from decimal import Decimal

import matplotlib.pyplot as plt

import math

# This program accesses a single text file containing raw point cloud data from a single elevation

# It processes the information by sorting it and sampling 96 different points at increasing angles (in 3.75 degree intervals)

def quadsplit():

"""

z = pc.split()

points = []

new = []

count = 0

for num in z:

if count == 2:

new += [float(num)]

points += [new]

new = []

count += 1

elif count == 0:

new += [float(num)]

count += 1

elif count == 1:

new += [float(num)]

count += 1

elif count == 9:

count = 0

else:

count += 1

"""

points = retrieve()

center = center\_calc(points)

q1 = []

q2 = []

q3 = []

q4 = []

for p in points:

if p[0] >= center[0] and p[1] >= center[1]:

q1 += [p]

elif p[0] >= center[0] and p[1] <= center[1]:

q4 += [p]

elif p[0] <= center[0] and p[1] <= center[1]:

q3 += [p]

else:

q2 += [p]

return q1, q2, q3, q4

def numdeci(points):

check = False

exp = 1

for p in points:

check = len((p.split("."))[1])

if check > exp:

exp = check

return exp

def deccheck(points):

top = max(points)

dig = len(str(int(top)))

return dig

def pad(points, decim, deca):

for p in points:

dis = str(p[3])

dig = str(dis)

dig = dig.split(".")

dig[0] = ("0" \* (deca - len(dig[0]))) + dig[0]

dig[1] = dig[1] + ("0" \* (decim - len(dig[1])))

p[3] = dig[0] + "." + dig[1]

return points

def radix(points):

test = len(points[0][3]) - 1

for i in range(test + 1):

if points[0][3][test - i] != ".":

buck = [[], [], [], [], [], [], [], [], [], []]

for p in points:

buck[int(p[3][test - i])] += [p]

points = []

for b in buck:

for point in b:

points += [point]

return points

def q1pro(q1):

# q1 processing

x1 = []

dist = []

for p in q1:

x1 += [p[0]]

sq1 = min(x1)

for p in q1:

if p[0] - sq1 != 0:

p += [p[0] - sq1]

dist += [p[0] - sq1]

else:

p += [p[0] - sq1]

dist += [p[0] - sq1]

deca = deccheck(dist)

for i in range(len(dist)):

dist[i] = f'{dist[i]:.50f}'

q1[i][3] = f'{q1[i][3]:.50f}'

decim = numdeci(dist)

q1 = pad(q1, decim, deca)

q1 = radix(q1)

return q1

def q2pro(q2):

# q1 processing

x2 = []

dist = []

for p in q2:

x2 += [p[0]]

sq2 = min(x2)

for p in q2:

if p[0] - sq2 != 0:

p += [p[0] - sq2]

dist += [p[0] - sq2]

else:

p += [p[0] - sq2]

dist += [p[0] - sq2]

deca = deccheck(dist)

for i in range(len(dist)):

dist[i] = f'{dist[i]:.50f}'

q2[i][3] = f'{q2[i][3]:.50f}'

decim = numdeci(dist)

q2 = pad(q2, decim, deca)

q2 = radix(q2)

return q2

def q3pro(q3):

# q1 processing

x3 = []

dist = []

for p in q3:

x3 += [p[0]]

sq3 = max(x3)

for p in q3:

if p[0] - sq3 != 0:

p += [sq3 - p[0]]

dist += [sq3 - p[0]]

else:

p += [sq3 - p[0]]

dist += [sq3 - p[0]]

deca = deccheck(dist)

for i in range(len(dist)):

dist[i] = f'{dist[i]:.50f}'

q3[i][3] = f'{q3[i][3]:.50f}'

decim = numdeci(dist)

q3 = pad(q3, decim, deca)

q3 = radix(q3)

return q3

def q4pro(q3):

# q1 processing

x3 = []

dist = []

for p in q3:

x3 += [p[0]]

sq3 = max(x3)

for p in q3:

if p[0] - sq3 != 0:

p += [sq3 - p[0]]

dist += [sq3 - p[0]]

else:

p += [sq3 - p[0]]

dist += [sq3 - p[0]]

deca = deccheck(dist)

for i in range(len(dist)):

dist[i] = f'{dist[i]:.50f}'

q3[i][3] = f'{q3[i][3]:.50f}'

decim = numdeci(dist)

q3 = pad(q3, decim, deca)

q3 = radix(q3)

return q3

def ang\_calc(point, center):

if point[0] > center[0] and point[1] > center[1]:

angle = math.degrees(math.atan(math.fabs((point[0] - center[0]) / (point[1] - center[1]))))

elif point[0] > center[0] and point[1] < center[1]:

angle = math.degrees(math.atan(math.fabs((point[1] - center[1]) / (point[0] - center[0])))) + 90

elif point[0] < center[0] and point[1] < center[1]:

angle = math.degrees(math.atan(math.fabs((point[0] - center[0]) / (point[1] - center[1])))) + 180

elif point[0] < center[0] and point[1] > center[1]:

angle = math.degrees(math.atan(math.fabs((point[1] - center[1]) / (point[0] - center[0])))) + 270

elif point[0] == center[0] and point[1] > center[1]:

angle = 0

elif point[0] == center[0] and point[1] < center[1]:

angle = 180

elif point[1] == center[1] and point[0] > center[0]:

angle = 90

elif point[1] == center[1] and point[0] < center[0]:

angle = 270

return angle

def rearrange(points):

ypoints = []

for p in points:

ypoints += [p[1]]

# rearranging the data set with gate 24.5 at the start

cut = ypoints.index(max(ypoints))

out = points[cut:len(points)]

points = out + points[0:cut]

return points

def center\_calc(points):

xp = []

yp = []

for point in points:

xp += [point[0]]

yp += [point[1]]

center = [(max(xp) + min(xp)) / 2, (max(yp) + min(yp)) / 2]

return center

def point\_sample(points, center):

sample = []

track = 0

ang = [7.5, 11.25, 13.125, 15, 18.75, 22.5, 26.25, 30, 33.75, 37.5, 41.25, 45, 48.75, 52.5, 56.25, 60, 63.75, 67.5,

71.25, 75, 78.75, 82.5, 97.5, 112.5, 127.5, 142.5, 157.5, 172.5, 187.5, 202.5, 217.5, 232.5, 247.5, 262.5,

277.5, 292.5, 307.5, 322.5, 337.5, 352.5]

for targ\_ang in ang:

point = None

for z in range(track, len(points)):

check = targ\_ang - ang\_calc(points[z], center)

if check < .001 and check > -1:

diff = check

point = points[z]

track = z

break

sample += [point[0:3]]

return sample

def retrieve():

f = open("points56.txt", "r")

new = f.readlines()

points = []

for p in new:

x = []

z = p.replace("[", "").replace("]", "").replace(",", "").split()

x += [float(z[0])]

x += [float(z[1])]

x += [float(z[2])]

points += [x]

return points

q1, q2, q3, q4 = quadsplit()

q1 = q1pro(q1)

q2 = q2pro(q2)

q3 = q3pro(q3)

q4 = q4pro(q4)

points = q4 + q3 + q2 + q1

"""

for p in new:

print(p)

f = open("points52.txt", "w")

for p in new:

f.write("[" + str(p[0]) + ", " + str(p[1]) + ", " + str(p[2]) + "]" + "\n")

f.close()

"""

points = rearrange(points)

"""

xp = []

yp = []

for point in points:

xp += [point[0]]

yp += [point[1]]

top = yp.index(max(yp))

bottom = yp.index(min(yp))

right = xp.index(max(xp))

left = xp.index(min(xp))

"""

center = center\_calc(points)

sample = point\_sample(points, center)

print(sample)

x = []

y = []

for p in sample:

if p:

x += [p[0]]

y += [p[1]]

plt.scatter(x, y, s=1, alpha=0.5)

plt.show()