EE2703 : Images and Animation

Annangi Shashank babu (EE21B021)

March 8, 2023

Firstly we import the packages that helps us to animate

[10]:
```
import numpy as np
%matplotlib ipympl
import matplotlib.pyplot as plt
import matplotlib.animation
from matplotlib.animation import FuncAnimation
import math
```

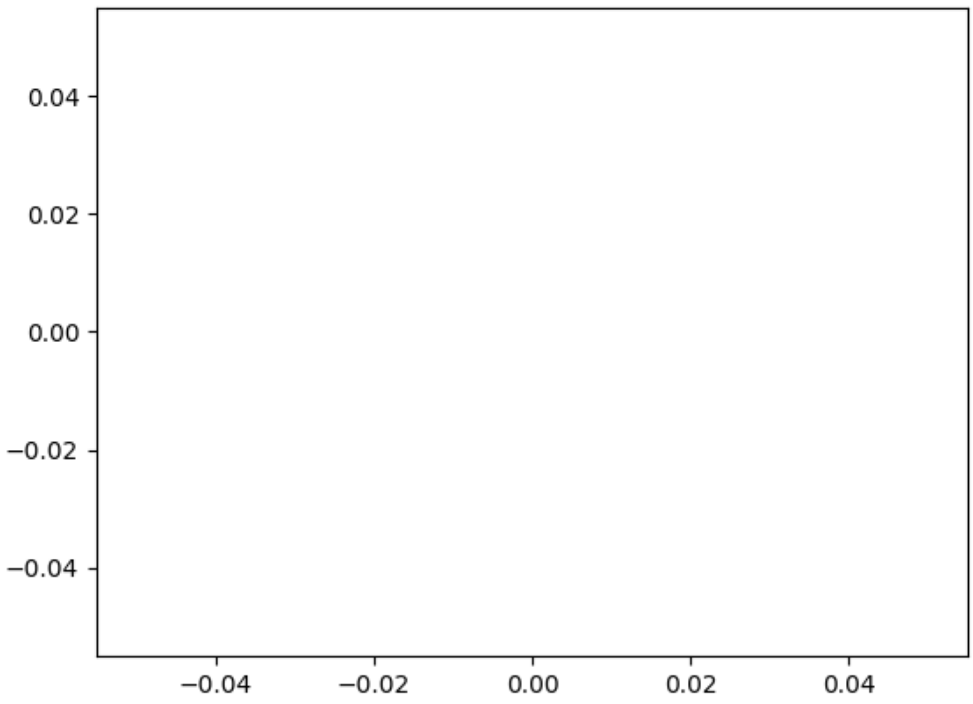
Setup required for the FuncAnimation to work

[12]:
```
fig, ax = plt.subplots()
xdata, ydata = [], []
ln, = ax.plot([], [], 'r')

#Helping Functions :
def factorial(n):
    """computes the factorial of the number

    :n: the number for which factorial is computed
    :returns: the factorial of n

    """
    ans = 1
    for i in range(1,n+1):
        ans *= i
    return ans
def init():
    ax.set_xlim(-1.2, 1.2)
    ax.set_ylim(-1.2, 1.2)
    return ln,
def morph(x1, y1, x2, y2, alpha,side_current):
    #print(type(x1))
    xm = alpha * x1 + (1-alpha) * x2
    ym = alpha * y1 + (1-alpha) * y2
    return xm, ym
def update(frame):
    #this function will first decode which 2 polygons to morph and stores
```

```
    #the morphed image in xdata and ydata
    global upflag
    if upflag == 1 :
        temp2 = []
        for i in range(no_of_sides - 3) :
            temp2 = np.append(temp2 , np.linspace(i,i + 1,frame_max))
        side_current = 3
        for f in temp2:
            if frame == 0 :
                break
            if f == 0 :
                continue
            if f == int(f) :
                side_current = int(f) + 3
            if f == frame:
                break
        if side_current >= no_of_sides:
            side_current = no_of_sides - 1
            #print('came')
            upflag = 0
        xdata, ydata = morph(polygons_x_coordinates[side_current - 2],␣
↪polygons_y_coordinates[side_current - 2],␣
↪polygons_x_coordinates[side_current - 3],␣
↪polygons_y_coordinates[side_current - 3], frame  - side_current +␣
↪3,side_current)
    else :
        temp3 = []
        for i in reversed(range(no_of_sides - 3)):
            i += 1
            temp3 = np.append(temp3 , np.linspace(i,i - 1,frame_max))
        side_current = no_of_sides
        for f in temp3:
            if frame == 5 :
                break
            if f == 5 :
                continue
            if f == int(f) :
                side_current = int(f) + 3
            if f == frame:
                break
        if side_current < 4 :
            side_current = 4
        xdata, ydata = morph(polygons_x_coordinates[side_current - 3],␣
↪polygons_y_coordinates[side_current - 3],␣
↪polygons_x_coordinates[side_current - 4],␣
↪polygons_y_coordinates[side_current - 4], frame - side_current +␣
↪4,side_current)
```

```
    ln.set_data(xdata, ydata)
    return ln,

no_of_sides =  8
size1 = factorial(no_of_sides)
t = np.linspace(0*np.pi/4, 8*np.pi/4, size1)
if len(t) % factorial(no_of_sides) != 0:
    raise BaseException("Number of points should be multiple of␣
 ↪{factorial(no_of_sides)}...")
def Polygon_generator(radius,angle,no_of_sides):
    """returns the x and y coordinates for a n sided polygon

    :radius: the radius of the circum circle of the polgon
    :angle: a list of angles at which the x and y coordinates have to be␣
 ↪calculated
    :returns: two lists containing the x and y coordinates

    """
    n = int(len(angle)/no_of_sides)
    x_polygon = []
    y_polygon = []
    for i in range(no_of_sides):
        for j in angle[n*i:n*(i+1)] :
            rad_eff = radius*(np.sin(2*np.pi/no_of_sides))/(np.sin(j - 2*np.
 ↪pi*(i)/no_of_sides) + np.sin(2*np.pi/no_of_sides - j + 2*np.pi*(i)/
 ↪no_of_sides))
            x_polygon = np.append(x_polygon,rad_eff*np.cos(j))
            y_polygon = np.append(y_polygon,rad_eff*np.sin(j))
    return x_polygon,y_polygon

#print(f"Square: {np.shape(x_polygon)}")
radius = 1
polygons_x_coordinates = []
polygons_y_coordinates = []
for i in range(no_of_sides - 2):
    polygons_x_coordinates.append([0])
    polygons_y_coordinates.append([0])
    polygons_x_coordinates[i], polygons_y_coordinates[i] =␣
 ↪Polygon_generator(1,t,i + 3)
frame_max = 50
#plt.plot(polygons_x_coordinates[5],polygons_y_coordinates[5])
#plt.show()
temp = []
upflag = 1
for i in range(no_of_sides - 3):
    temp = np.append(temp ,np.linspace(i,i + 1,frame_max))
```

```
for i in reversed(range(no_of_sides - 3)):
    i += 1
    temp = np.append(temp ,np.linspace(i,i - 1,frame_max))
ani = FuncAnimation(fig, update, frames=temp, init_func=init, blit=True,␣
 ↪interval=10, repeat=False)
plt.show()
```



1 Different approach

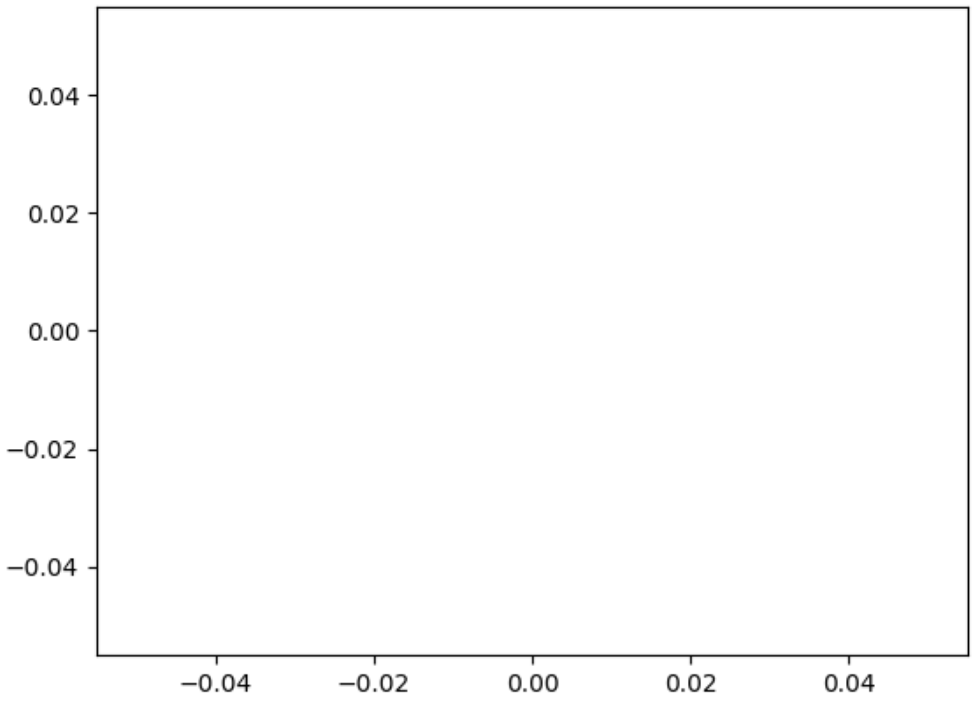
In the previous approach we were morphing every point in a n sided figure to every point in n+1 sided figure(Both figures have equal number of total points), Now we only take the corners of the initial and final figure and morph only the corners to get a transition corner point in the circum circle itself and draw straigh line between these transition points to get a smooth polygon transition

[13]:
```
def circle(t):
    return np.cos(t), np.sin(t)

def morph2(alpha,side_current,upflag,change):
    if upflag == 0 and change != 1 :
        side_current -= 1
```

```
    t = []
    t = np.append(t,np.linspace(0,np.pi*2,side_current + 1))
    x4,y4 = circle(t)
    t = []
    t = np.append(t,np.linspace(0,np.pi*2,side_current + 2))
    x2,y2 = circle(t)

    xm = np.zeros(2*len(x4))
    ym = np.zeros(2*len(x4))
    #taking a transition point on the circum-circle now
    for i in range(len(x4)):
        xm[2*i] = (alpha*(x2[i]) + (1-alpha)*(x4[i]))
        xm[2*i + 1] = (alpha*(x2[i + 1]) + (1-alpha)*(x4[i]))
        ym[2*i] = (alpha*(y2[i]) + (1-alpha)*(y4[i]))
        ym[2*i + 1] = (alpha*(y2[i + 1]) + (1-alpha)*(y4[i]))
    return xm, ym
def update2(frame):
    global upflag
    #print(f'{upflag}')
    change = 0
    if upflag == 1 :
        temp2 = []
        for i in range(no_of_sides - 3) :
            temp2 = np.append(temp2 , np.linspace(i,i + 1,frame_max))
        side_current = 3
        for f in temp2:
            if frame == 0 :
                break
            if f == 0 :
                continue
            if f == int(f) :
                side_current = int(f) + 3
            if f == frame:
                break
        if side_current >= no_of_sides:
            side_current = no_of_sides - 1
            #print('came')
            upflag = 0
            change = 1
        xdata, ydata = morph2(frame  - side_current +␣
↪3,side_current,upflag,change)
    else :
        temp3 = []
        for i in reversed(range(no_of_sides - 3)):
            i += 1
            temp3 = np.append(temp3 , np.linspace(i,i - 1,frame_max))
        side_current = no_of_sides
```

```
        for f in temp3:
            if frame == 5 :
                break
            if f == 5 :
                continue
            if f == int(f) :
                side_current = int(f) + 3
            if f == frame:
                break
        if side_current < 4 :
            side_current = 4
        xdata, ydata = morph2(frame - side_current +␣
 ↪4,side_current,upflag,change)

    ln.set_data(xdata, ydata)
    return ln,
```

[15]:
```
fig, ax = plt.subplots()
xdata, ydata = [], []
ln, = ax.plot([], [], 'r')

no_of_sides =  8
size1 = factorial(no_of_sides)
t = np.linspace(0*np.pi/4, 8*np.pi/4, size1)
if len(t) % factorial(no_of_sides) != 0:
    raise BaseException("Number of points should be multiple of␣
 ↪{factorial(no_of_sides)}")


#print(f"Square: {np.shape(x_polygon)}")
radius = 1
frame_max = 100

temp = []
upflag = 1
for i in range(no_of_sides - 3):
    temp = np.append(temp ,np.linspace(i,i + 1,frame_max))
for i in reversed(range(no_of_sides - 3)):
    i += 1
    temp = np.append(temp ,np.linspace(i,i - 1,frame_max))
ani2 = FuncAnimation(fig, update2, frames=temp, init_func=init, blit=True,␣
 ↪interval=10, repeat=False)
plt.show()
```


0.04
0.02
0.00
−0.02
−0.04
−0.04
−0.02
0.00
0.02
0.04

[ ]: