# Machine Vision Assignment 1

Name: Melwyn D Souza
Student Number: R00209495
Date: 18/Feb/2022
Module: Machine Vision
Course: MSc in Artificial Intelligence

Resulting Images from Assignment 1:

Task 1:

Task 1A:

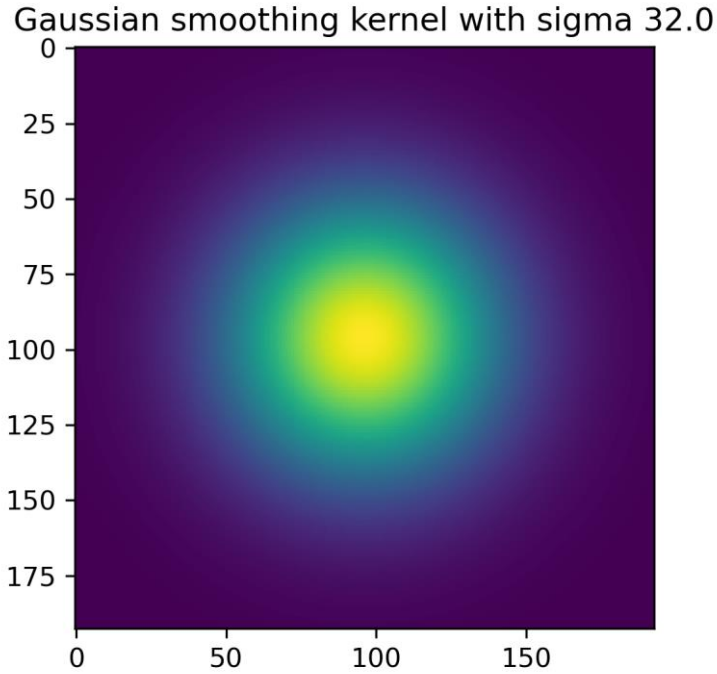
Greyscaled Image

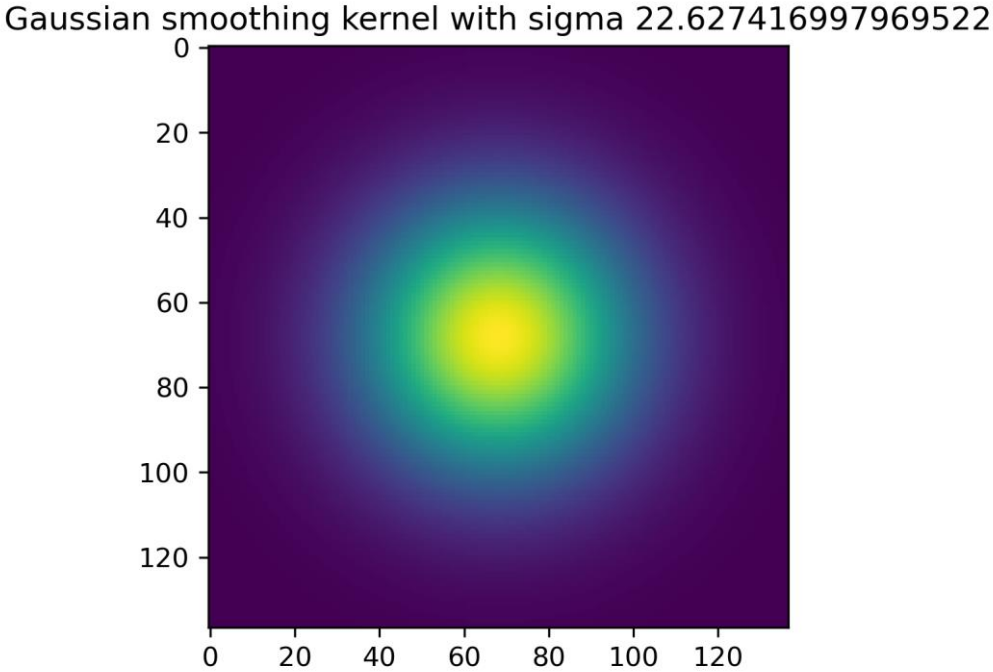
Code snippet – line 31

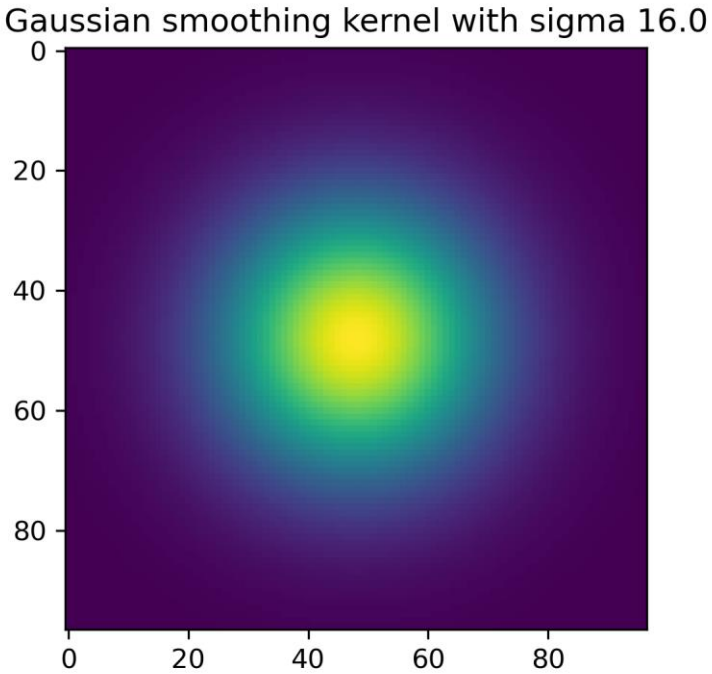
```
greyscaled = cv.cvtColor(input_image.astype('float32'), cv.COLOR_RGB2GRAY)
display("Grey Input", greyscaled)
```

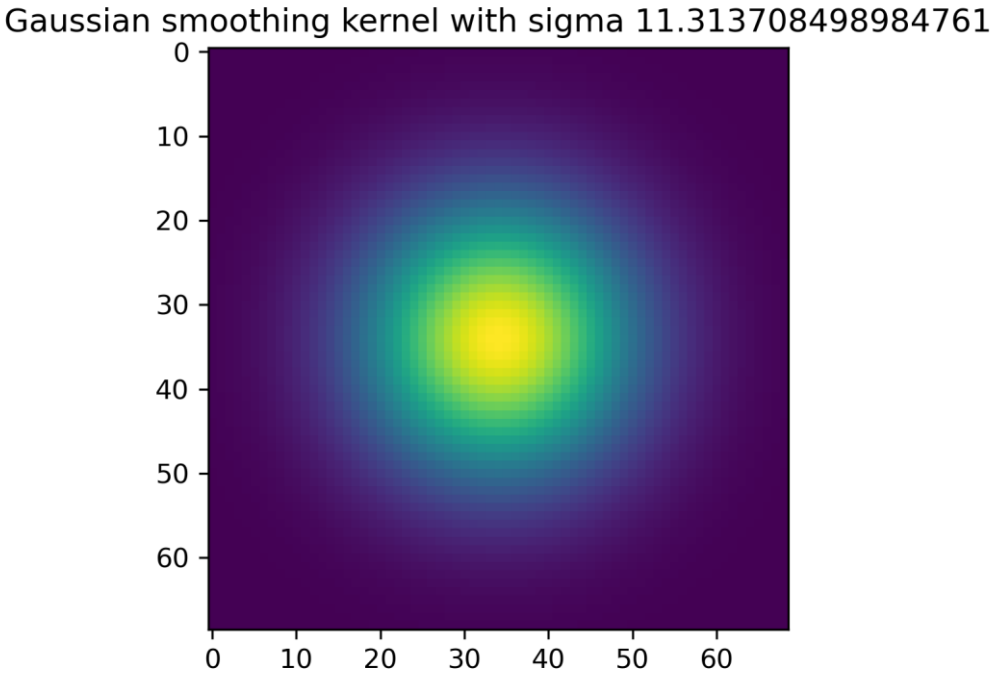
Task 1B:

Gaussian Smoothening Kernels

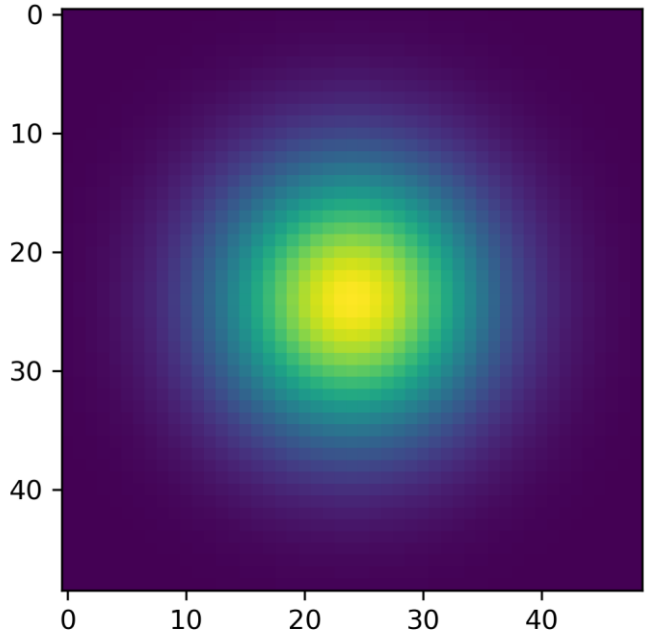


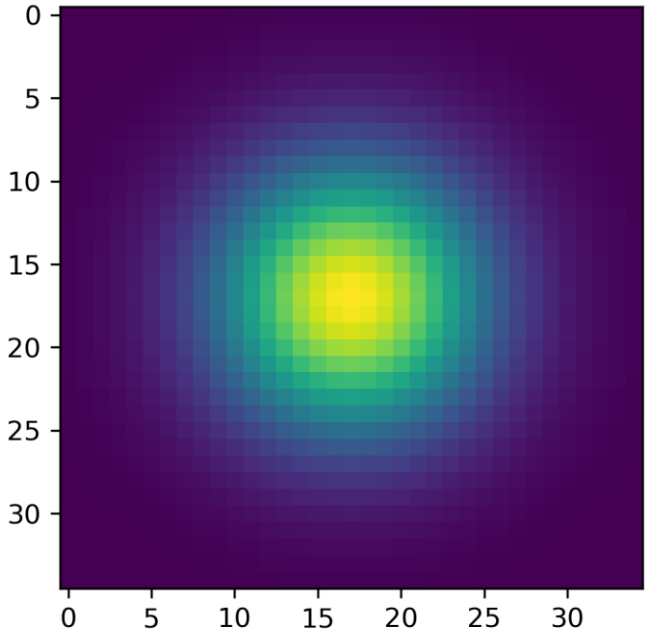

Gaussian smoothing kernel with sigma 16.0
0
20
40
60
80
0
20
40
60
80


Gaussian smoothing kernel with sigma 11.313708498984761
0
10
20
30
40
50
60
0
10
20
30
40
50
60

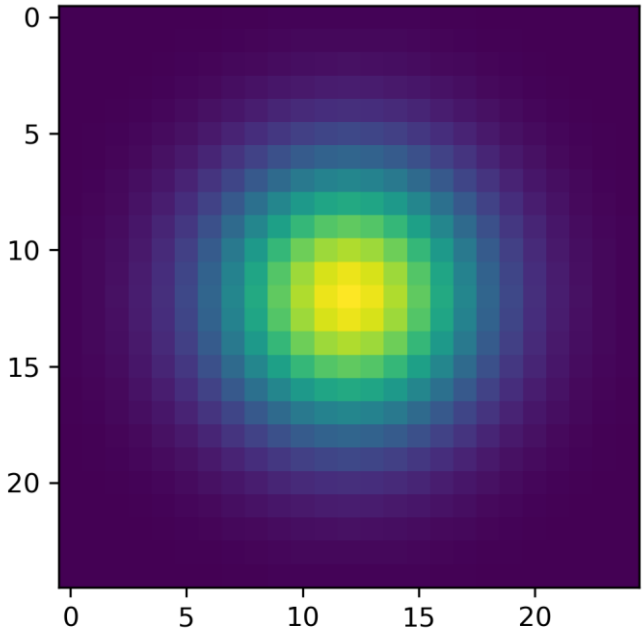
Gaussian smoothing kernel with sigma 8.0



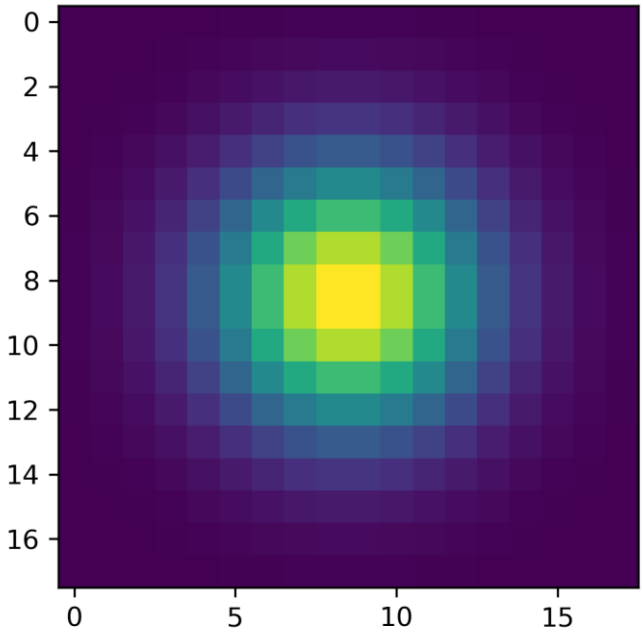

Gaussian smoothing kernel with sigma 5.656854249492381



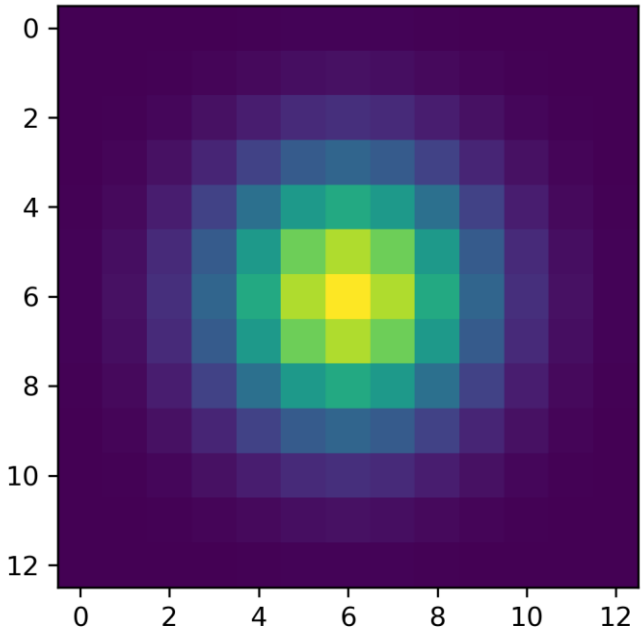
Gaussian smoothing kernel with sigma 4.0



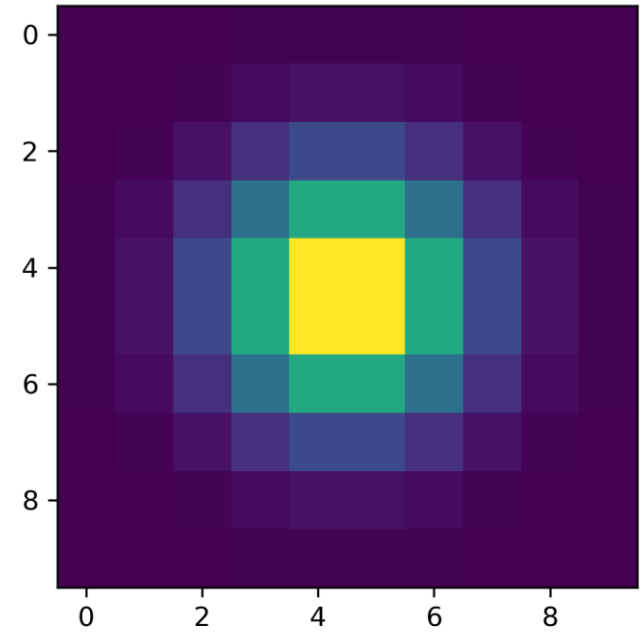

Gaussian smoothing kernel with sigma 2.8284271247461903

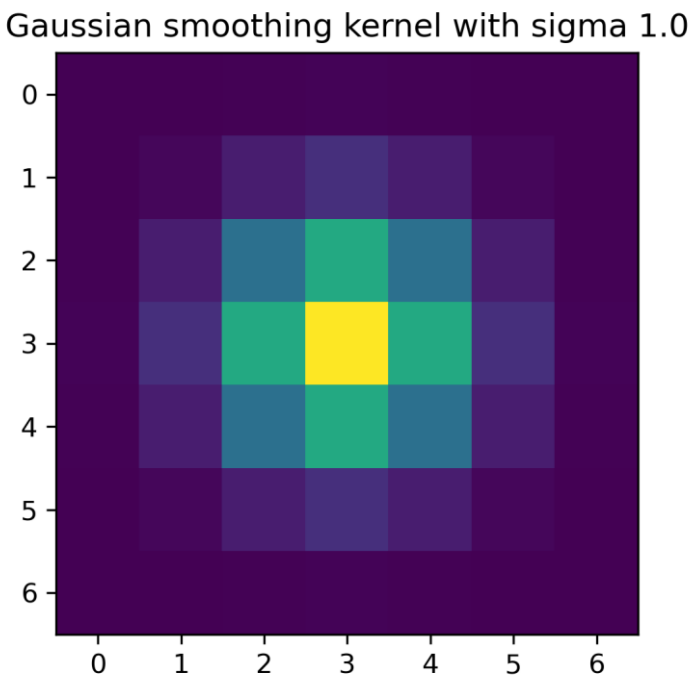


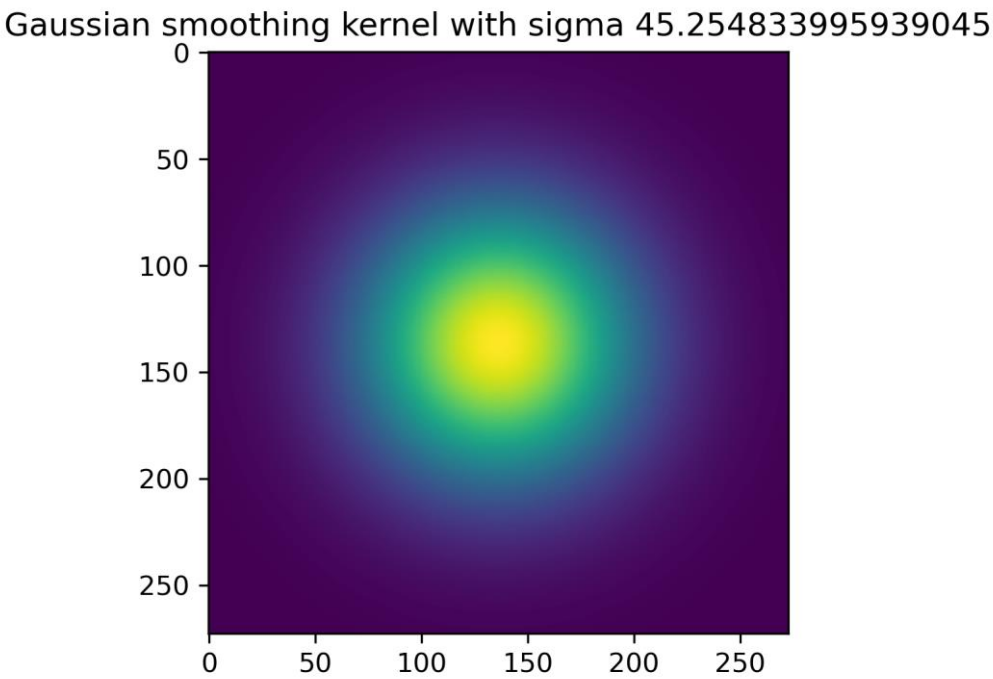
Gaussian smoothing kernel with sigma 2.0




Gaussian smoothing kernel with sigma 1.4142135623730951






Code Snippet – Line 38

```
for i in range (12):
    sigma = 2**(i/2)
    sigmaList.append(sigma)
    scaleSpace[sigma] = gauss_blur("Image", greyscaled, sigma)
```

Gaussian Smoothened Images

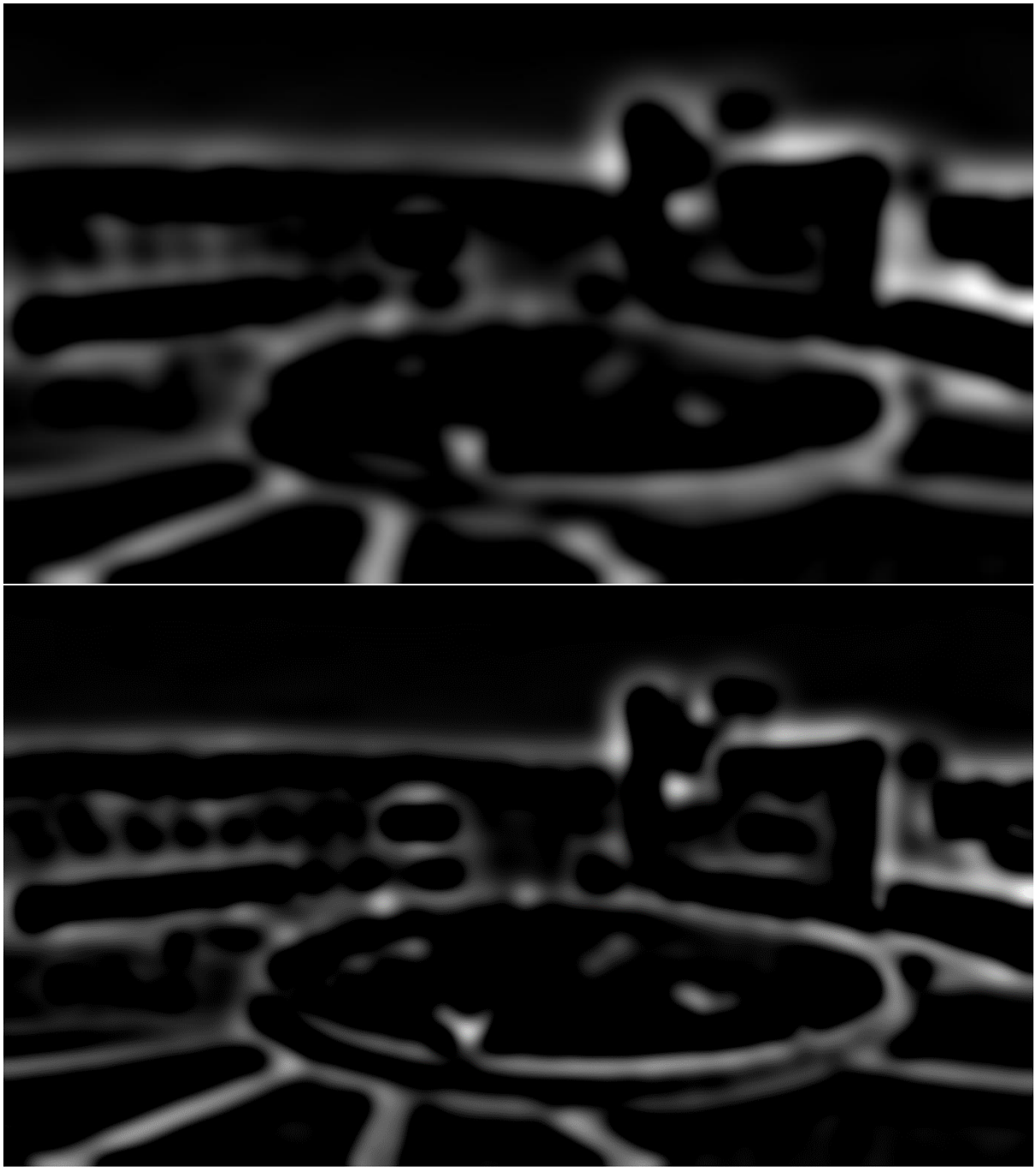
Code Snippet – Line 263

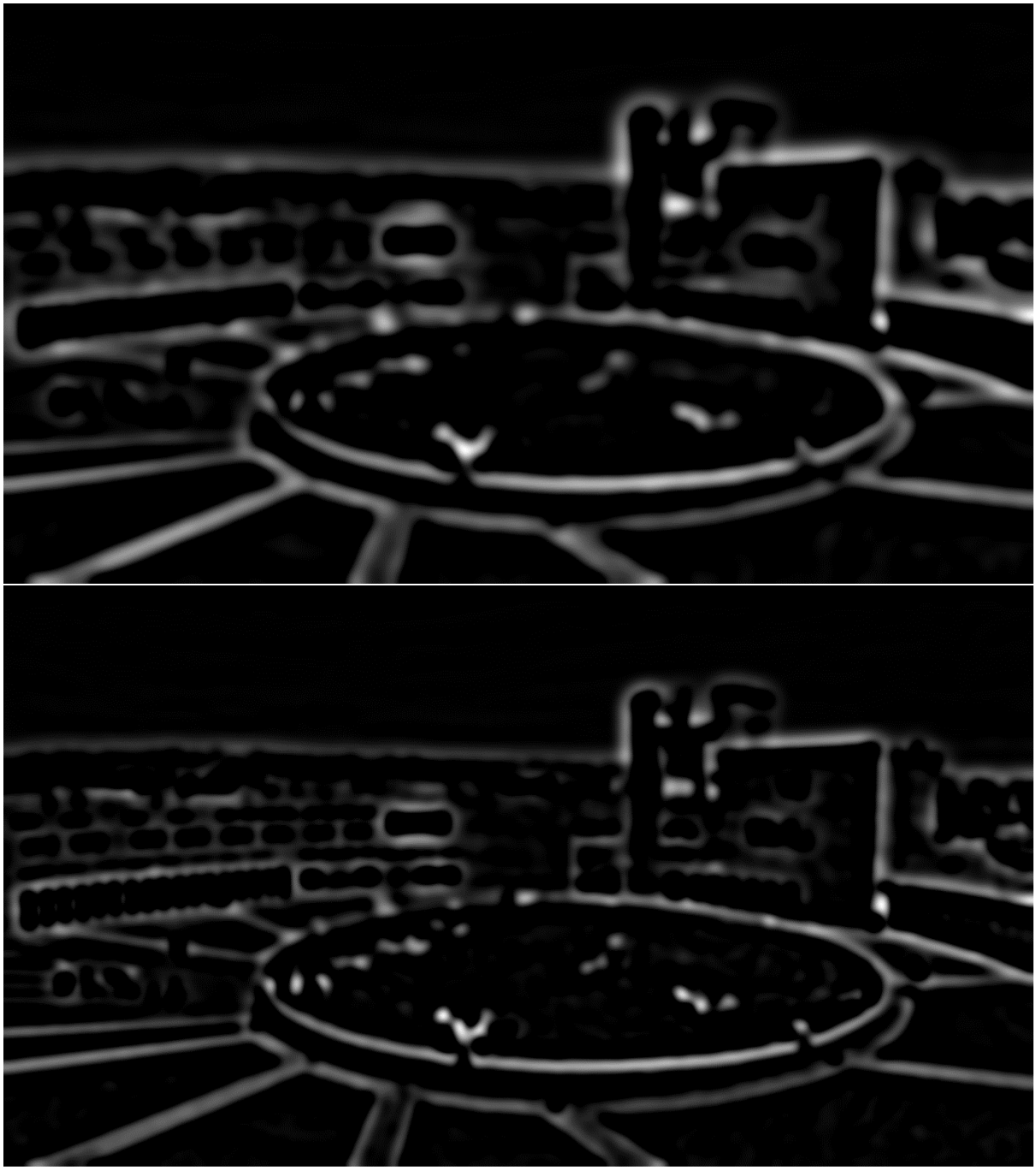
```
def gauss_blur(windowname, image, sigma):
    """Gaussian Smoothing Kernel"""
    size = 3*sigma
    name =  windowname + " Gauss Smoothened Image - sigma " + str(sigma)
    x, y = np.meshgrid(np.arange(-size, size+1),np.arange(-size, size+1))
    # x,y = np.meshgrid(np.arange(6*sigma),np.arange(6*sigma))
    smooth_kernel = np.exp(-((x-np.mean(x))**2 + (y-np.mean(y))**2)/(2*(sigma**2))) / (2*np.pi*(sigma**2))
    smooth_image = cv.filter2D(image, -1, smooth_kernel)
    plt.imshow(smooth_kernel)
    plt.title("Gaussian smoothing kernel with sigma {}".format(sigma))
    plt.show()
    display(name, smooth_image)
    return smooth_image
```

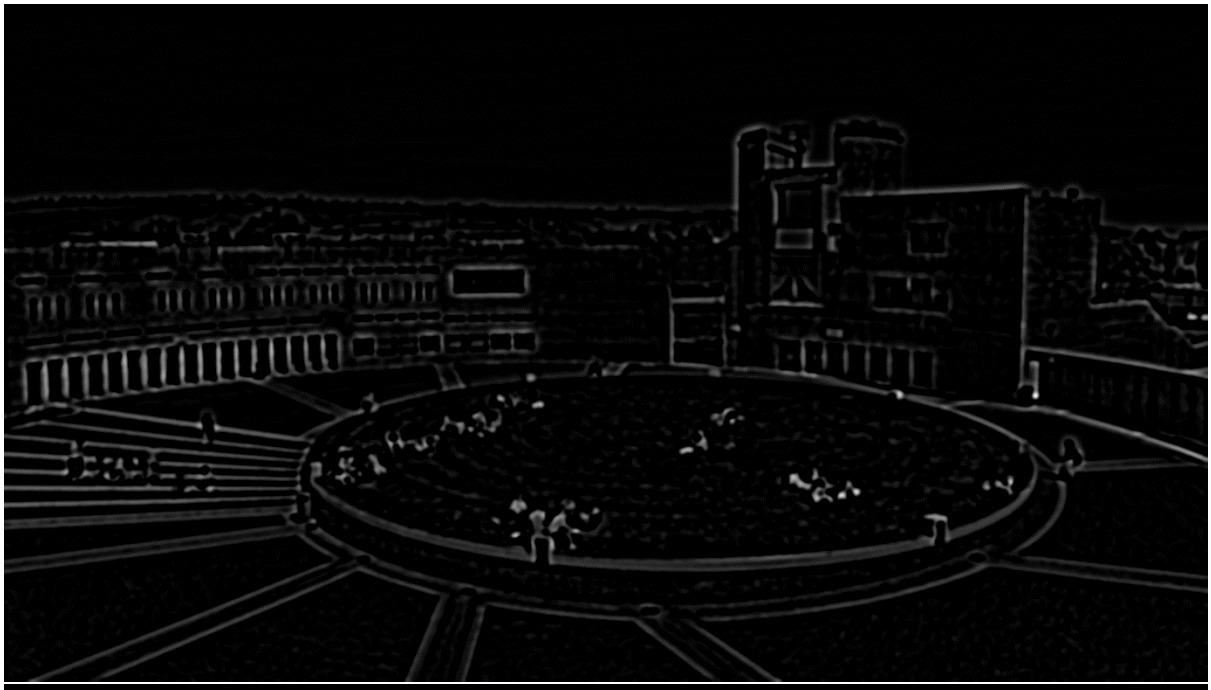
Task 2:

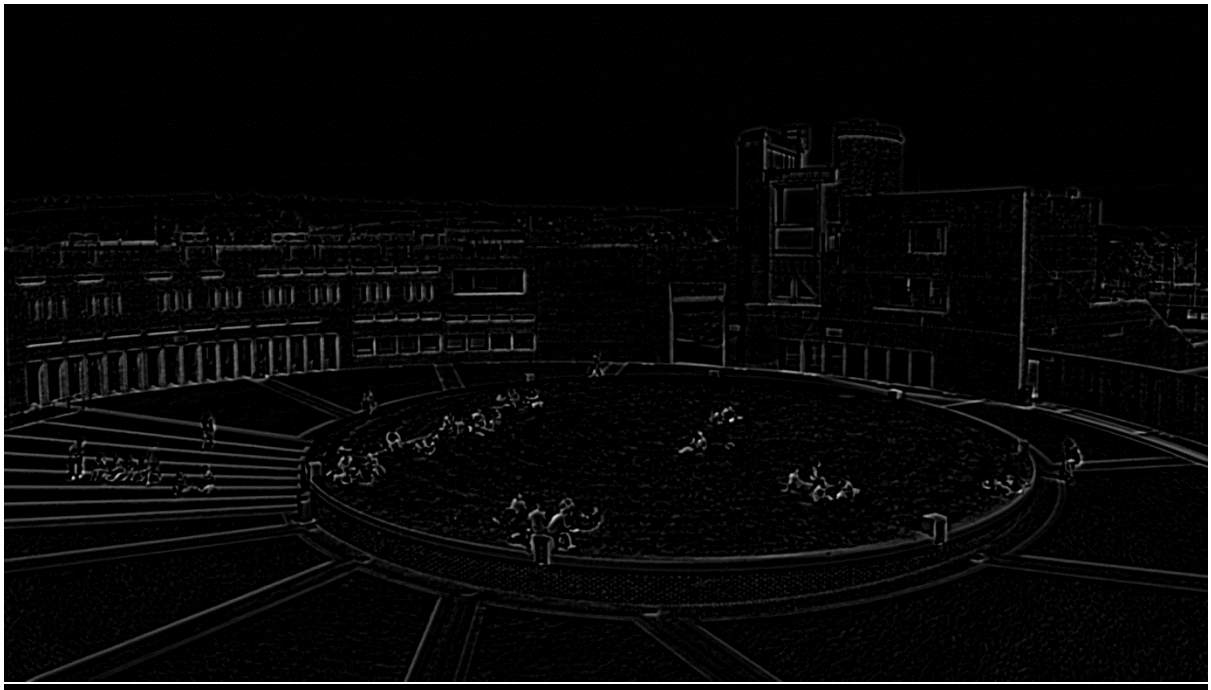
Task 2A:

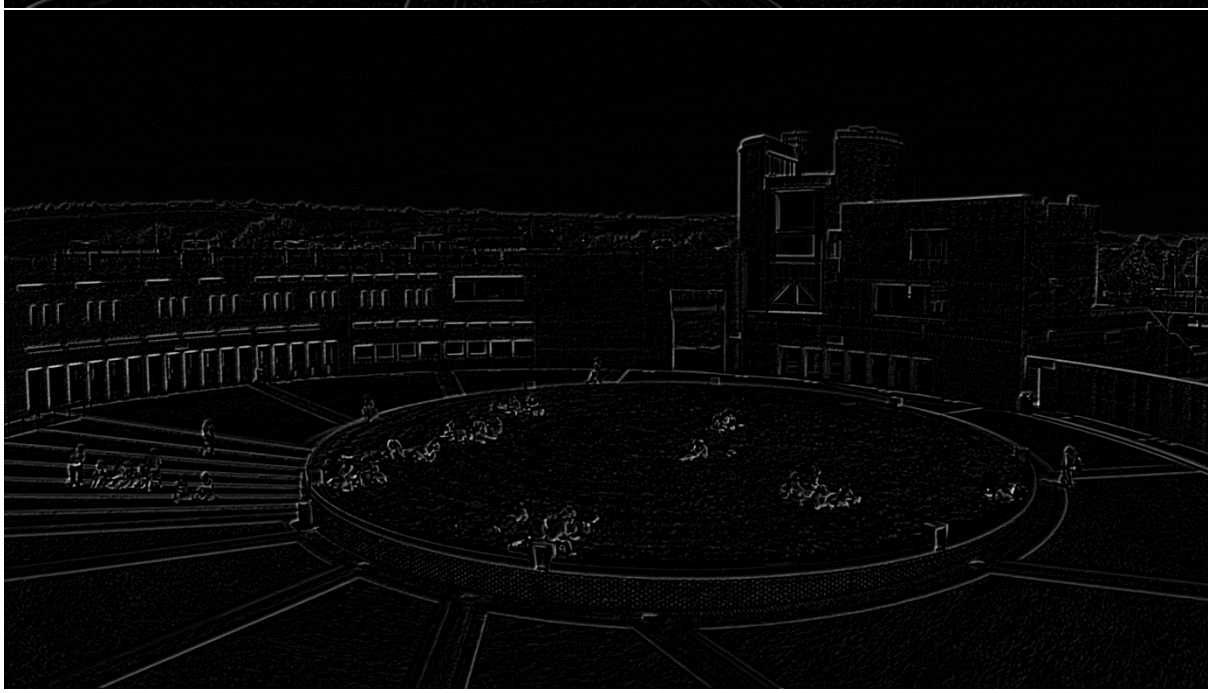
DoG Images

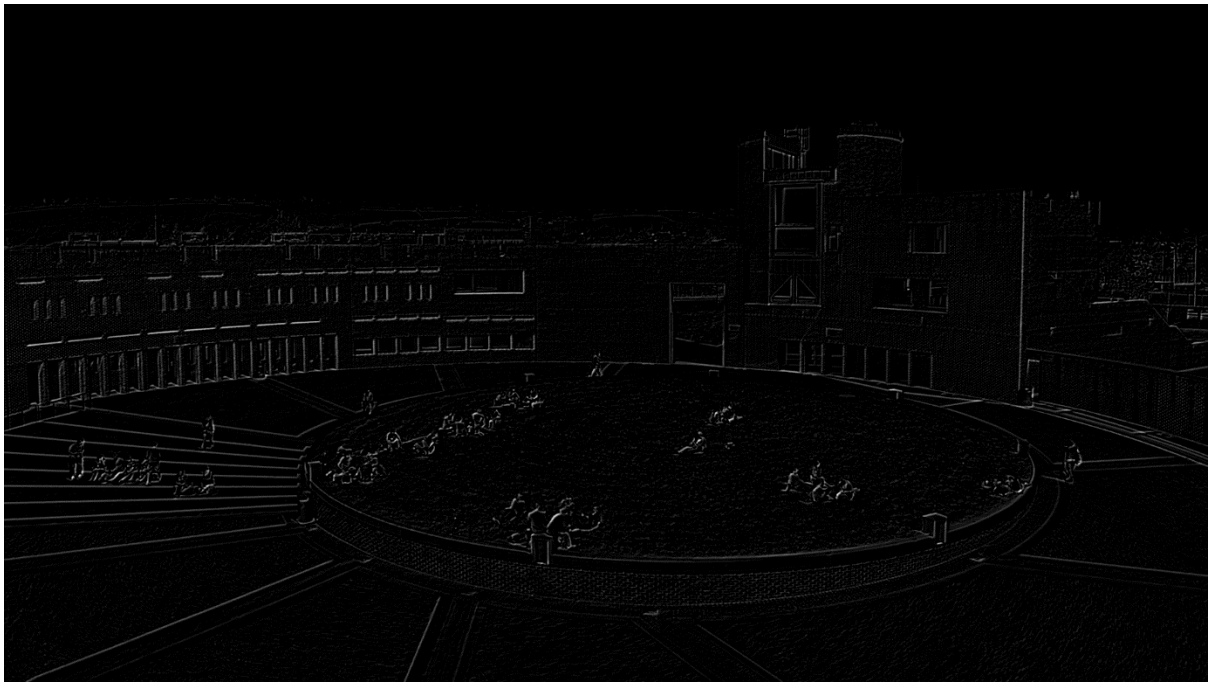












Code Snippet – Line 46

```
"""   TASK 2
DoGs and Key-Points
Iterates through all sigma values and its scale images to get DoG images"""
for i in range(len(sigmaList)-1):
    dog = scaleSpace[sigmaList[i]] -scaleSpace[sigmaList[i+1]]
    doglist[sigmaList[i]] = dog
    windowname = "DoG - Sigma "+str(sigmaList[i])+" - Sigma "+str(sigmaList[i+1])
    display(windowname, dog)
#Theresold is set to 10 and non_maxima_3D - function - checks if the current keypoint is maximum in 3D space+scale
threshold = 10
keyPoints = non_maxima_3d(doglist, threshold)
```

Task 2B/2C:

Circle the Key-Points with radius = 3*sigma

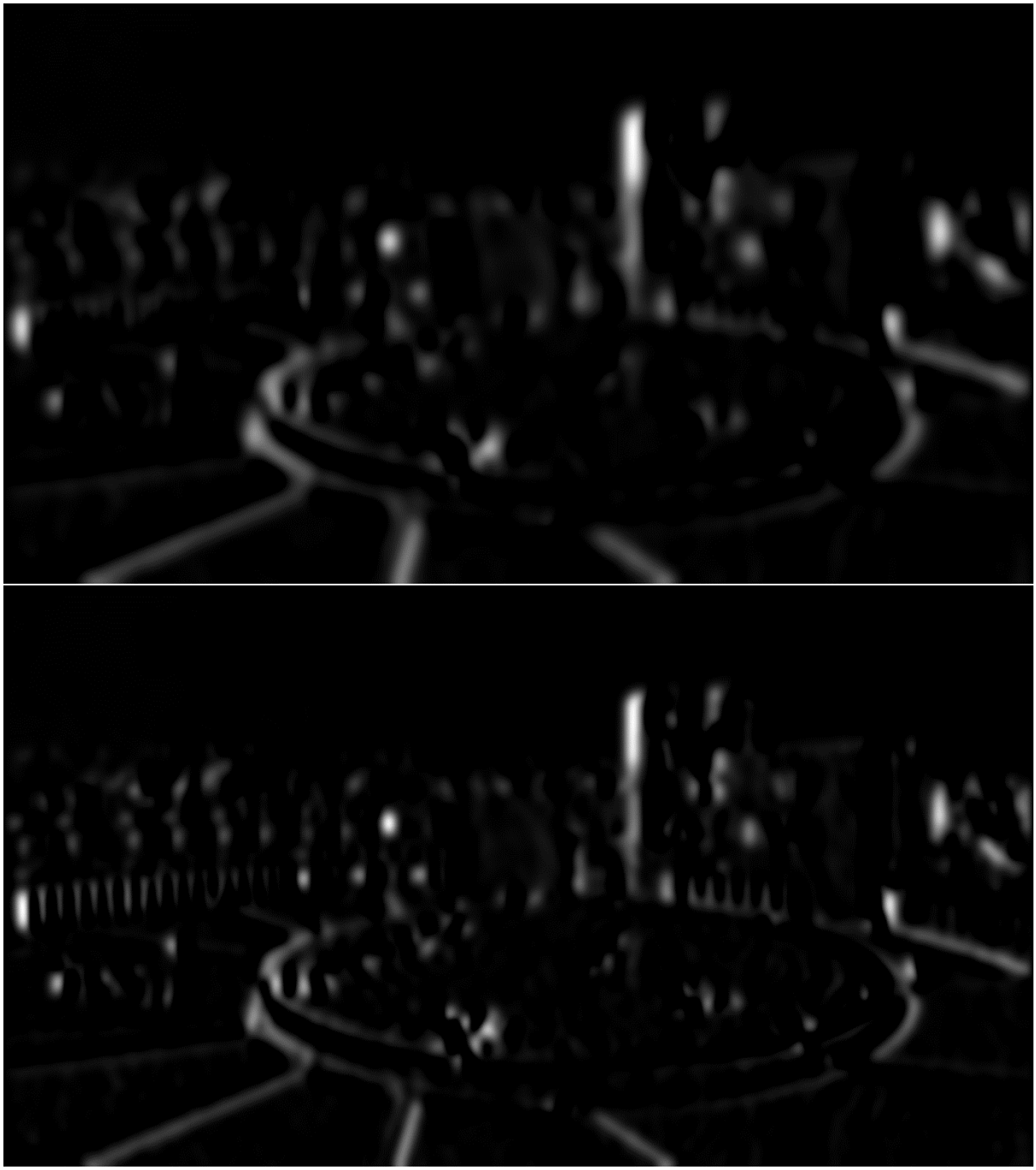
Code Snippet – Line 55

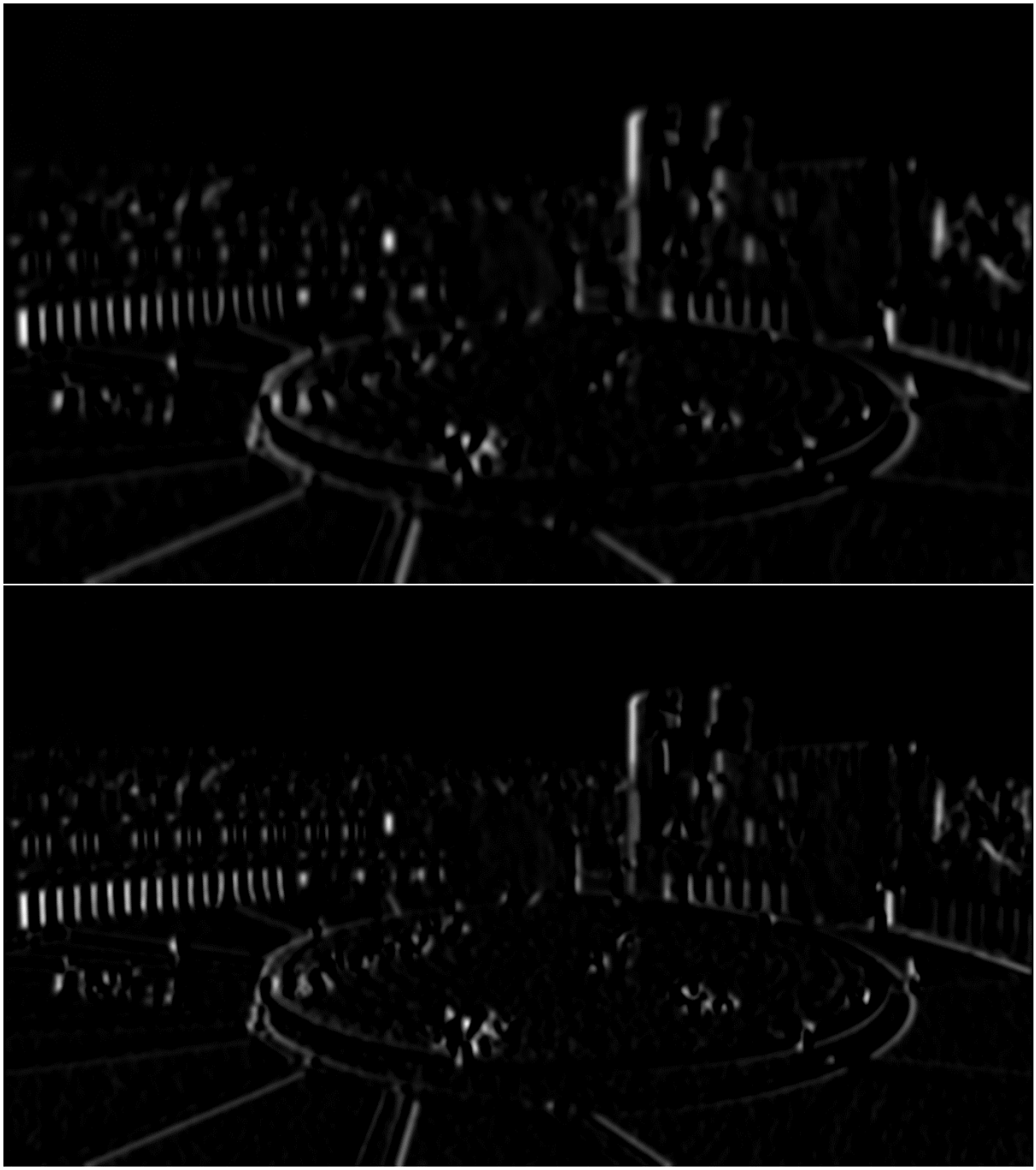
```
keyImg = copy.deepcopy(input_image)
for point in keyPoints:
    radius = round(3 * point[2])
    img = cv.circle(keyImg, (point[1], point[0]), radius, (0,0,255))
```

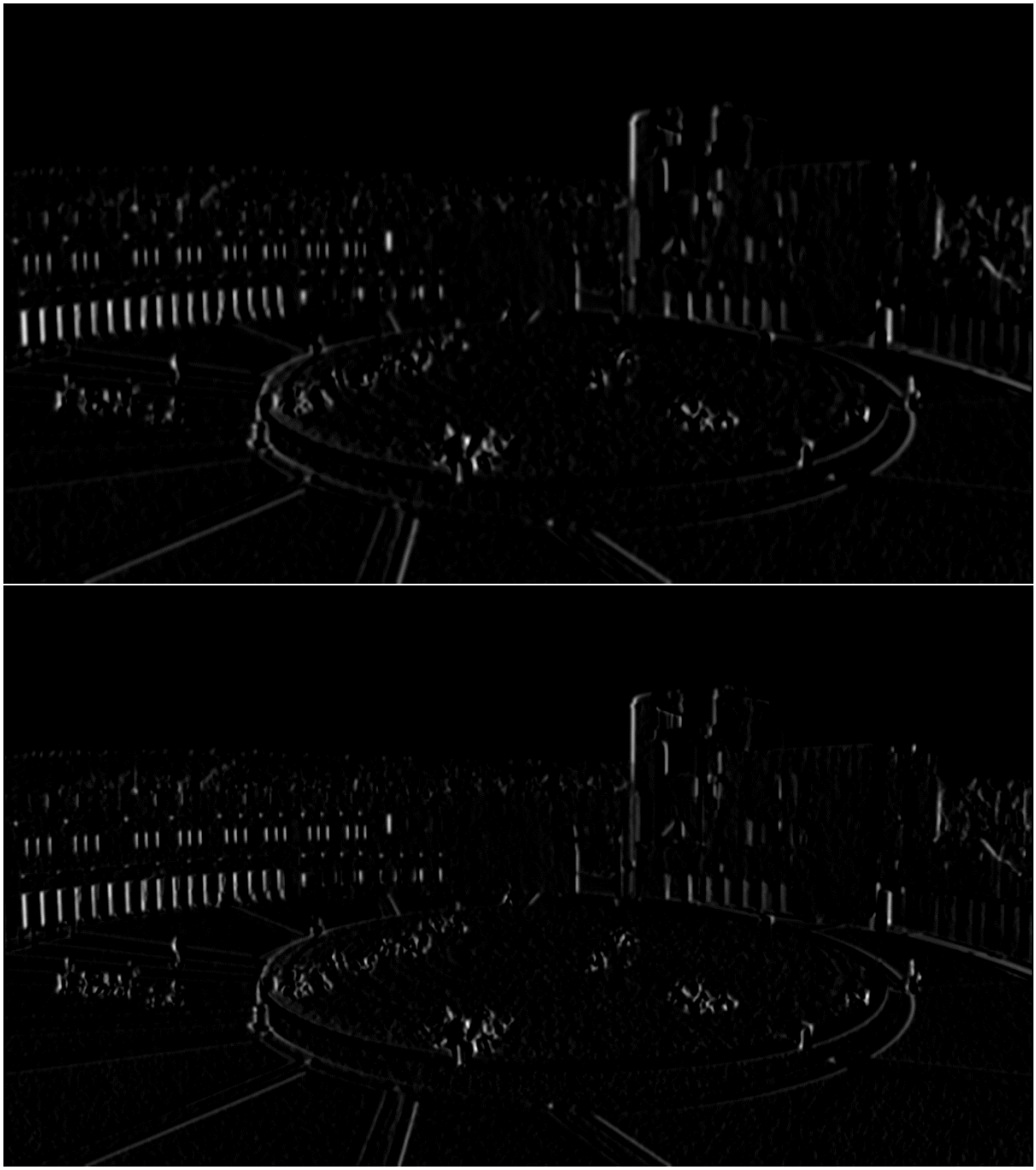
Task 3:

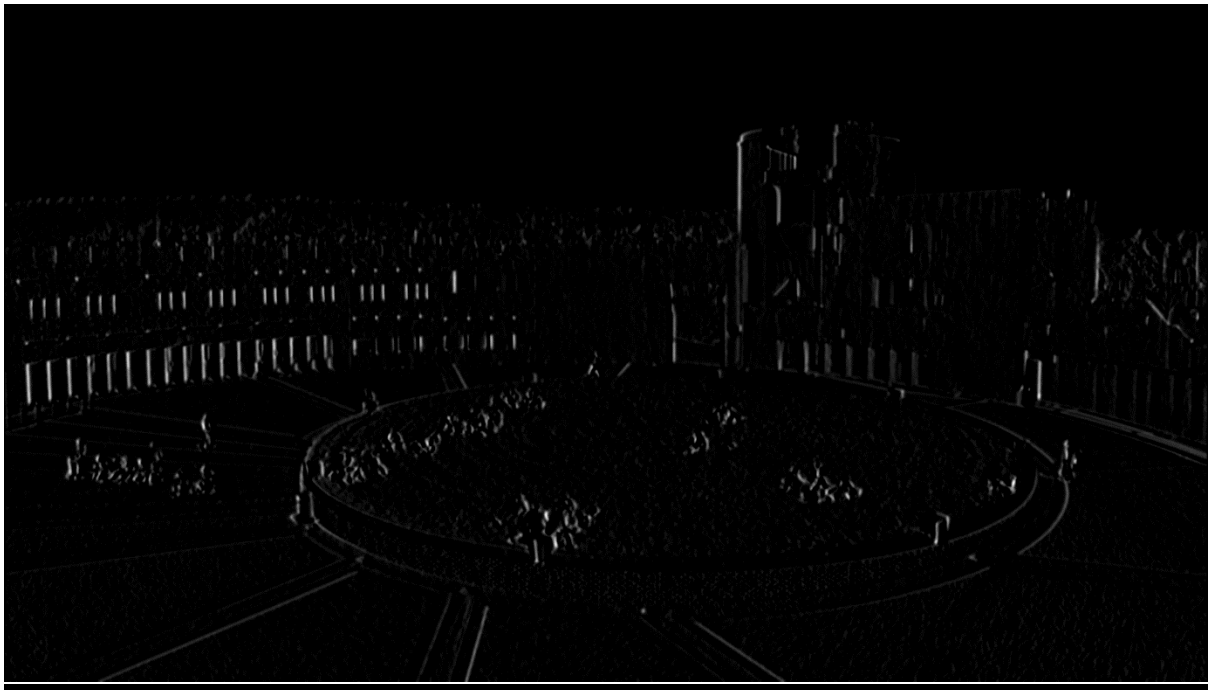
Task 3A:

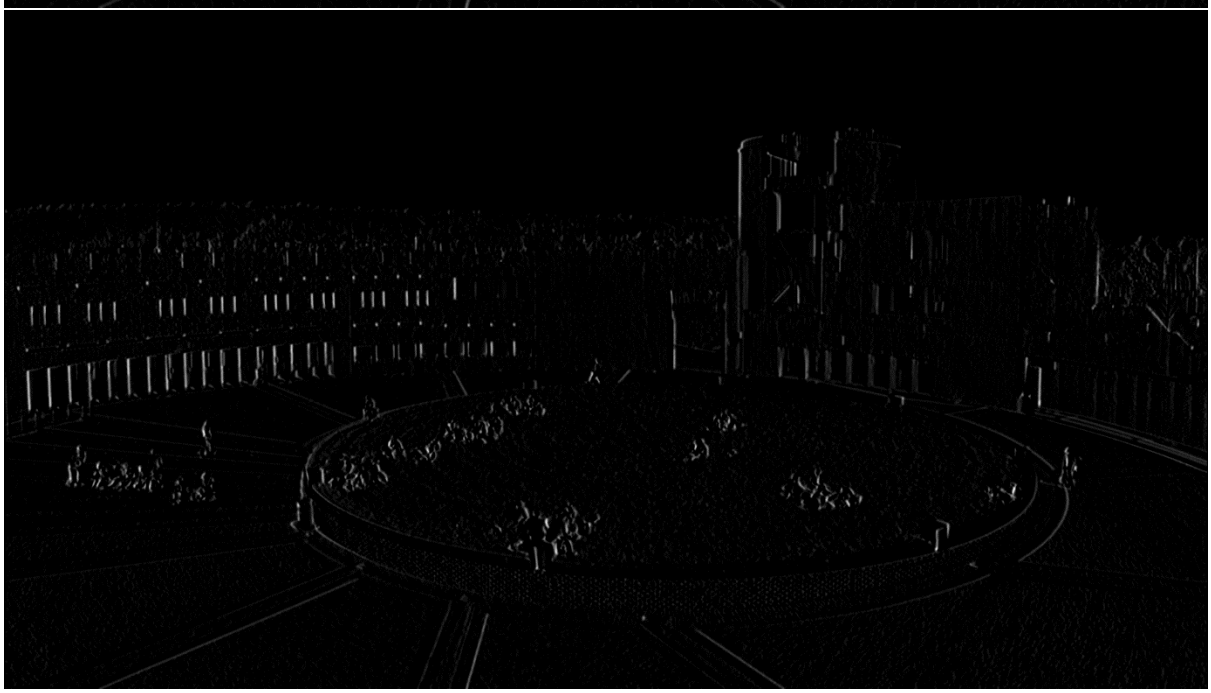
Derivatives gx

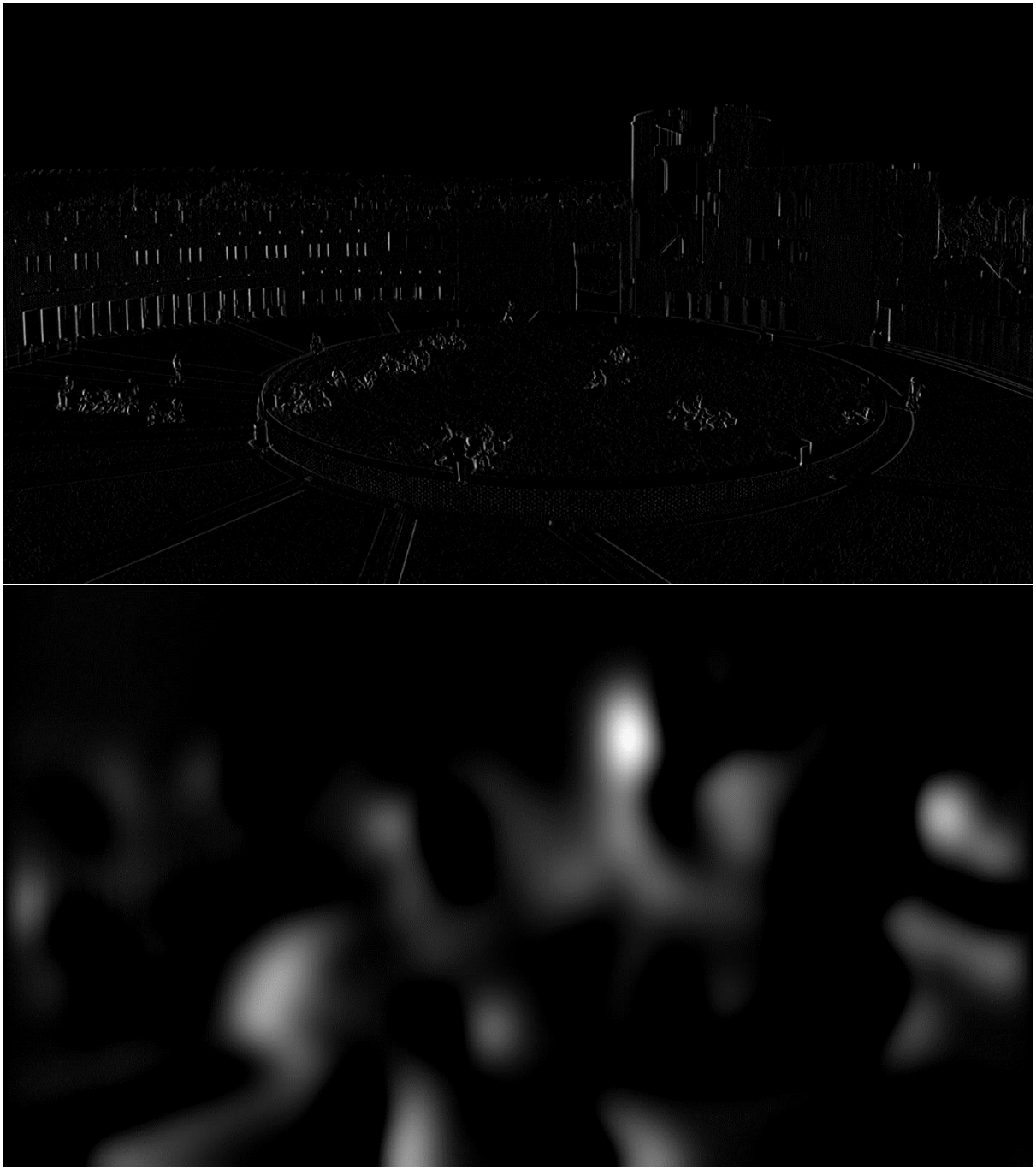




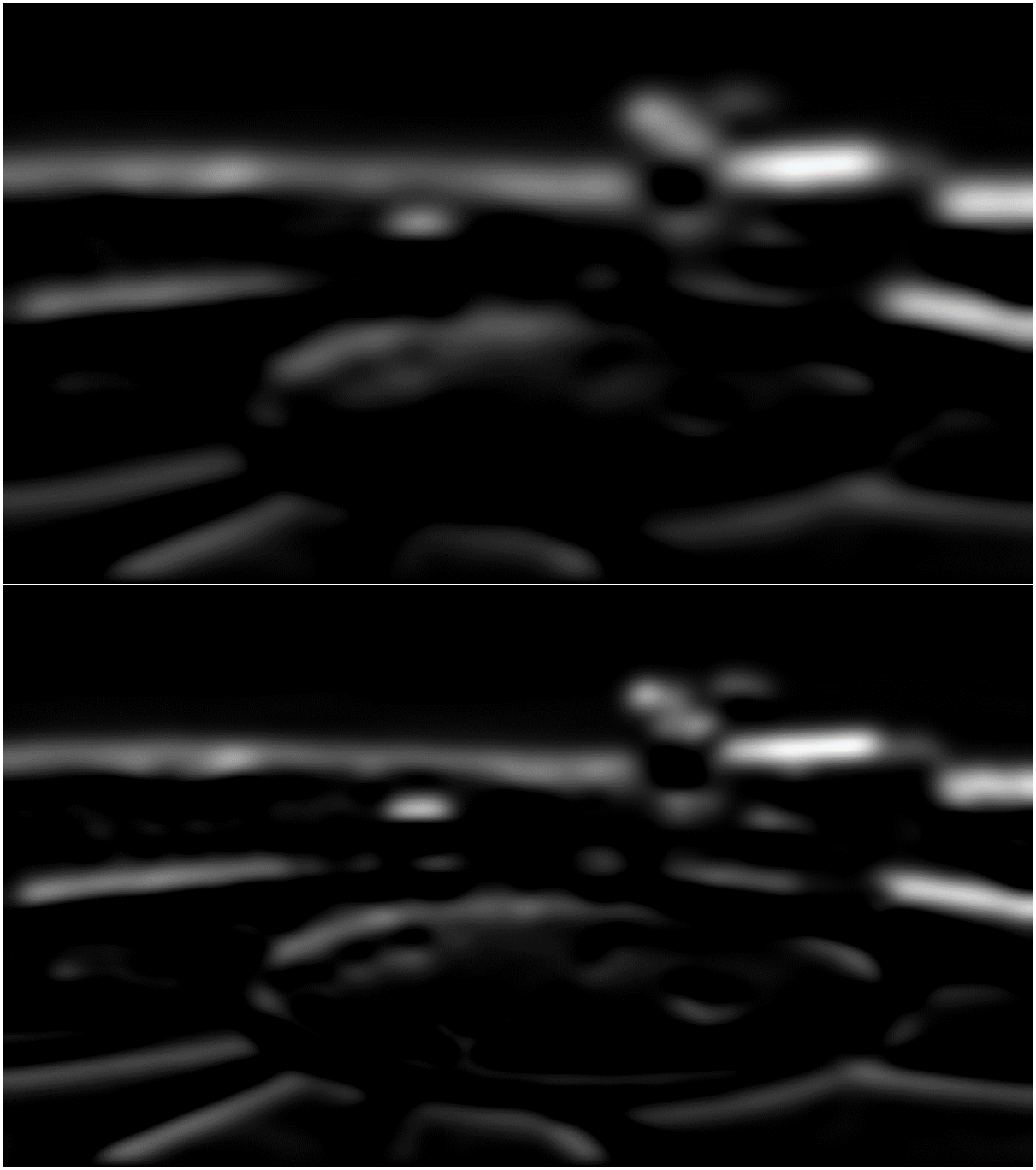


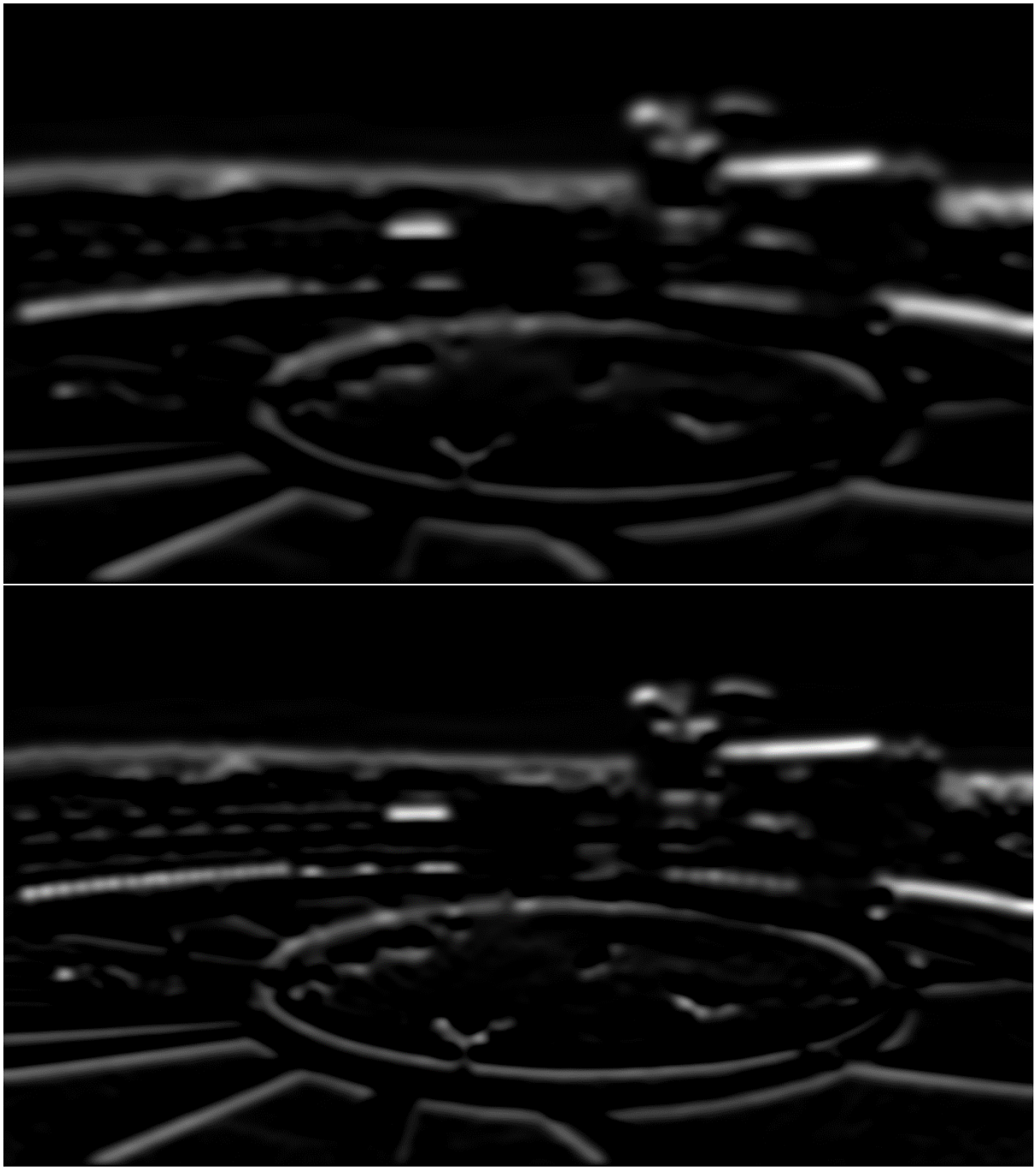


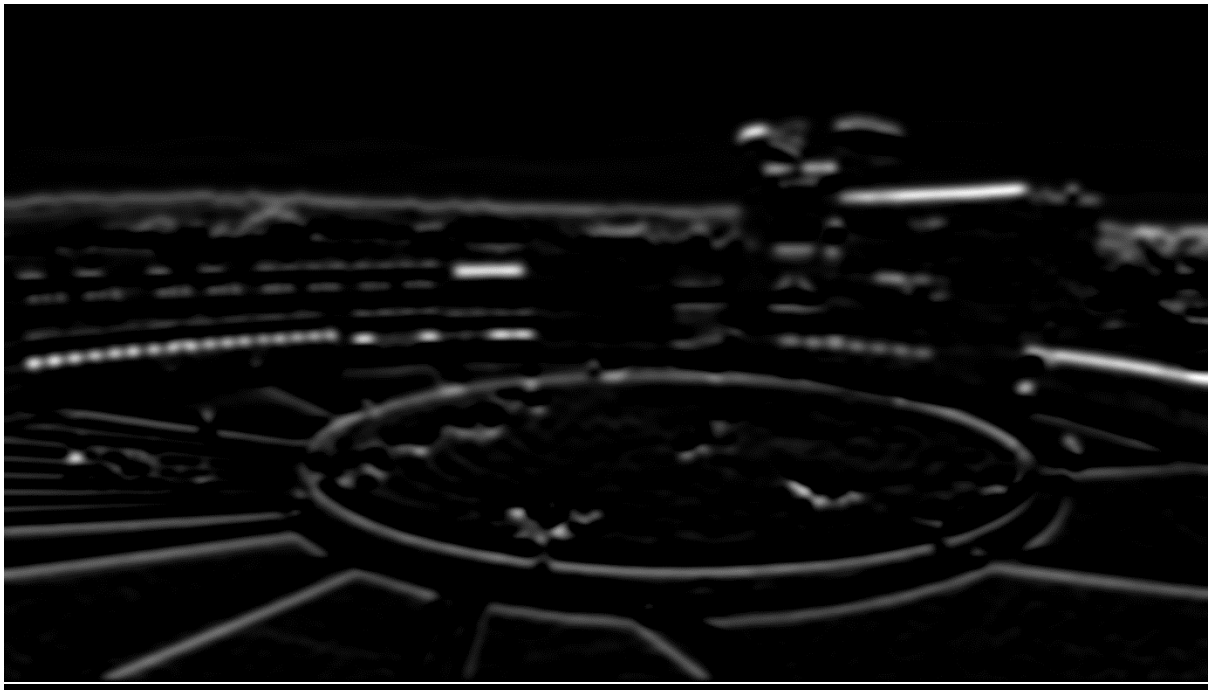


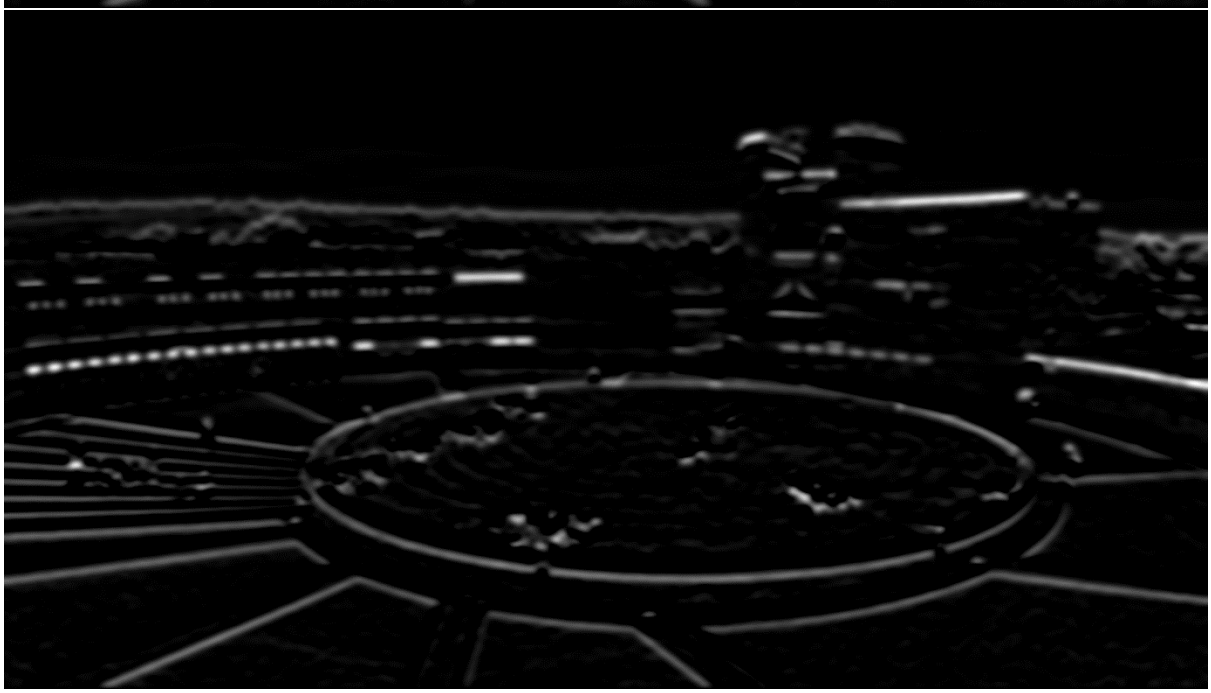


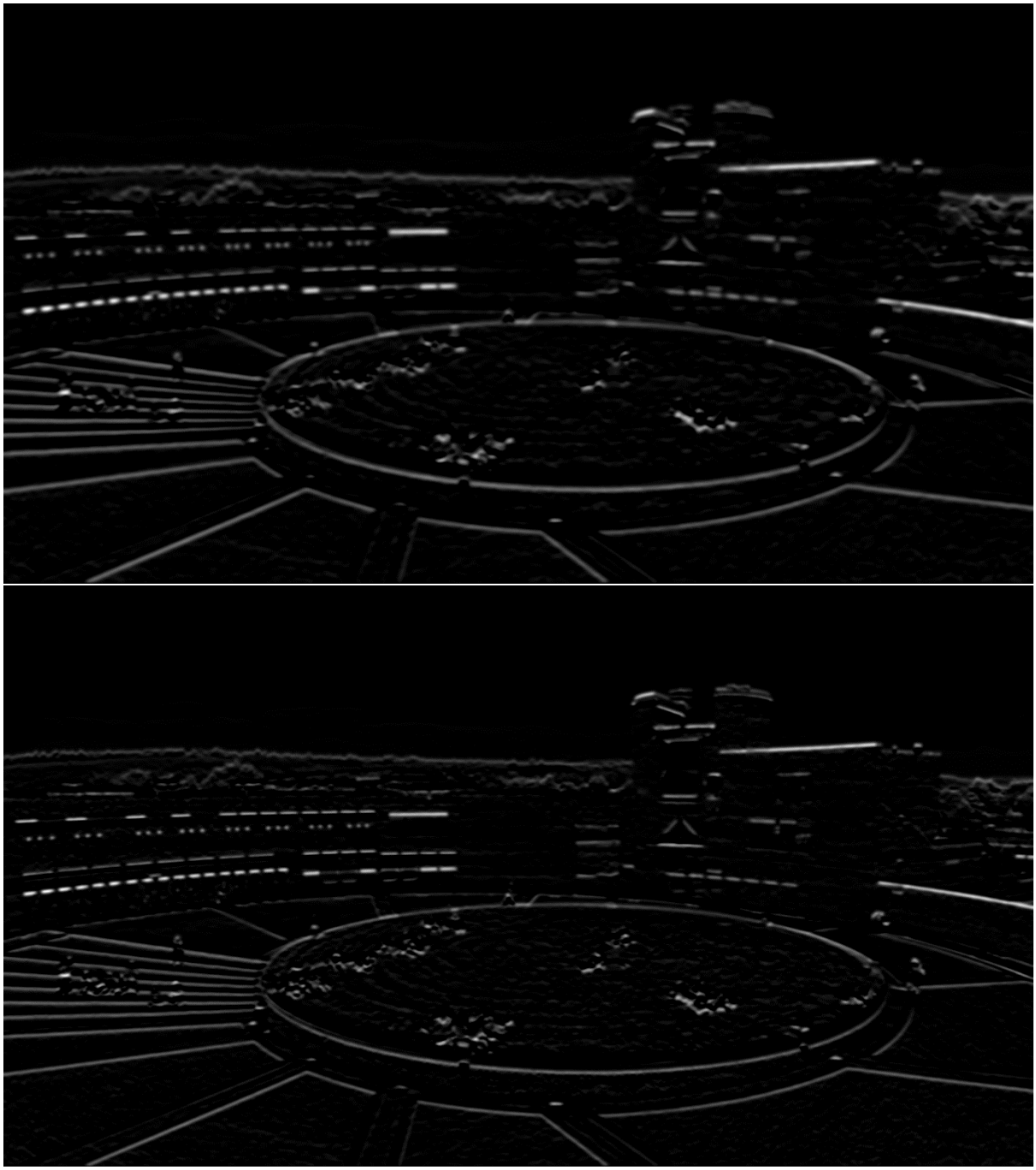
Derivative gy

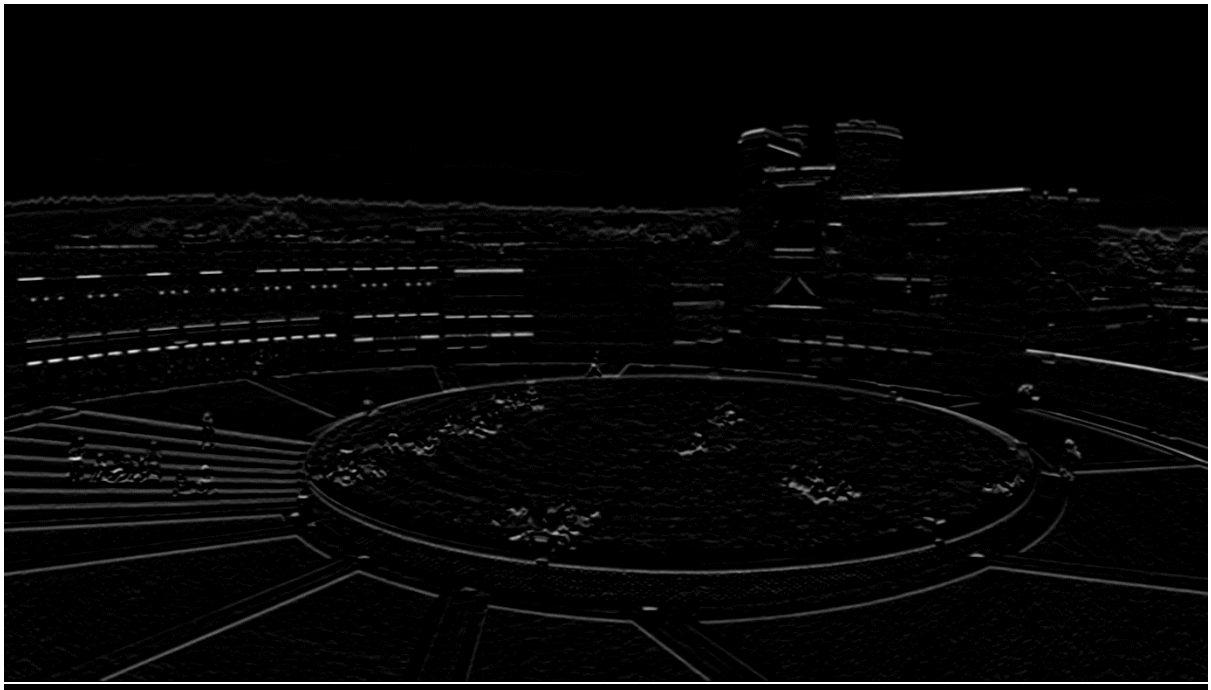


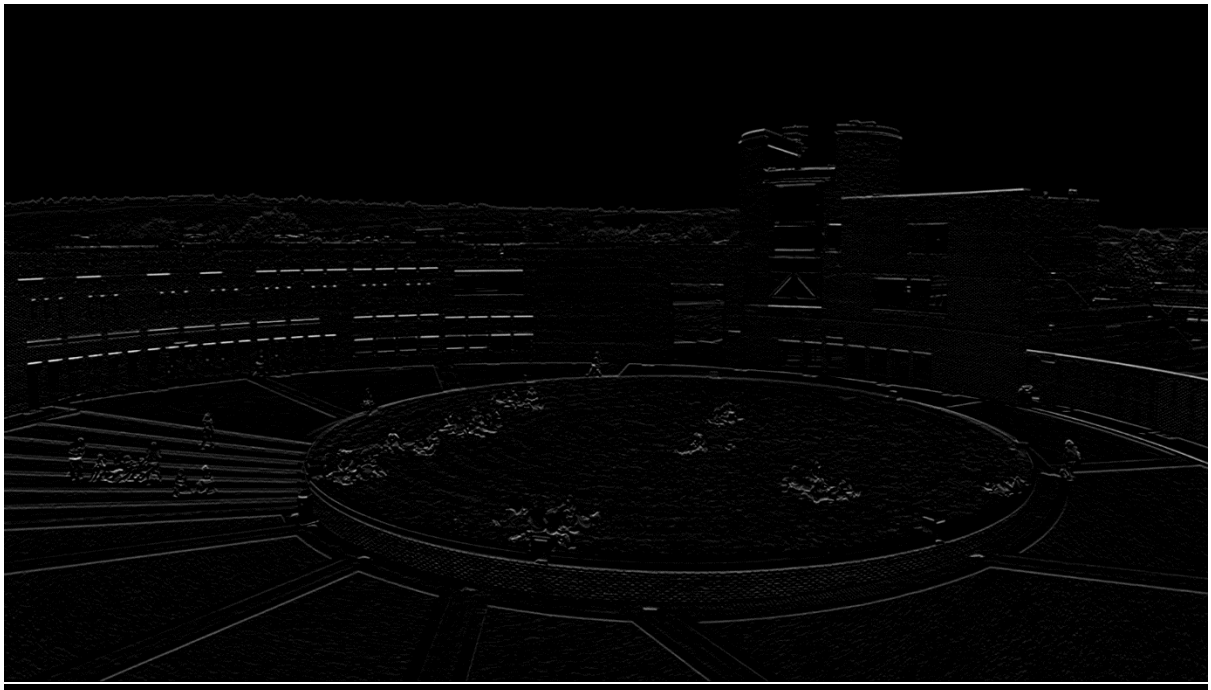


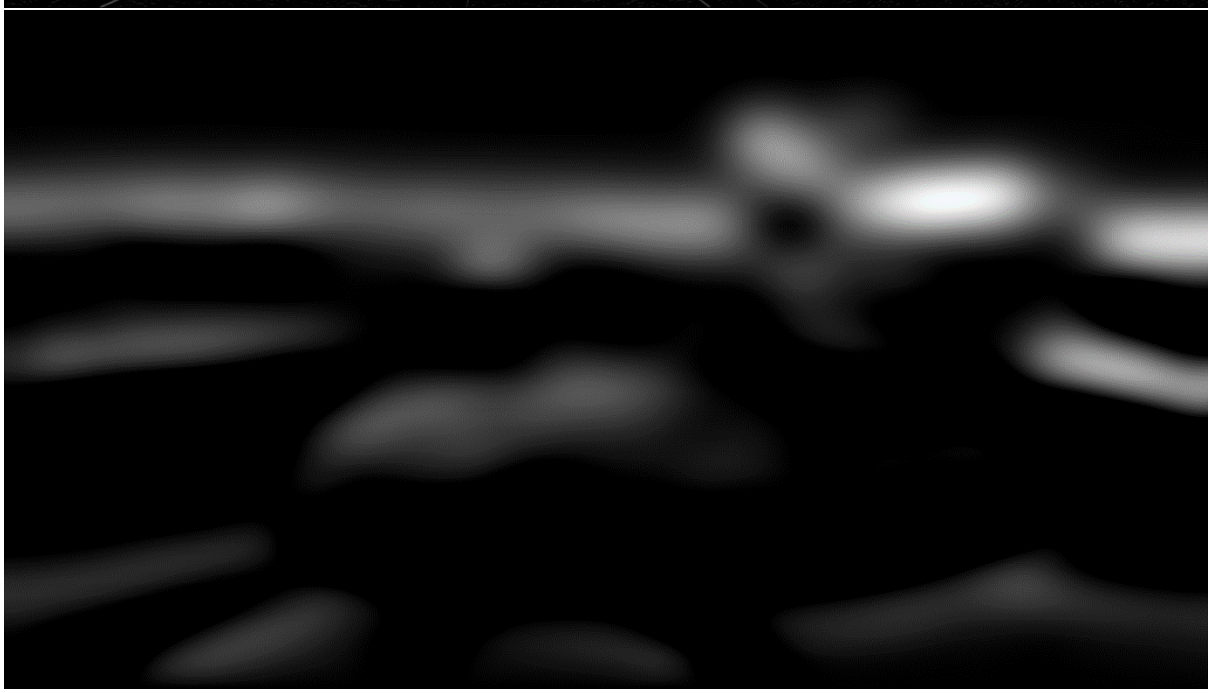












Code Snippet – line 73

```
"""task 3.a"""
for k,v in scaleSpace.items():
    gx[k] = cv.filter2D(v, -1, dx)
    windowname = "Derivative Image gx, Sigma = "+str(k)
    display(windowname,gx[k])
    gy[k] = cv.filter2D(v, -1, dy)
    windowname = "Derivative Image gy, Sigma = "+str(k)
    display(windowname,gy[k])
```

Task 3B/3C/3D:

Direction of Orientation

Code Snippet – Line 128

```
"""task 3.d"""
#draw a circle around the keypoint with radius 3*sigma and ann arrow pointing
for point in keyPointsdo:
    x,y,sigma,radians= point[0], point[1], point[2], point[3]
    radius = round(3*sigma)
    cords = (y,x)
    cv.circle(keyImg, cords, radius, (0,0,255))
    y1 = math.sin(radians)*radius
    x1 = -math.cos(radians)*radius
    end = (y + round(y1), x + round(x1))
    cv.line(keyImg, cords, end, (0,0,255))

display("KeyPoint Direction Orientation",keyImg)
```