IS71021C Mathematics and Graphics for Computer Games 1 | Assignment 2 of 2

Jamie Corr[1] [SN: 3372 8543] and Elliot Walker[2] [SN: 3368 6408]

[1] Goldsmiths, University of London. 8 Lewisham Way, London, SE14 6NW, United Kingdom
`jcorr005@gold.ac.uk`

[2] Goldsmiths, University of London. 8 Lewisham Way, London, SE14 6NW, United Kingdom
`ewalk008@gold.ac.uk`

J. Corr:	Programming and Design.
E. Walker:	Design and Audio Composition.

Abstract. This report provides an overview of experimental and descriptive research conducted in the field of music visualization; specifically, how the Turtle Graphics of Lindenmayer Systems may be manipulated with parameters in digital audio, and how they overlap to expand upon their use cases. Included is a review of L-Systems and digital audio (particularly the latter's association with Discrete Fourier Transforms).

Keywords: Graphics • Lindenmayer • Music • Procedural • Visualization.

Demo 1:
https://www.youtube.com/watch?v=VGTevKkUtA&ab_channel=JamieCorr

Demo 2:
https://www.youtube.com/watch?v=mlyeGSEw_cU&ab_channel=JamieCorr

Project .ZIP:
https://1drv.ms/u/s!AuqdY7Pu7csHny4WB1TwOTZ4B2aN?e=DkN7IA

Final Build .ZIP:
https://drive.google.com/drive/folders/12TQkPZfkMdvIbVoalUXze-OcUBSyWipH?usp=sharing

1 Background

1.1 Music Visualization

The field of **music visualization** is a technical art comprising of studies regarding the interface between audio and its graphical interpretations. Mathematical aspects include acoustics, particularly its digitization with Discrete Fourier Transforms (DFTs). Music visualization dates to approximately 1963 when British inventor Frederick Judd created an apparatus which generated abstract color patterns in response to sound waves from musical instruments.

Music visualization may be applied either recreationally or practically. Recreational applications may include simulating or expanding upon light shows at musical concerts using digital mediums such as conventional monitors or virtual reality headsets.

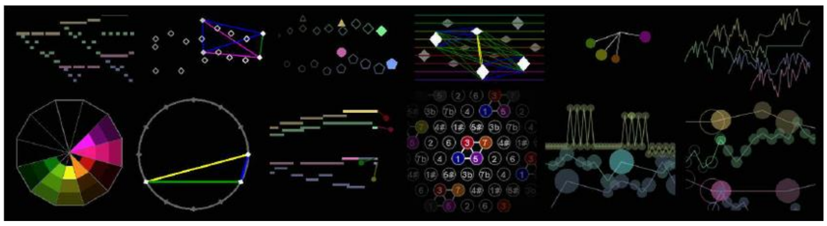

Fig. 1.1. Music visualization as seen on the Music Animation Machine [19][20].Visualizations include a piano roll (top-left), partial motions (right), and tonal compasses (bottom-left). Image by W.Y. Chan [21].

Practical applications included enabling musicians who are either deaf or hard of hearing to see what they are playing. Such applications have been demonstrated; Dr Richard Burns of Birmingham City University (BCU) developed a software as part of his PhD which visualizes input from a MIDI keyboard piano across two viewports – one displaying which key is pressed, and one displaying the waveform of the audio played. Collectively, these visuals stimulate the senses of deaf or hard of hearing individuals to enhance their bond with music [7][8].

Another example is the Music Animation Machine (MAM) developed by Stephen Malinowski [19][20], comprising of three unique displays demonstrating how aspects of digital audio may be mapped to graphical entities:

- **Piano Roll**
  Represents each note of music with a color and height depending on the pitch.
- **Partial Motion**
  Partial Motion represents notes by connecting circles of varying height and radii, the screen scrolls over the path of circles generated.
- **Tonal Compass**
  Displays notes as circles of different radii, these notes are arranged around a circle of fifths depending on the pitch of their tone. Ideal for visualizing harmony and chords within music.

1.2 Digital Audio and Unity

Digital audio is interpreted as a linear stream of binary data, where each byte represents a particular value for the amplitude of the audio. The process of converting analogue audio into digital format is **quantization**. The quality at which the audio is quantized is determined by two parameters:

- **Sampling rate** – frequency at which samples are taken from an audio source (~44kHz for Music, ~11kHz for Voice)

- **Sample size** – the number of bits per sample (8-bit sounds with 256 possible values which ate good for **synthetic music**, 16-bit sounds with 65,536 possible values which are good for **natural music**).

Audio samples may be considered as packages containing binary data representing the digital audio (a computer's interpretation of audio). Audio played at a sampling rate of 15kHz offers 15,000 of these "packages" per second, where the following relationship is implied:

$$Audio\ Frequency\ (Hz) \equiv Samples\ per\ second$$

Digital audio may be mapped to an audio frequency spectrum consisting of seven bands.

Table. 1.1 Tabulated overview of the audio frequency spectrum.

Band	Range (Hz)	Offset (Hz)	Size
Sub-bass	0 to 60	60	0.3%
Bass	60 to 250	190	0.95%
Low Midrange	250 to 500	250	1.25%
Midrange	500 to 2k	1.5k	7.5%
High Midrange	2k to 4k	2k	10%
Presence	4k to 6k		
Brilliance	6k to 20k	14k	70%

Frequencies in the 0kHz to ~20kHz range are audible to humans, and that each sample constituting this audio spectrum is offset from its neighbor by a constant value given by the following formula:

$$offset = \frac{20kHz}{n}$$

where n is the number of samples, equal to a power of 2.

For a spectrum of 512 samples, the following offset k is given by:

$$\boldsymbol{k} = \frac{20kHz}{512} = \frac{20kHz}{2^9} = \boldsymbol{40Hz}$$

and the location (frequency) at which a sample lies on the audio spectrum is equal to the product of this offset and the sample's index (e.g., the 100th sample in this collection of 512 would lie at $ik = (100)(40Hz) = \boldsymbol{4kHz}$, which is in the high midrange to presence bands).

From a mathematical perspective, digital audio is derived when a **continuous-time signal (CTS)** or analogue sound wave is rendered using a **Discrete Fourier Transform (DFT)** – specifically the **Fast Fourier Transform (FFT)** algorithm – into a **discrete-time signal (DTS)**, or digital sounds wave. Any signal (including soundwaves) may be expressed as a wavefunction whose independent variable is either space or time (usually the latter in the context of soundwaves): the DFT converts this into a frequency domain $\{X_k\}$.

$$\mathcal{F}(x) = \sum_{n=0}^{n=N-1} x_n e^{-\frac{i2\pi}{N}kn}$$

where n and k are indexes for the sets $\{x_n\}$ and $\{X_k\}$, respectively, and N is the number of values in either set.

$$\mathcal{F}(x) : \{x_n\} \rightarrow \{X_k\}$$

The Unity engine offers developers the ability to query data from **AudioSource** components, which are assigned audio clips to play during application runtime. Each AudioSource component comprises of audio spectrum data which can be queried and assigned to a sample buffer.

```
AudioSource.GetSpectrumData(float[], int, FFTWindow);
```

Parameter 1 is an array of floats representing the samples retrieved from the AudioSource component ; parameter 2 is an integer representing the channel that is consulted for audio data ; and parameter 3 is the FFT (Fast Fourier Transform) window (interval) applied to mitigate leakages between frequency bins and bands. There are various types of FFTWindow which may be applied. Algorithmic complexity increases down the list [15]:

Table. 1.2 Summary of the different FFTWindow types, including associated expressions.

Window Type	Expression
Rectangular	$W[n] = 1$
Triangle[1]	$W[n] = TRI(2n/N)$
Hamming	$W[n] = 0.54 - (0.46 * \cos(n/N))$
Hanning	$W[n] = 0.5 * (1 - \cos(n/N))$
Blackman	$W[n] = 0.42 - \left(0.5 * \cos\left(\frac{n}{N}\right)\right) + (0.08 * \cos(2n/N))$
BlackmanHarris	$W[n] = 0.35875 - \left(0.48829 * \cos\left(\frac{n}{N}\right)\right) + \left(0.14128 * \cos\left(\frac{2n}{N}\right)\right) - (0.01168 * \cos(3n/N))$

[1] Also known as the Bartlett Window, or the Triangle function.

1.3 Lindenmeyer Systems

A Lindenmeyer System (L-System) is any procedurally generated sequence which arises from a specific collection of symbols (grammars) which incur either deterministic, stochastic, or context-sensitive rulesets that alter the sequence. They may be interpreted graphically using Turtle Graphics, which comprise of rudimentary operations such as drawing a unit line, generating a node, or rotating it by a fixed amount. Examples of L-Systems include the Sierpinski Triangle, Koch Curve, Koch Snowflake, and many more including those which produce patterns similar or identical to those seen in plants or trees: often these are applied in **procedural graphics**.

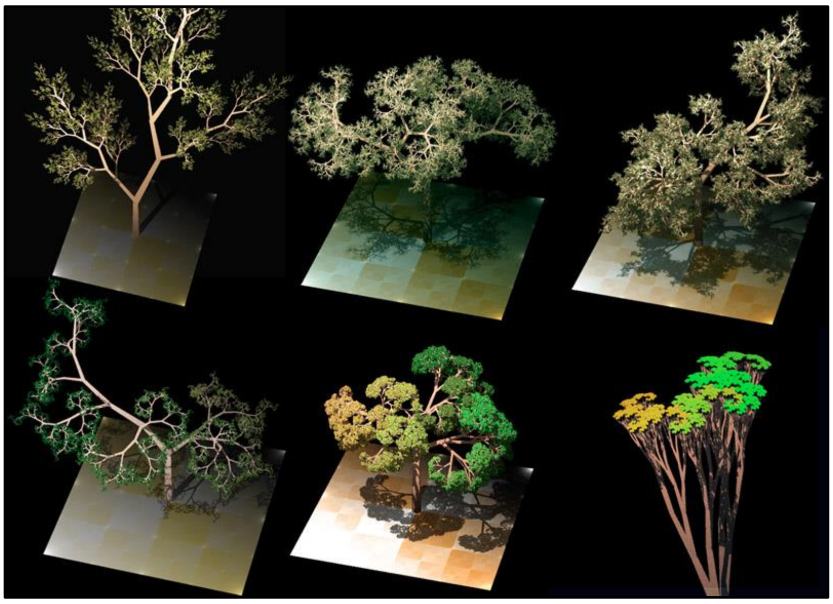

Fig. 1.2. 3D graphical interpretations of plant-like L-Systems, each one comprising of branches, leaves, and nodes.

Table. 1.3 Deterministic plant L-Systems, symbols, and rules. Symbols which do not incur changes are constants, whereas those that do are variables.

System Name	**Symbols**	**Axiom**	**Rules**
Plant A	F, +, -, [,]	F	$F \rightarrow F[+F]F[-F]F$
Plant B			$F \rightarrow F[+F]F[-F][F]$
Plant C			$F \rightarrow FF + [-F + F + F] - [+F - F - F]$
Plant D	F, X, +, -, [,]	X	$F \rightarrow FF$ $X \rightarrow F[+X]F[-X] + X$
Plant E			$F \rightarrow FF$ $X \rightarrow F[+X][-X]FX$
Plant F			$F \rightarrow FF$ $X \rightarrow F - \left[[X] + X\right] + F[+FX] - X$

2 Methodology

2.1 Research Proposition

The aim of our project is to expand upon the possible use cases for music visualization discussed in §1.1 using L-Systems and their Turtle Graphics as components. Our project is conducted under the experimental paradigm with a focus on qualitative data and involves two deliverables:

- An investigation of means by which Turtle Graphics for L-Systems may be affected by audio across different frequency bands. Specifically, how sample intensity and musical tempo can be mapped to these graphics.

- A complimentary and demonstrative software created with the **Unity3D** game engine that implements music visualization with these L-Systems.

2.2 Applied L-System Grammars

The L-System grammar which we have applied in our project is that which produces a plant-like structure.

Table. 2.1a Tabulated overview of the plant L-System used in our application.

Symbols	Axiom	Rule
F, +, -, [,]	F	$F \rightarrow F[+F]F[-F]F$
		$F \rightarrow F[+F]F[-F][F]$
		$FF + [+F - F - F] - [-F + F + F]$
		$F[+F + FF]FF[-F - FF]$[2]

Table. 2.1b Tabulated overview of the Turtle Graphics instructions which accompany the plant L-Systems outlined in table 2.1a.

Symbol	Turtle Graphics
F	Generate a branch (unit line).
+	Add a rotation value of 25° to the current branch's orientation.
-	Subtract a rotation value of 25° to the current branch's orientation.
[	Add (push) the current branch transform to the stack.
]	Remove (pop) the current branch transform to the stack.

[2] This is a custom-made L-System grammar composed by J.Corr.

2.3 Visualizing Music

Audio spectrum data may be mapped to game object components on a per-sample basis, where each sample's magnitude (volume level) may different parameters attached to these components. Such parameters may fall be physical such as position, scale (e.g., height, radius), and rotation, or graphical such as texture and colour.

Design for the project software was partially inspired by a series of programming tutorials [9-12] which explained how audio sample data may be mapped to graphics in Unity. Further aided with the official Unity documentation covering API features which enable audio sample data to be queried programmatically [13, 14].

In the project application, each sample (of which collections comprising of 64, 128, and 512 were tested) corresponds to a single L-System instance and a frequency on the audio spectrum. Each instance of the L-System grammar is preconstructed, such that its post-iterative state is applied, although only portions of its Turtle Graphics are rendered based on each sample's value during application runtime. The application's functionality is distributed across seven C-sharp scripts. Below is a tabulated summary of what each script achieves:

Table. 2.2 Tabulated overview of our application's scripts, and their respective functionality.

Script Name (.cs)	Functionality
DeactivateScript	Destroys the game object that wields this script.
	DeactivateTimer Invoked as a coroutine which waits for a period of time determined by a timer variable (DeactivationTime). Per iteration, the method increments a value "T" corresponding to the elapsed time; once $T = 1$, the connected game object is deleted.
GenerateXISystems	Attached to the spawning node for each L-System, and is responsible for L-System presentation, placement, and audio scaling.
	InputLSystemAudioValues Maps the audio bands to different components of each L-System's turtle (and glowing) graphics. The audio value (volume) corresponds to sub-bass and bass bands, the lower midrange band is mapped to branch length, the midrange and upper midrange bands are mapped to the branch angles across the xz-axis, and the presence band is mapped to tree colour.

	IterationTimer A coroutine that scales audio output by a factor of 100 for each iteration in the specified L-System.
	InitiateIterationTimer Checks if the iteration timer has been successfully found, then starts it via a coroutine.
	ScaleAudioOutput Returns a scaled value of a provided audio output volume. Applied to set specific intensities.
	Rotate Rotates the spawning node about its y-axis to project Turtle Graphics in three dimensions.
	SetupShape Maps the specified layout shape for the L-System collection to a corresponding setup instructions.
	SetupCircle Organizes the L-Systems in a circle.
	SetupSquare Organizes the L-Systems in a square.
	SetupSpiral Organizes the L-Systems in a spiral.
	ResetSystem Refreshes the setup of L-Systems during runtime.
	ResetLSystemsTimer A coroutine which maps iteration timer values to reset instructions. Once time $t = 1$, the L-System instance is refreshed.
	BasicTransition Collects game objects in the scene tagged with "LineHolder" or "LSystem" and removes them, subsequently invoking the "ResetLSystemsTimer" coroutine.
	SlowTransition Invokes a gradual transition between L-System graphical configurations.
	SlowTransitionTimer Wipes a leaf from the scene every 10 milliseconds. Once all the leaves are destroyed, the "BasicTransition" method is invoked.

	ChooseRandomPresets Applies RNG to select a random graphical configuration for a given L-System.
	ChooseRiggedPresets ***<depreciated>*** Sets the L-System configuration to a circle.
	ReturnRandomBoolean Applies RNG to generate a random boolean value (50:50 probability between true and false).
IterationNumber	Responsible for mapping audio to toggling constituent game objects for each L-System instance.
	VaryIterationNumber Sets particular branches and leaves to be active based on its sample's value.
LeafColor	Ensures the colours of the leaves spawned at the tips of each L-System branch are LERPed (linearly interpolated) over a fixed period.
	ColorTimer A coroutine method which generates a randomly-selected colours for each L-system leaf. Invoked once a leaf instance is instantiated.
	SmoothColorChange Invoked every tick in the script's "Update" method to LERP each leaf's colour, but only if the "bSmoothColorChange" flag is set to *true*.
	ChangeLeafColor Invoked as an alternative to "SmoothColorChange". Leaf colours are changed immediately rather than gradually.
LSystemAudioGenerator	Initializes the AudioListener and AudioSource (with audio clip) components which play the sounds specified in the attached AudioClip.
	CalculateAudioRanges Determines the audio frequency ranges (represented as floats) for each applied frequency band. Sub-bass and bass bands are grouped into a single band for convenience.
	Implements a single L-System grammar based upon an axiom and specified rulesets, in addition to a set of corresponding transform operations for Turtle Graphics.

LSystemTest	**LSystemRoutine**
	Invoked as a coroutine. Iterates through the current L-System state and applies Turtle Graphics (spawning branches, leaves, or rotating nodes) to each symbol encountered. Upon reaching the end of the state, a sibling method "Generate" is invoked to produce a new state.
	
	When generating the turtle graphics, each L-System takes into account audio values. The branch length is dictated by the 'LowMid' audio range, angle 1 by the 'Mid' audio range, angle 2 by the 'UpMid' range and finally the branch color by the presence range.
	
	// need to map branch width to bass
	//need to map leaf scale to brilliance

	Generate Iterates through the symbols constituting the current state of the L-System and mutates them via corresponding rules.
TransformInfo	A class which comprises of two public member variables: a Vector3 for an object's position, and a Quaternion for its rotation. Instances of this class are stored in a stack which tracks an L-System's growth at each iteration.

3 Reflection

3.1 End Product

The following images illustrate the project software and the visuals it produces:

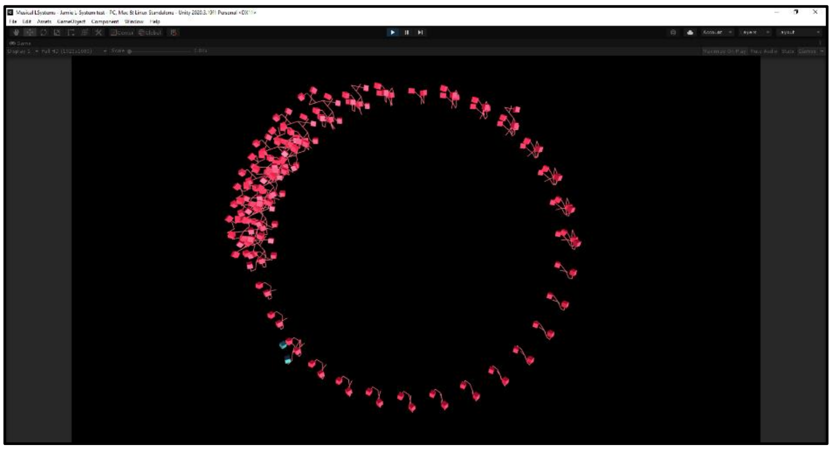

Fig. 2a. Example 1 of 4, a circular configuration.

When first starting our audio visualization project, we set up 64 L-Systems arranged in a circle and split the music into 64 audio channels (each of a different frequency range) across its circumference. Each L-System corresponded to an audio channel and the branch’s length, where angle and color were determined by the volume of the audio channel. This worked quite well and notable changes in the modal audio channels were observed, despite the visuals being limited beyond this.

We next created the LeafColor script, allowing us to change the appearance of the leaves throughout the song. The smooth color change made our L-Systems relaxing and visually engaging whilst the sudden color changes gave a clear indication for the beat and tempo of the music. The most important addition to our L-Systems was the varying iteration function, by storing each L-System's branches/ leaves in empty game objects called LineHolders, we were able to dynamically activate and deactivate the number of branches/ leaves visible. We made the iteration number dependent on the volume of each audio channel; this transformed our visuals – we were now able to clearly see which frequencies were loudest at any given point in the music by seeing each L-System rise and fall to the music. It also allows the user to track the rhythms in each instrument, giving a visual texture to the music.

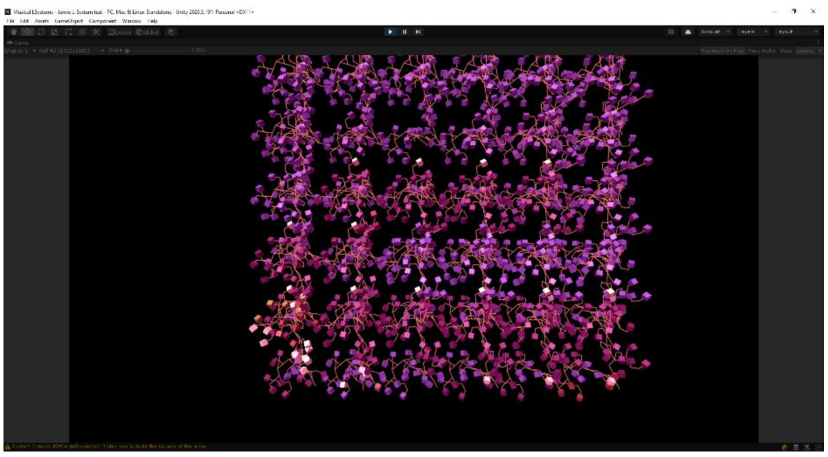

Fig. 2b. Example 2 of 4, a square configuration.

We also experimented with different rates of L-System growth, one smooth (generating 1 new leaf/branch every 50 milliseconds) and the other sudden (generating 10 new leaves/branches every beat, roughly every 400 milliseconds). The smooth growth gave a relaxing feel to the effects, although did not allow the user to clearly see the beat of the song. Other properties such as sudden color change and varying iteration numbers gave a beat indication, so we often paired the smooth growth with these properties. The sudden growth was excellent for showing the tempo and beat of the song although detracted from the visuals of other effects (such as varied iterations).

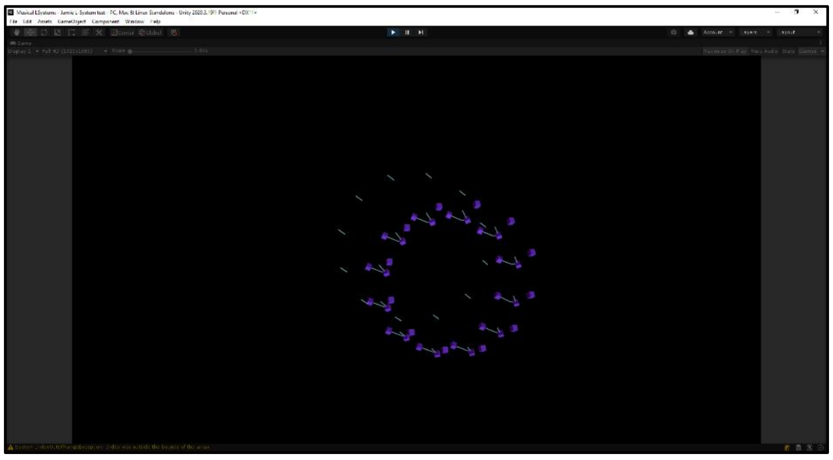

Fig. 2c. Example 3 of 4, a shifting circular configuration.

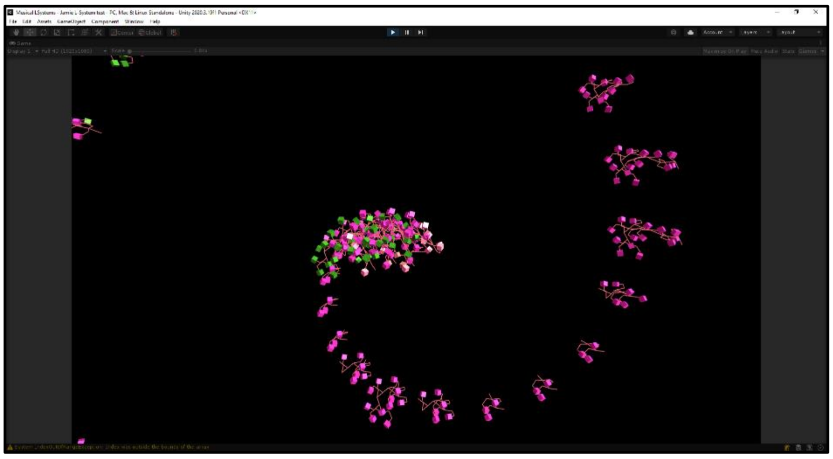
Fig. 2d. Example 4 of 4, a spiral configuration.

When testing our project, the frame rate would drop significantly as the song progressed. Starting off with frame rates around 60Hz, as the L-Systems became more complex and bloated, the rate would drop 50% to approximately 30Hz. To combat this, we introduced new properties for our L-Systems; deactivation time and iteration cap. The deactivation time destroyed the leaves and branches x seconds after generation, this created a new effect, as if each L-System was momentarily dancing across the screen. Pairing the deactivation time with the sudden growth made the L-Systems jaggedly dance to the beat. Iteration cap limits the number of branches and leaves that can be generated, this was good as towards the end of songs, the screen was often cluttered and overpopulated, making it hard to see the effects of the music.

Unity's **GetSpectrumData** method splits the music's audio into N channels, where N must be a power of 2 and at least 64 ($N \in 2^n, where\ n\ \geq 6$) – this implies we were restricted to using at least 64 L-Systems: a memory-intensive demand. Furthermore, some channels measuring the frequency in the brilliance range often had little variation, leading to Systems with minimal visual effects. To solve this problem, we created the **CalculateAudioRanges** method, which creates a new array and populates it with audio ranges. The audio ranges were calculated by taking the **GetSpectrumData** audio channels and splitting them into the six defined audio ranges (SubBass/Bass, Lower Middle, Middle, Upper Middle, Presence and Brilliance), each of these ranges were than further divided into x ranges. As the length of each audio range varies massively, we had to average varying numbers of channels. This worked well it allowed us to have as little as 6 L-Systems running with each System showcasing more impactful visuals. We also had to further introduce a scaling method as the raw output gave low frequency volumes.

This software is also demonstrated via videos uploaded to YouTube (see title page links).

3.2 Use Cases

Our research into music visualization with Unity lead to us contemplating the following use cases for music visualization and graphical representations of L-Systems:

- **Tempo-based Gameplay**
  Music visualizations which include manipulating game object transform and mesh geometry (like those seen in Turtle Graphics for L-Systems) may be used to create levels which respond to their in-game music. Gameplay may be influenced by this visualization to focus more on tempo and encourage players to adhere to the pace set by the music. For instance: platforms which the player must traverse that shrink and expand based on note pitch and volume.

- **Engaging Visuals for Impaired Listeners**
  The visual effects we created help highlight the tempo, volume, pitch and texture of the music, transferring an auditory experience to visual. This could help engage impaired listeners with musical properties that were previously inaccessible.

- **Alternative to Hallucinogens**
  Hallucinogens are a class of illicit drugs which cause the user to become disassociated with their surroundings and experience imaginary visuals which appear real, at the expense of their physiological health. The striking visuals which can be produced with music visualization (particularly those which are situated in immersive VR experiences) may provide healthier (and legal) alternatives to hallucinogens such as LSD or PCP.

- **Light Shows and Visuals for Concerts**
  The visuals seen in our project application may be applied in virtual experiences such as virtual concerts, warranting industrial applications for musical L-Systems. Similarly, such an application may act as an alternative to hallucinogenic experiences during musical festivities and similar venues.

4 Conclusion

We successfully explored the fundamental Mathematics and methods used in music visualization. When paired with L-Systems, we were able to create dynamic graphics which highlight the tempo, volume, pitch, and texture of digital music. Creating adjustable L-System properties gave us a wide range of effects to help make listening to music more engaging to the average user, and more accessible to impaired listeners.

In our exploration of these effects, we expanded upon the initial applications of music visualization to overlap with aspects of gameplay, therapeutic experiences, and accessibility. We also conclude that L-Systems (particularly those of a plant-like structure) offer aesthetically-pleasing parameters to music visualization.

End of Report

Bibliography

[1] D.W., Fourney, et al., “Creating Access to Music through Visualization,” Research Contribution, Centre for Learning Technologies, Ryerson University, Toronto, Canada, 2009.

[2] A., Rodrigues, et al., “Evolving L-Systems with Musical Notes,” Conference Paper, CISUC, Department of Informatics Engineering, University of Coimbra, Coimbra, Portugal, 2016.

[3] V.L., Nguyen., et al., “A Visualization Technique of a Music Emotion Represented in Dimensional Approach on Infinity Mirror Based LED Wall,” Research Contribution, Department of Media, Soongsil University, Soongsil, South Korea, 2017.

[4] A.S., Cowen, et al., “What music makes us feel: At least 13 dimensions organize subjective experiences associated with music across different cultures,” Research Contribution, Department of Psychology, University of California, Berkeley, CA; Department of Psychology, University of Amsterdam, Amsterdam, The Netherlands; Department of Psychology, York University, Toronto, Canada, 2019.

[5] S., Manousakis, “Musical L-Systems,” Master’s Thesis, Institute of Sonology, Royal Conservatoire, The Hauge, The Netherlands, 2006.

[6] P., Worth, et al., “Growing Music: music interpretations of L-Systems,” Conference Paper, Department of Computer Science, University of York, York, UK, 2005.

[7] Birmingham City University. (n/a). *Richard Burn: Music-making for the deaf* [Online], Available: < https://www.bcu.ac.uk/research/midlands4cities/richard-burn-music-making-for-the-deaf >

[8] YouTube, Birmingham City University. (2015, Nov. 15th). *Music-making for the deaf at Birmingham City University* [Online]. Available: < https://www.youtube.com/watch?v=lbZ-VYHS9tQ&t=89s >

[9] YouTube, Peer Play. (2016, Sep. 17th). *Audio Visualization – Unity/C# Tutorial [Part 1 – FFT/Spectrum Theory]* [Online]. Available: < https://www.youtube.com/watch?v=4Av788P9stk&list=RDCMUCBkub2TsbCFIfdhuxRr2Lrw&index=1>

[10] YouTube, Peer Play. (2016, Sep. 17th). *Audio Visualization – Unity/C# Tutorial [Part 2 – GetSpectrumData in Unity]* [Online]. Available: < https://www.youtube.com/watch?v=4Av788P9stk&list=RDCMUCBkub2TsbCFIfdhuxRr2Lrw&index=2 >

[11] YouTube, Peer Play. (2016, Sep. 18th). *Audio Visualization – Unity/C# Tutorial [Part 3 – Visualize 512 samples]* [Online]. Available: < https://www.youtube.com/watch?v=Ri1uNPNlaVs&list=RDCMUCBkub2TsbCFIfdhuxRr2Lrw&index=3 >

[12] YouTube, Peer Play. (2016, Sep. 20th). *Audio Visualization – Unity/C# Tutorial [Part 4 – Eight Frequency Bands]* [Online]. Available: < https://www.youtube.com/watch?v=mHk3ZiKNH48&list=RDCMUCBkub2TsbCFIfdhuxRr2Lrw&index=4 >

[13] Unity Documentation. (2021, Dec. 23rd). *Unity – Scripting API: AudioSource.GetSpectrumData* [Online]. Available: < https://docs.unity3d.com/ScriptReference/AudioSource.GetSpectrumData.html >

[14] Unity Documentation. (2021, Dec. 23rd). *Unity – Scripting API: FFTWindow* [Online]. Available: < https://docs.unity3d.com/ScriptReference/FFTWindow.html >

[15] National Instruments Corp., "Understanding FFTs and Windowing," Instrument Fundamentals Series [Online], 2021. Available: < https://lumen.ni.com/nicif/US/GB_INFOINSTFUNDGUIDE/content.xhtml >

[16] Michigan Technological University. (2021). *Frequencies for equal-tempered scale* [Online]. Available: < https://pages.mtu.edu/~suits/notefreqs.html >

[17] National Instruments. (2022). *Understanding FFTs and Windowing* [Online]. Available: < https://download.ni.com/evaluation/pxi/Understanding%20FFTs%20and%20Windowing.pdf >

[18] LDS Dactron. (2003). *Application Note AN014 | Understanding FFT Windows* [Online]. Available: < https://www.egr.msu.edu/classes/me451/me451_labs/Fall_2014/Understanding_FFT_Windows.pdf >

[19] Musanim. (2022, Jan. 29th). *Stephen Malinowski and the Music Animation Machine* [Online]. Available: <http://www.musanim.com/Background/>

[20] YouTube, TEDx Talks. (2012, Nov. 28th). *Music Animation Machine: Stephen Malinowski at TEDxZurich* [online]. Available: <https://www.youtube.com/watch?v=EAWSonBN3Pk>

[21] W.Y. Chan, et al., "Visualizing the Semantic Structure in Classical Music Works," Thesis, Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, 2009.