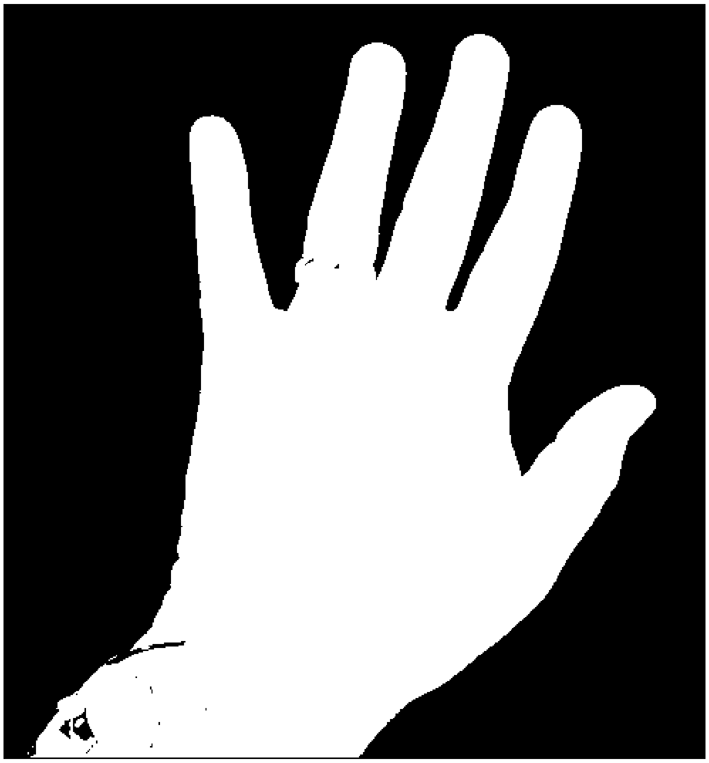
Andrew Garcia
8/3/2022

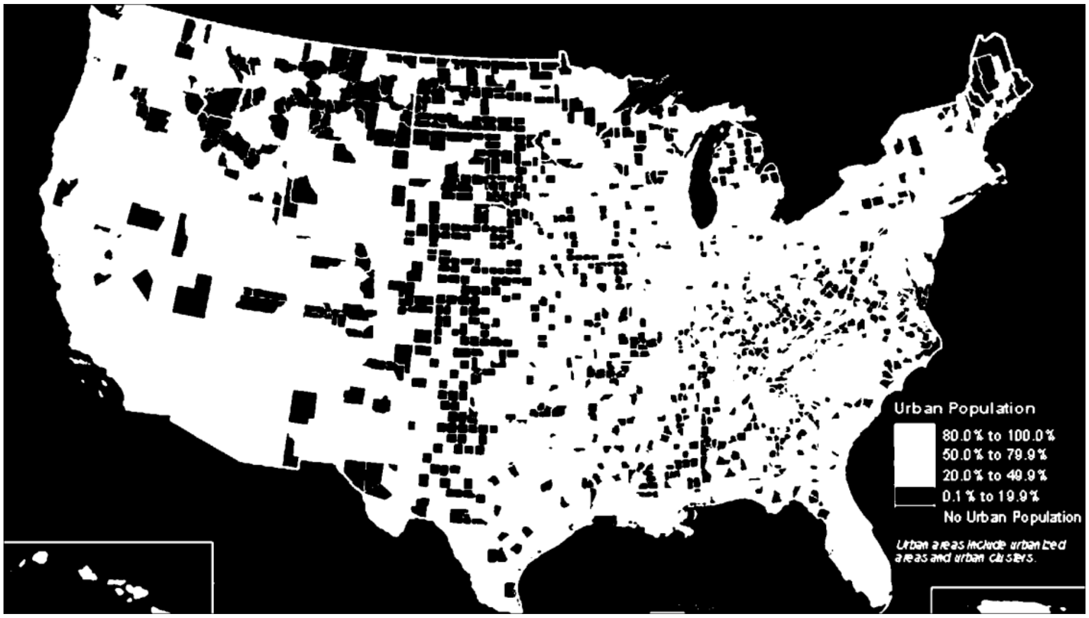
# ECE 5554 SU22 – Dr. Jones – HW4

**All Images:**

binary_img_hand0.png



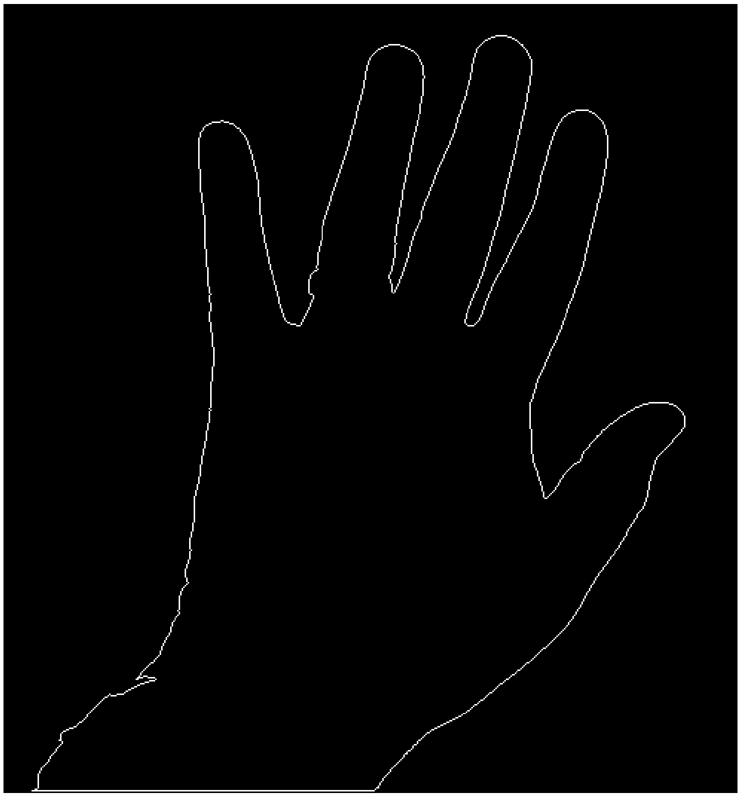
binary_img_US.png



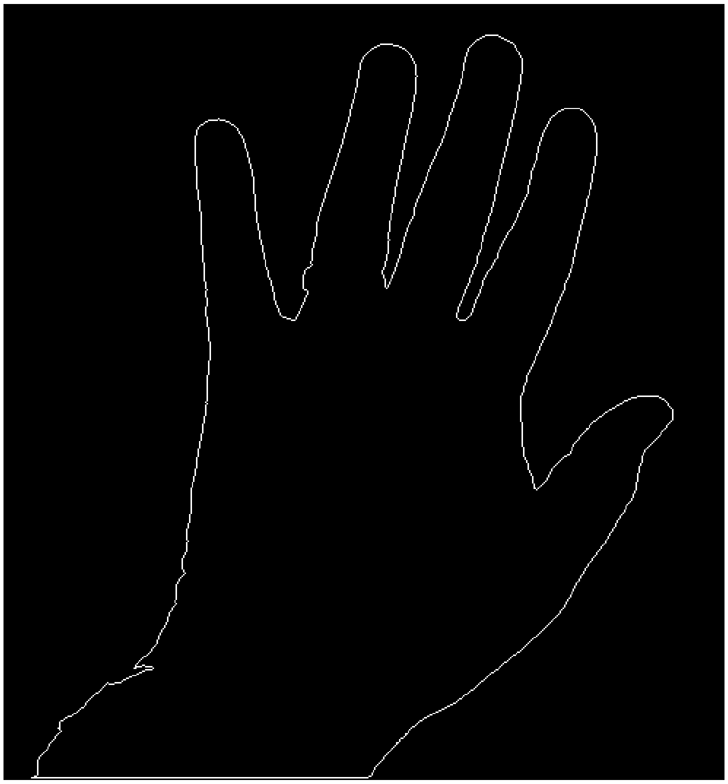
Andrew Garcia
8/3/2022

Andrew Garcia
8/3/2022

dce_1_hand0.png

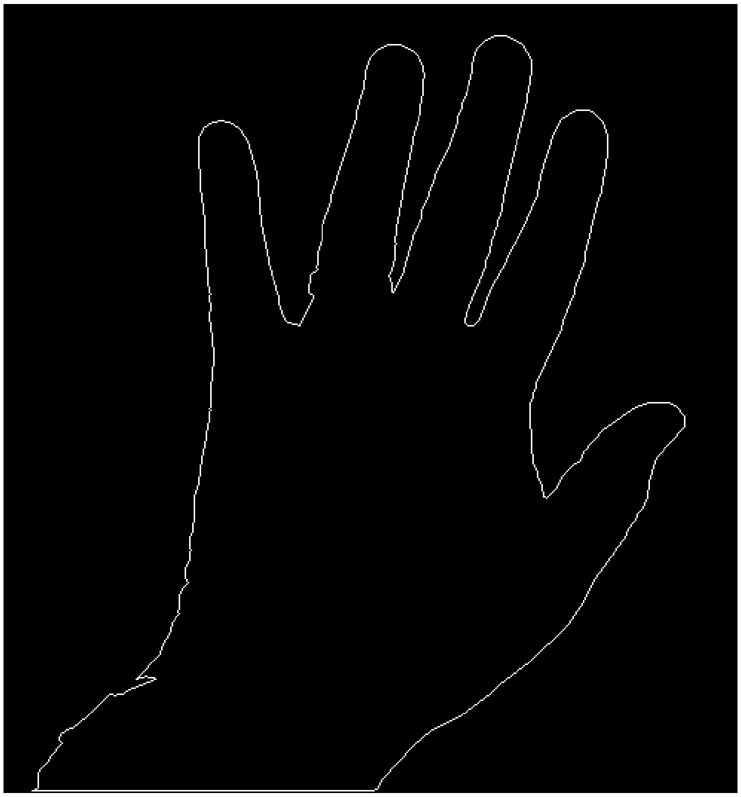


dce_2_hand0.png

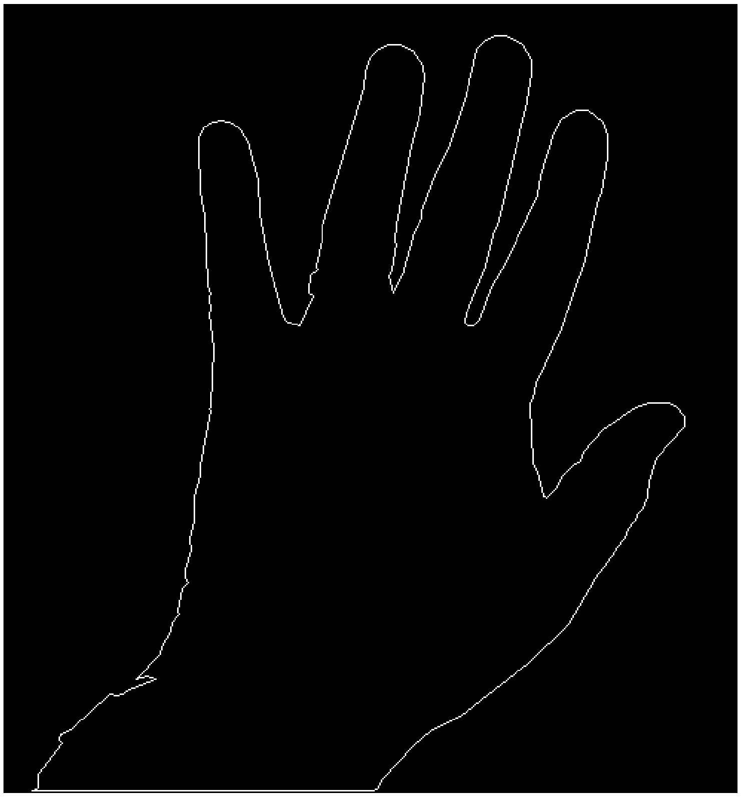


Andrew Garcia
8/3/2022

dce_3_hand0.png

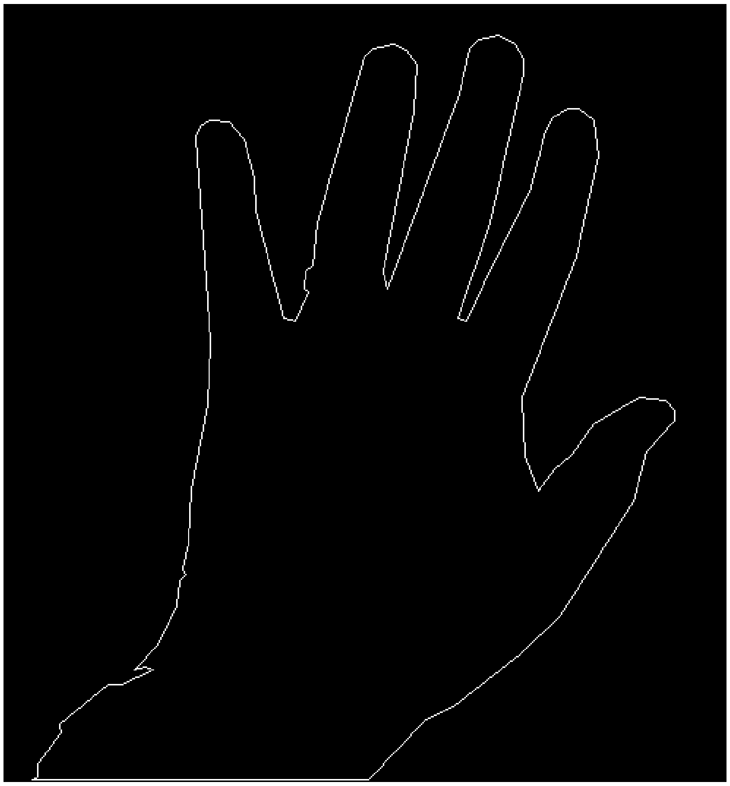


dce_4_hand0.png

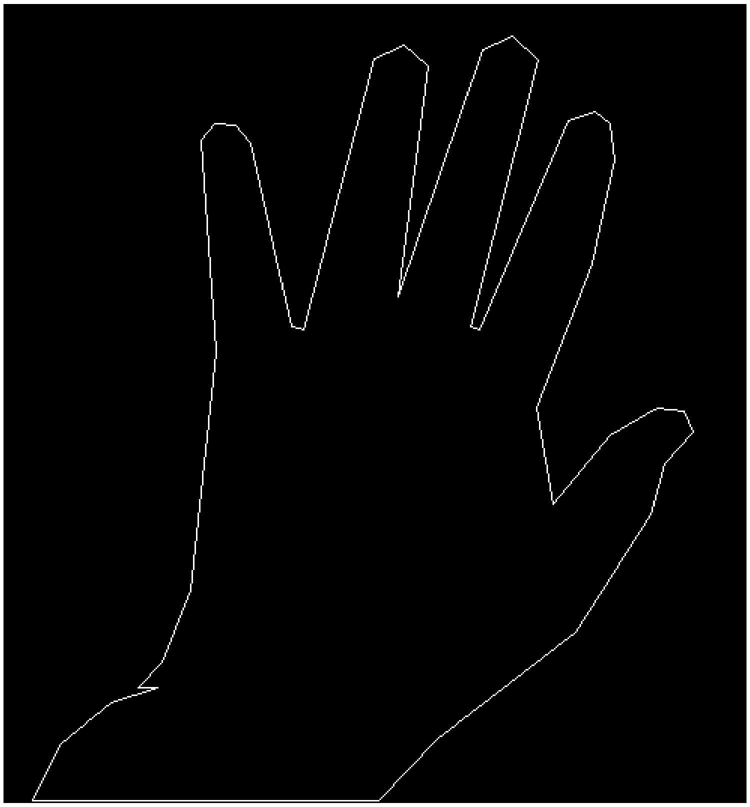


Andrew Garcia
8/3/2022

dce_5_hand0.png

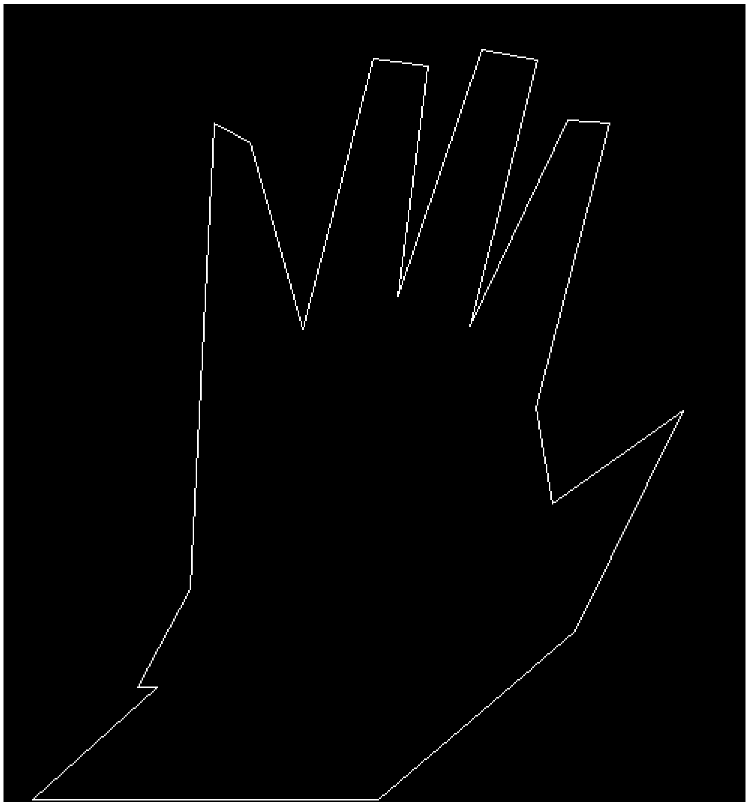


dce_6_hand0.png

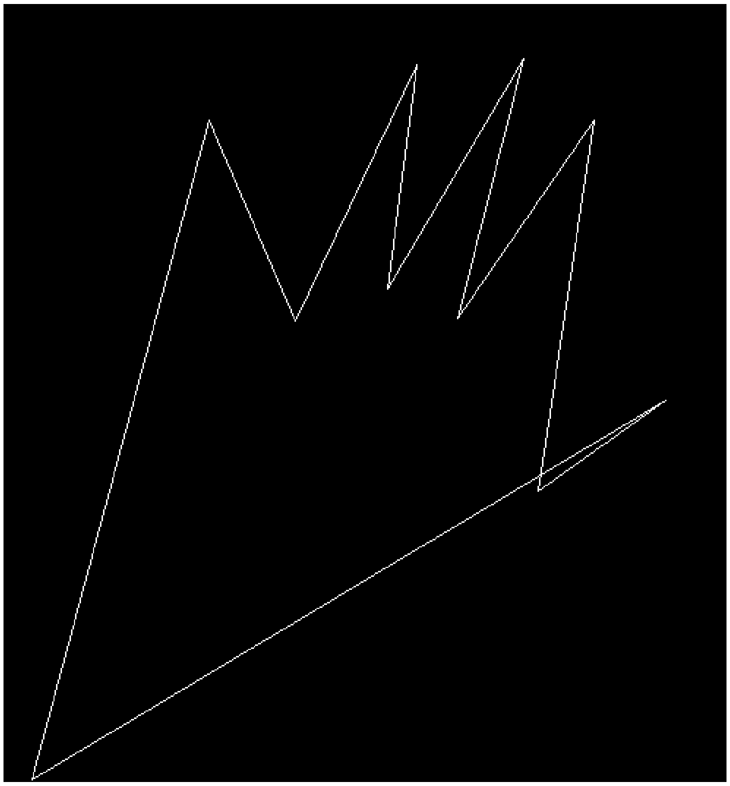


Andrew Garcia
8/3/2022

dce_7_hand0.png

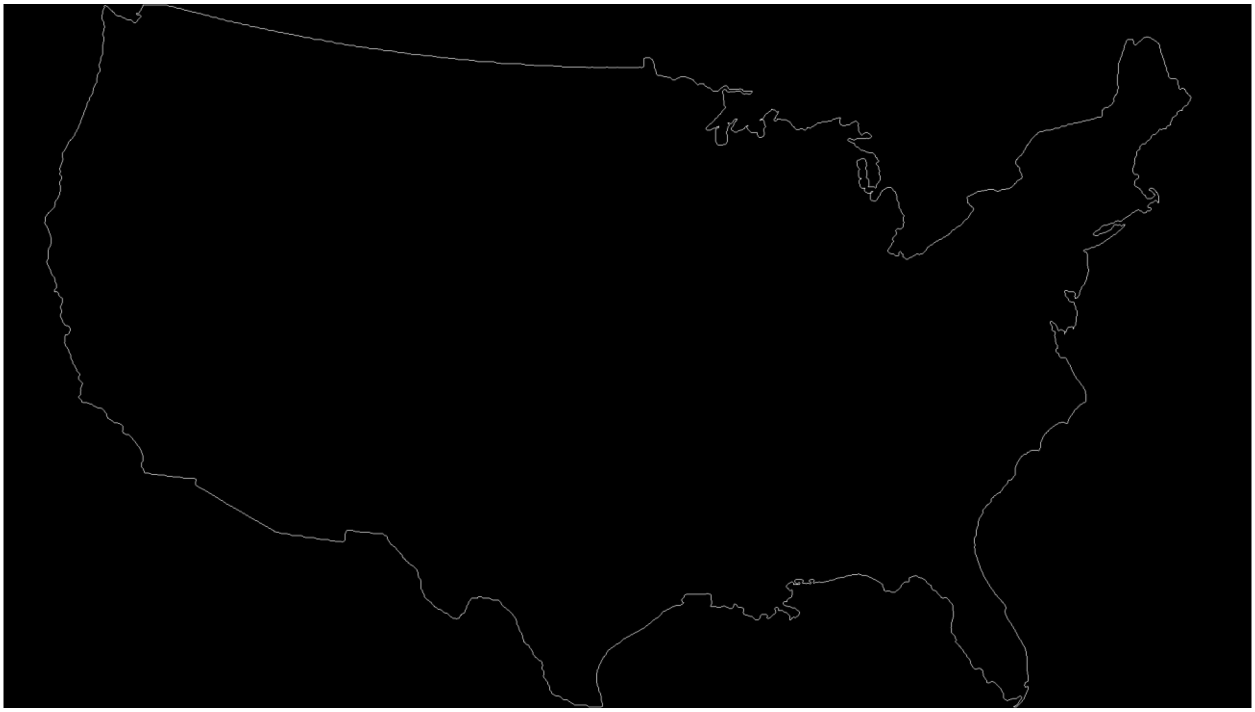


dce_8_hand0.png

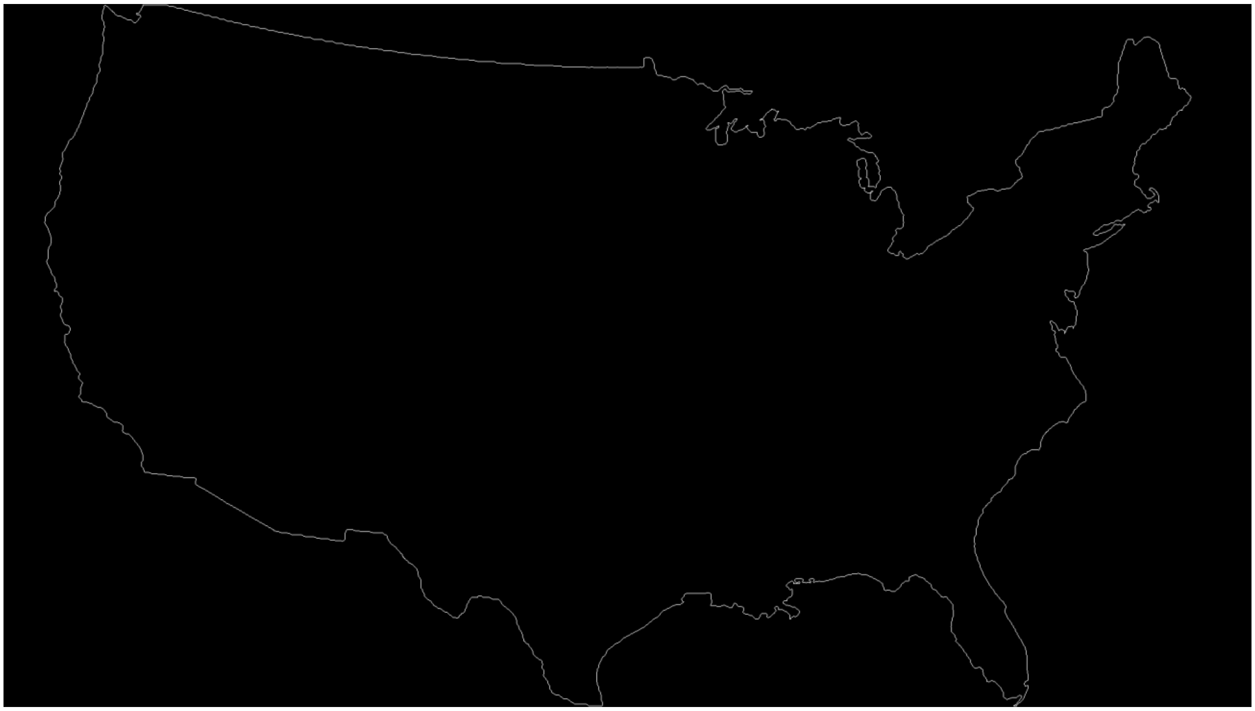


Andrew Garcia
8/3/2022

dce_1_US.png

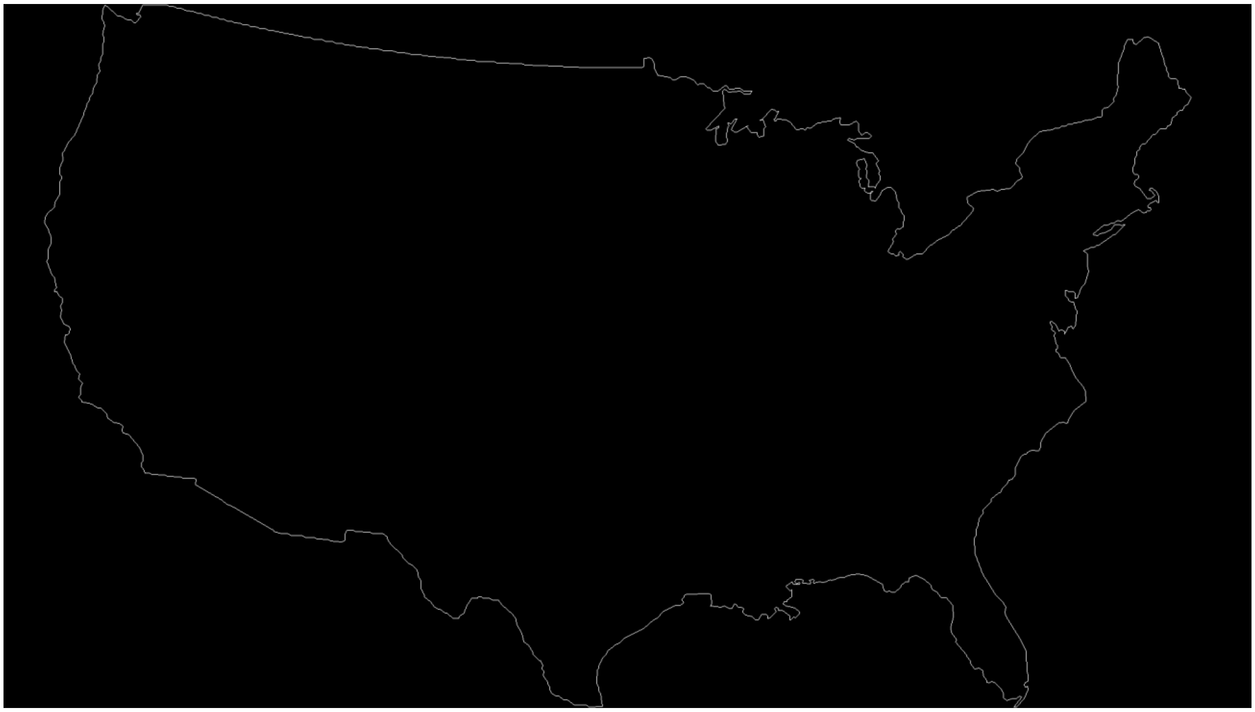


dce_2_US.png

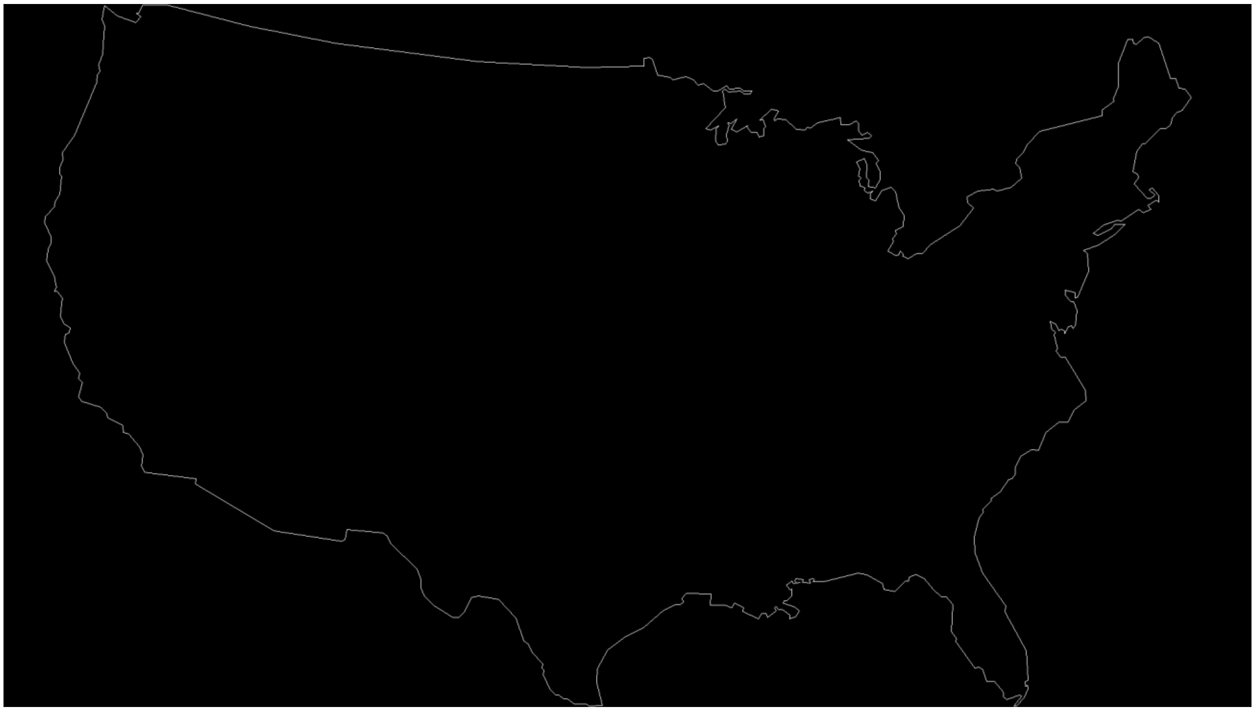


Andrew Garcia
8/3/2022

dce_3_US.png

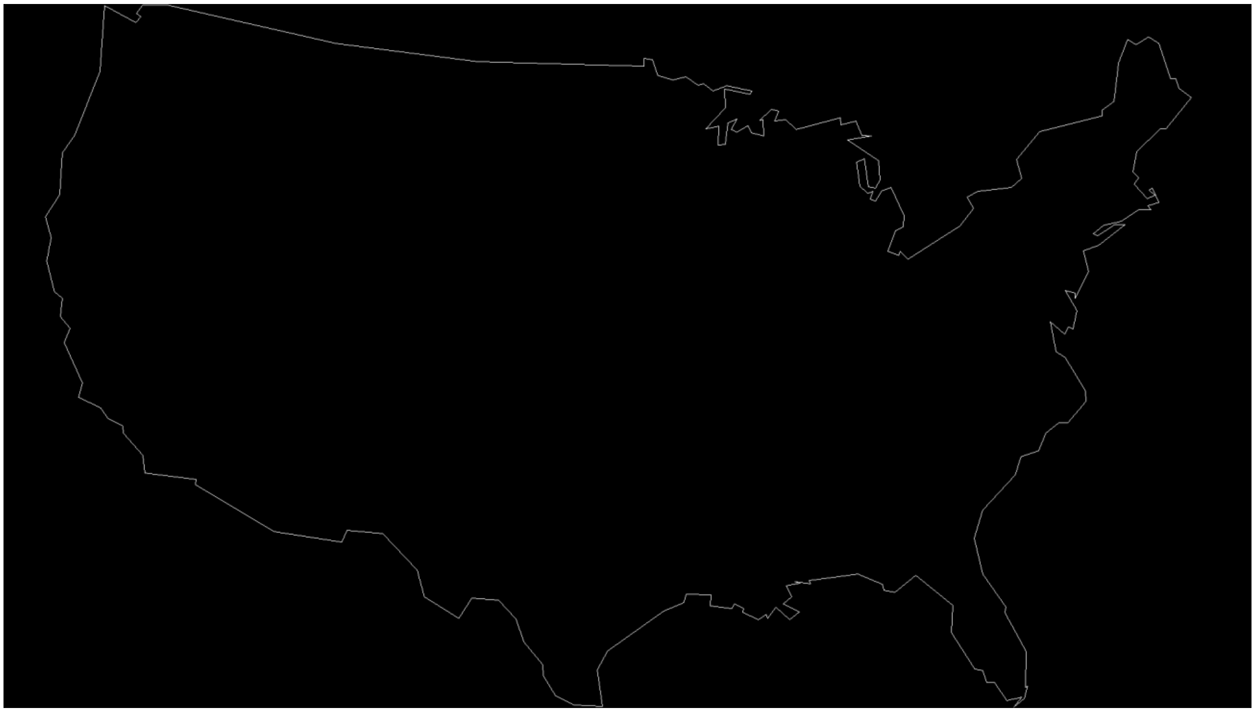


dce_4_US.png

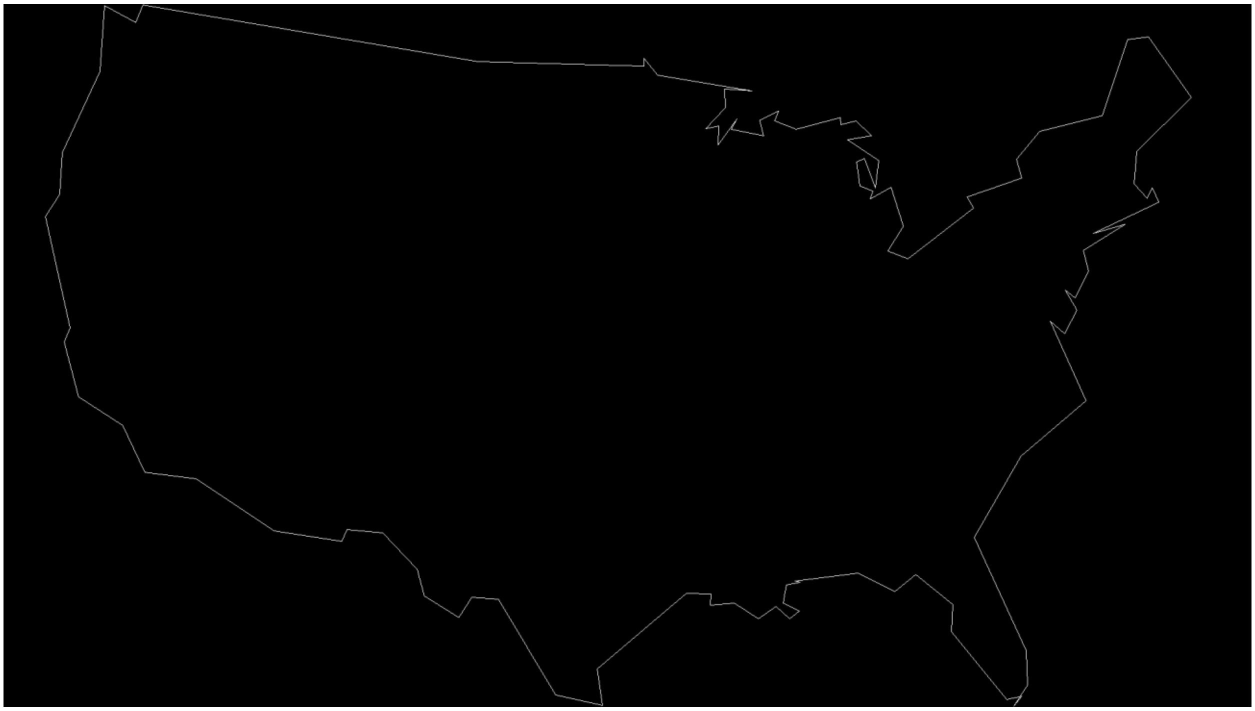


Andrew Garcia
8/3/2022

dce_5_US.png

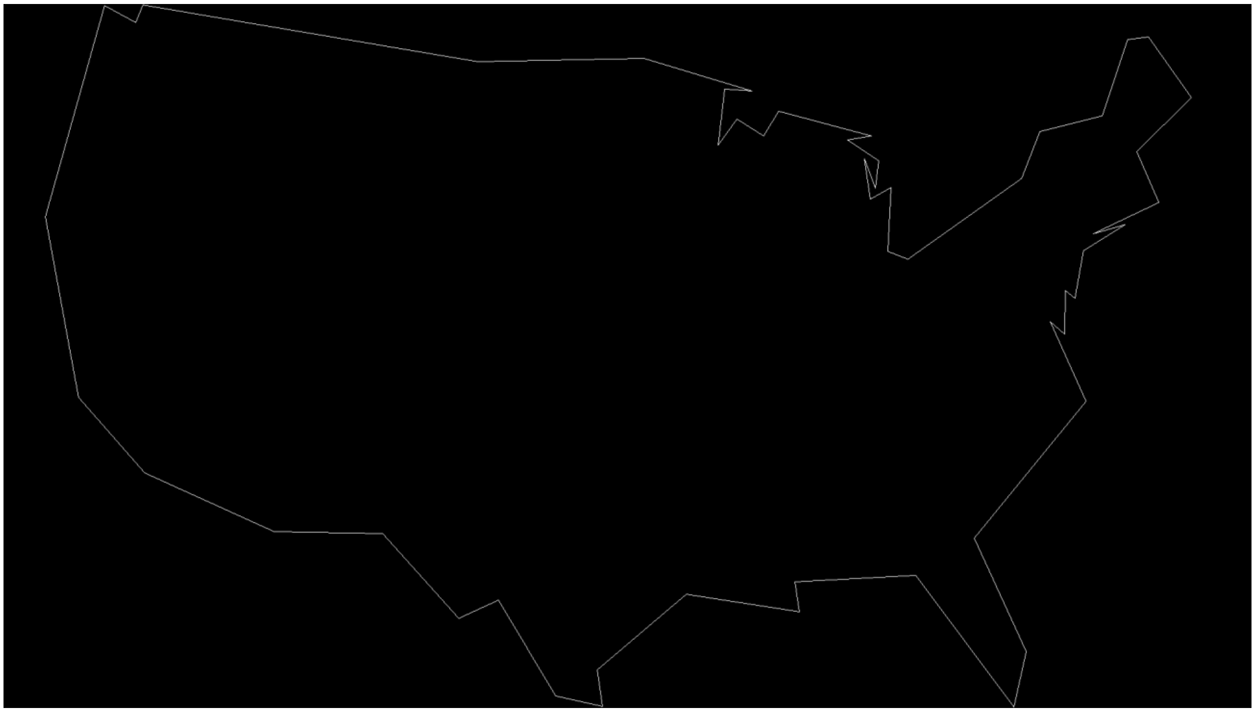


dce_6_US.png

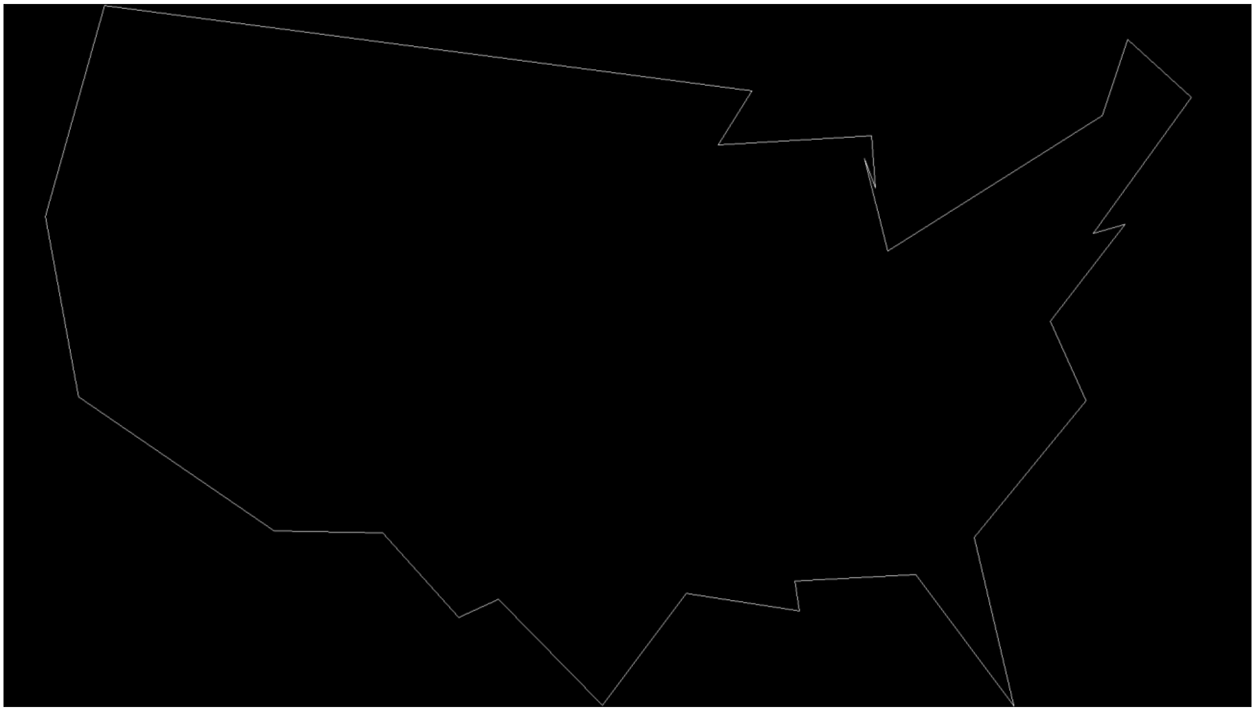


Andrew Garcia
8/3/2022

dce_7_US.png



dce_8_US.png



Andrew Garcia
8/3/2022

**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
```

Andrew Garcia
8/3/2022

# 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
```

Andrew Garcia
8/3/2022

```
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
```

Andrew Garcia
8/3/2022

```
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)
```