Goldsmiths, University London

Exploring Unreal Engine 4 as a Live Audio-Visual Tool

Final Report

Philip Kinshuck
5-6-2022

Contents

Section 1: Abstract

This project explores the creation and development of a live audio-visual performance, combining both audio analysis and Open Sound Control data (hereinafter “OSC”) to synchronise generative art driven by electronic music. With many tools available to create audio visuals this thesis will explore the possibility of using Unreal Engine 4 as a new medium for creating generative audio visual in real time.

Exploring Unreal Engine 4 (hereinafter “UE4”) as a tool maximises the use of modern graphics cards allowing the construction of high-resolution computer graphics and the functionality of event driven logic to provide suitable variations in visualisation complementing the movement in sound.

Section 2: Introduction

Audio visualisations play an important role in media such as tv, film, games and in supporting artists work in interactive installations or musical performances, enhancing our experience using audio visual communication provides an extra layer of comprehension and cognition by heightening the awareness of the audience. The power of audio-visual communication can be clearly seen in neurological studies such as the Mcgurk effect [1] for example “the way people recognize a sound can be modulated by visual sensory information accompanying the sound” [2]. Audio visual integration also creates strong emotional links especially when the sound and visual displays are synchronised, reflecting each other both sonically and visually. This has been documented in scientific studies which conclude that “emotion could be enhanced when the audio and visual emotions were consistent” [3].

Electronic music has incorporated visual representations to enhance live performance and the audience experience. Flying Lotus uses large projections and lighting to provide an immersive experience, this is displayed in a recording from one of his concerts [4] where the visual effects draw in the viewer by projecting numerous generative pieces on stage.

The aim of this thesis is to evaluate whether UE4 is an effective tool to create a music visualization which can be used in a live performance setting. The decision to use UE4 was made due to its ability to create high quality graphics combined with the multiple functionalities of audio analysis and OSC to communicate with modern digital audio workstations (hereinafter “DAWS”). Taking full advantage of modern CPU and GPU processing power allows the user to provide high framerate and resolution through modern displays or projectors whilst allowing customizable interactivity in a live scenario. Through development of generative art in UE4 it will be possible to examine whether this tool would be suitable to create audio visuals and communicate effectively with an audience.

Though there are few professional examples of audio-visual performance using UE4 the diversity of the engine and comparable visual scripting system allows the possibility of creating similar effects to professional productions whilst being suited to work efficiently in a performance setting.

Section 3: Background Research

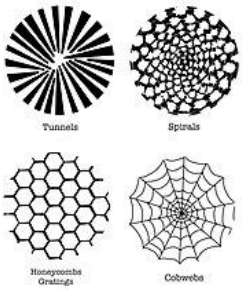
Tepantitla mural [5]

Audio visuals have been an integral part in the creation of expression in musical performance and the combination of art and sound is not a recent technological revelation but has origins dating back to Mayan rituals which combined the use of psychoactive drugs and tribal rhythms to induce a trance like state. It is argued that much of their art and design was influenced by these visions with works depicting fluid organic design with vivid colours, and geometric shapes. Examples such as "the Tepantitla mural in Teotihuacán, dating to 500 CE, shows the Toltec rain god Tlaloc, with priest-like figures bearing hallucinogenic mushrooms springing up where his raindrops fall." [5] .

808 State [6]

In the late 80's the birth of acid house partly reflects these ancient practices developing repetitive beats, tribal rhythms, and the use of psychoactive substances. Sound and art reflect this period and coincide with the development of technology with audio visuals combining early computer graphics and VJ equipment [6].

Henrich Kluver carried out psychological trials with mescaline and noted similar depictions. His "analysis of hallucinatory phenomena appearing chiefly during the first stages of mescaline intoxication yielded the following form constants (a) grating, lattice, fretwork, filigree, honeycomb, or chessboard; (b) cobweb; (c) tunnel, funnel, alley, cone or vessel; (d) spiral." [7]

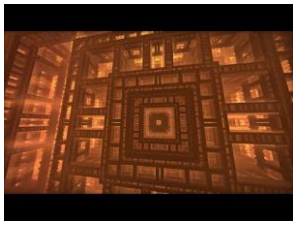


Kluver's four form constants

This work is clearly relevant today and aspects from Flying Lotus's performance [4] display visualisations clearly documented in this early research such as the tunnel, with the graphics developing and transforming from these original forms. Kluver also noted the tendency towards geometrization and that the "forms are frequently repeated, combined, or elaborated into ornamental designs and mosaics of various kinds" or "the elements constituting these forms, such as squares in the chessboard design, often have boundaries consisting of geometric forms". [7]. One technique which often depicts this style of visual uses GLSL raymarching to create 3D infinite geometric patterns which can be programmed to transform over time [8].

Flying Lotus Live Performance Visuals [4]

Currently there are many types of software available for live audio-visualisation and Synaesthesia [9] makes use of GLSL allowing the ability to port code from Shadertoy [10] to create complex generative scenes which react to sound. This program allows the user to integrate features such as midi, OSC and audio analysis. UE4 does allow the use of custom shaders written in HLSL which can allow the creation of such visualisations such as ray marching [8].



Ray marching Fractal Shadertoy [8]

On visiting the ‘Electronic’ exhibition at the design museum I have been inspired to discover new techniques used to convey the connection between electronic music, vision and design leading me to consider how to create professional graphics whilst implementing strong audio analysis features. One exhibit displayed at the exhibition was a music video from Wierdcore [11] who produce exciting visuals for Aphex-Twin [12] and as I discovered later, visuals for Simian Mobile Disco [13]. T-69 [11] is a 3D computer generated visual using midi and OSC to synchronise the visuals with the music taking into account of beats per minute, there is also real-world mapped data which has been implemented into the 3D visuals which are then manipulated creating an effective compilation of footage relating to the theme of the music. The visuals created for Simian Mobile Disco are much simpler but retain high resolution and compel a different style which slowly evolve over time.

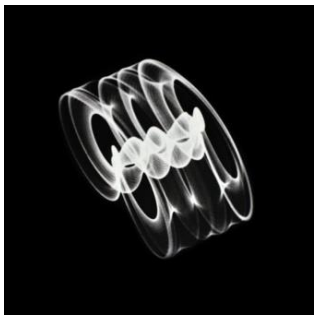
Wierdcore, Aphex-Twin Music Video [11] *Wierdcore Simian Mobile Disco visuals [13]*

On further research of their work, I discovered the software used to create their graphics which included: Adobe After-Effects, Cinema 4D, Max-Msp / Jitter, Quartz Composer and VMDX. Wierdcore’s visuals will set a benchmark for my own production of an audio-visual performance and UE4 has provided me with similar capabilities to the software listed above, such as audio analysis, high-quality graphics, and high performance whilst being open source and freely available to use.

Generative designs often feature as the visualisations for audio visual productions, with their properties modulated by audio and OSC data. Casey Reas [14] has developed numerous works of generative art some digital while others created as physical pieces. His work often explores recreating organic phenomena using emergence whilst some work explores themes relating to the transference of data to visualisation. Generative design is an important aspect when combining the technical skill with artistic process to create original work, and I am interested to explore these techniques through development in a game engine.

Casey Reas, Micro Image Software 1 [14]

Ben F Laposky's [15] work from the 1950's used a cathode ray oscilloscope to visualize and manipulate sound waves. These images demonstrate the close connection between audio and visual representation. Modern techniques pay homage to his ingenious use of modified analogue equipment to create colour and light often using similar digital techniques such as envelope followers, filters, pitch, and amplitude. This early exploration of sound and vision shows the innovation of creative technologists and how it is possible to take a system with a specific purpose and use it for a creative medium, this has inspired me to research and develop this project within UE4. In relation to style the vivid designs on dark background help create depth to his work with the sound waves creating complex organic structures, this choice of contrast can help to clearly define the visualisation and can be applied into my own generative work.



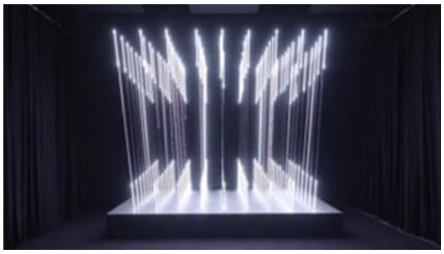
Ben F Lapsky, Oscillons [15]

Colour, sound, and emotion are closely connected and in some cases people with synaesthesia can see colours through the stimulus received via sonic expression. This can also lead to strong visual memories and the choice of colour is an important factor in applying the overall feel of the visualisation as "pleasantness is experienced when synesthetic and actual stimulus features match" [16]. Using a color chart [17] to consider matching the visual with sound can map out the emotional response expected to be perceived from the performance whilst maintaining consistency.

Colour psychology and emotional associations [17]

Games such as GTA 5 [18] have developed audio visualisation within their game by developing a virtual nightclub combining a variety of different effects and visualisations including audio reactive virtual lighting rigs. Based upon real design these lighting effects create ambience and sense of space and this can be compared with designs from modern exhibitions, such as Core by 1024 Architecture [19], who create physical architectural designs with sound controlling patterns of light. Game engines can be a suitable platform to create audio visualisations with the ability to build complex systems, virtual lighting rigs, and employ custom shaders.

GTA 5 Nightclub Demo [18]



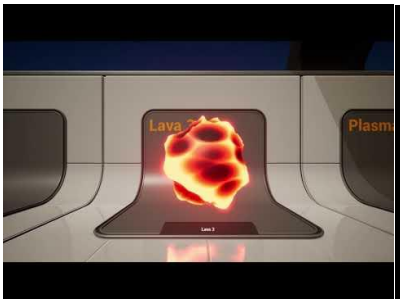
1024 Architecture [19]

UE4 provides all the above techniques, however one drawback is the lack of documentation in relation to using C++, this led me to use UE4's visual scripting system Blueprint (hereinafter "BP") as there is additional learning material currently available as well as open-source examples from creators.

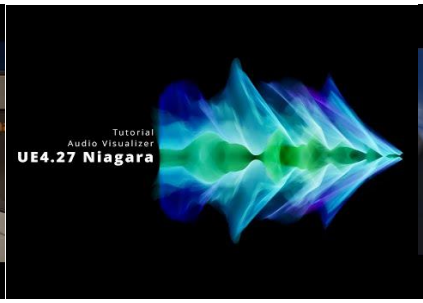
One example of a similar project using UE4 [20] and Ableton Live [21] demonstrates how a DAWS can communicate with UE4 with changes being reflected in graphics in real time using OSC. Ableton also provides the ability to setup custom parameters which can be sequenced or controlled instantly by the user using external midi controllers making a performance feel more intuitive. Communicating with UE4 via OSC is also possible using techniques discovered online [22].

UE4 and Ableton Live [20]

To further understand UE4's abilities user example projects have been researched to understand how to effectively use and manipulate audio analysis features. "Arthurs Audio-Visual Toolkit" [23] showcases methods on how to implement audio analysis tools and methods on passing data between classes. One other main feature of UE4 is the built in Niagara (hereinafter "NI") particle system which is customizable with BP which can add interaction as well as audio spectrum features. Three example projects [24] [25] [26]display the broad range of features and uses which can be implemented.



NI Audio Visualization Project [24]



NI Audio Visualizer [25]

NI Abstract Features [26]

To effectively map sound to visualisation it is necessary to analyse the audio by considering the frequency spectrum and amplitude. Controlling and applying methods to smooth the raw input enables events to be triggered at prominent moments or rhythmic patterns within the song. To effectively analyse audio Fast Fourier Transform can applied be splitting the magnitudes into frequency bands, these magnitudes then need to be processed to create smooth signals which are more predictable allowing for more precise animation. This can be applied using linear interpolation or low pass filtering. An example of this technique can be seen in the audio visualizer NI tutorial [25]. This input data may be used to apply synchronisation by controlling variables within the programme such as colour, velocity, position of an object or trigger envelopes (timeline) to sustain smooth movements.

As this is a live performance the audio will be created in Ableton and with previous experience in live performance this DAWS is suited towards use in this setting, incorporating midi controllers to manipulate sound provides physical controls as well as allowing customizable parameters. One tool I have decided to use is Live Grabber from show sync [27], this plugin includes modules to grab and send OSC data.

Live Grabber plugin for Ableton Live [27]

Many tools have been considered for this project and experiments early on have been made using openframeworks as the base framework, combined with addons such as Maxim, Supercollider and Max for live. Early demos I created worked effectively but did not inspire me graphically to continue developing using these technologies.

Demo Openframeworks *Demo UE4*

Section 4: Specification and Design

The aim of the project is to apply an effective audio-visual production suitable for display in a nightclub and projected onto a large screen, the visuals would help to enhance the total experience with the aim of the graphics to contribute towards synchronizing the audio performance with matching visual elements. The performance should take place in a dark room allowing the visuals to provide subtle lighting helping to draw a deep and calm ambience. The colors of each piece have been considered to provide consistency and a color chart [17] has been used to help mapping the colors. The use of blue purple and black help to draw on emotions such as tranquility, spirituality with the black background creating a dramatic mood in contrast with the lighting and color. This design choice has been inspired by Laposky's oscillons [15].

Structured Overview:

On initialization it is necessary to place the main actors into the level prior to loading. These main actors are responsible for communication, analysis, and spawning the visualizations:

Main Level:

Actor Spawner Blueprint:

The main controller of the project. This blueprint class controls which visuals are displayed and communicates between Blueprints receiving incoming OSC, audio data as well as triggering custom events to spawn or destroy actors and select camera.

Audio Analysis Blueprint:

Activates the microphone input and creates an array of selected frequencies.

Analysis Smoother Blueprint:

Receives the raw audio analysis array from the audio analysis class

OSC Receiver:

Receives OSC data from Ableton Live and applies logic by triggering events depending on which note, or parameter has been sent.

Camera Spawner:

Spawns and positions cameras and receive data from the actor spawner declaring which camera should be triggered and which target the camera should focus on.

UE4 Main Level components:

Blocking Volume: UE4's blocking volume creates an area which cannot be crossed and provides access to the physics engine to create a confined space.

Exponential Height Fog: UE4's fog effect allows the use of attenuation and atmospheric lighting.

Brief Description of Visuals used:

Visual 1: Elastic Grid visual uses NI particle system and is triggered by OSC from Ableton Live. This visual was one of my initial developments and features in the introduction and towards the end of the track. The ability to control the overall tightness of the grid makes this visual diverse and can help to shape various emotions of the sound, either at quieter points in the song where the movement is more fluid to louder times where the grid can break up and create a more dramatic emotion.

Visual 2: Noise wave grid uses NI particle system and has a custom audio analysis modules built to react to low frequencies. The simple emissive colour creates a strong contrast and helps to define a sharp change of movement in the song, in addition the gyroid shader was placed behind and the opacity is controlled through Ableton Live creating a new depth and preparing the audience for a change of scene.

Visual 3: Swarm NI particle system with custom audio spectrum module combines frequency to particle position this sketch was not used in the final cut but may be developed in later work.

Visual 4: Audio reactive visualisation using NI particle system based on audio frequency analysis this sketch displays frequency and additional reaction to low frequencies. This visual appears towards the end of the track and implies a more euphoric state helping to begin to slow the movement of the song and apply a brighter colour to the room.

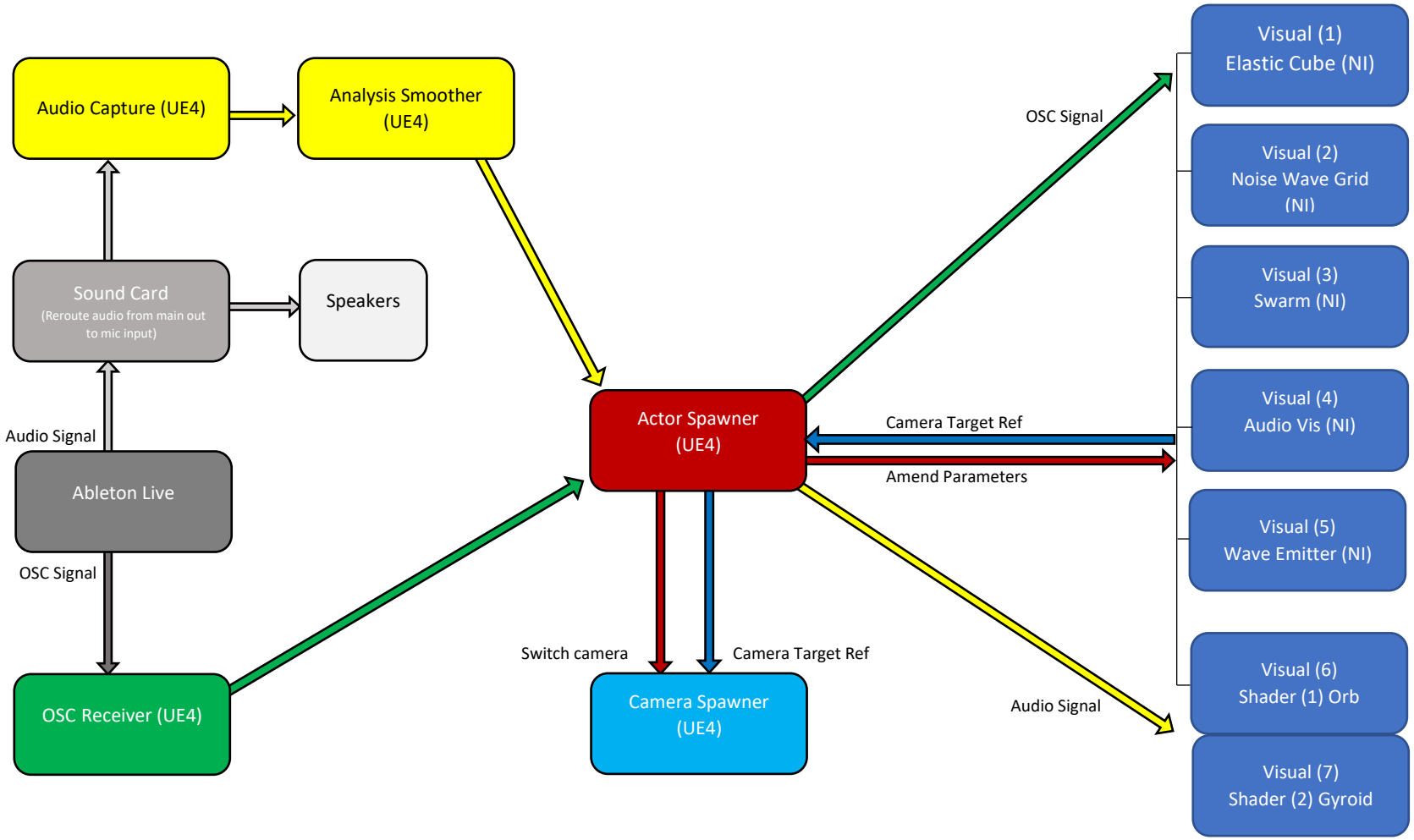
Visual 5: NI wave emitter grid based on visual 2 was not used in the final production but may be further developed.

Visual 6: Ray march shader alien orb using HLSL reacts to audio using a selection of timelines (envelopes). This sketch features at the end of the song during quieter movements in audio and depicts natural elements on the see floor, the audio analysis triggers long flowing envelopes to apply smooth change over a longer period with various frequencies triggering different properties of the shader.

Visual 7: Ray march shader gyroid using HLSL uses a combination of audio analysis to trigger timelines and sequenced parameters from Ableton Live which control predefined scalar parameters. The visual displays complex organic structures with a high level of detail. I have chosen to use this sketch in two parts of the song, one reduces the scale creating a more spatial structure whilst the second variation increase speed and produces a more complex structure.

The diagram below depicts the flow of data in this UE4 project.

Section 5: Flow chart of Data



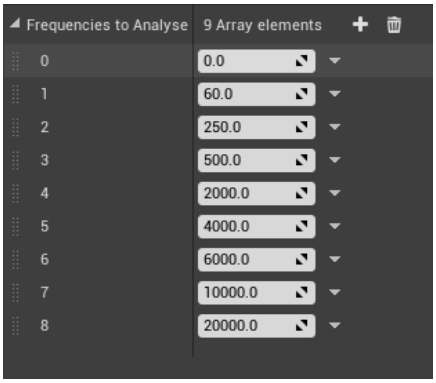
Section 6: Low Level Technical Detail

Audio Analysis Blueprint:

As this project will focus on live performance it will be necessary to take an external audio signal by patching live audio into the line input of a sound card.

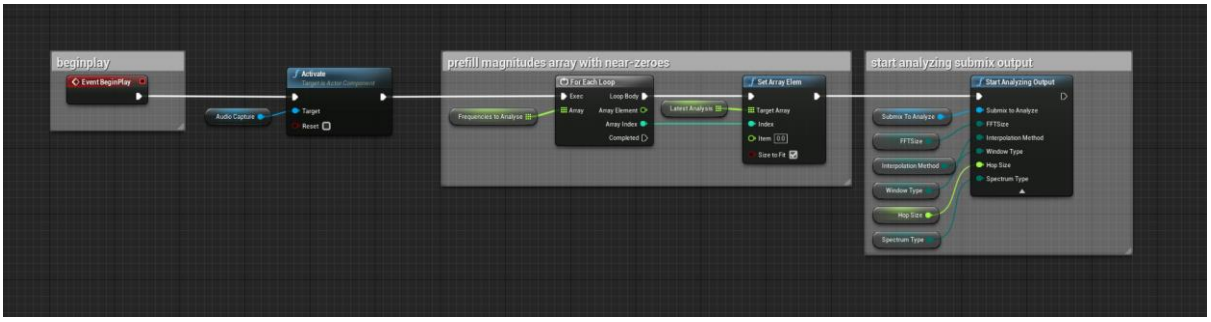
To activate audio input in UE4 it is necessary to create an actor which contains an audio capture component. The audio capture component can be automatically initialized as soon as the game engine has launched the level (event begin play) using the 'activate node'.

To create an array of the frequency spectrum firstly it is necessary to declare the number of frequency bands and the designated frequencies to place within these bands (frequencies to analyse).



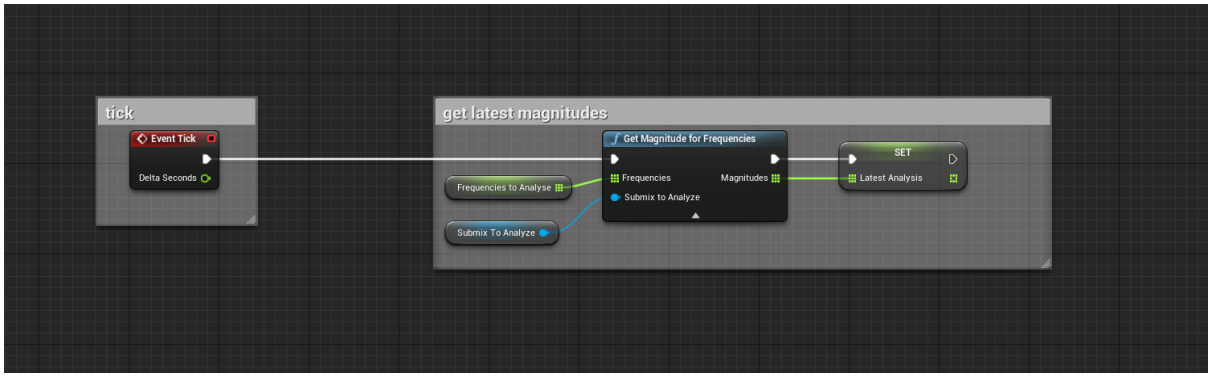
Selecting the frequencies to analyse

Once the frequencies have been declared we prefill a new array (latest analysis) with zeros and set the start analysing function to begin.



Audio Analysis Event Begin Play

In the 'Event Tick' node (update) we pass the selected 'frequencies to analyse' into the unreal function 'Get Magnitudes' and select the 'sub mix to analyse'. This function returns an array of magnitudes which are then applied using 'SET' to the 'Latest Analysis' array.



In a live performance situation, the same computer may run UE4 whilst performing the music through Ableton Live, however once audio input has been activated in UE4 a feedback loop is created. To stop the feedback loop, it is necessary to create a sub mix within UE4 which leads into a muted sub mix, this prevents the feedback loop whilst still allowing the main output from Ableton Live to be sent to the speakers.

Analysis Smoother Blueprint:

By applying steps to smooth the raw magnitudes and create an envelope and threshold it is possible to tune the audio signal to effectively apply the data to animations.

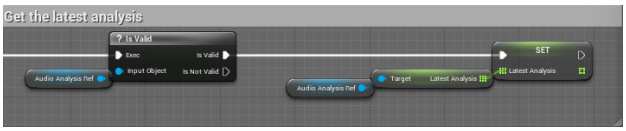
This script was followed from a creator's example [23] from which I have studied and applied to my project.

Event Begin Play (setup):

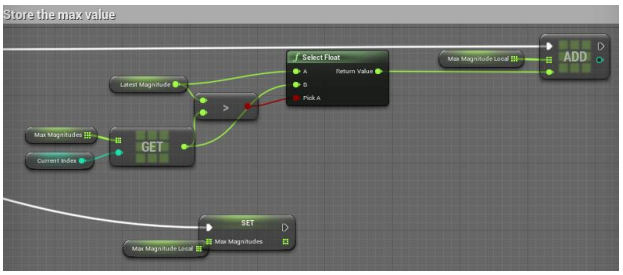
- Check that the audio analysis actor has been initiated using the '? Is Valid' node and pass by reference 'frequencies to analyse' array.
- Initialize an array to hold the maximum magnitudes

Event Tick:

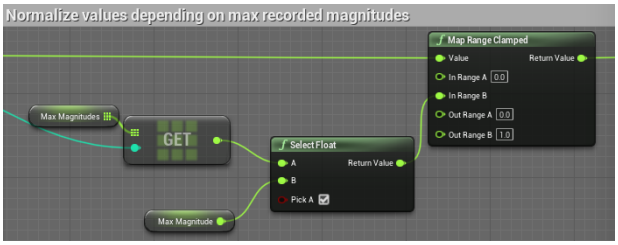
- Check that the Audio Analysis actor has been initiated using the '? Is Valid' node and pass by reference 'Latest Analysis' array.



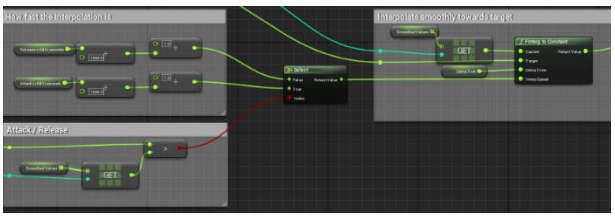
- *Function - StoreMaxMagnitudes*. Store the Maximum magnitudes by checking if the latest magnitudes is greater than the currently stored magnitude.



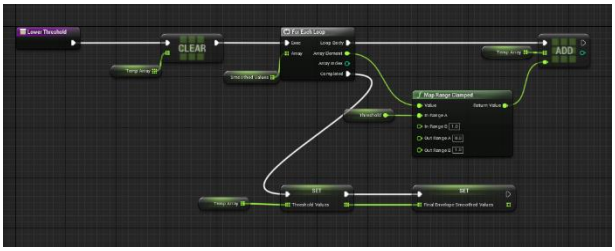
- *Function – NormalizeMagnitudes*. Normalize the magnitudes using the 'Map Range Clamped' function



- *Function – Smoothing*. Smooth Values by interpolating between normalized magnitudes and interpolated values. The attack and release of the envelope is created by changing the interpolation speed.



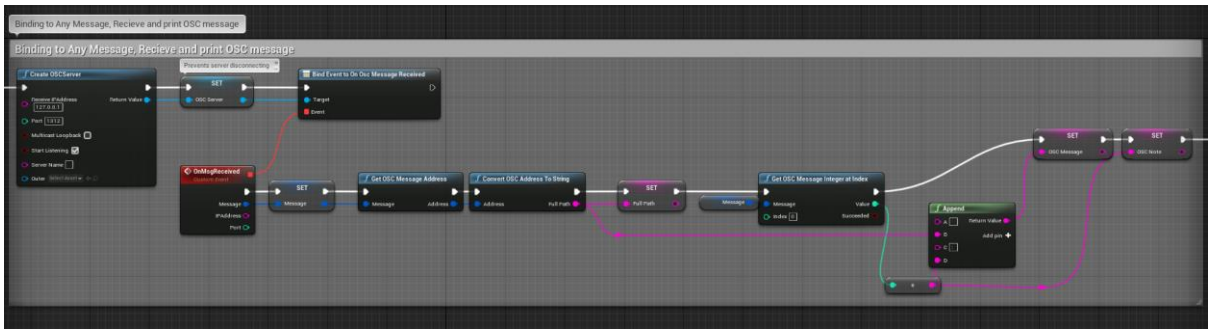
- *Function – LowerThreshold.* Apply a lower threshold to the smoothed values by adjusting the lower threshold of 'MapRangeClamped'.



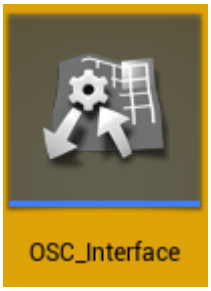
OSC Receiver Blueprint:

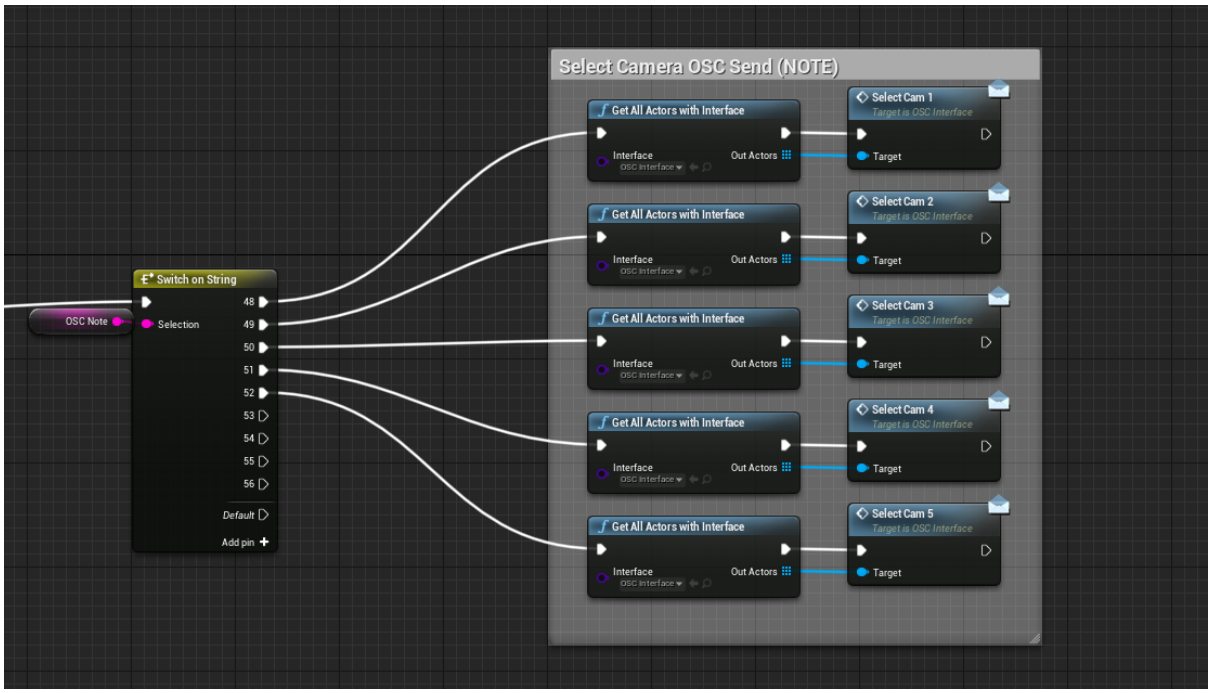
The OSC receiver actor takes OSC signal from Ableton Live, it is then possible to create a custom Interface within UE4 to allow custom events to be triggered.

The image below displays how the OSC receiver has been setup. It is possible to choose to append the message as a string or access the value received.

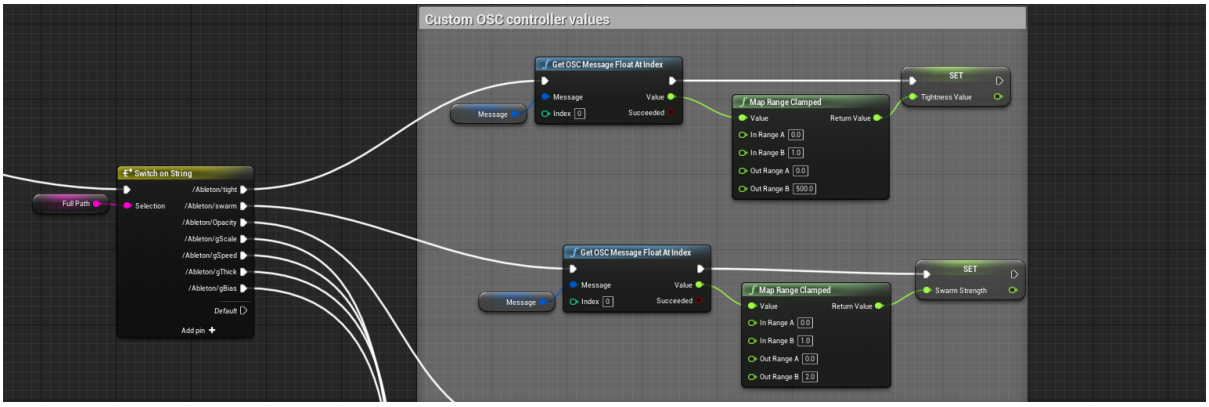


The image below displays how the signal is processed for selecting the camera. Firstly, we check for incoming midi notes and use the 'Switch on String' node which access's the first element in the array of OSC messages. If the message has been received the custom interface triggers an event which can be accessed by any actor required.





This image depicts how it is possible to access parameter values from Ableton Live.

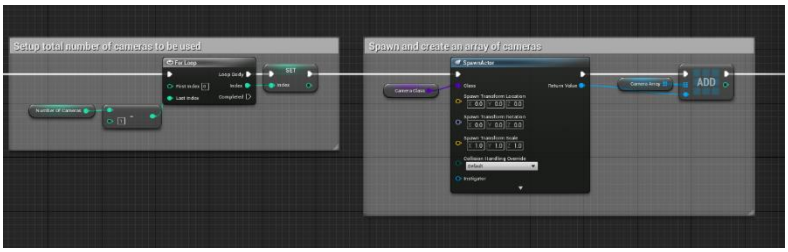


The same method is used to ‘Switch on String’ however we now access the value at index[0] of the OSC message array, ‘GetOSCMessageFloatAtIndex’, clamp the value and set this value to a variable within the OSC receiver class. In UE4 it is possible to access variables from other classes by passing reference to an actor, making it possible for any actor to access this data.

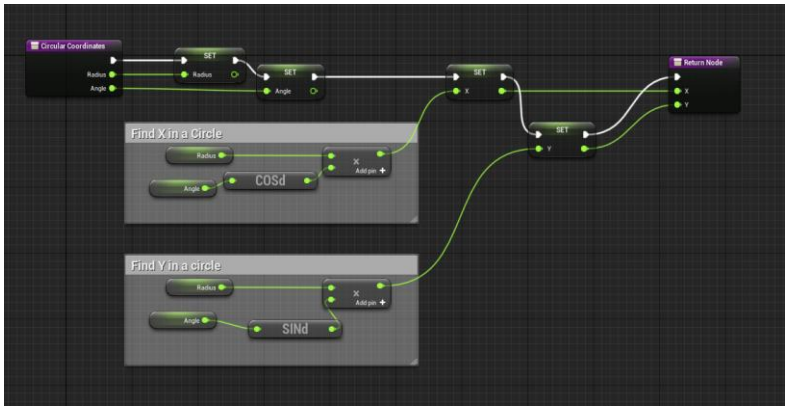
Camera Spawner Blueprint (BP)

The camera spawner places cameras into position within the scene

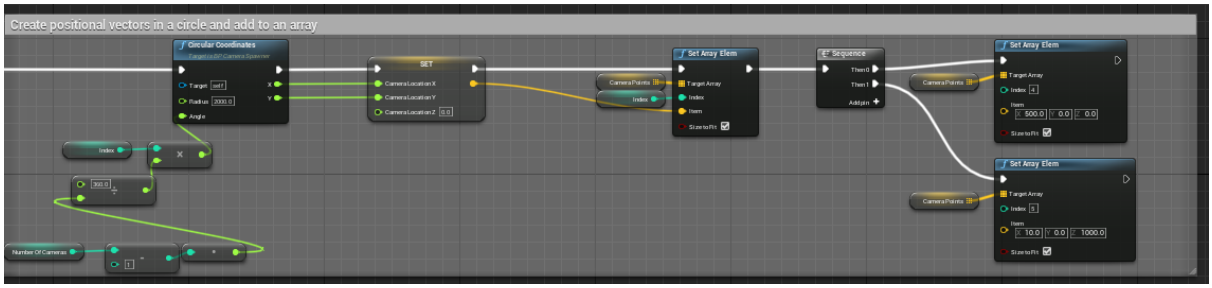
Event Begin Play:



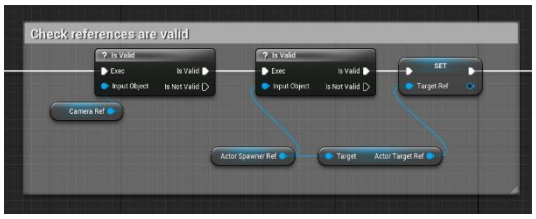
Using a for loop we can initialize the total number of cameras required. Then it is possible to spawn the camera actor adding them to an array.



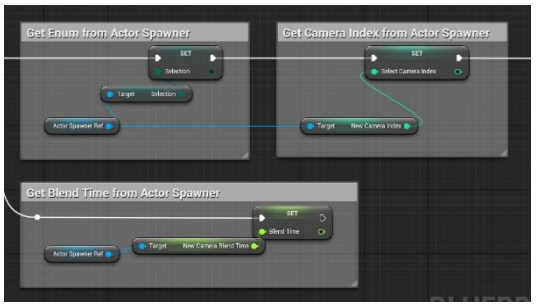
By creating a custom function 'CircularCoordinates' which takes inputs 'Radius' and 'Angle' it is possible to place numerous cameras into position on initialization, storing these vectors into an array using 'Set Array Elem'. Additional positional vectors were created later to add various camera perspectives.



Event Tick:



The camera spawner references two classes, the camera actor BP, and the actor spawner BP. Both must be checked for validation prior to launch to avoid errors in runtime.



Keeping the logic contained in the actor spawner keeps the code modular.

- **Selection**: *Enum to define which visual is currently selected.*
- **New Camera Index**: *The camera to select.*
- **New Camera Blend Time**: *The blend time to move between cameras.*
- **Actor Target Ref:** *This enables the camera spawner to target the currently selected actor and access information such as position. This is necessary as the visuals are spawned and destroyed.*

Set View Target with Blend:

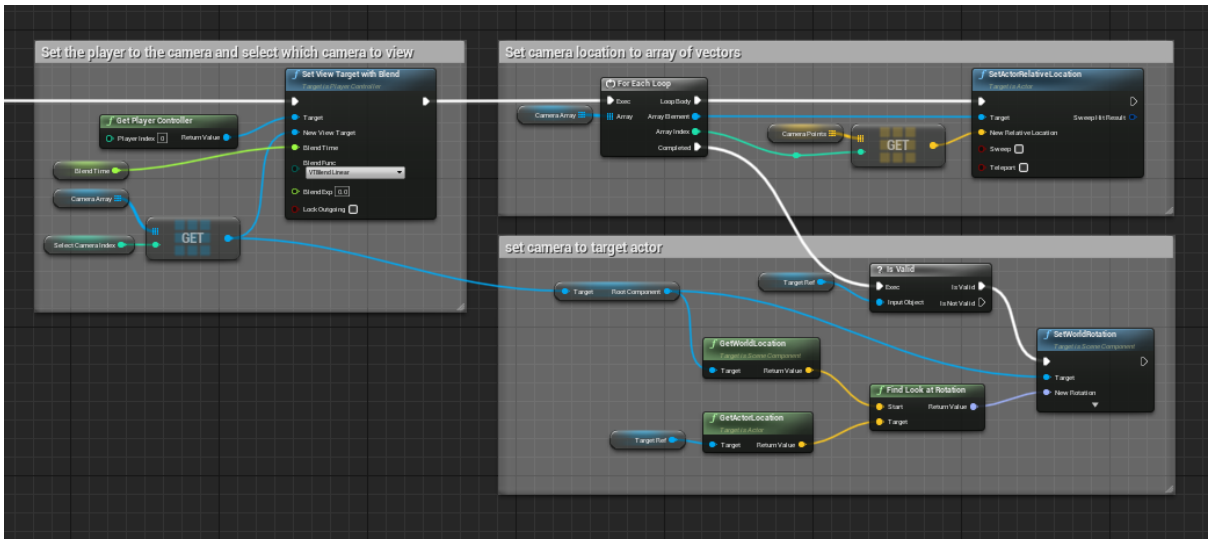
- To initialize the camera, it is first necessary to set its target using ‘GetPlayerController’ node, by default player index is 0.
- Blend time can now be set accessed from the Actor Spawner, allowing variation in time taken to switch between cameras.
- Using the ‘GET’ node it is possible to access an element in the camera array. The index is accessed from the Actor Spawner and allows the user to change camera selected when required.

Setting Camera Location:

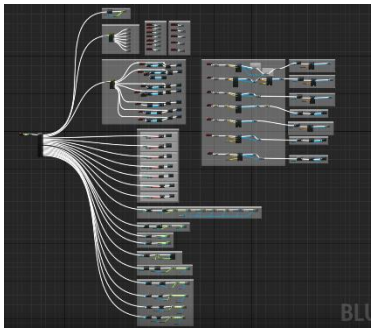
- A for loop iterates through each element in the camera array.
- Using ‘SetActorRelativeLocation’ we set the ‘new relative location’ using an array of vectors created in the event begin play.

Setting Camera target:

- ‘GetWorldLocation’ node regards to the cameras location here we pass reference to the current camera selected in the camera array.
- ‘GetActorLocation’ node requires a reference to the actor selected, this reference is taken from the actor spawner and is updated when a new visual is selected
- ‘FindLookAtRotation’ and ‘SetWorldRotation’ nodes allow the camera to track the positional vector of the actor, this allows the currently selected visual to always be in the center of the screen.
- One difficulty experienced with the camera spawner was receiving references for the camera target this need to be set when the actor is spawned, and the reference passed from the actor spawner to the camera spawner.
- The blend time also needed to be adjusted with the addition of the shader visuals, by setting the blend time to 0 this instantly set to the next position and scene to create smooth transitions.



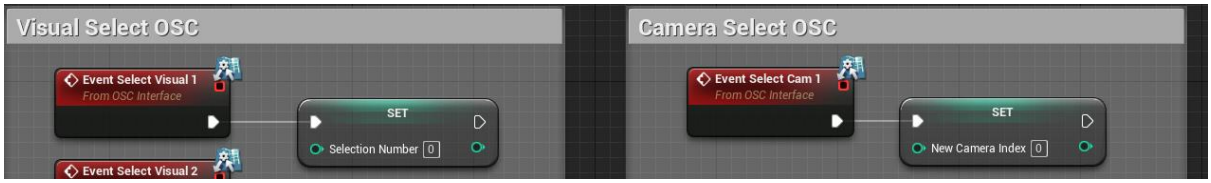
Actor Spawner Blueprint (BP):



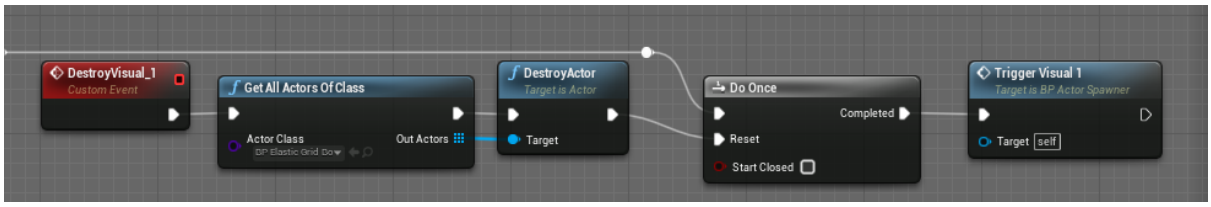
The Actor Spawner is the main control point and communicates with all the actors within the project.

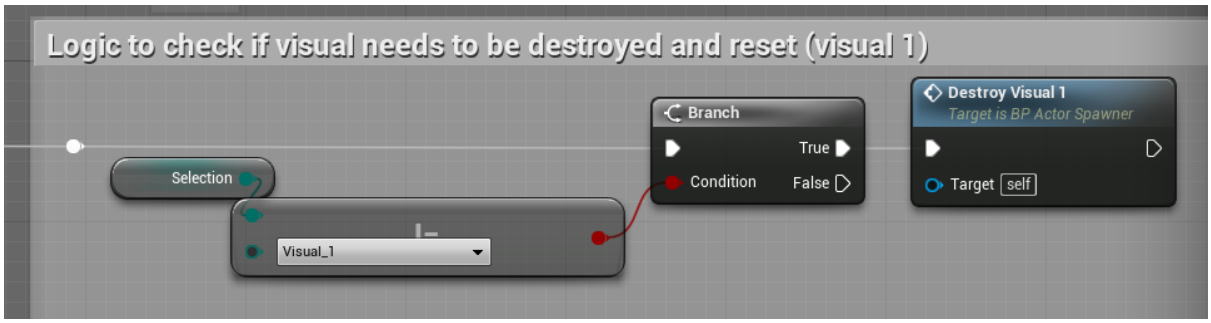
Features:

- Receives smoothed audio data from the analysis smoother BP.
- Enums are used to define which visual should be selected. The selection is initially made in Ableton Live processed in the OSC receiver and triggered by the OSC interface which switch's which visual is currently selected.

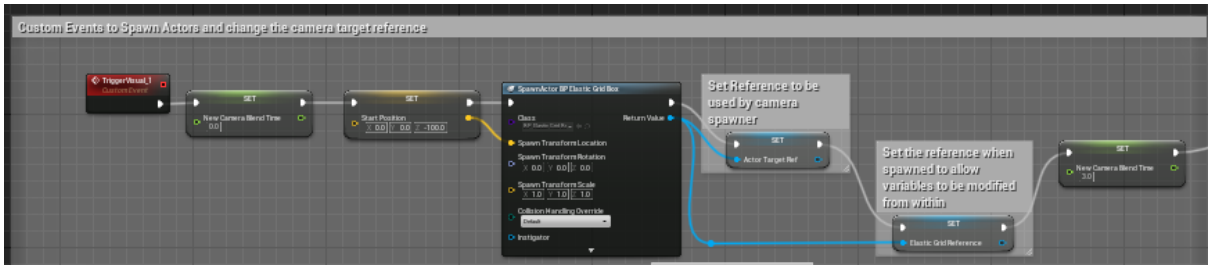


- Once the visual has been selected the "Do Once" node is used to trigger a custom event
- The destroy visual may also be selected if the visual is not selected.



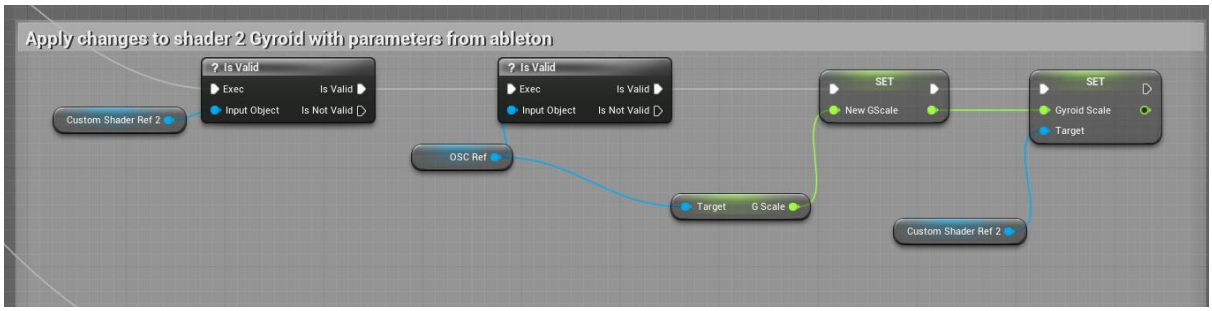


- The custom event then sets the camera blend time, spawns the actor at the selected position and sets the actor reference which is necessary to change parameters within the selected blueprint and apply the camera target position.

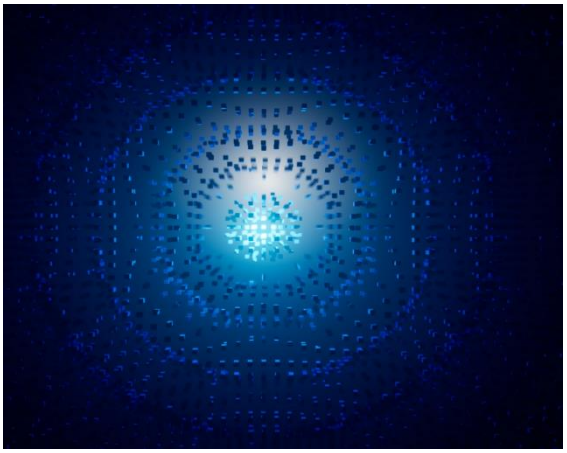


The actor spawner is also responsible for passing updated parameters received from the OSC receiver and sent to the selected blueprint allowing for precise control over movement and

synchronization from Ableton Live. The image below shows that the references must be checked to prevent any errors, the updated variables within the OSC receiver are targeted then the variable is updated within the selected Blueprint.



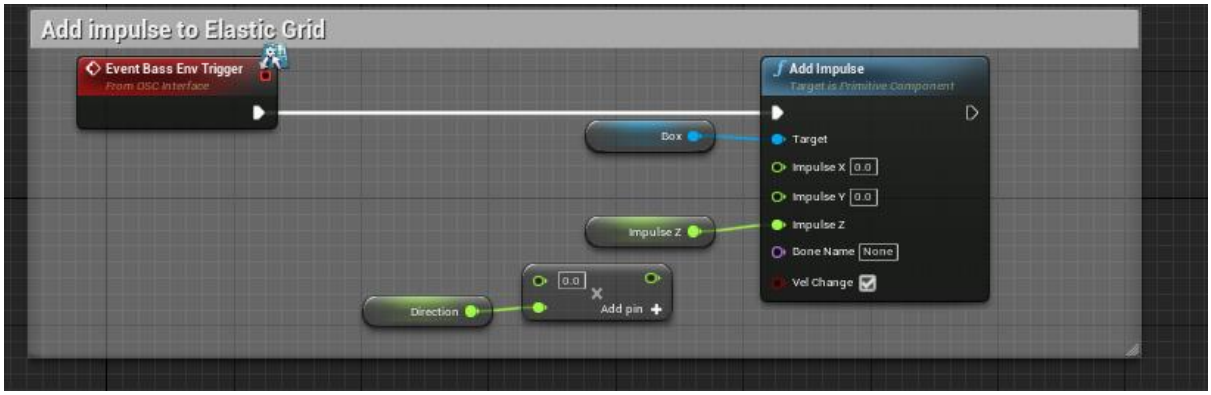
Visual 1: The elastic grid visual uses NI particle system and deforms when physical force is applied. This visualization is based on the use of standard NI modules.



This module contains custom user parameters which can be manipulated in real time through the actor.

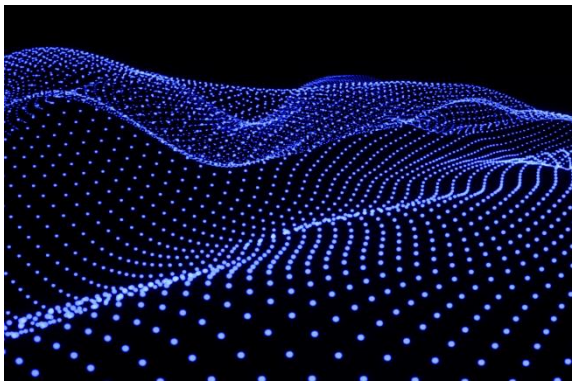
The NI system is place within a Blueprint where it is possible to add several features to help with interaction. A box extent is used to allow hit detection and user variables are created to allow customization of size weight and position.

This BP receives OSC data from Ableton Live's envelope follower and the signal triggers a custom event, the "Add Impulse" node creates movement by adding velocity. The grid reacts when a hit is detected, and a fluid movement is created through the grid.



Lighting is also added to the center of the grid to help create an atmospheric effect. This effect was first developed using an online tutorial [26].

Visual 2, Visual 5: NI Noise wave grid

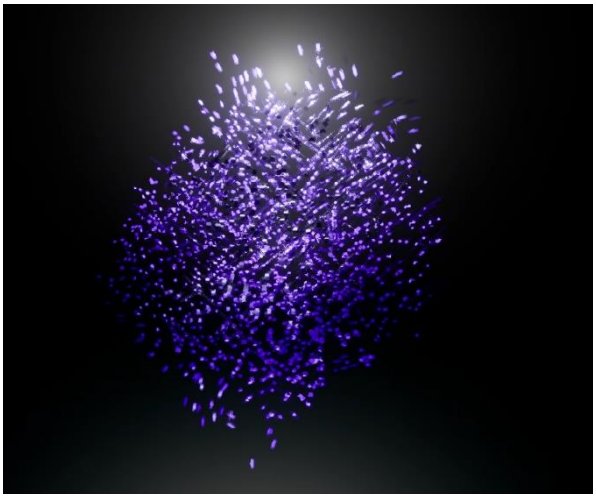


The noise grid uses the NI particles system and several built-in modules. A custom noise function has been created here and applied in NI to create a static frame as the particles will drift apart losing their form.

Here we sample the noise and apply it to particle position creating noise deformation.

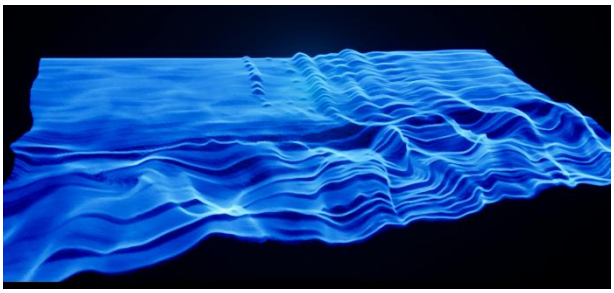
To animate the movement connecting to the bass frequency it is possible to create a custom module which access's the audio spectrum node. By applying the power function to exponentially increase the difference in value allows for a more pronounced variation in amplitude, helping to define the beat. This new value can now be added to the drag module within NI creating a smooth wobbly movement when the bass amplitude is high enough.

Visual 3: NI Swarm

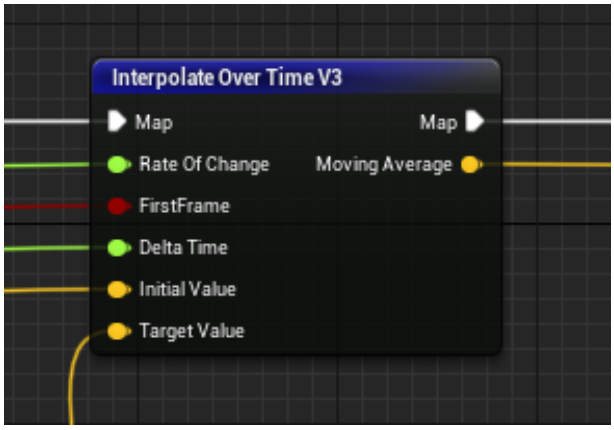


The audio reactive particle system uses NI modules however a custom Blueprint has been created to accelerate particles depending on their position within the audio spectrum. By adding scaling and amplitude to the velocity an explosive swarm like effect is created.

Visual 4: Abstract audio visualization:



This visualization applies the audio spectrum to the NI particles system and is based on tutorial [25]. A custom module must be built to apply the normalized execution index and spawn the particles in a row. In another custom module we can now apply each particle to its relevant position in the audio spectrum. Smoothing can be applied from within NI using the interpolate over time node

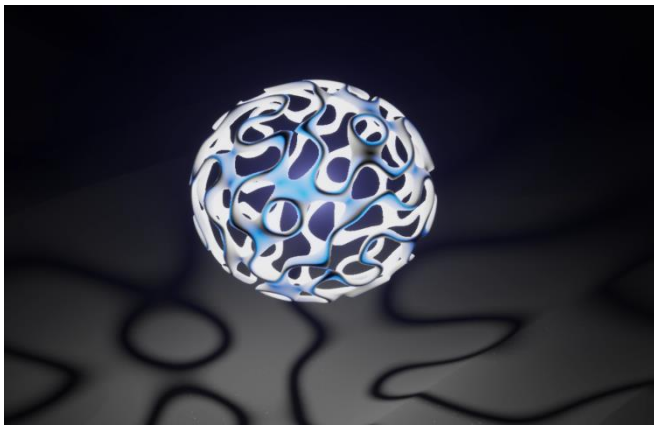


To create the wave like effect a new particle system is spawned from the original and by adding gravity and noise the initial wave deforms over time. Further to this, the same technique was used as in visual 2 and the previously built module can be added to create additional movement when analyzing the bass frequencies.

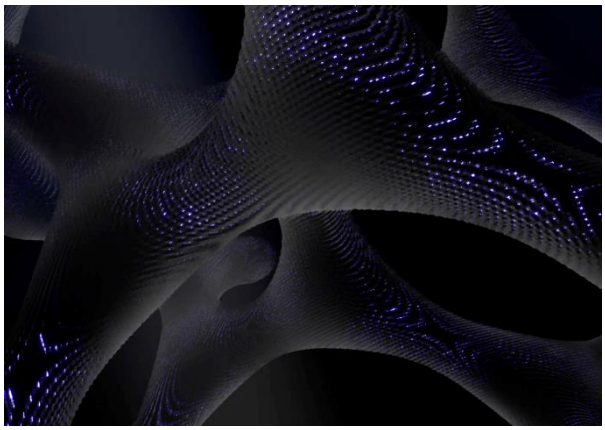
Lighting was added after extensive personal testing to help continue the color theme with attenuation visible in the dark background.

Visual 6 (Orb) / 7 (Gyroid): Ray marching in HLSL

It is possible use HLSL code in a separate file and link this directly to the material created in UE4.



Visual 6



Visual 7

Parameters are passed into the shader through the custom node essentially creating the uniforms outside the Unreal shader file (usf.). Two ray marching pieces have been created following from online tutorials [27], this was then converted from GLSL to HLSL, both explore the use of gyroids. Using techniques such as boolean intersection it is possible to cut shapes out of an image (visual 6) or apply details by adding or subtracting slight variation and applying offsets (visual 7). This code start with basic ray marching functions and additional functions have been created such as the 'sdGgyroid' function.

To create a gyroid we take the dot product of two vectors the first vector being sin(p) (with p being the 3D marching position) and cos(p.zyx), the xyz coordinates are switched around to create the gyroid shape.

```
return abs(dot(sin(p*sinRatio), cos(p.zxy*cosRatio)) + bias) / (scale*offset) - thickness;
```

In the main function of the shader, it is necessary to check if the point has been hit and then we can apply details, material, and color.

With UE4 it is possible to have a high number of steps and distance increasing the amount of detail that it is possible to render on screen without losing framerate.

Within the material in UE4 scalar parameters are initialized, these parameters are accessible from any BP.

The shaders are applied to a plane in 3D space and the parameters can be modified. This caused some errors when changing camera position as the plane would sometimes not be visible from head on

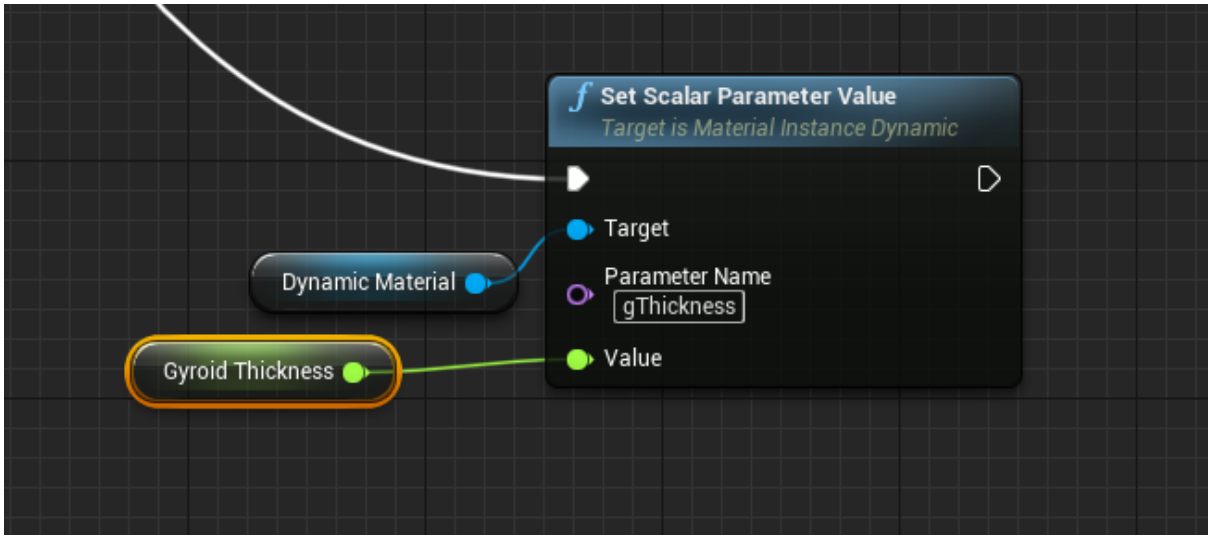
```
if(d<MAX_DIST) { //if we hit the object apply color / lighting
  float3 p = ro + rd * d;
  //float dif = Common.GetLight(p);
  float3 n = Common.GetNormal(p);

  //material
  p = Common.Transform(p); // move wrold position

  float dif = n.y*.5+.5;
  col += dif*dif; //float3(dif,dif,dif);

  float g2 = Common.sdGyroid(p, bScale + 5, bThickness, bBias, bSinRatio, bCosRatio);
  col *= S(-.1, .1, g2)*gLightShadow; //place light shading on gyroid

  float crackWidth = -.02 + S(0., -.5, n.y)*.04; //change the width of the crack depending on the normal, at the botto
  float cracks = S(crackWidth, -0.03, g2); //add color to the crack

  //animate the color of the cracks using a gyroid
  float g3 = Common.sdGyroid(p+t*.1, bScale, bThickness, 0, bSinRatio, bCosRatio);
  float g4 = Common.sdGyroid(p-t*.15, bScale*.5, bThickness, 0, bSinRatio, bCosRatio);
  cracks *= g3*g4*10+.02*S(.2, .0,  n.y);

  col += cracks*float3(.1, .04, 1.5); // add color to the cracks

}
```

visual 7

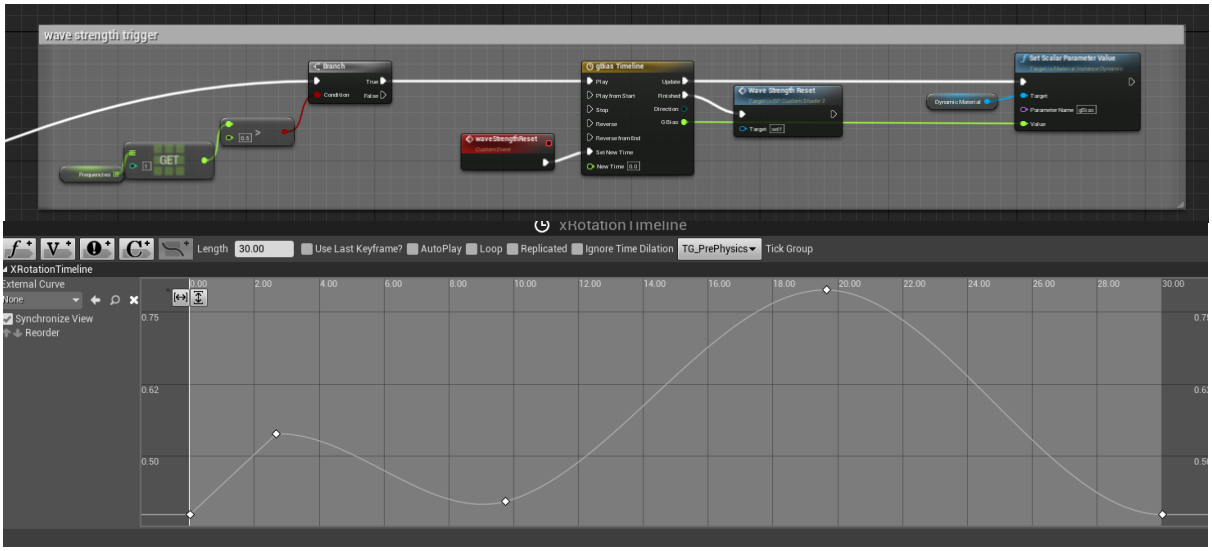
ruining the illusion. To rectify this issue, I have added to logic to change the blend time so when the shader visuals where spawned the camera instantly moved position and focused on the plane head on.

There are two different techniques used to modify the parameters:

The first takes parameters from Ableton Live passed to UE4 and the OSC receiver BP-> actor spawner BP -> material BP. We create a dynamic material instance in UE4 to allow real time changes to be applied in the shader. The values received directly update the scalar parameter as seen in the image below.



The second techniques use audio analysis and applies an envelope when the threshold is reached on a selected frequency band. The image below shows how we access the bass frequency from the analysis smoother, check to see if the threshold has been reached and trigger a timeline.



I have created an event that only retriggers the timeline when it has finished this is to allow for smooth control over animation over longer periods of time.

List of features:

Visual 6:

- waves strength(timeline)
- The size of the orb (timeline)
- The camera position x and y coordinates (timeline)
- Volumetric light strength (timeline)
- Wave direction (timeline)
- Ground position (timeline)

Visual 7:

- bBias (layered detail in the gyroid shader) is triggered by the timeline, this is tuned to the higher frequencies and is apparent when the high hat is playing creating a pulsing light effect.
- The scale of the gyroid controlled from parameters within Ableton Live.
- Gyroid thickness controlled from parameters within Ableton Live.
- World transform speed controlled from parameters within Ableton Live.

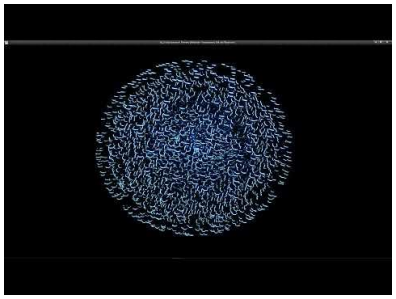
Using ray marching proved to be extremely effective, and the result created a new depth to the entire project emulating infinite space and organic details. This is an area I would like to continue to study and explore beyond this project as the visuals can display much more complexity whilst retaining high framerate.

Section 7: Testing

Testing the project visuals has been a continuous process from the start to finish with work being constantly shown among peers, friends, and family, this has helped to develop the project and subtle changes have been made. As an artistic endeavor it has been difficult to create a user testing form, I have therefore provided the development of the project in videos with statements taken at the time.

Prototype testing:

December 17th, 2021.



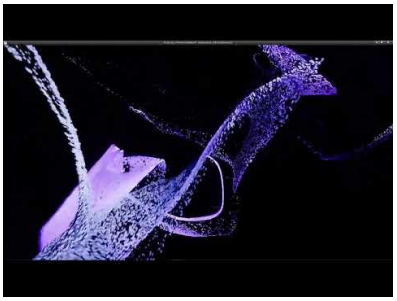
This is the first sketch created in UE4 using NI particle system analyzed an early demo of the song in Ableton Live audio reactivity has been incorporated into NI.

“God that’s hypnotic!! Is the music yours as well?”

“Looks like exploding rice. Nice audio, sounds like film music”

Response: Continued to experiment with NI particle system and positive reaction to the audio led to me to continue developing the track.

January 13th, 2022.



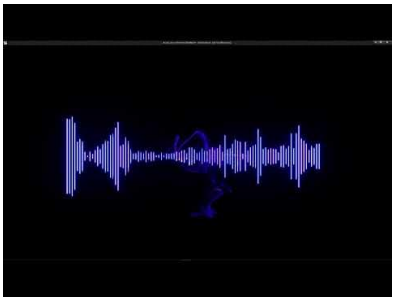
Continuing development with NI particle system.

“Wow that’s cool”

“No real structure needs to develop more over time”

Response: generally positive reaction to the visuals however more variation and form needed to be developed.

January 22nd, 2022.



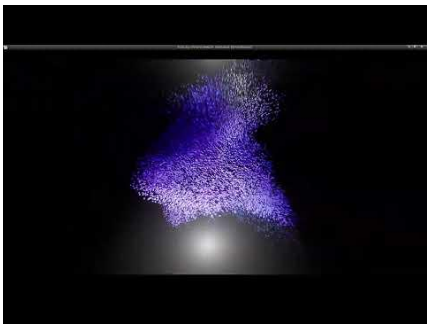
Combining particle system with audio analysis and analysis smoother Blueprints, Song Fuego (Julien Jeweil Remix) by Dubfire and Oliver Huntemann

“Good quality graphics”

“Looks like an audio visualizer”

Response: At this point in the project, I decided to focus on trying to develop more generative work and move away from audio visualization and begin to integrate various camera perspectives.

March 11th, 2022.



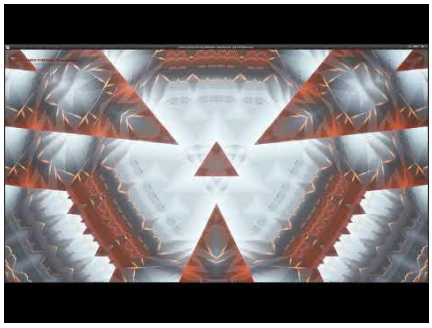
Presentation demo test, combined camera controls, and OSC.

“Great quality Graphics”

“Very 90’s not enough connection with the audio”

Response: Focused on implementing more interaction with audio and OSC.

March 21st, 2022.



First experiment porting GLSL Shadertoy into UE4, no audio or OSC had been implemented.

“Sick!”

“Not enough interaction with sound”

Response: With an overall positive response I decided to continue exploring writing shaders in UE4.

March 28th, 2022.

Final presentation demo includes more audio interactivity fractal shader and ray march shader.

Response: General response from the presentation was positive however there was still inconsistency with the colors and matching visuals, I decided to continue exploring ray marching and thinking about adding more interaction into the shader visuals.

April 26th, 2022.

Final project video, there were numerous changes and fine tuning of color, speed (shader gyroid visual), and length of performance.

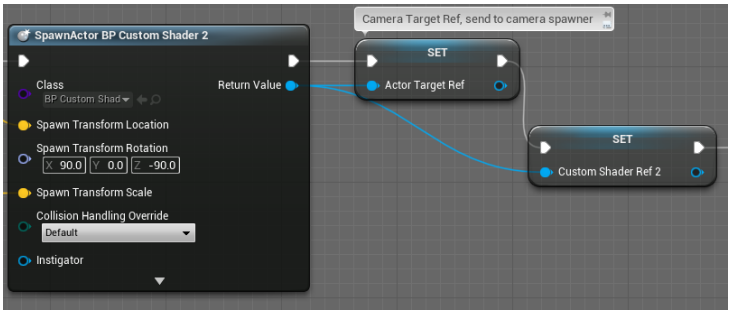
“that’s impressive so many variations with a great soundtrack”

Watching the development of the project from start to finish shows how testing was successful in that all criticism was taken on board and the project amended at each stage displaying refinement and progress.

Section 8: Debugging

Learning to debug in UE4 did take considerable time as I have no experience with Blueprint. When trying to find an error the console explains in which BP the error has been found, however understanding why the error occurred took much time searching for answers online. One of the main problems faced early on was no reference was found when trying to communicate between BP's this was easily fixed by using the 'isValid' node, this is like checking if a variable or array is not NULL. Most of my knowledge gained was through watching online tutorials and searching forums to consider why there where errors.

Problem solving was difficult at first trying to understand the flow of data and the structure of UE4 and much time and effort was put into searching for similar problems online. The first main problem I faced was understanding the state of the audio in UE4 and when using the audio analysis tool I received a feedback loop, this was due to the fact I was running Ableton Live and UE4 on the same machine and the audio output was patched to the microphone in on the sound card whilst being active in UE4. Creating a muted audio mix within UE4 stopped the feedback loop allowing the audio to come directly from Ableton Live which retaining higher quality audio than the processed mic input.



Another problem difficult to solve was passing the reference of an actor. This was most relevant when trying to target a newly spawned actor with the camera and was easily fixed by updating the reference in a single variable once the actor had been spawned.

HLSL was difficult to implement and the slight variation in code style to GLSL took some time to complete without error. Once I was able to link shader files to UE4 I discovered that there was no main function in HLSL and instead a struct can be used to place functions in which are called using the named prefix. Error logs appear in the console as in most code editor and the file can then be edited. One benefit of using HLSL in UE4 is that it almost instantly compiles the code.

```
struct _CommonLib
{
    //------------------------
    float2x2 Rot(float a) {=}
    //------------------------
    float smin( float a, float b, float k ) {=}
    //------------------------
    float sdCapsule(float3 p, float3 a, float3 b, float r) {=}
    //------------------------
    float sdCylinder(float3 p, float3 a, float3 b, float r) {=}
    //------------------------
    float sdTorus(float3 p, float2 r) {=}
    //------------------------
    float sdBox(float3 p, float3 s) {=}
    //------------------------
    float sdGyroid(float3 p,float scale, float thickness,  float bias, float sinRatio, float cosRatio) {=}
    //------------------------
    float3 Transform(float3 p) {=}
    //------------------------
    float GetDist(float3 p) {=}
    //------------------------
    float RayMarch(float3 ro, float3 rd) {=}
    //------------------------
    float3 GetNormal(float3 p) {=}
    //------------------------
    float GetLight(float3 p) {=}
    //------------------------
    float3 R(float2 uv, float3 p, float3 l, float z) {=}
    //------------------------
    float3 Background(float3 rd) {=}

} Common;

ro.yz = mul(ro.yz, Common.Rot(-m.y*3.14+1.)); //rotation
```

Section 9: Evaluation

This project has developed over time and with the initial prototype using openframeworks I am confident that I have made the correct decision to use UE4. Whilst I have not coded the project in C++ the visual scripting system took much time to learn, mostly through researching user content, searching forums and online video tutorials. By using initiative and knowledge learnt through coding in C++ it has been possible to transfer skills to the visual scripting system. Blueprint is extremely useful for creating quick prototypes however it lacks the satisfaction received from writing working code and solving problems. I would like to continue working with UE4 and start developing in Unreal Engine 5, however I would like to consider also using C++.

The final production of the project shows a higher quality than I expected using UE4 and the development of prototypes in user testing show a clear improvement in understanding of the system and development of problem-solving skills especially using the visual scripting system.

As UE4 already contains many built-in functions and modules such as basic physics engines and particle systems allowing the user to create interesting effects it is necessary to display full understanding of its functionality. This is most prominent in NI particle system which provides the user with instant access to high quality effects, learning how to add modules through NI allows for more creative control and I have displayed some control over this but will need to continue my practice to create more original works.

It took effort to begin to feel confident in using UE4 and there were moments were I lost focus on how to accomplish the project in relation to the artistic direction and techniques to be used. One area which I need to build on is my creative process and understanding the overall expectation on what must be delivered for an artistic project in media, this may include more consideration in creating a concept early on and improving my writing skills to convey my overall idea. The sense of control over creative process must be refined in future projects and planning must be made more thoroughly at the beginning. Technical areas which need improving in UE4 will relate to creating classes and functions however, now I feel more confident in continuing to develop these skills.

One part of the project which I would also need to implement for future development is a GUI, this was initially stated in my proposal, and would allow control once the project once it had been launched creating a more diverse product which could easily allow tuning of audio or selection of visuals depending on the performance and music.

The testing of the final performance in a live setting was not completed as there was no available location at the time the project was completed. This would have helped to see how the emotional response would have been received by a test group in the specific stated setting. One other issue in performance is that my current computer setup is a desktop making it more complicated to move however, as seen in a similar project [20] this can be performed using a sufficient laptop.

The tools experience gained from this project have allowed me to learn a new platform and develop my generative skills, which have diverse application from media, television, film, and games. These fields continue to flourish, and I hope to build the experience to apply my skills at a professional level.

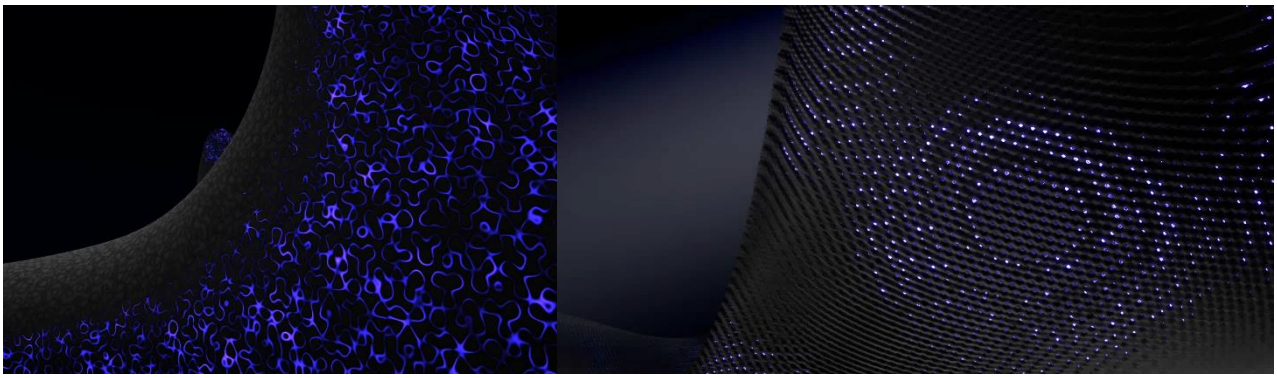
Section 10: Conclusion

As a result of this thesis UE4 has displayed results beyond my expectation and since early development in openframeworks I have seen a major improvement in the 3D graphics, this can be seen in the Niagara particle system and lighting and the fog attenuation used to create atmosphere and depth to the project. Stability has not been an issue which will be essential in a live performance setting however a high-performance computer with graphics card would be needed.

The audio spectrum tools have offered the ability to create detailed spectrum analysis applying calculations to smooth the values whilst providing the ability to render high quality 3D graphics, in comparison with openframeworks which struggled to retain frame rate during more complex 3D scenes whilst also processing audio and OSC data.

One area that comes to personal preference will be the choice of using Blueprint or C++ and this option will depend on the end user's preference, under further inspection there is also a slight performance drop whilst using Blueprint and this may become more noticeable on larger projects, and it should be considered which is more suited to the task as Blueprint allows for quicker prototyping.

The depth and color are also of a much higher quality when considering UE4 for generative work. When comparing work created GLSL and HLSL there is a clear difference in the amount of detail when ray marching, this can be seen in the images below comparing the same code in openframeworks and UE4.



Openframeworks Gyroid GLSL *Unreal 4 Gyroid HLSL*

The openframeworks version was unable to process detail far away and became unclear when too much detail was added. The HLSL gyroid was able to handle increased distance and the detail becomes more textured without losing framerate when increasing the number of steps in the ray march. It is also noticeable the level of background color in the HLSL version which creates much more depth and ambience, this may be due to the lighting system as UE4 easily allows emissive light to be passed as an output color. Whilst there is clearly more detail in the HLSL version it can be difficult to eliminate noise and this technique will need more time to master. UE4 also managed a much faster compile time with the whole project loading instantaneously in certain view modes

making it much easier to test work. On returning to openframeworks this did become more of an issue and slowed down production.

With UE4 being open source and free to use I am excited to continue to develop audio visuals using this tool, with the addition of Unreal Engine 5 soon to be released I can see this as an essential tool for any designer or creative innovator wanting to make high quality graphics with the addition of including interactivity or even developing VR experiences. I believe this tool to excel in developing a generative audio-visual experience.

Section 11: Bibliography

Bibliography

[1] J. a. M. H. MacDonald, “Visual influences on speech perception processes,” 1978. [Online]. Available: https://link.springer.com/content/pdf/10.3758/BF03206096.pdf.

[2] S. P.-A. a. L. Pan, “Skilled musicians are indeed subject to the McGurk effect,” 2019. [Online]. Available: https://doi.org/10.1098/rsos.181868.

[3] Z. L. O. Y. Z. X. Pan F, “The audio-visual integration effect on music emotion:,” 2019. [Online]. Available: https://doi.org/10.1371/journal.pone.0217040.

[4] S. Ellision, “Flying Lotus, Live at Form, Arconsanti,” Youtube, 2018. [Online]. Available: https://www.youtube.com/watch?v=ibFIChvLnUI.

[5] F. Carod-Artal, “Hallucinogenic drugs in pre-Columbian Mesoamerican cultures,” 2011. [Online]. Available: URL: http://whereareyouquetzalcoatl.com/Xochipilli_Carod_Artal_2011.pdf.

[6] 8. State, “Pacific 98,” zttrecords, Youtube, [Online]. Available: https://www.youtube.com/watch?v=0euY8QU1cP4.

[7] H. Kluver, “The Divine Plant and its Psycological Effects,” 1928. [Online]. Available: https://www.samorini.it/doc1/alt_aut/ek/kluver-mescal-the-divine-plant-and-its-psychological-effects.pdf.

[8] C. Webb, “Alien Structures,” 2018. [Online]. Available: https://www.shadertoy.com/view/MtdBD8.

[9] “Synaesthesia,” Gravity Current, 2022. [Online]. Available: https://synesthesia.live/.

[10] I. Q. a. P. Jeremias, “Shadertoy,” 2022. [Online]. Available: https://www.shadertoy.com.

[11] Wierdcore, “Aphex Twin, T69-Collapse,” 2019. [Online]. Available: https://www.youtube.com/watch?v=SqayDnQ2wmw&t=182s.

[12] R. D. James, “Aphex-Twin,” 2022. [Online]. Available: https://aphextwin.warp.net/.

[13] Wierdcore, “ ‘Simeon Mobile Disco (DJ visuals) – Tokyo’,” Vimeo, 2010. [Online]. Available: https://player.vimeo.com/video/14089551?h=0f82f604c3&app_id=122963.

[14] C. Reas, “Micro Image (Software 1),” Youtube, 2011. [Online]. Available: https://player.vimeo.com/video/81527119?h=1aee43fc01&app_id=122963.

[15] B. Laposky, “Oscillons,” 1952. [Online]. Available: https://collections.vam.ac.uk/item/O187634/oscillon-40-photograph-laposky-ben/.

[16] S. N. Safran AB, “Color synesthesia. Insight into perception, emotion, and consciousness.,” Curr Opin Neurol, 2015. [Online].

[17] “www.usertesting.com,” Uesr Testing, 2022. [Online]. Available: https://www.usertesting.com/blog/color-ux-conversion-rates.

[18] Rockstar, “GTA 5, night club lighting,” Youtube, 2013. [Online]. Available: https://www.youtube.com/embed/pnFp6WlLB8k?feature=oembed.

[19] 1. Architecture, “Core Reboot,” Youtube, [Online]. Available: https://player.vimeo.com/video/341863415?h=6bf59eae90&app_id=122963.

[20] T. Gieler, “Ableton & Unreal Engine AV set - first look,” 2020. [Online]. Available: https://www.youtube.com/watch?v=0QdOqG0BBAI.

[21] “Ableton Live,” Ableton AG, 2022. [Online]. Available: ableton.com.

[22] A. Wilson, “Controlling Unreal with OSC,” Youtube, 2020. [Online]. Available: https://www.youtube.com/watch?v=9CkKPCBys44&t=1494s.

[23] ArthurBarthur, “Arthurs Audio BP's,” GIT, 2021. [Online]. Available: https://github.com/ArthurBarthur/ArthursAudioBPs/tree/main/ArthursAudioBPs.

[24] M. Magi, “‘Niagara Audio Visual Particle Project’,” GIT, 2021. [Online]. Available: https://github.com/Lahe/AudioVisual.

[25] yeczrtu, Youtube, 2021. [Online]. Available: https://www.youtube.com/watch?v=el_c8WftaAE.

[26] “Art Hiteca,” Youtube, [Online]. Available: https://www.youtube.com/watch?v=DmgMDtC8wGk.

Section 12: Appendix

Appendix:

Git: https://gitlab.doc.gold.ac.uk/pkins001/audio_visual/-/tree/Unreal_Dev

Includes:

- Full Unreal Project Files, note project uploaded on Unreal_Dev branch.
- HLSL Shaders folder

Supporting Documents Folder:

- Ableton Live Project file “Techno_Dig”.
- MP4 Demo Video

Final video demo youtube: https://www.youtube.com/watch?v=2k6hsvVGbRs