SPATIAL AND KINEMATIC MODELS FOR PROCEDURAL AUDIO IN 3D VIRTUAL ENVIRONMENTS

Ignacio Pecino

NOVARS Research Centre,

University of Manchester, UK

joseignacio.pecinorodriguez@postgrad.manchester.ac.uk

ABSTRACT

In the last years an increasing interest can be observed for developments in game engine technologies as a versatile creative tool. In particular, the possibility to visualize and simulate real-time complex physical behaviors facilitates the design and implementation of 3D virtual music instruments and the exploration of sound gesture as a result of their kinematic and spatial properties. This paper describes two case examples in the form of linear compositions based on non-conventional instrumental designs where audio is procedurally generated using custom-built APIs in the game engine scripting language (Unity3D-Javascript/C#). Sound events are also organized as a sequence of flexible code instructions, resulting in a quasi-fixed piece duration with subtle timbral variations over multiple playbacks. In both cases, the model presented shows inherit spatial characteristics, which are useful in order to build spatialization patterns in a multichannel loudspeakers configuration and emphasize the strong causal connection between the visual and sonic aspects of these works.

1. INTRODUCTION

1.1 Game Engines as Multidisciplinary Creative Environments.

The need for rich development environments, especially in the field of sound art and interactive media, has led to the incorporation of new tools [1] [2] which, although originally devised for the video game industry, have proven to be an extremely convenient and affordable solution for many sound artists.

This is easily understandable as video games development often requires similar software and hardware capabilities as many of the interactive works proposed by sound artists, including graphics rendering, time management methods, an audio playback engine, input interfaces, etc.

In some occasions, this type of software can also be used for prototyping complex art installations where an impact assessment is required [3], yet the usage of game engine technologies goes far beyond real-world simulation models, allowing for innovative and original approaches to instrumental design and composition where virtual tridimensional space can be considered as the main structural variable that shapes the musical output.

Notable examples of compositional applications of a game engine [4] can be found in the works of Robert Hamilton [5], Ricardo Climent [6] or Andrew Dolphin [7] [8], whose "sound toys" are a very interesting demonstration of interactive tools with a focus on space and real time synthesis (in MAX/MSP) using control data from a 3D engine (Unity). However, it is possible to appreciate how most of these and other works using game engine technologies are essentially interactive in nature, as they need some sort of live user input in order to produce sound, whether it is coming from a member of the audience (e.g. a sound installation) or the artist himself (during a performance or at the studio).

In contrast, the two cases described later in this paper are an attempt to reconcile the more traditional approach of fixed-media and acousmatic composition with the emerging trends and procedural techniques that these new development environments have to offer. In particular, the potential for 3D objects manipulation, integrated physics (collisions detection, gravity, etc.), along with the built-in scripting languages, are exceptionally helpful when mapping dynamically generated spatial and kinematic data into the audio engine of our choice (e.g. SuperCollider [9]), while sound events can be programmatically sequenced over time from a custom script, analogously to how we organize sound materials or MIDI events in a DAW (e.g. Protools).

1.2 Procedural Audio

Given the real-time nature of most processes running in a game engine, a procedural approach to sound generation is possible and desirable, even in a non-interactive compositional context. An audio sample playback engine is generally provided in every game engine, but these are usually very limited in their functionality, permitting only basic volume and pitch controls. Consequently, it is often advisable to replace the default internal audio engine with a more specific and powerful software for real time audio

processing such as Supercollider or MAX/MSP. Both sides of the system (game engine and external audio software) are then interconnected using a standard communication protocol such as OSC [10].

Procedural audio [11], as opposed to recorded sound, is created in real time according to a set of rules or algorithms (sound as a process), reducing the amount of raw audio data needed to play a music piece although considerably more demanding in terms of CPU usage.

1.3 Live Coding vs. Sequenced Code.

The use of live coding (on-the-fly programming) as a form of musical expression is an increasingly popular practice within the music community. Supercollider and Chunk [12] are well-known examples of object-oriented programming languages that allow live interaction with code and improvisation.

On the other hand, the scripting languages that most game engines include (JavaScript and C# in Unity, Python in Blender, etc.) do not permit code modifications at runtime but, in most situations, they are still capable of managing time efficiently, as long as we do not require sample accurate audio processing.

In the two case examples proposed here, the scripting programming language is utilized in three different ways, corresponding to the three stages of the proposed compositional method.

First, it is required to implement the basic structural design of our virtual instrument, including the characteristic variables and components of the system.

Secondly, it allows us to define a set of functions or subroutines describing what our virtual instrument is able to perform (instrumental techniques) as result of the manipulation of the previous variables and objects.

Thirdly, as we mentioned earlier, all these functions (API) can be called as an organized sequence of events to create the temporal structure of the final musical piece (specific examples will be presented later in this paper).

Across all these three stages, minor random elements are introduced into the instrumental design and related algorithms, so the final output will be a piece with an approximate total duration that exhibits a macroscopic deterministic behavior while presenting interesting small variations at a microscopic level which are not directly determined by the composer (in contrast with conventional fixed-media pieces).

2. VIRTUAL 3D INSTRUMENT IMPLEMENTATION

Efforts to build virtual 3d instruments for musical interaction can be found prior to the emergence of game engines technologies [13], but the workflow involved in this process and the quality of the final output has benefited from the general features present in this accessible and "ready-to-use" development framework.

2.1 Space as the starting point

The two pieces we are about to discuss next, make extensive use of these virtual 3D environments, proposing non-conventional instrumental models with strong spatial attributes. The first piece ("Singularity") uses a sonification of cellular automata to explore the growth and dissemination patterns of its cells as a motif generator for spatial concatenative synthesis. On the other hand, the second piece studies the kinematic behavior and sonic potential of a set of virtual rings, organized as a fractal geometric construct known as Apollonian Gasket [1], which is recursively generated from triples of circles where each circle is tangent to the other two. Given the elements involved—materials/physical objects, sound and movement—this second piece displays manifest similarities with generic sound sculpture works [14].

Nevertheless, none of the instruments employed in these pieces imitate any real-world conventional instrumental design, but suggest more abstract prototypes based on mathematical models. Thus, the focus is not on simulation but on the use of space and the possibility of composing original pieces using bespoke experimental instruments that might otherwise be difficult and expensive to implement in a real-world situation.

2.2 The Audiovisual Dilemma

It seems to be clear that the possibility of real-time visualization and manipulation of 3D objects within a game engine is certainly helpful when designing and implementing 3D virtual instruments, as we are establishing direct causal relationships between the operation/behavior of the simulated physical system and the following sonic output. However, since the main goal of the proposed method is to deliver a non-interactive music composition, it is not so clear what the nature of the final piece should be: acousmatic or audiovisual, especially when the music language used resembles that of the electroacoustic acousmatic tradition. In other words, should the audience be presented with a visualization of the virtual instrument performing the piece?

In principle, both points of views are equally valid (with their pros and cons) but, given the spatial attributes of the pieces being discussed, I tend to believe that the sonic gesture is better explained when we are able to see the object and movement involved, in the same way as a performer and his instrument can be observed in a traditional concert situation. Yet categorizing the piece as "audiovisual" might not always be correct, as the visual side of it does not necessarily have an aesthetic intention, in the way the musical output does.

Furthermore, the visuals of the mentioned pieces have deliberately been kept simple and minimalistic, avoiding

[1] http://en.wikipedia.org/wiki/Apollonian_gasket

unnecessary elements not related with the sonic mechanisms and reinforcing the original musical concept.

2.3 Audio Sources and Spatialization

One of the advantages of using game engines to play sound materials is that the number of final audio channels is not fixed, not even needs to be considered during the compositional process. Consequently, spatialization is mostly based on the movement and positions of virtual objects (audio-sources) in the 3D virtual environment. Therefore, sounds are automatically panned by the built-in spatial audio engine according to the current loudspeaker setup and the audio-listener object position (also included in the scene). Once the piece is completed, we are able to switch the desired multichannel output configuration from a list of available options, usually including stereo, quadraphonic, 5.1 and 7.1 surround.

Alternatively, if we decide to integrate an external audio engine, such as one of the previously discussed, this spatial functionality is lost and needs to be re-implemented. The solution considered in one the mentioned examples (Apollonian Gasket) is to translate Unity's Vector3 positions (x,y,z) into angular positions (r, θ) and send this data to Supercollider where sounds can be panned into any speakers array using VBAP [15]. The downside of this approach is that our piece will no longer work as a single standalone application but it will depend on external software to play our sounds. This is, however, a distribution issue and not a compositional one.

3. SINGULARITY: SONIC AUTOMATA

A sonic implementation of a cellular automaton is presented along this piece[2], consisting in an orthogonal grid of square cells (represented as cubes) that "live or die" according to a given set of rules known as Game of Life [16].

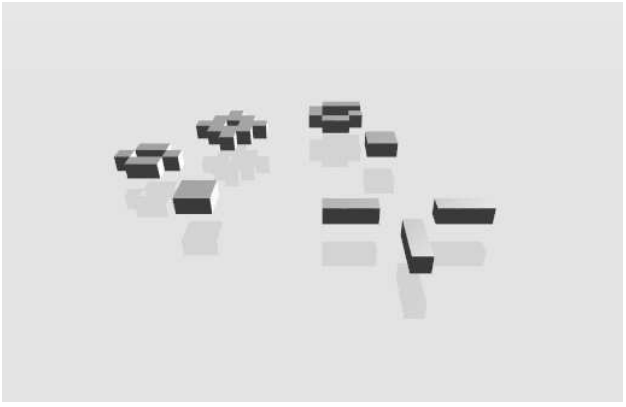

Figure 1. Screenshot of cellular automata in Singularity

The rules are quite simple but powerful enough to originate complex emergent phenomena. The originated shapes and patterns depend on the initial configuration of the automaton and evolve over time (using discrete time intervals/ steps) spreading across the different areas of the grid (see Figure 1). This behavior is particularly interesting from a compositional point of view, since it exhibits "life form" characteristics: growth, decline, chaos-order alternation, etc., which are structurally suitable for a musical discourse.

Also, the dynamic spatial dissemination of the living cells can be used to generate spatialization motifs within a surround loudspeakers configuration. In this case all sounds are played within Unity using the default audio engine.

3.1 Sonification of the Cellular Automaton

In order to translate the successive cells states into sonic gestures, an array of very short audio samples (<100ms) dissected from original recordings at the Museum of Science and Industry of Manchester, was used. Then, a piece of JavaScript code selects a random sample and plays it whenever a new cell comes to live. This sound is played at the cell's position, contributing to the surround sonic image perceived from the point of view of the audio-listener object at the center of the scene.

The rapid concatenation of micro-samples across a bi-dimensional space produces a macroscopic textural effect that can be categorized as a form of spatial concatenative synthesis, as long as the evolution rate is fast enough; although sometimes this rate is also kept relatively low and variable, favoring the apparition of distinct rhythmic patterns.

3.2 Software Control Interface

A number of variables are defined in the automaton's code, so we can modify its properties and state; for example: size (number of cells in the grid), initial density (how many cells are likely to be alive on start), initial area (size of the sub-grid where cells can be initially instantiated), pitch-shift (option to slightly detune audio samples), random-volume, random-rate, etc.
All the cells are initially disabled (dead state) but we can activate them from the main sequence script using the cell "state" property as follows:

```
var automaton_A: GameObject;
function Awake(){
        //Reference to the control script in the A automaton
        a = automaton_A.GetComponent(Automata);
}
function Start(){
        a.rate=0.1;
        a.state[10,12]=true;
        a.state[11,13]=true;
        a.state[9,13]=true;
}
```

The previous example would initiate a particular automaton configuration, which would evolve under the given rules (on a separate script) until a stationary state is achieved or all the cells are dead. As we can observe, the reference to the corresponding cell is provided by its (x,y)

[2] Recording of the piece available at http://vimeo.com/79344883

position in the grid (bi-dimensional array) while other properties can also be modified here.

In addition, specific functions are also created as methods to control the automaton's behavior, particularly, tempo related functions such as Accelerando, Ritardando, etc. The evolution rate (time between consecutive steps) can be fixed or randomly selected between a minimum and maximum rate. The next example shows a routine that progressively changes these values over a given period of time:

```
function ChangeRate(targetMin:float,targetMax:float,seconds:float){
   var t = 0.0;
   var currMin= minRate;
   var currMax= maxRate;
   while (t <= 1.0) {
      t += Time.deltaTime/seconds;
      minRate = Mathf.Lerp(currMin,targetMin, Mathf.Lerp(0.0, 1.0, t));
      maxRate = Mathf.Lerp(currMax,targetMax, Mathf.Lerp(0.0, 1.0, t));
      yield;
   }
}
```

3.3 Creating the final sequence

Once the instrument with its custom control interface is ready, we need to select a number of automata configurations as possible candidates for the final sequence, taking in consideration their musical potential and functionality within our piece. This process is analogous to the act of selecting sound materials for an acousmatic piece, although we are now working with shorts space-timbre motifs generated by virtue of the fast concatenative action of our cellular automata.

The sequence is possible thanks to several time related instructions such as the yield instruction of the Unity scripting language. For instance, if we need to wait 1.5s before a new motif is generated, it is just a matter of typing the following line:

```
yield WaitForSeconds(1.5);
```

Using a while statement, it is also possible to link two events so the second one will only take place once a certain condition involving the first event is met. For example:

```
//automaton A is ritardando
//Wait until its minimum rate is above 0.85
while (a.minRate<0.85){
   yield; //Wait 1 frame
}
//Play motif in automaton B
automata_B.SetActive(true);
b.state[1,1]=true;
b.state[1,2]=true
b.state[3,2]=true;
//etc.
```

Other subroutines are also created when multiple code threads need to run in parallel, facilitating the operation of simultaneous automata within our scene from a single master script (sequencer).

4. APOLLONIAN GASKET: A VIRTUAL SOUND SCULPTURE

As we mentioned earlier, this second case example explores the kinematics of a set of rings geometrically arranged as an Apollonian Gasket. This mathematical object was chosen not just because of its spatial characteristics, but also because of its simplicity and recursive elegance. Moreover, the kinematic model of the rings movement has multiple similarities with a popular sound toy known as Euler's Disk[3] and also resembles the natural behavior of a spinning coin.

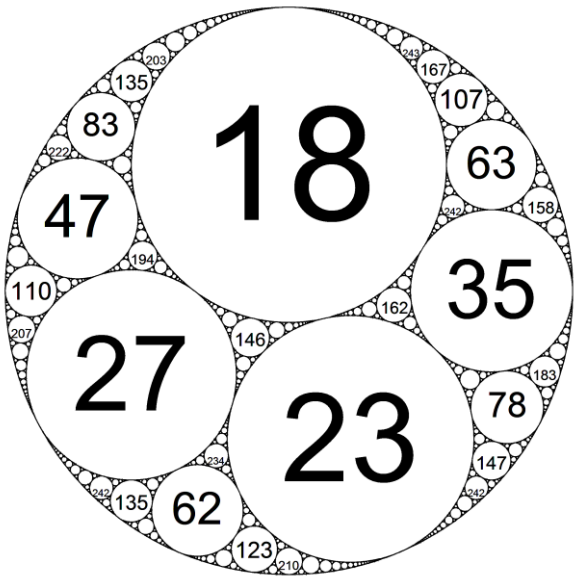

Figure 2. Spatial distribution of the Apollonian Gasket

The original Gasket is a bi-dimensional object as we can see in Figure 2, but it has been extended for this piece[4] into a third vertical dimension, in order to allow a richer kinematic experience (see Figure 3).

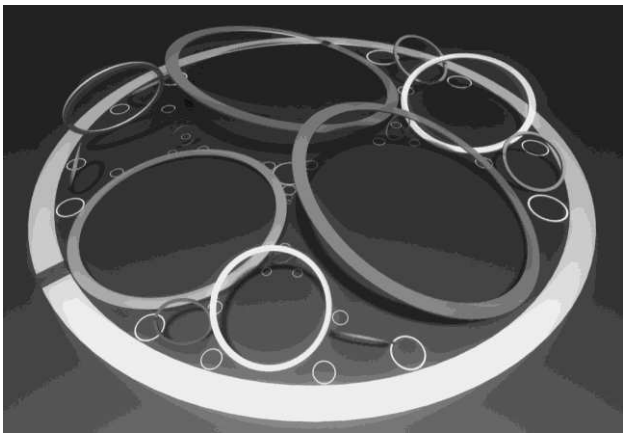

Figure 3. Screenshot of the 3D Apollonian Gasket Virtual Instrument.

[3] http://www.youtube.com/watch?v=ug2bKCG4gZY

[4] Recording of the piece available at http://vimeo.com/83995079

Given that the curvatures of the component rings are being used as a variable to feed sound parameters, I opted for an asymmetric shape given by the initial values 18, 23 and 27, thus avoiding excessive regularity and repetition.

4.1 Sonification of Kinematic Data

Audio is generated procedurally, in real time, using synthesis techniques that respond to specific changes in physical variables in the virtual objects (rings).

These objects are animated using a custom Unity script and the consequent kinematic data is sent to an external audio engine (Supercollider) via OSC[5] messages. Again, this approach permits a tight interconnection between the visual and sonic elements of the piece, since audio-video synchrony is nearly perfect and the movement/gesture of the virtual object can be effectively translated into sound.

The algorithms used in Supercollider include different types of synthesis such as Physical Modelling, FM/AM or Granular Synthesis and the input variables used to feed these generators include: ring curvature, tilt, rotation speed, precesion speed, collision strength, horizontal angular position (azimuth), etc.

One of the advantages of this kind of procedural sound generation is, for instance, the high number of sound variations that can be achieved with very light algorithms. For example, the .scd file used by SuperCollider to generate all the sounds in this piece, despite it contains more than 1500 lines of code, it only weights 30Kb (plus another 30Kb on the Unity scripting side). Also, these sonic variations are responsive and necessarily linked to the virtual object's kinematic variables, while a similar complex response using raw recorded audio would take a considerably larger amount of data.

4.2 Control Interface

Similarly to how we did in the first case example (Singularity), we need some kind of software interface to regulate the mechanics of the proposed instrument.

The system suggested is now based on an "energy" variable that determines the kinematic state of the rings. Then, we can introduce energy variations by calling the following function (C#):

```
IEnumerator ChangeEnergy (GameObject ring, float targetEnergy,
float seconds){
        float t = 0.0F;
        var precesion = ring.GetComponent<Precesion>();
        float currentEnergy = precesion.energy;
        while (t <= 1.0F) {
                t += Time.deltaTime/seconds;
                precesion.energy=
                Mathf.Lerp(currentEnergy,targetEnergy,
                Mathf.Lerp(0.0F, 1.0F, t));
                yield return 0;
        }
}
```

However, before any energy is introduced into the system, the multiple rings need to be initialized, both in Unity and SuperCollider. For example (Unity code):

```
public void StartRotor(){
        arcTan = Mathf.Atan(transform.position.x/transform.position.z);

        //Surround Panning
        if(transform.position.z>=0){
                azimut = (arcTan*180/Mathf.PI);
        }else{
                if(transform.position.x>=0){
                        azimut = 180 + (arcTan*180/Mathf.PI);
                }else{
                        azimut = (arcTan*180/Mathf.PI)-180;
                }
        }
        //Send OSC Messages
        List<object> values = new List<object>();
        values.AddRange(new object[]{curvature, azimut});
        OSCHandler.Instance.SendMessageToClient("SuperCollider",
        "start"+curvature.ToString(),values);
        rotorSynth=true;
}
```

The previous functions can obviously be reused as many times as needed from the master sequence script:

```
aro18.GetComponent<Precesion>().StartRotor();
StartCoroutine(ChangeEnergy(aro18,10.0F,5.0F));
```

These are just two example of the Apollonian Gasket API functionality, but a number of other methods can also be called to easily modify any required property.

4.3 Instrumental Techniques

The Apollonian Gasket instrument presents at least three distinctive timbres, depending on the characteristic movements/behaviour of the rings in 3D space, namely:

a) Rotation. Circular movement around the ring's main symmetry axis (Figure 3).

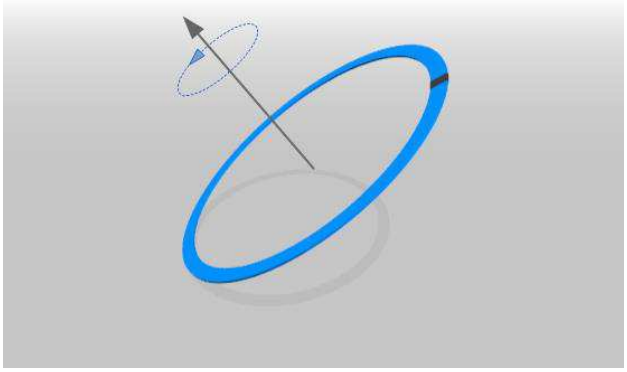

Figure 3. Rotation movement

[5] Using UnityOSC (https://github.com/jorgegarcia/UnityOSC)

b) Precesion. Changes in the orientation of the rotational axis around a vertical axis (secondary rotation). Rotation and precesion speeds are often related (Figure 4) and the consequent movement is translated into sound using real-time noise filtering and recursive modulation techniques in SuperCollider.

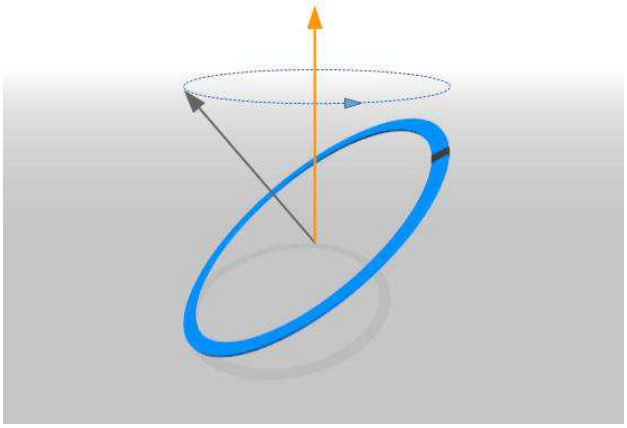

Figure 4. Precesion movement

c) Collision. A small percussive element is sometimes attached to the ring in a way that it will hit the floor at intervals, producing a collision sound as a consequence of the rotation/precesion movement (see Figure 5). In that case, the ring's tilt angle and the relative velocity of the collision can be used to feed several input parameters of an associated physical modelling synthesizer (Karplus-Strong UGen). Furthermore, the contact points generated by the physics engine provide the necessary vector data to position our collision sound using VBAP (see Section 2.3).

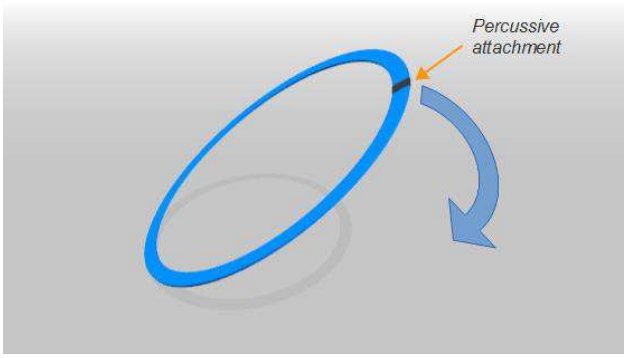

Figure 5. Collision gesture

4.4 Spatialization and Sound Diffusion

The geometrical distribution of the rings, according to the mathematical fractal structure, provides a quasi-symmetric layout with three main groups of rings placed on the Front-Left, Front-Right and Rear-Center (see Figure 6). The audio listener is then placed at the approximate centre of the set (ring146), providing an optimal listening position that can later be reproduced in the concert hall when playing the piece in a multichannel speakers setup (e.g. MANTIS[6] system). Every sound event is panned in Supercollider using the VBAP object, allowing for azimuth and elevation data on any array of speakers. This approach reinforces the original idea of channel independent composition where sound objects are spatialized in virtual space, regardless of the final number of output channels.

Furthermore, the Apollonian Gasket presents a sort of inside-outside separation given by the three original tangent circles, as all remaining circles will either be inside or outside of these three circles. The inner set is considerably smaller in size but has the same number of rings and similar spatial distribution as the outer set. This in/out characteristic can be easily used to create two separate multichannel buffers or stems (e.g. 4+4) and send them to different groups of speakers in the sound diffusion system (close vs. distant speakers) allowing for further exploration and manipulation of the piece during the live concert performance [17].

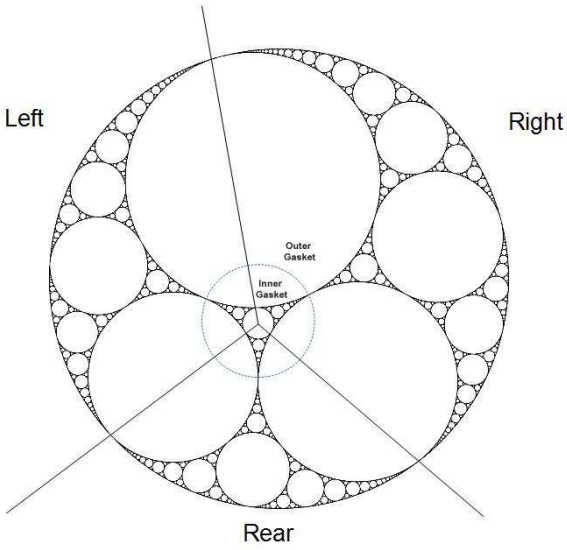

Figure 6. Spatial distribution of the Apollonian Gasket

5. CONCLUSIONS

Subsequent analysis of the ideas and techniques proposed in these two works seem to suggest that all the usual phases of music creation (e.g. instrument design/implementation, exploration of instrumental techniques, performance and composition) are also present within a game-engine environment and can be formalized using a programming language. This is indeed one more example of the existing virtualization trend that can be observed in many other disciplines; usually driven

[6] www.novars.manchester.ac.uk/mantis/

by economic factors, but focused here on extending the limits of musical expression.

We can also appreciate how real-time procedural sound generation is not necessarily linked to interactive media but may also be adopted for compositional purposes, for example, when a fixed macroscopic output is pursued while maintaining a certain degree of microscopic randomness or organic behaviour. Using code to organize our sound-generating processes (in contrast with traditional DAW timelines) has proven to be an effective way to achieve this goal, as it permits a flexible implementation and reinterpretation (playback) of these functions.

On the other hand, by extending our notion of space to the virtual domain, we have been able to explore sonic gesture as a result of simulated physical actions and spatial distribution/dissemination models that can be consolidated as viable 3D virtual musical instruments.

The complexity and level of detail of the processes involved are only limited by the actual processing speed of our CPU/GPU, but it is also possible to use separate networked machines for specific tasks (e.g. physics engine vs. audio engine) when the proposed model requires more processing power.

6. FUTURE WORK

By the time of writing this paper, all live performances of these pieces have been carried out in a concert hall using multiple loudspeakers and a conventional projection system (large screen at the stage). The most important limitation of this approach is that it requires some effort from the audience in order to apprehend the proposed correspondence between the visual on-screen space and the aural concert hall space. For example, for the perspective view displayed in Figure 1 and 3, objects rendered at the top of the screen would normally be played from the front loudspeakers, while objects at the bottom of the screen would be played from the rear ones (anything in-between is panned accordingly). Therefore, we find a physical separation between both spaces which is only overcome thanks to the listener's intuition and imagination.

As an optimal future solution, a fully immersive visual system such as Oculus Rift[7] could be adopted. The transition to this new technology should be quite straight forward given that the game engine used for these pieces (Unity) has already been tested with this kind of hardware. Such an approach would be compatible with the current system using loudspeakers and VBAP panning, but alternative audio rendering methods based on binaural or Ambisonics techniques might also be considered.

[7] http://www.oculusvr.com/

7. REFERENCES

[1] "Unity3D", http://unity3d.com/. Last accessed March 2014.

[2] "Blender.org". http://www.blender.org. Last accessed March 2014.

[3] I.Bukvic, "Using gaming engine for virtual prototyping and impact assessment of complex interactive art installations", in Proceedings of the International Computer Music Conference, Huddersfield, UK, 2011.

[4] A.Dolphin, A. "Compositional Applications of a Game Engine", in Proceedings of the Games Innovations Conference (GIC), London, UK, 2009, pp. 213-222.

[5] R.Hamilton, "UDKOSC: An immersive Musical Environment", in Proceedings of the International Computer Music Conference, Huddersfield, UK, 2011.

[6] R.Climent, http://acousmatic.org. Last accessed March 2014.

[7] A.Dolphin, "Cyclical Flow: Spatial Synthesis Sound Toy as Multichannel Composition Tool", in Proceedings of the International Computer Music Conference, Ljubljana, Slovenia, 2012.

[8] A.Dolphin, "SpiralSet: A Sound Toy Utilising Game Engine Technologies", in Proceedings of the International Conference on New Interfaces for Musical Expression (NIME), Pittsburgh, USA, 2009, pp. 56-57.

[9] "SuperCollider", http://www.audiosynth.com/. Last accessed March 2014.

[10] M.Wright, A.Freed, "Open Sound Control: A New Protocol for Communicating with Sound Synthesizers", in Proceedings of the International Computer Music Conference, Thessaloniki, Greece, 1997.

[11] A.Farnell, "An introduction to procedural audio and its application in computer games", available online at http://obiwannabe.co.uk/html/papers/proc-audio/proc-audio.html, 2007. Last accessed March 2014.

[12] G.Wang, P.R.Cook, "On-the-fly Programming: Using Code as an Expressive Musical Instrument" in Proceedings of the International Conference on New Interfaces for Musical Expression (NIME), Hamamatsu, Japan, 2004.

[13] A.Mulder, S. Fels, K. Mase, "Design of Virtual 3D Instruments for Musical Interaction", in Proceedings *of Graphics Interface '99*, Kingston, Canada, 1999, pp.76-83.

[14] J.Grayson. Sound sculpture: a collection of essays by artists surveying the techniques, applications,

and future directions of sound sculpture. Vancouver: A.R.C. Publications, 1975. ISBN 0-88985-000-3

[15] V. Pulkki, “Virtual Source Positioning Using Vector Base Amplitude Panning.” J. Audio Eng. Soc., 45(6): 456–466, Jun. 1997.

[16] M.Gardner, “The fantastic combinations of John Conway's new solitaire game "life", Scientific American 223, Oct.1970: 120-123

[17] D.Berezan, “Flux: Live-Acousmatic Performance and Composition”, EMS: Electroacoustic Music Studies Network, De Montfort/Leicester, 2007.