PH3170 Fractal Visualisation Report

Abstract

Geometrical shapes called fractals can be modelled using the Chaos Game algorithm and its generalisation. A C++ program was developed to implement these algorithms and visualise the output as an image. The program was used to explore and investigate 2D fractals and the effects of changing parameters of the Iterating Function System code for different attractors.

1 Introduction

A C++ program was created with the aim of allowing the user to explore the concept of fractals through the visualisation of different user-defined fractal shapes. A user-friendly interface was designed to provide three main options to model 2D fractals, guided by the algorithms of the Chaos Game and its generalisation. The interface can take either vertices of 2D polygons or transformation sets as input, providing a default example of the latter, which produces the fractal called Barnsley's fern, as an extra option. Other parameters such as probability of selecting a vertex or transformation, fraction of the distance to move towards the next selected vertex, and number of iterations of the algorithm can also be varied by the user. Once all parameters have been defined, the code then outputs a BMP image of the shape corresponding to those parameters.

2 Fractal modelling

Fractals are geometric shapes which display a pattern at every scale. They are generally self-similar as well, meaning that they exhibit the same kind of pattern whether you look at them as a whole or you magnify them by any amount [1]. Similar shapes are frequently found in nature (though never true fractals as there is a limit to the scale they can magnify to), taking the form of snowflakes, plants or clouds to name a few examples [2].

Fractals are shapes which were not defined by classical Euclidean geometry, and have been explored with modern mathematics [3]. Michael Barnsley published several methods in 1988 to model these

shapes from a small set of numbers, including the Chaos Game and the Iterated Function System (IFS).

2.1 The Chaos Game

The Chaos Game is a method which, in its simplest version, takes the vertices of a pre-defined shape and draws points within it, usually resulting in the repeating pattern characteristic of fractals, though the method does not always generate a fractal [4]. It obeys a set of rules, which may be modified to create a wider variety of patterns, but in the simplest case they are as follows: a vertex is selected as the starting position, and after selecting a vertex at random (with equal probabilities of selecting any vertex), a point is drawn between the two. This process is repeated for any number of iterations chosen starting from the last point created. The fraction of the distance between the two points d at which a new point is drawn may be any proper fraction, but must remain constant throughout the process.

How defined the shape produced is depends on the number of iterations done, but will always approach the same unique shape given the same parameters, a shape which is called the attractor of the dynamical system. The fractal structure of an attractor like the Sierpinski triangle (as shown in Fig. 1, which will later be discussed in Sec. 4.1) generated from this method can be clearly observed. The Sierpinski triangle, where $d = 1/2$, is a large triangle composed of three smaller triangles of half the original size with the same pattern, and the same can be said about each of these smaller triangles. All three smaller triangles are the same because the same transformation is essentially being applied to each vertex [5]. The transformations that can be applied have specific constraints, which will now be discussed.

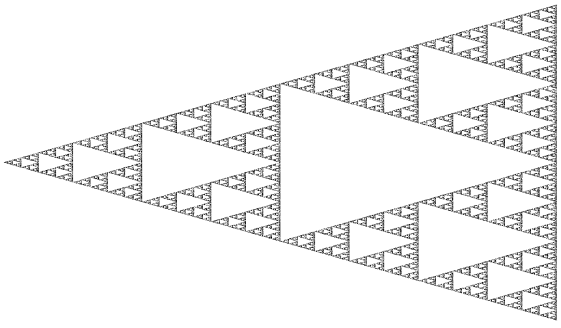

Figure 1: Sierpinski triangle resolved after 100,000 iterations of the Chaos Game algorithm.

2.2 Generalised Chaos Game

The Chaos Game can be generalised by using the IFS method. The IFS can represent complex fractal objects using descriptions of the smaller geometrical objects that compose them and the relationships between these. The relationships can be expressed quantitatively with transformations, which must be affine transformations in particular, as these preserve co-linearity and will therefore lead to shapes that have the same pattern of parallel lines as the original one. An affine transformation takes the following form:

$$w\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \tag{1}$$

where the matrix $\mathbf{A}$ represents rotation and scaling and the vector $\vec{b}$ represents translation. The resulting shapes must also be smaller than the original one such that they continue the fractal pattern at all scales. Therefore, contraction mappings, a subset of affine transformations, must be used, where the matrix $\mathbf{A}$ must meet the following constraints:

$$a_{11}^2 + a_{21}^2 < 1, \tag{2}$$

$$a_{12}^2 + a_{22}^2 < 1, \tag{3}$$

$$a_{11}^2 + a_{12}^2 + a_{21}^2 + a_{22}^2 - [\det(\mathbf{A})]^2 < 1. \tag{4}$$

With a valid set of transformations w_n and their associated probabilities of being selected p_n, the IFS code for their corresponding attractor can then be expressed by

$$\{w_n, p_n : n = 1, 2, \ldots, N\}, \tag{5}$$

where N is the number of transformations.

3 The program

The program was built to demonstrate the implementation of the Chaos Game algorithm and its generalisation with each option available to the user. Each option makes use of a different class object, where the classes were developed with increasing complexity and therefore contain more advanced exception handling and more complex method structures. Each class contains several private methods which are used by the only public method `fractal`. This method takes in a set of input parameters, defines the Chaos Game algorithm and outputs the coordinates for the set of points in the resulting shape (potentially a fractal). The set of points is then transformed into an image using the EasyBMP visualisation library, drawing pixels in its array grid at the coordinates matching those of the shape.

3.1 User interface

A user menu is displayed when the program is run, prompting the user to choose one of three options via user input. Once an option is chosen, the `switch` statement selects the corresponding sub-panel to be displayed, which asks for the input parameters corresponding to the `fractal` method of the class object that will be used. When the option for the Generalised Chaos Game is chosen, a sub-menu is displayed, giving the option of the default example, Barnsley's fern, or using different transformations. The input for this part of the code is taken as the name of a file. The transformations and probabilities contained within are validated using Eqs. 2, 3 and 4 and are stored as part of the construction of the corresponding class object. Both `switch` statements for the menus also make use of a default statement, which prompts the user to choose another option if their input option is not valid, and restarts the program by falling through the rest of the while loop. The sub-menu "go to main menu" option operates in the same way. Finally, the main menu also has an option to exit the program, which ends the while loop and therefore terminates the program.

3.2 Chaos Game algorithm

Note all `fractal` methods operate in a similar way, and therefore only the most complex version, that of the Generalised Chaos Game class, will be explained. Once it has been checked that the probabilities for all transformations have been stated in the input file and that the probabilities are valid, the starting point for the iteration is set to be a pre-defined point. A transformation is then selected at random according to the associated probabilities using the method `getRandomTransformation`, the transformation is applied to the starting point using the method `transformPoint` and the coordinates for the new point created are added to the set of coordinates contained in member variable `_vertices` using the method `addVertex`. The process is repeated, replacing the starting point with the last point created, for the number of iterations defined by user input, obtaining a set of coordinates for the new shape at the end of the loop.

3.3 Scaling function

For the Generalised Chaos Game option in the menu, a scaling function was also added before creating an image. After the fractal set of coordinates are obtained, it is frequently necessary to scale them to a size that fits into the image created by EasyBMP, such as with the Barnsley's fern example. The `scalingFactor` function takes the coordinate set from the fractal and the size of the image to be used, and finds the maximum and minimum x and y values in the coordinate set. These are then used to find the scaling factor by which to multiply each coordinate such that the shape fits in the image. The

minimum x and y values found are then used outside the function to translate the coordinate set so it's contained by the image.

4 Output

As mentioned in Sec. 3, the output of the program is a BMP image with a size of 1920 x 1200 BMP by default. The colour of the pixels is set to black for most cases, and green for Barnsley's fern to mimic the Black Spleenwort fern it models. The program was used to observe and investigate the shapes produced from the algorithms after varying different parameters.

4.1 Chaos Game

An example of a shape that can be produced using either the first or second option in the user menu is a Sierpinski triangle, shown in Fig. 1 and previously mentioned. The one shown in the figure was resolved after 100,000 iterations, which is a sufficiently large number in comparison to the ratio of the viewing resolution (1920 x 1200) and the triangle size in order to reveal the detail in this attractor. Other shapes that can be created using the Chaos Game include more Sierpinski shapes for polygons of different sides, such as the Sierpinski pentagon found at d equal to the golden ratio, or other pentagonal and hexagonal snowflakes at $d = 1/3$ or $3/8$ and the interior of a square at $d = 1/2$, all with equal probabilities for every vertex.

By modifying the algorithm to also include certain rules could allow producing shapes such as the Koch snowflake, which requires the point at the center of the polygon to be included as part of the set of vertices selected at random. The Sierpinski Carpet can also be achieved by using $d = 2/3$ and including the midpoints of the edges of the square used in the set of vertices. Other interesting rules that can be explored are not being able to select the same vertex in succession or having to select the vertices in a certain order.

Changing the probabilities of selecting certain vertices resulted in a less defined area of the shape close to the least probable vertex at a certain number of iterations, such as when maintaining the 100,000 iterations for the previous Sierpinski triangle. However, once the number of iterations is increased enough, there should be no change in the final shape since the attractor will always be final state of the system, even if one of the probabilities is very small.

4.2 Generalised Chaos Game

The Generalised Chaos Game algorithm was tested using the IFS code for Barnsley's fern (found in Ref. [6]), producing the image shown in Fig. 2 after 1,000,000 iterations. It has to be taken into

account that the shape was at its maximum size within the image as a result of the scaling function applied and the number of iterations needed is therefore at a maximum for the chosen parameters.

When the probabilities for the different transformations were varied, decreasing the probability of w_2 resulted in a larger number of iterations needed to resolve the structure, reaching an impractical point when all transformations have equal probabilities. When the opposite approach was taken and the probability of w_2 was increased to a maximum whilst maintaining the other probabilities above 0, the geometrical shape of the structure was conserved but only the veins of the leaves were observed within, without filling the inside of the leaves with points.

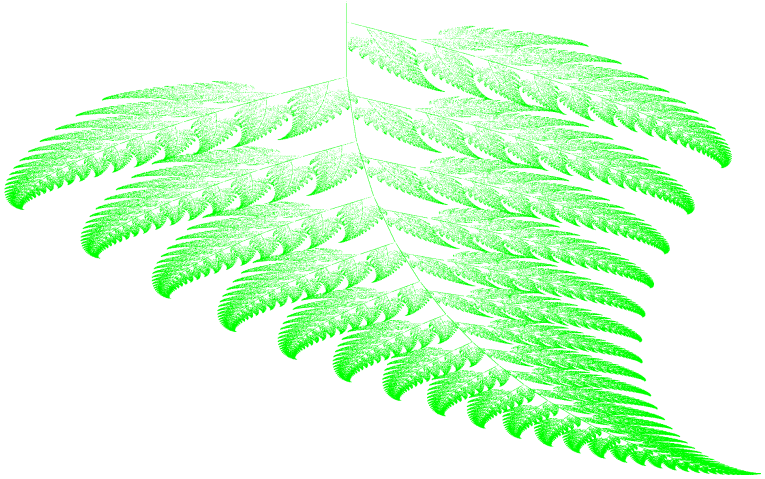

Figure 2: Barnsley's fern resolved after 1,000,000 iterations of the Generalised Chaos Game algorithm.

5 Evaluation of code

The program successfully modelled certain fractals like the Sierpinski triangle using the Chaos Game algorithm or Barnsley's fern using the Generalised Chaos Game and the corresponding IFS code. However, one issue with the code was found as the number of iterations was increased. As iterations increase, member variables like `_vertices` increase its number of elements quickly, and as these elements are arrays of two numbers, these take up too much stack memory. This problem could be solved, or at least improved, by using smart pointers pointing to these values placed on the heap.

Another issue which was found during implementation was the inability to include exact fractions such as 1/3 for values such as the probabilities or the distance fraction. It is firstly not possible for the program to understand fractions in such a form through user input or file reading and even if this could be solved, the exact value would not be stored with enough accuracy by the computer, resulting in the exception for sum of probabilities not equal to 1.0 being thrown.

Finally, as explained in Sec. 3, the different classes have been developed with increasing level

of complexity and level of exception handling, and as three separate entities for the purpose of the exercise. For better coding practice, these classes could be linked by inheritance, reducing the amount of repeated code. The scaling function could have also been used on the coordinates obtained for the first two user options, so as to ensure the shape size never exceeds the size of the image.

6 Summary

The program effectively models fractals using the Chaos Game algorithm and its generalisation, and allows the user to explore and investigate these structures by visualising the set of coordinates of the fractal and providing ways of easily modifying the starting parameters or IFS code used. Images of a Sierpinski triangle and Barnsley's fern were successfully generated, and the behaviour of the structures upon modifying their parameters was investigated and recorded, observing different behaviours depending on the iterative method used. Despite limitations such as lack of memory management and inability to use certain exact fraction values as parameters, the program seems fairly robust and achieves the task of fractal modelling at reasonable numbers of iterations.

References

[1] Weisstein, Eric W. *Fractal.* From MathWorld–A Wolfram Web Resource. `https://mathworld.wolfram.com/Fractal.html`

[2] Mathigon. *Fractals.* `https://mathigon.org/course/fractals`

[3] Mandelbrot, B. B. *The Fractal Geometry of Nature.* New York, NY: W. H. Freeman and Company (1982).

[4] Weisstein, Eric W. *Chaos Game.* From MathWorld–A Wolfram Web Resource. `https://mathworld.wolfram.com/ChaosGame.html`

[5] Peitgen, H. O. & Saupe, D. *The Science of Fractal Images.* New York, NY: SpringerVerlag (1988).

[6] Royal Holloway Physics department *PH3170 (C++ Programming) Fractal Modelling Project*, accessed on Moodle (2020).