Beyond the Dragon: A Novel Class of Fractal Curves

Sarah Bricault

Athinoula A. Martinos Center at MGH, Massachusetts, USA; sbricault@mgh.harvard.edu

Abstract

The Dragon Curve is a fractal that can be created by iteratively folding a single line according to a set algorithm. Every line segment is replaced in the subsequent iteration by two line segments that intersect at right angles with the end points unchanged. The following novel implementation results in a class of fractals referred to here as Folding Curves. One can fold the initial line according to a repeating pattern of fold commands, called here the fold command list. When one reaches the end of the fold command list, one returns to the beginning. For example, a fold command list of [0 1 1 0] produces an interesting semi-self-similar fractal. Additional complexity and beauty can be added by starting with a shape that is not a single line or implementing the fractals in a physical medium.

Introduction

The Dragon Curve, as well as the related Levy C curve, represent a type of fractal that can be created by iteratively folding a single line according to a set algorithm [4]. This is classically represented according to an L-system [1]. However, this is not the only way to imitate this pattern of lines.

In my initial implementation of the Dragon Curve fractal, I made a design choice of abstracting away the pattern of folds one would take to create the fractal, allowing me to use the same code for both the Levy C curve (all right turns) and the Dragon Curve (alternating right and left turns). Serendipitously, I did not limit the sequence of folds (the fold command list) to stop after each line segment. According to standard definitions (including via an L-system) the Dragon Curve at iteration i is created by replacing each line segment in iteration i-1 with two new line segments according to its algorithm [1]. Each line is replaced individually and independently. That's why it can be represented so well by an L-system.

However, the design of my code begged the question- what if that independence is lost and the fold command list can extend across multiple lines? In other words, what if the curve at iteration i is dependent on applying the fold command sequence sequentially to every line segment of the curve at iteration i-1? I spent time examining different fold command lists, and in so doing discovered a broader class of semi-self-similar fractals I refer to here as Folding Curves.

Notably, beautiful fractals can also be created using parametric L-systems in which the substitution rule is able to affect a parameter involved in that substitution [2]. The folding curve method described here is implemented more simply because iteration i of the fractal is computed from the sequence of folds in the fractal at iteration i-1, and the position along the fold command list. That is not to say a parametric L-system implementation is impossible, but I have not found this particular class of fractals in the literature, regardless of implementation.

Creating the Curves

The premise of Folding Curve creation is reliant on several parameters, most notably the fold command list. The fold command list specifies a set of folding directions (right or left, represented by 0's and 1's respectively). With each iteration, every existing line will be replaced by two new lines created by "folding" to the right or the left by a given angle (typically 90 degrees from each other, or 45 degrees each from the

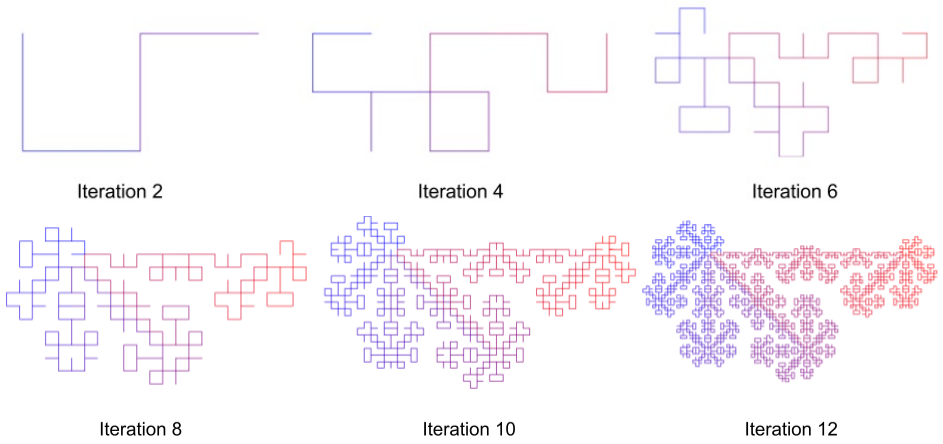
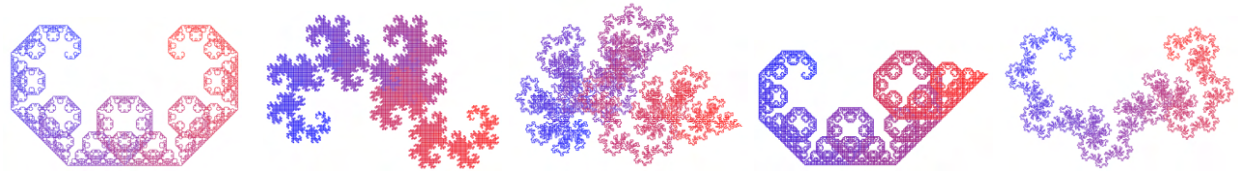


Figure 1: *The first 6 even iterations of the Half-Pinwheel Folding Curve. This is a Folding Curve with fold command list = [0110]. Note that iteration 0 is a horizontal line.*

line in the previous iteration). The default start for each folding curve (iteration 0) is a straight horizontal line. For example, the first few even iterations of the Folding Curve with fold command list = [0 1 1 0] are shown in Figure 1.

One way to look at the sequence of folds for the dragon curve is that it is inherently repetitive. That is, when you reach the end of the fold command list [0 1] you return to the beginning and continue. Therefore, the full sequence of folds will be something like RLRLRLRLRLRL... where R and L represent right and left turns, respectively. However, if you instead use a longer fold command list (say, [0 1 0]) and apply the same logic, you get a different repetitive pattern of turns. For a fold command list of [0 1 0] the fold list will look like RLRRLRRLRRLRRLRRLRR, which generates a different fractal when taken to sufficient iterations. Also note that for each new iteration, I restarted at the beginning of the fold command list. The code used to create these fractals can be found in the supplement to this paper.



(a) *[00]* **(b)** *[10]* **(c)** *[100]* **(d)** *[010]* **(e)** *[001]*

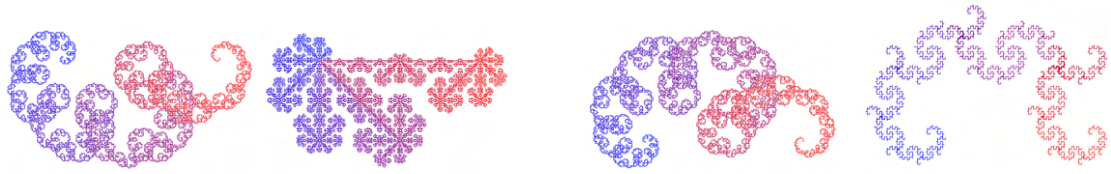
Figure 2: *One-segment folding curves with folding command lists of length 2 or 3. (a) The Levy C Curve imitated with folding command list [00], (b) The Dragon Curve imitated with folding command list [01], (c) Folding command list [100], (d) Folding command list [010], (e) Folding command list [001].*

Exploring the Parameter Space

As referenced above, running this code with fold command list = [0] or [0 0] (or [1] or [1 1] for an upside-down variant) imitates the Levy C curve (Figure 2a). Likewise, running the code with fold command list = [0 1] or [1 0] imitates the dragon curve, as demonstrated in Figure 2b. The output changes dramatically when the length of the fold command list is greater than 2. See, for example, results created with fold command lists of length three are shown in Figure 2c, 2d, 2e and results created with fold command lists of length four are shown in Figure 3.

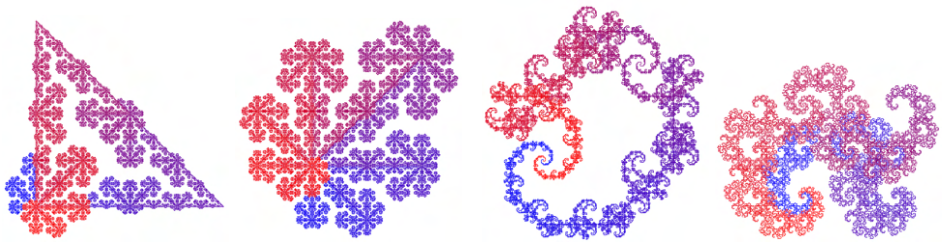
The curves presented so far have all begun with iteration 0 being a straight line. However, given the preponderance of 90 degree angles in the Folding Curve fractals, one natural alternate starting figure is a square. A few examples of the beautiful fractals that can be formed from a square start at iteration 0, along with their fold command lists, are shown in Figure 4. Note that these curves depend rotationally on the order in which the square is drawn; here I begin in the bottom left and progress clockwise to draw the square, though other implementations are possible as well (see the code in the supplement for more information).

The command list is in essence a binary sequence. Thus, Folding Curves offer us the opportunity to visualize infinite binary sequences in a unique and novel way if we implement those sequences as the fold command list. See Figure 5 for examples of such implementation. Note that to create these images, I generated the infinite binary sequence to at least length n, (where n is the number of lines present in the Folding Curve at the maximum iteration I was examining) and used that "static" sequence for each iteration leading up to the final one. In this way, the end of the fold command list is never reached. The application of Folding Curves to infinite binary sequences is here presented as a preliminary examination, and could be more deeply examined in the future. In particular, it might be of interest to see if there is a convergence in fractal form as the length of the binary sequence computer approaches infinity.



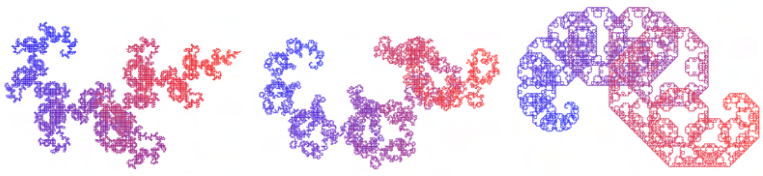
(a) *[0100]* **(b)** *[0110]* **(c)** *[1011]* **(d)** *[1100]*

Figure 3: *One-segment folding curves with folding command lists of length 4. Captions indicate the fold command list for each fractal.*



(a) *[0100]* **(b)** *[0110]* **(c)** *[0111]* **(d)** *[1000]*

Figure 4: *Folding curves initialized with a square. Captions indicate the fold command list for each fractal.*

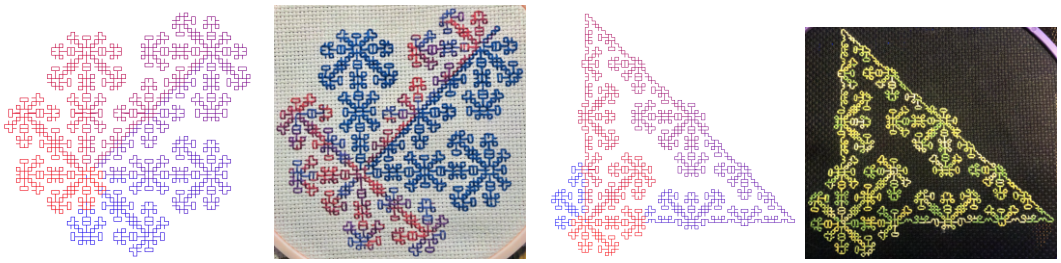


(a) *Thue Morse* **(b)** *Fibonacci word* **(c)** *Baum-Sweet*

Figure 5: *Visualizing infinite binary sequences. This figure utilizes the Folding Curve to visualize three well-known infinite binary sequences, (a) the Thue–Morse sequence [7], (b) the Fibonacci word [6], and (c) the Baum-Sweet sequence [5].*

Exploring Different Mediums

All even iterations of the Folding Curves contain only vertical and horizontal lines arranged on a grid. This property means that these fractals can be quite easily implemented in physical media, specifically through the use of backstitch cross-stitch [3]. Folding Curves can be implemented as fully backstitched cross-stitch – or the fully enclosed parts of the fractal could be space-filled if desired. Examples of Folding Curves implemented by hand in cross-stitch can be found in Figure 6.



(a) *[0110] (start with a square)* **(b)** *[0110] (start with a square)* **(c)** *[0100] (start with a square)* **(d)** *[0100] (start with a square)*

Figure 6: *Folding Curves in physical form: folding curves outlined at 10 iterations (a,c) and their corresponding handmade implementations in cross-stitch (b,d).*

Summary and Conclusions

The class of folding curves described in this paper was discovered through an atypical programmatic design choice. Exploration of this novel design space resulted in many beautiful fractals, and the figures presented here are in no way exhaustive. It remains to be seen if there is any greater mathematical significance to be found in this fractals.

Acknowledgements

Many thanks to Span Spanbauer, Jayson Lynch, Miranda Dawson, and Autumn Turner for their feedback.

References

[1] B. F. Lourenço, J. C. L. Ralha, and M. C. P. Brandao. "L-systems, scores, and evolutionary techniques." *Proceedings of the SMC 2009–6th Sound and Music Computing Conference,* Porto, Portugal, 23–25 July, 2009, pp. 113-118.

[2] M. McClure. "Parametric L-Systems and borderline fractals." *A preprint version of a "Mathematical graphics" column from Mathematica in Education and Research,* vol. 10, no. 3, 2005, pp. 1-10.

[3] L. Riddle. "Levy Dragon (Counted Cross Stitch and Back Stitch)." https://larryriddle.agnesscott.org/artwork/artwork.htm.

[4] S. Tabachnikov. "Dragon curves revisited." *The Mathematical Intelligencer,* vol. 1, no. 36, 2014, pp. 13-17.

[5] E. W. Weisstein, "Baum-Sweet Sequence." *From MathWorld–A Wolfram Web Resource.* https://mathworld.wolfram.com/Baum-SweetSequence.html.

[6] E. W. Weisstein, Eric W. "Rabbit Sequence." *From MathWorld–A Wolfram Web Resource.* https://mathworld.wolfram.com/RabbitSequence.html.

[7] E. W. Weisstein. "Thue-Morse Sequence." *From MathWorld–A Wolfram Web Resource.* https://mathworld.wolfram.com/Thue-MorseSequence.html.