EE2703 - Week 5

Hardik Gagrani, EE21B047

March 7, 2023

VERTEX SPLITTING:

[7]:
```
%matplotlib ipympl
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import math

def poly(n):
    # n = number of sides of a polygon
    x = []
    y = []
    theta = 2 * np.pi / n
    for i in range(n):
        x.append(np.cos(theta * i))
        y.append(np.sin(theta * i))
        x.append(np.cos(theta * i))
        y.append(np.sin(theta * i))
    x.append(x[0])
    y.append(y[0])

    x = np.array(x)
    y = np.array(y)

    return x, y
```

The `poly(n)` function generates the coordinates of a regular polygon with n sides, represented as a closed path. It takes a single argument n which is the number of sides of the polygon. I am using the roots of unity method to calculate the coordinates of the respective polygon lying on the unit circle.

[8]:
```
def mapping(x, y):
    xprime = []
    yprime = []

    a = len(x) - 1
    for i in range(0, a, 2):
        xprime.append(x[i])
```

```
        yprime.append(y[i])
        xprime.append(x[i + 2])
        yprime.append(y[i + 2])

    popx = xprime.pop(-2)
    popy = yprime.pop(-2)

    xprime = np.array(xprime)
    yprime = np.array(yprime)

    return xprime, yprime
```

Since I am using vertex splitting algorithm to make the animation work. So for this I need to map the coordinates of next polygon to that of current polygon in such a way that while morphing correct vertex splitting will take place. As we know in vertex splitting a vertex splits into 2 vertices, so I have a doubly generated coordinates of current polygon. For eg, if `a`, `b`, `c` are coordinates of triangle in anticlockwise order, so for vertex splitting I am creating a array which stores **[a, a, b, b, c, c, a]** such elements.

After that I map the coordinates of next polygon in such a way that **[a, a, b, b, c, c, a]** goes to their respective places. Lets say `x`, `y`, `z`, `w` are coordinates of square in anticlockwise order. So here first `a` will go to `x`, second `a` to `y`. Similarly first `b` to `y`, second `b` to `z` and first `c` to `z` and second `c` to `w` and the last `a` goes to `x` to ensure a closed loop is formed.

This way the mapped array for **[a, a, b, b, c, c, a]** we get is **[x, y, y, z, z, w, x]**. This process is generalised in this `mapping` function for a given x, y array of current polygon coordinates.

[9]:
```
fig, ax = plt.subplots()
xdata, ydata = [], []
ln, = ax.plot([], [], 'r')

def init():
    ax.set_xlim(-1.2, 1.2)
    ax.set_ylim(-1.4, 1.4)
    return ln,

def morph(x1, y1, x2, y2, alpha):
    xm = alpha * x1 + (1-alpha) * x2
    ym = alpha * y1 + (1-alpha) * y2
    return xm, ym

def animate(n):
    x, y = poly(n)
    xprime, yprime = poly(n + 1)
    xmap, ymap = mapping(xprime, yprime)

    return x, y, xmap, ymap
```

Here `init` function initialises the animation and creates the limits of x and y coordinates which is to be displayed on graph.

The `morph` function is typically used to generate a smooth transition between two sets of points, such as two different shapes or positions of an object. By calling `morph` with increasing alpha values in each frame of the animation, the output points transition smoothly from the first to the second set, creating the illusion of motion or transformation.

The `animate` function just generates 4 arrays, in which two of them are the doubly generated coordinates of current polygon an other two arrays are the mapped coordinates for the next polygon to the current polygon

[10]:
```
n = 8
def update(frame):
    count = 1
    shape_count = 3
    if count == 1:
        for i in range(3, n):
            if (math.ceil(frame)) == count:
                x, y, xmap, ymap = animate(shape_count)
                xdata, ydata = morph(xmap, ymap, x, y, frame - (count - 1))
                ln.set_data(xdata, ydata)
            count += 1

            shape_count += 1

    if count == n-2:
        for i in range(3, n):
            if math.ceil(frame) == count:
                x, y, xmap, ymap = animate(shape_count - 1)
                xdata, ydata = morph(x, y, xmap, ymap, frame - (count - 1))
                ln.set_data(xdata, ydata)
            count += 1
            shape_count -= 1

    return ln,
```

Here `update` funtion is the final step which will animate and change one polygon to another. I am doing this continuous conversion of polygons by taking each conversion in one frame and at the end the total number of frames which are used becomes 2*n where n is the maximum number of sides of polygon specifed earlier only. I am maintaining the frame count and shape count by introducing two variables `count` and `shape_count` which will ensure smooth transition among shapes

[11]:
```
ani = FuncAnimation(fig, update, frames=np.linspace(0, 2*n, 512),␣
↪init_func=init, blit=True, interval=10, repeat=False)
plt.show()
```

Finally this line of code will give us the required `animated output`. Here I am specifying the nmuber of frames as `2*n` as I said earlier each conversion is happening under 1 frame, so total number of frames I require is `n + n - 3`. Subtracting 3 because starting shape is triangle, that's why!