UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Maert Mägi

Interactive Music Visualization in Unreal Engine

Bachelor's Thesis (9 ECTS)

Supervisor: Madis Vasser, PhD

Tartu 2021

Interactive Music Visualization in Unreal Engine

Abstract:

This thesis explores music visualization as a whole and describes the general process of visualizing music. A module to streamline and simplify music visualization in Unreal Engine 4 Niagara was developed as a part of this thesis. The module can be used to synchronize objects in Unreal Engine with audio. The size, velocity, position, rotation, and color of the objects can be transformed by audio loudness in real-time. Along with the module, seventeen unique music visualizers were created. This thesis also contains the implementation details of the developed module and descriptions of each of the visualizers. The ease of use and functionality of the project were evaluated by conducting usability testing.

Keywords:

Music visualization, audio visualization, computer graphics, Unreal Engine, Niagara

CERCS: P170 Computer science, numerical analysis, systems, control

Interaktiivne muusika visualiseerimine mängumootoris Unreal

Lühikokkuvõte:

Käesolev bakalaureusetöö uurib muusika visualiseerimist tervikuna ning kirjeldab muusika visualiseerimise üldist protsessi. Töö raames loodi Unreal Engine 4 Niagara jaoks moodul, mis kiirendab ja lihtsustab muusika visualiseerimist antud mängumootoris. Mooduli abil on võimalik heliga sünkroniseerida Unreal Engine'i objekte. Helivaljuse abil on võimalik reaalajas muuta objektide suurust, kiirust, positsiooni, pöörlemist ja värvi. Lisaks moodulile loodi seitseteist unikaalset muusika visualiseerijat. Bakalaureusetöö sisaldab ka mooduli implementatsiooni üksikasju ning iga loodud visualiseerija kirjeldust. Projekti kasutuskergust ja funktsionaalsust hinnati kasutatavuse testi läbiviimisel.

Võtmesõnad:

Muusika visualiseerimine, heli visualiseerimine, arvutigraafika, Unreal Engine, Niagara

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1 Introduction

Music visualization is the process of representing music graphically. Historically, music notation has been used to visualize music, but due to its complexity it is not an effective means of conveying musical emotion to the average listener [1]. As such, various different forms of visualizing music have emerged and can be seen today in popular media player applications, like WinAmp[1], Apple iTunes[2], and Windows Media Player[3]. These visualizers use colorful abstract animations to visualize audio attributes as they move to the rhythm of a song [2].

In order to produce audiovisual animations, a mapping between the musical information and visual data must exist. Audio features, like frequency spectrum, amplitude, and pitch, have been reliably mapped to graphical properties, like size, position, brightness, visual repetitiveness, and texture granularity, for any arbitrary object in 2D or 3D space [2–4].

The purposes of any visualization usually fall into either technical or artistic categories [5]. Such is also the case with music visualization. Technical music visualizations are connected with more functional and analysis-oriented work, such as audiovisual therapy, music comprehension, and musical genre classification [6–8]. Artistic music visualizations, on the other hand, feature works, like live audiovisual performances, audio reactive artwork in museums, and providing musical experiences to people with hearing disabilities [9–12]. Sound driven animations can also be incorporated into video games to enhance the player experience.

The goal of this thesis was to create a flexible and reusable module that would assist artists, designers, and other enthusiasts in creating audio visualizations in the game engine Unreal Engine 4[4]. The game engine was chosen due to its graphical fidelity, interactivity, powerful audio engine, and the new visual effects system Niagara[5]. Seventeen unique visualizers were created to assist with the learning process of the module, but they can also be used as standalone visualizers.

Chapter 2 of the thesis focuses on the background and necessity of music visualization. Chapter 3 focuses on the design aspects of the visualizers. Chapter 4 goes in-depth about

[1] https://www.winamp.com/
[2] https://www.apple.com/ee/itunes/
[3] https://en.wikipedia.org/wiki/Windows_Media_Player
[4] https://www.unrealengine.com/en-US/
[5] https://docs.unrealengine.com/en-US/RenderingAndGraphics/Niagara/index.html

the implementation details and gives an overview of the architectures used. Chapter 5 provides the results and analysis of the conducted usability test.

The project's asset files and the feedback questionnaire are available in Accompanying Files (Appendix I). The project's repository is located in Repository (Appendix II). A video demo of the visualizers in action can be seen in Video Demo (Appendix III).

2 Background and Related Work

Music visualization has been around for a long time. In order to appropriately judge the necessity of music visualization, it is important to understand its background and purposes, which are described in this chapter. Subchapter 2.1 focuses on the history of music visualization. Subchapter 2.2 focuses on some of the applications of music visualizers, and subchapter 2.3 introduces some contemporary music visualizers and other works that influenced and inspired this thesis.

2.1 History

According to Fourney *et al.* [13], the viewing experience of visualized music should be similar to what one gets through sound. In addition to the lyrics, music visualizations must also convey the rhythm, melody, emotional content, and overall entertainment value to reach similar levels of visual enjoyment. Changes in the music's mood should also have perceivable differences in the visualizations.

Dances, paintings, and other kinds of visual arts have complemented music for thousands of years. Some of the earlier forms of audiovisual performances incorporating moving imagery include 18th century magic lantern shows, phantasmagoria, Javanese shadow puppet shows, and silent films with musical accompaniment [10].

Alexander *et al.* [9] mention that building on top of earlier attempts to visualize music, and with the rise of filmmaking in the early 20th century, the genre of abstract filmmaking, also known as visual music, was born. Films of this genre were largely made by abstract painters who were looking to bring the element of time into their works. Visual music films featured abstract animations often synchronized to music (see Figure 1), but as there did not exist a clear formula, the execution of the films were up to the artists' interpretation.

Figure 1. Still from Oskar Fischinger's *Optical Poem* [14].

One of the most famous examples of combining music and visuals on screen is the 1940 Disney film *Fantasia*, which uses abstract and literal art to communicate the music's emotion, entertainment, and other properties to the viewer [13]. The process of manually synchronizing animations to sound takes a lot of human effort, though, and is difficult to do in a live performance.

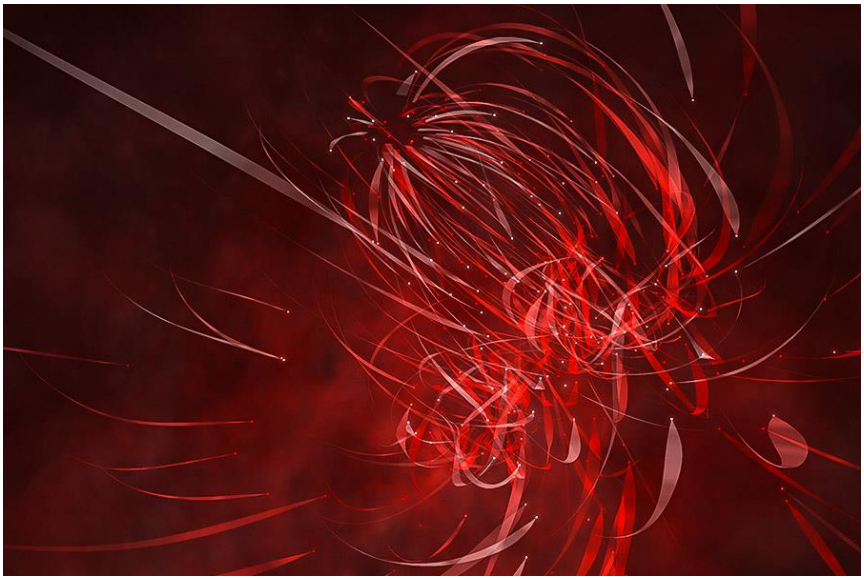

Figure 2. iTunes music visualizer [15].

One of the first digital music visualizers was the Atari Video Music, created in 1976. It was a hardware solution which could produce colorful visuals on an animated video display in real-time by interpreting the intensity of audio waves into various colors and shapes [16]. Since then, software solutions have become widespread and are included in popular media players, like Apple iTunes (see Figure 2), Windows Media Player, and WinAmp [17]. Their use is mostly limited to entertainment.

2.2 Benefits of Music Visualization

As music visualization has many useful purposes, it is difficult to choose which ones to represent in this thesis. The next subchapters expand on what the author feels like are some of the most important use cases for music visualizers. Subchapter 2.2.1 focuses on the therapy aspect of visual music. Subchapter 2.2.2 gives an idea how auditory visuals can aid with physical disabilities, and subchapter 2.2.3 explores how visuals can be incorporated in virtual concerts.

2.2.1 Audiovisual Therapy

Music therapy has been an effective way to relieve conditions such as autism, anxiety disorders, and stress. It is also shown to be very useful in rehabilitating stroke victims and assisting people with deafness or blindness, as described in subchapter 2.2.2.

McGowan *et al.* [6] mention that music therapy has been widely used to provide a non-verbal means of communication with those with Autism Spectrum Condition. People with autism often have trouble with communication and social interaction, so music can give them a suitable channel to express themselves. It was shown that when enhancing music therapy with real-time audio reactive visuals, patients with autism showed significant improvements in communication abilities.

A solution using image-based virtual reality (VR) technology to provide virtual environments of photorealistic quality was used to relieve stress in the workplace [18]. Due to the nature of this therapy being guided image therapy at its core, the patient was asked to imagine themselves in a relaxing scene. But instead of having to imagine a scene, the patient could enter the virtual environment and virtually navigate and explore it. Calm music was used to enhance the environment and help the patient get immersed in the world. The voice of a therapist was used to further reduce stress levels in participants.

Argo *et al.* [19] explored music therapy and its extensions as a means of working with anxiety, depression and trauma sufferers. A method of exposure therapy, featuring a rich soundscape that incorporates a patient's anxiety triggers as a form of habituation, was proposed. Instead of current exposure therapies' reliance on visual stimuli, like VR, it could be more effective to focus on the sound aspects more than the visuals. Using sound can negate the need for a visual overload of information. It does not matter how realistic or universal the visuals are, replicable sound events are much more reliable at triggering personal memories and experiences. As only some stimuli in virtual environments are required to increase anxiety in patients, creating more universal environments with diverse imagery and soundscapes could help a wide range of sufferers.

2.2.2 Physical Disability Relief

In addition to mental disorders, visualizing music can also help those with physical disabilities. Perception is the tool with which humans transform outside information, like sounds and colors, into senses, creating unique experiences and new knowledge [12].

From the work done for COL.diesis, Rossi *et al.* [20] acknowledged that the environment and culture affects how humans perceive colors. In addition to actually seeing the colors, the human conscience and subconscious uses previous experiences and aspects, like cultural influences, primordial imagery, and symbolism, to construct our unique perception of color. Even though blind people cannot see color, they can create their own concepts and associations to the descriptive words using emotions, experiences, and the senses of hearing, smell, and touch. In the case of COL.diesis, by associating musical cues to colours, it was discovered that there is a similarity between the perception of color and its representation in music. This approach can be used to give blind people a new way to express themselves and socialize, while also helping pedagogists understand the emotions and other mental problems of visually impaired people.

As mentioned before, the visual experience of music should be similar to its auditory experience. Fourney *et al*. [13] say that even though deaf people make their own music, they sometimes have difficulties trying to access music made by hearing people, most often due to the missing visual aspect. To hard of hearing people, hearing aids are not the solution either, as they are said to distort music. At live performances, dance and the vibration of the floor helps convey the emotion and experience of the music, but as soon as the performance has been recorded and gone digital, the entertainment value instantly drops for hearing impaired people. As for music visualizations, people with hearing disabilities prefer vibrant and visually appealing visualizations to more informative visualizations, which is to be expected, as most people listen to music for their own entertainment.

2.2.3 Live Performances

Continuing on the point brought up in paragraph 2.1, live music performances have featured visual elements in many different forms for a long time. Dance, moving imagery, lightshows, and pyrotechnics are just a small subset of how music can be augmented during live shows.

Live audiovisual performances often incorporate abstract imagery and music to provide unique viewing experiences to audiences. Video jockeys (VJ) are artists who prepare pre-recorded video and audio material, and during live performances, while still adhering to the flow of the performance, they process, mix, and combine these media sources to provide brand new audiovisual experiences [9]. Oftentimes, the music drives some parts of the visuals shown on screen, but many VJs like to have full manual control over their visuals.

Contemporary VJs use software, hardware, or most commonly, a mix of the two to create their live visuals. The equipment differs per artist, and while some might just use a software program, others might bring multiple instruments, like MIDI keyboards and mixers, into their performance (see Figure 3). In fact, VJ equipment has recently become quite a profitable venture for companies, both in the form of software and hardware [9]. Fortunately, there is a wide variety of entry-level freeware to start off with, and for extra flexibility, paid solutions can be considered.

Figure 3. A contemporary VJ rig [21].

Apart from the equipment, the type of VJing work also differs from artist to artist [22]. Some might be working along with a band, providing backing visuals and conforming to the vision of the band. Others might be complementing a disc jockey's performance in a club, but being autonomous in deciding the progression of the visual work. Live cinema, on the other hand, puts both mediums in the hands of the artist [9]. Unlike club VJing, live cinema focuses more on the linear structure and narrative of the performance. Audio and video are developed together by the same artist, and more often than not, the music is meant to be an accessory to the visuals, not the other way around.

Another use for music visualization in a performance context is for virtual concert performances. Such concerts take place in virtual environments using VR technology. The artists and their motions are captured using sensors and presented to audiences watching through VR displays in real-time [23]. The audience itself can be a fake simulated crowd, or can consist of virtual avatars of the people actually participating in the show. In addition to the artist and the crowd, virtual concerts usually feature a modeled stage, including visual

effects and a realistic concert environment, namely an outdoor field or an indoor concert hall [24].

Linjia He *et al.* [25] compared the psychological effects of watching a performance via conventional video to watching the same performance recorded with a panoramic camera on a VR display. It was discovered that the panoramic video in a virtual environment offered a much more immersive experience for the viewer. The participants using a VR display felt a heightened sense of presence and engagement, along with a higher desire to see the live performance afterwards.

2.3 Inspirations

This thesis was largely influenced by smaller music visualization projects, but also some commercial audio-driven video games. The next subchapters focus on some of these works as sources of inspiration. Subchapter 2.3.1 introduces some notable audiovisual artworks, and subchapter 2.3.2 goes in depth about the rhythm game AudioSurf[6].

2.3.1 Audiovisual Art

As far as computer-generated abstract art goes, fractal artwork might possibly by the most visually impressive of the sort, especially when rendered in 3D. A YouTube content creator by the name of CodeParade[7] used a rendering technique called ray marching to generate 3D fractals in real-time[8]. He then animated these fractals to react to audio, thus creating a fractal-based music visualizer, shown in Figure 4. Other examples of abstract music visualizers that were explored for this thesis include the Cubic Julia fractal music visualizer[9] and Conway's Game of Life music visualizer[10]. The Game of Life visualizer was rendered in a 2D environment, while the other mentioned visualizers were rendered in 3D environments. A difference in the complexity of their visual compositions can also be observed. As advanced rendering techniques were used to create the fractal-based visualizers, they look very visually complex compared to the Game of Life visualizer. Furthermore, with the exception of CodeParade's visualizers, the mentioned visualizers are not interactive.

[6] https://store.steampowered.com/app/12900/AudioSurf/
[7] https://www.youtube.com/channel/UCrv269YwJzuZL3dH5PCgxUw
[8] https://www.youtube.com/watch?v=svLzmFuSBhk
[9] https://www.reddit.com/r/proceduralgeneration/comments/gtxl8j/cubic_julia_fractal_music_visualiser/
[10] https://www.youtube.com/watch?v=R0bSaLr8bBU

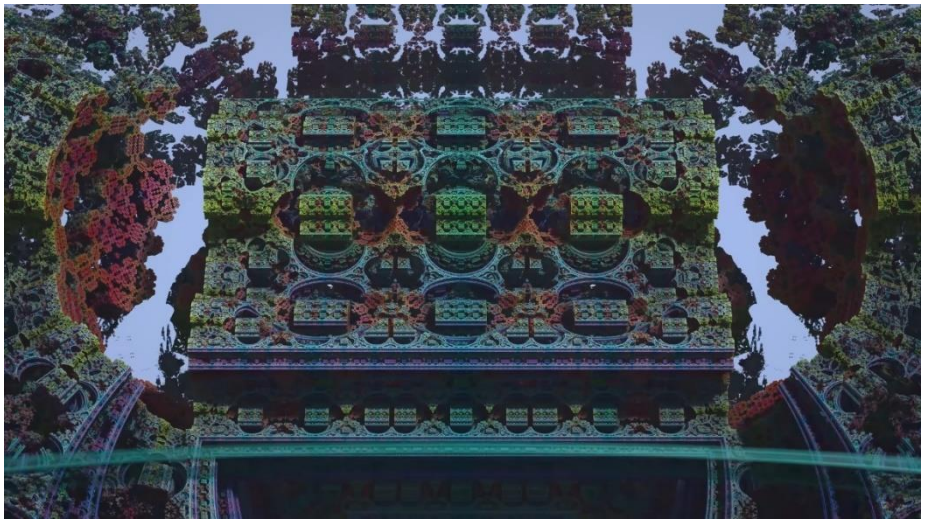

Figure 4. Still from CodeParade's *Audio Reactive Fractals* [26].

Adding to the topic of video jockeys above, during live performances VJs usually have a constant delivery of video material on the screen, so as not to let the entertainment value of the performance drop [22]. This constant flow of media is often called a *loop*. When selecting new visual samples, VJs must also think about how the video material can be looped seamlessly to give the illusion that a video clip is endless. Many artists use 3D modeling software to create these endless loops. Mike Winkelmann, also known as Beeple[11], uses Cinema4D[12] to create high quality VJ material, including loops, and releases them under the Creative Commons license to be used for free[13]. Although Beeple's loops are not audio reactive out of the box, they can still convey the emotion of a song very well when synchronized to the animation.

2.3.2 AudioSurf

AudioSurf is a rhythm game featuring a levitating vehicle driving across multiple lanes on a highway in an abstract and colorful environment (see Figure 5). The game allows players to select a piece of music, which is then used to generate a track that the vehicle follows. The goal of the game is to gather points by collecting colored blocks from the

Figure 5. Screenshot from *AudioSurf*.

[11] https://www.beeple-crap.com/
[12] https://www.maxon.net/en/cinema-4d/
[13] https://www.beeple-crap.com/vjloops

track. Different properties of the track, like its elevation, layout, colors, and scenery, vary depending on the music used to generate it. Faster music also increases the vehicle's speed while playing. Even though AudioSurf is not a visualizer per se, it still shares many similarities with the visualizers that have been previously mentioned, mainly due to the fact that the structure of the game environment is greatly determined by the type of music being played.

3 Design

The book *Information Visualization: Perception for Design* by Colin Ware [27] mentions the four stages of visualization as being: collection of data, pre-processing of the data, mapping the data to a visual representation, and the human perception of the visualized information. In the case of this thesis, data is the sound waves of a music track, pre-processing is the process of turning those sound waves into numeric values used to drive the visualizations, and mapping is the act of using the processed numeric values to affect some graphical properties of an arbitrary object in a virtual environment. The stages are expanded on in detail in subchapter 3.1.

The type of work required to make visually appealing music visualizers can oftentimes be more creative than technical. As such, certain design principles and guidelines should be followed. Ware gives a list of 168 guidelines for the design of visualizations in his book [27], elaborating on brightness, colors, visual clarity, patterns, and more. But due to the fact that the visualizations created as a part of this thesis are more for demonstration purposes, and because the author is not an artist, these guidelines were for the most part ignored. Subchapter 3.2 gives an overview of the scope of the visualizers and describes each of the created visualizers in detail.

Some of the terms used to describe the visualizers' composition:

- ***Meshes*** are pieces of geometry in game engines and 3D modeling software that consist of vertices, edges, and faces to define the shape of an object. Because meshes are cheap to render on a graphics card, they are widely used for creating complex 3D objects.
- ***Sprites*** are two-dimensional images that can be incorporated in three-dimensional scenes for extra visual identity.
- ***Materials*** are assets that can be applied to objects in a virtual environment to define the appearance of their surfaces. Materials are responsible for the objects' surface color, transparency, shininess, bumpiness, and many other properties.
- ***Particle emitters*** act as sources for particles and hold control over the spawned particles and their attributes in a virtual environment.

3.1 Audio

Music is a composition of sound waves of different frequencies. Sound waves can be thought of as a sum of sine waves that are characterized by properties like frequency and amplitude. The frequency of a sound wave, measured in Hertz (Hz), determines the pitch of the sound, and the amplitude determines its volume [28]. When sound is processed by a computer, all of the sound waves are translated into digital audio signals, and their amplitudes are added together. This can be represented by an audio waveform – the display of audio amplitude dependent on time. As audio waveforms are not very informative for music visualization [4], some processing of the audio has to be performed to extract more information from it. The next two chapters go through the process of the audio processing and describe how the audio can be mapped to various graphical properties.

3.1.1 Audio Processing

In order to gather more information about the audio, Chaudhary [29] suggests transforming the audio signal representation from time-domain to frequency-domain. Meaning that instead of showing the amplitude of the audio changing with time, it would show what frequencies are present in the signal, along with their magnitudes. The transformation is done using the Fast Fourier Transform (FFT) algorithm, which, in essence, decomposes the audio signal into its individual component frequencies. The frequencies can then be mapped to the audio frequency spectrum, which was done for the visualizers in this thesis.

In the context of this thesis, the audio frequency spectrum is a list of frequencies. It spans from 20 Hz to 20 000 Hz, which represents the human hearing range [30]. Each index in the spectrum holds the amplitude of the corresponding frequency at a single stage of the simulation. Because the extreme high and low end frequencies are not represented much in music, the audio frequency spectrum for the visualizers was limited to frequencies between 55 Hz and 10 000 Hz. To avoid a performance loss from passing a list this large to the visualizers, the frequencies were divided into a smaller list, where each index corresponds to the average amplitude of a small range of frequencies.

To further process the audio, optional equalizer and threshold filters can be applied. Both of these filters are used to vary the audio amplitude as a function of frequency. The threshold filter can be used to reduce the amplitude at specific frequency ranges or to cut the ranges entirely. The filtered amplitude values can be calculated using the formula

$$A_i' = \max\left(0,\ \frac{A_i - x_i}{1 - x_i}\right),$$

where A_i is audio amplitude at index i in the audio frequency spectrum list, x_i is the value at index i in the threshold filter list, ranging from 0 to 0.95, and A_i' is the new amplitude value at index i in the audio frequency spectrum.

The equalizer filter, on the other hand, just multiplies the amplitude value at index i in the frequency spectrum list with the value at index i in the filter list. Meaning that in addition to reducing the amplitude, this filter can also be used to amplify certain frequencies.

3.1.2 Audio Mapping

Giannakis [4] explains some of the issues behind designing useful audiovisual mappings. The first issue is related to the characteristics of sound used for mappings. Audio waveforms are time-domain representations of sound that offer information about audio amplitude variation over time, but other than that, they have a real lack of perceptual information. When looking at waveforms, it is difficult to determine what they will sound like. In addition, two identical sounding waveforms may look completely different. Frequency-domain representations, though, offer much more information about the audio due to various spectrum characteristics and, in theory, should aid in the perception of sound much better.

The second issue explores different graphical representations for audio visualization. Sensory mappings are mentioned when referring to representations that are innately understandable and that do not require knowledge about the visualizations beforehand. Arbitrary mappings, on the other hand, require some learning before they are fully understood. Sensory mappings include graphical properties, like color, texture, space, and motion. For auditory visualizations, sensory mappings have often been used to map changes in frequency and amplitude to height and color, respectively. In some cases, sound pitch has been mapped to color hue, tones to saturation, and loudness to color brightness. The problem with mappings like these is the lack of evidence to validate the correlations between the graphical properties and audio characteristics.

Taking these issues into account, the mappings in this thesis use frequency-domain representations for the audio data and sensory mappings for the visual components. The term *scale* is used when audio amplitude at a specific index in the audio frequency spectrum is used to transform some graphical property. Size, position, velocity, rotation, and color are the properties that can be scaled by the amplitude. Scaling techniques include adding new

values to the initial property values, multiplying the initial values, and continuously multiplying the current value on every frame of the simulation.

3.2 Visualizers

As mentioned above, the visualizers created for this thesis are meant to serve as examples for music visualization in general. Each visualizer is aiming to show different aspects and possibilities of audiovisual mappings. For clarity and comprehensibility, and in order not to confuse any potential users, the visual compositions of the visualizers were mostly kept very simplistic. In some cases, tutorial content was used to create the baseline visualizers, which were then modified to fit the needs of the project.

To keep the file size of the project at a minimum, external 3D modeling software was not used to create any meshes used in the visualizers. Instead, some primitive meshes were used from the Starter Content[14] asset pack, provided by Unreal Engine 4. By default, the primitive meshes were visually bland, so materials had to be created to give them some substance. Most of the materials created are very basic and only consist of single colors. Some of the more complex materials, though, are animated and were made with external assistance.

Each visualizer is composed of one or multiple particle emitters that either continuously spawn particles with a specified lifetime or just spawn a set of persistent particles once. Each spawned particle acts as a separate entity. The particles are rendered as either sprites or meshes, materials are applied to all particles, and audio is mapped to particles in such a way that they respond to the processed audio values individually. The next subchapters go in depth about the visual composition of the visualizers and also mention how audio was mapped to the graphical properties of the particles.

3.2.1 Volcano 1 & 2

The *Volcano* visualizers, pictured in Figure 6, are a pair of conical particle emitters that are conceptually similar but visually different. Both emitters feature tiny pyramid-shaped particles continuously floating upwards. Parallels can be drawn between these emitters and real life volcanoes, as due to their color and conical shape, *Volcano 1* resembles an active volcano and *Volcano 2* a dormant one.

[14] https://docs.unrealengine.com/en-US/Basics/Packs/index.html

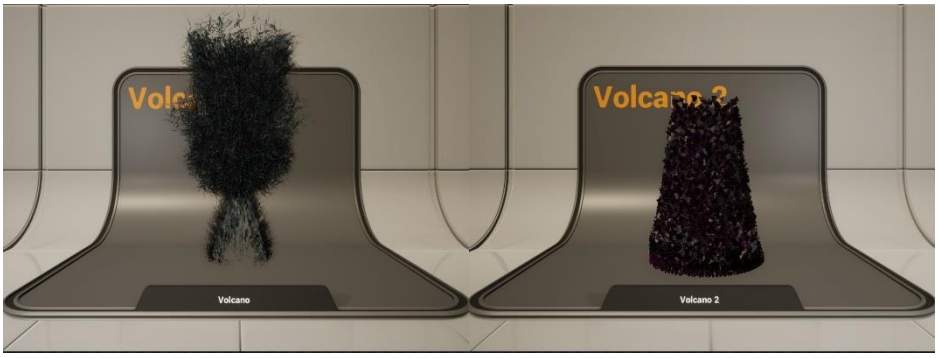

Figure 6. The *Volcano* visualizers.

For both visualizers, all particles are mapped to a single index in the frequency spectrum. In the case of *Volcano 1*, that index is at the beginning, and for *Volcano 2*, it is at the end of the spectrum. The emitters also differ in graphical mappings. For *Volcano 2*, the audio amplitude at a specific index in the frequency spectrum is used to scale the particle size. For *Volcano 1*, both velocity and size are scaled by the amplitude.

3.2.2 Lava 1-3 & Plasma Ball

The *Lava* and *Plasma Ball* visualizers, illustrated in Figure 7, are mechanically very similar. With the exception of *Plasma Ball*, they consist of a single persistent mesh particle with a custom animated material applied to it. The material resembles the look of moving lava and was created with the assistance of Ashif Ali's tutorial[15]. *Plasma Ball* consists of four particles and uses the same material as the *Lava* visualizers, but with different speed and color settings.

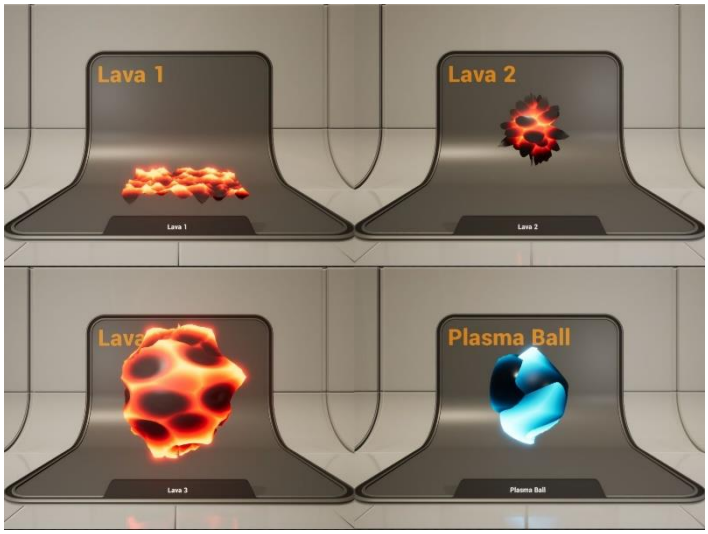

Figure 7. The *Lava* and *Plasma Ball* visualizers.

[15] https://www.youtube.com/watch?v=47_W3PUiics

The aim of these visualizers is to demonstrate how different properties of materials can be affected by audio. In all cases, the distortion value of the material is scaled by the audio amplitude. This gives the *Lava 1* visualizer a wave-like effect, *Lava 2* looks like it is imploding, and *Lava 2* looks like it is exploding. As the name suggests, *Plasma Ball* looks like an abstract ball of plasma, expanding and shrinking according to the audio amplitude.

3.2.3 Tentacles 1 & 2

The *Tentacles* visualizers, shown in Figure 8, consist of multiple particle emitters, each emitter making up the visual appearance of a single tentacle. The *Tentacles 1* visualizer features six such emitters, and the *Tentacles 2* visualizer features eight. Due to their complexity, a tutorial by Ashif Ali[16] was used to create the base tentacle emitters.

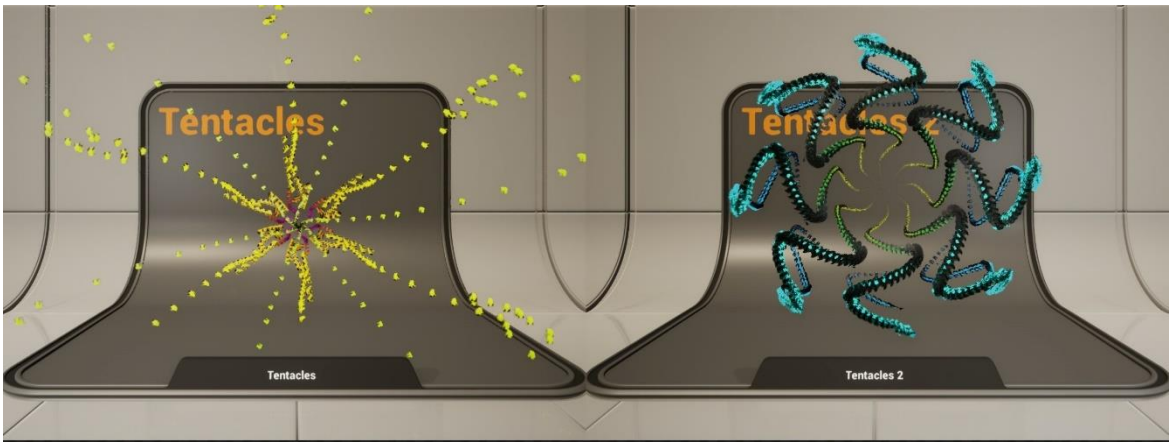

Figure 8. The *Tentacles* visualizers.

They were then modified to be audio reactive, and the hue shift feature was enabled. Hue shift will be explained in depth in chapter 4, but in short, it makes the color of each particle change over time based on the intensity of the particle's original color.

In terms of audio mappings, both visualizers use audio amplitude at the same index in the frequency spectrum to scale particle size and velocity. The difference being that only the initial velocity of the particles of *Tentacles 2* is scaled, but for *Tentacles 1*, the velocity of its particles is scaled continuously over time. This gives *Tentacles 1* a much more chaotic appearance.

[16] https://www.youtube.com/watch?v=4GSCk4FSCpc

3.2.4 Cloud

Figure 9 shows the *Cloud* visualizer, which was created with the assistance of Sem Schreuder's tutorial[17] and features a simple cloud of meshes with a basic material applied to every particle. The cloud moves around chaotically, and the color of the particles changes depending on how far they are from the center of the visualizer. Both particle size and velocity are scaled continuously over time according to the audio amplitude at a single index in the frequency spectrum.

Figure 9. The *Cloud* visualizer.

3.2.5 Holorays

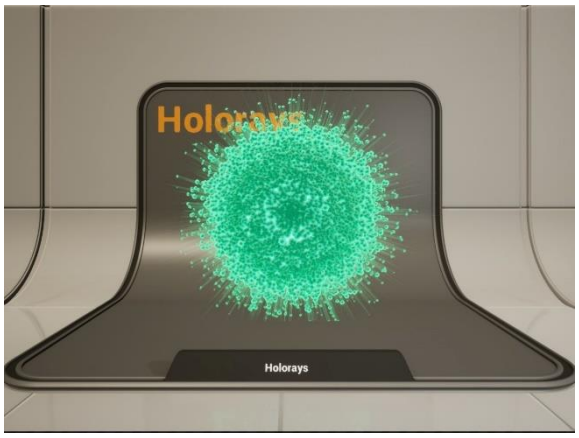

Figure 10. The *Holorays* visualizer.

The design for the *Holorays* visualizer was adapted from the tutorial by Ashif Ali[18]. It features a single particle emitter, where transparent particles are following other spherical opaque particles to create a holographic effect. It is illustrated in Figure 10. The emitter was modified to be audio reactive, and hue shift was enabled on the particles. Just as before, the audio amplitude at a single index in the frequency spectrum is used to scale the velocity and size of the particles.

[17] https://www.youtube.com/watch?v=UETAS5g-q4M
[18] https://www.youtube.com/watch?v=zvXGckZybIA

3.2.6 Ribbons

Figure 11. The *Ribbons* visualizer.

The *Ribbons* visualizer takes the form of a typical audio frequency spectrum visualizer (see Figure 11). It consists of 256 persistent ribbon particles laid out sequentially, and each ribbon corresponds to a specific index in the frequency spectrum in the order they are laid out in. The color of the ribbons also changes depending on the distance from the first ribbon. Audio amplitude at a specific index in the spectrum is used to scale the height of the ribbon corresponding to that index.

3.2.7 Stars

The *Stars* visualizer features white sprite particles resembling stars sparsely populating a rectangular space, as shown in Figure 12. The stars have forward momentum, giving the illusion that the user is moving through space. Each particle corresponds to a specific index in the frequency spectrum, but their order is random. This visualizer demonstrates how the brightness of a particle can be scaled by audio amplitude. In addition to the brightness, the velocity of the particles is also scaled.

Figure 12. The Stars visualizer.

3.2.8 Tunnel

The *Tunnel* visualizer, illustrated in Figure 13, is relatively similar to the *Volcano* visualizers. It also features continuously floating mesh particles that are spawned in a circular shape, but the direction of their movement is now different, and the velocity is also much lower. This visualizer uses yet another technique to map the particles to indexes in the frequency spectrum – normalized particle position. It takes the distance of each particle from the center of the visualizer, normalizes the distance, and maps it to an index in the frequency spectrum. Then, the size of each particle gets scaled by audio amplitude depending on their indexes in the frequency spectrum.

Figure 13. The *Tunnel* visualizer.

3.2.9 Torus Knot

As seen in Figure 14, the *Torus Knot* visualizer spawns persistent pyramid-shaped particles in a helix-like shape, and rotational velocity is applied to each particle to spin them around the centerpoint of the visualizer. Over time, the particles form a knot around the center of the visualizer. In addition, hue shift is enabled on each particle.

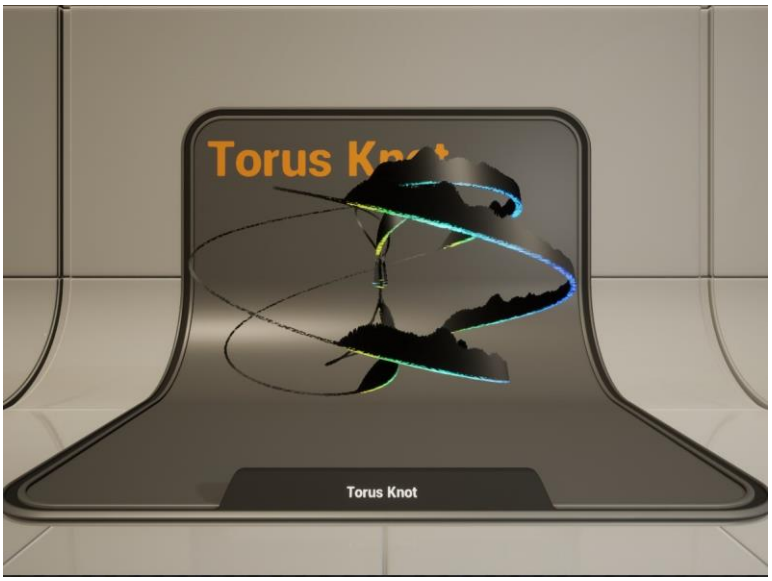

Figure 14. The *Torus Knot* visualizer.

Because the particles have velocity, the normalized initial position of each particle is used to calculate their index in the frequency spectrum. This means that, when spawned, the particles assigned to an index in the spectrum always correspond to that specific index, even when the particles move around in the environment. This method is used to make the visualizer seem more dynamic. Only the size of the particles is scaled by the audio amplitude for the *Torus Knot* visualizer.

3.2.10 Sphere 1 & 2

The *Sphere* visualizers are a pair of particle emitters that are quite similar to each other at their core. Both visualizers spawn persistent mesh particles in the shape of a sphere with rotational velocity applied to them to make the spheres rotate (see Figure 15). *Sphere 1* consists of pyramid-shaped meshes, and *Sphere 2* is made from regular cube meshes. The orientation of these meshes differs per emitter. For *Sphere 1*, the meshes are orientated sideways along the edge of the sphere, but for *Sphere 2*, the meshes are pointed at the center of the sphere. The spheres also differ in color and materials. *Sphere 1* has a custom material applied to each particle and has hue shift enabled, but *Sphere 2* only uses the default cube mesh material.

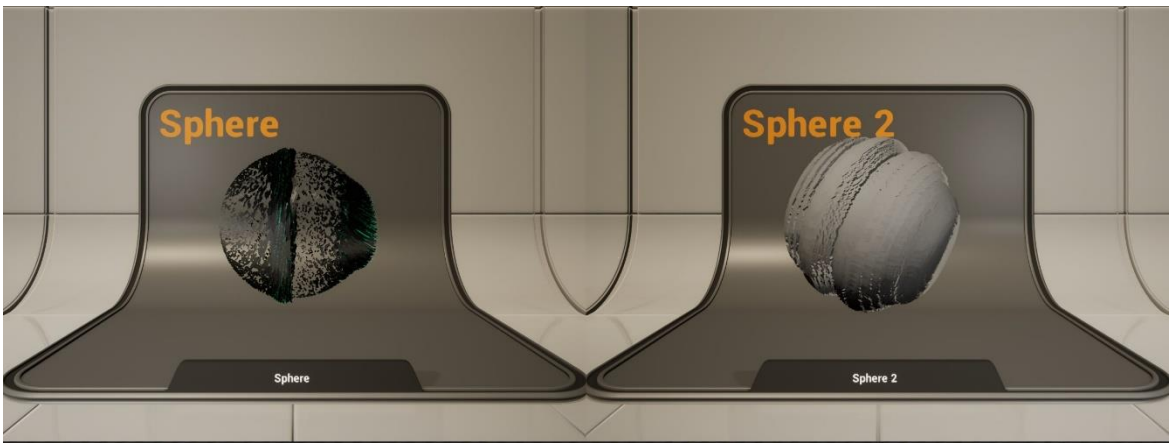

Figure 15. The *Sphere* visualizers.

In terms of audio, both spheres use the normalized initial particle position method for their frequency spectrum index mapping, just like the *Torus Knot* visualizer. However, this is not the best way to map spectrum indexes on a sphere, as the method does not account for the symmetry of the sphere and only maps the spectrum linearly from one side of the sphere to the other. Both *Sphere* visualizers also only scale the size of the particles by audio amplitude.

3.2.11 Circle Plane

The *Circle Plane* visualizer spawns persistent mesh particles in a cylindrical shape (see Figure 16). The meshes use the same custom material that is used in previous visualizers, like *Sphere 1*, *Torus Knot*, and the *Tentacle* and *Volcano* pairs. This visualizer also has hue shift enabled.

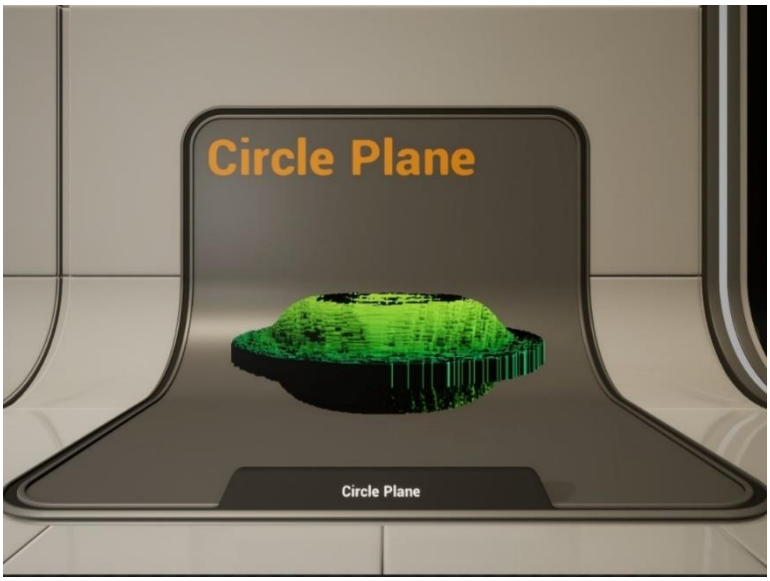

Figure 16. The *Circle Plane* visualizer.

This visualizer demonstrates how particles can be mapped to the frequency spectrum for more complex particle emitters. During spawn time, a custom spectrum index is saved for each particle. This index is created by normalizing the distance of a particle from the edge of the cylinder and then subtracting it from 1 to reverse the spectrum. Then, a harsh equalizer is applied to the audio to reduce most of the higher frequency sounds. Additionally, the height of the particles is scaled by the audio amplitude. As a result, the visualizer pulsates symmetrically, responding to low frequencies in the center and higher frequencies on the edges.

4 Implementation

In order to even start building music visualizations, some kind of frameworks have to exist to display the visualization environment and the objects and colors inside it. Such frameworks can be in the form of hardware or software, as mentioned in chapter 2. Due to the advancement of computer software and the various limitations of hardware, like availability and customizability, this thesis focuses more on the software side of music visualization.

Even on the software side, there are many options for picking the fitting technologies. One option would be to make everything from scratch, starting from displaying objects on the screen and ending with writing complex visualization code. Another option would be to use various graphical libraries that common programming languages offer. That would still require writing a large amount of custom code, but the process would be simplified by the building blocks that the libraries offer. To take it a step further, game engines can be used to further simplify the development of music visualizations. With built in systems to handle audio, lighting, visual effects, and physics of the environment, more effort can be spent on specifically the visualization elements, which can greatly speed up the entire development process. For more freedom, many game engines offer a choice between writing custom code, using visual scripting tools, or using a combination of both.

Unreal Engine version 4.26.1[19] was used as the game engine of choice to create the music visualizers for this thesis. The reasons are explained in subchapter 4.1. Unreal Engine's new visual effects system, Niagara, is introduced in subchapter 4.2. Subchapter 4.3 expands on the technical details behind the visualizers, and an overview of the main result of this thesis, the *Visualize Audio Spectrum* module for Niagara, is given.

4.1 Unreal Engine 4

Unreal Engine 4 is a free open-source game engine that supports development for multiple platforms, including desktop, mobile, VR, and consoles. Apart from games, the engine has also been used in many different industries, like architecture, automotive, film and television, and many more. It offers a wide toolset and many powerful built-in systems to ease development.

[19] https://docs.unrealengine.com/en-US/WhatsNew/Builds/ReleaseNotes/4_26/index.html

The main factors for choosing Unreal Engine for this thesis are: it is free, it features a robust visual scripting system in the form of Blueprints[20], its audio engine recently received an overhaul [31], and it has a powerful new visual effects system, called Niagara. Additionally, the engine offers great graphical fidelity out of the box, it offers an easy way to create custom materials, and it has great synergy with various 3D modeling or animation software, like Cinema4D and Houdini[21].

Visual scripting was created to provide non-programmers a way to automate tasks without needing the expertise of a programmer [32]. Unreal Engine 4 uses visual scripting in the form of Blueprints. Blueprints are visual node-based interfaces where code is replaced by nodes. Nodes can be connected with wires to create gameplay objects, events and functions. Blueprints can also be used to extend classes written in the C++ programming language, so it is possible to have a flexible workflow where programmers write gameplay classes in code and designers build on top of them with Blueprints to create custom gameplay elements.

4.2 Niagara Visual Effects System

The Niagara Visual Effects System was built as a replacement for Unreal Engine 4's old particle system, Cascade[22]. Niagara still offers much of the functionality Cascade did, but many of the core design processes have been overhauled.

According to the official documentation [33], Niagara features four hierarchical core components for creating visual effects:

- Systems
- Emitters
- Modules
- Parameters

Systems are the highest level components of Niagara. They can be placed in the game environment, and all of the other components are contained in the systems. Their main purpose is to be a container for emitters. A system can have one or multiple emitters inside

[20] https://docs.unrealengine.com/en-US/ProgrammingAndScripting/Blueprints/index.html
[21] https://www.sidefx.com/products/houdini/
[22] https://docs.unrealengine.com/en-US/RenderingAndGraphics/ParticleSystems/Cascade/index.html

it, and the emitters' parameters can be overwritten on a per-system basis without modifying the emitters themselves.

Emitters are reusable components that are, in essence, a collection of modules. Emitters are mostly single-purpose. For example, a projectile visual effect could be made using multiple emitters. One emitter might be used to spawn the projectile mesh, another could be used for the projectile trail, and a third emitter might be used to produce some sparks on the projectile's collision with an object.

Modules are the base components of Niagara. They control how particles are spawned and updated. There are modules for adjusting particle velocity, spawning particles in certain shapes, changing how particles are rendered, controlling particle spawn speed, and much more. Modules are built using node-based graphs, like Blueprints, but they do not share the same functionality. Niagara modules are limited to Niagara-specific data types, functions, and expressions. Modules can also be made with High-Level Shading Language (HLSL), and inline HLSL can be added to graph-based modules.

Parameters are an abstraction of data in Niagara [33]. They can be used to define primitive numeric data types, enumerated types, structured types, or custom data interfaces. All properties of particles are in the form of parameters, which are located in the Niagara parameter map. Parameters can have namespaces to further distinguish the purposes of each parameter.

Niagara combines the graph paradigm and the stack paradigm to provide a hybrid approach for making visual effects [34]. Graphs are used to build modules, and modules in an emitter are assigned to groups, in which they are executed in a stack, from top to bottom.

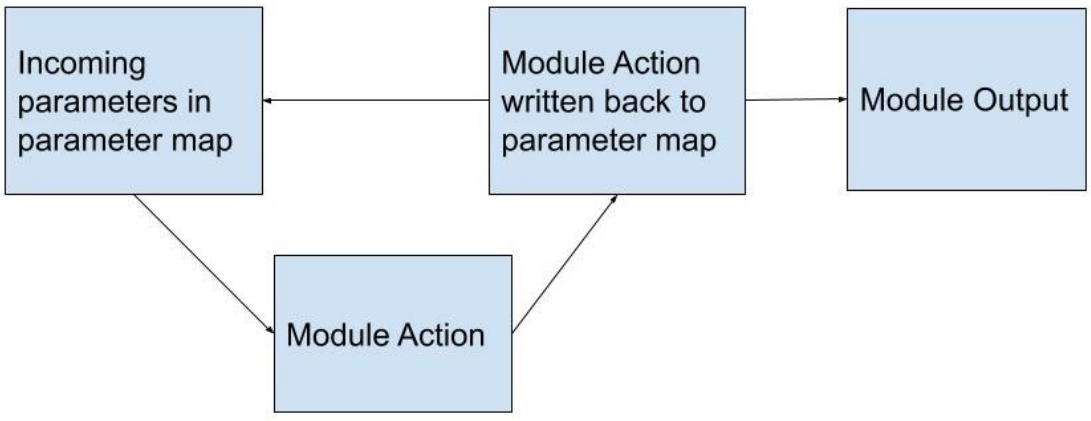

Figure 17. Niagara module function flow [33].

The Niagara workflow relies on the parameter map, which holds data about every particle in a Niagara emitter. The parameter map receives updated particle data on every frame and hands it to the module stack, where the data is modified and put back into the parameter map or returned (see Figure 17).

Due to Niagara still being relatively new, it lacks extensive documentation and tutorial content for creating visual effects, thus the learning curve for it is very high. Nonetheless, the features Niagara offers makes up for the difficulty and beats the alternatives for the purposes of this thesis. As of writing this thesis, Niagara does not include a default module for visualizing audio, so a custom module was developed for that purpose.

4.3 Visualize Audio Spectrum Module Overview

Visualize Audio Spectrum is a module for Unreal Engine 4 Niagara that was created as the main purpose of this thesis. It is meant to streamline the process of music visualization in Unreal Engine. It was created because, as mentioned in the previous chapter, such a module does not exist in Niagara by default, and there is a serious lack of free tutorial content for Niagara. All of the visualizers described in chapter 3 use this module to make them audio reactive. The visualizers were created to demonstrate the various possibilities of Niagara and the module, and to lay down stepping stones for real-time music visualization in Unreal Engine 4.

As mentioned in chapter 3, each particle spawned by a Niagara emitter is a separate entity, so modules only modify a single particle at a time. The general flow of events for particles passing through the module is:

1. Calculate audio frequency spectrum index.
2. Calculate audio amplitude.
3. Apply hue shift.
4. Scale particle parameters.

To elaborate on the flow further, when the parameter map gets passed to the module, for the first step, each particle gets assigned an index in the audio frequency spectrum. The methods for calculating the index are listed in Figure 18 and were described in chapter 3 for the visualizers using them.

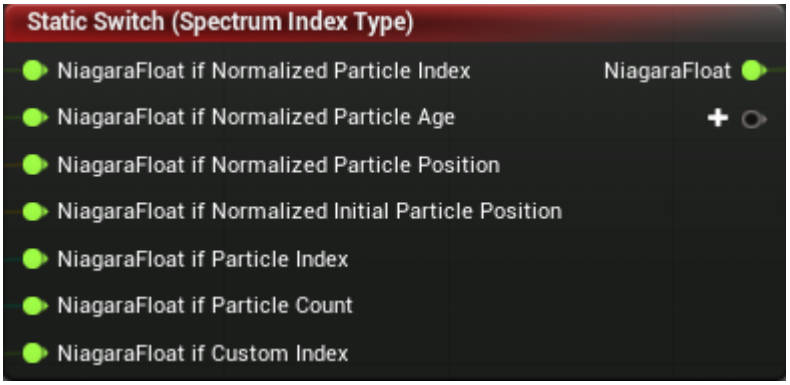

Figure 18. Methods for calculating the audio spectrum index for a particle.

Proceeding onwards, Niagara internally processes the audio that is playing in the engine by applying the FFT algorithm on it and produces an audio frequency spectrum. The index calculated in the previous step is now used to extract an amplitude from the audio frequency spectrum. Next, the threshold and equalizer filters mentioned in chapter 3 are applied to the amplitude, and the new amplitude is saved in the parameter map.

```
function HueShift(color, shift, saturation, brightness) do
    HSV = RGBtoHSV(color) // converts color from RGB to HSV
    (H, S, V) = Convert(HSV) // unpacks the color

    H += shift // hue
    S *= saturation // saturation
    V *= brightness // value (brightness)

    HSV = Convert(H, S, V) // packs the color back together
    RGB = HSVtoRGB(HSV) // converts color from HSV to RGB
    return RGB
end
```

Code 1. Pseudocode of the hue shift helper function.

As the next step, hue shift can be applied to the particles based on their spectrum index calculated in step 1. Some helper functions were created to make the process clearer. The hue shift function, depicted in Code 1, takes the initial color of the particle, the extent of the shift, and saturation and brightness values as inputs, and outputs a new color based on the shift amount. Inside the hue shift function, the input color is converted from the RGB color model to the HSV color model by using a custom HLSL node within Niagara graphs. The algorithm by Taylor *et al.* [35] was adapted to make the conversion. Then, the shift amount received as input is added to the hue value of the color, and saturation and brightness values are multiplied by the corresponding input values. Next, the color is converted back to RGB

and is returned by the function. The returned value is set as the new particle color and stored in the parameter map.

For the last step, a selection of particle parameters can be scaled by the audio amplitude calculated in step 2. As mentioned in chapter 3, the size, velocity, position, and rotation parameter values of mesh and sprite particles can be scaled. An option to scale arbitrary float or vector type values is given as well. Two groups of helper functions were created to assist with the scaling. One group scales the parameter values additively and the other group multiplicatively. All functions belonging to a single group are functionally identical, but differ in what types of parameters they accept. The input parameters have to be either floats, two-vectors, or three-vectors.

```
function ScaleAdd(map, amplitude, sensitivity, value, max, extra) do
    target = max * amplitude

    // converges to the target, starting from value
    newValue = Converge(sensitivity, value, target)
    newValue += extra
    return newValue
end
```

Code 2. Pseudocode of the additive scaling functions.

Code 2 describes the first group of functions. These functions take the parameter map, audio amplitude, scaling sensitivity, the initial parameter value, the maximum value, and an extra value as inputs. A target value is calculated by multiplying the maximum value with audio amplitude. Next, the initial input value converges to the new target value over time. The speed of the convergence is controlled by the sensitivity. After convergence, the extra value is added to the resulting value, and the final value is returned.

```
function ScaleMultiply(map, amplitude, sensitivity, value,
                       multiplier, extra) do
    target = value * multiplier * amplitude

    // converges to the target, starting from value
    newValue = Converge(sensitivity, value, target)
    newValue += extra
    return newValue
end
```

Code 3. Pseudocode of the multiplicative scaling functions.

The functions in the second group, shown in Code 3, take the parameter map, audio amplitude, sensitivity, the initial parameter value, the multiplier value, and an extra value as inputs. Similarly to the last function, a target value is calculated. The difference here being that the initial parameter value is multiplied with the input multiplier value and the audio amplitude. Because of this, the initial parameter value must not be zero, otherwise the output value will always stay at zero. Then, the initial value converges to the target value over time. As previously, the extra value is added to the parameter value after convergence, and the final value is returned.

The final parameter value from the function is then added to the parameter map in the case of all functions. The purpose of the extra value in both function groups is to give the functions more flexibility. Additionally, the smoothness of the animations can be controlled by the user by tweaking the sensitivity value.

As the module is designed to be very flexible, all of the scaling functions, including hue shift, are optional. This is achieved by having a static switch before each scaling function. The user just has to click on a checkbox to enable or disable the scaling of a parameter type. Multiple scaling functions can be enabled at once, and the module itself is exposed to other Niagara modules, so very complex visualizations can be created using this module.

5 Testing

Software testing is the activity of gauging the quality of a program, discovering problems, and determining if the software is achieving its goals [36]. To appropriately judge the ease of use and practicality of the module developed as a part of this thesis, usability testing of it and the pack of visualizers was conducted. Subchapter 5.1 explains usability testing and goes in depth about the testing methodology used in this thesis. Subchapter 5.2 describes the results of the testing process, and subchapter 5.3 mentions some of the improvements that could be made from user feedback.

5.1 Methodology

According to Jenkins [37], usability testing is a software testing technique where some target users are handed the product and asked to try to use it as if they were the end-users. Any problems the users encounter while testing are written down by the designers and evaluated in the next development stages. The users themselves have to be specifically chosen to represent the end-users. Various properties, like age, experience, education, and technical expertise are taken into account when selecting the target users. Jenkins mentions that only four or five evaluators can be enough to provide sufficient feedback about the product, and that any more could bring diminishing returns.

The module and the visualizers were packaged in a project, named *AudioVisual*. To conduct testing, the project was distributed by sending the the project's repository link (see Appendix II) and the demo video (see Appendix III) to the *labor.ati.cgvr* mailing list, and by sharing the links in online communities that focus on audio and audio visualization in Unreal Engine 4. The supervisor of this thesis also sent the project to some of his acquaintances familiar with Unreal Engine 4. With the exception of the mailing list, all target users were expected to have at least some experience with Unreal Engine 4. Strict guidelines were not imposed on the testers, but they were strongly encouraged to fill the feedback form regarding the project, which can be found in the Accompanying Files (Appendix I). The testers were asked to rate their expertise with Unreal Engine 4 and the overall usability of the module. They also had the option to note down any problems they encountered and could offer feature suggestions for the project.

5.2 Results

The feedback form included eight questions and was open for two weeks. Four responses were collected during that time, which can be found in the Accompanying Files (Appendix I). The amount of responses may seem low, but as mentioned above, four responses is often enough to make valuable conclusions. As the topic of music visualization is relatively niche, and coupled with the fact that only a small handful of people use Unreal Engine 4 to visualize music, this amount of responses was to be expected. Nonetheless, all of the responses were thorough and offered constructive feedback.

All but one person rated their skill with Unreal Engine as either 4 or 5 on a scale from 1 (*beginner*) to 5 (*professional*). The remaining respondent rated their skill as a 1. This occurrence should be noted for the subsequent questions, as that person also had difficulties using the module and rated its ease of use as a 1 (*very difficult to use*). The rest of the respondents rated the ease of use as either 4 or 5 (*very simple to use*), as shown in Figure 19.

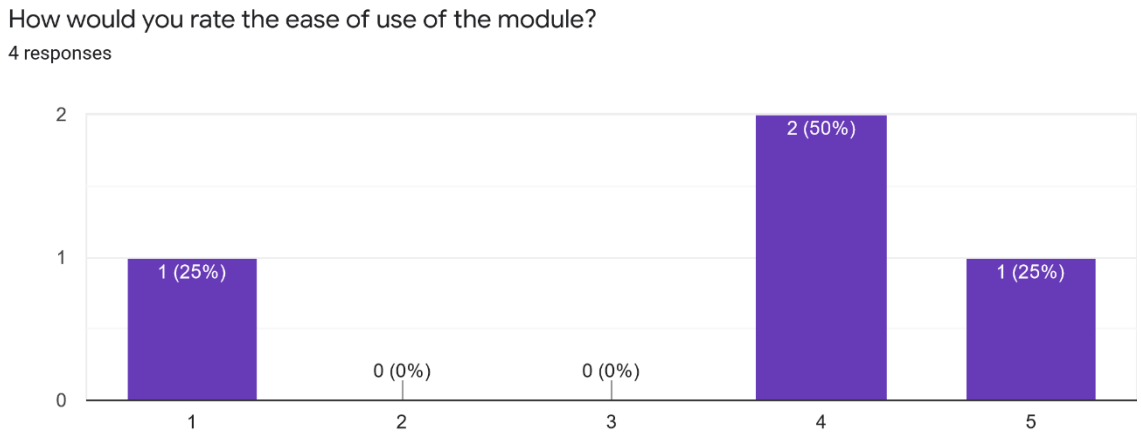

Figure 19. Ease of use of the module.

Coincidentally, the next two questions also had very similar responses, with most people being either likely or very likely to use the module in a project of theirs and finding the example visualizers helpful or very helpful for understanding how the module works. Yet again, the person who rated their skill the lowest is unlikely to use the module in a project of theirs and found the example visualizers to be unhelpful.

Figure 20 shows which example visualizers the respondents liked the most. As seen, all responses were unique. Due to the low amount of responses and relatively high amount of visualizers, noteworthy conclusions cannot be drawn from this question.

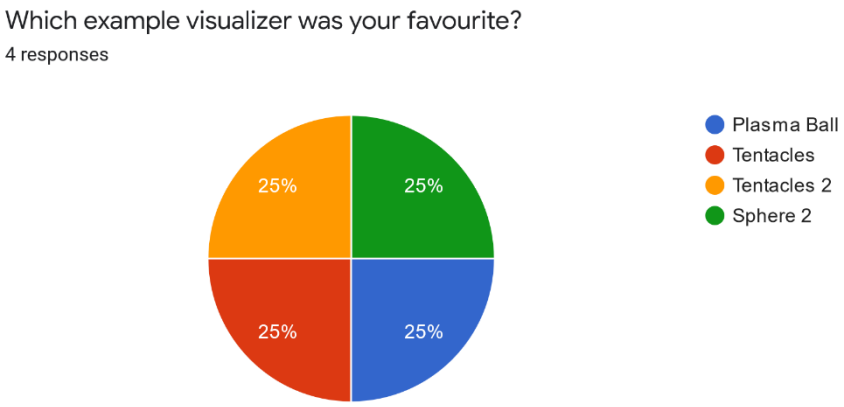

Figure 20. Which example visualizers the respondents liked the most.

The next set of questions offered the most qualitative feedback for the project. Two respondents, although having not used it extensively yet, encountered no problems with the module, besides some typical Unreal Engine-related hiccups. One user had issues fine-tuning the properties of the visualizers due to the design of Niagara's interfaces, and the last user could not get the project to work at all. When queried about new features, all respondents requested for more assistance in the form of better documentation, more tutorials, or more examples. One person also specifically asked for a cartoon mouth visualizer to visualize speech animations. Another person believes that the module could cut down on development times significantly. Finally, a respondent suggested that the project will kickstart a lot of audio visualization projects and will be a stepping stone for both beginners and professionals.

5.3 Improvements

From the responses, it can be concluded that experience with Unreal Engine 4 and Niagara is required to use the module. The complexity and documentation issues with the module could partly be attributed to Niagara's learning curve and its lack of extensive documentation, as mentioned in chapter 4.2. However, more tutorial content and examples are still a must for the *AudioVisual* project going forward. More extensive examples could also reveal feature limitations with the module, which then have to be alleviated once again.

6 Conclusion

As a result of this thesis, the *Visualize Audio Spectrum* module for Unreal Engine 4 Niagara was developed. The module was created due to the fact that Niagara does not include such a module by default at the time of writing this thesis. It is used to streamline and simplify visualizing music and other audio in Unreal Engine 4 by synchronizing various 3D objects with audio. Along with the module, seventeen example visualizers were created to provide assistance with learning how to use the module.

In this thesis, the history of music visualization and different purposes for visualizing music were explored, including therapies, live performances, and virtual concerts. Various audio-visual works that inspired this thesis were also reviewed and compared.

The design process of music visualization was described, starting with the process of analyzing and processing of the audio. To combine the auditory and visual mediums, different audio-visual mappings were investigated. As such, the time-domain and frequency-domain representations of audio were compared, and suitable graphical mappings were chosen for the visualizers in this thesis.

Descriptions of each of the visualizers developed for this thesis were given. Their visual compositions and audiovisual mappings were examined in detail. Implementation details for the *Visualize Audio Spectrum* module, which is used to make the visualizers audio reactive, were also described. In addition, the flexibility of the module was emphasized.

Usability testing of the module and the visualizers was conducted. Its aim was to gauge the ease of use of the module and to see if it was achieving its goals. The feedback was mostly positive, and the practicality of the project was validated. Though, some documentation problems were highlighted. In addition, a strong correlation between expertise with Unreal Engine 4 and the ease of use of the module was observed.

The author plans to continue developing the module by improving the documentation and creating more example visualizers. A visualizer involving fractals is being planned. Additionally, with further development, the possibility of releasing the project for free on the official Unreal Engine marketplace might become reality.

References

[1] Smith SM, Williams GN. A visualization of music. In: *Proceedings. Visualization '97 (Cat. No. 97CB36155)*, 1997, pp. 499–503.

[2] Lee Y, Fathia RN. Interactive music visualization for music player using processing. In: *2016 22nd International Conference on Virtual System Multimedia (VSMM)*, 2016, pp. 1–4.

[3] McGowan J, Leplâtre G, McGregor I. CymaSense: A Real-Time 3D Cymatics-Based Sound Visualisation Tool. In: *Proceedings of the 2017 ACM Conference Companion Publication on Designing Interactive Systems*. Edinburgh United Kingdom: ACM, 2017, pp. 270–274.

[4] Giannakis K. A comparative evaluation of auditory-visual mappings for sound visualisation. *Organised Sound*, 2006, 11, pp. 297–307.

[5] Kosara R. Visualization Criticism - The Missing Link Between Information Visualization and Art. In: *2007 11th International Conference Information Visualization (IV '07)*, 2007, pp. 631–636.

[6] McGowan J, Leplâtre G, McGregor I. CymaSense: A Novel Audio-Visual Therapeutic Tool for People on the Autism Spectrum. In: *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*. Baltimore Maryland USA: ACM, 2017, pp. 62–71.

[7] Ciuha P, Klemenc B, Solina F. Visualization of concurrent tones in music with colours. In: *Proceedings of the international conference on Multimedia - MM '10*. Firenze, Italy: ACM Press, 2010, p. 1677.

[8] Schindler A, Rauber A. An Audio-Visual Approach to Music Genre Classification through Affective Color Features. In: Hanbury A, Kazai G, Rauber A, et al. (eds) *Advances in Information Retrieval*. Cham: Springer International Publishing. 2015, pp. 61–67.

[9] Alexander A, Collins N. Live audiovisuals. In: *The Cambridge Companion to Electronic Music*. 2007, pp. 126–139.

[10] Cooke G. Liveness and the machine: improvisation in live audio-visual performance. *Screen Sound*, 2011, 2, pp. 9–26.

[11] Schwelling E, Yoo K. Automatic 3D modeling of artwork and visualizing audio in an augmented reality environment. In: *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*. Tokyo Japan: ACM, 2018, pp. 1–2.

[12] Szucs V, Kovacs B, Tasnadi B. Music for seeing – visualization of sounds. In: *2018 9th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*, 2018, pp. 123–128.

[13] Fourney DW, Fels DI. Creating access to music through visualization. In: *2009 IEEE Toronto International Conference Science and Technology for Humanity (TIC-STH)*, 2009, pp. 939–944.

[14] Fischinger O. Optical Poem, 1938, https://arthur.io/art/oskar-fischinger/optical-poem (accessed 2 May 2021).

[15] Technologies I. Getting trippy with the iTunes visualizer. *InCare Technologies*, 2017, https://incaretechnologies.com/getting-trippy-with-the-itunes-visualizer/ (accessed 3 May 2021).

[16] Brown RJ. *Audio activated video display*. US4081829A, 1978.

[17] Pramerdorfer C. *An Introduction to Processing and Music Visualization*. TU Wien. 2011.

[18] Dayang R, Awang R, Suziah S, et al. VReST: An image-based virtual reality stress therapy web application. In: *2011 IEEE International Symposium on IT in Medicine and Education*, 2011, pp. 733–737.

[19] Argo J, Ma M, Kayser C. Immersive Composition for Sensory Rehabilitation: 3D Visualisation, Surround Sound, and Synthesised Music to Provoke Catharsis and Healing. In: Ma M, Oliveira MF, Baalsrud Hauge J (eds) *Serious Games Development and Applications*. Cham: Springer International Publishing, 2014, pp. 134–149.

[20] Rossi J, Perales FJ, Varona J, et al. COL.diesis: Transforming Colour into Melody and Implementing the Result in a Colour Sensor Device. In: *2009 Second International Conference in Visualisation*, 2009, pp. 30–35.

[21] chris butscher (@chrisonvizz) is on Instagram, https://www.instagram.com/chrisonvizz/ (accessed 3 May 2021).

[22] Lamb N. *VJING: A NEW ZEALAND CASE STUDY*. Victoria University of Wellington. 2010.

[23] Yakura H, Goto M. Enhancing Participation Experience in VR Live Concerts by Improving Motions of Virtual Audience Avatars. In: *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2020, pp. 555–565.

[24] Yılmaz E, Çetin YY, Erdem ÇE, et al. Music Driven Real-Time 3D Concert Simulation. In: Gunsel B, Jain AK, Tekalp AM, et al. (eds) *Multimedia Content Representation, Classification and Security*. Berlin, Heidelberg: Springer, 2006, pp. 379–386.

[25] He L, Li H, Xue T, et al. Am I in the theater?: usability study of live performance based virtual reality. In: *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*. Tokyo Japan: ACM, 2018, pp. 1–11.

[26] CodeParade. Audio Reactive Fractals (4K) - Let's Go for a Walk, 2019, https://www.youtube.com/watch?v=EkZsPcsV7yE (accessed 3 May 2021).

[27] Ware C. *Information visualization: perception for design, third edition*. 3rd ed. Waltham, Mass: Morgan Kaufmann. 2013.

[28] Elert G. Music and Noise. *The Physics Hypertextbook*, https://physics.info/music/ (accessed 22 April 2021).

[29] Chaudhary K. Understanding Audio data, Fourier Transform, FFT, Spectrogram and Speech Recognition. *Medium*, 2020, https://towardsdatascience.com/understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520 (accessed 23 April 2021).

[30] Rosen S, Howell P. *Signals and Systems for Speech and Hearing*. BRILL. 2011.

[31] Unreal Engine 4.25 Release Notes, https://docs.unrealengine.com/en-US/WhatsNew/Builds/ReleaseNotes/4_25/index.html (accessed 2 May 2021).

[32] Beljajev A. *Dwarf Block Game Development - Dwarf Simulation*. 2020.

[33] Niagara Overview, https://docs.unrealengine.com/en-US/RenderingAndGraphics/Niagara/Overview/index.html (accessed 27 April 2021).

[34] Niagara Key Concepts, https://docs.unrealengine.com/en-US/RenderingAndGraphics/Niagara/NiagaraKeyConcepts/index.html (accessed 27 April 2021).

[35] RGB to HSV/HSL/HCY/HCL in HLSL - chilliant.com, https://www.chilliant.com/rgb2hsv.html (accessed 28 April 2021).

[36] Quadri SMK, Farooq SU. Software Testing – Goals, Principles, and Limitations. *IJCA*, 2010, 6, pp. 7–10.

[37] Jenkins N. *A Software Testing Primer*. 2008.

Appendix

I. Accompanying Files

The ZIP-file, containing the project, the questionnaire and its responses, and the demo video, is structured as follows:

- *AudioVisual* – the Unreal Engine 4 project folder containing the source code (see Appendix II) of the module and the visualizers.
- *AudioVisual.mp4* – demo video of the seventeen example visualizers and four extra visualizers demonstrating the functionality of the module.
- *Questionnaire.pdf* – the feedback questionnaire used to conduct usability testing and to gather feedback for the module.
- *Questionnaire_responses.csv* – the results of the feedback form.

II. Repository

The project source files are available in the GitHub repository at:
https://github.com/Lahe/AudioVisual

The project can be downloaded as a ZIP-file from the repository and imported into Unreal Engine 4. The project was tested on Unreal Engine version 4.26.1.

III. Video Demo

An online video demo of the visualizers can be seen on YouTube:

https://www.youtube.com/watch?v=b1e0I4jdkNE

IV. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Maert Mägi**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,
   Interactive Music Visualization in Unreal Engine,
   supervised by **Madis Vasser**.
2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Maert Mägi

07.05.2021