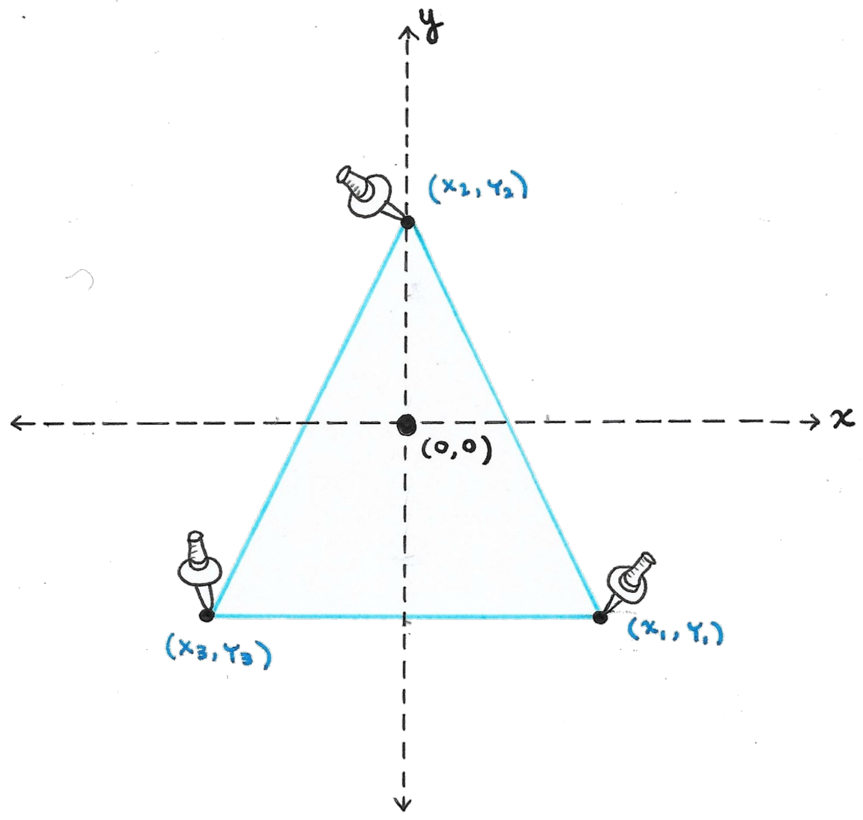
The Chaos Game

July 2019

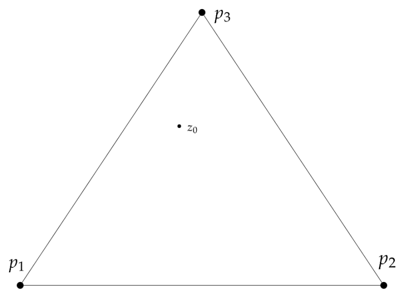


Abstract

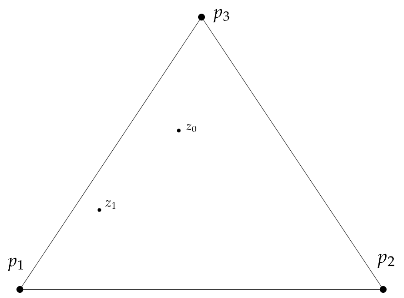
The Chaos Game at heart is a group of algorithms that produce fractal shapes. This text focuses on the most popular version of the Chaos Game: the Sierpiński Chaos Game. Along with introducing these algorithms, the overall purpose of the paper is proving the appearance of the Sierpiński triangle within the Sierpiński Chaos Game. Sections I to III are preliminaries that are valuable to familiarize the reader with the problem, notation, and terminology. Detailed in section IV is the proof and to close the paper, section V briefly speaks to applications of the Chaos Game and fractal geometry.

I. The Sierpiński Chaos Game

Imagine three points $\{p_1, p_2, p_3\}$ called *pins* that are arranged in a triangle. Accompanied by our pins is an arbitrary point, z_0 with coordinates (x_0, y_0). Generally, all pins and points lie in the Cartesian plane $\mathbb{R}^2$.

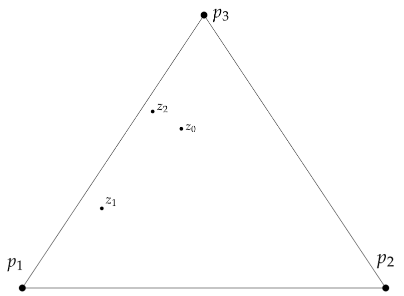


From here we begin our game. To start, randomly select a pin from $\{p_1, p_2, p_3\}$ and place the next point z_1 halfway between our chosen pin and the initial point z_0.



Ex: Random pin is p_1

Similarly, to find z_2 we calculate half the distance between z_1 and a new randomly chosen pin (this may or may not be the same pin selected prior).



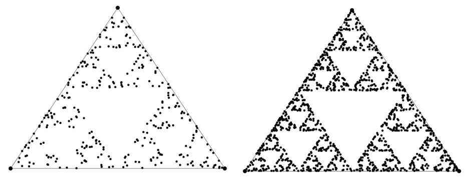
Ex: Random pin is p_3

Let $p_i \in \{p_1, p_2, p_3\}$ be a randomly selected pin with coordinates (x_{p_i}, y_{p_i}). Generally, to compute z_n (where n is a non-zero integer),

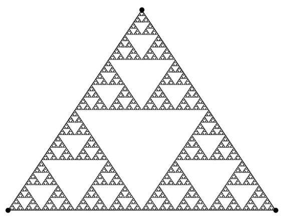
$$z_n = \left(\frac{x_{n-1} + x_{p_i}}{2}, \frac{y_{n-1} + y_{p_i}}{2}\right)$$

Simply put, to locate a point z_n, we calculate half of the distance between the previous point z_{n-1} and a randomly selected pin p_i.

For small n we can play this game with a pen and paper. After a few turns we have what appears to be a bunch of randomly plotted points. Where the image becomes interesting is for large n, or more formally the $\lim_{n\to\infty} z_n$. In this case, it clearly becomes quite tedious to plot each point by hand so we typically use a computer to simulate this process.[1]



The result may be familiar for those who have seen fractal shapes or have learned recursive programming. For large n, this game produces a Sierpiński Triangle, named after Polish mathematician Wacław Sierpiński. Roughly speaking, a Sierpiński Triangle can be considered as a triangle that has been subdivided recursively into scaled down versions of itself.



This is called by many the *Sierpiński Chaos Game*, one of many variations of the Chaos Game. The Chaos Game is considered a family of games with the following characteristics:

[1]visit owenpetersmith.com for visuals

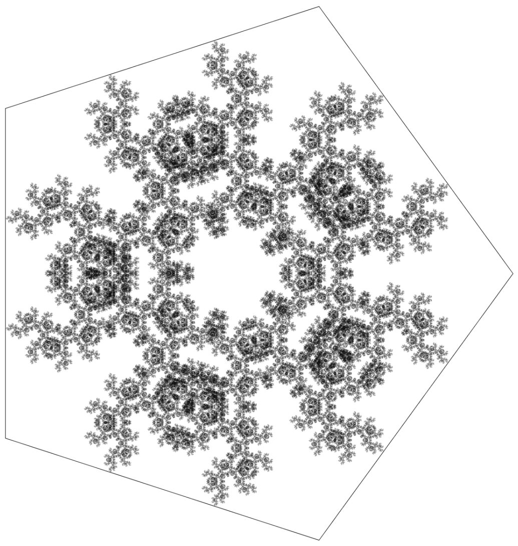
1. A starting point z_0
2. A set of pins $\{p_1, p_2, \ldots, p_k\} \;\; k \in \mathbb{N}$
3. The probability of choosing pin p_i is p'_i $(1 \leq i \leq k)$ such that

   $$p'_1 + p'_2 + \ldots + p'_k = 1 \quad [1]$$

   Note: In the Sierpiński Chaos Game, $p'_1 = p'_2 = p'_3 = \frac{1}{3}$
4. Every iteration we choose a pin p_i based on the probabilities of selecting each pin.
5. The point z_n is determined by some 'rule' dependent on z_{n-1} and p_i.

   Note: In the Sierpiński Chaos Game, our rule was: z_n is half the distance between z_{n-1} and p_i.

There are an numerous versions of the Chaos Game that produce different fractals.



Chaos Game with 5 pins. Same formula for z_n except p_i cannot equal the pin selected prior.

However, due to particular characteristics in these variations, sometimes no significant image appears.

An obvious question arises from the Sierpiński Chaos Game. Why and how does the Sierpiński Triangle appear? Before we can prove this result, we must introduce how we build such a shape. Which in itself begins with a brief, high-level overview of self-similar fractal shapes.

II. Fractals

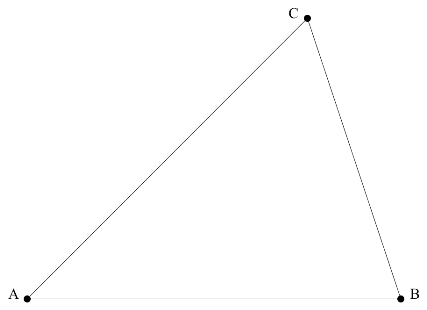
The most commonly known types of Fractals are *self similar*. Informally, an object is self-similar if we can zoom in on a particular part and the result is exactly or approximately the original object.

Fractals are obtained by beginning with an image and iterating a particular function. This function is called the *Iterated Function System* (IFS) - made up of several linear (or affine to be more precise) transformations. Each transformation making up the IFS determines the type of self-similarity within the fractal. [1]

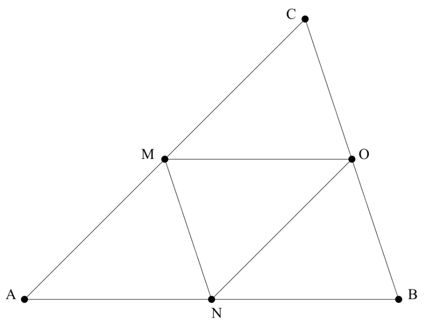
III. Building a Sierpiński Triangle

The Sierpiński Triangle is a fascinating shape. It has the theoretical property of infinite perimeter and yet zero area. Just like other self-similar fractals, this shape has a set of linear transformations that defines its self-similar structure. In particular, the Sierpiński Triangle has three linear transformations. The best way to develop an intuition for these transformations is to build the triangle from scratch.

Begin with a triangle with corners A, B, C ($\triangle ABC$). Without loss of generality we fix A, B at coordinates $(0, 0)$ and $(1, 0)$ respectively, and let C sit arbitrarily above the x-axis at (x_C, y_C).

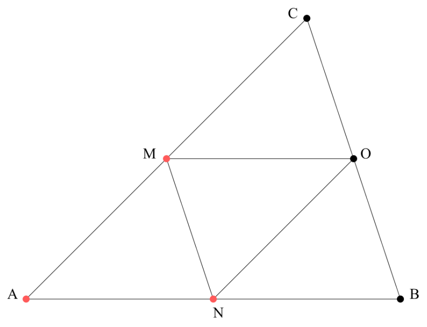


Then divide the triangle into four equal triangles - discarding the center.



To repeat this process, we require three distinct transformations $\{f_A, f_B, f_C\}$, each to map a particular corner to its respective corner in each of the three smaller, scaled down triangles.

For starters, lets define f_A. We take some (x, y) coordinate as input and map it to a new coordinate (ie. $f_A : \mathbb{R}^2 \to \mathbb{R}^2$). Essentially, f_A will have the task of assigning a new left corner (red) to each of the smaller triangles ($\triangle ANM$, $\triangle NBO$, $\triangle MOC$). [2]



Note that f_A equivalently maps each corner of $\triangle ANM$[2]

So we obtain the following definition for f_A given the diagram above.

$$f_A(x,y) = \begin{cases} (0,0) & \text{if } (x,y) = (0,0) \\ (x_N, y_N) & \text{if } (x,y) = (1,0) \\ (x_M, y_M) & \text{if } (x,y) = (x_C, y_C) \end{cases}$$

Similarly with f_B and f_C

$$f_B(x,y) = \begin{cases} (x_N, y_N) & \text{if } (x,y) = (0,0) \\ (1,0) & \text{if } (x,y) = (1,0) \\ (x_O, y_O) & \text{if } (x,y) = (x_C, y_C) \end{cases}$$

$$f_C(x,y) = \begin{cases} (x_M, y_M) & \text{if } (x,y) = (0,0) \\ (x_O, y_O) & \text{if } (x,y) = (1,0) \\ (x_C, y_C) & \text{if } (x,y) = (x_C, y_C) \end{cases}$$

An important observation is that regardless of its side length, in order to divide the triangle into four equal smaller triangles, each must be a scaled version of the original by a factor of $\frac{1}{2}$.

With this observation, we can redefine our mapping function into a more concrete set of linear transformations. From above we know f_A maps the origin to the same place. The function also maps (1,0) to $(x_N, y_N) = (\frac{1}{2}, 0)$ and (x_C, y_C) to $(x_M, y_M) = (\frac{x_C}{2}, \frac{y_C}{2})$. Generally we can redefine f_A as, [2]

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

Using the same logic we can derive,

$$f_B(x,y) = (\frac{x+1}{2}, \frac{y}{2})$$

$$f_C(x,y) = (\frac{x + x_C}{2}, \frac{y + y_C}{2})$$

Thus f_A and f_B are completely independent of the shape of the Sierpiński Triangle; only f_C depends on the top corner C. The set of these three transformations defines our IFS for the Sierpiński Triangle. As we will see in the next section, understanding the IFS will be the key to proving its appearance in the Sierpiński Chaos Game.

IV. Proving the Result of the Sierpiński Chaos Game

Our goal is to prove that the algorithm in the Sierpiński Chaos Game does indeed produce a Sierpiński Triangle. Specifically we are certifying the algorithm's correctness by showing that it is equivalent to the IFS in the section prior.

Recall our pins $\{p_1, p_2, p_3\}$. Without loss of generality we place p_1 and p_2 at coordinates (0,0) and (1,0) respectively. As you may have guessed, we let p_3 sit arbitrarily above the x-axis. From section I we defined a equation that calculates z_n.

$$z_n = (\frac{x_{n-1} + x_{p_i}}{2}, \frac{y_{n-1} + y_{p_i}}{2})$$

Knowing that there are three choices for p_i, we can split the equation into, [2]

$$z_n = \begin{cases} (\frac{x+x_{p_1}}{2}, \frac{y+y_{p_1}}{2}) & \text{if } p_i = p_1 \\ (\frac{x+x_{p_2}}{2}, \frac{y+y_{p_2}}{2}) & \text{if } p_i = p_2 \\ (\frac{x+x_{p_3}}{2}, \frac{y+y_{p_3}}{2}) & \text{if } p_i = p_3 \end{cases}$$

To simplify, plug in the coordinates for each pin.

$$z_n = \begin{cases} (\frac{x}{2}, \frac{y}{2}) & \text{if } p_i = p_1 \\ (\frac{x+1}{2}, \frac{y}{2}) & \text{if } p_i = p_2 \\ (\frac{x+x_{p_3}}{2}, \frac{y+y_{p_3}}{2}) & \text{if } p_i = p_3 \end{cases}$$

Which is identical to the IFS for the Sierpiński Triangle! Easy! As n becomes arbitrarily large, the image produced will be a Sierpiński Triangle since the Sierpiński Chaos Game mimics its' IFS.

V. Applications of Fractal Shapes

The main applications for self similar fractal shapes appear in the world of computer graphics. We often see fractal shapes in nature (eg. snowflakes, mountains ranges, lightning bolts, leaves, etc.) and with the help of the IFS, it takes a relatively small amount of code to reproduce these shapes. Modern animators and graphic artists use this as a tool to recreate real life fractals within a virtual environment.

[2]For simplicity (x_{n-1}, y_{n-1}) are replaced with (x, y)

The Barnsley Fern is one of the most famous fractal shapes and can be computed with a variation of the Chaos Game. [3]

VI. Closing

In my second year of university, I was introduced to the Sierpiński Triangle in a recursive programming assignment. I was struck immediately with frustration and complexity of building such a dynamic and intricate shape. Months later I was shown the Chaos Game for the first time. In an algorithm a toddler can follow with crayon and paper, to my surprise the Sierpiński Triangle magically appears out of thin air. Prior to starting this paper, I believed that it was nothing more than a mere coincidence. However, in mathematics, that truly is never the case.

References

[1] Pyke, R. (2009). Fractals and the Chaos Game. Presentation, Simon Fraser University.

[2] Steemson, K., & Williams, C. (n.d.). Generalised Sierpinski Triangles. Retrieved July 9, 2019, from https://arxiv.org/

[3] Weisstein, E. W. (n.d.). Barnsley's Fern. Retrieved July 14, 2019, from http://mathworld.wolfram.com/BarnsleysFern.html