The Sound of Fractals: Applying Fractal Parameters to Generate Visual and Auditory Outputs

Claire Kang
CPSC 490: Senior Project
Yale University
Advised by Dr. Scott Petersen

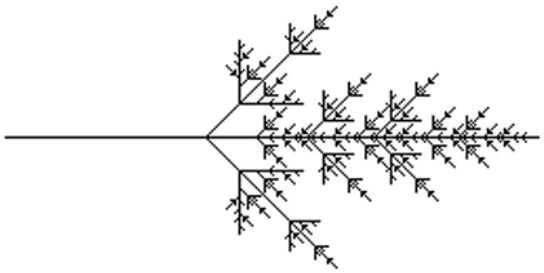

Figure 1. Example L-system generated fractal leaf

Abstract

Fractals have been a fascinating subject of study in mathematics and computer science for their intricate self-similar patterns that can be modeled using recursion. In the growing age of digital art, mathematical formulas have been used to generate fractal art. Interestingly, due to music theory's high degree of formalism, mathematically composing music has been equally prevalent, with its origins dating back to ancient Greece.[1] Fractals and their link to music have opened doors for immense creative possibilities, and this project delves into the multifaceted applications of fractal parameters to create a user-interactive web application, *The Sound of Fractals,* that generates fractal art and fractal music.

The key to fractal music is mapping, or defining the direct relationships between numerical outputs from mathematical algorithms to musical parameters. This paper specifically maps geometrical fractal parameters to parameters in a parallel generative grammar known as L-systems. L-systems have traditionally been used for graphical interpretation, but there have been studies that explore their potential for musical expression.[2] By exploring the intricacies of fractals, this paper aims to highlight the artistic possibilities that emerge when visual fractal parameters are harnessed as creative tools to produce music.

Keywords

fractals, L-systems, musical grammars, fractal music

1 Introduction

The earliest discussion of fractals dates back to the 17th century. Mathematician and philosopher Gottfried Leibniz studied recursive self-similarity, using the term "fractional exponents" to refer to scaling properties.[3] From his original studies of self-similar recursion, Leibniz considered straight lines to be the most fundamental self-similar figure.[4] Late 19th century, mathematician Karl Weierstrass discovered the Weierstrass function that is "continuous everywhere but differentiable nowhere."[5] His student, Georg Cantor went on to study subsets of self-similar lines which we know today as the Cantor set and is considered to be a fractal pattern.

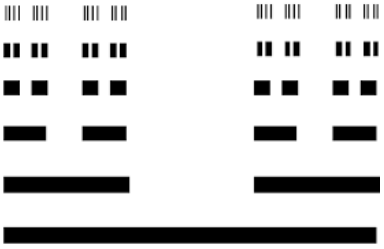

Figure 2. Cantor set

Early 20th century, Helge von Koch was dissatisfied with Weierstrass' definition of fractals and instead turned

[1] Fauvel, J., Flood, R., & Wilson, R. *Music and Mathematics: From Pythagoras to Fractals.* Oxford University Press, 2003.
[2] Worth, P., & Stepney, S. "Growing Music: Musical Interpretations of L-Systems." In EvoMUSART workshop, EuroGP 2005, Lausanne, Switzerland, edited by L. M. S. Rocha, 545-550. In LNCS 3449. Springer, 2005.
[3] Pickover, C. A. The Math Book: From Pythagoras to the 57th Dimension, 250 Milestones in the History of Mathematics. Sterling Publishing Company, Inc., 2009.
[4] Ibid.
[5] Vesneske, S. Continuous, nowhere differentiable functions. Whitman College Press, 2019..

towards a geometrical definition.[6] He started with an equilateral triangle that added identical repeating patterns with each iteration. His hand-drawn figure is now known as the Koch snowflake.

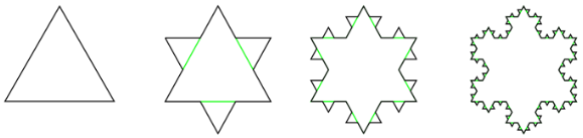

Figure 3. Koch snowflake of 3 iterations

A decade later, Waclaw Sierpinski described what is known as the Sierpinski Triangle. The pattern had been observed in stonework centuries ago, but it was Sierpinski who mathematically generated the pattern by repeatedly removing subsets of an equilateral triangle, also known as a finite subdivision. In his original article, he went on to show an example of a self-similar curve that uses two substituting production rules: (A → B-A-B) and (B → A+B+A).[7] The +/- indicated rotating 60 degrees in either the left or right direction. The result of the iterated curve is known as the Sierpinski arrowhead curve.[8]

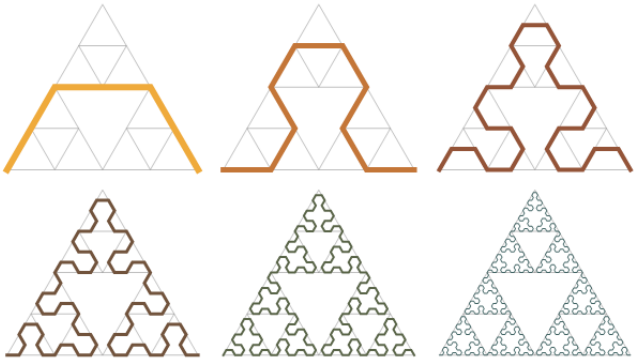

Figure 4. Six iterations of the Sierpinski arrowhead curve

Later, Gaston Julia and Pierre Fatou began relating their work to focus on patterns and deterministic laws of dynamic systems that are dependent on initial conditions: Chaos theory.[9] This is commonly known as the butterfly effect, named after one of Edward Lorenz's chaotic system plots that looks like a butterfly in his paper "Predictability: Does the Flap of a Butterfly's Wings in Brazil Set Off a Tornado in Texas?"[10] Counterintuitive to its name, chaos systems are far from random. Complex patterns create what is called a strange attractor that attracts the solutions to a range. When these attractors are visualized, they are often referred to as fractals.[11]

In this paper, sections 2 through 4 will discuss the related geometric information on the visual rendering portion of the project. Sections 5 through 7 will then discuss the musical components of the project.

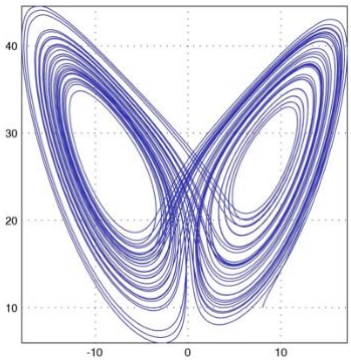

Figure 5. Plot of Lorenz attractor "butterfly"

2 Modern Definition of Fractals

Modern interpretations of fractals center around geometric patterns with self-similarity, bounded by a rule iterated by a desired number. Benoit B. Mandelbrot was the mathematician who coined the term "fractal" in 1977.[12] His work specifically focused on fractal geometry that can be observed in nature. There is an abundance of different types of fractals, and one such example that comes from nature is known as a fractal tree. Fractal trees are generated by splitting a line segment into two smaller segments and repeating this split a desired number of iterations.

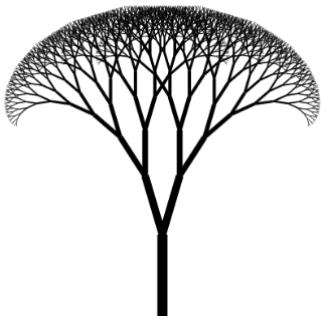

Figure 6. Fractal canopy with angle 2π/11 and segment ratio 3/4

A fractal tree is defined by a set of properties.[13] First, is that the angle between any two neighboring segments must be constant. Second, the ratio of the length of consecutive segments must be constant. As argued by Mandelbrot, the beauty of fractal trees, and fractals in general, is that they mimic the self-similar figures in nature.[14]

[6] Edgar, G. *Classics on Fractals.* Westview Press, 2004.
[7] Kaszanyitzky, A. "The generalized Sierpinski Arrowhead Curve." arXiv preprint arXiv:1710.08480, 2017.
[8] Ibid.
[9] Oestreicher, C. *A history of chaos theory.* National Library of Medicine, 2007.
[10] Lorenz, E. "Predictability: Does the flap of a butterfly's wing in Brazil set off a tornado in Texas?" 1972.
[11] Oestreicher, supra note 9.
[12] Mandelbrot, B. B. *The Fractal Geometry of Nature*. W. H. Freeman and Company, 1983.
[13] Thiriet, M. *Anatomy and Physiology of the Circulatory and Ventilatory Systems.* Springer Science & Business Media, 2013..
[14] Ibid. Supra note 12.

3 Visual Fractal Parameters

As mentioned before, there are an abundance of different types of fractals, and depending on the fractal there can be different properties that characterize them.[15] Looking ahead to fractal music generation, there are several musical parameters that must be considered in order to generate "pleasant" sounding music. To use the same fractal parameters for the visual rendering, as well as the music generation, I had to pick a type of fractal that can be defined by a multitude of properties. As a result, this paper will focus on symmetric snowflake fractals inspired by fractal trees.[16] The snowflake fractal consists of a set number of fractal trees, meaning at the end of each segment, it will split into two segments. The snowflake fractal can be defined by the following user-modifiable visual parameters:

1) **Sides**: The number of initial segments
2) **Branches**: The number of symmetric offshoots, or pairs of branches, *along* a given segment
3) **Depth**: The number of recursive iterations
4) **Spread**: The constant angle between any two neighboring segments

When considering the artistic characteristic of a fractal, the snowflake fractal will have a fifth parameter that can be modified by the user:

5) **Color**: The color of the fractal segments

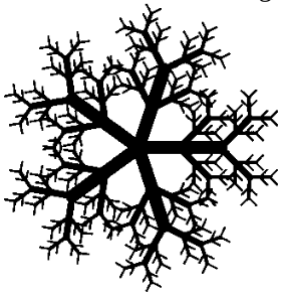

Figure 7. Snowflake fractal with 5 sides, 2 branches, 3 levels of depth, 0.8 rad spread, and black segments

Based on the definition of a fractal tree, scale – the ratio of the length of consecutive segments – is also a parameter in the snowflake fractal. However, to properly see the details of the user-generated fractal it was decided that a fixed ½ ratio was needed. Thus, it is not listed as part of the user-modifiable parameters.

4 Visual Rendering Methods

The Sound of Fractals is a web application that allows users to customize fractal parameters outlined in section 3 to both visually and auditorily generate an output. The final mockup of the application had the goal of creating a simple application that focuses on the visual fractal. The nature of the fractal parameters also required that the application provide users with granular control.

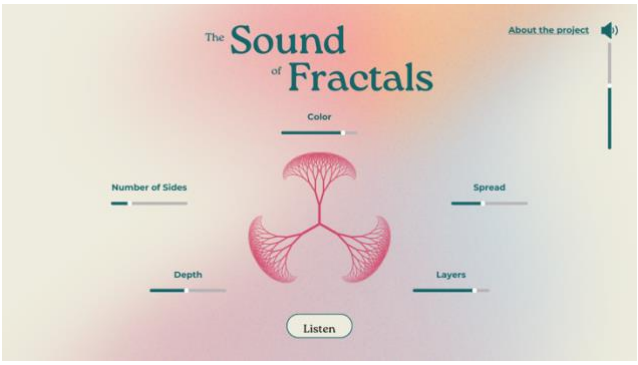

Figure 8. Final mockup of "The Sound of Fractals"

To visually render the fractal, I chose to use JavaScript and the HTML Canvas component. Drawing the fractal consists of drawing a *side*, which corresponds to one side of the fractal that recursively calls itself for *n = depth* number of iterations. Then this side is drawn *m = sides* number of times, where the initial segments are equally spaced around 2π radians to keep its symmetry. Sample code to generate the fractal is below:

```
function drawFractal() {
   ctx.clearRect(0, 0, canvas.width, canvas.height);
   ctx.save();
   ctx.strokeStyle = hslColor;
   ctx.translate(canvas.width/2, canvas.height/2);
   for(let i = 0; i < sides; i++) {
      ctx.rotate((Math.PI * 2)/sides);
      drawSide(layers);
   }
   ctx.restore();
}


function drawSide(layers) {
   if(layers == 0) return;
   ctx.beginPath();
   ctx.moveTo(0, 0);
   ctx.lineTo(size, 0);
   ctx.stroke();
   for(let i = 0; i < branches; i++) {
      ctx.save();
      ctx.translate(size - (size/branches) * i, 0);
      ctx.scale(scale, scale);

      ctx.save();
      ctx.rotate(spread);
      drawSide(layers - 1);
      ctx.restore();

      ctx.save();
      ctx.rotate(-spread);
      drawSide(layers - 1);
      ctx.restore();

      ctx.restore();
```

[15] By parameter I mean different inputs to a potential fractal tree generating algorithm that can visually modify a fractal tree. Fractal parameter used here is distinct from fractal measures, such as dimension, lacunarity, or succolarity.

[16] Snowflake fractal as used here and the rest of the paper will refer to the fractals defined by the properties mentioned in section 3. It is not referring to the Koch snowflake fractal.

```
    }
}
```

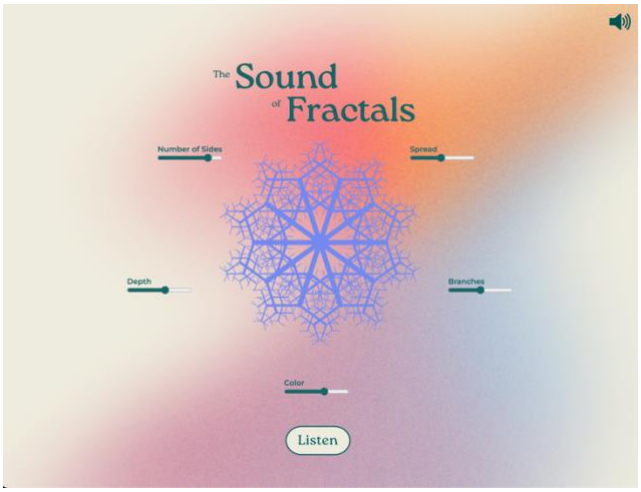

Figure 9. Final front-end implementation of "The Sound of Fractals"

5 L-Systems

L-systems are parallel generative grammars.[17] One starts from an axiom string and for several iterations, the defined grammar rules are applied in parallel. For example, consider the following L-system:

ω: X P_1: X → F[-F]X P_2: F → F+F

The string sequences for the first two iterations are:

0: X
1: F[-F]X
2: F+F[-F+F]F[-F]X

L-systems were originally defined to model plant development where each element can be interpreted as turtle graphics commands.[18] Peter Worth and Susan Stepney considered non-graphical applications of L-systems, specifically by considering music as grammar and reinterpreting the elements to represent pitch, duration, and timbre.
They defined the following elements:

F : Increase note duration by initial duration
+ : Move up one note in the scale of the chosen key
- : Move down one note in the scale of the chosen key
[: Push current state and set note duration to 0
] : Play note according to current state, then pop state

This musical rendering, also defined as a *Schenkerian rendering* produces a semi-musical melody where notes are played sequentially.
Worth and Stepney also defined *Stochastic L-systems*, where a variety of musical renderings can be produced in the same style. The same elemental interpretations were applied but with probabilities of rules being chosen from the same predecessor. For example, a stochastic L-system grammar could have the following rules:

$$F \rightarrow F+F$$
$$F \rightarrow F-F$$

Whether the predecessor 'F' gets expanded to 'F+F' or 'F-F' would depend on the probabilities assigned to them.

The Schenkerian rendering of the stochastic system in Worth and Stepneys' work produced three different pieces that sounded similar to each other. One could take these three different renderings and combine them to form a single piece of music.

While the stochastic systems helped to produce similar styles of music, they still sounded rather random. Context sensitivity in L-systems could be useful since a generated piece could create useful musical patterns like chord progressions based on neighboring notes. Worth and Stepney combine the idea of stochastic and context-sensitive L-systems to define a rule set that can add more musical nuance to the generated pieces. They also include a new element 'd' to be interpreted as halving the duration. Variations of the insertion rule that provide tonal information[19] and subdividing long notes by halving durations to form rhythm are as follows:

System A: Stochastic context-sensitive L-system[20]:

ω: F++F++F+++F---F--F—F

Identity: F $\rightarrow_{1/2}$ F
Repetition: F $\rightarrow_{1/26}$ [dFF]
Appoggiatura: F $\rightarrow_{1/26}$ [d-F+F]
Appoggiatura: F $\rightarrow_{1/26}$ [+F-F]
Neighbor note: FF $\rightarrow_{1/26}$ [Fd+F-F]
Neighbor note: FF $\rightarrow_{1/26}$ [Fd-F+F]
Skip1: F+F $\rightarrow_{1/26}$ [Fd++F-F]
Skip1: F+F $\rightarrow_{1/26}$ [Fd+++F--F]
Skip1: F+F $\rightarrow_{1/26}$ [Fd-F++F]
Skip1: F+F $\rightarrow_{1/26}$ [Fd--F+++F]
Skip2: F-F $\rightarrow_{1/26}$ [-Fd++F-F]
Skip2: F-F $\rightarrow_{1/26}$ [-Fd+++F--F]
Skip2: F-F $\rightarrow_{1/26}$ [-Fd-F++F]
Skip2: F-F $\rightarrow_{1/26}$ [-Fd--F+++F]

[17] Worth & Stepney, supra note 2.
[18] Prusinkiewicz, P., & Lindenmayer, A. *The Algorithmic Beauty of Plants.* Springer, 1990.
[19] Baroni, M., Dalmonte, R., & Jacobini, C. "Theory and Analysis of European Melody." In *Computer Representations and Models in Music*, edited by A. Marsdon & A. Pople, 187-206. Academic Press, 1992.
[20] Worth & Stepney, supra note 2.

This is context-sensitive because now the grammar rules are not just mapped to a single element, like 'F', but a series of elements, like 'F+F'. All rules have given probabilities defined in the arrow subscripts and thus, interpreting the rules considers all possible expansions, and a rule is chosen based on the probability. For example, if a given sequence is 'F+F', this could be expanded based on the Identity, Repetition, Appoggiatura, or the Skip1 rules.

The resulting pieces produced from system A sound more musically pleasing with interesting rhythms. The Repetition rule lengthens the current state's note by the initial duration. Appoggiatura rules create an appoggiatura, which are grace notes that are usually one step higher or lower than the main melody note. Neighbor note rules result in playing two neighbor notes that are one step away from each other. The skip rules result in playing notes that are more than one step away from each other.

The Sound of Fractals uses system A and the direct application will be explained in section 7.

6 Auditory Fractal Parameter Mapping

Aside from defining the axiom and ruleset for the L-system, many other mappings must be determined, such as the key of the scale, whether the scale is major or minor, and the initial duration of the note. Taking the visual fractal parameters outlined in section 3, the auditory fractal parameter mapping is as follows:

1) **Sides → Scale**: The number of sides map 1:1 to the scale key from A♭ to G
2) **Color → Major/Minor**: The HSL color values are mapped to major/minor
3) **Depth → Grammar expansion iterations**: The number of layers in the fractal determines how many iterations we want to expand the L-system sequence. Higher iterations result in more complicated pieces.
4) **Spread → Initial note duration**: The initial note durations are mapped to the spread values. Larger spread values correspond to slower notes.
5) **Branches → Axiom:** The number of branches determines which axiom we will use. Higher number of branches corresponds to longer axioms which result in longer complete pieces.

Other musical decisions made that are not determined by the fractal parameter are as follows:

1) Common time signature
2) Initial note is the 4th octave key (e.g. C4)

7 Audio Rendering Methods

To render the audio, I used Tone.js, a Web Audio framework.[21] Tone.js uses an audio context which is split into three parts: input, effects, and destination. Input refers to the audio source which could be created using a synthesizer or audio sampled from an existing file. This input is routed to effect nodes that can modify the audio source which is finally routed to the destination to be played, like the computer speakers. Tone.js comes with preloaded synthesizers that can be used to create and play notes. It also provides a Sampler instrument where audio files can be mapped to note pitches. *The Sound of Fractals* uses the Tonejs-Instruments external sample library to use a xylophone instrument.[22]

Audio rendering consists of three parts: mapping the fractal parameters, generating the L-system sequence, and processing the sequence through the audio context.

7. 1 Mapping the fractal parameters

Every time a user presses the "Listen" button a new Music object is created. The Music constructor passes in the fractal parameters and has the following properties:

1) **Iterations** – The number of L-system expansion iterations
2) **Initial Duration** – The initial duration of the note to be used in the sequence processing
3) **Key Name** – The scale root note in the 1st or 2nd octave (e.g. 'D2')
4) **Major** – 'Major' or 'Minor'
5) **Scale Notes** – An array of notes in the corresponding scale from the root note up to 4 octaves
6) **Axiom** – The axiom of the L-system
7) **L-System Sequence** – The final sequence after N number of iterations
8) **Note Queue** – An array of tuples that define the note name and note duration (e.g. { 'D3', '8n' }) that will be played in the final audio output
9) **Is Muted** – A conditional value that indicates whether the audio should be muted

How the fractal parameters then get used to define these properties can be seen through the sample code below.

```
constructor(sides, layers, spread, branches, color){
   this.iterations = layers + 1;
   this.initialDuration =
      setInitialDuration(spread);
   this.keyName = setScale(sides);
   this.major = color % 2 == 0 ? "major" : "minor";
   this.scaleNotes =
      makeScale(this.major, this.keyName);
   this.axiom = setAxiom(branches);
   this.lSystemSequence =
      lSystemGenerator(
         this.axiom,
         RULES,
         Number(this.iterations)
      );
```

[21] Tone.js. "Home." Tone.js. https://tonejs.github.io/.

[22] Brosowsky, N. P. "Tone.js Instruments." GitHub. https://github.com/nbrosowsky/tonejs-instruments.

```
this.noteQueue =
   interpretLSystem(
      this.lSystemSequence,
      this.scaleNotes[15],
      this.initialDuration,
      this.scaleNotes
   );
this.isMuted = false;
```

In order to produce a melody with at least 15 notes, it requires that there be at least 2 iterations of L-system expansion. The fewest number of layers a fractal can have is 1. Thus, the number of iterations was defined to be the number of layers + 1.[23]

7. 2 Generating the L-system sequence

As mentioned before, *The Sound of Fractals* uses system A outlined in section 6. The rules are defined as a map with the rule successor as keys and the predecessor as the values. For example, consider the following rule:

$$FF \rightarrow_{1/26} [Fd+F-F]$$

It is stored in the map as ['[Fd+F-F]', 'FF'] with '[Fd+F-F]' being the key and 'FF' being the value.

There are three possible axioms. Starting with the axiom, a sequence with the defined rules are passed into an applyRules function. This function gets called a specified number of times where the resulting sequence from the previous iteration gets passed in the subsequent calls.

The applyRules function has an initially empty "results" string. It starts with the first element and for each element in the sequence it creates a "potentialRule" empty array. For each rule defined in the rules map we check to see if the rule predecessor matches either the current element, the current element and one element after it, or the current element and two elements after it. For all matches, we push the successor into the potentialRules array. Once it has iterated through all the rules, if potentialRules is *not* empty, we randomly select one of the successors, add the successor to the result string, and update the for-loop iterator index based on the length of predecessor of the successor selected. The length of the predecessor is calculated by calling get on the rules map with the successor. This means if the successor selected had the predecessor 'F+F', we skip the '+F' elements and move to the element after them. If the potentialRules array is empty, we add the current element to the result string and move on to the next element.

After applyRules gets called the proper number of times, the resulting final L-system sequence is stored as this.lSystemSequence in the Music object.

7. 3 Processing the final sequence

With the final L-system sequence, we must now interpret the elements based on the element definitions outlined in section 5. One modification to the interpretation was made for '['. Instead of pushing the current state and setting note duration to 0, we instead set the duration to the initial duration. This change was made because setting the duration to 0 resulted in null notes for higher number of iterations.[24]

Thus, with that modification the final element interpretations are:

F : Increase note duration by initial duration
d : Halve the note duration
+ : Move up one note in the scale of the chosen key
- : Move down one note in the scale of the chosen key
[: Push current state, set note duration to initial duration
] : Play note according to current state, then pop state

The sequence interpretation is done through the interpretLsystem function that passes in the final sequence string, the starting note, the initial note duration, and the array of notes in the given scale as parameters. The starting note is the base note in the scale at the 4th octave e.g. 'G4'. This function acts as a state machine that keeps track of the current stack and the resulting note queue. The code for the state machine details what each state does:

```
for (const symbol of lSystemSequence) {
   switch (symbol) {
      case 'F':
         currentDuration =
            addDurations(
               currentDuration,
               initialDuration);
         break;
      case 'd':
         currentDuration =
            halveDuration(currentDuration);
      case '+':
         currentNote =
            moveNote(currentNote, 1, scale);
          break;
      case '-':
         currentNote =
            moveNote(currentNote, -1, scale);
         break;
      case '[':
         stack.push(
            { note: currentNote,
            duration: currentDuration }
         );
         currentDuration = 0;
         break;
      case ']':
         const prevState = stack.pop();
         if (prevState) {
            result.push(
               {  note: prevState.note,
                  duration: prevState.duration
               });
            currentNote = prevState.note;
            currentDuration = prevState.duration;
         }
         break;
      default:
         break;
   }
}
```

The resulting note queue from this function is used to create a Tone.Part. Part allows for a set of notes to be

[23] Layers refers to the depth value of the fractal

[24] This is due to Tone.js's note duration handling. It does not process a duration value of 0.

looped and started/stopped as a single unit.[25] We want the resulting melody to loop itself, thus the entire note queue is mapped under Tone.Part. Instead of using the built in synth instrument, I used the Tonejs-Instruments external sample library to have the notes be played with a xylophone. After the xylophone samples are loaded, it is routed to the destination which is the system speaker. Then the triggerAttackAndRelease function, which takes in the note name and note duration value, is called on the xylophone and the notes mapped in Tone.Part are played.

8 Further Works and Conclusion

The visual rendering of fractals was successful in providing users with granular control to produce a wide range of fractals. It would be interesting to provide other controls, such as changing the scale of the lines with each layer of depth or producing other types of fractals other than the fractal snowflakes.

The musical L-system's sequential rendering is a relatively naïve approach but produces pleasant results. The current algorithm produces repetitive, uninteresting melodies at higher iterations. While the context-sensitive L-system provides potential for creating longer pieces that incorporate musical progression, more work on rendering pieces from a music theory standpoint would be valuable for producing even more musically sounding pieces. For example, the current implementation uses the same key with similar timbre and rhythm throughout the entire piece. Incorporating chord progressions and mimicking a sonata form (exposition, development, and recapitulation), would help the resulting pieces sound more pleasing. In addition, the current output also sequentially plays single notes. It would be interesting to incorporate chords and use them to add chord progressions to the music.

In terms of the user experience for the web page, users may be interested in knowing the exact parameter values for their fractal, and how these values are impacting the music that is being generated. Displaying this information somewhere on the website would be useful in case users want to recreate a fractal. Moreover, due to the stochastic L-systems, the same fractal parameters can produce different pieces. There is no way of "saving" the pieces that are created from a fractal. Even if the user were to put the same fractal parameters, it would most likely output a different piece. Providing a way to save the musical output, either by generating a MusicXML score for pieces that the user wants to save, or implementing database storage so users can save the note queues that can be played back whenever they wish to, would be valuable additions to the web application.

Overall, *The Sound of Fractals* successfully produced both visual and auditorial outputs. The final music output went through several iterations, but it was a great experience getting to work with an audio library for the first time. *The Sound of Fractals* can be visited at https://sound-of-fractals.pages.dev/

[25] Tone.js. "Part." Retrieved from https://tonejs.github.io/docs/14.7.77/Part.html

9 Acknowledgements

I would like to give many thanks to my advisor, Dr. Scott Petersen, for encouraging me and helping me navigate through the new world of web music creation. I also want to thank Noa Choi, who continuously tested, provided helpful feedback, and supported me throughout the entire process. I send big thank-yous to my friends for supporting me through my journey as a CS major and growing with me at Yale in general.

Finally, I would like to thank my family for supporting my education and being my rock. Without their constant support and encouragement, I would not be where I am today.

Bibliography

Baroni, M., Dalmonte, R., & Jacobini, C. "Theory and Analysis of European Melody." In Computer Representations and Models in Music, edited by A. Marsdon & A. Pople, 187-206. Academic Press, 1992.

Brosowsky, N. P. "Tone.js Instruments." GitHub. https://github.com/nbrosowsky/tonejs-instruments.

Edgar, G. Classics on Fractals. Westview Press, 2004.

Fauvel, J., Flood, R., & Wilson, R. Music and Mathematics: From Pythagoras to Fractals. Oxford University Press, 2003.

Kaszanyitzky, A. "The generalized Sierpinski Arrowhead Curve." arXiv preprint arXiv:1710.08480, 2017.

Lorenz, E. "Predictability: Does the flap of a butterfly's wing in Brazil set off a tornado in Texas?" 1972.

Mandelbrot, B. B. The Fractal Geometry of Nature. W. H. Freeman and Company, 1983.

Oestreicher, C. A history of chaos theory. National Library of Medicine, 2007.

Pickover, C. A. The Math Book: From Pythagoras to the 57th Dimension, 250 Milestones in the History of Mathematics. Sterling Publishing Company, Inc., 2009.

Prusinkiewicz, P., & Lindenmayer, A. The Algorithmic Beauty of Plants. Springer, 1990.

Thiriet, M. Anatomy and Physiology of the Circulatory and Ventilatory Systems. Springer Science & Business Media, 2013.

Tone.js. "Home." Tone.js. https://tonejs.github.io/.

Tone.js. "Part." Retrieved from https://tonejs.github.io/docs/14.7.77/Part.html.

Vesneske, S. Continuous, nowhere differentiable functions. Whitman College Press, 2019.

Worth, P., & Stepney, S. "Growing Music: Musical Interpretations of L-Systems." In EvoMUSART workshop, EuroGP 2005, Lausanne, Switzerland, edited by L. M. S. Rocha, 545-550. In LNCS 3449. Springer, 2005.