UNIVERSITY OF SILESIA
FACULTY OF SCIENCE AND TECHNOLOGY

Antonio Brimes Romero

Fractal generator aplication

(Fractal Methods in Computer Graphics)

Sosnowiec 2023

1. Introduction

In this project we implement a fractal generator by implementing four different algorithm in a python script, in which the user can use a menu, who is implemented too, to choose the type of fractal that he want to see, also, there is an extra option implemented after execute each algorithm, where the program ask the user if he want to save the generated fractal in a '.png' file to see a more detailed image of the chosen one

2. Theory and algorithms

Definition 2.1. lmplemented algorithms

In this case, we implement four algorithms, and some of them have modifications compare to the original one.

- The Julia Set algorithm generates a fractal pattern by iterating a complex quadratic polynomial over each point in the complex plane. Given a complex constant \(c \), the algorithm applies the function $f(z) = z^2$ and then iterates, for $f^0(z) = z^2$ and finally for $f^n(z) = f\left(f^{n-1}(z)\right) + c$ for $n \geq 1$. A point (z) is considered part of the Julia Set if the sequence of iterated values remains bounded, typically within a certain distance from the origin (2) after a set number of iterations.

- Mandelbrot modified algorithm, is a procedure to identify points in the complex plane that belong to the Mandelbrot Set. For each complex number (c), the algorithm iterates the function $(f(z) = z^2 + c)$ starting with $(z = 0)$, and checks if the iterated values remain bounded. If the magnitude of the iterated values does not exceed a certain threshold (2) after a certain number of iterations, the point represented by (c) is part of the Mandelbrot Set, the modification consist in changing the quadratic function substituting it to a squared to n index$((f(z) = z^2 + c) \to (f(z) = z^n + c))$

- Phoenix algorithm, is a pattern generated through iterative mathematical processes. This fractal is derived by iterating a modified version of the complex quadratic polynomial typically used in Julia sets. The iterative formula used is $(z_n = z_{n-1}^2 + c + p \cdot z_{n-2})$,where $(z)$is a complex number, (c) is a complex constant defining the specific form of the fractal, and (p) is another complex parameter that introduces the influence of the second-to-last term in the sequence.

- Multi-step Polynomiograph algorithm, is a method for visualizing the dynamics of polynomial root-finding algorithms. This algorithm generates the patterns by tracing the paths of points as they converge towards the roots of a given polynomial through successive iterations. Each point's trajectory and its eventual convergence (or divergence) to a root is visually represented. The multi-step aspect of the algorithm refers to the use of multiple iterations or layers in the process and using an 'iteratePoint' function to get the number of iterations in the current step and the last point obtained in the iteration process.

Algorithm 1: Julia set algorithm

Input: array julia [height, width], x_range [width], y_range [height], max_iter, c

Output: Approximation of a Julia set

```
1 for i ← 1 to width do
2   x ← x_range[i]
3   for j ← 1 to height do
4       y ← y_range[j]
5       z ← complex(x, y)
6       ic ← 0
7       for k ← 1 to max_iter do
8           z ← z^2 + c
9           if |z| > R then
10              ic ← k
11              break
12    julia[j, i] ← ic
```

Table 2.1. Julia set algorithm variables

Width	*Height*	*x_min, x_max*	*y_min, y_max*	*max_iter*	*c*
1600	*1600*	*-2,2*	*-2,2*	*256*	*-0.8 + 0.156i*

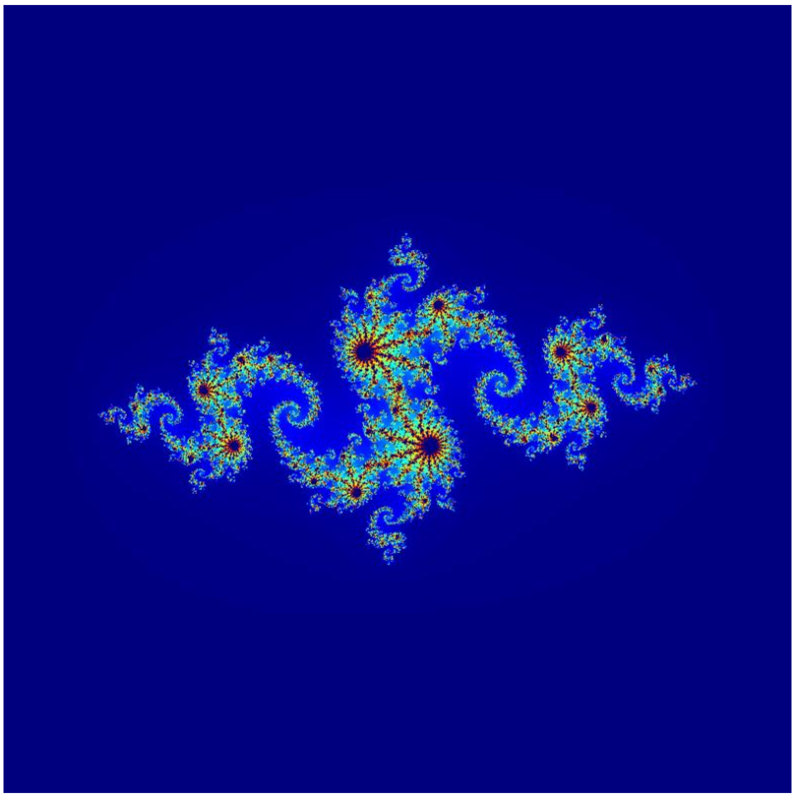

Figure 2.1. Julia set fractal

Algorithm 2: Modified Mandelbrot algorithm

Input: area_real: array [1 ... width], area_imag: array [1 ... height], n: integer (exponent for fractal equation), max_iter: integer (maximum iterations)

Output: Approximation of the modified Mandelbrot set

```
1. for re_index = 1 to width do
2.   for im_index = 1 to height do
3.     c = complex(area_real[re_index], area_imag[im_index])
4.     z = c
5.     ic = 0
6.     for i = 1 to max_iter do
7.         z = z^n + c
```

```
8.          if abs(z) > 2 then
9.              ic = i
10.             break
11.     fractal_image[im_index][re_index] = ic
```

Table 2.2. Modified Mandelbrot algorithm variables

N	*max_iter*	*c*	*R*	*z*	*area_real*	*area_imag*	*Width*	*Height*
2-7	*100*	-	max(|c|, 2)	-	*-2, 2*	*-2, 2*	*1600*	*1600*

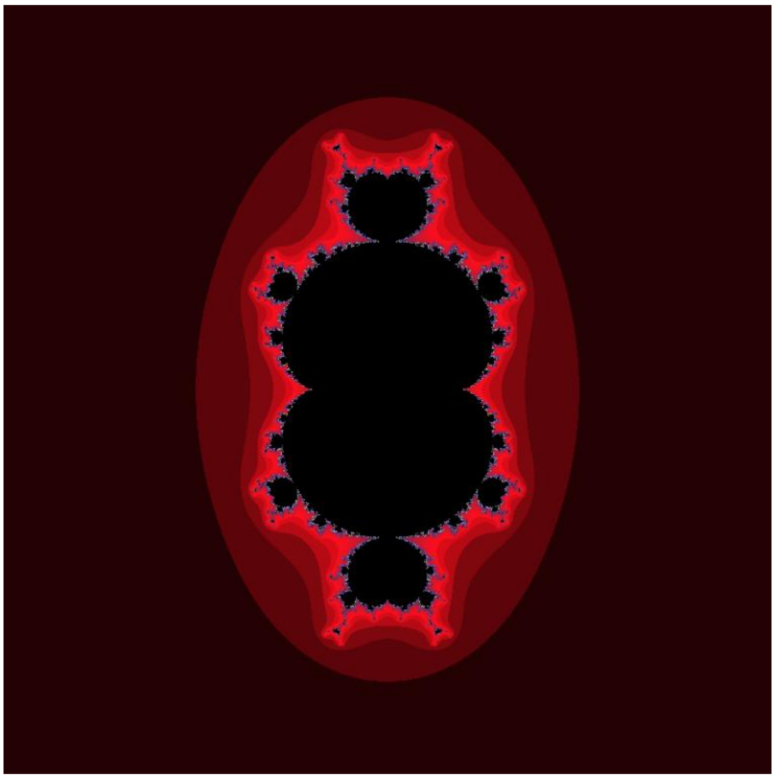
Figure 2.2. Modified Mandelbrot fractal

Algorithm 3: Phoenix algorithm

Input: width, height, x_min, x_max, y_min, y_max, max_iter, c, p, escape_radius
Output: Approximation of the Phoenix set

```
1. Initialize fractal[height][width] to zero
2. for i = 0 to width do
3.    for j = 0 to height do
4.        z = complex(x_min + i*(x_max-x_min)/width, y_min + j*(y_max-y_min)/height)
5.        z_prev = 0
6.        for k = 0 to max_iter do
7.            z_new = z^2 + c + p*z_prev
8.            if abs(z_new) > escape_radius then
9.                fractal[j][i] = k
10.               break
11.           else
12.               z_prev = z
13.               z = z_new
```

Table 2.3. Phoenix algorithm variables

escape_radius	*max_iter*	*c*	*p*	*x_min,x_max*	*y_min,ymax*	*Width*	*Height*
4	*40*	*0.56667+0.5i*	*0.5+0.007i*	*-2,2*	*-2,2*	*800*	*800*

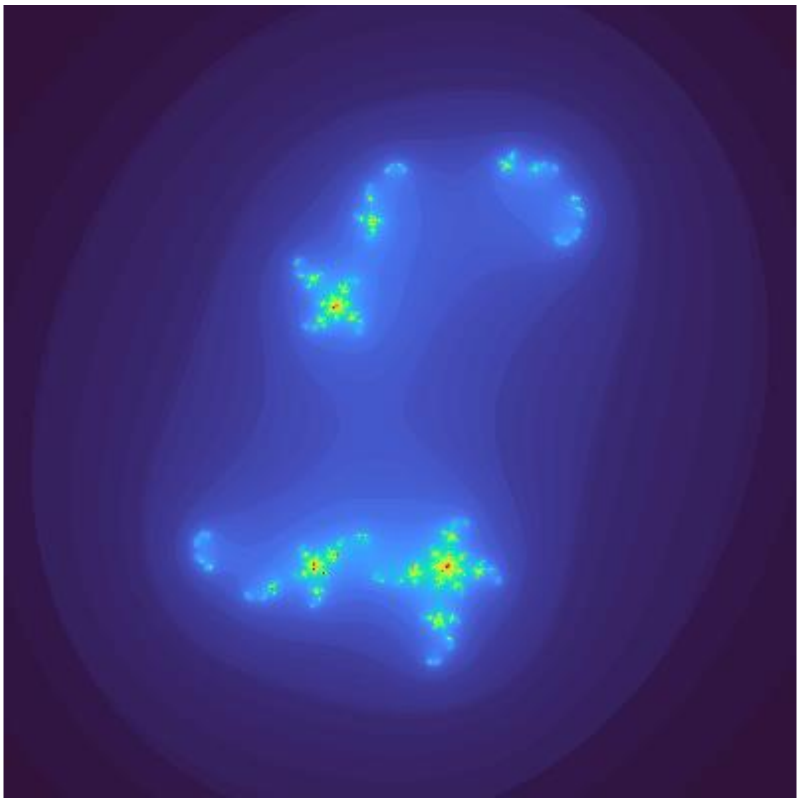
Figure 2.3. Phoenix fractal

Algorithm 4: Multi-step polynomiograph algorithm

Input: area: (xmin, xmax, ymin, ymax), polys: [polynomial functions], iters: [iteration functions], max_iters: [maximum iterations for each polynomial], conv_test: [convergence test functions], area_transforms: [area transformation functions]

Output: The Multi-step polynomiograph

```
1. xmin, xmax, ymin, ymax ← area
2. Set width and height
3. x ← linspace(xmin, xmax, width)
4. y ← linspace(ymin, ymax, height)
5. Z ← x + 1j * y (complex grid)
6. polynomiograph ← zeros([height, width, 3])
7. for each (poly, Iv, M, Cv, transform) in zip(polys, iters, max_iters, conv_tests, area_transforms) do
8.     for i in 0 to height-1 do
9.         for j in 0 to width-1 do
10.            n_iter ← iterate_point(Z[i][j], poly, Iv, Cv, M)
11.            color ← colormap(n_iter/ M)
12.            polynomiograph[i][j] ← [255 * c for c in color[:3]]
13.            break
```

Algorithm 4.1: IteratePoint function

Input: - z0: The initial complex point from which the iteration starts, p: The polynomial function that defines the iteration rule, Iv: The iteration function that applies the polynomial to the current point, Cv: The convergence test function that determines whether the point has escaped or converged, M: The maximum number of iterations to perform.

Output: n_iter: The number of iterations it took for the point to converge or escape, z: The last computed point in the iteration sequence.

```
1. z ← z0
2. for i in 0 to M-1 do
```

```
3.    z ← Iv(z, p)
4.    if Cv(z) then
5.        return (i, z)
6. return (M, z)
```

Table 2.4. Multi-step polynomiograph algorithm variables

area	*polys*	*iters*	*max_iters*	*conv_test*	*area_transforms*
-2,2	*lambda z: z^2 + (0.285+0.01i), … lambda z: z^2 + (-0.4+0.6i)*	*lambda z, p: p(z) for _ in polys*	*[60, 70, 80, 90]*	*lambda z: abs(z)>2 for _ in polys*	*lambda z: z for _ in polys*

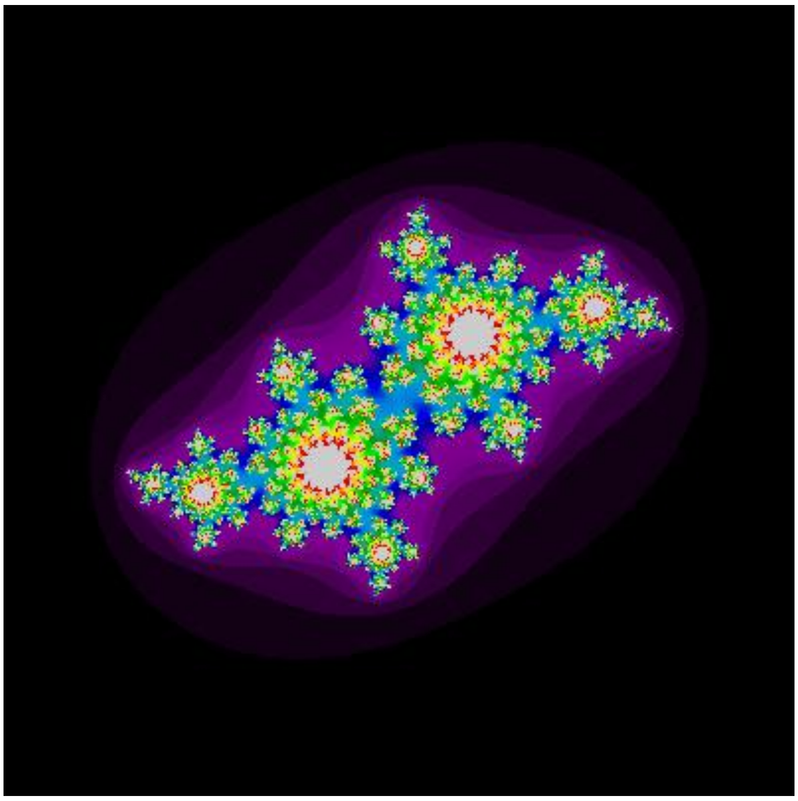
Figure 2.4. Multi-step polynomiograph fractal

3. Description of the program

The program is written in Python and the code utilizes the following libraries:

- sys: This module provides various functions and variables that are used to manipulate different parts of the Python runtime environment.
- numpy: Python library used for working with arrays, a fundamental package for scientific computing in Python.
- matplotlib.pyplot: A library for creating different kind of visualizations in Python.
- os: To work with the folders in order to save the fractal images.

3.1. Program's capabilities

The program can generate and visualize four different kinds of fractals, each with its own algorithm by using a code in which they iterate using several mathematical formulas:

1. Julia Set Fractal: Create the Julia Set fractal, a classic fractal pattern that we obtain iterating by tree loops and applying his function, there can be variations in the final result by changing the parameters.
2. Modified Mandelbrot Fractal: Generates a modified version of a Mandelbrot fractal in which we can choose a number between two and seven, to set it as a power in the Mandelbrot function and

generate the fractal based on this.

3. Phoenix Fractal: This iterates using the quadratic formula and adding feedback from the previous values of the iteration, we can modify the complex number of the formula to change the shape of the fractal.
4. Multi-step Polynomiograph: There we obtain a visual representation of the roots of a sequence of polynomials, and in this case, we use an auxiliar function ‘IteratePoint’.

Each fractal generation algorithm allows the user to save the generated fractal image as a PNG file to see a more details of them.

3.2. Description of the program

1. Starting the Program: Run the script to start the program or execute the ‘.exe’ version of the project(The executable version of the program only runs on mac computer because my computer is a mac and I can only create the EXE version for my operative system, I hope it will not cause you too much inconvenience). The main menu offers four options for fractal generation and an exit option.
2. Choose a Fractal to Generate:
   - Enter `1` to generate a Julia Set fractal.
   - Enter `2` for a Modified Mandelbrot fractal and then enter a number between 2 and 7 to adjust the fractal.
   - Enter `3` to obtain a Phoenix Fractal.
   - Enter `4` for a Multi-step Polynomiograph.
3. Viewing and Saving the Fractal:
   - After choosing an option, the program will run the correspondent algorithm to generate and display the fractal.
   - After that, the program will ask the user to save the fractal as a PNG file in a folder called ‘Fractal Images’ in the same directory as the script or the executable version. Enter `yes` to save it or `no` to skip.
4. Exiting the Program: Enter `0` at the main menu to exit the program

Bibliography

[1] https://www.sciencedirect.com/science/article/pii/S0960077922007366

[2] https://www.ultrafractal.com/help/index.html?/help/formulas/standard/phoenix.html Shoemake,

[3] https://www.researchgate.net/publication/319979740_Fractal_patterns_from_the_dynamics_of_combined