Author : Group « Fractal Patterns »

Mini Algo Project Report

I. Introduction

I.1. About the subject

"A fractal is a never-ending pattern driven by recursion. Fractals are infinitely complex patterns that are self-similar across different scales. They are created by repeating a simple process over and over in an ongoing feedback loop" – according to *fractalfoundation.org*.

Fractals can be found easily in nature: trees, leaves, snowflakes, rivers, etc… They have various applications in a wide array of domains: art, image/graphics design, biological study, image compressing. The study of fractals is often associated to a branch of maths.

I.2. Functionality of the program

The aim of this project is to provide a program allowing the users to discover some of the most representative fractals. Concretely, the user is able to:

- Choose amongst a variety of patterns.
- Study the advancement of the patterns step-by-step (control the generations of advancement).
- Customize some patterns: modify the shape, zoom in and out, change their colors, etc.
- Switch between light mode/dark mode.

II. Problems & Challenges

- Use various tools to draw different geometrical shapes (circles, polygons, vectorised lines, etc)
- Handle mathematical problems: calculating angles, changing coordinate systems, determining points, etc…
- Determine the algorithm that generates fractals from basic geometrical shapes.
- Figure out a way to control the advancement steps.
- Limit the generations to a certain level so that the program does not lag during the interaction with the user while keeping the pattern complex enough (no considerable difference compared to its successive generation).
- Ensure the stability of the program when the user moves from one pattern to another.
- Optimize the interface: making it as large as possible to facilitate observation while making sure it adapts to different screen sizes.
- Allow zooming on a fractal.

- Provide additional functionality to make the interaction more interesting.
- The most challenging part: merge the programs written by different members of the team to form a unified, functional one.

III. Principle of the algorithm & Solutions

- The geometrical shapes can be generated from the code written in first semester TPs: class Courbe, Cercle, Polygone, PolygoneRegulier, Apoint. For advanced manipulation such as a directional point, we extended the existing class (Apoint) to obtain a more suitable one.
- The principle to obtain a fractal is using **recursion**: begin with the first simple shape (generation 0), then draw the very first generation. After that, turn the method into a recursive one by calling itself. Remark: there must always be a base case to escape from the recursion.
- The number of generation is passed as a parameter of type integer to control the step of advancement.
- The patterns are created separately in their own classes (object-oriented) in order to be independent of each other, thereby ensure the stability of the program.
- Apply the knowledge of adaptative graphical interface acquired in first semester to make the program versatile (instead of setting fixed coordinates).

IV. Bibliography

The patterns in the program are selected from google image searching with the keyword: "simple fractal patterns", there is no concrete source to see all of them at the same time. However, the idea is inspired by the sources below:

- What Is A Fractal (and what are they good for)? - MITK12Videos <https://www.youtube.com/watch?v=WFtTdf3I6Ug>
- Mandelbrot zoom 10^227 [1080x1920] - tthsqe12 <https://www.youtube.com/watch?v=PD2XgQOyCCk&t=108s>

V. Structure of data (hierarchy of objects)

The hierachy is defined as below:

1. The geometrical classes from first semester TPs (Courbe, Cercle, Polygone, APoint…) are fundamental classes.
2. Advanced geometrical forms are extended from the above classes. For example: Snail is in fact APoint with an angle as an additional attribute.
3. The fractal patterns are classes based on the geometrical forms above.
4. All of the patterns are instantiated in the class Fractal, which includes the interface.
5. The main class instantiates Fractal.

VI. Suggestion of possible improvements

Due to the limit of time, we put the variety of patterns as our first priority. There are in fact some more functionalities that can be added later on: rotation of figures, multiple patterns in the same window, some more complex patterns, etc…

VII. "Carnet de route" – Timetable

1. First week: Start discovering in detail the principle of generating fractals, observe their evolution via the GIFs, learn how a recursive function works.
2. Second week: select patterns to be generated, discuss to have a mutual convention on the structure of the program, deploy a hierarchy of classes, allocate tasks to each member.
3. Third week: create the first fractal (Sierpenski triangle) and test it with a simple interface. Continue to create KochSnowFlake.
4. Forth week: enrich the interface, create the other patterns, test the program on different laptops.
5. Fifth week: finalize the interface, adding unique extra functionality to each of the pattern. Test the program again on different laptops.