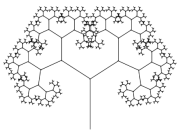
Controlling Generative Systems with Audio Signals



Oskar Schachtschneider

Fachbereich VI - Medieninformatik

Berlin Hochschule für Technik

Contents

1 Introduction

The description of physical processes of sounds or of music and their origins has always been an important and widespread research topic. Already the Greeks, especially Euclid and Archytas, tried to describe the properties of music physically and mathematically, for example, the question of the origin of sound and the mathematical description of harmony. The description of the physical laws also serves the understanding of the studied instruments and can help to analyze them more precisely. Nowadays, audio signals are mainly analyzed digitally. This analysis is universally applicable and finds appeal in all areas. There are sleep trackers that use the sounds at night to determine the sleep phase, AI voice assistants that respond to mood, or systems that can detect malfunctions in mechanical components. There are a variety of character styles that can be extracted from an audio signal. These characteristics are mostly abstract data describing the audio signal. Generative art or process art is a form of contemporary avant-garde art developed in the 1960s based on the ideas of minimal art and performance art. This art form is mostly digitally generated and expresses itself in abstract images, videos, or animations. Early on, the industry adopted these systems and let generative do the work of architects, game designers, and engineers. For the creation of a generative work, it needs certain input data, which are mostly generated randomly under framework conditions. This thesis deals with generating this input data from an audio signal. Randomness also plays an important role here. It is randomly selected which generative system is used. Once this system is initialized, it continues to use the audio signal as input data.

This thesis deals with the

1.1 Motivation

Generative System is becoming more and more popular, and their techniques are used for live performances, art exhibitions but also commercial purposes. In the past, artists used many creative approaches and algorithms to create Images, Music, and Videos. The modern Day approaches lean more towards 3D renderings and physical machines. The demands for computers have also changed; complex 3D renderings are in need of heavy computational power and optimizations. This is especially true for live performances and real-time generation. Artists strive for new tools to communicate their vision and try to find new ways of developing their own style. Long-standing techniques like the Mandelbrot do not qualify for this anymore. So the Mandelbulb is the next iteration of this famous algorithm, featuring complex 3D scenes and endless artworks with a high grade of Detail. Artists see Generative Art as a tool in their toolbox next to the brush and the Canvas. These systems are able to generate an endless amount of artworks within a given style. These artworks are mostly created by using controlled randomness.

leans more towards to

1.2 Goals

For this Bachelor thesis, a system is created that generates data-driven 3D real-time artworks. The System can be accessed through a native application that runs on every major desktop operating system. The artworks will be generated by layering 3D Models, Textures, and Particle Effects. These artworks will be created semi-randomly, an incoming audio signal is analyzed, and based on the audio features, they will be affected. This includes Camera position, Animation speeds, and general Asset properties. These systems will be designed to make them suitable for live audio / visual performances with a low setup cost. Therefore the user interaction will be kept to a minimum, and this will mostly be a plug-and-play system. For the artworks, there shall be some degree of variance to them. However, since the artworks are created by a system using models and premade textures, there is a certain degree on how much it will differ each time. The goal here is not to create algorithmic art that creates art purely out of code but rather to control a generative system with audio signals that comes with its assets. To use this System in a live audio/video performance, the artist needs to use its models and textures to create a unique style since this application base only comes with generic assets. In addition, the System shall be designed to be easily extensible using Unity3D. The artist who has modeling and some programming skills shall create and load their models to create a unique composition. Additionally, the System shall be built with performance in mind. Finally, the System will be loosely coupled to make it easily extensible and allow other data sources for the artworks.

Delete this syste

1.3 Structure

In the next chapters basics like Generative Art techniques, Audio Analysis and related work will be covered. The last chapters of this thesis will focus on the requirements, the design and implementation and will finish of with a retrospect. In the retrospect alternatives and possible future work is discussed.

2 Generative Systems

2.1 Algorithmic Art

Algorithmic or Computer Art is created by a computer program for the purpose of being exhibited as art. It is a genre of art in which the artist is an algorithm. The algorithm is usually more complex than those used to generate patterns and pictures in computer animation, but the defining features of the genre are that the output of the algorithm is exhibited in a gallery and that the algorithm is not intended to be comprehensible to human viewers. The term was coined by Roy Ascott in 1986, who then proceeded to found the Algorithmic Arts Society. Algorithmic art was initially developed by artists associated with the algorithmic and generative art communities.

The most famous generative art program is probably "AARON" by Harold Cohen of the University of California and the "Cypher" program by Barry Truax, who is often considered the father of computer art. In the early 2000s, the genre began to be adopted by mainstream video game companies and art institutions. For example, "Proteus" was an experimental game developed by Ed Key and David Kanaga and released in 2013 by Key in conjunction with the show "Sound States: The Art of Video Games" at London's Barbican Centre. The art in the game was generated by a computer program that generates fictional flora and fauna and changes the player's surroundings. In the mid-2000s, after about a decade of popular video games being created using the same technology that drives films, television series, and cartoons (i.e., 3D rendering and CGI), a few companies began looking towards the video game industry's roots, and towards the arts, in search of new ideas which could be used to improve video games. In 2012, Nintendo released "Pikmin 3", a game that features an artist protagonist and that allows players to paint their surroundings. The same year, Electronic Arts released "FIFA 13", which allows players to control a virtual ball with the gamepad, moving it around and painting it with the controller's buttons, and then use the ball to play a game of football. "FIFA 13" was not the first game to allow players to control a virtual ball, but it was the first to allow players to paint the ball. In 2013, the art and technology magazine "Wired" named "FIFA 13" and "Proteus" among the five most important art projects of the year, alongside "Videogames For Artists" (a game development competition hosted by the Tate Museum and the Tate Britain).

2.2 Generative Art

Generative Art is not only created by computers but also by humans. Some artists have also found a way to generate Art by playing a computer games.

In the early 1960s, computer art was created by directly manipulating a computer's memory. Programs were written to populate the memory with numbers randomly, and then a program would be written to generate an image by controlling how the pixels could be modified.

by playing computer gam

how the pixels are modified

same sentence 2 timse

The first generative art program was created in 1966 by Brion Gysin, a Beat author and artist from New York. He created the program to explore the potential of computers in Art. Generative Art was often created by a jitterbug program, a more complex method of randomizing pixels. The jitterbug and data randomization is used to create randomness in Generative Art. A *jitterbug program* is an algorithm used to create randomness in Generative Art. It is a recursive algorithm that uses a few simple rules to generate a larger dataset. The algorithm can be thought of as a loop and a conditional statement.

The jitterbug algorithm runs in two modes: "forward" and "reverse." In the forward mode, the program will start from the first element in a dataset and apply a series of rules to that data. In the reverse mode, the program will start from the last element of the dataset, apply rules to it, go to the last element, and so on.

go to the next ele

The p5.js framework is widely used for web-based generative systems. This framework is based on processing, a programming language that makes it generatively easy to create shapes, patterns, and colors. Ultimately, a generative system is not dependent on the programming language or framework; it can be created with almost any programming language/framework. In this thesis, Unity3D is used, which simplifies processes like rendering, analysis, and management.

2.3 Definition of Generative Art

If you search for a definition for Generative Art, you can find several definitions. For this thesis, the definition of Philip Galanter is used: "Generative Art refers to any art practice where the artist uses a system, such as a set of natural language rules, a computer program, the machine, or other procedural invention which is set to into motion with some degree of autonomy contributing to or resulting in a completed work of art." What is interesting about this definition is that it does not limit generative art to computers that generate images.

2.4 Randomness

2.4.1 Pseudo-randomness

Pseudo-randomness is a form of Randomness that approximates the properties of Randomness. It is often implemented by a deterministic algorithm that generates numbers according to a (pseudo-) random number generator algorithm. In some cases, the algorithm can be made to have the same general properties as a random one, such as being unbiased.

For example, one can choose a random number in the range 0–1 by generating a random integer in the range 1–n. This works well if n is large but will not work well for small values of n.

A better algorithm is to generate a random number in the range 0–1 by generating a random fraction x in the range 0–1 and then generating x times a random

number in the range 0–n. This way, one can generate random numbers for any n.

The Perlin noise is another way of creating Randomness. The Perlin noise is based on the idea of creating a smooth "random" function. The following Python code uses the Perlin noise to generate a smooth "random" function.

delete

2.4.2 Randomness from physical systems

Randomness can also be obtained from physical sources. Many of these methods are often also called "noise" and are used in simulations of natural systems.

For example, consider a marble rolling on a flat surface. If you measure the motion of the marble, it will seem to be determined, but if you measure the marble at a larger scale, the trajectory of the marble will seem to be random. This is because, on a larger scale, the forces of the rebound of the surface will be averaged out and will no longer be visible.

non formal wı

Another example is the radioactive decay of an atom. The decay process is completely determined by the laws of physics, but the time of the decay is random.

One can also use the thermal noise of a resistor to come up with a physics-based random value

2.4.3 Randomness from humans

Humans are often also an important source of Randomness. A human-generated random number can be obtained, for example, by flipping a coin. The probability for a head or a tail is 50%, which makes the outcome of the coin flip random.

Another way is to use dice to generate Randomness. A dice has six sides, and each side has a number. If you roll the dice many times, the numbers will show up in random order. In fact, there are 6! possible outcomes of a dice roll.

non forn

2.5 Games, Movies and Music

In the film industry Generative Systems are heavily used. These systems are used to create the whole movie, from story to editing. Whole scenes are generated, ranging from a large skyline of skyscrapers to other wordly alien planets. An example of this is Sky Captain and the World of Tomorrow. The movie Sky Captain and the World of Tomorrow is a movie that uses artificial intelligence to generate the whole video, with no one scene the same as the last. However staying true to the script of the artificial intelligence an ending was generated which is considered by its fans as "one of the worst endings ever". This movie was a perfect example of how using a Generative System can go wrong, as the ending was so bad that it ruined the whole movie.

Another example of a Generative System is the video game Spore. Spore is a video game where the player creates an organism, and then the player can control their organism, explore the world, and create new organisms with their unique body shape and behaviors. As well as this, the player can also create vehicles and buildings. The player also has to battle against other organisms, for example, for survival purposes. The Organisms evolve, as the player creates new DNA, which then allows the organism to be able to reproduce, but not only that, the new offspring can also be unique, this is because the offspring is created from a combination of its parent's DNA. The game also allows the player to explore a whole new planet, and there is a galaxy of planets that the player can explore. The player can also explore other planets through spaceships or build their spaceship. The video game has also been used as a learning tool for AI, as it is an excellent example of how a Generative System works and how it can be used in the future.

A *generative music system* is a computer system that can create musical pieces based on rules. These pieces are typically relatively short, but they can be combined to create longer pieces. There are many different ways to create a generative music system. One popular approach is to use a Markov chain. A Markov chain is a system where the current state determines the next state but not the previous states. This approach can create pieces that are never heard before but still sound like they are part of a larger piece. Another popular approach is to use a genetic algorithm. A genetic algorithm is a system where pieces are created by combining different parts of other pieces. This approach can create pieces that are never heard before and sound very different.

Generative music systems have been used in several different areas. One important area is music composition. Many composers have used generative music systems to create new pieces of music. For example, in 2001, Steve Larson created a piece of music using a Markov chain. This piece was later used in a commercial for a large telephony company. Another important area is music for games. Many games use generative music systems to create music for the game. This music is typically used to create music while the player is just moving around, waiting for something to happen, or just standing around.

2.6 Techniques

The most used techniques for Generative Art will be discussed in the following. Most of these techniques are inspired by nature laws or mechanics. For example, the L-system or the Lindenmayer system is formal grammar and a string rewriting system that generates fractals, trees, and plants. Most of the generative techniques can be categorized by predictable and nonpredictable execution. This distinction is important because it relates to the ability of the techniques to leverage the re-use of the code. The techniques based on randomness or unpredictable execution are not re-usable because they are not predictable, and the same result cannot be obtained in a different context. In contrast, the techniques based on an algorithm or known pattern can be re-used in different contexts as long as the input and the external environment are the same. The algorithms based on randomness are not re-usable because they depend on context, and the same code will not work in different contexts. For example, the genetic algorithm uses randomness and some function to optimize a value. This function depends on the context, and it would not converge into the same result if the input data is different. For example, if someone wants to use a genetic algorithm to solve a new problem, he needs to create a new code and hope to get the same result, but this is not guaranteed. The algorithms based on algorithms are re-usable because they are predictable, and the user can get the same results in different contexts. For example, analogical reasoning generates a solution based on a known analogy, and the user can use the same code in different contexts. Research shows that most Generative Art algorithms are based on Genetic Algorithms. This class of Evolutionary Algorithms grows a population of possible solutions to a problem and then evaluates these solutions to determine the quality of each. The best solutions are then allowed to "survive" to the next generation. These methods create the most visually pleasing results but pose some challenges for the artist in the form of complexity and limits. Artists have developed their methods for Generative Art, which tend to be more flexible and accessible to the user. The most commonly used algorithms are Cellular Automata, Rule 30, Reaction-Diffusion, and Perlin Noise.

2.6.1 Predictable Methods

in his book

2.6.1.1 Fractals Fractals are objects with a very complex structure or shape generated by a simple iterative algorithm. Fractals are commonly approximated by iterative methods. Benoit Mandelbrot mentiones the term “fractal” first in Book “The fractal geometry of nature.” According to Mandelbrot, “Fractals are geometric objects usually studied in relation to the study of dynamical systems, of which self-similarity is a first typical characteristic.” Fractals are generated by self-similarity (similarity of parts to the whole). This self-similarity can be observed at different scales.

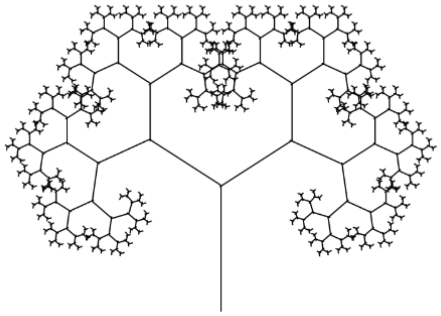


Figure 1: Artwork based on the L-System using p5.js

2.6.1.2 The Mandelbrot set One example of a fractal is the Mandelbrot set. The Mandelbrot set can be generated by a simple iterative algorithm. This particular fractal is generated by iterating the following equation: Where X is a complex number, z0 is a complex number, and z is a complex number. The value of Z after each iteration is the result of the previous iteration, which means that the value of Z is a function of the value of Z. This process is iterated until a certain iteration count or a boundary condition is reached.

To plot the Mandelbrot set, map each pixel on the screen to a complex number. Then check if it belongs to the set by iterating the formula, and color the pixel black if it does and white if it doesn't. Since the iteration may never end, we set a maximum.

non formal la

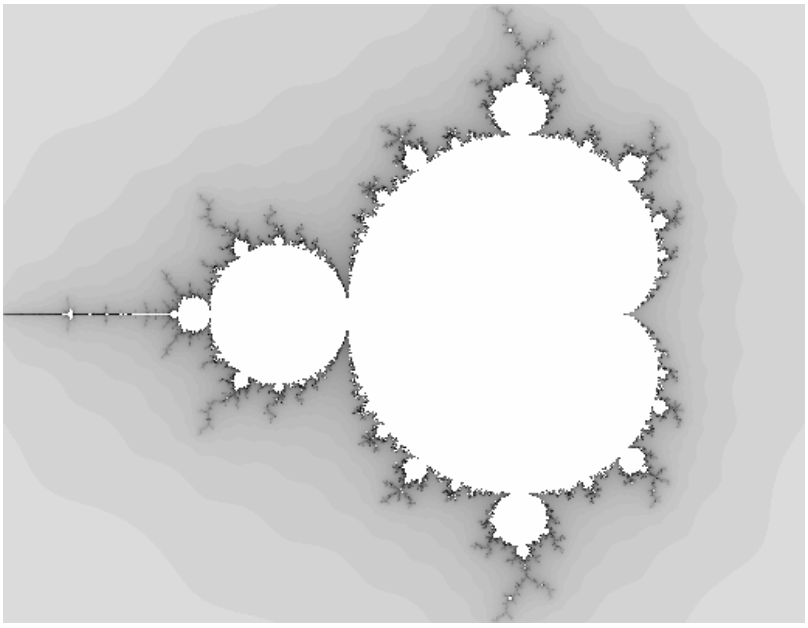


Figure 2: Rendering of the mandelbrot set with p5.js

2.6.1.3 Mandelbulb The Mandelbulb is the 3D analogue of the Mandelbrot set. It is created by applying a different equation to the set of complex numbers. It is a 3D space filled with tiny Mandelbrot sets.

The result is a 3D set of complex numbers. The following figure shows the result of applying the iterations to the set of numbers [-1.4, 1]x[-1.4, 1]. The result of applying the iterations is represented by a black point. The iterations will stop when the resulting point is very small.

Just like the Mandelbrot set, the Mandelbulb is defined by a simple recurrence; but in 3D.

A point (x,y,z) is in the Mandelbulb set if (and only if) the following formula is satisfied:

f(z) = z^2 + c.

Where: f(z) is the iteration and c is a constant.

The Mandelbulb is computed by solving the 3D recurrence. The Mandelbulb itself is not the result, the result is a 3D grid of points (x,y,z) which are the locations of the Mandelbulb set. These points can be plotted to create the image of the Mandelbulb.

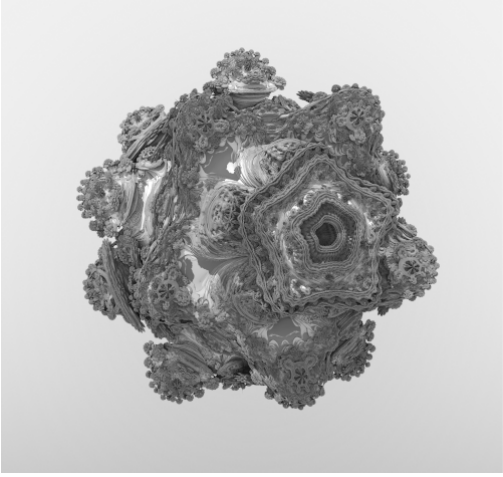


Figure 3: Rendering of a Mandelbulb using mandelbulb

2.6.1.4 L-systems The Lindenmayer system is a method of describing the production of shape and color in plants and animals. The Lindenmayer system is like a language, with symbols that have a meaning. It is also like a code where the symbols have no meaning until they are decoded. The Hungarian Aristid Lindenmayer invented the Lindenmayer system in 1968. Lindenmayer made two important discoveries. The first was that any object, such as a plant, animal, human, building, machine, or river, can be described in terms of how it is put together from a few essential elements. The second discovery was that the way the essential elements are put together could be described using a language of symbols that have a simple meaning. If you do not know the meaning of the symbols, but you do know the language, then you can decode the message contained in the symbols to see what the object looks like. Lindenmayer noticed that although the world is complex, it is possible to describe the world by repeating a few essential elements or shapes. So, for example, if you look at a plant, a few basic shapes make up the plant. These basic shapes are of different types. For example, there are linear shapes, such as straight lines, curved lines, and circles. There are also looping shapes, such as bends and corners. There are branching shapes, such as forks and branches. Moreover, some other shapes do not fit neatly into these categories.

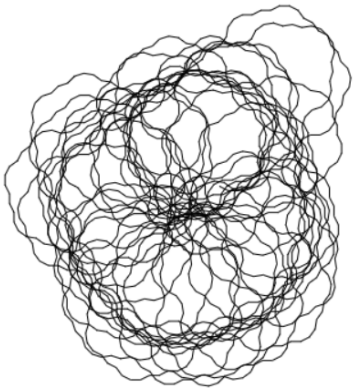


Figure 4: Artwork based on the L-System using p5.js

2.6.1.5 Cellular Automata Cellular Automata consists of a grid of cells, and each cell has some state. The cells can change state according to some rules. For example, a cell could be black, red, or blue. There are several cellular automata rules which are used:

In the book "a new kind of science," Stephen Wolfram introduces four classes for cellular automata. The first class has the highest complexity, which requires the most amount of steps to reach a stable state. The next class has lower complexity, and so on.

Conway's Game Of Life is one example of this. It is a cellular automaton that has a complexity of 2. It has the following rules:

- Any live cell with fewer than two live neighbors dies as if by loneliness.
- Any live cell with two or three live neighbors lives on as if by reproduction.
- Any live cell with more than three live neighbors dies as if by overcrowding.
- Any dead cell with exactly three live neighbors becomes a live cell as if by reproduction.

The rules of this cellular automata eventually reach a stable state.

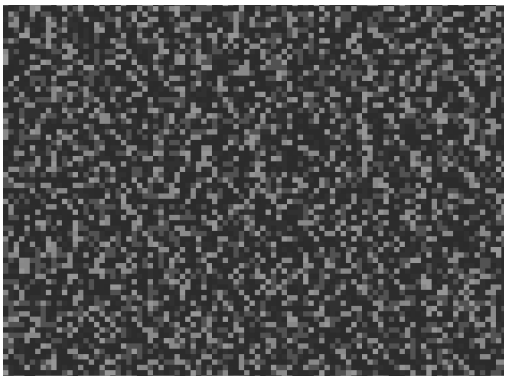


Figure 5: Rendering of example cellular automata algorihm

2.6.1.6 Genetic Algorithms Genetic Algorithms (GA) use the same principles as nature in evolution. Individuals are created, and the individuals are evaluated and ranked. The best individuals are selected to breed, and the offspring is created by applying a genetic operator. The offspring is again evaluated, and the ranking is updated. This is the so-called survival of the fittest. The outcome of a genetic algorithm is usually split into two parts: The genotype and the phenotype. The genotype is the description of the individual, and the phenotype is the result of the evaluation. The algorithm is usually evaluated by the phenotype and not by the genotype. The phenotype is evaluated by a fitness function that returns a number between 0 and 1. The fitness function is the most crucial part of the genetic algorithm. If it is not working well, the genetic algorithm will not work well. The fitness function should be clear, meaningful, and well-defined. It should be clear so that the programmer knows what it means. It should be meaningful so that it is a good way to evaluate the results. The fitness function is described in a very general way. Moreover, it should be well defined so that it is mathematically defined and can be used in an automated fashion. The programmer will have to fill in how the fitness function is evaluated. The fitness function is often used to evaluate the result of the algorithm and is used to determine the next generation. The better the fitness function is, the better results can be expected. The fitness function is often called the goal function. It is used to determine the goal of the algorithm.

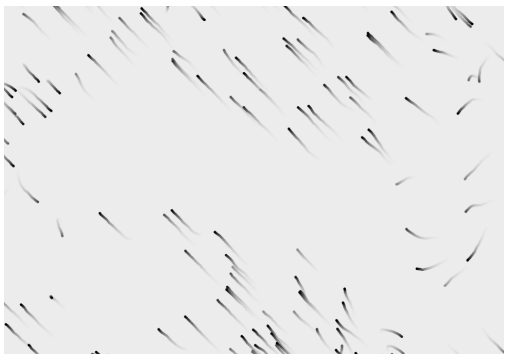


Figure 6: Particle system rendered with p5

2.6.1.7 Particle Systems Systems that consist of many small objects ('particles') that are emitted from a source are called particle systems.

The behavior of the particles is governed by simple rules. The particle system can be seen as a particle pipeline. The result is a sequence of particles that can be used for modeling many phenomena.

rewrite

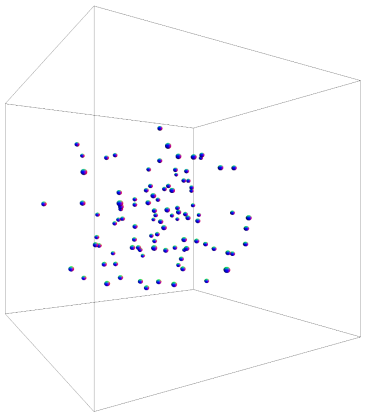


Figure 7: Particle system rendered with p5

2.6.2 Less predictable Methods

2.6.2.1 Data driven Data-Driven approaches for Generative Art try to get away from the need for human interaction by converting data into instructions for the model. Then the instructions are executed to draw the artwork. Data-driven artworks follow a deterministic approach and are predictable if the input data is known.

Often times data-driven approaches are chosen for data visualisations since they map directly to the data and often do not alter the input.

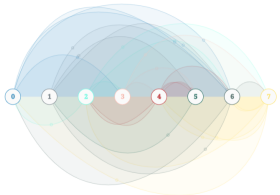


Figure 8: Data driven rendering with p5.js

2.6.2.2 Noise Functions The noise function has many different applications in creative coding. It is used for procedural textures, particle systems, and defining probability distributions. Noise functions are heavily used in creative coding when pseudo-random numbers are needed. The Perlin noise, for example, is a function that generates random numbers that resemble the noise in nature.

One of the essential noise functions is the Perlin noise. Out of frustration with machine-like generated imagery, Ken Perlin developed the Perlin noise in 1983. Perlin noise is a procedural texture, a gradient noise used by visual artists and musicians to achieve more realistic random functions. The creation of Perlin noise is achieved through turbulence (which can be described as a chaotic computation).

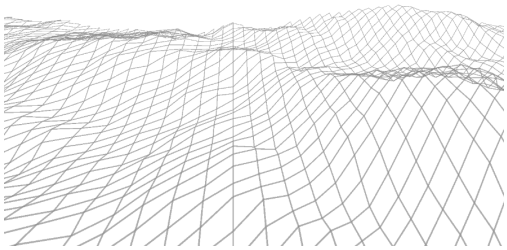


Figure 9: Perlin Noise based generated Terrain with p5.js

2.6.3 Evaluation of less predictable methods

An advantage form the less predictable methods over the more predictable ones is, that there is more potential for greater details without handcraftig these. Moreover the less predictable method may be used to create a series of artworks. The same algorithm is able to generate an endless stream of images that are each different from each other but share the same style. For example a data-driven approach that uses realtime data from a social media feed will never recieve the same stream twice. This can be a very exiting approach because the artwork is still altered after its creation in contrast to static artforms like painting and music. However, the less predictable methods may lead to unwanted results.

3 Audio Analysis

3.1 Frequency

Frequency describes the number of waves that pass a fixed place in a given amount of time. Frequency is an essential concept in physics because it can describe any phenomena that exhibit periodic or rhythmic behavior. Frequency is also an important parameter used engineering to specify the rate of oscillatory and vibratory phenomena, such as vibrations, audio signals, radio waves, and light.

used in

When talking about frequency in the context of audio signals, the frequency determines the pitch of the sound. The greater the frequency, the higher the pitch. Frequency is measured in Hertz (Hz), named after the German physicist Heinrich Hertz. One Hertz is equal to one cycle per second or a frequency of one cycle per second. The Hertz is the SI unit of frequency, which measures the number of occurrences of a repeating event per unit time. In the International System of Units (SI), the Hertz is defined as a multiple of the SI unit of frequency, the Hertz, and can be used to express other units of frequency; the kilohertz, megahertz, gigahertz, terahertz, and petahertz are commonly used multiples.

The information in the frequency spectrum is not filled equally, i.e., higher frequencies are more densely packed than lower frequencies, so it is essential to specify the range of frequencies being considered. The frequency spectrum of a signal is a graph of the signal's amplitude versus frequency. In a simple case where the signal is the output of a single sinusoidal oscillator, the signal consists of a single sinusoidal waveform. In this case, the signal's frequency spectrum consists of a single line at the frequency of the signal. The frequency spectrum of a signal is often measured using a spectrogram.

A spectrogram is a visual representation of the signal's frequency spectrum. A spectrogram is a two-dimensional representation of the signal's frequency spectrum. A spectrogram can be generated by filtering the signal using a bandpass filter and plotting the magnitude of the resulting signal over time. The frequency spectrum can be used to identify various components in the signal. For example, if the frequency spectrum has a single line at a specific frequency, then the signal is a sinusoidal function with that frequency. If the frequency spectrum has several lines at different frequencies, then there are multiple sinusoidal components in the signal, such as a series of harmonics. The frequency spectrum can also measure the amount of energy in each of the frequency components. This is done by determining the power of each component of the signal. The power of a signal is the square of the signal's amplitude (or the square of the magnitude of the complex-valued signal).

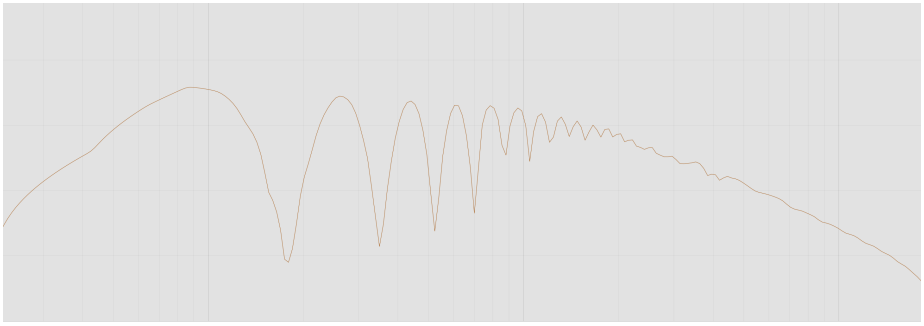


Figure 10: Spektogram showing a oscilating saw wave

3.2 Sampling

Sampling is the reduction of a continous-time signal to a discrete-time signal. In the Audio world this translates to converting a sound wave to a sequence of samples (a discrete-time signal). One sample represents a point in time which holds the value that was converted from the sound wave. To capture a sound wave digitally that enough information is present to fully reconstruct that sound digitally the sample rate needs to be chosen properly. The sample rate defines the resolution that is used to represent a sound wave. Higher values will give more accurate representation of the original sound wave but will require more storage space and bandwidth. The sample rate of a sound wave is not the same as the frequency that represents the pitch of that sound wave. The sample frequency (in Hz) is calculated as the sample rate divided by the length of the sample. The sample rate is actually the frequency of the digital representation of the sound wave, not the frequency of the sound wave itself. So the sample frequency is much higher than the frequency of the sound wave. The Nyquist Theorem states that a sample frequency of at least twice the highest frequency in the sound wave is needed to recover the original sound wave.

in a way

The Nyquist Theorem states that in order to fully reconstruct a sound wave from a sampled version of that sound wave, the sample rate has to be at least twice the highest frequency in the sound wave. If the sample rate is less than twice the highest frequency in the sound wave, the sound wave can not be fully reconstructed. This is one reason why digital audio often sounds “thin” and “hollow” and is a common complaint about MP3s. The digital samples do not have to be taken at regular intervals. If the samples are taken irregularly and more frequently around the wave peaks, the resulting digital waveform is more faithful to the original waveform.

Here is a example of a poorly choosen sample rate and its reconstruction.

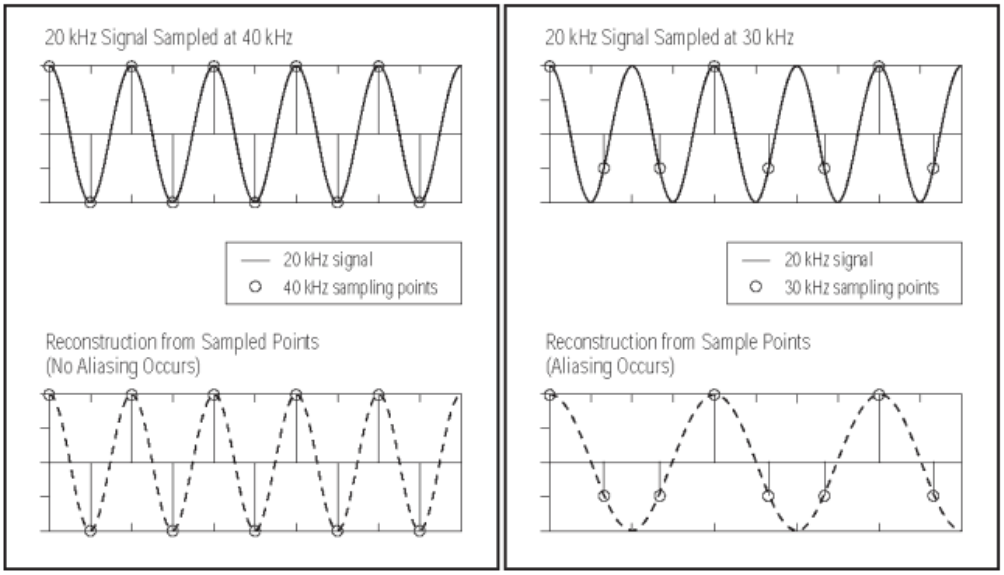


Figure 11: Showcase of different sample rates and its effects

3.3 Fast Fourier Transformation

The human ear automatically and involuntarily performs a calculation that takes the mathematicians years of education to accomplish. The ear transforms the waves of pressure that move through the air over time and converts it into a spectrum that describes the sound as a a series of volumes at specific pitches. The brain turns this information into percieved sound.

A similar conversion can be done using mathematical methods on the same sound waves. The Fourier transform [1] is a mathematical method to convert waveform data in the time domain into the frequency domain. The reverse Fourier transform turns the frequency domain back into a waveform. The Fast Fourier transform is build on the same principles

principles .

In 1965 the fast Fourier transform [2] used enourmus amounts of computational power which lead James W. Cooley and Jown W. Tukey to the invention of the Fast Fourier transformation. The Fast Fourier Transformation (FFT) has been used for the past decades to obtain the frequency domain representation of the waveforms. In this work, FFT's are used to convert the signal form from its original domain to a representation in the frequency domain. The FFT is performed on the windowed version of the signal. The features are extracted by applying a vector quantization on the frequency domain representation obtained. The results show that the features extracted using the Hamming function are very different from the features extracted using the FFT without the Hamming function. The proposed method is tested on speech signals, music signals and

audio signals. The results show that the proposed method is able to distinguish between the different categories of signals and offers the best result for musical pieces.

3.4 Window function

A fifth window function works by dividing the signal into a number of segments and then using the same function on each segment. The result is that the frequency components of the signal are preserved, and only the magnitude is scaled. The window function determines the number of segments. This can be done by calculating the length of a side of a triangle. You then divide the length into the number of segments you want. The window function also determines the number of points that are used in the windowed FFT. This means that the windowed FFT is only good for a given size of the signal. A triangle window is a good choice for most signals. It is a good trade-off between not losing too much information and not wasting too many operations. A Hann window is used for signals with a large range of frequencies, such as audio signals. A Hamming window is used for signals that have large amounts of noise, such as images.

3.4.1 fft_sinc

- This function is a good choice for signals with a large range of frequencies.
- This function is not good for signals with a lot of noise.
- This is a good choice for audio signals because it retains the low frequencies and it is fast.

3.4.2 fft_hamming

- The fft_hamming function does a Hamming windowed fft. This function is even faster than the fft_sinc function because it uses less points.
- This function is a good choice for signals with a lot of noise. This is not a good choice for audio signals because it removes the low frequencies.

3.4.3 fft_blackman

- The fft_blackman function does a Blackman windowed fft. This function is ideal for audio signals because it does not remove the low frequencies.
- This function is not good for signals with a lot of noise. This is not a good choice for signals with a very large range of frequencies.

3.4.4 fft_kaiser

- The fft_kaiser function does a Kaiser windowed fft. This function is good for signals with a large range of frequencies and that also have a lot of noise. This function is not good for audio signals.
- This function is not good for signals with a large amount of noise.

3.5 Audio Features

3.5.1 Brightness

Brightness either alludes to a solitary note or harmony played at a specific time, or to a track as an entirety, at the point when the splendor, everything being equal/harmonies, is summarized. In the setting of a note or harmony, splendor would be a discrete occasion, whereas the brightness of a track would be a worldwide element. Brightness is normally characterized by a note's pitch, e.g., a C' (played in the principal octave) is thought of as less splendid than a C" (played in the subsequent octave). Albeit musically talking that is a similar note (same key), it has an alternate splendor. Similar applies to harmonies. Brilliance is additionally to some degree associated with harmonicity - significant harmonies are generally thought of to be "cheerful," which overall likewise prompts a sensation of expanded brilliance, whereas minor harmonies are depicted as "sad" or "melancholic," providing the audience with the sensation of diminished brilliance. Actually talking, "brightness portrays the unearthly dispersion of frequencies and portrays whether a sign is overwhelmed by low or high frequencies, individually. A sound becomes more splendid as the high-recurrence content turns out to be more predominant, and the low-recurrence content turns out to be less predominant." As far as MIDI, the splendor is straightforwardly associated with the pitch of a note, which is worth somewhere in the range of 0 and 127.

splendor = brigl

3.5.2 Loudness

Loudness essentially depicts how clearly a note has been played. Loudness makes characteristical spikes in waveforms (sound) or influences the volume quality of MIDI notes. Over the previous years, music makers have combat in an alleged tumult war [22], attempting to make tracks with an enormous and serious sound. This leads to more thick and smaller waveforms, making it much harder to effectively dissect sound records. As far as representation, tumult can be utilized to influence the power of tones or visual occasions.

tumult = loudness

3.5.3 Tonality

The resonance or key portrays which notes are utilized inside a song. In spite of the fact that resonances hold a substantial arrangement of notes, most notes are essential for a few resonances. Also, due to compositional opportunity, the resonance can change whenever inside the track, and it isn't said that assuming a melody is written in a specific key, that it just holds back notes from that specific scale. This is pretty normal, particularly in Jazz and Latin music, and makes it much harder to distinguish the right resonance while dissecting a melody. Like brilliance and harmonicity, resonances have a specific inclination joined, which can be either happy or melancholic. This makes resonance identification an intriguing assignment while attempting to picture bits of music.

3.5.4 Tempo / Beats per Minute

The beat of a track is a decent base for changing the overall style of any melodic representation. Quicker melodies transport a sensation of energy or aggressivity, whereas more slow tracks typically actuate a laid-back and repressed feeling. BPM represents beats per minute, which is a mathematical worth that portrays the number of quarter notes each moment played during a track. The rhythm or beat of a track is without a doubt what gives the track the force and flare that is unique to the track. Rhythm and pace are the most vital components when it comes to the representation of a track. The rhythm and pace of a track is a standout amongst the most important components of music and melody.

The formula to calculate the BPM is:

$$(60/MS) * 1000$$

3.6 Extracting Audio Features

Digital analysis of audio signals emerged with the advent of the digital age. Julius O. Smith III can be considered one of the pioneers who did significant work in the field of physical signal analysis and processing. Today, digital audio analysis is used in DAWs (Digital Audio Workstation), DJ programs, and AI voice assistants. The main distinction is between real-time and look-ahead analysis; usually, a mix of both systems is used.

There is a range of audio features which can be extracted from a given audio signal through either parametric or non-parametric methods. In parametric methods, a frequency table is constructed from the audio signal, and the audio features are extracted from the table. In non-parametric methods, the audio signal is transformed in a way that makes the feature extractable. For example, in the frequency domain, the low-frequency components are extracted.

The audio signal can be segmented into frames, usually of size 20ms. The segmented signal can be transformed into different domains to extract the audio features. The domains used are:

Frequency domain: This domain is used to extract the frequency and its harmonics. Usually, the signal is first transformed into the frequency domain using a Fourier Transform. This is an efficient method, and it is fast. However, it requires a large amount of memory, and the computation of the frequency spectrum is not trivial.

Spectral-domain: This domain is used to extract the spectral amplitude at each frequency. It is a subset of the frequency domain.

Energy: It is the total energy of the signal. It is computed as the sum of the square of the amplitudes of the frequency components.

Onset Detection: Onset refers to a sudden spike in amplitude which can be measured.

4 Related Work

In the following, two related artworks will be discussed. Both works use a genetic algorithm in order to create artworks. In 3.1 a playground is given. The computer explores the creation of new artworks by tiling this playground and by re ordering the tiles to. This gives an unpredictable outcome but does clearly feature a visual style. This initial visual style is defined by the rules of the playground. The second work explores how music as data can be used to paint and to visualize artworks. Each work in their gallery does implements this data differently to create new and unique artworks. One of the central questions is "How does music actually look like" is explored by trying to interpret music in different ways. Furthermore this project explains how music is made and what it is made of on basic level. Similar to this thesis this project does view music as data that can be visualized differently.

4.1 Generative & Conceptual Audio-Visual Art

In their Paper "An Evolving Musical Painting on the Boundary Between Permanence and Change" [3] by Yiannis Papadopoulos*, David Parker, Martin Walker, Emma-Jane Alexander an ever changing painting was drawn. They describe the project as "The research vision of this project is to create a new strand of research that intersects computer science, philosophy, medicine and art with potentially novel applications and impact on conceptual art, art therapy and educational games."[3].

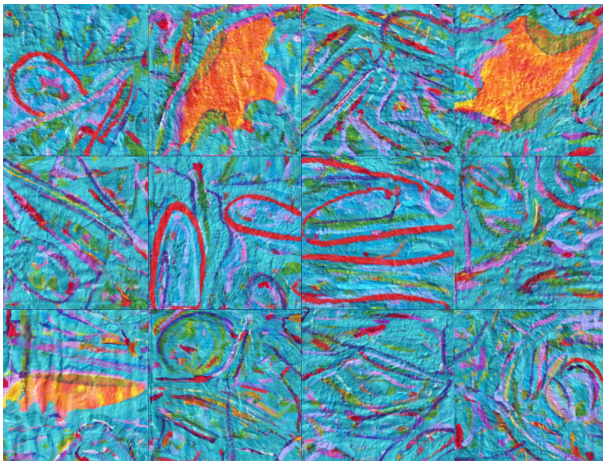


Figure 12: "East" (Musical painting #1): Visual artwork by Roberto Bono, "Dhakartu Siqilliyata" - Music by Bob Salmieri

Additionally music by the musicians Roberto Bono, Andrea Alberti and Bob Salmieri have been used to create a matching landscape by applying the same logic to their music. The painting just like the music is created by defining a

"playground", a set of rules and systems that has been put into place by the artists. Once the playground is set up the computer explores new sounds and images by tiling the playground into multiple tiles and starting to rearrange them creating new paintings. In the same manner the music is chopped and tilled and does make up the sound scape for this artwork. Furthermore this project explores the potentials of therapeutic effects of these artworks. Informal feedback by patients and clinicians that have experienced this art suggest that this may have a therapeutic effect on people with long term conditions like autism, tinnitus or dementia.

Similar to this thesis, the project explores the idea of a mix between generative art and human intervention. By defining a playground that the generative algorithm can manipulate, new artworks are created. The process is defined in the way that a user can participate in creating new artwork by making a decision. By shaping the playground in a different way the resulting artwork will differ.

Figure 13: "West" (Musical painting #2): Visual artwork by Roberto Bono, "Porta Ossuna" - Music by Andrea Alberti

4.2 vi.sion mixing senses

In the audio visual art project “vi.sion mixing senses”[4] von Benjamin Doubali, Leon Fuchs and Guido Schmidt the question “What does music actually look like?” is explored. In their Gallery the project features 3 generative artwork that does try to answer this question. The first Artwork “dot”[1] lets the user choose to input audio via the system default microphone, an uploaded media file or an predefined musical piece. Based on this music the logo of the Project is generated as it can be seen in Figure 12. The next artwork[2] is playing back five samples and lets the user choose for each sample wether it represents a color, emotion or shape. Based on the user input the artwork is created containing all the user selected properties. Furthermore all samples are played simultaneously to create a new track. The artwork becomes interactive when the music is played back as it is possible to be rotated in a 3D room while it changes shape and color based on the music.

back. It is also possible ...

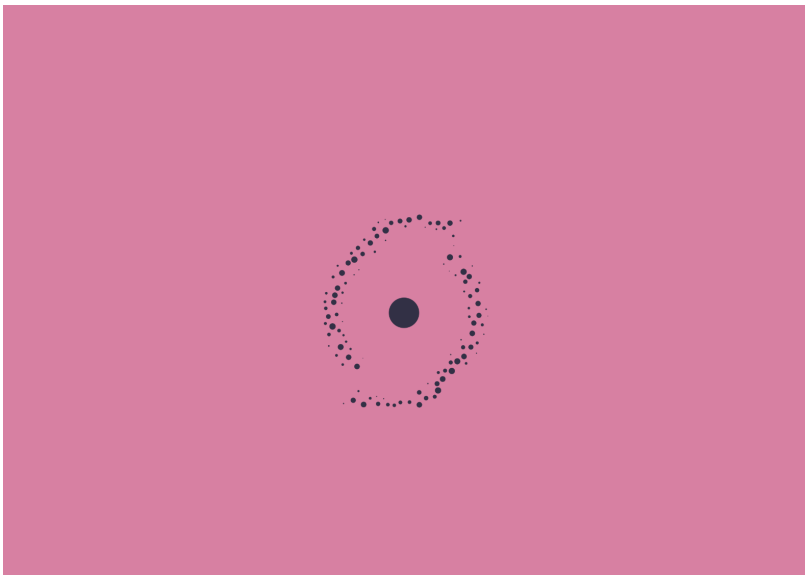


Figure 14: Music visualisation Dot

[1] https://dot.mixing-senses.art/
[2] https://audiovisio.mixing-senses.art/totem

The final and last artwork named “patterns” [3] is a interactive audio visual experience that runs in the web just like the other pieces. The user is able to define a curve in a 3D room. This curve can be saved and used to create the Artwork with. As it can be seen in Figure 13, this curve is used to spawn random shapes with random colors around. These shapes and color are generated when the music progresses. The music is split into 5 Stems. Each stem can be played back separately, by muting one the artwork is evolving differently since the input data is not the same anymore.

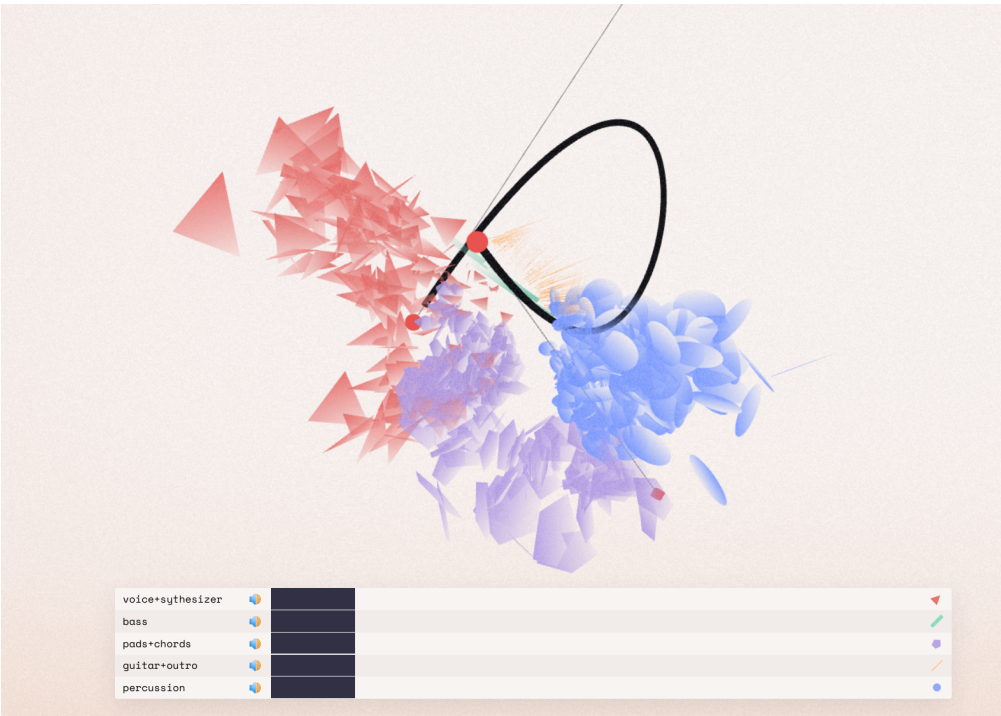


Figure 15: Music visualisation Dot

[3] https://patterns.mixing-senses.art/

5 Requirements

In the following, the requirements analysis will be presented. Since this application can be split into two main parts, the audio analysis, and the rendering, each part comes with its requirements, which do not apply vice versa. In 1, the functional requirements are laid out and discussed. These requirements feature that the application **must** fulfill to be called successful. In 2, the non-functional requirements are described. The requirements are not needed to prove the application's functionality but to enhance the system. As stated, this application will create artworks based on events that are extracted from an incoming audio signal.

use chapter numbers ins

5.1 Functional Requirements

5.1.1 Audio

5.1.1.1 Input The application must be able to recieve input Data in form of an audio signal. The application must be able to select the default audio device.

5.1.1.2 Analysis The application must be able to analyse the incoming audio and convert it to a machine readable format. The application must create a audio buffer. The sample size of the buffer can be specified.

5.1.1.3 Conversion The application must be able to extract following features:

- onset detection
- short break detection
- long break detection
- loudness

The application must feature an event system that dispatches event when the feature is extracted.

5.1.2 Rendering

5.1.2.1 Render artwork This applicaiton must be able to render a artwork consisting of 5 Layers. Each layer can hold various GameObjects.

5.1.2.2 Compute new Artworks This application must be able to compute new artworks without an audio input signal. When an input audio signal is received the scene becomes responsive and starts to regenerate artworks.

5.1.2.3 Effects This application must feature effects like a camera shake and transitions.

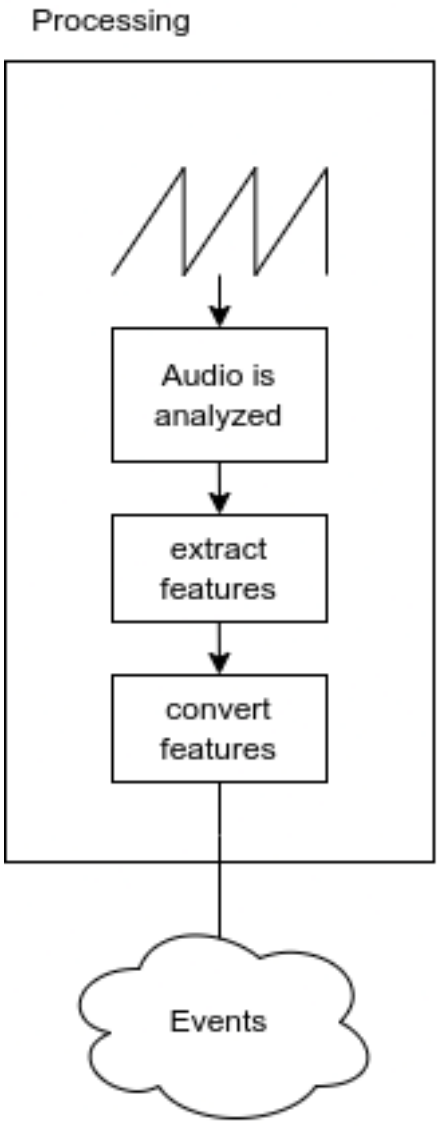


Figure 16: diagram showing audio signal flow

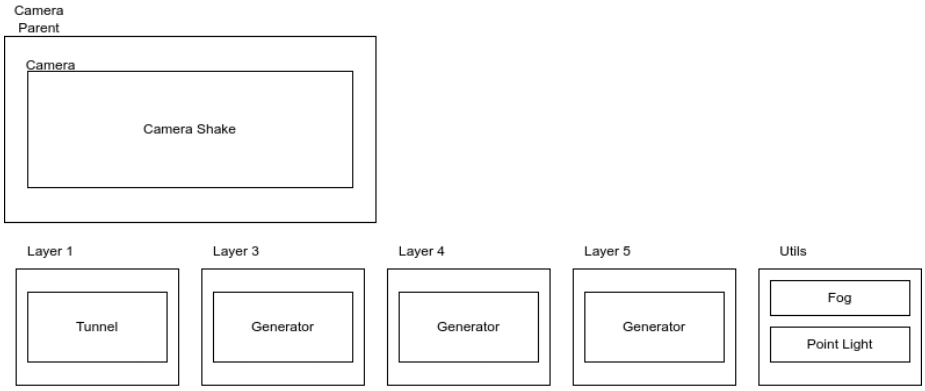


Figure 17: diagram showing the rendering layers

5.2 Non-functional Requirements

5.2.1 Accesibility

In order to provide this application to a bigger audience, this system should be accessible. As a consequence, this application should be accessible from as many operating systems as possible.

5.2.2 Variance

The system shall create generations of artworks that are well distinguishable from each other. Therefore it is helpful to create an unpredictable approach. On the other hand, it is important that the approach is not completely random because the system shall be able to generate visually pleasing artwork. To generate a visually pleasing artwork, there needs to be some supervision involved. By using many different models and Generators, it is possible to create artworks with a higher variance.

5.2.3 Extensibility

It should be easy for another artist to add a function to the primitive set; for example, a new type of Generator could be added, or a new lighting system might be implemented. This is desirable because this allows fast testing of new ideas.

5.2.4 Loose coupling

There are some steps that are necessary for any 3D application. Mainly the rendering pipeline and setting up the scene graph, and so on. Therefore the components shall be coupled loosely so that parts of this application can be reused in other projects

6 Design and implementation

6.1 Tech Stack

6.1.1 Unity3D

Unity is a cross-platform game engine developed by Unity Technologies and developed video games for PC, consoles, mobile devices, and websites. Games can be developed using C#, JavaScript, and the Unity Scripting API. Unity has been used to develop games in various genres, including RPGs, platformers, fighting games, and MOBAs. Unity Technologies likens the design of the Unity engine to that of standard movie production software – the director, cinematographer, actors, lighting technician can work independently of each other but still create a unified product, in this case, a video game. Unity was initially developed by David Helgason, creator of the game "The Crystal Hammer," as a game engine for his company, Over the Edge, in 2004. Helgason was joined by two other co-founders, Nicholas Francis and Joachim Ante; as they were working on another project, "ToyWars," they found they needed a game engine. Unity 1.0 was released in 2005. Unity Technologies' original slogan was "Developed by gamers, for gamers ."The company changed the slogan to "Develop once, deploy everywhere" in 2013.

While Unity is used chiefly for Game Development, it shows great potential for Animation Films and Art projects due to its flexible nature. In this paper, Unity was chosen because it features excellent potential to build a performant cross-platform application with access to the System I/O devices. The framework makes creating easy since the Model can be visually previewed and altered inside the Unity editor while the application logic can be written in C#.

6.1.2 C Sharp

C# is a programming language that runs on the .NET Framework. It is a multi-paradigm programming language that supports structured, imperative, functional, generic, object-oriented, and component-oriented programming styles. C# is an object-oriented language supporting encapsulation, inheritance, polymorphism, multi-threading, operator overloading, and exception handling. Unlike C++, the C# language does not support multiple inheritances. The C# language is syntactically similar to C++ and Java. Unlike Java, C# has explicit support for the Object-Oriented Paradigm. However, C# is statically typed and does not require the explicit declaration of variables. The .NET Framework provides a managed execution environment called the Common Language Runtime (CLR), which executes common language runtime bytecode that has been compiled from C# source code. The CLR provides memory management, type safety, exception handling, garbage collection, security, and thread management. In this thesis, C# was used to implement the logic for the generators, the audio processor, and the camera scripts.

6.1.3 P5.js

P5.js is a javascript framework and the successor to the famous processing framework [5]. Casey Reas and Ben Fry developed it. The framework is based on the same concept as Processing, an open-source language built on Java. The framework provides a Javascript API for writing code for the web browser. The code is written in the Processing language, based on a java-like syntax. P5.js allows us to draw different primitives, create images, videos and audio, text, shapes, and 3D models. The framework also provides a library for the browser and mobile devices. This library contains the core processing functions like the mouse, keyboard, color, and built-in functions. P5.js is often used to create generative artworks since the framework offers superb flexibility while easy to use. The framework is based on the p5 core. The core is a library that contains all the functions of the framework. The core is available for all platforms since it is written in plain Javascript. It is a simple library with less than 8000 lines of code. P5 core is written to enable it to be easily integrated into other projects. The core is powered by another library called p5.dom. The dom can be used to manipulate the webpage. By using this library, elements on a webpage can be manipulated. We can create elements like objects, images, and shapes or change the web page. The framework also uses the p5.sound library. This library contains functions for playing music and sound effects. The library is based on the html5 audio API. The library contains predefined functions for playing sounds and music. Using the web audio API makes it possible to get input from the system's default microphone. P5 does feature to apply a Fast Fourier Transformation to convert the time-based input audio to the frequency domain.

while being easy to use

6.1.4 Gimp

GIMP or the Gnu Image Manipulation Program is a freely distributed software for such tasks as photo retouching, image composition, and image authoring. It works on many operating systems in many languages. It is freely available to anyone and is distributed under the GNU General Public License. It can be used as a simple paint program, an expert quality photo retouching program, an online batch processing system, a mass production image renderer, an image format converter, and much more. It is expandable and extensible. It is designed to be augmented with plug-ins and extensions to do just about anything. The advanced scripting interface allows everything from the most straightforward task to the most complex image manipulation procedures to be easily scripted. Gimp can also be used to generate Textures and patterns. This feature alone makes GIMP an essential tool for any Artist. In this Thesis, GIMP is mainly used to create textures and fine-tune generated Images to suit the overall visual experience better.

break after Gimp

6.2 Input Data

In order to regenerate artworks, this application does require an Audio Signal. While every audio signal can be used, the system was trained on musical pieces.

In contrast to speech, music does feature a broad frequency spectrum that contains more information than recorded speech. Furthermore, to prove the concept, this application is specifically trained on the *Trance* Music Genre, consisting of a full low end and crisp highs. This genre has been chosen because it features a deterministic song structure. Moreover, this genre's Tracks can be easily mixed, creating a seemingly endless musical piece. The following figure shows a typical song structure for a modern trance track.

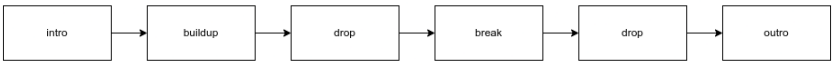


Figure 18: Song structure of a modern Trance Song

The song structure does indicate brakes and breakdowns. This genre usually features a quiet intro followed by a clean kick and bass combination. This setup builds up until the drop is reached after a short break. These short brakes are usually 4 to 8 bars long and are resolved by a sudden spike in the low end. Using digital audio allows for a near-lossless signal to pass through to the application and only contains non-audible noise. Theoretically, it is possible to use any input data, whether white noise, alternative rock from the 80's or amplified transistor noises captured by the digital-analog converter. However, bass-heavy music with a clear track structure provides enough information for the system to function correctly. This system has been tested with normalized audio, which results in minimal change in loudness. The musical pieces have been shuffled to provide a maximal range of variance while staying true to the genre. These musical mixes calibrate the input to prevent overfitting since they contain different energy levels, track arrangements, and harmonics.

Figure 19: waveform of a modern trance song

6.3 Project Setup

6.3.1 Hardware

There are several ways to input audio into the application. Audio interfaces made for music production and recording feature an excellent input audio quality. Microphones are a great way to record voices and go on zoom calls. However, the standard built-in microphone is optimized to record speech, not high-quality music.

In this paper, the Denon x1600 Mixer filters the audio signal before it reaches the audio interface. Using the hardware knobs, this mixer can filter out the mid, highs, and lows. Additionally, the mixer applies a -3db headroom to the audio signal to reduce the distortion effects. Finally, this audio signal is routed through 2 RCA cables to the Audio DJ 4 interface. This interface can pick up an analog signal and transform it into a digital one. A so-called Digital Analog Converter (DAC) works by converting a digital signal into an analog one. This interface can pick up a more comprehensive range of analog signals. This interface is superior to the simple ADC since it can make a wide range of analog signals digital. Within this interface, there are mainly two kinds of interfaces. The first one is the ADC. The ADC is an interface that can pick up an analog signal and turn it into a digital one. The second one is the DAC. It is an interface that can pick up digital signals and turn them into analog ones. In this thesis, the ADC converts an analog signal to a digital one. This setup allows for triggering certain events in the application which are based on a particular frequency spectrum.

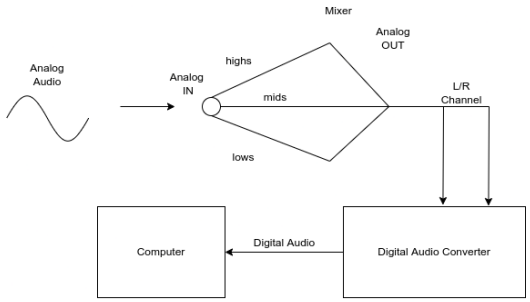


Figure 20: Visualizing the hardware setup

6.3.2 Native Application

The primary use case of a native application will be to provide performance optimizations and easy access to the GPU and the peripherals. The application can be executed on every x86 platform while providing similar features. Using Unity3D allows for GPU optimizations. This way, the artwork can be rendered on the GPU while keeping the analysis and data translation to the CPU. This approach will spread the workload between computational systems and allow higher performances.

Unity3D does feature its build process, which allows compiling the application for a wide variety of platforms. This application targets the desktop operating systems, mainly GNU/ Linux, Windows, and macOS, and a native version can be compiled for each system. To build a native application for a desktop, Unity first compiles the C# code to an intermediate language. This language is used alongside the metadata to make up the byte code. The mono runtime can execute that bytecode and interpret it on each platform accordingly.

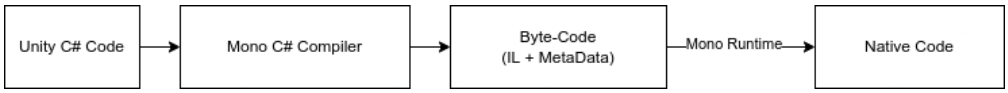


Figure 21: Overview of the Unity build process ,IL = Intermediate Language

6.3.3 Architecture

This system is split into 2 Parts: the Processing and the Generative Part. The processing mainly consists of an AudioPeer class responsible for analyzing the audio and saving the current incoming stream to a list. The Feature Extraction / Event System picks up that list and extracts audio features. See 5.4.3. When the features are successfully detected, this class does send out an Event. These classes are added to the Scene by attaching them to empty GameObjects.

The Generative Part consists of 5 Layers, a camera, and some Utils. Each layer does contain a Generator that can instantiate a part of the artwork, see 5.4.5. The utils help create Transitions while the camera provides a viewport for the Scene. The following chapters go into more detail about each Part.

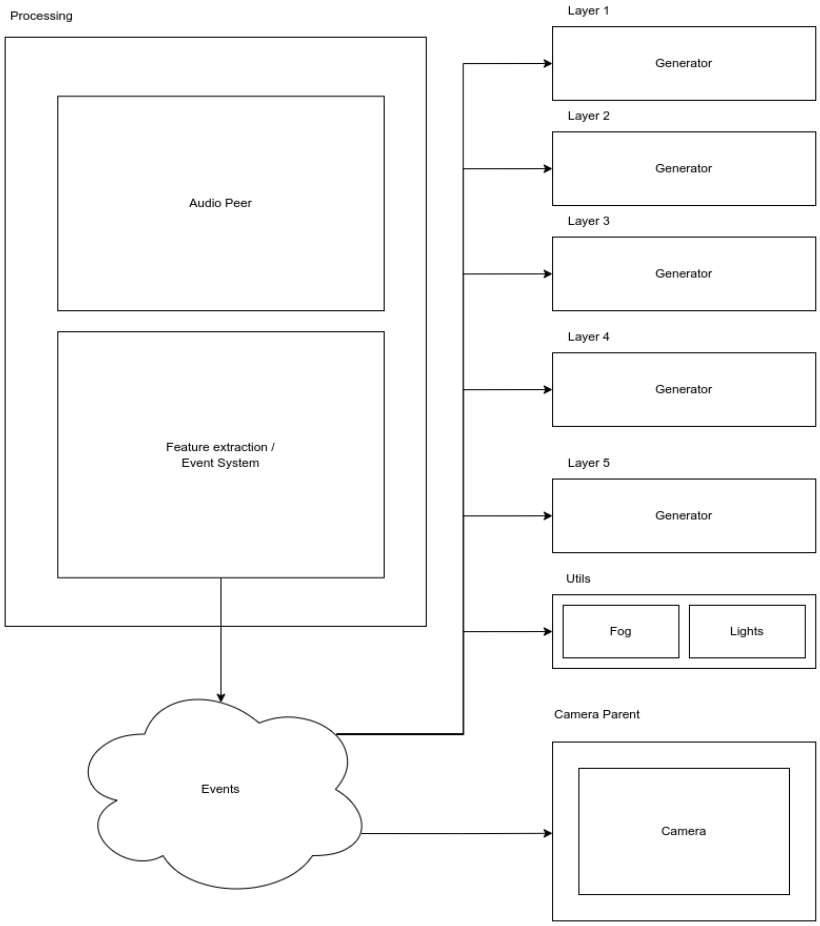


Figure 22: Diagram showing the architecture

page break

6.4 Components

This application consists of 2 Parts, the Audio Analysis, and the rendering. These parts build a chain where firstly the Audio Signal is analyzed, then translated into events, and finally, based on these Events, the artworks are rendered.

6.4.1 Audio Analysis

A list of all Input devices can be obtained using Unity3D. Usually, the Operation System does flag one input as the primary one. The application will start to

listen to input data using the primary input. The data is analyzed with the fast Fourier transform algorithm using the Blackman window function.

The information inside an Audio Spectrum is not equally split up. Instead of following an equal distribution, it follows an unequal distribution. This means that there is much more information between 20 - 60hz than between 10000 - 10040hz. The lowest channels would feature the bass, while the higher channels feature the treble. Since the audio signal does have 20000hz individual values, it is best to dumb down the values, usually into six channels. For further analysis the Input data will be devided into 6 Channels. Based on this pattern:

20 - 60hz (channel 1) 60 - 250hz (channel 2) 500-2000hz (channel 3) 2000-4000hz (channel 4) 4000-6000hz (channel 5) 6000-20000hz (channel 6)

Adding to this, there is also an average of all channels computed that will be holding the amplitude. Since we are listening to non-normalized input data, there is no way to know how high the values will be. Dealing with a quiet Signal will result in low values, while loud signals result in high values. To deal with that uncertainty, a normalization process is used. To normalize the values, the frequency value of each channel is divided by the highest recorded frequency for that specific channel. This will result in normalized values between 0 and 1. To adapt to specific loudness changes, the highest frequency will be constantly decreased by a factor of 0.005. This is always used to re-evaluate the highest frequency and quickly adapt to loudness changes.

To smooth out the values for each channel, there is a smoothing process when the values are decreased. Instead of abruptly jumping to a low value, it is decreased gradually. This effect can be seen when visualizing each channel. The animation is less jumpy and more smoothed out.

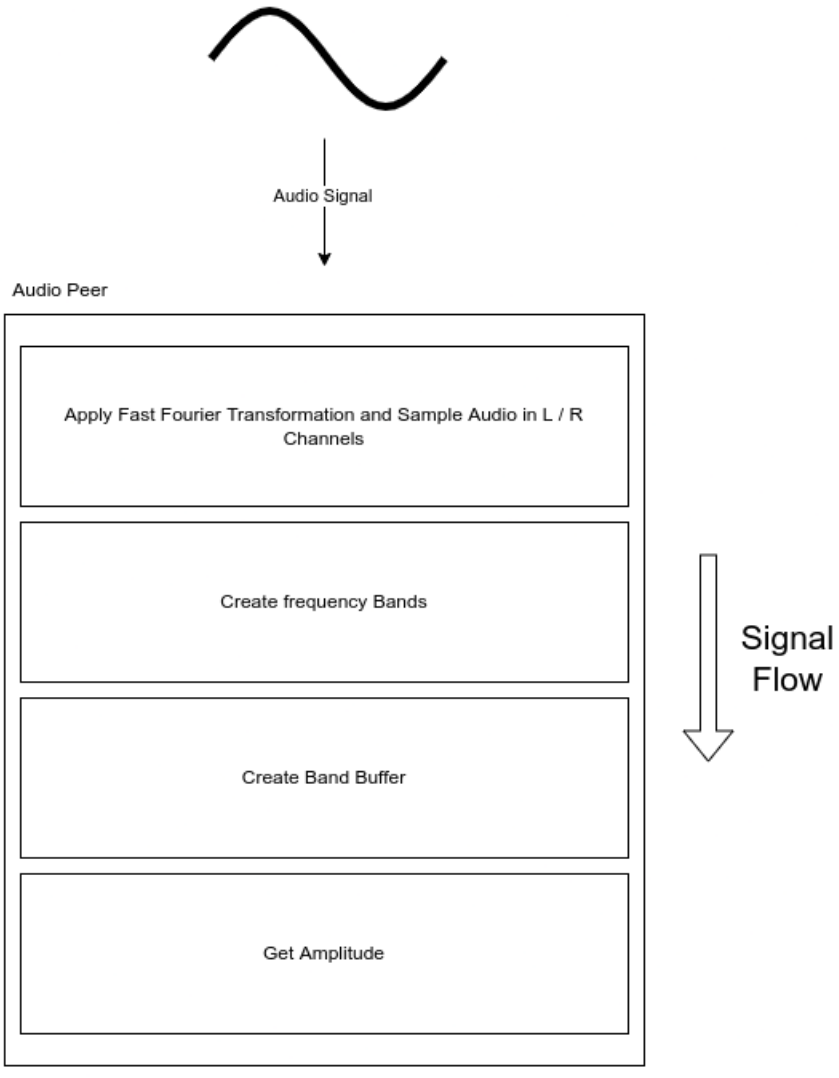


Figure 23: Audio Analysis Signal flow

6.4.2 Audio Feature extraction

Computed features give a rough understanding of the Audio signal. Two kinds of audio features can be extracted. These include energy levels, Frequency domain, and spectral spread. These features are computed directly from the audio signal's frequency and work with only one snapshot of the frequency. Furthermore, these values are not normalized by default, making them unreliable to analyze real-time audio, which is prone to change in tonality, amplitude, and energy.

6.4.2.1 Onset detection The onset detection feature is extracted using an algorithm on a series of frequencies and cannot apply to a single frequency snapshot. Each incoming frequency snapshot needs to be buffered to create a series of frequencies. This buffer does contain the last 512 frequencies snapshots of each channel.

This enables one of the most used features in this project called onset detection. The onset detection algorithm works by comparing the buffer to the current audio signal and evaluates if a sudden change in amplitude occurs. This algorithm can be fine-tuned to detect big spikes in audio that likely appear after long breaks or to listen to subtle changes, for example, the beating of a drum.

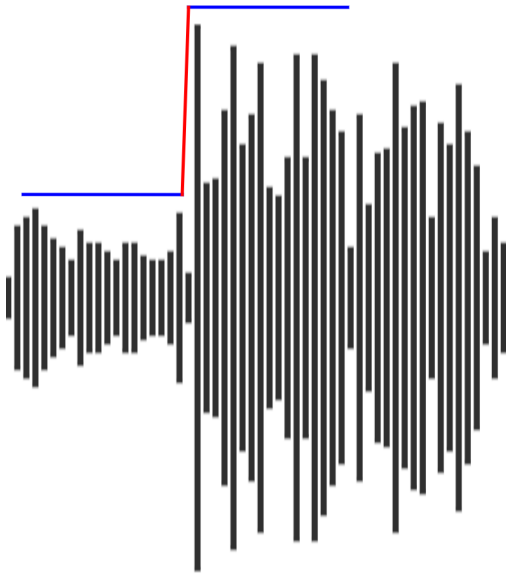


Figure 24: visualizing onset detection. RED=onset BLUE=normal audio

6.4.2.2 Break detection One of the features implemented was the break detection algorithm. The break detection can differentiate between long and short breaks. These detection algorithms are specifically tailored to the chosen genre of music used since they use a reference to compare the current running audio signal against. A break is indicated by a low low-end and little energy in the treble. To detect a break, the detection algorithm looks at the first channel of the deconstructed audio to compare the average of the buffered values against the reference values. To classify for a long break, the average of the first channel is not allowed to reach the threshold within a given timeframe. The algorithm looks at the audio signal three times to save processing power, each one second apart. Each iteration of the break detection algorithm, the previous state of the run is saved. When the average of the first channel is not able to cross the threshold on three consecutive runs of the detection algorithm, a long break-ready event will be triggered. This long break-ready event indicates that the song is currently in a long break phase. With the subsequent low-end onset, this phase will end, and the saved values used for the long break detection algorithm are cleared again.

The values of the first channel are compared against reference values which are defined in the application. These reference values contain averages of the first channel of various reference songs. The reference songs have been chosen to feature a range of different audio characteristics. These songs have been analyzed and converted into a machine-readable format based on tonality, spectral spread, and intensity. The average of the first channel of each reference track is computed and concatenated into a list to serve as a threshold reference for the real-time audio.

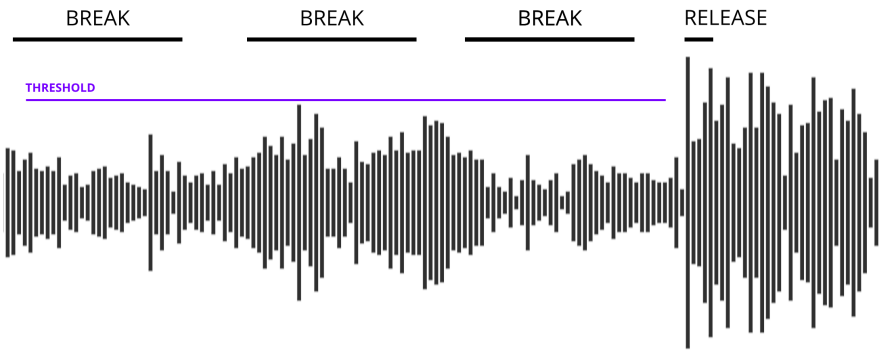


Figure 25: long break detection algorithm visualized

To detect a short break, the same mechanism is used. The only difference is that only one is used instead of using three saved values to detect a break. The onset of the bass after one successful iteration of the breaking algorithm indicates a short break. Afterward, the saved values will also be cleared.

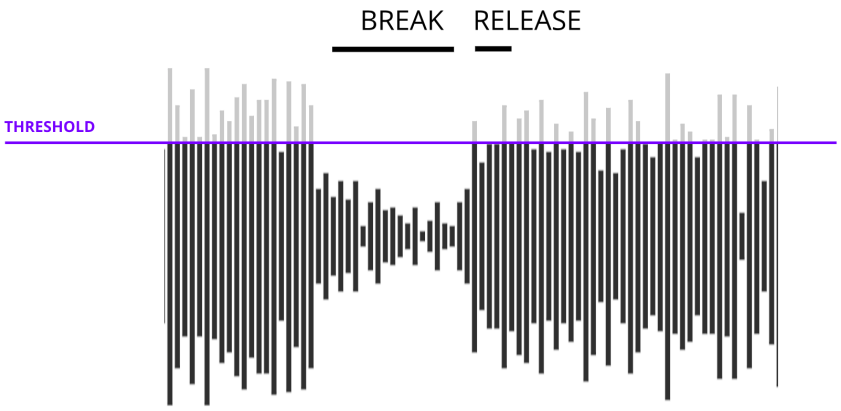


Figure 26: short break detection algorithm visualized

6.4.3 Feature conversion

To use the analyzed features, they need to be converted into a machine-readable form. Unity3D offers an event system that lets the application send events and make them globally accessible. These events can contain pieces of data or extra information. Usually, an event consists of a name and adds some data. Using an event system provides an easy way to communicate data between the components and allows for loose coupling. The event system acts as a middleware between the audio analysis and the graphical user interface by providing an accessible API. In contrast to modern web development, the event system dispatches events globally, and the client subscribes to them instead of calling the event system for updates.

Having an event system also allows for easy extensibility in the form of plugins. The analyzed audio could also be translated into midi data, text input, or raw pixels—more on this in the Outlook section.

use number

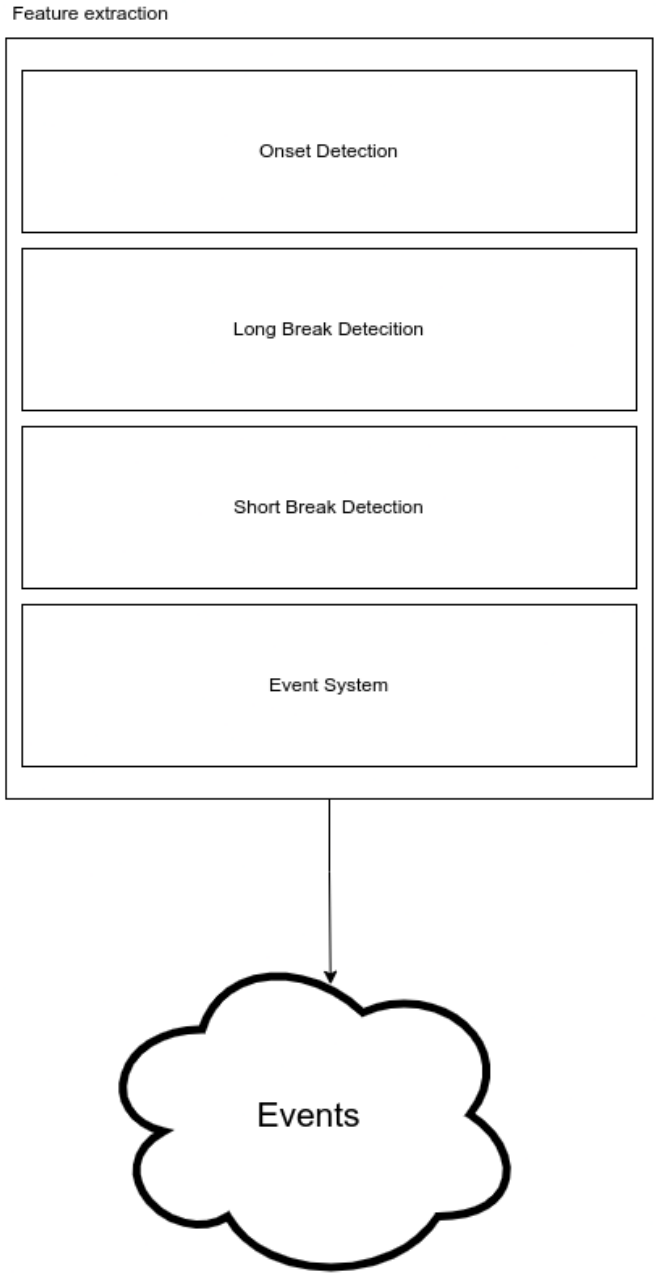


Figure 27: Showing the feature extraction workflow

6.4.4 Scene setup

The scene is made up out of 5 Layers, a processing unit, a camera and lights.

The Camera is wrapped with another component which can be controlled individually. The processing unit does provide the AudioPeer which acts as the AudioAnalyser and the EventSystem which converts the audio featuers into events and sends them accordingly. Each inidividual layer does contain a Script which dynamically loads a Model or Visual Effect. These Scripts are listening for Events and modify there parameters accordingly. Finally there are some global lights to illuminate the scene.

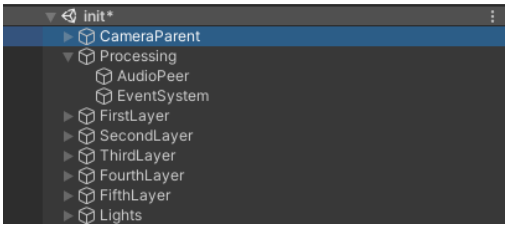


Figure 28: Unity3D Scene Hierachy

6.4.5 Generators

Each layer does feature its own Generator Script. This setup makes sure that each layer is rendered in the right position and does feature the right assets and models. In general there are 5 Layers each assigned to its own Generators. The Generator Script in itself is simple and generic. It exposes a list which can be filled in the Unity Editor UI with GameObjects. When initialized one random GameObject from that list is choosen and displayed. While its technically possible to assign every GameObject to a certain Generator it is advised to reserve the upper Layers for smaller Objects which can be easily layered above the rest. The Layers are stacked across the Z axis to minize clipping.

initialized, ...

Here is the code for the Generator Script:

```
void AddRandomGameObject()
{
    var r = Random.Range(0, GameObjects.Length);
    var tp = transform.position;
    var position = new Vector3(tp.x,tp.y, tp.z)
    GameObject gb = GameObjects[r];
    var randomObj = GameObject.Instantiate(gb, position, Quaternion.identity);
    randomObj.transform.parent = gameObject.transform;
}
```

6.4.6 Tunnel

The so called tunnel is one of the main elements in the scene. It builds the base of the scene by providing an ever changing background. In itself it layers 4 Textures and moves them forward or backwards following a cylindrical shape. It features properties like movement speed, rotation and twist per each layer. One of the most used feature is the dynamic texture assignment via an API. While it is possible to control each layer on its own, the global control does unify most of them. Similar to the properties on each layer there is a global speed and rotation next to a global brightness.

On the application start it slowly gets brighter. This is due to the fact to hide the adjusting phase of the audio analysis. When starting out the maximum frequency could not be determined yet thus the normalized values do not take the whole spectrum into account. This usually shows itself in really shaky values which tend to jump between high and low.

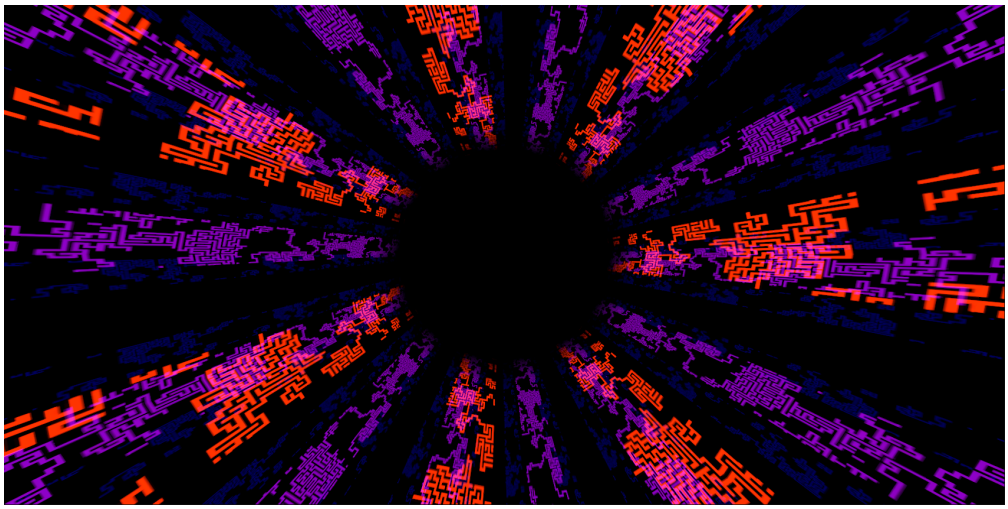


Figure 29: Screenshot of the background tunnel

6.4.7 Fractals

The mandalas are used as a secondary visual. These are meant to be layered on top of Layer 1 and act as a backdrop for the rest of the scene. These mandalas are generated with p5.js and included in the final project.

p5.js is free and open-source javascript library that runs on the web. Generally p5 is used for experimental web projects, data visualizations and generative art.

This is the code to generate the mandalas

```
const makeMandala = (position) => {
  const layers = layerConstructors.map((lcon) => {
    let picker = random(1);
    const draw = picker > lcon.weight;
    return lcon.init({
      position,
      draw,
    });
  });
  return layers;
};
```

This makeMandala function does generate a random number between 0 and 1. This value is then compared against a weight. This weight defines the likely hood that a specific layer is choosen. This is controlled randomness

Each layer constructor does contain a name, init and weight field. The name does make the debugging process easier by providing a human readable identifier for the drawn shape. The init function does contain the actual computation of the layers. This can be the drawing of a hexagon, a line or a group of squares. The weight indicates the likely hood that this particular shape is choosen as explained above.

Here is the code for a basic shape that either draws a hexagon, a circle or square background.

```
const layerConstructors = [
  {
    name: "Shape",
    init: (props) =>
      outlineShape({
        ...props,
        ...setState(state),
      }),
    weight: 0.3,
  },
];
```

This relatively simple code is able to generate an endless amount of fractals with

a given style. This style is defined by the weight of each layer, the init function and the color palette. This generative system is used to generate the assets for another generative system. In order to save on performance these visuals are rendered before hand and are saved as an .png. Displaying an Image does come with a much lower performance overhead then creating the image in realtime.

Each of these layers is rendered with a transparent background and does contain only black and white colors. This allows to easily re color the fractals in the final application to match the general theme better.

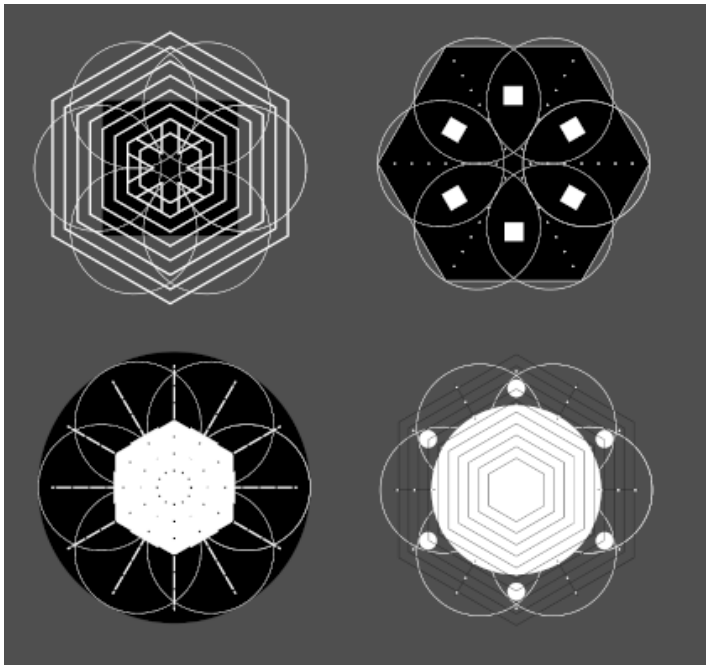


Figure 30: rendering of the isolated fractals

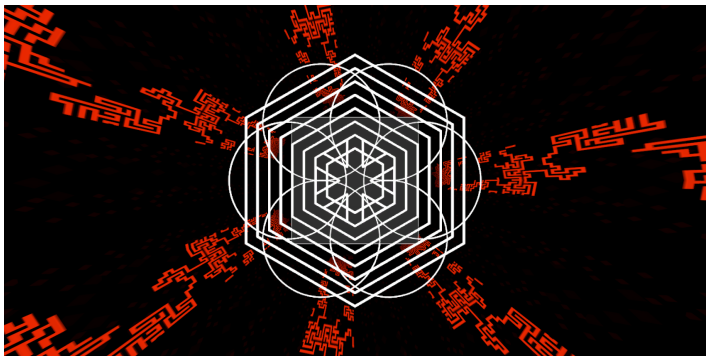


Figure 31: layering the fractals with the background

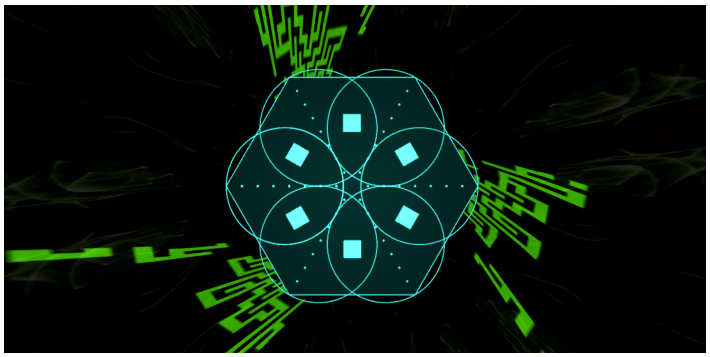


Figure 32: regenerating the layers and background with the background

6.4.8 Lighting

One exception to the workflow of generating the layer contents are the lighting systems. It is required to have atleast on Light Source in the scene to be able to illuminate the models. This Lights are pre backed to the scene and thus live in its own GameObject. A Point Light is used for the illumination since it provides a Radius, Intensity and additionally an parameter to control the color. Having one global light does simplify the scene setup since its possible to fine tune more easily to the scene. Also when transitioning between different Artworks this light can be used as a transition effect to darken the whole scene to allow for a smooth transition.

used

The color of the light is used to give the scene a certain look. Every model that does contain a lot of white textures or colors will get tinted by the light color. This could be use to match the background or to paint every model in a complimentary color compared to the background.

6.4.9 Particle System

In the base example there are 2 Particle Systems used. These System can easily co exist on the same layer since they add a small visual footprint. The build in Unity Particle engine is really powerful and allows for all kind of visual particle effects. Each particle contains a 2D Texture which can be randomly allocated. Also speed, direction and gravity can be adjusted. For sake of simplicity the Particle System will not be created in a generative manner but will be provided as an Unity Prefab instead. This allows to fine tune the behavior of the system without adding much complexity.

Unity build in particle System is really performant out of the box already so there are not many optimizations that can be done. One factor that has the most impact on the performance of the System is the Particle Count or the Spawn rate. The Spawn rate indicates how many particle are spanned in a second, afters its initial spawn process these particles then follow there own path which is generated by the parent system. Another optimization is to only generate Particles that may be visible in the viewport.

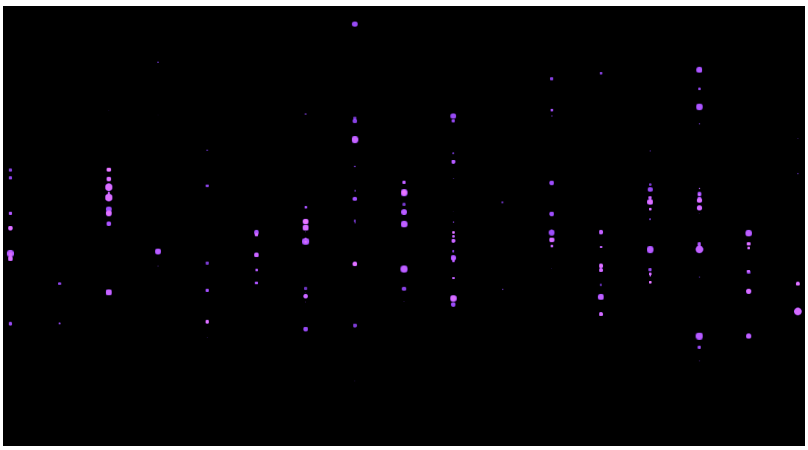


Figure 33: regenerating the layers and background with the background

6.4.10 Generative Camera movements

One common technique to generate realistic Camera movements is to use the Perlin noise to generate a random value for the x, y and z axis intensity over time. For each axis one perlin noise is used that generates smooth values which are then normalised and finally applied to the corresponding axis. To control the intensity each axis does expose a intensity property which multiplies the the normalised value to strengthen or weaken the effect of the perlin noise.

Camera shake is widely used effect to emphasis certain events. Mainly used in the Gaming and Video industry this effect can also be adapted to our project. Paired with the onset detection after a long break this emphasis the Artwork and gives and dynamic approach to it. When a Camera Shake Event is emitted the Camera Processor picks it up. To shake the camera the intensity of the camera movement script is quickly increased on decreased. This leads to a camera shake effect.

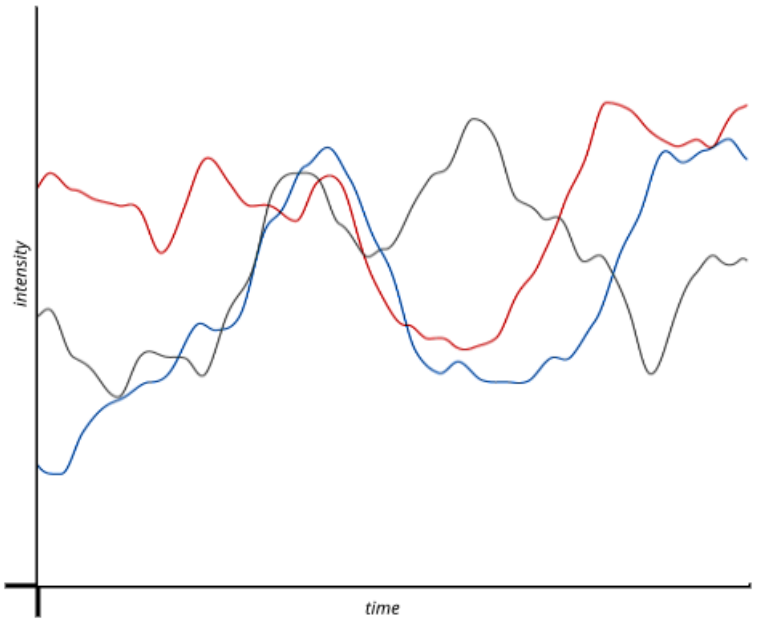


Figure 34: 2D Graph showcasing X=RED, Y=BLUE, Z=BLACK

6.4.11 Transitions

To be able to regenerate Artworks in a visually pleasing and smooth way theref is a transition Event. This event is triggered when a long break occurs without an immediate onset afterwards. The transition event sets a lock with a cooldown to the event trigger. Having a lock in place limits the Event from getting spammed during really long breaks where the long break event might be triggered multiple times. The cooldown is 2 minutes or 120 seconds long, according to the song structure this results in one Artwork change per Song.

Visually one Transition effect is achieved by adding fog to the scene where the intensity and thickness is increased or decreased. This darkens the scene and allows each layer to regenerate its content. Another transition is achieved by rotating the camera upwards and back again just in time to see a new artwork.

6.4.12 Examples

This section showcases some artworks that have been generated with this system, note that these are still images and do not show the movement of the scene which is a important part of the artworks.

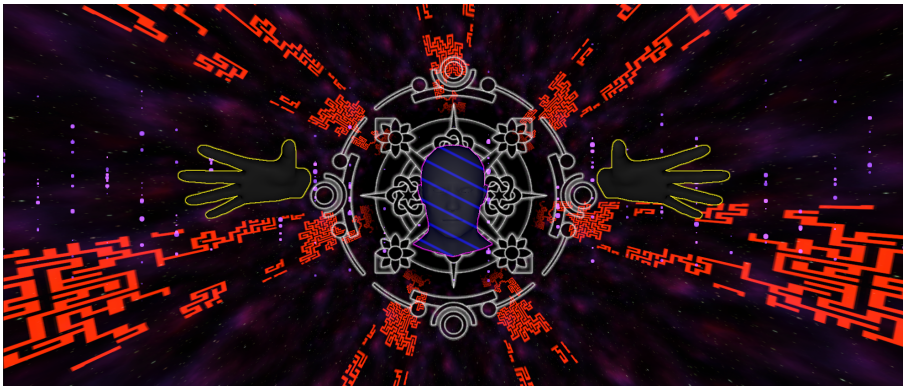


Figure 35: Fully rendered scene

6.5 Conclusion

The work discussed in this chapter is intended to fulfill the requirements set in chapter 4. The first step is to limit the input data to a specific genre of music. In 5.1.1, the project setup is discussed so that the application receives normalized audio levels to function correctly and make this system reproducible. This is the groundwork for the rest of the application to build on and to have the audio analysis work predictably. This chapter continues to introduce the components which this system consists of. Each of those components does fulfill one functional requirement set out it 4. In addition, it is shown that by choosing to build a native application with unity3d, the non-functional requirement of cross-platform compatibility is possible. Furthermore, the implementation of the Layers shows that the application can fulfill the requirement in 4.1.2 that is to render artworks, create effects and compute new artworks. Moreover, the introduced scene graph does allow to extend the system further without much need of programming skills and fulfills the non-functional requirement: extensibility from 4.2.3. The non-functional requirement to create an artwork with a high level of variance from 4.2.2. is partly fulfilled. In this implementation, only a small set of Models and Textures was used, leading to a small variance level. The artist would need to add more models and textures to add more variance to the system. Since time was running out during the end of the project, the non-functional requirement of loose coupling could not be fulfilled. The application follows the monolithic approach and is tightly coupled. The reason for this is some unexpected pitfalls which are discussed in 5.4. An outlook on what yet has to be implemented is given in chapter 6.1

6.6 Lessons Learned

In the following, the pitfalls that emerged during the design and implementation will be presented. Although creating a proof of concept that can fulfill the functional requirements was reached, it consumed significantly more time than estimated. Firstly the onset detection which is discussed in 5.6.1. Secondly the generative approach to create textures will be discussed in 5.4.2.

6.6.1 Onset Detection

This Thesis explores the extraction of specific features from an audio signal. The onset detection was one feature that required more work then expected. To detect the onset of a song a series of frequency snapshots needed to be analyzed. These series of snapshot have been compared to a previous saved reference snapshot. This reference data was collected by developing and running a special state to the application that is able to save these frequency snapshots with a press of a button. To select convincing reference data the incoming music has been chosen carefully to feature the characteristic that are needed, mainly a break followed by a drop. These reference frequencies have been averaged and put into a reference buffer. This buffer does contains the frequency response samples. The real-time audio signal has been compared to this frequency time snapshot if show a certain pattern. If yes the onset event has been triggered. This feature did took longer then expected since a lot of pre work was needed to get it to function properly. This feature also led to the decision to use normalized audio when trying to extract features of different audio tracks.

to see if it matches a certain pattern

6.6.2 Generative Texture generation

The fractal textures that have been generated in 5.4.7 are created with P5.js and imported into Unity3D. The Texture is applied to a material that respects the alpha channel of the Fractral Texture which is finally applied to a plane object. Each Fractal consists of many different shapes and patterns, while the color palette is kept with only black or white the rest of the artwork is created randomly. Each shape does contain a weight, the weight determines the likelyhood of this shape being selected. Each shape is created individually by creating a combination of loops and conditions. To get a visually pleasing endresult there is a amount of shapes required. The generated image is not supposed to be to simplistic or to complex. To find the right balance there was a lot tweaking done in this project. When these fractals are generated they are exported as a .png and added to the asset folder in Unity.

7 Conclusions

This thesis explored how to control generative systems with audio signals and create a constant stream of real-time generated artworks. Therefore, the designed system performs well and creates visually pleasing, ever-changing artworks which are regenerated based on features extracted by the incoming audio signal. The artworks are composed of 3D objects, 2D textures, and animations.

Chapter 2 conveys an overview of generative art and its techniques. It starts with defining what generative art means in the context of this thesis. Afterward, the most common techniques are detailed and explained. Furthermore, the use cases of Generative art are summarized and explained by drawing examples from the Music, Video, and Gaming industry. The chapter finished by explaining the difference between algorithmic and generative art. The used system in this thesis aligns with the definition of Generative Art, not algorithmic art.

The next chapter gives an overview of the basics of Audio Analysis. The term 'Sampling,' the basics of the fast Fourier transformation, and its corresponding window functions are explained. The window function that is used for this bachelor thesis is also outlined. The first part of this chapter describes the techniques to analyze audio files. In contrast, the second part of the chapter goes into more detail about the features that can be extracted from the analyzed audio signal. Some of these features are used for the requirements analysis in chapter 4.

Based on these findings, a requirement analysis has been put together in chapter 4. The requirements are categorized into functional and non-functional requirements. In chapter 5, it is shown that these requirements have been implemented and fulfilled. In 5.1, the input data is discussed, and its general structure is showcased. The characteristics of the music genre are explained and broken down. 5.2.1 promises a performant and easy distributable approach by choosing and native application over a web application. Up to this point, the theoretical groundwork is done, and the project basics are set up. The chapter explains how the Audio is Analyzed, which frequency spectrum is chosen and why. The flow of the incoming signal is broken down and visualized. Afterward, the feature extraction is explained, and the essential features are shown in more detail. These features are converted into Events that are used throughout the application. The second part of the chapter details how the scene is set up and which layers construct a new artwork. The chapter goes into more detail about the used scripts and how they link together with the event system. Some example Assets are discussed to fulfill the requirement of a minimal viable working project. Up to this point, the system aims to implement a proof of concept that shows how the components can work together with each other in order to create artworks. As this thesis is published, the artworks will consist of the example assets shown in 5.2. Unfortunately, there was no time left to present more complex artworks and feature a wide range of different assets. Lastly, the chapter discusses the pros and cons of a monolithic approach and how they compare against the approach

taken in this thesis.

7.1 Improvements

The following improvements are discussed. These improvements come with a performance hit that needs to be critically evaluated if this is suitable for the given system.

7.1.1 Beat detection algorithm

As stated in 3.6, the time between the onsets in the lower frequency range can determine the BPM (Beats per Minute). To detect the BPM, the ms between 3 onsets is measured. By knowing the BPM, a grid can be fitted aligned with the onsets of the low frequency. The Grid allows to count the bars of the musical piece and to predict brakes before the frequency change has even indicated this. One approach could be to change the rerender of the artwork after 128 beats automatically.

7.1.2 Tonality detection

In 3.5.3, tonality is discussed; tonality is an indicator for the key of the musical piece. Each key conveys a different feeling to the listener. An interesting improvement would be extracting the key and mapping its value to a color. This color could then be used to control the lighting of the scene **[inproceedings?]**. The key can also manipulate the outcome of a texture generation algorithm.

7.1.3 Generated options

The system's generated options make up for a significant improvement to allow for more variance. Having more models and textures to choose from does make the application more interesting and worthwhile. Textures can be generated inside the application by modifying pixel values directly at the program start. A model can be generated too. By stacking and combing random meshes, a new model can emerge. To optimize for performance, these models are created on program start or are generated before the new scene is shown to minimize the perceived performance.

7.1.4 Transitions

In the base application, the Transition is always the same. With a quick fade to a dark screen, the new scene is rendered in the background and displayed when the fog subsides. This Transition does not change is not generated in any way but rather a static animation. This Transition can be generated. For example, the color fades to change every time a transition is initiated. Furthermore, camera effects are possible. To Transition, the camera could rotate and focus a new point in the space where the new artwork will be rendered.

7.1.5 Export MIDI

MIDI is the standard for communication between musical instruments and software. Modern lighting and laser systems use midi to communicate the mode they are operated in. This way, the technician can play the lights and lasers just like they would play an instrument. Virtually all modern keyboards, lighting systems, and recording equipment can be set up to communicate with a Computer via MIDI for years. In order to make use of that, this application could additionally export MIDI notes via an audio interface. This MIDI data can then be mapped to a musical instrument or a lighting system. The MIDI does contain the extracted audio features mapped to a virtual keyboard. Each Note does represent an event that has been converted from the audio signal. For example, an extended break release could map to the key of C.

Bibliography

1. Bracewell RN. The Fourier Transform. *Scientific American* 1989;**260**:86–95. https://www.scientificamerican.com/article/the-fourier-transform

2. Schnelle Fourier-Transformation (FFT) und Fensterfunktion - NI. https://www.ni.com/de-de/innovations/white-papers/06/understanding-ffts-and-windowing.html

3. Generative Art & Media Website. http://generativeart.net.dcs.hull.ac.uk/

4. Vi.son Exhibition. https://exhibition.mixing-senses.art/

5. Bohnacker H, Gross B, Laub J *et al. Generative Design: Visualize, Program, and Create with Processing.* New York: Princeton Architectural Press, 2012.