E:\3) Image Lab\Lab4_Fourier_Domain_Filtering\fourier_assignment\fourier_assignment.py

fourier_assignment.py

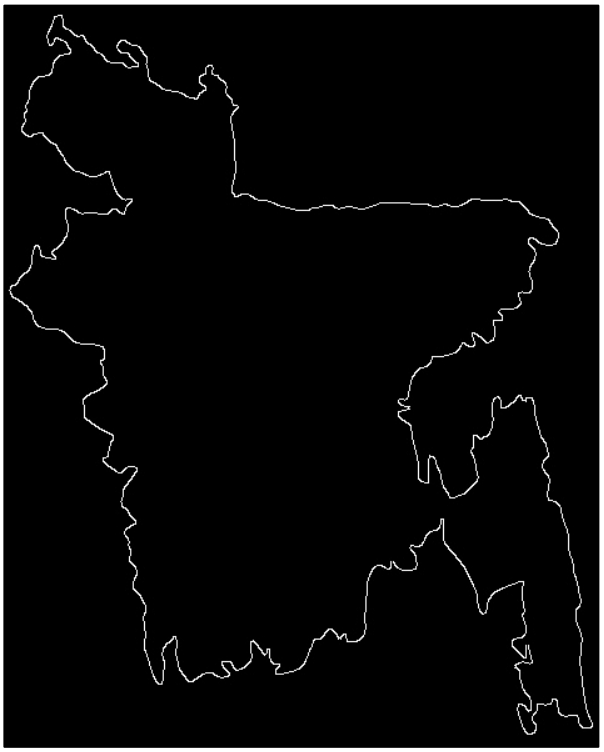
```python
import cv2 # for finding contours to extract points of figure
import matplotlib.pyplot as plt # for plotting and creating figures
import numpy as np # for easy and fast number calculation
from math import tau # tau is constant number = 2*PI
from scipy.integrate import quad_vec # for calculating definite integral
from tqdm import tqdm # for progress bar
import matplotlib.animation as animation # for compiling animation and exporting video

# function to generate x+iy at given time t
def f(t, t_list, x_list, y_list):
    return np.interp(t, t_list, x_list + 1j*y_list)

# reading the image and converting to greyscale mode
img = cv2.imread("bangladesh_input.jpg")
img = cv2.resize(img, (500,500))
cv2.imshow('Input Colored Image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('Grey Image', img_gray)
cv2.imwrite("bangladesh_grey.jpg", img_gray)
cv2.waitKey(0)
cv2.destroyAllWindows()

# finding the contours in the image
# making pure black and white image
ret, thresh = cv2.threshold(img_gray, 127, 255, 0)
# finding available contours i.e closed loop objects
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
# contour at index 1 is the one looking for
contours = np.array(contours[1])

# displaying the contours on the original image
cv2.drawContours(img, contours, -1, (0,255,0), 3)
cv2.imshow('Contours', img)
cv2.imwrite("bangladesh_contours.jpg", img)
cv2.waitKey(0)
cv2.destroyAllWindows()

# splitting the co-ordinate points of the contour
# reshaping this to make it 1D array
x_list, y_list = contours[:, :, 0].reshape(-1,), -contours[:, :, 1].reshape(-1,)

# centering the contour to origin
x_list = x_list - np.mean(x_list)
y_list = y_list - np.mean(y_list)

# visualizing the contour
fig = plt.figure()
ax = fig.add_subplot(111)
ax.plot(x_list, y_list)

# later these data will be needed to fix the size of figure
xlim_data = plt.xlim()
ylim_data = plt.ylim()
plt.show()
```

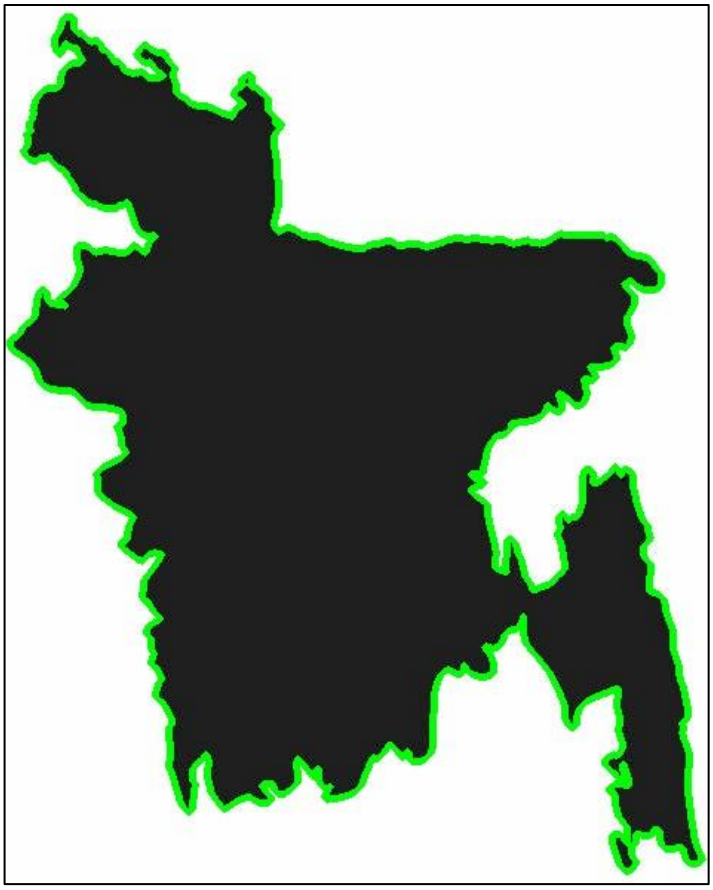
```python
# time data from 0 to 2*PI as x,y is the function of time
t_list = np.linspace(0, tau, len(x_list))
# now we can relate f(t) -> x,y

# Now finding fourier coefficient from -n to n circles
# ..., c-3, c-2, c-1, c0, c1, c2, c3, ...
order = 100 # -order to order i.e -100 to 100

print("generating coefficients ...")
# computing fourier coefficients from -order to order
c = []
pbar = tqdm(total=(order*2+1))
# calculating the coefficients from -order to order
for n in range(-order, order+1):
    # calculating definite integration from 0 to 2*PI using formula
    coef = 1/tau*quad_vec(lambda t: f(t, t_list, x_list, y_list)*np.exp(-n*t*1j), 0, tau, limit=100,
    c.append(coef)
    pbar.update(1)
pbar.close()
print("completed generating coefficients.")
```

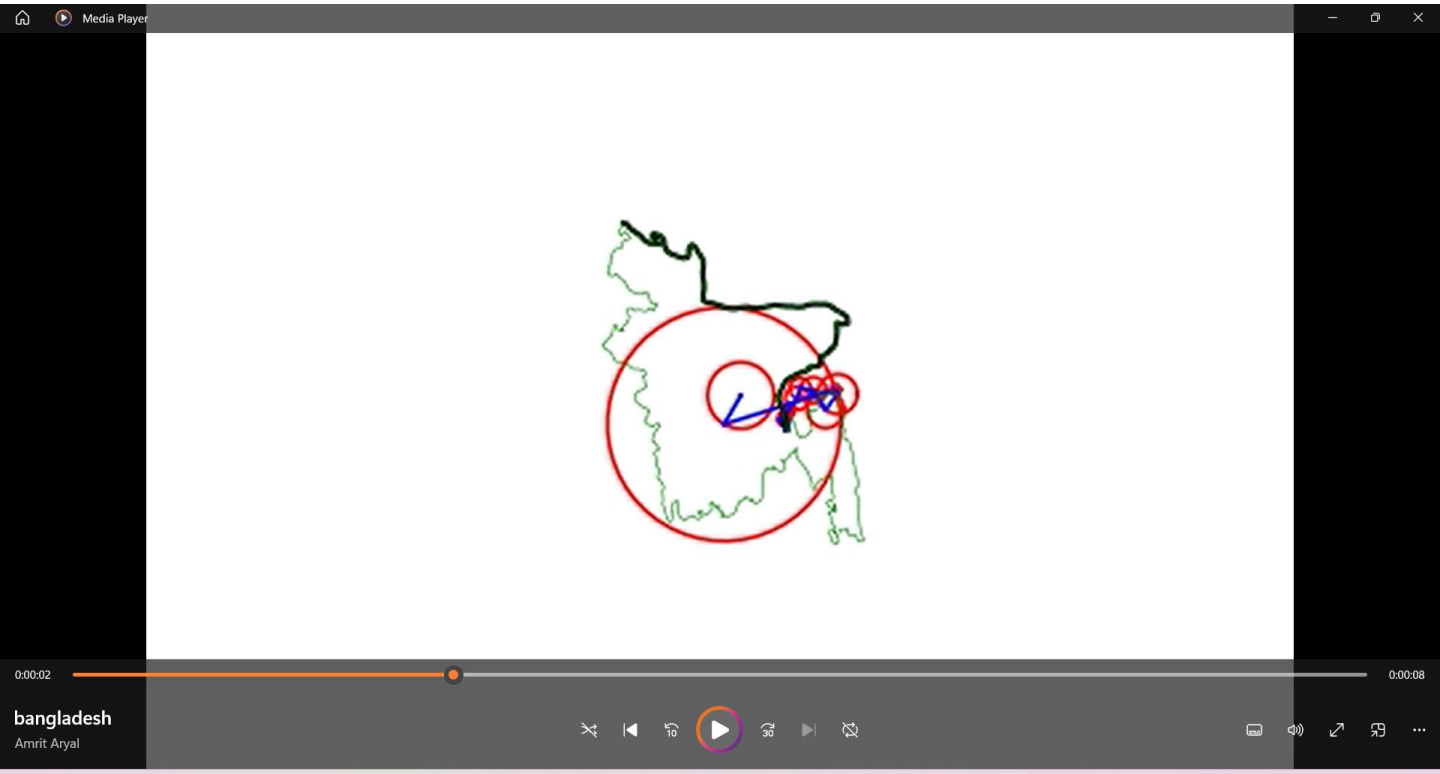
```python
# converting list into numpy array
c = np.array(c)

# saving the coefficients for later use
np.save("coeff.npy", c)

## -- now to making animation with epicycle -- ##

# this is to store the points of last circle of epicycle which draws the required figure
draw_x, draw_y = [], []

# making figure for animation
fig, ax = plt.subplots()

# different plots to make epicycle
# there are -order to order numbers of circles
circles = [ax.plot([], [], 'r-')[0] for i in range(-order, order+1)]
# circle_lines are radius of each circles
circle_lines = [ax.plot([], [], 'b-')[0] for i in range(-order, order+1)]
# drawing is plot of final drawing
drawing, = ax.plot([], [], 'k-', linewidth=2)

# original drawing
orig_drawing, = ax.plot([], [], 'g-', linewidth=0.5)

# to fix the size of figure so that the figure does not get cropped/trimmed
ax.set_xlim(xlim_data[0]-200, xlim_data[1]+200)
ax.set_ylim(ylim_data[0]-200, ylim_data[1]+200)

# hiding axes
ax.set_axis_off()

# to have symmetric axes
ax.set_aspect('equal')

# setting up formatting for the video file
Writer = animation.writers['ffmpeg']
writer = Writer(fps=30, metadata=dict(artist='Amrit Aryal'), bitrate=1800)
```

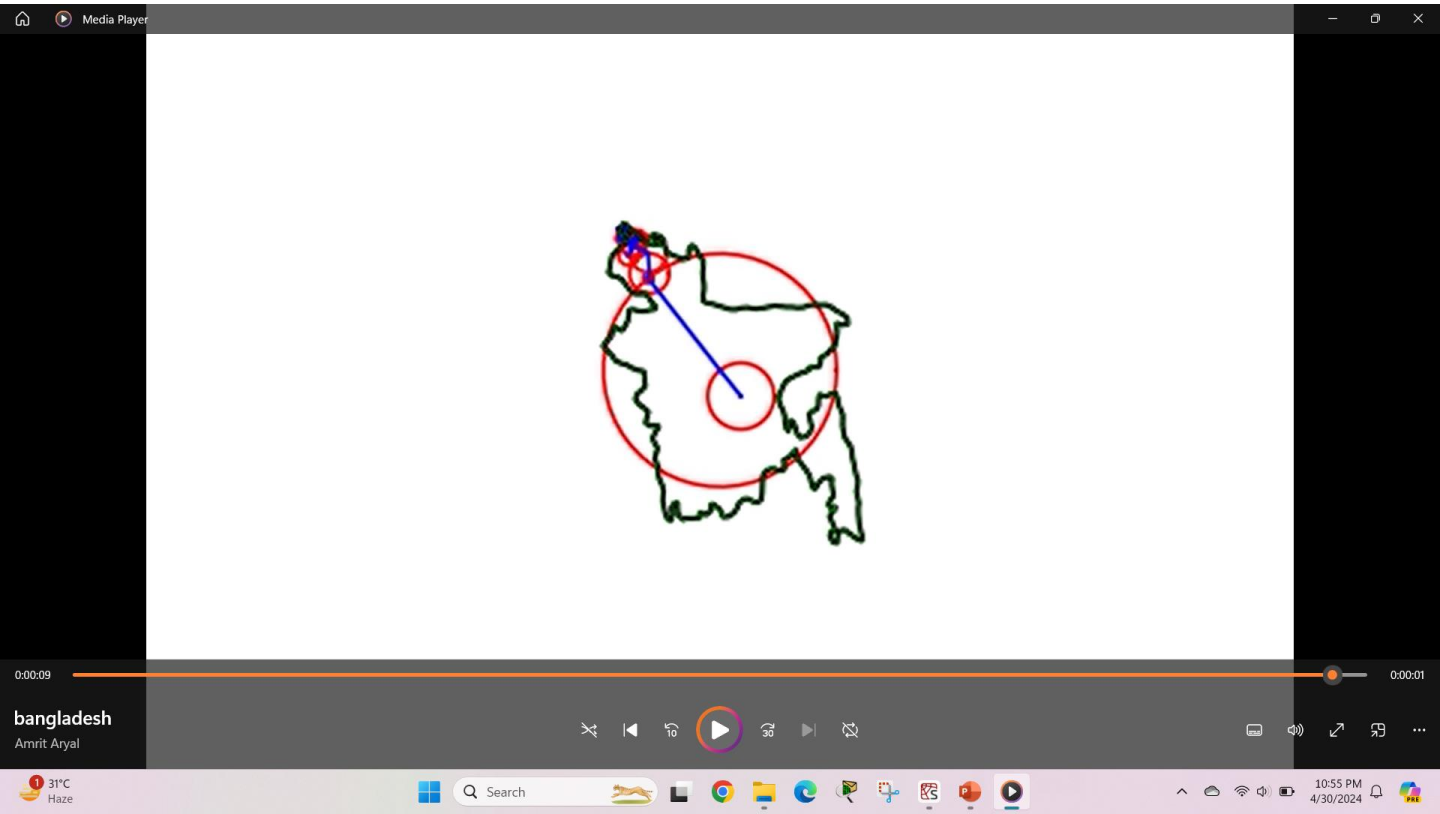
```python
print("compiling animation ...")
# setting number of frames
frames = 300
pbar = tqdm(total=frames)

# saving the coefficients in order 0, 1, -1, 2, -2, ...
# it is necessary to make epicycles
def sort_coeff(coeffs):
    new_coeffs = []
    new_coeffs.append(coeffs[order])
    for i in range(1, order+1):
        new_coeffs.extend([coeffs[order+i],coeffs[order-i]])
    return np.array(new_coeffs)
```

```python
# making frame at time t
# t goes from 0 to 2*PI for complete cycle
def make_frame(i, time, coeffs):
    global pbar
    # getting t from time
    t = time[i]

    # exponential term to be multiplied with coefficient
    # this is responsible for making rotation of circle
    exp_term = np.array([np.exp(n*t*1j) for n in range(-order, order+1)])

    # sorting the terms of fourier expression
    coeffs = sort_coeff(coeffs*exp_term) # coeffs*exp_term makes the circle rotate.
    # coeffs itself gives only direction and size of circle

    # splitting into x and y coefficients
    x_coeffs = np.real(coeffs)
    y_coeffs = np.imag(coeffs)

    # centering points for fisrt circle
    center_x, center_y = 0, 0

    # making all circles i.e epicycle
    for i, (x_coeff, y_coeff) in enumerate(zip(x_coeffs, y_coeffs)):
        # calculating radius of current circle
        r = np.linalg.norm([x_coeff, y_coeff]) # similar to magnitude: sqrt(x^2+y^2)

        # draw circle with given radius at given center points of circle
        # circumference points: x = center_x + r * cos(theta), y = center_y + r * sin(theta)
        theta = np.linspace(0, tau, num=50) # theta should go from 0 to 2*PI to get all points
        x, y = center_x + r * np.cos(theta), center_y + r * np.sin(theta)
        circles[i].set_data(x, y)

        # drawing a line to indicate the direction of circle
        x, y = [center_x, center_x + x_coeff], [center_y, center_y + y_coeff]
        circle_lines[i].set_data(x, y)

        # calculating center for next circle
        center_x, center_y = center_x + x_coeff, center_y + y_coeff

    # centering points now are points from last circle
    # these points are used as drawing points
    draw_x.append(center_x)
    draw_y.append(center_y)

    # drawing the curve from last point
    drawing.set_data(draw_x, draw_y)

    # drawing the real curve
    orig_drawing.set_data(x_list, y_list)

    # updating progress bar
    pbar.update(1)

# making animation
# time is array from 0 to tau
time = np.linspace(0, tau, num=frames)
anim = animation.FuncAnimation(fig, make_frame, frames=frames, fargs=(time, c),interval=5)
anim.save('bangladesh.mp4', writer=writer)
pbar.close()
print("completed: epicycle.mp4")
```






Media Player
0:00:02
0:00:08
bangladesh
Amrit Aryal
31°C
Haze
Search
10:55 PM
4/30/2024


Media Player
0:00:09
0:00:01
bangladesh
Amrit Aryal
31°C
Haze
Search
10:55 PM
4/30/2024