QuBits, a System for Interactive Sonic Virtual Reality

Jonathan Kulpa
CNMAT - UC Berkeley
kulpajj@berkeley.edu

Edmund Campion
CNMAT - UC Berkeley
campion@berkeley.edu

Carmine Cella
CNMAT - UC Berkeley
carmine.cella@berkeley.edu

ABSTRACT

This article describes the QuBits system, a virtual reality environment offering an expanded medium for musical experience with space and visuals. The user and the computer jointly shape many of the events. Sound engines were designed to explore an aesthetic of algorithmically generated sonic structures, sound mass, interactive evolution, and spatial sound. Real-time challenges are discussed. A network of software is diagramed, solving initial issues with latency. Finally, the principles and methods utilized in the current project are evaluated with implications for future iterations.

We present *QuBits*, a virtual reality (VR) system surrounding a user in a virtual sound space. The user participates with headphones and the Vive VR System [1]. They encounter many kinds of *VR characters* (see Figure 1), beings with a presence in virtual space. These characters are detailed in Section 2.3. When each VR character makes a sound, it also has a visual behavior, unifying audio and image into a single event. Using hand controller inputs and by movement through the space, the user influences VR character audiovisual behavior and evolution. The user discovers what behaviors they can interact with through explorative input. They can then shape these behaviors further. Two vantages of the virtual space are possible, the *rightside-up* (see Figure 1) and the *upside-down* (see Figure 2). In the upside-down, visuals are distorted and the sounds of the right-side up are processed with a chain of effects (see Section 2.5).

The system is released under the GPLv3 license and can be downloaded at:
`https://github.com/kulpajj/QuBits_Max`
`https://github.com/kulpajj/QuBits_Unity`

A demonstration of the system can be found at:
`https://www.youtube.com/watch?v=2iZRtcW0X9k`

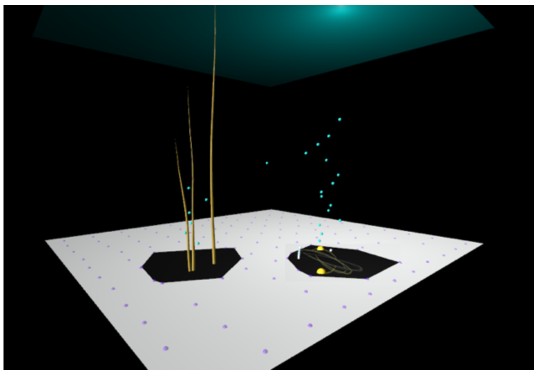
Figure 1. Various VR characters in the *rightside-up*.

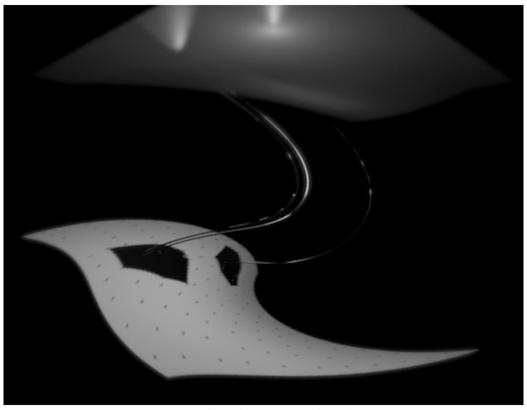
Figure 2. A vantage in the *upside-down*.

1. STATE OF THE ART

Before discussing our system, we review works from other sound artists and composers exploring generative sound. To discuss this type of music, we find it useful to have a conceptual model, consisting of three main elements. These elements are 1) rules, lying in wait to be activated or not, 2) input values (numbers) that seed the rules, and 3) timings of the input values, causing output and generating a sonic event. An example will clarify this idea:

> If an input force > 10, then make the timbre brighter, else make the timbre darker.

The statement is the rule. Sending an input value, a strength of force, at a particular moment in time produces an output value, making the sound brighter or darker. Successive input values shape sound over time.

Additionally, we propose the notion of the *creator*. Any creator (human, computer, or nature) can be responsible for any of the three elements of the conceptual model (rules, values, or timing).

In the following review, we examine works according to who or what creates the timing of events.

Nature Creates Timing. Åsa Stjerna's *Sky Brought Down* [2] uses weather stations to measure data about atmospheric pressure, precipitation, wind speed, and light intensity. These values represent the rhythms and conditions of nature, which Stjerna then maps to synthetic sounds that can be short or sustained.

Computer Algorithms Create Timing. Robert Henke's laser and sound installations, such as *Fragile Territories* [3], map a set of statistical algorithms simultaneously to audio and visual; thus, both media evolve in tandem.

These works can be infinite in duration due to constant algorithmic variation.

Human User and Computer Jointly Create Timing. Michael Musick's *Sonic Spaces Project* [4] features sculptures that each contain a microphone, microcontroller, and speaker. Users inject the system with energy by making sound in any way they wish. Thereafter, algorithmic analysis of the sound determines which processing engines activate; algorithms also govern the engines themselves. In Natasha Barrett's *OSSTS* [5], a user sits in a chair and uses a joystick controller to traverse a virtual space projected on the floor (visuals move around the stationary chair). Visuals correspond to one of 30 composed sound spaces, with many embedded subspaces. Users navigate these sounds by their own accord, able to leave a sonic scene and move to another. Paul Weir produces interactive sounds for video games where randomly selected phrases are layered together in unique recombination [6]. Also, a user's position affects an interpolation value, morphing sound between a start and end state.

2. OUR MODEL

2.1 Aesthetics

2.1.1 Generative Music

QuBits engages with generative music composition. In this project, we created the rules, and there are three modes of interaction between computer and user for the creation of values and timing. 1) The computer entirely determines the timing of an event and also the values shaping the event. 2) The user triggers the beginnings of an event, but thereafter values are selected by the computer. 3) The user triggers the beginnings of an event, and thereafter user and computer jointly shape values, *a duet of influence*. When the user inputs to the system, they influence the sound, and when they no longer input, the computer completes the sound.

2.1.2 User-Interactive Evolution

Embedded in the environment are rules that remain dormant until the user discovers how to activate them, using their hand controllers and by moving through virtual space. The new rules introduce new VR characters or change the behavior of those already present, providing a sense of evolution. Many of these discoverable rules are independent, i.e. not part of a longer-term plan.

We also made a trajectory of four rule sets, or states, that progress in a specific order, most likely over a longer duration. We refer to this as the *long-term evolution*. Along this trajectory, the user causes each change in state. The trajectory progresses from maximum noise/least pitch to maximum pitch/least noise. Also, VR characters become ever more interconnected. When a user encounters any change in the system, they do not need to perceive whether it is independent or part of this long-term plan. Both are strategies to create an environment that evolves over time.

2.1.3 Sound Masses and Particulate Sound

QuBits explores particulate sound in which many grains of activity make up a larger sound mass. In this project, masses of VR characters enable the exploration of masses of sound. There are 121 individual *qubit* characters, each contributing one particle of sampled real-world sound to a greater mass. There are also 36 *ceiling light* characters, each contributing energy to a synthesized sound mass.

2.1.4 Spatial Sound

In *QuBits*, there are two ways in which space, here virtual space, is important for the user's experience of sound. A user's hand controllers allow them to place input gestures in virtual space, then affecting the sound of VR characters near those locations. Also, audio and visual effect engines are activated and shaped by the user's movement through virtual space.

2.2 Sound Materials

The sounds in this project are a mixture of sampled real-world sounds, phrases of granular synthesis (based on the samples), and synthesis (generated without samples).

2.2.1 Sampling of Real-world Sounds

A large collection of objects was obtained to make an extensive sample library. The objects include an assortment of metal, laminated wood, bricks, and pins. The pins were used to excite the resonances of the other objects, but also resonate themselves. The sound sources were detailed with multiple microphones placed close in proximity in a room with a low noise floor. Very short samples were made as follows.

For each metal and wood object, each sample consists of striking the object once with a pin, at a unique node and with a variable amount of force, then letting the object resonate (we refer to these samples as *pin strikes*). A hyper-cardioid, omni, and large diaphragm microphone were used; 15 to 80 samples were made per object.

For the bricks, contact microphones were applied to capture vibrations traveling inside the material. For each sample, one of the pins was dropped from above, first bouncing onto the bricks followed by the sound of the pin rolling around (we refer to these samples as *pin ricochets*); 100 samples were made.

Longer samples were also made with a collection of baskets made of various dried grasses or metal wire. Each basket was squeezed by hand to produce longer samples of crackling sounds.

2.2.2 Phrases of Granular Synthesis

We wanted to explore the generation of longer musical phrases by chaining together the short samples described above. Granular synthesis driven by code is the technique employed. The granular synthesis engine used is o.granubuf~ [7], which runs in Max/MSP [8].

Granular synthesis is usually implemented by splitting a long sample into evenly-spaced windows of sound, or grains, resulting in a variable amount of attack and decay per grain. In our design, we wanted each grain to have a predictable attack and decay. The short samples described above were recorded with this goal in mind, each having the desired sonic morphology. Each short sample is a single grain.

We built granular engines driven by code to explore various rhythms and effects. One engine creates accelerating and decelerating rhythms. Another creates various densities of sound mass. For this engine, the singular grain is a pin ricochet sample, the mass then sounding like pins endlessly dropping on bricks. A code engine controls the duration between grains, producing a desired mass density. We refer to these sounds as *pin ricochet masses*. We will later describe how each of 121 qubits quietly plays this sound, constructing a larger mass from these smaller masses. Other engines were created for granular time-stretching and granular freeze (where a single temporal slice resonates for as long as desired).

2.2.3 Synthesis (Without Samples)

In this project, synthesis is used to recreate the sound of a suspended metallic washer that has been struck with a pin (causing it to spin and produce audible pulsations). We had the desire to control timbre and rate of pulsation, using a model. The synthesis engine used is resonators~ [9], running in Max/MSP. The washer resonance model and the spin pulsation are produced by separate algorithms.

In making each washer resonance model, we explore degrees of harmonicity/inharmonicity. In [10], the author proposes a generalized series to variate timbral quality. A collection of frequencies is generated from three parameters:

$$G = \sum_{n=1}^{\infty} f\left(n^{\alpha} \cdot \beta^{n} + \gamma\right) \tag{1}$$

f is the fundamental frequency and n is the partial number. The three parameters are harmonicity α, deformation β, and size γ. α is our primary focus, explored with values between .1 to 2. For variety, at run-time we generate 50 models at a time, within random constraints, many which could sound simultaneously. A score script defines a progression of conditions. Each condition consists of 1) harmonicity for *fundamentals across* all models and 2) harmonicity for *partials within* an individual model. Another script, the model-making script, references the score's current value for parameter 1) to develop a pool of fundamentals as a series of tones in its own right. Then, 50 models are generated. For each model, f is randomly selected from the fundamentals pool and higher partials are produced using parameter 2). Individual ceiling light VR characters randomly select one of the models to play.

To add the spinning pulsation effect, we use amplitude modulation. By introducing an additional frequency near a model's fundamental, beatings occur as the waveforms interfere. We first calculate the critical band distance around the fundamental. At this distance, the beatings are very fast, not perceptible as rhythmic, but this gives us a reference. The critical band around fundamental f is approximated by the equivalent rectangular bandwidth (ERB) [11]:

$$\mathrm{ERB}(f) = 24.7 \cdot (4.37 \cdot f + 1) \tag{2}$$

Values between .5% to 8.5% of the ERB result in the rhythmic speeds we desire. For deceleration, the extra tone begins as a higher percentage and interpolates lower.

2.3 VR Characters

2.3.1 Anatomy of Each VR Character Instance

Each individual VR character is an instance of an abstraction, e.g. a particular qubit is an instance of the qubit abstraction. As a technical definition, each VR character is an instance of a prefab in Unity [14] running an instance of a C# Monobehavior class and communicating with a dedicated instance of a poly~ audio engine in Max/MSP. In Unity, a Monobehavior is the class that runs the `Update()` method, executing code once per visual frame. The `Update()` method enables each instance to live out an independent life, animating rules and values over time, affected by the wider system and user input. We next describe three of the VR character abstractions.

2.3.2 Qubits

There are 121 qubits located on the virtual floor, initially appearing as small purple spheres collectively arranged in a grid. The computer and user affect individual qubits, causing those instances to change behavior, or type. We will illustrate some of the major qubit types.

As the experience begins, all qubits default to type 1. These are completely stationary qubits, unaffected by user or computer, appearing slightly translucent.

The computer triggers and shapes type 2 qubits. A type 2 embarks upon a random walk, bounded by a small area near its most recent position, for a random duration. Each type 2 plays the sound of a pin ricochet mass, a density of pins dropping on bricks. This complex sound is reduced to a sonic particle by playing very quietly. As a community of qubits perform random walks, the larger mass becomes a fluttery sound with its own distinct timbre. This can be described more generally as an interesting principle of composition: take a complex sound and greatly reduce its amplitude; use this as the atom to build a new quality of sound mass timbre. The system produces various densities of type 2 activity. Each type 2's translucence becomes more solid, and thus the visual representation of the mass corresponds to the collective sonic density.

The user initiates type 3 behavior. With the use of their hand controllers, they can click-drag on the virtual floor, creating a force at those locations. Qubits close to the force become type 3, repelling away. The sound is similar to type 2, a pin ricochet mass; however, the amplitude is louder, revealing the composite pin ricochet particles. The speed of repelling determines amplitude, and this is a duet of influence. The user determines the strength of force applied to a qubit through proximity of click, resulting in a speed of repelling and corresponding increase in volume. When the user stops inputting, the computer applies a force of friction to slow the qubit, lowering its volume. This is the first independent interactive rule the user discovers, through explorative input. They can then shape this with intent. We will discuss type 4 and type 5 qubits after introducing the voids.

2.3.3 Voids

As the user repels qubits, they likely discover another independent rule: if the negative space between a local group of qubits is large enough, a void will open, bounded by those qubits (see Figure 1). The void is reshaped as the user repels the bounding qubits. An interrelationship arises: qubit positions form and shape voids; the voids also restrict where the bounding qubits can go.

The user creates the timing of a void's sound, but the sound itself is algorithmically determined. The user may discover that by click-dragging on a void, it becomes displaced a short distance from its centroid and when letting go, it springs back. This is another independent rule the user can discover and shape with intent. For as long as the user displaces the void and while it springs back, the engine plays crackling sounds, randomly selected samples of the grass and wire baskets.

Additionally, a particle system, called a *geyser*, emits from the center of a void at algorithmically determined intervals, then traveling upwards (see Figure 1). Discussed shortly, this is the mechanism that activates the ceiling light VR characters.

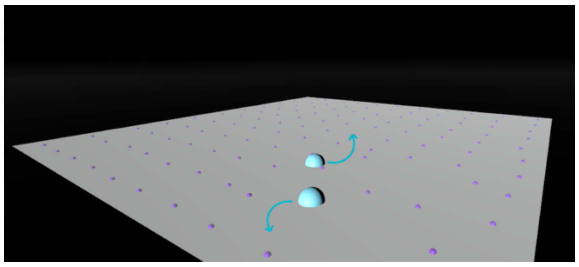
Figure 3. Two entangled qubits. Drawn arrows demonstrate how each qubit mirrors the direction of the other.

2.3.4 Qubits Revisited

The user initiates type 4 qubits, the computer terminates this behavior, and in between, user and computer jointly shape the phrasing. As the user repels qubits, if two collide, they become type 4, another independent rule. As a type 4, instead of repelling from the user's click, they attract to it. The sound is bright noise, made from granularly freezing a wood pin strike sample. Similar to type 3, the user's click applies a force to increase speed and volume and algorithmic friction decreases the volume. Both qubits involved in the collision become an entangled pair (see Figure 3). If the user moves one of them, the other also sounds and moves in a mirrored direction. The computer allows the pair to remain entangled for only a short duration. An additional rule that may be uncovered is that a type 4 is able to pass inside a void, then becoming captured there. This establishes another relationship between qubit and void.

Upon being captured in a void, the qubit becomes a type 5, transforming into a new shape, a tentacle (see Figure 4). By default, a type 5 is algorithmically driven, a tentacle that elongates towards the ceiling and shrinks back into the void (then repeating). This corresponds to swells and fades in volume of darker noise, made from freezing a play position at the end of a wood pin strike sample.

When a user click-drags on a void containing tentacle qubits, they begin a duet of influence over those tentacles. The engine switches from playing dark noise to playing a time stretched metal tube sound, low and resonant pitches. When time stretching in the forward direction, these sounds have a timbral morphology from brighter to darker. Direction of time stretching is jointly determined between user and computer. While a user displaces the void, its tentacles grow brighter (time stretching in reverse). If the user lets go of the void, the tentacles grow darker (forward time stretching). When a void is displaced containing multiple tentacles, harmonies arise between the multiple low tones. Harmonic shifts are also jointly determined. The user prolongs all active tones by continually displacing a void. Tones change when the user lets go and the tentacles shrink into the void. The next time the void is displaced, the algorithm solely determines which pitches change, ensuring there are common tones between harmonic shifts.

Though not detailed in this paper, additional qubit behaviors were designed and implemented: a qubit can briefly split into two spheres and come back together into one; also, a captured qubit can orbit inside its void.

2.3.5 Ceiling Lights

The 36 ceiling lights are activated by chance collisions, resulting from computer algorithms. As previously stated, each void periodically emits a particle system geyser that travels upward. When a ceiling light is struck by a geyser, it grows in visual brightness and then fades away again. This corresponds to swelling and fading of synthesized sound, a randomly selected washer model composed of mid to high frequencies. When multiple lights are struck, they collectively create harmonies. Computer and user jointly contribute to the frequency range of harmonies, scene-wide. The algorithmically controlled ceiling lights contribute higher frequencies and the user-controlled tentacles contribute lower frequencies.

2.4 Long-term Evolution

Again, the long-term evolution is a composed trajectory of four states. The total number of qubits captured in all voids determines the evolution state. As a forward trajectory, the user cannot skip states, capturing one qubit at a time. A user can dissolve a void (freeing its captured qubits), potentially returning to a state several steps back.

In the beginning state, all sounds are noisy. When a user opens voids, geysers disappear before reaching the ceiling. The user is not yet aware of the ceiling lights.

In the second state, there is an algorithmic counterpoint of dark noise, multiple tentacles growing and shrinking. This state introduces the geysers reaching the ceiling, revealing ceiling lights and the first mid to high-pitched harmonies. These tones intermingle with lower harmonies when the user displaces a void containing tentacles.

In the third state, tentacles sometimes reach the ceiling, colliding with and causing lights to spin (see Figure 4). This introduces the synthesized spin pulsation. While displacing a void containing tentacles, the user determines when the tentacles *could* reach the ceiling, and the computer determines whether they travel high enough on that displacement to spin the lights. In this state, there is an interconnectedness among the major VR characters: qubits create voids, voids capture qubits and turn them into tentacles, and both geysers and tentacles play the ceiling lights.

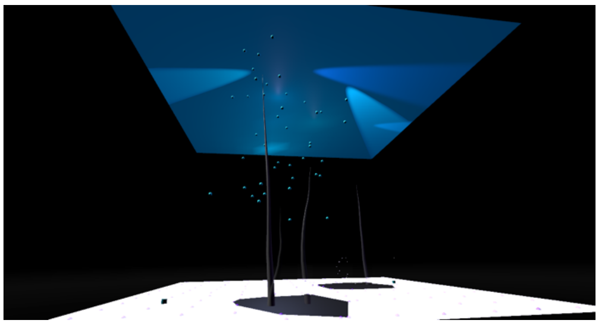

Figure 4. Tentacles eventually cause the lights to spin.

In the final state, geysers no longer cause ceiling lights to glow. Each light runs on its own algorithmic timer, creating a much more active collective. The score script switches to a new progression of 18 conditions, beginning as more harmonic and higher in frequency and gradually becoming more inharmonic and lower. This progression then repeats.

2.5 Global Filtering and Effects

2.5.1 Directional Lighting

An algorithmically-controlled directional light either illuminates the entire scene or rotates away from it, creating various degrees of visual darkness. A fully-darkened scene filters all sounds and a fully-lit scene leaves all sounds unfiltered. The light's rotation angle is mapped to a crossfade, a continuum between filtered and unfiltered.

2.5.2 The Rightside-up and the Upside-down

Again, the user can hear and see *QuBits* through two different lenses, the rightside-up and the upside-down (see Figures 1 and 2). The rightside-up is visually colorful, sounding as previously described. The upside-down is visually warped and in grayscale. In the upside-down, all the sounds of the rightside-up are processed with a chain of effects.

The user begins their experience in the rightside-up. They are transported to the upside-down by passing through a void. In the upside-down, a user's movements shape audio effects. While the user rotates around the floor, transposition and downsampling are shaped. A threshold distance to the floor's center is a switch between two collections of values, sent to the effect engines. The user can zoom in and out to contrast each set of conditions. If a user comes close to a void, they are pushed away and induce feedback from a delay line. The user can repeatedly alternate between zooming in on a void and allowing the algorithmic push, flirting with this explosive feedback. If the user manages to pass through a void, they return to the rightside-up.

2.6 Real-time Challenges

2.6.1 Computational Latency in the First Iteration

Initially, the visuals were created with Jitter and GenJitter, Max/MSP's 3D vector graphics engine. The algorithms driving the system were driven by Max and Jitter data objects, the GenExpr expression language, and the *odot* expression language [12]. odot operates on Open Sound Control (OSC) [13] data bundles and translates them to Max/MSP's native data types. After much development, this version resulted in visual latency.

To troubleshoot, an attempt was made to consolidate multiple GenExpr code boxes into one, however, this proved to be extremely challenging, resulting in many compile errors. We collaborated with multiple experts in the field to troubleshoot these errors and consider alternate designs for the system.[1]

2.6.2 Software and Hardware in the Rebuild

We made the decision to rebuild the visual and control data system in Unity and C#. Max/MSP is retained as the sound engine. The visuals and data processing in this version perform much faster. Unity and Max/MSP communicate by sending OSC data bundles via the User Datagram Protocol (UDP). OSC bundles cannot be understood by either Max/MSP or C#, so additional software runs inside each platform, translating to native data types. In Max/MSP, odot handles this task and in Unity, OSC Simpl [15] translates to C# data types.

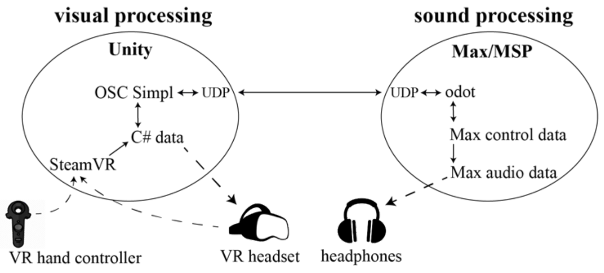

Figure 5. Data flow between software and hardware.

A canvas is then created, where visuals and algorithms in Unity can trigger sounds and algorithms in Max/MSP, and vice versa (see Figure 5). To centralize logic and by preference, we drive the system with algorithms in C#. An additional software component needed is SteamVR [16] to map VR hardware input data in C# scripts.

2.6.3 Granular Sounds Recorded to Samples

Many of the granular rhythms and effects described previously require a lot of CPU due to a high number of overlapping grains. Furthermore, our intentions were to explore an entire mass composed of granular phrases, here as dense as 121 simultaneous voices. Our CPU limit did not permit this many granular engines. Thus, we recorded these granular phrases, assembling them into a large library.

[1] Rob Ramirez, who previously worked on Max/MSP Jitter development helped us with the compile errors. For alternate design strategies, we consulted with Björn Hartmann, Professor of Electrical Engineering and Computer Science at the University of California, Berkeley.

2.6.4 CPU-Intensive Audio Engine Replaced

To render the upside-down, we originally time stretched the sound of the rightside-up, processing this in real-time. Though this sound was very compelling, it was computationally demanding. As a compromise, we redesigned the upside-down with an efficient chain of effects, still providing an alternate lens on the rightside-up.

3. CONCLUSIONS

We think this project is an interesting exploration of generative composition, interactive evolution, sound mass, and sound in virtual space; however, we also believe these aesthetic goals can be better developed in future iterations of *QuBits*.

We are compelled by the sound of the type 2 qubit sound mass, smaller masses of pin ricochets summing to a larger mass with its own distinct timbre. In a future iteration, we will take this idea and scale it to more layers of qubits, each layer having its own timbral identity that weaves in and out of the scene. We also like the quality of sound offered by sampled real-world sound and granular synthesis, however, in experimenting with ever-more layers of activity, synthesis could offer greater control, especially when needing to edit timbral identity.

Additionally, the experience of space in *QuBits* is not as immersive as we desire. One issue in this iteration is that distance to a sound source does not affect the volume of that source. This can be addressed with a spatial audio engine. Also, the user is constrained to a small area of virtual space they cannot move beyond. Rather than solve this issue of scope and immersion for VR, we would rather *QuBits* be adapted to physical space. With physical space, even given the same spatial scope, we believe a more immersive experience would result. Some of the virtual physics would need to be remapped.

The premise that the user influences the long-term evolution is interesting to us, but the rule deserves reexamination. Each state change in the current iteration is an instantaneous switch the moment a qubit is captured in a void. Instead, an accumulation of energy could be employed. For instance, a twice-repelled qubit could become more frenetic in its type 2 random walk, growing louder in volume. The mass would build towards a threshold number of these louder qubits, eventually triggering the next state. The rule that evolves the evolution state could also change as the user's experience progresses.

Figure 5 diagrams the hardware and software comprising the system, solving initial latency issues.

We believe that this system will be helpful as a starting point for composers and sound artists aiming to create experiences with audiovisual systems.

4. REFERENCES

[1] *Vive VR System*. (2019). Vive. [Hardware].

[2] A.H. Stjerna, *Sky Brought Down*. Gothenburg, Sweden: Sahlgrenska University Hospital, 2017.

[3] R. Henke, *Fragile Territories*. Lima, Peru: Espacio Fundacion Telefonica, 2019.

[4] M. Musick. "A physically-distinct, multi-agent, sonic space ecosystem," in *Proc. 2018 International Computer Music Conference*, Daegu, Korea, pp. 276 – 281.

[5] N. Barrett, *OSSTS*. Oslo, Norway: Oslo Contemporary Music Festival, 2014.

[6] P. Weir. "Stealing sound: the use of generative music in the next Thief", presented at the *2011 Game Developers Conference*, San Francisco, CA, Feb. 28 – Mar. 4, USA, 2011.

[7] R. Gottfried. *o.granubuf~*. (2016). Regents of the University of California. [Online]. Available: https://github.com/CNMAT/CNMAT-odot

[8] *Max 7*. (2016). Cycling '74. [Online]. Available: https://cycling74.com/downloads

[9] A. Freed. *resonators~*. (1999). Regents of the University of California. [Online]. Available: https://github.com/CNMAT/CNMAT-Externs

[10] C.E. Cella, "Generalized series for spectral design," www.carminecella.com, 2013.

[11] B.C.J. Moore and B.R. Glasberg, "Suggested formulae for calculating auditory-filter bandwidths and excitation patterns," *J. Acoust. Soc. Am.*, vol. 74, pp. 750-753, 1983.

[12] J. Maccallum, R. Gottfried, and I. Rostovtsev, J. Bresson, and A. Freed. "Dynamic message-oriented middleware with Open Sound Control and odot," in *Proc. 2015 International Computer Music Conference*, Denton, TX, USA, pp. 58 – 64.

[13] M. Wright and A. Freed. "Open Sound Control: a new protocol for communicating with sound synthesizers," in *Proc. 1997 International Computer Music Conference*, Thessaloniki, Hellas, pp. 101–104.

[14] *Unity*. (2018). Unity Technologies ApS. [Online]. Available: https://store.unity.com

[15] *OSC Simpl*. (2018). Sixth Sensor. [Online]. Available: https://assetstore.unity.com/packages/tools/input-management/osc-simpl-53710

[16] *SteamVR*. (2019). Valve Corporation. [Online]. Available: https://store.steampowered.com/app/250820/SteamVR