Generation of Fractals: IFS & Chaos Game

A Project Work Presented for the Degree of

BACHELOR OF SCIENCE

To

ST. XAVIER'S COLLEGE, KOLKATA (AUTONOMOUS)

By

KOESHA SINHA

Roll No.:0073

Under Guidance of

Prof. DIPTIMAN SAHA

Department of Mathematics

ST. XAVIER'S COLLEGE, KOLKATA (AUTONOMOUS)

Kolkata, 700016, India

April, 2021

I affirm that I have identified all my sources and that no part of my dissertation paper uses unacknowledged materials.

Supervisor's Signature

Head of the Department's Signature

Koesha Sinha

Student's Signature

Acknowledgement

The project entitled **'Generation of Fractals: IFS & Chaos Game'** comprises of my research work. I would like to express my sincere gratitude towards my supervisor, **Professor Diptiman Saha**, Associate Professor, Department of Mathematics, SXC Kolkata; without whom this project would not have been possible. His constant support and encouragement, motivated me to successfully complete my dissertation.

I would also like to thank **Professor Sucharita Roy**, H.O.D, Assistant Professor, Department of Mathematics and St. Xavier's College (Autonomous), Kolkata, for providing me with the opportunity to undertake this topic as my project.

Lastly, I would like to thank my parents Mr. Sajal Kumar Sinha , Mrs. Keya Sinha Bhadra and my friends for being supportive throughout the process of this project.

Abstract

The main objective of this project is to understand the generation of fractals and creating some of them using various mathematical algorithms including ***Iteration of Functions*** and ***Chaos Game***. Fractals are never-ending patterns that repeats itself at different scales and this project studies the generation of these geometric objects from the basic level with help of some well-known theorems, including “Collage Theorem". I have tried to generate two fractals, one of which is ***“The Koch Snowflake”***, along with ***“The Koch Curve”***, which is generated with help of the IFS and another one is ***“The Sierpinski Triangle”***, which is generated with help of the Chaos Game and I have tried to visualize the process of convergence of these two fractals through a Python Programme. Study of the generation of fractals has a huge application in modern world and that is also an inseparable part of my project.

Contents

1. Introduction

How would you describe a fern using only triangles, polygonal shapes, circles, and points? It would be quite hard to do so with only the tools of Euclidean geometry. They do a good job of simplifying shapes in order for us conjecture about the properties of these smooth objects, but it doesn't translate to what we see around us every day. That's why we turn to fractal geometry- it allows us to describe and classify more complex objects in a simple, but precise manner.

Many processes in our complex world are "***chaotic***". With this, we mean that small changes may have very large influences and detailed features of such a process are very hard to predict. An example is the complex firing pattern of neurons in our brain as we think and learn.

However, through our studies, it's important to keep in mind that fractals are abstractions, a result of an ***infinitely iterative process***.

In mathematics, the term Chaos Game originally referred to a method of creating a fractal, using a polygon and an initial point selected at random inside it.

1.1 IFS & Chaos Game:

Iterated function systems (IFS) are a formalism for generating exactly self

similar fractals based on work of ***Hutchinson*** (1981) and ***Mandelbrot*** (1982), and popularized by ***Barnsley*** (1988).

The roots of Chaos Theory date back to about 1900 when the mathematician Henri Poincaré was working with the three-body problem. While working with the three-body problem, he became the first person to discover a chaotic deterministic system. His work laid the foundation for Chaos Theory. In 1960, the meteorologist Edward Lorenz became the first true pioneer to work with Chaos Theory. His interest in Chaos Theory came about accidentally as he was studying weather patterns.

The so-called chaos game (also called a random iteration algorithm) is a method to generate iterative fractals with the help of polygons where the polygons being repeated converge very slowly to a certain geometric figure, simply known as Fractals.

In this paper we will introduce all the basic mathematical ideas needed to understand the generation of fractals using various methods including IFS & chaos game.

1.2 Fractal Terminology:

Before we get into any more detail, we need to cover some basic terminology that will help us understand the unique qualities that fractals possess.

1. ***Self-similarity***: the details of image is made up of small copies of the whole image.

2. ***Structure at different scales***: the details of the image is intricate regardless of magnification .

3. ***Fractal dimension***: a non-integer number in between two dimensions

Now taking all of that, and we can plainly see that a ***pure fractal*** is a geometric shape, that is self-similar through infinite iterations in a recursive pattern and through infinite detail and we are on generating it.

2. Tools Used in My Project

A ***Metric Space*** (X,d) is a space X together with a real-valued function d:X × X → ℝ, which measures the distance between any two points x and y in X.The function 'd' is known as the metric which satisfies the following:

- d(x, y)=d(y, x) ∀ x, y ∈ X.
- 0 < d(x, y) < ∞ ∀ x, y ∈ X, x ≠ y.
- d (x, x)=0 ∀ x ∈ X.
- The triangle Inequality, that is, d(x, y) ≤ d(x, z) + d (z, y) ∀ x, y, z ∈ X.

2.1 Transformations on Metric Space:

Let (X, d) be a metric space. A ***transformation*** on X is a function f: X⟶X which assigns exactly one point f(x)∈X to each point x∈ X.

If S ⊂ X, then f(S) ={f(x): x ∈ S}. f is *one-to-one* if x, y ∈ X with f(x) =f(y) implies x=y. Function f is *onto* if f(X)=X. f is called invertible if it is one-to-one and onto: in this case it is possible to define a transformation f^{-1}:X→X , called the inverse of f, by f^{-1}(y)=x where x∈ X is the unique point such that y=f(x).

2.1.1 Definition:

Let f: X⟶X be a transformation on a metric space (X,d). The ***forward iterates*** of f are transformations f^{on} :X→X defined byf^{o0} (x)=x, f^{o1} (x)=f(x), $f^{o(n+1)} = f \circ f^{on}(x) = f(f^{on}(x))$, *for n=1,2,3,……..*

If f is invertible then the ***backward iterates*** of f are transformations $f^{o(-m)}$:X→X defined by $f^{o(-1)}$(x)=$f^{(-1)}$(x), $f^{o(-m)}(x) = (f^{om})^{-1}(x)$, *for m=1,2,3,……..*

Also if f is invertible then, $f^{on} \circ f^{om} = f^{o(m+n)}$, $\forall m, n \in \mathbb{Z}$.

2.1.2 Definition:

A transformation w: $\mathbb{R}^2 \to \mathbb{R}^2$of the form :

$$w(x_1, x_2) = (ax_1 + bx_2 + e, cx_1 + dx_2 + f)$$

where a, b, c, d, e and f are real numbers, is called a (two-dimensional) ***affine transformation .***

We will often use the following equivalent notations:

$$w(x) = w\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} = Ax + t$$

Here $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is a two-dimensional 2x2 real matrix and t is the column vector $\begin{pmatrix} e \\ f \end{pmatrix}$which we do not distinguish from the co-ordinate pair $(e, f) \in \mathbb{R}^2$.

2.1.3 Definition:

A transformation w: $\mathbb{R}^2 \to \mathbb{R}^2$ is called ***similitude*** if it is an affine transformation having one of the special forms:

$$w\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}rcos\theta & -rsin\theta\\ rsin\theta & rcos\theta\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \begin{pmatrix}e\\f\end{pmatrix}$$

Or,

$$w\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}rcos\theta & rsin\theta\\ rsin\theta & -rcos\theta\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \begin{pmatrix}e\\f\end{pmatrix},$$

For some translation $(e, f) \in \mathbb{R}^2$, some real number $r \neq 0$, and some angle θ, $0 < \theta < 2\pi$. θ is called the rotation angle while r is called the scale factor or scaling.

Similitude transformation preserves angle.

2.1.4 **Definition:**

A transformation $f: \mathbb{R} \to \mathbb{R}$ of the form

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_Nx^N$$

Where the coefficients $a_i (i = 0,1,2,3, \ldots, N)$ are real numbers, $a_N \neq 0$, and N is a non negative integer , is called a ***polynomial transformation***. N is called the ***degree*** of the transformation.

2.1.5 **Definition:**

A transformation $f: \hat{\mathbb{C}} \to \hat{\mathbb{C}}$ defined in the form

$$f(x) = \frac{ax+b}{cx+d},\ a, b, c, d \in \mathbb{C},\quad ad \neq bc,$$

Is called a ***linear fractional transformation*** or a ***Möbius transformation***.

If $c \neq 0$ then $f\left(-\frac{d}{c}\right) = \infty$, and $f(\infty) = \frac{a}{c}$. If $c = 0$ then $f(\infty) = \infty$.

A point $z_0 \in C_\infty$ is called a ***fixed point*** of a function f if f $(z_0) = z_0$.

Example:

A Möbius transformation takes circles onto circles.

2.1.6 **Definition:**

Analytic transformations are nothing but generalisation of Möbius transformation.

Let $((\mathbb{C}, d)$ denote the complex plane with the Euclidean metric.A transformation $f: \mathbb{C} \to \mathbb{C}$ is called *analytic* if for each $z_0 \in \mathbb{C}$ there is a similitude of the form

$$w(z) = az + b, \text{ for some pair of numbers } a, b \in \mathbb{C}$$

Such that $\frac{d(f(z),w(z))}{d(z,z_0)} \to 0\ as\ z \to z_0$. The numbers a and b depend on z_0.If corresponding to a certain point $z_0 = c$, we have $a = 0$, then c is called a ***critical point*** of the transformation ; and $f(c)$ is called a *critical value.*

3. Generating fractals using iterations of function

Let us try to create some basic fractals using iterations of functions. This would be the first step in our project to make fractals by our own.

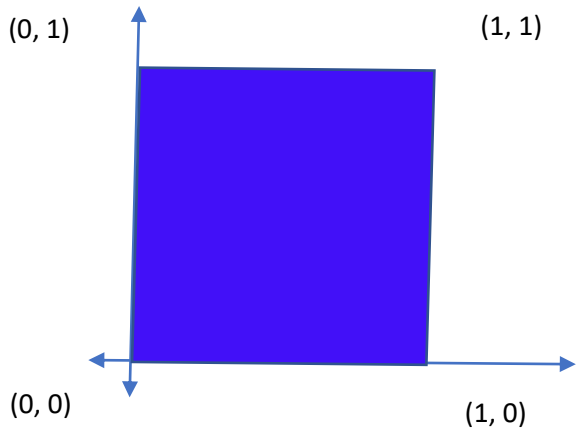
3.1 Sierpinski Triangle:

3.1.1 When initial image is a asquare-

Let's try to create some basic fractals using functions on the plane.

We can start with a square with corners at (0, 0), (1, 0), (0, 1), and (1, 1).

We will call our initial image of a square S_0.



We are interested in what happens to our square when we consider the functions:

$$f_1(x,y) = \left(\frac{x}{2}, \frac{y}{2}\right), f_2(x,y) = \left(\frac{x}{2} + \frac{1}{2}, \frac{y}{2}\right) \text{ and } f_3(x,y) = \left(\frac{x}{2} + \frac{1}{4}, \frac{y}{2} + \frac{1}{2}\right)$$

and evaluate them at the vertices of our square.

When we evaluate $f_1(x,y) = \left(\frac{x}{2}, \frac{y}{2}\right)$ at the vertices of S_0 we get the following:

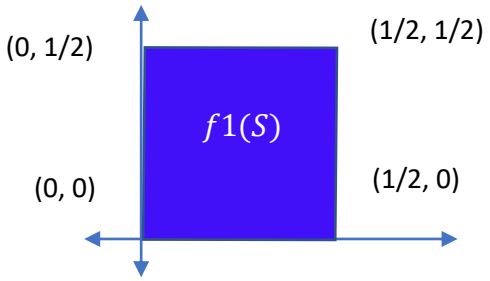
$$f_1(0,0) = (0,0)$$

$$f_1(1,0) = \left({}^1/_2, 0\right)$$

$$f_1(0,1) = \left(0, {}^1/_2\right)$$

$$f_1(1,1) = \left({}^1/_2, {}^1/_2\right)$$

Notice that this takes S_0 and shrinks it to half of its original x length and half of its original y height.



When we evaluate $f_2(x,y) = \left(\frac{x}{2} + \frac{1}{2}, \frac{y}{2}\right)$ at the vertices of S_0 we get the following:

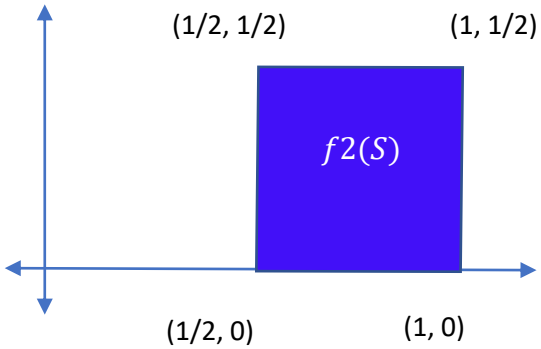
$$f_2(0,0) = \left({}^1/_2, 0\right)$$

$$f_2(1,0) = (1,0)$$

$$f_2(0,1) = \left({}^1/_2, {}^1/_2\right)$$

$$f_2(1,1) = \left(1, {}^1/_2\right)$$

Note that this is the same image as we get from f_1, but it is shifted ½ unit to the right.



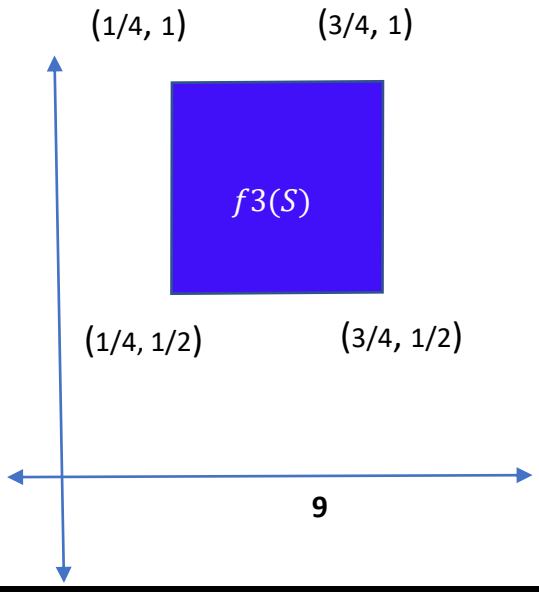
When we evaluate $f_3(x,y) = \left(\frac{x}{2} + \frac{1}{4}, \frac{y}{2} + \frac{1}{2}\right)$ at the vertices of S_0 we get the following:

$$f_3(0,0) = \left({}^1/_4, {}^1/_2\right)$$

$$f_3(1,0) = \left({}^3/_4, {}^1/_2\right)$$

$$f_3(0,1) = \left({}^1/_4, 1\right)$$

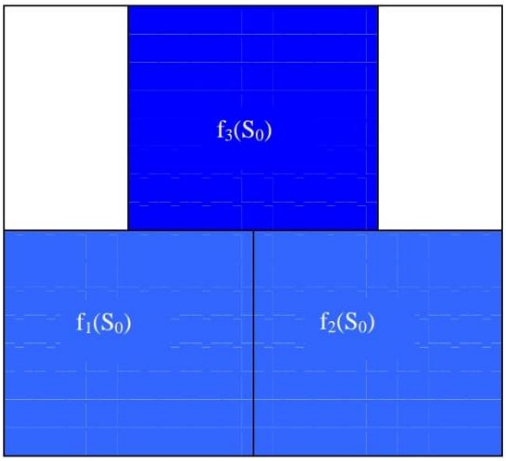
$$f_3(1,1) = \left({}^3/_4, 1\right)$$



Note that this, too, is the same image as we get from f_1, but it is shifted ¼ unit to the right and ½ unit up.

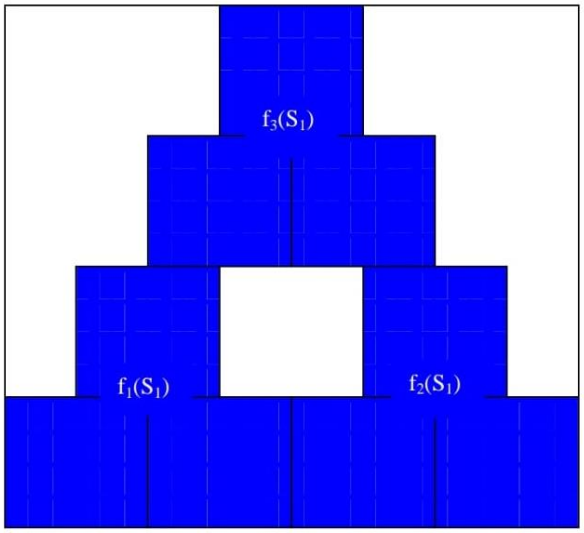
We define the new function formed as $F(S) = f1(S) \cup f2(S) \cup f3(S)$.

Here is *F(S₀)*. We will call this new image *S₁*.



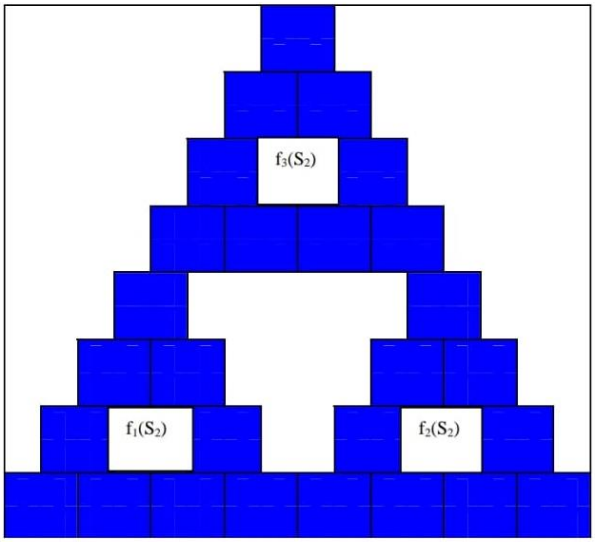
We can now iterate our image again by following the same pattern. To get *S₂* we can start by evaluating *f₁* at the vertices of *S₁* which would shrink *S₁* to half of its original *x* length and half of its original *y* height. Next, we would evaluate *f₂* at the vertices of *S₁*. This simply gives a ½ to the right translation of *f₁(S₁)*. Last, we would evaluate *f₃* at the vertices of *S₁*. This gives us a translation of *f₁(S₁)*, too. This one is shifted ¼ unit left, and ½ unit up.

We'll call this image *S₂*.



Iterate the image again by evaluating the same three functions at the vertices of *S₂*. The first function, *f₁*, will shrink the image, *f₂* will translate the shrunken figure to the right ½, and *f₃* will translate the shrunken figure to the right ¼, and up ½.

We'll call this image *S₃*.



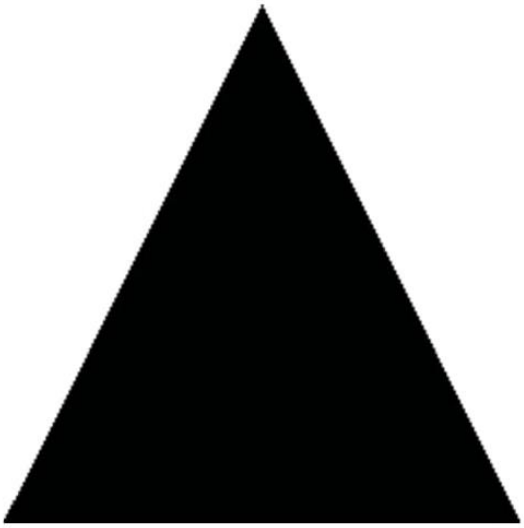
This is starting to look like the Sierpinski Triangle. We might wonder if our three functions gave us the Sierpinski Triangle because we started with four corners of a square.

3.1.2 When initial image is a triangle-

Let's see what happens if we start with an isosceles triangle instead of

a square. We start with an isosceles triangle with corners at (0, 0), (1, 0), and (½, 1)

The initial image is the isosceles triangle. We'll call this image S_0.



When we evaluate $f_1(x, y) = \left(\frac{x}{2}, \frac{y}{2}\right)$ at the vertices of S_0 we get the following:

$$f_1(0,0) = (0,0)$$

$$f_1(1,0) = \left({}^1/_2, 0\right)$$

$$f_1\left({}^1/_2, 1\right) = \left({}^1/_4, {}^1/_2\right)$$

Notice that evaluating f_1 with the vertices of S_0 shrinks the image to half of the original length of x and half the original height of y.

When we evaluate $f_2(x,y) = \left(\frac{x}{2} + \frac{1}{2}, \frac{y}{2}\right)$ at the vertices of S_0 we get the following:

$$f_2(0,0) = \left({}^1/_2, 0\right)$$

$$f_2(1,0) = (1,0)$$

$$f_2\left({}^1/_2, 1\right) = \left({}^3/_4, {}^1/_2\right)$$

Notice that this gives the same image that we achieved with f_1, but it has been shifted ½ to the right.

When we evaluate $f_3(x,y) = \left(\frac{x}{2} + \frac{1}{4}, \frac{y}{2} + \frac{1}{2}\right)$ at the vertices of S_0 we get the following:

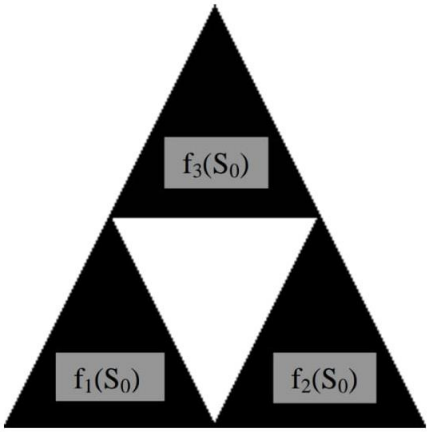
$$f_3(0,0) = \left({}^1/_4, {}^1/_2\right)$$

$$f_3(1,0) = \left({}^3/_4, {}^1/_2\right)$$

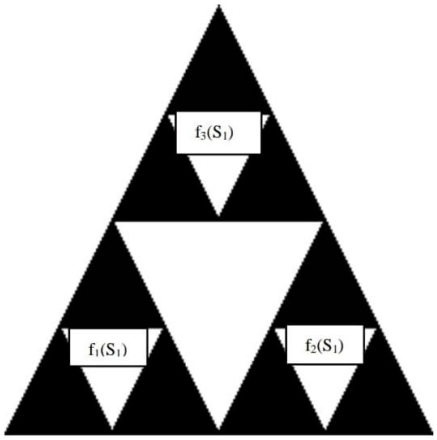
$$f_3\left({}^1/_2, 1\right) = \left({}^1/_2, 1\right)$$

This, too, gives the same image as in f_1, but it has been shifted ¼ to the right, and ½ up.

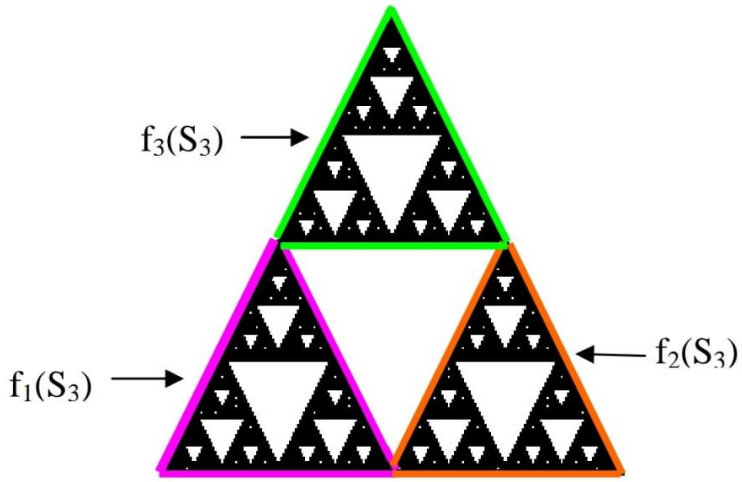
We will call this new image S_1.



After the second iteration we have the new image, S_2.



Continue with two more iterations. After the fourth Iteration, we have the new image, S_4.



After the fourth iteration, we can see that when we start with a triangle the functions are affecting the image in the same way as when we started with a square. In fact, we could start with any initial image, even a silhouette of Jim Lewis, and after enough iterations using our three functions we would begin to see the Sierpinski Triangle. The final image is actually independent of the initial image.

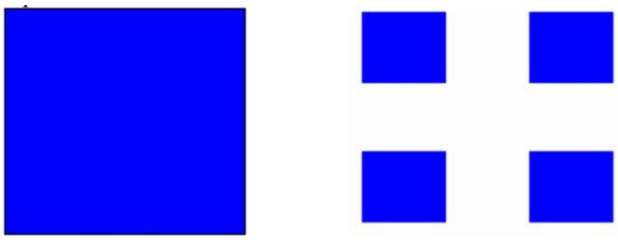
If we do enough iterations, the initial image gets smaller and smaller, becoming a dot, and so the final image is in a sense made up of an infinite number of dots. It does not matter what shape we start with, if we apply the same three functions,

$f_1(x,y) = \left(\frac{x}{2}, \frac{y}{2}\right), f_2(x,y) = \left(\frac{x}{2} + \frac{1}{2}, \frac{y}{2}\right)$ *and* $f_3(x,y) = \left(\frac{x}{2} + \frac{1}{4}, \frac{y}{2} + \frac{1}{2}\right)$,

we will get the Sierpinski Triangle.

3.2 Cantor Square:

Another famous iteration is known as the Cantor Square. The Cantor Square, in contrast, is an iteration of four functions. We learned that the first two iterations of the Cantor Square look like this:



From that, we were able to determine the functions that generate the fractal. To create the Cantor Square, we begin with a 1 x 1 square. To this image, we apply the following functions:

$$f_1(x,y) = \left(\frac{x}{3}, \frac{y}{3}\right),$$

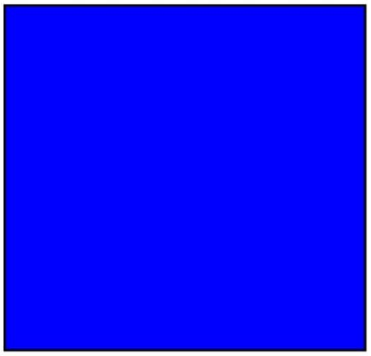
$$f_2(x,y) = \left(\frac{x}{3} + \frac{2}{3}, \frac{y}{3}\right)$$

$$f_3(x,y) = \left(\frac{x}{3}, \frac{y}{3} + \frac{2}{3}\right)$$

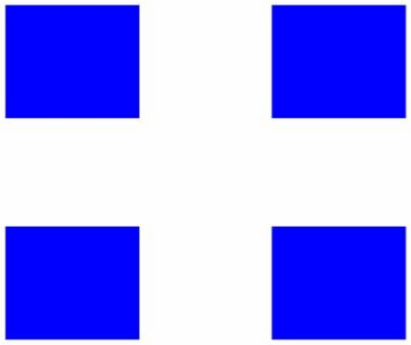
$$f_4(x,y) = \left(\frac{x}{3} + \frac{2}{3}, \frac{y}{3} + \frac{2}{3}\right)$$

We did not know these four functions before creating the fractal. We determined these functions by examining the S_0 and S_1 images. The functions came from discovering the shrinks and translations applied to the initial image, S_0.

The initial image of the Cantor Square is as follows. We will call this image S_0.

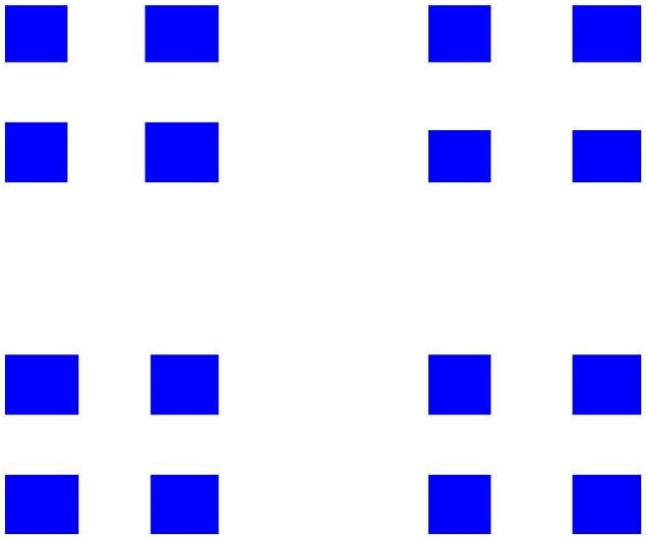


$F(S_0)$ is an image that looks like this: We will call this image S_1.

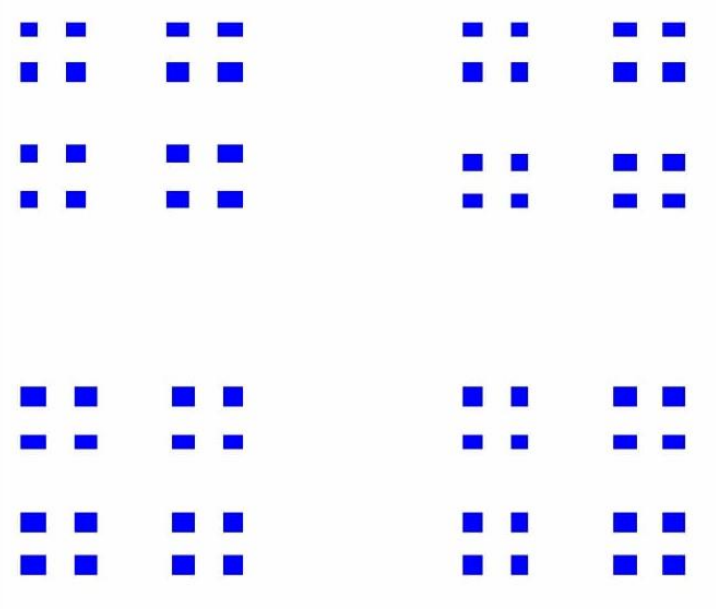


We get these four squares by applying the four functions. The function f_1 simply shrinks the image. The function f_2 translates the shrunken image to the right. The function f_3 translates the shrunken image up, and the function f_4 translates the shrunken image to the right and up.

If we repeat the iteration on the previous image, we get S_2:

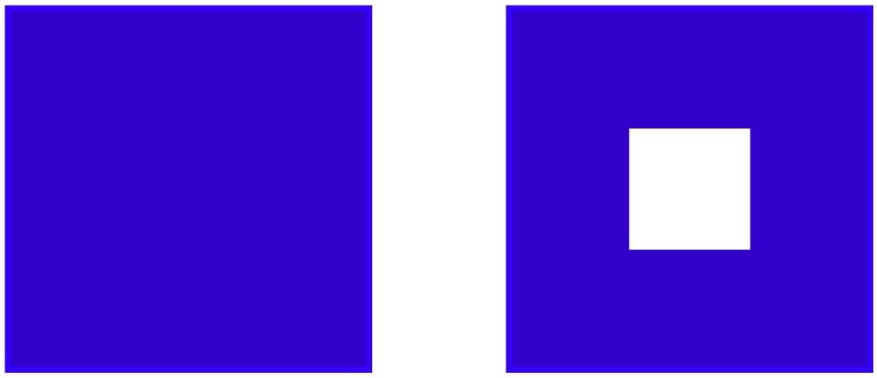


And, after one more iteration, we produce S_3:



3.3 Sierpinski Carpet:

The Sierpinski Carpet is another unique fractal. We learnt that the first two images of the Sierpinski Carpet look like the following:



By looking at these images, we determined the eight different functions necessary to generate the Sierpinski Carpet.

$$f_1(x,y) = \left(\frac{x}{3}, \frac{y}{3}\right),$$

$$f_2(x,y) = \left(\frac{x}{3} + \frac{1}{3}, \frac{y}{3}\right)$$

$$f_3(x,y) = \left(\frac{x}{3} + \frac{2}{3}, \frac{y}{3}\right)$$

$$f_4(x,y) = \left(\frac{x}{3}, \frac{y}{3} + \frac{1}{3}\right)$$

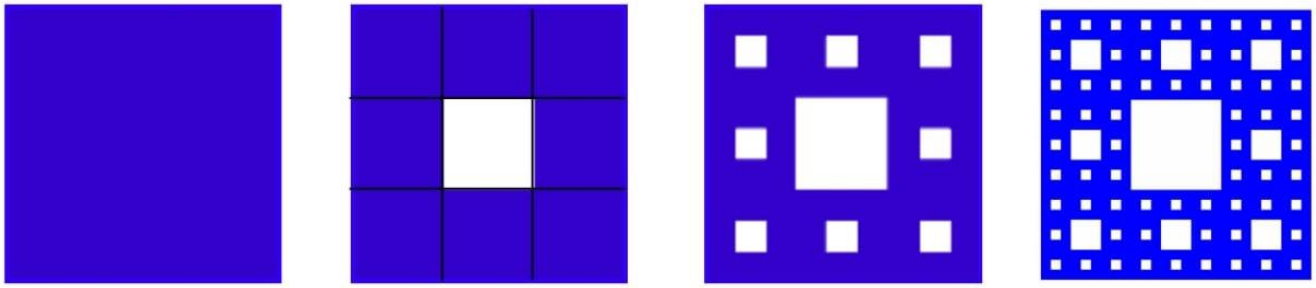
$$f_5(x,y) = \left(\frac{x}{3} + \frac{2}{3}, \frac{y}{3} + \frac{1}{3}\right)$$

$$f_6(x,y) = \left(\frac{x}{3}, \frac{y}{3} + \frac{2}{3}\right)$$

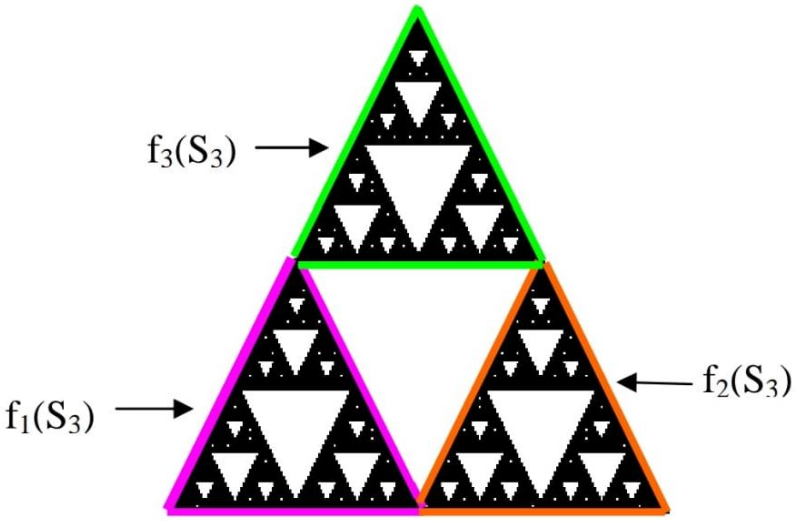
$$f_7(x,y) = \left(\frac{x}{3} + \frac{1}{3}, \frac{y}{3} + \frac{2}{3}\right)$$

$$f_8(x,y) = \left(\frac{x}{3}+\frac{2}{3},\frac{y}{3}+\frac{2}{3}\right)$$

We found these eight functions by observing the given images S_0 and S_1 and how S_0 was transformed to achieve S_1. Each part of S_1 was formed by either shrinking or shrinking and translating S_0. The first four images are shown below. Notice that the second image is actually eight shrunken copies of the previous image, seven of which are also translated. This is why we have eight functions to create the Sierpinski Carpet.



- All of the fractals we have looked at so far share the common characteristic in that they are ***self-similar***. An object is said to be self-similar if it looks "roughly" the same on any scale of magnification. We can choose a small part of the image and it will look very similar to the whole image. For example, consider the Sierpinski Triangle. Zoom in on a section, say the pink region. It is a miniature duplicate or a copy of the whole triangle, as is the yellow region. It is self-similar. In fact, all fractals share this characteristic of self similarity.



4. The Contraction Mapping Theorem , IFS & The Collage Theorem

4.1 Few Important Concepts:

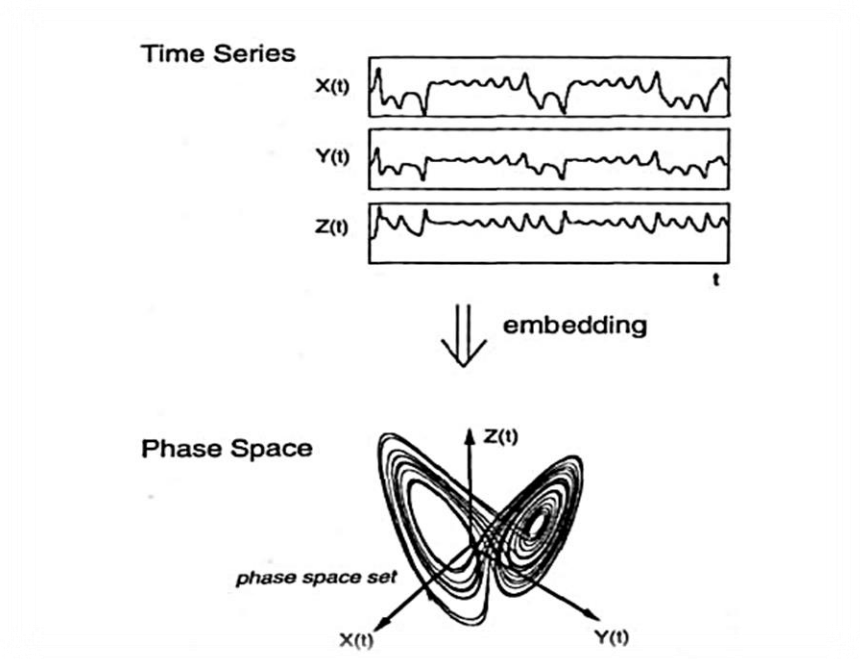
Before going to the main theorem and its application, we need to be introduced with two very important concepts-

1. Phase Space, &
2. Attractors.
3. Fixed point of an iteration.

4.1.1 Phase Space:

The mathematician named Henri Poincaré, gave us a way to transform a sequence of values in time into an object in space. This transformation replaces an analysis in time with an analysis in space. The space is called the ***phase space***. The object in the phase space is called the ***phase space set*** and the act of transforming the sequence of values in time into the object in space is known as ***embedding***. This transformation is useful because certain properties of the data are easier to determine from the phase space set than from the original sequence of values in time.

For example, let's follow the location of a baseball in a baseball stadium. Suppose the sequence of values in time of it's distance left, right and above home plate are given by X(t), Y(t) and Z(t) respectively. And at time t1, the location of the baseball is given by the values X_1, Y_1 and Z_1. The point in a three-dimensional phase space with coordinates $X=X_1$, $Y=Y_1$ and $Z=Z_1$, corresponds to the location of the baseball at that time. As the baseball moves, the values X, Y and Z changes. These values correspond to the coordinate in the phase space. Thus, the point moves in the phase space and this moving point traces out an object in the phase space. This object is the phase space set.



The sequence of values is not X(t), Y(t) and Z(t) is not limited to positions in space. For example, X(t) could be the value of the temperature, Y(t) the value of the potassium concentration and Z(t) is the value of the electrical voltage measured from a cell. Now here in this case, as the state of the cell changes, the values measured for X, Y and Z changes and the point representing the cell moves in the phase space. As it moves it traces out an object, which is the phase space set.

4.1.2 Attractors:

Let $f(t,\bullet)$ be a function which specifies the dynamics of the system. That is, if a is a point in an n-dimensional phase space, representing the initial state of the system, then $f(0,a) = a$ and, for a positive value of $t, f(t,a)$ is the result of the evolution of this state after t units of time. For example, if the system describes the evolution of a free particle in one dimension then the phase space is the plane $\mathbb{R}^2$with coordinates(x,v), where x is the position of the particle, v is its velocity , $a = (x,v)$, and the evolution is given by:

$$f(t,(x,v)) = (x + tv, v).$$

An ***attractor*** is a subset A of the phase space characterized by the following three conditions:

(i) A is ***forward invariant*** under f, that is, if a is an element of A then so is $f(t,a)$, for all $t > 0$.

(ii) There exists a neighbourhood of A, called the basins of attractions for A and denoted by *B(A)*, which consists of all points b that "***enter A in the limit*** $t \to \infty$". More formally, *B(A)* is the set of all points b in the phase space with the following property: For any open neighbourhood N of A , there is a positive constant T such that $f(t,b) \in N$ for all real$t > T$.

(iii) There is no proper (non-empty) subset A of having the first two properties.

Since the basin of attraction contains an open set containing A, every point that is sufficiently close to A is attracted to A. The definition of an attractor uses a metric on the phase space, but the resulting notion usually depends only on the topology of the phase space. In the case of $\mathbb{R}^n$, the Euclidean norm is typically used.

An attractor can be a point, a finite set of points, a curve, a manifold or even a complicated structure with a fractal structure. There are a few types of attractors. Two of them are fixed point attractors and strange attractors.

a) Fixed point attractors :

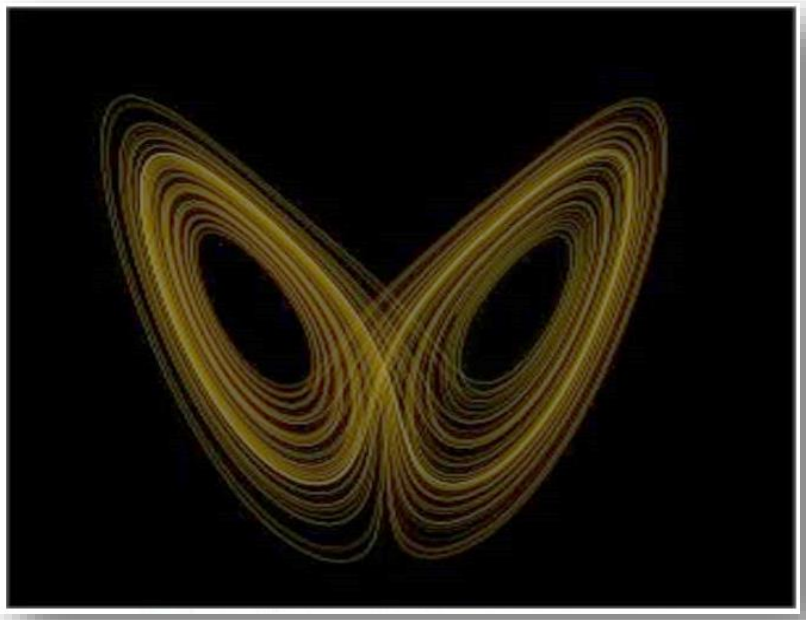
A fixed point of a function or transformation is a point that is mapped to itself by the function or transformation. If we regard the evolution of a dynamical system as a series of transformations, then there may or may not be a point which remains fixed under each transformation. The final state that a dynamical system evolves towards, corresponds to an attracting fixed point of the evolution function for that system, such as the bottom center of a bowl contain a rolling marble. But the fixed points of a dynamic system is not necessarily an ***attractor*** of the system. For example, if the bowl containing a rolling marble was inverted and the marble was balanced on top of the bowl, the center bottom (now top) of the bowl is a fixed state, but not an attractor. In the case of a marble on top of an inverted bowl, that point at the top of the bowl is a fixed point but not an attractor.

In addition, physical dynamic systems with at least one fixed point invariably have multiple fixed points and attractors due to the reality of dynamics in the physical world, including the non-linear dynamics of friction, surface roughness, deformation and even quantum mechanics. In the case of a marble on the top of an inverted bowl, even if the bowl seems perfectly hemispherical, and the marble's spherical shape, are both much more complex surfaces when examined under a microscope, and their shapes change or deform during contact. There are many points in a physical surface having a rough terrain that are considered as fixed points, some of which are categorized as attractors.

b) Strange attractors :

An attractor is called strange if it has a '***fractal structure'***. This is often when the dynamics is chaotic, that is, a dynamical system whose apparently random states of disorder and irregularities are actually governed by the underlying patterns and deterministic laws that are highly sensitive to initial conditions. Small differences in initial conditions, such as those due to error in measurements or due to rounding off errors in numerical conditions, can yield widely diverging outcomes for such dynamical systems. Although, strange non-chaotic attractors also exists.

If a strange attractor is ***chaotic***, exhibiting sensitive dependence on initial conditions, then any two arbitrarily close alternative initial points on the attractor, after any of various number of iterations, will lead to points that are arbitrarily far apart subject to the confines of the attractor and after any of various other numbers of iterations will lead to points that are arbitrarily close together. Thus a dynamic system with a chaotic attractor is locally unstable yet globally stable, that is, once some sequences have entered the attractor, nearby points diverge from one another but never depart from the attractor.



4.1.3 Fixed point of an Iteration:

Let $f : X \to X$ be a transformation on a metric space(X, d). A point $x_f \in X$ is called a ***fixed point of the transformation*** if $f(x_f) = x_f$. The fixed points of a transformation are very important as they tell us which parts of the space are pinned in space, and not moved by the transformation.

4.2 Contraction mapping:

A transformation $f: X \to X$ on a metric space (X, d) is called *contractive* or a ***contraction mapping*** if there is a constant $0 \leq s \leq 1$ such that

$$d\big(f(x), f(y)\big) \leq s.d(x, y), \forall x, y \in X.$$

Any such number *s* is called a *contractivity factor* for *f*.

4.3 The Contraction mapping Theorem:

Let $f: X \to X$ be a contraction mapping on a complete metric space (X, d).Then f possesses exactly one fixed point $x_f \in X$ and moreover for any point $\in X$, the sequence $\{f^{0n}(x): n = 0,1,2, \ldots\}$ converges to x_f. That is,

$$\lim_{n\to\infty} f^{0n}(x) = x_f, \text{ for each } x \in X.$$

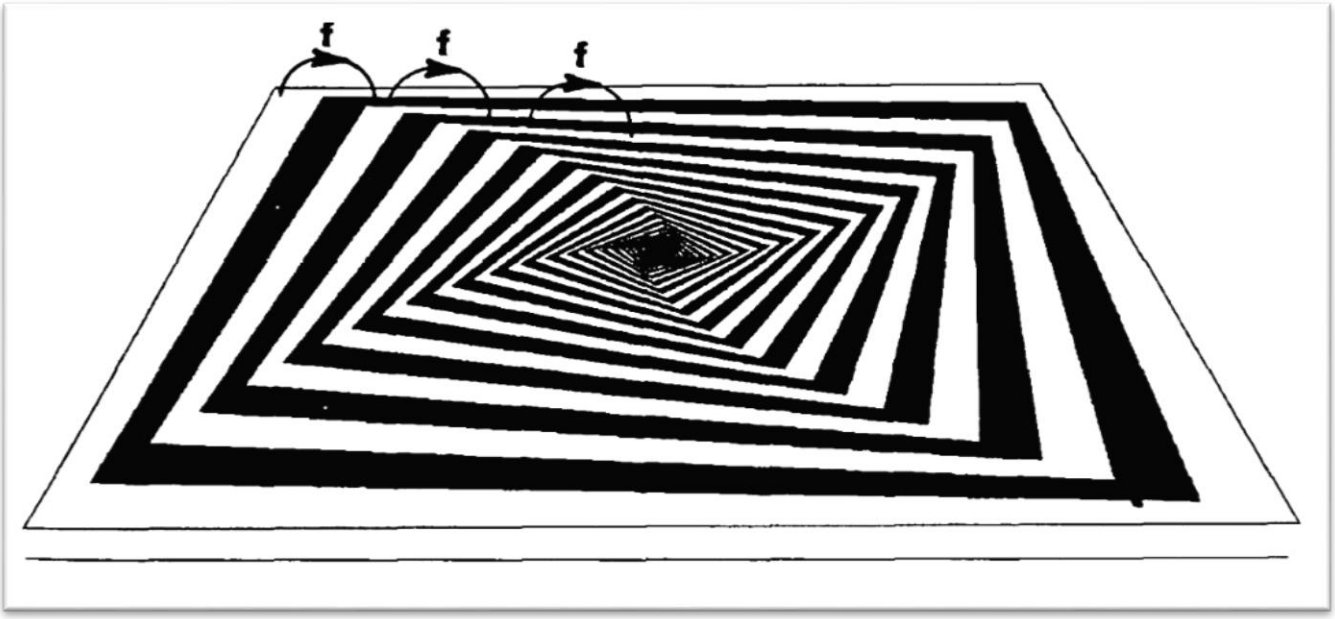


Fig: A contraction mapping doing its work, drawing all of a compact metric space X towards the fixed point

4.4 Some Results on Contraction Mappings on the Space of Fractals:

Let (X, d) be a metric space and let $(\mathcal{H}(X), h(d))$ denote the corresponding space of nonempty compact subsets, with the Hausdorff metric $h(d)$.

Result-1:

Let $w: X \to X$ be a contraction mapping on a complete metric space (X, d).

Then w is *continuous*.

Result-2:

Let $w: X \to X$ be a continuous mapping on a complete metric space (X, d). Then w maps $\mathcal{H}(X)$ into *itself.*

Result-3:

Let $w: X \to X$ be a contraction mapping on the metric space (X, d) with

contractivity factor s. Then $w: \mathcal{H}(X) \to \mathcal{H}(X)$ defined by

$$w(B) = \{w(x): x \in B\}, \forall B \in \mathcal{H}(X)$$

is a contraction mapping on $(\mathcal{H}(X), h(d))$ with contractivity factor *s*.

Result-4:

For all B, C, D and E in $\mathcal{H}(X)$

$$h(B \cup C, D \cup E) \leq h(B, D) \vee h(C, E)$$

where *h* is the Hausdroff metric.

Result-5:

Let (X, d) be a metric space. Let $\{w_n: n = 1,2, \dots, N\}$ be contraction mappings on $(\mathcal{H}(X), h)$. Let the contractivity factor for w_n be denoted by s_n for each n. Define $W: \mathcal{H}(X) \to \mathcal{H}(X)$ by

$$W(B) = w_1(B) \cup w_2(B) \dots \cup w_N(B)$$

$$= \cup_{n=1}^{N} w_n(B), \text{ for each } B \in \mathcal{H}(X).$$

Then W is a contraction mapping with contractivity factor $s = Max\{s_n: n = 1,2, \dots, N\}$.

4.5 Introduction to IFS:

A (hyperbolic) iterated function system consists of a complete metric space (X, d) together with a finite set of contraction mappings $w_n: X \to X$, with respective contractivity factors s_n for $n = 1,2, \dots, N$.

The abbreviation "IFS" is used for "***iterated function system***". The notation for the IFS just announced is $\{X;\ w_n, n = 1,2, \dots, N\}$ and its contractivity factor is $s = Max\{s_n: n = 1,2, \dots, N\}$.

The following theorem summarizes the main facts so far about a hyperbolic IFS.

4.5.1 Theorem:

Let $\{X;\ w_n, n = 1,2, \dots, N\}$ be a hyperbolic iterated function system with contractivity factor *s*. Then the transformation $W: \mathcal{H}(X) \to \mathcal{H}(X)$ defined by

$$W(B) = \cup_{n=1}^{N} w_n(B), \text{ for all } B \in \mathcal{H}(X),$$

Is a contraction mapping on the complete metric space $(\mathcal{H}(X), h(d))$ with contractivity factor *s*. That is

$$h\big(W(B), W(C)\big) \leq s.h(B, C)$$

for all $B, C \in \mathcal{H}(X)$. Its unique fixed point , A $\in \mathcal{H}(X)$, obeys

$$A = W(A) = \bigcup_{n=1}^{N} w_n(A),$$

and is given by $A = \lim_{n \to \infty} W^{0n}(B)$ for any $B \in \mathcal{H}(X)$.

The fixed point A $\in \mathcal{H}(X)$ is called the ***attractor*** of the IFS.

4.5.2 Condensation Set:

Let (X, d) be a metric space and let $\in \mathcal{H}(X)$. Define a transformation $w_0: \mathcal{H}(X) \to \mathcal{H}(X)$ by $w_0(B) = C$ for all $B \in \mathcal{H}(X)$. Then w_0 is called a condensation transformation and *C* is called the associated ***condensation set***.

4.5.3 IFS with Condensation:

Let $\{X;\ w_n, n = 1,2,\dots,N\}$ be a hyperbolic IFS with contractivity factor $0 \le s < 1$. Let $w_0: \mathcal{H}(X) \to \mathcal{H}(X)$ is a condensation transformation. Then $\{X;\ w_n, n = 1,2,\dots,N\}$ is called a (hyperbolic) ***IFS with condensation***.

4.6 Understanding the Notations for IFS:

For simplicity we restrict attention to hyperbolic IFS of the form $\{\mathbb{R}^2; w_n: n = 1,2,3,\dots,N\}$, where each mapping is an ***affine transformation***.

Let us understand the notations for an IFS of affine transformation by the following example:

The trasformations for an IFS whose attractor is a ***Sierpinski triangle*** is:

$$w_1\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}0.5 & 0\\0 & 0.5\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}1\\1\end{bmatrix},$$

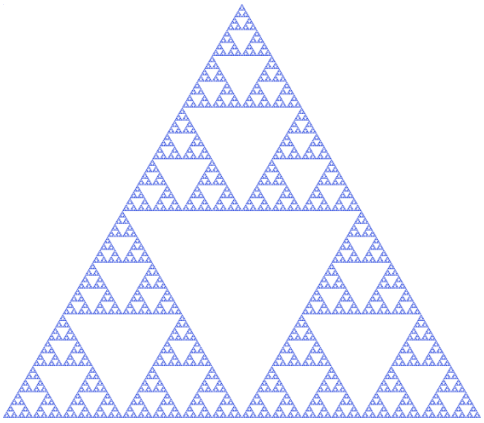
$$w_2\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}0.5 & 0\\0 & 0.5\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}1\\50\end{bmatrix},$$

$$w_3\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}0.5 & 0\\0 & 0.5\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}50\\50\end{bmatrix},$$

Which can in general be written as:

$$w_i(x) = w_i\begin{bmatrix}x_1\\x_2\end{bmatrix} = \begin{bmatrix}a_i & b_i\\c_i & d_i\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}e_i\\f_i\end{bmatrix} = A_i x + t_i.$$

The resulting image: *Sierpinski triangle*



Anyway the following table is a tidier way of conveying the same iterated function system-

w	***a***	***b***	***c***	***d***	***e***	***f***	***p***
1	0.5	0	0	0.5	1	1	0.33
2	0.5	0	0	0.5	1	50	0.33
3	0.5	0	0	0.5	50	50	0.34

Here, each of those transformations , w_i, $i = 1,2,\dots\dots\dots,N$ has a value p_i , $i = 1,2,\dots\dots\dots,N$, respectively. These p-values represents the probability that a particular transformation will be chosen and they are such that$\sum_{i=1}^{N} p_i = 1$, $p_i > 0, for\ i = 1,2,\dots,N$. These probabilities play an important role in the computation of images of the attractor of an IFS using the ***Random Iteration Algorithm***.To this end we take their values to be given approximately by

$p_i \approx \frac{|\det A_i|}{\sum_{i=1}^{N}|\det A_i|} = \frac{|a_i d_i - b_i c_i|}{\sum_{i=1}^{N}|a_i d_i - b_i c_i|}$, for $i = 1,2, \ldots \ldots \ldots, N$. (In the case of Sierpinski triangle N=3). If for some $i, det\ A_i$ = 0, then p_i should be assigned a small positive value, such as 0.001.

4.7 Steps for Creating Fractal Using IFS:

For generating a fractal using iterated function system steps given below should be considered.

1. Establishing a set of transformations.

2. Draw any initial pattern on the plan.

3. Applying transformations on the initial patterns which are defined in the first step.

4. Again apply transformation on the new image which is the combination of initial pattern and pattern after applying transformations.

5. Repeat step 4 again and again, this step 4 can be repeated infinite number of times.

The following theorem is central to the design of IFS's whose attractors are close to given sets.

4.8 The Collage Theorem [Barnsley, 1985b]:

Let (X, d) be a complete metric space.Let $L \in \mathcal{H}(X)$ be given , and let $\epsilon \geq 0$ be given. Choose an IFS (or IFS with condensation) $\{X: (w_0), w_1, w_2, \ldots, w_N\}$ with contractivity factor $0 \leq s \leq 1$, so that

$$h\left(L, \bigcup_{n=1(n=0)}^{N} w_n(L)\right) \leq \epsilon,$$

where $h(d)$ is the Hausdorff metric. Then

$$h(L, A) \leq \epsilon/(1-s)$$

where A is the attractor of the IFS. Equivalently,

$$h(L, A) \leq (1-s)^{-1} h\left(L, \bigcup_{n=1(n=0)}^{N} w_n(L)\right)$$

for all $L \in \mathcal{H}(X)$.

This theorem by ***Michael Barnsley***, a professor at Georgia Tech, actually gives the following idea :

If the image you want to get is called L, then you need to find functions f such that $F(L) = L$. Then no matter what initial image you start with, if you iterate F, you'll "eventually" get L, where "eventually" means you'll get closer and closer to it and after awhile your image will be indistinguishable form L. (direct correspondence, W. Hines)

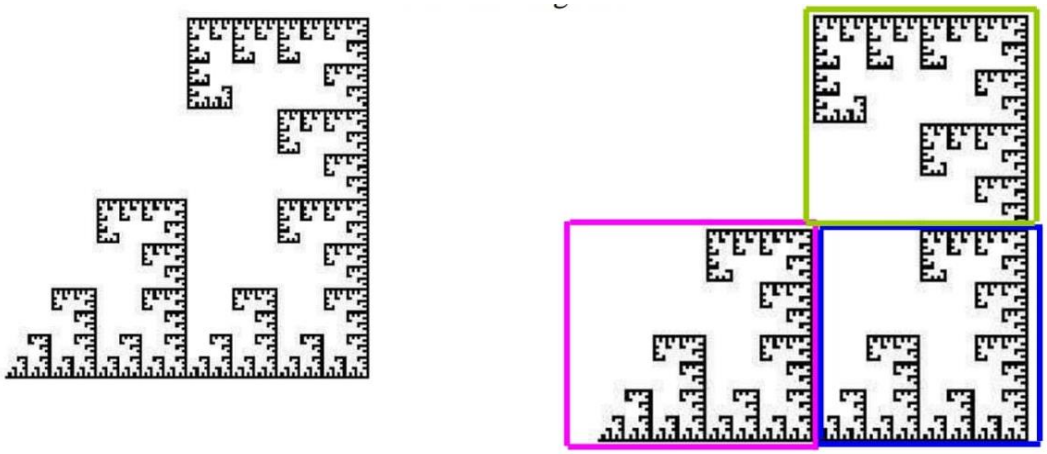
To make this more precise, we should add:

If $F(S) = S$ and S contains L, then $S = L$. That is, no larger set is 'fixed' by$F$. Without this, many F's will not work, the simplest F being the function

$F(x, y) = (x, y)$ for all (x, y).This function satisfies $F(L) = L$, but it will not generate L. (direct correspondence, G. Woodward)

4.9 Using Collage Theorem to Determine Underlying Functions of a Fractal:

My next challenge is to use The Collage Theorem to determine the functions that give me the following fractal:



I started by staring at the picture. I viewed this as part of a square with vertices (0,0), (0,1), (1,0), and (1,1). With a little prodding, I did see three, smaller replicas of the big picture. One of the pictures was simply a shrink of the big picture. Both dimensions were half of the original, x and y by the vertices used for the original image. (Marked in pink on the above diagram.) This gave me one of my functions:

$$f(x,y) = (\frac{x}{2}, \frac{y}{2})$$

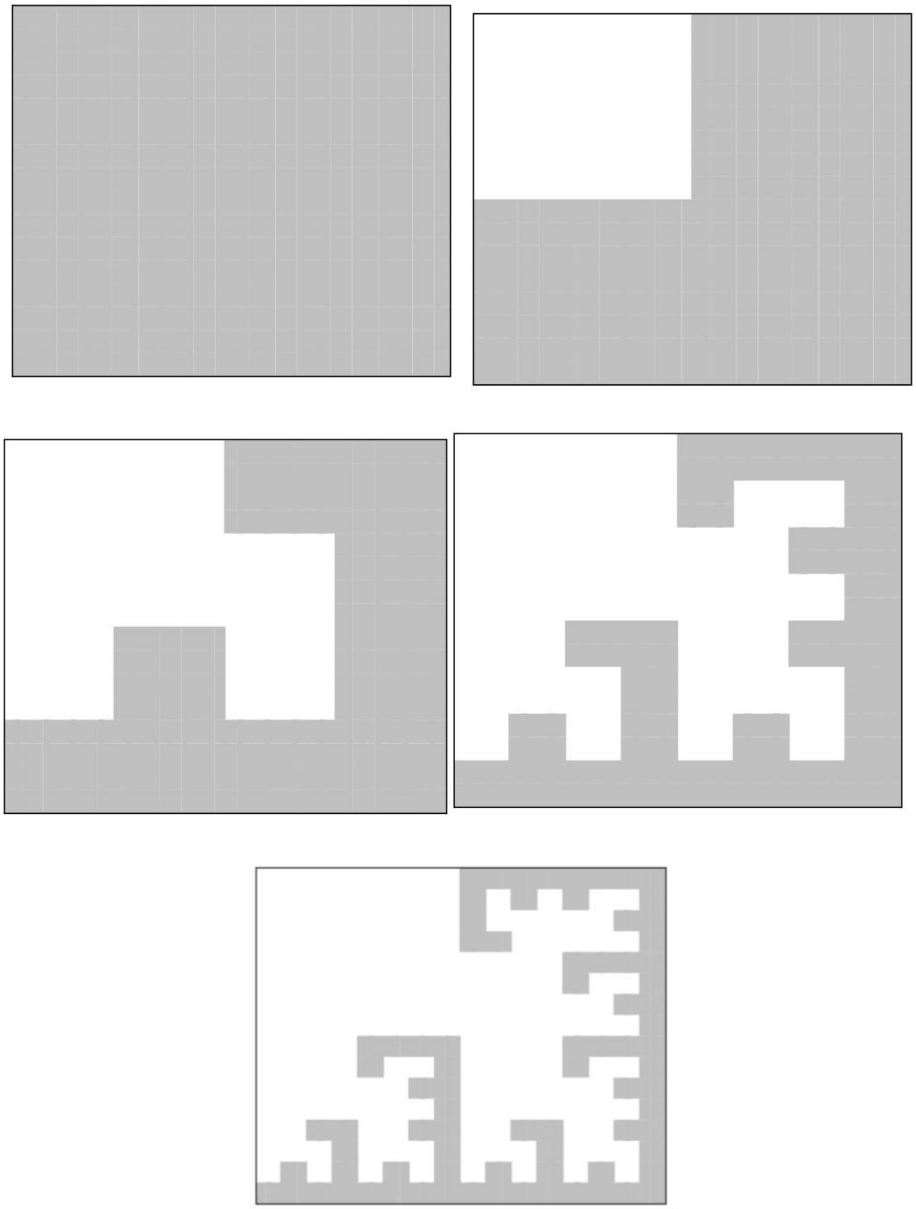
Another was a translation of the shrunken image. I determined that it was translated ½ to the right. (Marked in blue on the above diagram.) This allowed me to find the second function:

$$f(x,y) = \left(\frac{x}{2} + \frac{1}{2}, \frac{y}{2}\right)$$

It took me quite a while to find the third formula because I was not finding the third image! Finally, and with some assistance, I found the shrunken image translated to the right and up, and then rotated. (Marked in green on the above diagram.) Because of the rotational relationships found previously, this allowed me to write the third function:

$$f(x,y) = \left(\frac{-y}{2} + 1, \frac{x}{2} + \frac{1}{2}\right)$$

Now, I needed to test my formulas. I decided to use a 1x1 square as my initial (S_0) shape. I then applied four iterations, shown in progression below. If I were to continue, I would indeed have my starting image. My functions are appropriate for the fractal.



4.10 Algorithmic Generation of Fractals Using IFS & Collage Theorem:

In this section we take time out from the mathematical development to provide two algorithms for rendering pictures of attractors of an IFS.

The two algorithms presented are :

1. The Deterministic Algorithm

2. The Random Iteration Algorithm

Now that we know how to generate the attractor of our IFS, or at least what we should aim for when designing our IFS, we can implement our methods using the algorithms discussed in this section. Both the algorithms use the ideas that result from the Collage Theorem, which tells us that if the distance between our desired fractal and the union of the fractal under contractive mappings is small, then our IFS will converge to a unique attractor, whose distance to the fractal will be small. Thus, if we choose our IFS wisely, it will

produce an attractor which will resemble our desired fractal. Luckily, we already have some parameters from the examples in Barnsley and Peitgen et al and we will use those for our examples.

4.10.1 The Deterministic Algorithm:

This method starts with an image and apply some affine transformation on each subset of this image and try to find out next image which should be complete subset of $\mathbb{R}^2$space where image lie. After applying affine transformation again and again a sequence of image will be generated which should be converging at some point which will be the limit point? This limit point is nothing but an image. This apply some mapping to get an image from other image, this mapping should be contractive. This approach to generate fractal requires heavy amount of memory, because in each iteration generate some image and to store image generated by *affine transformation* requires large amount of memory.

Let $\{X;\ w_n, n = 1,2, \dots, N\}$ be a hyperbolic IFS. Choose a compact set $A_0 \subset \mathbb{R}^2$. Then compute successively $A_n = W^{0n}(A)$ according to

$$A_{n+1} = \bigcup_{j=1}^{N} w_j(A_n), for\ n = 1,2, \dots$$

Thus construct a sequence$\{A_n : n = 0,1,2, \dots\} \subset \mathcal{H}(X)$. Then by the IFS Theorem the ***sequence*** $\{\ A_n\}$ converges to the attractor of the IFS in the Hausdorff metric.

So, in short,this algorithm is an example of using the Hutchinson operator from the Collage Theorem. We draw a rough outline of the object and then cover it as closely as possible by a number of smaller similar or affine copies to form a sort of "collage". Then we iterate with our resulting collage until we start to see our desired fractal-like image emerge.

Using the terms we discussed above: we take any initial set H such as a unit square and iterate the operator *W* from our IFS *{w_1, w_2, . . . w_n} k* times to get the *k* th iterate/approximation W_k *(H)*to *H* for a suitable value *k*. Then for *k* large enough, we get the important identity *W(H) = H* where the *k* th iterate of the square starts to resemble the fractal and becomes the fixed point of our IFS.

The deterministic algorithm can also be described with an analogy of the Multiple Reduced Copy Machine (MRCM) with multiple lenses. As input, it will take an image and set of parameters for each lens. The parameters specify values for our transformations on our given image. The kinds of transformations that MRCM takes are called affine linear transformations, which include scaling, shearing, reflection, rotation and translation. They take the parameters *r, s, θ, ψ, e, f* where

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} r\cos\theta & -s\sin\psi \\ r\sin\theta & s\sin\psi \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

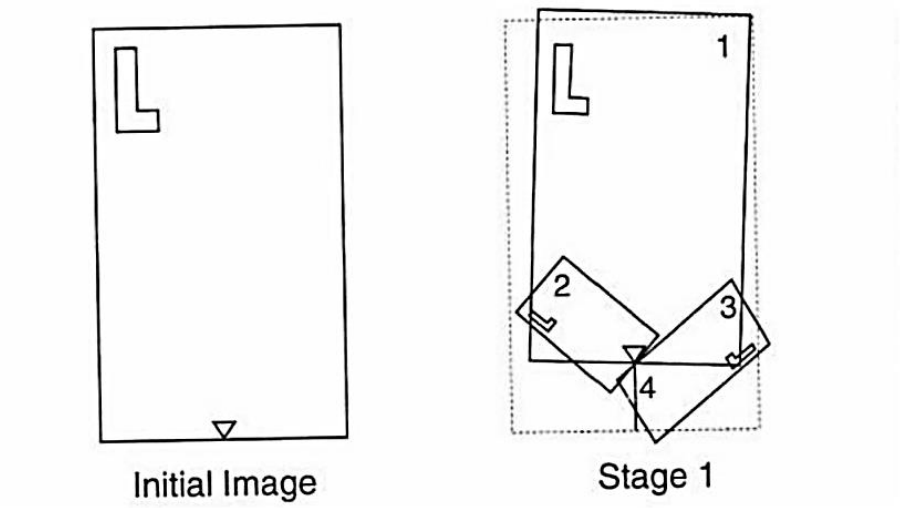
which we can simplify with the coefficients a, b, c, d, e, f and the matrix

$$\begin{bmatrix} a & b & e \\ c & d & f \end{bmatrix} where \begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} ax + by + e \\ cx + dy + f \end{bmatrix}$$

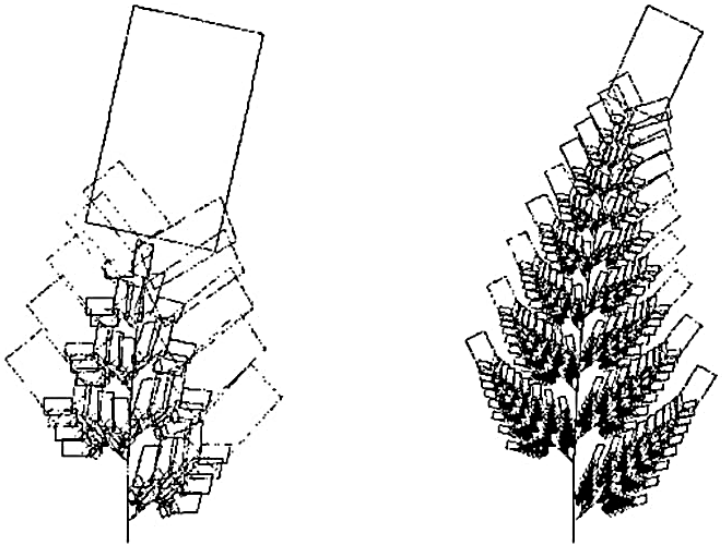
As output, it will assemble all of the transformed images ionto one image. The resulting image is then fed back into the machine as the input for iterative process.

Example:

To create a "blueprint" for the Barnsley fern, we have the following picture. One of the transformations make most of the fern, two transformations make the smaller leaves, and the last transformation forms the stem.



Then if we feed the blueprint back into the machine and then iterate successively with the obtained collages, we will eventually produce a relatively detailed fern. The fern on the left of the following figure is after 5 iterations, and on the right after 10 iterations.



However after a few more iterates, this process will take a considerably more time. Take as our initial set a rectangle whose length takes up 1000 pixels. Then if we want to reproduce our fern so that each rectangle becomes a point or 1 pixel, we use this formula to calculate how many iterations N we need to do. Since our fern contracts 85% in our blueprint, N is given by

$$1 = 1000 \times .85^{N}.$$

Thus it will take about iterations in order to get our fern to be somewhat detailed. Then we would have to calculate 42 iterates, which amounts to drawing $1 + 4^1 + 4^2 + \cdots + 4^N = (4^{N+1} - 1)/3$ rectangles. For N = 42, this is a lot of rectangles - on the order of 2×10^{25}. Even with a very fast computer, this would take many many years.

4.10.2 The Random Iteration Algorithm:

The random approach is different from the deterministic approach in that the initial set is a singleton point and at each level of iteration, just one of the defining affine transformations is used to calculate the next level. Which will also be a singleton point? At each level, the affine transformation is ***randomly selected and applied***. Points are plotted, except for the early ones, and are discarded after being used to calculate the next value. The random algorithm avoids the need of a large computer memory, it is best suited

for the small computers on which one point at a time can be calculated and display on a screen. On the other hand it takes thousand of dots to produce an image in this way that does not appear too skimpy.

Let $\{X; w_1, w_2,\ldots, w_N ; p_1, p_2, \ldots, p_N\}$ be an IFS with probabilities. Choose $x_0 \in X$ and then choose recursively, independently,

$$x_n \in \{w_1(x_{n-1}), w_2(x_{n-1}), \ldots, w_N(x_{n-1})\} \quad \text{for n = 1, 2, 3, \ldots,}$$

where the probability of the event $x_n = w_i (x_{n-1})$ is p_i . Thus, construct a sequence $\{x_n: n = 0, 1, 2, 3,\ldots\} \subset X$.

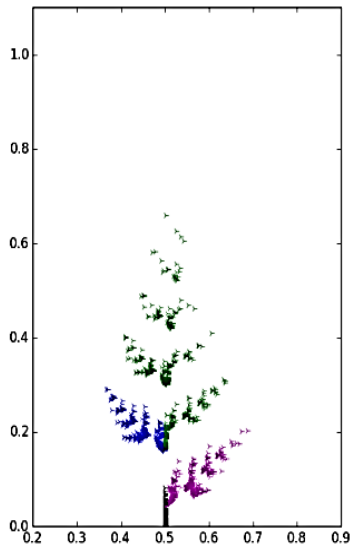
This algorithm uses the idea of the chaos game. The chaos game was formulated by Michael Barnsley as method to generate classical fractals. The game goes as follows:

1. Pick a polygon and any initial point inside that polygon.
2. Choose a vertex of the polygon at random.
3. Move a fraction of the length towards the chosen vertex.
4. Repeat steps (2-3) with the new point.

By successively iterating in this way, we will produce our fractal shape. This method will plot points in random order all over the attractor.

Example:

With a more natural fractal that doesn't necessarily have strictly self-similar properties, we will be fairly disappointed with our results when we assign equal probabilities, even with 10000 iterations.



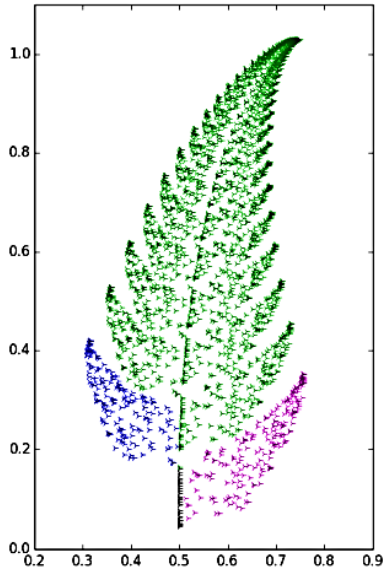
i	a	b	c	d	e	f
1	.849	.037	-.037	.849	.075	.1830
2	.197	-.266	.226	.197	.4	.0470
3	-.15	.283	.26	.237	.575	-.084
4	0	0	0	.160	.5	0

To improve on this picture, we can do two things: throw out the first few thousand iterations and assign probabilities to each transformation. Since our sequence is converging to the attractor, we don't need to plot the points that we obtained in the beggining of the sequence. The probabilities are mass distributions given by

$$p_i \approx \frac{|\det w_i|}{\sum_{i=1}^{N} |\det w_i|}$$

where the sum of all the probabilities is 1. The theory behind this involves measures and is beyond the scope of this project, but we can use this to produce much more satisfying result with the same number of iterations.

i	1	2	3	4
p	.01	.85	.07	.07



Our refined algorithm, taking in account of mass distributions is as follows:

1. Take any initial point $x_0 \in H$.

2. Choose a transformation w_i chosen based on probability p_i .

3. Apply the chosen transformation onto our point.

4. Repeat steps (2-3) with the new point.

This method will *converge to our attractor* much quicker than with equal probabilities.

5. Using IFS & The Collage Theorem to Generate One Fractal Models

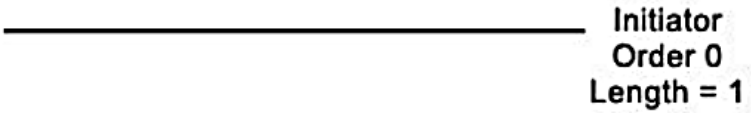
With the help of such codes or algorithm, we can create many beautiful IFS fractals. One of such IFS fractals discussed in the project are namely, ***'Koch Snowflake'.***

5.1 Koch Snowflake:

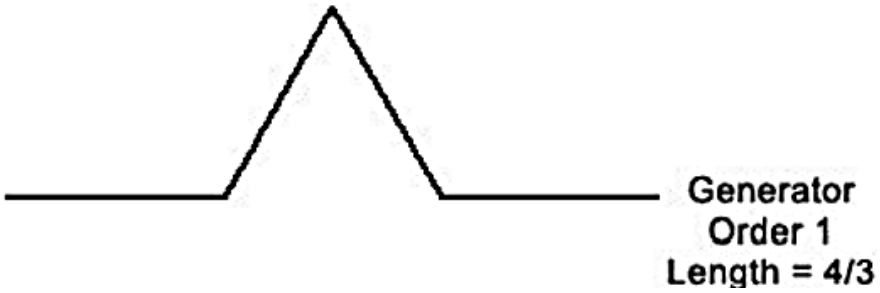
The Koch Snowflake is a fractal curve and is one of the earliest fractals to have been described. It is based on yet another fractal curve, known as the 'Koch Curve', which appeared in a 1904 paper titled, "On a Continuous Curve without Tangents, Constructible from Elementary Geometry", by the Swedish Mathematician, ***'Helge von Koch'***.

5.1.1 Koch Curve & it's Construction:

***STEP k=0*:** The initiator of this fractal curve is a unit line segment, that is length $L_0 = 1$.



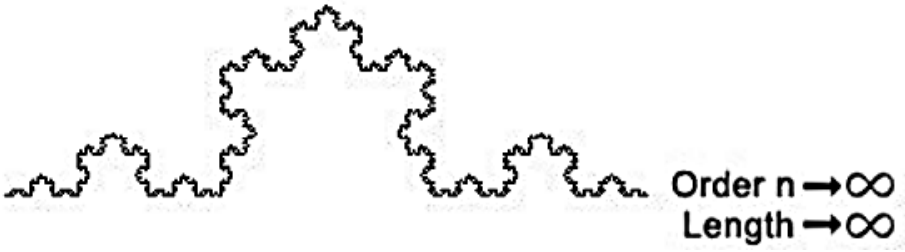
***STEP k=1*:** This unit line segment is then divided into three equal parts and the middle third is removed. The middle third is then replaced with two equal line segments of one-third in length, which forms an equilateral triangle. This step is the generator and the length of this prefractal is, $L_1 = \frac{4}{3} = \frac{4}{3} \times 1 = \frac{4}{3} \times L_0$.



***STEP k=2*:** Now, the middle third part is removed from each of the four line segments and each is replaced by two equal line segments of length 1/9 just as in the previous step. Hence, forming the prefractal with length, $L_2 = \frac{16}{9} = \frac{4}{3} \times \frac{4}{3} = \frac{4}{3} \times L_1$.

***STEP k=3*:** Similarly, we repeat the process on each of the 16 line segments to obtain the prefractal having length, $L_3 = \frac{64}{27} = \frac{4}{3} \times \frac{16}{9} = \frac{4}{3} \times L_2$.

After infinite iterations of the process we obtain the self-similar fractal, the Koch Curve.



- A noticeable property of this fractal is that it is seemingly infinite in length. At each step k of the construction, the length of the prefractal curve increases to $\frac{4}{3}L_{k-1}, k = 1,2,3, \ldots$, where L_k is the length of the prefractal curve at step k. As the number of generations increase, the length of the curve diverges. It is therefore apparent that 'length' is not a useful measure of the Koch Curve as its length is infinite. Also, it can be shown that Koch curve is effectively constructed from corners and hence no unique tangent occurs anywhere upon it. Thus, the Koch curve is not a smooth curve and is nowhere differentiable.

5.1.2 IFS Code for the Koch Curve:

The first iteration of the Koch curve consists of taking four copies of the unit line segment, each scaled by ε=1/3. Two segments must be rotated by 60°, one clockwise and the other, counter clockwise. Along with the required translations, this yields the following IFS:

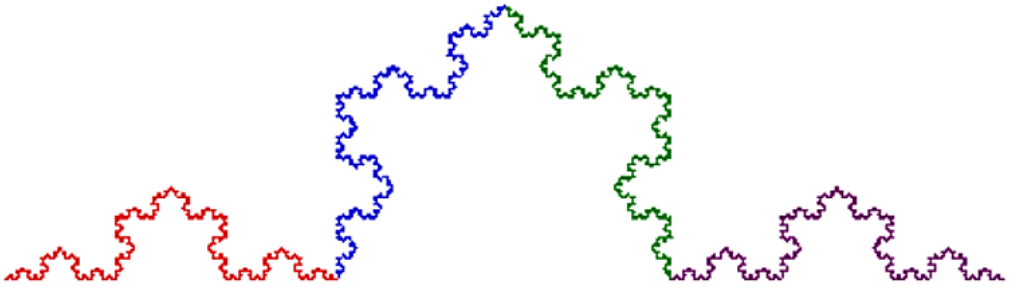
$$w_1\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}\frac{1}{3} & 0\\ 0 & \frac{1}{3}\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \begin{pmatrix}0\\0\end{pmatrix}, \text{scale by } \varepsilon.$$

$$w_2\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}\frac{1}{6} & \frac{-\sqrt{3}}{6}\\ \frac{\sqrt{3}}{6} & \frac{1}{6}\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \begin{pmatrix}\frac{1}{3}\\0\end{pmatrix}, \text{scale by } \varepsilon, \text{rotate by } 60°.$$

$$w_3\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}\frac{1}{6} & \frac{\sqrt{3}}{6}\\ \frac{-\sqrt{3}}{6} & \frac{1}{6}\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \begin{pmatrix}\frac{1}{2}\\\frac{\sqrt{3}}{6}\end{pmatrix}, \text{scale by } \varepsilon, \text{rotate by } -60°.$$

$$w_4\begin{pmatrix}x_1\\x_2\end{pmatrix} = \begin{pmatrix}\frac{1}{3} & 0\\ 0 & \frac{1}{3}\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \begin{pmatrix}\frac{2}{3}\\0\end{pmatrix}, \text{scale by } \varepsilon.$$

The fixed attractor of this IFS is the Koch Curve.

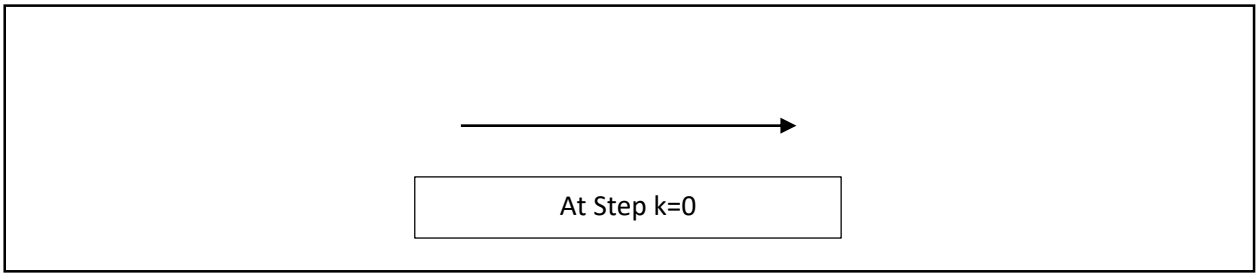


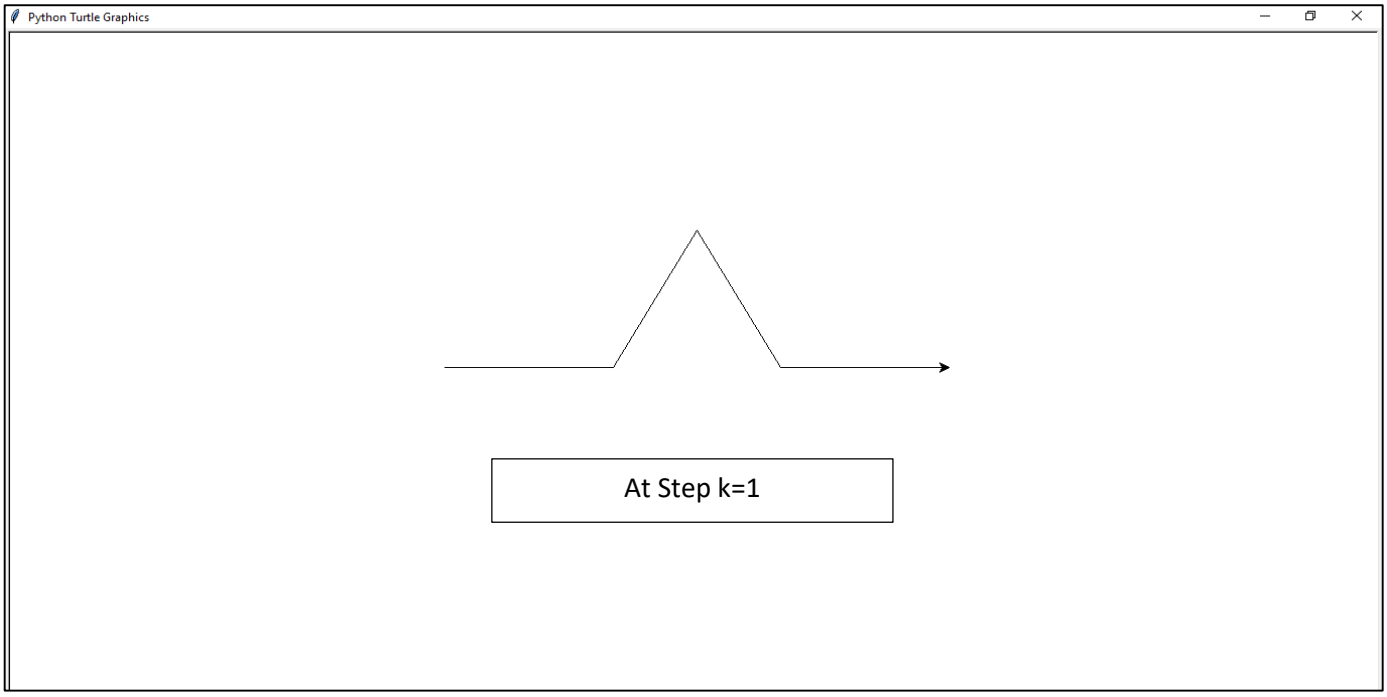
5.1.3 Python Code for Koch Curve:

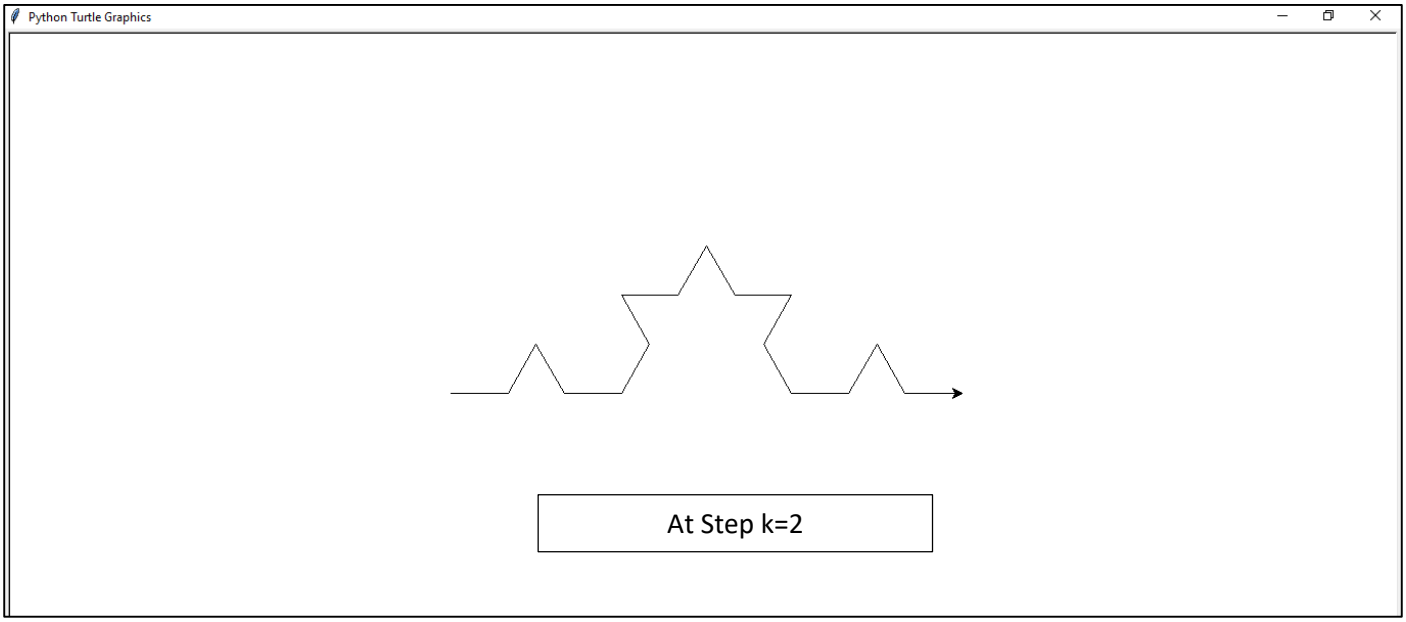
```
# Python program to print partial Koch Curve.
# importing the libraries : turtle standard
# graphics library for python
from turtle import *

#function to create koch snowflake or koch curve
def snowflake(lengthSide, levels):
    if levels == 0:
        forward(lengthSide)
        return
    lengthSide /= 3.0
    snowflake(lengthSide, levels-1)
    left(60)
    snowflake(lengthSide, levels-1)
    right(120)
    snowflake(lengthSide, levels-1)
    left(60)
    snowflake(lengthSide, levels-1)

# main function
if __name__ == "__main__":
```

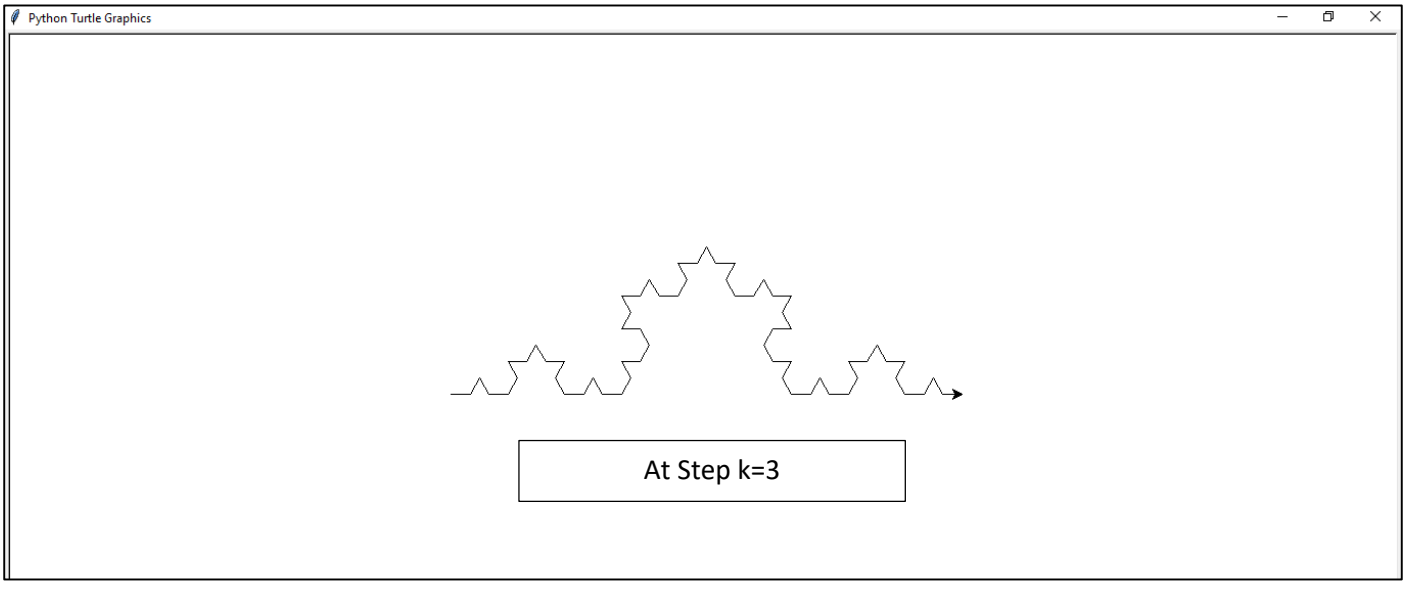
```
# defining the speed of the turtle
speed(0)
length = 500.0

# Pull the pen up – no drawing when moving.
penup()

# Move the turtle backward by distance,
# opposite to the direction the turtle
# is headed.
# Do not change the turtle’s heading.
backward(length/2.0)

# Pull the pen down – drawing when moving.
pendown()

snowflake(length, 5)

# To control the closing windows of the turtle
mainloop()
```

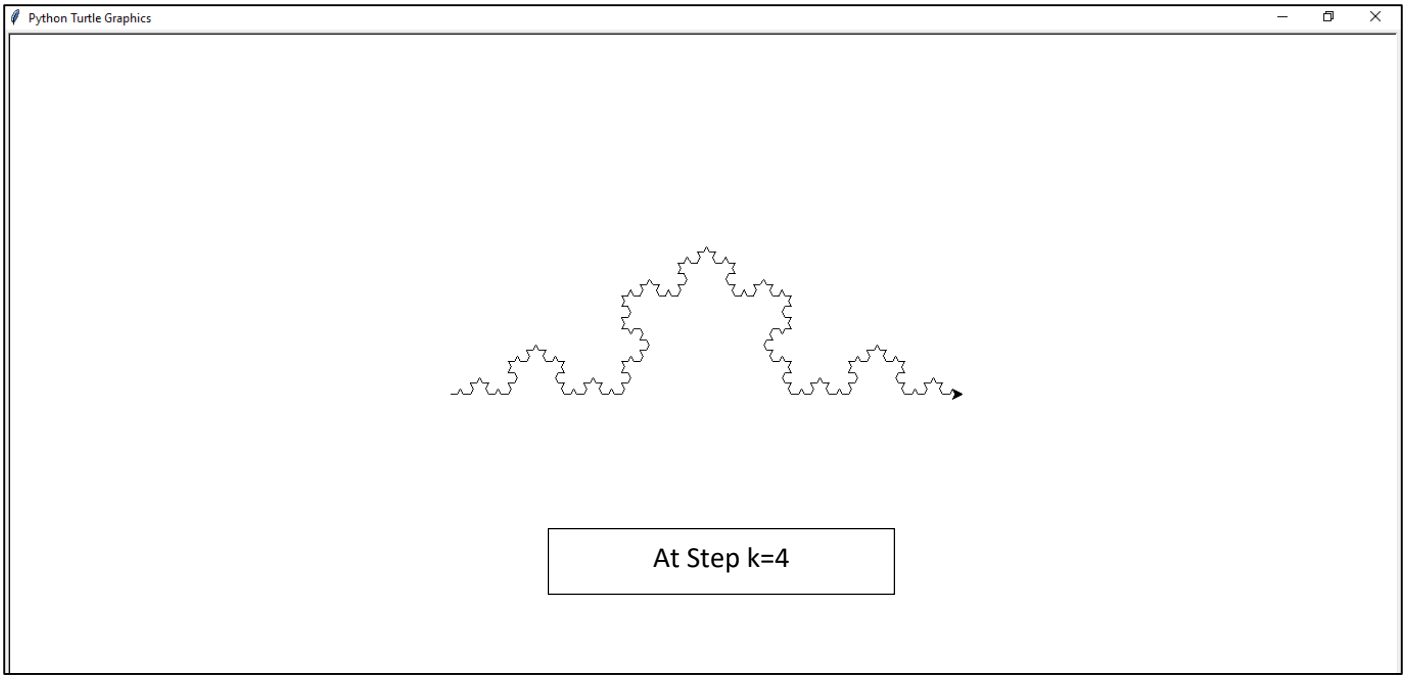
Output:



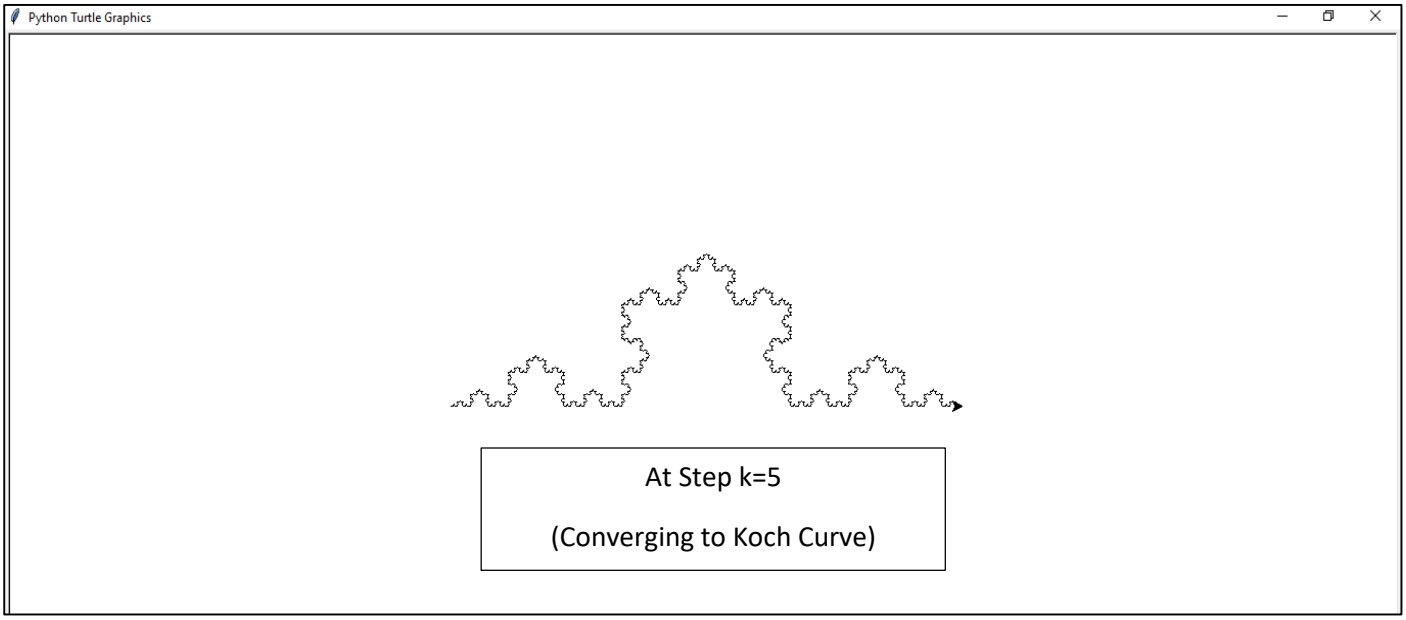

Python Turtle Graphics
At Step k=1


Python Turtle Graphics
At Step k=2


Python Turtle Graphics
At Step k=3



At Step k=4

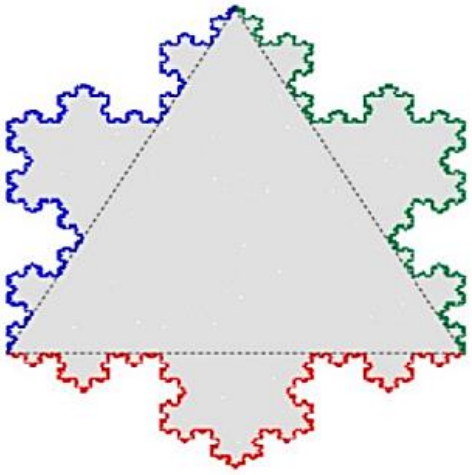


At Step k=5

(Converging to Koch Curve)

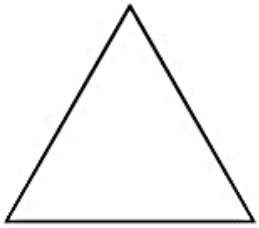
5.1.4 Construction of Koch Snowflake:

As mentioned earlier, Koch Snowflake is based on the fractal, Koch Curve. It is basically three copies of Koch Curve placed outward around the three sides of an equilateral triangle.

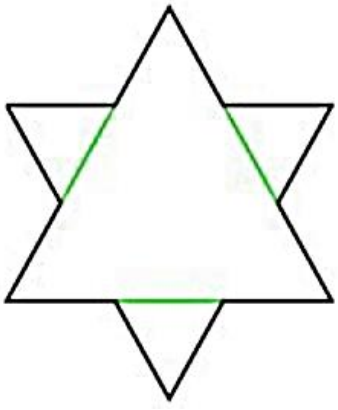


Koch snowflake is also known as the Koch Island or Koch Star. The construction is as follows:

STEP k=0: The initiator of the fractal is an equilateral triangle, of side length say, a. Area of the triangle is, $A_0 = \frac{\sqrt{3}}{4}a^2$.



***STEP k=1*:** Each side of the triangle is then divided into three equal parts. The middle third part of each line segment is removed and then replaced by two equal line segments of length a/3, thus giving us the following prefractal:



(The green part in the figure denotes the removed part). This step is the generator. In this step we basically added three small equilateral triangles of side length a/3 on the sides of the initial triangle. therefore the area is given by:

$$A_1 = A_0 + 3 \times \frac{\sqrt{3}}{4}\left(\frac{a}{3}\right)^2$$

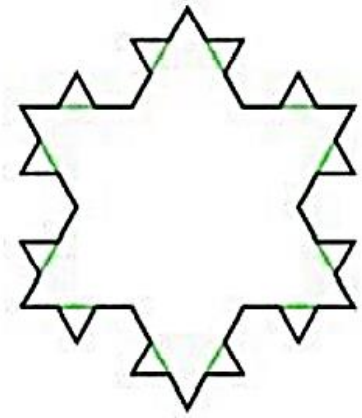
$$= A_0 + \frac{1}{3} \times \frac{\sqrt{3}}{4} a^2$$

$$= A_0 + \frac{1}{3} A_0$$

This can be rewritten as,

$$A_1 = A_0 + \frac{3 \times 4}{9} A_0.$$

STEP k=2: We repeat the same process on each of the 12 line segments and obtain the following prefractal curve:



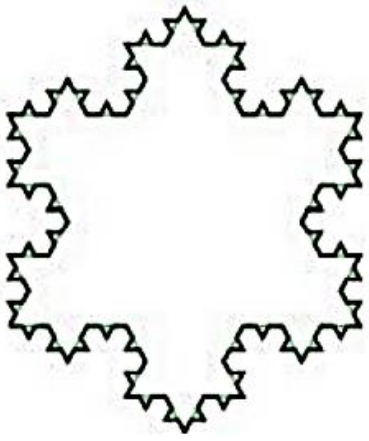
In this step as well, we have added 12 smaller equilateral triangles, each of side length a/9 this time. Therefore, the area bounded by this curve is given by:

$$A_2 = A_1 + 12 \times \frac{\sqrt{3}}{4}\left(\frac{a}{9}\right)^2$$

$$= A_1 + \frac{4}{27} \times \frac{\sqrt{3}}{4} a^2$$

$$= A_1 + \frac{3 \times 4}{9^2} A_0.$$

STEP k=3: Repeating the same process, now on 48 line segments we obtain:

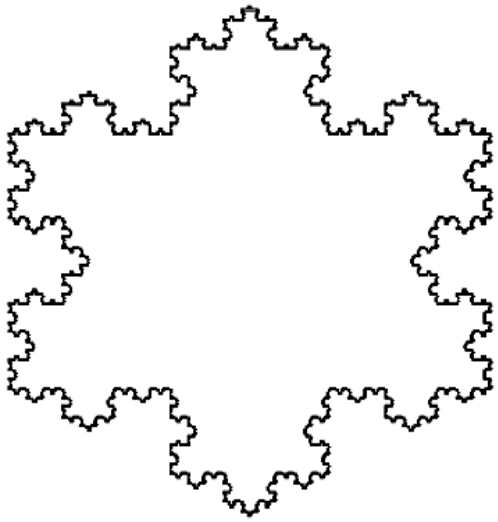


In the similar way, we can obtain the area bounded by this curve as well, which is

$$A_3 = A_2 + 48 \times \frac{\sqrt{3}}{4}\left(\frac{a}{27}\right)^2$$

$$= A_2 + \frac{3 \times 4^2}{9^3} A_0.$$

And hence, on iterating this process for infinite many times, we obtain our fractal:



Now, in the consecutive steps of the construction, we obtained the following relations:

$$A_0 = \frac{\sqrt{3}}{4} a^2,$$

$$A_1 = A_0 + \frac{3 \times 4}{9} A_0,$$

$$A_2 = A_1 + \frac{3 \times 4}{9^2} A_0,$$

$$A_3 = A_2 + \frac{3 \times 4^2}{9^3} A_0.$$

Therefore, in general we have,

$$A_k = A_{k-1} + \frac{3 \times 4^{k-1}}{9^k} A_0, for\ k \geq 1,$$

and where A_k is the area bounded by the curve obtained at k-th step; and

$$A_0 = \frac{\sqrt{3}}{4} a^2.$$

Now, substituting the values of $A_{k-1}, A_{k-2}, \ldots, A_1$ in the formula for A_k, we get:

$$A_k = \left(A_0 + \frac{3 \times 4^0}{9} A_0\right) + \left(\frac{3 \times 4^1}{9^2} A_0 + \frac{3 \times 4^2}{9^3} A_0 + \cdots + \frac{3 \times 4^{k-2}}{9^{k-1}} A_0 + \frac{3 \times 4^{k-1}}{9^k} A_0\right)$$

$$= \left(\frac{4}{3} A_0\right) + \left[\sum_{n=2}^{k} \left(\frac{3 \times 4^{n-1}}{9^n}\right) A_0\right]$$

$$= \left[\frac{4}{3} + \frac{1}{3} \times \sum_{n=2}^{k} \left(\frac{4^{n-1}}{9^{n-1}}\right)\right] A_0$$

$$\Rightarrow A_k = \left[\frac{4}{3} + \frac{1}{3} \times \sum_{n=2}^{k} \left(\frac{4}{9}\right)^{n-1}\right] A_0$$

Now, $\sum_{n=2}^{k} \left(\frac{4}{9}\right)^{n-1} = \frac{4}{9}\left(1 + \frac{4}{9} + \left(\frac{4}{9}\right)^2 + \cdots + \left(\frac{4}{9}\right)^{k-1}\right)$ is a geometric series whose sum is given by:

Sum, $s = \frac{4}{9}\left[\frac{1-\left(\frac{4}{9}\right)^{k-1}}{1-\frac{4}{9}}\right] = \frac{4}{5}\left[1 - \left(\frac{4}{9}\right)^{k-1}\right]$.

Since, $\left(\frac{4}{9}\right) < 1$, therefore as $k \to \infty$, $\lim_{k \to \infty} \left(\frac{4}{9}\right)^{k-1} = 0$.

Therefore, as $k \to \infty$, the sum of the geometric series converges to 4/5 .

And so, $\lim_{k \to \infty} A_k = \left(\frac{4}{3} + \frac{1}{3} \times \frac{4}{5}\right) A_0 = \frac{8}{5} A_0$,where, $A_0 = \frac{\sqrt{3}}{4} a^2$ =area of the initial equilateral triangle with side length a.

Hence, area of ***Koch snowflake*** $= \boldsymbol{A = \frac{2\sqrt{3}}{5} a^2}$ ***sq units***. Thus, Koch snowflake has an infinite perimeter yet it encloses a finite area.

5.1.5 IFS Code for Koch Snowflake:

The IFS code is given by:

$$w_1 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -\frac{1}{6} & \frac{\sqrt{3}}{6} \\ -\frac{\sqrt{3}}{6} & -\frac{1}{6} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} \frac{1}{6} \\ \frac{\sqrt{3}}{6} \end{pmatrix}, \text{scale by } \frac{1}{3}, \text{rotate by } -120°.$$

$$w_2 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{6} & -\frac{\sqrt{3}}{6} \\ \frac{\sqrt{3}}{6} & \frac{1}{6} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} \frac{1}{6} \\ \frac{\sqrt{3}}{6} \end{pmatrix}, \text{scale by } \frac{1}{3}, \text{rotate by } 60°.$$

$$w_3 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} \frac{1}{3} \\ \frac{\sqrt{3}}{3} \end{pmatrix}, \text{scale by } \frac{1}{3}.$$

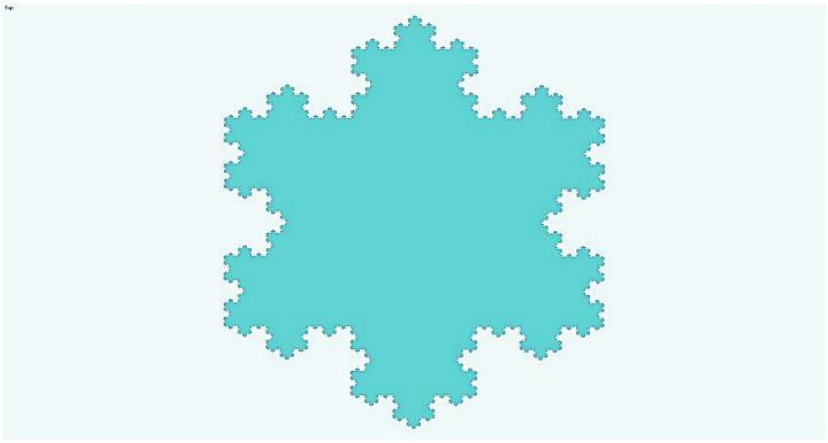
$$w_4 \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{6} & \frac{\sqrt{3}}{6} \\ -\frac{\sqrt{3}}{6} & \frac{1}{6} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} \frac{2}{3} \\ \frac{\sqrt{3}}{6} \end{pmatrix}, \text{scale by } \frac{1}{3}, \text{rotate by } -60°.$$

$$w_5\begin{pmatrix}x_1\\x_2\end{pmatrix}=\begin{pmatrix}\frac{1}{2} & -\frac{\sqrt{3}}{6}\\ \frac{\sqrt{3}}{6} & \frac{1}{2}\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix}+\begin{pmatrix}\frac{1}{3}\\0\end{pmatrix}, \text{scale by } \frac{\sqrt{3}}{3}, \text{rotate by } 30°.$$

$$w_6\begin{pmatrix}x_1\\x_2\end{pmatrix}=\begin{pmatrix}-\frac{1}{3} & 0\\ 0 & -\frac{1}{3}\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix}+\begin{pmatrix}\frac{2}{3}\\0\end{pmatrix}, \text{scale by } \frac{1}{3}, \text{rotate by } 180°.$$

$$w_7\begin{pmatrix}x_1\\x_2\end{pmatrix}=\begin{pmatrix}\frac{1}{3} & 0\\ 0 & \frac{1}{3}\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix}+\begin{pmatrix}\frac{2}{3}\\0\end{pmatrix}, \text{scale by } \frac{1}{3}.$$

The attractor for this IFS is the Koch Snowflake.



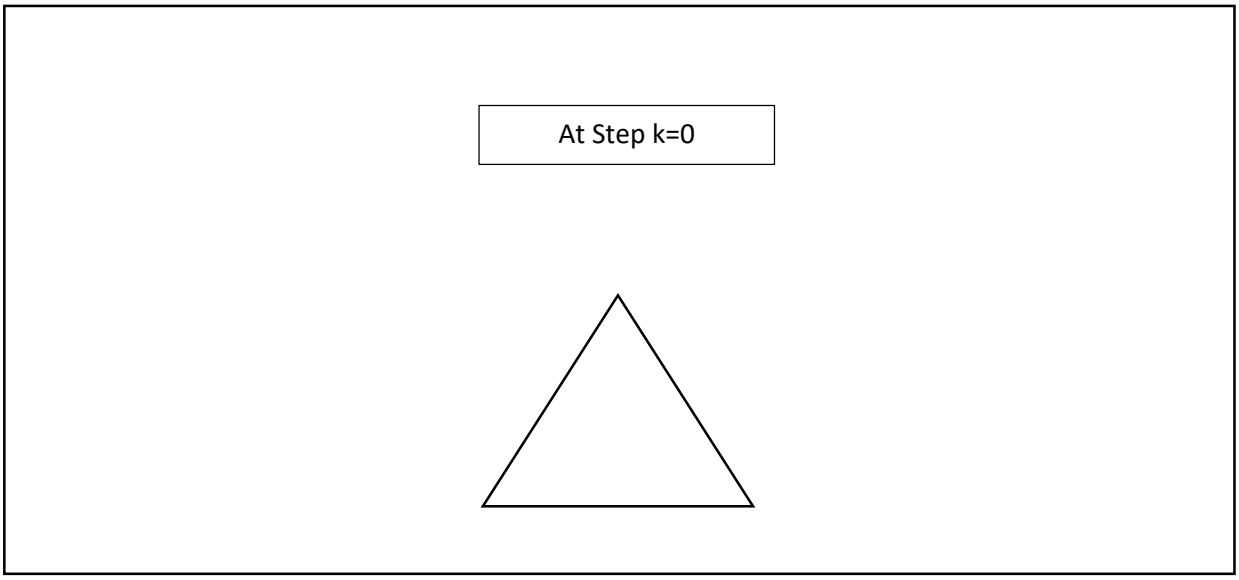
5.1.6 Python Code for Koch Snowflake:

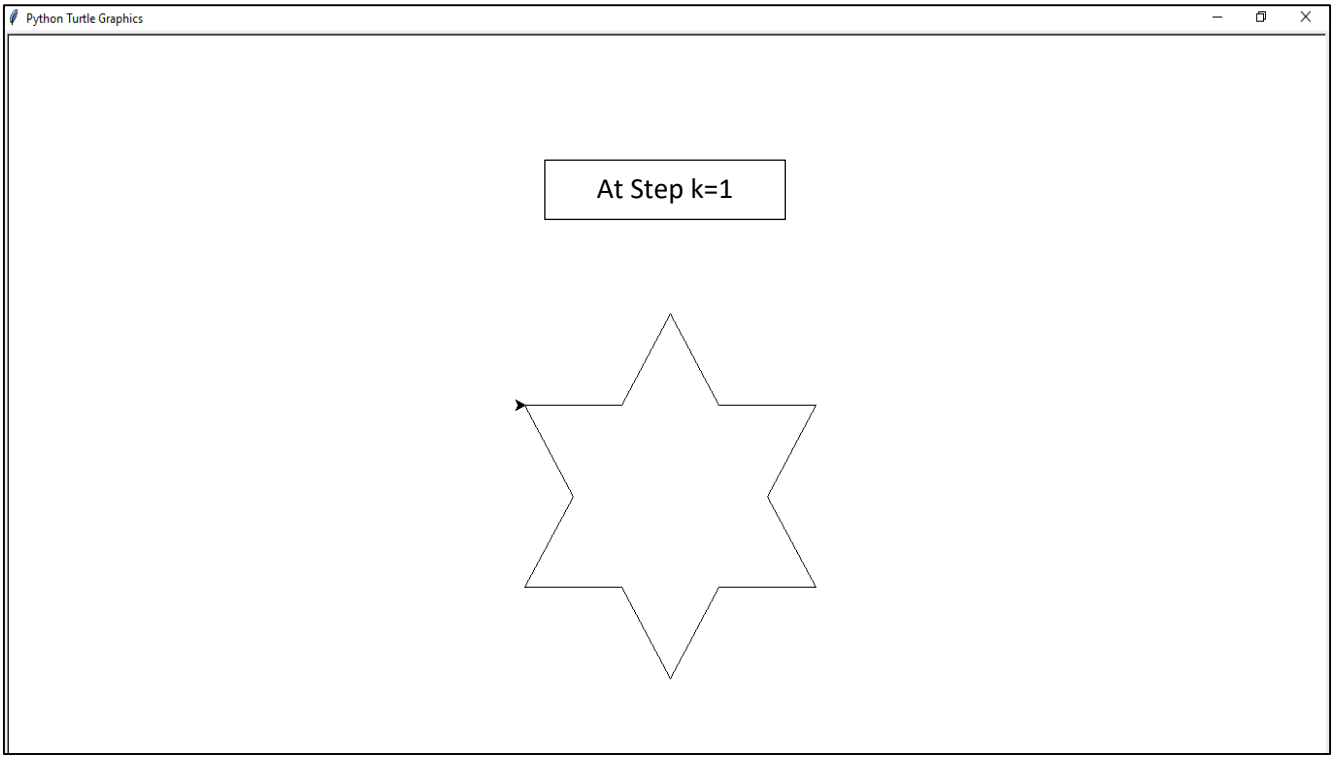
To create a full snowflake with Koch curve, we need to repeat the same pattern three times.

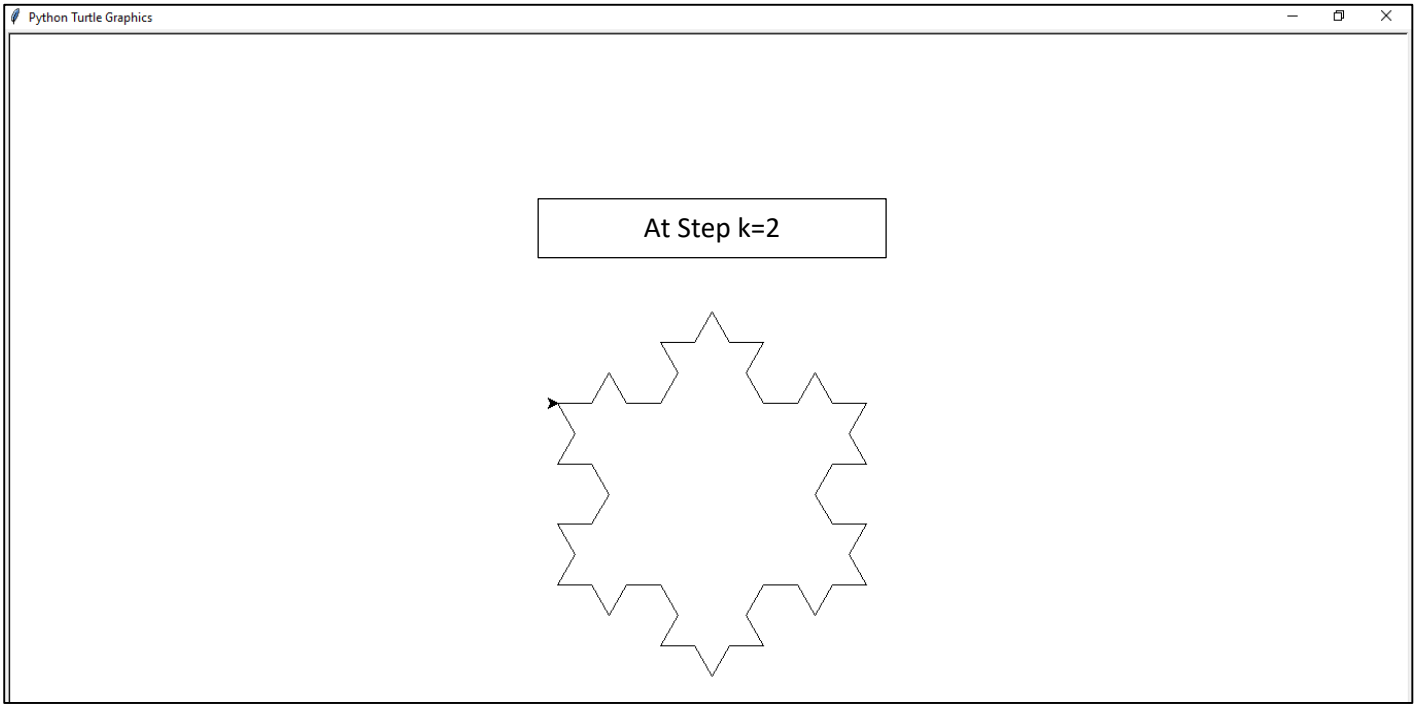
```
# Python program to print complete Koch Curve.
from turtle import *

# function to create koch snowflake or koch curve
def snowflake(lengthSide, levels):
    if levels == 0:
        forward(lengthSide)
        return
    lengthSide /= 3.0
    snowflake(lengthSide, levels-1)
    left(60)
    snowflake(lengthSide, levels-1)
    right(120)
    snowflake(lengthSide, levels-1)
    left(60)
    snowflake(lengthSide, levels-1)

# main function
```

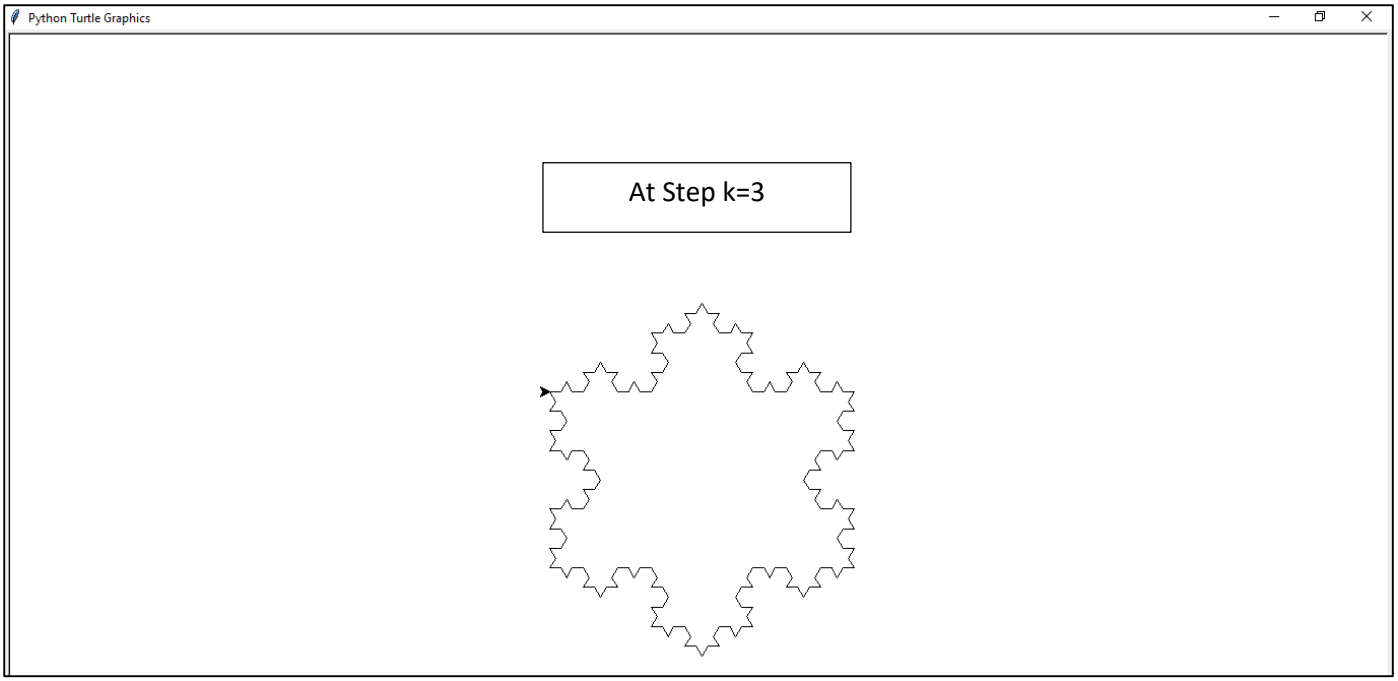
```
if __name__ == "__main__":
    # defining the speed of the turtle
    speed(0)
    length = 300.0

    # Pull the pen up – no drawing when moving.
    # Move the turtle backward by distance, opposite
    # to the direction the turtle is headed.
    # Do not change the turtle’s heading.
    penup()

    backward(length/2.0)

    # Pull the pen down – drawing when moving.
    pendown()
    for i in range(3):
        snowflake(length, 4)
        right(120)

    # To control the closing windows of the turtle
    mainloop()
```

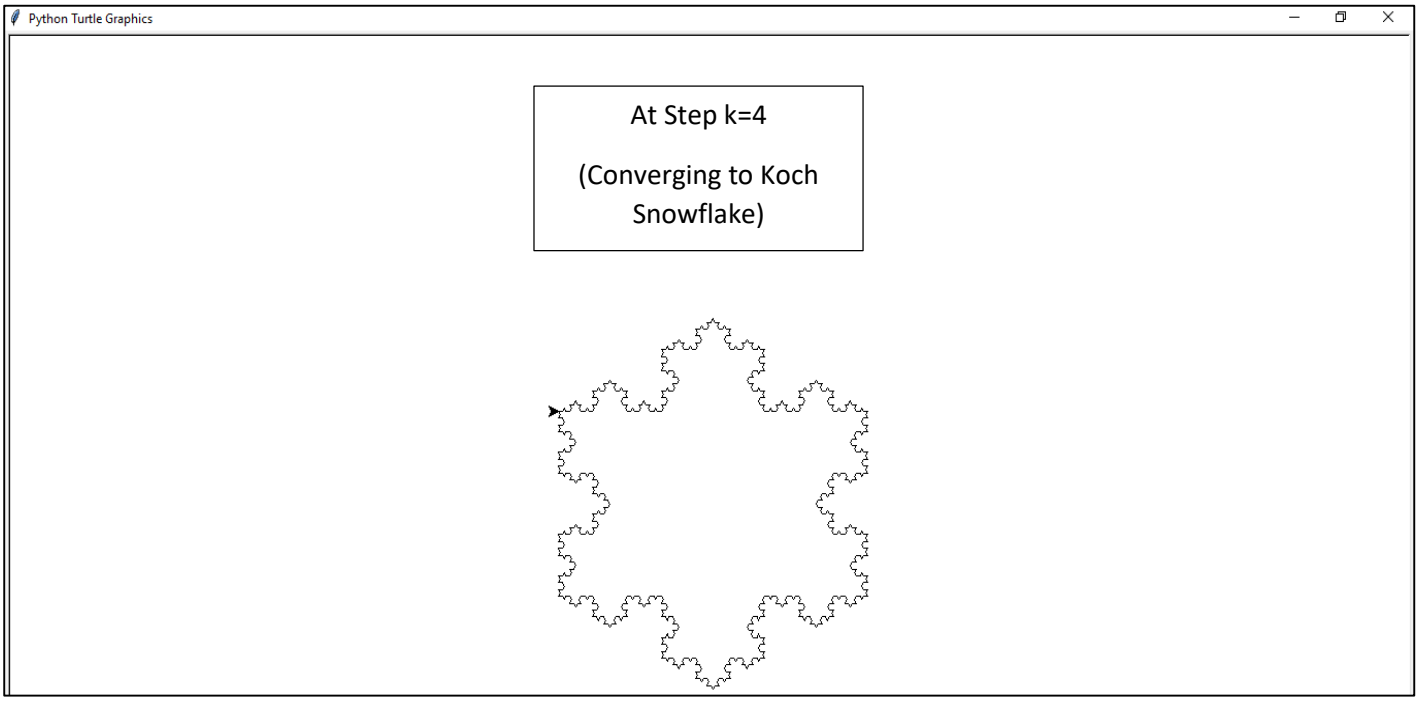
Output:




Python Turtle Graphics
At Step k=1


Python Turtle Graphics
At Step k=2


Python Turtle Graphics
At Step k=3


Python Turtle Graphics
At Step k=4
(Converging to Koch Snowflake)

6. Chaos Game(The Random Iteration Algorithm)

As we have already discussed The Random Iteration Algorithm aka The Chaos Game earlier in Section-4.6 as an implementation of IFS theorem and in Section-5.3 as an algorithm for generation of fractals using Collage theorem, in this section we will go deeper to the concept of Chaos Game.

When playing the Chaos Game it looks complicated and random, but the outcome is predetermined. Chaotic systems are complicated and their behavior appears random, but they are really deterministic. Chaotic systems don't always have to produce a pattern, though they do in the end in the Chaos Game as it generates the Sierpinski Triangle. Essentially, we shall use the Chaos Game to understand how Chaos Theory works.

6.1 IFS in Chaos algorithm:

Suppose IFS contain M affine transformation $A_1, A_2, A_3, \dots, A_M$. Each affine transformation in IFS has some associated probability $p_1, p_2, p_3, \dots, p_M$, respectively. Affine transformation with probability is applied on chaos game algorithm. In IFS select one affine transform according to its probability and apply to initial point (x_0, y_0) to find out new point (x_1, y_1) again apply another transform with another probability a new point (x_2, y_2) will be generated. Repeat this process to obtain a long sequence of points:

$$(x_0, y_0), (x_1, y_1), (x_2, y_2), \dots \dots$$

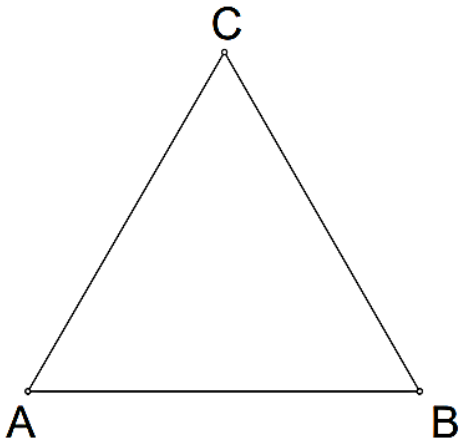
A basic result of the IFS theory is that this sequence of points will converge, with 100% probability, to the attractor of the IFS.

6.2 Creating Sierpinski Triangle using Chaos Game:

Earlier, we created the Sierpinski Triangle using functions. However, can we also create it using a different type of process? Perhaps we can generate the Sierpinski Triangle by random luck. This can be illustrated by the Chaos Game.

6.2.1 Process for playing the game:

- To play the Chaos Game, start with an isosceles triangle with the vertices labelled A, B, and C.



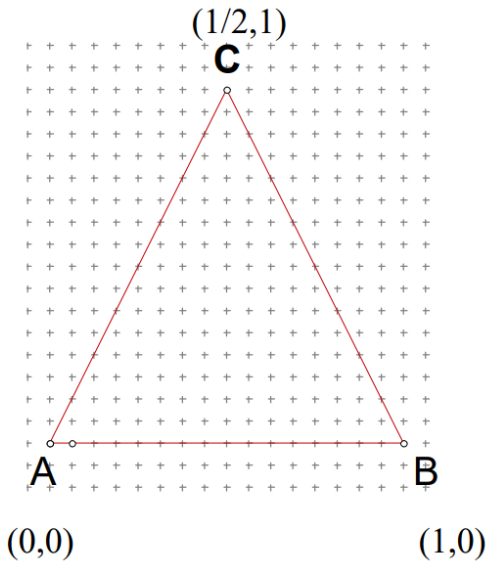
- We will now assign each vertex numbers of the die. Vertex A will be the numbers 1 or 2. Vertex B will be the numbers 3 or 4. Vertex C will be the numbers 5 or 6.
- We first start playing the game by picking a point. Let's make our starting point C.

- Whatever number we roll we will move half the distance toward the numbered vertex and plot a point. We will remain at that point and roll the die again.
- We will then move half the distance toward the numbered vertex and plot a point and remain there.
- We will repeat the process over and over and over. Actually, we will repeat this process an infinite number of times.
- We might hypothesize what image we will get in the end. Since the dots are being ***generated randomly***, we may think that we should end up with dots that are ***scattered chaotically***.

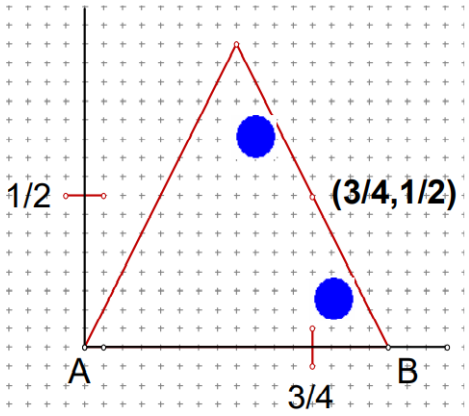
6.2.2 Play the Game:

We could now sit down and play the game. We will need a die, graph paper, a ruler, and a pencil. Let's pick the starting point C and start playing the game.

We can use a 16 x 16 grid with vertices at (0,0), (1,0), and (1/2,0).



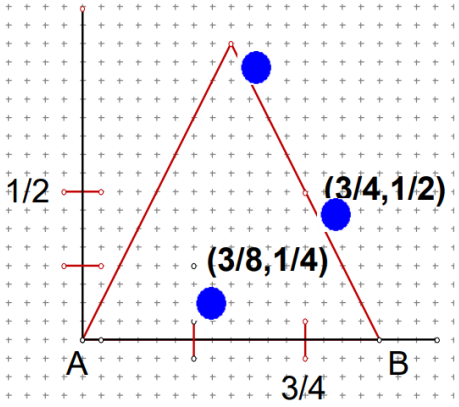
We can now roll the die and take note of what number we rolled. In this case, we rolled a three so we will use a ruler to measure half the distance from C toward B and plot a point and remain there.



Notice that we landed at (3/4, 1/2).

We can now roll again and repeat the same process of the Chaos Game.

This time we rolled a two and we will measure half the distance toward A and plot a point and remain there. Notice that we landed at (3/8, 1/4).



We can now see that this process of rolling the die and measuring with a ruler may take us more time than we have to devote to playing the Chaos Game. In addition to using a ruler to find the midpoint, we could also find the mean of the coordinates of the point we are at and the point we are moving toward. However, this would also take a great deal of time. Even if we roll the die 300 times and plot 300 points, we may not see a pattern. Now is when computers are a great asset to our goal of understanding the Chaos Game. Computers can be programmed easily to plot the points for us and we can take advantage of the available technology. Let's use the computer to jump ahead and see what image we get after 1,000 rolls of the die.

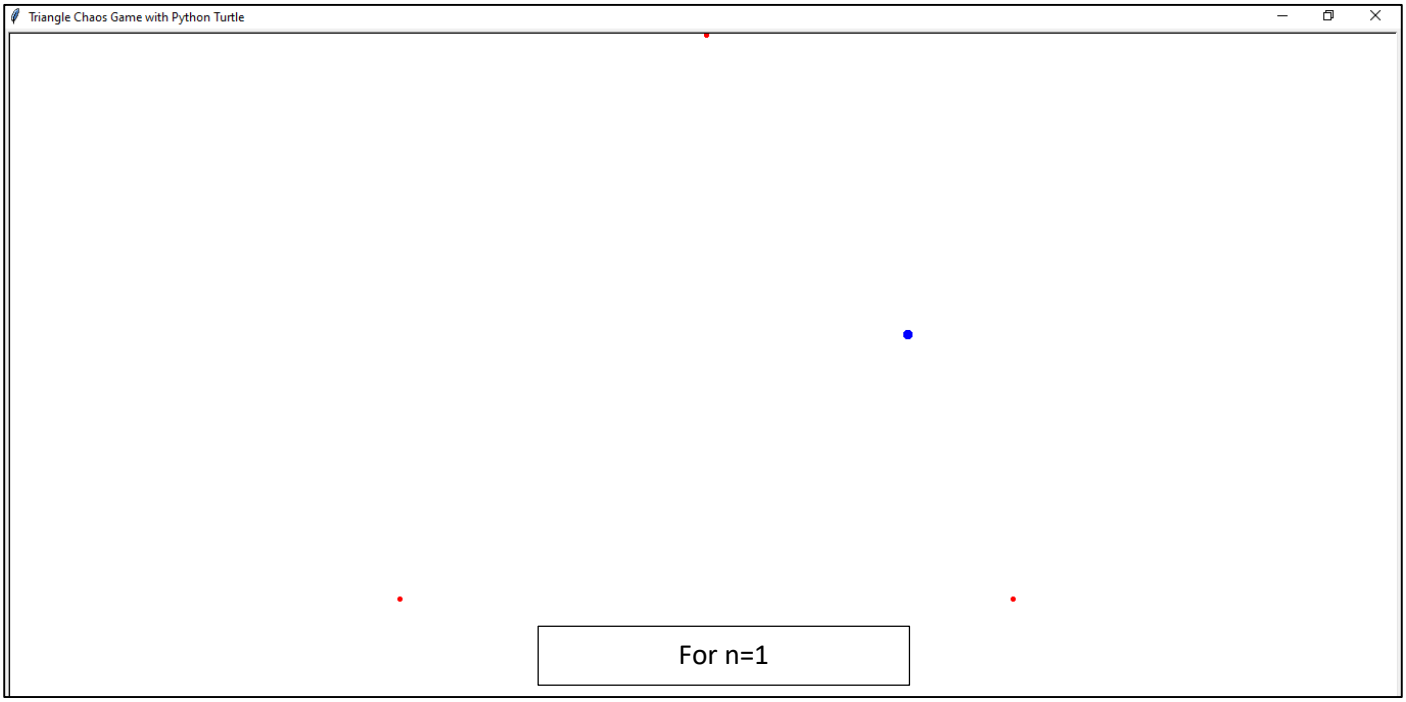
After 10,000 rolls of the die, we have a surprising image which seems to be approaching the exact Sierpinski Triangle.

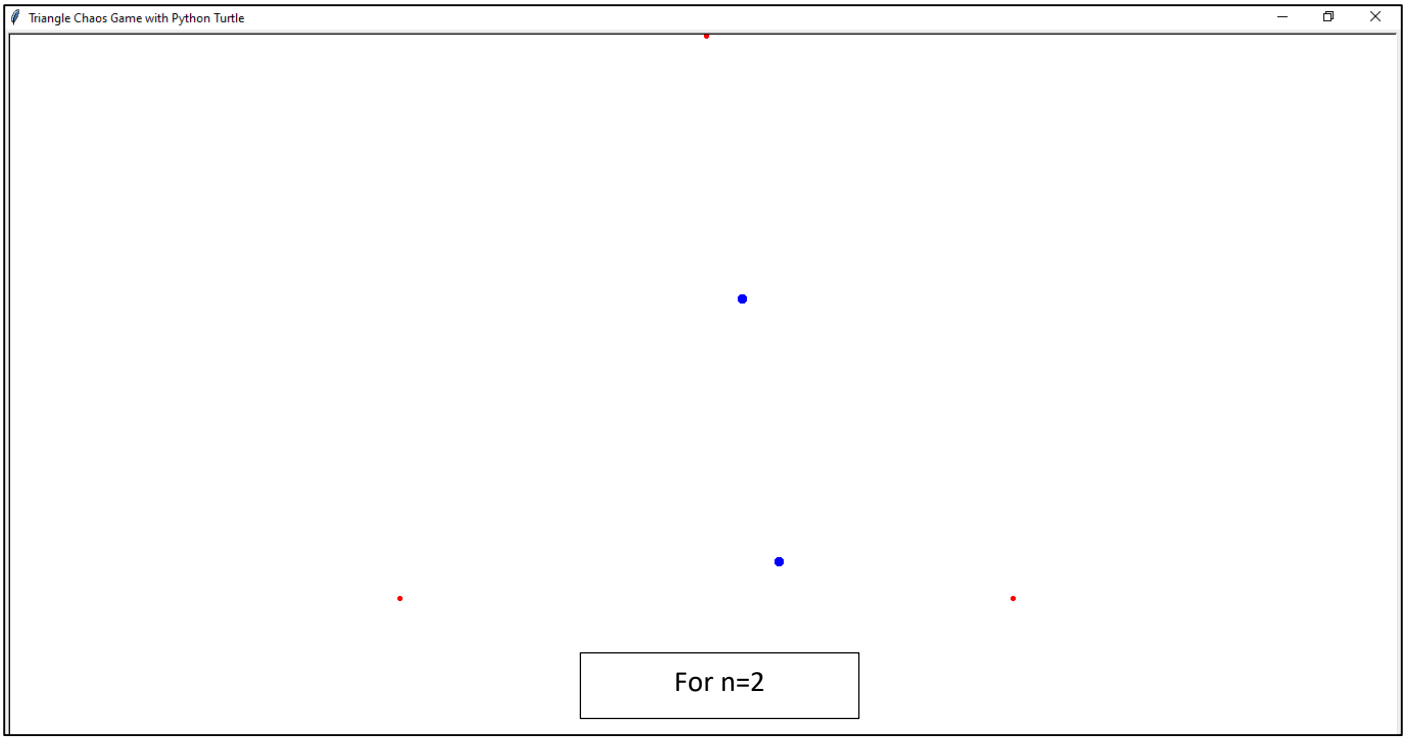
6.2.3 Code for Chaos Game- Sierpinski Triangle:

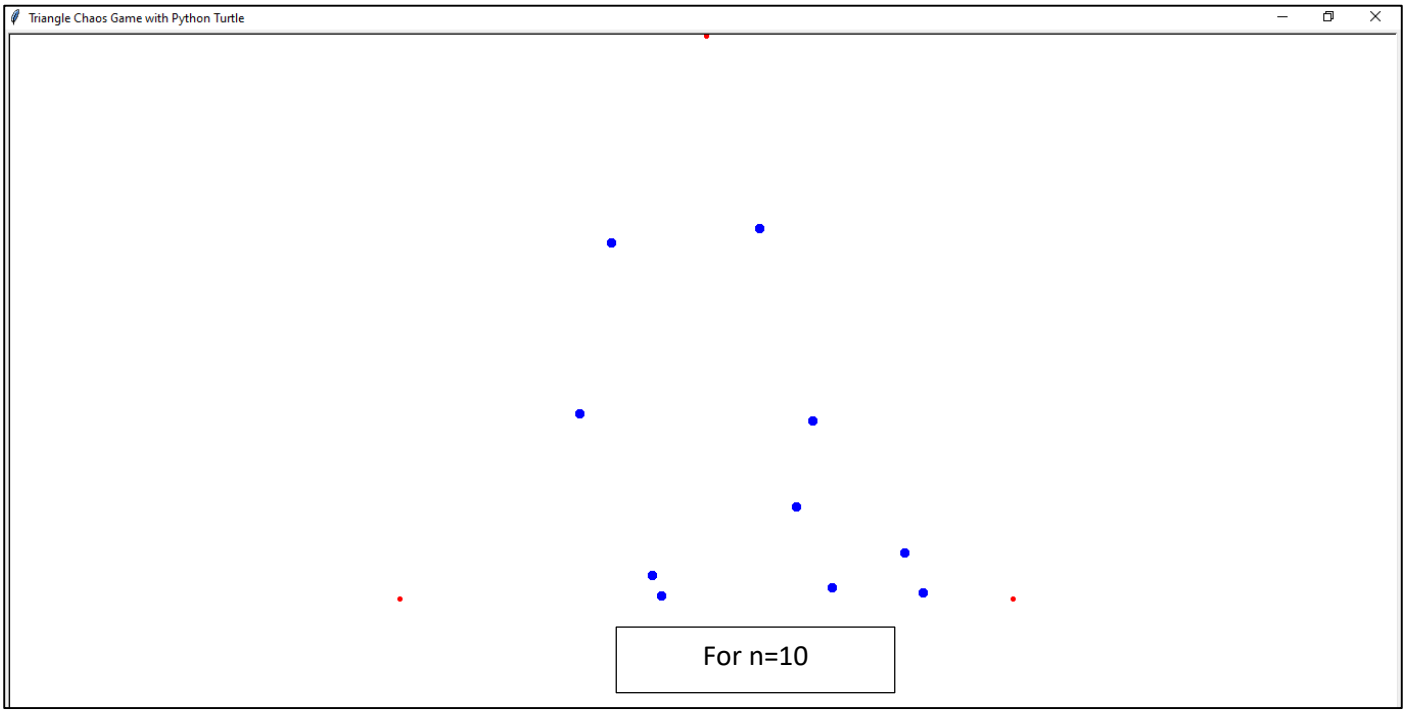
```
7     import turtle
8     import random
9
10    screen = turtle.Screen()
11    screen.title('Triangle Chaos Game with Python Turtle')
12    screen.setup(1000,1000)
13    screen.tracer(0,0)
14    turtle.hideturtle()
15    turtle.speed(0)
16    turtle.up()
17
18    A = (0,350)
19    B = (-300,-200)
20    C = (300,-200)
21    V = (A,B,C) # list of three vertices
22    for v in V:
23        turtle.goto(v)
24        turtle.dot('red')
25
26    n = 50000 # number of points to draw
27    p = (random.uniform(-200,200),random.uniform(-200,200)) # random starting point
28    t = turtle.Turtle()
29    t.up()
30    t.hideturtle()
31    for i in range(n):
```

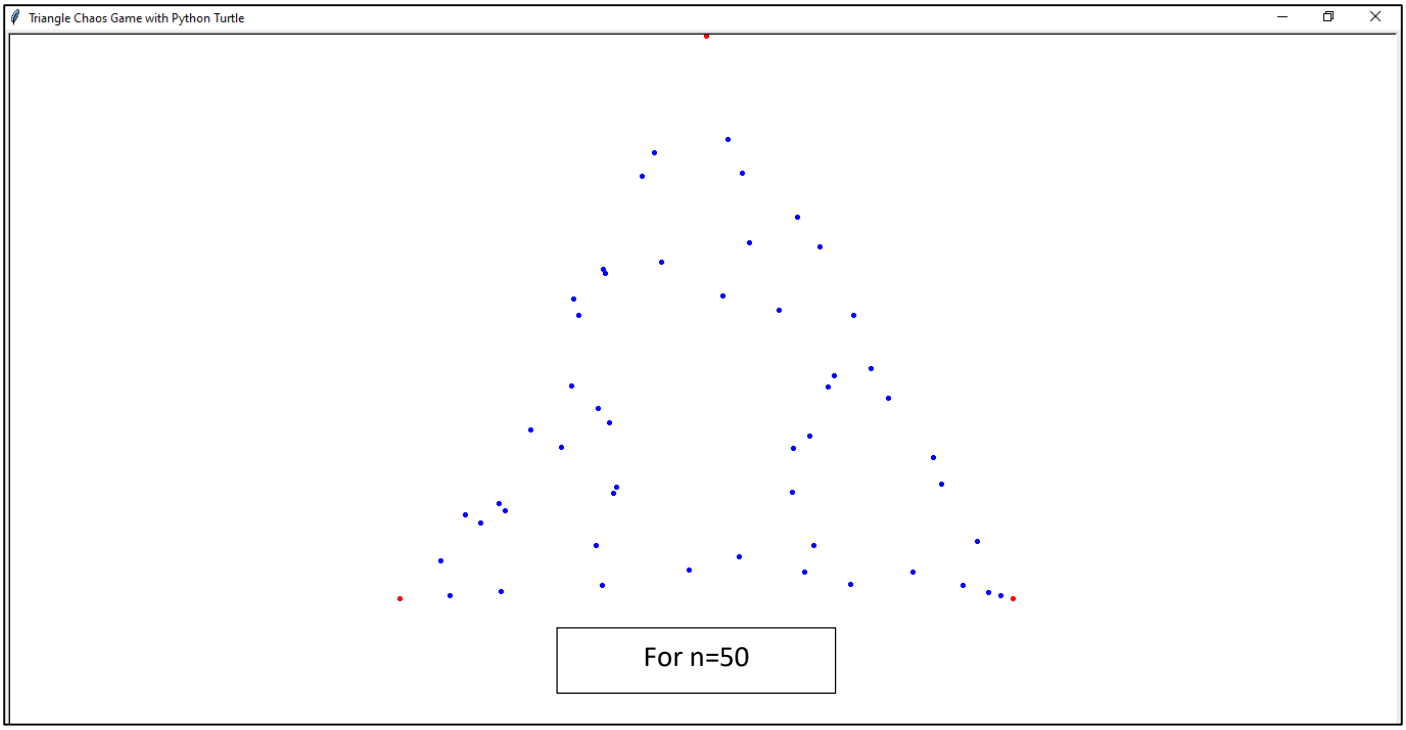
```
32          t.goto(p)
33          t.dot(2,'blue')
34          r = random.randrange(len(V)) # pick a random vertex
35          p = ((V[r][0]+p[0])/2,(V[r][1]+p[1])/2) # go to mid point between the random
        vertex and point
36          if i % 1000 == 0: # update for every 1000 moves, this part is for performance
        reason only
37              t = turtle.Turtle() # use new turutle
38              t.up()
39              t.hideturtle()
40              screen.update()
41      turtle.exitonclick()
42
```

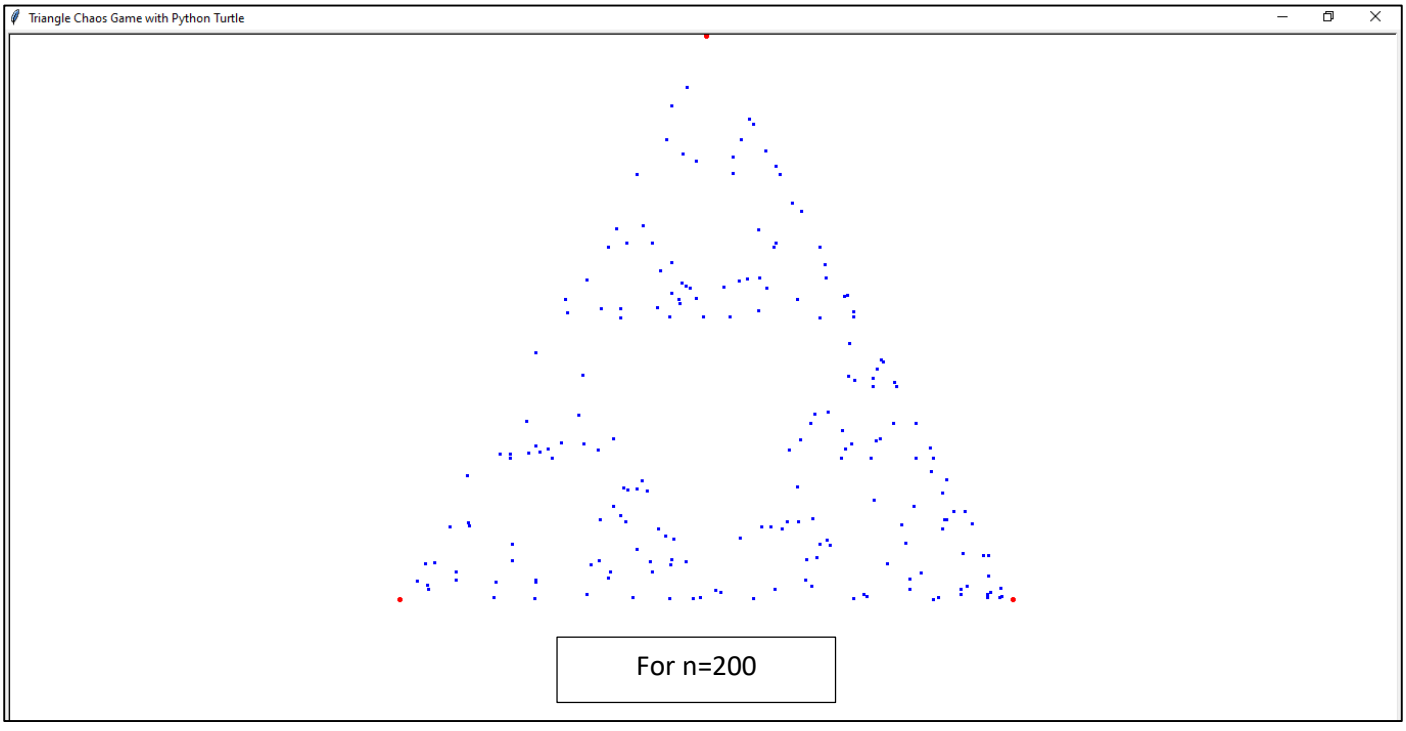
Output:

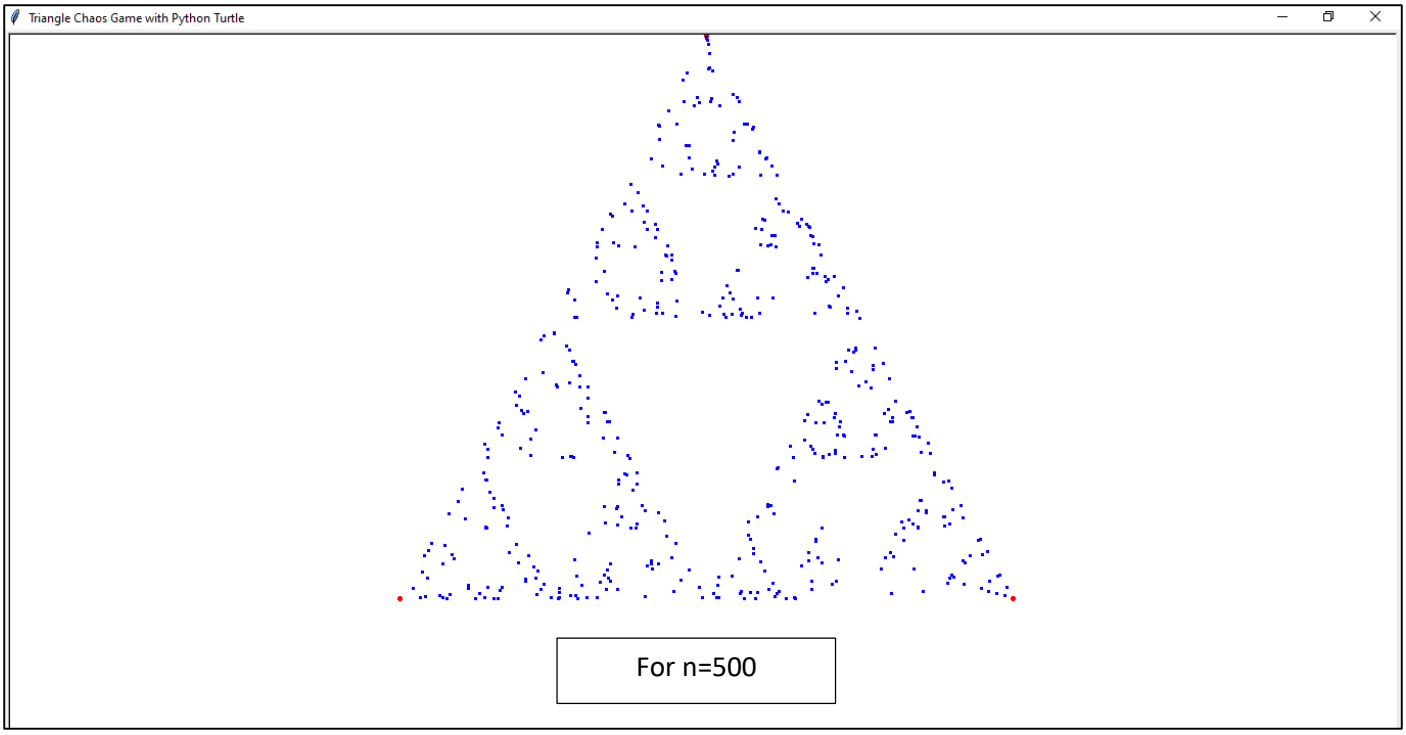



Triangle Chaos Game with Python Turtle
For n=2

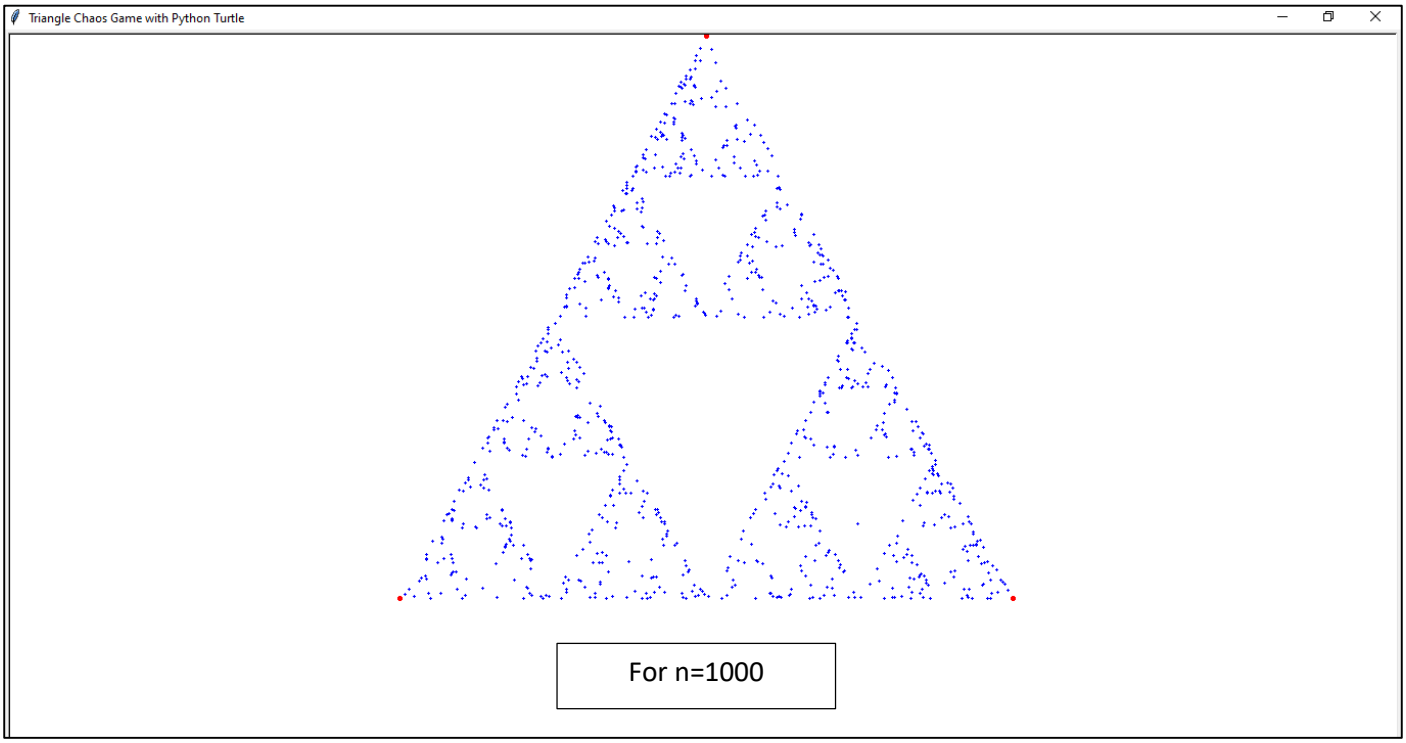

Triangle Chaos Game with Python Turtle
For n=10


Triangle Chaos Game with Python Turtle
For n=50

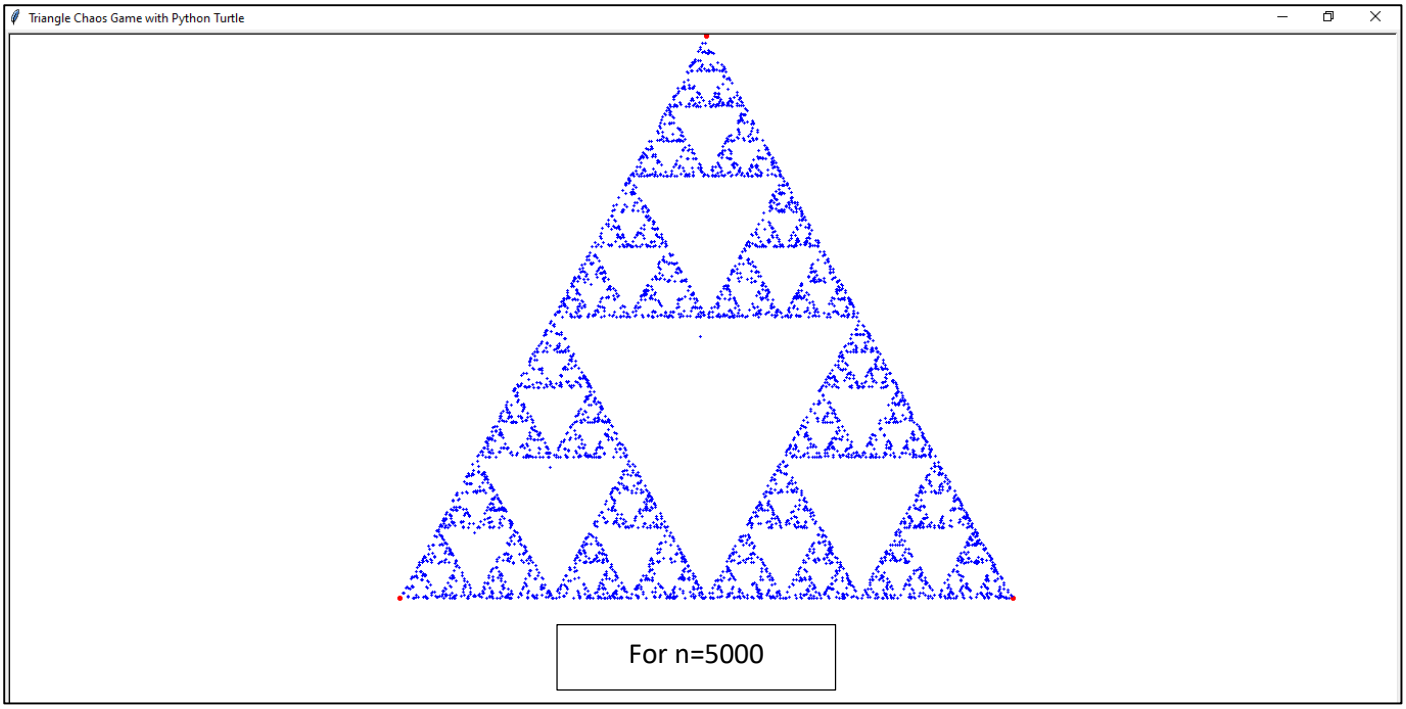

Triangle Chaos Game with Python Turtle
For n=200



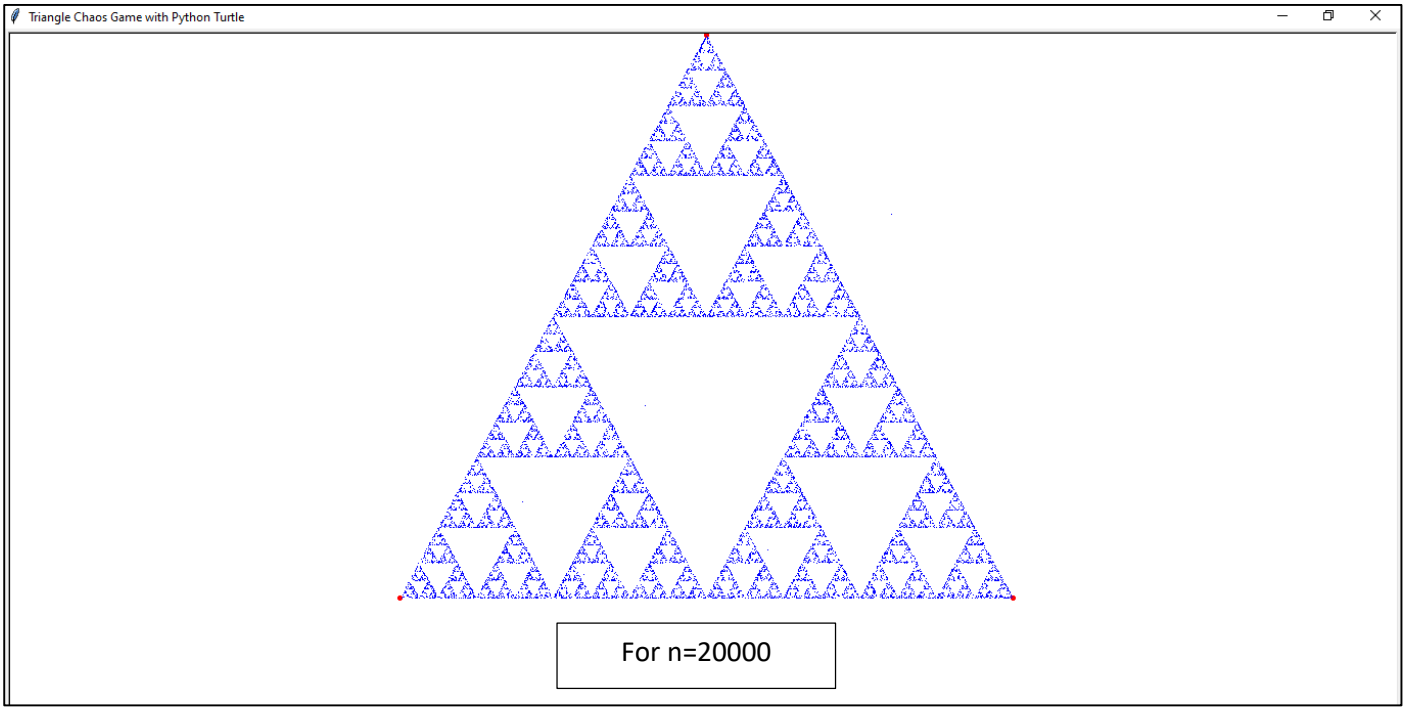
For n=500



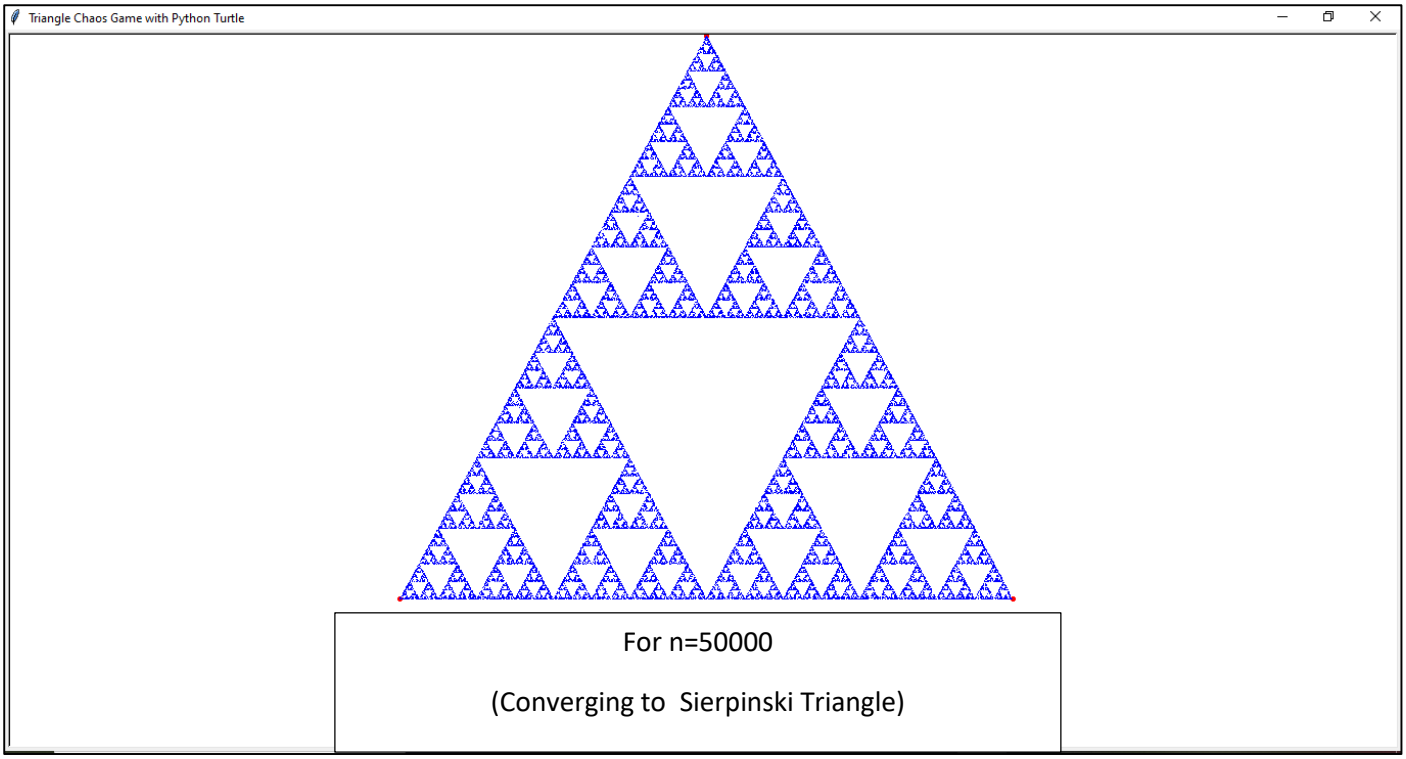
For n=1000



For n=5000



For n=20000



It's amazing that a random process actually generates a pattern. In fact, this Chaos Game was programmed to generate the Sierpinski Triangle. If we were to play the Chaos Game and roll the die an infinite number of times we would generate exactly the Sierpinski Triangle.

6.2.4 Why & How Chaos Game generates the Sierpinski Triangle:

Let's now try to think about why this process generates the Sierpinski Triangle. Remember that I was given the three functions at Section-3.1 of my project. The functions were

$$f_1(x,y) = \left(\frac{x}{2}, \frac{y}{2}\right), f_2(x,y) = \left(\frac{x}{2} + \frac{1}{2}, \frac{y}{2}\right) \text{ and } f_3(x,y) = \left(\frac{x}{2} + \frac{1}{4}, \frac{y}{2} + \frac{1}{2}\right)$$

We'll be keeping those functions in mind as we try to figure out why the Chaos Game works.

We will again start with an isosceles triangle with vertex A at the point (0, 0), vertex B at the point (1, 0), and vertex C at the point (1/2, 1). We will choose to start at a vertex, such as the vertex at the point C (1/2, 1) and roll the die. We rolled a three which tells us to plot a point half the distance to vertex B. We will plot the point (3/4, ½). Notice that our point is on the Sierpinski Triangle. We can now observe that this is the same point we would have gotten if would have applied the function

$$f_2(x,y) = \left(\frac{x}{2} + \frac{1}{2}, \frac{y}{2}\right)$$

to our starting point,i.e,

$$f_2(1/2, 1) = (3/4, 1/2).$$

We can now roll the die a second time. We rolled a 2 so we will plot a point half the distance from our last point to vertex A. Our point lands at (3/8, 1/4). Notice again that our point is on the Sierpinski Triangle. We can again observe that rolling a one or a two means that we are applying the function

$$f_1(x,y) = \left(\frac{x}{2}, \frac{y}{2}\right), \text{i.e,}$$

$$f_1\left(\frac{3}{4}, \frac{1}{2}\right) = \left(\frac{3}{8}, \frac{1}{4}\right).$$

If I roll a five or six, then I would apply the function

$$f_3(x,y) = \left(\frac{x}{2} + \frac{1}{4}, \frac{y}{2} + \frac{1}{2}\right).$$

and again land on the Sierpinski Triangle.

$$f_3(x,y) = \left(\frac{x}{2} + \frac{1}{4}, \frac{y}{2} + \frac{1}{2}\right)$$

$$\Longrightarrow f_3\left(\frac{3}{8}, \frac{1}{4}\right) = \left(\frac{7}{16}, \frac{5}{8}\right).$$

This process of applying *one of the three functions happens every time* we roll the die. That is, rolling the die tells us which one of the three functions to apply. If we roll a one or a two, the functions f_1 is applied. If we roll a three or a four, f_2 is applied. If we roll a five or a six, f_3 is applied. Using a die gives us a *random process* of choosing which function to apply. After an infinite number of rolls, each function is used one-third of the time. This guarantees that each part of the Sierpinski Triangle gets filled in. Our process of generating dots is random, but the dots we make are part of the Sierpinski Triangle.

Additionally, another way to convince ourselves why the Chaos Game works is that to notice that if we pick a point in the Sierpinski Triangle and apply one of the three functions we will always get a point in the Sierpinski Triangle. Every point on the Sierpinski Triangle has a *pre-determined outcome* of the Sierpinski Triangle. Although this is believable, it is very difficult to prove and the proof isn't necessary for our understanding of the Chaos game so we may take it on faith.

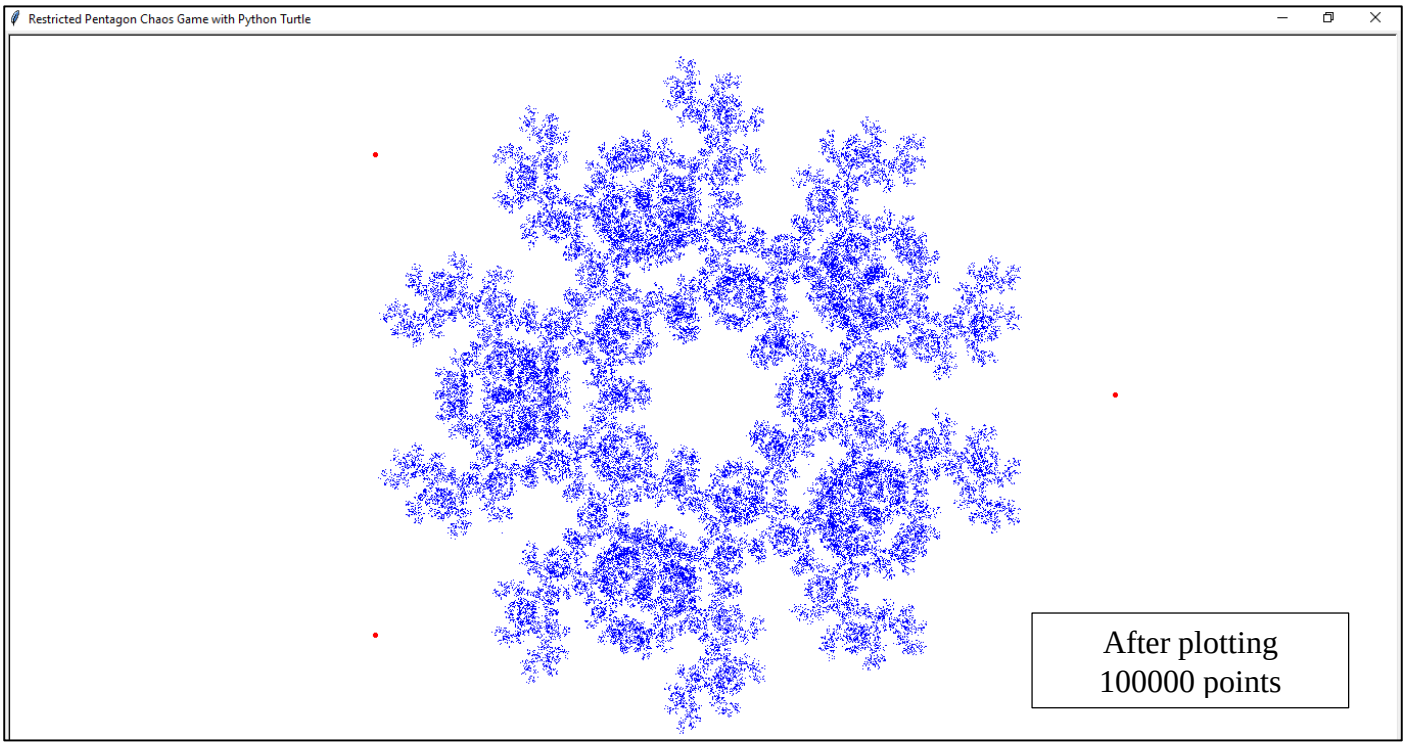
Conversely, if we start in a cut out section and roll the die an infinite number of times, we will always land on a cut out section. If we start in the largest cut out triangle, the points will eventually lie in successively smaller removed triangles. Actually, the removed triangles quickly become so small in size that it is essentially invisible. Starting in a cut out section will never allow us to land on the Sierpinski Triangle.

We can also explain the Chaos game in a slightly different way. We saw that when making the Sierpinski Triangle earlier we applied three functions. The first one shrunk the image and put it in the lower left area. The second one shrunk the image and put it in the lower right area. The third one shrunk the image and put it in the top middle area. This is just like the Chaos Game except instead of starting with the image and plotting the image we are starting with a point and plotting the point. Remember that we are using a random process with the die which means that due to *probability* we will plot points one third of the time in the lower left area and one-third of the time in the lower right area and one-third of the time in the top middle area. This means that after an infinite number of times of rolling the die and plotting points we would have the Sierpinski Triangle.

6.3 Restricted Pentagon Chaos Game:

```
import turtle
import random
import math

screen = turtle.Screen()
screen.title('Restricted Pentagon Chaos Game with Python Turtle')
screen.setup(1000,1000)
screen.tracer(0,0)
turtle.hideturtle()
turtle.speed(0)
turtle.up()

m=5
angle = 0
V = []
for i in range(m):
    p = (400*math.cos(angle),400*math.sin(angle))
    V.append(p)
    angle += math.pi*2/m

for v in V:
    turtle.goto(v)
    turtle.dot('red')

n = 100000 # number of points to draw
p = (random.uniform(-200,200),random.uniform(-200,200)) # random starting point
t = turtle.Turtle()
t.up()
t.hideturtle()
lastr = r = -1
for i in range(n):
    t.goto(p)
    t.dot(2,'blue')
    while r == lastr:
        r = random.randrange(len(V)) # pick a random vertex
    lastr = r
    p = ((V[r][0]+p[0])/2,(V[r][1]+p[1])/2) # go to mid point between the random vertex and point
    if i % 1000 == 0: # update for every 1000 moves, this part is for performance reason only
        t = turtle.Turtle() # use new turutle
        t.up()
        t.hideturtle()
        screen.update()
```

Output:

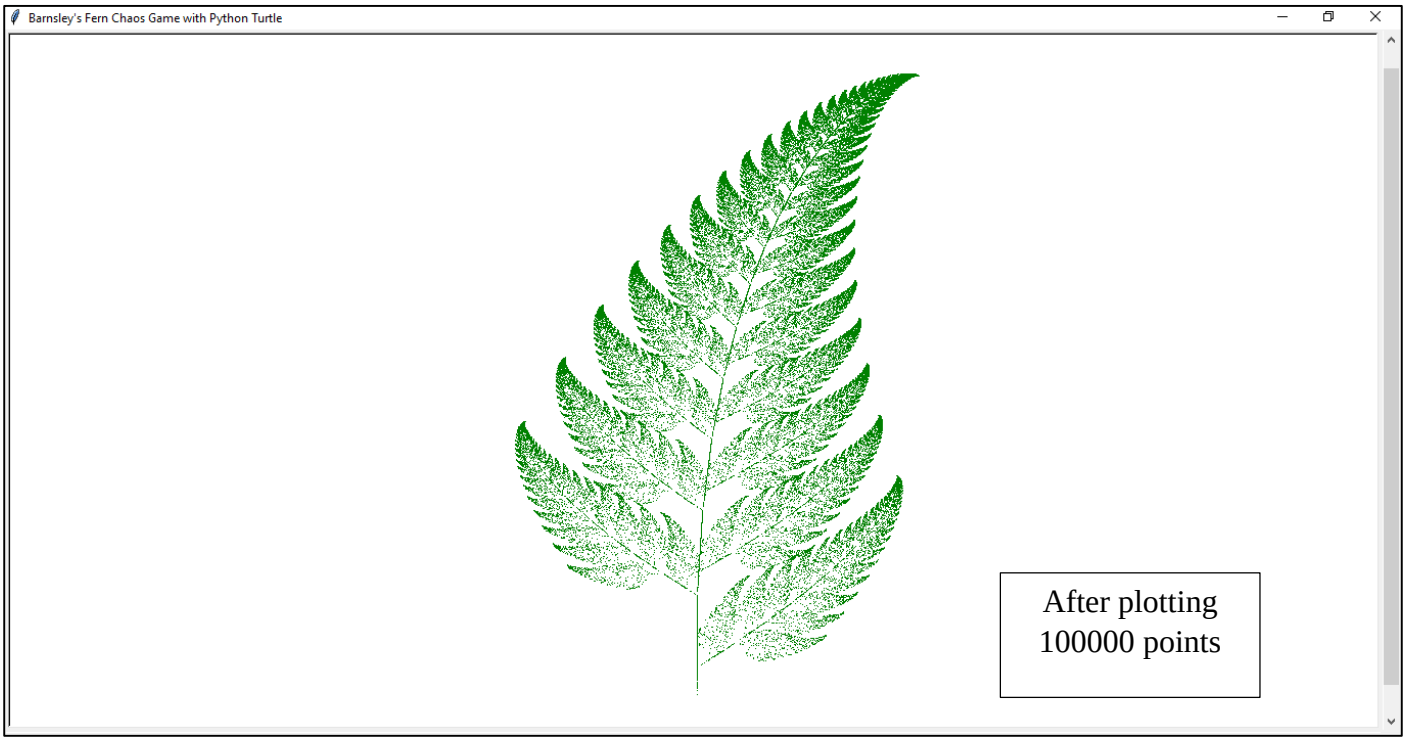


6.4 Drawing Barnsley's Fern with Chaos Game:

```
import turtle
import random

screen = turtle.Screen()
screen.title('Barnsley\'s Fern Chaos Game with Python Turtle')
screen.setup(1000,1000)
screen.setworldcoordinates(-6,-1,6,11)
screen.tracer(0,0)
turtle.hideturtle()
turtle.speed(0)
turtle.up()

n = 100000 # number of points to draw
p = (0,0)
t = turtle.Turtle()
t.up()
t.hideturtle()
for i in range(n):
    t.goto(p)
    t.dot(2,'green')
    r = random.uniform(0,1)
    if r < 0.01:
        p = (0,0.16*p[1])
    elif r < 0.86:
        p = (0.85*p[0] + 0.04*p[1], -0.04*p[0] + 0.85*p[1] + 1.6)
```

```
    elif r < 0.93:
        p = (0.2*p[0] - 0.26*p[1], 0.23*p[0] + 0.22*p[1] + 1.6)
    else:
        p = (-0.15*p[0] + 0.28*p[1], 0.26*p[0] + 0.24*p[1] + 0.44)

    if i % 1000 == 0: # update for every 1000 moves, this part is for performance reason
only
        t = turtle.Turtle() # use new turutle
        t.up()
        t.hideturtle()
        screen.update()
```

Output:

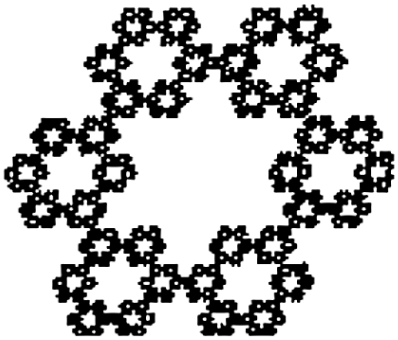


6.5 Some Other Similar Chaos Game:

For generating fractals using chaos game, we can use any number of vertices and not just three.

6.5.1 Sierpinski Hexagon:

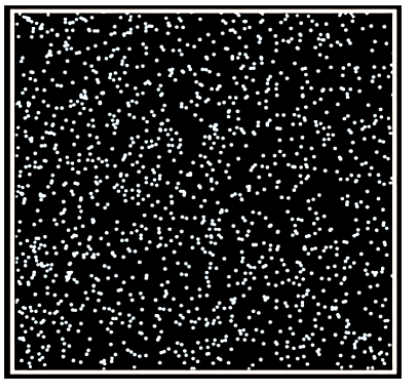
For example, let us consider a regular hexagon, a polygon with six vertices. Number the vertices of the hexagon from one through six and this time we take a regular numbered dice. The rule here is instead of half, we now move one-third of the distance towards the chosen vertex, that is, the compression ratio for this game is three. After rolling the die for a very large number of times we get a new fractal, 'Sierpinski Hexagon'.



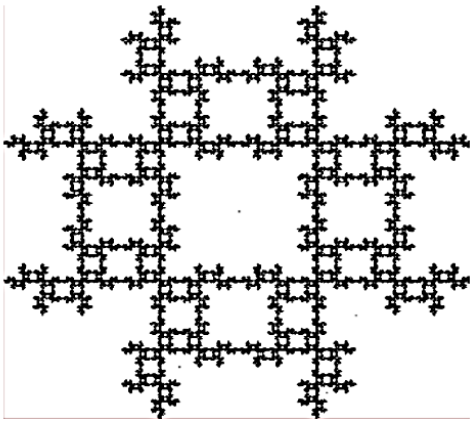
There is another fractal buried inside this fractal. The inner boundary of the Sierpinski Hexagon is actually the fractal curve, Koch Snowflake.

6.5.2 Now if we play the chaos game with four vertices and compression ratio

of two, then what we get is actually a square, which is obviously a self-similar object but not a fractal.

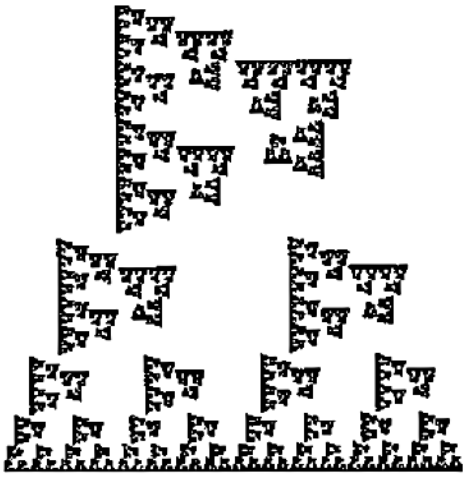


Although if we play the chaos game with four vertices and compression ratio of two, but with an additional rule that the current vertex cannot be chosen in the next iteration, then we obtain this fractal.



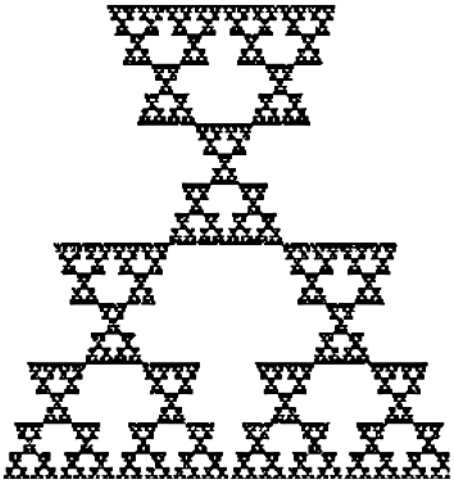
6.5.3 Rotational Chaos:

Another, more complicated type of chaos game results when we allow rotations. This is where the geometry of transformations becomes more important. Starting with the vertices of an equilateral triangle, for the bottom two vertices, the rules are as before, that we move half the distance towards these vertices when called. For the top vertex, the rule is, first we move the point half the distance to that vertex, and then rotate the point 90 degrees about the vertex in the clockwise direction. The result of this chaos game is as such:

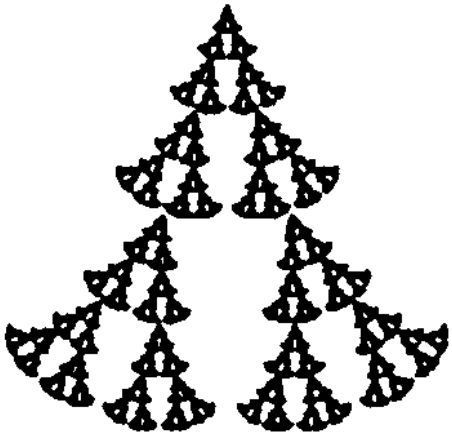


There are basically three self-similar pieces in the fractal, each of which is half the size of the original, but the top one is rotated by 90 degrees in the clockwise direction.

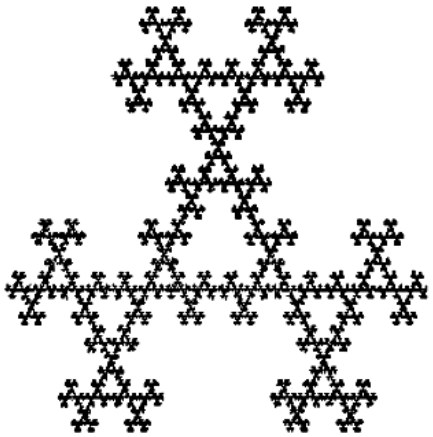
Now, if we change the rotation to 180 degrees around the top vertex we obtain this fractal.



The following fractal is obtained by rotating by 20 degrees in the clockwise direction around the lower left vertex, rotating 20 degrees in the counter clockwise direction around the lower right vertex, and no rotation around the top vertex.



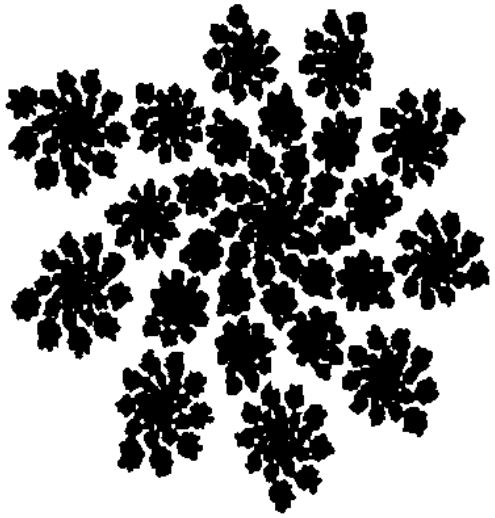
And the following fraction is obtained by rotating 180 degrees about each of the vertices.



6.5.4 Fractal Starfish:

When compression ratio is to be change, it often becomes necessary to change the probability of choosing a certain vertex. The reason for this is that if a compression ratio at a certain vertex is just slightly larger than one, then we need to choose that vertex over and over in succession in order to fill the entire portion of the fractal corresponding to that vertex.

For example, the fractal starfish was made with just two vertices, one in the upper left corner with compression ratio 5; and one in the center with compression ratio 1.04 and a rotation of 38 degrees. We actually placed 11 vertices all with the same compression ratios and rotations in the center to change the odds of moving toward the center vertex.



6.6 Linear Algebra behind the Chaos Game:

All of the chaos games thus far have been specified by giving just the location of the vertices, compression ratio, and rotation. Using linear algebra, we can specified these rules by providing an affine transformation of the form:

$$T\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{1}{a} & 0 \\ 0 & \frac{1}{a} \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

Here, (x_0, y_0) is the vertex, $a > 1$ is the compression ratio and θ is the rotation angle. More generally, we could allow any affine transformation, provided that the matrix involved is a contraction. When we allow this, the output of the chaos game produces a much richer collection of fractals, including not only fractals from geometry, but also fractals from nature, such as fern.

6.7 Why Chaos Game?

The Chaos Game is a very fast way to generate fractals. For a computer to generate a fractal by applying the functions used to generate the fractal it would take a great deal of time. For example for a computer to generate the Barnsley Fern it would require a full day or more. However, using the Chaos Game to generate the Barnsley Fern requires a second or two to see an image that for all intents and purposes looks like the final Barnsly Fern.

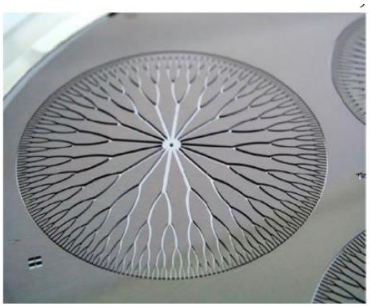
7. Conclusion

Interestingly, chaos theory is used to make sense of and study natural phenomena that recently was thought to be pattern-less and incapable of being described using mathematical modeling. Some natural phenomena that we now use chaos theory to study are weather patterns, the occurrences of earthquakes, and fluctuations in the stock market.

Some of these are described below:

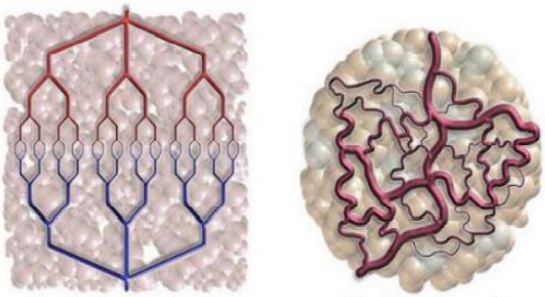
1) The use of chaos game in data compression. Like, for example, in order to generate Sierpinski Gasket, the data which is required to feed into the computer are just the three vertices, a compression ratio and the total number of iterations. Feeding these three informations to the computer allows us to store the incredibly complicated set of points making up the fractal. Images of many other fractals from nature, such as ferns, clouds, coastlines, etc., can be captured as a very small data sets. These ideas have been used to advantage in many diverse arenas as digital encyclopedias and Hollywood movies to construct and store lifelike, fractal images in a very efficient manner.

2) This is a computer chip cooling circuit etched in a fractal branching pattern. Developed by researchers at Oregon State University, the device channels liquid nitrogen across the surface to keep the chip cool.

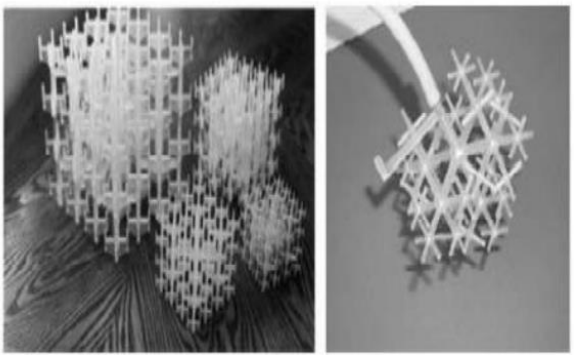


3) Fractal antennas developed by Fractenna in the US and Fractus in Europe, are making their way into cellphones and other devices. Because of their fractal shapes, these antennas can be very compact while receiving radio signals across a range of frequencies.

4) Researchers at Harvard Medical School and elsewhere are using fractal analysis to assess the health of blood vessels in cancerous tumors. Fractal analysis of CT scans can also quantify the health of lungs suffering from emphysema or other pulmonary illnesses.



5) Amalgamated Research Inc (ARI) creates space-filling fractal devices for high precision fluid mixing. Used in many industries, these devices allow fluids such as epoxy resins to be carefully and precisely blended without the need for turbulent stirring.



- **"<u>Every Event is the Inevitable Outcome of a Previous Event" in Light of Chaos:</u>**

In Chaos Theory, systems that show mathematical chaos are deterministic which means that they actually have a determined outcome despite the fact that they appear to be complicated and random. Chaos uses determinism which is the philosophical belief that every event is the inevitable outcome of a previous event. Historically, determinism dates back several thousand years while Chaos Theory is fairly recent. With Chaos Theory a random event produces a specific determined outcome. Chaos Theory was used as a central role in the movie with Ashton Kutcher titled The Butterfly Effect. The movie got its title from the idea of the butterfly effect which is where the flapping of a butterfly is imagined to produce some effect to the atmosphere that eventually leads to some dramatic event such as a tornado.

Chaos Theory is also portrayed in the book by Ray Bradbury titled Sound of Thunder which used Chaos Theory as a central role. This was made into a movie of the same name that was released in 2005. In the book, adventurers pay money to time travel back to the time of dinosaurs for a safari. The time travel company is very strict that the customers do not step off the path. However, at one point one customer steps off the path slightly and squishes a moth. Then, when they return to present time, the world as they knew it is drastically different. For instance the United States is now under the leadership of a ruthless dictator. The outcome of killing that one specific moth produced a specific determined outcome of the dictatorship. This is saying a random event (killing the moth) produces a specific determined outcome (the dictatorship). This is similar to the playing the Chaos Game where the random event of rolling the die an infinite number of times produces a specific determined outcome of the Sierpinski Triangle. Also, every point on the Sierpinski Triangle has a specific determined outcome of the Sierpinski Triangle.

8. References

- Fractals and Chaos – Paul S. Addison.
- Fractals Everywhere: The First Course in Deterministic Geometry – Michael Fielding Barnsley.
- Fractals and Chaos Simplified for the Life Sciences – L. Liebovitch.
- The Fractal Geometry of Nature – Benoit B. Mandelbrot.
- Chaos Rules! – A paper by Robert L. Devaney, Boston University; http://www.maa.org
- www.alunw.freeuk.com/pascal.html
- www.math.bu.edu
- www.FractalsFoundation.org
- Fractals and the Collage Theorem.pdf by Sandra S. Snyder.
- Geometric-Modelling-Of-Complex-Objects-Using-Iterated-Function-System.pdf by Garg, Agarwal, Negi.
- Fractals and the Chaos Game (1).pdf by Stacie Lefler.
- The Beauty of Fractals- H.-O. Peitgen & P.H. Richter.