Computer Assignment 1

CPE 261453 (Digital Image Processing)

โดย

นายเธียร สุวรรณกุล

รหัสนักศึกษา 620610176

เสนอ

ผศ.ดร.ศันสนีย์ เอื้อพันธ์วิริยะกุล

คณะวิศกรรมศาสตร์ มหาวิทยาลัยเชียงใหม่

Histogram and Object Moment

Task :

1. Compute the histogram of the image and determine how many objects are in the image and the gray level of each.

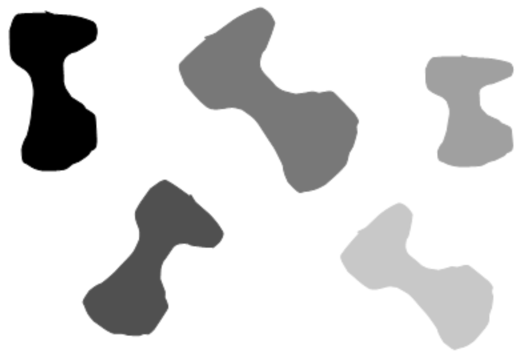


Image : scaled_shapes.pgm

2. Write a procedure to compute the (central)moment of an object given its gray level and use this procedure to compute the central moments μ20 and μ02. Using this value, compute ϕ1 and verify its invariance (proof that ϕ1 is constant).

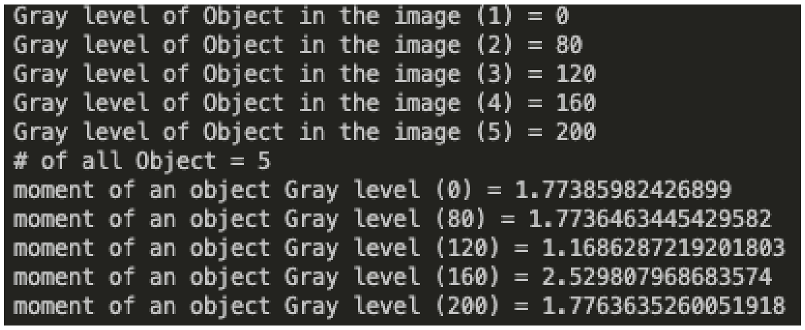
Algorithms & Formulas used :

1. Reads the header of the PGM file to obtain the image dimensions and maximum pixel value.
2. Reads the pixel values of the image into a 2D array and computes the histogram of the image.
3. Identifies objects in the image by finding gray levels with a histogram count greater than 1000, and calculates the moment of each object using the Hu moments algorithm.
4. Outputs the gray level of each object and its corresponding moment.

After implementing all of the given formulas, I modified the program to display the gray level and moment for each object in the given **scaled_shapes.pgm** file. This allows us to identify which grayscale and moment correspond to each object.

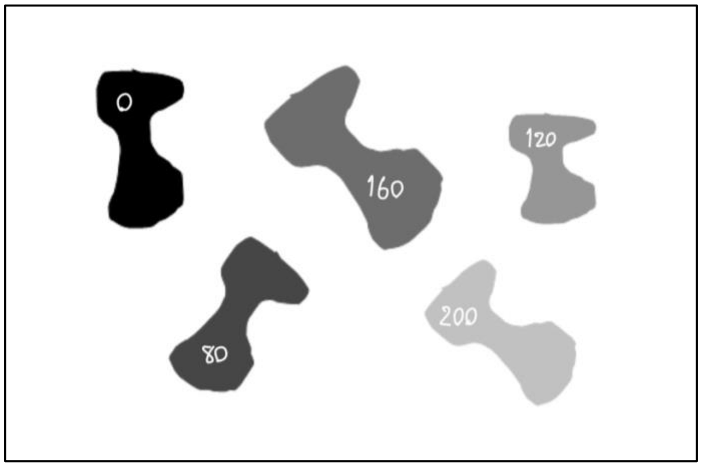
Output & Conclusion :

This is outputs of the program which show gray level of each object and count the number of objects then show each moment.



```
Gray level of Object in the image (1) = 0
Gray level of Object in the image (2) = 80
Gray level of Object in the image (3) = 120
Gray level of Object in the image (4) = 160
Gray level of Object in the image (5) = 200
# of all Object = 5
moment of an object Gray level (0) = 1.77385982426899
moment of an object Gray level (80) = 1.7736463445429582
moment of an object Gray level (120) = 1.1686287219201803
moment of an object Gray level (160) = 2.529807968683574
moment of an object Gray level (200) = 1.7763635260051918
```

From the output we can map values to each object

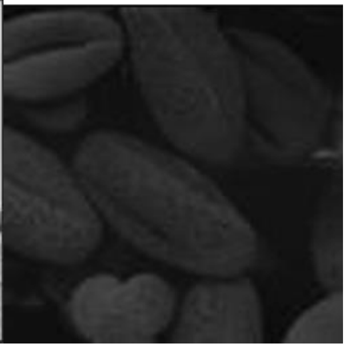


As we can see objects that have the same moment have the same size (object 1,2,5) , Larger moment value represent bigger object (object 4) , in the other hand smaller moment value represent smaller object (object 3).

Point Operation

Task :

This part involves two images, “Cameraman.pgm’ and “SEM256_256.pgm”



The goal is to improve the appearance and bring out the details using point operations.(Assume cameraman is too bright and seed is too dark)

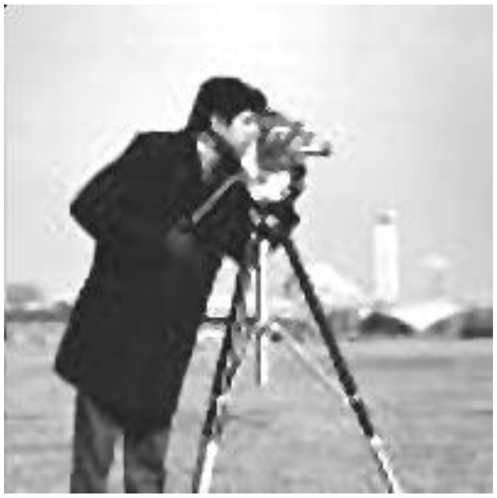
Algorithms & Formulas used :

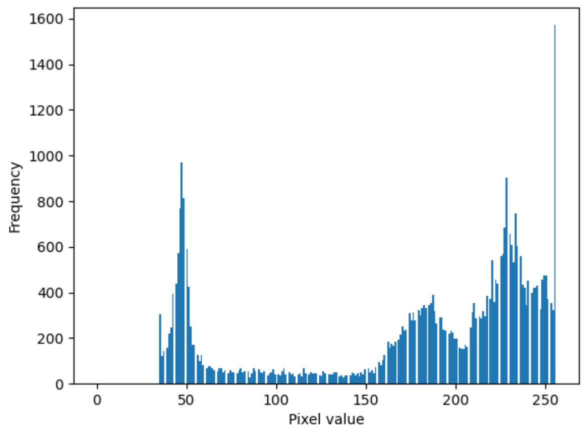
I’ve used equalization method.

1. Import image file to the program.
2. Find gray level of the image each gray level going through equalization method from the class.
3. Writing a program to show histogram of each image.

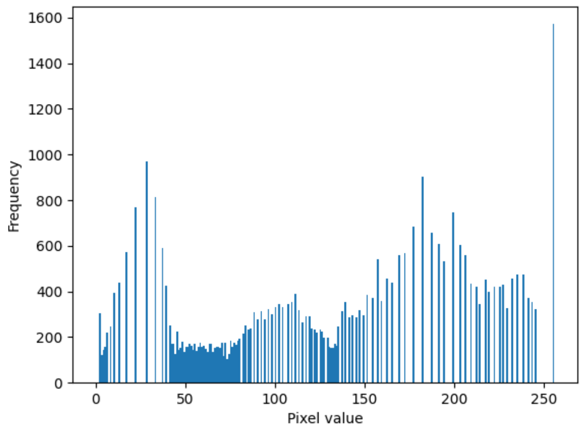
Output & Conclusion :

“Cameraman.pgm”



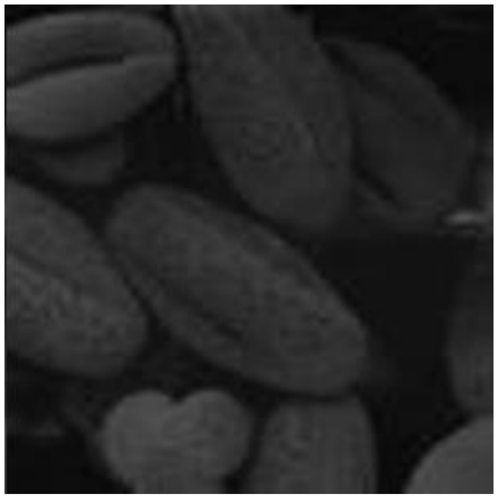


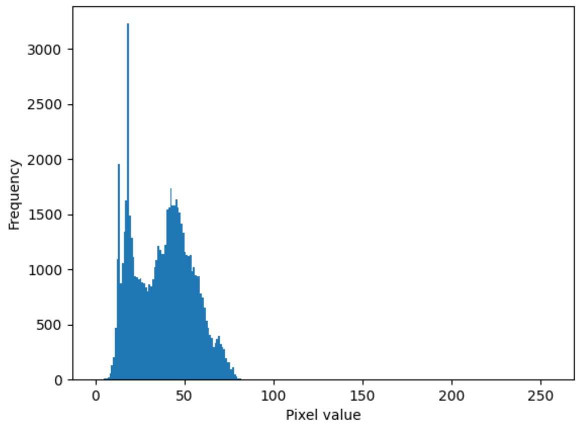
"Cameraman_improved.pgm"



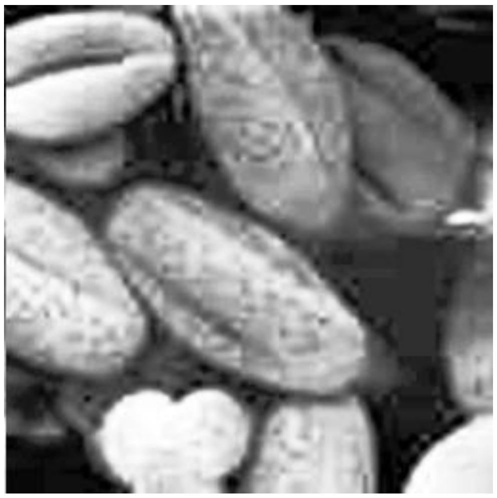
As you can see from the histogram, the first picture lacks 0-45 pixel values which results in the image not having the full range of dark pixels that it should have. Therefore, I have created an improved version of the program that covers all the gray scale values in the histogram, resulting in the improved image having more color dimensions.

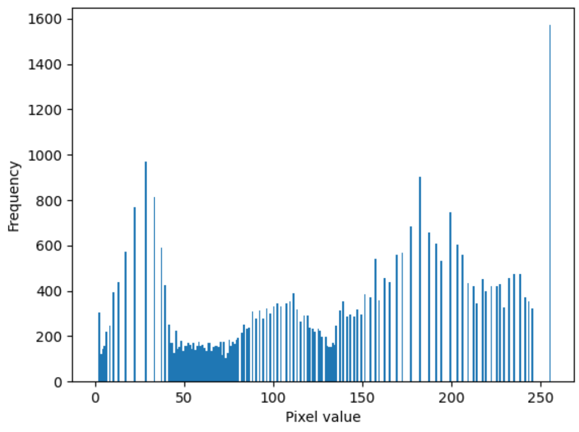
"SEM256_256.pgm"





“SEM256_256_improved.pgm”





In the same way of the previous image . The given seed image was too dark, as can be seen from the histogram. After improving it, the seed image is much brighter than the original one.

Algebraic Operations

Task:

Combine three component of R,G,B separate image to the final image

Algorithms & Formulas used :

Use a simple method by calculating gray levels.

1. Input 3 given images in different color channels to the program.
2. Use each image gray level to calculate for the result.

The 2g-r-b image is obtained by applying the following formula for each pixel (i, j) of the input images:

equl1 = 2 * p2 - p1 - p3

where p1, p2, and p3 are the pixel values of red, green, and blue channels of the input image at location (i, j).

The pixel value for the 2grb image at location (i, j) is set to equl1, after clipping it between 0 and 255.

The r-b image is obtained by applying the following formula for each pixel (i, j) of the input images:

equl2 = p1 - p3

where p1, and p3 are the pixel values of red, and blue channels of the input image at location (i, j).

The pixel value for the rb image at location (i, j) is set to equl2, after clipping it between 0 and 255.

The r+g+b/3 image is obtained by applying the following formula for each pixel (i, j) of the input images:

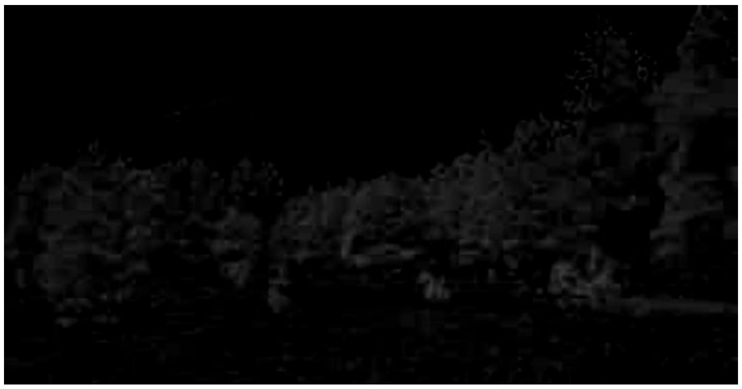
equl3 = round((p1 + p2 + p3) / 3)

where p1, p2, and p3 are the pixel values of red, green, and blue channels of the input image at location (i, j).

The pixel value for the r+g+b/3 image at location (i, j) is set to equl3, after clipping it between 0 and 255.

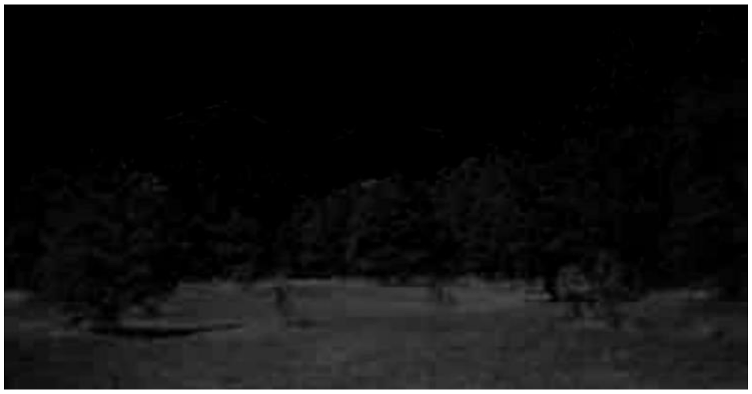
Output & Conclusion :

2 g-r-b (excess green) this generally emphasizes trees and grass.



As you can see, the image has a remarkable amount of detail in the trees and leaves, cause of the blue channel has been removed along with the red channel.

r-b (red-blue difference)



This image show most of the ground area and a little bit of trees, cause of the ground has high r channel and then we remove b channel so it’s mostly on the ground.

sorry for my grammar though

(r+b+g)/3

This image almost like the original given images despite of the equation we use this should be the finish image.

Geometric Operation

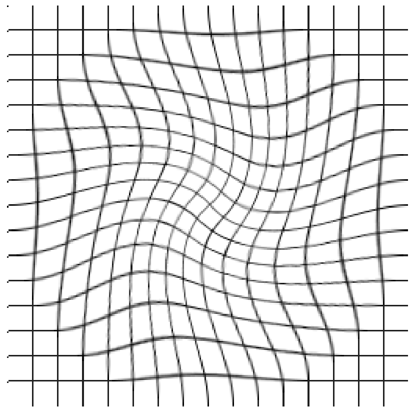
Task;

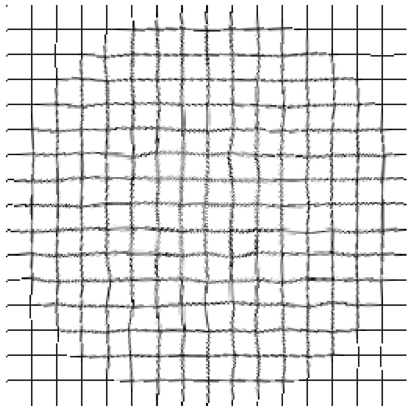
Use given "grid.pgm" and "disgrid.pgm" to undistort lady Lenna "distlenna.pgm"

Algorithms & Formulas used :

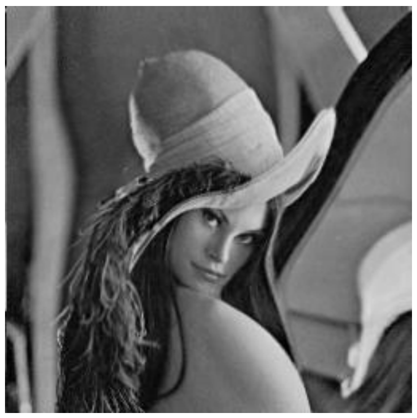
1. Define all grids angle and stored it in array (16x16)
2. Compare 1. to the distorted grid
3. Calculate w1-w8 of Bilinear Interpolation
4. Bring the value to re position of grid if there's no defined position then round it for nearest neighbor
5. Apply to lady Lenna

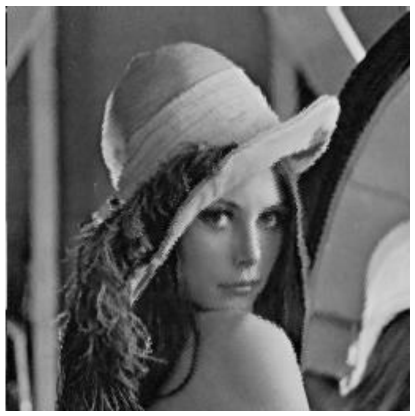
Output & Conclusion :





After running the program, the grid is undistorted but not perfectly aligned because some of the lines in the given disgrid.pgm file are not in the exact position of the array. Therefore, we use rounding to make it more accurate.





Now I’ve apply the program to poor lady Lenna the result is quite good , if you don’t zoom in close enough it’s almost perfect lady Lenna!

Code (All implement in Python)

Histogram and object moment

```python
import math

def next(stream):
    bytes = []
    while True:
        b = stream.read(1)
        if b != b'':
            c = b.decode('utf-8')
            if c == '#':
                while True:
                    d = stream.read(1)
                    if d == b'' or d == b'\n' or d == b'\r':
                        break
            elif not c.isspace():
                bytes.append(b)
            elif len(bytes) > 0:
                break
        else:
            break
    return b''.join(bytes).decode('utf-8')

def pqmoment(p, q, image):
    m = 0
    row, col = len(image), len(image[0])
    for x in range(row):
        for y in range(col):
            if image[x][y] > 0:
                m += math.pow(x, p) * math.pow(y, q) * 1
            else:
                m += 0
    return m

def pqHu(p, q, image):
    u = 0
    row, col = len(image), len(image[0])
    m10 = pqmoment(1, 0, image)
    m00 = pqmoment(0, 0, image)
    m01 = pqmoment(0, 1, image)
    for x in range(row):
        for y in range(col):
            if image[x][y] > 0:
                u += math.pow((x - (m10 / m00)), p) * math.pow((y - (m01 / m00)), q)
            else:
                u += 0
    return u

def pqN(p, q, image):
    n = 0
    row, col = len(image), len(image[0])
    n = pqHu(p, q, image) / math.pow(pqHu(0, 0, image), ((p + q / 2) + 1))
    return n

if __name__ == '__main__':
    with open('DIP1/HistrogramAndObjectMoment/scaled_shapes.pgm', 'rb') as stream_in:
        next(stream_in)
        row = int(next(stream_in))
        col = int(next(stream_in))
        max_val = int(next(stream_in))

        image = [[0] * col for _ in range(row)]
        D = [0] * (max_val + 1)
        q = []

        for i in range(row):
            for j in range(col):
                p = int.from_bytes(stream_in.read(1), byteorder='big')
                image[i][j] = p
                D[p] += 1


        for i in range(max_val):
            if D[i] > 1000:
                if not q:
                    print('Gray level of Object in the image (1) = ' + str(i))
                else:
                    print('Gray level of Object in the image (' + str(len(q)+1) + ') = ' + str(i))
                q.append(i)

        q = sorted(q, key=lambda x: x == 0, reverse=True)
        print('# of all Object =', len(q))
        for removedele in q:
            m = [[0] * col for _ in range(row)]
            for i in range(row):
                for j in range(col):
                    if image[i][j] == removedele:
```

```
            m[i][j] = 1
    theata = pqN(2, 0, m) + pqN(0, 2, m)
    print('moment of an object Gray level (' + str(removedele) + ') = ' + str(theata))
```

Point Operation

```
import math

def next(stream):
    bytes = []
    while True:
        b = stream.read(1)
        if b != b'':
            c = b.decode('utf-8')
            if c == '#':
                while True:
                    d = stream.read(1)
                    if d == b'' or d == b'\n' or d == b'\r':
                        break
            elif not c.isspace():
                bytes.append(b)
            elif len(bytes) > 0:
                break
        else:
            break
    return b''.join(bytes).decode('utf-8')

def main():
    with open('DIP1/PointOperation/SEM256_256.pgm', 'rb') as stream_in, open('DIP1/PointOperation/SEM256_256_improved.pgm', 'wb') as 
stream_out:
        next(stream_in)
        row = int(next(stream_in))
        col = int(next(stream_in))
        max_val = int(next(stream_in))

        image = [[0] * col for _ in range(row)]
        HA = [0] * (max_val + 1)
        PA = [0] * (max_val + 1)
        DA = [0] * (max_val + 1)
        DP = [0] * (max_val + 1)
        DB = [0] * (max_val + 1)

        for i in range(row):
            for j in range(col):
                p = int.from_bytes(stream_in.read(1), byteorder='big')
                image[i][j] = p
                DA[p] += 1
        for i in range(max_val + 1):
            HA[i] = DA[i] / (row * col)
            if i == 0:
                PA[i] = HA[i]
            else:
                PA[i] = PA[i - 1] + HA[i]
            DP[i] = max_val * PA[i]
            DB[i] = round(DP[i])

        for i in range(max_val + 1):
            print(f"{i} {DA[i]} {HA[i]} {PA[i]} {DP[i]} {DB[i]}")

        stream_out.write("P5\n".encode())
        stream_out.write(f"{col} {row}\n".encode())
        stream_out.write(f"{max_val}\n".encode())
        for i in range(row):
            for j in range(col):
                p = image[i][j]
                image[i][j] = DB[p]
                stream_out.write(bytes([image[i][j]]))

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

Histogram render

```
import matplotlib.pyplot as plt
import numpy as np
# Open the PGM image in binary mode
with open('Cameraman.pgm', 'rb') as f:
    # Read the header
    f.readline() # skip the P5 magic number
    width, height = map(int, f.readline().split())
    max_val = int(f.readline())
    # Read the pixel values into a 2D array
    pixels = []
    for i in range(height):
        row = []
        for j in range(width):
            row.append(int.from_bytes(f.read(1), byteorder='big'))
        pixels.append(row)
# Flatten the pixel values into a 1D array
pixels_flat = np.array(pixels).flatten()
```

```
# Plot the histogram
plt.hist(pixels_flat, bins=range(max_val+2))
plt.xlabel('Pixel value')
plt.ylabel('Frequency')
plt.show()
```

Algebriac Operation

```
import io

def main():
   with open("SanFranPeak_red.pgm", "rb") as stream_in1, \
       open("SanFranPeak_green.pgm", "rb") as stream_in2, \
       open("SanFranPeak_blue.pgm", "rb") as stream_in3, \
       open("2grb.pgm", "wb") as stream_out1, \
       open("rb.pgm", "wb") as stream_out2, \
       open("rgb_3.pgm", "wb") as stream_out3:

        next_line(stream_in1)
        col = int(next_line(stream_in1))
        row = int(next_line(stream_in1))
        next_line(stream_in1)

        for _ in range(4):
           next_line(stream_in2)
           next_line(stream_in3)

        imageR = [[0 for _ in range(col)] for _ in range(row)]
        imageG = [[0 for _ in range(col)] for _ in range(row)]
        imageB = [[0 for _ in range(col)] for _ in range(row)]
        image1 = [[0 for _ in range(col)] for _ in range(row)]
        image2 = [[0 for _ in range(col)] for _ in range(row)]
        image3 = [[0 for _ in range(col)] for _ in range(row)]

        for i in range(row):
           for j in range(col):
              p1 = stream_in1.read(1)[0]
              imageR[i][j] = p1
              p2 = stream_in2.read(1)[0]
              imageG[i][j] = p2
              p3 = stream_in3.read(1)[0]
              imageB[i][j] = p3
```

```
                equl1 = 2 * p2 - p1 - p3
                equl1 = min(max(equl1, 0), 255)
                image1[i][j] = equl1

                equl2 = p1 - p3
                equl2 = min(max(equl2, 0), 255)
                image2[i][j] = equl2

                equl3 = (p1 + p2 + p3) // 3
                equl3 = min(max(equl3, 0), 255)
                image3[i][j] = equl3

        write(image1, stream_out1)
        write(image2, stream_out2)
        write(image3, stream_out3)


def next_line(stream):
    line_bytes = bytearray()
    while True:
        b = stream.read(1)[0]
        if b == ord('#'):
            while b not in (ord('\n'), ord('\r'), -1):
                b = stream.read(1)[0]
        elif not isspace(b):
            line_bytes.append(b)
        elif line_bytes:
            break
    return line_bytes.decode('ascii')


def write(image, stream):
    stream.write(b'P5\n')
    stream.write(str(image[0].__len__()).encode('ascii'))
    stream.write(b' ')
    stream.write(str(image.__len__()).encode('ascii'))
    stream.write(b'\n255\n')

    for row in image:
        stream.write(bytes(row))

    stream.close()



def isspace(b):
    return chr(b).isspace()



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

Geometric Operation

```
import numpy as np

# Creating array for grid and disgrid
grid = [[[0 , 0],[16, 0],[32,0],[48, 0],[64, 0],[80,0],[96, 0],[112, 0],[128, 0],[144, 0],[160, 0],[176, 0],[192, 0],[208, 0],[224, 0],[240, 0],[256, 0]],
        [[0 , 16],[16, 16],[32,16],[48, 16],[64, 16],[80,16],[96, 16],[112, 16],[128, 16],[144, 16],[160, 16],[176, 16],[192, 16],[208, 16],[224,
16],[240, 16],[256, 16]],
        [[0 , 32],[16, 32],[32,32],[48, 32],[64, 32],[80,32],[96, 32],[112, 32],[128, 32],[144, 32],[160, 32],[176, 32],[192, 32],[208, 32],[224,
32],[240, 32],[256, 32]],
        [[0 , 48],[16, 48],[32,48],[48, 48],[64, 48],[80,48],[96, 48],[112, 48],[128, 48],[144, 48],[160, 48],[176, 48],[192, 48],[208, 48],[224,
48],[240, 48],[256, 48]],
        [[0 , 64],[16, 64],[32,64],[48, 64],[64, 64],[80,64],[96, 64],[112, 64],[128, 64],[144, 64],[160, 64],[176, 64],[192, 64],[208, 64],[224,
64],[240, 64],[256, 64]],
        [[0 , 80],[16, 80],[32,80],[48, 80],[64, 80],[80,80],[96, 80],[112, 80],[128, 80],[144, 80],[160, 80],[176, 80],[192, 80],[208, 80],[224,
80],[240, 80],[256, 80]],
        [[0 , 96],[16, 96],[32,96],[48, 96],[64, 96],[80,96],[96, 96],[112, 96],[128, 96],[144, 96],[160, 96],[176, 96],[192, 96],[208, 96],[224,
96],[240, 96],[256, 96]],
        [[0 , 112],[16, 112],[32,112],[48, 112],[64, 112],[80,112],[96, 112],[112, 112],[128, 112],[144, 112],[160, 112],[176, 112],[192,
112],[208, 112],[224, 112],[240, 112],[256, 112]],
        [[0 , 128],[16, 128],[32,128],[48, 128],[64, 128],[80,128],[96, 128],[112, 128],[128, 128],[144, 128],[160, 128],[176, 128],[192,
128],[208, 128],[224, 128],[240, 128],[256, 128]],
        [[0 , 144],[16, 144],[32,144],[48, 144],[64, 144],[80,144],[96, 144],[112, 144],[128, 144],[144, 144],[160, 144],[176, 144],[192,
144],[208, 144],[224, 144],[240, 144],[256, 144]],
        [[0 , 160],[16, 160],[32,160],[48, 160],[64, 160],[80,160],[96, 160],[112, 160],[128, 160],[144, 160],[160, 160],[176, 160],[192,
160],[208, 160],[224, 160],[240, 160],[256, 160]],
        [[0 , 176],[16, 176],[32,176],[48, 176],[64, 176],[80,176],[96, 176],[112, 176],[128, 176],[144, 176],[160, 176],[176, 176],[192,
176],[208, 176],[224, 176],[240, 176],[256, 176]],
        [[0 , 192],[16, 192],[32,192],[48, 192],[64, 192],[80,192],[96, 192],[112, 192],[128, 192],[144, 192],[160, 192],[176, 192],[192,
192],[208, 192],[224, 192],[240, 192],[256, 192]],
        [[0 , 208],[16, 208],[32,208],[48, 208],[64, 208],[80,208],[96, 208],[112, 208],[128, 208],[144, 208],[160, 208],[176, 208],[192,
208],[208, 208],[224, 208],[240, 208],[256, 208]],
        [[0 , 224],[16, 224],[32,224],[48, 224],[64, 224],[80,224],[96, 224],[112, 224],[128, 224],[144, 224],[160, 224],[176, 224],[192,
224],[208, 224],[224, 224],[240, 224],[256, 224]],
        [[0 , 240],[16, 240],[32,240],[48, 240],[64, 240],[80,240],[96, 240],[112, 240],[128, 240],[144, 240],[160, 240],[176, 240],[192,
240],[208, 240],[224, 240],[240, 240],[256, 240]],
        [[0 , 256],[16, 256],[32,256],[48, 256],[64, 256],[80,256],[96, 256],[112, 256],[128, 256],[144, 256],[160, 256],[176, 256],[192,
256],[208, 256],[224, 256],[240, 256],[256, 256]]]

disgrid = [[[0, 0],[16, 0],[32, 0],[48, 0],[64, 0],[80, 0],[96, 0],[112, 0],[128, 0],[144, 0],[160, 0],[176, 0],[192, 0],[208, 0],[224, 0],[240, 0],[256, 0]],
        [[0, 16],[16, 16],[32, 16],[48, 16],[64, 16],[79, 16],[97, 17],[114, 19],[130, 18],[146, 19],[160, 18],[176, 17],[192, 16],[208, 16],[224,
16],[240, 16],[256, 16]],
        [[0, 32],[16, 32],[33, 32],[48, 32],[67, 31],[85, 35],[103, 37],[121, 40],[136, 42],[150, 43],[162, 41],[177, 37],[192, 35],[208, 32],[224,
32],[240, 31],[256, 32]],
        [[0, 48],[16, 48],[32, 48],[51, 49],[72, 49],[94, 53],[112, 56],[128, 60],[141, 63],[154, 65],[166, 65],[178, 62],[192, 57],[206, 52],[224,
48],[240, 48],[256, 48]],
        [[0, 64],[16, 64],[34, 64],[56, 63],[80, 66],[99, 68],[116, 72],[132, 76],[144, 80],[156, 84],[167, 85],[177, 83],[190, 80],[204, 74],[222,
66],[240, 64],[256, 64]],
        [[0, 80],[16, 80],[37, 78],[63, 78],[84, 78],[103, 81],[119, 85],[132, 89],[144, 94],[154, 100],[165, 103],[176, 102],[188, 100],[203,
94],[221, 85],[240, 80],[256, 80]],
        [[0, 96],[16, 96],[41, 93],[65, 91],[86, 90],[102, 90],[118, 96],[130, 102],[141, 108],[152, 116],[161, 117],[172, 119],[184, 116],[200,
112],[217, 105],[237, 97],[256, 96]],
        [[0, 112],[18, 110],[42, 106],[65, 103],[84, 101],[100, 102],[114, 105],[127, 112],[136, 119],[145, 126],[154, 130],[167, 132],[180,
132],[196, 128],[215, 122],[237, 115],[256, 112]],
        [[0, 128],[19, 126],[41, 120],[64, 113],[81, 112],[96, 112],[109, 115],[121, 120],[129, 128],[137, 135],[148, 141],[161, 143],[174,
144],[193, 142],[213, 137],[236, 131],[256, 128]],
        [[0, 144],[18, 141],[40, 135],[60, 129],[76, 125],[90, 124],[101, 125],[113, 129],[121, 136],[131, 144],[142, 150],[156, 154],[172,
154],[190, 154],[212, 149],[236, 145],[256, 144]],
        [[0, 160],[17, 160],[38, 151],[57, 144],[72, 140],[85, 138],[96, 138],[106, 141],[115, 148],[126, 153],[138, 161],[153, 165],[169,
167],[190, 167],[214, 163],[238, 161],[256, 160]],
        [[0, 176],[16, 177],[34, 170],[53, 162],[66, 156],[81, 153],[92, 153],[102, 156],[112, 158],[124, 165],[137, 171],[153, 174],[171,
178],[192, 178],[217, 177],[240, 176],[256, 176]],
        [[0, 192],[17, 192],[33, 191],[51, 182],[66, 175],[78, 170],[90, 169],[101, 172],[113, 176],[124, 181],[139, 184],[155, 188],[174,
189],[198, 193],[221, 192],[240, 192],[256, 192]],
        [[0, 208],[16, 208],[31, 208],[49, 204],[64, 197],[80, 193],[89, 190],[101, 190],[113, 191],[128, 195],[144, 198],[161, 203],[182,
205],[204, 206],[224, 208],[240, 208],[256, 208]],
        [[0, 224],[16, 224],[32, 224],[48, 223],[63, 221],[80, 217],[92, 213],[106, 212],[119, 212],[133, 215],[150, 217],[168, 220],[189,
222],[208, 224],[223, 224],[241, 224],[256, 224]],
        [[0, 240],[16, 240],[32, 240],[48, 240],[64, 240],[80, 239],[95, 238],[110, 237],[125, 236],[142, 237],[158, 238],[175, 239],[192,
240],[208, 240],[224, 240],[240, 240],[256, 240]],
        [[0, 256],[16, 256],[32, 256],[48, 256],[64, 256],[80, 256],[96, 256],[112, 256],[128, 256],[144, 256],[160, 256],[176, 256],[192,
256],[208, 256],[224, 256],[240, 256],[256, 256]]]

# read .pgm file
```

```python
def reader(stream):
    byte_list = []
    while True:
        b = stream.read(1)

        if b != b'':
            c = chr(b[0])
            if c == '#':
                d = b''
                while d != b'\n' and d != b'\r' and d != b'':
                    d = stream.read(1)
            elif not c.isspace():
                byte_list.append(b[0])
            elif len(byte_list) > 0:
                break
        else:
            break

    return bytes(byte_list).decode('utf-8')


# implemented controlgrid
def controlGrid(disgridXY_new, gridXY, pic, row, col):
    x_new = np.zeros(4, dtype=int)
    y_new = np.zeros(4, dtype=int)
    x = np.zeros(4)
    y = np.zeros(4)
    w = np.zeros(8)
    xy = np.zeros((4, 4))
    inverse = np.zeros((4, 4))
    image_new = np.zeros((row, col), dtype=int)

    gridXY = np.array(gridXY)

    for i in range(gridXY.shape[0]-1):
        for j in range(gridXY.shape[1]-1):
            x_new[0] = gridXY[i,j,0]
            x_new[1] = gridXY[i,j+1,0]
            x_new[2] = gridXY[i+1,j,0]
            x_new[3] = gridXY[i+1,j+1,0]

            y_new[0] = gridXY[i,j,1]
            y_new[1] = gridXY[i,j+1,1]
            y_new[2] = gridXY[i+1,j,1]
            y_new[3] = gridXY[i+1,j+1,1]

            for num in range(4):
                xy[num,0] = x_new[num]
                xy[num,1] = y_new[num]
                xy[num,2] = x_new[num] * y_new[num]
                xy[num,3] = 1

            inverse = np.linalg.inv(xy)

            x[0] = disgridXY_new[i][j][0]
            x[1] = disgridXY_new[i][j+1][0]
            x[2] = disgridXY_new[i+1][j][0]
            x[3] = disgridXY_new[i+1][j+1][0]

            y[0] = disgridXY_new[i][j][1]
            y[1] = disgridXY_new[i][j+1][1]
            y[2] = disgridXY_new[i+1][j][1]
            y[3] = disgridXY_new[i+1][j+1][1]

            w[0] = (inverse[0][0] * x[0]) + (inverse[0][1] * x[1]) + (inverse[0][2] * x[2]) + (inverse[0][3] * x[3])
            w[1] = (inverse[1][0] * x[0]) + (inverse[1][1] * x[1]) + (inverse[1][2] * x[2]) + (inverse[1][3] * x[3])
            w[2] = (inverse[2][0] * x[0]) + (inverse[2][1] * x[1]) + (inverse[2][2] * x[2]) + (inverse[2][3] * x[3])
            w[3] = (inverse[3][0] * x[0]) + (inverse[3][1] * x[1]) + (inverse[3][2] * x[2]) + (inverse[3][3] * x[3])
            w[4] = (inverse[0][0] * y[0]) + (inverse[0][1] * y[1]) + (inverse[0][2] * y[2]) + (inverse[0][3] * y[3])
            w[5] = (inverse[1][0] * y[0]) + (inverse[1][1] * y[1]) + (inverse[1][2] * y[2]) + (inverse[1][3] * y[3])
            w[6] = (inverse[2][0] * y[0]) + (inverse[2][1] * y[1]) + (inverse[2][2] * y[2]) + (inverse[2][3] * y[3])
            w[7] = (inverse[3][0] * y[0]) + (inverse[3][1] * y[1]) + (inverse[3][2] * y[2]) + (inverse[3][3] * y[3])

            for k in range(y_new[0], y_new[2]):
                for l in range(x_new[0], x_new[1]):
                    xp = int(round(w[0]*l + w[1]*k + w[2]*l*k + w[3]))
                    yp = int(round(w[4]*l + w[5]*k + w[6]*l*k + w[7]))
                    image_new[k][l] = pic[yp][xp]
    return image_new


def invert(a):
    n = len(a)
    x = [[0 for _ in range(n)] for _ in range(n)]
    b = [[0 for _ in range(n)] for _ in range(n)]
    index = [i for i in range(n)]
    for i in range(n):
        b[i][i] = 1
    gaussian(a, index)

    for i in range(n - 1):
        for j in range(i + 1, n):
            for k in range(n):
                b[index[j]][k] -= a[index[j]][i] * b[index[i]][k]

    for i in range(n):
        x[n - 1][i] = b[index[n - 1]][i] / a[index[n - 1]][n - 1]
        for j in range(n - 2, -1, -1):
            x[j][i] = b[index[j]][i]
            for k in range(j + 1, n):
                x[j][i] -= a[index[j]][k] * x[k][i]
            x[j][i] /= a[index[j]][j]
    return x

def gaussian(a, index):
    n = len(index)
    c = [0] * n

    for i in range(n):
        c1 = 0
        for j in range(n):
            c0 = abs(a[i][j])
            if c0 > c1:
                c1 = c0
        c[i] = c1

    k = 0
    for j in range(n - 1):
        pi1 = 0
        for i in range(j, n):
            pi0 = abs(a[index[i]][j])
            pi0 /= c[index[i]]
            if pi0 > pi1:
                pi1 = pi0
                k = i
        index[j], index[k] = index[k], index[j]
        for i in range(j + 1, n):
            pj = a[index[i]][j] / a[index[j]][j]
            a[index[i]][j] = pj
            for l in range(j + 1, n):
                a[index[i]][l] -= pj * a[index[j]][l]
```

```
def main():
    with open("distlenna.pgm", "rb") as stream_in1, open("lenna.pgm", "wb") as stream_out:
        reader(stream_in1)
        row = int(reader(stream_in1))
        col = int(reader(stream_in1))
        reader(stream_in1)

        image = [[0 for _ in range(col)] for _ in range(row)]
        image_new = [[0 for _ in range(col)] for _ in range(row)]

        for i in range(row):
            for j in range(col):
                p1 = stream_in1.read(1)[0]
                image[i][j] = p1

        image_new = controlGrid(disgrid, grid, image, row, col)

        stream_out.write(b"P5\n")
        stream_out.write(str(len(image[0])).encode())
        stream_out.write(b" ")
        stream_out.write(str(len(image)).encode())
        stream_out.write(b"\n")
        stream_out.write(b"255\n")
        for i in range(row):
            for j in range(col):
                p = image_new[i][j]
                stream_out.write(bytes([p]))

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