Homework 1

In [1]:
```
%load_ext nb_black
```

In [2]:
```
import math
import numpy as np
import scipy as sc
from scipy import ndimage
from skimage import filters
from PIL import Image
from IPython.display import display

import plotly.express as px
```

Section 1: Coding Assignment

Problem 1

In [3]:
```
image = Image.open("hw1_data/Seattle.jpg")  # .convert("L")
image = np.asarray(image)
print(image.shape)
px.imshow(image, color_continuous_scale="gray")
```

```
(600, 800, 3)
```

Gaussian filter is defined as:

$$G(\mathbf{x};\sigma^2) = \frac{1}{2\pi\sigma^2}\exp -\frac{\mathbf{x}^T\mathbf{x}}{2\sigma^2}$$

In [4]:

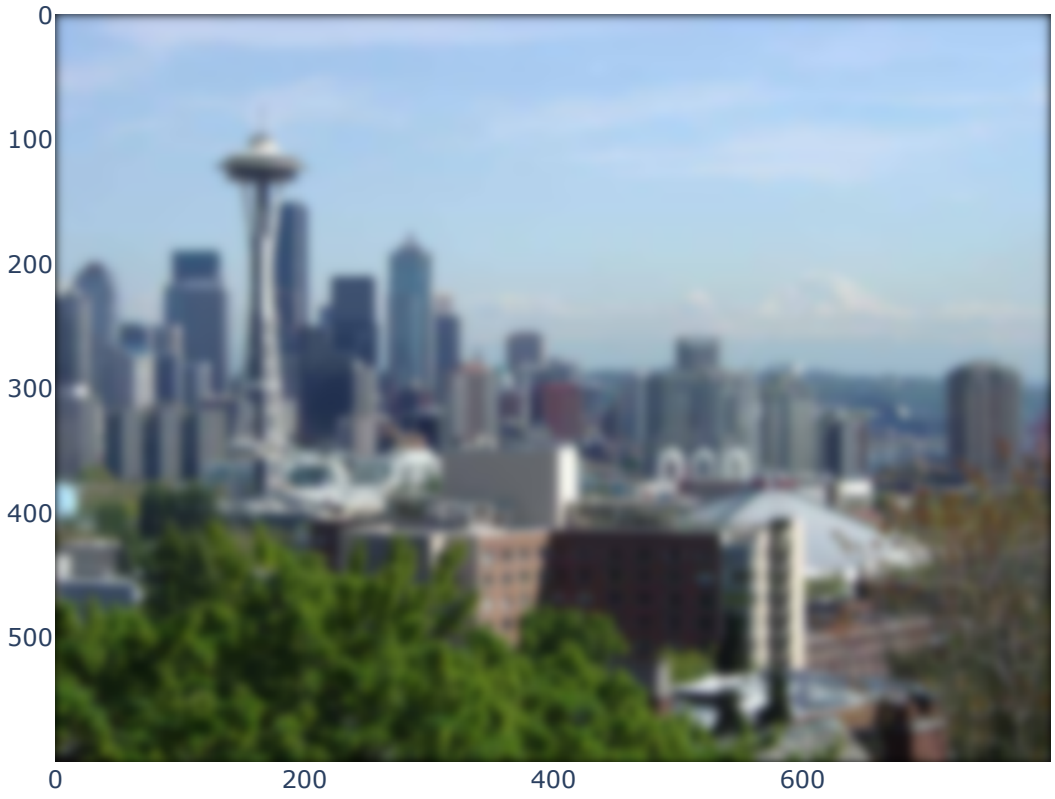
```
# Trick for colors
# np.stack([np.arange(16).reshape(4,4)])*3, axis = -1).shape

# # First lets write a convolve function
def convolve(image, flter):
    filter_size = flter.shape[0]

    padimage = imagepadding(image, filter_size)
    rowsize = image.shape[0]
    colsize = image.shape[1]

    im_filtered = np.zeros_like(padimage, dtype=np.float32)

    for i in range(rowsize):  # rows
        for j in range(colsize):  # columns
            submat = padimage[i : i + filter_size, j : j + filter_size]
            im_filtered[i, j] = np.sum(np.multiply(submat, flter))
    return im_filtered


# Image Padding function
def imagepadding(image, sizeofpad):
    sizeofpad = sizeofpad // 2
    rowsize = image.shape[0]
    colsize = image.shape[1]

    # Multiply by two since padding is double sided
    padimage = np.zeros(shape=(rowsize + 2 * sizeofpad, colsize + 2 * sizeofpad))

    # Plant the image in the middle of the new matrix, equidistant from the edges
    padimage[sizeofpad:-sizeofpad, sizeofpad:-sizeofpad] = image
    return padimage
```

In [5]:

```
def GaussianBlurImage(image, sigma):
    # Builds the filter
    filter_size = 2 * int(4 * sigma + 0.5) + 1
    gaussian_filter = np.zeros((filter_size, filter_size), np.float32)
    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))
            )

    # Zero matrix
    im_filtered = np.zeros(
        shape=(
            image.shape[0] + gaussian_filter.shape[0] - 1,
            image.shape[1] + gaussian_filter.shape[0] - 1,
            3,
```

```
        )
    )

    # Filter for each RGB
    for c in range(image.shape[2]):
        im_filtered[:, :, c] = convolve(image[:, :, c], gaussian_filter)

    # RGB version
    im_filtered = im_filtered[
        : image.shape[0], : image.shape[1], :,
    ]

    print("The final resolution is ", im_filtered.shape)
    fig = px.imshow(np.clip(im_filtered, 0, 255).astype(np.uint8))
    fig.write_image("1.png")
    display(fig)
    return im_filtered


gb = GaussianBlurImage(image, 4)
```

```
The final resolution is  (600, 800, 3)
```



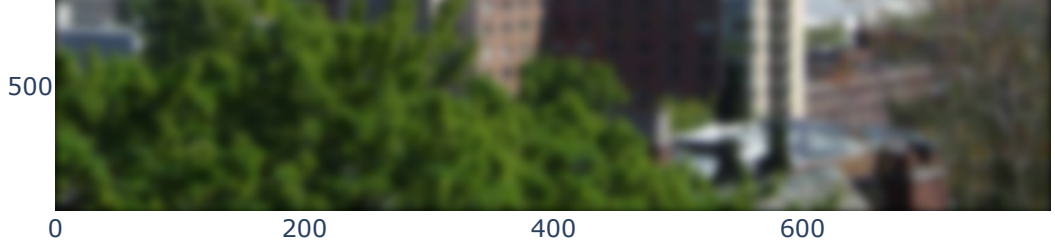
In [6]:

```
# im_filtered_skimage = filters.gaussian(
#     image, sigma=4, mode="constant", cval=0.0, multichannel=True, preserve_range=True
# )
# print(im_filtered_skimage.shape)
# px.imshow(im_filtered_skimage)
```

Problem 2

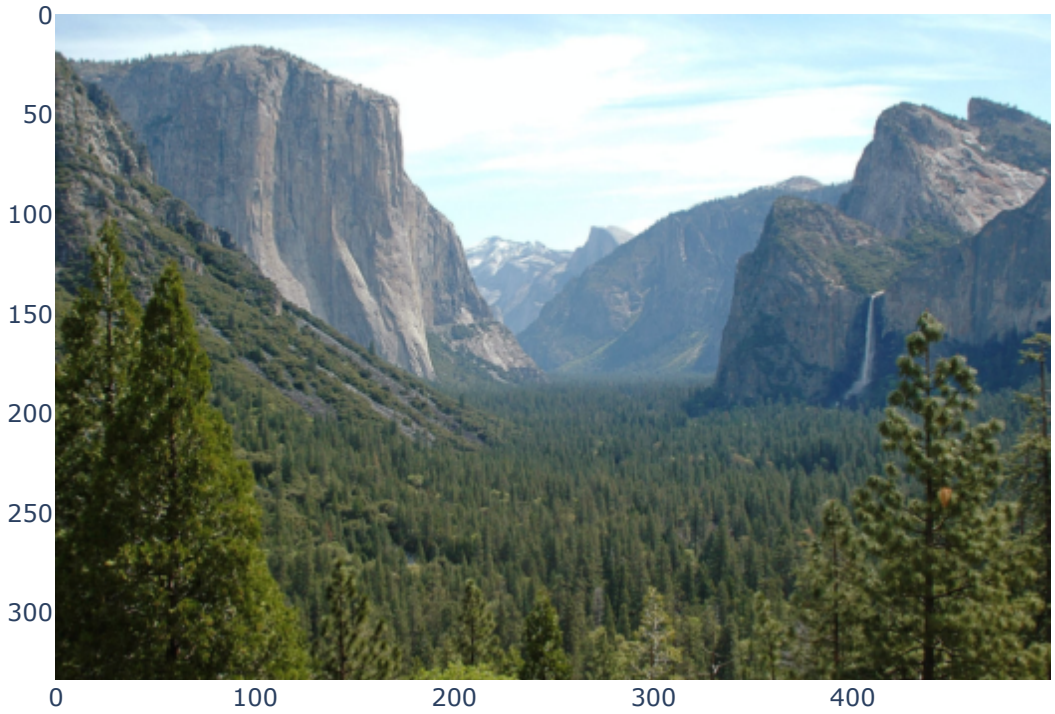
In [7]:

```
def SeparableGaussianBlurImage(image, sigma):
    # Create filter
    filter_size = 2 * int(4 * sigma + 0.5) + 1
    gaussian_filter = np.fromfunction(
        lambda x: math.e
        ** ((-1 * (x - (filter_size - 1) / 2) ** 2) / (2 * sigma ** 2)),
        (filter_size,),
    )
    gaussian_filter = gaussian_filter / np.sum(
        gaussian_filter
    )  # Normalizing for sum to 1

    # Pad Image for rgb
    padimage = np.zeros(
        (image.shape[0] + filter_size - 1, image.shape[1] + filter_size - 1, 3)
    )
    for rgb in range(3):
        padimage[:, :, rgb] = imagepadding(image[:, :, rgb], filter_size)
    # Create empty matrix
    im_blur = np.zeros(
        (padimage.shape[0] - filter_size + 1, padimage.shape[1] - filter_size + 1, 3)
    )

    for c in range(3):
        img = padimage[:, :, c].astype(np.uint8)
        rowblur = np.zeros((img.shape[0], img.shape[1] - filter_size + 1))
        for i, val in enumerate(gaussian_filter):
            rowblur += val * img[:, i : img.shape[1] - filter_size + i + 1]
        colblur = np.zeros((rowblur.shape[0] - filter_size + 1, rowblur.shape[1]))
        for i, val in enumerate(gaussian_filter):
            colblur += val * rowblur[i : img.shape[0] - filter_size + i + 1]
        im_blur[:, :, c] = colblur

    fig = px.imshow(im_blur)
    fig.write_image("2.png")
    display(fig)

    return im_blur


sep = SeparableGaussianBlurImage(image, 4.0)
```



Problem 3

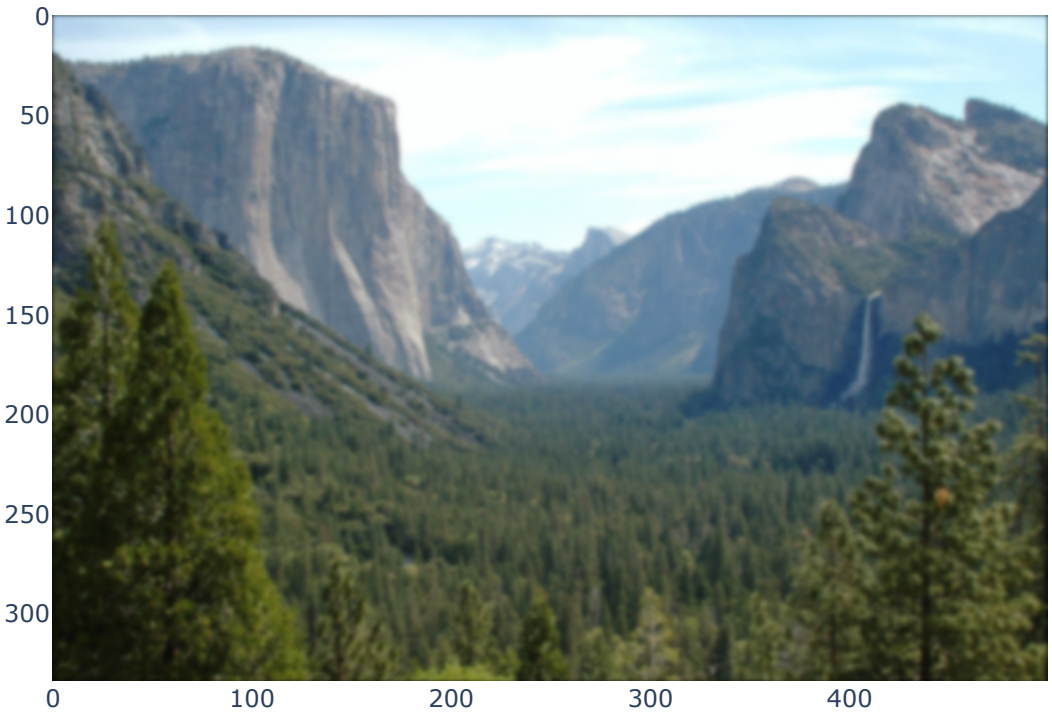
In [8]:
```
yosemite = Image.open("hw1_data/Yosemite.png")  # .convert("L")
yosemite = np.asarray(yosemite)
px.imshow(yosemite, color_continuous_scale="gray")
```



In [9]:
```
def SharpenImage(image, sigma, alpha):
    image = image.astype(np.uint8)
    blur = GaussianBlurImage(image, sigma)
    edges = blur - image

    # Replicating Matlabs functionality for uint8()
    edges[edges < 0] = 0
    edges[edges > 255] = 255
```

```
    sharp = image.astype(np.uint8) - alpha * (edges)   # im_filtered
    fig = px.imshow(sharp)
    fig.write_image("4.png")
    return fig


SharpenImage(yosemite, 1, 5)
```

```
The final resolution is  (334, 500, 3)
```



Problem 4

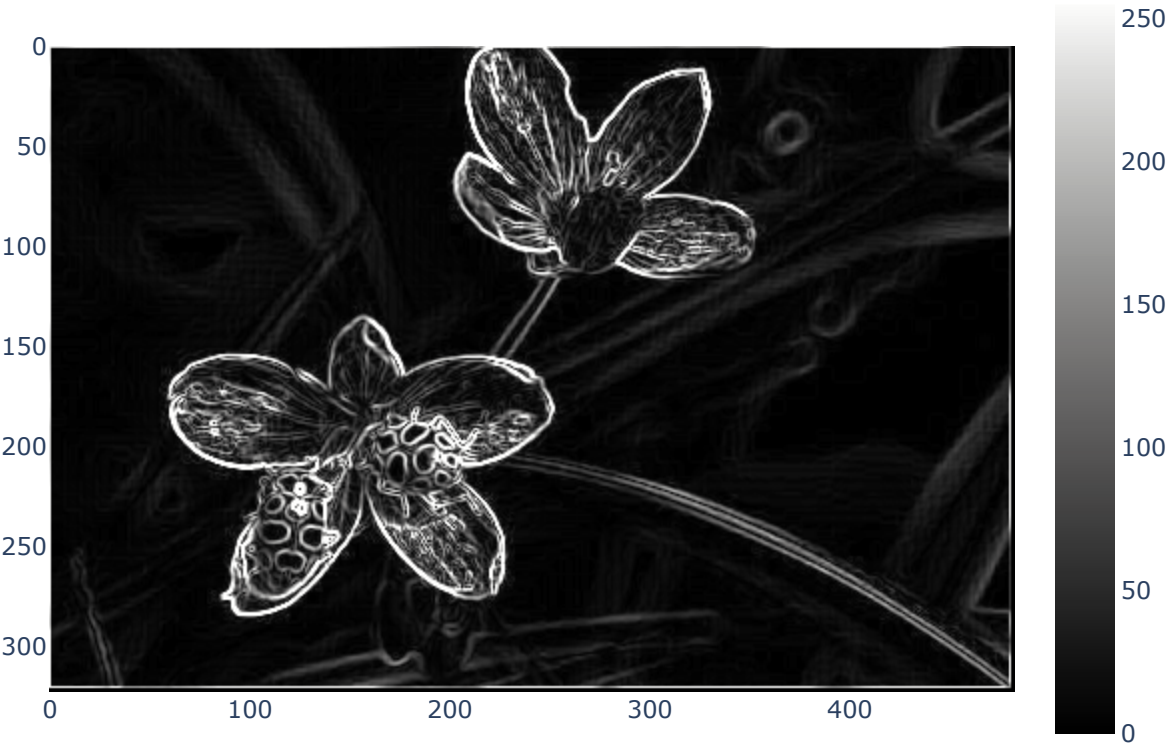
In [10]:

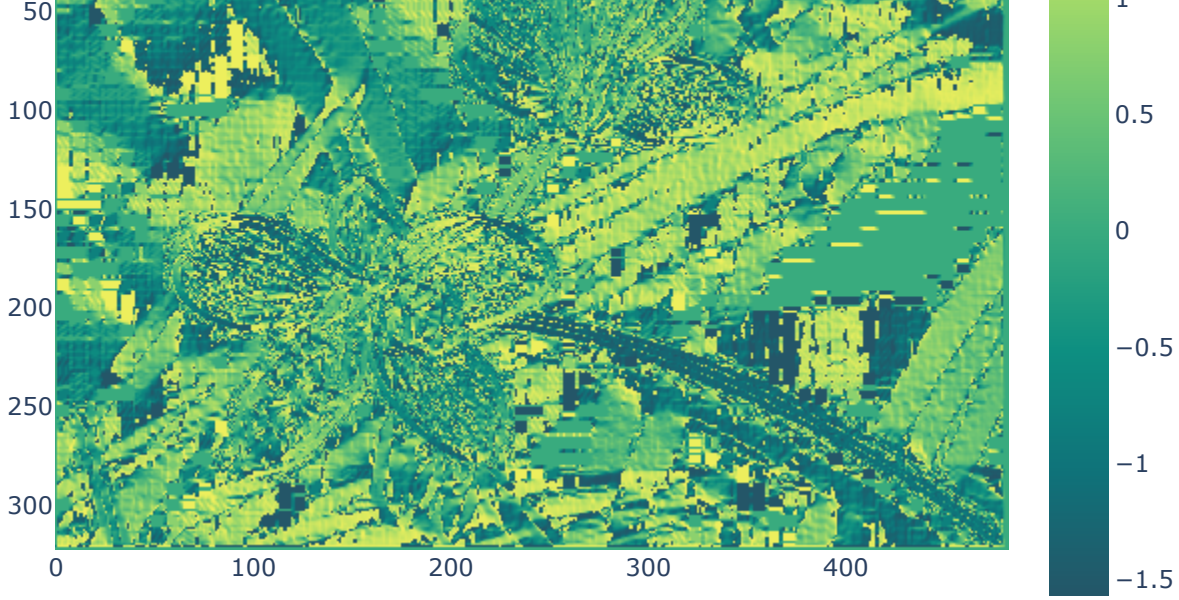
```
ladybug = Image.open("hw1_data/LadyBug.jpg").convert("L")
ladybug = np.asarray(ladybug)
px.imshow(ladybug, color_continuous_scale="gray")
```

In [24]:

```
def SobelImage(image):
    horiz = np.array([[1, 0, -1], [2, 0, -2], [1, 0, -1]])
    vert = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

    gx = convolve(image, horiz)
    gy = convolve(image, vert)

    # Convert 0 to machine epsilon
    gx[gx == 0] = np.finfo(float).eps  # 2.22044604925e-16
    # gy[gy == 0] = np.finfo(float).eps
```

```
    mag = np.sqrt(gx ** 2 + gy ** 2)
    orient = np.arctan(gy / gx)

    # Replicating Matlabs functionality for uint8()
    mag[mag < 0] = 0
    mag[mag > 255] = 255
    px.imshow(mag, color_continuous_scale="gray")

    #     orient[orient < 0] = 0
    #     orient[orient > 255] = 255

    fig1 = px.imshow(mag, color_continuous_scale="gray")
    fig2 = px.imshow(orient, color_continuous_scale="aggrnyl")

    fig1.write_image("5a.png")
    fig2.write_image("5b.png")

    display(fig1)
    display(fig2)
    return (mag, orient)


magnitude, orientation = SobelImage(ladybug)
```

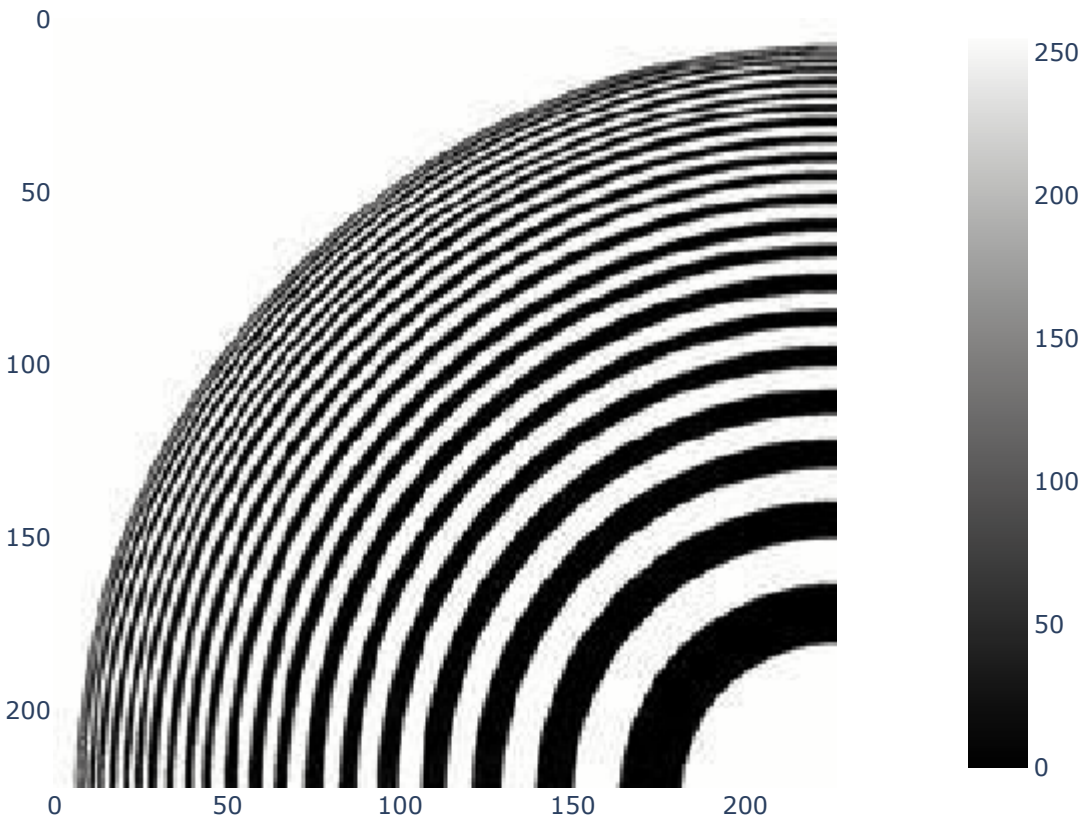




Problem 5

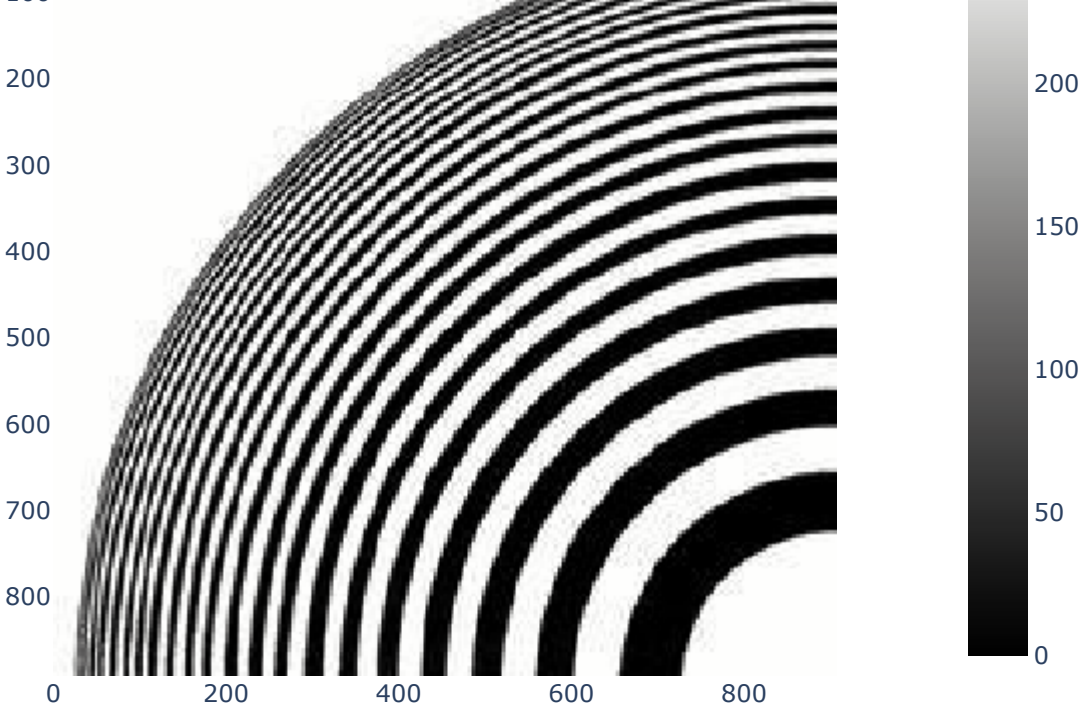
In [12]:

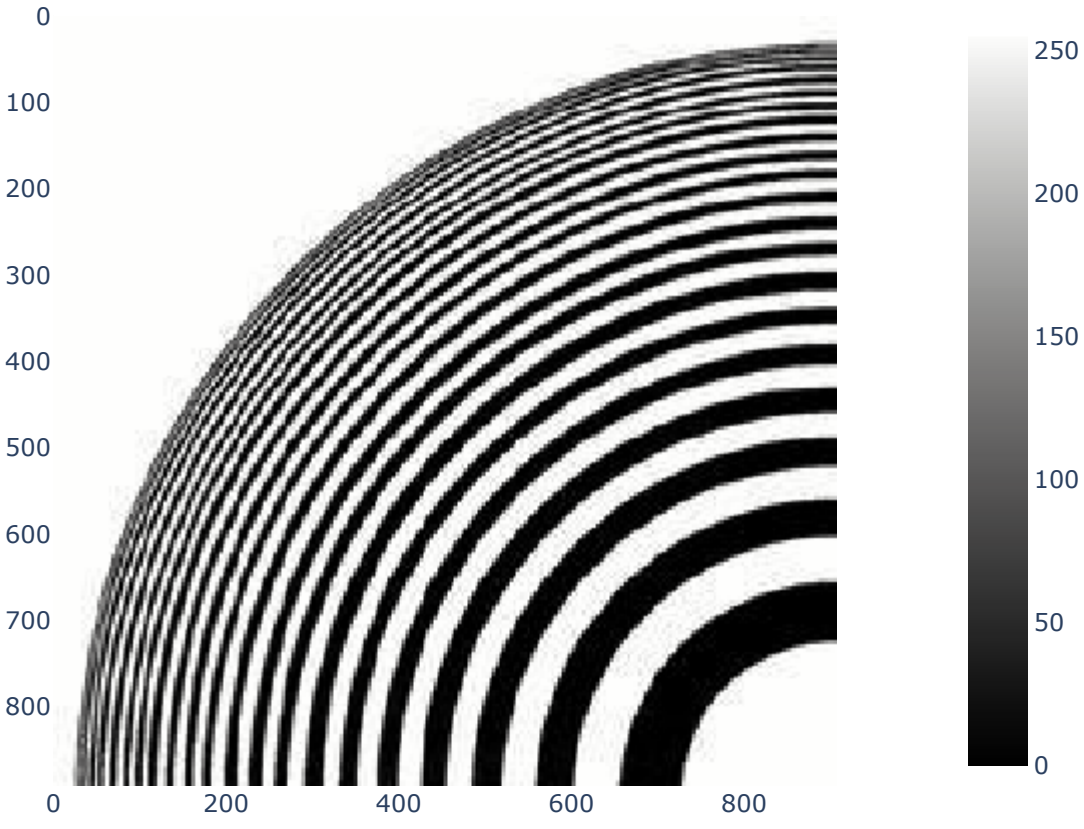
```
msmall = Image.open("hw1_data/Moire_small.jpg").convert("L")
msmall = np.asarray(msmall)
px.imshow(msmall, color_continuous_scale="gray")
```



In [151…

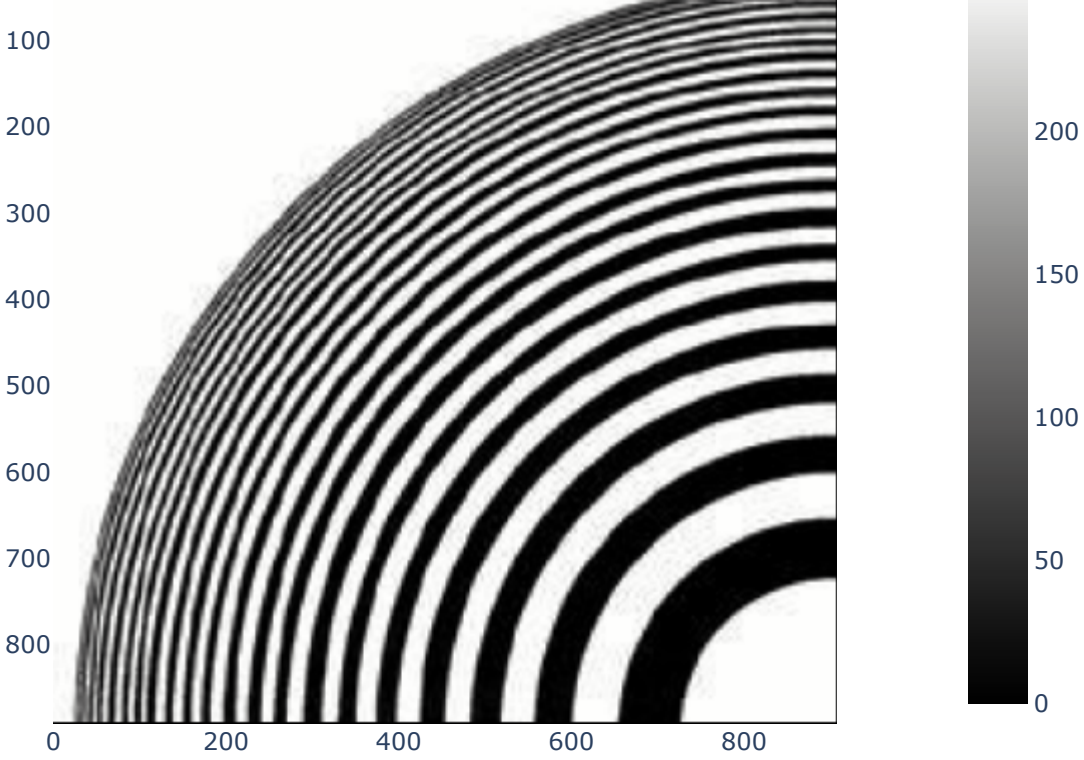
```
def BilinearInterpolation(image, x, y, scalefac=4, method="NN", imagewrite=True):
    size = np.shape(image)
    im_scaled = np.zeros(shape=(size[0] * scalefac, size[1] * scalefac), dtype=np.uint8)
    if method == "NN":
        for i in range(scalefac * size[0]):
            for j in range(scalefac * size[1]):
                x1 = int(np.floor(i / scalefac))
                y1 = int(np.floor(j / scalefac))
                im_scaled[i, j] = image[x1, y1]

        fig = px.imshow(im_scaled, color_continuous_scale="gray")
        display(fig)
        fig.write_image("6a.png")
    else:  # Bilinear
        for i in range(
            size[0] * scalefac - 3
        ):  # I am not quite sure why minus 3 works yet minus 2 doesnt
            # print(i)
            x1 = int(math.floor(i / scalefac))  # x
            xp1 = int(math.ceil(i / scalefac))  # x plus 1
            a = i / 4 % 1

            for j in range(size[1] * scalefac - 3):
                y1 = int(math.floor(j / scalefac))  # y
                yp1 = int(math.ceil(j / scalefac))  # y plus 1
                b = j / 4 % 1

                # Corners
                topleft = image[x1, y1]
                topright = image[xp1, y1]
                botleft = image[x1, yp1]
                botright = image[xp1, yp1]

                newval = (
                    ((1 - a) * (1 - b) * topleft)
                    + (a * (1 - b) * topright)
                    + (b * (1 - a) * botleft)
                    + (a * b * botright)
                )
                im_scaled[i, j] = newval

    if imagewrite == True:
        fig = px.imshow(im_scaled, color_continuous_scale="gray")
        display(fig)
        fig.write_image("6b.png")

    #     print("Original resolution", image.shape)
    #     print("New Resolution", im_scaled.shape)
    return (x, y)


bilinNN = BilinearInterpolation(msmall, 465, 518, method="NN")
print(bilinNN)
bilinBL = BilinearInterpolation(msmall, 465, 518, method="Bilinear")
print(bilinBL)
```


200
300
400
500
600
700
800
0
200
400
600
800
200
150
100
50
0


0
100
200
300
400
500
600
700
800
0
200
400
600
800
250
200
150
100
50
0

(465, 518)


0
250

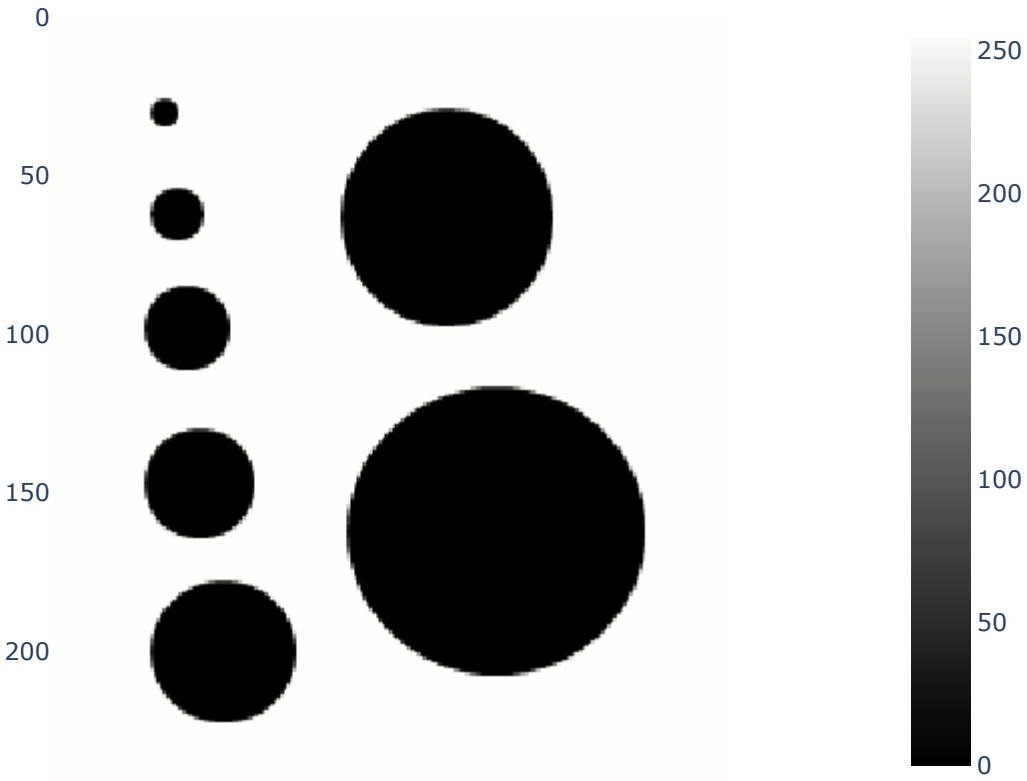


(465, 518)

Question 6

In [14]:

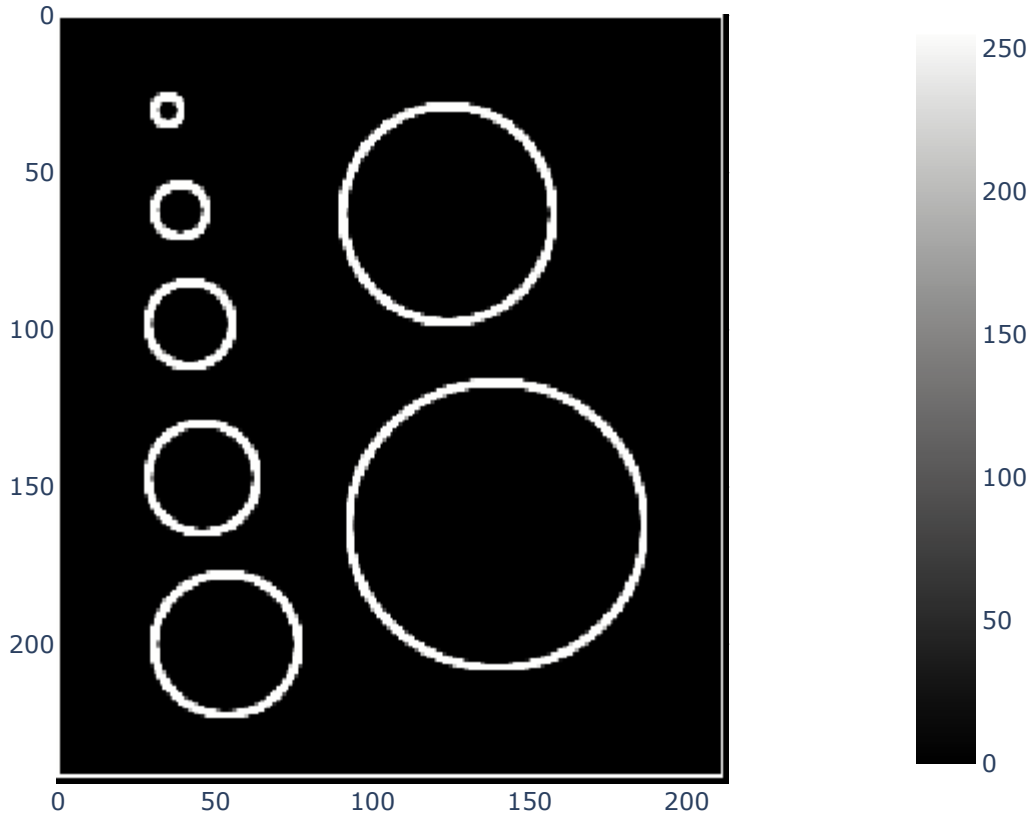
```
circles = Image.open("hw1_data/Circle.png").convert("L")
circles = np.asarray(circles)
px.imshow(circles, color_continuous_scale="gray")
```

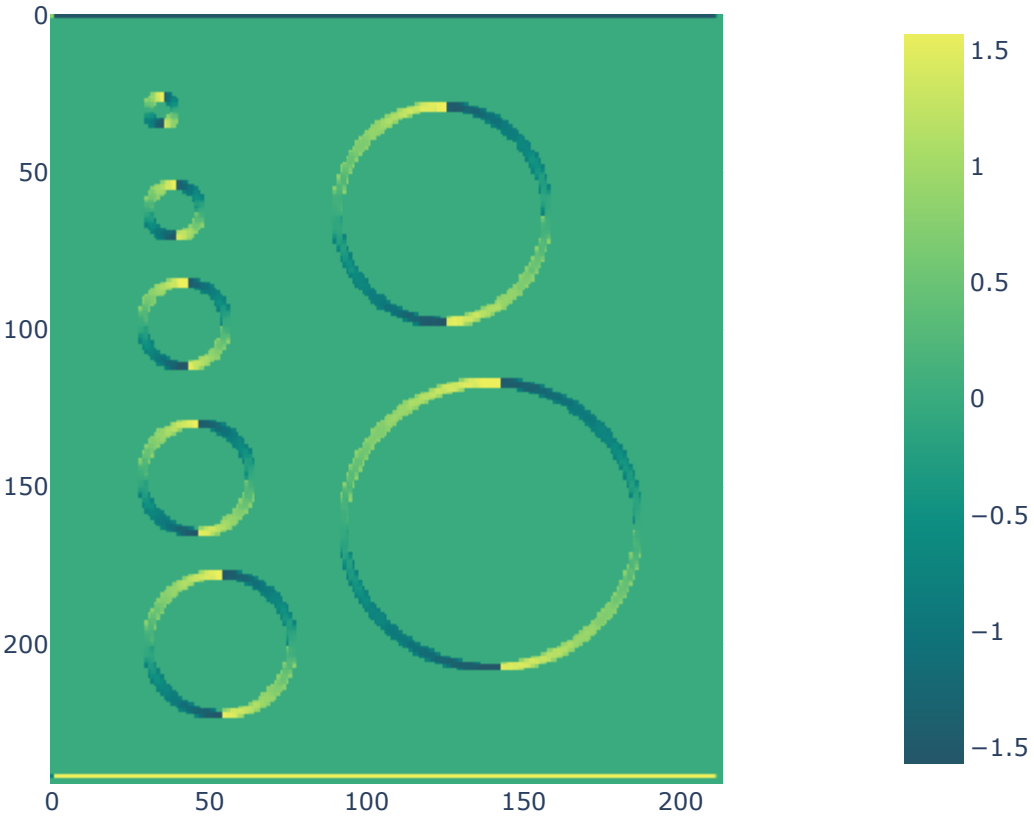


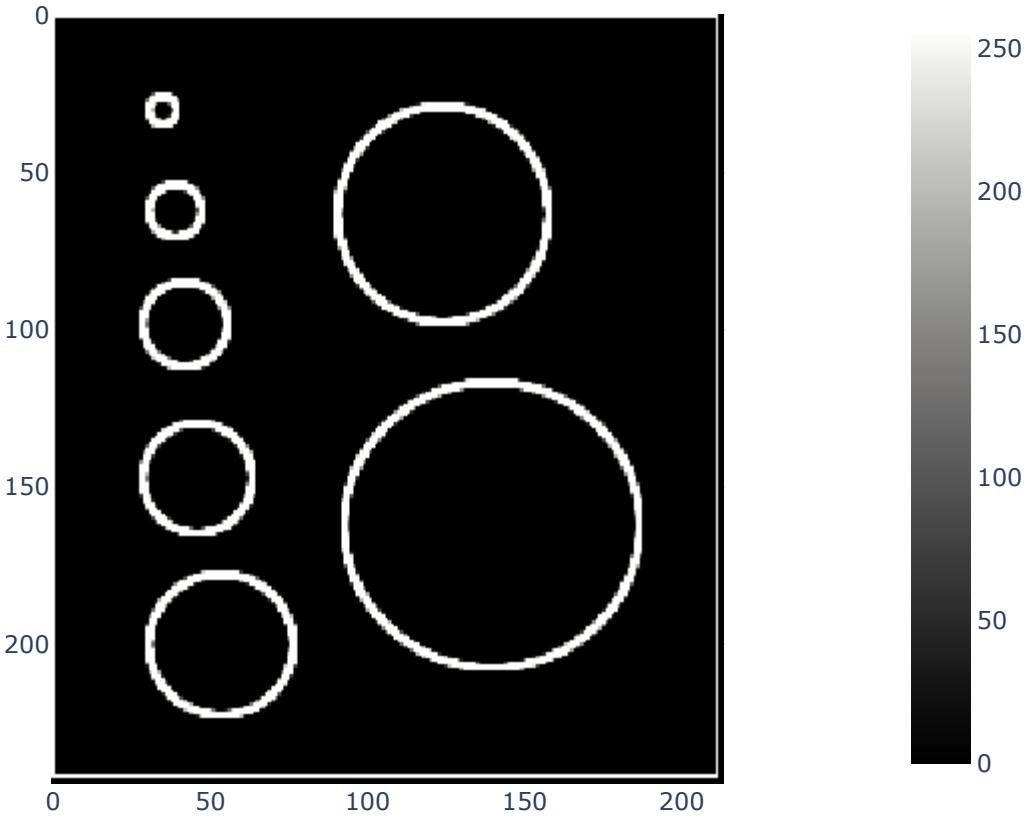
0 50 100 150 200

In [162…

```
def FindPeaksImage(image, thres):
    mag, theta = SobelImage(image)
    # print(orient)

    for i in range(
        mag.shape[0] - 3
    ):  # minus three since we do not want to go over bounds of matrix
        for j in range(mag.shape[1] - 3):
            #             e0, e1 = BilinearInterpolation(
            #                 image, i + 1, j + 1, method="Bilinear", imagewrite=False
            #             )

            submag = mag[i : i + 3, j : j + 3]
            subtheta = theta[i : i + 3, j : j + 3]

            # find which cells are in same direction as center
            centermag = submag[1, 1]
            centertheta = subtheta[1, 1]

            neighbors = np.argwhere(subtheta == centertheta)
            # print(neighbors)
            # Remove the center one (redundant neighbor to itself)
            mid = [1, 1]
            neighbors = [i for i in neighbors if not np.any(i == mid)]

            if len(neighbors) == 2:
                n1 = neighbors[0]  # first neighbor
                n2 = neighbors[1]  # second neighbor

                # now that same direction cells are found, look at intensity
                # if less than center magnitude and threshold, turn to zero
                # here we compare the first neighbor
                if (submag[n1[0], n1[1]] < thres) & (submag[n1[0], n1[1]] < centermag):
                    submag[n1[0], n1[1]] == 0
                if (submag[n1[0], n1[1]] > thres) & (submag[n1[0], n1[1]] > centermag):
                    # change the center to zero since it is not as intense as neighbor
                    submag[1, 1] == 0
                    submag[n1[0], n1[1]] == 255
                # Now second neighbor
                if (submag[n2[0], n2[1]] < thres) & (submag[n2[0], n2[1]] < centermag):
                    submag[n2[0], n2[1]] == 0
                if (submag[n2[0], n2[1]] > thres) & (submag[n2[0], n2[1]] > centermag):
                    submag[1, 1] == 0
                    submag[n2[0], n2[1]] == 255

            mag[i : i + 3, j : j + 3] = submag

    fig = px.imshow(mag)
    fig.write_image("7.png")

    return mag


px.imshow(FindPeaksImage(circles, 40), color_continuous_scale="gray")
```


0
50
100
150
200
0
50
100
150
200
250
200
150
100
50
0


0
50
100
150
200
0
50
100
150
200
1.5
1
0.5
0
−0.5
−1
−1.5



Section 2: Written Assignment

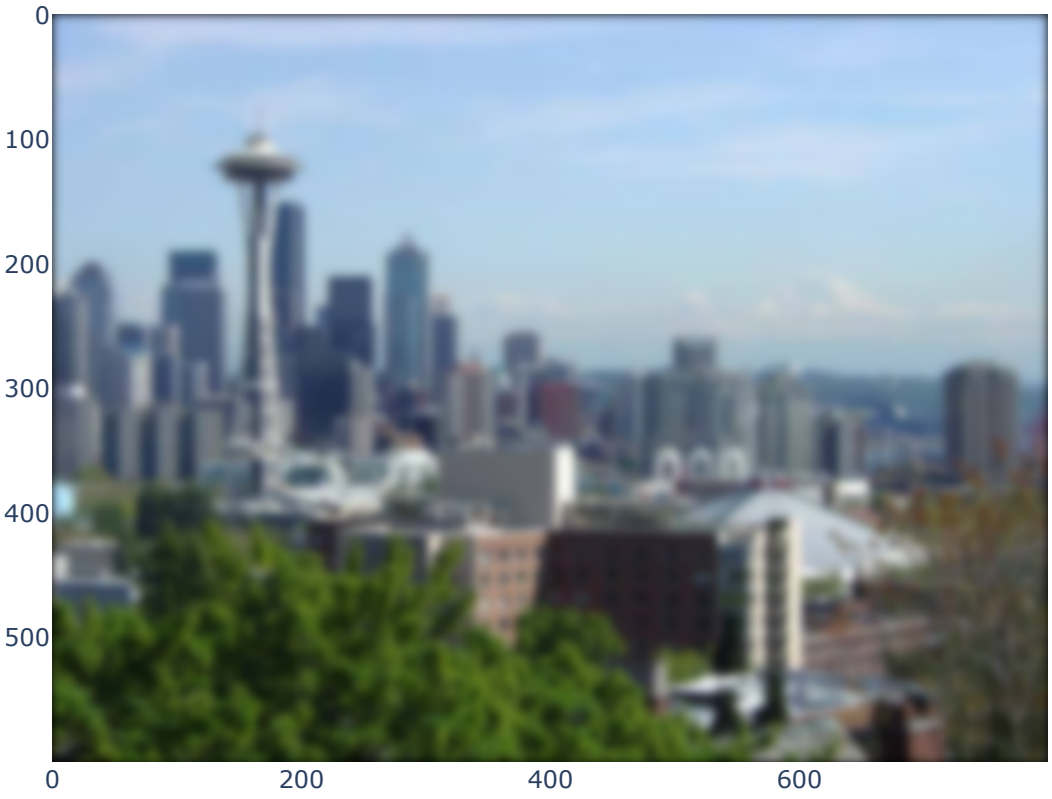
Question 1

In [16]:
```
import time
```

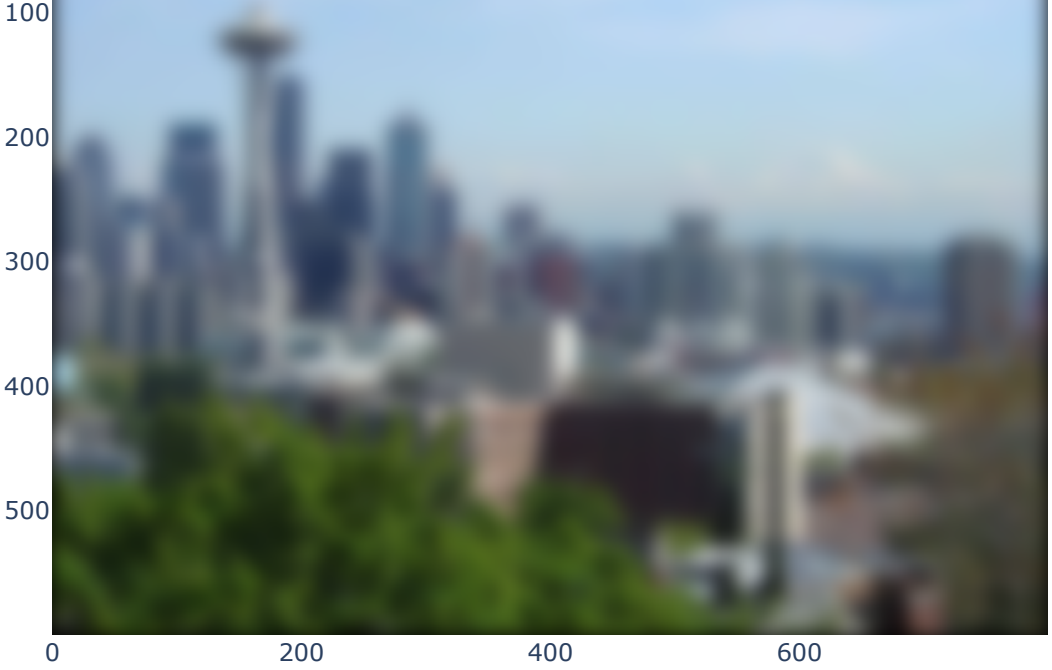
In [17]:
```
start = time.time()
GaussianBlurImage(image, 2)
print("--- %s seconds ---" % (time.time() - start))
start = time.time()
GaussianBlurImage(image, 4)
print("--- %s seconds ---" % (time.time() - start))
start = time.time()
GaussianBlurImage(image, 8)
print("--- %s seconds ---" % (time.time() - start))
```

```
The final resolution is  (600, 800, 3)
```


```
--- 7.835134029388428 seconds ---
The final resolution is  (600, 800, 3)
```




```
--- 9.607639074325562 seconds ---
The final resolution is  (600, 800, 3)
```

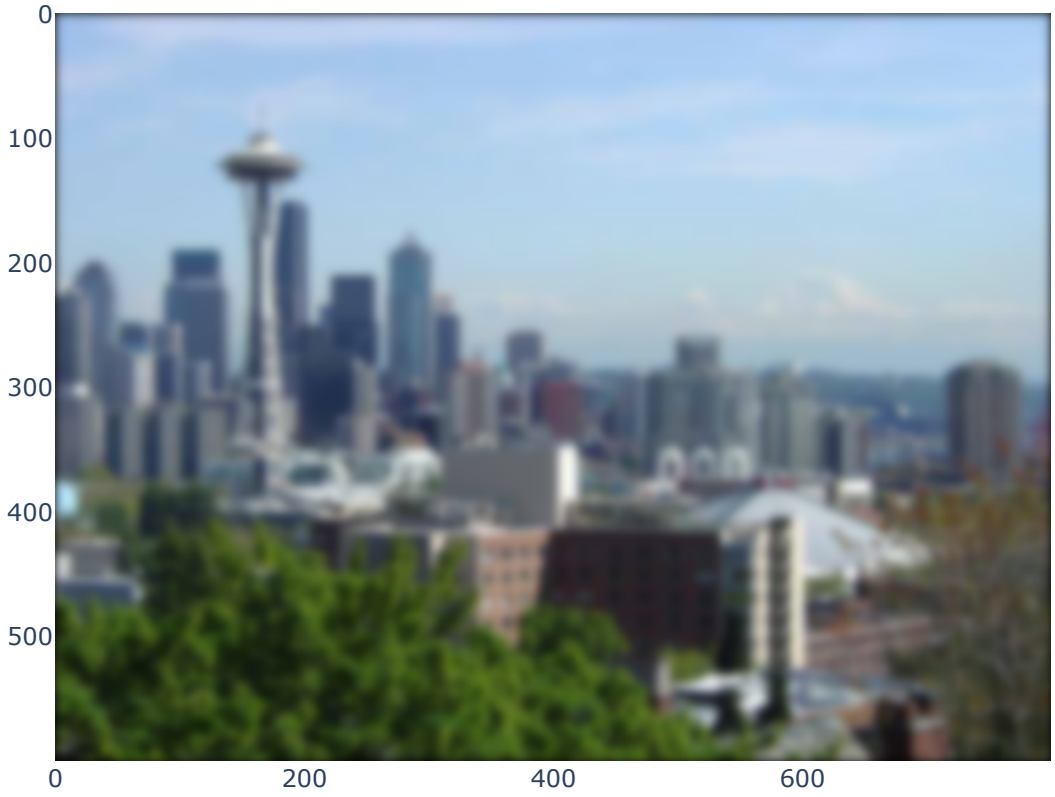



```
--- 17.2693293094635 seconds ---
```

In [18]:

```
start = time.time()
SeparableGaussianBlurImage(image, 2)
print("--- %s seconds ---" % (time.time() - start))
start = time.time()
SeparableGaussianBlurImage(image, 4)
print("--- %s seconds ---" % (time.time() - start))
start = time.time()
SeparableGaussianBlurImage(image, 8)
print("--- %s seconds ---" % (time.time() - start))
```


```
--- 0.6198980808258057 seconds ---
```




```
--- 0.7106447219848633 seconds ---
```

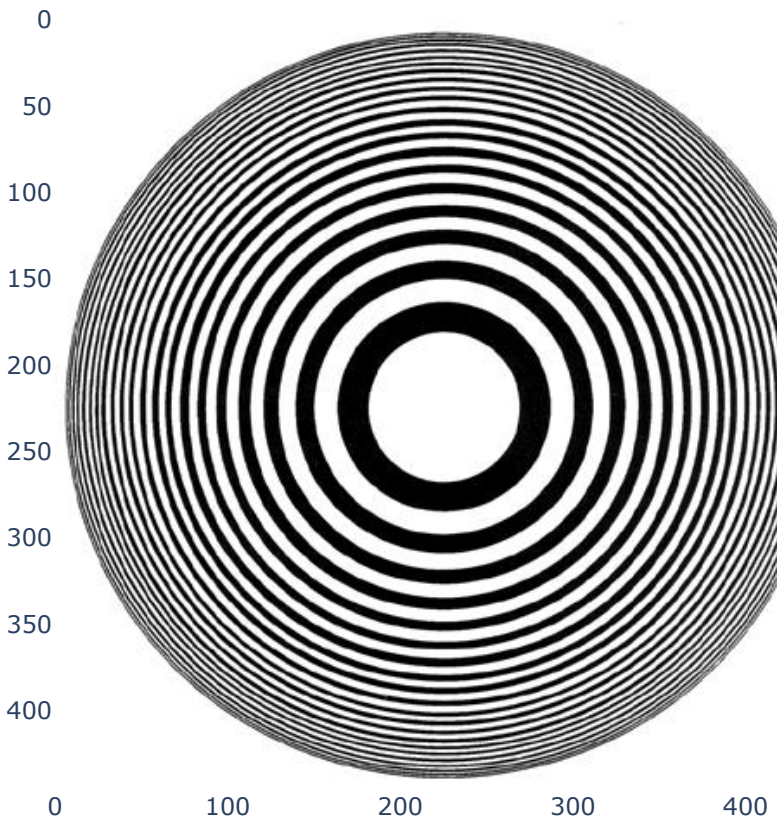

```
--- 0.8921654224395752 seconds ---
```

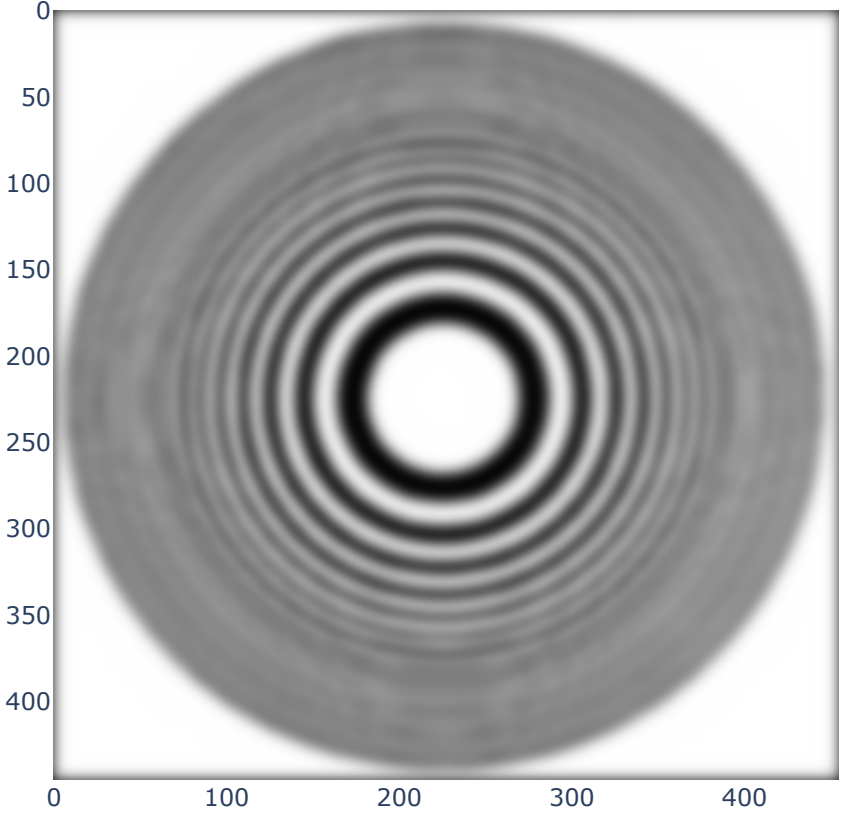
For a $\sigma = 32$ I believe it would take over 30 seconds for GaussianBlurImage() to perform the task and over 2 seconds for SeparableGaussianBlur() to finish. We can see from above that an increase in σ increases the run time of the blurring.

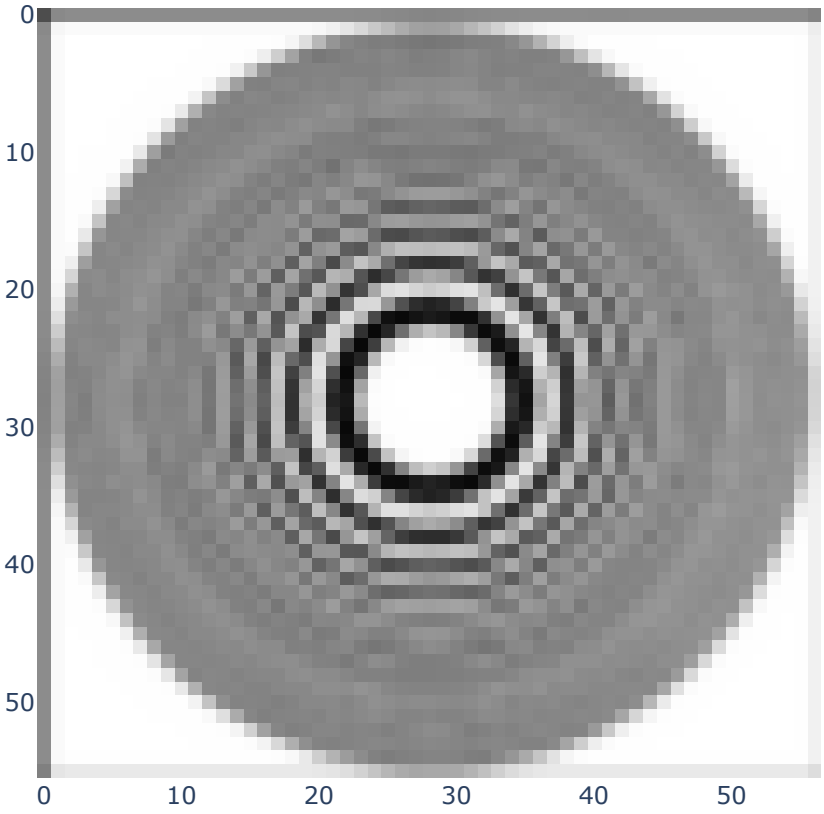
Question 2

In [180…

```
moire = Image.open("hw1_data/Moire.jpg")   # .convert("L")
moire = np.asarray(moire)
display(px.imshow(moire, color_continuous_scale="gray"))
moire = SeparableGaussianBlurImage(moire, 4)

for i in range(3):  # Half size 3 times down sampling
    moire = moire[0::2, 0::2]

display(px.imshow(moire, color_continuous_scale="gray"))
```

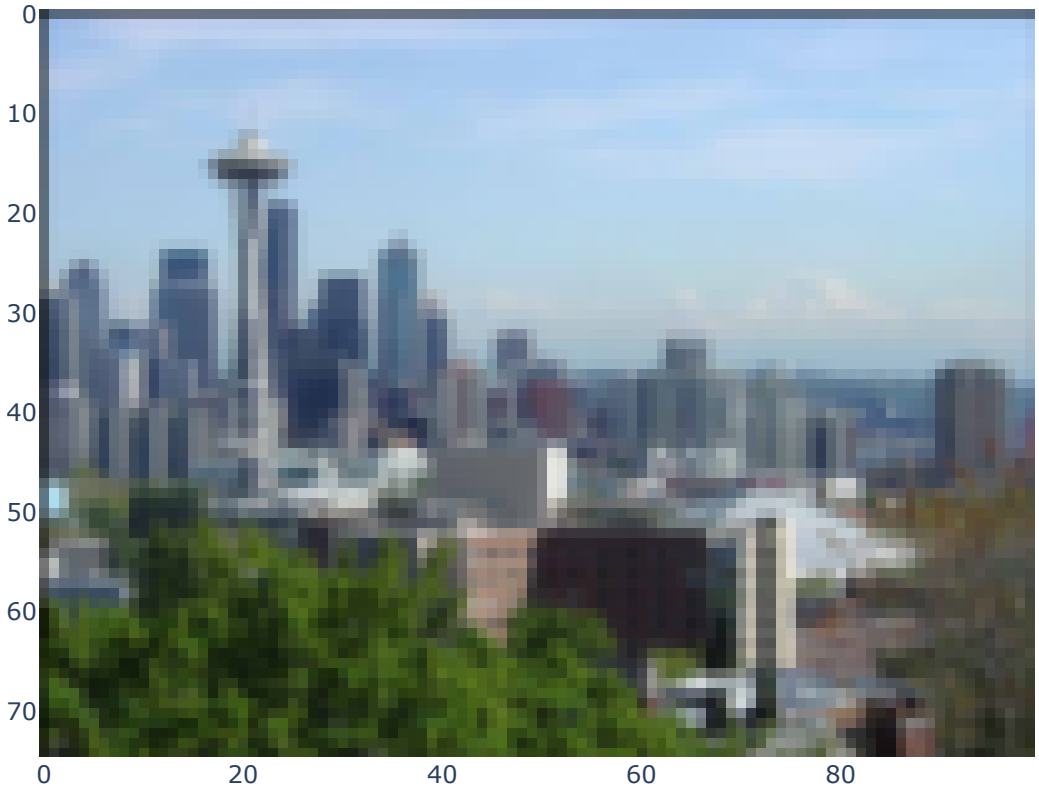



0
50
100
150
200
250
300
350
400
0
100
200
300
400


0
10
20
30
40
50
0
10
20
30
40
50

In [182…

```
seattle = Image.open("hw1_data/seattle.jpg")   # .convert("L")
seattle = np.asarray(seattle)
display(px.imshow(seattle, color_continuous_scale="gray"))
seattle = SeparableGaussianBlurImage(seattle, 4)

for i in range(3):  # Half size 3 times down sampling
    seattle = seattle[0::2, 0::2]

display(px.imshow(seattle, color_continuous_scale="gray"))
```



The best amount of blur for the Moire image seems to be with a sigma of 4, however, this value does not perform the most ideal output. In this question, I am assuming best is that we can blur an image with a downsampled version of the image and maintain a optimum amount of information such that the blurring looks very close to blurring the orignal image.

Question 3

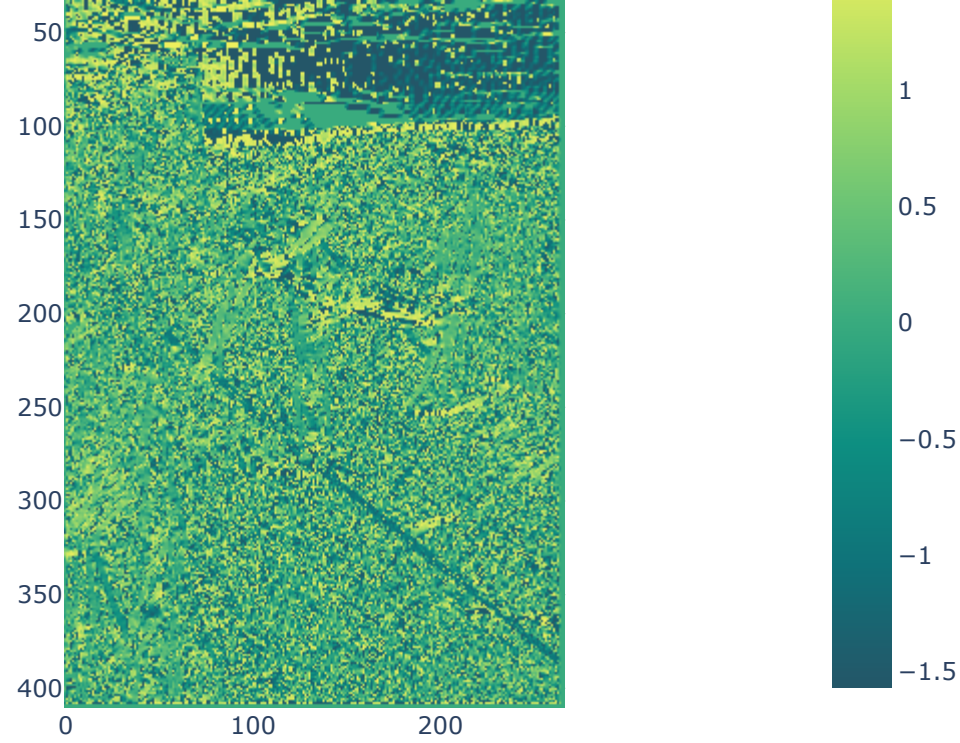
In [26]:

```
tightrope = Image.open("hw1_data/TightRope.png").convert("L")
tightrope = np.asarray(tightrope)
display(px.imshow(tightrope, color_continuous_scale="gray"))
x = SobelImage(tightrope)
```


50
100
150
200
250
300
350
400
0
100
200
200
150
100
50
0


0
50
100
150
200
250
300
350
400
0
100
200
250
200
150
100
50
0


0
1.5

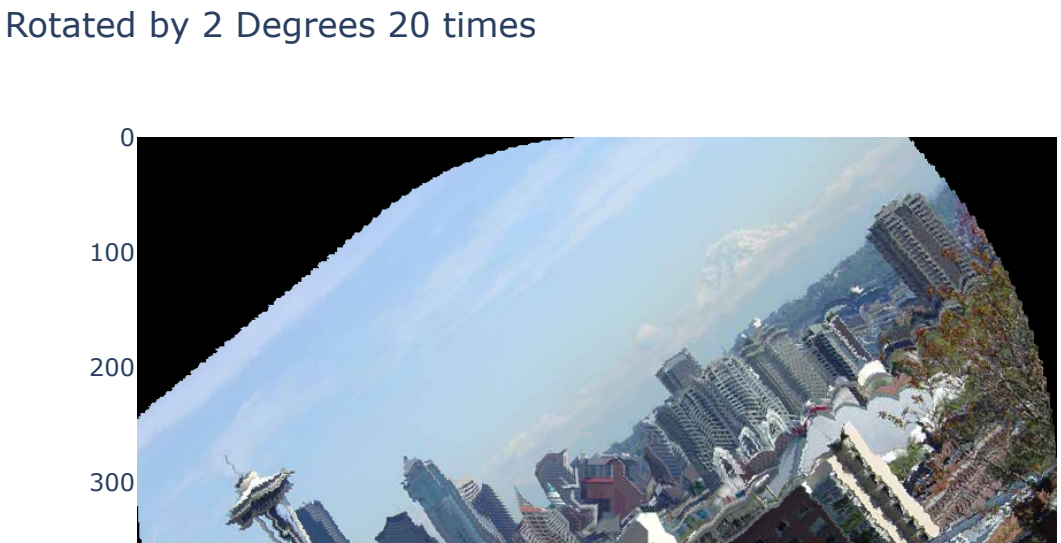


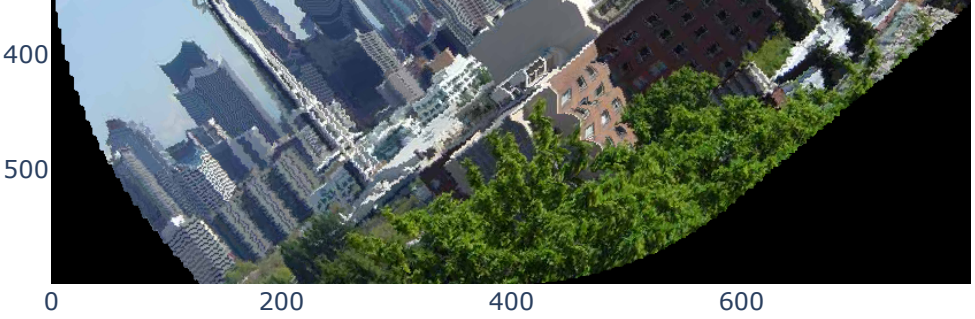
From the tight rope image, we can see that an obvious edge would be the cliff at the top left, however, the Sobel filters do not give this edge a lot of strength in the orientation. Another not so well seen edge to the filter is the side of the man's left leg.

Question 4

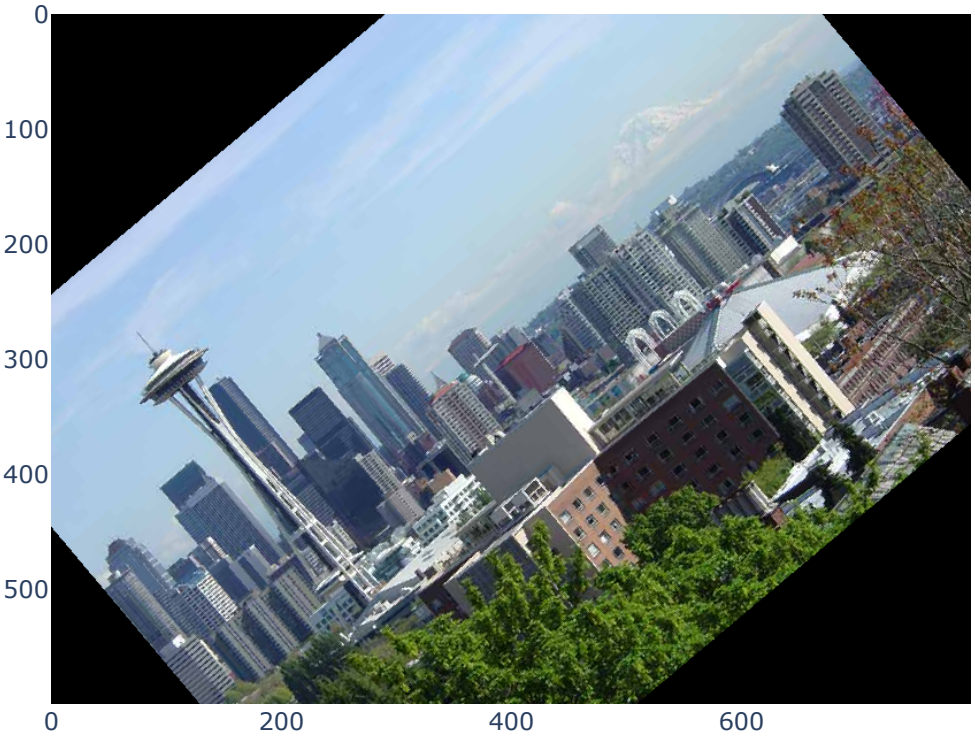
In [160…

```
seattle = Image.open("hw1_data/seattle.jpg")
for i in range(20):
    seattle = seattle.rotate(2)


display(px.imshow(seattle, title="Rotated by 2 Degrees 20 times"))

seattle = Image.open("hw1_data/seattle.jpg")
display(px.imshow(seattle.rotate(40), title="Rotated 40 Degrees"))
```





Rotated 40 Degrees



As we can see here, rotating an image 20 times by 2 degrees yields a different result than rotating an image 40 degrees. Performing so many small rotations begins to misplace pixels each rotation causing a warping effect. A single rotation maintains the quality of the original image without making any visible warping.