# Code journal: line flowers

03.06.2020
*Mariana Ávalos Arce*

## 1 From a circle to a flower

The flower begins with the logic of a circle drawn using polar coordinates. Usually, one loops for each radius until the maximum and then paints a black pixel for each of the desired angles. This time, I inverted the logic: first finish a complete line and then move to the next angle. In this way, the beginning algorithm looks like the following:

```
num_lines = 6
radius = 20
angle_step = 360.0 / num_lines

for i in range(num_lines):
    curr_angle = i * angle_step - (angle_step / 2.0)
    for r in range(radius):
        rads = curr_angle * (3.1416 / 180.0)
        x = r * math.cos(rads)
        y = r * math.sin(rads)
        x += w / 2.0
        y = h / 2.0 - y
        img[y, x] = [0, 0, 0]
```

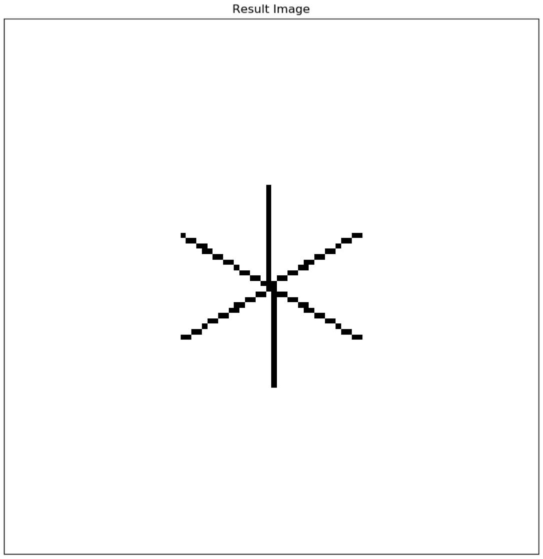

Figure 1: Circle with 6 spikes

After that, what I defined a height differential, which is called **dh** in the code. It is a small change to be applied to the next radius, either added or substracted, let's say 5. Whether this change is added or taken off of the next radius will be defined with a random number between 0 or 1. Therefore, in the outer loop, right before entering to the next radius, I calculate the new radius based on the accumulation of the change with the last radius.

```
num_lines = 60
radius = 50
angle_step = 360.0 / num_lines
last_r = radius
dh = 5

for i in range(num_lines):
    curr_angle = i * angle_step - (angle_step / 2.0)
    rand = randrange(2)
    last_r += dh if rand == 0 else -dh
    for r in range(last_r):
        rads = curr_angle * (3.1416 / 180.0)
        x = r * math.cos(rads)
        y = r * math.sin(rads)
        x += w / 2.0
        y = h / 2.0 - y
        img[y, x] = [0, 0, 0]
```

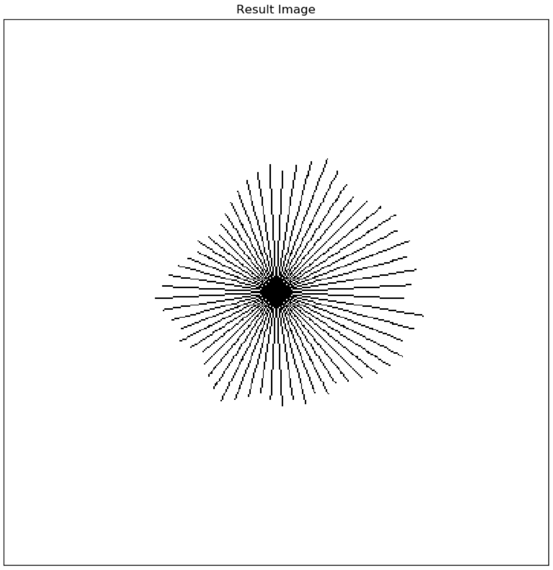

Figure 2: Circle with varying radius

Then, the next thing I added was two constant parameters, **iterations** and **translation speed**, which defined the aesthetic basically. The translation is only in x axis and the factor multiplies the radians, so that translation varies also with time.

```
num_lines = 150
radius = 50
angle_step = 360.0 / num_lines
last_r = radius
dh = 5
iterations = 5
translation_speed = 2

for i in range(num_lines * iterations):
    curr_angle = i * angle_step - (angle_step / 2.0)
    rand = randrange(2)
    last_r += dh if rand == 0 else -dh
    for r in range(last_r):
        rads = curr_angle * (3.1416 / 180.0)
        x = r * math.cos(rads) + translation_speed * rads
        y = r * math.sin(rads)
        x += w / 2.0
        y = h / 2.0 - y
        img[y, x] = [0, 0, 0]
```

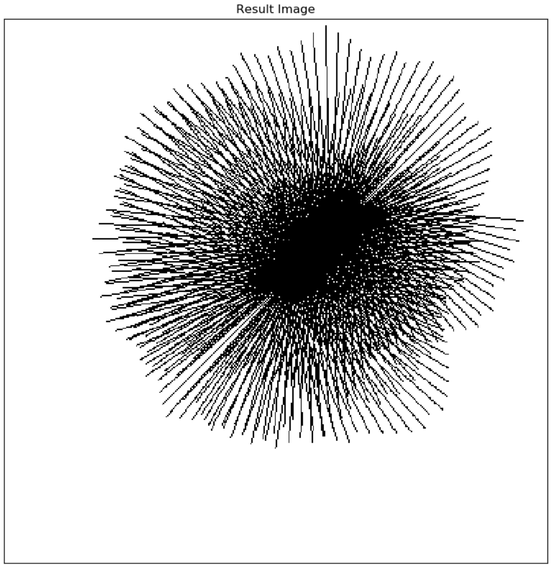

Figure 3: Circle with multiple iterations and translation

Finally, the last thing I did was paint a line that unites the current line's tip with the last line's tip. For this, I stored in **last point** the coordinates of the last line painted. At the end of the new line, and right before storing it again, I paint the line using opencv line function, for simplicity.

```
num_lines = 100
radius = 50
angle_step = 360.0 / num_lines
```

```
last_r = radius
dh = 5
iterations = 5
translation_speed = 2
last_point = (0, 0)

for i in range(num_lines * iterations):
    curr_angle = i * angle_step - (angle_step / 2.0)
    rand = randrange(2)
    last_r += dh if rand == 0 else -dh
    for r in range(last_r):
        rads = curr_angle * (3.1416 / 180.0)
        x = r * math.cos(rads) + translation_speed * rads
        y = r * math.sin(rads)
        x += w / 2.0
        y = h / 2.0 - y
        img[y, x] = [0, 0, 0]

        if i > 0 and i < (num_lines * iterations) and r == (last_r -1):
            img = cv2.line(img, (x, y), last_point, (0, 0, 0), 1)
        if r == (last_r - 1):
            last_point = (x, y)
```

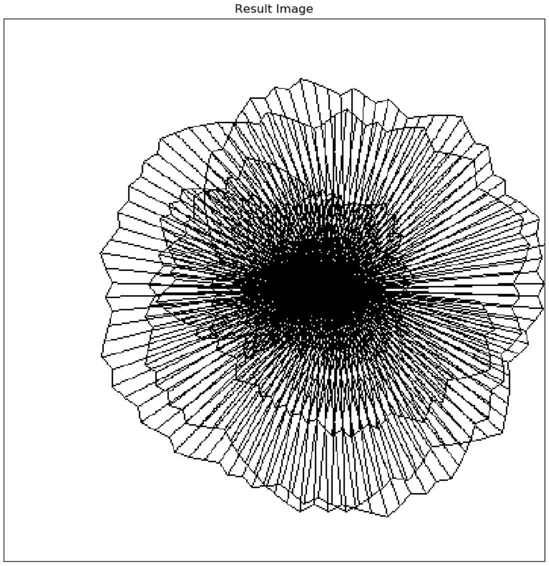

Figure 4: Circle with each line united to the last