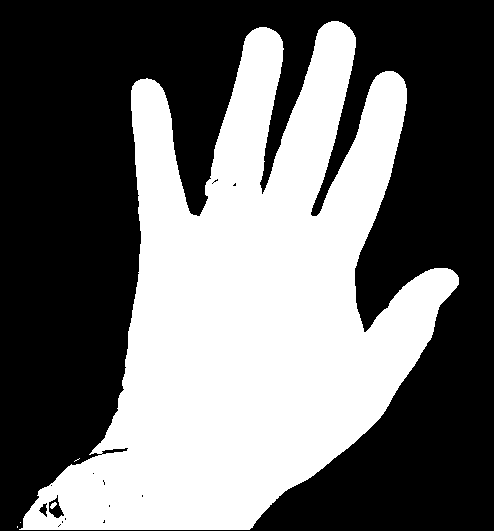
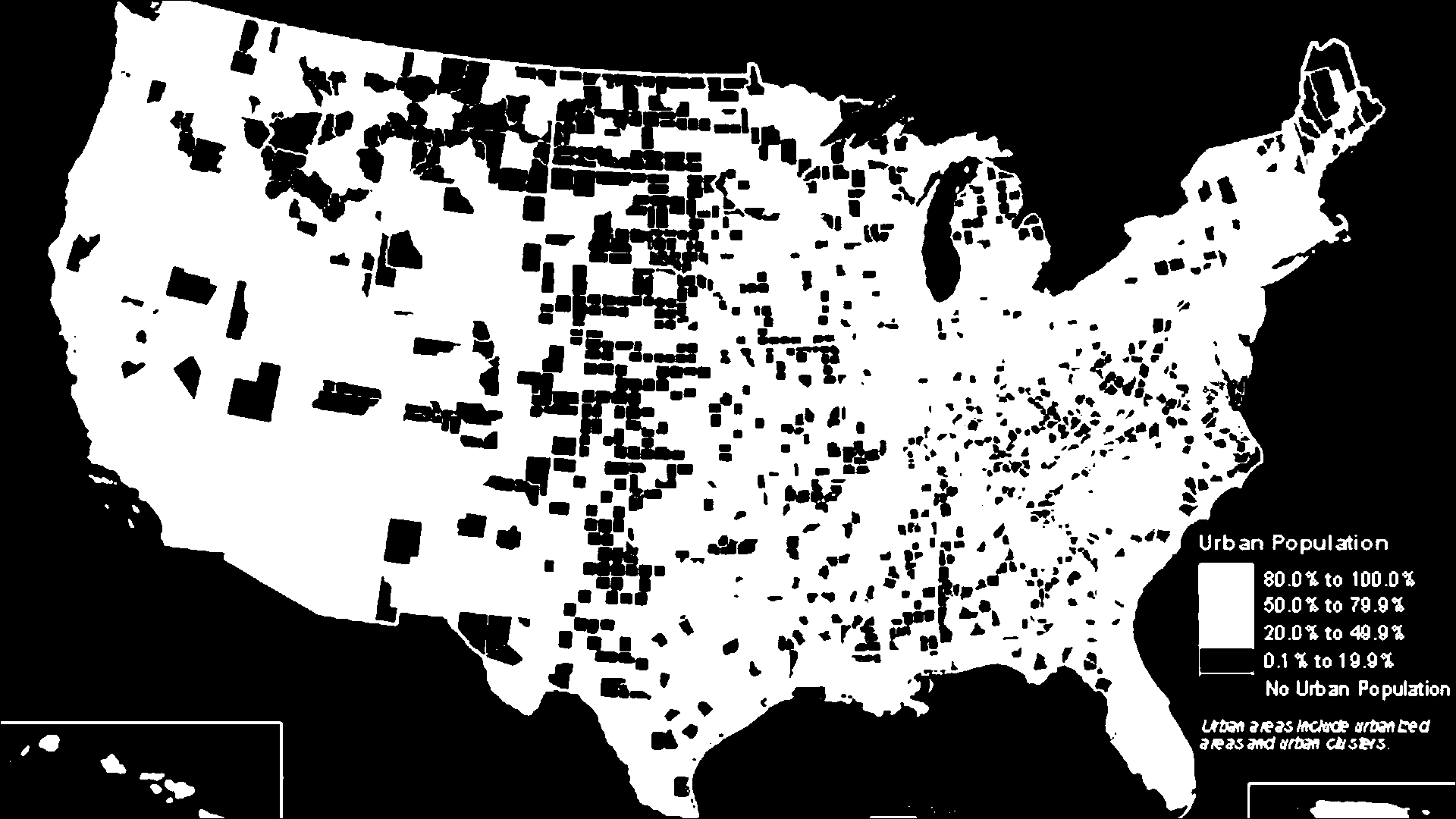
ECE 5554 SU22 – Dr. Jones – HW4

**All Images:**

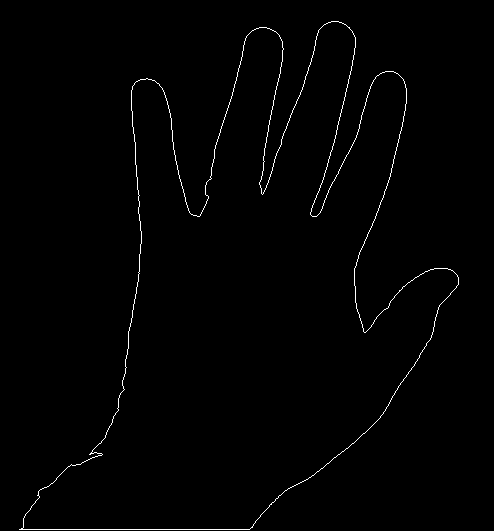
binary\_img\_hand0.png



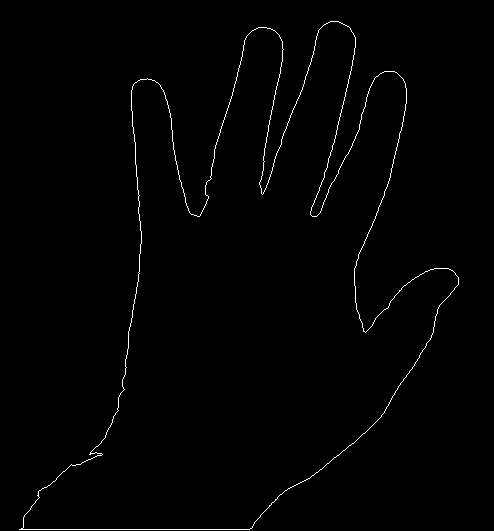
binary\_img\_US.png



dce\_1\_hand0.png



dce\_2\_hand0.png

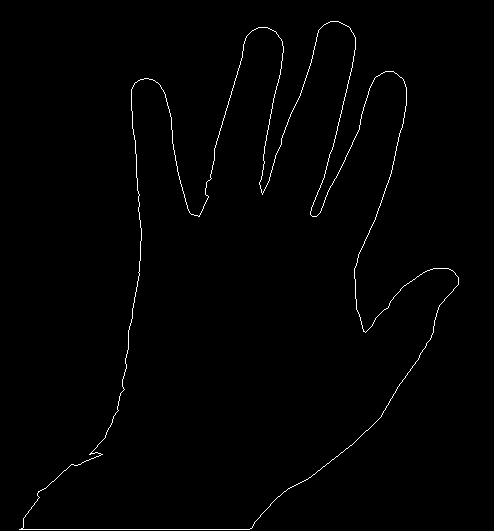


dce\_3\_hand0.png

A picture containing black, silhouette

Description automatically generated

dce\_4\_hand0.png



dce\_5\_hand0.png

A picture containing black

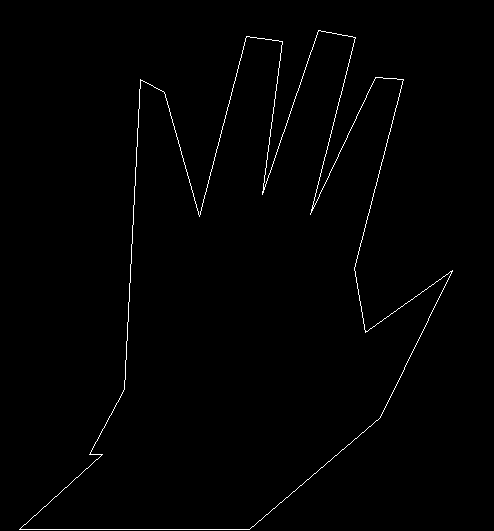
Description automatically generated

dce\_6\_hand0.png

A picture containing black

Description automatically generated

dce\_7\_hand0.png

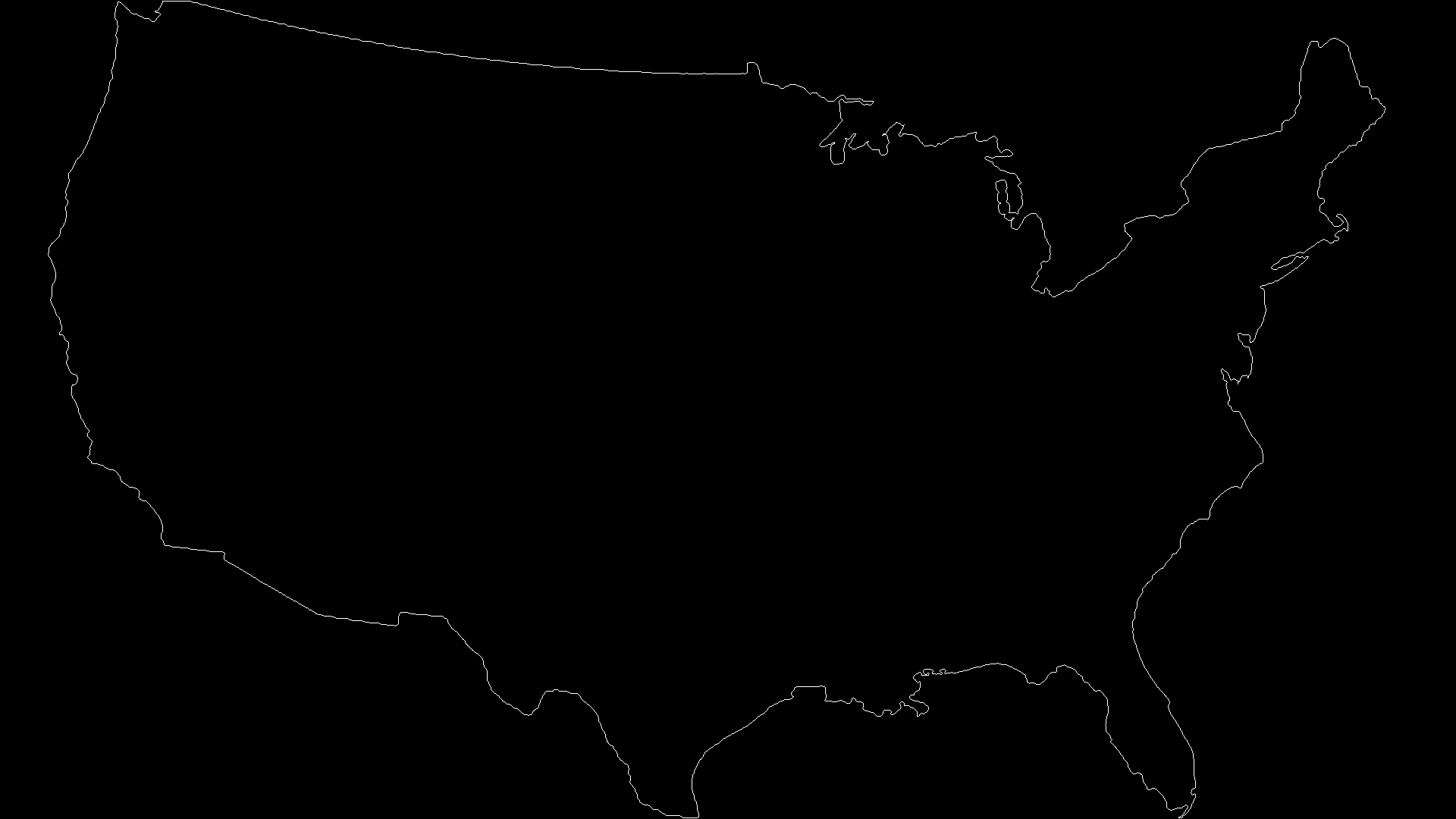


dce\_8\_hand0.png

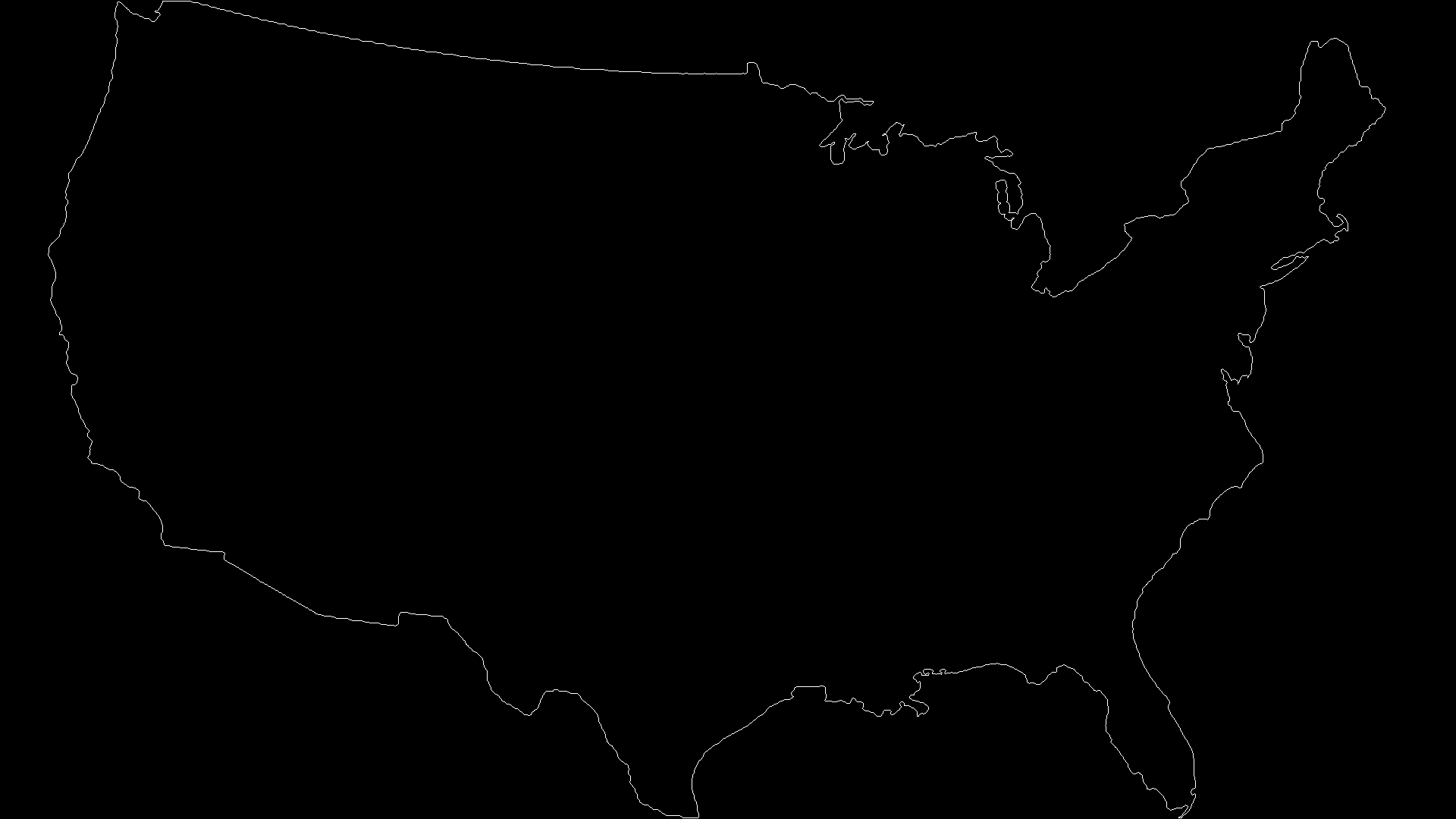
A picture containing line

Description automatically generated

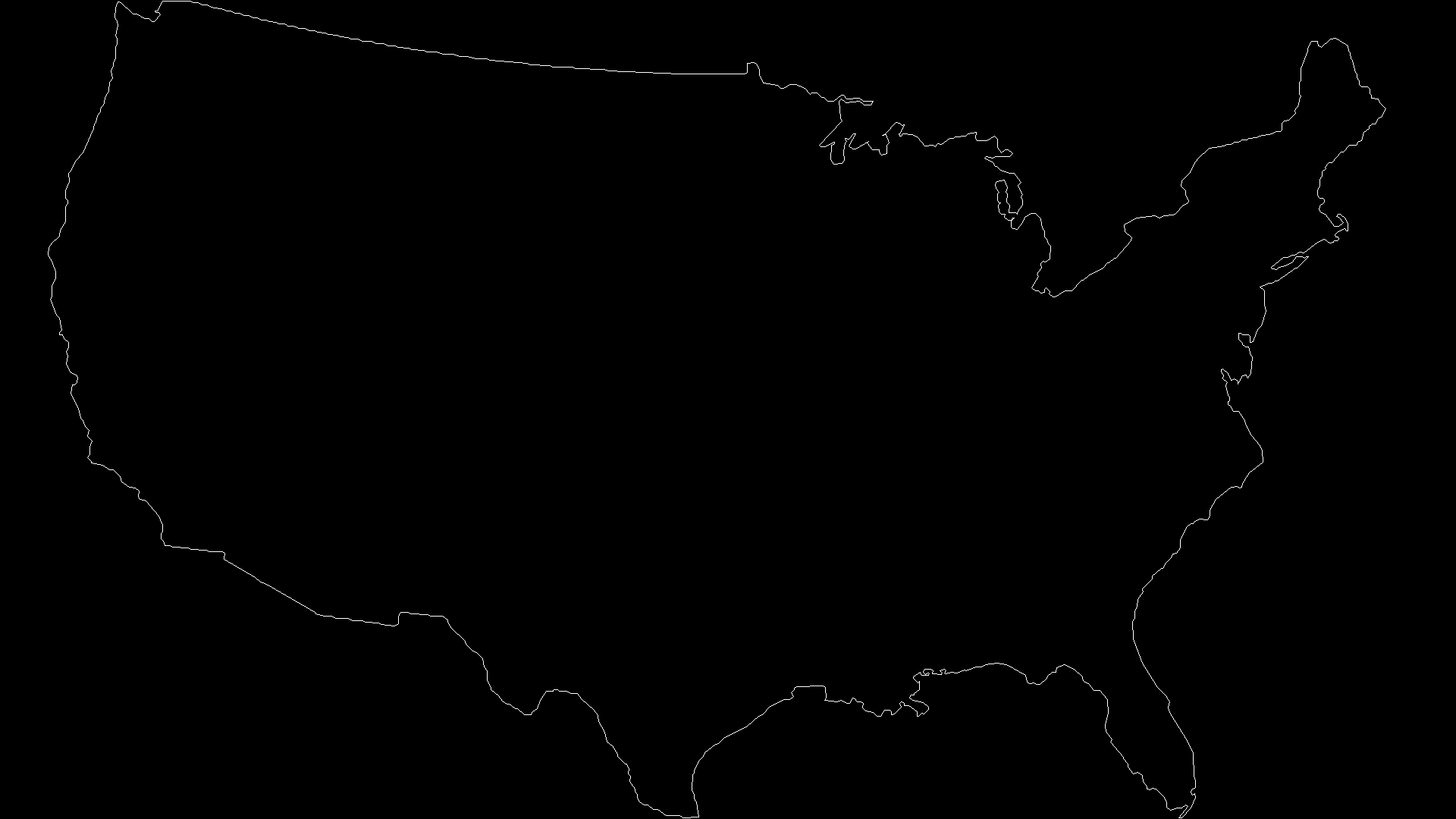
dce\_1\_US.png



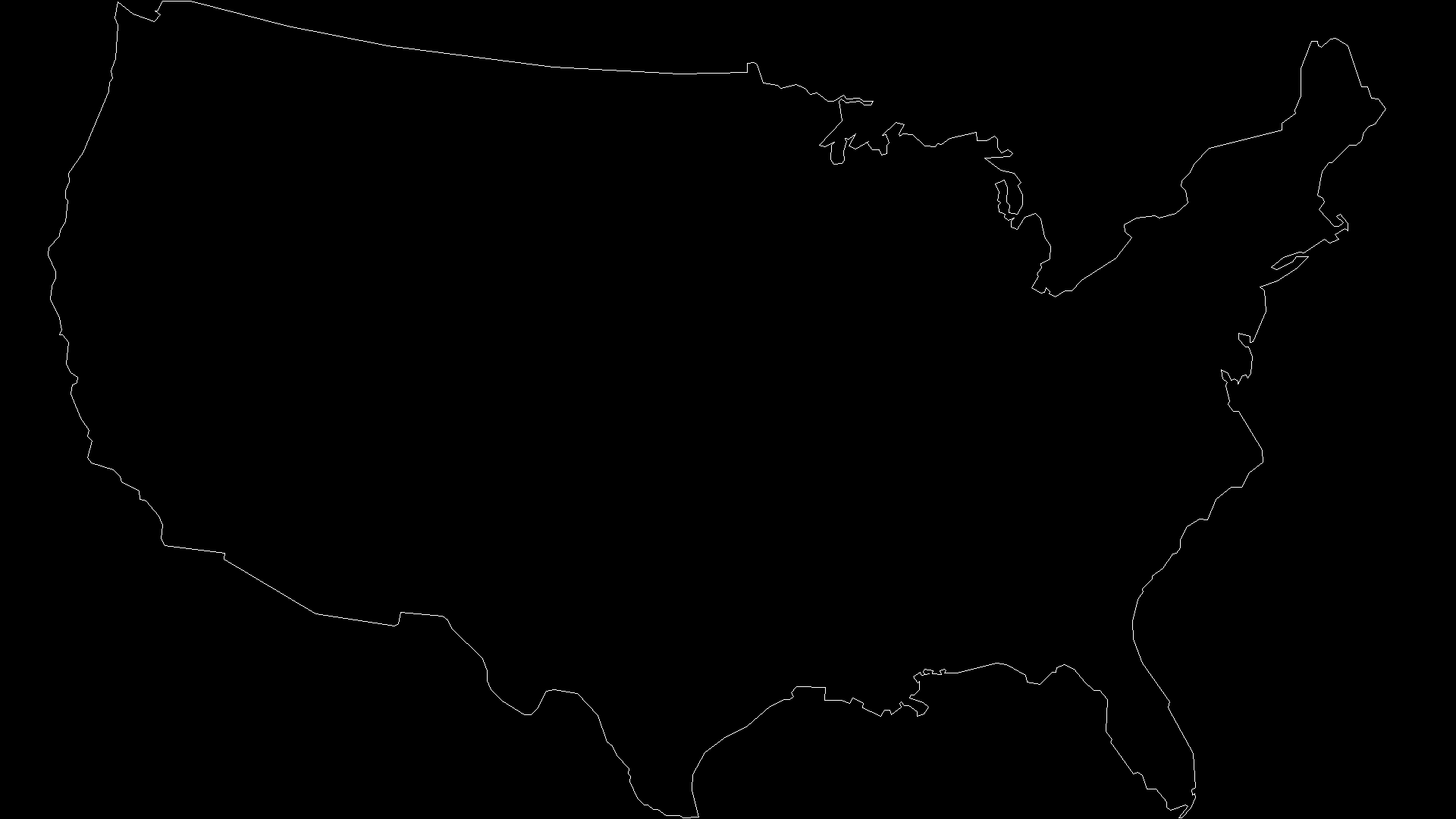
dce\_2\_US.png



dce\_3\_US.png



dce\_4\_US.png



dce\_5\_US.png

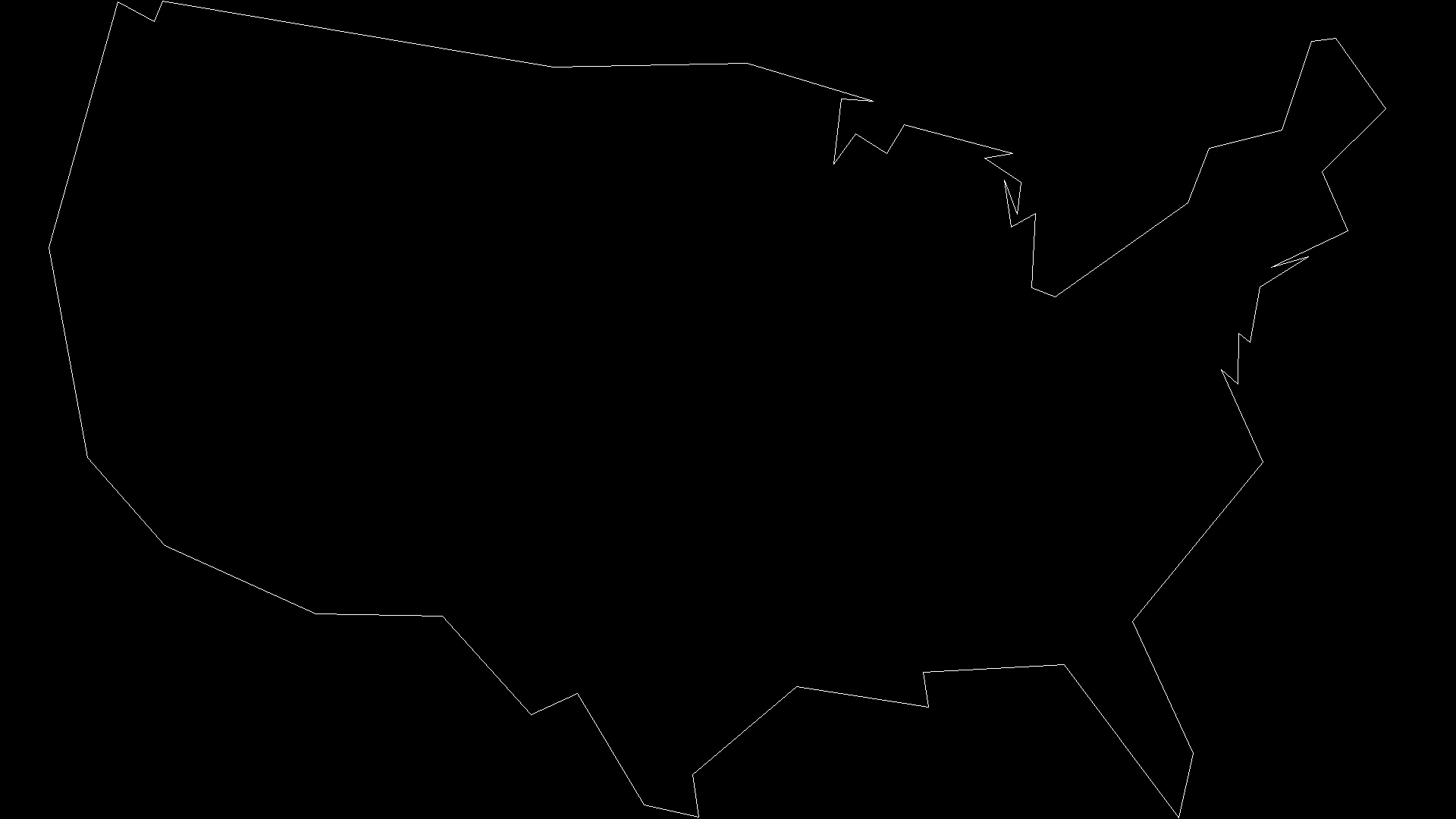


dce\_6\_US.png

A picture containing tree, outdoor object, web

Description automatically generated

dce\_7\_US.png



dce\_8\_US.png

A picture containing web

Description automatically generated

**Console Output:**

Filename: hand0.png Contour points = 2451 Area = 109708.0 Gauss Area = 111107.5

Filename: hand0.png Contour points = 1226 Gauss Area = 111107.5

Filename: hand0.png Contour points = 613 Gauss Area = 111538.5

Filename: hand0.png Contour points = 307 Gauss Area = 111486.5

Filename: hand0.png Contour points = 154 Gauss Area = 111537.0

Filename: hand0.png Contour points = 77 Gauss Area = 111749.0

Filename: hand0.png Contour points = 39 Gauss Area = 112664.5

Filename: hand0.png Contour points = 20 Gauss Area = 107527.0

Filename: hand0.png Contour points = 10 Gauss Area = 111673.0

Filename: US.png Contour points = 6781 Area = 530935.0 Gauss Area = 1156608.5

Filename: US.png Contour points = 3391 Gauss Area = 1156715.5

Filename: US.png Contour points = 1696 Gauss Area = 1156768.5

Filename: US.png Contour points = 848 Gauss Area = 1158022.5

Filename: US.png Contour points = 424 Gauss Area = 1158636.5

Filename: US.png Contour points = 212 Gauss Area = 1160126.5

Filename: US.png Contour points = 106 Gauss Area = 1163999.5

Filename: US.png Contour points = 53 Gauss Area = 1173259.0

Filename: US.png Contour points = 27 Gauss Area = 1140969.0

Code

Homework4.py:

import cv2

import numpy as np

import math

RED = (0, 0, 255)

GREEN = (0, 255, 0)

BLUE = (255, 0, 0)

NORTH = 'NORTH'

EAST = 'EAST'

SOUTH = 'SOUTH'

WEST = 'WEST'

DIRECTION = {

0: NORTH,

1: EAST,

2: SOUTH,

3: WEST,

}

# this little function calculates the area of an object from its contour

# using simple pixel counting

# holes in the object are not considered (they count as if they are filled in)

# it's really klugey - not great code

def fillArea(ctr):

maxx = np.max(ctr[:, 0]) + 1

maxy = np.max(ctr[:, 1]) + 1

contourImage = np.zeros( (maxy, maxx) )

length = ctr.shape[0]

for count in range(length):

contourImage[ctr[count, 1], ctr[count, 0]] = 255

cv2.line(contourImage, (ctr[count, 0], ctr[count, 1]),

(ctr[(count + 1) % length, 0], ctr[(count + 1) % length, 1]),

(255, 0, 255), 1)

fillMask = cv2.copyMakeBorder(contourImage, 1, 1, 1, 1,

cv2.BORDER\_CONSTANT, 0).astype(np.uint8)

areaImage = np.zeros((maxy, maxx), np.uint8)

startPoint = (int(maxy/2), int(maxx/2))

cv2.floodFill(areaImage, fillMask, startPoint, 128)

area = np.sum(areaImage)/128

return area

def gauss\_area(pts):

area = 0

for idx, xy in enumerate(pts):

#print(f"idx:{idx} x:{xy[0]} y:{xy[1]}")

# Make sure its not on the last one

if idx != len(pts)-1:

x = pts[idx][0]

y = pts[idx][1]

x1 = pts[idx+1][0]

y1 = pts[idx+1][1]

temp = x\*y1 - x1\*y

area = area + temp

return area/2

def look\_ahead(binary\_img, rowcol, direction):

r, c = rowcol

# Case Statement for directions

if DIRECTION[direction] == NORTH:

left\_rowcol = (r-1, c-1)

mid\_rowcol = (r-1, c)

right\_rowcol = (r-1, c+1)

elif DIRECTION[direction] == EAST:

left\_rowcol = (r-1, c+1)

mid\_rowcol = (r, c+1)

right\_rowcol = (r+1, c+1)

elif DIRECTION[direction] == SOUTH:

left\_rowcol = (r+1, c+1)

mid\_rowcol = (r+1, c)

right\_rowcol = (r+1, c-1)

elif DIRECTION[direction] == WEST:

left\_rowcol = (r+1, c-1)

mid\_rowcol = (r, c-1)

right\_rowcol = (r-1, c-1)

# Get pixel values for left, mid, and right

left = binary\_img[left\_rowcol]

mid = binary\_img[mid\_rowcol]

right = binary\_img[right\_rowcol]

# Set new rowcol and direction

if left == 255:

new\_rowcol = left\_rowcol

direction = direction - 1

elif left == 0 and mid == 0 and right == 0:

new\_rowcol = rowcol

direction = direction + 1

elif left == 0 and mid == 255 and right == 0:

new\_rowcol = mid\_rowcol

direction = direction

elif left == 0 and mid == 255 and right == 255:

new\_rowcol = mid\_rowcol

direction = direction

elif left == 0 and mid == 0 and right == 255:

new\_rowcol = right\_rowcol

direction = direction

# Fix if rotated out of dictionary

if direction == 4:

direction = 0

elif direction == -1:

direction = 3

return new\_rowcol, direction

def pavlidis\_contour(binary\_img, rowcol\_start):

# Setup variables

abs\_encoding = [rowcol\_start]

r\_start, c\_start = rowcol\_start

prev\_rowcol = rowcol\_start

# Start facing North

direction = 0

# Run Look Ahead once to initialize new\_rowcol and direction

new\_rowcol, direction = look\_ahead(binary\_img, (r\_start, c\_start), direction)

# Check to see if it moved, if it did move then add to list

if new\_rowcol != prev\_rowcol:

abs\_encoding.append(new\_rowcol)

prev\_rowcol = new\_rowcol

# Do this while in the starting position

while new\_rowcol == rowcol\_start:

# Look ahead to get new\_rowcol and new direction

new\_rowcol, direction = look\_ahead(binary\_img, new\_rowcol, direction)

# Append to list if it moved

if new\_rowcol != prev\_rowcol:

abs\_encoding.append(new\_rowcol)

prev\_rowcol = new\_rowcol

# Do this until it goes back to the start

while new\_rowcol != rowcol\_start:

# Look ahead to get new\_rowcol and new direction

new\_rowcol, direction = look\_ahead(binary\_img, new\_rowcol, direction)

# Append to list if it moved

if new\_rowcol != prev\_rowcol:

abs\_encoding.append(new\_rowcol)

prev\_rowcol = new\_rowcol

# Drop last one if it the same as start

if rowcol\_start == abs\_encoding[len(abs\_encoding)-1]:

abs\_encoding = abs\_encoding[:-1]

y = []

x = []

for rowcol in abs\_encoding:

y.append(rowcol[0])

x.append(rowcol[1])

contour\_pts = np.array([x,y])

contour\_pts = contour\_pts.T

return contour\_pts

def find\_first\_edge(img):

# Find all pixel locations that are 255

rows, cols = np.where(img == 255)

# Iterate over loc to create list of (r,c)

loc\_list = [(r, c) for r,c in zip(rows, cols)]

# Find one with least row and least col

row = loc\_list[0][0]

col = loc\_list[0][1]

return row, col

def OnePassDCE(pts):

relevence = []

last\_idx = len(pts)-1

for idx, v in enumerate(pts):

if idx == 0:

v\_1 = pts[last\_idx]

v\_2 = v

v\_3 = pts[idx+1]

elif idx == last\_idx:

v\_1 = pts[idx-1]

v\_2 = v

v\_3 = pts[0]

else:

v\_1 = pts[idx-1]

v\_2 = v

v\_3 = pts[idx+1]

# Calc distance

d\_12 = math.dist(v\_1, v\_2)

d\_23 = math.dist(v\_2, v\_3)

# Create Vectors

s\_i1 = np.array([v\_2[0]-v\_1[0], v\_2[1] - v\_1[1]])

s\_i2 = np.array([v\_3[0]-v\_2[0], v\_3[1] - v\_2[1]])

# Get Angle between the two

theta = math.acos(np.dot(s\_i1,s\_i2)/ (d\_12 \* d\_23))

score = (theta\*d\_12\*d\_23)/(d\_12 + d\_23)

relevence.append(score)

# Drop lowest relevent point

lowest\_idx = relevence.index(min(relevence))

pts = np.delete(pts, lowest\_idx, axis = 0)

return pts

def main(filename):

# Part a: Load image and convert to greyscale

img = cv2.imread(filename)

img\_grey = cv2.imread(filename,0)

# Part b: Convert binary image using Otsu's

(T, thresh\_img) = cv2.threshold(img\_grey, 0, 255, cv2.THRESH\_BINARY\_INV + cv2.THRESH\_OTSU)

# Part c: Add black border to img

h, w = thresh\_img.shape

thresh\_img[0] = 0

thresh\_img[h-1] = 0

thresh\_img[:,0] = 0

thresh\_img[:,w-1] = 0

# Part d: Save and show img

cv2.imshow('Before Theshold', img\_grey)

cv2.waitKey(0)

cv2.imshow('After Theshold', thresh\_img)

cv2.waitKey(0)

cv2.imwrite(f'binary\_img\_{filename}', thresh\_img)

# Part e: Find a point on the edge of the object

row, col = find\_first\_edge(thresh\_img)

# Part f: Call your own developed Pavlidis function

contour\_pts = pavlidis\_contour(thresh\_img, (row, col))

# Part g: Call provided function

contour\_area\_Creed = fillArea(contour\_pts)

# Part h: Call your own Gauss area estimation

contour\_gauss\_area = gauss\_area(contour\_pts)

# Part i: Print to console filename, number of points, actual area, and Gauss' est

print(f'Filename: {filename} Contour points = {len(contour\_pts)} Area = {contour\_area\_Creed} Gauss Area = {contour\_gauss\_area}')

# Part j: Do this 8 times

new\_pts = contour\_pts

for n in range(8):

num\_of\_pts = len(new\_pts)

# Part j.i: DCE Function

for m in tqdm(range(int(num\_of\_pts/2))):

new\_pts = OnePassDCE(new\_pts)

# Part j.ii: Connect the dots

contour\_img = np.zeros([h, w])

for idx, rowcol in enumerate(new\_pts):

if idx == len(new\_pts)-1:

cv2.line(contour\_img, rowcol, new\_pts[0], 255)

else:

cv2.line(contour\_img, rowcol, new\_pts[idx+1] , 255)

# Part j.iii: Save img

cv2.imshow(f'After {n+1} DCE', contour\_img)

cv2.waitKey(0)

cv2.imwrite(f'dce\_{n+1}\_{filename}', contour\_img)

# Part j.iv: Call Gauss function

contour\_gauss\_area = gauss\_area(new\_pts)

# Part j.v: Print to console filename, points, and Gauss est

print(f'Filename: {filename} Contour points = {len(new\_pts)} Gauss Area = {contour\_gauss\_area}')

if \_\_name\_\_ == '\_\_main\_\_':

filenames = ['hand0.png', 'US.png']

for filename in filenames:

main(filename)