UNIVERSITY OF CALIFORNIA, SANTA BARBARA

Lithe: An object-based audio-graph framework for spatial composition and sound design

by

Akshay Cadambi

Project documentation submitted in partial fulfillment for the degree of

Master of Science
in
Media, Arts and Technology

January 2017

Lithe: An object-based audio-graph framework for spatial composition and sound design

Abstract

by Akshay Cadambi

Over the recent years, content creation in spatial audio has moved away from multi-channel to object-based approaches, where instead of specifying audio signals for each speaker channel, audio is created with positional data. This is then rendered to speakers using one of the many spatial sound rendering techniques (VBAP, DBAP, Ambisonics, etc.). However, workflows for creating content in such systems are usually ad-hoc and vary vastly between software, algorithms, and controllers. The unit generator (UGen) is an abstraction used in MUSIC-N languages to structure and define sound processing as an interconnected graph. This project will present a novel extension of the UGen graph for object-based spatial audio. By abstracting spatial position as a continuously varying signal transmitted along with the audio, we can allow for interconnections between processing blocks that exert transformations on both the sound and its spatial information. This project attempts to define a framework for trajectory processing, while being independent of speaker layouts, rendering methods, and compositional motivations. Finally, this project presents *Lithe*, a C++ framework to build object-based audio-graphs. It will also present an proof of concept use-case of this library with a modular synthesis application called *Lithe Modular*, as well as a sound installation *HIVE*.

Acknowledgements

I would like to express my gratitude to my committee Dr. Curtis Roads, Dr. Andrés Cabrera, and Dr. Clarence Barlow for their guidance toward this project. I would also like to extend my gratitude to the MAT community including faculty, friends, colleagues, and alumni, for providing such an inspiring, academically stimulating atmosphere. Finally, I thank my family for their unceasing support.

Contents

Chapter 1

Introduction and Motivation

The use of space as a musical parameter has existed for many decades now. In instrumental music this involved physically positioning the instrumentalists throughout the concert hall. The invention of the loudspeaker extended this idea into the electronic medium, where instead of instrumentalists, composers were concerned with the placement of loudspeakers. A notable early example is Varèse's *Poeme Electronique* that was showcased at the 1958 Brussel's Worlds Fair distributed sounds across 400 loudspeakers and used switching circuits to orchestrate movement of sounds. Another example is John Chowning's piece *Turenas* where trajectories were based on the Lissajous curves, and used a custom algorithm [1] to spatialize it over four loudspeakers.

Over time the growing need to make compositions portable, pushed toward the development of standard speaker layouts like stereo, quadraphonic, octaphonic, 5.1, 7.1, etc. Of these, the two channel stereo layout is by far the most ubiquitous. Quadraphonic and octaphonic speaker layouts, consisting of 4 and 8 speakers respectively, arranged in the form a ring around the listener. These are more common in electroacoustic music concerts and conferences. 5.1 and 7.1 were born out of the needs of cinema industry, where the 5 and 7 respectively represent the number of channels surrounding the listener. The '.1' represents a separate sub-woofer channel used to reproduce low-frequencies that the typical speaker drivers could not, owing to its other name – the Low Frequency Effects (LFE) channel [2]. In all cases, standard speaker layouts allowed composers to compose and mix in their studio with the expectation that the performance space had the same speaker layout.

The common assumption with this approach is that the studio and performance space both have the same similar speaker layouts, acoustics, and dimensions. Any discrepancies between the two in these aspects could result in inaccurate spatial imaging. Further, once printed, a piece is largely unchanged and any performance of the piece is usually tied to the speaker layout that it was made with. This approach is generally called the channel-based approach since the composer composes for a set of speaker channels.

An alternative to this is the *object-based* or *object-oriented* workflow. Here, the composers create sounds with positional metadata in a virtual space. Most of these systems allow for the specification of a spatial trajectory for a sound source using coordinates. It then uses a rendering algorithm to determine the individual speaker signals needed to position the sound for a given speaker system. Each sound source together with its positional metadata is called as an *audio object* (or sometimes *sound object*). Examples of rendering algorithms include DBAP [3], VBAP [4], Ambisonics [5], Wave-field Synthesis [6], and proprietary technologies like Dolby Atmos [7], DTS:X [8] and Auro 3D [9].

Several software tools [10][11][12][13][14][15][16] currently exist for composing spatial music using object-based approaches. Fig 1.1 shows a representation of what occurs in most such tools in order to render a single sound source. The audio signal is produced in the sound synthesis block and includes of audio processed from a live input, synthesized, recorded, or any combination of these. The position information associated with the audio signal is produced by the trajectory generation block. The position and trajectory information are combined to obtain the audio object. Finally, spatializer takes the audio object and renders it using one of the many rendering techniques like VBAP, WFS, Ambisonics etc. In some cases [17] the sound processing and trajectory generation are made to correlate using an overarching simulation algorithm.

Trajectory generation typically takes place in one of two ways: it is manually via user input, or algorithmically generated. Some tools allow a hybrid of both these approaches but lean heavily on one or the other as far as their workflow is concerned. Zirkonium MKIII [12] allows for trajectories to be drawn using an interface and then allows for queuing these trajectories as events in the time line. It also allows for algorithmic control via OSC. *Spatium* [13] implements interfaces that run simulations of gravitational forces, pendulum motion, and spring motion to generate trajectories. Spatial Swarm Granulation [17], is an example where an overarching simulation algorithm informs both the sound processing and trajectory generation. It uses the boids flocking algorithm to both generate and position sound grains in space.

Since each software uses a separate workflow, this leads to pieces largely being tied down to the software that they are made with. E. Lyon [18] talks about the diminishing returns yielded from exploration of timbre composition in the recent years in comparison to the early days of computer music and suggests the need for the exploration of spatial composition. Further, composers often have to re-orchestrate compositions due to the differing speaker layouts in the venues. From a workflow perspective, it is true that we've reached a point in timbre composition where most Digital Audio Workstations

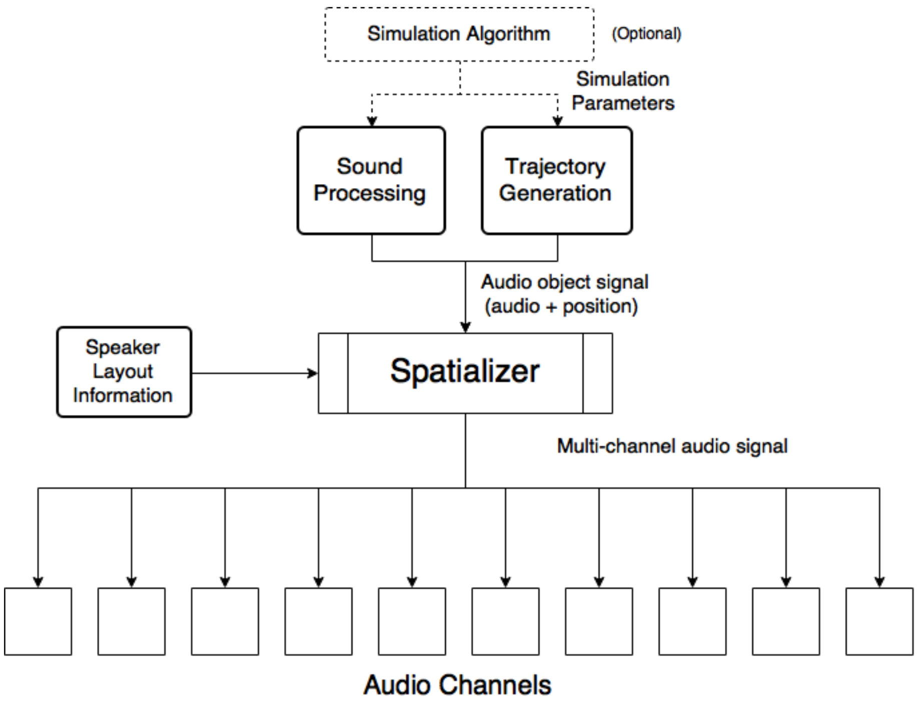


FIGURE 1.1: A general view of the signal flow in an object-based audio rendering system. The sound synthesis process produces the audio signal, and the trajectory generation process (GUI or some algorithm) produces coordinate information. The combined audio+position signal is passed into the spatializer which uses a rendering method (like VBAP, DBAP, Ambisonics etc.) to produce the individual channel signals.

(DAWS) are capable of recording, synthesizing, mixing, and potentially mastering a stereo composition with in-the-box tools and plug-ins. A user who is familiar with the idioms of sound synthesis and processing will be able to achieve similar results in most DAWs. Additionally re-mixing a piece (timbrally) to acoustically suit different venues is also achievable with most DAWS. Comparatively in spatial composition, the workflow incompatibilities between software tools hinders the development of similar idioms and obviates easy re-orchestration for different venues.

Newer synthesis methods make use of trajectories and spatial information as a means to explore timbre. Spatial Modulation Synthesis [19] for example uses fast-moving sound sources affected by Doppler effect to generate FM-like tones. Spatio-temporal granulation [20] makes use of sound-field recordings to explore both temporal and spatial information as a way to parameterize grain generation. However, these new methods exist either as ad hoc or proof of concept implementations [15] since typical DAWs lack the ability to explore them. Further, there is also a potential range of effects that could employ cross-adaptive processing between the position and audio allowing for audio to be modulated based on its spatial position, or trajectories modified based on the real-time analysis of audio signal. Negrão's Immlib library [21] is a step in this direction, however, workflow limitations in most tools limit the exploration of such effects.

This project will draw inspiration from the *unit generator* [22], an abstraction that was introduced in the MUSIC-N languages that is still ubiquitous in sound processing. It will propose a novel extension of unit generator graph to allow for the processing of audio objects. It will also attempt to define a framework for spatial position processing that is independent of the rendering methods, is similar across regular and irregular speaker layouts, and inherently allows for cross-adaptive processing between audio and position signals. It will then implement these ideas in the form of a C++ library. It will show the use cases for such a framework via two projects *Lithe Modular* and HIVE.

The following chapter will lay out some of the design considerations taken towards the development of the object-based audio graph. It also discusses the strategy for the development an abstract representation of spatial position using 2-d manifolds. Chapter 3 specifies the signal specification of the audio object signal, the signal that is passed between the nodes of the audio-graph. Chapter 4 discusses the implementation of the object-based audio graph library *Lithe*, as well as the modular synthesis application *Lithe Modular*, and the sound installation HIVE. Chapter 5 presents potential implications for spatial computer music and points to directions for future work. Chapter 6 presents a concluding discussion.

Chapter 2

Design

The *unit generator* (UGen) a concept first introduced in the MUSIC-N languages [22], is still one of the most widely used abstractions in computer music software today [23]. A UGen is a basic building block for sound synthesis capable of processing input signals to produce output signals. Examples include filters, oscillators, adders and multipliers. A number of fairly simple UGens could be interconnected to design vastly complex sounds. In the words of to Max Matthews, one of the developers of the MUSIC-N, this allowed complexity of the program to "vary with the complexity of the musician's desires" [24]. Figure 2.1 shows a typical UGen graph[1] with three UGens.

[1]The terms audio graph and UGen graph are used synonymously in this document.

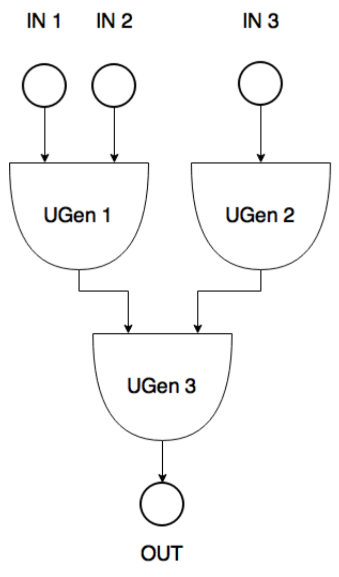


FIGURE 2.1: A typical audio graph comprised of three UGens. The inputs to the UGens may either be a signal from a prior UGen, or may be static parameters that influence that UGen's processing.

Several computer music languages and software incorporate the unit generator concept within their workflows. Examples include CSound [25], SuperCollider [26], ChucK [27], MAX/MSP [28], PureData [29], and Reaktor. Even in DAWs, where the audio graph structure is mostly hidden from the user, still follow a UGen-graph based signal flow for plug-ins, inserts and, side-chained effects.

The concept of UGens share an analog with the world of hardware modular synthesizers, which also rely on the principles of modularity and interconnections to achieve complexity in timbre or musical structure. With the growing resurgence of hardware modular synthesizers especially in the Eurorack format[2], many new modules have digital processors capable of advanced DSP, allowing digital techniques like granular synthesis to take on a hardware interface in a modular form.

However, in order to make this extension of the unit generator generic enough to serve as a framework for spatial audio, considerations need to be made with respect to spatial sound rendering techniques, speaker layouts, and co-ordinate systems. The following two subsections describe these considerations.

2.1 Independence from rendering algorithms and speaker layouts

Many spatial music performances are acoustically nuanced, and the workflows depend on the composition, the performance space, and the composer. Some composers take a "set and forget" approach where a composition, once finished is not modified any further. Performance in such cases involves pressing the play button. Other composers like Jonty Harrison, Natasha Barrett, and Curtis Roads employ a technique called live diffusion [30] where they actively modify aspects of a finished piece during the performance. This is to better adapt the piece to the space and speaker layout.

Additionally, rendering methods like WFS or Ambisonics work best with specific kinds of speaker layouts and dictate a certain regularity of arrangement and spacing between the speakers. WFS [6] for example, requires a regularly spaced array with a hard upper limit on the distance between adjacent speakers. Others like DBAP function despite irregular layouts. An example for an irregular layout includes a multichannel sound sculpture HIVE[3] that the author worked on that consisted of speakers pointed divergently outward from the center at different directions, elevations.

[2]EuroRack is a hardware modular format introduced by the manufacturer *Doepfer* (http://www.doepfer.de/home_e.htm)

[3]This project is further discussed in Chapter 4.4

By making the object-based audio graph independent of both speaker layout and rendering method, it will be able to cater to both regular and irregular speaker layouts using a similar workflow. It will also allow for the ability to make minute changes or even completely replace the rendering algorithm without affecting the spatial orchestration. Since the studio is often acoustically different from performance spaces it allows the composer to fine-tune the rendering algorithm to better suit an acoustically different performance space.

2.2 Choosing a coordinate system

A Cartesian system would be an obvious and general way to convey spatial position. However, it presents several limitations. Unless qualified by additional parameters, a signal consisting of Cartesian coordinate positions is of unbounded range. Further, it conveys no information about the expected geometry of movement leaving little room for signal manipulation and processing. To use a term from information theory, a Cartesian position signal can be considered as a high entropy signal.

In comparison, lower entropy signal would be in the spherical coordinate system, where two of the three signals (azimuth and elevation) are of bounded range, and the third (radial distance) is unbounded. This lowers the entropy because we can expect modulations in azimuth to occur with a circular geometry and idioms for processing this signal can be better defined. For example, an elevation signal normally in $[-\frac{\pi}{2}, \frac{\pi}{2}]$ can be ‘compressed’ to a smaller range thereby reducing the height content in the piece. However, using spherical coordinates to represent trajectories that lie on a flat plane presents a few hurdles. In general, using a parametric coordinate system like spherical or cylindrical coordinates ties the composer down to trajectories that are geometrically similar to the base geometry of the co-ordinate system.

Therefore, in order to best suit modulation and processing idioms that are similar to those used in audio, it desirable to use a position signal that abstracts away the base geometry of the coordinate system. Additionally, having a system that is of bounded range makes it easier to define dynamic range-based processing effects on the signal. These make it easier to develop UGens that are highly decoupled, yet easy to interconnect, to be able to process signals similarly despite the geometry of the coordinate system. The solution chosen for this project is the use of 2-dimensional manifolds because they can be represented by a set of common local coordinates, but yet can map onto many different base geometries.

Chapter 3

The Audio Object Signal

This project relies on the abstraction of coordinate as points on a 2-dimensional manifold. The immediately following section describes the specification of the signal, and succeeding sections further describe the sub-components of these signals and how they are used.

3.1 Signal Specification

Every sample of an audio object signal contains the following essential components:

Audio :	Monophonic audio signal that is associated with this audio object
u :	Horizontal coordinate on the manifold's local coordinate system
v :	Vertical coordinate on the the manifold's local coordinate system
d :	Distance parameter projected perpendicular to the manifold surface at (u, v)

u and v are the local coordinates of the 2-dimensional manifold used.

d refers to the distance parameter. The use of this will be further explained in section 3.3.

3.2 Usage of Manifolds

In context of this project, a 2-dimensional manifold can be thought of as a 2-d surface embedded in 3-d space. Fig. 3.1 shows some examples of familiar manifold surfaces. The

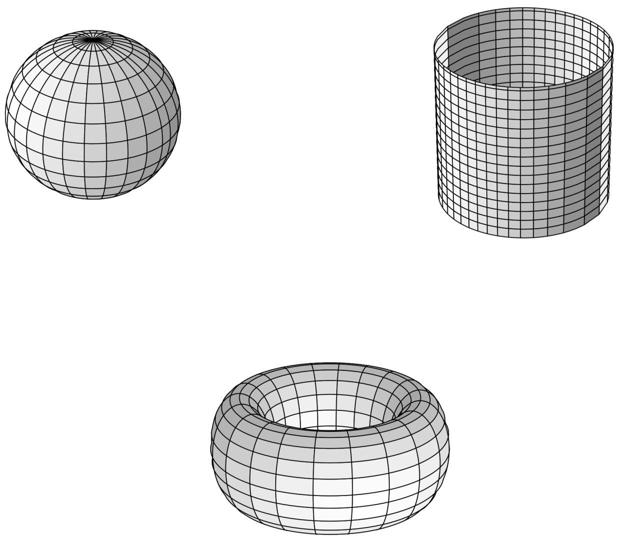


Figure 3.1: A few examples of the manifold surfaces that the (u,v) plane can be parametrically mapped to: sphere (left), cylinder (right), and torus (bottom)

use of 2-dimensional manifold surfaces as a strategy to abstract coordinate information is not new in the field of spatial audio. Manifold Interface Amplitude Panning (MIAP) [31] is a design paradigm by MeyerSound that uses an interface called *SpaceMaps* that considers the speaker layout as points on a manifold. It was intended to cater to live-diffusion as well as stereo diffusion performances [32]. The Immlib library [21] for the SuperCollider environment uses a technique called parameter field spatialization [33], where scalar fields defined on manifold surfaces are used as a way to derive sound synthesis parameters

Manifolds allow differently shaped smooth surfaces to be mapped using a common local coordinate system consisting of a 2-dimensional euclidean plane with the coordinates (u, v). Fig. 3.2 shows a (u, v) plane with both u and v in the range [-1, 1] (as implemented in this project). As an example, mapping a spherical manifold would be as shown below. The u and v being similar to the longitude and latitude respectively on the earth's globe:

$$u[-1,1] \rightarrow azimuth[-\pi,\pi]$$
$$v[-1,1] \rightarrow elevation[-\tfrac{\pi}{2},\tfrac{\pi}{2}]$$

A mapping function, or a set of mapping functions exist to transform all points on the (u, v) plane to their respective 3-dimensional Cartesian coordinates of points on the

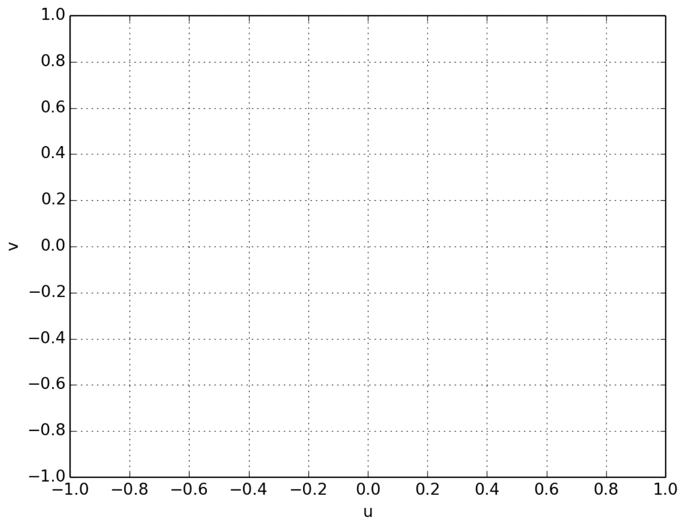


FIGURE 3.2: A (u, v) plane is used to represent the local coordinates of a 2-d manifold.

manifold. In topology, this mapping function is commonly called a chart, and a set of charts that together describe all points on the manifold is called an atlas.

Dynamic manipulations on these u and v signals result in spatial transformations on the trajectories of the sound sources. DC shifting results in the trajectory to be moved to a different location on the manifold[1]. High frequency signals directly relate to faster movement, and lower to slower movement. This allows the usage of filters in the spatial domain as a means for velocity manipulation. Dynamic compression[2] can be used to 'compress' the span of movement while preserving the overall shape.

Some manifolds wrap around the extremities of the ranges of the u and v signals. In this case, DC shifting can be applied with a wrap-around to the other extremity of the range to allow for continuous manipulation. For example, discontinuous signals like a full range sawtooth wave can create continuous motions when used as the u signal in a radially symmetric manifold like cylinders and spheres. In this case, simple oscillator waveforms like sine, saw, triangle result in motions along the circumference of a circle. Any other trajectories can simply be sampled as a wave table and played back; with speed of playback relating directly to the speed of motion. Common techniques from tape music like splicing, varispeed, and reversal, result in truncated trajectories, modulated velocities, and trajectory reversal respectively. Thus, it is easy to see that with little

[1]Admittedly, this assumes that the manifold doesn't contain any inherent spatial distortions. This is further discussed in Section 5.3.1.

[2]The only caveat being that care must be taken to apply any DC shifts *before* compression since unlike audio signals, u and v signals can contain DC content.

modification, commonly known audio processing techniques can be re-used in the spatial realm allowing for simple, yet powerful trajectory generation idioms to develop.

3.3 Usage of d-maps

If one is using a spherical manifold, d can be used as a metric to map radial distance. By extending this idea further, d can be used to map distance as mapped onto the perpendicular drawn to the surface of a point on the manifold. Essentially, this can then be used to map points outside the manifold surface while still keeping object motions congruent with the geometry of the manifold. Additionally, this allows for very irregular speaker layouts to operate in a consistent manner.

The following equation gives the vector calculation of a point based on the u, v, and d taken from the audio object signal.

$$\vec{P} = \vec{P_m} + (\hat{N} * multiplier)$$

where,

$\vec{P}$	is a point to be calculated
$\vec{P_m}$	is the point on the manifold obtained from the (u, v) coordinates
$\hat{N}$	is the unit normal vector to the manifold at point $\vec{P_m}$
$multiplier$	is the perpendicular distance from the manifold from the d-map function

The d-map is a functions used to obtain the multiplier used in the above equation using the d value in the audio object signal. An example d-map would be a piecewise linear map where d in [-1, 0, 1] is linearly mapped to [-a, 0, b]. Then, a d of 0.5 would result in a multiplier of value $-\frac{a}{2}$, and that of -0.5 would be $\frac{b}{2}$. Other mapping functions could be used with different bounds to reach points at different proximities from the manifold with different, possibly non-linear, response curves. Fig 3.3 shows examples of piecewise linear, piecewise quadratic d-map function responses. In general, the larger the multiplier, the further the point is from the surface of the manifold.

3.4 System Diagram

Fig 3.4 shows the system diagram of the proposed object-based audio graph, here called a Lithe audio graph. It can be compared with Fig 1.1, where the sound processing and

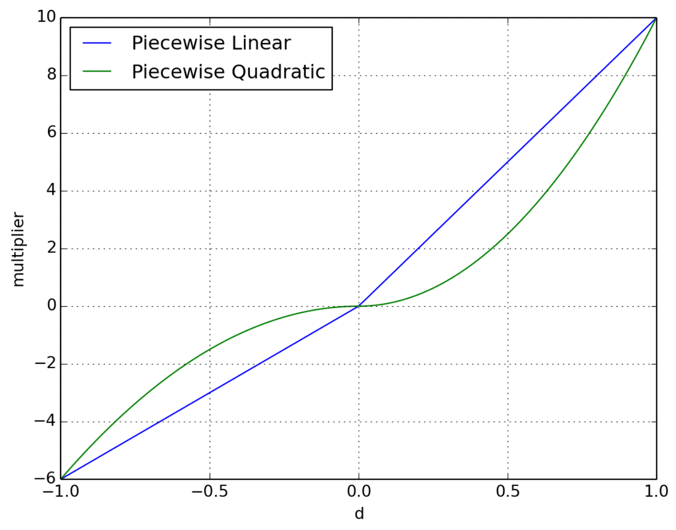


FIGURE 3.3: Examples of d-map functions that map from d in [-1, 1] to [-6, 10]. Examples shown here use a piecewise linear and piecewise quadratic model that ensures that d=0 maps to a multiplier also equal to 0.

trajectory generation are fairly separate processes. Here they occur simultaneously since the signals that pass between UGens are audio object signals which inherently consist of audio and positional information.

The final UGen in the audio graph (which will be called the *Sink UGen*) wraps the rendering algorithm within it. It takes the speaker layout information as a parameter and renders the incoming audio objects to the speaker layout by making use of the Manifolds and d-maps to convert the audio object signal's position information contained in the u, v, d signals into Cartesian coordinates. Since all rendering methods are wrapped as UGens and hence have a common interface, it is easy to switch out one Sink UGen for another, effectively changing the rendering algorithm.

For most typical surround systems, a radially symmetric manifold like the sphere, torus or cylinder would be an obvious choice. However, for more unconventional speaker layouts one may choose more complicated geometries to better suit the speaker layout or the intended trajectory patterns.

Further, by adding additional metadata that specifies the manifold and d-map to be applied to the audio object to those already proposed in 3.1, it is possible for audio objects on different manifolds to be processed by the same graph. These kinds of groupings of semantically related audio objects allows easier control when working with a very large number of objects.

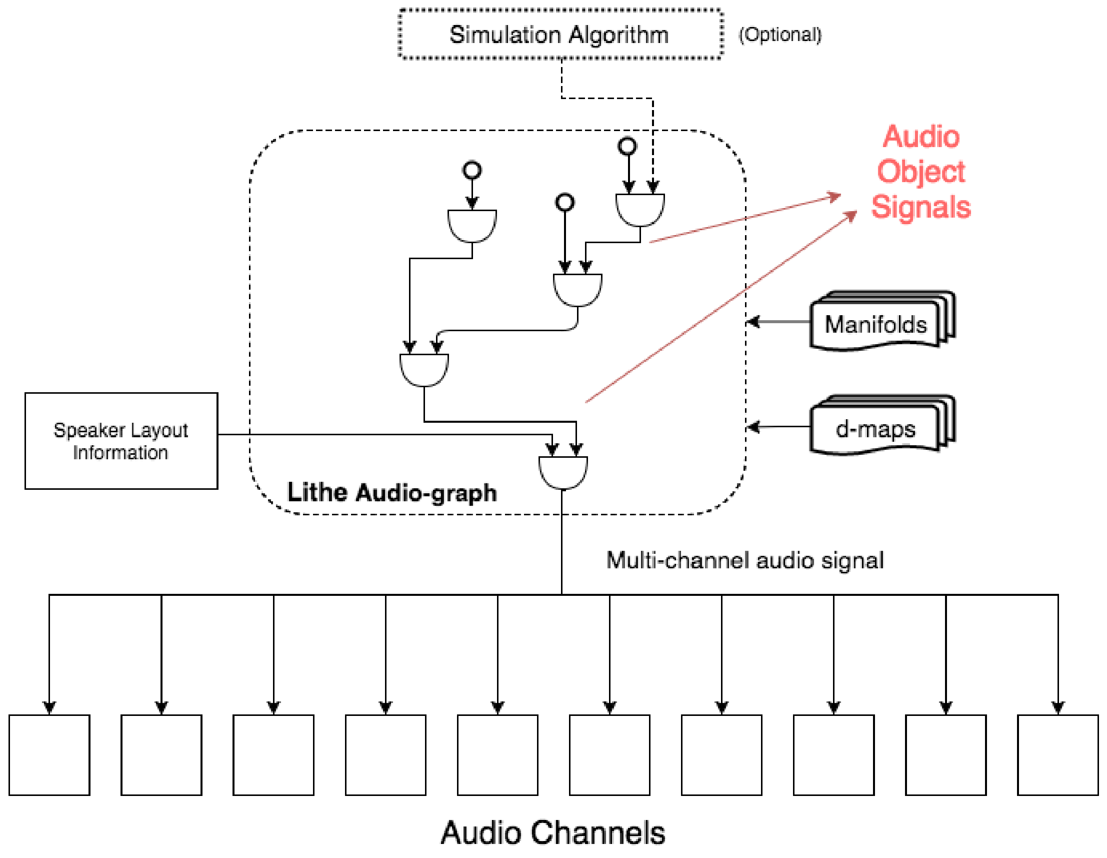


FIGURE 3.4: System diagram of an object-based audio graph, here called a Lithe audio graph. Signals passed between UGens are audio object signals consisting of both audio and positional information. Audio objects are processed as a whole, and the final UGen in the graph renders this to the actual speaker layout.

Chapter 4

Implementation and Usage

This project is implemented in three distinct parts: *Lithe*, *allolithe*, *Lithe Modular*. *Lithe* is a C++ library acts is a framework for creating and processing an object-based audio graph. *allolithe* is a module based on Lithe for the *AlloSystem* framework. And finally, *LitheModular* is a proof of concept modular synthesizer inspired application built using allolithe.

4.1 Lithe

Lithe is a C++ library intended for creating and processing object-based audio graphs. Lithe has no dependencies other than the C++ STL.

Lithe can be functionally divided into three modules: AudioGraph, Atlas, Utilities. The AudioGraph module consists of audio graph components implements the graph processing functionality. The Atlas module provides a class interface for defining manifolds and d-maps. Currently implemented are spherical and torus manifolds. Finally, the Utilities module consists of classes that provide functions such as topological sorting, signal range processing, and modulation.

Lithe's audio graph provides the following classes: *Inlet*, *Outlet*, *Node*, *Patcher*. Table 4.1 provides further description of these classes. Of these, the *Node* class is the skeleton for creating a UGen. It has a processing function that processes the audio object signal at the inlets to produce the output signals at the Outlets. This processing function is defined by the user of the library. Every *Inlet* in-turn triggers the processing function of the connected upstream UGen, this is common to most pull-based graph processing architectures like the Jamoma Audio Graph [34]. It uses per-sample processing under the hood, but can be easily overridden to do buffered processing of audio object samples.

Class	**Description**
Inlet	Pulls samples from as single connected Outlet. Has an internal single sample delay that can be enabled by a sorter to allow for feedback.
Outlet	Holds samples that result from a Node's processing function that can be read by connected Inlets. It can connect to multiple inlets at once, allowing for 'fanned' connections.
Node	Consists of Inlets, Outlets, and a processing function that can be defined by the user of the library. Provides the framework for creating a UGen.
Patcher	An interface that is used to connect or disconnect Inlets and Outlets

Table 4.1: Description of some of the classes within the AudioGraph module. The AudioGraph module consists of components used to build and process the audio graph.

Lithe also provides a class interface for topologically sorting an audio graph using the *Sorter* class in order to enable and disable these single sample delays as they seem fit. Currently implemented is a Breadth First Search (BFS) based sorting algorithm that traverses a graph from the bottom-up and turns on sample delays on any *Inlet* that is connected to a back edge. However, the interface allows for other sorting methods like Tarjan's algorithm to sort the graph and process the UGens in a specific order.

Lithe is intended as a core processing framework upon which applications are built and hence does not implement for interfacing with audio IO, GUI, and other application-level functionality.

4.2 allolithe

allolithe is a submodule of AlloSystem [1] that depends on Lithe. It wraps many of the Lithe classes and incorporates the AudioIO, lock-free parameter classes within AlloSystem to allow for easy development of UGen applications. Additionally, it also implements classes to create a GUI using the GLV [2] library for visualizing the audio graph patching UGens together in real time.

The main UGen-related classes in allolithe are Module, SinkModule, and InputModule. *Module* wraps the Lithe *Node* class and is essentially the equivalent of a UGen. It also provides an interface to utilize the lock-free *Parameter* class from AlloSystem enabling the control of UGen parameters using GUIs or potentially, OSC-based interfaces over the network. The *SinkModule* extends the Module class to provide access the Audio IO device to write to the speaker channels. A SinkModule is typically where the spatial sound rendering algorithm is placed. The *InputModule* is similar to the SinkModule

[1]AlloSystem [35] is an open-source C++ framework for developing audio visual applications.
[2]GLV [36] is C++ based cross platform GUI library.

except that it uses the Audio IO device to pull input from an external source (line-in or microphone) and process it within the audio graph.

In addition to the above classes, it also provides an *SoundEngine* class that acts as an audio graph manager. It is able to dynamically spawn new instances of Modules, make graph connections, maintains a list of all active module instances, and also owns all the available Manifolds and d-maps that will be used in the graph. Since it provides a uniform entry point to the entire audio-graph, it can be considered as the starting point to allow for a distributed processing architecture, like the client-server architecture found in SuperCollider. Future versions might move this functionality into the Lithe framework itself.

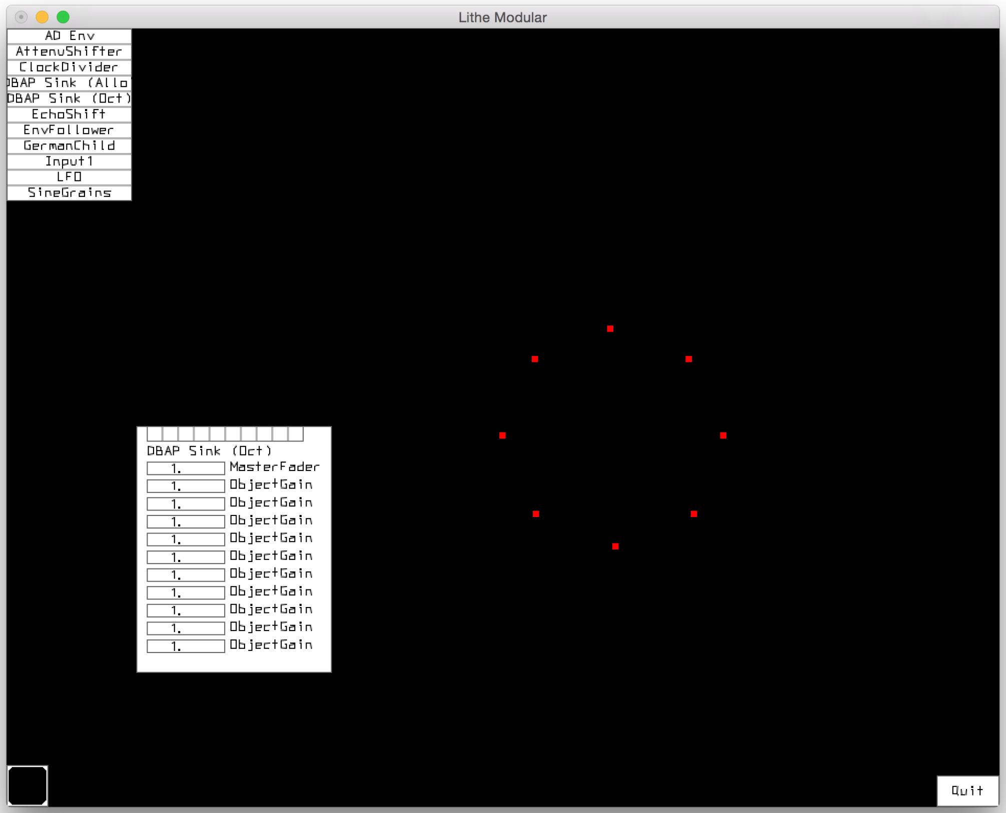
4.3 Lithe Modular

Lithe Modular is proof of concept application built using allolithe and AlloSystem. It is intended to be a modular synthesis environment for spatial sound. At it's core is the Lithe audio-graph, which is wrapped by the allolithe submodule mentioned in the previous section.

Figures 4.1, 4.2, 4.3, 4.4 demonstrate some of the functionality of the Lithe audio graph. The basic interface of the application is shown in Fig 4.1. On the top left are the list of available modules. On the bottom left is the button to run/stop graph processing, and finally on the bottom right is the quit button. When a sink module is instantiated, it shows the speaker layout associated with that sink from the top view. The red dots indicate the individual speakers. Here we have a sink module with an octaphonic ring, that uses DBAP as the rendering algorithm. Note that the sink itself has several parameters (in this case, per-object gains and a master gain), as well as inputs.

While most modules that were used in Lithe Modular like LFOs, AD Envelopes, Clock Dividers, are common to modular synthesizers, they take on an additional function in the patch since they are also able to affect trajectories of sound sources. A discussion of some of the implemented modules can be found in Appendix A.

Lithe Modular was publicly demonstrated in the AlloSphere at Elings Hall, UC Santa Barbara on January 13th 2017. The AlloSphere's audio system [37] consists of 54 speakers arranged at three different elevations and pointed inward through the surface of a sphere. It was possible to move sounds both below and above ear level as well as around the audience.



DBAP Sink (Oct)
1. MasterFader
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain

Figure 4.1: The interface of Lithe Modular. Here we have a sink module which uses DBAP as the rendering method for an octaphonic speaker layout.

4.4 HIVE

allolithe was also used in the project HIVE as the core sound generation algorithm. HIVE is a sculptural sound installation that uses 16 channels of audio with embedded speakers pointed outward from the center. Each speaker points at a different direction at three different elevations. Adding to the complexity, an additional quadraphonic speaker system was placed external to the sculpture in the four corners of the room with the intent to provide a foreground-background dichotomy in the soundscape that was emitted by the sculpture. Figure 4.6 shows they top view of the installation.

All the sounds used were taken from recordings, and then manipulated and spatialized using modules similar to those used in Lithe Modular. This was approached using a spherical manifold, with the radius that is halfway between the sculpture and the external quadraphonic setup. By selecting a piecewise linear d-map, positive d values resulted in sounds projected from the sculpture and negative d values on the quadraphonic setup. This proved to be an easy and effective way to control call and response-like sonic events

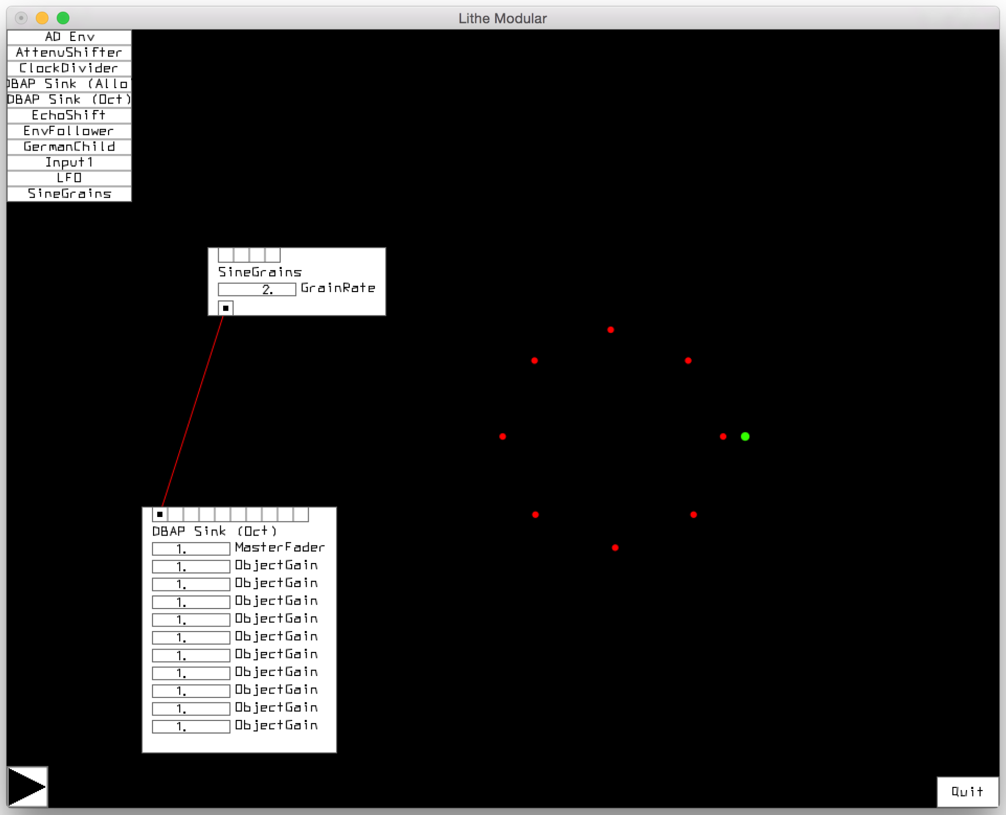
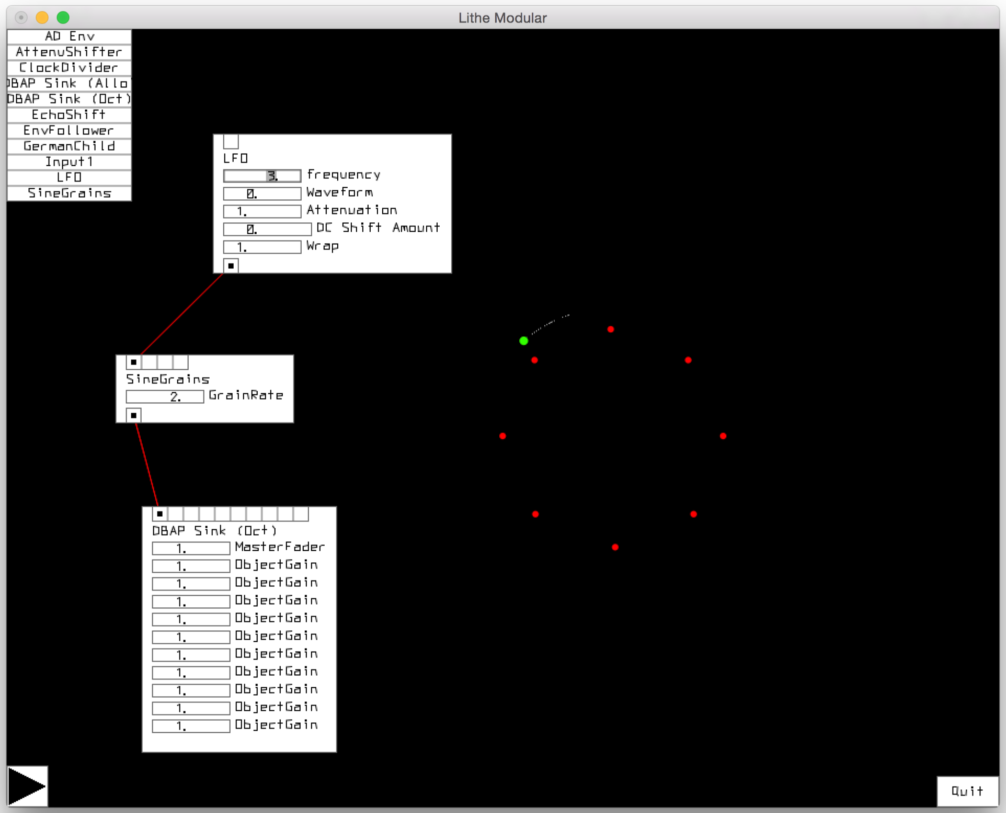


FIGURE 4.2: When an inlet is connected to something, the sink module automatically draws that object in its position. In this case it is a static object mapped to a spherical manifold currently at zero azimuth (u = 0). Here we have the SineGrain module, a grain generator that generates grains at a specified rate from a 1000Hz sine wave.

between the sculpture and its "environment" (i.e., the quadraphonic system). Further, audio input was taken from piezoelectric sensors placed under the carpets in the space. This was then used with onset detection modules to trigger envelopes that modulated various other modules that modulated the trajectories of the sound sources. This added an additional dimension of interactivity to the installation allowing the audience to affect the soundscape of the installation by walking around the space.

HIVE[3] was conceived in collaboration with Şölen Kıratlı and was shown from December 1st to 11th, 2016 in Studio F at the *Santa Barbara Center for Art Science and Technology* (SBCAST), Santa Barbara, CA. It was created in association with Prof. Yon Visell and the Re-touch Lab, and was also partly funded by the Interdisciplinary Humanities Center (IHC).

[3]More information can be found here: akshaycadambi.com/hive

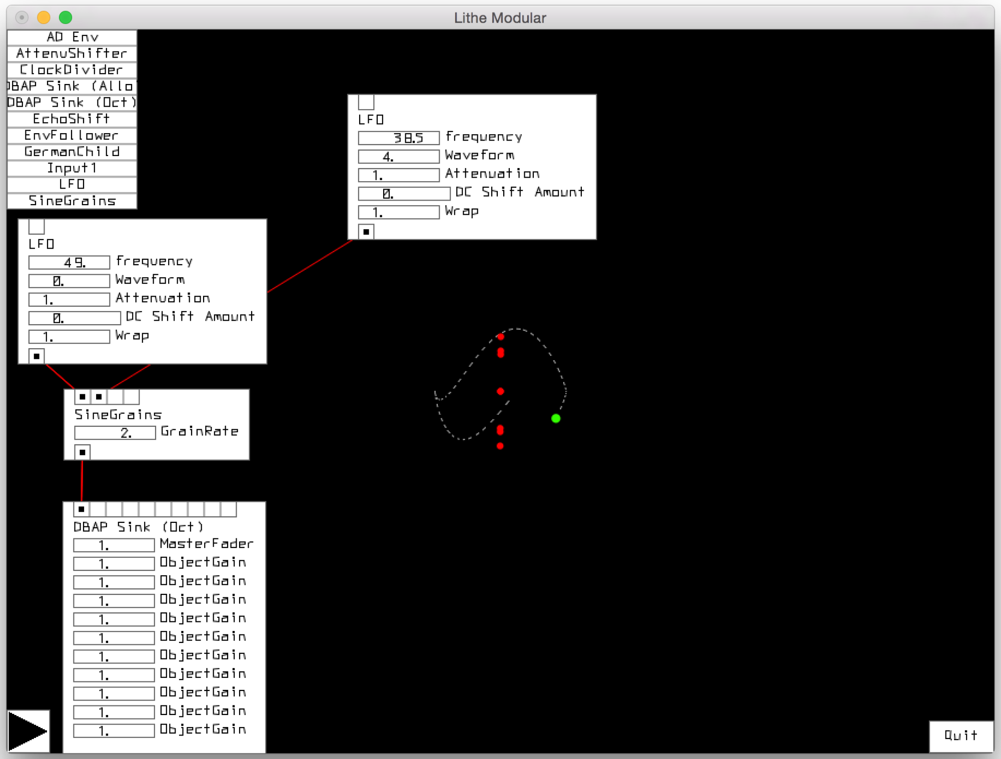


LFO
3. Frequency
0. Waveform
1. Attenuation
0. DC Shift Amount
1. Wrap

SineGrains
2. GrainRate

DBAP Sink (Oct)
1. MasterFader
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain

FIGURE 4.3: An sawtooth LFO that modulates the u of the SineGrain module to produce circular motion with a rotational speed of 3Hz



LFO
38.5 frequency
4. Waveform
1. Attenuation
0. DC Shift Amount
1. Wrap

LFO
49. frequency
0. Waveform
1. Attenuation
0. DC Shift Amount
1. Wrap

SineGrains
2. GrainRate

DBAP Sink (Oct)
1. MasterFader
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain
1. ObjectGain

FIGURE 4.4: Side view of the speaker layout with two LFOs modulating the SineGrain module. The first one modulates the u with a sawtooth wave to produce circular motion. The second one modulates the v with a sine wave to move the object up and down the surface of the sphere.

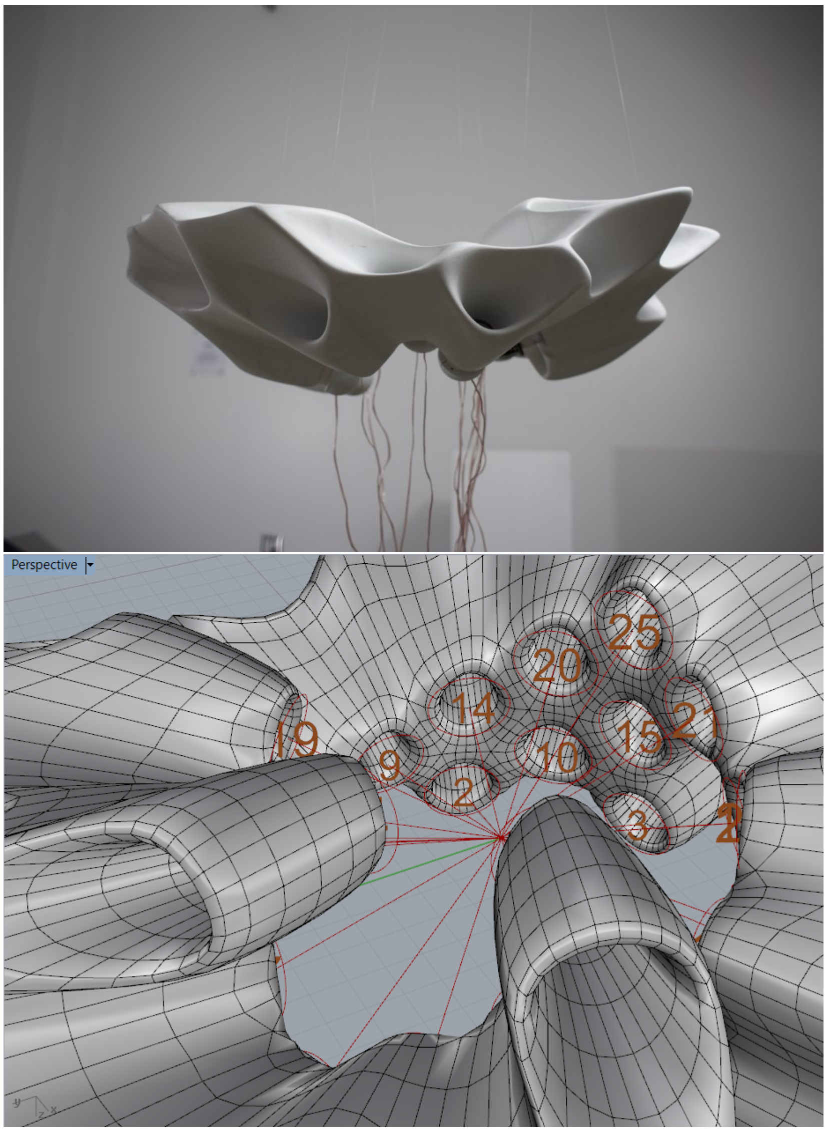


Figure 4.5: HIVE is a sound sculpture with cavities that contain speaker drivers on the inner end. 16 channels of audio with speakers pointed outward from the center. Each speaker points at a different elevation and direction. The numbers in bottom figure show the positions of the speakers

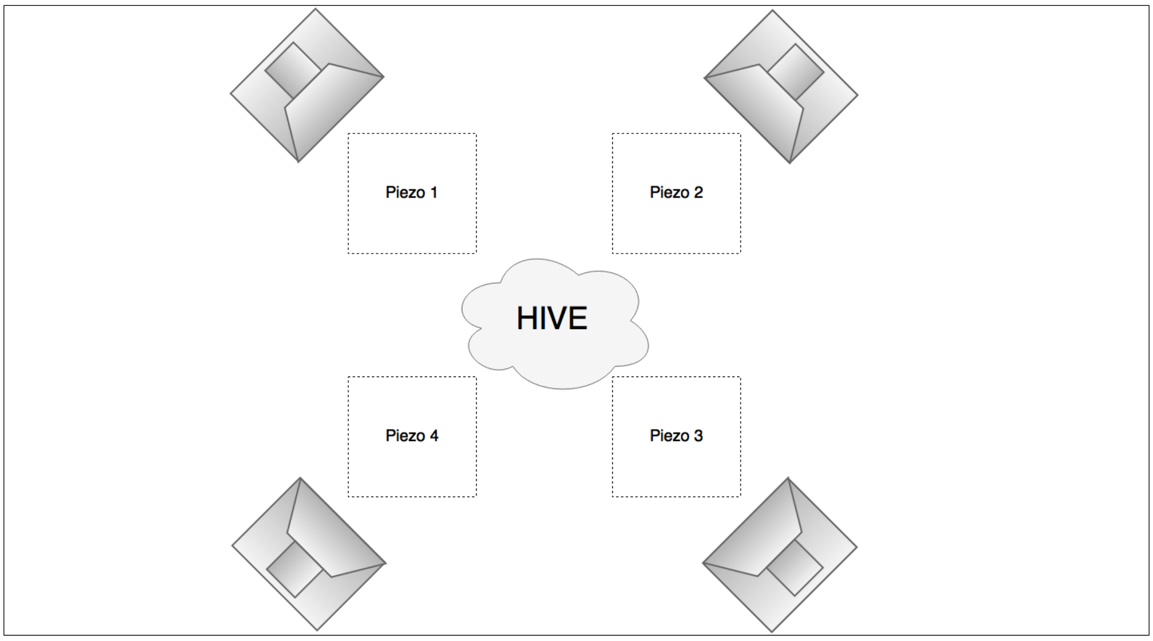


Figure 4.6: The installation at SBCAST Studio F. The HIVE sculpture was placed in the center and four external speakers placed in a quadraphonic setup. The sculpture's 16 speakers projected outward in all directions, and the quadraphonic setup projected inward toward the center. As the audience walks around in the space, they affect the piezo sensors placed in the carpets which triggered sonic and spatial events.

Chapter 5

Implications and Future Work

5.1 Implications with regards to compositional domains

Due to the position signals being abstracted and being bounded in range, this opens up a wide range of primarily timbre oriented effects and techniques to be reinterpreted into the spatial domain. discussed in Appendix A. Effects used in timbre composition like gates, compressors, oscillators, filters, etc. can be reinterpreted in the spatial domain adding a sense of familiarity while still being open for experimentation.

In hardware modular synthesizers, a notion that "everything is a voltage" exists – parameters are all treated in the same manner and can be modulated by any other signal. This makes it possible to obtain an infinite number of interconnections and combinations between modules. This project has a similar notion since all position signals are of the same bounded range (in this implementation, [-1, 1]), infinite cross manipulations between position signals are possible.

Finally, it also opens up the potential to re-interpret known compositional ideas into the spatial domain. For example, Clarence Barlow's Harmonicity formula [38] primarily used for tuning, temperament, and note-selection can be used to inform the frequencies of oscillators of multiple sound objects to explore notions spatial harmony and disharmony[1]. Further, approaches like per-grain effects in granular synthesis could be spatially informed using scalar fields defined on manifolds. Newer implementations could port the Immlib library [21] to facilitate this kind of parameter automation.

[1]Although, since one would probably prefer lower frequency oscillators for trajectories, a more appropriate term might be rhythm. However, certain techniques like SMS [19] make use of audio-rate trajectories.

5.2 Implications for the development of new software

As discussed in Chapter 1 many tools, while providing a platform for creating object-based content, don't completely unify sound processing and trajectory generation within the same workflow. Having a common processing architecture potentially opens up a space for object-based plug in standards similar to the existing standards for audio plug-ins like VST, AAX, Audio Units, and JSFX. This allows for a user to extend their own software's functionality without drastically modifying to the software itself.

In addition to new plug-in formats, this workflow also points toward the refinement of the development efforts toward object-based audio formats like SpatDIF [39]. Since all position information is interpreted as signals in the Lithe audio graph, an audio object signal can, in its simplest form, be stored as a 4 channel wave file. One possible use case would be to allow for the storage of atlases and d-maps as a part of the file allowing for better portability, while still leaving the possibility to edit and modify audio scenes. Having a generic container format also paves the way for Digital Audio Workstations to be developed for inherently object based workflows. Further, since this uses the concept of a unit generator, such DAWS would inherently be compatible with non-object based content as well.

Finally, as a future work is to extend this audio graph to be able to process an object group. This is when UGens are able to process complete sets of object in similar manner. This can be seen as an extension of multi-channel processing to be able to control large groups of semantically related sound sources. Since each sound source has its individual position information, any UGen that applies effects based on position will produce heterogeneous results from a monolithic interface.

5.3 New areas for exploration

5.3.1 Handling spatial distortions

One of the drawbacks of using an abstracted coordinate system like the (u,v) coordinates of a manifold, is that spatial distortions that result of the mapping function aren't apparent while working within the (u, v) coordinate space. With the example of a globe, the latitude circles have a smaller radius as we get closer to the poles. In other words, the apparent dynamic range that u is mapped to changes depending on the v value. This is why most Mercator projections of the globe distort the size of continents the closer they are to the poles.

This may be desired in some cases, manifolds like spheres and toruses are fairly intuitive to navigate despite these distortions. In other cases, with more irregular manifolds, it may be pertinent to investigate the representation as well as workflows that better characterize these distortions and how they affect the trajectory. This may be approached along two different vectors: one in the realm of UI design, and the other in re-defining the mapping functions and (u, v) space itself such that the distortions are obvious even in a waveform representation.

A different aspect of this is the creation of UGens that use spatial distortions to create velocity based effects, for example, UGens that modulate the velocity of an object based on the expansion and contraction of space due to the distortions in the manifold. This can be thought of similar to Kepler's second law of planetary motion where planet's velocity of changes such that the rate of change of area of a line connecting one of the foci to the planet is zero.

5.3.2 UI Research

In a time-line view, audio is commonly represented as a waveform, and event based signals like MIDI have been represented using a piano roll view. Since this project suggests the representation of position signals using abstract coordinates, a waveform based approach may not be sufficient. Most tools discussed in Chapter 1 have a live animation view which is not ideal for viewing a time-line of an audio object. Some tools like Zirkonium MK3 [12] have a time-line view for events representing trajectories but these are limited to top-view of trajectories and are not extensible to irregular speaker layouts.

Typically the primary functions while using a time-line view is to be able to chop, edit, fade, and stitch audio clips. There is an avenue for investigating UIs that best represent trajectories in a time-line view to allow for these kinds of manipulations on the trajectories. Ideas from UIs of popular 3D-animation software could be considered as a starting point for these investigations.

5.3.3 Creation of manifolds

Currently this project allows for manifolds to be defined mainly using parametrically defined shapes. Since an atlas is a collection of mapping functions that together cover an entire manifold, a non-parametrically defined manifold can easily be segmented into multiple mapping functions in order to fully define it. Extending this idea further, mapping functions can extracted from spatially sampling 3-D models thereby allowing for any[2] irregular shape to be used as a source for manifolds.

While the use of very irregular manifolds in the context of spatial composition requires some study, this nevertheless provides a vehicle for composing spatial pieces based on movements from architecturally derived forms modeled in software. The author's collaboration on the project HIVE mentioned in 4.4 was conceived with this in mind. The object itself was parametrically modeled in software and then physically realized using 3-d printing. Hence, instead of using simple manifolds like spheres and toruses, one may be able to obtain manifolds that better conform to the shape of the object to potentially achieve a tighter sonic-spatial relationship.

[2]With reasonable mathematical limitations.

Chapter 6

Conclusion

While there are several tools to create content using object based audio, the workflows vary vastly between them. This puts limitations on the portability of compositions, the types of speaker layouts that can be used, and the scope of interfaces to create trajectories for sound sources. The project presented here builds on the ubiquitous unit generator concept as a way to structure spatial compositions. Further by abstracting the representation of position as a point on a 2-dimensional manifold, it suggests a generic workflow for spatial trajectory processing that is compatible with both regular and irregular speaker layouts, while being independent of the rendering methods used.

These design considerations were then used to build the C++ software library *Lithe* that allows for the construction of object-based audio graphs. This library was used in a proof-of-concept modular synthesis application *Lithe Modular* to demonstrate the potential uses of this workflow. Several modules were developed and shown in the context of manipulating spatial trajectories, noteworthy are two hybrid delay modules (mentioned in Appendix A) that have the ability to manipulate spatial and temporal aspects of a sound source with relation to each other. *Lithe* was also used in the sound installation HIVE – a sculptural sound installation – as the core sound generation system, showing the use of the same workflow irregular and non-conventional speaker setups.

Finally, a discussion on the potential future prospects of an object-based audio graph processing architecture for spatial audio were discussed, highlighting areas of potential exploration in terms of UI design, software design, as well as composition.

Appendix A

Lithe Modular Modules

This chapter discusses some noteworthy modules that were built as a part of Lithe Modular. Obviously, the possibilities in the design space for object-based modules are endless, but hopefully these would go to show the potential usefulness of an object-based audio processing architecture.

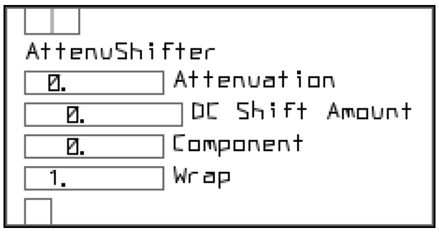
A.1 AttenuShifter

The AttenuShifter module is a multi-component attenuator, inverter, DC shifter. It takes a single audio object and outputs a single audio object. It has four parameters: Attenuation, DC Shift Amount, Component, and Wrap. The Component is used to select which component of the coming audio object signal should be affected. It currently ranges from 0-3, representing the four components of the audio object signal[1]. The Attenuation attenuates the selected component by the specified amount. It also allows for negative attenuations (i.e., functioning as an inverter)[2].

The DC Shift shifts the attenuated signal by the specified amount and allows for full range positive and negative shifting. Since this has the potential to shift the signal out of range, the Wrap specifies how to handle this. Wrap is a boolean that takes a value of either 0 or 1. If set to 0, it hard clips any sample value that goes out of range to the corresponding range maximum or minimum. If set to 1, it wraps any value that goes out of range to the opposite side of the range. For example, if the range was [-1, 1], then a value of 1.4 would be wrapped to -0.6.

[1]From a UI design standpoint, this is a less than ideal way to specify the component. However given that this is a proof of concept and a prototype, it was sufficient.

[2]In many Eurorack hardware modular systems, this type of attenuator-inverter functionality is commonly called an *Attenuverter*



AttenuShifter
0. Attenuation
0. DC Shift Amount
0. Component
1. Wrap

Figure A.1: A prototype UI of the AttenuShifter module. This module is capable of attenuating, inverting, DC shifting. It allows for 'voltage' control of attenuation from the second input allowing it to also function as a VCA.

The second input of the AttenuShifter can be used for 'voltage' control of the attenuation amount. It takes the audio component of any incoming audio object connected to this input and uses that to set the attenuation value. The UI setting for attenuation is ignored and overridden if anything is connected to this inlet. Thus, this allows the AttenuShifter to also behave as a Voltage Controlled Amplifier (VCA).

A.2 LFO

The Low-frequency Oscillator (LFO) is not unlike commonly available low-frequency oscillators and is able to produce sine, saw, triangle, and square waves at frequencies ranging from 0-150 Hz. However, it has the additional ability to attenuate and DC shift, and wrap its output. These additional functions allow for spatial manipulations when used to modulate the u, v, and d parameters of another audio object.

If for example a spherical atlas were used, modulation of the u would result in motions that are along the circumference of the azimuthal circle. A sawtooth wave would result in continuous circular motions; a triangle wave would result in back and forth oscillations along the circumference, and a sine wave would be similar to a triangle wave except that it slows down toward the extremities of the oscillation. Attenuating any of these waveforms would reduce the span of movement from the full circumference, to a small section of it. DC shifting it would center this attenuated movement to other locations on the circumference.

A.3 EchoShift

The EchoShift module is an echo module with multiple cascaded delay lines. It takes a single audio object input and produces multiple echoes of the original. Figure A.3 shows the prototype UI of the module with four outputs. The first output is the original

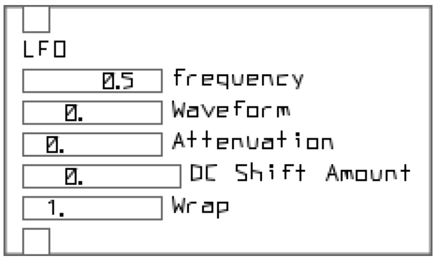


FIGURE A.2: A prototype UI of the LFO module. It has added attenuation and DC shifting functionality to allow it to be used for trajectory modulation.

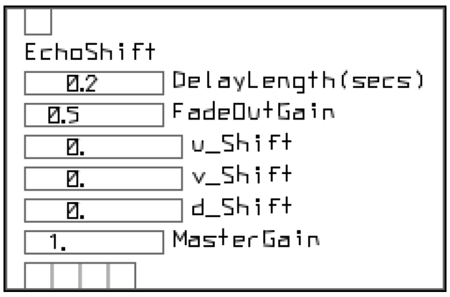


FIGURE A.3: Prototype UI of the Echoshift module with four outputs. It takes a single audio object as an input and produces four successive echoes of specified delay length. The u, v, and d, are also shifted successively from the first output to the fourth creating a cascaded spatial arrangement of the echoes.

object passed through without any modification. The second output's audio component is delayed according to the value specified in *DelayLength*. Further, the u, v, and d, components are DC shifted by their specified values. The third output is a similarly delayed and shifted version of the second, and so on. Additionally, each successive echo can be attenuated from the previous using the fadeout gain, and the level all the outputs can be simultaneously set using the master gain. Figure A.4 shows the signal flow diagram of the cascaded delay line used.

Any trajectories and movement in the incoming audio object will cause the echoes to move in a similar, but shifted fashion. Using this, is possible to spread the sources out spatially, with a short delay length to create a perceptually wide sound source. It is also possible to increase the time delay in order to perceive each echo as a discrete event to create a call and response sort of arrangement.

A.4 EchoDelay

The EchoDelay module is similar to the EchoShift module presented in A.3 except that instead of shifting the u, v, and d signals, they are delayed in a similar cascaded manner

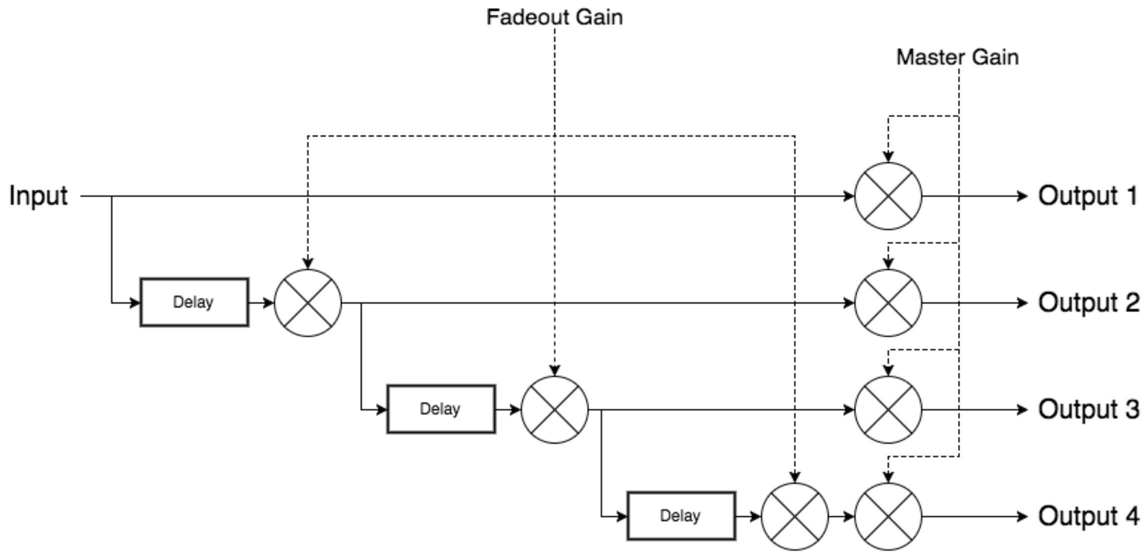
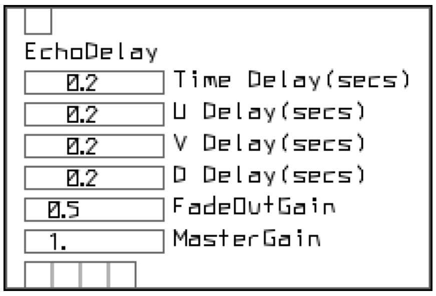


FIGURE A.4: Signal flow diagram representing the cascaded delay lines used. Note that this figure represents the signal flow of only the audio within audio object signal.



```
EchoDelay
0.2   Time Delay(secs)
0.2   U Delay(secs)
0.2   V Delay(secs)
0.2   D Delay(secs)
0.5   FadeOutGain
1.    MasterGain
```

FIGURE A.5: A prototype UI of the EchoDelay modules with four delays, one for each, audio, u, v, and d. It produces four outputs with each successive output delayed from the previous output by the specified amount on each component.

as the audio. This allows for large spatio-temporal manipulation of a moving audio object within a simple parameter space.

For example, with smaller but similar delays on the u, v, d signals, it is easy to get a trail of echoes behind the movement of the input object. WIth larger delays relative to the velocity of motion, the locations of the echoed audio objects start to spread out more and more in space. Manipulating the audio delay in either case can shift the perception from spatialized comb filtering effects, large wide sources, room effects like slap-back, to wide bundles of echoes and repetitions. With dissimilar delays on the u, v, and d signals, the spread of sources can be even less correlated to the motion of the input object.

It is also worth noting that the output objects can be further processed both sonically and spatially by other modules in the audio graph increasing the possibilities to orchestrate the resultant spatial textures.

Bibliography

[1] John M Chowning. The simulation of moving sound sources. *Journal of the Audio Engineering Society*, 19(1):2–6, 1971.

[2] Tomlinson Holman. *Surround sound: up and running.* CRC Press, 2014.

[3] Trond Lossius, Pascal Baltazar, and Théo de la Hogue. Dbap - distance-based amplitude panning. In *ICMC*, 2009.

[4] Ville Pulkki. Virtual sound source positioning using vector base amplitude panning. *Journal of the Audio Engineering Society*, 45(6):456–466, 1997.

[5] David G Malham and Anthony Myatt. 3-d sound spatialization using ambisonic techniques. *Computer music journal*, 19(4):58–70, 1995.

[6] Augustinus J Berkhout, Diemer de Vries, and Peter Vogel. Acoustic control by wave field synthesis. *The Journal of the Acoustical Society of America*, 93(5):2764–2778, 1993.

[7] Dolby Laboraories Inc., . URL https://www.dolby.com/us/en/brands/dolby-atmos.html.

[8] DTS Inc., . URL http://dts.com/dtsx.

[9] Auro Technologies N.V. URL http://www.auro-3d.com/system/concept/.

[10] Thibaut Carpentier, Markus Noisternig, and Olivier Warusfel. Twenty years of ircam spat: looking back, looking forward. In *International Computer Music Conference*, pages 270–277. The International Computer Music Association, 2015.

[11] Jens Ahrens, Matthias Geier, and Sascha Spors. The soundscape renderer: A unified spatial audio reproduction framework for arbitrary rendering methods. In *Audio Engineering Society Convention 124*. Audio Engineering Society, 2008.

[12] Chikashi Miyama, Götz Dipper, and Ludger Brümmer. Zirkonium mk iii-a toolkit for spatial composition. *Journal of the Japanese Society for Sonic Arts*, 7(3):54–59.

[13] Rui Penha and J Oliveira. Spatium, tools for sound spatialization. In *Proceedings of the Sound and Music Computing Conference*, 2013.

[14] Thibaut Carpentier. Panoramix: 3d mixing and post-production workstation. In *Internaltional Computer Music Conference*. The International Computer Music Association, 2016.

[15] Muhammad Hafiz Wan Rosli and Andres Cabrera. Angkasa: A software tool for spatiotemporal granulation. In *International Symposium on Computer Music Multidisciplinary Research, University of São Paulo, Brazil*, 2016.

[16] Jean Bresson. Spatial structures programming for music. In *Spatial Computing Workshop (SCW)*, pages 1–1, 2012.

[17] Scott Wilson. Spatial swarm granulation. In *International Computer Music Conference*. The International Computer Music Association, 2008.

[18] Eric Lyon. The future of spatial computer music. 2014.

[19] Ryan McGee. Spatial modulation synthesis. In *International Computer Music Conference*. The International Computer Music Association, 2015.

[20] Muhammad Hafiz Wan Rosli and Curtis Roads. Spatiotemporal granulation. In *International Computer Music Conference*. The International Computer Music Association, 2016.

[21] Miguel Cerdeira Negrao. Immlib-a new library for immersive spatial composition. In *Proceedings of the Sound and Music Computing Conference*, 2014.

[22] Max V Mathews, Joan E Miller, F Richard Moore, John R Pierce, and Jean-Claude Risset. *The technology of computer music*, volume 9. MIT press Cambridge, 1969.

[23] Julius O Smith III. Viewpoints on the history of digital synthesis. 1999.

[24] C. Roads and Max Mathews. Interview with max mathews. *Computer Music Journal*, 4(4):15–22, 1980. ISSN 01489267, 15315169. URL http://www.jstor.org/stable/3679463.

[25] Richard Charles Boulanger. *The Csound book: perspectives in software synthesis, sound design, signal processing, and programming*. MIT press, 2000.

[26] James McCartney. Supercollider, a new real time synthesis language. In *International Computer Music Conference*, pages 257–258. The International Computer Music Association, 1996.

[27] Ge Wang and Perry Cook. Chuck: a programming language for on-the-fly, real-time audio synthesis and multimedia. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 812–815. ACM, 2004.

[28] Miller Puckette. Combining event and signal processing in the max graphical programming environment. *Computer music journal*, 15(3):68–77, 1991.

[29] Miller S Puckette. Pure data: another integrated computer music environment. 1997.

[30] Jonty Harrison. Sound, space, sculpture: some thoughts on the what,howand whyof sound diffusion. *Organised Sound*, 3(02):117–127, 1998.

[31] Zachary Seldess. Miap: Manifold-interface amplitude panning in max/msp and pure data. In *Audio Engineering Society Convention 137*. Audio Engineering Society, 2014.

[32] Enda Bates. Spacemaps, manifolds and a new interface paradigm for spatial music performance.

[33] Miguel Negrao. *Parameter Field Spatialization: The development of a technique and software library for immersive spatial audio*. PhD thesis, Queens University Belfast, February 2016.

[34] Timothy Place, Trond Lossius, and Nils Peters. The jamoma audio graph layer. In *Proceedings of the 13th International Conference on Digital Audio Effects*, pages 69–76, 2010.

[35] AlloSystem. URL `https://github.com/AlloSphere-Research-Group/AlloSystem`.

[36] Graphics Library of Views (GLV). URL `https://github.com/AlloSphere-Research-Group/AlloSystem`.

[37] Curtis Roads Andres Cabrera, JoAnn Kuchera-Morin. The evolution of spatial audio in the allosphere. *Computer Music Journal*, 2017 (forthcoming).

[38] Clarence Barlow. *On Musiquantics*. University of Mainz, 2012.

[39] Trond Lossius. Controlling spatial sound within an installation art context. In *Proceedings of the International Computer Music Conference*, 2008.