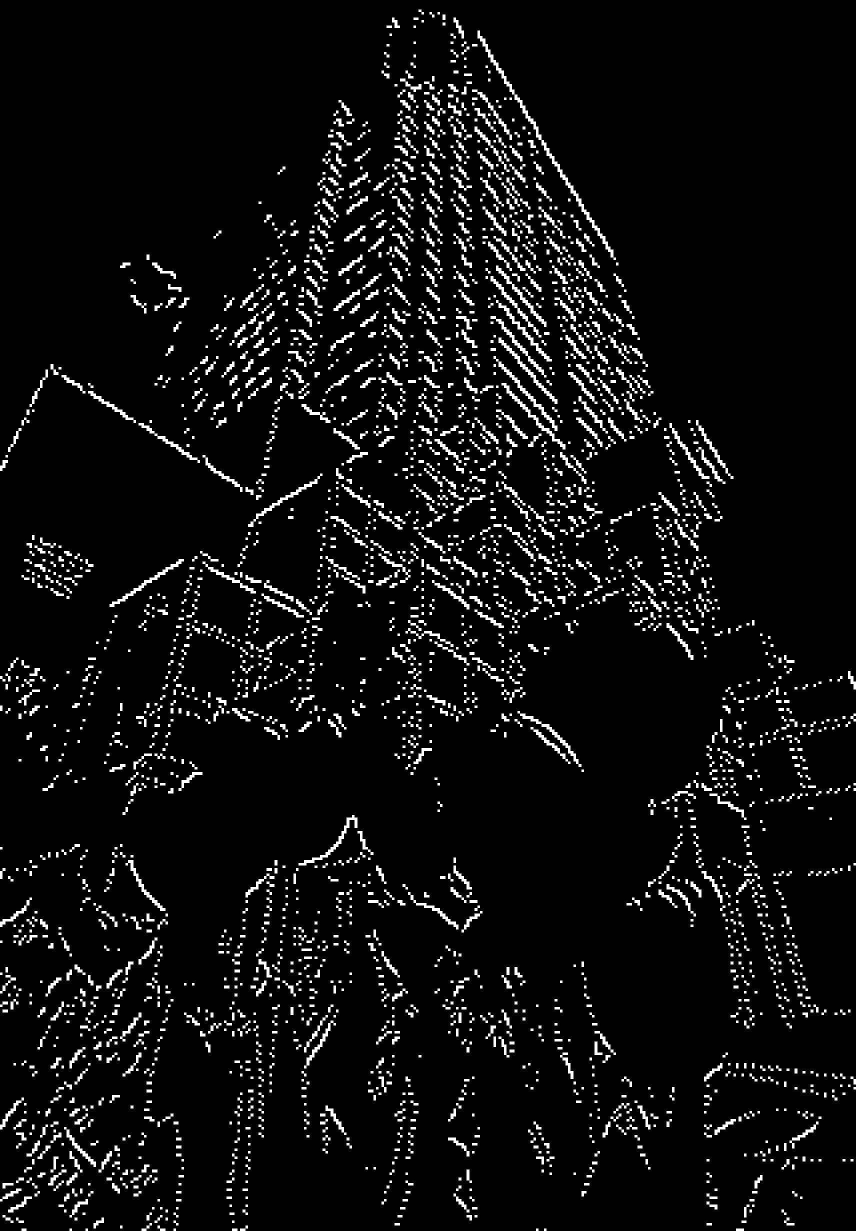
```
import math
import copy
import sys
from copy import deepcopy

img_width = 0
img_height = 0

#Source (altered slightly):
https://stackoverflow.com/questions/35723865/read-a-pgm-file-in-python
def read_pgm(img_file):
    assert img_file.readline() == b'P5\n'
    #skips the 'created by irfanview' line
    img_file.readline()

    global img_width, img_height
    (img_width, img_height) = [int(i) for i in img_file.readline().split()]
    depth = int(img_file.readline())
    assert depth <= 255

    #creating raw image matrix
    img_matrix = []
    for y in range(img_height):
        row = []
        for y in range(img_width):
            row.append(ord(img_file.read(1)))
        img_matrix.append(row)
    return img_matrix


def gaussian_blur(img_width, img_height, sigma, img_matrix):
    #creating Gaussian kernel
    kernel = []
    calc1 = 1 / (2 * math.pi * sigma**2)
    calc2 = -1 / (2 * sigma**2)
    for x in range(0-sigma*3, 0+sigma*3+1):
        row = []
        for y in range(0-sigma*3, 0+sigma*3+1):
            row.append(calc1 * math.exp(calc2 * (x**2 + y**2)))
        kernel.append(row)



    padded_img = img_matrix
    # padding existing rows by copying edges
    for p in range(img_height):
        for s in range(sigma*3):
            padded_img[p].append(padded_img[p][img_width-1+s*2])
            padded_img[p].insert(0, padded_img[p][0])
    # adding padding to top and bottom by copying edges
    for z in range(sigma*3):
        padded_img.append(padded_img[img_height-1+z*2])
        padded_img.insert(0, padded_img[0])

    #applying Gaussian kernel to padded image
    blurred_img = []
    matrix_sum = 0
    for x in range(img_height):
        blurred_row = []
        for y in range(img_width):
            for i in range(sigma*6+1):
                for j in range(sigma*6+1):
                    matrix_sum += kernel[i][j] * padded_img[x+i][y+j]
            if matrix_sum > 255:
                matrix_sum = 255
            blurred_row.append(round(matrix_sum))
            matrix_sum = 0
        blurred_img.append(blurred_row)
    return blurred_img


def sobel_edge(blurred_img):

    sobel_x = [[1, 0, -1], [2, 0, -2], [1, 0, -1]]
    sobel_y = [[1, 2, 1], [0, 0, 0], [-1, -2, -1]]

    # padding existing rows by copying edges
    for a in range(img_height):
        for b in range(1):
            blurred_img[a].append(blurred_img[a][img_width-1+b*2])
            blurred_img[a].insert(0, blurred_img[a][0])
    # adding padding to top and bottom by copying edges
    for c in range(1):
        blurred_img.append(blurred_img[img_height-1+c*2])
        blurred_img.insert(0, blurred_img[0])

    #applying sobel filters to smoothed image
    sobel_img = []
    sobel_dir = []
    sobel_sumx = 0
    sobel_sumy = 0
    for d in range(img_height):
        sobel_row = []
        sobel_dir_row = []
        for e in range(img_width):
            for f in range(3):
                for g in range(3):
                    sobel_sumx += sobel_x[f][g] * blurred_img[d+f][e+g]
                    sobel_sumy += sobel_y[f][g] * blurred_img[d+f][e+g]
            sobel_row_val = ((sobel_sumx)**2 + (sobel_sumy)**2)**(1/2)
            #clamping values that go over 8 bit storage limit
            if sobel_row_val > 255:
                sobel_row_val = 255
            #making sure I don't divide by 0 in arctan
            if sobel_sumx == 0:
                direction = math.pi/2
            else:
                direction = math.atan2(sobel_sumy, sobel_sumx)
            mag = round(sobel_row_val)
            #gradient magnitude threshold
            if mag > 90:
                sobel_row.append(mag)
            else:
                sobel_row.append(0)
            sobel_dir_row.append(direction)
            sobel_sumx = 0
            sobel_sumy = 0
        sobel_img.append(sobel_row)
        sobel_dir.append(sobel_dir_row)

    #keep values separate so sobel_img matrix can still be easily turned into an image
    return sobel_img, sobel_dir


#non-maximum suppression
def canny_edge(sobel_image, sobel_direction):

    maxsup_image = deepcopy(sobel_image)

    for y in range(len(maxsup_image)):
        for x in range(len(maxsup_image[0])):
            #checking horizontal, between 67.5 and 112.5 or -67.5 and -112.5 degrees
            if((sobel_direction[y][x] < 1.9634954 and sobel_direction[y][x] > 1.178097) or
            (sobel_direction[y][x] > -1.9634954 and sobel_direction[y][x] < -1.178097)):
                #right OOB (out of bounds)
                if(x+1 > len(maxsup_image[0])-1):
                    if maxsup_image[y][x] >= maxsup_image[y][x-1]:
                        maxsup_image[y][x-1] = 0
                    else:
                        maxsup_image[y][x] = 0
                #left OOB
                elif(x-1 < 0):
                    if maxsup_image[y][x] >= maxsup_image[y][x+1]:
                        maxsup_image[y][x+1] = 0
                    else:
```

```
            maxsup_image[y][x] = 0
    else:
        if(maxsup_image[y][x] >= maxsup_image[y][x-1] and
        maxsup_image[y][x] >= maxsup_image[y][x+1]):
            maxsup_image[y][x-1]= 0
            maxsup_image[y][x+1] = 0
        elif(maxsup_image[y][x+1] >= maxsup_image[y][x] and
        maxsup_image[y][x+1] >= maxsup_image[y][x-1]):
            maxsup_image[y][x-1] = 0
            maxsup_image[y][x] = 0
        else:
            maxsup_image[y][x] = 0
            maxsup_image[y][x+1] = 0
#checking vertical, between 22.5 and -22.5 or 157.5 and -157.5 degrees
if((sobel_direction[y][x] < 0.3926991 and sobel_direction[y][x] > -0.3926991) or
(sobel_direction[y][x] > 2.7488936 or sobel_direction[y][x] < -2.7488936)):
    #bottom OOB
    if(y+1 > len(maxsup_image)-1):
        if maxsup_image[y][x] >= maxsup_image[y-1][x]:
            maxsup_image[y-1][x] = 0
        else:
            maxsup_image[y][x] = 0
    #top OOB
    elif(y-1 < 0):
        if maxsup_image[y][x] >= maxsup_image[y+1][x]:
            maxsup_image[y+1][x] = 0
        else:
            maxsup_image[y][x] = 0
    else:
        if(maxsup_image[y][x] >= maxsup_image[y-1][x] and
        maxsup_image[y][x] >= maxsup_image[y+1][x]):
            maxsup_image[y-1][x] = 0
            maxsup_image[y+1][x] = 0
        elif(maxsup_image[y+1][x] >= maxsup_image[y][x] and
        maxsup_image[y+1][x] >= maxsup_image[y-1][x]):
            maxsup_image[y-1][x] = 0
            maxsup_image[y][x] = 0
        else:
            maxsup_image[y][x] = 0
            maxsup_image[y+1][x] = 0
# checking diagonal 1 (top left to bottom right and vice versa),
# between 22.5 and 67.5 or -112.5 and -157.5 degrees
elif((sobel_direction[y][x] <= 1.178097 and sobel_direction[y][x] >= 0.3926991) or
(sobel_direction[y][x] >= -2.7488936 and sobel_direction[y][x] <= -1.9634954)):
    #bottom right OOB
    if(y+1 > len(maxsup_image)-1 or x+1 > len(maxsup_image[0])-1):
        if maxsup_image[y][x] >= maxsup_image[y-1][x-1]:
            maxsup_image[y-1][x-1] = 0
        else:
            maxsup_image[y][x] = 0
    #top left OOB
    elif(y-1 < 0 or x-1 < 0):
        if maxsup_image[y][x] >= maxsup_image[y+1][x+1]:
            maxsup_image[y+1][x+1] = 0
        else:
            maxsup_image[y][x] = 0
    else:
        if(maxsup_image[y][x] >= maxsup_image[y-1][x-1] and
        maxsup_image[y][x] >= maxsup_image[y+1][x+1]):
            maxsup_image[y-1][x-1] = 0
            maxsup_image[y+1][x+1] = 0
        elif(maxsup_image[y+1][x+1] >= maxsup_image[y][x] and
        maxsup_image[y+1][x+1] >= maxsup_image[y-1][x-1]):
            maxsup_image[y-1][x-1] = 0
            maxsup_image[y][x] = 0
        else:
            maxsup_image[y][x] = 0
            maxsup_image[y+1][x+1] = 0
# checking diagonal 2 (top right to bottom left and vice versa),
# between 112.5 and 157.5 or -22.5 and -67.5 degrees
elif((sobel_direction[y][x] <= 2.7488936 and sobel_direction[y][x] >= 1.9634954) or
(sobel_direction[y][x] >= -1.178097 and sobel_direction[y][x] <= -0.3926991)):
    #top right OOB
    if(y+1 > len(maxsup_image)-1 or x-1 < 0):
        if maxsup_image[y][x] >= maxsup_image[y-1][x+1]:
            maxsup_image[y-1][x+1] = 0
        else:
            maxsup_image[y][x] = 0
    #bottom left OOB
    elif(y-1 < 0 or x+1 > len(maxsup_image[0])-1):
        if maxsup_image[y][x] >= maxsup_image[y+1][x-1]:
            maxsup_image[y+1][x-1] = 0
        else:
            maxsup_image[y][x] = 0
    else:
        if(maxsup_image[y][x] >= maxsup_image[y-1][x+1] and
        maxsup_image[y][x] >= maxsup_image[y+1][x-1]):
            maxsup_image[y-1][x+1] = 0
            maxsup_image[y+1][x-1] = 0
        elif(maxsup_image[y+1][x-1] >= maxsup_image[y][x] and
        maxsup_image[y+1][x-1] >= maxsup_image[y-1][x+1]):
            maxsup_image[y-1][x+1] = 0
            maxsup_image[y][x] = 0
        else:
            maxsup_image[y][x] = 0
            maxsup_image[y+1][x-1] = 0
    #brightening remaining edges
    for a in range(len(maxsup_image)):
        for b in range(len(maxsup_image[0])):
            if maxsup_image[a][b] > 0:
                maxsup_image[a][b] = 255

    return maxsup_image



def main():
    img = open(sys.argv[1], 'r+b')
    image_matrix = read_pgm(img)
    blur_img = gaussian_blur(img_width, img_height, int(sys.argv[2]), image_matrix)
    sobel_img, sobel_dir = sobel_edge(blur_img)
    maxsup_img = canny_edge(sobel_img, sobel_dir)

    file1 = input("Enter name for Sobel Gradient image: ")

    fout = open(file1, 'wb')

    pgmHeader = 'P5' + '\n' + str(img_width) + ' ' + str(img_height) + '\n' + str(255) +  '\n'
    fout.write(bytearray(pgmHeader, 'ascii'))

    for j in range(img_height):
        bnd = list(sobel_img[j])
        fout.write(bytearray(bnd))

    fout.close()

    file2 = input("Enter name for non-maximum suppression image: ")

    fout = open(file2, 'wb')

    pgmHeader = 'P5' + '\n' + str(img_width) + ' ' + str(img_height) + '\n' + str(255) +  '\n'
    fout.write(bytearray(pgmHeader, 'ascii'))

    for j in range(img_height):
        bnd = list(maxsup_img[j])
        fout.write(bytearray(bnd))

    fout.close()


if __name__ == "__main__":
    main()
```



A