Abdullah Qarni

CODING SECTION

NOTE all code done in Jupiter

QUESTION 1:

```
import numpy as np
import scipy as sc
import timeit
import math
from scipy import ndimage
from skimage import filters
from matplotlib import pyplot as plt
from PIL import Image

# #1
def GaussianBlurImage(image, sigma):
    #open image as np array
    image = Image.open(image)
    image = np.asarray(image)

    filter_size = 2 * int(4 * sigma + 0.5) + 1
    #initialize filter
    gaussian_filter = np.zeros((filter_size, filter_size), np.float32)
    #from notes
    for i in range(filter_size):
        for j in range(filter_size):
            x = i - filter_size // 2
            y = j - filter_size // 2
            gaussian_filter[i, j] = 1 / (2 * np.pi * sigma ** 2) * np.exp(-(x ** 2
+ y ** 2)/(2 * sigma ** 2))
    im_filtered = np.zeros_like(image, dtype=np.float32)
    #print(gaussian_filter.sum())
    for c in range(3):
        im_filtered[:, :, c] = sc.ndimage.convolve(image[:, :, c], gaussian_filter,
mode='constant', cval=0)
    #plt.imshow(np.clip(im_filtered, 0, 255).astype(np.uint8))
    fil = np.clip(im_filtered, 0, 255).astype(np.uint8)
    image = Image.fromarray(fil)
    image.save('1.png')
```

OUTPUT:

QUESTION2:

```
# #2
def SeparableGaussianBlurImage(image, sigma):
    image = Image.open(image)
    image = np.asarray(image)

    filter_size = 2 * int(sigma * 4 + 0.5) + 1
    gauss_filter_x = np.zeros(filter_size, dtype=np.float32)
    gauss_filter_y = np.zeros(filter_size, dtype=np.float32)

    #same as above exppt we use 2 seperate loops for 2 seprate filters
    for i in range(filter_size):
        x = i - filter_size // 2
        gauss_filter_x[i] = 1.0 / np.sqrt((2 * np.pi * sigma ** 2)) * np.exp(-(x **
2)/(2 * sigma ** 2))
    for j in range(filter_size):
        y = j - filter_size // 2
        # sep guas fltr equation
        gauss_filter_y[j] = 1.0 / np.sqrt((2 * np.pi * sigma ** 2)) * np.exp(-(y **
```

```
2)/(2 * sigma ** 2))
    gauss_filter_x = np.expand_dims(gauss_filter_x, axis=0)  # or axis=1
    gauss_filter_y = np.expand_dims(gauss_filter_y, axis=1)  # or axis=1

    #plt.imshow(gauss_filter_y)

    im_filtered = np.zeros_like(image, dtype=np.float32)
    for c in range(3):
        im_filtered[:, :, c] = sc.ndimage.convolve(image[:, :, c], gauss_filter_x,
mode='constant', cval=0)
    for c in range(3):
        #convolve the filtered image with itself vertically
        im_filtered[:, :, c] = sc.ndimage.convolve(im_filtered[:, :, c],
gauss_filter_y, mode='constant', cval=0)
    #plt.imshow(np.clip(im_filtered, 0, 255).astype(np.uint8))
    fil = np.clip(im_filtered, 0, 255).astype(np.uint8)
    image = Image.fromarray(fil)
    image.save('2.png')
```

OUTPUT:

QUESTION3:

```
# #3
#remaking an separable gaussian filter that returns a filtered np image
def gauss(image, sigma):
    image = Image.open(image)
    image = np.asarray(image)

    filter_size = 2 * int(sigma * 4 + 0.5) + 1
    gauss_filter_x = np.zeros(filter_size, dtype=np.float32)
    gauss_filter_y = np.zeros(filter_size, dtype=np.float32)


    for i in range(filter_size):
        x = i - filter_size // 2
        gauss_filter_x[i] = 1.0 / np.sqrt((2 * np.pi * sigma ** 2)) *
np.exp(-(x ** 2)/(2 * sigma ** 2))
    for j in range(filter_size):
        y = j - filter_size // 2
        gauss_filter_y[j] = 1.0 / np.sqrt((2 * np.pi * sigma ** 2)) *
np.exp(-(y ** 2)/(2 * sigma ** 2))
    gauss_filter_x = np.expand_dims(gauss_filter_x, axis=0)  # or axis=1
    gauss_filter_y = np.expand_dims(gauss_filter_y, axis=1)  # or axis=1

    #plt.imshow(gauss_filter_y)

    im_filtered = np.zeros_like(image, dtype=np.float32)
    for c in range(3):
        im_filtered[:, :, c] = sc.ndimage.convolve(image[:, :, c],
gauss_filter_x, mode='constant', cval=0)
    for c in range(3):
        im_filtered[:, :, c] = sc.ndimage.convolve(im_filtered[:, :, c],
gauss_filter_y, mode='constant', cval=0)
    return im_filtered

def SharpenImage(image, sigma, alpha):
    filtered = gauss(image, sigma)

    image = Image.open(image)
    image = np.asarray(image)

    #sharpned = alpha(gaussian - image)
    sharpened = np.multiply(alpha, (np.subtract(filtered, image)))
    plt.imshow(np.clip(sharpened, 0, 255).astype(np.uint8))
```

```
    fil = np.clip(sharpened, 0, 255).astype(np.uint8)
    image = Image.fromarray(fil)
    image.save('4.png')


SharpenImage('./hw1_data/Yosemite.png', 1, 5)
```

OUTPUT:

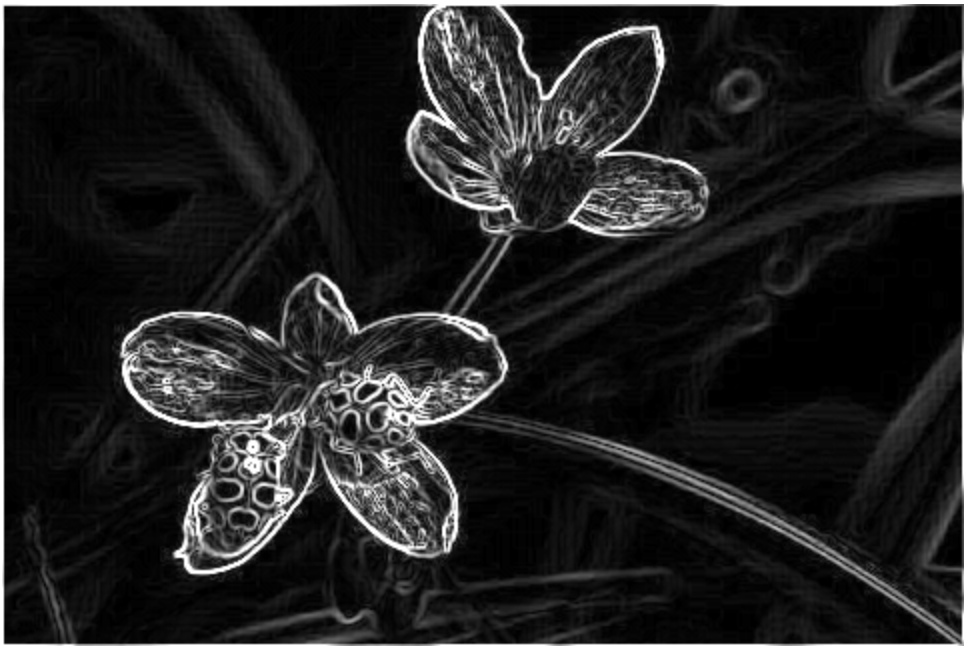
QUESTION 4:

```
#4
def SobelImage(image):
    image = Image.open(image)
    image = np.asarray(image)
    #making image grey scale
    R, G, B = image[:,:,0], image[:,:,1], image[:,:,2]
    imgGray = 0.2989 * R + 0.5870 * G + 0.1140 * B
    #plt.imshow(imgGray, cmap='gray')

    #declaring my 3x3 Gy and Gx matrices
    gx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]], np.float32)
    gy = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]], np.float32)
    #print(gx)
    #print(gy)
    #declaring 2 np arrays for the 2 seperate filteres (Gx, Gy)
    gx_filter = np.zeros_like(imgGray, dtype=np.float32)
    gy_filter = np.zeros_like(imgGray, dtype=np.float32)
```

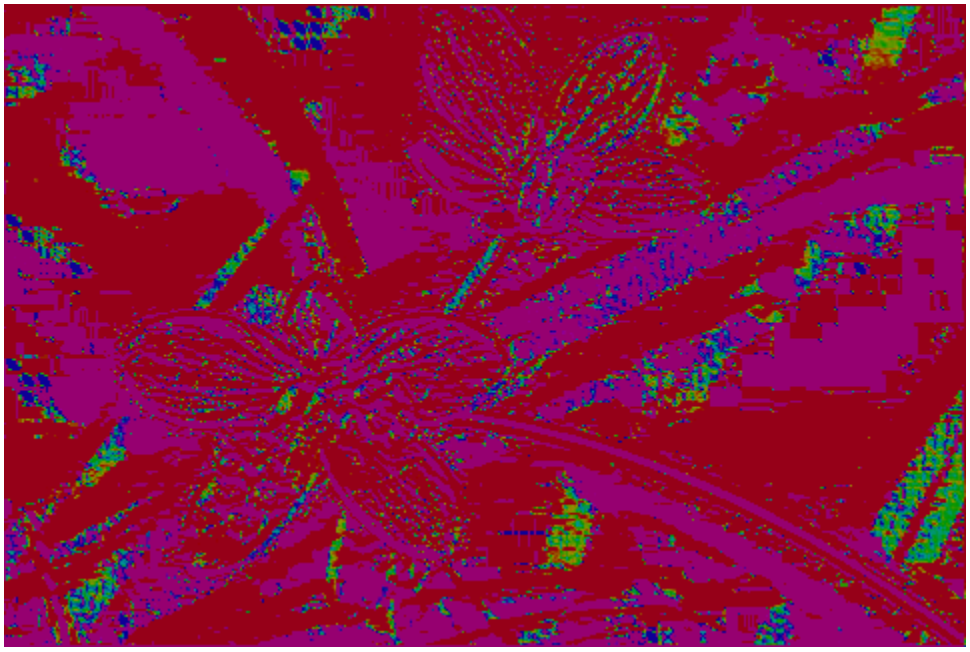
```
    #filter here
    for c in range(2):
        gx_filter[:, :] = sc.ndimage.convolve(imgGray[:, :], gx,
mode='constant', cval=0)
    for c in range(2):
        gy_filter[:, :] = sc.ndimage.convolve(imgGray[:, :], gy,
mode='constant', cval=0)

    #plt.imshow(np.clip(gy_filter, 0, 255).astype(np.uint8))

    #now compute the magnitude
    magnitude = np.zeros_like(imgGray, dtype=np.float32)
    for i in range(imgGray.shape[0]):
        for j in range(imgGray.shape[1]):
            #magnitude formula from slides
            magnitude[i, j] = np.sqrt((gx_filter[i,j]**2) +
(gy_filter[i,j]**2))
    #plt.imshow(np.clip(magnitude, 0, 255).astype(np.uint8))
    fil = np.clip(magnitude, 0, 255).astype(np.uint8)
    image = Image.fromarray(fil)
    image.save('5a.png')

    #compute the oreintation
    theta = np.zeros_like(imgGray, dtype=np.float32)
    for i in range(imgGray.shape[0]):
        for j in range(imgGray.shape[1]):
            # theta = arctan(gy/gx)
            theta[i, j] = np.arctan2(gy_filter[i,j],gx_filter[i,j])
    #plt.imshow(np.clip(theta, 0, 255).astype(np.uint8))
    #color map
    cm = plt.get_cmap('gist_rainbow')
    colored_image = cm(theta)
    Image.fromarray((colored_image[:, :, :3] *
150).astype(np.uint8)).save('5b.png')


SobelImage('./hw1_data/LadyBug.jpg')
```

OUTPUT:

5A:



5B:



QUESTION 5a:

```
def NearestNeighborInterpolation(image, x, y):
    #open image
    image = Image.open(image)
    image = np.asarray(image)
    #plt.imshow(np.clip(image, 0, 255).astype(np.uint8))
    #initialize output matrix
    output = np.zeros( (image.shape[0]*4,image.shape[1]*4, 3) )
```

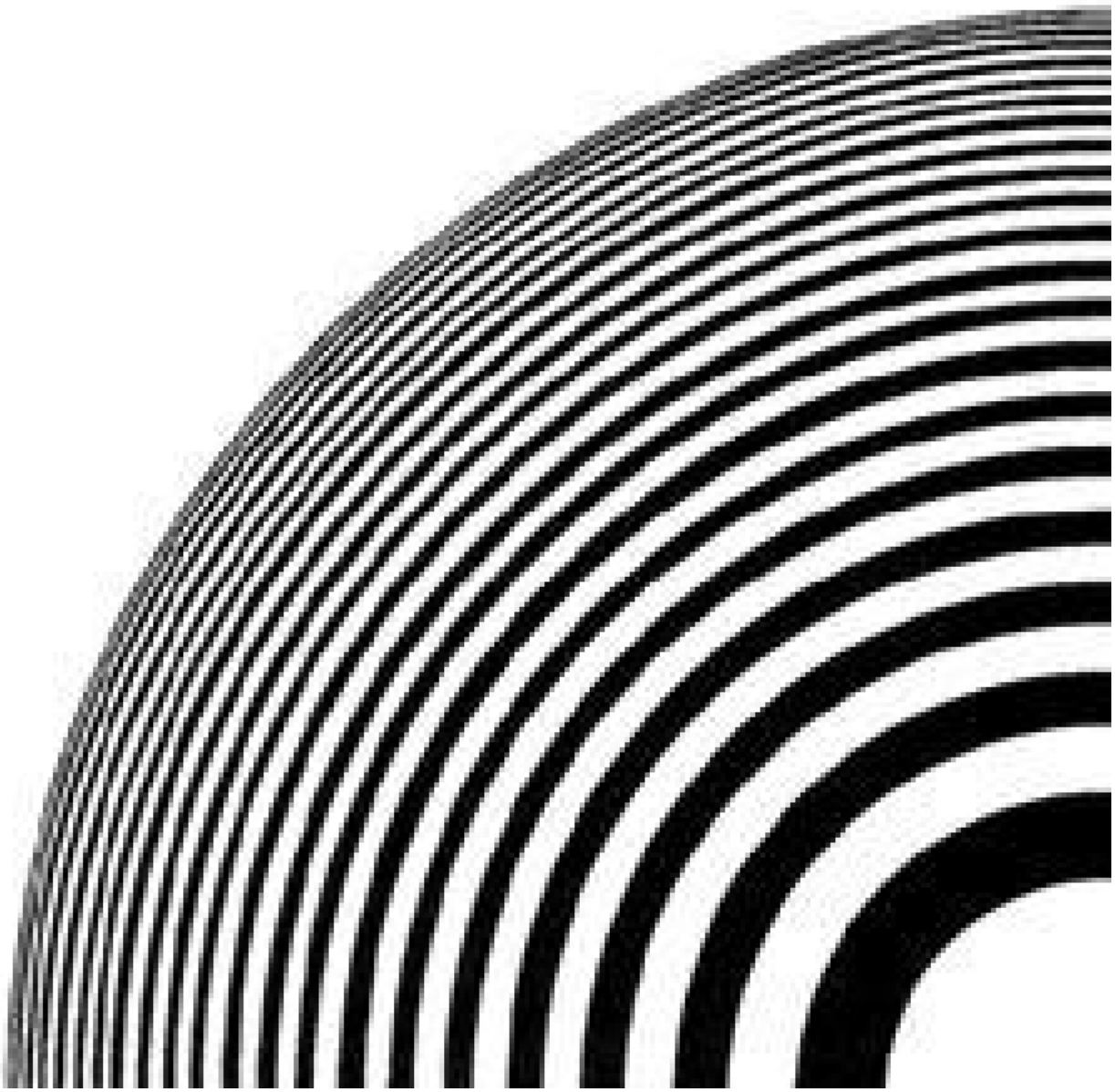
```
    for j in range(len(output)):
        for i in range(len(output[j])):
            #scaling in this case .25 which is 4x
            proj_x = math.floor(i*x)
            proj_y = math.floor(j*y)
            output[j][i] = image[proj_y][proj_x]
    plt.imshow(np.clip(output, 0, 255).astype(np.uint8))
    fil = np.clip(output, 0, 255).astype(np.uint8)
    image = Image.fromarray(fil)
    image.save('6a.png')
NearestNeighborInterpolation('./hw1_data/Moire_small.jpg', 0.25, 0.25)
```

5B:

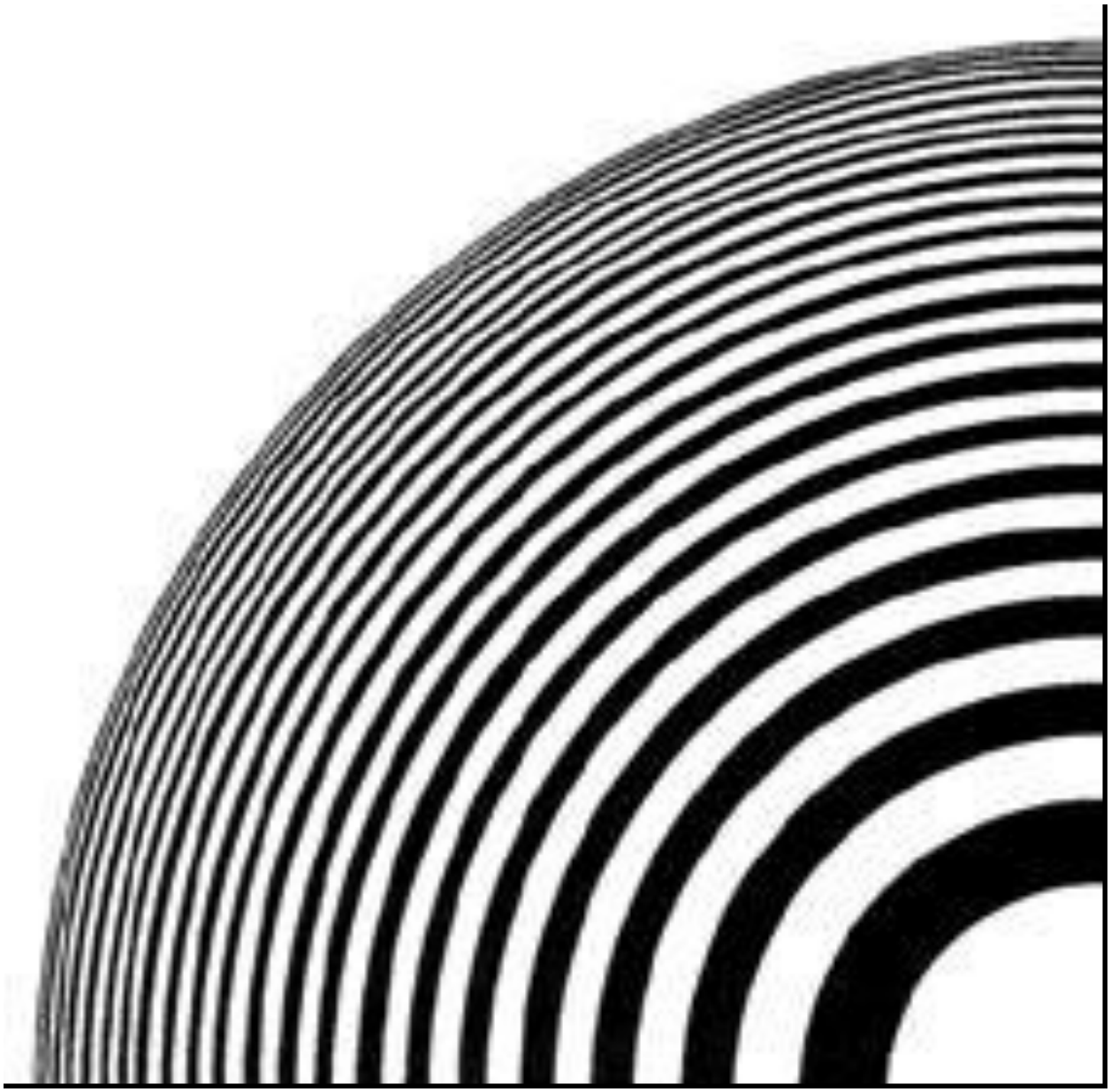
```
# #5
#this function for earch index evaluates the BilinearInterpolation
#we use this function at each pixel in the image to upcasle with less
aliasing
def BI(x, y, points):
#https://stackoverflow.com/questions/8661537/how-to-perform-bilinear-interp
olation-in-python

    points = sorted(points)               # order points by x, then by y
    (x1, y1, q11), (_x1, y2, q12), (x2, _y1, q21), (_x2, _y2, q22) = points

    if x1 != _x1 or x2 != _x2 or y1 != _y1 or y2 != _y2:
        raise ValueError('points do not form a rectangle')
    if not x1 <= x <= x2 or not y1 <= y <= y2:
        raise ValueError('(x, y) not within the rectangle')

    return (q11 * (x2 - x) * (y2 - y) +
            q21 * (x - x1) * (y2 - y) +
            q12 * (x2 - x) * (y - y1) +
            q22 * (x - x1) * (y - y1)
           ) / ((x2 - x1) * (y2 - y1) + 0.0)


def BilinearInterpolation(image, x, y):
    image = Image.open(image)
    image = np.asarray(image)

    output = np.zeros( (image.shape[0]*4,image.shape[1]*4, 3) )
    #-4 because I was runing into some bound issues, and we are dividing by
4
```

```
    for i in range(output.shape[0]-4):
        for j in range(output.shape[1]-4):
            # these are the x/y cordinates
            proj_x = (i*x)
            proj_y = (j*y)
            #math.floor is the lower bound
            x0 = math.floor(proj_x)
            y0 = math.floor(proj_y)
            #this function takes some time, print staement is basically a
loading
            print("x0 = ", x0)
            #this is the vector that gets pssed into BI x0/y0+1 is upper
bound
            n = [(x0, y0, image[x0][y0]),
                 (x0, y0+1, image[x0][y0+1]),
                 (x0+1, y0, image[x0+1,y0]),
                 (x0+1, y0+1, image[x0+1][y0+1]),
                ]
            output[i][j] = BI(proj_x,proj_y, n)
    plt.imshow(np.clip(output, 0, 255).astype(np.uint8))
    fil = np.clip(output, 0, 255).astype(np.uint8)
    image = Image.fromarray(fil)
    image.save('6b.png')

BilinearInterpolation('./hw1_data/Moire_small.jpg', 0.25, 0.25)
```

OUTPUT:
File 6a:



File 6b:



WRITTEN SECTION

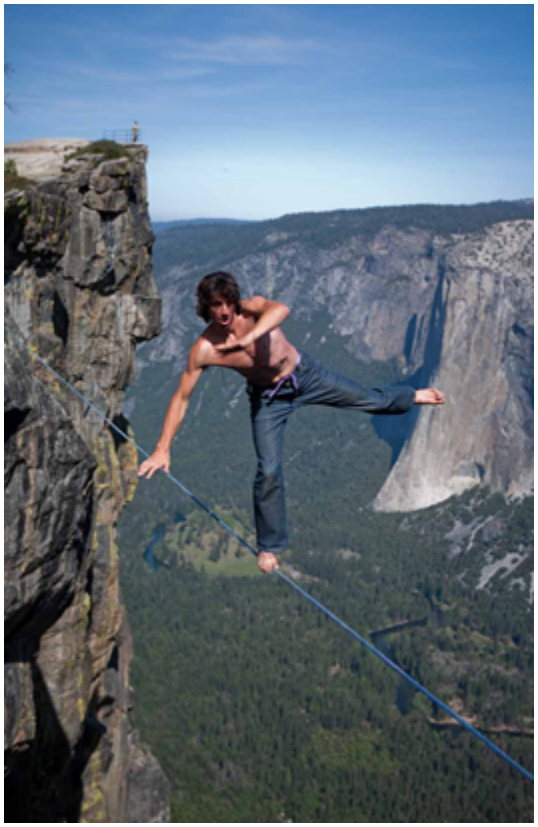
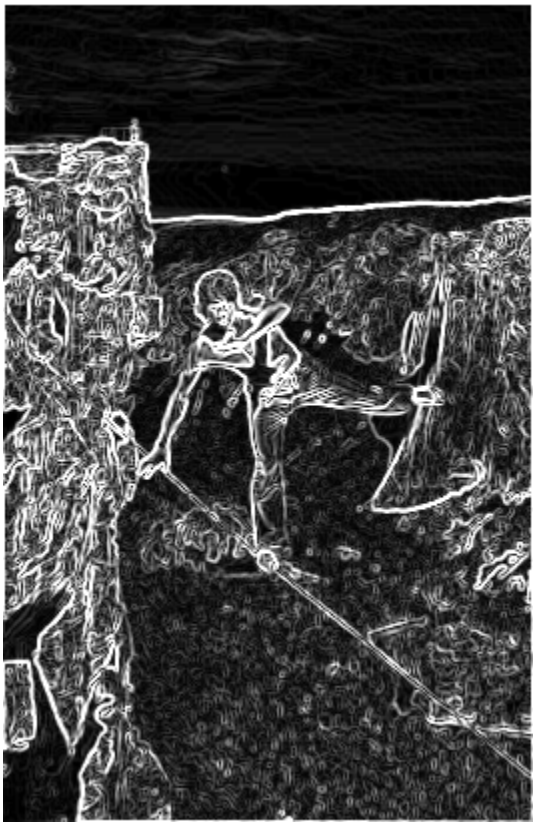
Question1:

SIGMA	Gausian (time in secs)	Sep Gausian (time in secs)
2	0.7656475000003411	0.3138857000012649
4	2.1704167000007146	0.34582310000041616
8	7.918284800000038	0.4189799999985553

For sigma 32 I think regular gaussian mayl take upwards of a couple mins. The trend looks to be that for each doubling of sigma the time it takes for execution goes up by ^x+1 so if sigma 8 is ^3 then 16 will be ^4 and 32 will be ^5. The time it takes most definitely increases exponentially. When trying to run sigma 32 my jupiter host crashed, the filter size is simply massive.
Sigma 32 when doing the separate gaussian will take less than a second, we are doing a couple orders of magnitude less computation with the separate gaussian compared to the regular. When running it the time took around .6-.7 secs.

Question2:
According to Sampling Notes slide 21, Filter size should double for each ½ size reduction. This is regardless of the image in question in order to avoid aliasing. So for a ⅛ size image thats simply half 3 times we will double the filter 3 times, so 8x blur for both images.

Question3:

We notice that the rope in front of the rock face on the left side is very apparent to the human eye, but applying the filter does not seem to highlight it best. It looks like the rope gets lost in the sharp angles of the mountain.

Question 4:
I don't believe so because when rotating it cuts off part of the pixels that don't fit unless it's at 90 degree intervals. Will test.
Code for image rotation

```
OG_image = Image.open('./hw1_data/LadyBug.jpg')
rt_image = OG_image.rotate(40)
rt_image.save('rt40.png')
rt_20 = OG_image
for i in range(20):
    rt_20 = rt_20.rotate(2)
rt_20.save('rt2x20.png')
```


rt40.png


rt2x20.png