Instituto Politécnico Nacional

Escuela Superior de Cómputo

Evolutionary Computing

Laboratory session #05: Fractals

Student: Vargas Romero Erick Efrain

Professor: Rosas Trigueros Jorge Luis

Practice completion date: November 3, 2021

Report delivery date: October 14, 2021

1 Theoretical framework

1.1 Fractals

Many object in the nature can be created by applying the concept of clasical geometry like lines, circles, conic sections, polygons, spheres, quadratic surface and so on. There are various objects of nature which can not be modelled by applying Euclidean geometry, hence there is need to deal with such complicated and irregular object which can only be constructed by *fractal* geoetry [1].

To generate such complicated object iteration process is required which is called iterated function system. The main property of every *fractal* object is self similarity. Upon magnification of a *Fractal*, we can find subsetsof it that look like the whole figure. If we zoom a picture of a mountain and again and again we still see a mountain. This is the self similarity of a *fractal* [1].

1.2 Properties of a fractal

1. Self similarity.
2. Fractional dimension.
3. Fractals are non-differentiable.
4. Fractals has infinite length, yet they enclose infinite area.

Fractals are figures with an infinite amount of detail. After magnifying fractal it looks more complicated as they were without magnification.

Self similarity

Fractals are self similar an any level of magnification; many things around us look the same way no matter how you magnify them. When parts of some object are similar to the entire object, we call it *self-similarity*.

According to perfect self similarity each of these fractals is composed of smaller versions of itself. When fractal with perfect self similarity is magnified, they turn out to be identical to the entire picture.

Sometimes, however, the object is not so perfectly self-similar. In such case, it is called approximate self-similarity.

Dimension

A point has a dimension of 0, a line has a dimension of 1, a square is 2-dimensional, and a cube is 3-dimensional. Such dimension is called *topological dimension.*

Fractal has fractional dimension. Dimension of object means how the object feels the space. Other types of dimension can be Box dimension, Hausdroff dimension.

A Line in 1 dimensional with magnification factor 2, will give 2 identical line segments. Let's use a variable D for dimension, e for magnification, and N for the number of identical shapes. Now take a 2-dimensional square and a triangle. With the magnification of 2, you get 4 identical shapes in both of them. Finally, take a 3-dimensional cube. Again, magnify it 2 times. Now, you will get 8 identical cubes. To calculate dimension of a fractal objects following formula can be used eq. (1.1).

$$D = \frac{log(N)}{log(e)} \tag{1.1}$$

To Calculate dimensions ,take a look at another fractal, called the Koch Snowflake img. 1.1. In it, you can see four identical snowflakes (N = 4). Each of them is 1/3 of the entire snowflake, so e = 3.

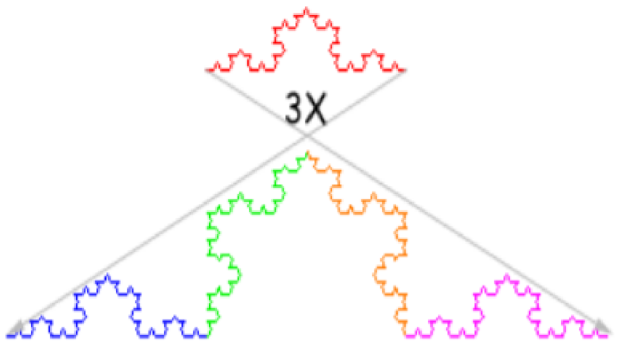


Figure 1.1: Coach curve

Calculating the fractal dimension, we get: $D = \frac{log(4)}{log(3)} = 1.26$. The dimension is a fraction — something you can never see in standard Euclidean geometry [1].

2 Material and equipment

The necessary material for this practice is:

- A computer with the latest *Python* stable version installed
- A text editor

Or is possible to use the google site `https://colab.research.google.com/` that allows us to use a virtual machine with an *Ubuntu* operative system with *Python* installed.

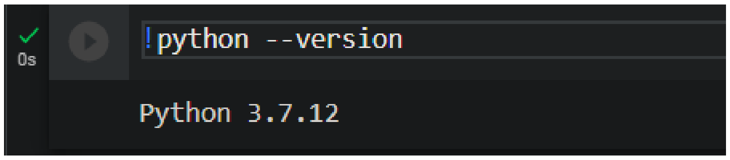
3 Practice development

3.1 Fractals

To develop this practice we used the Google platform called *Colab* as this platform uses a virtual machine with linux (specifically Ubuntu) we can install some packets and of course we can verify if we have already *Python* installed. To check it we must use the command:

```
python --version
```

If we run this command in our linux terminal using *Colab* we can see the next as the result:



```
!python --version

Python 3.7.12
```

Figure 3.1: Verifying python version

For this practice we used *cv2*, it is a library focused in solve computer vision problems. This library help us to draw the landscape.

Based in the class video of October, 8th and in the code provided by the professor Trigueros, I build a landscape of mountains and trees in winter (at least I tried it). First of all I created some functions to make easier to add elements in the landscape, the first function that I created was called *sun* and as you can imagine this function creates a very interesting figure as you can see in fig. 3.2.

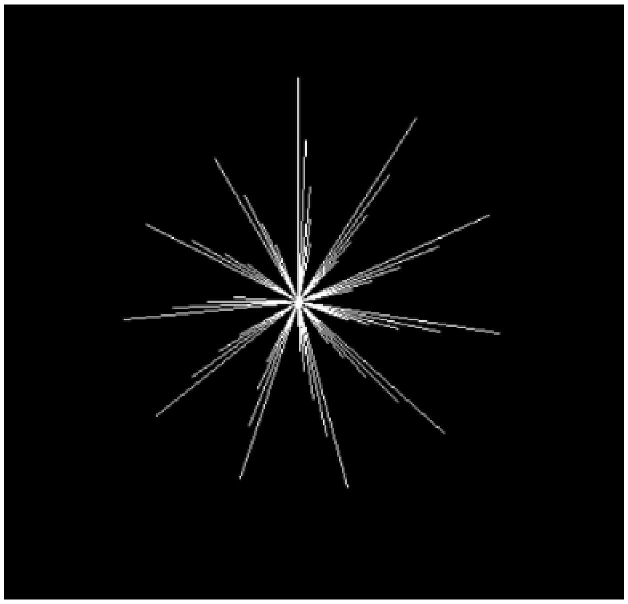


Figure 3.2: Sun or star figure

In code we get the next as the result:

```python
def sun(img,x,y,l,a,sl,da,n):
    if n == 0:
        return
    x2 = x + l
    y2 = y
    ar = math.radians(a)
    coseno = math.cos(ar)
    seno = math.sin(ar)
    xrot = (x2 - x) * seno + (y2 - y) * coseno
    yrot = (x2 - x) * coseno - (y2 - y) * seno
    x2 = xrot + x
    y2 = yrot + y
    if random.randrange(0, 2) == 0:
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), (0, 255, 247), (1))
    else:
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), (0, 220, 200), (1))
    sun(img, x, y, l * sl, a - da, sl, da, n - 1)
    return
```

The next function is called *drawMountain* and basically we generate something like the img. 3.3 but we create at the same time two lines, the first one is the background of the mountain and the second one is the border, the differences between both lines are basically two, the first one is the color and the second one is the line thickness.

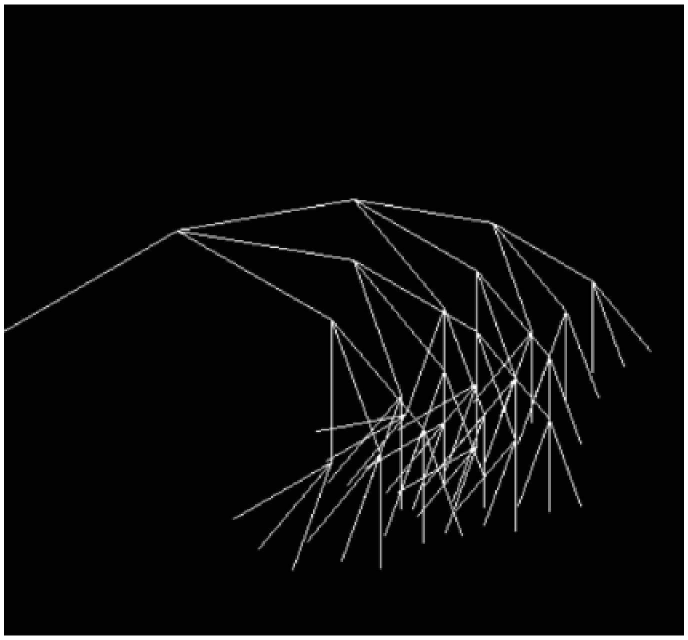


Figure 3.3: Mountain figure

A mountain is draw with the next code:

```python
def drawMountain(img,x,y,l,a,sl,da,n):
    if n == 0:
        return
    x2 = x + l
    y2 = y
    ar = math.radians(a)
    coseno = math.cos(ar)
    seno = math.sin(ar)
    xrot = (x2 - x) * seno + (y2 - y) * coseno
    yrot = (x2 - x) * coseno - (y2 - y) * seno
    x2 = xrot + x
    y2 = yrot + y
    cv2.line(img, (int(x),int(y)), (int(x2),int(y2)), (255, 229, 204), (35))
    cv2.line(img, (int(x),int(y)), (int(x2),int(y2)), (255,255,255), (1))
    if(y > 280):
        return
    drawMountain(img, x2, y2, l + sl, a - da, sl, da, n - 1)
    drawMountain(img, x2, y2, l + sl, a - 1.7 * da, sl, da, n - 1)
    return
```

Finally, the most interesting function is the called *drawTree*. This function generates as you can imagine a tree figure and in img [**?**] you can see the result without any modification.

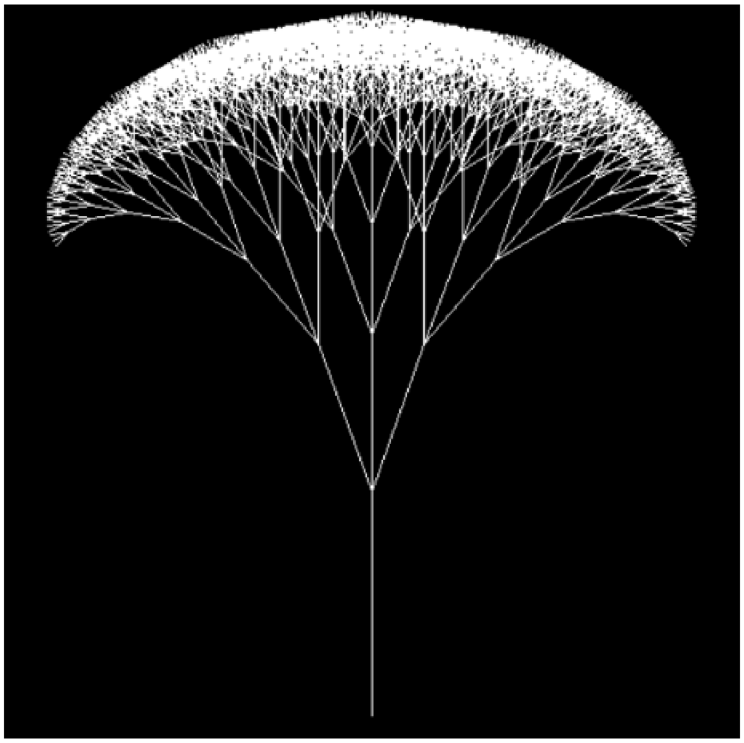


Figure 3.4: Tree figure

This function is very similar to the previous ones but as you can see we receive an extra paramater called *isWood*, basically as you can suppose it is a flag that indicates if we are in the first recursive call of the figure it is with purposes to use different colors. Additionally, each tree that is generated theoretically is different of the rest because we are using random numbers to create each tree, because as we known in nature we do not have the same tree, all of them are different so in order to make it more interesting we decided to use this characteristic of the real world.

```
def drawTree(img, x, y, l, a, sl, da, n, isWood):
    if n == 0:
        return

    x2 = x + l
    y2 = y
    ar = math.radians(a)

    coseno = math.cos(ar)
    seno = math.sin(ar)
    xrot = (x2 - x) * seno + (y2 - y) * coseno
    yrot = (x2 - x) * coseno - (y2 - y) * seno

    x2 = xrot + x
    y2 = yrot + y
```

```python


    if isWood:
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), (19, 69, 139), n)
    else:
        leafColorPosition = random.randrange(0, 3)
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), leafColors[
    leafColorPosition], 2)

    if n <= 8:
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), (255, 255, 255), 1)

    treeType1 = random.randrange(0, 2)
    treeType2 = random.randrange(0, 2)

    #First branch
    if treeType1 == 0:
        drawTree(img, x2, y2, l * sl, a - da, sl * random.random(), da * (random.
    random() + 1), n - 1, False)
    else:
        drawTree(img, x, y, l * sl, a - da, sl * random.random(), da * (random.
    random() + 1), n - 1, False)

    #Second branch
    if treeType2 == 0:
        drawTree(img, x2, y2, l * sl, a + da, sl, da, n - 1, False)
    else:
        drawTree(img, (x + x2) / 2, (y + y2) / 2, l * sl, a + da, sl, da, n - 1,
    False)

    #Third branch
    drawTree(img, x2, y2, l * sl, a, sl, da, n - 1, False)

    return
```

Here you have the entire code of this practice if you want to check how it works:

```python
import cv2
import random
import numpy as np
from google.colab.patches import cv2_imshow
import math

width = 1500
height = 640

leafColors = [(0, 100, 0),(0,128,0),(34,139,34)]

def sun(img,x,y,l,a,sl,da,n):
    if n == 0:
        return
    x2 = x + l
    y2 = y
    ar = math.radians(a)
    coseno = math.cos(ar)
    seno = math.sin(ar)
    xrot = (x2 - x) * seno + (y2 - y) * coseno
    yrot = (x2 - x) * coseno - (y2 - y) * seno
    x2 = xrot + x
    y2 = yrot + y
    if random.randrange(0, 2) == 0:
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), (0, 255, 247), (1))
    else:
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), (0, 220, 200), (1))
    sun(img, x, y, l * sl, a - da, sl, da, n - 1)
    return

def drawMountain(img,x,y,l,a,sl,da,n):
    if n == 0:
        return
    x2 = x + l
    y2 = y
    ar = math.radians(a)
    coseno = math.cos(ar)
    seno = math.sin(ar)
    xrot = (x2 - x) * seno + (y2 - y) * coseno
    yrot = (x2 - x) * coseno - (y2 - y) * seno
    x2 = xrot + x
    y2 = yrot + y
    cv2.line(img, (int(x),int(y)), (int(x2),int(y2)), (255, 229, 204), (35))
    cv2.line(img, (int(x),int(y)), (int(x2),int(y2)), (255,255,255), (1))
    if(y > 280):
        return
    drawMountain(img, x2, y2, l + sl, a - da, sl, da, n - 1)
    drawMountain(img, x2, y2, l + sl, a - 1.7 * da, sl, da, n - 1)
    return

def drawTree(img, x, y, l, a, sl, da, n, isWood):
    if n == 0:
        return

    x2 = x + l
    y2 = y
    ar = math.radians(a)

    coseno = math.cos(ar)
    seno = math.sin(ar)
    xrot = (x2 - x) * seno + (y2 - y) * coseno
    yrot = (x2 - x) * coseno - (y2 - y) * seno

    x2 = xrot + x
    y2 = yrot + y


    if isWood:
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), (19, 69, 139), n)
    else:
        leafColorPosition = random.randrange(0, 3)
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), leafColors[
    leafColorPosition], 2)

    if n <= 8:
        cv2.line(img, (int(x), int(y)), (int(x2), int(y2)), (255, 255, 255), 1)

    treeType1 = random.randrange(0, 2)
    treeType2 = random.randrange(0, 2)

    #First branch
    if treeType1 == 0:
        drawTree(img, x2, y2, l * sl, a - da, sl * random.random(), da * (random.random() + 1), n - 1, False)
    else:
        drawTree(img, x, y, l * sl, a - da, sl * random.random(), da * (random.random() + 1), n - 1, False)

    #Second branch
    if treeType2 == 0:
        drawTree(img, x2, y2, l * sl, a + da, sl, da, n - 1, False)
    else:
        drawTree(img, (x + x2) / 2, (y + y2) / 2, l * sl, a + da, sl, da, n - 1, False)

    #Third branch
    drawTree(img, x2, y2, l * sl, a, sl, da, n - 1, False)

    return

img = np.zeros((height, width, 3), dtype='uint8')
cv2.rectangle(img, (0,0), (width, height), (150,44,0), (-1))

cv2.circle(img,(int(260), int(600)), int(400), (45, 82, 100), -1)
cv2.circle(img,(int(1200), int(600)), int(400), (45, 82, 100), -1)
cv2.circle(img,(int(1000), int(521)), int(400), (45, 82, 100), -1)
cv2.circle(img,(int(685), int(300)), int(100), (45, 82, 100), -1)

cv2.circle(img,(int(50), int(40)), int(50), (0,128,255), -1)
sun(img, 50, 40, 200, 120, 0.99, 33, 150)

drawMountain(img, 0, 280, 170, 120, 0.8, 20, 15)
drawMountain(img, 555, 280, 150, 120, 0.8, 20, 2)
drawMountain(img, 830, 180, 150, 120, 0.8, 20, 8)

for i in range(0, 7):
    drawTree(img, width - (100 + (200 * i)), height, random.randrange(130, 201), 180, 0.7, 20, random.randrange(9, 14), True)

cv2.line(img, (int(0), int(height)), (int(width), int(height)), (255, 255, 255), 10)

cv2_imshow(img)
```

4 Screens, graphs and diagrams

Using the code that we described previously we can create interesting things, for example this landscape img. 4.1. As you can see, we created several trees, some mountains and the sun but we used some colors, to make it so much interesting.

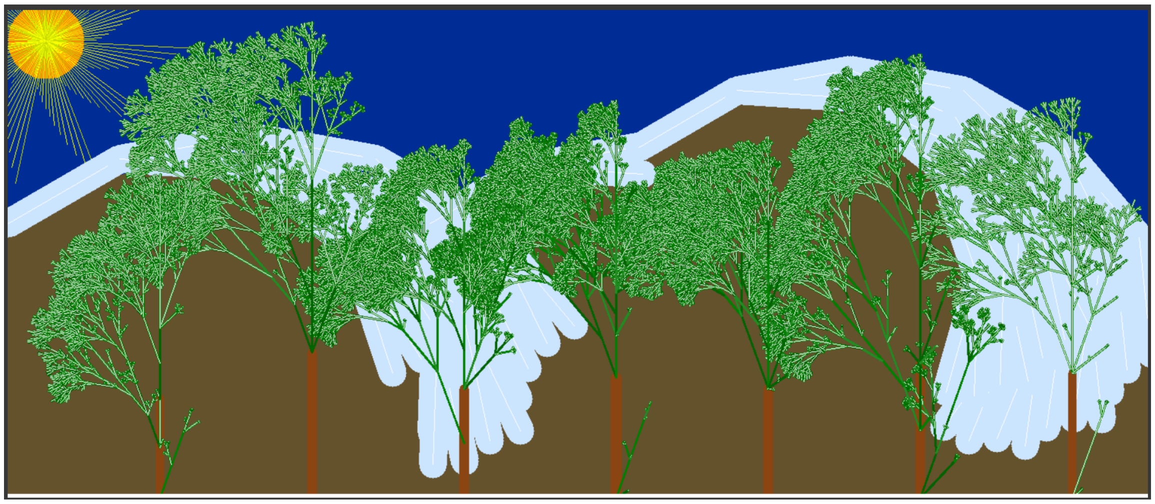


Figure 4.1: Landscape

5
Conclusions

Fractals are interesting figures that are in nature and a lot of times we are not aware of them. Fractals have lots of interesting properties and applications. In our case understand how can we create some fractals was awesome because the concept of recursion is very important in the process of creating of a fractal merging fractals with genetic algorithms come to my mind the concept of evolution, because at least that concept and each recursion are pretty simmilar. Something cool about this practice was use python to display the result, normally (at least in my experience) is too complicated create a graphical application (to draw) and in python has been super simple and you do not need a lot of lines of code.

Bibliography

[1] A. Garg, A. Agrawal and A. Negi *A review on Natural Phenomenon of Fractal Geometry*. Article in International Journal of Computer Applications. 2014. Accessed on November 2nd, 2021. Available online: `https://www.researchgate.net/publication/262374722_A_Review_on_Natural_Phenomenon_of_Fractal_Geometry/link/00b49537709e49a553000000/download`