Making Melodious Music From Fractals

William D. Reames

Christopher Newport University

HONR 379: Fractals and Infinity

May 04, 2021

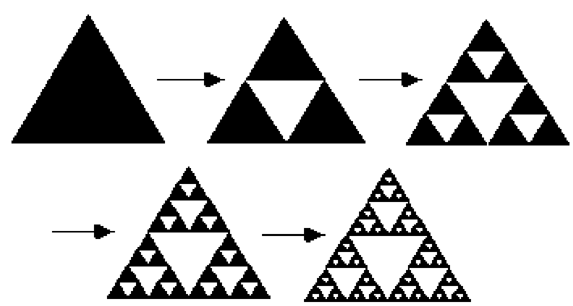
Making Melodious Music From Fractals

Fractals and other various symmetrical or self-similar patterns can be found everywhere in the natural world (Gleick, 2008). Whether it is the branches on trees, the arrangement of clouds in the sky, or the jagged edge of a coastline, fractals can be seen in many different places (Gleick, 2008). One of these areas that is especially interesting is how fractals relate to music. More specifically, the idea of converting a fractal into a piece of music or a melody. That is the goal of this paper: to research a method that could be used to convert a fractal into a melodious piece of music and to explore the creation of a computer program to automate this conversion.

Literature Review

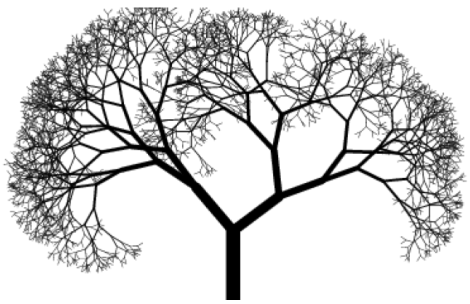
Fractals

"In the mind's eye, a fractal is a way of seeing infinity," (Gleick, 2008, p. 98). This is what Gleick says to help describe fractals in their book, *Chaos: Making a New Science*, and it helps paint a good picture of what fractals are. Fractals, in essence, are geometric shapes that are *infinitely* self-similar (Gleick, 2008). What this means is that if a person zoomed in on a specific portion of a fractal, they would see the same image, even if they continued to zoom in an infinite number of times (Gleick, 2008). This is what it means for something to be infinitely self-similar. The Sierpinski Triangle (also known as the Sierpinski Gasket) is a great example of this. To form the Sierpinski Triangle, take an equilateral



triangle, scale it down by ⅓, then make two additional copies of the triangle that are shifted over to the two corners of the original (Gleick, 2008; Mishra & Mishra, 2007). After completing this process, an image of the original triangle with the center removed can be seen. From here, this process can be repeated again, but with the resulting image as the new base. As this process is repeated to infinity, the image of the Sierpinski Triangle is formed. The first five iterations of the Sierpinski triangle can be seen on the right of the previous page (Devaney, 1995).

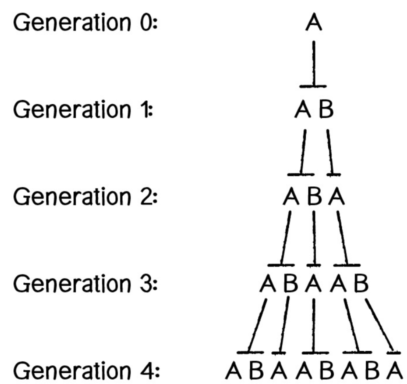
Fractals are not just a theoretical concept, though, and they can appear in various natural forms as well (Gleick, 2008). This includes certain features of nature such as clouds, trees, coastlines, and blood vessels, just to name a few (Gleick, 2008). As an example, the generation



of a tree fractal can be seen depicted to the left (Poltorak, 2019). However, even though all of these natural structures exhibit self-similarity in their geometry, they do not *perfectly* exhibit the characteristics of a fractal as they usually have some degree of variance in their self-similarity and do not iterate all the way to infinity (Gleick, 2008). Even so, they do still show that fractals are found in many different parts of the natural world. One area of the natural world where fractals can appear is music, with one aspect of this being the process of turning a fractal into a musical composition. This is where Lindenmayer systems come in.

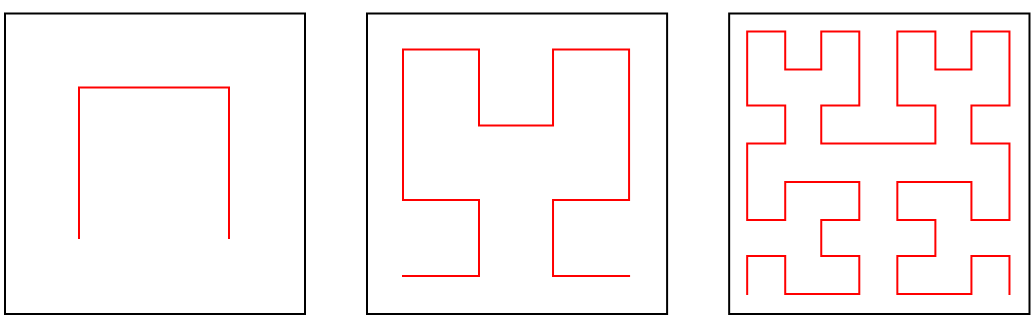
L-Systems

Lindenmayer systems (L-systems) were originally created by Arstid Lindenmayer to describe the growth patterns that can be found in biological organisms (Mason & Saffle, 1994; Mishra & Mishra, 2007; Prusinkiewicz, 1986a; Prusinkiewicz, 1986b). L-systems represent a way to generate a string of characters by recursively and simultaneously rewriting certain symbols in the string (Mason & Saffle, 1994; Mishra & Mishra, 2007; Prusinkiewicz, 1986a; Prusinkiewicz, 1986b). The L-systems described in this paper will be D0L-systems, which means that they are all deterministic and context-free, rather than non-deterministic and context-sensitive (Mason & Saffle, 1994; Mishra & Mishra, 2007). In this process, an L-system starts with an initial rule or axiom (e.g., an initial character), along with additional rules that are substituted for the axiom when the L-system is iterated (Mason & Saffle, 1994; Mishra & Mishra, 2007; Prusinkiewicz, 1986a; Prusinkiewicz, 1986b). L-systems can get a bit more complex than that, but that is the most basic form. To give an example, suppose the axiom of a given L-system is set to 'A', with rules A ⇒ AB and B ⇒ A. From here, the axiom is iterated based on the given rules, making the 'A' convert into 'AB'. This process can continue further converting 'AB', into 'ABA', since the first character 'A', is converted into 'AB', and the second character 'B' is converted into 'A' based on the rules described previously. This eventually gives us the sequence: A, AB, ABA, ABAAB, ABAABABA, and so on. This L-system expansion is visualized to the right (Shiffman, 2012). From this, certain aspects of fractal-like self-similarity can be seen, and it is clear how a person can begin to formulate an idea on how the two could be related (Mason & Saffle, 1994; Mirsha & Mirsha, 2007).



Not only can L-systems generate these self-similar patterns, but they can also be used to create various different pre-existing fractals such as the Koch Curve, the Sierpinski Gasket, the Cantor Set, and the Hilbert Curve, among others. (Mirsha & Mirsha, 2007; Prusinkiewicz, 1986a). In order to do this, the L-system strings need to be converted into images, and something called a "turtle" is used to aid in this process (Mason & Saffle, 1994; Mishra & Mishra, 2007; Prusinkiewicz, 1986a; Prusinkiewicz, 1986b). The turtle can be thought of as a pen in some ways that simply follows a given set of instructions based on a specific iteration of an L-system

(Mason & Saffle, 1994; Mishra & Mishra, 2007; Prusinkiewicz, 1986a; Prusinkiewicz, 1986b). In the most basic sense, the instructions a turtle can follow are: F : move forward and draw a line, + : Turn right or rotate clockwise based on a predetermined angle, - : Turn left or rotate counterclockwise based on a predetermined angle (Mason & Saffle, 1994; Mishra & Mishra, 2007; Prusinkiewicz, 1986a; Prusinkiewicz, 1986b). All other characters are ignored by the turtle (Mason & Saffle, 1994; Mishra & Mishra, 2007; Prusinkiewicz, 1986a; Prusinkiewicz, 1986b). There are other settings that could be implemented, but these are the settings that will be used in this paper. To give an example of this, start with an axiom of A, with rules A ⇒ +BF−AFA−FB+ and B ⇒ −AF+BFB+FA−, and use the same turtle rules described previously but with + and - each turning 90° (π/2 radians) respectively. The resulting sequence after the L-system is iterated is A, -BF+AFA+FB-, -+AF-BFB-FA+F+-BF+AFA+FB-F-BF+AFA+FB-+F+AF-BFB-FA+-, and so on. When each iteration is drawn using the given turtle settings, the following images are produced:



(Chakravarty, n.d.). From this, the Hilbert Curve can be seen emerging via this L-system (Chakravarty, n.d.; Prusinkiewicz, 1986a). Now that there is an understanding of how to represent fractals using L-systems, the process of converting the L-system (or fractal) into a musical composition can begin.

Making Music From L-System Fractals

The conversion of an L-system drawing to a musical composition follows a fairly simple process. This begins by drawing the given L-system or fractal shape using the turtle described earlier (Mason & Saffle, 1994; Prusinkiewicz, 1986b). Following this, the process continues by traversing the line drawn by the turtle and interpreting every horizontal line segment as a note, with the note duration being the length of the segment and the pitch being represented by its height in the image (Mason & Saffle, 1994; Prusinkiewicz, 1986b). To give a visual representation, this is what the resulting score would look like for the Hilbert Curve, taken from an example in Prusinkiewicz's (1986b) work describing this process:

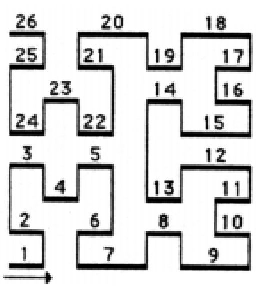


Fig. 3. Traversing the Hilbert curve.

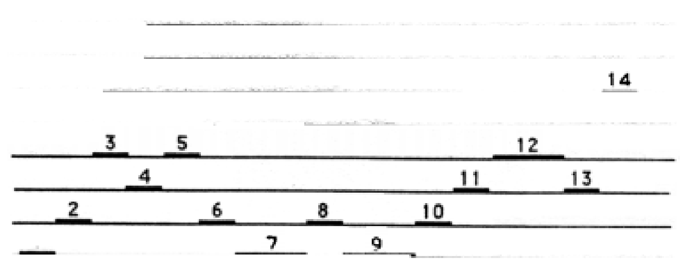
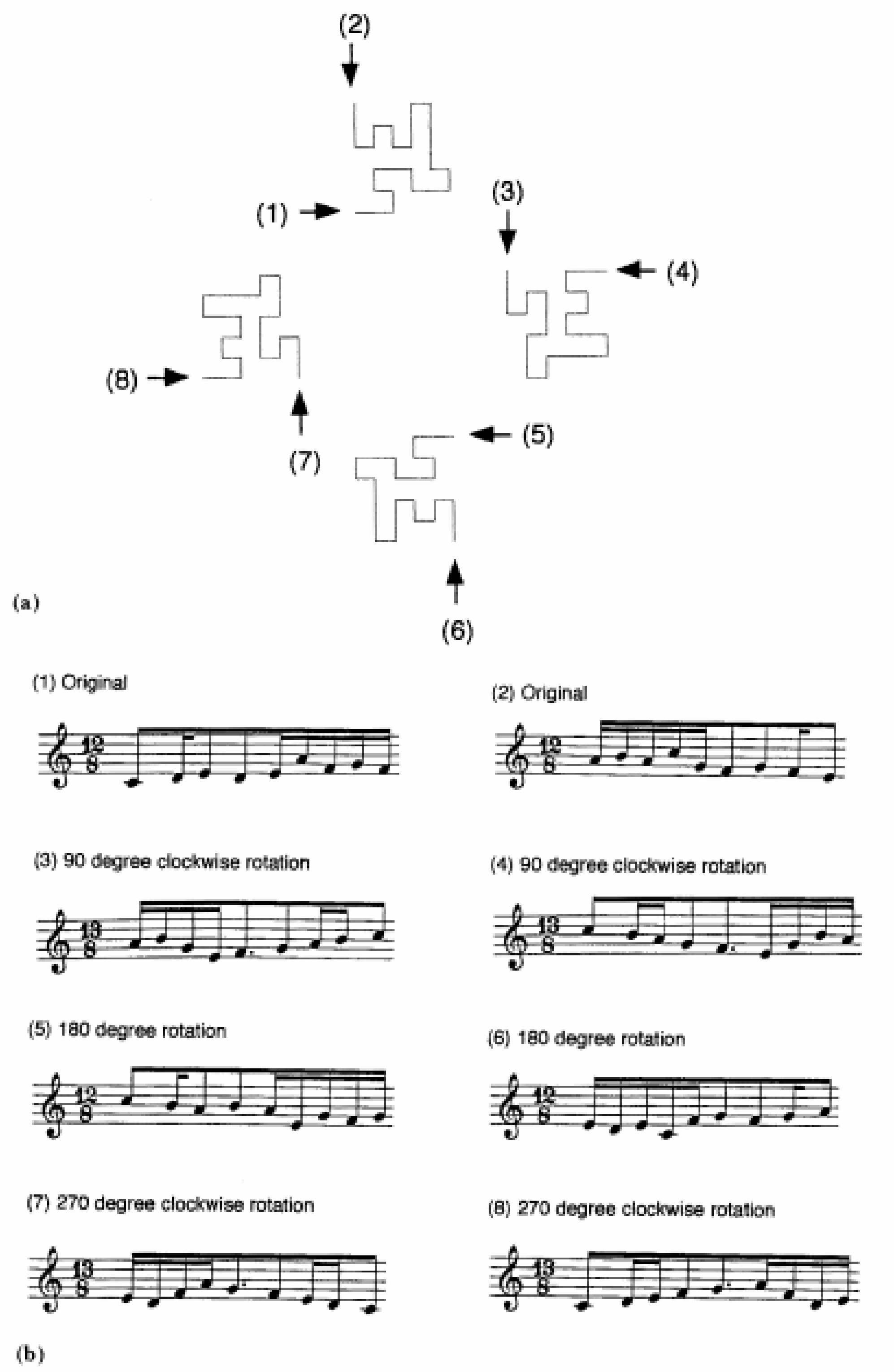


Fig. 4. The score associated with the Hilbert curve in the piano-roll notation.

Fig. 5. The score associated with the Hilbert curv in the common musical notation.

Furthermore, it should also be noted that every image resulting from the L-system can be interpreted into eight different melodies, with options to start the traversal at the beginning or end of the image, and to rotate it 0° (0 radians), 90° (π/2 radians), 180° (π radians), or 270° (3π/2 radians) (Mason & Saffle, 1994). Additionally, it is also possible to combine some of these different melodies to create harmonies from a single image.

An example of the eight different melodies that can be created from one L-system or fractal are shown in the following image from Mason and Saffle's (1994) work on the subject:



Interestingly, even though this process seems fairly straightforward and easy to implement, I was not able to find any software during my research to replicate this process for my own selection of L-system fractals. This is what led me to the main focus of this paper: creating a program that could be used to convert a fractal into music through the use of L-systems.

Methodology

This program was implemented using Python. To aid in this process, I found preexisting code on GitHub that could be used for L-system expansions as well as for forming musical sounds in Python. These repositories were https://github.com/Mizzlr/L-Systems-Compiler-And-Renderer (Ahamed, 2016) and https://github.com/weeping-angel/Mathematics-of-Music (Jain, 2020). My plan was to create a bridge between these two repositories to allow for the L-system expansion to be converted into musical notes and sounds.

Results

In order to implement this program, there were several major steps I had to take. Firstly, I needed to be able to expand an L-system based on a given set of rules. Secondly, I had to modify the L-system expansion so it could be read as musical notes. Lastly, I had to output the music to a sound file so that the user could play the music generated by the program. Thankfully, the first and last steps of this process were implemented by the preexisting repositories I had found and only required some minor changes such as altering the output, changing the turtle drawing to be optional, and adding an option to control note duration. Therefore, the focus of this project was on the second step: converting the L-system expansion into a series of notes while following the guidelines specified by the works of Prusinkiewicz (1986b) as well as Mason and Saffle (1994).

After using the L-Systems-Compiler-And-Renderer to expand a set of L-system rules, I looped over the final string and processed it just like the turtle would (Ahamed, 2016). I kept track of the current direction that the turtle would be facing, and recorded the pitch and duration of the various notes accordingly. If the turtle moved up, I increased the pitch by one; if the turtle moved down I decreased the pitch by one; if the turtle moved horizontally, I increased the note duration by one; if the turtle moved vertically, I reset the duration to zero. Finally, after recording the duration and pitch of a given note, I stored the values into a list of custom Note objects whenever the turtle turned off the horizontal plane. This did run into a few issues, though, when evaluating sequences such as ‘F+-F’, since the turtle would turn off of the horizontal axis, preparing to move vertically, but turn back to continue the same note before moving again. To solve this, I removed the last note from the list, and replaced it with the new (current) note whenever the current pitch and duration matched that of the previous note. The implementation of this method is shown in the figure on the right of this page.

```
for character in generated_str:
    if character in movement_chars:

        # Records the duration and pitch of the tones based on the turtle movements
        if direction == Dir.LEFT or direction == Dir.RIGHT:
            current_duration += 1
        elif direction == Dir.UP:
            current_pitch += 1
            current_duration = 0
        elif direction == Dir.DOWN:
            current_pitch -= 1
            current_duration = 0

    elif character == '+' or character == '-':
        if (direction == Dir.LEFT or direction == Dir.RIGHT) and current_duration:
            # Pops out the previous tone if the pitches match.
            # This is to keep the current pitch duration accurate.
            if notes and notes[-1].pitch == current_pitch and \
                    (notes[-1].duration == current_duration - 1 or
                     notes[-1].duration == current_duration):
                notes.pop()
            notes.append(Note(current_pitch, current_duration))

        if character == '+':
            direction = turn_right(direction)
        elif character == '-':
            direction = turn_left(direction)
```

Following this, I converted the list of Notes that I made into a single string that could be read by the music-making system I used for the program, Mathematics-of-Music (Jain, 2020). The Mathematics-of-Music program takes an input string consisting of character representations of a note and dashes (Jain, 2020). For example, the tune, *Twinkle Twinkle Little Star*, would look like this:

‘C-C-G-G-A-A-G--F-F-E-E-D-D-C--G-G-F-F-E-E-D--G-G-F-F-E-E-D--C-C-G-G-A-A-G--F-F-E-E-D-D-C’ (Jain, 2020). To aid in this process, I made a method in my Note class to convert it into a string representation. This string was the mapping of the pitch to the note character (i.e., 0 ⇒ ‘C’, 1⇒ ‘D’, 2 ⇒ ‘E’, …) formatted as ‘<character>-’ and repeated as many times as the duration. This way, the note would be held out for as long as the duration specified. For example, if a note had a value of one, and a duration of three, the resulting string would be ‘D-D-D-’. Using this str() method, I was able to iteratively add more characters to a running string of music notes. This string was then sent to the Mathematics-of-Music program to be outputted as a “.wav” sound file (Jain, 2020). The implementation for this code is found on the left side of this page.

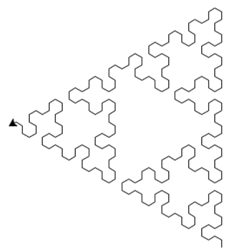
```
def __str__(self):
    note_char = self.pitch_map[self.pitch]
    return '{}-'.format(note_char) * self.duration
```

```
music_str = ''
for note in notes:
    music_str += str(note)
```

Furthermore, I also implemented input fields to allow for the user to select the number of iterations as well as the initial direction used to interpret the music. This was done in order to allow the user to produce all eight melodies from the single image as described by Mason and Saffle (1994). The full implementation of this program can be found on my GitHub repository here: https://github.com/wdreames/fractals_to_music.

Discussion

After completing this process and creating the functional program, I was successfully able to implement Prusinkiewicz’s (1986b) along with Mason and Saffle’s (1994) method of converting fractals, or more specifically, L-system representations of fractals, into music. Furthermore, this Python script could be applied to any fractal pattern as long as it can be formulated using L-systems with a turning angle of 90° ($\pi/2$ radians) such as the Hilbert curve or Peano curve (Mason & Saffle, 1994; Prusinkiewicz, 1986b). L-systems that use angles other than



90° ($\pi/2$ radians) still produce music when run by the program, but since the intended algorithm uses horizontal and vertical lines, it is questionable whether or not the resulting melodies still represent the original fractal. For example, the Sierpinski Gasket (displayed to the left) still produces melodies even though it uses an angle of 60° ($\pi/3$ radians) (Ahamed, 2016). For most fractals, though, the program works as intended and outputs the proper sound file relative to that fractal pattern.

However, there are several aspects of the program that could be improved upon in the future. For starters, the program only allows for melodies in the key of C and for the first seven notes of the scale to be played. These are both very obvious limitations on the accuracy of the melodies in relation to the fractals and would be good areas of focus in future development because of that. Although adding more options for keys could be implemented relatively easily (by creating additional pitch mapping dictionaries, i.e. rather than only having an option for $0 \Rightarrow$ C, $1 \Rightarrow$ D, $2 \Rightarrow$ E, there could be an additional mapping for $0 \Rightarrow$ A, $1 \Rightarrow$ B, $2 \Rightarrow$ C#, etc.), the issue with the note range is harder to implement.

Since the fractal will have an unknown height due to the variable number of L-system iterations and vertical step counts, the range of possible notes is also unknown. Because of this, it becomes very hard to manage all of the available notes. One potential solution to this is to change the note representations in the Mathematics-of-Music program to use a range of numbers rather than a set list of characters (Jain, 2020). This would allow the program to generate an infinite range of notes as large as the fractal. Unfortunately, this solution proved to be too challenging as it became difficult to implement an accurate key for the melody to reside in.

Rather than sticking to a predefined key, the numbers represented every note and therefore also included all flats and sharps. This meant any melody created by this implementation would use a chromatic key containing every note possible. Because of this, I chose to use a modulus operator to keep the pitch range limited to one octave instead. This is definitely something that could be improved upon in the future.

Another area of the program that could be developed further is the creation of harmonies. I was only able to produce simple melodies in my implementation, and was not able to find a way to easily create harmonies or overlay multiple melodies from the same fractal pattern as Mason and Saffle (1994) had suggested. To remedy this, I set the program to produce two wav files: one that represented the original output, and one that was reversed. This way, the user could combine the two or listen to them individually if they desired. However, it would be much better if it were possible to overlay or combine multiple melodies into one sound file during the program's execution.

Lastly, I believe it would also be interesting to see an updated implementation so that a multitude of instruments could be used. My implementation only allows for basic drone noises to be used for the tone of each note, and it would be much more engaging if a variety of sounds could be played for each melody. Especially when incorporated with the idea of combining different melodies into one file, this would be a great aspect of the program to develop further.

In total, however, the program turned out fairly well and was able to produce a musical melody from a fractal based on L-system rules. Even if it is not exactly a "fractal" being used to create the melody, and instead uses an L-system representation of a fractal, I still believe it to be a fascinating, and relatively simple, method used to convert a fractal into musical sounds.

Bibliography

Ahamed, M. A. (2016). L-systems compiler and renderer in Python. Retrieved April 10, 2021, from https://github.com/Mizzlr/L-Systems-Compiler-And-Renderer

Chakravarty, M. (n.d.). *Hilbert space filling curves* [Computer Generation]. The Use of Context in Pattern Recognition. https://www.bic.mni.mcgill.ca/~mallar/CS-644B/hilbert.html

Gleick, J. (2008). *Chaos: Making a new science*. New York, New York: Penguin Books.

Devaney, R. L. (1995). *The sierpinski triangle* [Computer Generation]. https://math.bu.edu/DYSYS/chaos-game/node2.html

Jain, S. (2020). Mathematics of music. Retrieved April 10, 2021, from https://github.com/weeping-angel/Mathematics-of-Music

Mason, S., & Saffle, M. (1994). L-systems, melodies and musical structure. *Leonardo Music Journal*, 4, 31-38. doi:10.2307/1513178.

Mishra, J., & Mishra, S. N. (2007). *L-system fractals*. Amsterdam: Elsevier.

Poltorak, A. I. (2019). *Fractal tu b'shevat* [Computer Generation]. The Times of Israel. https://blogs.timesofisrael.com/fractal-tu-bshevat/

Prusinkiewicz, P. (1986a). Graphical applications of L-systems. *Proceedings of Graphics Interface '86 / Vision Interface '86*, 247-253. Retrieved April 10, 2021, from http://algorithmicbotany.org/papers/graphical.gi86.pdf

Prusinkiewicz, P. (1986b). Score generation with L-systems. *Proceedings of the 1986 International Computer Music Conference*, 455-457. Retrieved April 10, 2021, from https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.324.3009&rep=rep1&type=pdf

Shiffman, S. (2012). *L-system generation* [Illustration]. The Nature of Code.
https://natureofcode.com/book/chapter-8-fractals/