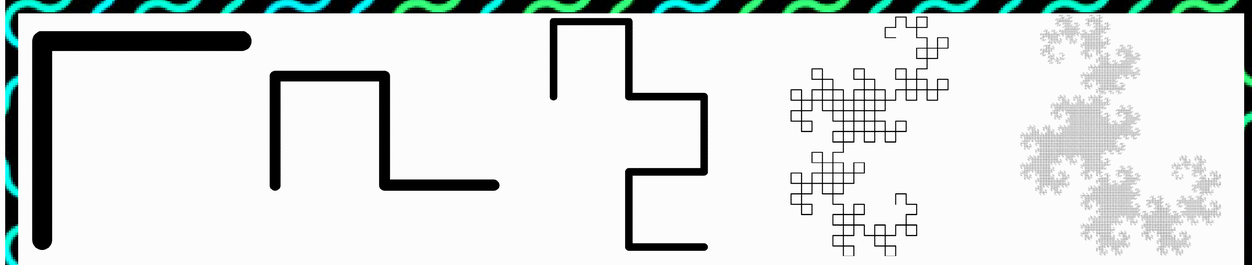
Dragon curve boundaries on the triangular grid

Kiran Pearce, supervised by Helena Verrill

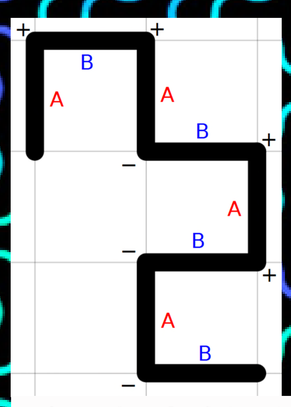
UNIVERSITY OF WARWICK

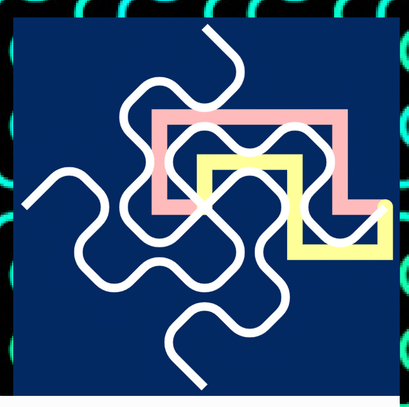
Introduction

Try folding a strip of paper in half, again and again, then unfold it so that all angles are 90°. The shape you have made is called a dragon curve. You may have noticed the curve fits on a square grid, and we can describe it with forwards directions (A,B) and left and right turns (+,−), in a word such as “A+B”. Then folding one more time is equivalent to the set of replacement rules: A→A+B, B→A−B (other letters invariant), which can be customised to form different curves. My supervisor Helena Verrill devised an algorithm (1) to convert this to a set of rules for drawing a boundary line around the outside, and my goal was to replicate this for folding curves drawn on the triangular grid.



Iterations 1, 2, 3, 8, and 15 of the original dragon fractal A→A+B





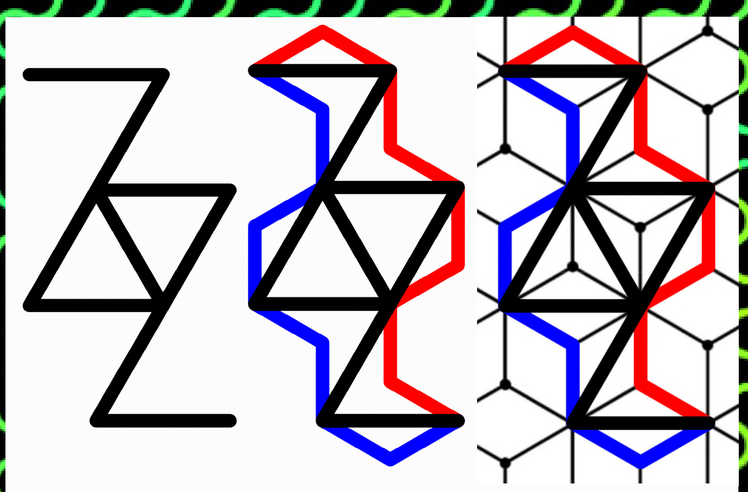
Iteration 3 of the dragon fractal- overlaid on a square grid, and separately with left (pink) and right (yellow) boundaries drawn

L-system

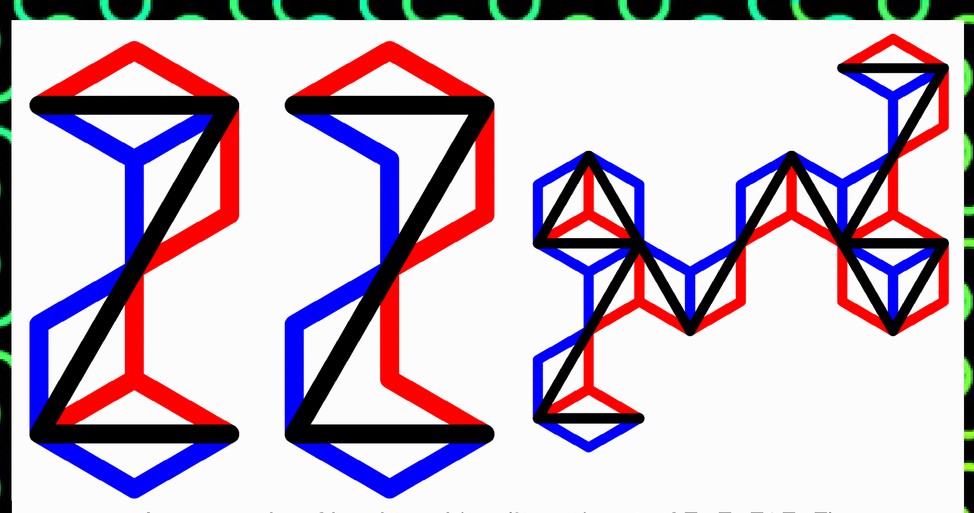
The set of directions is called an L-system, and on the square grid we use the alphabet {A,B,+,−}, where “A” and “B” distinguish between vertical and horizontal movement, since this allows us to produce more complex patterns. Arndt (2) devised a set of conditions for folding curves to be “plane-filling”- (fractal) dimension 2 and tileable. Note while the curve itself is 2D, its boundary has non-integer fractal dimension, so we call it a “fractile”.

Triangular grid

On the triangle grid it suffices to use the alphabet {F,0,+,−}, where “F” is any forward direction, but the turning angle is 120° and we also include the non-turn “0”. However, the boundary is drawn on a different grid, the rhombille tiling. I propose the alphabet {R,L,r,l,+,−}, where R and r look the same (like A and B).



Iteration 2 of F→F+F−F



An example of backtracking (Iteration 1 of F→F+F0F−F)
And double backtracking (Iteration 2)

Algorithm

Inspired by Verrill (1), I created the following algorithm:

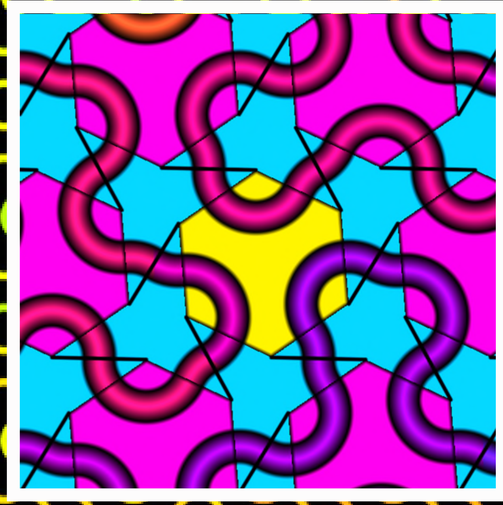
1. Draw a left boundary element (“R”) on each line
2. Replace small parts of the boundary until there is no more “backtracking”. The result is the replacement rule for “R”
3. Invert this rule to obtain the rule for l
4. Repeat for the right boundary

The left boundary consists only of {R,l,+,−}, and the right boundary {r,L,+,−}, so that we effectively have two independent L-systems.

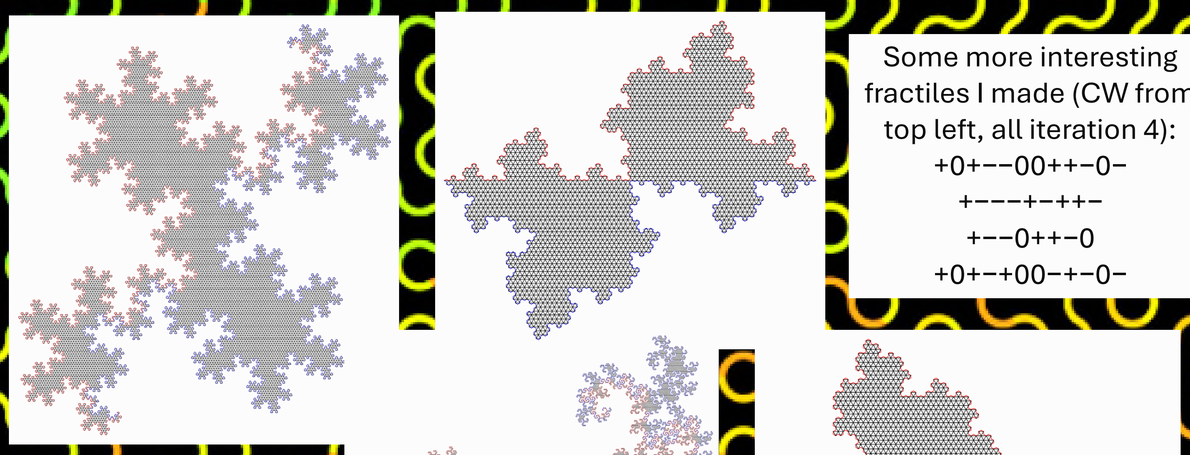
For a symmetric curve, we only need {R,L,+,−} and can omit step 3.

Hinged Truchet Tiles

The most promising proof approach was hinged tiles, which is a different visualisation of the iterative folding procedure. On the square grid, boundary lines can be drawn on tiles, and two different hinging operations used to produce tiles. Two different hinged tilings (triangular and hexagonal) can produce triangular fractiles, but only with 60° turns, and I couldn’t find a version that would cover all plane-filling triangular fractile examples.



Partway through the hinging process on the hexagonal grid, with new tiles forming in blue, and paths overlaid



Some more interesting fractiles I made (CW from top left, all iteration 4):
+0+−−00++−0−
+−−−+−++−
+−−0++−0
+0+−+00−+−0−

Further work

Unfortunately, I haven’t yet been able to formalise a proof of this result, so it will remain as a conjecture for now. The triangular grid functions very differently to the square one, with respect to L-systems, hinging, and symmetry. Further research is required and may need a new approach. Folding curves are also common on the hexagonal grid, the final regular Euclidean tiling, and I suspect this approach could be extended there too. Please do check out my GitHub page, where you can find the code to create your own folding curves, and where I will post any future developments!

← Check out the code! https://github.com/KPearce-13/fractals

1) https://doi.org/10.1016/j.tcs.2025.115363 (Verrill, 2025)
2) http://arxiv.org/abs/1607.02433 (Arndt, 2017)