ศิวกร วงศ์พลกานันท์ 600610781

Histogram and Object Moment

1.Histogram and Object Moment

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

ทำการหาค่า histogram ของรูปทั้งรูปก่อน แล้วนำมาตัด threshold ค่าสีเทาที่มีความถี่มากกว่า 1500 เพื่อหาวัตถุที่มีอยู่ในรูป และตัดค่าสี 255 ที่เป็น Background ออก จะได้ dictionary ที่มีแต่ ค่าสี : ความถี่ ของแต่ละวัตถุซึ่งจะได้ออกมา 5 วัตถุ ที่มีค่าสีเทา 0,80,120,160,200

```
Histogram of image :
{0: 4969, 4: 13, 5: 7, 7: 7, 8: 7, 9: 7, 10: 13, 11: 5, 12: 33, 14: 21,
15: 11, 16: 54, 17: 1, 18: 7, 19: 4, 20: 10, 21: 6, 23: 1, 24: 3, 26: 6,
 27: 1, 32: 1, 33: 6, 37: 2, 40: 11, 44: 2, 48: 13, 56: 5, 62: 1, 65: 9,
 74: 8, 80: 4956, 81: 5, 83: 15, 84: 23, 85: 7, 86: 14, 87: 4, 88: 27, 8
9: 1, 90: 30, 91: 22, 92: 37, 93: 7, 94: 25, 95: 4, 96: 1, 97: 13, 98: 6
, 99: 1, 101: 5, 102: 2, 103: 5, 105: 7, 106: 2, 108: 10, 109: 1, 110: 6
, 113: 22, 115: 3, 116: 3, 119: 11, 120: 7529, 121: 3, 122: 23, 123: 11,
 124: 7, 125: 30, 126: 19, 127: 58, 128: 40, 129: 50, 130: 32, 131: 23,
133: 24, 134: 6, 135: 11, 136: 6, 137: 5, 138: 11, 140: 4, 141: 14, 143:
 10, 144: 1, 145: 2, 146: 6, 147: 5, 148: 6, 149: 12, 150: 8, 151: 9, 15
2: 1, 153: 4, 154: 1, 155: 9, 156: 8, 157: 2, 158: 5, 159: 20, 160: 3460
, 161: 9, 162: 13, 163: 21, 164: 23, 165: 47, 166: 44, 167: 20, 168: 20,
 169: 7, 170: 3, 171: 3, 172: 4, 173: 2, 174: 8, 175: 14, 176: 7, 177: 3
, 178: 28, 179: 1, 180: 4, 181: 15, 183: 15, 184: 11, 185: 1, 186: 5, 18
7: 13, 188: 6, 189: 6, 190: 2, 191: 37, 192: 13, 193: 2, 194: 4, 195: 6,
 196: 3, 197: 9, 198: 4, 199: 32, 200: 4955, 201: 35, 202: 31, 203: 44,
204: 71, 205: 54, 206: 16, 207: 59, 208: 23, 210: 10, 211: 30, 212: 4, 2
13: 12, 214: 18, 215: 24, 216: 17, 217: 21, 218: 8, 219: 7, 220: 3, 221:
 22, 222: 19, 223: 16, 224: 10, 225: 17, 226: 2, 227: 9, 228: 19, 229: 7
, 230: 19, 231: 36, 232: 7, 233: 21, 234: 21, 235: 3, 236: 9, 237: 20, 2
38: 22, 239: 40, 240: 7, 241: 15, 242: 20, 243: 29, 244: 11, 245: 25, 24
6: 17, 247: 43, 248: 10, 249: 10, 250: 36, 251: 30, 252: 38, 253: 18, 25
5: 151196}
```

(a) ค่า histogram ของทั้งรูป

```
Number of Object : 5
Gray level of each object
{0: 4969, 80: 4956, 120: 7529, 160: 3460, 200: 4955}
```

(b) ค่า ระดับสีเทา : ความถี่ ของแต่ละวัตถุ

1.2 Write a procedure to compute the (central)moment of an object given its gray level and use this procedure to compute the central moments m_{20} and m_{02}. Using this value, compute f_1 and verify its invariance (proof that f_1 is constant)

จากสูตรที่ได้จากโจทย์ โดยจะใช้ dictionary ของ ทำการสร้าง list ของแต่ละวัตถุโดยการ mask วัตถุ โดยตำแหน่งไหนที่มีค่าสีเทาของวัตถุนั้น ๆ จะให้ mask ตำแหน่งนั้น ๆ เป็นค่า 1 และ ให้ตำแหน่งอื่น ๆ เป็น 0 จะได้ list ที่มีเพียง object เดียว จากนั้น นำไปผ่าน ฟังก์ชั่น ตามสูตรที่ได้จากโจทย์ เพื่อคำนวณหาค่า Central moment ของแต่ละ วัตถุ

(a) รูป scaled shapes ที่ได้จากโจทย์ มีค่าระดับสีเทา 0,120,160,80,200 เรียงจากซ้ายไปขวา บนลงล่าง เมื่อคำนวณหาค่า central moment แล้วจะได้ค่า ดังนี้

Central moment of an Object on gray level(0) :

m20 = 6345057.412758953, m02 =1100035.4952706748, Normalized moment = 0.30153111124470

Central moment of an Object on gray level(80) :

m20 = 4941000.965899928, m02 =2456890.0379338027, Normalized moment = 0.30119331814209

Central moment of an Object on gray level(120) :

m20 = 7875173.002258037, m02 =8460663.084340539, Normalized moment = 0.28818194805645

Central moment of an Object on gray level(160) :

m20 = 2194070.5838150145, m02 =975991.6254335531, Normalized moment = 0.26479854065025

Central moment of an Object on gray level(200) :

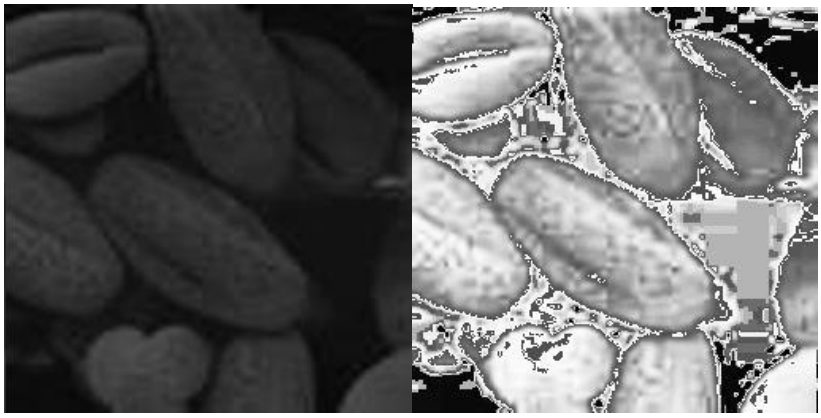
m20 = 3007131.8700302355, m02 =4792344.160242188, Normalized moment = 0.31767139493676

เมื่อนำค่า moment ที่ได้ มาเปรียบเทียบกันจะทำให้รู้ว่า ค่า central moment น้อย จะแสดงว่า รูปมีขนาดเล็ก และถ้าค่า central moment ทีค่ามาก จะแสดงให้เห็นว่า รูปว่ามีขนาดใหญ่ รวมถึง Normalized moment จะแสดงให้เห็นถึงอัตราส่วนของแต่ต่างระหว่าง coordinates ของ center of mass ของวัตถุ

2.Point Operations

ทำการ histogram equalization รูปภาพ Cameraman กับ SEM256_256 โดยตอนแรกให้ทำการหา histogram ของรูปภาพ จากนั้นนำ histogram ไปสร้างตารางสำหรับการทำ histogram equalization โดยการหา ค่า Dmax , PDF , CDF , จากนั้นนำมา round ค่า เพื่อ map กับรูป input จะได้รูป output ที่ถูก histogram equalization แล้ว

(a) (b)



(c) (d)

(a) รูป cameraman ก่อน histogram equalization (b) รูป cameraman หลัง histogram equalization
(c) รูป SEM256_256 ก่อน histogram equalization (d) รูป SEM256_256หลัง histogram equalization

จากรูป cameraman จะทำให้เห็นว่า ก่อนผ่าน equalization จุดมืดของภาพจะออกสีเทาๆละรูปส่วนใหญ่จะออกไปทางสว่างมากกว่าจึงทำให้รูปมี contrast ต่ำ แต่เมื่อผ่าน equalization แล้ว จะทำให้จุดมืด มีค่าสีเทาน้อยลง จึงทำให้เห็น detail ในส่วนของเสื้อคนมากขึ้นและให้เห็น background ที่เป็นตึกชัดขึ้นด้วยและจากรูป SEM_256 เมื่อผ่าน equalization แล้วจะทำให้ค่าสีเทากระจายไปในแต่ละพื้นที่ ทำให้จุดที่มืดมาก ๆเมื่อ equalization แล้วจะไปดันให้ส่วนสว่างของรูปสว่างขึ้นมาก ๆด้วยเช่นกัน

3. Algebraic Operations

สามารถทำการคำนวณค่าระดับสีเทาได้จากรูปภาพเลย โดยการรับภาพ input มาทั้ง 3 ภาพ และนำค่าของระดับสีเทาของแต่ละ pixel ไปคำนวณตามสูตรที่โจทย์กำหนด โดยถ้าคำนวณแล้วได้ค่าน้อยกว่า 0 ให้กำหนดค่านั้นเป็น 0 และ ถ้าหากค่าที่ได้ มากกว่า 255 ให้กำหนดค่านั้นเป็น 255

2G-R-B (excess green)

Red-blue difference

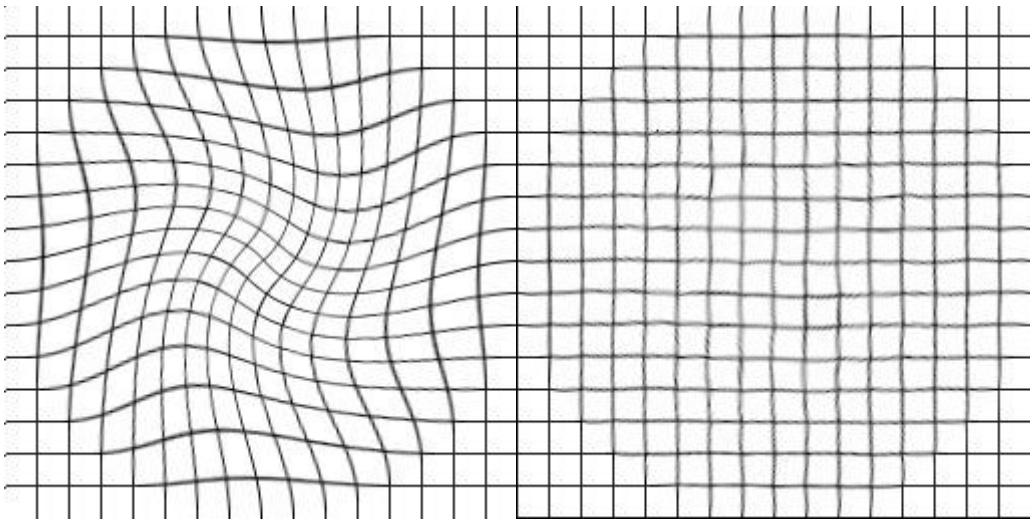
Intensity ((red + blue + green)/3)

จากรูป จะได้ว่าการทำ Algebraic Operation แบบ Intensity จะทำให้ได้รูปที่สมบูรณ์ที่สุดเมื่อเทียบกับ สองวิธีการที่เหลือ เพราะ เป็นการหาค่าเฉลี่ยของระดับสีเทาทั้ง 3 channel ของสี ส่วนสองวิธีที่เหลือเป็นการรวมค่าระดับสีเทาที่ทำให้สีๆใดสีนึง โดดขึ้นมา หรือ ทำให้หายไปเลย ทำให้ได้รูปที่ไม่สมบูรณ์แบบ

4. Geometric Operations

กำหนดค่ามุมแต่ละมุมของ grid และ กำหนดค่ามุมแต่ละมุมของ distgrid จากนั้นนำมาเทียบกัน หลังจากนั้นคำนวณหา w1-w8 เพื่อทำการ control grid interpolation จากนั้นนำค่า w มาย้อนกลับหาค่าพิกัดใหม่จากตำแหน่งของ distgrid แทน และนำไปทำการ bilinear interpolation สำหรับค่าพิกัดที่ไม่เป็นจำนวนเต็ม เพื่อทำให้เกิดรูปใหม่ที่ไม่บิดเบี้ยวละทำให้ขอบของรูปมีความชัดขึ้น

(a) (b)



(c) (d)

(a) รูป lenna ดั้งเดิม (b) รูป lenna ที่ผ่าน Geometric Operations

(c) dist_gird ดั้งเดิม (d) dist_grid ที่ผ่าน Geometric Operations

จากรูป จะทำให้เห็นว่า Geometric Operations จะสามารถทำให้รูปที่บิดเบี้ยวสามารถทำให้กลับมาตรงเหมือนเดิมได้ได้ และการทำ bilinear interpolation จะทำให้ขอบของรูปมีความชัดเจนขึ้น ถึงจะไม่ชัดเท่ารูปต้นแบบแต่ก็ชัดเจนในระดับที่ถือว่าใช้ได้

ภาคผนวก

Source Code : https://github.com/Thanatossan/image_processing_hw1.git

#อ่านไฟล์ pgm

```
def read_pgm(filename, col, row):
    f = open(filename, encoding="ISO-8859-1")
    img = ""
    list_img = []
    comment = False
    # matirx_img = []
    for i, line in enumerate(f):
        if i == 1 and line[0:1] == "#":
            comment = True
        elif i == 1 and line[0:1] != "#":
            col = int(line[0:3])
            row = int(line[4:])
        if i == 2 and comment == True:
            col = int(line[0:3])
            row = int(line[4:])
        if i >= 4 and comment == True:
            img = img+line
        elif i >= 3 and comment == False:
            img = img+line
    f.close()
    # print(len(img))
    for i in range(len(img)):
        list_img.append((ord(img[i])))
    list_img = fix_miss_pixel(list_img)
    return list_img, col, row
```

#แปลงจาก list 1 มิติ เป็น list 2 มิติ

```
def list_to_2D_list(lists, list_2D, col, row):
    for i in range(row):
        inner_list = []
        for j in range(col):
            inner_list.append(lists[i*(col)+j])
        list_2D.append(inner_list)
    return list_2D
```

#copy list 2 มิติ

```
def copy(lists):
    copy_list = []
    for i in range(len(lists)):
        inner_list = []
        for j in range(len(lists[i])):
            inner_list.append(lists[i][j])
        copy_list.append(inner_list)
    return copy_list
```

#สร้าง histogram

```
def createHistogram(converted_img):
    gray_level = []
    frequency = []
    histogram = {}
    for i in range(len(converted_img)):
        if converted_img[i] not in gray_level:
            gray_level.append(converted_img[i])
    gray_level.sort()
    for i in range(len(gray_level)):
        frequency.append(converted_img.count(gray_level[i]))
    histogram = dict(zip(gray_level, frequency))
    return histogram
```

#เขียน pgm file

```
# Must be done someday
def writepgm(filename, mattriximg, col, row):
    string = ""
    filename = "./output_img/"+filename
    for i in range(len(mattriximg)):
        for j in range(len(mattriximg[i])):
            mattriximg[i][j] = chr(int(mattriximg[i][j]))
            # print(mattriximg)
            string += mattriximg[i][j]
    f = open(filename, "a", encoding="ISO-8859-1")

    f.write("P5 \r")
    f.write(str(col) + " " + str(row) + "\r")
    f.write("255 \r")
    f.write(string)
    f.close
```

Histogram and Object Moment

```
rom readpgm import read_pgm, list_to_2D_list, copy
from momentFunction import pqmoment, pqN, pqHu
from etc_function import createHistogram
filename = "./image/1./scaled_shapes.pgm"
converted_img = []
mattrix_img = []
col = 0
row = 0

converted_img, col, row = read_pgm(filename, col, row) #อ่านไฟล์ pgm
mattrix_img = list_to_2D_list(converted_img, mattrix_img, col, row)
                                                        #ทำให้เป็น mattirx



object_dict = {}
count = 0

histogram = createHistogram(converted_img) #สร้าง histogram
# count object
for key in histogram:
    if histogram[key] > 1500 and key != 255: #กำหนด threshold และ ตัด backgroundอ ออก
        object_dict[key] = histogram.get(key)
        count = count+1

print("Histogram of image :")
print(histogram)
print("Number of Object : " + str(count))
print("Gray level of each object")
print(object_dict)

maskesOb_img = copy(mattrix_img)


for key in object_dict:   #mask วัตถุที่กำลังคำนวณอยู่ให้เป็น 1

    for i in range(row):
        for j in range(col):
            if mattrix_img[i][j] == key:
                maskesOb_img[i][j] = 1

            else:
                maskesOb_img[i][j] = 0
      #ฟังก์ชั่นคำนวณ central moment
    U20 = pqHu(2, 0, maskesOb_img, row, col)
    U02 = pqHu(0, 2, maskesOb_img, row, col)
    N20 = pqN(2, 0, maskesOb_img, row, col)
    N02 = pqN(0, 2, maskesOb_img, row, col)
```

```
    theata = N20 + N02

    print("Central moment of an Object on gray level(" +
          str(key) + ") : U20 = " + str(U20) + ", U02 =" + str(U02) + ", Theat
a = " + str(theata))
    maskesOb_img = copy(mattrix_img)
```

#ฟังก์ชั่นคำนวณ central moment

```
def pqmoment(p, q, img, row, col):
    m = 0
    for i in range(len(img)):
        for j in range(len(img[i])):
            if img[i][j] == 1:
                m += (pow(i, p) * pow(j, q))
            else:
                pass

    return m


def pqHu(p, q, img, row, col):
    u = 0
    m10 = pqmoment(1, 0, img, row, col)
    m00 = pqmoment(0, 0, img, row, col)
    m01 = pqmoment(0, 1, img, row, col)
    for i in range(len(img)):
        for j in range(len(img[i])):
            if img[i][j] == 1:
                u += (pow((i - (m10/m00)), p) * pow((j - (m01/m00)), q)) * 1
            else:
                pass
    return u


def pqN(p, q, img, row, col):
    Upq = pqHu(p, q, img, row, col)
    U00 = pow(pqHu(0, 0, img, row, col), (((p+q)/2)+1))

    n = Upq / U00
    return n
```

Point Operations

```
from readpgm import read_pgm, list_to_2D_list, copy
from writepgm import writepgm
from etc_function import createHistogram
filename = "./image/2./Cameraman.pgm"
# filename = "./image/2./SEM256_256.pgm"

converted_img = []
mattrix_img = []
col = 0
row = 0

converted_img, col, row = read_pgm(filename, col, row)
mattrix_img = list_to_2D_list(converted_img, mattrix_img, col, row)
histogram_old = createHistogram(converted_img)
current_D = []
miss_frequency = []
miss_update = {}
for i in range(256):  # find miss colour #หาค่าสีที่หายไป
    if i not in histogram_old:
        miss_frequency.append(i)
# complete with colore frequenct = 0 #เติมค่าสีที่หายไปด้วยความถี่ 0
miss_update = {i: 0 for i in miss_frequency}
for key in miss_update:
    histogram_old.update({key: miss_update[key]})

histogram_old = {key: histogram_old[key]                    #แปลงเป็น dictionary
                 for key in sorted(histogram_old.keys())}  # sorted to dict
histogram_db = histogram_old.copy()
area = col*row

for D in histogram_db: #ทำ Histogram equalization ตามปกติ
    histogram_db[D] = histogram_db[D] / area  # H(D) / area
Pcxy = 0
for D in histogram_db:
    Pcxy += histogram_db[D]
    histogram_db[D] = Pcxy  # convert to PDF
for D in histogram_db:
    histogram_db[D] = histogram_db[D]*255  # convert to CDF
for D in histogram_db:
    # histogram_db[D] = int(histogram_db[D] // 1)  # floor number
    temp = int(histogram_db[D])
    if temp - (histogram_db[D]) >= 0.5:
        histogram_db[D] = temp + 1
    else:
        histogram_db[D] = int(histogram_db[D])
# convert from {DA : DB} to {DB : DA}
histogram_db = dict((y, x) for x, y in histogram_db.items())
```

```
# current dict is {DB : DA }
test_array = []
test = 0
for i in range(len(mattrix_img)):  #map histogram ใหม่ กับ histogram เดิม
    for j in range(len(mattrix_img[i])):
        for key in histogram_db:
            if mattrix_img[i][j] == histogram_db[key]:
                mattrix_img[i][j] = key

# print(mattrix_img)
# referred https://stackoverflow.com/questions/29244286/how-to-flatten-a-2d-list-to-1d-without-using-numpy/29244327
new_histogram_list = [i for subarray in mattrix_img for i in subarray]
new_histogram = createHistogram(new_histogram_list)
print("Before " + str(histogram_old))
print("--------------------")
print(histogram_db)

print("-------------")
print("After " + str(new_histogram))
filename_new = "cameraman.pgm"
writepgm(filename_new, mattrix_img, col, row)
```

Algebraic Operations

```
from readpgm import read_pgm, list_to_2D_list, copy
from writepgm import writepgm


def checklimit(gray_level):     #เช็คว่าค่าสีที่คำนวณเกิน 255 หรือน้อยกว่า 0
    if gray_level > 255:
        gray_level = 255
    if gray_level < 0:
        gray_level = 0
    return gray_level


#excess green formula
def formula1(mattrix_img_blue, mattrix_img_red, mattrix_img_green, col, row):
    mattrix_img_new = copy(mattrix_img_blue)
    for i in range(row):
        for j in range(col):
            mattrix_img_new[i][j] = 2*mattrix_img_green[i][j] - \
                mattrix_img_red[i][j] - mattrix_img_blue[i][j]
            mattrix_img_new[i][j] = checklimit(mattrix_img_new[i][j])
    return mattrix_img_new


#red-blue difference
def formula2(mattrix_img_blue, mattrix_img_red, mattrix_img_green, col, row):
    mattrix_img_new = copy(mattrix_img_blue)
    for i in range(row):
        for j in range(col):
            mattrix_img_new[i][j] = mattrix_img_red[i][j] - \
                mattrix_img_blue[i][j]
            mattrix_img_new[i][j] = checklimit(mattrix_img_new[i][j])
    return mattrix_img_new


#intensity
def formula3(mattrix_img_blue, mattrix_img_red, mattrix_img_green, col, row):
    mattrix_img_new = copy(mattrix_img_blue)
    for i in range(row):
        for j in range(col):
            mattrix_img_new[i][j] = int((
                mattrix_img_green[i][j] + mattrix_img_red[i][j] + mattrix_img_
blue[i][j])/3)
            mattrix_img_new[i][j] = checklimit(mattrix_img_new[i][j])
    return mattrix_img_new
```

```
filename_blue = "./image/3./SanFranPeak_blue.pgm"
filename_red = "./image/3./SanFranPeak_red.pgm"
filename_green = "./image/3./SanFranPeak_green.pgm"
col = 0
row = 0
mattrix_img_red = []
mattrix_img_green = []
mattrix_img_blue = []
listimg_red, col, row = read_pgm(filename_red, col, row)
listimg_green, col, row = read_pgm(filename_blue, col, row)
listimg_blue, col, row = read_pgm(filename_green, col, row)

mattrix_img_red = list_to_2D_list(listimg_red, mattrix_img_red, col, row)
mattrix_img_blue = list_to_2D_list(listimg_blue, mattrix_img_blue, col, row)
mattrix_img_green = list_to_2D_list(listimg_green, mattrix_img_green, col, row
)

exceedgreen = formula1(mattrix_img_blue, mattrix_img_red,
                       mattrix_img_green, col, row)
red_blue_diff = formula2(
    mattrix_img_blue, mattrix_img_red, mattrix_img_green, col, row)

intensity = formula3(mattrix_img_blue, mattrix_img_red,
                     mattrix_img_green, col, row)


exceedgreen_filename = "exceedgreen.pgm"
red_blue_diff_filename = "red-blue-differentce.pgm"
intensity_filename = "intensity.pgm"

writepgm(exceedgreen_filename, exceedgreen, col, row)
writepgm(red_blue_diff_filename, red_blue_diff, col, row)
writepgm(intensity_filename, intensity, col, row)
```

Geometric Operations

```
from readpgm import read_pgm, list_to_2D_list, copy
from etc_function import solve4eqaultion, Bilinear
from dist_list import disgrid, grid
from writepgm import writepgm
filename = "./image/4./distlenna.pgm"
# filename = "./image/4./distgrid.pgm"

col = 0
row = 0
mattrix_img = []
listimg, col, row = read_pgm(filename, col, row)
mattrix_img = list_to_2D_list(listimg, mattrix_img, col, row)

grid = grid() #กำหนดค่ามุมของ grid
dist = disgrid() #กำหนดค่ามุมของ disgrid

x = [0]*4
y = [0]*4
w1_to_4 = [0]*4
w5_to_8 = [0]*4
w = [0]*8
x_dist = [0]*4
y_dist = [0]*4
xy = []
Am = []
x_vertor_dist = [0]*4
for i in range(4):
    xy_in_loop = []
    Am_in_loop = []
    for j in range(4):
        xy_in_loop.append(0)
        Am_in_loop.append(0)
    xy.append(xy_in_loop)
    Am.append(Am_in_loop)
image_new = []
for i in range(row):
    image_new_in_loop = []
    for j in range(col):
        image_new_in_loop.append(0)
    image_new.append(image_new_in_loop)
#กำหนดค่า x1-x4 และ y1-y4
for i in range(len(grid)-1):
    for j in range(len(grid)-1):
        x[0] = grid[i][j][0]
        x[1] = grid[i][j+1][0]
        x[2] = grid[i+1][j][0]
        x[3] = grid[i+1][j+1][0]
```

```
        y[0] = grid[i][j][1]
        y[1] = grid[i][j+1][1]
        y[2] = grid[i+1][j][1]
        y[3] = grid[i+1][j+1][1]
        for k in range(4):
            xy[k][0] = x[k]
            xy[k][1] = y[k]
            xy[k][2] = x[k]*y[k]
            xy[k][3] = 1
#กำหนดค่า x'1 - x'4 และ  y'1 - y'4
        x_dist[0] = dist[i][j][0]
        x_dist[1] = dist[i][j+1][0]
        x_dist[2] = dist[i+1][j][0]
        x_dist[3] = dist[i+1][j+1][0]

        y_dist[0] = dist[i][j][1]
        y_dist[1] = dist[i][j+1][1]
        y_dist[2] = dist[i+1][j][1]
        y_dist[3] = dist[i+1][j+1][1]
        xy_new1 = copy(xy)
        xy_new2 = copy(xy)
# แก้สมการ 4  ตัวแปร หาค่า w  โดยวิธีการ guassian elimination
        w1_to_4 = solve4eqaultion(xy_new1, x_dist)
        w5_to_8 = solve4eqaultion(xy_new2, y_dist)
        for k in range(len(w1_to_4)):
            w[k] = w1_to_4[k]
        for k in range(len(w5_to_8)):
            w[k+4] = w5_to_8[k]
#หาค่าพิกัดใหม่ด้วยวิธีการย้อนกลับ
        for k in range(y[0], y[2]):
            for l in range(x[0], x[1]):
                xp = (w[0]*l + w[1]*k + w[2]*l*k + w[3])
                yp = (w[4]*l + w[5]*k + w[6]*l*k + w[7])
                image_new[k][l] = Bilinear(mattrix_img, yp, xp)
                                                #นำมาผ่าน Billinear interpolation
writepgm("new_lenna.pgm", image_new, col, row)
```

#มุมของ grid และ dist_grid

```
def disgrid():
    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], [255, 0]],
               [[0, 15], [16, 15], [32, 15], [48, 15], [64, 15], [80, 15], [97, 16], [114, 17], [130, 18], [
                   145, 18], [161, 17], [176, 15], [192, 15], [208, 15], [224, 15], [240, 15], [255, 15]],
               [[0, 31], [16, 31], [32, 31], [48, 31], [66, 32], [85, 34], [103, 36], [121, 39], [136, 41], [
                   150, 42], [163, 40], [177, 37], [192, 33], [208, 31], [224, 31], [240, 31], [255, 31]],
               [[0, 47], [16, 47], [32, 47], [51, 48], [72, 49], [93, 52], [112, 56], [128, 59], [141, 62], [
                   154, 64], [166, 64], [178, 62], [192, 56], [207, 50], [224, 47], [240, 47], [255, 47]],
               [[0, 63], [16, 63], [34, 63], [56, 63], [79, 64], [99, 67], [117, 71], [131, 75], [144, 79], [
                   156, 83], [167, 84], [178, 83], [191, 79], [206, 72], [223, 65], [240, 63], [255, 63]],
               [[0, 79], [16, 79], [38, 78], [62, 77], [84, 77], [103, 80], [118, 83], [132, 89], [144, 94], [
                   155, 99], [165, 102], [176, 102], [188, 100], [203, 93], [221, 85], [240, 79], [255, 79]],
               [[0, 95], [18, 95], [41, 92], [65, 90], [86, 89], [103, 91], [118, 94], [131, 101], [141, 107], [
                   151, 113], [161, 116], [172, 117], [184, 117], [200, 111], [218, 104], [238, 97], [255, 95]],
               [[0, 111], [19, 110], [43, 106], [65, 102], [84, 100], [101, 101], [115, 105], [127, 111], [136, 118], [
                   145, 125], [155, 130], [167, 131], [180, 131], [196, 127], [215, 121], [237, 113], [255, 111]],
               [[0, 127], [19, 125], [42, 119], [63, 114], [81, 111], [96, 111], [109, 114], [121, 119], [129, 127], [
                   138, 135], [149, 140], [161, 143], [176, 143], [193, 140], [213, 136], [236, 129], [255, 127]],
               [[0, 143], [18, 141], [40, 135], [60, 128], [77, 124], [91, 122], [103, 125], [113, 129], [121, 135], [
                   131, 143], [142, 149], [156, 153], [172, 155], [190, 153], [212, 149], [236, 145], [255, 143]],
               [[0, 159], [17, 158], [38, 151], [57, 143], [72, 139], [85, 137], [96, 137], [106, 141], [115, 146], [
                   126, 153], [138, 159], [153, 164], [170, 166], [190, 165], [214, 162], [238, 160], [255, 159]],
               [[0, 175], [16, 175], [35, 170], [54, 161], [69, 155], [81, 152], [93, 152], [102, 154], [113, 159], [
                   124, 164], [137, 170], [153, 174], [172, 177], [192, 177], [217, 176], [239, 175], [255, 175]],
               [[0, 191], [16, 191], [33, 189], [51, 181], [66, 175], [79, 170], [90, 169], [101, 170], [112, 174], [
                   125, 178], [139, 183], [156, 187], [175, 189], [198, 191], [221, 191], [240, 191], [255, 191]],
               [[0, 207], [16, 207], [32, 207], [49, 204], [65, 197], [78, 192], [90, 189], [102, 189], [114, 191], [
                   127, 194], [143, 198], [161, 201], [182, 204], [204, 206], [224, 207], [240, 207], [255, 207]],
               [[0, 223], [16, 223], [32, 223], [48, 223], [64, 220], [79, 216], [93, 213], [106, 211], [119, 212], [
                   134, 214], [151, 217], [169, 219], [190, 222], [208, 223], [224, 223], [240, 223], [255, 223]],
               [[0, 239], [16, 239], [32, 239], [48, 239], [64, 239], [80, 239], [95, 237], [110, 236], [126, 235], [
                   142, 236], [158, 237], [175, 238], [192, 239], [208, 239], [224, 239], [240, 239], [255, 239]],
               [[0, 255], [16, 255], [32, 255], [48, 255], [64, 255], [80, 255], [96, 255], [112, 255], [128, 255], [
                   144, 255], [160, 255], [176, 255], [192, 255], [208, 255], [224, 255], [240, 255], [255, 255]]
               ]
    return disgrid
```

```
def grid():
    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],
            [[0, 15], [16, 15], [32, 15], [48, 15], [64, 15], [80, 15], [96, 15], [112, 15], [128, 15], [
                144, 15], [160, 15], [176, 15], [192, 15], [208, 15], [224, 15], [240, 15], [255, 15]],
            [[0, 31], [16, 31], [32, 31], [48, 31], [64, 31], [80, 31], [96, 31], [112, 31], [128, 31], [
                144, 31], [160, 31], [176, 31], [192, 31], [208, 31], [224, 31], [240, 31], [255, 31]],
            [[0, 47], [16, 47], [32, 47], [48, 47], [64, 47], [80, 47], [96, 47], [112, 47], [128, 47], [
                144, 47], [160, 47], [176, 47], [192, 47], [208, 47], [224, 47], [240, 47], [255, 47]],
            [[0, 63], [16, 63], [32, 63], [48, 63], [64, 63], [80, 63], [96, 63], [112, 63], [128, 63], [
                144, 63], [160, 63], [176, 63], [192, 63], [208, 63], [224, 63], [240, 63], [255, 63]],
            [[0, 79], [16, 79], [32, 79], [48, 79], [64, 79], [80, 79], [96, 79], [112, 79], [128, 79], [
                144, 79], [160, 79], [176, 79], [192, 79], [208, 79], [224, 79], [240, 79], [255, 79]],
            [[0, 95], [16, 95], [32, 95], [48, 95], [64, 95], [80, 95], [96, 95], [112, 95], [128, 95], [
                144, 95], [160, 95], [176, 95], [192, 95], [208, 95], [224, 95], [240, 95], [255, 95]],
            [[0, 111], [16, 111], [32, 111], [48, 111], [64, 111], [80, 111], [96, 111], [112, 111], [128, 111], [
                144, 111], [160, 111], [176, 111], [192, 111], [208, 111], [224, 111], [240, 111], [255, 111]],
            [[0, 127], [16, 127], [32, 127], [48, 127], [64, 127], [80, 127], [96, 127], [112, 127], [128, 127], [
                144, 127], [160, 127], [176, 127], [192, 127], [208, 127], [224, 127], [240, 127], [255, 127]],
            [[0, 143], [16, 143], [32, 143], [48, 143], [64, 143], [80, 143], [96, 143], [112, 143], [128, 143], [
                144, 143], [160, 143], [176, 143], [192, 143], [208, 143], [224, 143], [240, 143], [255, 143]],
            [[0, 159], [16, 159], [32, 159], [48, 159], [64, 159], [80, 159], [96, 159], [112, 159], [128, 159], [
                144, 159], [160, 159], [176, 159], [192, 159], [208, 159], [224, 159], [240, 159], [255, 159]],
            [[0, 175], [16, 175], [32, 175], [48, 175], [64, 175], [80, 175], [96, 175], [112, 175], [128, 175], [
                144, 175], [160, 175], [176, 175], [192, 175], [208, 175], [224, 175], [240, 175], [255, 175]],
            [[0, 191], [16, 191], [32, 191], [48, 191], [64, 191], [80, 191], [96, 191], [112, 191], [128, 191], [
                144, 191], [160, 191], [176, 191], [192, 191], [208, 191], [224, 191], [240, 191], [255, 191]],
            [[0, 207], [16, 207], [32, 207], [48, 207], [64, 207], [80, 207], [96, 207], [112, 207], [128, 207], [
                144, 207], [160, 207], [176, 207], [192, 207], [208, 207], [224, 207], [240, 207], [255, 207]],
            [[0, 223], [16, 223], [32, 223], [48, 223], [64, 223], [80, 223], [96, 223], [112, 223], [128, 223], [
                144, 223], [160, 223], [176, 223], [192, 223], [208, 223], [224, 223], [240, 223], [255, 223]],
            [[0, 239], [16, 239], [32, 239], [48, 239], [64, 239], [80, 239], [96, 239], [112, 239], [128, 239], [
                144, 239], [160, 239], [176, 239], [192, 239], [208, 239], [224, 239], [240, 239], [255, 239]],
            [[0, 255], [16, 255], [32, 255], [48, 255], [64, 255], [80, 255], [96, 255], [112, 255], [128, 255], [
                144, 255], [160, 255], [176, 255], [192, 255], [208, 255], [224, 255], [240, 255], [255, 255]],
            ]
    return grid
```

#Gaussian elimination แก้สมการ 4 ตัวแปร

```
def solve4eqaultion(A, b):
    n = len(A)
    M = A

    i = 0
    for x in M:
        x.append(b[i])
        i += 1

    for k in range(n):
        for i in range(k, n):
            if abs(M[i][k]) > abs(M[k][k]):
                M[k], M[i] = M[i], M[k]
            else:
                pass

        for j in range(k+1, n):
            q = float(M[j][k]) / M[k][k]
            for m in range(k, n+1):
                M[j][m] -= q * M[k][m]

    x = [0 for i in range(n)]

    x[n-1] = float(M[n-1][n])/M[n-1][n-1]
    for i in range(n-1, -1, -1):
        z = 0
        for j in range(i+1, n):
            z = z + float(M[i][j])*x[j]
        x[i] = float(M[i][n] - z)/M[i][i]
    return x
```

#Billinear interpolation

```
def Bilinear(old_image, pixelXP, pixelYP):
    # point
    x = int(pixelXP//1)
    xplus = int((pixelXP+1)//1)

    y = int(pixelYP//1)
    yplus = int((pixelYP+1)//1)

    xScale = pixelXP - x
    yscale = pixelYP - y
    # read color from old image
    a = old_image[xplus][y] - old_image[x][y]
```

```
    b = old_image[x][yplus] - old_image[x][y]
    c = old_image[xplus][yplus] + old_image[x][y] - \
        old_image[x][yplus] - old_image[xplus][y]
    d = old_image[x][y]
    # caculate
    color = (a * xScale) + (b * yscale) + (c * xScale * yscale) + d

    color = int(round(color))
    return color
```