Lindenmayer systems, fractals, and their mathematics

Alasdair McAndrew
`Alasdair.McAndrew@vu.edu.au`
College of Engineering and Science
Victoria University
PO Box 14428, Melbourne,
Victoria 8001, Australia

Abstract

Students are always asking for applications of mathematics. But more often than not, textbooks are filled with "applications" which are unimaginative, contrived, unrealistic, and uninteresting. However, there are of course numerous areas in which mathematics can be, and is, deployed to great effect. The purpose of this article is to introduce one such area: *Lindenmayer systems*, which beautifully joins mathematics and graphics, and investigate some of the mathematics—in particular their fractal dimension—and also the beauty of some of the graphics. We claim that the simplicity of the systems, and the complexity of their outputs, make for a simple way to introduce complexity—and modelling of the natural world—into a mathematics course, especially a course on finite mathematics, geometry, or computational mathematics.

1 Introduction

Lindenmayer systems, or more simply *L-systems* were initially developed by the biologist Aristid Lindenmayer in 1968 as a way of describing the growth of organisms. The initial object of study was a cynobacterial algae *Anabaena Catenula* which forms long strands like strings of pearls. The technique developed by Lindenmayer was found to be extraordinarily flexible and powerful, and was shown to be able to model plants of all sorts, as well as fractals. Although L-systems have been studied and used now for nearly 50 years, there are still many unsolved problems about their behaviour, and new uses for them are still being found [5, 7, 13].

2 Brief descriptions

Term rewriting

Simply put, an L-system is a method of creating strings of characters, according to a list of rules which describe how certain characters in the string are replaced by other characters. For example, we can construct strings consisting of 0's and 1's, starting with the string "0" and replacing characters according to the rules

$$\begin{aligned}0 &\longrightarrow 1\\ 1 &\longrightarrow 01\end{aligned}$$

The important point is that in any string, all characters are replaced simultaneously; that is, in parallel; so that *all* 0's are replaced with 1's, and all 1's are replaced with 01's. This produces the following sequence of strings, where n denotes the generation:

n	Strings
0	0
1	1
2	01
3	101
4	01101
5	10101101
6	0110110101101

To show this in more detail, consider the generations 5 and 6, starting with all the characters in generation 5:

Characters:	1	0	1	0	1	1	0	1
Result of rules:	01	1	01	1	01	01	1	01

A system such as this is known as a "term-rewriting system". Note that even with this simple system we can perform some mathematics; for example, to count the numbers a_n of 0's and b_n of 1's in each generation. Considering the rules above, we have

$$a_{n+1} = b_n, \qquad b_{n+1} = a_n + b_n$$

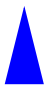
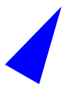
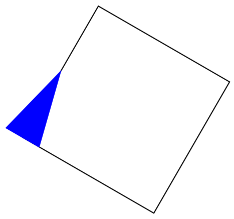
which can be written as a single equation in b_n as

$$b_{n+2} = a_{n+1} + b_{n+1} \Longrightarrow b_{n+2} = b_{n+1} + b_n$$

and this is the generating formula for the Fibonacci numbers F_n. With $F_0 = 0$ and $F_1 = 1$ it follows that $b_n = F_n$, $a_n = F_{n+1}$ and so the total length of the string at generation n is F_{n+2}.

Turtle graphics

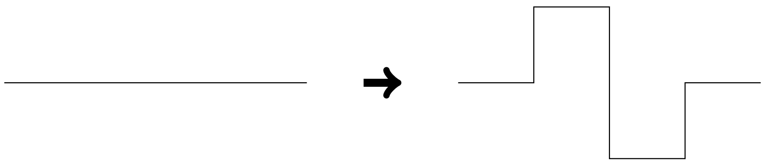
In order to turn such a string into graphics, the tool of choice is turtle graphics. This has its antecedents in the computer language Logo, pioneered by Seymour Papert at MIT in the late 1960's [9]. The "turtle" can move around, drawing as it goes (or moving without drawing, or even erasing); the importance of turtle graphics is that all commands are defined from the perspective of the turtle. For example, with the turtle starting facing up, the following Logo commands will draw a square on an angle:

Command:	`clearscreen`	`right 30`	`repeat 4 [forward 100 right 90]`
Explanation:	Clear the screen and return the turtle to its home position	Turn right 30 degrees	Move forward a distance of 100 and turn right 90 degrees. Repeat these two instructions four times.
Outcome:			

In some modern implementation of Logo, or of turtle graphics, the turtle is denoted by an arrowhead, as here, which shows its current position and direction.

A fractal example

To show how L-systems, turtles, and graphics mix, we shall look at Koch's "quadratic curve", where a line is transformed into a square wave shape:



This is continued recursively; the new shape may be considered to have eight line segments each one-quarter the length of the original line, and each of these is transformed, and so on. Using the symbols:

`F`:	move forward one step
`+`:	turn left 90 degrees
`-`:	turn right 90 degrees

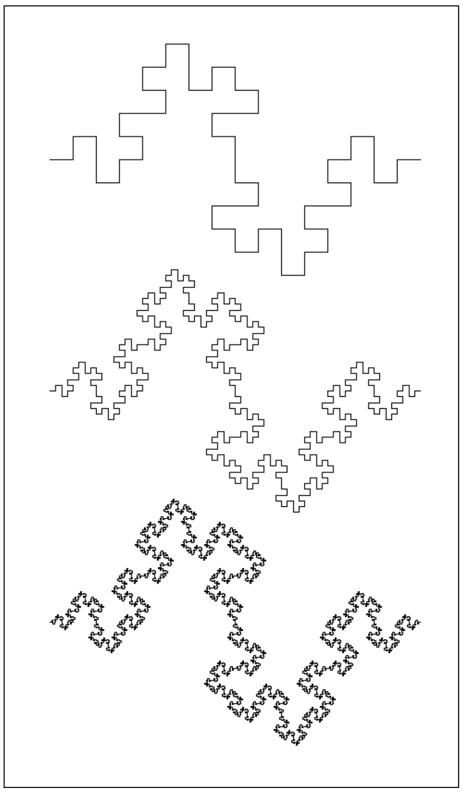
the line transformation can be written as the L-system rule:

$$\mathtt{F} \longrightarrow \mathtt{F}+\mathtt{F}-\mathtt{F}-\mathtt{FF}+\mathtt{F}+\mathtt{F}-\mathtt{F}$$

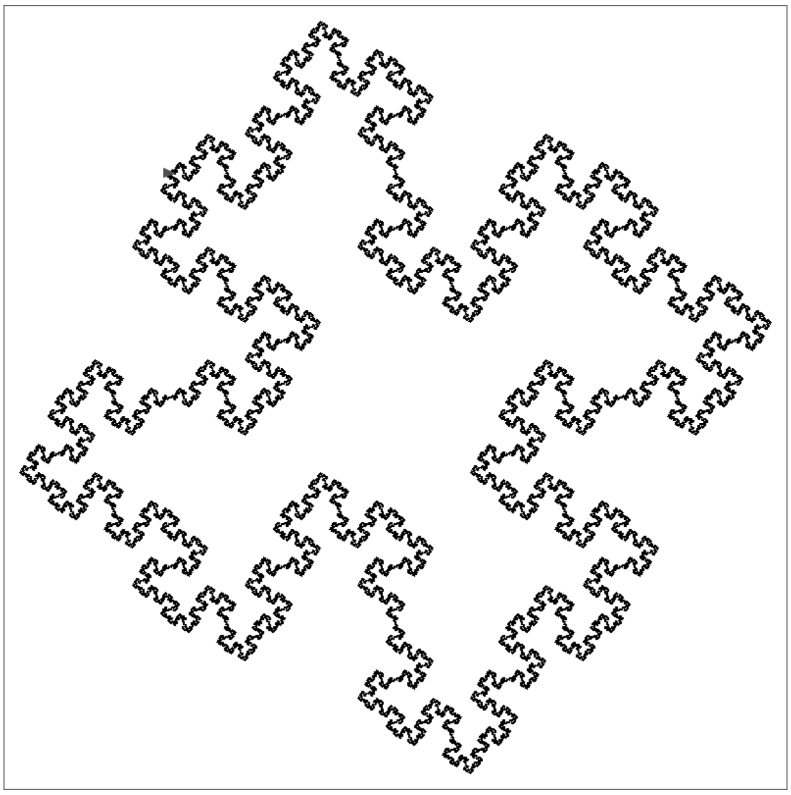
Since there is no current modern implementation of Logo available for all platforms, we shall develop this using the turtle graphics library of Python. Note that there are in fact several excellent L-systems implementations in Python [3], but we shall adopt Buchberger's "black-box, white-box" approach [4] and build up some simple procedures from scratch. This allows a much deeper understanding of the workings of L-systems. Python's turtle graphics library is full featured, and programs written in Logo (or any other turtle graphics implementation) can be translated into Python with little effort.

```
def kochq(level, size):
    if (level==0):
        t.fd(size)
    else:
        kochq(level-1, size/4); t.lt(90)
        kochq(level-1, size/4); t.rt(90)
        kochq(level-1, size/4); t.rt(90)
        kochq(level-1, size/4)
        kochq(level-1, size/4); t.lt(90)
        kochq(level-1, size/4); t.lt(90)
        kochq(level-1, size/4); t.rt(90)
        kochq(level-1, size/4)
```

If this transformation is applied to each side of a square, the result is called "Koch's quadratic snowflake", and the results at different levels are shown in Figure 1.



$n = 2, 3, 4$



$n = 6$

Figure 1: The Koch quadratic snowflake

As $n \to \infty$, the figure becomes a *fractal*, in that it is self-similar at all magnifications: in a sense it has an infinite complexity. Note that at each generation the length of the curve is doubled, so the final result has infinite length but bounds a finite area. L-systems can be used to draw a very large variety of fractal shapes, as well as space-filling curves [10].

In order to extend the shapes achievable by L-systems, it will be necessary to consider branching. Standard symbols for branching are the (square) brackets, which translate to turtle graphics as follows:

`[`:	take note of the turtle's current position and direction,
`]`:	return the turtle to the position and direction (without drawing the path).

Normally these would be implemented using a stack, so that each closing bracket pops from the top of the stack the values which had been pushed onto it by the corresponding opening bracket.

One very simple system is given by the rule

$$\texttt{F} \longrightarrow \texttt{F[+F]F[-F]F}$$

where as above the `+` and `-` symbols indicate a turn—in this case a branch—to the left or right respectively. This means that the group of symbols `[+F]` correspond to constructing a branch to the left. Starting with a straight line, one application of the rule will produce the shape shown in Figure 2.

The rule is now applied recursively to every length in the new shape, of which in the first generation there are five, corresponding to the five `F`'s in the rule. Rather than using a stack, we can implement branching by the simple expedient of turning in the required direction, drawing the object, then bringing the turtle back and turning the other way, so as to be oriented in the initial direction.

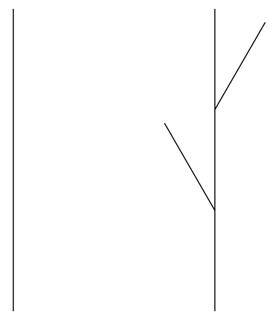


Figure 2: The branching shape defined by the rule $\texttt{F} \longrightarrow \texttt{F[+F]F[-F]F}$

```
def edgetree(level, size, angle):
    if (level==0):
        t.fd(size)
    else:
        edgetree(level-1, size/3, angle); t.lt(angle)
        edgetree(level-1, size/3, angle); t.bk(size/3); t.rt(angle)
        edgetree(level-1, size/3, angle); t.rt(angle)
        edgetree(level-1, size/3, angle); t.bk(size/3); t.lt(angle)
        edgetree(level-1, size/3, angle)
```

Three different generations are shown in Figure 3, with an angle of 26°. Already this has a lifelike look, and it's remarkable that such simple rules can lead to such an organic form.

Edge rewriting and node rewriting

The examples so far have used *edge rewriting* where a single edge is replaced by scaled copies of itself. However, another way of applying L-systems is by *node rewriting*, where as well as edges we include "nodes". A node is indicated by an `X`, and may be considered as a placeholder, where something will be added in future generations. In biological terms, an `X` is like a bud. For example, consider the rules

$$\texttt{F} \longrightarrow \texttt{FF} \tag{1}$$

$$\texttt{X} \longrightarrow \texttt{F[+FX][-X]F[+FX]--FX} \tag{2}$$

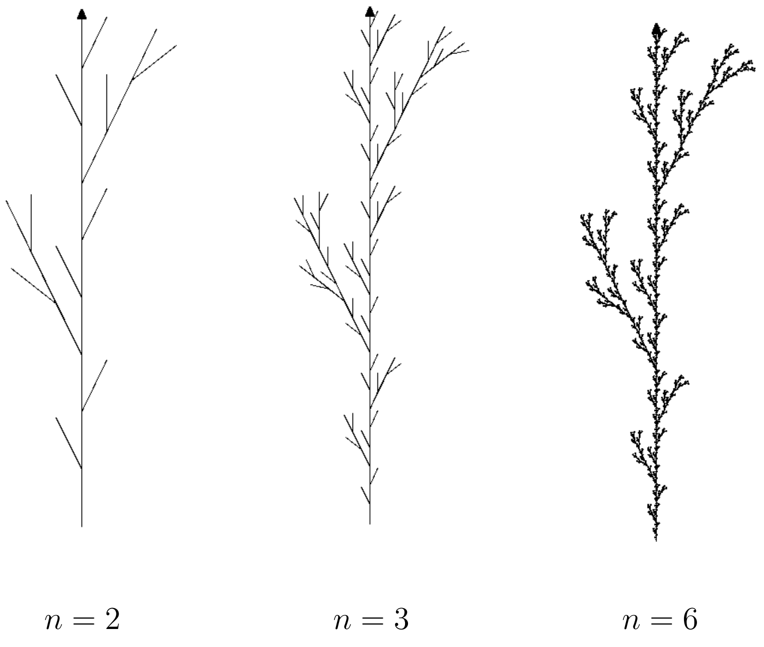


Figure 3: Different generations of `F`$\longrightarrow$`F[+F]F[-F]F`

and we start with `X`. To make sense of this, consider the second rule without the `X`'s:

`F[+F][-]F[+F]--F.`

This may be considered the "base" tree, and is shown in Figure 4.

We can easily use the brackets to save and retrieve the turtle's position. What we can do is rewrite the strings first, and then finally draw the resulting string.

```
rules = {'F':'FF', 'X':'F[+FX][-X]F[+FX]--FX'}
ls = 'X'
for i in range(4):
    ls = ''.join(rules[s] if s in rules else s for s in ls)
```

This last somewhat curious command is simply a way of travelling through the current string `ls` and replacing each 'X' and 'F' according to the rules. Having got the string, we can easily draw it:

```
def draw_lsys(str,size,angle):
    stk = []  # a stack to keep positions and headings, implemented as a list
    L = len(str)
    for i in tqdm(str,total=L):   # the tqdm library adds a progress bar
        if i=='F': t.fd(size)
        elif i=='+': t.lt(angle)
        elif i=='-': t.rt(angle)
        elif i=='[':
            p = t.pos()
            h = t.heading()
            stk.insert(0,[p,h])   # insert current position and heading at top of stack
```

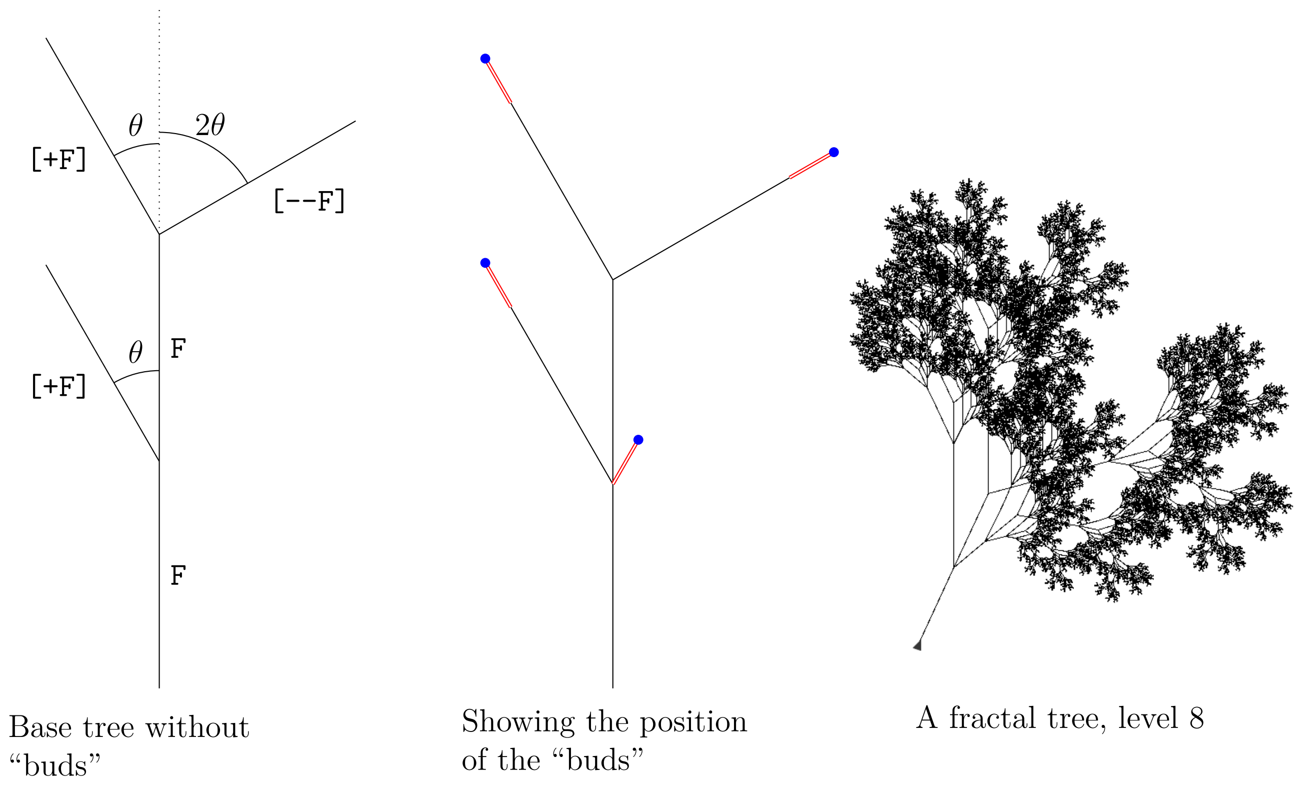


Figure 4: A tree created with node rewriting

```
        elif i==']':
            t.penup()
            ph = stk.pop(0)   # retrieve position and heading from top of stack
            t.setpos(ph[0])
            t.seth(ph[1])
            t.pendown()
```

An example (drawn on an angle) with level 8 is shown on the right in Figure 4. Even with the unrealistic straight line branches, this has a remarkably tree-like shape and form. Greater realism can be obtained by changing the width of the branches, and adding a little randomization—both beyond the scope of this article. Note that this generic technique can be used to draw the branching weed shown in figures 2 and 3.

3 Fractal dimensions and lengths

We have used the term "fractal" without defining it, and in fact we won't provide a formal definition. For our purposes, a *fractal* will be any non-smooth shape which retains its complexity at arbitrary magnification. One way of distinguishing a fractal is by its *dimension*.

We can define the dimension of an object as the power by which its value changes as the linear dimension is increased by a scale factor k. For example, if the sides of a cube are increased by k, then its volume is increased by k^3, whence its dimension is 3. Consider the Koch "quadratic snowflake" transformation described earlier. Each transformation of a line produces eight segments each one quarter the length of the original line; in other words if each new segment is the same length as the original line, we have the line length increasing by four but the total length of all segments by 8. Since

$$8 = 4^{3/2}$$

it follows that the dimension of this curve is 3/2. This notion of dimension is variously known as the *Hausdorff dimension* or the *Hausdorff-Besicovitch dimension*.

Consider again the branching tree shown in figures 2 and 3. At each iteration, the tree is divided in length by 3, for five new segments. If the segments are the same length as the original line, then the line is increased by a scale factor of 3, and the total length of all parts of the tree by 5. The dimension d therefore can be computed as:

$$5 = 3^d \implies d = \frac{\log 5}{\log 3} \approx 1.465.$$

A fractal can be defined as an object with a non-integer Hausdorff dimension.

The branching tree just discussed was an edge-rewriting system; it is more difficult to compute the dimension of a node-rewriting system, such as the example shown in Figure 4, as the self-similarity is more subtle. But we can approximate the fractal dimension by investigating the *box-counting dimension*; also known as the *Minkowski-Bouligand dimension*. This works by overlaying the fractal with square boxes of size length b, and counting the number of boxes $N(b)$ required to cover the fractal. The dimension is then defined as

$$\lim_{b\to 0} \frac{\log(N(b))}{\log(1/b)}.$$

To see why this makes sense, consider a planar figure: a square, circle for example. We would in general expect that the number of boxes of size b required to cover the figure will be roughly the area of the figure divided by the area of a box, so that

$$N(b) \approx \frac{A}{b^2}$$

with the approximation becoming more accurate as b becomes smaller. Then

$$\begin{aligned}\frac{\log(N(b))}{\log(1/b)} &\approx \frac{\log(A/b^2)}{\log(1/b)}\\ &= \frac{\log(A) + 2\log(1/b)}{\log(1/b)}\\ &= \frac{A}{\log(1/b)} + 2.\end{aligned}$$

But as $b \to 0$, $\log(1/b) \to \infty$ and so the first term vanishes, leaving 2.

We can approximate this result by considering the fractal as an image of size $N \times N$, broken into n^2 subimages of size b^2, where $nb = N$. The value $N(b)$ will then be the number of subimages intersecting the fractal. The quotient to be computed will then be

$$\frac{\log(N(b))}{\log(N/b)}.$$

Note that the denominator here is $\log(N/b) = \log(n)$: this is because in a discrete situation as here, $b = 1$ is the smallest possible value.

For our tree, table 1 shows the various box-counts and quotients, for decreasing values of b, from which we infer that the dimension of the tree in Figure 4 is approximately 1.7.

b	512	256	128	64	32	16	8	4	2	1
n	2	4	8	16	32	64	128	256	512	1024
$N(b)$	4	16	60	178	528	1608	5152	16656	51168	132939
$\dfrac{\log(N(b))}{\log(n)}$	2.0000	2.0000	1.9690	1.8689	1.8089	1.7752	1.7616	1.7530	1.7381	1.7020

Table 1: Computing the box-counting dimension

A very good, simple, clear account of the box-counting dimension is given by Falconer [6]. For applications of dimensions of L-systems, see Alfonseca et al [1]. Further investigations of the fractal dimension can be found in the bibliography of Prusinkiewicz and Lindenmayer [10].

One final matter is to consider the length of the strings obtained by the rewriting. We have only touched on the simplest possible strings; strings (and especially in three dimensions) may have many more symbols. But for example consider the tree with the rules defined in rules (1) and (2). Suppose we define f_n, x_n, z_n to be the numbers of `F`'s, `X`'s and other symbols respectively at generation n. From examination of the rules, we see that

$$\begin{aligned} x_n &= 4x_{n-1}, \quad x_0 = 1 \\ f_n &= 2f_{n-1} + 5x_{n-1}, \quad f_0 = 0 \\ z_n &= z_{n-1} + 11x_{n-1} \end{aligned}$$

The first equation can be solved immediately as $x_n = 4^n$, and now the second equation can be written as

$$f_n = f_{n-1} + 5(4^{n-1}), f_0 = 0$$

and this can be solved by standard techniques for recurrence relations as

$$f_n = \frac{5}{2}(4^n - 2^n).$$

The last equation is

$$z_n = z_{n-1} + 11(4^{n-1})$$

which can be solved as

$$z_n = \frac{11}{3}(4^n - 1).$$

This means that the total length of the string at generation n is

$$\begin{aligned} x_n + f_n + z_n &= 4^n + \frac{5}{2}(4^n - 2^n) + \frac{11}{3}(4^n - 1) \\ &= \frac{43}{6}4^n - \frac{5}{2}2^n - \frac{11}{3} \\ &= \frac{1}{6}\Big(4^n - 15(2^n) - 22\Big). \end{aligned}$$

Further examples and discussions are given by Rozenberg and Artomaa [11].

4 Conclusions

Fractals are seen as a modern geometry for describing the natural world [8]. In contrast to standard geometrical figures such as squares and circles, fractals are better able to describe the complexity of natural shapes. Fractals have been used to model branches of trees, river systems, blood vessels and nerves in the human body, mountain ranges, and many others. However, the complexity of fractals has precluded their adoption in the classroom; mathematics teachers and learners have instead used simple shapes as approximations (as in the famous "spherical cow" joke [2]), and of course such approximations have been used to great effect. But instead of simplifying the natural world to suit our mathematics, fractals allow us to work with the natural world using a greater complexity. Of all fractal shapes, L-systems are amongst the simplest to understand, to program, and to analyse. Given the importance of computer coding in modern mathematics curricula [12], there would appear to be a strong case for the consideration of L-systems in such curricula.

References

[1] Manuel Alfonseca and Alfonso Ortega. "Determination of fractal dimensions from equivalent L systems". In: *IBM Journal of Research and Development* 45.6 (2001), pp. 797–805.

[2] Lera Boroditsky and Michael Ramscar. "First, we assume a spherical cow..." In: *Behavioral and Brain Sciences* 24.4 (2001), pp. 656–657.

[3] Frédéric Boudon et al. "L-Py: an L-system simulation framework for modeling plant architecture development based on a dynamic language". In: *Frontiers in plant science* 3 (2012).

[4] Bruno Buchberger. "Should students learn integration rules?" In: *ACM SIGSAM Bulletin* 24.1 (1990), pp. 10–17.

[5] Laura Ciobanu, Murray Elder, and Michal Ferov. "Applications of L-systems to group theory". In: *arXiv preprint* `https://arxiv.org/pdf/1705.02809.pdf` (2017).

[6] Kenneth Falconer. *Fractals: A very short introduction.* OUP Oxford, 2013.

[7] Jeff Hanes and R Paul Wiegand. "Using L-Systems to Generate Fault Trees for Benchmarking and Testing". In: *The Twenty-Ninth International Flairs Conference.* 2016.

[8] Benoit B Mandelbrot. *The Fractal Geometry of Nature, Revised and Enlarged Edition.* New York, WH Freeman and Co, 1983.

[9] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas.* Basic Books, Inc., 1980.

[10] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty Of Plants.* Springer Science & Business Media, 2012.

[11] Grzegorz Rozenberg and Arto Salomaa. *The Mathematical Theory of L-systems.* Vol. 90. Academic press, 1980.

[12] Conrad Wolfram. "A practical approach to teaching maths". In: *MSOR Connections* 4.4 (2004), p. 60.

[13] Yanhe Zhu et al. "A distributed and parallel control mechanism for self-reconfiguration of modular robots using L-systems and cellular automata". In: *Journal of Parallel and Distributed Computing* 102 (2017), pp. 80–90.