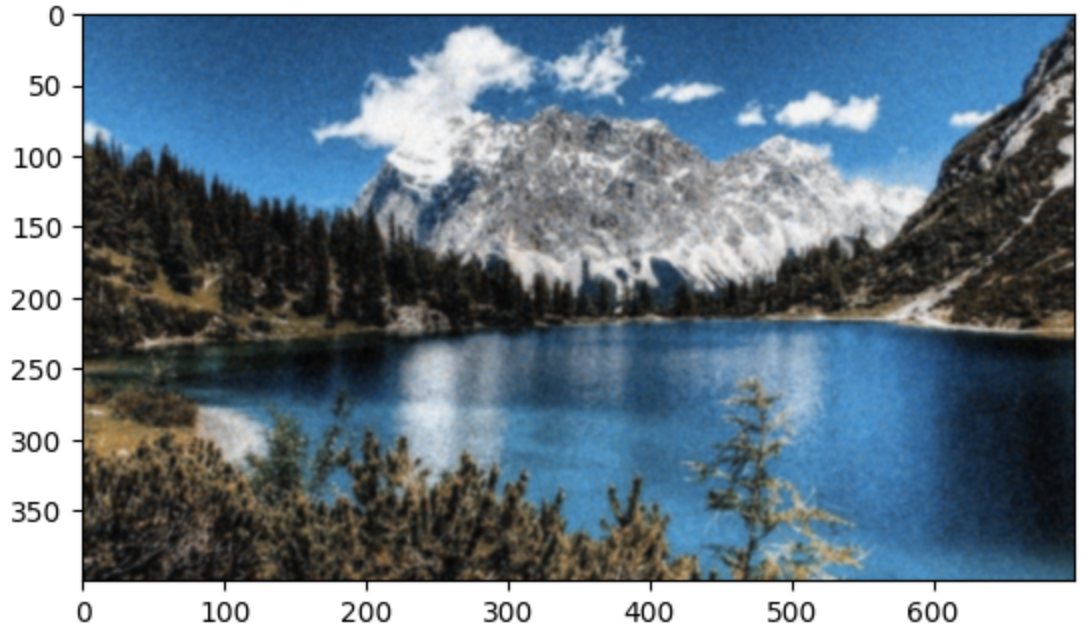
Out[4]:
```
<matplotlib.image.AxesImage at 0x7fa278293400>
```



Now let's try with the more blurred gaussian filtered image.

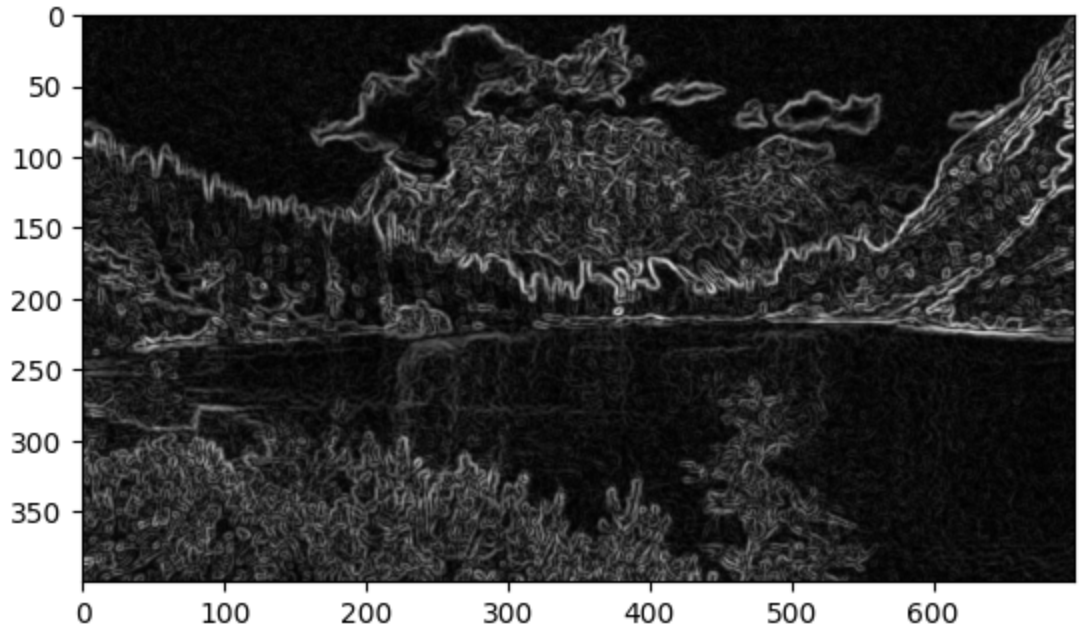
In [5]:
```
img_gauss = gauss(img, sigma=1)
plt.imshow(img_gauss)
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x7fa278db0fd0>
```



In [6]:
```
img_edge = edge_sobel(img_gauss)
plt.imshow(img_edge, cmap="gray")
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x7fa278458b50>
```



Canny Edge Detection

Steps:

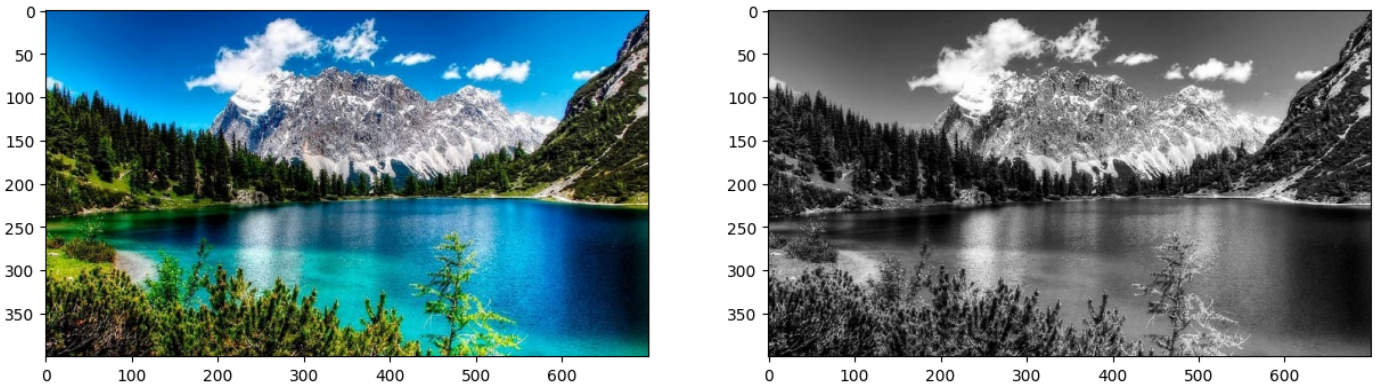
1. Noise Reduction
2. Gradient Calculation
3. Non-maximum Supression
4. Double Threshold
5. Edge Tracking by Hysteresis

Pre-requisite: Convert the image to grayscale before algorithm.

In [7]:
```
img = cv2.imread('data/image.jpg')
```

```
img_color = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

plt.figure(figsize=(15, 8))
plt.subplot(1, 2, 1)
plt.imshow(img_color)
plt.subplot(1, 2, 2)
plt.imshow(img_gray, cmap="gray")
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x7fa278d7b700>
```



1. Noise Reduction

Edge detection are highly sensitive to image noise due to the derivatives behind the algorithm.

We can apply a Gaussian Kernel, the size of the kernel depends on the expected blurring effect. The smaller the less blurring effect.

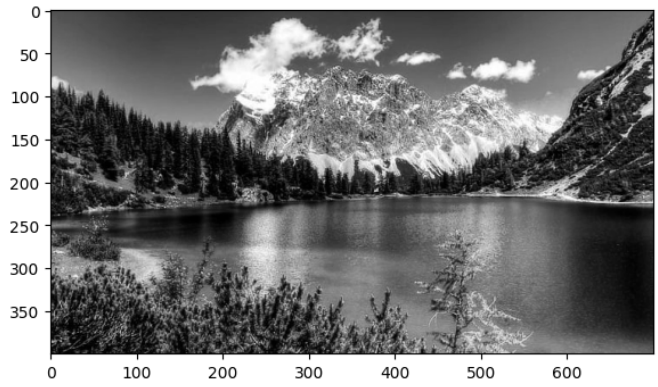
Equation for Gaussian Kernel of size $(2k+1) \times (2k+1)$

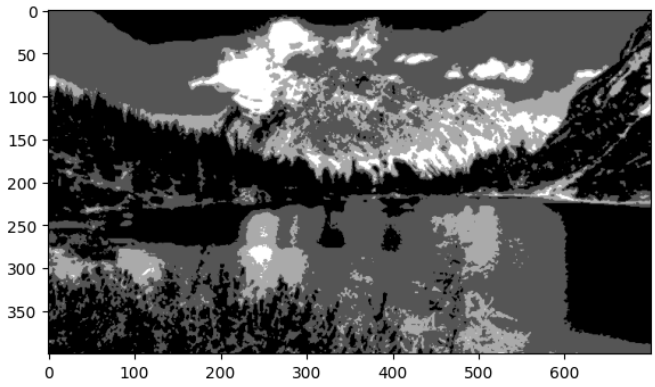
$$H_{i,j} = \frac{1}{2\pi\sigma^2} exp(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}); 1 \leq i, j \leq (2k+1)$$

In [8]:
```
def gaussian_kernel(size, sigma=1):
    size = int(size) // 2
    x, y = np.mgrid[-size:size+1, -size:size+1]
    normal = 1 / (2.0 * np.pi * sigma**2)
    g =  np.exp(-((x**2 + y**2) / (2.0*sigma**2))) * normal
    return g
```

In [9]:
```
img_gaussian = convolve(img_gray, gaussian_kernel(3, 10))

plt.figure(figsize=(15, 8))
plt.subplot(1, 2, 1)
plt.imshow(img_gray, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(img_gaussian, cmap="gray")
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x7fa2795d90d0>
```





2. Gradient Calculation

Edges correspond to a change of pixels intensity.

To detect it, the easiest way is to apply filters that highlight this intensity change in both directions:

- horizontal (x)
- and vertical (y)

It can be implemented by convoling I with *Sobel kernels* Kx and Ky

$$K_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, K_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

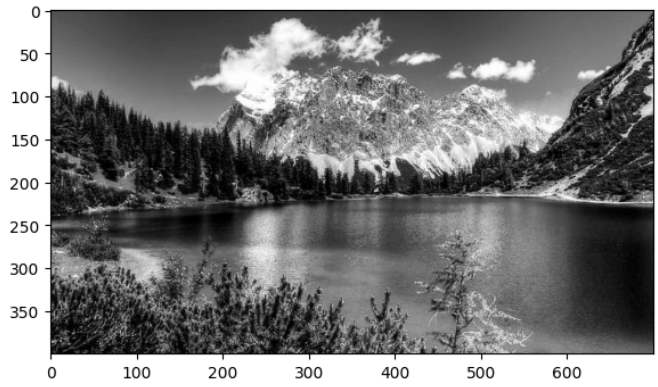
Then, the magnitude G and the slope θ of the gradient are calculated as follow:

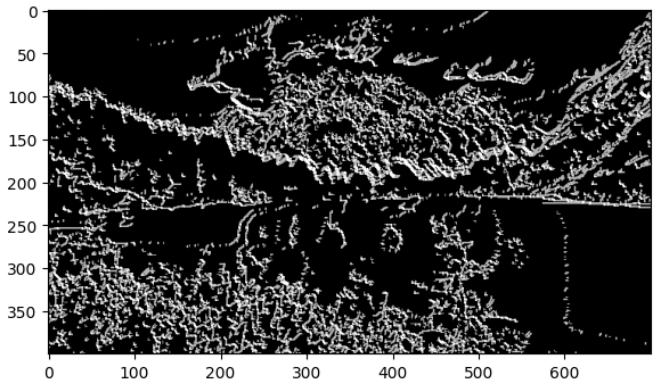
$$|G| = \sqrt{I_x^2 + I_y^2}, \theta(x,y) = arctan(\frac{I_y}{I_x})$$

In [10]:
```
def sobel_filters(img):
    Kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)
    Ky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]], np.float32)

    Ix = ndimage.convolve(img, Kx)
    Iy = ndimage.convolve(img, Ky)

    G = np.hypot(Ix, Iy)
    G = G / G.max() * 255
    theta = np.arctan2(Iy, Ix)

    return (G, theta)
```

In [11]:
```
G, theta = sobel_filters(img_gaussian)

plt.figure(figsize=(15, 8))
plt.subplot(1, 2, 1)
plt.imshow(img_gray, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(G, cmap="gray")
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x7fa27b455790>
```





3. Non-Maximum supression

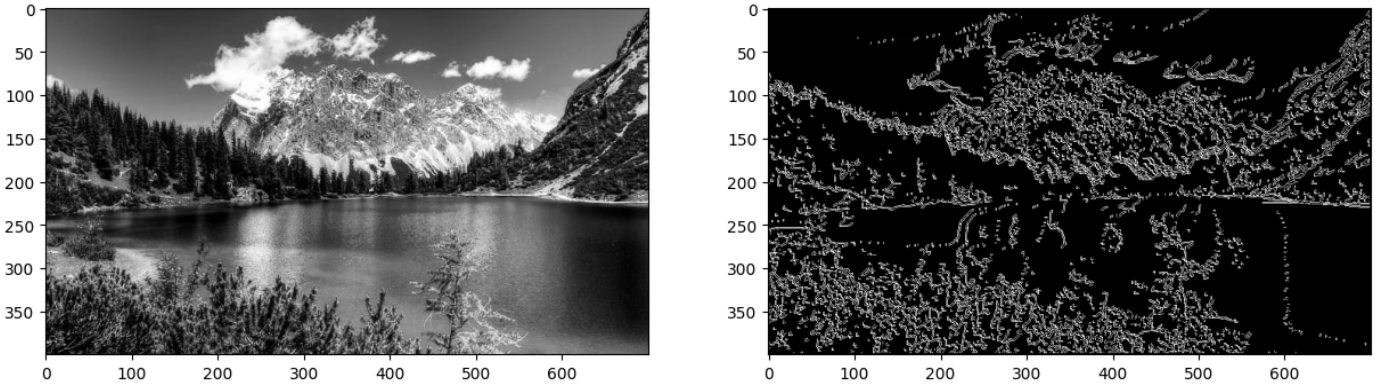
1. Create a matrix initialized to 0 of the same size of the original gradient intensity matrix
2. Identify the edge direction based on the angle value from the angle matrix
3. Check if the pixel in the same direction has a higher intensity than the pixel that is currently processed
4. Return the image processed with the non-max suppression algorithm.

In [12]:
```
def non_max_suppression(img, D):
    M, N = img.shape
    Z = np.zeros((M,N), dtype=np.int32)
    angle = D * 180. / np.pi
    angle[angle < 0] += 180


    for i in range(1,M-1):
        for j in range(1,N-1):
            try:
                q = 255
                r = 255

               #angle 0
                if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <= 180):
                    q = img[i, j+1]
                    r = img[i, j-1]
                #angle 45
                elif (22.5 <= angle[i,j] < 67.5):
                    q = img[i+1, j-1]
                    r = img[i-1, j+1]
                #angle 90
                elif (67.5 <= angle[i,j] < 112.5):
                    q = img[i+1, j]
                    r = img[i-1, j]
                #angle 135
                elif (112.5 <= angle[i,j] < 157.5):
                    q = img[i-1, j-1]
                    r = img[i+1, j+1]

                if (img[i,j] >= q) and (img[i,j] >= r):
                    Z[i,j] = img[i,j]
                else:
                    Z[i,j] = 0

            except IndexError as e:
                pass

    return Z
```

In [13]:
```
img_nonmax = non_max_suppression(G, theta)
```

```
plt.figure(figsize=(15, 8))
plt.subplot(1, 2, 1)
plt.imshow(img_gray, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(img_nonmax, cmap="gray")
```

Out[13]:
```
<matplotlib.image.AxesImage at 0x7fa27c611a60>
```



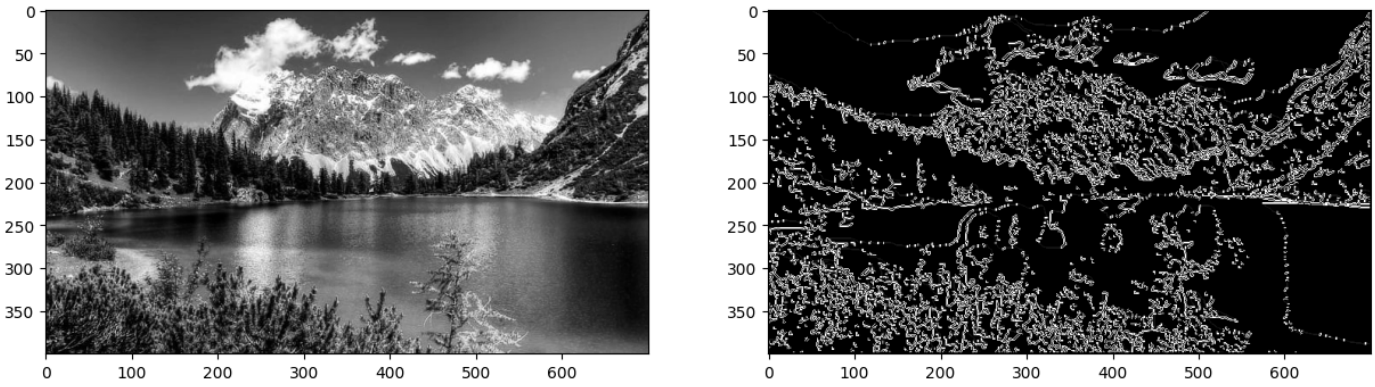
4. Double threshold

- Strong pixels are pixels that have an intensity so high that we are sure they contribute to the final edge.
- Weak pixels are pixels that have an intensity value that is not enough to be considered as strong ones, but yet not small enough to be considered as non-relevant for the edge detection.
- Other pixels are considered as non-relevant for the edge.

In [14]:
```
def threshold(img, lowThresholdRatio=0.05, highThresholdRatio=0.09):

    highThreshold = img.max() * highThresholdRatio;
    lowThreshold = highThreshold * lowThresholdRatio;

    M, N = img.shape
    res = np.zeros((M,N), dtype=np.int32)

    weak = np.int32(25)
    strong = np.int32(255)

    strong_i, strong_j = np.where(img >= highThreshold)
    zeros_i, zeros_j = np.where(img < lowThreshold)

    weak_i, weak_j = np.where((img <= highThreshold) & (img >= lowThreshold))

    res[strong_i, strong_j] = strong
    res[weak_i, weak_j] = weak

    return (res)
```

In [15]:
```
img_threshold = threshold(img_nonmax)

plt.figure(figsize=(15, 8))
plt.subplot(1, 2, 1)
plt.imshow(img_gray, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(img_threshold, cmap="gray")
```

Out[15]:
```
<matplotlib.image.AxesImage at 0x7fa27c79f610>
```



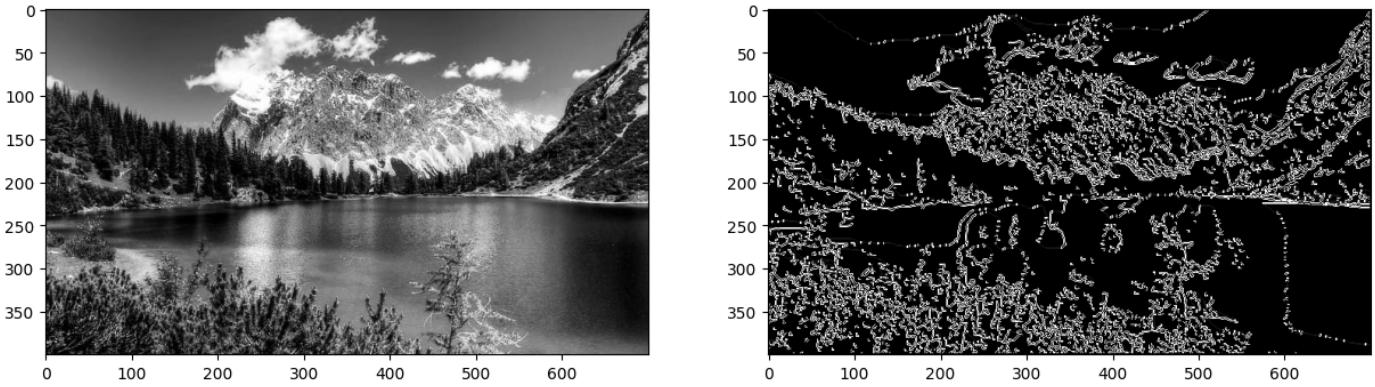
5. Edge Tracking by Hysteresis

The hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one

In [16]:
```
def hysteresis(img, weak = 75, strong=255):
    M, N = img.shape
    for i in range(1, M-1):
        for j in range(1, N-1):
            if (img[i,j] == weak):
                try:
                    if ((img[i+1, j-1] == strong) or (img[i+1, j] == strong) or (img[i+1
                        or (img[i, j-1] == strong) or (img[i, j+1] == strong)
                        or (img[i-1, j-1] == strong) or (img[i-1, j] == strong) or (img[
                        img[i, j] = strong
                    else:
                        img[i, j] = 0
                except IndexError as e:
                    pass
    return img
```

In [17]:
```
img_final = hysteresis(img_threshold, 150, 255)

plt.figure(figsize=(15, 8))
plt.subplot(1, 2, 1)
plt.imshow(img_gray, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(img_final, cmap="gray")
```

Out[17]:
```
<matplotlib.image.AxesImage at 0x7fa25df5b550>
```



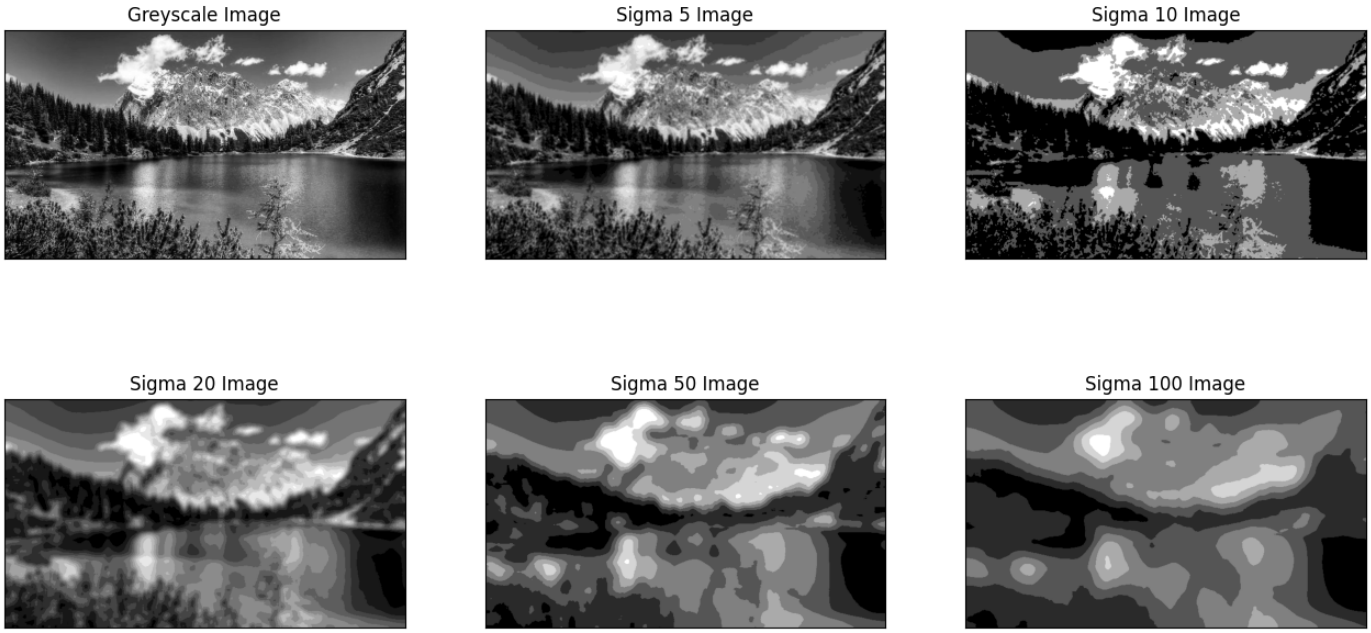
Exercises

In [18]:
```
# Function to plot original, grayscaled and Edge Detected Images with Different Sigmas
def plot_images_sigmaChanges(gray, noisy5, noisy10, noisy20, noisy50, noisy100):
```

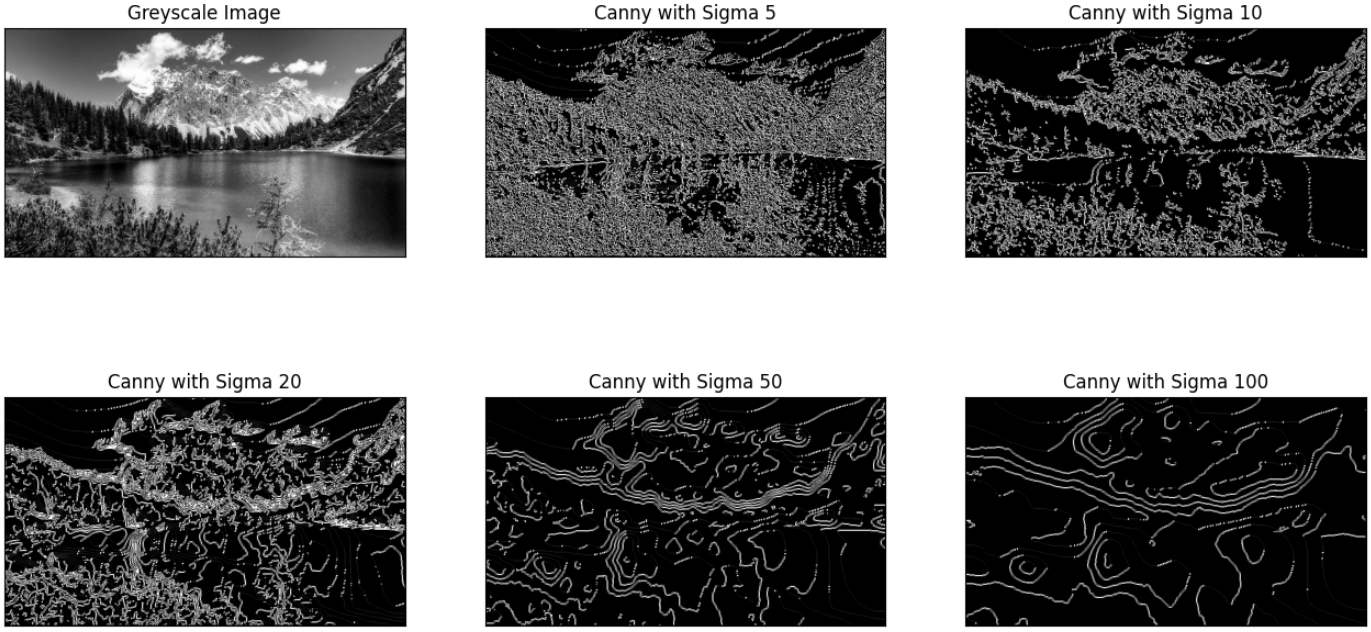
```
    plt.figure(figsize=(16,8))
    plt.subplot(2,3,1),plt.imshow(gray["img"], cmap=plt.get_cmap('gray'))
    plt.title(gray["title"]), plt.xticks([]), plt.yticks([])
    plt.subplot(2,3,2),plt.imshow(noisy5["img"], cmap=plt.get_cmap('gray'))
    plt.title(noisy5["title"]), plt.xticks([]), plt.yticks([])
    plt.subplot(2,3,3),plt.imshow(noisy10["img"], cmap=plt.get_cmap('gray'))
    plt.title(noisy10["title"]), plt.xticks([]), plt.yticks([])
    plt.subplot(2,3,4),plt.imshow(noisy20["img"], cmap=plt.get_cmap('gray'))
    plt.title(noisy20["title"]), plt.xticks([]), plt.yticks([])
    plt.subplot(2,3,5),plt.imshow(noisy50["img"], cmap=plt.get_cmap('gray'))
    plt.title(noisy50["title"]), plt.xticks([]), plt.yticks([])
    plt.subplot(2,3,6),plt.imshow(noisy100["img"], cmap=plt.get_cmap('gray'))
    plt.title(noisy100["title"]), plt.xticks([]), plt.yticks([])
```

A) Modify Sigma Value and Observe detected lines

In [19]:
```
noisy5 = {"img": convolve(img_gray, gaussian_kernel(2, 5)) , "title": "Sigma 5 Image"}
noisy10 = {"img": convolve(img_gray, gaussian_kernel(3, 10)), "title": "Sigma 10 Image"}
noisy20 = {"img": convolve(img_gray, gaussian_kernel(10, 20)), "title": "Sigma 20 Image"
noisy50 = {"img": convolve(img_gray, gaussian_kernel(20, 50)) , "title": "Sigma 50 Image
noisy100 = {"img":convolve(img_gray, gaussian_kernel(40, 100)) , "title": "Sigma 100 Ima


plot_images_sigmaChanges({"img": img_gray, "title": "Greyscale Image"}, noisy5, noisy10,
```



In [20]:
```
def apply_Canny(noisy_img, minThreshold = 75, maxThreshold = 255):
    gMag, slope = sobel_filters(noisy_img)
    nms = non_max_suppression(gMag, slope)
    dbthres = threshold(nms)
    return hysteresis(dbthres, minThreshold, maxThreshold)
plot_images_sigmaChanges(
    {"img": img_gray, "title": "Greyscale Image"},
    {"img": apply_Canny(noisy5["img"]), "title": "Canny with Sigma 5"},
    {"img": apply_Canny(noisy10["img"]), "title": "Canny with Sigma 10"},
    {"img": apply_Canny(noisy20["img"]), "title": "Canny with Sigma 20"},
    {"img": apply_Canny(noisy50["img"]), "title": "Canny with Sigma 50"},
    {"img": apply_Canny(noisy100["img"]), "title": "Canny with Sigma 100"})
```

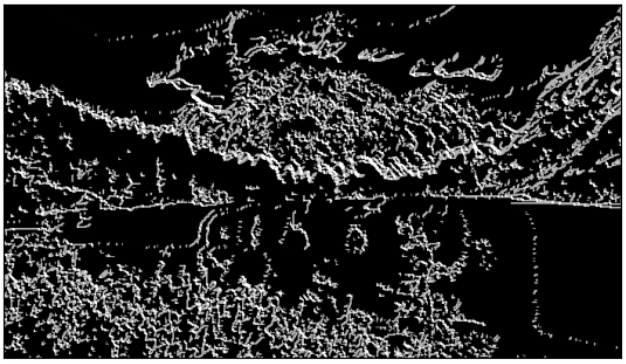


B) Observe Canny Detector behavior with and without noise reduction

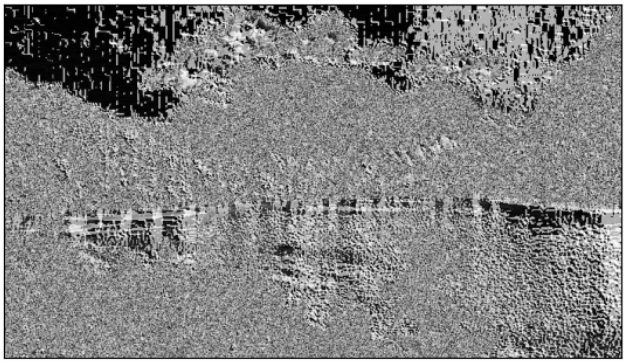
In [21]:
```
# In the gray image we apply the Sobel Filter directly, no noise added (skipping step 1
# We go directly to Step 2.
GmagOriginal, slopeOriginal = sobel_filters(img_gray)

# Once we got the G Magnitud and the Slop we proceed to step 3, obtaining the NMS for th
orig_nonMax = non_max_suppression(GmagOriginal, slopeOriginal)

# After Obtaining the NMS we proceed to step 4, Applying the Double Threshold.
orig_threshold = threshold(orig_nonMax)

# Finally on step 5 we transform weak pixels to strong ones.
orig_final = hysteresis(orig_threshold, 150, 255)

plt.figure(figsize=(15,20))
plt.subplot(4,2,1),plt.imshow(G, cmap=plt.get_cmap('gray'))
plt.title('Sobel Filter for Noised Reduced Image'), plt.xticks([]), plt.yticks([])
plt.subplot(4,2,2),plt.imshow(GmagOriginal, cmap=plt.get_cmap('gray'))
plt.title('Sobel Filter for No Noise Reduction Image'), plt.xticks([]), plt.yticks([])
plt.subplot(4,2,3),plt.imshow(img_nonmax, cmap=plt.get_cmap('gray'))
plt.title('NMS For Noised Reduced Image'), plt.xticks([]), plt.yticks([])
plt.subplot(4,2,4),plt.imshow(orig_nonMax, cmap=plt.get_cmap('gray'))
plt.title('NMS For No Noise Reduction Image'), plt.xticks([]), plt.yticks([])
plt.subplot(4,2,5),plt.imshow(img_threshold, cmap=plt.get_cmap('gray'))
plt.title('Double Threshold for Noised Reduced Image'), plt.xticks([]), plt.yticks([])
plt.subplot(4,2,6),plt.imshow(orig_threshold, cmap=plt.get_cmap('gray'))
plt.title('Double Threshold for No Noise Reduction Image'), plt.xticks([]), plt.yticks([
plt.subplot(4,2,7),plt.imshow(img_final, cmap=plt.get_cmap('gray'))
plt.title('Hysteresis For Noised Reduced Image'), plt.xticks([]), plt.yticks([])
plt.subplot(4,2,8),plt.imshow(orig_final, cmap=plt.get_cmap('gray'))
plt.title('Hysteresis For No Noise Reduction Image'), plt.xticks([]), plt.yticks([])
```

Out[21]:
```
(Text(0.5, 1.0, 'Hysteresis For No Noise Reduction Image'), ([], []), ([], []))
```

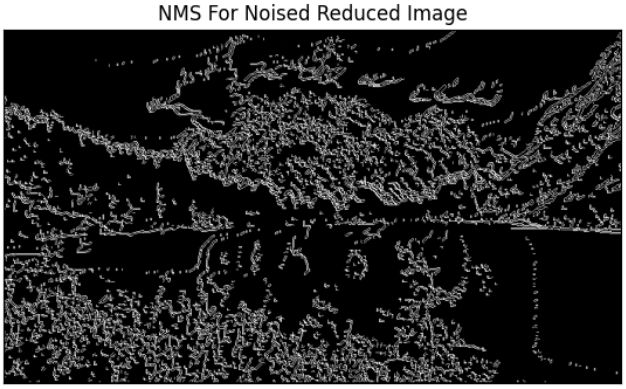
Sobel Filter for Noised Reduced Image



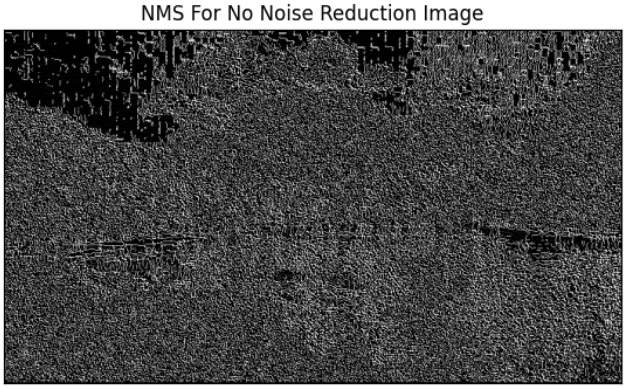
Sobel Filter for No Noise Reduction Image



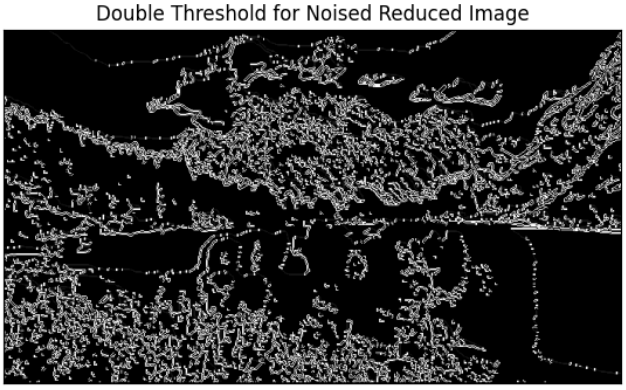
NMS For Noised Reduced Image



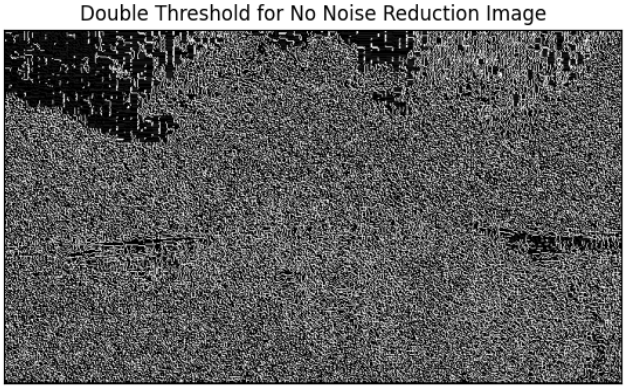
NMS For No Noise Reduction Image



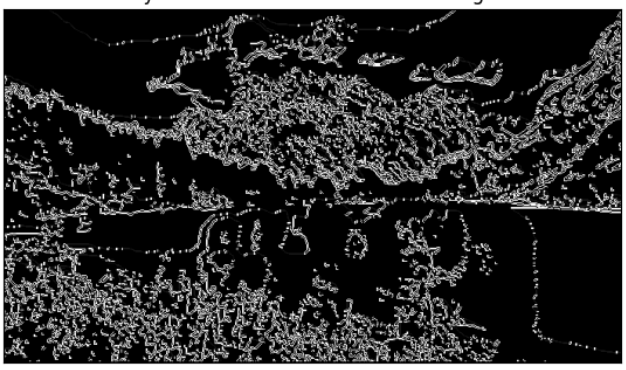
Double Threshold for Noised Reduced Image



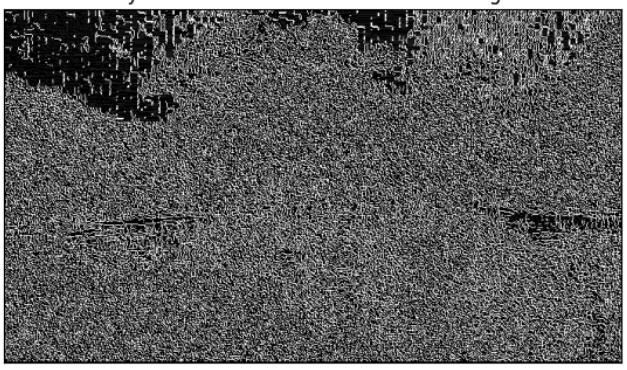
Double Threshold for No Noise Reduction Image



Hysteresis For Noised Reduced Image



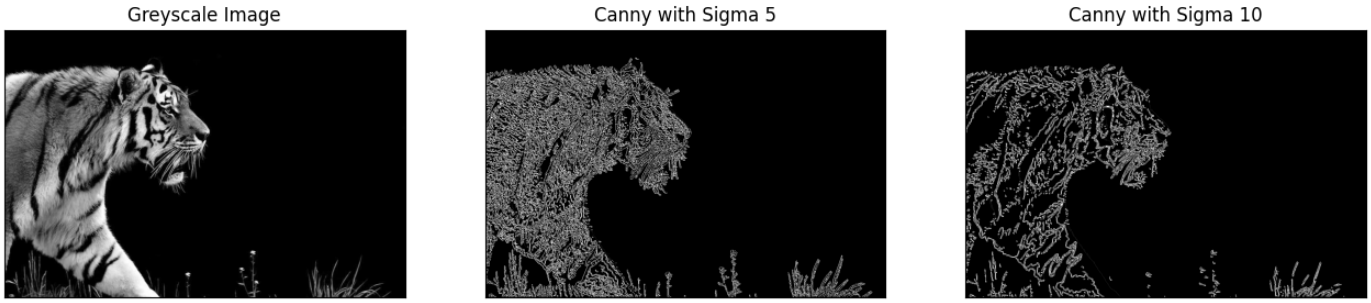
Hysteresis For No Noise Reduction Image

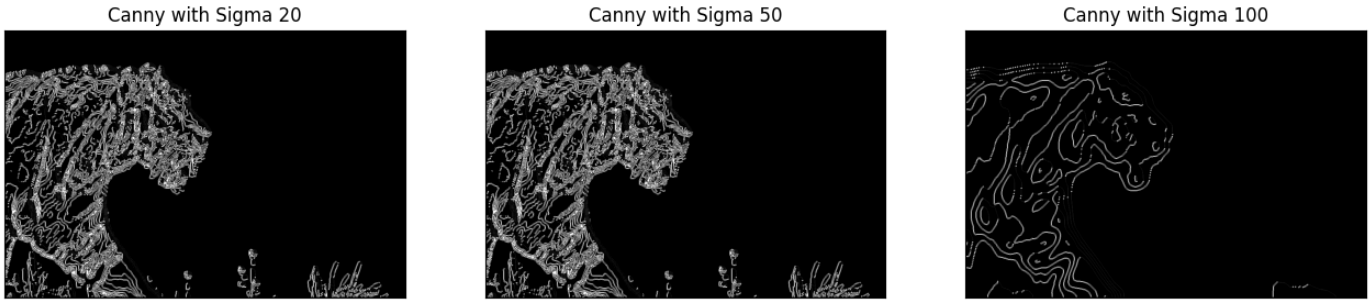


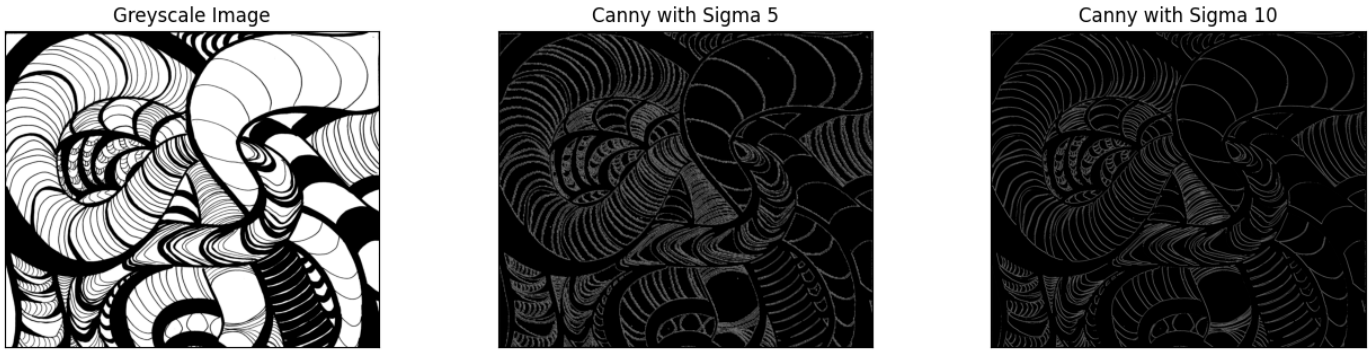
C) Use of images with different amount of lines and textures to observe the algorithm's behavior

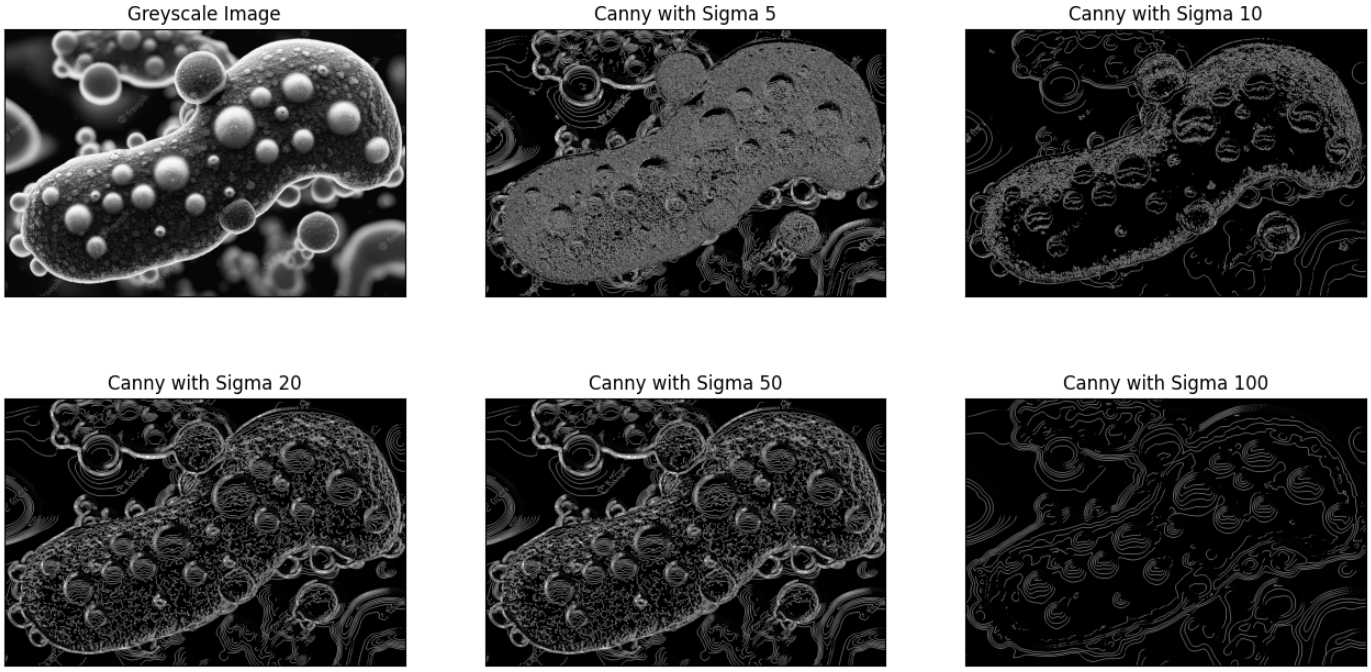
In [22]:
```
imgs_toRead = ['data/tiger.jpeg', 'data/textures1.jpg', 'data/microscope.jpeg']
```

```
for myimg in imgs_toRead:
    gray = {"img": cv2.cvtColor(cv2.imread(myimg), cv2.COLOR_BGR2GRAY), "title": "Greysc
    noisy5 = {"img": apply_Canny(convolve(gray['img'], gaussian_kernel(2, 5))), "title":
    noisy10 ={"img": apply_Canny(convolve(gray['img'], gaussian_kernel(3, 10))), "title"
    noisy20 = {"img": apply_Canny(convolve(gray['img'], gaussian_kernel(10, 20))), "titl
    noisy50 = {"img": apply_Canny(convolve(gray['img'], gaussian_kernel(10, 20))), "titl
    noisy100 = {"img": apply_Canny(convolve(gray['img'], gaussian_kernel(40, 100))) , "t
    plot_images_sigmaChanges(gray, noisy5, noisy10, noisy20, noisy50, noisy100)
```









Conclusions

As we saw in the examples the importance of the sigma parameter in the Canny algorithm relies in the control of the amount of the smoothing applied to the image before edge detection. Therefore, in the selection of the sigma value is important to find an equilibrium between the reduction of noise and the preservation of the relevant details in the image that should be kept, this could lead in a certain amount of experimentation y adjustment of the parameters for each image case.

Also, the selection of the optimal parameters in the hysteresis is important because it can help us to unite the fragment segments of the edges on an image which can improve to have a more accurate and robust edge detection.

References

[1] Gonzalez, R. C., & Woods, R. E. (2008). Digital Image Processing (3rd ed.). Pearson Education.