Spiralis Fractalis: the Evolution of Generative Art

Mattia Carolo, Laura Corso, Sebastiano Dissegna, Chiara Camilla Rambaldi Migliore

***Abstract*—One of the main application of AI Systems is in generative art. Following this trend, inspired by the work from Wannarumon et al.[1], we developed *Spiralis Fractalis* a software that automatically generates new fractal images starting from their mathematical form.**

To do this, we resorted to fractals made using IFSs also known as Iterated Function Systems (a system in which images are created following a predetermined set of function which describes the inner features of the fractal itself) and genetic algorithms. This combination resulted in a framework capable of generating different fractals starting from the ones selected by an user.

Moreover, since the simple generation process would give only a binary representation of the fractals, we used the same IFS methodology to achieve colouring on the resulting images.

***Index Terms*—Fractal, IFS, Evolutionary Algorithm, AI, Computational Aesthetics, Evolutionary Art, Interactive Evolutionary Design**

I. Introduction

In recent years AI technologies have been applied with success to different fields such as Computer Vision, Natural Language Understanding, etc. Moreover, these systems, not only have been proved useful in these technical sectors, but also in other context like Art or Poetry, which, until the recent past, have always been considered as human-only domains. Recently, instead, it has been established that AI can serve also as a tool to generate artistic contents that resemble the ones created by human artists.

In particular, narrowing the discussion to **Generative Art**, which refers to "art that in whole or in part has been created with the use of autonomous system"[2], the main technique used nowadays to realize it are *Generative Adversarial Networks*. In fact, in the literature we can find lots of application of these GANs to the art fields. One example for all is Chen and Chen, 2020[3].

Differently, for our project we have decided to approach the Generative Art problem with **Interactive Evolutionary Computation**. This technique uses *Genetic Algorithms* in combination with human subjective judgement. Where Genetic Algorithms are a class of methods that belong to the Evolutionary Algorithms. These try to find the optimal solution to a given problem using concept like selection, population and fitness that classically characterize the biological world and its Darwinian description. Therefore, we have implemented a Genetic Algorithm that, given an initial population of images, through cross-over and mutation produces the offspring population. These new images are then evaluated by a user that, for each, assigns a point between 0 and 100 accordingly to the aesthetic appearance of the picture. Images with higher score will then be propagated in the next generation through *elitism* (i.e. they will be part of the offspring) and, also, they will have a higher probability to become parents for the next generation. We have decided to only use the user evaluation as fitness function without any additional non-user based score.

In addition, inspired by the work of Wannarumon et al.[1], we have decided to produce images using **fractals**, i.e. geometric shapes created repeating in a feedback loop a never-ending pattern. Generally, fractal patterns are extremely familiar since nature is full of them (e.g. trees, rivers, seashells, hurricanes, etc.).

In order to build fractal pictures, always following the the work of Wannarumon et al.[1], we have used the **Iterated Function System** (IFS) encoding which is a set of affine transformations defined by any combination of scaling, rotation, shearing and translation of point sets. An IFS fractal is made up of the union of several copies of itself, each copy being transformed recursively by an affine transformation.

To sum up, in our work we have developed an aesthetic-driven genetic algorithm for the generation of fractal images with an user-centered scoring system.

II. Methodologies

A. Genetic Algorithm

Before talking about the techniques used in our evolutionary algorithm we must first describe how a fractal can be encoded in a genotype. The **genotype** of a fractal is a list of a varying number of transformations, where each one it's a gene and it is encoded as a vector of seven real values: the first six being the coefficients used to represent the transformation, and the seventh, the weight associated to it, whose role can be seen as how much that specific transformation will be used in the process of creating the resulting image, which will be the **phenotype** of the fractal.

For the choice of the **initial population** there are two possibilities, the first, starting from randomly initialized fractals and the second, using some well known typologies of fractals. We choose to implement the latter in order to start from a solid base.

As for the **number of individuals** belonging to a generation, since the evaluation is done by a human, it is kept to a low number (10), to keep the process as fast as possible.

Another important choice to take was which **fitness function** to use, and as said in the section above, we leave the task of assigning a score (in the range $[0, 100]$) to the resulting images, to a human, via a simple interface made in Python.

1) Mutation: After an user assigns the score to the fractals displayed on the interface, each of them is mutated with a probability of 0.5.

The mutation consists of adding a random value, sampled from a Normal Distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$, multiplied by an hyperparameter set to 0.1, to each parameter in all the transformations of each fractal. After the

mutation, the mutated fractal is added to the population, with the same score of the original, along with a copy of the original before the mutation.

2) Parent selection: For the parent selection, a **fitness-proportionate method** is used, where the probability of an individual of being selected to generate an offspring is proportional to its fitness with respect to the total fitness of the population. In our project, the fitness of an individual is the score given by the user and, in order to obtain the probability of becoming a parent, this number is divided by the sum of all the fitness of the whole population. This selection method comes with some problems, like a high selection pressure, which leads to low-fitness individuals to have a very slim chance of being selected. But this is exactly what we wanted, since we don't want genes from fractals with a low score to be used in future generations.

3) Crossover: For the crossover operation, in the case of fractals, **multiple parents** can be used at the same time and also **multiple crossover points** are possible in the intersections between different genes. In our project we select a random number n between 2 and 4. Next, we extract n parents from the population, which now includes the mutated fractals and their original counterpart, and we start the crossover step. To choose how many genes one offspring will have, since at least one gene will be taken from every parent, the final number of genes is a random number between n and the number of transformations of the parent with the highest number of them.

4) Final Population: All the offspring will be used for the next population, by replacing the previous one, with the exception of the three highest evaluated fractals, which are classified as **elite individuals**. Therefore they will be included without any crossover or mutation operation in the new population.

B. Fractal Generator

1) Creating the Fractals: At the beginning of the rendering, each transformation of each fractal generated by the Genetic Algorithm will have a probability p_i associated to it. Those single probabilities will be summed up to get the total probability p_{tot} of each fractal. It's important to notice that we don't mind if p_{tot} surpasses the value of 1 as it will be later explained.

Next, we create two set of tuples: one containing the points for the binary fractal that would be the skeleton of the new image - i.e. *drawing IFS* -, and one containing the points for the fractal used in the **color stealing** process - i.e *coloring IFS*. The process of building a new fractal will be the same for both and both will be initialized with the origin point set to $(0, 0)$. This will be the anchor point of the fractal from which we will generate all the subsequent points of the image.

In order to create new points of the fractals, for every point in the set:

- We iterate through the IFS transformations of a fractal and select one of them with probability equal to $\frac{p_i}{p_{tot}}$. We can see that the selection follows the principle of the roulette wheel selection [4] where we can avoid to treat the p_{tot} as a joint probability with value set to one. This permits us to avoid normalizing the probability of each transformation every time we compute a new fractal.
- After selecting a transformation, it will be applied to the point creating a new one that will be added to the set.

We repeat this process, where we grow new points, until we reach the number of iterations (set at the start of the algorithm). The final result will be a set of tuples made of two values, one for the x and one for the y axis but these values have not being referred as spatial coordinates yet.

During the initialization of the image we define, along with the structure of the fractals, also the size of the images to be rendered. With this constraint we can't treat generated points as coordinates of the image already. This is because we don't apply bounds to the generation of points so they can infinitely grow towards a direction with only the maximum number of iteration as a limit. To counter this, we defined a scale property which maps the tuples from the set in actual coordinates of the image in order to avoid having images with portions of the fractal outside the image scene.

2) Coloring of the fractal image: In order to obtain colored fractal images we tried two different methods:

- **Color mapping**: in this method we perform the following operations:
  1) produce a point of the fractal applying the IFS transformations;
  2) map the generated point to a pair of spatial coordinates (x, y) so that each point of the fractal will have a position in the final image associated to a pixel;
  3) the color of the pixel is determined extracting it from the corresponding position on a natural image. Therefore, if the natural image in the pixel $(0, 0)$ is colored in green (identified by the RGB tuple $(0, 128, 0)$) also the pixel $(0, 0)$ of the fractal image will be colored in the same way.
- **Color stealing**: following the work of M. Barnsley[5], we perform the following steps:
  1) compute a point of the fractal image using the *drawing IFS* transformations. For each applied transformation, a transformation of the *coloring IFS* is applied to generate a new point. Therefore, we have two IFS that runs in parallel using the same weights: the one of the *drawing IFS*;
  2) map the point generated with the *coloring IFS* to a pair of spatial coordinates (x, y) and "steal" the color present in the pixel (x, y) of a natural image;
  3) assign the stolen color to the point generated by the *drawing IFS*.

  where *drawing IFS* is the IFS used for generate the points of the fractal image. Since, *drawing IFS* can have at most a number of transformations equals to the maximum number of transformations in the parents that have generated it, *coloring IFS* is selected as the IFS with the highest number of transformations in the initial dataset. Consequently, *drawing IFS* will have less or same number (at most) of transformations than the *coloring IFS*. This guarantee that, when a transformation

(a) Fractal made with Color Mapping

(b) Fractal made with Color Stealing

Fig. 1: Results produced by the two different coloring methods

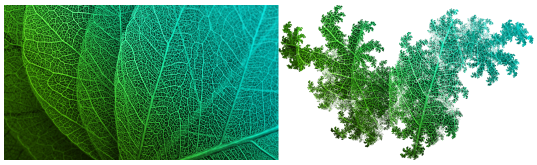

Fig. 2: Example of image crop obtained using Color Mapping. On the left the original image is displayed. On the right the generated fractal is shown

of the *drawing IFS* is applied to generate a point, a corresponding transformation of the *coloring IFS* will be always available. The transformation of *coloring IFS* to perform at each step is chosen in sequential order: first the transformation 1, second the transformation 2, etc.

The two methods produce different results as can be seen in ***Fig. 1***. In fact, while the first approach is equivalent to crop the natural image with the generated fractal shape, the second one, applying a second IFS, maps the colours of the image in the different pixels of the fractal.

As an ultimate choice we decided to color the fractal image with the second approach because with noisy fractals, the natural image appear almost without changes as can be seen in ***Fig. 2***. Instead, with the colours mixing guaranteed by the *Colour stealing* approach we're able to obtain a more uniform coloration of the fractal. In addition, to provide a smooth colorization we decided to not use natural images but gradient ones. Moreover, to produce fractals of different colors we selected ten gradient images, and, each time a fractal has to

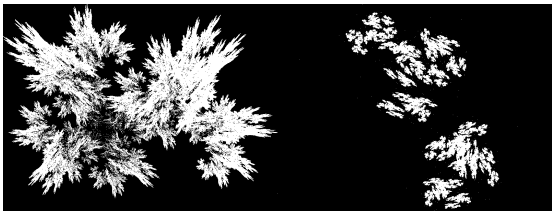

Fig. 3: Example of two Binary Fractals obtained after evolution

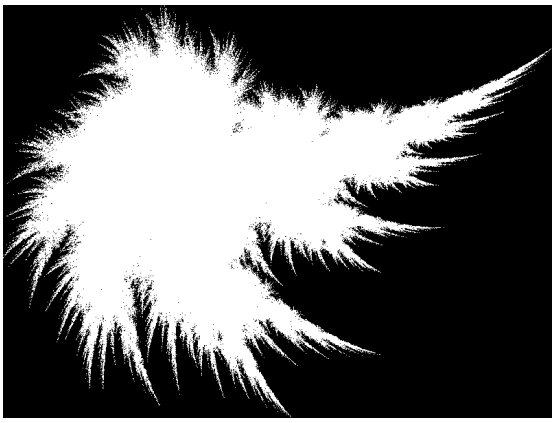

Fig. 4: Example of Noisy Binary Fractal

be generated, one of them is selected randomly. As a result, the final ten images, displayed on the interface, are colored each with a different colour.

III. Difficulties

The most of the difficulties in the development of our project were encountered during the the implementation part.

In particular, during the rendering of the fractal, we started the generation algorithm using classical IFS Fractals like the Barnsley fern [6] in order to understand if our algorithm was working properly.

The first results, while encouraging, since we were already getting some good pictures as ***Fig. 3*** shows, were mainly noisy images like the one showed in ***Fig. 4***. This is mainly due to fractals self-similarity property [7], i.e. an object is exactly or approximately similar to a part of itself. In fact, in our circumstances the base element to be replicated is a point, consequently, the algorithm will tend to generate dense clouds in part of the images.

Furthermore, since the transformations are modified by our EA instead of custom tailor each transformation, the outcome of having some noisy pictures is always a possibility even when starting with only fit fractals. A possible resolution that could be implemented in future work is to modify the starting element of the fractal in order to achieve different results and maybe highlight more effectively some patterns.

IV. Results

We can distinguish two phases of results as listed below. In fact, encouraged by the good results obtained since the first developing stage, we progressed and further improved our software obtaining even more better results.

Fig. 5: Example of application of *Spiralis Fractalis*. Here one of our fractal has been used as a landscape for a drawing.

A. First phase - Binary Images

In this first phase we generated Black and White (binary) images encountering very good results.

After few generations we were able to generate very well fitted individuals as you can see in ***Fig. 3***. These images, showed us how effectively our evolution algorithm worked. In fact, we avoided the generation of cloudy and messy fractals thanks to the idea of upper bounding the number of transformations for each individual. In addition, we kept a bit of randomness even using the same transformations (that propagates through different generations) mutating also the probability for each transformation to be picked. These promising results inspired us to further improve our algorithm, even if in a purely aesthetic way.

B. Second phase - Colored Images

The second phase outstanded our previously obtained results.

The combination of well constructed fractals with the technique of Color Stealing[5] brought us some incredible masterpieces ***Fig. 1b***. Moreover, including more than one color gradient and using some randomness to match each fractal with a different color, led to a very eye catching generative art generation.

V. Conclusions

Spiralis Fractalis is an example of successful application of Genetic Algorithm to the Generative Art field. In fact, it provide an efficient and effective tool to design colored fractal images that can be used for various purposes such as wallpapers, posters, etc. as it's possible to see in ***Fig. 5***

In addition, since, the fitness used in order to iterate through the execution of the software, is user-based, an user is free to interrupt the procedure as fast as a satisfying image is generated and obtain almost effortless an abstract image to use.

VI. Contributions

In order to develop our software, all the components of the group equally contributed to it.

In particular, the Fractal Generator piece of code was written by M. Carolo and C. C. Rambaldi, the part of the Genetic Algorithm was written di S. Dissegna and L. Corso.

In addition, the colouring code was developed by M. Carolo and L. Corso, while the interface was implemented by C.C. Rambaldi and S. Dissegna.

Moreover, to write this paper, we divided its different parts assigning each of them to the person that developed the corresponding portion of code.

References

[1] E. B. S. Wannarumon and K. Annanon, "Aesthetic evolutionary algorithm for fractal-based user-centered jewelry design," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 22, no. 1, pp. 19–39, 2007.

[2] Wikipedia. Generative art. [Online]. Available: https://en.wikipedia.org/wiki/Generative_art

[3] Z. Chen, L. Chen, Z. Zhao, and Y. Wang, "Ai illustrator: Art illustration generation based on generative adversarial network," in *2020 IEEE 5th International Conference on Image, Vision and Computing (ICIVC)*, 07 2020, pp. 155–159.

[4] A. Lipowski and D. Lipowska, "Roulette-wheel selection via stochastic acceptance," *CoRR*, vol. abs/1109.3627, 2011. [Online]. Available: http://arxiv.org/abs/1109.3627

[5] M. Barnsley, "Ergodic theory, fractal tops and colour stealing," *Lecture Notes, Australian National University*, 01 2004. [Online]. Available: https://maths-people.anu.edu.au/~barnsley/pdfs/fractal_tops.pdf

[6] Wikipedia. Barnsley fern. [Online]. Available: https://en.wikipedia.org/wiki/Barnsley_fern

[7] B. B. Mandelbrot, "Self-Affine Fractals and Fractal Dimension," *physscr*, vol. 32, no. 4, pp. 257–260, Oct. 1985.