Collaboration and Embodiment in Networked Music Interfaces for Live Performance

Chad McKinney

Submitted for the degree of D.Phil.

University of Sussex

October, 2016

Acknowledgements

I am incredibly indebted to the tremendous support and generosity of my wife Jessica, and our two amazing children, Lorelei and Tristan. They have followed me across the world and helped me follow my dreams. They believed in me when I didn't believe in myself, and kept me grounded when I became lost.

I am also incredibly grateful to have had two of the most inspiring, knowledgeable, and supportive supervisors, Nick Collins and Martin Berger. Nick began me on this journey and never led me astray. Witnessing his daring work and hearing his sharp wit are two things I'll dearly miss about this time. As well, Martin has been been an incredibly consistent and energizing figure. He provoked interesting and new ideas and helped build my confidence when I sorely lacked it.

The execution of this research could not have been possible without my collaborators and friends, Cole Ingraham, Curtis McKinney, and Benjamin O'Brien. They suffered through my mad whimsy and endured experimentations with software version of highly dubious stability. Their creativity and insight made this research much better than it could have been otherwise.

I would like to especially thank Curtis, who started down a path that I've only been following. He is my most ardent supporter and my harshest critic; someone who never held back, but never let me quit. That said, we can all agree that I came up with the better song titles.

I was lucky enough to meet and befriend some tremendously great people during my time at Sussex. Thanks to Thor Magnusson, Chris Kiefer, Anna Jordanous, Alice Eldridge, Gareth White, and Pejman Mirza-Babaei for making those years so exciting.

I greatly appreciate the contributions of time and insight from the various participants of the studies in this thesis. They literally helped define my research and expanded my ideas about music and technology, for which I am incredibly grateful.

Finally, I would like to thank all my friends and family who have shown so much support over the years. They never stopped believing in me, and helped me strive to achieve my greatest goals.

Collaboration and Embodiment in Networked Music Interfaces for Live Performance

Chad McKinney

Summary

Research regarding liveness and embodiment in electronic music has tended to explore the relationship of bodies and instruments, audience perception, interfaces, and shifting definitions, less theoretical and empirical study has considered network situations, perhaps given their relative cultural novelty. Network music has seen many advances since the time of the Telharmonium, including the invention of the personal computer and the widespread proliferation of internet connectivity. These advances have fostered a unique approach to live electronic music that facilitates collaboration in a field where solo performance is perhaps more common. This thesis explores the design of network music interfaces, and how those interfaces mediate collaborations.

Three new network music system interfaces, each using different a different paradigm for interface design are presented in this study. One an instrument for creating modular feedback lattices. Another is a three dimensional virtual pattern sequencer. And the last is a web based collaborative live coding language. Accompanying each system is an evaluation using quantitative and qualitative analysis to frame these instruments in a larger context regarding network music. The results highlight important themes concerning the design of networked interfaces, and the attitudes of musicians regarding networked collaborations.

Submitted for the degree of D.Phil.

University of Sussex

October, 2016

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Liveness, Embodiment and Networked Performance

Luciano Berio, in a 1983 interview with Roassan Damlonte, said of electronic music:

> "With or without new tools and technologies, electronic music as a means for musical thinking reached a dead end. Moreover, the new tools detached it even further from the global and comprehensive idée of music making which is perceived not only by its technical, historical, and expressive terms, but in contemporary and social terms as well." (Berio & Dalmonte, 2007)

In contrast, writers such as Philip Auslander and Marc Leman have proposed that outdated notions of liveness and embodiment are inadequate in the age of mass media, the internet, and mobile technology (Auslander, 2008; Leman, 2008). Social media and the web have transformed not just the format of an act, but also the fundamental audience. Sites like Instagram, YouTube and Sound Cloud have become common as 21st century mediums created by increasingly fragmented groups of authors for increasingly fragmented audiences. Some researchers have argued that newer audiences who have been raised watching DJs in clubs, playing video games, and who have never known a time before the Internet, don't feel the same need for a one to one connection between effort and output (d'Escriván, 2006; Dean, 2003). An example supporting this is video game streaming and eSports, which have grown tremendously in popularity in recent years. In 2014 Twitch.tv, a popular game streaming site, accounted for 43 percent of all live video streaming by volume, with a reported 55 million monthly viewers (Delaney & Madigan, 2015). In 2015 the League of Legends world championship finals had a world record 36 millions viewers (Riot Games, 2015). The electronic music industry is also flourishing with the International Music Summit estimating that the industry is worth 7.1 billion dollars in their 2016 report, a 60 percent increase since 2013 (Watson, 2016). This growth is fueled in part by a steady increase in the popularity of live electronic music events and festivals.

With these developing trends in mind, network music is considered as a unique case in live electronic music. Electronic music is a popular genre, but networked techniques and approaches have largely been ignored by the mainstream practitioners. Academic research regarding the field is also less popular than many other topics. Whilst research regarding liveness in electronic music has tended to explore the relationship of bodies and instruments, audience perception, interfaces, and shifting definitions, less theoretical and empirical study has considered network situations, perhaps given their relative cultural novelty. Network music has seen many advances since the time of the Telharmonium, including the invention of the personal computer and the widespread proliferation of internet connectivity (Rohrhuber, 2007a). These advances have fostered a unique approach to live electronic music that facilitates collaboration in a field where solo performance is perhaps more common. Furthermore, the interdependencies, presentations, locations, and structures of these groups introduce new variables with regards to the perception of liveness and embodiment for both performers and audiences (Föllmer, 2005; Tanaka, 2006).

It is difficult to separate the study of live network music performance from the study of interfaces. Often these performances employ novel technology to facilitate some form of relationships between the performers. This thesis explores the relationship of these networked musicians with their interfaces. It also probes how those interfaces can mediate or encumber the dynamic interactions that the network music often is attempting to facilitate.

1.2 Research Questions and Themes

The overarching research theme is to investigate the experience of performing music in networked environments and how to enhance that experience. The key questions of the thesis are:

1. How do distribution, virtual spaces, communication, and autonomy impact a network music interface and collaboration?
2. What characteristics define a successful live networked music interface and collaboration?
3. Do computer musicians consider interfaces with virtualized environments to embody those characteristics?
4. Do musicians have a preference regarding co-location and distribution in collaborations?

These questions will be explored through the design and evaluation of three new networked music interfaces, with final conclusions in Chapter 7.

1.3 Contributions

The contributions of this thesis are as follows:

- Three new network music system interfaces, each using a different paradigm for interface design; one is a 2D modular feedback network instrument, another is a 3D pattern sequencer and the third is a web based live coding language. All the code is open source and freely available (McKinney, 2012, 2013b, 2014a). *Lich.js* can be used by going to `chadmckinneyaudio.com/lich`. Binaries for *Yig, the Father of Serpents* and *Shoggoth* have been made available at `chadmckinneyaudio.com/YigAndShoggoth.zip`.
- Evaluations of the new interfaces, as well as a novel study regarding co-location and distribution in network music collaborations
- A collection and evaluation of opinions and thoughts of established network music practitioners on the issues regarding the medium historically, and to this day.

1.4 Relevance

Since the last four years when the initial research for this thesis began, music technology, especially web and mobile based music technology, has expanded tremendously. New interfaces, systems, instruments, and experiments are directed at wider audiences with a lower barrier to entry. Specific areas of research such as live coding and mobile device based synthesizers have seen rapid activity. Furthermore, social media has asserted itself as an era defining establishment. The 'meme' culture of the 21st century fanatically shares and remixes, dynamically and continuously. While there is active research regarding live networked music performance, the technologies that research has developed have not seen such wide adoption. We need to further understand why performers may be interested in collaborative technologies, but also what complications and issues, technical and aesthetic, arise out of the use of those technologies. By understanding these relationships, instruments and interfaces can be better designed to facilitate collaborations by musicians of varying backgrounds.

1.5 Structure

Chapter 2 examines historic uses of networking technology in musical contexts, and notable works using those technologies. This is followed by Chapter 3 where a survey of veteran network musicians are queried about their ideas regarding liveness in network music. Subsequently Chapters 4, 5, and 6 each propose a new interface for network music that addresses some of the issues raised. Chapter 4 presents *Yig, the Father of Serpents*, a networked interface for creating synthesized feedback lattices in a two dimensional space. Additionally this chapter evaluates *Yig, the Father of Serpents* in a comparative group study where collaborators use *Yig* and Max Neuhaus's *Auracle* in both co-located and distributed networks. Chapter 5 presents *Shoggoth*, which is a pattern sequencer in a three dimensional virtual space, followed by an analysis of reports by two experienced users. Chapter 6 presents *Lich.js*, which is a collaborative live coding language for music and graphics using web technologies. This chapter also contains the analysis of an anonymous survey of *Lich.js*. Finally, in Chapter 7 the combined results from these studies are analyzed and conclusions are presented.

1.6 Related Publications

Some parts of this thesis contain material that has already appeared in conference publications:

Chapter 3 contains material from *Liveness In Network Music Performance*. In Proceedings of the Live Interfaces: Performance, Art, Music Symposium, Leeds, 2012 (McKinney & Collins, 2012a).

The initial development of *Yig, the Father of Serpents*, in Chapter 4, was first described in *Yig, the Father of Serpents: A Real-Time Network Music Performance Environment*. In Proceedings of the Sound and Music Computing conference, Copenhagen, 2012 (McKinney & Collins, 2012b).

The initial development of *Shoggoth*, in Chapter 5, was first described in *An Interactive 3D Network Music Space*. In Proceedings of New Interfaces for Musical Expression, Seoul, 2013 (McKinney & Collins, 2013).

The initial development of *Lich.js*, from Chapter 6, was first described in *Quick Live Coding Collaboration In The Web Browser*. In Proceedings of New Interfaces for Musical Expression, London, 2014 (McKinney, 2014e).

1.7 Related Performances

The three systems presented in this thesis, *Yig*, *Shoggoth*, and *Lich.js* have each seen multiple international (and transcontinental) performances over several years, mainly with the band Glitch Lich (McKinney, McKinney, O'Brien, & Ingraham, 2012).

1.7.1 Yig Performances

- NOISE=NOISE – January 4th, 2012, London, UK
- Network Music Festival – January 28th, 2012, Birmingham, UK
- TEDxSussex – April 27th, 2012, Brighton, UK
- International Computer Music Conference – September 14th, 2012, Ljubljana, Slovenia
- Amersham Arms – October 17th, 2012, London, UK
- University of Florida – October 26th, 2012, Gainesville, Florida, USA
- Pendulum New Music – November 28th, 2012, Boulder, Colorado, USA
- Texas AM – February 5th, 2013, College Station, Texas, USA
- Mills College – March 6th, 2013, Oakland, California, USA
- University of Colorado – April 11th, 2013, Boulder, Colorado USA
- Algorave – April 17th, Brighton UK
- SuperCollider Symposium – May 20th, 2013, Boulder, Colorado, USA
- FaceArt Institute of Music – September 27th, 2013, Shanghai, China
- /* vivo */ – November 26th, 2013, Mexico City, Mexico
- Frost – February 1st, 2014, New York City, New York, USA
- Algorave – April 26th, 2014, Gateshead, UK
- Zoomin' Night – May 5th, 2014, Beijing, China
- Valuing Electronic Music – June 6th, 2014, London UK
- Algorave – June 9th, 2015, Santa Barbara, California, USA

1.7.2 Shoggoth Performances

- Network Music Festival – February 24th, 2013, Birmingham, UK
- University of Colorado – April 11th, 2013, Boulder, Colorado USA
- Slave to the Algorithm – April 13th, 2013, London, UK
- Algorave – April 17th, 2013, Brighton, UK
- New Interfaces for Musical Expression – March 28th, 2013, Daejeon, Korea
- /* vivo */ – November 26th, 2013, Mexico City, Mexico

1.7.3 Lich.js Performances

- live.code.festival – April 19th, 2013, Karlsruhe, Germany
- Algorave – January 5th, 2014, Tokyo, Japan
- Loft 345 – February 20th, 2014, Guangzhou, China
- Soup – April 5th, 2014, Tokyo, Japan
- Algorave – May 10th, 2014, Shanghai, China
- The Shelter – June 5th, 2014, Shanghai, China
- Algorave – July 4th, 2014, Brighton, UK

Chapter 2

Network Music

2.1 Network Music

Computer network music has benefited from over three decades of development, including the experiments of the San Francisco Bay Area network band pioneers, research into streaming and latency issues, and the introduction of the OSC protocol (Wright, 2002). Making an infrastructure suitable for network performance in the face of highly distributed participants and online security roadblocks remains a challenging task, and many approaches have been developed to facilitate this type of music composition and performance. Programmers and musicians have created wildly different technologies and aesthetics, but all of them share the appreciation for the importance of collaboration, communication, and interaction.

Without a clear definition, the use of the term network music can quickly become vague and meaningless. In their book *Networking Foundations* Patrick Ciccarelli and Christina Faulkner define a network as *a system that allows communication to occur between two people or machines* (Ciccarelli & Faulkner, 2004). Interconnectivity may be the essence of network music and any attempt to define it must be formed around that idea. Be that as it may, the concept of interconnectivity is broad and can easily define ensemble performances of any kind, including acoustic music. John Lazarro and John Wawrzynek supply this definition in their paper *A Case for Network Musical Performance*,

> *A Network Musical Performance (NMP) occurs when a group of musicians, located at different physical locations, interact over a network to perform as they would if located in the same room* (Lazzaro & Wawrzynek, 2001).

This definition, while well formed, is too specific to a particular approach and excludes a large portion of the network music performances that actually occur. Instead, following the lead of Alvaro Barbosa, I prefer Jason Freeman's explanation from his lecture opening the workshop on Network Music at ICMC in 2005 (Barbosa, 2006):

> *What I want to say about Networked Music in general is that all music is networked. You can think about an Orchestra as a client-server network, where a conductor is "serving" visual information to the "client" musicians, or a peer-to-peer networking model in an improvising Jazz Combo, where there is no one directing, and the musicians are all interacting, so, any performance context we can think of in some way there is a network connecting the performers (...). Networked Music with capital N and capital M (the kind we are talking about) is about performance situations where traditional aural and visual connections between participants are augmented, mediated or replaced by electronically-controlled connections.*

This definition works well for the general case including the historic examples from the literature discussed in the following sections, but not entirely for the research discussed in this thesis. The three systems discussed later in chapters 4, 5, and 6 were all developed using the band *Glitch Lich* as a test bed for research. Glitch Lich performances can be described using Freeman's definition, although further clarification and specification is possible. The four key features of an interface designed for Glitch Lich are distribution, virtual spaces, communication, and autonomy. Glitch Lich performances are commonly distributed, with members having lived across several continents for most of the band's history. Communication is a key part of Glitch Lich performances; not just amongst the performers, but also displaying that communication to the audience. This extends to even letting the audience join in the public conversation in real–time via twitter. Additionally, Glitch Lich performances rely heavily on experimentation, exploration, and improvisation. Notably, Glitch Lich pieces are commonly concerned with creating virtual spaces for these performances, and using some visual component to convey this virtual space. Electronic connections form the physical network between the players, but a virtual space allows for a virtual network to create connections between the players, who are embodied in this abstracted system. Additionally these virtual spaces are often imbued with a certain amount of autonomy, often via feedback or using algorithmic and procedural techniques. This autonomy is also a fundamental part of the network, treated as just another entity, like the performers themselves. This is where Freeman's definition fails. To describe the network music of Glitch Lich, and the research in the following chapters, a different definition is required. The network music of Glitch Lich is a virtual space which is mapped to and modulated by relationships between virtual entities, where those relationships are defined and modified in real–time.

Following are a survey of compositions, installations, events, and technologies related to network music. It is not exhaustive, but it summarizes the important trends and notable developments in the field. Afterwards a survey of the published theories and taxonomies of network music will be presented with examples and critiqued.

2.2 Origins and Early Network Music

Music has been used to bridge long distances for thousands of years. Horn instruments were very common in ancient times and were used for communicating with herds or between villages (Jones, 2006). As the instruments developed, the complexity of the music increased allowing for rich melodies that, as with the Ranz des Vaches (also known as Kuhreihen), could even be used to call the names of specific cows. Hector Berlioz famously referenced this musical communication in his programmatic work *Symphonie Fantastique*. Berlioz begins the description of the third movement, *Scène aux champs*, with:

> *One evening in the countryside he hears two shepherds in the distance dialoguing with their 'ranz des vaches'; this pastoral duet, the setting, the gentle rustling of the trees in the wind, some causes for hope that he has recently conceived, all conspire to restore to his heart an unaccustomed feeling of calm and to give to his thoughts a happier colouring.* (Berlioz, Malherbe, & Weingartner, 1900)

To achieve this effect in a concert setting he wrote a musical dialogue between an English Horn and an offstage Oboe, attempting to create a sense of distance.

Charles Ives, ever the ambitious composer, imagined a scenario much more grandiose with his unfinished *Universe Symphony*. The epic symphony was to be his largest creation and he worked on it since before 1915 up until his death in 1954. Ives is said to have envisioned a scene where

> *(...) several different orchestras, with huge conclaves of singing men and women, are to be placed about in valleys, on hillsides, and on mountain tops.* (Lambert, 1997)

The score and his notes do not indicate this directly, but the anecdote is attributed to his secretary Christine Loring (Perlis, 1974).

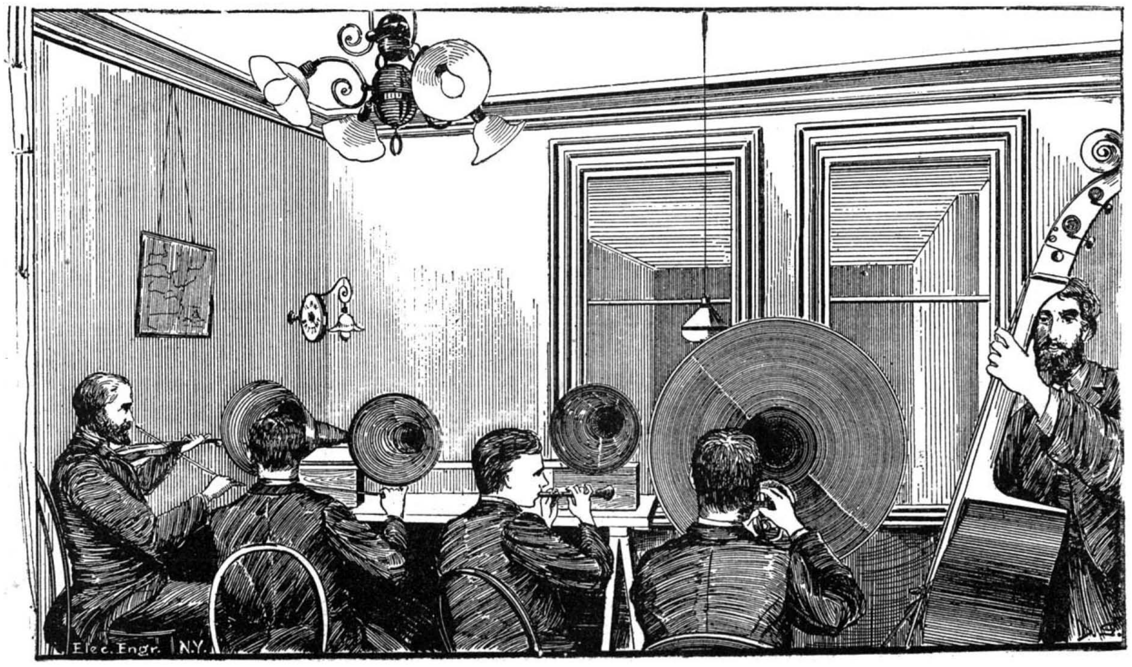


Figure 2.1: An artist's depiction of a music performance over telephone in 1891 (Sci, 1891a).

Until the late 1800s it was not possible to transmit sound without simply making it very loudly. This all changed with the invention of the telephone, of which there is some confusion and controversy about the history. What is known is that Alexander Graham Bell did receive the first patent for the telephone with some of his earliest demonstrations of the technology involving music being sung by participants in other locations (Pasachoff, 1996). As early as 1891 companies like the Long Distance company would pipe music performances over telephone lines for hundreds of miles to concert halls and homes (Connections, 1891b). By 1909 subscription services were available for playing phonograph recordings over the telephone by demand (Anonymous, 1909).

Around this time many experiments with electronic instruments were taking place, such as Elisha Gray's Musical Telegraph and William Du Bois Duddell's Singing Arc (Kirk & Hunt, 1999). It is interesting that one of the very first inventions using electricity to produce musical tones, and by far the most ambitious, was designed specifically for networked performance. The Telharmonium, developed by Thaddeus Cahill throughout the 1890s and early 1900s, was an impressive machine that used an array of tone wheels to produce organ like sounds (Holmes & Holmes, 2002). Thaddeus Cahill was a gifted engineer, but also forward thinking and business oriented. Instead of using the Telharmonium for concerts (although this did happen on occasion) Cahill envisioned piping live music to paying subscribers using the new telephone technology of the era. Thaddeus Cahill was truly ahead of his time considering that this distributed music service predates radio broadcasts and is remarkably similar to technologies that we commonly associate with computers, mobile devices, and the internet. Ultimately the Telharminum proved to be a commercial failure due to a lack of venture capital and the discovery that the machine interfered with telephone calls (Burkart & McCourt, 2006).

While telegraph lines were quickly becoming popularized and incorporated in broader infrastructures, many saw the possibility for another technology to revolutionize communication and subsequently music (Fahie, 2011). The so-called "wireless telephony" , which is now commonly referred to as radio, had the potential to deliver services similar to telephone connections, but without the burden of connecting to users directly. Beginning with the early experiments of Heinrich Rudolf Hertz and Nikola Tesla that showed radio broadcast viability, several ambitious inventors embarked on a race to invent the first functional prototypes for the broadcast and reception of communications via radio waves. Several attempts had been able to attain limited results, but in 1906 it was Dr. Lee De Forest who held the first public radio broadcast of music where he read the news and played phonograph recordings of a wide range (Adams, 2011).

Radio broadcasting technology transformed entire industries and the culture at large as it was assimilated into large scale infrastructures. As with any technology, the proliferation was followed by the experimentation of artists and musicians looking for new and unexpected ways to use it. John Cage's *Imaginary Landscape No. 4* (1951) begins to show some of the characteristics of network music that moves beyond simple broadcast (Nicholls, 2002). 12 transistor radios are each controlled by two performers following a score comprised of a set of parameter charts created using the I Ching. The execution relies on the coordination of the performers sharing radios and the sound of the piece presents a group texture that moves throughout the ensemble.

Fifteen years after writing *Imaginary Landscape No. 4* Cage expanded on this idea in *Variations VII* (1966) by using not only radio, but television and ten telephone connections from around New york including the New York Times press room, a German restaurant, and Merce Cunningham's studio (Cunningham, Kluver, Tudor, Moog, Coker, Kompfner, & Riley, 2008). This expansion beyond just radio station broadcasts opened a new dimension for exploration. Not only was just music or speech being used; instead other spaces themselves were being melded together in a Marshall McLuhan type mass media wash.

Cage was not the only composer who saw the potential that distance and space can play in music. Alvin Lucier composed the work *Quasimodo the Great Lover* (1970) (Nyman, 1999) as an exploration of communication over long distance. Here is the full score:

> *Quasimodo the Great Lover (1970) – for any person who wishes to send sounds over long distances through air, water, ice, metal, stone, or any other sound carrying medium, using the sounds to capture and carry to listeners far away the acoustic characteristics of the environments through which they travel.* (Simon, 1980)

Performances feature a relay of microphone and speaker pairs so that whale like sounds can traverse over long distances creating a sense of space as well as the accrued acoustic characteristic of the nodes in the chain. Lucier has a continuing interest in the role that sound plays as an information medium which is illuminated in his previous work *Vespers* (1968), where performers using Sondols (hand-held echo-location devices) explore a space by scanning the environment with impulses. *Quasimodo, the Great Lover*, while similar to *Vespers* in that electronic sounds are used to excite ambient acoustics, differs because the distance traversed is as important as the acoustic phenomena.

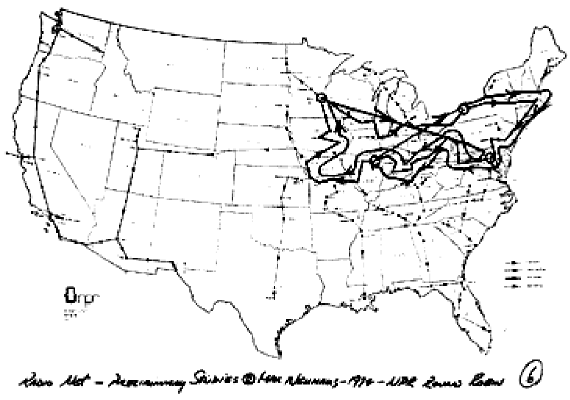


Figure 2.2: Max Neuhaus's illustration demonstrating the signal flow for his work *Radio Net* (Neuhaus, 2004c)

Max Neuhaus's *RadioNet* (1977) is a behemoth sized work that utilized the United States' national public radio (NPR) nation wide circuit to create a two hour long feedback network that spanned hundreds of miles (Neuhaus, 2004c). There were two hundred stations involved in the performance, all of which were connected together through telephone lines. For the performance Neuhaus created five closed sub-loops, all of which went through Washington D.C. Because of the complexity of the system Neuhaus created a self adjusting mixing system that worked by multiplexing multiple inputs. Higher pitched sounds were given more fractions per second of time, corresponding to a higher level in the apparent mix. Listeners were invited to call in and whistle, and in doing so participate in a nation sized effects feed back loop.

As noted in *Indigenous to the Net*, the world's first network computer band, the League of Automatic Music Composers, began as an extension of the home brew circuit tinkering that was characteristic of the San Francisco Bay Area in the mid-1970's (Brown & Bischoff, 2002). Their computers, MOS Technology KIM-1 models, were modest with only 1 kilobyte of memory and which could only be programmed using assembly language. The League of Automatic Music Composers created interactive programs by directly soldering connections between computers and writing programs which would listen and transmit data on these lines. The network was fragile and error prone. It was also difficult to set up as all the connections had to be re-soldered each time the band rehearsed (Brown & Bischoff, 2002). Although beginning with modest machines according to modern standards, the group was able to create rich and complex compositions. Each performer's computer would take an input, use it in some way, and create an output. The analysis of the input signals, and the nature of the output is completely individual to each of the

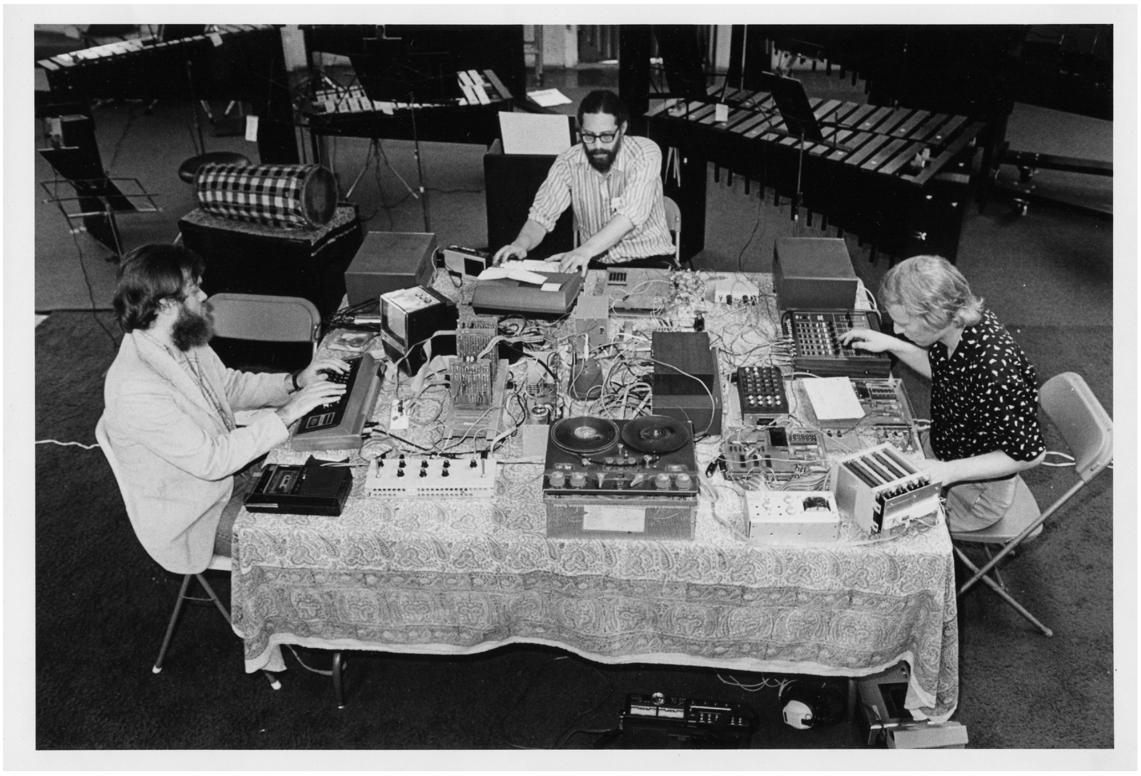


Figure 2.3: The League of Automatic Music Composers (Perkis, Horton, and Bischoff, left to right) performing at Ft. Mason, San Francisco 1981 (Brown & Bischoff, 2002).

members (Bischoff, Gold, & Horton, 1978).

In what can be seen as a natural evolution of the technology, the spiritual successor to the League of Automatic Music Composers, the Hub, utilized a server based system. This system, from which the Hub claims their namesake, provided for a standardized interface for connections between members with varying computer models and shared memory for the ensemble. The hardware for the first hub initially used the same KIM-1 as the League of Automatic Music Composers. The important distinction is the configuration of the ensemble shifted from peer-to-peer to server and client. After the first KIM-1 Hub the group upgraded to a setup that included two computers with expanded power and memory to support the six person ensemble. In an online interview for the Networked Music Review Scot Gresham-Lancaster noted that,

> *From the period of the first HUB gigs in 1985 until 1990 we used two Synertek 6502 based single board computers that talked to each other over RS-232 at 300 baud. We all had a shared scratch pad of memory that we could retrieve and alter numbers from 0-255. A whopping 1 K of memory was all we had.* (Thorington, 2007)

Even though the Hub's first public appearance was a trans-site performance, their work has tended to deemphasize the importance of the distance between the performers. As Chris Brown noted of the Hub "the band itself was always far more interested in the aspects of performer interactivity, algorithmic complexity, and the web of mutual influence that the network provided. The fact that the chamber could be expanded in distance was not entirely irrelevant, but never really the point." After several years of using the twin Synertek configuration, the Hub changed their setup by switching to a Musical Instrument Digital Interface (MIDI) based system. This change allowed for an even more universal interface for the computers as well as simpler network maintenance. This shift was at the cost of a change in the configuration of the network. There was still a kind of server, an Opcode Studio 5 MIDI interface, but the interface only served the role of routing information between users as well as imposed the limitations on data formatting that is characteristic of MIDI. What was lost was the shared memory of the network, and subsequently changed the fundamental way that the hub created its music (Perkis, 1999). In 1998 the group officially disbanded noting a lack of enthusiasm from years of dealing with technical problems and complex setups. This hiatus did not last as they later reformed and continue to give performances around the globe.

2.3 Telepresence

2.3.1 Early Telepresence and Art

Telepresence has interested researchers, businesses, musicians, and artists for decades. There has been tremendous research dating back to the late 1950s on how to create some form of presence at a distance using electronics (Fisher, 1991). Initially these experiments were very simple, such as moving a camera using a head mounted display, the exploration of remote environments in a "virtual safari", or the simulation of driving through Aspen, Colorado using camera stills and a touch screen. Even before these developments artists were experimenting with how presence and authorship can be stretched or redefined using telegraph, telephone and radio technology. Since as early as 1919 artists have embraced the telegraph as a unique medium for communication. The Dadaists Richard Huelsenbeck, Johannes Baader, and George Grosz sent this telegram in reaction to Italian soldier and writer Gabriele D'Annunzio's invaded and anexed Fiume (now called Rijeka):

> *Please phone the Club Dada, Berlin, if the allies protest. Conquest a great Dadaist action, and will employ all means to ensure its recognition. The Dadaist world atlas Dadaco already recognises Fiume as an Italian City.* (Kac, 2005b)

Telegrams continued to be employed by artists well into the century by artists such as Marcel Duchamp, Robert Rauschenberg, and On Kawara.

With the proliferation of telephone connections artists found another medium. The Bauhaus painter and photographer László Moholy-Nagy sought to prove the validity of intellectual motivations in the creation of art by ordering the fashioning of three paintings over a phone call for his *Telephone Pictures* (Galenson, 2009). Moholy-Nagy was greatly influenced by constructivism, as evidenced by his call to a sign factory and not another painter, and his method foreshadows digital techniques by using a grid of pixels and instructing the order similarly to playing chess by correspondence. This approach was echoed decades later in the *Art by Telephone* exhibition in the 1960s which featured works that were created after verbal instruction by artists over the telephone (Kac, 2005b).

Other artists also saw the potential for new a expression with communication technology. In 1966 Max Neuhaus's first network piece, *Public Supply* mixed together incoming phone calls to a radiostation (Neuhaus, 2004b) and in 1971 Richard Teitelbaum transmitted his brain waves by telephone to manipulate jumping beans in his work *Alpha Bean Lima Brain* (Rosenboom, 1976; Rohrhuber, 2007b).

In the 1960s the concept of Mail Art explored the transmission of the artwork itself through the postal service. Ray Johnson along with his New York Correspondence School institutionalized mailed artwork and soon members of the Fluxus movement were experimenting with the genre which persists to this day (Saper, 2001). Continuing in this vein, Fluxus associated artist Paulo Bruscky merged his Xerox performances with mail art when he started creating works using Fax machines in the 1980s (Osthoff, 2005).

Soon televisions would be commonplace in households and satellite could be used to transmit information across the globe to millions of people. Nam June Paik had an ongoing interest in using satellite feeds in his video art. His first work in this vein, *Global Groove* (1973) is a psychedelic collage of video and audio including performances from John Cage, Merce Cunningham, Allen Ginsberg, and Charlotte Moorman lasting thirty minutes and broadcast by WNET-TV (Paik, 1997). His subsequent satellite works *Good Morning Mr Orwell* (1984), *Bye Bye Kipling* (1986), and *Wrap Around the World* (1988) would greatly expand in duration and included live performances from a wide array of musicians, poets, and artists including David Bowie, Laurie Anderson, Oingo Boingo, a Brahms concert, and a Punk Rock show (Harris, 2011; Kac, 2005a).

Connecting together with other people across the world became increasingly easier with the advent of the internet age. Internet users frequented chat rooms, visited websites, and played online games. The technology fosters interactivity instead of just broadcast and in doing so opens up a vast new territory for exploration by musicians and artists. Eduardo Kac created some of the earliest internet works with his installations *Ornitorrinco in Eden* (1989 – 1994) (Kac, 1996) and *Rara Avis* (1996) (Kac, 1997). In *Ornitorrinco in Eden* Kac, with some help from Ed Bennett, created a mobile robot with a mounted camera and interactive controls. Users could call a certain phone number and use their telephone keypad (dial tones were translated into movement commands) to move the robot through the space. The footage taken from the robot's camera was then streamed over the internet for the public to see. *Rara Avis* was similar to *Ornitorrinco in Eden* because it combined internet technology with robotics, but the specifics of the implementation differed. The robot used was fashioned to appear similar to a macaw, where the eyes were actually two cameras, and it had both a microphone and speaker for audio. A "virtual reality headset" was used to allow participants in the venue to see the atrium with stereoscopic vision from the point of view of the bird and allowed for the control of the robots viewing direction. Audio and video from the robot was streamed over the internet, and viewers online could participate by sending audio to the bird's speaker.

2.3.2 Performance Streaming Over the Internet

In 1993, at the University of Southern California, the first experiments with music performance utilizing internet connections streaming audio for collaboration by distance were taking place (Schooler, 1993). Their approach is remarkably similar to the research still happening now in the field, indicating perhaps a fundamental problem that will only mediated by advances in broadband speed and perhaps never fully conquered because of the laws of physics. The performers (playing a Haydn piano trio) and audiences were spread across several sites from California to Washington D.C. User Datagram Protocol (UDP) packets were timestamped and sent over the network for both audio and clock synchronization. Lower rate audio streams were used to save bandwidth and buffering was used to prevent jitter. Despite these measures the delay was still quite noticeable (80 – 350 ms) and the researchers noted that the performance was novel, but contrived. Their suggestions again foreshadow much of the research in the field, stating that latency is not an anomaly and requires acceptance suggesting that new music will need to be written and new approaches developed to work specifically with this medium.

In the titled paper *Network Audio Performance and Installation* Atau Tanaka describes his experiments with networked audio and control data during the years of 1994 through 1999 (Tanaka, 1999). His approach was programmatic, using off the shelf teleconferencing equipment and software. It demonstrated what was possible at the time without access to custom built software or dedicated research connections. The research mainly focuses on bi–directional performance settings where he noted that latency and divergence were an eventuality of the medium and should be accepted, and that the video functioned better at lower resolutions, which facilitated higher frame rates.

Researchers at the University of Geneva conducted similar streaming studies from 1996 through 1999. Their research was more concerned with rehearsal and included an orchestra, a conductor, and video streaming in addition to audio (Konstantas, Orlarey, Gibbs, Carbonel, Moulin, Lyon, & Augustin, 1997). They achieved lower delays (audio 31 ms, video 81 ms) utilizing compression on the streams and higher bandwidth connections, but their distance scale was about a fourth the distance of the USC tests (Konstantas, 1998; Konstantas, Orlarey, Carbonel, & Gibbs, 1999). The analysis of their results concluded that latency was still problematic, but that the technology could provide for some worthwhile opportunities and cost savings.

With the development of the high-performance research networks in the mid-to-late 1990s, new studies were called for that could test the limits of the new connections' capabilities (Paper, Council, Bargar, Church, Systems, Keislar, Fish, Pavo, Microsoft, & Pennycook, 1998). The first Wide Area Network (WAN) study was conducted in 1999 by McGill University which held a swing band concert that was broadcast to an audience at New York University. Their methodology concerned streaming high quality AC-3 audio (encoded Dolby 5.1) and MPEG–1 video and only required single direction transmission, having forgone the performer separation in the earlier noted studies. They used a mixture of Transfer Datagram Protocol (TCP) messages for control information regarding streams and User Datagram Protocol (UDP) for the actual stream (Postel, 1981, 1980). The initial conditions introduced a 20 second buffer to remove jitter and request lost packets which worked without interruption. Upon lowering the buffer to three seconds the audio remained consistent, but the video streamed received several interruptions. The next year the same research group conducted a follow up study similar to the previous, but that streamed 12 channels of uncompressed 24bit/96kHz Pulse-code Modulation (PCM) audio, but without video. The audio was received at the University of Southern California and mixed there to produce the final product (Oliver, Pierce, & Shannon, 1948; Cooperstock & Spackman, 2001).

Soon after the McGill experiments, the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford began developing new techniques and software to take advantage of the burgeoning high speed networks for full duplex bi-directional multi–channel audio streams (Chafe, Wilson, Leistikow, Chisholm, & Scavone, 2000). Their goal was to develop an approach that would remove as many bottlenecks from their system as possible and achieve the closest possible approximations to the latency a musician is accustomed to working with. They did not use compression, which increases latency for the sake of bandwidth, because they had a high bandwidth test connection. They also implemented a multithreading scheme and several parameters for fine tuning the connection, such as adaptable packet size, fragment size, circular buffer size, threshold, and thread priority. Their final tests concluded stereo audio streamed with 75ms round trip time (RTT) across the United States from their site in California to two facilities, one in New York and another in North Carolina.

On June 13, 2002, McGill and Stanford held the first transcontinental jam session utilizing high quality audio and video over research network connections. While previous uses of video and audio were unidirectional, this performance attempted to create a shared space where video projection would bridge the two separated ensembles into a single virtual room. Audio and video did not maintain synchronicity, and the researchers claimed that because the intermodal latency between the performers' sense of sight and vision was less than 30 ms that it was tolerable. They were able to achieve a minimum latency of 50 ms for audio and 80 ms for video during (Woszczyk, Cooperstock, Roston, & Martens, 2005).

The University of Southern California, noting the developments made by Stanford and McGill, developed their own technologies for what they called "Distributive Immersive Performance" (DIP). They referenced the McGill and Stanford collaboration's lack of multimodal synchrony and created their own system which sought to create, initially, unidirectional performances with audio and video synchronicity, over 16 channels of 24bit/48 khz audio, and ambient acoustic emulation of the performance space (Sawchuk, Chew, Zimmermann, Papadopoulos, & Kyriakakis, 2003). Similar work continued subsequently by these groups as well by others. Notable studies include the multi–user investigations at the Technische Universität Braunschweig utilizing client/server architecture, the several detailed studies on performer latency thresholds run by USC, Alexander Carot's work on SoundJack, and CCRMA's development of JackTrip (Gu, Dick, Noyer, & Wolf, 2004; Gu, Dick, Kurtisi, Noyer, & Wolf, 2005; Kurtisi, Gu, & Wolf, 2006; Chew, Sawchuk, Tanoue, &

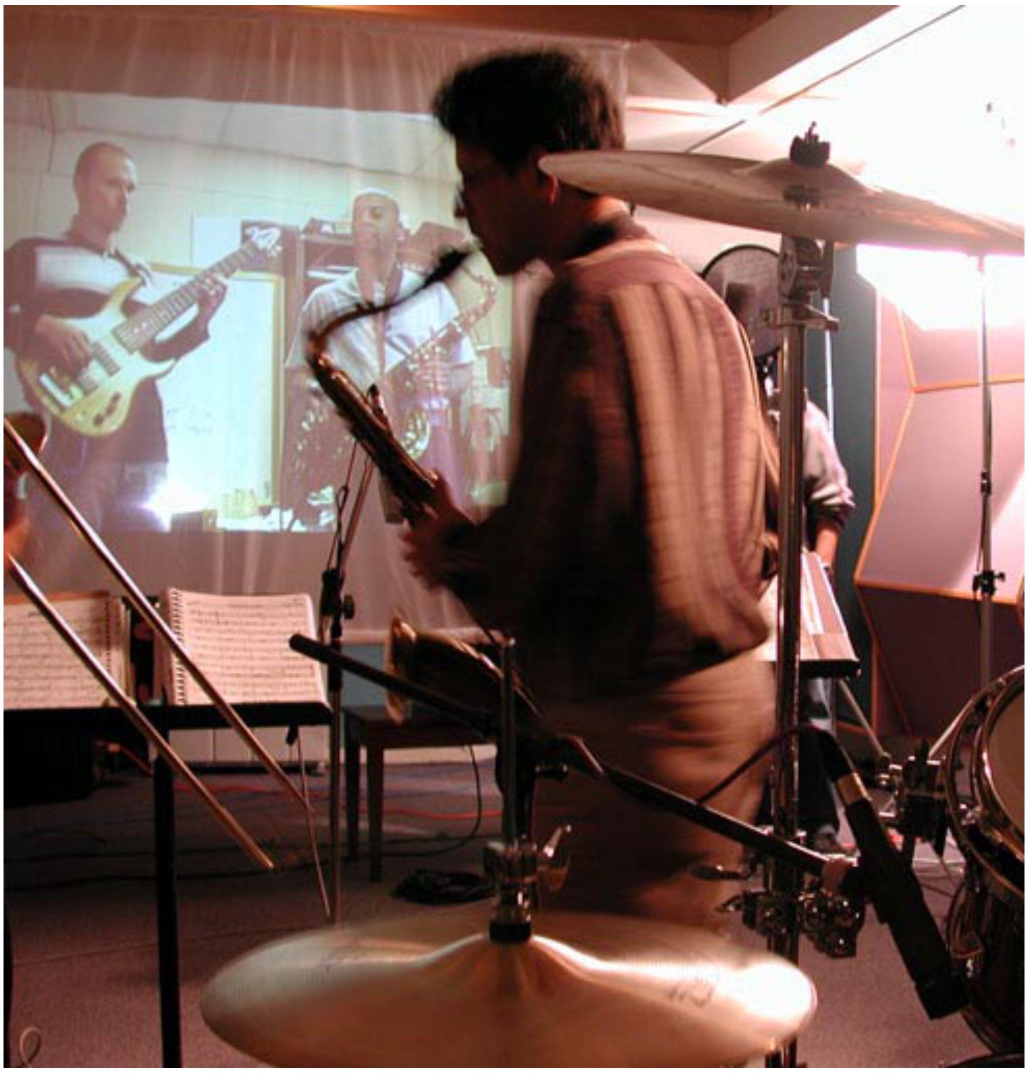


Figure 2.4: Distributed jam session between Stanford and McGill (Carôt, 2004).

Zimmermann, 2005; Zimmermann, Chew, Ay, & Pawar, 2008; Carôt, Krämer, & Schuller, 2006).

Internet speeds consistently increased throught the 1990s and 2000s enabling lower latency and higher quality Voice Over Internet Protocol (VoIP) services such as Skype (Microsoft, 2014) and Google Voice to a broader user base (Networks, 2011). The proliferation of these services makes telepresence performance possible for anyone with a computer and an internet connection. Pauline Oliveros, Scott Gresham-Lancaster, Sam Ashley and others have demonstrated the viability of the medium, although it is inherently imperfect because of the lack of bandwidth, speed, and Quality of Service (QoS) features characteristic of Internet2 and other high performance networks (Gresham-Lancaster, 2007).

2.3.3 Strategies for the Acceptance of Latency and Networking Idiosyncrasies

Much research has been conducted towards investigating the issues that latency presents to instrumentalists when streaming audio as well as strategies to cope with this latency in performances (Barbosa, 2003). Often solutions favor research grade connections between sites, providing lower latency, but also not offering a solution that is widely available to the public (Xu & Cooperstock, 2000; Schroeder, Renaud, Rebelo, & Gualda, 2007). Given the fundamental laws of physics, there are certain distances which produce latencies that no amount of technological advance can overcome. This latency creates a distinct separation amongst the performers, between the performers and the audience, as well as possibly amongst the several audiences. Attempting to perform traditional music given these factors cannot result in a traditional performance. Because of this, other researchers such as Atau Tanaka advocate that the medium has inherit temporal characteristics that should be accepted and explored instead of mitigated (Tanaka, 2006, 2003; Chafe, 2009). Tanaka notes:

> *Transmission delays will be considered a hindrance as long as we try to superpose old musical forms onto the network. Rather, a new musical language can be conceived, respecting notable qualities of the medium, such as packetized transmission and geography independent topology. These features can be said to define the "acoustics" of the network, a sonic space that challenges existing musical notions of event, authorship and time (Tanaka, 2000).*

Several of Tanaka's works have looked towards methods for this acceptance, notably his work with Kasper Toeplitz on *Global String*, a networked sound art installation. The installation comprises of a steel string that is fastened to the ground and stretched across the space towards the ceiling. Physical vibrations are converted into digital information and sent across the internet to another site with another cable. Software synthesis uses the excitations of the cable to activate a locally synthesized digital string emulation with the length of the distance measured (Tanaka & Bongers, 2001).

Researchers at CCRMA devised a similar solution to this problem in 2002 where they used the inherit latencies in a network to synthesize a plucked string (Chafe, Wilson, & Walling, 2002). The Karplus–Strong algorithm is a common one dimensional Waveguide algorithm that produces surprisingly rich results from a simple delay line and filter (Karplus & Strong, 1983). The research used audio feedback in the network with excitations to create harp like sounds that sonified the distances of network nodes, treating the delay as a resource as opposed to a problem. The same group went on to develop techniques for using comb filters in a network to create network based reverberation (Chafe, 2003). Research between the Sonic Arts Research Centre (SARC) at Queen's University Belfast and CCRMA has extended on these experiments utilizing the longer delays between San Francisco and Belfast (approximately 130 milliseconds) to produce interlocking rhythmic music (Renaud & Caceres, 2010). A similar experiment consisted of a performance of Terry Riley's monumental minimalist work *In C* using the same feedback locking technique to synchronize two distributed groups, one at Stanford and other at Peking University in China (Cáeres, Hamilton, Iyer, Chafe, & Wang, 2008).

Jörg Stelkens also approached the issue of latency with an optimistic attitude in his *peerSynth* software. Recognizing that the kinds of connections used by McGill and Stanford would not be available to most musicians, Stelkens created a patching system that allows the users to map network latency as a modulator onto a number of different synthesis parameters (Stelkens, 2003). The resulting music is naturally divergent for the musicians while participation is still codependent and presents a unique performance medium. While most of this type of research looks to incorporate latency as a musical resource there have been some efforts to completely eliminate latency using predictive techniques. Mihir Sakar combined machine listening and predictive modeling with distributed tabla performance to effectively eliminate latency completely. Because tabla music works within a highly structured framework, predicting possible future actions is simpler than in a more open genre, and incorrect decisions will still fall within a narrow range. In his system two performers in different locations played with a locally synthesized tabla that is driven by a predictive model based on what the performer on the other end is currently playing. The result is naturally divergent because the models make occasional mistakes. Reportedly the performers felt as if playing with another musician as if they were in the same room, albeit with some idiosyncrasies (Sarkar, 2007).

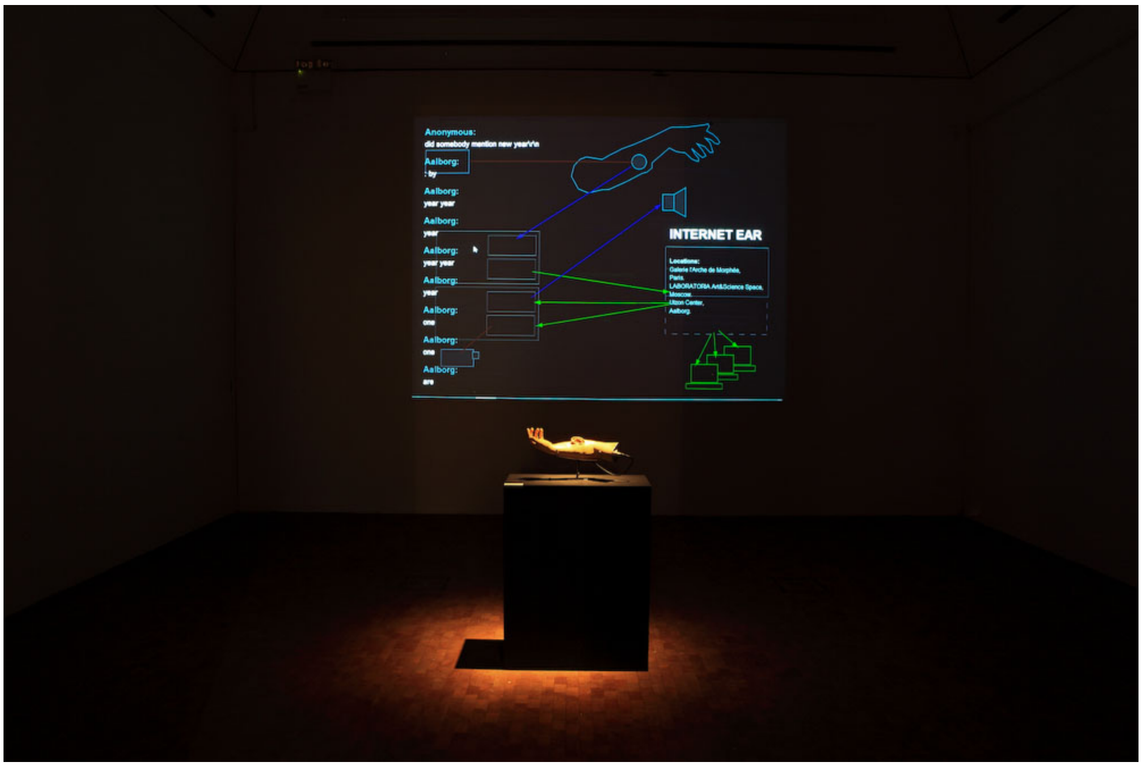


Figure 2.5: One installation of Stelarc's *Internet Ear* (Stelarc, 2011).

Internet Ear is an installation by Stelarc that uses a softcast of his actual ear on arm, with an implanted microphone that is attached to the internet, installed at two locations (Stelarc, 2011). When visitors speak a speech recognition program interprets their speech and attempts to repeat it. The microphone in the ear then sends that audio over the internet to another location, which could be picked up by the speech recognition software there, or also commented on by visitors in the other location. This results in a chaotic wash of voices and feedback, with the distance between the installations having as much of an impact on the resulting sound as the two installation spaces themselves.

2.3.4 Collaborative Composition and Jamming

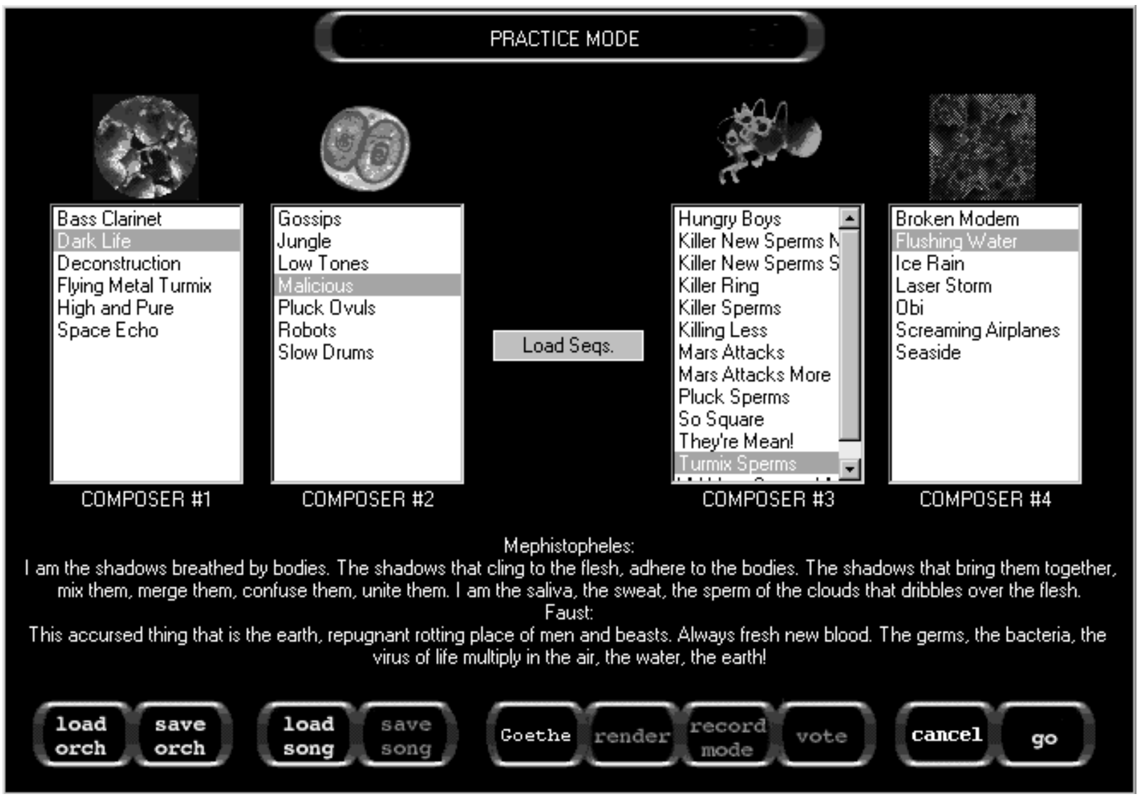
Throughout the 1990s and 2000s many other approaches were being developed to facilitate network music. The previous efforts mentioned focused on the real-time performances of instrumentalists, but many other methodologies explored asynchronous, non-real-time, or virtual instrument collaborations. NetJam, dating back to 1990, is an early example where users could instigate collaborations via an e-mail list to collaboratively edit data files (in the form of MIDI files, Max patches, or PostScript files) on an anonymous File Transfer Protocol (FTP) server (Latta, 1991a, 1991b). A “Read Me” file was placed next to the data file allowing for a running dialogue on the direction of the piece.

In 1994 ResRocket began as a similar project to NetJam, where contributors posted sound files to an FTP server as well as contributed to the continuing conversation on their mailing list (Notes from the NetJam Project, 2004). By 1995 the virtual band claimed 600 members, who named the band ResRocket by voting in a poll of ten randomly generated names. Two of the users developed an online MIDI networking program and offered their service along with a new website for a monthly subscription fee. After investors became involved the project soon collapsed. Faust Music Online, created by Sergi Jordà in 1997, was another program similar to the NetJam’s asynchronous collaborative composition paradigm. Instead of pop or rock songs Faust Music Online was designed to create experimental electronic music for users of varying backgrounds (Jordà, 1999). The interface was mouse driven GUI allowing users to manipulate and store proprietary score files in an online database which allowed for recursive editing with history. Only these score files, and no audio was ever passed between programs and initially it did not allow for real–time interactions between performers (Jordà, 2002).

Towards the end of the 1990s these type of collaborative online programs became more common. Rabiscas and Cordas (2002) were java programs that allowed for real–time MIDI performances utilizing a client server architecture, while Georg Hajdu’s Quintet.net used Max/MSP to achieve similar results (Fernando Lindner Ramos & Manzolli, 2003; Hajdu, 2004). Andrew Brown’s jam2jam software takes influence from previous work such as FMOL to allow for real–time network collaboration using generative techniques. Local and distributed ensembles use the jam2jam software which only networks control data circumventing many of the issues related to audio and video streaming (Brown, 2010).

A newer trend in commercial software is to allow for peer-to-peer audio and possibly video streaming between users. Instead of attempting to reduce latency below detectable levels, the latency is instead made a factor of metric division. Programs like NinJam, eJamming, and Digital Musician Net offer varying features such as chat, video, and social networking. They argue that the metric based delay is an acceptable caveat for their users (Cockos Incorporated, 2004; Ninjam, 2010; GMBH, 2012; Kleimola, 2006). Using this technique multiple people can participate in a near real–time jam or active collaboration using real and/or virtual instruments depending on the software implementation. The music tends to be rhythmic with a regular pulse, yet it generally lacks sudden changes because of the increased temporal distance between the users.

There has been some interest in using network music as an educational tool and for



Mephistopheles:
I am the shadows breathed by bodies. The shadows that cling to the flesh, adhere to the bodies. The shadows that bring them together, mix them, merge them, confuse them, unite them. I am the saliva, the sweat, the sperm of the clouds that dribbles over the flesh.
Faust:
This accursed thing that is the earth, repugnant rotting place of men and beasts. Always fresh new blood. The germs, the bacteria, the virus of life multiply in the air, the water, the earth!

Figure 2.6: Screen capture of a configuration window in *Faust Music On Line* (Jordà, 1999).

novices. Traditionally these systems have been made for technologically adept musicians. Systems like the networked version of Drumstep or the recent CODES seek to widen the user base beyond the current niche groups (Bligh, Jennings, & Tangney, 2005; Miletto, Pimenta, Bouchet, Sansonnet, & Keller, 2011; Pimenta, Miletto, Flores, & Hoppe, 2011). Both of these systems allow multiple users to collaborate on a piece of music through a simple iterative editing process that happens in near real–time. By simplifying the interface and utilizing more popular styles in a loop based format, users with little or no musical background can quickly begin collaborating in a group effort. Similarly, Smule's AutoRap phone application allows for networked freestyle rap battles utilizing auto–tune and rhythmic manipulation (Smule, 2014). Potential contenders invite their friends to a battle and the software manipulates and mixes the two performances into a final track for sharing.

2.4 Interactive Music in the Browser

During the 1990s internet connections proliferated into homes and businesses across the globe (Ryan, 2010). Browsers and the websites they visited became increasingly sophisticated with more in depth interactivity and deeper content. The introduction of new technologies like Java, Javascript, Shockwave, Flash, Quick Time and Real Audio allowed for web developers to create complex sites that went beyond the simple display of information (Wöhrmann & Ballet, 1999). By the late 1990s websites such as *Cathedral* by William Duckworth were being made that could only be described as an experience (Duckworth, 2001). When *Cathedral* went online in 1998 it was a unique destination because it took the interactivity of the available technologies and created a rich audio and visual environment where the users could explore and interact with sound in an intuitive but aesthetic way (Duckworth, 1999b). Previously websites might provide a background song or sound effects when the mouse hovered over certain items. *Cathedral* allowed for user manipulation of sequencing elements in addition to simple playback allowing users from any background to experiment with approaches that in the past would require special software and expertise, but also with a dramatic and aesthetic presentation (Duckworth, 1999a). Soon after *Cathedral* went online a very different website named *Integer* hosted a network performance where users dictated the playlist of a radio by selecting tracks in a browser. The performance brought in large numbers and the site recorded 35,000 clicks during the 22 minute performance by almost 500 unique visitors (Iber, 1999; Tanzi, 2001). A similar approach was taken by Seionshin Yamagishi and Kohji Setoh in their work *Variations for WWW* which took user input from a website and used it to create variations on a melody which was being broadcast by audio online (Yamagishi, 1998).

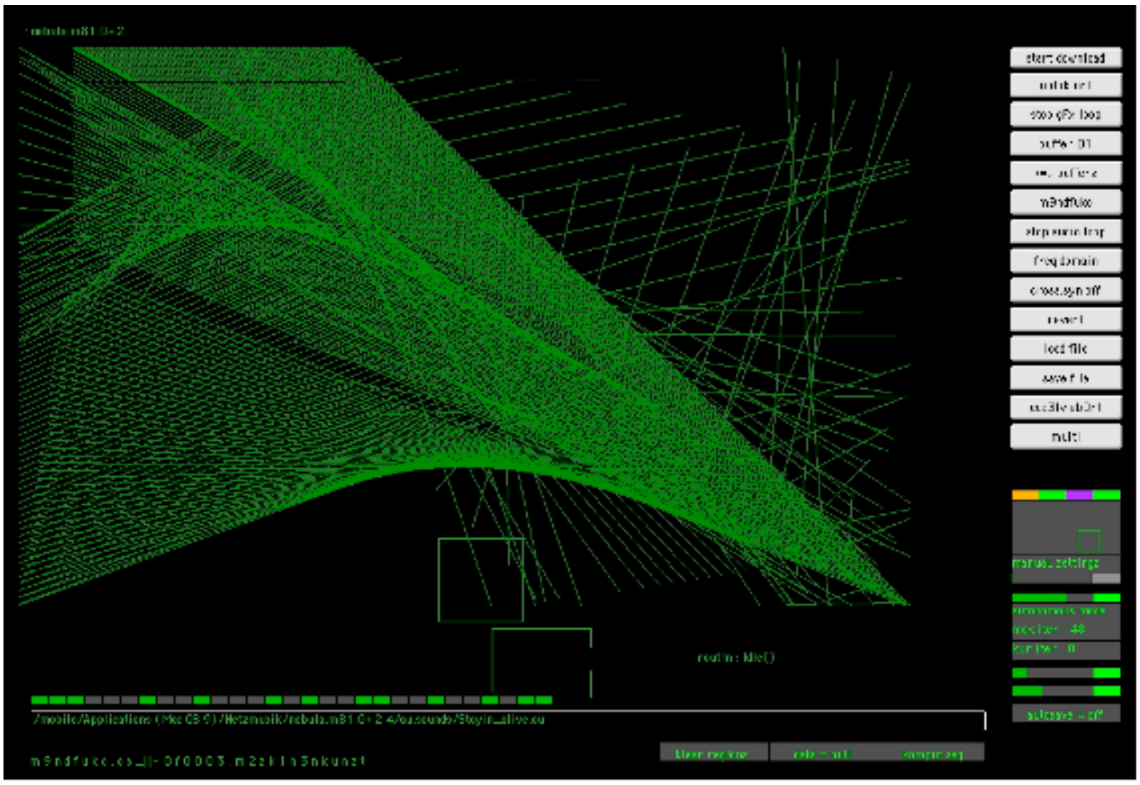


Figure 2.7: Screen capture of *nm.81* by Netochka Nezvanova (Föllmer, 2005)

In the late 1990s Rebekah Wilson (and possibly other programmers and artists collectively) using the pseudonym Netochka Nezvanova and other associated names such as “punktprotokol” and “0f0003” created a stir on the internet through disruptive posts in mailing lists but also because of the interesting software that the identity was putting out (Mieszkowski, 2002; Nezvanova & Föllmer, 2002). The first programs released under the name included *0f0003 propaganda* and *b1257+12* (both released in 1998) facilitated the creation of multimedia works using algorithmic techniques. Soon after in 1999 she/it/they released *m9ndfukc.0+99* and *k!berzveta.0+2* which were programs written in Java that interpreted network data and were likely precursors to the transmediale award winning *nm.81*. *nm.81* was an experimental web browser that rendered html code into a generative audio and visual environment, somewhat similar to *Cathedral* although more abstract (Cramer, 2005; Nezvanova, 2000).

Two artists using the moniker [The User] developed a project throughout the early 2000s to utilize the unique acoustics of an abandoned grain storage facility in Montreal. The project *Silophone* invited participants to visit a website or call by phone, where they could submit sounds to be amplified through the space (McIntosh & Madan, 2012). A microphone would record the resulting transformation and broadcast it back over the internet to create a new kind of site specific, yet network–based installation. To date 17,115 different sounds have been submitted online which accounts for a diverse collection of sound effects, music, speech, and sound art. The installation differs from the previous examples in that it merged a real space with a virtual site, creating a synthesized hybrid. In 2004 another online sound piece named *Radio Astronomy* was coordinated by the art group *r a d i o q u a l i a* with several radio telescopes throughout the world. The scale of the work is massive when considering the literal distances involved. Radio telescopes in Hawaii, Florida, and Irbene capture signals transmitted by distant planets and stars and then converted the mix into an audio signal which was broadcast by a radio station in New Zealand and online (Hyde & Harger, 1998; Astronomy, 2004). The resulting sounds have striking similarities to many experimental electronic pieces and fused the realms of art and science, much like Alvin Lucier's *Sferics* (1988) where he recorded and amplified ionospheric disturbances in the Earth's atmosphere (Weiss, 2008).

Phil Burk began working on *JSyn* and *Transjam* during the late 1990s and throughout the 2000s, which have been used as the technological framework for several works. JSyn is an audio API written in Java on top of a C based browser plugin (although recently he released a version that no longer requires the plugin) that enables developers to synthesize audio in an Internet browser (Burk, 1998). Transjam is a client/server architecture for facilitating group collaboration, accounting for locking of edited data and maintaining of user groups (Burk, 2000). Burk's test program *Webdrum* (2000) utilized the technologies to allow for visitors of his TransJam site to collaborate together creating drum loops over the internet (Burk, 1997). Subsequently, the Hub members John Bischoff and Chris Brown used these technologies to develop the works *Aperture* (2003) and *Eternal Network music* (2003), respectively, that allow for sonic exploration and collaboration in a browser (Bischoff, 2003; Brown, 2003). The two works do not have very developed or detailed user interfaces. Their approaches were decidedly more experimental than *Webdrum* and the contributions of other users were easily heard as well as notable on screen. Soon after *Aperture* and *Eternal Network Music* Max Neuhaus directed the creation of *Auracle* (2004), which was built using JSyn and TransJam with help from Phil Burk, Jason Freeman, C. Ramakrishnan, and Kristjan Varnik (Neuhaus, 2004a). Visitors to the site can log in and join others in a group "jam" where the voice is used to create gestures which are analyzed to create control data. This control data is then sent to a central server and broadcast to the network to control a software synthesizer. None of the actual audio from the voice is transmitted, instead 43 parameters including pitch and the statistical analysis of emotion were fed into a neural network for projecting changes onto a three dimensional parameter space for controlling the synthesizers (Freeman, Varnik, Ramakrishnan, Neuhaus, Burk, & Birchfield, 2005). The graphical interface provides a visualization of the last five vocal gestures that the users in the network have created, where amplitude, pitch, and time are mapped (Ramakrishnan, 2004).

Auracle collaborator Jason Freeman went on to create *Graph Theory* (2006) which takes a very different approach. There is no group collaboration and instead the user is presented with a clean interface depicting a graph of short repeated cells. The only thing available for the user to the control is the direction of traversal through the graph (Freeman, 2005, 2007). While sparse, the site allows the visitor to explore the relationships of individual figures in a crafted piece that is evocative of Morton Feldman's string quartets. The work is also notable because it uses only a few violin samples to create all of the cells and forgoes the synthesized approach of many of the previously mentioned works. The work is influenced and closely related to the New York Miniaturist Ensemble's collaborative composition site which let site visitors add or remove notes and musical expression marking from a single staff system (theory: interfacing audiences into the compositional process, 2005).

Peter Traub took a very different direction from all the previously mentioned online works with his quirky *ItSpace* (2007) project. Traub was interested in the concept of spaces, both real and virtual, and created an online conceptual sound art installation (Truab, 2010) utilizing the social network Myspace (Myspace, 2012). Traub created nine music accounts representing nine objects in his household and uploaded a picture of them along with a short piece of music created only using that object. The nine objects (metal bowl, pair of vases, recliner, bainster, wine glasses, shower head, pillow, egg timers, and folding table) friended each other on the site and other users were invited to create similar accounts. Traub accounts for at least thirty new accounts that friended each other, creating a growing network of objects that existed both in real space and in virtual space divorced from their original intent, but also representative of it.

Batuhan Bozkurt's *Otomata* (2011) (Bozkurt, 2011b) and *Circuli* (2012) (Bozkurt, 2011a) are two recent examples of music created in the browser using some newer technology, in this case the language Haxe (Ponticelli & McColl-Sylveste, 2008), to create elaborate generative sequencers. *Otomata* uses cellular automata type rules to determine direction changes for cell movement and sounds are created when cells collide with walls. The simple rules generate surprisingly complex and varied results that continuously evolve. *Circuli* sheds the typical sequencer grid and instead uses the collision of growing circles to produce codependencies and emergent behavior.

A slightly more recent work is *Plink* by Dinamoe Labs, which is advertised as "a super intuitive multiplayer music experience" (Dinamoe Labs, 2013). In *Plink* players use mouse clicks on an endlessly forward marching beat grid to create pentatonic melodies and sample based drum beats. The only inputs are mouse height, mouse button down/up,

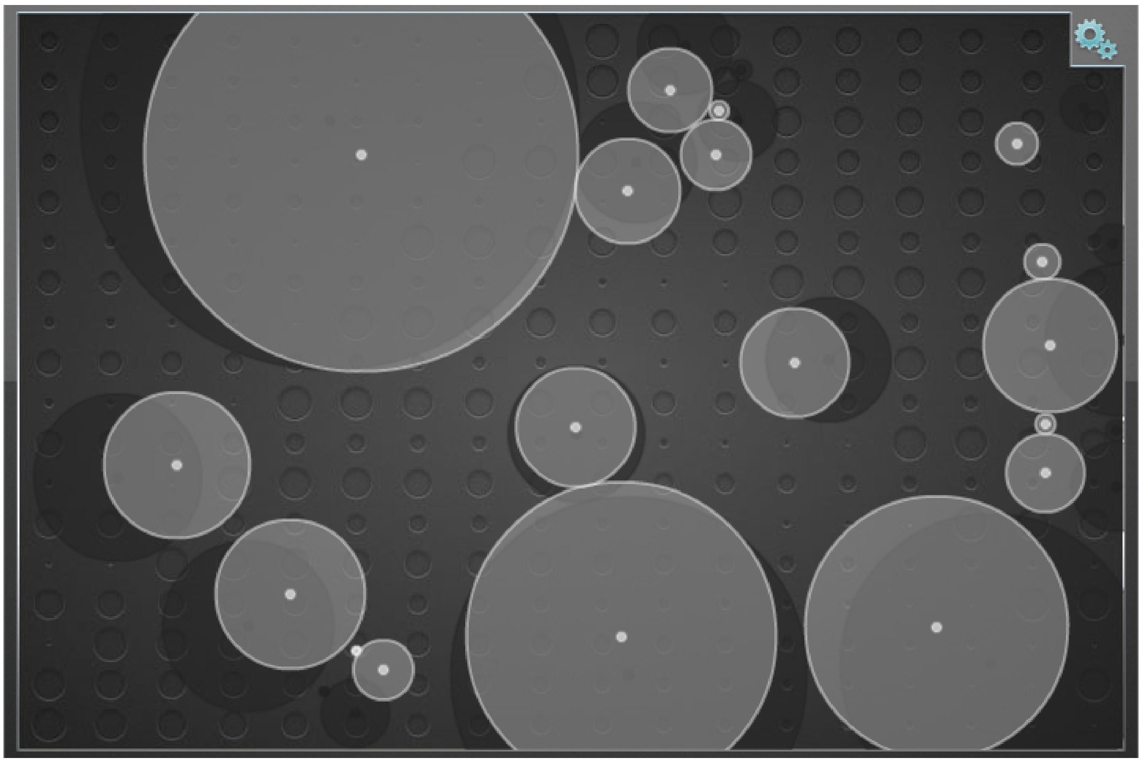


Figure 2.8: One possible configuration of circles in *Circuli* (Bozkurt, 2011a).

and instrument selection. While incredibly simple, the interplay with other players is immediately engaging.

2.5 Network Music, Present and Future Trends and Technologies

2.5.1 Laptop Orchestras and Bands

The laptop orchestra is an effort to create an ensemble with the functionality of a traditional orchestra, but with laptop technology. The Princeton Laptop Orchestra (PLOrk), which uses an identical laptop and hemispheric speaker setup for their performers has garnered much publicity (Fiebrink, Wang, & Cook, 2007; Wang, 2007), yet they were not the first. The *laptop orchestra*, an ensemble originating in Japan, likely held the first performance for such an orchestra in Tokyo in 2002 (Orchestra, 2012). Since then many of these ensembles have been created at other institutions such as the Stanford Laptop Orchestra (SLOrk) (Stanford University, 2012) and the Boulder Laptop Orchestra (BLOrk) (University of Colorado at Boulder, 2012). Laptop bands such as PowerBooks_UnPlugged and the Hub differ from these laptop orchestras, which often have a traditional composer and performer relationships, by encouraging a technical virtuosity among members as well as shared authorship (Rohrhuber, de Campo, Wieser, van Kampen, Ho, & Hölzl, 2007). Some bands such as the Birminham Laptop Ensemble (BiLE) or the author's own band Glitch Lich go as far as writing reactionary manifestos (Purloined Letters and Distributed Persons, 2011; McKinney et al., 2012).

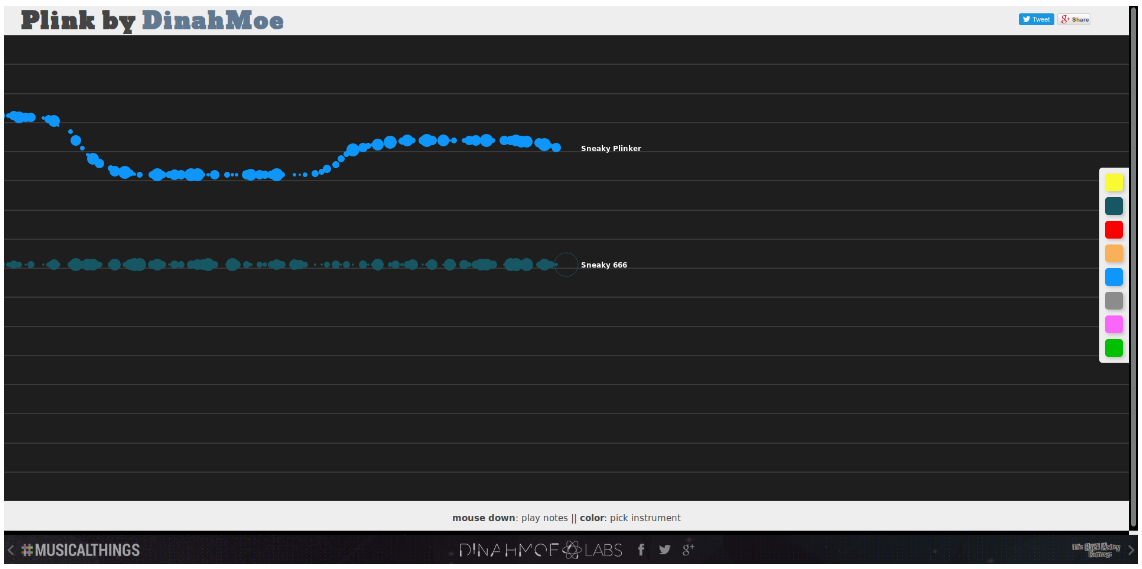


Figure 2.9: Two players playing *Plink* online at the same time (Dinamoe Labs, 2013).

2.5.2 Developments in Network Technology

Network music is directly tied to developments in technology, therefore it is important to recognize these advances in order to begin to understand some of the ways in which the field will change and grow. One looming change is the transition from IPv4 to IPv6, which is required to fulfill the growing demand for IP addresses (Li, Jinmei, & Shima, 2007). IPv4 uses a 32bit IP address which only allows for 4.29 billion unique values. This had been a large enough number to not cause concern, but given the rapid increase in worldwide internet usage, 4.29 billion addresses has become inadequate. In contrast IPv6 uses a 128bit address scheme, allowing for 340 undecillion addresses. Many of the technologies used by network musicians are ill prepared for the change to IPv6 and there will need to be further developments for a smooth transition. The proliferation of Global Positioning System (GPS) enabled devices will surely be a valuable resources for network musicians and net artists (Xu, 2003). There has already been some work utilizing this technology including the *Nomadic Milk* project which used GPS to trace routes for milk delivery, the GPS Beatmap that utilizes global position for audio track mixing (St. Pierre, Stiles, & Bahn, 2006), and the *NoTours Project* which uses it to created augmented reality sound walks (Polak, 2012).

Cloud computing is an interesting development that could have large and lasting effects. Operating systems like Google's Chrome OS, which stores most of its information and applications in a cluster of servers, will make portable devices both cheaper and smaller, which will benefit the development of music and art by and for these devices (Buyya, Broberg, & Goscinski, 2011). This could have far reaching implications on issues related to privacy, data sovereignty, and security (Alleweldt, Kara, Fielder, Brown, Weber, & McSpedden-Brown, 2012). The so called Internet of Things (IoT) is beginning to take shape and soon everything in a home or office will be connected to the internet (Hersent, Boswarthick, & Elloumi, 2011). Hewlitt -Packard has recently announced the Central Nervous System for the Earth (CeNSE) which extends the IoT concept beyond appliances to the entire Earth. The project envisions a vast network of sensors placed in roadsides, bridges, lakes, oceans, and deep underground which will give a global set of senses to the internet (Packard, 2009). This massive growth is bound to have repercussions on, and reactions by, the community of network musicians who will have increasing resources at their disposal.

Bandwidth and transfer speeds are always a concern in network music. There is continuing research for new strategies and techniques to increase performance including recent experiments that implement photon Orbital Angular Momentum (OAM) to transfer 2.5 terabits of data per second over wifi (Wang, Yang, Fazal, Ahmed, Yan, Huang, Ren, Yue, Dolinar, Tur, & Willner, 2012) or research into photonic chips to decrease bottlenecks caused by silicon processors in network routers(Koka, McCracken, Schwetman, Zheng, Ho, & Krishnamoorthy, 2010). No matter what advances are made, we are bound by the laws of physics and the speed of light. Many theories have been suggested to achieve superluminal transfer speeds such as Nimtz's quantum tunneling experiments (highly disputable), quantum entanglement (no classical information can be encrypted then decrypted), neutrinos (incorrect measurements due to a loose cable), and the existence of tachyons (never observed and a likely sign that a theory is unstable) (Nimtz, Heitmann, Roy-Brehonnet, & Jeune, 1997; Marinescu & Marinescu, 2011; Adam, Agafonova, Aleksandrov, Altinok, Sanchez, & Aoki, 2011; Tipler & Llewellyn, 2007). Barring some striking future development, we will someday reach a maximal speed of communication.

2.5.3 Languages, Frameworks, and Live Coding

Network music is often created using specialized languages and software to simplify and abstract the lower level issues related to the practice. Many programming languages have audio libraries, but using a specialized language can greatly shorten development time and prevent repeating established work. CSound is a veteran among computer music languages with over 30 years of development. The language was not originally designed for real–time use or networking but there now exists extensions to the language to enable these features (Boulanger, 2000). SuperCollider was originally developed in the early 1990s but the most recent major revision (SC3) features networking as a core feature. The language is split into a client/server architecture that allows for modularity and distributed configurations (McCartney et al., 2016). ChucK is similar to SuperCollider in many respects, although much younger and without the underlying server/client architecture (Wang, 2002). Max/MSP and Pure Data are two closely related graphical programming languages which allow for development using a patching scheme (Cycling '74, 2014; Chung, 2013). Assessing differences between programming paradigms is difficult but there has been some research into the differences between visual data flow languages and text based languages (Green & Petre, 1996). It has been found that visual programming contributes to a lack of understanding of dependencies and control flow, but can give the user a better understanding of data flow (Navarro-Prieto & Cañas, 2001). That said, This research does not assess their use in large projects. Frameworks for developing two dimensional and three dimensional graphics are often used in conjunction with the previously mentioned audio languages to create multimedia performances and installations. These frameworks are often simply large libraries written in a pre–existing language such as Processing which is written in Java (Reas & Fry, 2007), and openFrameworks and Cinder which are written in C++ (Perevalov, 2013; Rijnieks, 2013).

Often these languages and frameworks make use of the Open Sound Control (OSC) (Wright, 2002) protocol to share information between computers and programs. While powerful, these libraries require a large amount of effort to create some framework for groups to use. This has led to a proliferation of higher level network libraries and frameworks such as OSCGroups (Bencina, 2005), OSCthulhu (McKinney & McKinney, 2012), Benoit Lib (Borgeat, Ballweg, & Romero, 2012), the Co-Audicle (Wang, Misra, & Cook, 2006), and the Republic (J. Rohruber, A. de Campo, 2011) which all serve to simplify collaboration by building on top of existing technologies. These tools are rich and powerful but are difficult to use, requiring some amount of expertise and with varying levels of stability and operating system support.

Figure 2.10: Live coding band Slub performing with Tidal and Scheme Blocks (Aagaard, 2013).

Live coding, where performance write code to create and manipulate programs in real–time, is a growing trend in computer music performances. There have been reports of live coding performances as early as 1986 when Ron Kuivila used Forth in a performance and subsequently crashed, but not before making “some quite interesting music” (Ward, Rohrhuber, Olofsson, McLean, Griffiths, Collins, & Alexander, 2004). Live coding is a fundamentally computer based performance style and for that reason lends itself well to networking. The information being transferred is small, yet can produce long lasting results and the time specific parameters for execution are much less restrictive than in a traditional performance. The practice leads to organic interdependencies and produces layers of uncertainty which afford unexplored modes of performance, composition, and listening (De Campo & Rohrhuber, 2004)

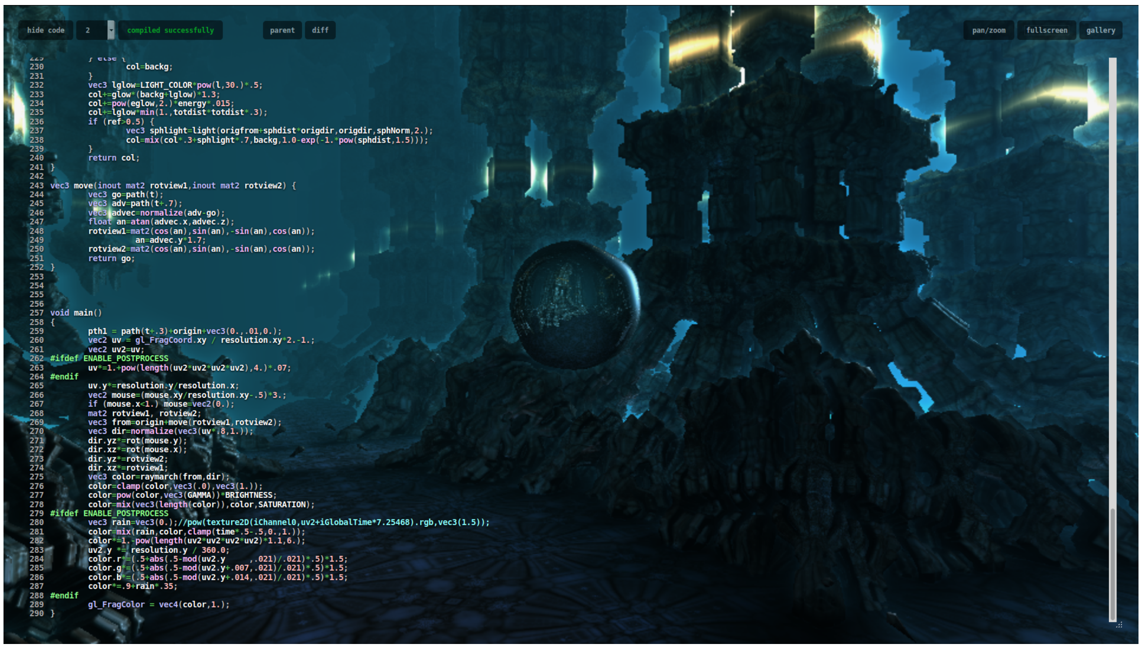
Unlike audio and video streaming approaches, the sharing of code between users requires very little bandwidth and adverse effects from dropped packets can be easily corrected or even ignored depending on the intent of the system design. PowerBooks Unplugged were early pioneers in merging networking techniques with live coding performances (Rohrhuber et al., 2007). Eschewing the stage to sit among the audience, members use only the sound produced by their laptop speakers, although sometimes augmented by an amplified signal for mid and low frequencies, using OSC messages to share their code among members. Live coding performances have utilized many different languages and approaches often with an emphasis on honesty and communication with audiences (Ward et al., 2004).

Because of the difficulty of programming live with traditional compiled languages such as C++ or Java, a plethora live coding frameworks and languages have been created. These languages can largely be divided into two sub groups: Languages which are essentially extensions of an existing language, or languages with new live coding specific syntax and subsequently requiring custom compilers. There are libraries and frameworks in many of the most popular languages such as Lua (Doornekamp, 2013; Lee & Essl, 2013), Ruby (Aaron & Blackwell, 2013), Scheme (Sorensen & Gardner, 2010; Sorensen, 2014; Griffiths, 2014), Clojure (Aaron, 2014), C (Heiland-Allen, 2012), SuperCollider (J. Rohruber, A. de Campo, 2011), and Haskell (Bell, 2011). These frameworks and environments build upon existing languages providing audio or visual libraries with special functionality for creating patterns and other generative features. New languages have begun to be written that use special syntaxes to attempt to make live coding as easy and fast as possible, often trading versatility for speed. Two examples of these are The Haskell based Tidal (McLean, 2011), the SuperCollider based IxiLang (Magnusson, 2011), and Cyril (Mothersele, 2014), written in c++. Tidal utilizes a small embedded language for layout based pattern generation and manipulation while still allowing for code to be written in traditional Haskell. IxiLang is written on top of SuperCollider but provides a layout based syntax for creating melody and drum patterns and also features special functionality for scheduling and effects routing. Cyril provides a terse Python–like syntax for creating recursive functions for graphics generation. Additionally there have been several live coding languages written that utilize a visual programming paradigm. These languages such as Beta Blocker, Scheme Blocks, Al Jaziri, and Texture provide either a highly graphical interface for text based input or even utilize other kinds of input such as a mouse (McLean, Griffiths, Collins, & Wiggins, 2010; McLean & Wiggins, 2011).

2.5.4 Web Technology and Live Coding

Web development technologies are often used for music and sound art, especially now that the concept of a web application has become established. Flash and JavaScript are common solutions because of their simple implementation and ease of deployment, although Flash is currently being phased out of development (Grover, 2011; Flanagan, 2006). When a server is necessary, Node.js, which is built on top of Google's V8 JavaScript engine, provides easy development and can service multiple connections without I/O blocking (Joyent, Inc., 2012; Node.js, 2012). Sites using the new HTML5 version are becoming increasingly common. HTML5 fills out many of the shortcomings of traditional HTML by including tools for graphics, video, geo-location, and audio among others (Pilgrim, 2010). Programmers already familiar with Processing can easily port their code for the web by using Processing.js which compiles down to javascript and embeds in HTML5 (Up & Running, 2012). For 3D graphics, THREE.js (Cabello, 2010) is a popular JavaScript library for utilizing WebGL scene rendering using a more familiar node scene API. Audio has seen dramatic improvements with the adoption of the Web Audio API (Rogers, 2013) by several major browsers such as Chrome, FireFox, and Safari. Web Audio allows developers to read and write raw audio, utilize built–in audio unit nodes, and even employ FFT analysis, all from JavaScript and without the need for any plugins.

Browser based technologies are a good choice for collaborative frameworks because web browsers are common and their development is widely supported with heavily funded development. The recent advent of web standards such as WebGL (Khronos Group, 2013) and more recently Web Audio (Rogers, 2013) has led to many early efforts for web based graphics and audio applications. WebGL and Web Audio are important because they give developers access to powerful functionality such as OpenGL Shader Language (GLSL) support and real-time audio synthesis that was previously unavailable. Among these online graphics and audio applications are several live coding environments. Live Coding is a natural fit for web development, not just because of WebGL and Web Audio, but also because of the built in support for text editing in HTML documents and the ability to use JavaScript as a target language for code generation.



```
	} else {
		col=backg;
	}
	vec3 lglow=LIGHT_COLOR*pow(l,30.)*.5;
	col+=glow*(backg+lglow)*1.3;
	col+=pow(eglow,2.)*energy*.015;
	col+=lglow*min(1.,totdist*totdist*.3);
	if (ref>0.5) {
		vec3 sphlight=light(origfrom+sphdist*origdir,origdir,sphNorm,2.);
		col=mix(col*.3+sphlight*.7,backg,1.0-exp(-1.*pow(sphdist,1.5)));
	}
	return col;
}

vec3 move(inout mat2 rotview1,inout mat2 rotview2) {
	vec3 go=path(t);
	vec3 adv=path(t+.7);
	vec3 advec=normalize(adv-go);
	float an=atan(advec.x,advec.z);
	rotview1=mat2(cos(an),sin(an),-sin(an),cos(an));
		  an=advec.y*1.7;
	rotview2=mat2(cos(an),sin(an),-sin(an),cos(an));
	return go;
}




void main()
{
	pth1 = path(t+.3)+origin+vec3(0.,.01,0.);
	vec2 uv = gl_FragCoord.xy / resolution.xy*2.-1.;
	vec2 uv2=uv;
#ifdef ENABLE_POSTPROCESS
	uv*=1.+pow(length(uv2*uv2*uv2*uv2),4.)*.07;
#endif
	uv.y*=resolution.y/resolution.x;
	vec2 mouse=(mouse.xy/resolution.xy-.5)*3.;
	if (mouse.x<1.) mouse=vec2(0.);
	mat2 rotview1, rotview2;
	vec3 from=origin+move(rotview1,rotview2);
	vec3 dir=normalize(vec3(uv*.8,1.));
	dir.yz*=rot(mouse.y);
	dir.xz*=rot(mouse.x);
	dir.yz*=rotview2;
	dir.xz*=rotview1;
	vec3 color=raymarch(from,dir);
	color=clamp(color,vec3(.0),vec3(1.));
	color=pow(color,vec3(GAMMA))*BRIGHTNESS;
	color=mix(vec3(length(color)),color,SATURATION);
#ifdef ENABLE_POSTPROCESS
	vec3 rain=vec3(0.);//pow(texture2D(iChannel0,uv2+iGlobalTime*7.25468).rgb,vec3(1.5));
	color=mix(rain,color,clamp(time*.5-.5,0.,1.));
	color*=1.-pow(length(uv2*uv2*uv2*uv2)*1.1,6.);
	uv2.y *= resolution.y / 360.0;
	color.r*=(.5+abs(.5-mod(uv2.y      ,.021)/.021)*.5)*1.5;
	color.g*=(.5+abs(.5-mod(uv2.y+.007,.021)/.021)*.5)*1.5;
	color.b*=(.5+abs(.5-mod(uv2.y+.014,.021)/.021)*.5)*1.5;
	color*=.9+rain*.35;
#endif
	gl_FragColor = vec4(color,1.);
}
```

Figure 2.11: Browser based generative visuals in GLSL Sandbox (Fontan & Goberna, 2013).

Before the creation of WebGL and Web Audio, sites like Jsaxus (Brodsky, 2013) and Flaxus (Ivanoff & Jimenez, 2006) used JavaScript and Adobe Flash to allow users to program graphics based applications in real–time. The standardization of the Web GL specification now allows for 3D accelerated programs to be written entirely inside a web browser. Sites like livecoding.io (Florit, 2013), Livecodelab (Casa, McDonald, Stutters, & Ryan, 2013), Livecoder (Obermeyer, 2013), WebGL Playground (Samp, 2013), and GLSL Sandbox (Cabello, 2013) have harnessed this API to create live programming environments that can be easily accessed from any computer using a web browser. More recently the Web Audio specification has been developed, but the project is younger than WebGL and is still undergoing active development and lacks a complete cross-browser implementation. CoffeeCollider (Yonamine, 2013) is a newer framework that looks to bring SuperCollider like functionality to the browser. It leverages CoffeeScript for its more elegant syntax while providing basic functionality for synthesis and sequencing, but does not support graphics or collaborative programming. Gibber is an early adopter and provides a thorough audio synthesis environment for live music and graphics programming (Roberts, Wakefield, & Wright, 2014). Gibber has recently added collaborative sessions, making it one of the best choices for web based live coding collaborations (Roberts, 2014).

2.5.5 Mobile Development

Mobile devices evolved dramatically in the last 10 years from simple utilitarian phones to powerful tangible interfaces with internet connectivity. Music has benefited greatly from this transition, not only because of the obvious music playback capabilities inherent in the devices, but also because they open up a large design space. Even before smart phones were developed composers saw the potential for the medium. *Dialtones (A Telesymphony)* (2001), direct by Golan Levin with work by several other contributors, is a large scale piece where the entirety of the audio output is generated by the audience members' cell phones. Audience members were asked to register their phone numbers and download ringtones before the concert. Throughout the performance their phones were called in sequence (Levin, Gibbons, Shakar, Sohrawardy, Gruber, Lehner, Schmidl, & Semlak, 2001) generating a naturally spatially diffused music. Greg Shiemer wrote a series of microtonal pieces called *Mandala* for the Pocket Gamelan. Control messages were passed between phones using Bluetooth connections which control chord selection and by swinging the phones subtle changes in pitch were be achieved (Schiemer & Havryliv, 2006).

The spread of smart phones in the marketplace created a new platform for developers to target. Stanford built on the Laptop Orchestra concept by creating the Mobile Phone Orchestra (MoPhO) which utilizes the lightweight, yet powerful technology to create electronic music where the performers can literally be mobile (Wang, Essl, & Penttinen, 2008; Oh, Herrera, Bryan, Dahl, & Wang, 2010). Mogees (Mogees Ltd., 2014) is a new product with a recently successful KickStarter campaign (Kickstarter Inc., 2014). Mogees that combines smart phone software with contacts mics and generative synthesis to create a dynamic system for electronic music improvisation using acoustic sound sources. Similarly Phonotonic is developing Interative Music Battle (Phonotonic, 2014), another mobile phone application and hardware combination. The Interactive Music Battle hardware consists of a small silicon ball with a gyroscope, accelerometers, and WiFI connectivity.

Users manipulate melodies and drum beats by rotating, moving, and throwing the small ball and the system is designed to be used for collaboration.

Phones are not the only important mobile devices. The tablet computer market was rejuvenated when Apple released the iPad in 2010. Tablet devices lack the pocket portability of smart phones, but the larger touch screen displays enable richer interfaces. There is a rapidly growing collection of commercial music applications designed for tablets such as the Reactable Mobile application (Reactable Systems, 2014) that utilizes the larger touch screen on tablets to emulate the larger Reactable tangible controller system (Jordà, 2009). This genre is large and growing with a variety of different touch based interfaces (Apple Inc., 2014; Google, 2014a; Liljedahl, 2014). While the networking and collaborative components of most of this genre is often limited to track sharing there are projects such as UrMus (Essl, 2011) that utilize mobile technology in conjunction with networking to create collaborative touch based interfaces.

2.6 Theory and Taxonomy

Although computer network music has over thirty years of history (and network music as a whole much longer) it is only recently that researchers have started to develop theories and taxonomies to define and categorize the field. Two such taxonomies demonstrate some strong overlap; one developed by Alvaro Barbosa in his PhD thesis *Computer-Supported Cooperative Work for Music Applications* (Barbosa, 2003) and another by Andrew Hugill described in his book *Internet Music: An Introduction* (Hugill, 2005). Both provide insights for how to categorize the various approaches to network music and yet are separate in their particular methodologies. Barbosa takes direct cues from the Computer-Supported Cooperative Work field by presenting a four square cube that emphasizes the difference in time and place between the users (performers). The four square classification is an adaptation of Robert Johansen's Classification space (although Barbosa refers to a slightly reworded interpretation from Tom Rodden) which divided software designed for group use into four distinct categories (Johansen, 1988; Rodden, 1992). The grid is defined in two dimensional space where each axis represents time and location respectively as a function of proximity. Barbosa defines four classes of network music that fall along these axes, although not completely segregated in the dimension space, as Co–Located Musical Networks, Music Composition Support Systems, Remote Music Performance Systems, and Shared Sonic Environments.

Co–Located Musical Networks are networks where the performers are in the same location at the same time, but use some kind of local network connection to produce interdependency. This can include acoustic and electronic instruments in any kind of style, but the emphasis is on the close proximity of the performers in both space and time. Performances by the Hub can be categorized as Co–Located Musical Networks. *Music Composition Support Systems* allow synchronous or asynchronous collaboration on music from a more traditional composition–based approach. This can include notation systems for multiple users and networked digital audio workstations. The type of information that

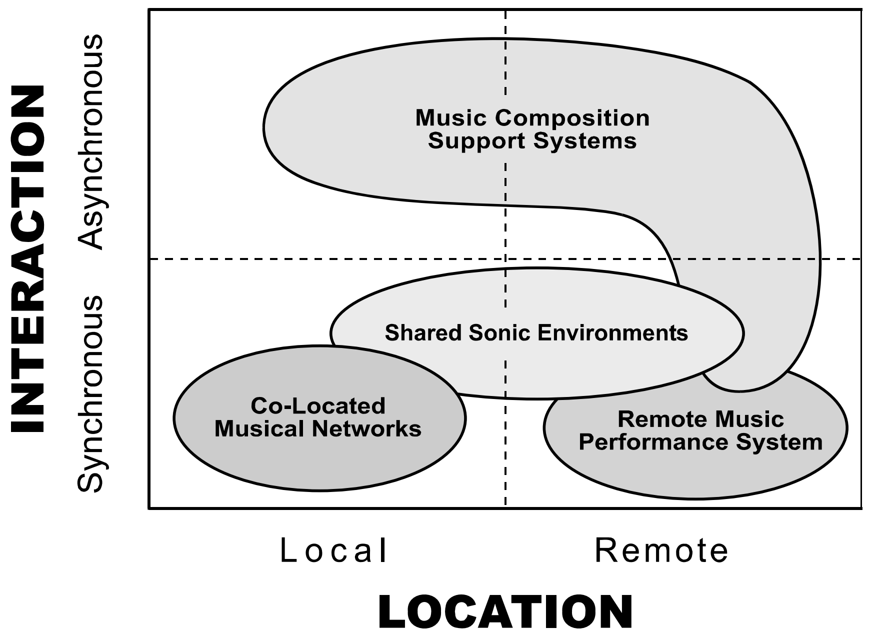

INTERACTION
Asynchronous
Synchronous
Music Composition Support Systems
Shared Sonic Environments
Co-Located Musical Networks
Remote Music Performance System
Local
Remote
LOCATION

Figure 2.12: Barbosa's Network Music classifications placed in Johansen's four-square dimension space

is shared may differ from program to program, however the emphasis is on collaborating on a composed piece of music using groupware. NinJam and eJamming are examples of Music Composition Support Systems.

Remote Music Performance Systems allow for performers to play music from different locations but in close temporal proximity. This can include anything from audio and video streaming telepresence scenarios to virtual instrument collaborations. In any scenario, latency and network behavior is always a consideration. The 2002 distance based collaboration between McGill and Stanford is an example of a Remote Music Performance System. Barbosa describes *Shared Sonic Environments* as lacking a specific time scale, using the internet as a kind of interactive audio realm. This can include sound art installations or other kinds of audio experiments that draw on networking or the internet for data mining or control. Stelarc's *Internet Ear* could be classified as a Shared Sonic Environment.

Barbosa's taxonomy uses two fundamentals of physics, time and space, to create a grid for categorization. Because time and space are so fundamental, this taxonomy is still useful thirteen years after the original publication. This is impressive when considering how much the internet and networking technologies have evolved in that time. While that is true, the taxonomy accomplishes this by using a very high level methodology, and therefore some network music pieces and systems can be grouped together while having large differences such as instrumentation, genre, interfaces, and the presence of added visuals.

Music that Uses the Network to Connect Physical Spaces or Instruments
Music that is Created or Performed in Virtual Environments, or Uses Virtual Instruments
Music that Translates into Sound Aspects of the Network Itself
Music that Uses the Internet to Enable Collaborative Composition or Performance
Music that is Delivered via the Internet, with Varying Degrees of User Interactivity

Table 2.1: Andrew Hugill's five internet music types.

Hugill focuses less on the time and place of network music and more on the specific techniques employed. His taxonomy is a collection of possible approaches, yet there is some clear overlap with Barbosa's classifications. Hugill posits five categories for internet music, though there is the possibility for further developments and expansions in his framework given new technologies and trends. It is important to note that Hugill uses the term Internet Music where I and others like Barbosa refer to it simply as Network Music. It is a subtle distinction, though he does note that this kind of music can and does occur on local and other kinds of networks besides just the larger internet.

Music that Uses the Network to Connect Physical Spaces or Instruments includes a large field of music where the collaborators connect to each other via some kind of connection to transmit data as command information, audio, or video. This includes all real–time collaborations where the performers are actively engaged with each other either through a simple audio/video broadcast or more interconnected dependencies. An example would be the 2002 McGill and Stanford collaboration. Hugill defines *Music that is Created or Performed in Virtual Environments, or Uses Virtual Instruments* as a unique group that uses artificial and synthetic environments for collaboration. This music relies less on specific communications between users and more on maintaining the illusion of a persistent virtual world. Participants in the virtual world can be human or artificial, allowing for a plethora of possible configurations of interactivity. *Plink* is an example of this kind of virtual environment.

Music that Translates into Sound Aspects of the Network Itself comprises music and often sound art that uses the network as a source for timing, data, and as an aesthetic resource. It tends to be concerned with internet culture and attempts to use the internet, or artifacts of the internet, as metaphor or simulacrum. *Global String* is an example of this kind of approach. *Music that Uses the Internet to Enable Collaborative Composition or Performance* comprises mostly asynchronous or at least time neutral collaborations focused on crafting an end product, such as a composition or recording. This category is very similar to Barbosa's *Music Composition Support System*, so similarly Ninjam and eJamming can be included in this group. With *Music that is Delivered via the Internet, with Varying Degrees of User Interactivity* Hugill groups audio visual website content in the form of highly interactive websites available to the open public with other kinds of online interactive media. Hugill states that there is an emphasis on short and intense engagement without the effort for prolonged usage, such as *Cathedral*.

In contrast to Barbosa's taxonomy, Hugill's five internet music types are more ad hoc and without any real fundamental dimensions for categorization. There can be clear overlap between these types such as *Music that Uses the Network to Connect Physical Spaces or Instruments* and *Music that Translates into Sound Aspects of the Network Itself*. The five types listed are somewhat broad, but certainly doesn't cover every possible configuration of network music. For example, collaborative live coded performances could possibly be grouped in as either *Music that is Created or Performed in Virtual Environments, or Uses Virtual Instruments* or *Music that Uses the Internet to Enable Collaborative Composition or Performance*, yet both of these are somewhat inadequate descriptions.

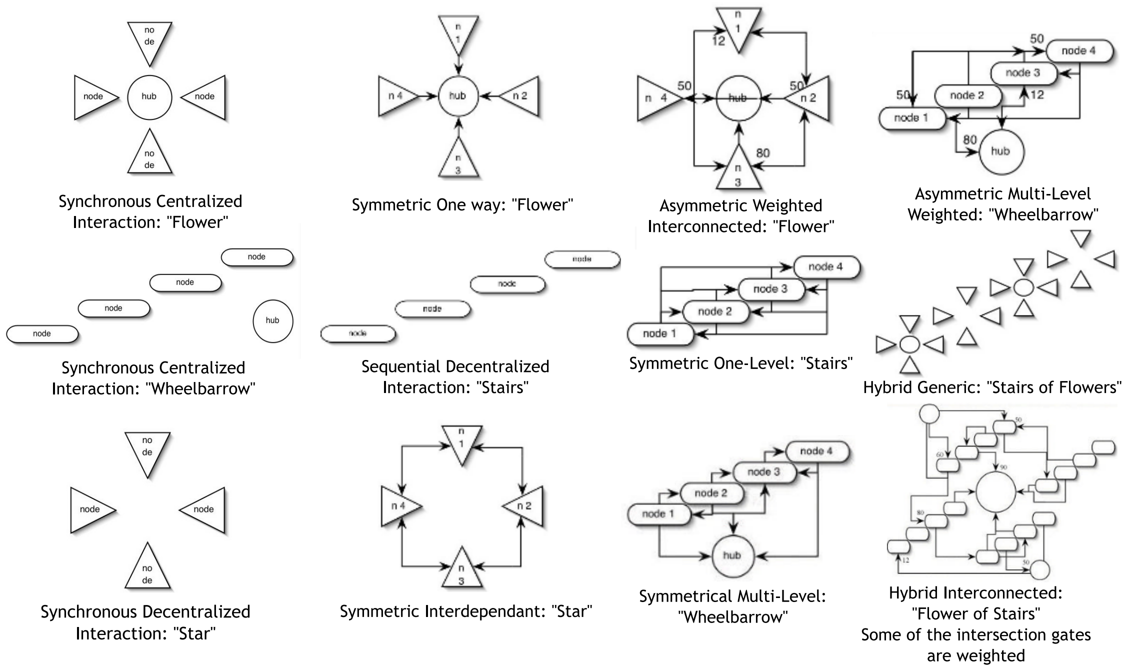

Synchronous Centralized Interaction: "Flower"
Symmetric One way: "Flower"
Asymmetric Weighted Interconnected: "Flower"
Asymmetric Multi-Level Weighted: "Wheelbarrow"
Synchronous Centralized Interaction: "Wheelbarrow"
Sequential Decentralized Interaction: "Stairs"
Symmetric One-Level: "Stairs"
Hybrid Generic: "Stairs of Flowers"
Synchronous Decentralized Interaction: "Star"
Symmetric Interdependant: "Star"
Symmetrical Multi-Level: "Wheelbarrow"
Hybrid Interconnected: "Flower of Stairs" Some of the intersection gates are weighted

Figure 2.13: Weinberg's enumeration of various network configurations

Gil Weinberg took a slightly different approach in his PhD thesis *Interconnected Musical Networks: Bringing Expression and Thoughtfulness to Collaborative Group Playing* as well as his paper *Interconnected Musical Networks: Toward a Theoretical Framework* (Weinberg, 2003, 2005). In these documents Weinberg presents a collection of architectures and topologies for the various kinds of Interconnected Musical Networks (IMNs). The largest category for consideration is whether or not the network architecture is centralized or decentralized. Centralized networks often function in a more traditional manner, mainly serving to duplicate traditional modes of performance and interaction. Decentralized networks allow for more complicated interdependencies and often the focus of the music is exploring the ways that those dependencies play out.

Centralized and Decentralized networks can also each be divided into two groups: Synchronous (real-time) and Sequential (non-real-time). Synchronous networks account for simultaneous collaboration, allowing for continuous action and reaction, and often the networking occurs during a performance. Sequential networks on the other hand facilitate slower interactions, but in doing so bypass many of the problems that plague Synchronous networks. Bandwidth usage can be higher because information disseminated in the network can be uploaded and downloaded over longer periods of time. This allows for high quality audio, video, or any other kind of important data to be shared. Collaborations that last over longer periods (days/weeks/months/years) benefit from this approach, and here the network serves more as a functional tool for composition or development instead of as an important aesthetic contribution to the resulting music. It should be noted that Weinberg's use of the term 'Sequential' may be somewhat misleading. 'Asynchronous' would work better in this context because 'Sequential' implies linearity, but this kind of networking does not necessarily have to be so. For example, version control using Git (Loeliger & McCullough, 2012) or Subversion (Pilato, Collins-Sussman, & Fitzpatrick, 2008) allows for asynchronous networking that can last over days, weeks, months, and years, but does not have a linear order of events.

Elaborating on the architectural dichotomies of Centralized/Decentralized and Synchronous/Sequential, Weinberg further accounts for specific topologies of connections between the various network nodes and the hub (if one is present) as well as the added complexity of the notion of weighted gates on traffic. Twelve configurations of architectures and topologies are presented with varying amounts of complexity. The Flower, Star, Stairs, and Wheelbarrow configurations are all possible topologies with varying characteristics, and thusly varying impacts on the flow of information in the network. A flower configuration is a synchronous topology that contains a centralized hub with nodes in various configurations where as a Star topology lacks a central hub. Wheelbarrow topologies are sequential and centralized configurations and Stairs are sequential, but decentralized.

When these topologies are used to enumerate hardware connections, many are well represented by the usual types of network ensembles such as the Synchronous Decentralized Interaction (Star), or the Synchronous Centralized Interaction (Flower). These two configurations are common in network bands and ensembles because they allow for transmission configurations that aren't overly complicated and maximize the effectiveness of the communication channels in the group. Other configurations such as the Symmetric Interdependent (Star) configuration are less resistant to network problems and others such as the Hybrid generic "Stairs of Flowers" are niche, if not completely theoretical. These topologies may also define emulated or virtual connections for data flow or progression of events. In this way many types of topologies can be built on top of a more traditional framework. Additionally, the frameworks could be expanded easily with other configurations, weights, characteristics, and now, types of devices (mobile, etc..) so it is sufficient to say that the theory is not holistic or absolute. Nevertheless, it does provide some genuine insight into the complexities of network configuration and its impact on group dynamics and aesthetics for network music.

Within his PhD thesis Weinberg also presented an analysis of the design goals for IMNs. According to his categories, IMNs that emphasize complexity, performance scenarios, virtuosity, long timescales, and experimentation are generally intended to be used by experts in the field. The systems that are used by groups such as The Hub or Powerbooks Unplugged demonstrate these kinds of characteristics and their music is usually presented to

Table 2.2: Weinberg's comparison between novice and expert IMN systems.

Novice IMN systems	Expert IMN systems
Emphasize the process and the experience – collaboration, creation, or learning. Aimed mainly at performers.	Emphasize the final product – musical composition or stage performance. Aimed mainly at audiences.
Technology's main use is to simplify interaction for players by constraining musical possibilities	Technology's main use is to create complex and rich interdependent topologies.
Low floor / low ceiling learning – fast and easy learning curve but low long- term depth value	High floor/high ceiling learning – pre-required skills and knowledge, richer longer term learning value
Designed for short interactions (seconds to minutes) in public places.	Designed for long interactions (minutes to hours) in concerts hall or on-line.
When the system leads to a coherent musical product, the music tends to be of the popular variety.	Historically, musical product tended to be of the high-art music variety. Little accessibility to wide audiences.

smaller and more technically adept audiences. Weinberg suggests that IMNs that emphasize play, simplicity, short timescales, education, or commercial value are usually designed for broader user bases and audiences, often with less musical or technical expertise. Weinberg differentiates these groups as Novice IMN systems and Expert IMN systems, although he does not use the term novice disparagingly. In fact he encourages the proliferation and usage of network technology to increase music education and appreciation amongst wider audiences. An example of a Novice IMN is *Plink*, and in contrast the Republic could be described as an expert IMN. While the terms may not be used derisively, the use of two categories to represent multiple axes of attributes is too simplistic. There are several examples that break this dichotomy such as the appropriation of toys in 'expert' performances or the use of incredibly sophisticated software by new users.

Golo Föllmer also sought to adopt a theoretical framework that could account for the many different approaches to network music. Upon a survey of the field he devised a slightly complicated multidimensional spatial framework with types and clusters. Föllmer recognizes twelve unique types of network music that can be plotted in a three dimensional spatial order. He defines the three fundamental dimensions as "interplay with network characteristics", "interactivity/openness" and "complexity/flexibility" where each is a continuum of representation in a particular approach (Föllmer, 2005).

Within the twelve groups Föllmer proposes five clusters that reduce and clarify the topology of network musics. The Forum cluster contains the Discussion Forum, Remix Lists, and Archive Projects types. Discussion Forums provide support for non–real–time collaborations, Remix Lists are similar but with a more open network for participants, and Archive Projects simply provide a repository for storage. Sites like SoundCloud could be classified in this forum cluster (SoundCloud, 2016). The Game cluster contains the Soundtoys and Flash/Shockwave Soundtoys types. Soundtoys are interfaces that emulate conventional musical instruments but with limited functionality and repertoire. Flash/Shockwave Soundtoys offer easy to use browser based interactivity in place of an instrumental

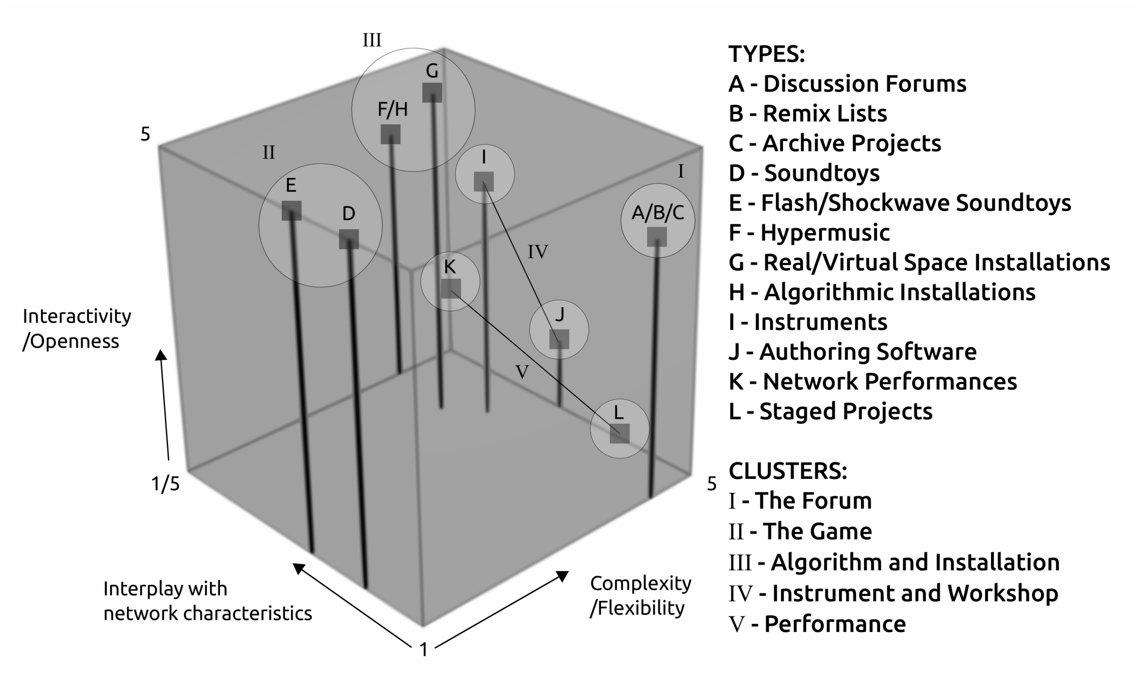


Figure 2.14: Föllmer's Spatial Order of the Twelve Types of Net Music.

interface. *Otomata* is an example of a Flash/Shockwave Soundtoy in the Game cluster.

The Algorithm and Installation cluster contains the types for Hypermusic, Real/Virtual Space Installations, and Algorithmic Installations. Hypermusics are complex soundscapes that are explored and not performed in a traditional sense. Real/Virtual Space Installations explore transitions in real and virtual space and Algorithmic Installations explore the properties of an electronic space. Stelarc's *Internet Ear* can be classified as a Real/Virtual Space Installation, residing in the Algorithm and Installation cluster. The Instrument and Workshop cluster encapsulates the Instruments and Authoring Software types. Software–based Instruments and Authoring Software offer higher complexity than the previously mentioned Soundtoys with higher degrees of control and user interactivity. An example would be *Faust Music On Line*. Finally the Performance cluster contains the Network Performance and Staged Projects types. Network performances minimize interactivity by a larger audience for the sake of utilizing the skills of experienced performers and Staged Projects contextualize their performance with a kind of libretto or theme. Performances by the Hub can be classified as a Network Performance in this cluster.

Föllmer's taxonomy uses three basic dimensions, Interactivity/Openness, interplay with network characteristics, and Complexity/Flexibility. These are fundamental descriptions of characteristics in network music performances, albeit not an exhaustive collection. In this way it is like Barbosa's taxonomy, using simple dimensions to demonstrate basic relationships. However, Föllmer takes the extra steps to add a list of types to this space, as well as a meta grouping of types, dubbed clusters. This direction makes the spatial order much more ad hoc, and therefore more similar to Hugill's internet music types. Because of these added layers of complexity the taxonomy is actually less useful. It has a large amount of redundancy, but also notable holes. Ideal usage should likely avoid dealing too much with the types and clusters, and instead consider the basic dimensions as the most important aspect of the taxonomy.

2.6.1 Computer Supported Cooperative Work

Computer Supported Cooperative Work (CSCW) is a field that has many similarities to research led in network music, but it also offers worthwhile theories and analyses on how groups can better function using technology. Perhaps the most famous taxonomy in the field is the four square map of groupware types (mentioned earlier) which creates categories as a function of time and place (Johansen, 1988). Grudin and Poltrock suggest an expansion to a nine square model that accounts for the predictability of the time and place of the participants (Grudin & Poltrock, 1991). Others have suggested higher dimensional taxonomies that account for things like group size and member proximity (DeSanctis & Gallupe, 1987; Nunamaker, Dennis, Valacich, Vogel, & George, 1991).

Researchers Paul Dourish and Victoria Bellotti investigated awareness and coordination in shared work spaces, finding that shared feedback (information that is collected and distributed automatically in the background) provides significant advantages for awareness and group efficiency (Dourish & Bellotti, 1992). CSCW usually utilizes qualitative methodologies which can be useful for analyzing network music systems because of the inherit subjectivity involved in music composition and performance. Grudin and Poltrock suggest that the future of CSCW will see a combination of quantitative and qualitative methodologies which can utilize hard data to support and expand on current models (Grudin & Poltrock, 2011). This combinatorial approach could prove useful for future network music studies.

2.7 Conclusion

This chapter began with some ruminations on the definition of network music followed by a short literature review. Academic review for the field is relatively young, resulting in a collection of smaller and more recent commentaries. This was followed by a survey of taxonomies with critique. None of these taxonomies are ideal, demonstrating that the field still has room for the development of useful critical analysis tools. Given that network music is so fundamentally tied to the development of technology, this may be an inevitable fate for any attempt at a taxonomy of network music. These taxonomies will be used in the following chapters to categorize and analyze the author's own network music systems. By considering the previous examples given, a context is provided regarding these new network music systems.

Chapter 3

Liveness In Network Music Performance

3.1 Introduction

This chapter presents a small qualitative survey on liveness in network music; specifically how issues unique to a network ensemble can affect performance dynamics. The survey focuses on a small set of current performers in the network music field, probing their experiences and opinions on how networks might influence a feeling of liveness. The themes in question include perception of the presence of the performance to external parties, their experience as audience members for others' works, as well as their interactions with other members of their ensembles. Network bands and orchestras present a scenario where communication, co-ordination, and timing are important factors to performance. Especially for the case of distributed ensembles over multiple locations, focused engagement in liveness remains a great challenge. Nonetheless, such ensembles may also have unique opportunities to convey musical efforts and their results to audiences, exploring a sense of meaningfulness of presence and action.

3.2 Questionnaire

This survey uses a questionnaire to explore notions of liveness in the context of network music performance. It is a qualitative survey, with questionnaire responses solicited via email, drawing on the experiences and opinions of current practitioners. Actively performing network musicians were chosen because they have a unique perspective on the subject as both performers and as audience members, but also because their technical background allows them to respond in detail and with specifics to the questions posed. Because the surveyed group is small (twenty four requests sent, seven received) and personally known to the research team, an option for non-anonymous response was made available. The default was anonymity, though, and non-anonymity only extends to attribution of direct quotes rather than comparative across group analysis. Ethics review approval was gained from the University of Sussex to run the survey in this manner. Ethics approval can be found in Appendix A.2 on page V. The full questionnaire is listed in Appendix A.1 on page III.

Several topics are covered in the 23 questions, starting with some initial interrogation regarding several technical parameters of the performers' ensembles. Performance practice, communication, visualization, perception, presentation, and anxiety are subsequently probed in the remaining questions with the hope of sparking longer responses. While only seven responded, they represent several actively performing ensembles with a range of experience, make up, location, and approach. Still, given such a small sample size, any results from these purely anecdotal responses can by no means be claimed to be conclusive or statistically significant. Instead, the goal is to establish an initial collection of responses and themes. Further studies in later chapters will explore these themes more formally.

The following is an overview of the ensembles represented, their preferred technology, and their general approach to networking. The average size of the ensembles represented by the respondents is 4.57 (no laptop orchestras were represented). All the respondents claimed to use laptops with Macbooks being the most noted. Software and languages used covered a wide range including SuperCollider, Max/MSP, Pure Data, Processing, C++, Lisp, and Forth. Wireless and ethernet connections were used by all the respondents, but also MIDI and single board custom servers were noted as having been used historically by one. Only two respondents claimed their ensemble performed distributed, with the others stating that they had experience with distribution, yet don't currently perform as such.

3.3 Emerging Themes

Because of the small sample size no formal qualitative content analysis was conducted. Instead, informal cross comparisons of the responses was conducted, linking them to published literature on arising topics. There were several notable themes that emerged after collecting and comparing the questionnaires including the roles that live coding, communication, controllers, and visuals play in network music performance.

3.3.1 Communication

The participants were asked several questions regarding communication, including:

- How does your ensemble communicate with each other during performances?
- How do your ensemble's channels of communication impact on group awareness?
- Do you find this to be successful and how do you compare it to more traditional ensembles using acoustic instruments?

All the ensembles represented utilized some kind of text based chat, and for those ensembles who perform without distributed members the chat system is augmented by visual communication such as gestures or facial expressions, and occasionally vocalized speech. Communication in musical performances is often considered vital and network music is no different (Miell, MacDonald, & Hargreaves, 2005). Unlike other practices though, network music performances often incorporate some kind of projection, and for this very reason all the responses noted that their communication is projected to the audience. These projected communications aim to increase the audience's appreciation for the liveness of a given performance, though not all the responses indicated a preference for text based chat. One response indicated that gestures such as head nodding to the beat or hand movements are preferable. Juan Romero details the differences of the two modes by explaining "It's a trade, gestural communication is faster but simpler, it helps for the synchronicity and to show approval or disapproval and other basic responses. While chatting, the ensemble can write longer ideas and the others can respond to it, complement it and develop it, before it is executed. Chat is much more democratic, but in trade it takes more time."

Curtis McKinney from the network band Glitch Lich lauded text based chat for the ability to foster group awareness by stating "We find it to be successful, and it goes well beyond the traditional means of communication, being able to instantaneously and quietly communicate musical ideas, thoughts, or gestures.” In contrast, Patrick Borgeat amusingly bemoans any effort for communication during a performance: “It’s a general problem that both with chat and visual cues you don’t have any guarantee that all members a) noticed it b) agreed with that. This is the same problem that traditional bands have. If the bass player and the drummer and guitarist all agree by looking at each other that they’ll extend the solo part you’re almost sure that the singer will start singing the chorus nonetheless.”

Communication, in any form, can be a powerful tool in rehearsal and performance, though much of the utility is predicated upon group dynamics and politics (Williamon, 2004). These group dynamics are especially highlighted in improvisatory contexts, where the music can be heavily influenced in real–time (De Jong, 2006). Here was a missed opportunity to directly inquire about the role of improvisation, especially with regards to communication and liveness. One response alludes to the role of improvisation, noting their ability to change their performance in reaction to the audience or the ensemble, but it would have been beneficial to have focused responses on the subject.

3.3.2 Control and Performance

Several of the questions inquired about interfaces and controllers, including:

- What hardware (laptops, phones, kinect, instruments, etc..) does your ensemble use?
- What kinds of software, languages, and environments does your ensemble use? Does everyone use the same collection or is there a mix?
- Given the network music context, in using any controller interface for your music, how does the hardware effect the connection between effort and sonic output?

As mentioned earlier, novel controllers and interfaces have become a common technique among electronic musicians to increase perceived connectivity between effort and output, as well as alter the musician’s relationship to their system. For this reason it is important to understand how they might be used in a network context as well as the opinions of the musicians about their usage. Questionnaire responses varied on their virtues, while none of the ensembles widely incorporated much more than laptops into their standard setup. One respondent explained the lack of controller proliferation because “Our pieces tend to emphasize a group network behavior, and this in turn de-emphasizes individual performance. However, group members are free to use whatever input controls they desire; it’s just that the demands of playing the actual piece and supporting the desired collaboration often preclude concentration on virtuosic, individual performance.” In some ways this group dynamic can be compared to a Javanese Gamelan, where the virtuosity of each performer is superseded by the importance of group cohesion, and where group virtuosity is more important than any given individual (Brinner, 1995).

In contrast, another respondent regretted his group’s dearth of options: “I feel that the one aspect that is lacking for the entire group is getting away from the keyboard and mouse. Granted, it would be difficult and expensive for us all to have the exact same setups but, in solo and group performances, I’ve found that not sitting in front of a laptop is a tremendous boost to the feeling of things being live, no matter what else you may be doing." This sentiment is echoed by some researchers, claiming that the more a performer incorporates the body into live electronic music, the more familiar the performance will be to an audience, and subsequently easier to appreciate (La Rosa, 2008).

Interestingly, when asked to discuss any differences between their solo and group performances, some respondents came back to the subject of control. Tim Perkis highlighted that his solo performances are often very gestural and instrumental, but his network music while still feeling live, was also more composer-like. In contrast Patrick Borgeat pondered his solo performance ambitions, stating "I wouldn't be that much interested in liveness here, but maybe just because I got all the liveness I want with my ensemble."

3.3.3 Live Coding

The participants were asked "Does your ensemble live code during performance? If so, do you show your screens?" Live coding is practiced by many network ensembles and therefore it was important to inquire about the role it plays in the respondents' own ensembles as well as how they consider it to impact their performance and sense of liveness. Three respondents claimed their ensemble live codes, with Tim Perkis of the Hub musing that "live coding only happens if things have gone very, very badly." On a more serious note, Patrick Borgeat of Benoît and the Mandelbrots celebrates the approach by stating "I believe that blank slate live coding is as live as computer music can get." His band mate Juan Romero tempers the sentiment somewhat by saying "It is hard for live coding to make a big show out of it, but for us, the combination of screen displaying, group interaction, communication and our music has had good acceptance as a live act."

Live coding practitioners have claimed that the practice shores up some of the short comings of laptop performance such as the obscurantism of the back of a laptop screen (Ward et al., 2004). By showing their screens they claim to allow the audience to have a better understanding of the intent and efforts of the performers (Wang & Cook, 2004). On the other hand, there is a risk of further obscuring the act, as Alex McLean notes in his Ph.D. thesis on the topic "Most people do not know how to program computers, and many who do will not know the particular language in use by a live coder. So, by projecting screens, do audience members feel included by a gesture of openness, or excluded by a gibberish of code in an obscure language?" (McLean, 2011). One respondent, Juan Romero, also suggests that live coding could have an effect on the interaction of the performers with the audience, stating that "After some concerts people remark how we write our code so fast, and we are fixed on our screens in a kind of 'Tunnel Vision', but then we start being more social and make the music collectively. So this kind of effort is more appreciable during the beginning of our concerts, but also visible throughout the whole performance."

3.3.4 Visual Presentation

Several questions asked specifically about visual presentation, such as:

- Do you use any kind of visual element during performances? If so please describe the presentation?
- How do you feel your ensemble's visual presentation is effected by your networking setup? Does this effect your feeling of connection to the other performers during a performance?
- Does your ensemble live code during performance? If so, do you show your screens?

Laptops (which are used by all the respondents) have had many criticisms with regard to their use as a musical instrument. These criticism include issues such as performer disembodiment, the appearance of an introverted demeanor, lack of social conventions or legacy, minimal physical effort, and a lack of authenticity (Cascone, 2003; Magnusson & Mendieta, 2007). As one respondent eloquently put it, "It's a bit ironic; the performance practice we have embraced in order to make electronic music that is very, very live, can look very, very dead from the audience's perspective." The previous section on live coding addressed some of these issues, and how live coding could possibly help, yet four out of seven respondents did not claim to live code. All of the represented ensembles utilize some form of visual projection during performance. For the live coding band Benoît and the Mandelbrots, this consists of showing their screens and the utilization of some visual effects on the signal. Other groups cited the use of chat displays, visualization of the network and flow of data, and two dimensional and three dimensional graphics as techniques that were employed.

It would have been useful to further inquire if the respondents were making choices with regards to approach and visuals representation in reaction to the previously cited criticisms of laptops. The fact that all the groups have some visual component to their performance beyond simply sitting behind their computers might imply that there are conscientious efforts to mitigate these issues, but the claim cannot be made with the current responses to the questionnaire. Juan Romero does offer some interesting insight with regards to audience opinion on the perception of liveness: "Other people have suggested we should use more light, and other kinds of gimmicks (e.g. using uniforms, walking on stage on Segways, marching while live coding, perform solos, virtuoso laptop air coding, boy band choreographies, etc.) which would help for a live situation, at least make it more interesting (and funny I guess), but our easy set up, and sitting in front on the computer is also acceptable for us, and for the interested audience."

3.3.5 Perceptions of Liveness

The following questions inquired directly regarding liveness:

- Broadly, how do you feel network performance, and in particular your ensemble's approach to network music effects a sense of liveness as a performer?
- If you perform electronic music as a solo performer as well, could you please describe how your solo performance and networked performance work differs with respect to liveness?
- What do you think has worked well for your ensemble, and what do you think has not, in regards to fostering a general sense of liveness during performance?

All the responses indicated that they felt networked performance to be highly engaging. Tim Perkis explains “It’s very personally engaging. Over time I’ve come to realize that the actual interactions and personalities and humor of the performers is the most compelling aspect of the music.” Nevertheless, none of the network musicians felt that networking itself had any effect on liveness (as opposed to engagement), as evidence by this quote from Patrick Borgeat: “I don’t think that networking enhances or diminishes the ‘live factor’ of our performance.” Another response expounds “I don’t think the networked aspect causes an inherent difference in liveness; it much more depends on the priorities of the musicians involved.” Tim Perkis, earlier touting the engagement inherit in networking, only replied “Adversely, probably.” Another response simply stated “I don’t know :(” These answers are interesting because they imply that there might not be a direct correlation between performer engagement in a performance, and a sense of liveness for an audience.

Performance anxiety can have a large effect on some musicians, and it could even be said to be the result of a performance feeling *too* live. With this in mind, musicians were asked specifically about their opinions regarding the effect that networked performances have, if any, on their feelings of anxiety. Responses claimed a range of anxiety during performance, both in networked and non-networked settings. None of the responses claimed to have increased anxiety in networked performances, but several claimed a reduction for various reasons. Patrick Borgeat feels that performing network music moderates several problems that performance anxiety can create. Here he compares instrumental and laptop performances: “My traditional instrument is the saxophone, though I never played it professionally. I haven’t played it for several years but two months ago I played with it again in public. Here I realized that stage anxiety does much more influence my body than my mind: My air and lip pressure trembled and badly influenced my playing. Even if my fingers would tremble in this way I could still type code (maybe a little slower) so here the ‘digitalliness’ of our interface filters out the noise of my anxiety.” The added presence of other musicians was mentioned several times, such as this humorous response: “For me the slightly higher degree of anonymity in a laptop ensemble, mostly due to the relative difficulty to discern which member of the ensemble just exploded the filter, really seems to have an effect on the level of stress involved.”

Performing music with computers introduces the possibility for technical problems to impede the performance. One respondent noted the improvement of software over the years: “Back in the day, I remember a great deal of anxiety about technological failure, and for good reason! Now that the tech is much more stable, that is less of a concern. Having five noisy bandmates can cover a host of problems, as long as the whole network doesn’t fail. I don’t think I’ve ever played a concert that didn’t generate a great deal of excitement for me. It’s why I perform, after all.” Tim Perkis describes how the Hub copes with these issues: “Our music is complex and difficult enough to perform that there is often at least one person not working at any one moment, so there is little anxiety about that, we just expect it.” Glitch Lich, Curtis McKinney’s laptop band, performs distributed, and he described the effect on anxiety by saying that “Network music while playing dislocated and away from the actual audience severely diminishes this, but it also serves to somewhat dull the adrenaline rush and immediate sense of contentment with a well done performance." Other responses supported the sentiment that distribution dampens the adverse effects of nervousness.

3.3.6 Ensemble Structure

Ensemble structure was the subject of the following questions:

- Does your ensemble perform with members physical distributed among several locations?
- How does your ensemble's structure and approach influence your sense of involvement in performance?
- If your ensemble performs physically distributed, do you feel this effects a sense of liveness or connectivity?

Some participants suggested that fostering individuality in the network with regards to audio production and reaction to network activity has been vital towards creating more lively and interesting performances. A respondent explains, "Our design and performance practice, from the very early days, has concentrated on the emergent behavior of the network / ensemble, and I think this has led to consistently surprising and lively performance. One interesting thing that we've found is that it is very important that each member realize each piece specification in their own individual manner – sharing of piece code tends to homogenize and 'deaden' the resulting perfomance." On the other hand, Juan Romero suggested that there is an advantage to a symmetrical ensemble, stating, "We had a mix in our first performance (Max + SuperCollider) but then all the members of the band recognized that having all SuperCollider would be better to make a framework for staying in sync and sharing data. Also for learning from each other." One respondent felt that having a shared visual interface helped foster performer interactivity as well as increase audience understanding, explaining "If we were all doing something completely different and just trying to make it work together sonically, I don't feel like anyone, including us, would feel as connected to what we are doing as when we can all see and interact with the same environment."

Ensemble distribution is one of the unique possibilities afforded to networked ensembles, but only one of musician claimed that there was any merit in this structure. Responding to an inquiry regarding multi-site distribution and liveness, Curtis McKinney explains "It certainly affects it, though it is not all negative. It's a different performance medium, with different possibilities and restrictions." Others had much more negative opinions. Responding to the same question, another musician plainly states "It affects it quite negatively; this has been our invariant experience. Nothing (in current technology) can approach the moment-to-moment live interaction to be enjoyed with musicians sitting in the same room together." Tim Perkis agreed, saying "I don't find multi-location playing very interesting. it seems like a gimmick that offers no particular advantage in any way." Another had a more nuanced opinion, suggesting "I think that remote performers need some kind of visual presence (by video projection or with an avatar in the visuals), otherwise you don't really recognize them as a performer who has influence over the current piece."

3.4 Summary

Several important themes emerged from the responses of the network musicians who participated in the survey. Generally the responses were concerned with certain fundamental themes including communication, control, visual representation, individuality, and group behavior. All the ensembles represented by participants in the survey utilized some kind of communication during performances, but not all the responses indicated that this communication was mediated by their interface. The respondents who represented ensembles with distributed members all used communication channels via their network music interfaces. This may indicate that while some communication channel may not be a requirement for a successful network music interface, that it may be a requirement for a successful performance. Additionally all the respondents recognized the importance of communication in collaborative rehearsals and performances.

Another important theme is the reported lack of use of input controllers. Although one response bemoaned the lack of variety, most responses indicated that their ensemble emphasized group dynamics over individual virtuosity. This fact may also be a consequence of the fact that several respondents used live coding in their performances.

All the participants indicated that they utilize some form of visual component in their performances. These visual components serve to make the performance more exciting or interesting for an audience. Most of the groups employed the strategy of "Show us your screens", and projected the direct interface of the performers. Others used some kind of external visualization program that ran parallel to the music software, listening and reacting, but not participating. Network music interfaces may benefit from these visual approaches, but there was no consensus in the responses.

Some participants reported working in ensembles that don't user a single music interface, but instead an array of interfaces. In these groups individuals defined their own behavior according to a predefined set of communication protocols. By contrast other groups reported using a single or unified interface for the entire ensemble.

Chapter 4

Yig, The Father of Serpents

4.1 Introduction

Yig, the Father of Serpents is an application for creating and manipulating feedback matrices in real-time over the internet. The name was chosen as a reference to H.P. Lovecraft's *The Curse of Yig* (Z.B. Bishop and H.P. Lovecraft, 1953) as well as the ancient symbol of the ouroboros[1], both as metaphors for feedback and recursion. A video of a Glitch Lich performing with *Yig* at the 2012 Network Music Festival in Birmingham, UK can be found at `https://vimeo.com/50402941`. The program was created with the philosophy that performers don't require identical experiences to have a successful performance. Research in the field of network music has historically attempted to create perfect reproductions for performers in separate locations in an attempt to unite the clients within one absolute and real performance (Barbosa, 2003; Weinberg, 2005; Schroeder et al., 2007). By contrast Yig is designed as a system that, while using techniques for low latency and high levels of synchronicity, has naturally divergent sonic results in the network.

Yig is an Open Sound Control (OSC) client to the SuperCollider scsynth server utilizing a patch based visual interface. Synths running on scsynth are displayed as circular objects, with parameter modulation via object rotation, and collision based cable creation. When one of these circular synth objects is dragged so that it overlaps with another circular synth object a connection is created from the output of the source synth to the input of the target synth. See figure 4.3 for an example of three synth objects linked using these kind of cable connections. These synths use their input audio to modulate audio or control signals, as well as to add to their own output. These connections can be cyclical, allowing for feedback networks to be created using several synths and connections. Control information that defines object states is networked using a server based OSC synchronization system. Stochastic Unit Generators (UGens) within synth definitions are combined with input analysis, such as pitch and onset detection, inside feedback matrices creating complex dependencies and chaotic behavior. While the performers' object states may be syntactically identical, they can in fact be sonically divergent. This is because of the codependencies of the synths and the combined minute differences in timing and synchronization. Yig does not try to combat this, but in fact embraces these differences as a fundamental concept of network performance.

Performers do not perform with each other, but along side each other in parallel, yet fundamentally different, experiences. None of the sub-performances in the web of the network is any more real than the others. The most considerable efforts to ensure an identical reproduction will never produce true copies. Even if it were possible, the differences in presentation, venue, sound systems, audience presence and many other uncountable details creates a fractured image of the concert with no singular source.

Composing, performing, and improvising in a fractured ensemble is a unique opportunity for the network ensemble. In 1987, one of the first multi-site performances featured the pioneer network band *the Hub* performing in two spaces simultaneously (Brown & Bischoff, 2002). This concert presents the six member ensemble as bifurcated sub ensembles networking locally. The two trios communicated with each other via a phone line

[1]The Ouroboros is a symbol depicting a snake eating its own tail.

modem, however only control data was shared. The ensemble was informationally joined but acoustically divided.

By allowing some openness into the network, unique perspectives can flourish and decisions can have unanticipated results. Incidental findings have shown Yig to be a viable choice to compose and improvise music for laptop bands. Yig provides a robust framework for network performances over extremely large distances while preserving low latencies and high levels of syntactic synchronicity. As an open source instrument, Yig provides laptop musicians with the ability to create unique music or to reuse the codebase (McKinney, 2012) for their own projects.

4.2 Design Philosophy

The core design philosophy is to have an instrument that allows for collaboration between players and the system itself, where choices are meaningful, but also produces unexpected results. Yig was designed for Glitch Lich performances as well as any other interested parties to use. As mentioned in Chapter 2, Glitch Lich performances have a certain set of requirements technically, and aesthetically. First is that the interfaces must facilitate distributed performances. In Yig all nodes in the physical network render the full output of the audio, allowing for performances from multiple locations at the same time. Next the system must support some kind of communication, in the case of Yig, this is a simple chat system.

Next, the system must emulate some kind of virtual space. Yig uses a simple two dimensional space for the collaboration. The spatial and collision based design allows for this simple virtual environment to facilitate constantly changing group dynamics. Additionally, this virtual space must have some kind of visual component for Glitch Lich performances. Yig has seen several performances where the interface was projected directly to the audience, but the interface was not designed to fit the visual aesthetic of the band. Instead, performances would often use a second program written in C++ using Cinder and OpenGL to render a highly stylized version of the Yig program state for the audience to see. A picture of a Glitch Lich performance using this custom visualization program can be found in Figure 4.1. Finally Glitch Lich requires some form of autonomy in the system. Yig uses complex feedback networks to create behaviors that change and evolve without direct input.

In addition to the requirements for a Glitch Lich performance, Yig was designed to create fast changing chaotic and noisy music. The knob–like design of the main synth objects are intended to allow for a simple but efficient input mechanism for fast changes. The collision based connections are also designed to facilitate fast exploration and experimentation, allowing for the feedback network to be reconfigured instantly just by dragging objects across the screen.

The design philosophy outlined above produces an interfaces with inherit limitations. The 2D space allows for easy viewing of the entire virtual space, but also forces this homogenous view onto the entire group. The knob–like controls are quick and easy, but

Figure 4.1: Glitch Lich performance at the Network Music Festival using Yig with an external visualization program.

also lack precision. This is especially apparent when the knobs control parameters that are directly mapped to pitch. Having such a heavy reliance on autonomy can produce undesirable results at times. The feedback networks in Yig can resist change at times, making performances occasionally less dynamic.

The next section will describe various features as well as the development process. This is followed by an analysis of the system using Andrew Hugill's internet music types as well as Thor Magnusson's epistemic dimension space. Subsequently, the networking behind Yig is explained and the concept of divergence in network music is discussed. Finally, the conclusion establishes several potential improvements for the system and ideas for using divergence in networks.

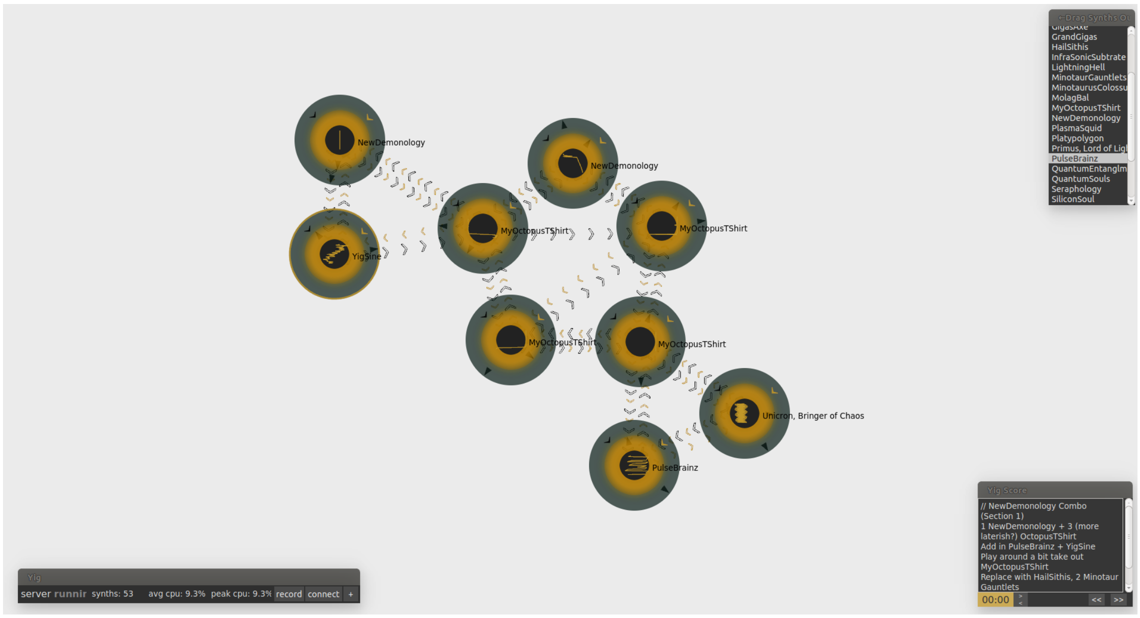
4.3 The Interface

Yig is an instrument designed for real-time performance. Because of this, many design decisions seek to streamline actions necessary to create and manipulate sound while preserving depth and flexibility. Originally the cable creation was envisioned to be similar to Max/MSP or Pure Data where the input and output nodes are connected through explicit mouse clicks (Cycling '74, 2014; Chung, 2013). However, this approach was abandoned very quickly for something more similar to the mobile platform version of the ReacTable, which has proximity based connections, for the sake of increased agility (Jordà, 2009). While there are some similarities to the ReacTable graphical user interface (GUI), Yig is not just an imitation. The interface was designed with performance over a network as the main feature.

4.3.1 Features

Making music with Yig requires four main actions: Synth creation, synth deletion, cable creation, and cable deletion. Synth instances are created by dragging items from the synth menu on to the playing area. Synths are represented as concentric circles with an animated oscilloscope in the middle. Each synth has two modulatable parameters, two audio feedback inputs and one audio feedback output. By default all synths route audio directly to the main outputs in addition to the feedback output which allows for complex chains of non-linear structures.

Synths can be linked together through colliding one synth over one of the input nodes of another synth. Collision detection automatically generates an animated cabled connection showing the channel(s) of the connection as well as the flow of audio, which can be bi-directional. When two synths are dragged past a predetermined length, the cables automatically detach. This is similar to the iPhone mobile application Jasuto, although in Yig only disconnections are proximity based while cables are created during collisions (Wolfe & Wolfe, 2014). It is important that cable creation and deletion is easy and intuitive in order to allow for fast manipulations of the audio connections. Such fast changes in routing are difficult to reproduce with physical equipment.



```
// NewDemonology Combo
(Section 1)
1 NewDemonology + 3 (more
laterish?) OctopusTShirt
Add in PulseBrainz + YigSine
Play around a bit take out
MyOctopusTShirt
Replace with HailSithis, 2 Minotaur
Gauntlets
```

```
server runnir synths: 53   avg cpu: 9.3%  peak cpu: 9.3%
```

Figure 4.2: Screen shot of Yig, the Father of Serpents.

A server window floats on the bottom of the screen showing the current status of scsynth, including peak and average cpu usage and the number of running synths. There are also the recording, connect, and option buttons. When a user clicks the connect button, given that they have properly set up the OSCthulhu client, the server window automatically extends itself to include a current list of users online and a chat window is created in the bottom right corner of the screen. Having built in communication is paramount to successful network performances. When connected to the network, other users' cursors and selected synths are visible. These visual cues provide additional modes for communication during a performance.

4.3.2 GUI Development

Yig is written in C++ with heavy reliance on the Qt framework (Summerfield, 2010). Qt was chosen because of its strong cross platform GUI application programming interface (API) which greatly simplifies and streamlines the development process. Creating Yig with Qt was found to be highly flexible because the API allows for highly encapsulated design. This encapsulation was beneficial to our object oriented approach to GUI programming by allowing the interface between widgets and GUI elements such as buttons and displays, to operate in a high level manner, reducing complexity and increasing robustness.

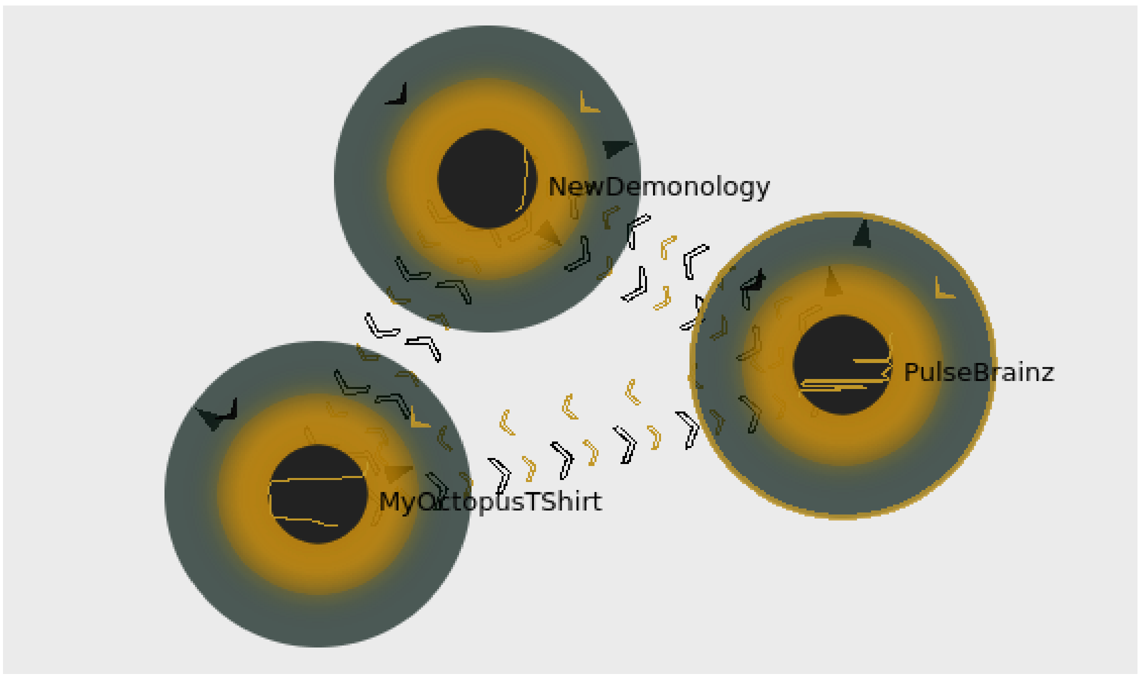


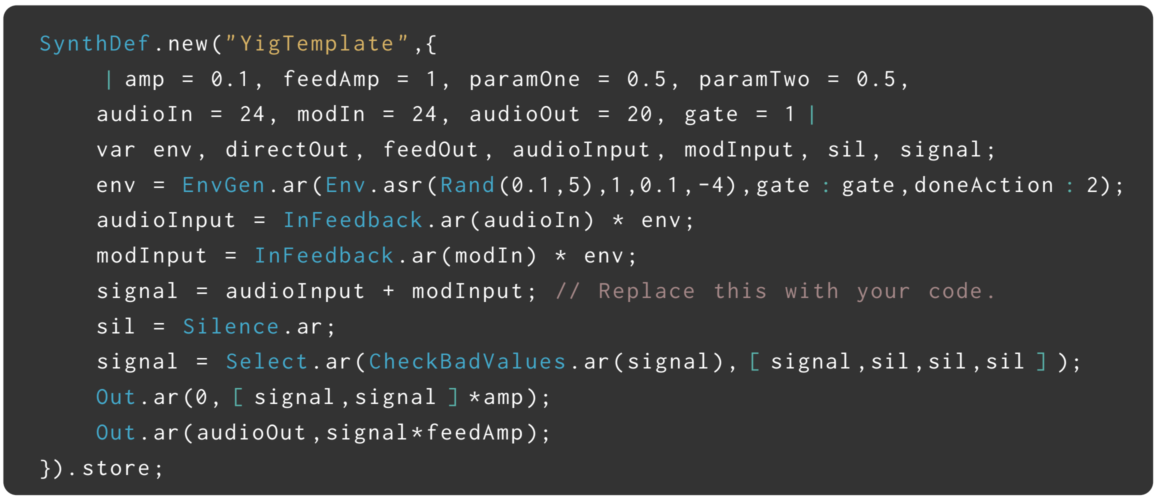
Figure 4.3: Three Yig synth objects connected to eachother.

Qt proved invaluable during development, however the early decision to use the QGraphicsView and QGraphicsScene paradigm for the creation and manipulation of graphics elements proved to be a mistake. QGraphicsView enables incredibly easy implementations of drawable items with the convenient inclusion of important features such as mouse and keyboard interfacing and collision detection. These features made early development much easier, however the performance of the system was much worse than anticipated. With only six synth objects, which are very simple circle graphics, the system idled at 20 percent CPU usage on a two year old Macbook Pro. When the synths were moved, triggering redrawing of the items and collision detection, the CPU usage could jump dramatically, upwards to 70 percent. Many approaches were attempted to increase performance, such as multithreading and pre-caching reusable graphics items, however the results were still slower than preferable. This required addressing as initial experiments with the system showed that it was easy for a player to create and manipualte many of these synth objects at once. With multiple performers controlling multiple synths this could render the interface sluggish and unresponsive.

When work began on animation for the oscilloscopes and cable connections, an OpenGL approach was chosen over extending the use of QGraphicsView. This proved beneficial, however only the animated graphics currently benefit from it. In retrospect, OpenGL should have been used for all of the graphics items, which would have required some extra programming for the previously mentioned features, however the results would be a faster interface that takes advantage of the computer's GPU, freeing up crucial CPU processing power for other important tasks such as audio rendering and networking.

4.3.3 Synth Definition Development

At its core, Yig is an scsynth client. However, scsynth does not run as a separate process, but instead, using libscsynth, operates internally within the same process. This configuration allows for the seamless retrieval of audio buses for oscilloscope animation, but also serves to maintain a singular package for distribution. Libscsynth was chosen over other options, including a custom audio engine, because it is open source, lightweight, robust, and high quality (McCartney et al., 2016). Most importantly, Yig is intended to be an instrument that allows for easy extension. The synth def plugin architecture provided by scsynth allows for high level synth creation by users without the need to recompile the program. Any user with a working knowledge of SuperCollider based synthesis can simply modify and extend the Yig synth definition template (found in figure 4.4) to create new sounds.



```
SynthDef.new("YigTemplate",{
      | amp = 0.1, feedAmp = 1, paramOne = 0.5, paramTwo = 0.5,
    audioIn = 24, modIn = 24, audioOut = 20, gate = 1 |
    var env, directOut, feedOut, audioInput, modInput, sil, signal;
    env = EnvGen.ar(Env.asr(Rand(0.1,5),1,0.1,-4),gate : gate,doneAction : 2);
    audioInput = InFeedback.ar(audioIn) * env;
    modInput = InFeedback.ar(modIn) * env;
    signal = audioInput + modInput; // Replace this with your code.
    sil = Silence.ar;
    signal = Select.ar(CheckBadValues.ar(signal), [ signal,sil,sil,sil ] );
    Out.ar(0, [ signal,signal ] *amp);
    Out.ar(audioOut,signal*feedAmp);
}).store;
```

Figure 4.4: Yig's basic SuperCollider synth definition template

When creating new synths for Yig, some considerations are helpful. No synth exists in a vacuum, instead the synths behave differently in varying feedback network configurations.

For this reason, it is important to thoroughly explore the range of sounds before settling on a final version. Often, unexpected and exciting, or disappointing results can appear. Also, because Yig is so focused on feedback, loss of sound can be an issue. It is highly recommended to use the CheckBadValues UGen, which is used in most synths to prevent unstable or malformed values from destroying a performance. The CheckBadValues UGen tests for infinity, NaN (not a number), and denormals. Used in combination with the Select UGen, a different output (Silence is used above) can be chosen when the signal becomes malformed. Yig heavily uses feedback between large numbers of synths, each of which use and modify their input signals in many different ways. This technique ensures that if a signal becomes unstable then there are some safety measures that could help rein it in. Currently there is no global tempo synchronization. In some performances demand rate UGens have been used to create sequenced rhythms. This technique cannot guarantee a synchronized pulse between instances without feeding this pulse through a specific bus. An example of one of the synths that was used in performances of Yig can be found in Appendix B.5 on page XX.

The design decision to focus on feedback is deliberate. The goal is to have a system that not only allows for the performers to collaborate with each other, but to also collaborate with Yig itself. A good Yig performance will utilize feedback networks to curate evolving and autonomous emergent behavior. With that in mind, these networks should not just be created and left alone. Instead, the feedback should be treated almost like another performer. There should be listening, reactions, counterpoint, and interruptions when playing with this emergent phenomenon. Yig performances work well when the system produces unexpected behavior, prompting reactions from the performers who make changes, subsequently prompting reactions from the system itself as well as the other performers, and on. These feedback networks do not always produce a wildly evolving chaotic maelstrom. Indeed, during performances Yig can be stubbornly monotonous or even annoying. This is also part of the design, as playing with Yig should be an exercise in challenging the performers' exceptions, even when those exceptions anticipate the unexpected. This design makes performance interesting and challenging, but it also means that some performances are more successful than others in a way that can be outside of the hands of the performers.

4.4 Networking

The networking for Yig was designed to ensure functionality over long distances while using slow and even intermittent internet connections. To accomplish this, Yig uses low bandwidth user datagram protocol (UDP) based OSC messages to ensure high speed transfer. Audio is rendered locally on each network node's computer and no audio is directly networked. Instead the state information, such as the synth objects and their parameters, is broadcasted across the network. This approach greatly increases the speed of the networking while ensuring that variance in the internet connection does not have a great immediate impact on the resulting output of the performance. When networking audio directly, if the internet connection is lost for a period of time or speed drops dramatically, the audio output of the system is directly and recognizably effected. When networking state information these variances will effect the performance, but in a less direct manner. Instead of audio jitter or loss, the local state of the machine may begin to diverge from the other nodes, however the audio fidelity of the system is not effected and often these events go unnoticed by audience members and even the performers themselves.

4.4.1 Synchronization

Because the networking in Yig uses control information instead of audio the issue of synchronicity becomes paramount. Without any steps taken to address dropped UDP packets, divergence in the system would steadily increase over time, rendering the networking completely unreliable. To address this issue Yig uses the OSCthulhu server and client system for OSC synchronization.

OSCthulhu is a synchronization system that attempts to create a program agnostic interface for networked variable serialization and synchronization (McKinney & McKinney, 2012). It should be noted that while the author has contributed to OSCthulhu, Curtis McKinney is the main contributor, and that system is not a part of this thesis. The benefits of the system are a simple API and a server based synchronization scheme that allows developers to bypass many of the issues in synchronization systems development. OSC messages are sent first to a local OSCthulhu client, altering local state information. Next the client messages the OSCthulhu server directly. Finally, the server broadcasts the message to the all participants. Because the server runs on a rented machine on the open internet, it bypasses many of the problems that other OSC systems such as OSCgroups have with routers and firewalls blocking traffic. A synchronization cycle every 1000 milliseconds ensures state mirroring across the network so that packet loss is usually adjusted for within one second.

The networking code in Yig is more similar to a video game than a typical network performance system. This is because in OSCthulhu messages are never sent directly to other players. All traffic passes through the server, and the server is the fundamental governing body of the ensemble, similar to *the Hub*. Objects known as *sync args* are created on the server that represent a network entity. This could be anything from a synth or cable, to a cursor. The entire ensemble is updated when a *set sync arg* message is sent to the server. If a client misses a synchronization cycle, they will be updated on the next pass, ensuring network symmetry.

This approach has been tested time and again not only in OSCthulhu, but in the video game industry as well (Sweeney, 1999). When utilizing the API there are some special considerations. Some messages, such as synth creation and deletion messages in Yig, can cause local volatility if the message is not received by the server. When using UDP it is important to assume that if messages are sent, it is likely that they may not be received at some point. The ramifications of that can be enormous, but OSCthulhu has a tool to ensure stability. These messages are sent out before interpretation so that they update the server first before updating the local machine. The local machine will only be updated when the server sends back a synchronization message. Missed messages from the server are far less severe than missed messages to the server.

As an example, consider the deletion of a synth. If the synth were deleted locally and the message never reached the server, the ensemble would still have representations of that synth on their system. At this point the local player has no way of bringing themselves back into step with the ensemble without another player serendipitously deleting it. However, when first sending the server these kinds of mission critical messages, the problem will always automatically resolve itself. There are three possible outcomes when sending messages directly to the server before interpreting them locally. First, the server never receives the message. Second the server receives the message, but the local client does not receive the synchronization reply. Finally the server and the client both receive their messages.

Back to the example of the deleted synth. If the deletion message is never received by the server the synth simply never changes on the local machine and is still available to make another deletion attempt. If the message is received by the server, but the synchronization isn't received by the client, the client will simply be updated by the next synchronization cycle. Finally if both messages are received, functionality is as predicted.

This approach is not always appropriate however. Often you have objects where speed is more important than accuracy. In Yig, setting the value of one of a synth's parameters will occur locally before the message is sent to the network. This is because the synths are set by mouse control. This type of gestural information spans several packets and is therefore more resilient to packet loss. When the synth is changed, a smooth transition occurs locally and the gesture is preserved. If part of the gesture is dropped on the way to the server, the whole does not suffer greatly for it.

4.4.2 Divergence in the Network

Using OSCthulhu synchronization ensures that the state of Yig is tightly mirrored between network nodes, however this does not guarantee that the audio output of the system itself is identical. Small differences in timing can greatly effect the audio and control feedback chains within Yig, in turn creating sometimes drastically different results. A realignment from a dropped packet could take longer than a second with very bad connections. In that time a chaotic ugen will continue to calculate output using differing states between the users as well as different audio input from the feedback chain. There is no opportunity to truly synchronize the audio output of these systems between users. This prompts us to ask a few questions, though there are no perfect answers.

First, why not just network the audio, even given the requirement of high quality network connections? Broadband connections will continue to improve, offering network music faster and more stable infrastructure. What is considered a high quality research connection could become typical for internet users in the future. Perhaps designing systems with faster connections in mind will ensure their relevancy and value? However, simply networking audio between nodes may only serve to reinforce traditional methods of performance.

Networking offers new possibilities for music composition and performance beyond simply allowing performers to be further apart during performance or rehearsal. Regardless of the distance, a system like Yig grants a group of performers the ability to collaborate that simply sending an audio signal cannot capture. The important part here is not the network connection itself, but rather the interactions of the group.

It would be possible to design a system where a server renders audio and sends streams to the performers. However the audio quality would suffer due to jitter in the network and a loss of internet would absolutely terminate the performance. In contrast, Yig can operate independently of the network allowing for continued performance in the event of connection loss. In the server audio scenario latency would also be an issue because it would restrict the ability for the performers create fluid gestures and hear immediate results. Furthermore, the network would become congested, potentially causing control packets to be lost more often. By using locally rendered audio, the trade off is that the node renderings might not be identical, but control is fast and fluid and the network is much more robust.

Why attempt to network naturally divergent systems? Given that these systems present difficulties for the technology, why shouldn't they just be avoided? Where something can be done musically, often someone will do it; composers and performers like to experiment, and often like to break things in the process. Is divergence in networking bad? Network music attempts to create a seamless and invisible framework for the users. Any noticeable artifact of the system is an irregularity and steps should be taken to eliminate them. However, this philosophy fails to capture the truly interesting aspects of network music, which are the defining features of the entire approach. There is a disconnection between users and an artificial attempt to traverse it. This could be seen as a problem, but the design philosophy was to engage it as a unique resource.

4.5 Categorization

It may be useful to understand Yig within the context of previous attempts to classify and analyze similar instruments. Barbosa's network music classifications plot a networked piece or system along a two dimensional space that describes interaction (synchronous/asynchronous) and location (local and remote). The space is divided in two four quadrants where different types of music or music systems can occupy one or more of these quadrants. First addressing the 'interaction' dimension, Yig is designed to be a synchronous instrument. From the server and client architecture to the lack of any kind of save feature (beyond simple recording), Yig is focused on live interactivity between players. The system could be run on a central computer or server where clients connect and leave behind feedback network lattices for other visitors to interact with at a later time. This is only conjecture and has not been attempted to date. Instead. Yig is plotted in the synchronous half of the space.

For 'Location' Yig can safely be categorized as both 'Local' and 'Remote' because the entire system was designed to facilitate both types of work. One consideration is that Yig can feature large amounts of divergence in behavior across the network, and for this reason it may be preferable to use a single computer for audio output, but this is a preference to the collaborators as the divergence may be considered to be an interesting layer added to the music. Barbosa labels systems that are synchronous but support both local and remote collaborations as 'Shared Sonic Environments' which is especially fitting for Yig as there is in fact a visual shared environment and interface to accompany the shared sonic environment.

As shown previously in table 2.1 on page 35, Andrew Hugill's internet music taxonomy provides five categories that broadly define several established approaches to creating music using the internet. Music created with Yig falls well into the category of *Music That Is Created or Performed in Virtual Environments, or Uses Virtual Instruments*. Yig provides a single synchronized environment within which users collaborate and perform. Given that the categories are broad, an argument could be made for a classification within *Music That Uses the Network to Connect Physical Spaces or Instruments* or *Music That Uses the Internet to Enable Collaborative Composition or Performance*. However, neither of these account well for the sort of interactions that Yig establishes through its usage. Yig provides a singular world within which to perform and does not attempt bridge multiple distinct locations.

Gil Weinberg's various network configurations are a veritable zoo of possibilities. While many of the theorized configurations could make for an interesting piece, Yig uses one of the simplest, the 'Flower'. The 'Flower' is a simple client and server hub configuration where clients don't communicate directly and all traffic is routed through the central hub. This particular configuration is a matter of pragmatism as this method is a traditional and stable approach to multi-user systems, and is still used in many modern video games. While Yig's technical networking infrastructure is not very unique, the feedback networks and lattices that users creates within the virtualized environments can form some of the more exotic configurations presented by Barbosa. In this way the physical networking is serving almost like an emulation layer for the virtualized connections that players make between running synths. This is similar to how higher level languages such as Lua and Python virtualize more complex behaviors and abstractions, while being calculated and executed in a lower level virtual machine, and ultimately executed as a series of machine code instructions by the CPU. This is because Yig creates a virtual network of synth objects and feedback connections that are simulated on each of the machines. The physical computer network is required to maintain the synchronization of this virtual network on the various machines in the physical network. The network that the players are dealing with directly is the virtual network in Yig, not the physical machine network.

Föllmer's spatial order of the twelve types of Net music is a three dimensional space plotting 'Interactivity/Openness', 'Interplay with Network Characteristics', and 'Complexity/Flexibility', each with a rating from one to five. For 'Interactivity/Openness' Yig can be rated moderately at a 3 as the system is very open and encourages chaotic exploration, yet this openness is only afforded to the performers, and general audiences cannot easily interact without downloading and installing the software, along with running their own server instance. Yig can also be plotted about a three in the 'Interplay with network characteristics' dimension. A large part of the design of Yig is to explore the inherit divergence caused by network characteristics, yet these divergences are not the sole focus of performances with the system like some other network pieces or systems. For 'Complexity/Flexibility' Yig again can be rated a three as the system can facilitate some rather complex and sophisticated behavior, yet much of that behavior is defined ahead of time in the synth defnitions that are precompiled from SuperCollider. There is a deep resource for complexity, but the particular infrastructure mutes the realistic utility of it in some cases. A rating of threes plots Yig directly in the middle of the space, putting it firmly in the K type: 'Network Performances' which is a part of the V cluster for 'performances'. This plotting lines up well with both the intended design for Yig as well as the history of performances Yig has seen throughout the past several years.

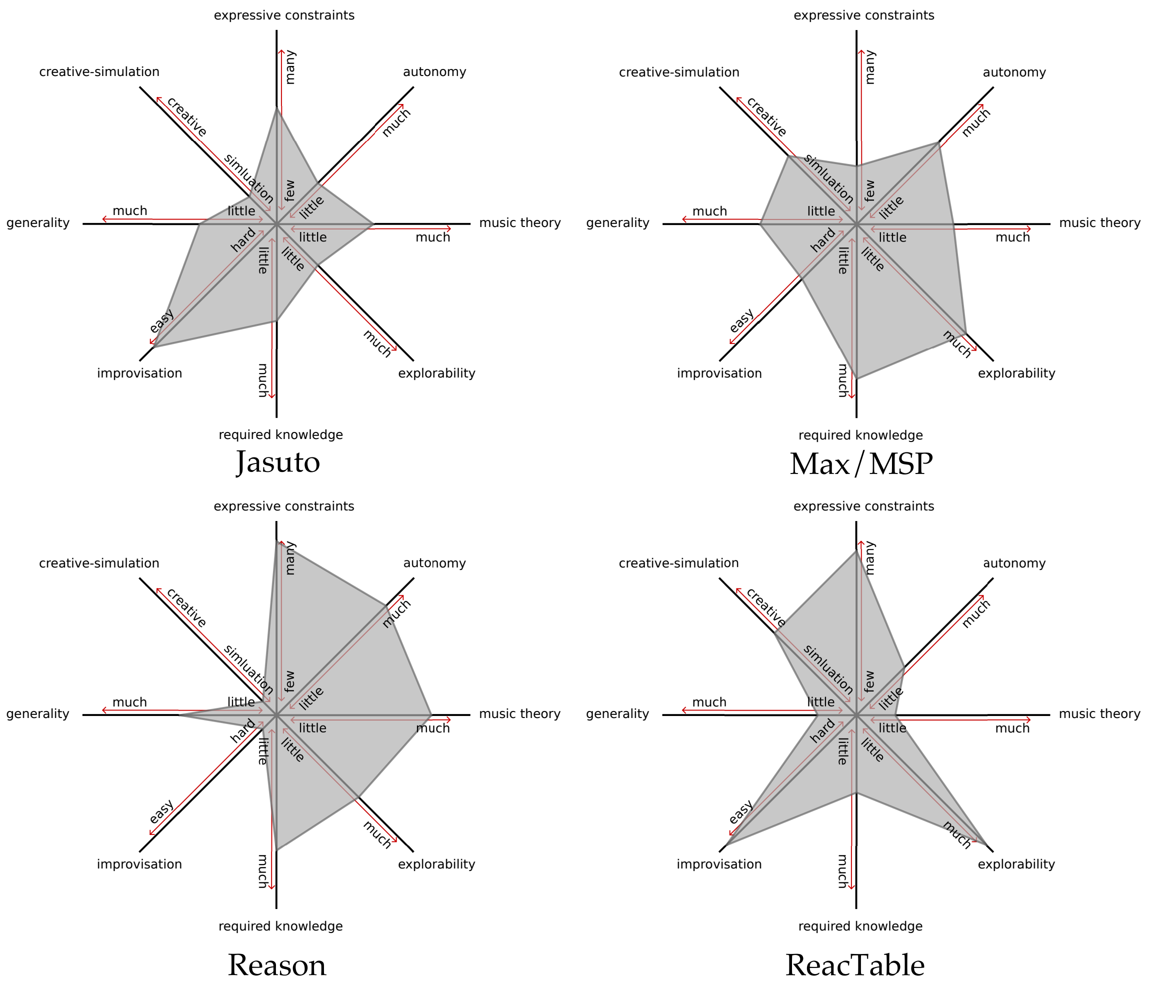


Figure 4.5: Jasuto, Max/MSP, Reason, and the ReacTable plotted in Magnusson's epistemic dimension space.

Using Magnusson's epistemic dimension space, Yig's characteristics as a performance instrument can be analyzed and graphed, providing insight as well as an opportunity for comparisons to other instruments. Parameters are mapped along 8 axes creating a polygonal field that describes the overall distribution of specific qualities. Expressive constraints, autonomy, music theory, exportability, required foreknowledge, improvisation, generality, and creative–simulation continuums are plotted for the software and comparisons can be

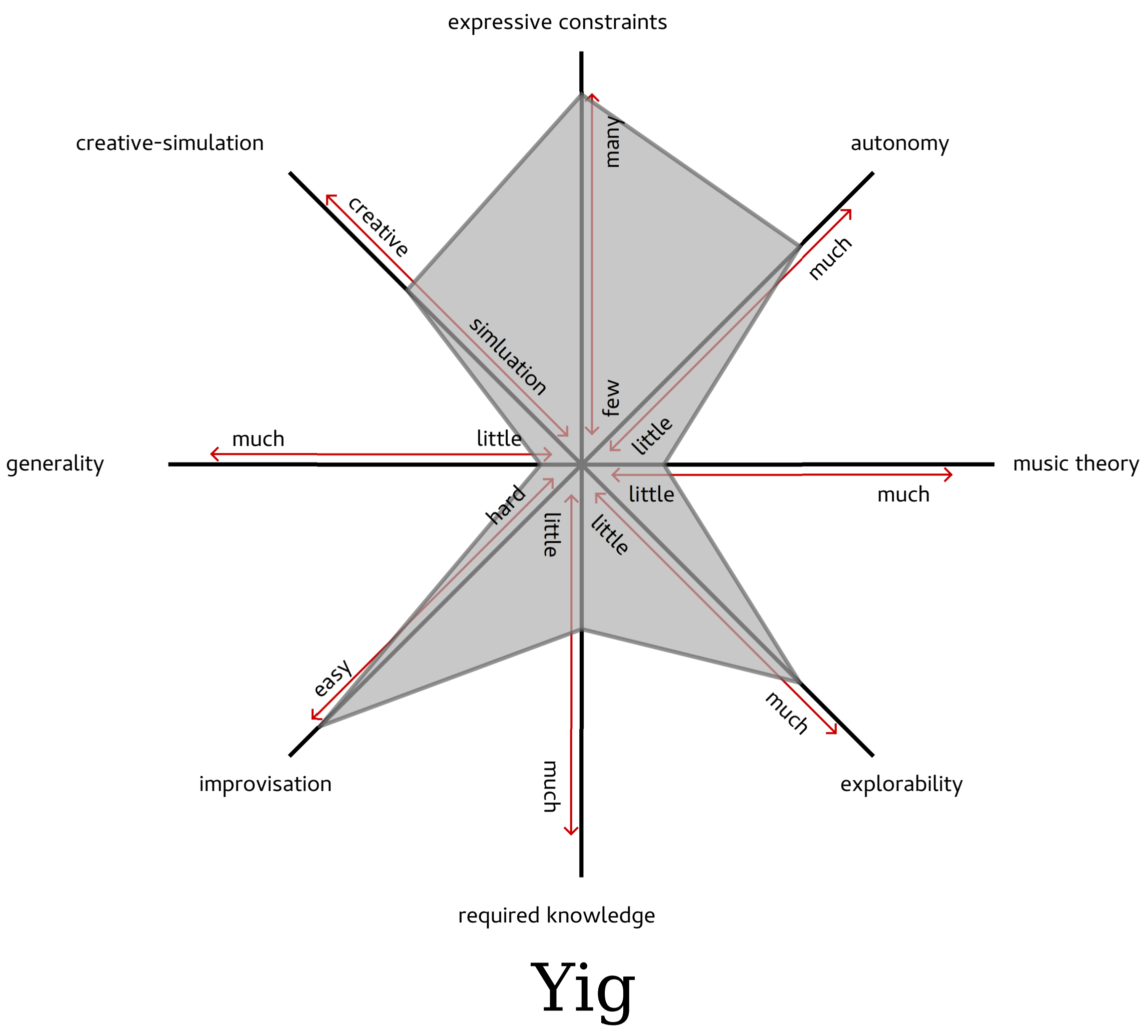


Figure 4.6: Yig in the epistemic dimension space.

made to other software. Once mapped in the dimension space, underlying design implications are revealed as exaggerated contours. Yig is shown to emphasize the ease of improvisation, while deemphasizing the required knowledge to play the instrument. The performer is not required to know any music theory and does not expect any other special skill beyond the ability to start and run the networking client correctly.

The instrument allows for high levels of exportability and autonomy. Creating complex feedback chains affords the user with many opportunities for detailed experimentation. However the simplification of the system also serves as an expressive constraint. There is some allowance for direct control over the music, but the full dexterity of the human hand is reduced to a one dimensional rotation.

4.6 Audio Divergence Test

To demonstrate how easily two nodes with identical states can diverge, a small case study was arranged using Yig with two participants, one in Brighton and the other in London. Both nodes initialized a recording and created a small feedback network. Once the synths were connected no further changes were made. The recording was allowed to run untouched for five minutes.

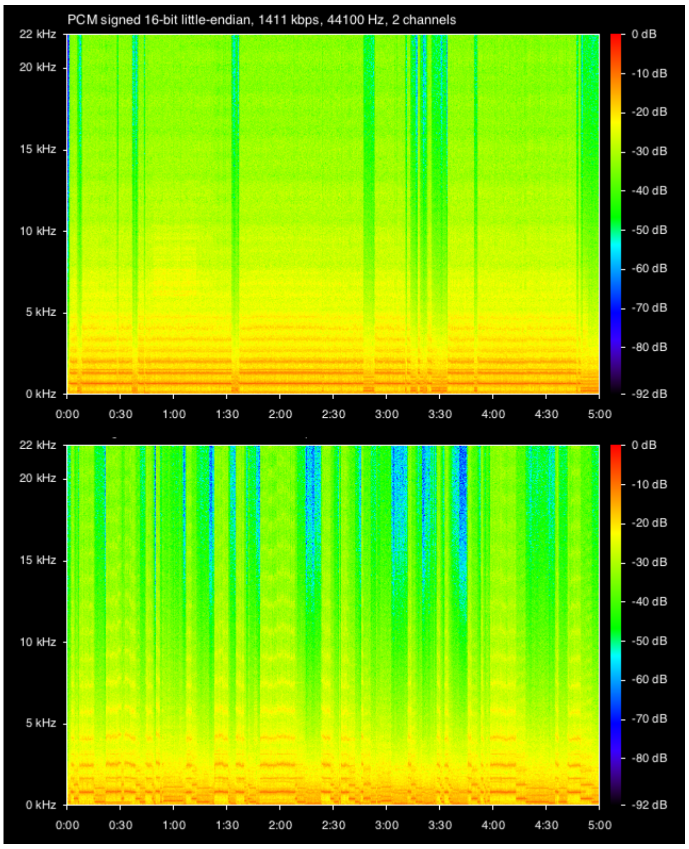


Figure 4.7: Comparison of recordings from two nodes with identical states.

The spectrogram analysis of the two recordings is provided here in Figure 2. The two recordings show some clear differences: both recordings have similarly noisy textures, but the punctuations do not align in number or placement, and there are difference in their harmonic content. The second recording is more punctuated and the harmonics move slightly more than the first. Additionally, the first recording consistently fills the entire spectrum while the second has several gaps in the high end of the range. What is not completely captured in the spectrograms is that while the harmonic content and rhythm differs between the two, the texture of both recordings is still quite similar.

4.7 Design Summary and Future Work

Now that the fundamentals of the program have been established more work can be done to improve upon the current model. There is much room for efficiency improvements through multithreading the collision detection as well as using more advanced OpenGL techniques such as display lists or virtual buffer objects (VBOs) for the graphics animation. A system for organizing, traversing, and switching between scores will be greatly beneficial to organizing rehearsals and performances. Beat based synchronization of demand rate synths will be a useful feature, but will require more changes to the fundamental synth creation process, and potentially to the Yig synth def template.

Yig demonstrates that divergence within a network can be embraced, though there is much more to explore for the concept. The proliferation of small electronic devices provides fertile territory for further developments, since mobile technology provides massive potential for complicated networks with asymmetrical configurations. New music technologies can be made that offer users ways to create music that is informed by social media and computational ubiquity. Divergence can be explored in a productive way to enrich our instruments.

4.8 User Evaluation

Yig was evaluated in a user study involving sixteen participants who were split into pairs and asked to engage in a musical collaboration. This collaboration was divided into two sittings, with each sitting using a different network music system, namely, *Yig, the Father of Serpents* and Max Neuhaus's *Auracle* (Freeman et al., 2005). *Auracle* is a web based network music system that lets players uses voice controlled synthesizers. *Auracle* was selected for the study because it represents a known network music system from the literature that is designed to be quick to learn and easy to use. The participants were given ten minutes using the first system, followed by filling out a questionnaire containing several seven point Likert items regarding the collaboration. The participants were then directed to collaborate using the other system for ten more minutes, followed by another questionnaire. Afterwards both participants were jointly interviewed with open ended questions regarding the experience. The survey items and interview questions can be found in Appendix B.3 on page X. Ethics approval can be found in Appendix B.2 on page IX. In addition to using two network music systems, half the participants collaborated in the same room, while the other half collaborated from different rooms, without being able to see each other or communicate through any other means than the facilities of the network music system at hand. To mitigate the impact of order effects on the results half the groups used Yig first and the other half used Auracle first.

4.8.1 Quantitative Results

The data derived from the participants is split into a two dimensional space containing four quadrants (Yig/Auracle and Co-located/Distributed), with each quadrant receiving a total of 336 responses to Likert items. This is because there were 16 participants who each answered 21 questions for each system, (16 participants * 21 questions = 336 responses). This group was split in half for the location dimension, with 8 participants being co-located while 8 others were distributed, but because two systems were evaluated each session the same number of items can also be attributed to the co-located and the distributed groups (8 participants * 2 systems * 21 questions = 336 responses). The particular count of 16 also guaranteed that each system was evaluated equally for both co-located and distributed participants, lending to a balanced data set.

It was important to analyze the responses to arrive at an evaluation of Yig as compared to a notable and historic example, but equally important, the study was a unique opportunity to study the effects of locality on networked collaborations. The author is not aware of any formal group study which evaluates potential differences in attitudes towards networked music collaborations that are co-located and distributed. For this reasons the data is analyzed with respect to both the systems and the locality.

The responses to Likert items are treated as ordinal data, forgoing any parametric tests. Instead a focus is placed on using the Mann-Whitney-Wilcoxon test to verify or reject a null hypothesis for two populations. The null hypothesis for each pair of populations (Yig/Auracle and Co-located/Distributed) is that the populations have identical data distributions. Additionally, the Bonferroni correction is used to counteract problems with multiple comparisons for each population. A p-value of 0.05 is used to obtain a high confidence in any results, but to account for statistical irregularities from multiple tests resulting in false rejection of the null hypothesis, the Bonferroni correction states that the results must be less than the p-value (0.05) / m, where m is the number of tests. The questionnaire contains 21 Likert items so any resulting p-value from the Mann-Whitney-Wilcoxon tests need to be lower than 0.05 / 21, approximately 0.00238. Additionally, as the Likert item responses are treated as ordinal data, median values are used to identify the central tendency of a given population and the interquartile range (IQR) of the responses are used to derive a measurement of consensus. Given that the study used a seven point Likert scale, an IQR of less than two is considered to indicate consensus, and an IQR of greater than five indicating strongly diverging opinions within a population. Finally, several of the Likert items have a negative scale, where a strongly agreeing score would denote a strong negative attitude. Note that these items have their results inverted during aggregated analysis, but not in the individual analysis or in the diverging bar charts presented in this chapter and in the appendix.

The first pair of populations considered are the results for Yig, the Father of Serpents and Auracle. The responses are presented as diverging stacked bar charts in figures 4.8 and 4.9. Note that the full page size versions of these charts can be found in Appendix B.5 on page XX. After applying the Mann-Whitney-Wilcoxon tests (with Bonferroni correction) roughly half the items rejected the null hypothesis. The questions that held results rejecting the null hypothesis, meaning that the data sets can not be said to have an identical data distribution, are the following items:

- It is useful for music collaboration.
- It is flexible.
- I am satisfied with it.
- It is fun to use.
- I felt involved with the collaboration.
- I enjoyed the collaboration.
- I felt satisfied with the result.
- Coordination was difficult.
- I would have played longer if given the option.
- If I were performing for an audience just now, I would be satisfied with the performance.
- The collaboration was gratifying.

The Yig responses for the item “It is useful for music collaboration” received a median

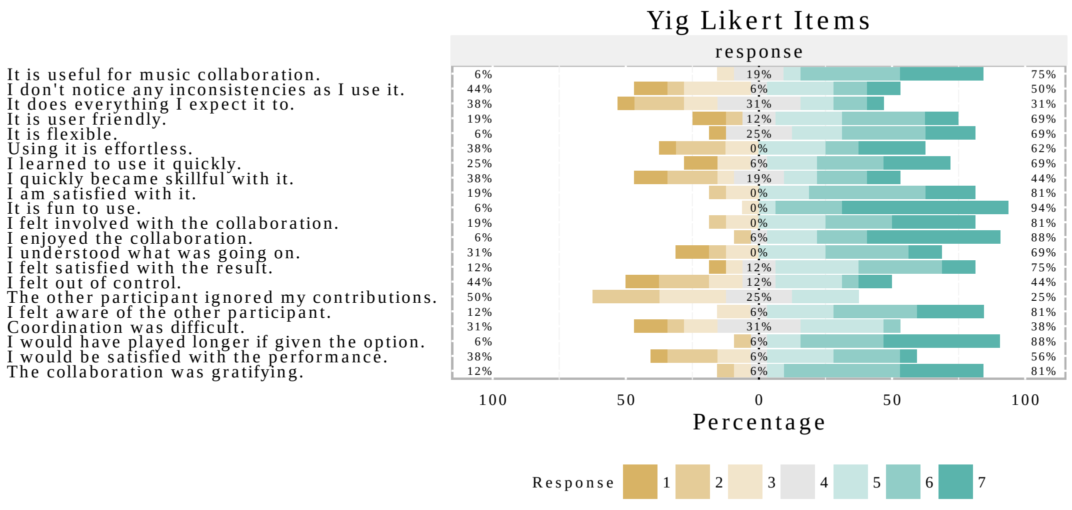


Figure 4.8: Yig Likert Items.

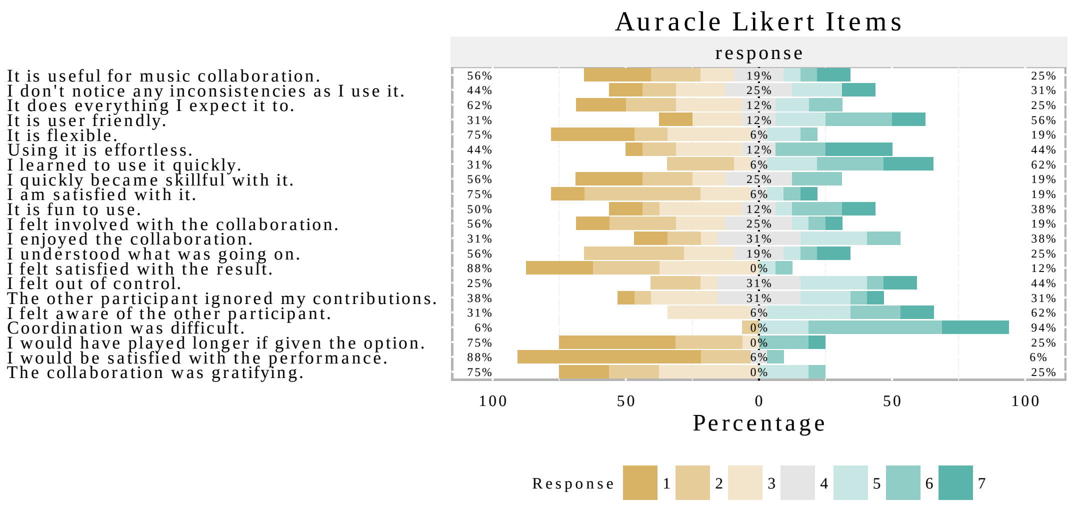


Figure 4.9: Auracle Likert Items.

value of 6 with an IQR of 2.25. Conversely, Auracle received a set of responses with a median of 3 and an IQR of 2.5. Both IQRs indicate that there was disparity in the attitudes of the users regarding the statement. Given the wide disparity in median results and the verification of significant results from the Mann-Whitney-Wilcoxon test, Yig is shown to be broadly considered to be more useful than Auracle for music collaboration by more users, albeit without a strong consensus from either population. These results are similar for the statement “It is flexible”, where Yig received a median of 5.5 and an IQR of 2, and where Auracle received a median of 3 and an IQR of 2.25. There is a measured increase in consensus, but the result is similarly that generally participants considered Yig to be more flexible than Auracle, but with divergence in the population.

The statement “I am satisfied with it” received stronger result than the previous two statements. Yig held a median value of 6 with an IQR of 1 and Auracle received a median of 2 with an IQR of 1.25. These result show that users felt decidedly more satisfied with the the collaboration using Yig than Auracle and that there is a high degree of consensus with that attitude. The statement “It is fun to use” measured an even stronger response for Yig, with a median value of 7 and an IQR of 1. The Auracle results for this item are much less definitive, with a median of 3.5 and an IQR of 3. Yig is shown to be considered fun to use with a strong degree of consensus, where as Auracle response were mixed and generally neutral.

The results for the previous statements start to identify a trend, where users are shown to have more positive attitudes to Yig than Auracle as an enjoyable system. The results for the statements “I felt satisfied with the result” and “If I were performing for an audience just now, I would be satisfied with the performance” corroborate that trend. Yig received median values of 5 for both statements and an IQR of 1.25 and 3.25 respectively. These responses indicate that attitudes about the music created using Yig were moderately positive with strong consensus about the satisfaction about the results, yet with diverging consensus regarding those results being presentable to an audience. This divergence demonstrates that participants can be be satisfied with musical results but that there is a subsection who have separate criteria for results that are performance worthy. Auracle similarly elicited strong attitudes, but instead, strongly negative attitudes for the two statements. There is a strong consensus that Auracle does not produce musically satisfying results, having received medians of 2.5 and 1, and IQRs of 1.25 and 1 respectively. Considering the results for all the previous statements, while there is some degree of diverging opinions, the larger consensus is that Yig produces a more enjoyable and satisfying musical experience than Auracle, as measured by responses to the previously mentioned statements.

The data confirms a musical consensus, but in order to understand the attitudes of these systems for producing satisfying collaborations the remaining statements from the list above need to be considered. The strongest result is for the statement “The collaboration was gratifying” where Yig received a median score of 6 with an IQR of 1.25, and Auracle received a median of 3 with an IQR of 1.5 These show much more positive attitudes for Yig than Auracle for collaboration. This is further supported by the statement “Coordination was difficult” where Yig received a median value of 4 with an IQR of 2. while this is one of the worst scores for Yig, Auracle scored a median value of 6 with an IQR of 0.5, indicating a very strong consensus that Auracle coordination using Auracle was difficult. Across all of the statements where the Mann-Whitney-Wilcoxon test indicated a significant result Yig’s worst reported scores were neutral and by contrast Auracle’s best performing results were neutral.

The previous data analysis establishes a case where participants held more positive attitudes for Yig than Auracle. Still, there were roughly half the questions where the Mann-Whitney-Wilcoxon test did not reject the null hypothesis, indicating identical or highly similar data distributions for the two population. The items that failed to reject the null hypothesis include:

- I don't notice any inconsistencies as I use it.
- It does everything I expect it to.
- It is user friendly.
- Using it is effortless.
- I learned to use it quickly.
- I quickly became skillful with it.
- I understood what was going on.
- I felt out of control.
- The other participant ignored my contributions.
- I felt aware of the other participant.

For these questions it is not useful to compare the results between the two systems, but it useful to consider notable data points where the responses are in accordance about the two systems. Generally most of these questions received neutral medians with a lack of consensus, such as "I don't notice any inconsistencies as I use it" where Yig and Auracle received a 4 and 4.5 median score, both with an IQR of 2.25. Given a lack of consensus in the populations about both systems, it can only be said that attitudes were varied for both systems regarding inconsistencies. Another example is "I felt out of control" where the attitudes were diverse (Yig M=4, IQR=3 and Auracle M=4, IQR=1.25). There were however several statements where both Yig and Auracle scored favorably. These include "It is user friendly", "I learned to use it quickly.", and "I felt aware of the other participant.", all with median scores of 5 to 6, although with a range of IQRs.

Comparing the entire population for Yig and Auracle using the Mann-Whitney-Wilcoxon test produces a p-value of 2.2×10^{-16}, which is much lower than the required threshold of 2.381×10^{-3} for a result to reject the null hypothesis. Yet, this only indicates that there is some difference in their distribution, so to find what that difference is the Likert items were aggregated to create overall results for the entire Likert scale for each participant. This was calculated by averaging the scores given to each question by an individual, but not the results of a single Likert item, which was treated as ordinal data. After calculating these Likert scale scores the median and standard deviation of these aggregates was calculated. Yig received a mean aggregated scale score of 4.961 with a standard deviation of 1.169 and Auracle received a mean of 3.432 with a standard deviation of 1.134. These values demonstrate that while many individual items presented starkly different attitudes to Yig and Auracle, as a whole the two are not as dramatically different regarding their overall assessment. Yet, Yig still leads Auracle with a generally positive attitude from the surveyed respondents where Auracle scored as slightly negative.

Moving on from the first two populations, Yig and Auracle, the next pair of populations are the co-located and distributed groups. The responses are presented as diverging stacked bar charts in figures 4.10 and 4.11. The full page size versions of these charts can be found in Appendix B.4 on page XXIII. After applying the Mann-Whitney-Wilcoxon tests (with Bonferroni correction) none of the individual Likert items rejected the null hypothesis. The individual analysis of the Likert items tells a similar story, where most of the items received neutral scores or had high IQRs indicating a lack of consensus. The only item with a non-neutral median and with general consensus was the item "I felt aware of the other participant." Participants generally agreed with the statement for both co-located and distributed groups, indicating that physical presence may not be a requirement for a sense of awareness in these kinds of network music systems, but that it also is not necessarily detrimental to it either.

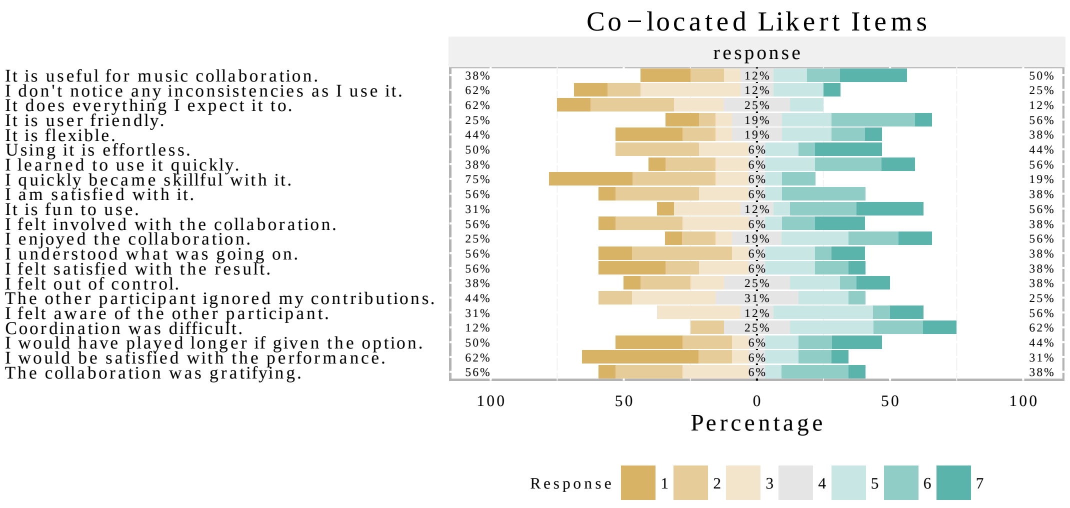


Figure 4.10: Co-Located Likert Items.

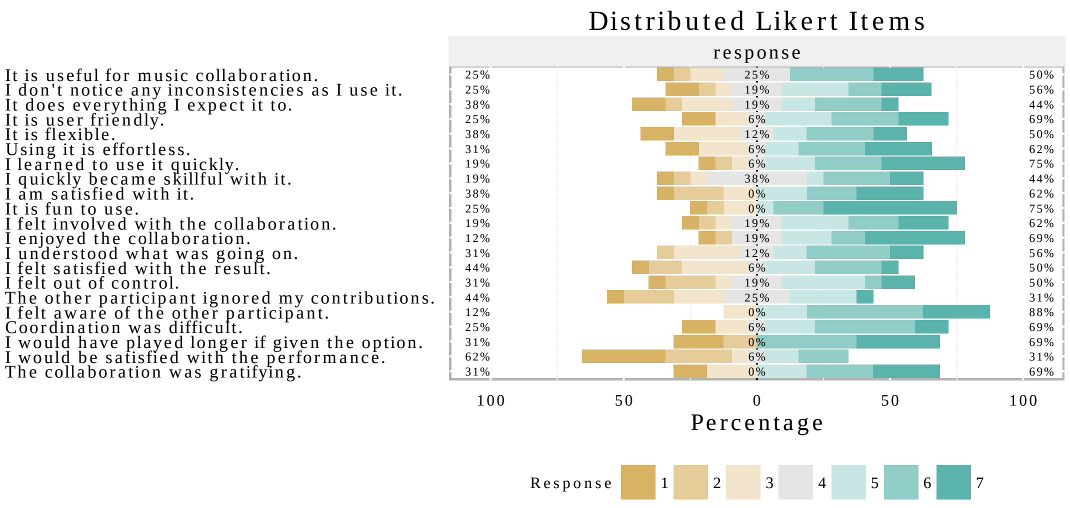


Figure 4.11: Distributed Likert Items.

The co-located and distributed Likert item results were compared as a whole similarly to the Yig and Auracle groups, first by running the Mann-Whitney-Wilcoxon test and then by aggregating the Likert scale scores. The Mann-Whitney-Wilcoxon test on these two populations produced a p-value of 4.237×10^{-7}, enough to reject the null hypothesis. No single Likert item for these two groups was able to meet the requirements to reject the null hypothesis, yet the overall scores for the populations was enough. After aggregating the individual participants scale scores, the co-located group received a mean Likert scale score of 3.818 with a standard deviation of 1.343 and the distributed group received an aggregated mean of 4.929 with a standard deviation of 1.339. It is of note that the distributed population performed slightly better (generally favorable) than the co-located group (generally neutral), which suggests that physical presence may not be a requirement for positive assessments of network music collaborations using systems such as Yig and Auracle. No single Likert item for these two populations met the requirements to reject the null hypothesis, so to better understand some potential reasons for why this is the case will require the qualitative analysis presented in the next section.

4.8.2 Qualitative Results

Each pair of participants were interviewed at the end of the evaluation session. Regardless of if the pairs collaborated co-located or distributed, the interview took place with both in person, where they were presented with the following eight questions:

- Briefly, could you describe your general opinion of the collaboration?
- How do you think the collaboration would have differed if you were in separate locations or the same location?
- How did the software effect your ability to develop musical ideas?
- How would you compare the anxiety of a normal performance with the anxiety from the collaboration that just happened?
- Would you be interested in this kind of collaboration in the future? Why or why not?
- Can you describe the most negative aspects of the collaboration?
- Can you describe the most positive aspects of the collaboration?
- Are there any further comments you wish to make?

Participants were directed to answer openly, and to allow for any natural flow of conversation that stemmed from each question. The answers to these questions were analyzed for themes using the methodology as defined by Braun and Clarke (Braun & Clarke, 2006). According to the methodology an initial set of codes should be developed by annotating the responses where small atoms of sentiment or concept are identified and labeled. By grouping codes together candidate themes are created, combined, or discarded. The codes developed for this evaluation can be found in Appendix B.4 on page XII.

Adding another layer of complexity to the analysis is the quadrant framework, where Auracle and Yig as well as co-location and distribution have to be considered when developing themes. The questions asked do not directly address either network music system, instead allowing for the participants to make any connections to the systems organically. By contrast, the relationship between co-located and distribution was directly referenced in the second question. Pairs of respondents had the opportunity to use both systems but collaborated only as either co-located or distributed. As the topic of distribution was of specific interest, this small effort was made to get the thoughts of the participants on the subject. While it is useful to group emerging themes into sections of this quadrant, it was important not to be overly ambitious about separating themes. Unlike the Likert items in the quantitative analysis, often responses cannot be simply and cleanly categorized into groups. Most of the interview questions were open about subject matter and answers from the participants could be vague or transition and change mid sentence. For this reason themes were identified and developed within identified contexts, but often these themes were emergent across boundaries and without a specific population axis as the source.

Several major themes were identified, but first Yig and Auracle will be addressed, before moving on to themes regarding co-location and distribution. There were a wide range of opinions and sentiments about Yig and Auracle, but by approaching these statements methodically a series of themes were developed. First, a strong theme emerged that the format of the evaluation inhibited developing a deep understanding of either Yig or Auracle. Both systems attempt to create an interface that is intuitive even for users who do not consider themselves musicians, yet a repeated concern was that the limited time frame was in part prohibitive. A shared opinion was that practice and time would make using either system more enjoyable through increased proficiency. This proficiency would allow for a more direct connection between intent and results. Often participants spent a large part of the allotted time learning the interface and rules of the systems, before meaningfully being able to concern themselves with more sophisticated observations such as the utility of the system as an instrument. This observation needs to be considered when evaluating both the results from the quantitative analysis as well as the proceeding themes discussed later. Evaluations by both new and experienced users provide useful data. For that reason, the next chapter will feature an evaluation of a different system by network music performers who draw upon the experience of several performances.

Another theme was that there were technical issues that effected both Auracle and Yig. Yig was reported to have disappearing objects and even a reported crash. Participants reported that Auracle would occasionally stop responding to input. While these bugs caused some issues, generally users did not consider them to be overly distracting. One participant noted that "If you write software yourself, it's going to have bugs in it, I mean it's not a piece of commercial software."

Opinions varied greatly regarding the ease of use and intuitive design of both systems. Auracle was noted as having a direct connection from vocal input to sonic output, as well as having an immediacy in that connection. One participant states "Auracle, was clearer because I knew I made a noise and it translated to a different noise." Many other users noted that while the apparent connection was obvious, the relationship between input and output was not. One user bemoans this behavior, saying "...maybe I want it to approximate something like that but it doesn't really correlate to what I'm doing vocally. It's just kind of a crapshoot." and another "The first one (Auracle) didn't make any god damned sense." This was exacerbated by the users' feelings that there was too long of a lag between input and results. On the other hand several users felt that Yig lacked a clear connection between user inputs and results or that it was not immediately obvious how to make connections between running synths. A participant explains, "So I wasn't quite sure all the time what happened if I was doing something that was actually changing something in the sounds." Others noted that they felt it was more intuitive than Auracle, and that the interface better reinforced actions and the actions of the other collaborator. A user explains “The visuals on the second one (Yig) were super easy to understand. This feeds into that which feeds into this.” Indeed, graphical representation was a small sub theme that emerged regarding the collaborations. Auracle’s spectral display was regarded as lacking: “the first one (Auracle), the graphic visualization of the sound is just far too abstract to understand as someone who doesn’t have a lot of experience.”, and by contrast Yig was lauded as reinforcing the collaboration because of the graphical implementation: “because it’s just like you sort of drag it over, it’s easier to sort of collaborate with that than the Auracle one.”

While there was no great consensus regarding ease of design, there was largely a consensus regarding the resulting sounds and music that was created during the sessions. Generally Auracle was panned as having a limited and uninteresting sonic palette. When asked about the effect of the software on musical ideas, one participant asserts “I don’t think Auracle gets there as far as where it’s worth asking the question”. Another user felt similarly that “Auracle definitely feels more like a toy than an instrument.” Another user remarks about the sonic range “...then I realized it just made one sound, then it got less fun again.” Although, some users enjoyed output, for example this user who found a technique for local feedback “I was recording yours as the same time you were recording mine that was quite cool.” On the other hand many users felt that Yig produced surprisingly interesting results. One user excitedly exclaimed “we’d like be connecting stuff and suddenly it’d be like a strange sound and I would be surprised like, like, shit that’s fucking awesome.”

Regarding the implications of locality of the collaborations, attitudes were diverse and occasionally conflicting. The quantitative analysis revealed that the participants did not favor co–location over distribution, which is the traditional approach to collaboration. The diverse and conflicting opinions could account for a lack of definitive quantitative results. Communication is one aspect of collaboration that is greatly effected by distribution. The innate communication channels found in co-located situations was argued by some to be a useful tool. For example, “The good thing about being in the same room is we could talk to each other about what we found out.” Other uses felt the distribution was detrimental or unusual, such as the user who said “I feel pretty akward just sitting alone and making noise.” By contrast there was a running theme that distribution may actually be desirable. When one user recalled that upon realizing Auracle required vocal control, they were “quite relieved in a way to be in a different room.” Another user stated that “...for uh, Auracle I think that it would not work at all if you were in the same location” The requirement for vocal control may be an innervating source, and may also be partially responsible for scores to slightly favor distribution over co-location in the quantitative analysis. Other users, from both co-located and distributed populations, felt there would be no difference if the locality was changed, from co-located to distributed, or vice versa. As a participant said, “I was pretty glued to the screen. It wouldn’t have made much of a difference.” Additionally, most users neglected to use the chat capabilities of either music system. Co-located users would occasionally speak to each other or make visual contact. Yet distributed users, lacking this channel for communication, did not turn to the available alternative. Instead, their sentiment is echoed by the comment that "I was too busy exploring that to actually type like 'Hey, let's do this.'"

Several other themes regarding collaboration emerged from the interviews. Yig uses a two dimensional space where circles in this space represent running synths. Furthermore there were only limited constraints on control input for these running sounds. While a particular user is controlling a specific synth no other user can move or delete it. Once they stop controlling it, that synth is open for any user to control. One theme that emerged is that in this virtual space there is some innate sense of etiquette and ownership regarding these circles. One user explained some self restraint when using the system, "And I also felt, about the second software, actually I wanted to delete some object but I didn't since he made it, so it might be rude." Another participant who accidentally deleted all the running synths at one point during the session, apologized to the other user in the interview, saying "Sorry I deleted everything. (laughs)" This digital space was not safe from good humored 'trolling', and one user reveled in their slight digital transgression by jesting "Yeah, it was, um, more fun trying to stop you from doing anything (laughs). Like putting one of your little circles so that you couldn't reach it, that was quite fun." Several users expressed some desire to have more independence during the collaborations, and several comments were made about the desire to have a mute function. Yet others enjoyed the serendipitous collisions of intent that fostered discovery, with one user noting that "...there were things kind of discovered by accident so like when I move my mouse at the same time as yours and then we got that random connection by mistake"

Overall the qualitative analysis supports the quantitative results. Yig was favored over Auracle for music collaboration, but both were offered criticism, especially concerning intuitive design. Two reasons that Yig performed better include a better graphical representation of the collaboration, and the observation that it fostered co-discovery. Additionally the sonic output was regarded as more interesting and more viable for consideration as an instrument. Regarding locality, the results were similarly mixed as with the quantitative results. Opinions varied and the data suggests that their is no clear indicator for preference in the two populations. Advantages were noted for both approaches, as well as disadvantages. Finally, etiquette and ownership were two emergent themes that could not be found in just the Likert item analysis. Some 'real world' principles were transposed into the virtual space. This was especially true for Yig, where the locality of presence may be an impetus for feelings of embodiment.

Chapter 5

An Interactive 3D Networked Music Space

5.1 Introduction

Shoggoth is a new network music program for real–time group performance with members distributed over potentially global distances. As a reference to the strange protoplasmic beings described in H.P. Lovecraft's *At the Mountains of Madness* (H.P. Lovecraft, 1931), Shoggoth allows users to reshape polymorphic terrains to create generative music in collaboration. The program is designed with a user interface that is both functional and highly visual. A video of Glitch Lich performing with *Shoggoth* at the 2013 Network Music Festival in Birmingham, UK can be found at `https://vimeo.com/94046155`. The interface design allows for an aesthetically pleasing presentation that serves to both enhance communication in the ensemble as well as provide a visual representation of performer actions to the audience. This is important because performances with physically separated ensembles present a unique stage presence where parts, and possibly all, of the group can only be represented through digital media. The separation in distributed ensembles amplifies several issues in traditional computer music performance, such as a lack of correlation between physical effort and sonic results. Furthermore, distributed ensembles lose fundamental components of communication such as visual cues and gestures.

These issues are not new (Berio & Dalmonte, 2007; Wessel & Wright, 2002) and there is a growing range of techniques and technologies which seek to mitigate or embrace these features of electronic music. Controllers and interfaces are a popular solution for computer musicians to reestablish or re-imagine the performance characteristics of traditional instrumentalists (Morris, 2008; Rebelo, 2006; Wilson-Bokowiec & Bokowiec, 2006). These interfaces lose value in networked performances if members are in different locations from each other or from an audience. Concerts in many forms, from experimental computer music in smaller clubs to popular music stadium shows, are now commonly performed with accompanying visuals to augment stage presence (Sexton, 2007; Brougher, Strick, Wiseman, & Zilczer, 2005). While there may be useful benefits from adding the visual medium, if the presentation isn't communicative of the non-present performers, their contributions will be deemphasized or lost entirely. For this reason network performances are occasionally realized using video and audio streaming between performance sites (Chafe et al., 2000; Sawchuk et al., 2003; Gresham-Lancaster, 2007). Latency and quality of connectivity are ever present concerns, and if performers aren't using traditional instruments or physical controllers then the same issues regarding computer music performance outlined earlier will still be present.

This is where virtual spaces can serve a useful role. Networked performances, distributed or not, that are performed in virtual spaces communicate performers' efforts while simultaneously increasing ensemble communication. Consideration must be given to both usability and presentation and a balance must be struck to facilitate a successful performance space. Video games are a natural source of inspiration, with their sprawling and detailed worlds, the largest of which are developed utilizing multi-million dollar budgets and over a period of several years. Music has been an important component of video games since the beginning and game music has become ingrained in our culture. Despite this, music has usually served a secondary role, similar to it's usage in movies to set the tone of a scene or level. Game music is commonly adaptive, not interactive, because there is usually no direct connection between player actions and changes in the music (Collins, 2008). Sound effects are the actual interactive components in a game, such as the triggering of a jump sound based on a button press. There is often some correlation between game state, such as the adjustment of tempo according to a game boss's life.

There is a history of utilizing games or game like worlds in music and sound art. A common approach has been to appropriate or modify an existing game for use in a work. Cory Arcangel's Nintendo cartridge hacks, including his celebrated *Super Mario Clouds*, and Tom Bett's glitch inducing quake engine modification *QQQ* are two examples of how an existing game can be appropriated to produce results never intended by their designers (Bittanti & Quaranta, 2006). Both modify the source code for a game, fundamentally altering it's logic, and creating something new. Not all game appropriations are as subversive. Rob Hamilton's work *Maps and Legends* (Hamilton, 2007b, 2007a) built using *q3apd* (Oliver & Pickles, 2002), a Quake III modification by Julian Oliver and Steven Pickles, is a network composition performed in virtual space. Player states such as position and view angle, and weapon selection, as well as certain actions such as jumping and firing are mapped using OSC to control a Pure Data patch (Chung, 2013). These mappings allow Hamilton to use the core logic of the Quake engine as the framework for a networked virtual performance.

In Shoggoth, instead of using an existing game engine, a new one was written specifically for the purpose of network music performance. This allowed for the customization of a system that attempts to find the right balance between usability, musical control, and visual aesthetics. The following section discusses system design and philosophy, as well as some technical aspects of the implementation. Next the system is categorized using established frameworks with a subsequent examination of the role of virtual spaces in music performance. Finally, informal assessments from two experienced Shoggoth performers are considered.

5.2 Design Philosophy

The basic design goal of Shoggoth is to have an emmersive virtual space for collaborative performances of harsh rhythmic music. As with Yig, Shoggoth was designed for Glitch Lich performances. The key design requirements are distributed performance support, communication support, autonomy, a virtual space, and visualization. Similarly to Yig, Shoggoth supports distribution with all clients rendering the full audio output. Shoggoth also supports a simple chat system for communication. The autonomy in Shoggoth is similar to Yig, using feedback to create emergent behavior.

For a virtual space Shoggoth supports a full three dimensional environment with keyboard and mouse based camera position and rotation. In this way Shoggoth's design is largely inspired by first person video games. Glitch Lich's previous systems, such as Yig, demonstrated increasingly graphical interfaces, often accompanied by a separate visualization program. This approach has worked well, although it also meant that audiences

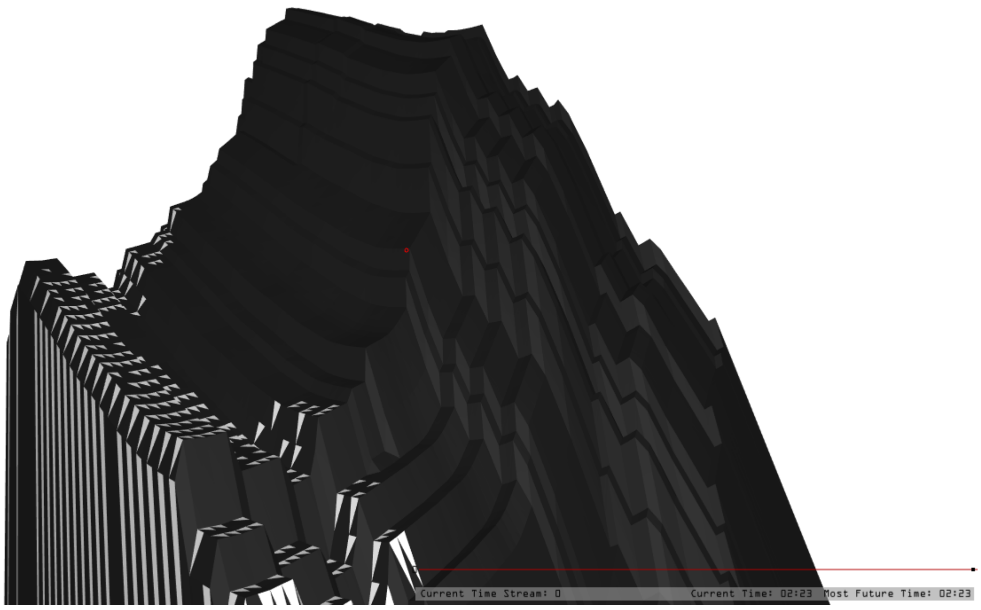


Figure 5.1: One possible terrain shape in Shoggoth. The heightmap defining the shape of the terrain is also used for the buffers of the wave terrain ugens in the running synths.

were not presented with the same visual information as the performers. The design goal of Shoggoth is to create an interface that is aesthetically rich while functioning as the interface through which the musicians collaborate.

Similarly to Yig, the design produces an interface with unavoidable trade–offs. Using a perspective camera in a 3D space introduces new issues regarding field of view. Where as in Yig each performer could see everything, in Shoggoth, each performer has their own perspective which can exclude large parts of the performance area. This makes it more difficult to be aware of what the other performers are doing at any given time. This also effects the projection for the audience. Shoggoth performances all used the video output of just one computer, often because the performance was distributed and only one of the performers was actually at the venue. This meant that the view of the performance was limited to one performer. A specific spectator mode for Shoggoth (and other 3D performance interfaces) similar to spectator mode in games like *Defense of the Ancients 2* would help mitigate these problems for the audience (Valve, 2016).

5.3 The Interface

Shoggoth is written in C++ and uses the Cinder framework (Rijnieks, 2013) for the graphics implementation. On startup the view comprises of a grid of flat black square islands suspended in white space. Users can fly around the space by employing controls similar to a first person shooter (FPS) game, but there is no gravity or physics. The flat grids are vertex buffer objects (VBOs) (McReynolds & Blythe, 2005) comprising of a triangle mesh

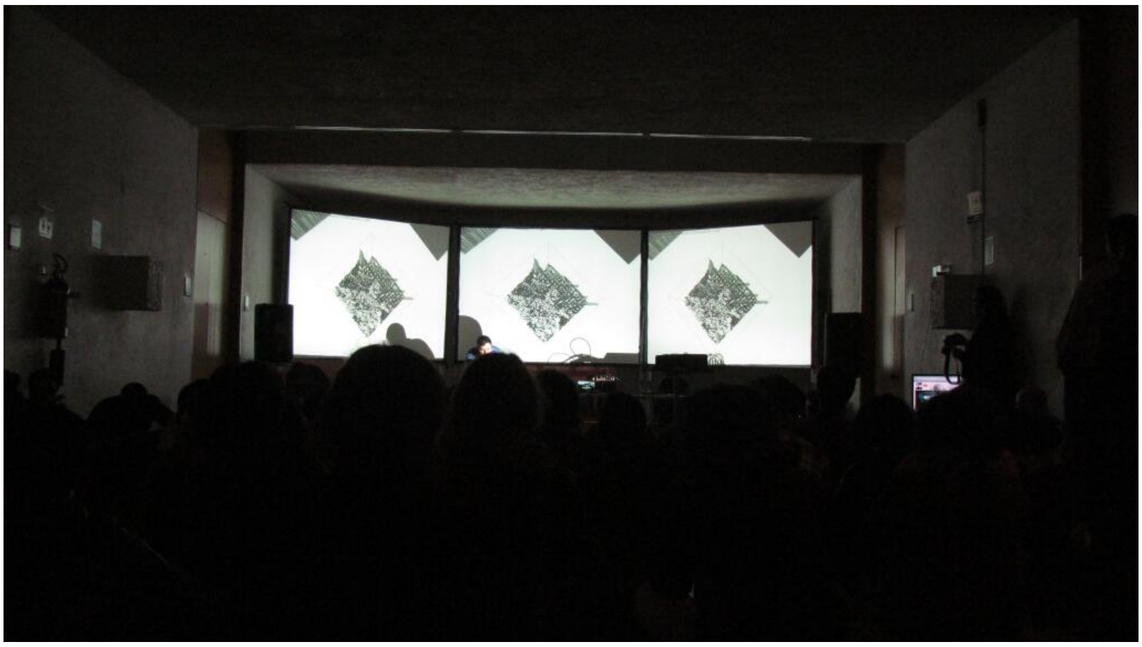


Figure 5.2: Glitch Lich performance of Shoggoth at /*vivo*/.

bound with important data such as color and identification numbers. The grids can be manipulated using a selection of number keys that trigger a morphing animation into various shapes dependent on one of several generative processes. These processes are each based on a particular algorithmic model, enumerated as as (0) Blank, (1) Diamond Square, (2) Cellular Automata, (3) Strange Attractor, (4) L-System, and (5) Flocking. Each process results in a height map and a series of intermediate steps are constructed between the existing mesh and the new version. Using a queued update system the mesh is updated each frame, incrementing through a thirty step animation list, until the final version of the mesh is reached. Earlier versions of Shoggoth did not have animations between meshes and for that reason mesh transitions were jarring, which inspired the added feature. Animations not only create smooth changes and striking visual effects, but also allow for the audio sequencing to follow the interpolation as well.

A triangle can be selected using 3D picking (Shreiner, 2009a), from the grid of a terrain mesh for sequence path creation or manipulation. 3D picking is a technique that allows users to select something in 3D space using 2D coordinates, usually via a mouse controlled camera view. 3D picking was implemented in Shoggoth using a graphics technique whereby the terrain meshes are rendered at a lower resolution into a frame buffer object (FBO) (Shreiner, 2009b), which is never shown to the user, and each triangle in each terrain mesh is colored according to a global identification system. When a picked triangle is requested, the color of the pixel in the exact center of the FBO is selected and then only has to be translated from an RGBA (reg, green, blue, alpha) value into an unsigned integer, resulting in the selected triangle's global identification number. This proved to be invaluable as each terrain contains over 10,000 triangles and previous attempts using ray casting were unusably slow.

A path can be created from a sequence of triangle picks, and once outlined, a read head

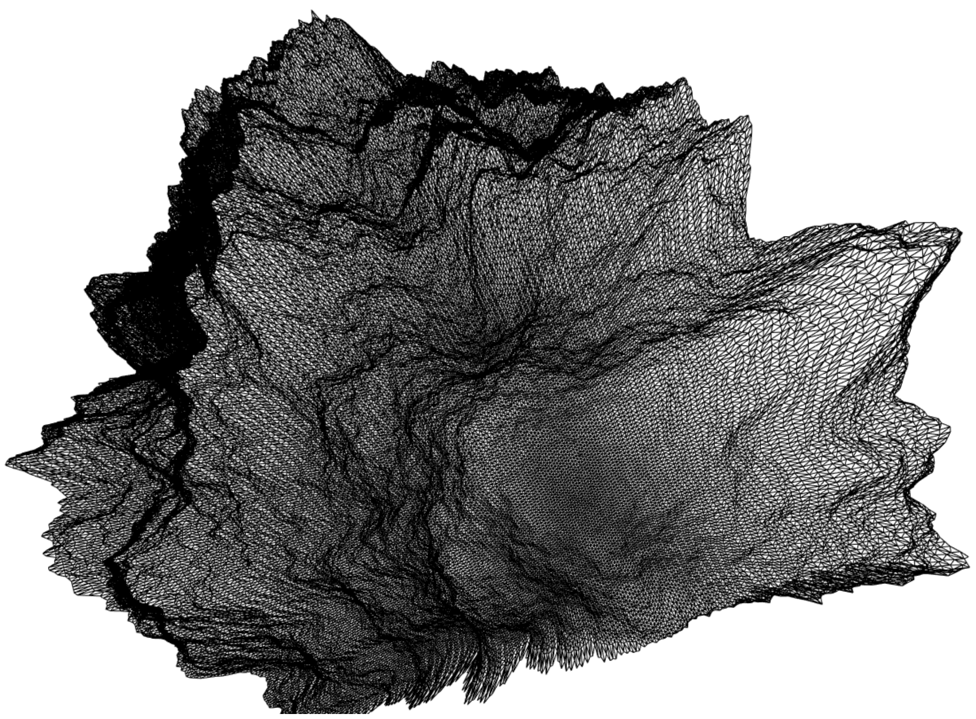


Figure 5.3: Wireframe render for an island terrain, demonstrating the triangle mesh and high polygon count.

immediately follows on the path, triggering and modulating monophonic synth instances. A triangle in the mesh of an island has two possible states: black (inactive) or white (active). If the triangle is active when a read head passes over it, then a coordinating synth is triggered, resulting in an opening of the envelope gate and an update to the parameters of the synth according to the triangle's height and location in the grid. Triangles are activated or deactivated according to a similar set of generative processes as the height map, and are triggered using the same number keys, but with the shift key pressed as well.

Player representation and communication are important in network music performance and Shoggoth has some simple, but effective, designs to facilitate them. Players are represented using minimalist tetrahedron models, which aren't complicated, but align well with the triangle based theme of the islands. Position and rotation information is mapped allowing performers to see not only where each other are, but what they're looking at, and the immediate results of their actions. This is an upgrade from the authors' previous systems where either no representation was made or only position data was represented. A chat system has been created to allow for communication, both with the other performers and the audience, and uses a multi-player game style 2D overlay.

5.4 Sound Design

Sound in Shoggoth is implemented using the SuperCollider (McCartney et al., 2016) libscsynth library in conjunction with libsc++ (McKinney, 2013a) to create an internal server built natively into the C++ application. Because the server is built internally, no exter-

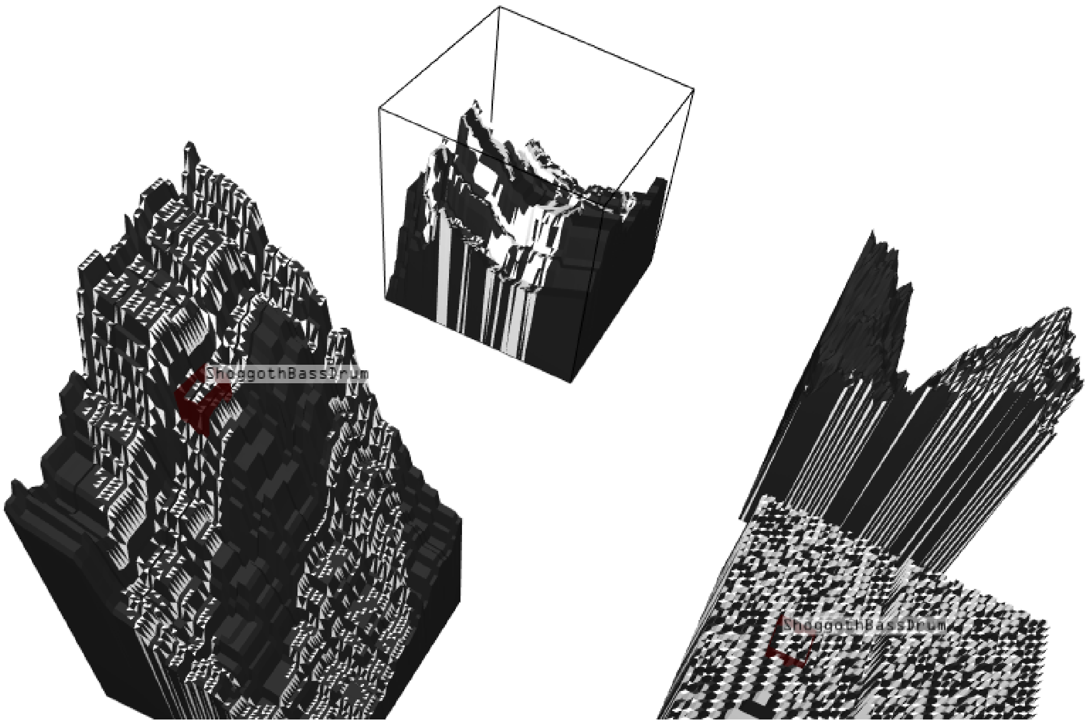


Figure 5.4: Multiple islands with sequences. The red area on the bottom left islands is a looping sequence that triggers synth onsets.

nal messaging is necessary, and all communication with the scsynth server and Shoggoth occur through native function calls. Shoggoth will fail completely without any hanging servers in the event of a crash, but if the server was running as a separate process on the local machine this would not be the case. Maintaining independence of the sound server, language, and now the IDE is a favorable characteristic of SuperCollider as an audio language. That level of independence is not favorable when distributing a program to users who may not be knowledgeable about the subtleties involved with multiple processes.

Synth design in Shoggoth is focused on the usage of wave terrain synthesis (Roads, 1996). Each synth definition utilizes at least one wave terrain oscillator that reads a buffer filled with the same 2D height map that defines the shape of the terrain that the synth's sequence resides on. This is a essential feature because it allows the terrains to effect not just the sequential triggering of synth instances or the modulation of synth parameters, but also to define the most fundamental components of the synths' timbre. Each generative process, such as the cellular automata, have a characteristic harmonic palette that forges a strong connection between the visuals and the sound. Furthermore, when the island meshes morph into new forms, the animation effects not only the visuals display, but also updates any running synths as well, creating a dramatic timbral shift.

Synth definitions must be compiled against the same version of the SuperCollider synthdef format that Shoggoth is compiled against. This creates a dependency on either SuperCollider itself or some other environment that can compile SuperCollider synth definitions. This might change given development in the libsc++ library that could allow for native or scripted synthdef compilation in Shoggoth itself. Even given this dependency,

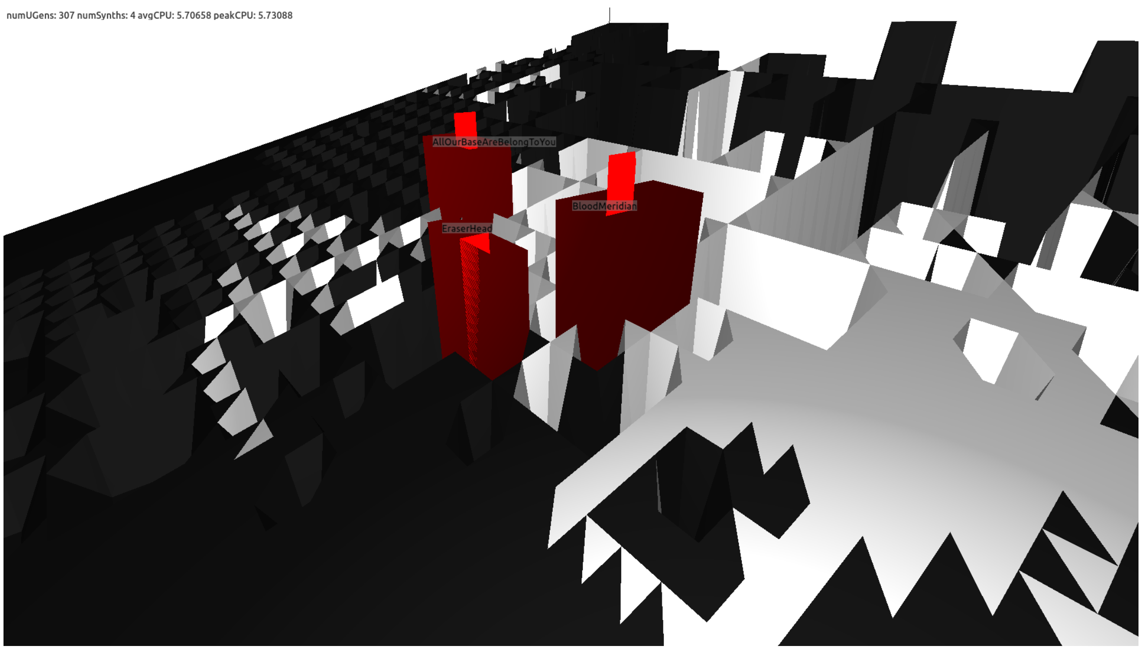


Figure 5.5: Three looping sequencers on an island. When the sequencers land on white triangles a synth onset is triggered.

SuperCollider is an excellent choice for sound design because it has an established code base with years of active development and supplies a well defined and terse interface for synthesis. Shoggoth can be used to create a wide range of sonic output, but given the looping sequential infrastructure and the often aggressive waveforms produced by the wave terrain synthesis, rhythmic noise is the most natural end result. While this style of music may not appeal to all, generative and networked music audiences are often interested in more experimental music.

5.5 Networking

The Shoggoth network implementation is very similary to Yig, using Open Sound Control (Wright, 2002) messaging with OSCpack (Bencina, 2013) to create and receive OSC packets and the OSCthulhu (McKinney & McKinney, 2012) server and client framework to synchronize state. In Shoggoth there are eight types of information networked: Player position, player orientation, terrain height maps, terrain step grids, sequence positions, sequence sizes, synth selections for sequences, and chat. This group contains a wide variety of data from character strings to high volume meshes. The terrain meshes proved to be the most challenging to network initially. Synchronizing 10,000 triangles with 3 points each as well as the step grid was daunting, inspiring odd attempts to reduce bandwidth such as using LZMA compression (Salomon, 2006). The solution was limiting and ultimately unused. Instead of manually synchronizing each triangle, instead the settings and random seed used to generate a given terrain mesh and step map were synchronized, allowing for an incredibly small amount of information to guarantee state across the network. The drawback is that manual deformation of the terrains had to be removed, leaving only the generative processes able to create and manipulate the islands. This changed the nature of the performance from guided intentionality to fast experimentation, but the reduction in bandwidth was extremely beneficial.

Player position and orientation were easy enough to network, but using seven arguments per user to define them (3 for position, 4 for quaternion defined rotation) sometimes generated asynchronous updates for their individual components and unnecessary traffic. This problem led to the use of bitpacking to package updates together. Using bitpacking the X/Y/Z components of the position of a player can be packed into a single integer value, as well as the W/X/Y/Z components of their rotation, reducing traffic while simultaneously enforcing unified updates. The same technique was used with the aforementioned island states. All the settings including the process number and random seed are packed into a single integer so that there is a guaranteed success or failure of an update. This prevents scenarios where only a portion of the information needed to update an island is received, while the others might be lost, resulting in an incorrect state. Because of these efforts, Shoggoth's networking is precise and fast despite the large amount of information represented on screen and even while using low quality wireless connections.

5.6 Categorization

As in the previous chapter with Yig, here is a set of classifications of Shoggoth according to several network music taxonomies including Alvaro Barbosa's network music classifications (Barbosa, 2003), Andrew Hugill's internet music taxonomy (Hugill, 2005), Gil Weinberg's enumeration of network configurations, Golo Föllmer's twelves types of net music (Föllmer, 2005) and Thor Magnusson's epistemic dimension space (Magnusson, 2010). Starting with Barbosa's network music classification, Shoggoth is similar to Yig in this taxonomy. While Shoggoth has the capacity to support asynchronous interaction, the system was designed for synchronous collaborations and performances, and lacks important features for asynchronicty such as state storage and recall. On the other hand the live configurations of patterns and synth parameters coupled with the client and server architecture highly encourages synchronous interaction. Similar to Yig, Shoggoth can be used both locally and remotely, as the networking was written to handle large distances. Similar to Yig, local performances may benefit from using a single computer's output for signal as some highly chaotic feedback sounds can producte different results across machines, although this may be an aesthetic choice as the layering may be considered to be a useful dimension to the performance. Barbosa labels music that has synchronous interactions but with local and remote locality as 'Shared Sonic Environments', which is especially apt given the three dimensional virtual space that Shoggoth performances inhabit.

Next, with Hugill's Internet music taxonomy, Shoggoth sits well in the second category of music that is *Created or Performed in Virtual Environments, or Uses Virtual Instruments*. Convincing arguments could be made that Shoggoth also facilitates music that *Uses the Internet to Enable Collaborative Composition or Performance*, but the defining features of Shoggoth align more closely with most of the pieces that could fit into Hugill's second category than his fourth.

As with Yig, Shoggoth uses the ‘Flower’ or ‘Synchronous Centralized Interaction‘ network configuration, as listed in Gil Weinberg’s network topologies. This configuration is useful as it provides a stable and standardized structure for network traffic, as well as provides a way for users to easily collaborate from behind firewalls, avoiding many of the technical issues with facilitating direct peer to peer communication. Similar to Yig though, the configurations of messages, feedback, and control flow is much more complicated, and these kinds of connections and interdependencies can form some of the more odd and exotic configurations in Weinberg’s collection.

Föllmer’s taxonomy is much more complicated than Hugill’s and requires more consideration. There are many similarities between Shoggoth and Föllmer’s description of the *Algorithmic Installations* type, but the emphasis on an installation as opposed to performance does not afford an easy fit. With that consideration, the *Performance* cluster, number 5, is the most natural, leaving a choice between *Network Performances* and *Staged Projects* types. Shoggoth makes no use of librettos or text and therefore *Staged Projects* makes little sense, leaving type K *Network Performances* in cluster V *Performance*.

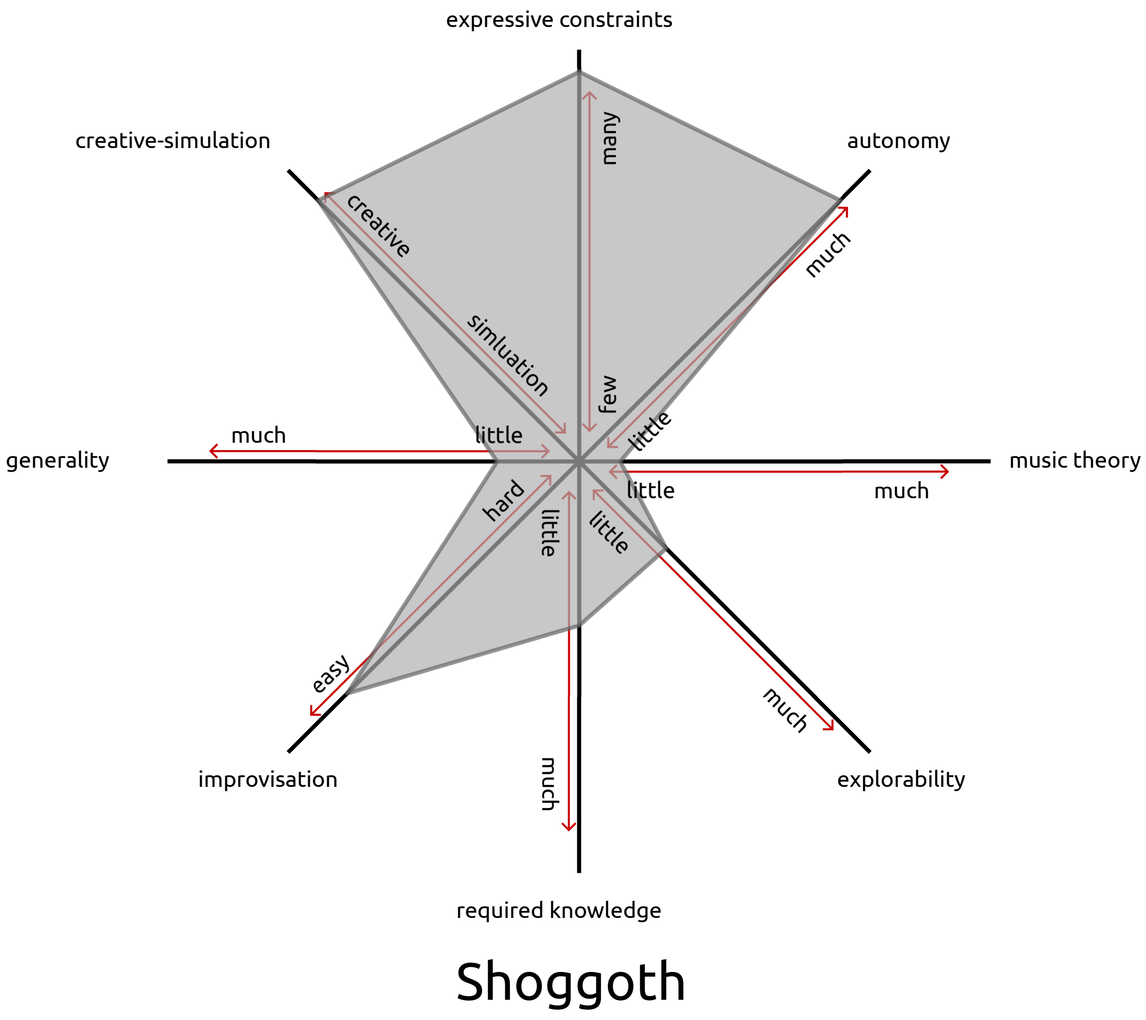


Figure 5.6: Shoggoth in the epistemic dimension space.

In figure 5.6 Shoggoth is plotted in Magnusson’s epistemic dimension space similar to

Yig in the previous chapter. Shoggoth is most similar to the ReacTable (Jordà, 2009), when compared to the list provided in figure 4.5. While Shoggoth is marked as having more autonomy than the ReacTable (because of the extended use of generative processes), they both impose significant constraints on expression, while lacking generality and inherent music theory constructs. Instead, both the ReacTable and Shoggoth emphasize improvisation in a creative and unique interface. In contrast, Reason contain more music theory infrastructure, and Max/MSP has more depth of explorability.

5.7 Performance in Virtual Space

Video game culture provides a useful reference for digital spectator events. The youngest generations have been raised in an era of video games and the internet, giving rise to online eSports such as *Star Craft* and *League of Legends* (Taylor, 2012; Jin, 2010). These games are not just for bragging rights and the winners stand to win hundreds of thousands of dollars in front of thousands of fans (Benedetti, 2012). Virtual music (Duckworth, 2013) performances have not yet reached this level of acceptance, but the concept has been proven that there is a potentially large audience for virtual performance.

There are many similarities between an online battle and an online musical performance (perhaps even an online music battle). The two are often group events and the depiction of embodiment is important to spectators. But where games have concrete goals and rules that dictate their achievement, musical performances have compositions and improvisations with a wide range of constraints and goal orientation. There are other considerations, such as the embodiment of the performers and their portrayal, or lack thereof, of physical and emotional state. These have an important role in how a performance is perceived by an audience, and by the performers themselves (Deutsch, 2012). If virtual music performances are to attain the popularity of virtual sports, more work will need to be put into the systems and infrastructure that supports those performances.

Online games such as *Star Craft* have budgets that rival hollywood movies, but more importantly there is a depth to the software that is simply missing in network music systems. For example the players in Shoggoth are represented as simple tetrahedron with only position and rotation as defining features. Characters in a video game on the other hand can have hundreds of animations. The amount of detail in the textures, meshes, lighting, and shading in a large game dwarfs the efforts of even the most ambitious network musician. Important steps can be taken to improve the situation, such as the development of open and abstracted tools sets to reduce duplicated work and the adoption of new skills such as modeling and animation. Perhaps the most useful step is to consider how some independent developers manage to compete against even the largest games despite tiny budgets and thin development teams. Games such as *Minecraft* (Mojang, 2013) attract massive audiences despite these issues because they use resources wisely, often employing minimalist or generative techniques, and create sophisticated and stylized game and art designs that don't require large resources.

5.8 Reflections on Development

After months of work, and many challenges along the way, Shoggoth has reached an initial release and is performance ready. The program is fairly stable and a recent feature lock down means that future development will be concerned with bug fixing and system efficiency. Other network musicians or software designers will benefit from learning about a few of the challenges throughout Shoggoth's development.

An important consideration is when to write a completely new engine from scratch or in contrast, recognizing when an existing engine is a viable option. Writing a new engine should not be considered lightly, and indeed the vast majority of the time spent developing Shoggoth was put into building basic functionality such as FPS style camera controls, mesh generation, and a chat system. The Quake III engine mentioned earlier, or something similar such as the Unreal Engine (Finch, 2014), will already have this kind of functionality built in, and will greatly reduce your development time. Only if something requires a unique feature (in the case of Shoggoth, the polymorphic terrains) should engine development be considered. Furthermore, if the decision is made to write an engine, the creation of an abstracted framework or library will benefit subsequent development. Shoggoth is written without such abstractions and for this reason much of the code base is not easily portable to other projects. For that reason, this development cycle inspired the creation of an engine with many of the basic functions underneath a network music program like Shoggoth implemented using a clean and abstracted interface.

Another large problem facing development was a lack of focus during some periods of design. Experimentation is a useful technique in music software design, but some amount of planning will minimize lost time. For example, in Shoggoth the fundamental way in which performers used the interface was not clearly defined until well into the development process, resulting in several abandoned efforts and wasted time. From a musical perspective, a lack of focus is also problematic because it creates a moving target for sound design, stunting the growth of the system's musical identity. Finally, allowing other musicians an opportunity to use and evaluate the project starting early in the process will help identifying problems not just with the code base, but also the design and vision of the project. For Shoggoth, that external assessment was not introduced early enough in the process, leading to some of the issues mentioned earlier. Moving forward the main concern is to perform with the interface, find and fix problems in the system, and to streamline the project where possible.

5.9 Experienced User Evaluation

No formal group evaluation was conducted for Shoggoth. Instead a survey was sent to the two users with the most experience with Shoggoth, Curtis McKinney and Cole Ingraham. While these responses are purely anecdotal, their feedback provides some insight into how successful the execution of the design was, from the perspective of having played several performances using Shoggoth with the author as the band Glitch Lich (McKinney et al., 2012). The full survey can be found in the appendix on page XXVII. The survey begins with five statements in a seven point Likert scale. The statements and responses are as follows:

- Shoggoth is useful for music collaboration: (7, 7)
- I quickly became skillful with Shoggoth: (6, 5)
- It was difficult to communicate and collaborate: (2, 2)
- The 3D graphics interface implementation was useful: (6, 3)
- I will use Shoggoth in a future performance: (2, 3)

Given that both Ingraham and McKinney have performed with Shoggoth multiple times, the first four responses, which show support for the system, are unsurprising. Yet, on the last point both users disagree with the statement "I will use Shoggoth in a future performance." To understand this sentiment will require addressing the answers they provided to the last five open ended questions. These questions will be addressed in turn. First is the question "How did the 3D graphics interface effect your ability to develop musical ideas?" Curtis McKinney responded with

> *The graphics give a kind of awareness for the music that is often lacking while playing electronic music. While sometimes interfaces can be a distraction to music, the interface in Shoggoth actually ended up accentuating it, due to the fact that it often produced novel patterns. This led to an interplay whereby I felt compelled to alter the music, or the visuals, in reaction to each other. This manner of performing was unique and enjoyable.* – Curtis McKinney

One emergent aspect of Shoggoth was that the performers would take actions in the piece not just for musical effect, but also for visual effect. This might include flying around the space to find interesting camera angles or changing the shape of the terrain to find an aesthetically pleasing contour. Performances engaged the user in a multi–modal context that the band's previous systems had not. Ingraham responded with

> *The interface itself was not a huge factor in the musicality of my experience with Shoggoth. The mode of interaction of a 3D FPS was intuitive for me (as someone with a gaming background) but did not influence my musical decisions. If anything, it forced me to make more random, rather than informed, decisions because of the inherent imprecision of the interface. This was of course compounded by the fact that the build I was using was heavily glitched and often times unusable.*
> – Cole Ingraham

One unfortunate side effect of building something as complex as Shoggoth for performances was that development became much more difficult, and subsequently there were several bugs in the software at various points, even during performances. Ingraham used a different platform (Apple OSX) than the other members (Linux), and platform specific issues could render builds unstable or even unusable. One example is the February 24th, 2013 Glitch Lich performance at the Network Music Festival in Birminghan, UK (Knotts & Hutchins, 2013). This was the premier of Shoggoth, but only the first piece, Simulacra, was checked with the projector at sound check. During the performance it was discovered that the first person camera controls did not work correctly with a projector using a difference resolution connected to Shoggoth, which restricted the movement of the performer in virtual space.

Next the two performers were asked "How did the 3D graphics interface effect your ability to communicate and collaborate?" McKinney answered

> *The interface provided the familiar (for video game users) interface of the command + t chat prompt, which made organizing the performance simple. Having a running chat log with fellow musicians made it easy to quickly guide how the performance played out in a direct way. Furthermore, the avatars for the players gave immediate and intuitive feedback for what activities each perform was up to. This often led to plays automatically assuming roles, or sometimes congregating around interesting activities happening in the piece.* – Curtis McKinney

McKinney references notions of locality and embodiment in the virtual space. The design of Shoggoth has four separate islands to allow for users to work independently or more directly together. Using an avatar in 3D space allowed performers to quickly assess intentions and activities. Ingraham echoes his first sentiment

> *For the specific build I was forced to use, the interface encumbered my interaction with the other performers. While I could communicate easily via the chat system, I was unable to see where the others were in the 3D environment. The result was a greater delay in my ability to react to the decisions of others.* – Cole Ingraham

One of the issues with the Apple build was that the avatars did not work correctly, which dampened the intended effect of the 3D implementation. These criticisms regarding the technical stability of Shoggoth were echoed by both McKinney and Ingraham in their response to the question "Can you describe the most negative aspects of your experience with Shoggoth?"

> *Shoggoth uses a sequencer-like system to control the beats made in the music. The system is easy to use and creates rich beats, but there were times I wished there were other direct ways to control sounds sources. Also, Shoggoth has issues being a consistent platform for performance due to both difficulties in maintaining the code and consistent bugs during performances.* – Curtis McKinney

> *Although this was obviously not a design issue, the instability of the build, and difficulty of compilation for that matter, make performing with Shoggoth particularly frustrating. Also for the interface critiques stated above, it feels rather difficult to develop a sense of mastery of the piece, at least in my experience with it.* – Cole Ingraham

Beyond the technical issues both Ingraham and McKinney note that there are some limitations to the input mechanisms afforded to the users for making musical decisions. For example, Shoggoth lacks continuous parameter modulation from user input, a feature most of the previous systems that Glitch Lich had used previously boasted. Still both Ingraham and McKinney praised Shoggoth for the sonic capacity of the system, when asked "Can you describe the most positive aspects of your experience with Shoggoth?"

> *Shoggoth makes feedback noise music an approachable sandbox. Modulating terrains and sounds in conjunction turns out to be rather tantalizing. Seeing manifestations of other plays during performance gives a sense of collaboration, even when performers are physically absent.* – Curtis McKinney
>
> *Beyond the aforementioned issues, it is very fun and compelling to see and hear. An audience member would likely greatly appreciate the interesting geometry and rich noise/feedback textures. As a performer it is also a fun and novel sandbox to play with.* – Cole Ingraham

Shoggoth has had several performances internationally, including performances in England, Mexico, South Korea, and the United States. The performances where Shoggoth was used all received positive feedback, despite the technical issues. Still, the problems arising from developing a 3D networked interface in C++ has prompted a reassessment of technical strategies, and the members of Glitch Lich have adjusted their priorities to include stable builds and build environments.

5.10 Summary

This chapter presented Shoggoth, a new interactive system for performing networked generative music within a 3D space. Discussion began with the unique problems that network music performances face, especially with regards to distributed ensembles. Next, the technical implementation of graphics, audio and networking was discussed and Shoggoth examined with respect to three established taxonomies, and associated contextual considerations. Finally Shoggoth was evaluated by network musicians who drew upon their experience rehearsing and performing with the interface.

Chapter 6

Quick Live Coding Collaboration in the Browser

6.1 Lich.js

The previous two chapters covered *Yig, the Father of Serpents* and *Shoggoth*, which are two graphics focused interfaces. While these interfaces provide graphical representation of action and state, they also greatly restrict the number and types of decisions that are made during performances. The decision was made to make a new live coding language to explore what possible advantages that type of interface might have in a networked context. A live coding language can maintain some of the focus on displaying effort while also greatly increasing the "search space" of potential activity. After some initial experiments with more a Lisp like language, Haskell was instead chosen as the main inspiration for the language. The resulting language is named *Lich.js*, as a combination of the band name Glitch Lich and the suffix ".js", which is often used for javascript based projects.

Lich.js was created to achieve several goals including quick collaboration, graphics and audio live coding, a terse and expressive language, and a set of powerful but safe language features. Collaboration is the main priority and it is for this reason that the language is web based. The goal of Lich.js is to make collaboration fast and painless for everyone, from the most experienced network musicians to the casual interested party. Having the language running on the web allows for members to join a group and start making art without installing a single piece of software. The end goal is for a group of people to sit down at any computer and start making music together, as if the computer were like any other instrument, lacking the requirement of some special software pre-installed on the device. A video of a live coded *Lich.js* performance by Glich Lich at Ochiai Soup in Tokyo, Japan can be found at `https://www.youtube.com/watch?v=SIW1TsrGWz0`.

6.2 Design Philosophy

The core design philosophy for Lich.js is to have a language that facilitates quick live coding collaboration. While Lich.js was designed for Glitch Lich performances, it is also targeted at a wider audience than intended for Yig or Shoggoth. This required a balance between the previously mentioned Glitch Lich design requirements and the requirements of an instrument or system intended for broad use. To recap, the four outlined Glitch Lich design requirements are distribution, communication, virtual space, and autonomy. As with Yig and Shoggoth, Lich.js uses full local rendering to facilitate distributed performances. Lich.js also has chat support built directly into the web based IDE. Because Lich.js is designed for live performance this chat is scaled to a large font size, to be easily read when projected to an audience.

Even though Lich.js is a text based live coding environment, there are two characteristics of a virtual space in the design. The first is simply that the web based IDE has support for multiple code views displaying the active editing area for each performer. This also allows the players to see what the other members are doing in real–time, allowing for simple code sharing. Second, and more abstract, is the semantic space of Lich.js. These live coding performances use a shared global space for the performers. This means that variables, synth definitions, and patterns defined by one player are available to all the other players to reference directly in their own code.

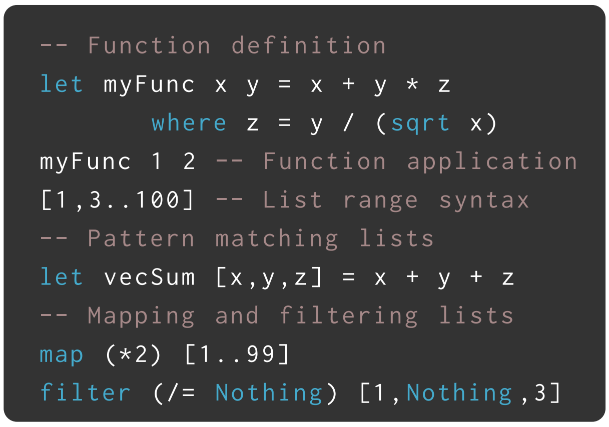
The final Glitch Lich design requirement is autonomy. In principle Lich.js is a turing complete language supporting all the standard features of a basic functional language. To fully utilize this language with more sophisticated and autonomous behavior would require a more developed set of libraries which the language currently lacks. There is support for some procedural techniques, but their range of expression is not that wide. Lich.js also supports feedback of various kinds, which is where most of the autonomy can be found. To date, feedback usage in Lich.js performances has been much simpler than the feedback networks in Yig.

Figure 6.1: Glitch Lich performance using Lich.js at an Algorave in Tokyo.

The Glitch Lich requirements need to be considered alongside the requirements of a live coding language targeted for larger public release. The core design principle was to allow for quick live coding collaboration. This basic principle informs other design decisions such as the necessity for easy synth and pattern definitions. Simplicity and terseness are both features of Lich.js that are meant to make collaborations faster to start and easier to edit and share code. Lich.js has many features, but the core features of rhythmic and melodic patterns, synth definitions, and live evaluation have all been streamlined, taking influence from other live coding languages, especially Tidal (McLean, 2011) and ixi lang (Magnusson, 2011). Sample based sound generation is equally as important as synthesis. While Lich.js syntax can be more verbose than both Tidal and ixi lang, Lich.js has more support for synthesis based sound editing. In general this has led to a design philosophy where efficiency and terseness are core principles, but also allowing some verbosity and complexity for features such as synthesis and visuals programming.

6.3 Language Design and Implementation

The consequence of choosing a web based design is that JavaScript now becomes the target language. The decision was made early on that the syntax provided by JavaScript is too verbose to allow for easy live coding. By creating a domain specific language, unique syntaxes can be created to make expressions terse and clear with respect to music and graphics. Lich.js is a pure functional language with syntax similar to Haskell, but unlike Haskell it is dynamically typed and compiles to JavaScript. Haskell-like syntax was chosen because it provides both an existing language as an established reference point and because the syntax is terse and declarative. It also provides some other useful syntax elements such as pattern matching, which allows function arguments and case statements to easily select and decompose objects by type. Lich.js also implements partial application, which is similar to currying in Haskell, and allows for functions to be passed a number of arguments less than the expected total. The return value is now a new function with that many fewer arguments, and with the curried arguments predefined in a new closure. Lich.js also widely uses a streaming operator >> for data flow, which takes inspiration from the |> forward pipe operator in F#. This operator fits well in a music language because it composes chains of computation that are expressed similarly to a signal path through a series of effects pedals or synth modules. For example "saw 440 >> lowpass 900 1 >> gain 0.1" expresses a saw oscillator feeding into a low pass filter and finally being lowered in volume.



```
-- Function definition
let myFunc x y = x + y * z
       where z = y / (sqrt x)
myFunc 1 2 -- Function application
[1,3..100] -- List range syntax
-- Pattern matching lists
let vecSum [x,y,z] = x + y + z
-- Mapping and filtering lists
map (*2) [1..99]
filter (/= Nothing) [1,Nothing,3]
```

Figure 6.2: Some example Lich.js code that demonstrates a similarity to Haskell.

Lich.js's lexer and parser are written in Jison (Carter, 2014), which compiles input strings of Lich.js into an abstract syntax tree. This syntax tree is then recursively traversed generating substrings of code which are concatenated together to produce the final target JavaScript code string. This JavaScript string is evaluated against a pre-built runtime that facilitates audio and graphics sequencing, and networking capabilities. The entire framework is hosted on a server and users simply visit a website to immediately begin collaborating. Immediacy is an important goal for the Lich.js design because it makes iteration and code dissemination incredibly simple for the ensemble. Juggling code bases can be difficult even for experienced ensembles, especially those using mixed operating systems. Lich.js is supported on Windows, Mac OSX, Linux, and can even be run on mobile devices such as tablets and phones.

A functional paradigm was chosen because it allows for highly modular and terse programs, two features that are useful when writing and using code in a real-time performance. Like Haskell, Lich.js enforces immutability of objects. This means that variables can not be rebound or mutated, including containers such as lists and dictionaries. By disallowing object mutation it becomes easier to reason about the behavior of a program, allowing for chains of pure functions to build up complex, but more predictable programs than imperative designs. This becomes especially important in networked coding contexts where collisions between code bases can easily cause unforeseen consequences.

Being a live coding language, it would be rather useless if there wasn't some way to allow for side effects in the system and Lich.js provides a few well defined ways to allow for these effects. The easiest is to use the interactive mode, which is the default mode when visiting a Lich.js website. In interactive mode, commands can be executed and global scope variables can be rebound by hand using a "let myVar = myValue" syntax similar to GHCI for Haskell. Functions or any other language construct still have no ability to change objects and so any mutation is directly requested by the user and only at global scope. For live coding this is very useful because changes to synth definitions and patterns are defined in the global scope. The user can now rely on the vast majority of their programs being created with pure functions. The other mode for writing code is the library mode which is similar to a basic Haskell file. In library mode, global variables don't require use of the "let" syntax but can only be defined once and never changed as well as anything else. Predefined user code can be compiled ahead of time with useful functions such as chord progressions, section changes, useful data structures, and so on, with the guarantees of immutability. In interactive mode, library files can be imported and called like any other system code.

For more complicated uses of state, Lich implements the State Monad as found in Haskell. The State Monad is a kind of wrapper that allows for state mutation to be emulated using a chain of pure computations. Regarding the type system, Lich.js is not a strict clone of Haskell, and eschews a strong type system . For this reason some semantics are more similar to Erlang or Scheme because of the ability to utilize mixed lists or other structures. This does introduce many more opportunities for confusion and unpredictability, offsetting some of the efforts of the pure functional design. Dynamic typing was chosen simply because it gives the user less to worry about while coding in the moment, which is useful for live coding, as evidenced by many of the other live coding languages using a similar type system. Additionally, enforcing type semantics would add another layer of computation during compilation which could be potentially introduce more audio glitches during performance.

6.4 Synths and Patterns

Web Audio is a relatively new API implemented in most major browsers including Firefox, Chrome, Safari, and Edge. The API provides a relatively low-level interface for creating real-time synthesis in pure JavaScript. The metaphor of a mixing console with effects channels is present throughout, lending itself well to use in web applications and games. Unfortunately the API is unwieldy for live coding because it is verbose and lacks many higher level features that you can find in other audio programming languages. For these reasons Lich.js compiles user created synth definitions into pure JavaScript code, generating the necessary node connections and managing unit generator life times. Currently the Web Audio API defines several oscillators (sine, triangle, saw, and square wave), filters (lowpass, highpass, bandpass, notch, and several shelf variants), a waveshaper, a limiter, buffer playback, convolution, mixing nodes, and FFT analysis. This is a good basis for any audio application, but pales in comparison to many other audio engines and as such requires augmentation. Lich.js currently supports all of the supplied audio types except for FFT analysis. Furthermore it includes a wide array of custom unit generators including noise generators, filters, distortion effects, reverb, bit crushing, decimation, buffer manipulation, envelopes, and a frequency shifter.

```
let s = osc >> filt >> dly >> gain 0.3 >> play
    where
        osc = (saw 80) * (square 3)
        lfo = sin 0.1 >> exprange 20 2000
        filt = lowpass lfo 10
        dly = delay (1/3) 0.9
```

Figure 6.3: Audio generation using the play method.

Lich.js also implements a robust synth definition syntax that allows for language constructs, such as pattern matching or conditional branching, to be used directly. This is possible because these synth definitions compile down to a graph of audio node connections in JavaScript and reference values in the same language. This contrasts with a language such as SuperCollider where synth definitions are walled off from many language semantics because the target language for audio processing is different. It should be noted that higher level language semantics in Lich.js such as conditionals won't run at audio rate in a synth definition, but instead will define the particular connections that are created inside the node graph upon compilation. One notable feature of the synth definition is the heavy use of the >> operator. This operator simply takes the left operand and applies it to the right operand. The syntax is simple but would not be easily implemented in other languages that don't utilize partial application. For this reason Lich.js synth definitions have a signal path like flow that declaratively describe transformations.

By using the *play* function, running synths can be generated from synth definitions, but using the generative pattern sequencers in Lich.js allows for more powerful control.

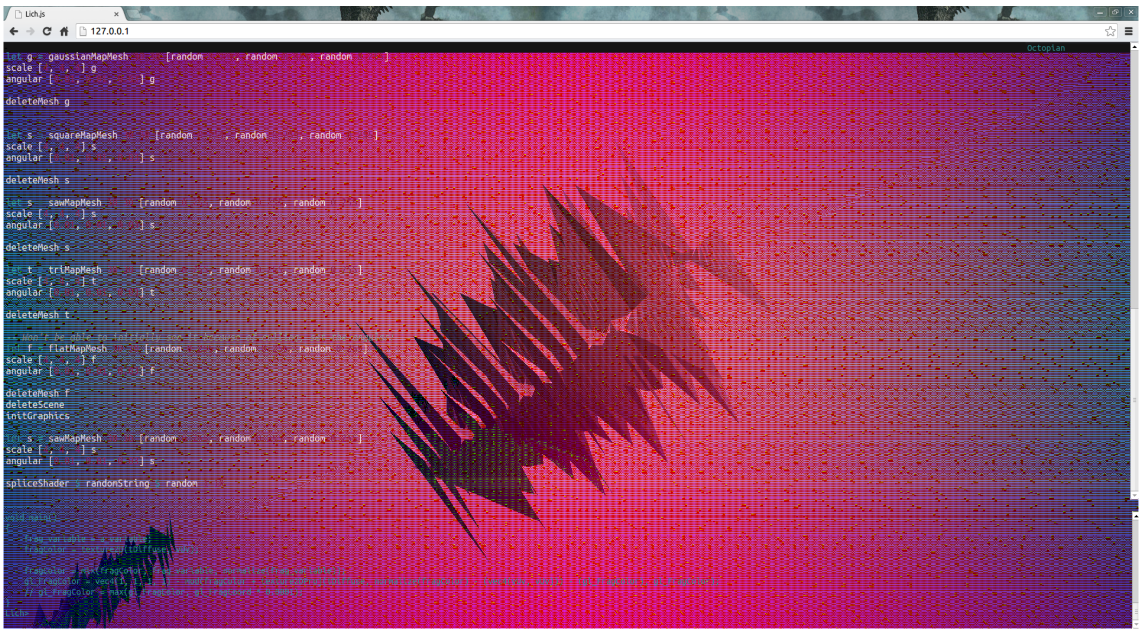
```
-- Synth definition with an argument named f
let m f => sin f >> delay 1 0.1 >> perc 1 0.1 1
-- Solo patterns sequence over argument values
lead ~> m 440 [660 990] 330 _ 220 _ 110 55
-- Synth definitions use the => operator
let b => tri 80 >> lowpass 320 1 >> perc 0 1 0.3
let s => white 1 >> bandpass 900 3 >> perc 0 1 0.2
-- Impulse patterns use a layout based syntax
drums +> b s [b b] s
```

Figure 6.4: Some expressions for synth and pattern generation.

Synth definitions that are used with patterns are defined using the => operator. Currently there are two different syntaxes for generating patterns in Lich.js. The first is impulse based patterns, such as a drum sequence, and is invoked using the +> operator. Impulse patterns define nested rhythms using variables and lists without the need for commas and with rests marked by the Haskell wildcard _ symbol. Rhythm duration modifiers can be added to the pattern to generate more complex expressions with a small amount of code. The other kind of pattern generator is the solo pattern invoked using the ∼> operator. Unlike impulse patterns, solo patterns only take a single synth variable and afterwards are supplied a sequence of values to call that synth with. This is useful for sequencing bass lines, melodies, or any function that requires an argument. Like impulse patterns, modifiers can be added to the sequencer, but here they modify the values passed to synth instead of durations.

6.4.1 Scheduling

The scheduler in Lich.js has been greatly improved upon since the original implementation. JavaScript timers can have a variation of tens to hundreds of milliseconds which is not reliable enough for an audio sequencer, and for this reason the original scheduling in Lich.js was uneven and inconsistent. In a massive revision the scheduler was rewritten to use a technique found in several other web audio applications (Wilson, 2013). The most important part of the technique is to recognize that Web Audio implementations are run in a different thread from the main user view and the Web Audio scheduler is highly accurate. The problem is finding how to leverage the accuracy of the Web Audio timer without having the standard JavaScript timers mar the fidelity. The solution is to write an event scheduler with a look ahead time. Because Web Audio Objects have an explicit start and stop time, if you can schedule events far enough ahead of time (100ms or so should suffice) then even if your scheduler has variance, the actual web audio event will still occur accurately. Furthermore the scheduler is more robust and can withstand higher variance in CPU activity from other parts of the program.



```
let g = gaussianMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] g
angular [0.01, 0.01, 0.01] g

deleteMesh g


let s = squareMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] s
angular [0.01, 0.01, 0.01] s

deleteMesh s

let s = sawMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] s
angular [0.01, 0.01, 0.01] s

deleteMesh s

let t = triMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] t
angular [0.01, 0.01, 0.01] t

deleteMesh t

-- Won't be able to initially see it because of culling, set the angular!
let f = flatMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] f
angular [0.01, 0.01, 0.01] f

deleteMesh f
deleteScene
initGraphics

let s = sawMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] s
angular [0.01, 0.01, 0.01] s

spliceShader $ randomString $ random 1 10


void main()
{
    frag_variable = a_variable;
    fragColor = texture2D(tDiffuse, vUv);

    fragColor = mix(fragColor, frag_variable, normalize(frag_variable));
    gl_FragColor = vec4(1, 1, 1, 1) - mod(fragColor + texture2DProj(tDiffuse, normalize(fragColor) - (vec4(vUv, vUv)) - (gl_FragColor), gl_FragColor);
    // gl_FragColor = max(gl_FragColor, gl_FragCoord * 0.0001);
}
Lich>
```

Figure 6.5: Generative visuals created with Lich.js

6.5 Graphics

Most of the graphics functionality in Lich.js is built upon the THREE.js (Cabello, 2010) library which is a WebGL based JavaScript library for 3D graphics. Although WebGL has been supported longer than Web Audio, the API is very low level, requiring tremendous effort to make something usable, let alone live coding. Pragmatism was the deciding factor for using THREE.js. Writing a new language, audio system, and client/server architecture were all tremendous tasks, but necessary to materialize the unique features required in Lich.js. WebGL on the other hand is older than Web Audio and THREE.js has well established itself as a stable library that implements most features that would be needed in a graphics based application. While THREE.js abstracts away most of the difficulties of OpenGL, it still requires more code than is feasible to write in real–time. For this reason most of the graphics work in Lich.js has been to take the implementation of THREE.js and build on top of it a collection of higher level functions and algorithms for live coding visuals.

Lich has several functions for 3D mesh generation from basic shapes such as spheres and boxes to complex generative algorithms. The functions take position and color data in addition to any other values they require, creating the mesh and directly drawing it in the scene. All meshes can have their position, rotation, angular and linear momentum, color, and scale manipulated in real-time. Furthermore, because the pattern sequencers described in the previous section support more than just audio functions, you can compose a definition that manipulates a mesh instead of audio. Beyond mesh manipulation, Lich.js has some preliminary code for shader generation using a mini-language called Splice. Splice is a very basic language that takes a string and translates every character into a GLSL function, then wraps up the result with some boiler plate code to create a new GLSL shader. The result is often glitchy with some surprising behavior. Passing native GLSL code directly is supported, but writing pure GLSL in real-time is cumbersome. At the moment there is no 2D Graphics API or support for textures. These are among several features that will be added in upcoming versions of the language.

```
let c = map (random 0) [255,255,255]
let s = sawMapMesh 20 20 c
s >> scale [7,7,7] >> angular [0.01,0.01,0.03]
spliceShader . randomString $ random 1 5
```

Figure 6.6: Lich.js code used to generate the scene in Figure 6.5

6.6 Networking

Networking was the greatest motivating factor for creating Lich.js with a web based implementation. The networking features currently supported are simple, but the focus has been to make the project stable and usable first, before adding more features. Networking in Lich.js uses a simple client and server architecture with the server written in Node.js (Joyent, Inc., 2012). There are three main networking features in the language: the networked IDE, chat based communication, and shared code execution. The Lich.js IDE is based on the ACE HTML5 code editor which supports syntax highlighting, matching bracket highlighting, and customizable keyboard bindings. The work on the IDE so far has served to create a completely automated user management experience. When a user visits a Lich.js site they are greeted with a request for a user name. The server receives the name and stores a cookie on the client computer so they don't have to reenter this information in the future. There is currently no password system so this is the extent of logging on. Once into the main Lich.js page they are presented with a code editor and a post view on the bottom. If another user joins the same site the code editing view is automatically split in half using a smooth transition animation, and now both users can see other's code. Originally the code editor was a single document, but this caused significant issues with unexpected code deletions and collisions. The current method affords many of the same features of the shared editor, namely the visibility of the code and efforts of the other users, but the networked code editors are read-only for everyone but the owner. A notable drawback of this approach is that it limits the number of collaborators to only about three or four before the editor sizes become too small to use reasonably.

A chat feature was added to the system because even though the post window could be used as a chat using print functions, it was easy to lose messages to system print out and was too small for an audience to read. The new chat feature prints chat messages very large and right justified over the entire window. The messages fade in and out and only last a few seconds. They are legible and invite attention without being too distracting. Finally, code is executed across the network. Whenever any user executes a function that function is sent across the network and executed for each client. While currently not implemented, it would be useful to have a more sophisticated implementation that uses time stamps with pre-calculated latency to execute changes synchronously across the system. Another concern is that while it is very simple to start using Lich, if an interested user wanted to create their own server, the procedure is more complicated. It requires the installation of Node.js and using some command line tools. A more elegant method needs to be developed to make independent servers easier to create for newer users.

6.7 Categorization

Categorizing Lich.js using the taxonomies mentioned in previous chapters will demonstrate some of the relationships between it and previous works. Given that Lich.js is a live coding language using a textual environment, a superficial consideration may treat it as very different from the heavily graphics oriented systems described in other chapters. In some regards this is true, however there are many meaningful similarities that are useful to consider as well.

Using Barbosa's Network Music classifications (Barbosa, 2003) immediately presents some interesting decisions regarding classification. Lich.js is hosted on an open public server, and receives multiple visits everyday from locations across the globe. These visits are isolated events, punctuated by occasional moments of overlap. It may be possible to consider these surreptitious meetings to be part of an asynchronous collaboration, and thus a case could be stated that Lich.js at least in part can be classified as an 'Asynchronous' system, per Barbosa's Taxonomy. This is not the case though, as all of the interactions in Lich.js are synchronized between users on server login and logout. User departures from the server leave no lasting effect for unknown future visitors to encounter. Lich.js can firmly be categorized as a 'Synchronous' network music system, but the 'Local' and 'Remote' axis is more open with Lich.js. Remote collaboration was one of the fundamental principles guiding the design and architecture of Lich.js. The language fully supports transcontinental efforts as evidenced by the multiple performances by Glitch Lich with members in North America, Europe, and Asia. While global collaborations are supported, local ones are not excluded. The barrier to entry may be slightly higher as this may require running an instance of the Lich.js server on a local host, but local collaboration is a completely functional capability of Lich.js. As such Lich.js can be categorized as a 'Synchronous' interaction system supporting 'Local' and 'Remote' locations per the taxonomy laid out by Barbosa. Barbosa's denotes this particular combination as a "Shared Sonic Environment."

Next is to consider Anrew Hugill's five internet music types. Hugill's categories are not plotted along dimensions and partly because of this their relationships to each other are less clearly defined. Instead they are presented as unique monolithic categories that allow little overlap. Considering each in turn, Lich.js cannot be claimed to create "Music that Uses the Network to Connect Physical Spaces or Instruments." The system still does not have live input support and doesn't even support MIDI input, so this can be safely ruled out. The type "Music that is Created or Performed in Virtual Environments, or Uses Virtual Instruments" is worth considering, but the particular interpretation of environments and instruments by Hugill suggests that this category is meant for music or systems that more closely attempt to emulate live or acoustic performance. Lich.js on the other hand is quite distant to this kind of work, and in fact seeks to create music more unique to the computer, given the live coding approach. The third type "Music that Translates into Sound Aspects of the Network Itself" does not even begin to hold true for Lich.js. While networking infrastructure is a fundamentally important to the behavior of the system, the end result is simply to as closely synchronize environments between players, and the network only imprints itself by the usual whims of packet transmission, packet loss, and the occasional disconnect. The strongest contender is "Music that Uses the Internet to Enable Collaborative Composition or Performance", which encapsulates both the performance and compositional capacity of Lich.js, as well as the basis in internet technologies. Lastly, "Music that is Delivered via the Internet, with Varying Degrees of User Interactivity" holds some validity, but the focus on delivery and end user interactivity makes this category more appropriate for traditional music sites such as Duckworth's *Cathedral* (Duckworth, 2001) than for the performer and composer focused functionality found in Lich.js. The three dimensions plot "Net Music" across the axes of 'Interactivity/Openness', 'Interplay with network characteristics', and 'Complexity/Flexibility.'

Weinberg's network configurations and topologies are incredibly varied, but as mentioned in earlier chapters, the vast majority of these configurations are rare. This pattern holds true with the evaluation of Lich.js which can be considered to be an example of the "Flower" topology. The "Flower" topology includes network configurations that are synchronous and centralized. In Lich.js every message travels first through the main server before being dispersed to the rest of the peers in the network, thusly it can be considered to be centralized. Lich.js is also synchronous because messages are sent and handled immediately with no structures built into the system for pan-session memory, storage, or permanency. This is in contrast to historical network music systems and pieces, including the works of the League Of Automatic Music Composers and the Republic, which can use more complicated or elaborate network configurations, often because these connections directly impact musical behavior. Interactions in Lich.js are synchronized and virtualized, and while these relationships can exists, often they are abstracted as variable references or function calls. The language design itself seeks to emphasize stateless and denotational formulations as opposed to state machine emulations.

My assessment of Weinberg's comparisons between so called "Novice IMN Systems" and "Expert IMN Systems" concluded that the infrastructure is too encumbered by the formulation of strict dichotomies. While Weinberg's categories have issues it is still useful to consider the question of just who is Lich.js for? A programming language for music may not be the first format that comes to mind when considering exoteric music collaboration. Still, Lich.js stands as the author's most concerted effort to make a general, open, and intuitive system for collaborative electronic music. Programming languages (especially higher level programming languages) are at their root an effort to express human ideas and mental abstractions. By using specialized syntax Lich.js attempts to simplify some of the difficulty in expressing musical structures as higher level abstractions.

Föllmer's Spatial Order of the Twelve Types of Net Music is a dense collection of various categories mapped across three dimensions. As opposed to Anrew Hugill's five internet music types there is a strong set of relationships established across the dimensions and because of this the various categories are more likely to overlap or contain both similarities and differences. Considering 'Interactivity/Openness' immediately presents an issue of category. Lich.js is a programming language and highly modular. For the performers and composers using Lich.js, the system is quite open and highly interactive. The utilization of Lich.js in a performance on the other hand is usually lacking any kind of audience or public interactivity. Still, the open server implementation acts as a public forum for day to day experimentation often with anonymous users. Föllmer's clusters themselves vary somewhat in the intended vantage point from which to consider the axes. Lich.js should be included near both the Algorithm and Installation cluster as well as the Instrument and Workshop cluster, which would situate it at a 3 or 4 along the 'Interactivity/Openness' axis (scale 1 through 5). The 'Interplay with network characteristics' axis is more clearly demarcated. As mentioned earlier, network characteristics have a limited impact on Lich.js collaborations however it is important to distinguish that live interactions are being networked, which is why it was as a 3 in this axis. Finally, the 'Complexity/Flexibility', which Föllmer uses to rate of Flash and Shockwave toys as being a rating of 1. Given the relationship between Flash and Web Audio as internet technologies, it may be tempting to group Lich.js closer to the Flash/Shockwave Soundtoys type, but the Turing complete language implementation differentiates Lich.js as having an incredibly deep capacity for complexity and flexibility. Still, the core language eschews some of the complexity and flexibility to focus on intuitive musical expressions, so I've considered Lich.js a 4 of 5 on the 'Complexity/Flexibility' axis. This particular configuration of an interactive and open system, with moderate interplay with network characteristic, but high amounts of complexity and flexibility plots Lich.js in a region of Föllmer's spatial order with no clearly established category or cluster. This may in fact be an artifact of the change of web technologies as historically similar systems such as the Flash and Shockwave Soundtoys had limited capabilities, but now more complex systems from a shared background can now be created, which was previously unexplored during the formulation of the established clusters.

Plotting Lich.js in Magnusson's epistemic dimension space reveals a shape that is different than the previous mappings. This in part due to the fact that the previous systems, including Max/MSP, Yig, and Shoggoth all put a focus on a graphical interface, where as Lich.js is a live coding language with textual input. Along the 'expressive constraints' axis, Lich.js is plotted moderately. Lich.js is a Turing complete functional language and for this reason has the capacity to emulate any other Turing complete state machine. While this is possible, the language is focused on streamlining improvisation and much of the more complicated functionality is hidden away in advanced features that lack an intuitive and simple API. As well, the 'autonomy' axis is marked at around 40 percent. There are built in functions to generate procedural sequences, and a library of filters and combinators to build up more complex and surprising behavior. These functions are all deterministic and even something as simple as feedback is not allowed within the Lich.js pattern system. The amount of autonomy is limited to a higher level description of mathematical evaluations and the system has no real capacity to make decisions beyond these formulations.

Regarding ‘music theory’, Lich.js has a large scale and tunings collection that can be used with melodic functions. Lich.js also has support for chord semantics, allowing for chord based sequences to be defined, and for melodic passages to use relative offsets from the root, or to specify chord voices for pitch. Lich.js lacks powerful theory based functions for describing more complex relationships and behaviors beyond that, such as larger scale structure, counterpoint, or rhythmic relationships. The ‘explorability’ axis is similar to the ‘autonomy‘ axis in that Lich.js has the capacity to produce complex and varied output, but the design of the language adds frictions to these kinds of explorations in favor of making fast changes to beat based tonal music easier and more intuitive. The ‘required knowledge’ axis reveals some of the potential shortcomings of the live coding approach for general purpose usage. The language was designed with simplicity as a key factor, however the choice of Haskell has proven to be confusing at times even for programmers, and additionally a certain amount of familiarity with music theory such as scales, chord progressions, and rhythmic structures is required to make more sophisticated expressions.

The ‘improvisation’ plotting demonstrates one of the strengths of Lich.js. Changes made to patterns are updated in real–time along with the changes of collaborators. This tight iteration loop combined with a plethora of generative functions allows for a continuous dialogue between the performers. For the same reasons the ‘generality’ is marked as low, because many of the features that make the system more easily used for improvisation also make Lich.js less useful for anything else, such as composition, or music that doesn’t focus on beat based tonal music. Finally, Lich.js is plotted high on the ‘creative/simulation’ axis as the system is an abstracted live coding language that eschews any effort to present the user with traditional instrumental input or even behavior.

6.8 Comparing Lich.js and Javascript Performance

It is useful to look at some metrics for how the code generated by the Lich.js parser performs in comparison to equivalent code written directly in JavaScript by hand. Before the tes was ran, it was hypothesized that there would be quite a disparity in performance. This is because the code generator produces code in a continuation passing style (CPS) and JavaScript is missing key optimizations that make CPS performant (Loitsch, 2007). The test setup consisted of creating four small examples of code that demonstrated different basic facilities of each language, such as binary operations, and case statements. The Lich.js code was compiled to JavaScript using the normal lexer/parser and the two versions of each test were run using jsPerf, an established website dedicated to JavaScript benchmarking (Bynens, 2010).

The results can be found in figure 6.8 (note: the scale is logarithmic and higher is better) and the test code used is listed in Appendix D.7 on page XL. The first test, calculating 9 digits of the Fibonacci sequence, demonstrates the widest difference. The Lich.js version only managed 352 operations per second while the direct JavaScript implementation came in with 1,965,147. This confirms my hypothesis, however the difference is larger

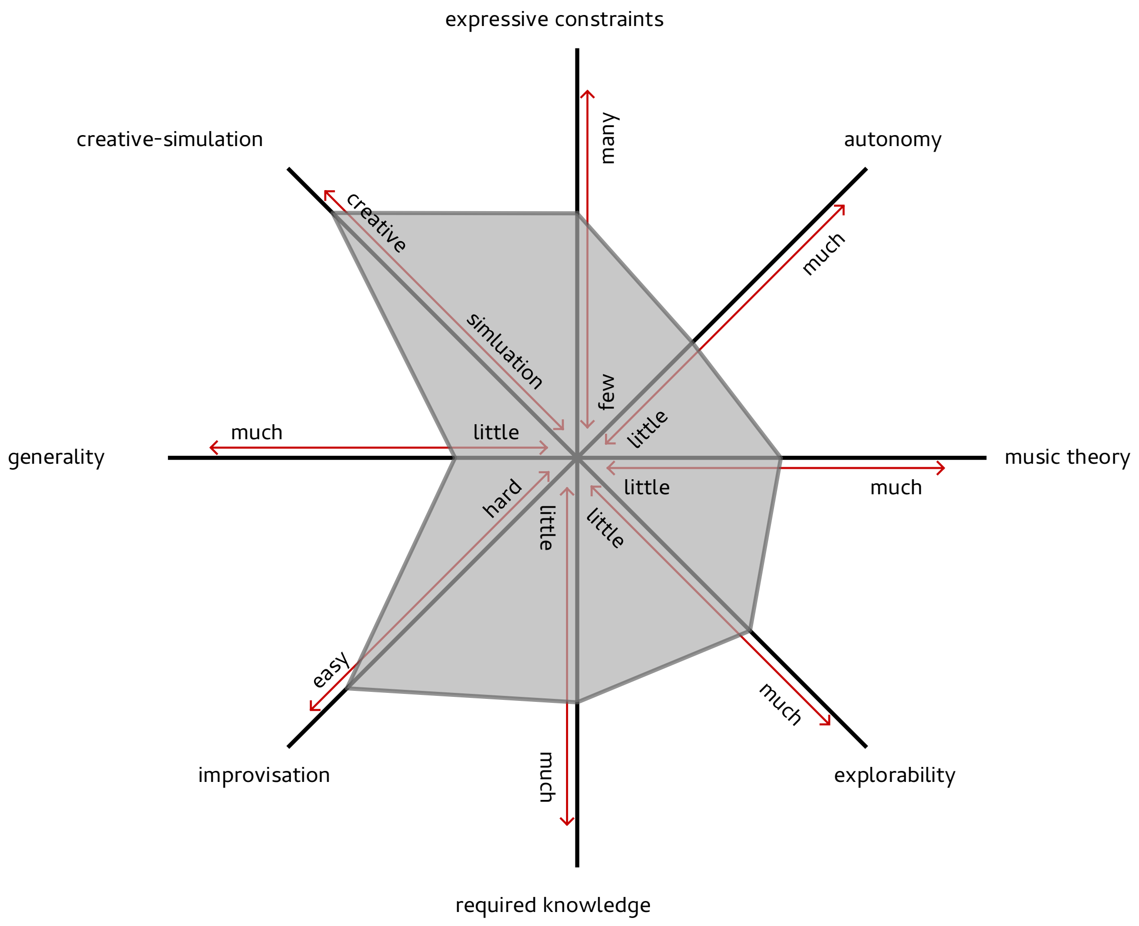


Figure 6.7: Lich.js in the epistemic dimension space.

than originally had postulated. The next test used case statements that values that fell all the way through to the default case. Case statements saw a 72% gap with 6,595,638 operations per second for Lich.js compared with 23,017,388 for JavaScript. The difference is large, but not nearly as large as in the Fibonacci sequence test. The third test was a simple arithmetic operation of “1 + 2 * 3". The binary operator sequence presented another tremendous disparity with 23,017,388 for Lich.js and 89,004,920 for the directly coded JavaScript. This result is interesting because the test was so basic. An optimization with the operators could have far reaching results in the performance of the entire language and will constitute further investigation. Finally the list test looked at list comprehension for a list of almost 10,000 items. This test saw a 46% difference between Lich.js and the hand written JavaScript. This is likely because the list comprehension syntax in Lich.js relies heavily on pure JavaScript primitives unlike the Fibonacci sequence test.

6.9 Design Summary and Future Work

The previous sections have argued that the continued difficulty of networking has stifled the creation and productivity of network bands and ensembles; that there is a need for a simpler approach for users of all experience and skill levels. Lich.js was designed in part to accomplish this goal, as a new live coding language in the burgeoning field of web based

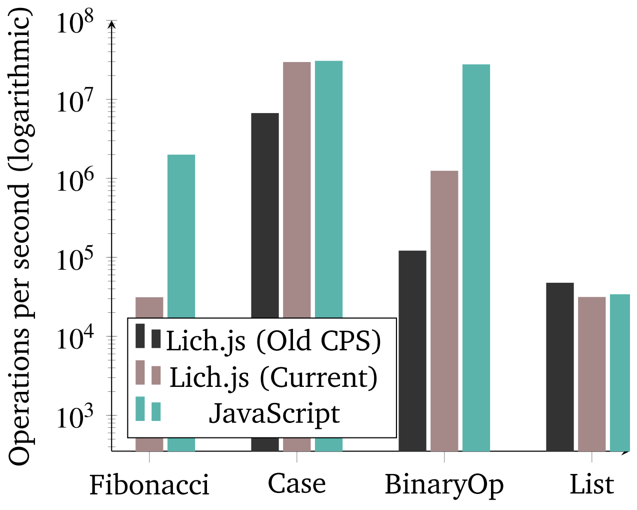


Figure 6.8: A comparison of the execution speed of Lich.js in continuation passing style, Lich.js as it is currently, and the equivalent JavaScript code.

languages. After surveying the current alternatives a justification was made for the need of yet another language; one based on collaboration from the beginning. Following the introduction, Lich.js was covered in detail, including the language syntax and semantics, synth definitions and pattern generation, graphics, and networking. Lich.js is open source and available on GitHub (McKinney, 2014a) but is still in development. New features are also being considered such as local area network clock synchronization, time stamped code execution, Open Sound Control responder implementation, and more graphics and audio functionality. Now that the design and basic analysis of Lich.js has been performed, a more formal user survey will be discussed in the next section that addresses in part how successful Lich.js is at achieving the design goals laid out previously.

6.10 User Evaluation

To evaluate Lich.js an online survey was created using the web based survey authoring site (SurveyMonkey, 2014). Ethics approval can be found in Appendix D.6 on page XXXIX. An open call for participants was announced on various social media sites such as Twitter (Twitter, 2014) and FaceBook (FaceBook, 2015), as well as on several computer music and art e-mail lists such as the SuperCollider list (McKinney, 2014d), and lurk.org (McKinney, 2014b). Users were directed to a web site (McKinney, 2014c) to use Lich.js and asked to answer an accompanying questionnaire after using Lich.js. Because Lich.js was developed using web technologies, a web format for the survey was chosen. This format allowed for anyone to access it from any computer regardless of operating system, and even supports mobile devices, although a keyboard is preferred given the requirement for programming. Compared to the Yig study, friction for participants to be involved was much lower. Participants did not have to have specific software installed beyond the Google Chrome web browser (Google, 2015) and could participate from any location and during any time of day.

The goal was to attract a larger pool of participants than the Yig study, but despite the low friction for participation only seventeen surveys were collected after several months of being posted. This is only one more than the number of participants in the Yig study, but there were several other advantages to the format. Participants were able to complete the survey completely anonymously, and did not have to be local to southern England to participate, which increases the potential for a wider variety of backgrounds for survey results. While this potential exists, it is important to note though the participants were a self selecting group participating in a web survey which directly impacts the potential audience. Participants are more likely be young and middle-aged adults, with high school or college degrees, and who hold jobs that pay salaries above the considered poverty line (Center, 2014). The survey and demo were only available in English, which excludes many potential visitors. Additionally Lich.js was featured on the Chrome Experiments (Google, 2014b) site, which increased traffic to the site from beyond the initial postings. No background information was requested in the survey so specific data points cannot be referenced, but given the sources of advertisement it is more likely that participants are familiar with computer music, live coding or recent web technologies such as Web Audio and WebGL.

6.10.1 Quantitative Analysis

The survey itself contained two sections, one for quantitative evaluation, and the second for qualitative. Section one included a series of Likert items where users were directed to "Rate the following statements with a number 1 through 7 according to how strongly you agree with them. 1 being strongly disagree and 7 being strongly agree." The statements for each Likert item are listed in Table 6.1. Likert responses were treated similarly to the Yig evaluation, using the R open source statistics environment (Crawley, 2012) for analysis. The Likert items are treated as ordinal data, using median, and IQR on individual items, forgoing any parametric analysis. First, each question will be addressed in turn, but it is important to note that too much importance shouldn't be placed on any individual Likert item, especially given the small sample size (17) of the data set. A stronger case about what the data suggests will be made by synthesizing the results of all of the Likert items as a resulting Likert scale. After addressing the quantitative data, the qualitative responses will be considered and how they might present potential themes and reasons for the attitudes that the quantitative results suggest.

Table 6.1: List of Likert item statements regarding *Lich.js*

Lich.js is useful for music collaboration.
I quickly became skillful with *Lich.js*.
It was difficult to communicate and collaborate.
The browser based implementation is useful.
I will use *Lich.js* in a future project.

The listing of Likert responses can be found in figure 6.9, which will be referred to throughout the rest of this section. First regarding "Lich.js is useful for music collaboration", the resulting responses have a median value of 5, suggesting a positive attitude towards the Lich.js collaborations. The interquartile range (IQR) is 2 representing a general consensus with the median, albeit not the strongest consensus. Next regarding "I quickly became skillful with Lich.js", the responses have a median value of 4, and an IQR of 2. This measurement indicates generally neutral attitude, although the responses for this item held the most normalized distribution around the neutral (4) response. This indicates that while Lich.js is scored as being useful for collaboration, that that usefulness does not directly correlate with ease of use. Continuing the neutral attitude is the item "It was difficult to communicate and collaborate". The scores have a median value again of 4 and an IQR of 2. Notably this item has the largest percentage (35%) of neutral responses, which might indicate that instead of having a large collection of neutral opinions, this may also include a large number of respondents who did not use the collaborative capabilities of Lich.js.

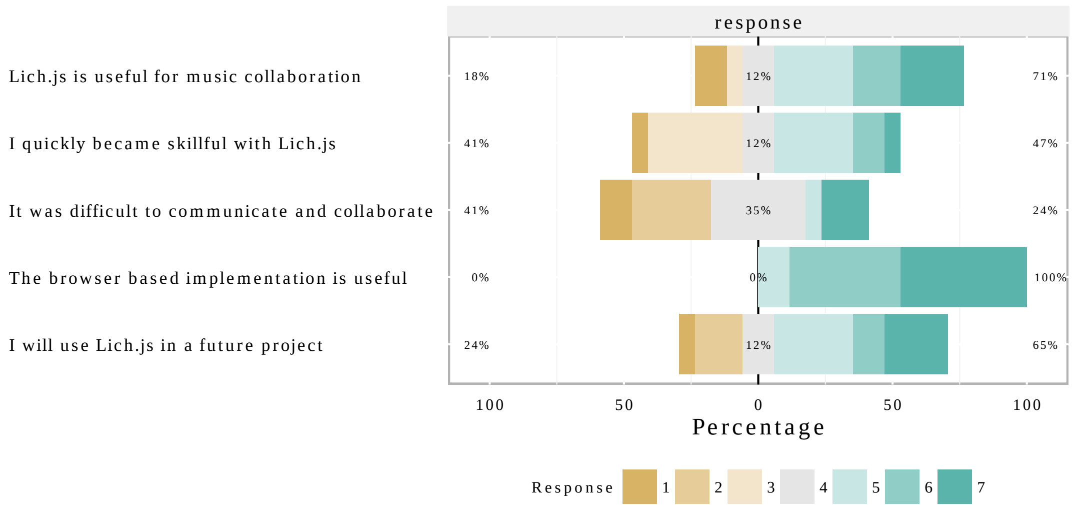


Figure 6.9: Lich.js User evaluation numbered responses.

Next, "The browser based implementation is useful" has a median value of 6 and an IQR of 1. This item has the highest median of the individual items as well as the greatest consensus on that value. Finally, "I will use Lich.js in a future project." has a median value of 5 and an IQR of 2. This has a very similar distribution to the first item, "Lich.js is useful for music collaboration", and although some responses of individuals differently between the two, the result as a whole is still generally positive. The results indicate that some individuals may consider Lich.js to be useful for collaboration but will not actually use it, and other don't think Lich.js is useful for collaboration, but do intend on using it in a future project. This suggestion holds with the indifferent response to "It was difficult to communicate and collaborate" and the highly positive response for "The browser based implementation is useful." Given these responses, the data indicates that Lich.js is considered to generally be a useful tool for collaboration, but that there is more excitement about a browser based music system than a collaborative music system.

Table 6.2: Sum and mean of Likert item responses for each participant

Sum of Likert item responses for each participant	26, 26, 26, 16, 27, 16, 22, 31, 16, 31, 27, 29, 23, 22, 26, 26, 27
Mean of Likert item responses for each participant	5.2, 5.2, 5.2, 3.2, 5.4, 3.2, 4.4, 6.2, 3.2, 6.2, 5.4, 5.8, 4.6, 4.4, 5.2, 5.2, 5.4

The sum for individual responses to each Likert item can be found in the first row of table 6.2, and the mean of these individual responses to all five items can be found in the second row of table 6.2. Note that this mean is applied to each individual respondent's results across all five Likert items, and not a median of the responses to a particular item. For this reason the data is evaluated as a scale of positive or negative attitudes to the Lich.js demo experience, and not as just ordinal data. The median value of the collection of means of attitudes is 5.2, with an IQR of 1. The mean value of individual means is 4.906, with a standard deviation of 0.957. These results indicate a generally positive, but not overly positive, attitude towards Lich.js and this result sees a high consensus. This consensus shows that indeed the synthesis of the individual items is a stronger indicator of actual attitudes than each individual response.

6.10.2 Qualitative Analysis

Now to better understand reasons behind the results from the quantitative analysis next there will be a thematic analysis of the responses to the second section of the survey. The participants were asked to answer five questions regarding their experience with Lich.js including "How did the language effect your ability to develop musical ideas?", "If you used the graphics functionality, how did the language effect your ability to develop visual ideas?", "Can you describe the most negative aspects of your experience?", "Can you describe the most positive aspects of your experience?", and "Are there any further comments you wish to make?" The answers to these questions were analyzed using thematic analysis using the methodology as defined by Braun and Clarke (Braun & Clarke, 2006).

According to Braun and Clarke, thematic analysis requires the researcher to make several decisions about the particular approach to the analysis first, and these decisions will impact the way that themes are developed and how closely tied to the specific data set the resulting analysis holds. The four decisions as outlined are "A rich description of the data set, or a detailed account of one particular aspect", "Inductive versus theoretical thematic analysis", "Semantic or latent themes", and "Epistemology: essentialist/realist versus constructionist thematic analysis." First, for "A rich description of the data set, or a detailed account of one particular aspect", a rich description was chosen as the goal was to find emergent themes from the data set, instead of introspecting about any specific theme. Next, with "Inductive versus theoretical thematic analysis" inductive analysis was chosen so that identified themes would be developed from the ground up and be tightly coupled to the data. Semantic themes were chosen over latent themes as the research is not necessarily concerned with finding larger cultural or ideological reasonings that form the basis for themes. Instead the goal is to specifically analyze the responses directly in relation to the evaluation of Lich.js. Finally, realist analysis was chosen over constructionist analysis because the responses are self selecting and anonymous for a topic with that isn't especially sensitive. For the purposes of the analysis it is more useful to assume a simple and largely unidirectional relationship between meaning and experience and language.

After making the required decisions regarding the particular approach to the thematic analysis, the next step is to familiarize yourself with the data set and after which to generate a series of codes in the data. These codes are like small atoms of meaning in the responses of the participants. These codes should provide a hook for identifying important thoughts, notions, and feelings while maintaining only a small amount of surrounding context. The resulting codes chosen for this analysis can be found in Appendix D.3 on page XXXV. The resulting collection is a list of 126 codes that range from a single word such as 'Fun' to nearly whole sentences such as 'Difficult to understand who produces which sound', and covers a wide range topics. Because the approach is data driven, no effort was made to go beyond these atoms, at their source of conception. These codes represented a slightly abstract but concise collection of the various feelings and opinions of the users, and identifies overlap and disparity between the responses.

After generating the code set, the next phase is to start grouping the codes together to find candidate themes and sub-themes. Often codes will reside in multiple themes and attention must be given to merging highly similar candidate themes or splitting up themes that begin to diverge or attempt to convey too much information. The generated sub-themes can be found in Appendix D.4 on page XXXVIII. This phase is exploratory and experimental, and while producing the list of candidate themes the analysis often resulted in contradictory statements. For instance three sub-themes emerged regarding the amount of information in the Lich.js demo. These themes include 'The demo had too much overwhelming material', 'The demo needed more material or was lacking information', and 'The demo was well made'. The contradictions here do not mean that the analysis has bad results, but in fact these disparities accurately reflect that the respondents held differing opinions regard how much information should be in a live coding demo, and this opinion can be orthogonal to other opinions, such as "There is interest in the continuing development of Lich.js."

Finally a series of major themes were generated by synthesizing these sub-themes, and identifying overlap and congruity, while discarding themes with little support. After applying this process four themes were developed:

- Web based music systems are desirable
- Anonymous live coding collaboration using Lich.js is difficult and potentially undesirable for some users
- The graphics implementation had several technical issues and many users were uninterested in this functionality
- Despite technical issues, the novelty and utility of Lich.js is recognized

These themes will be addressed in turn starting with "Web based music systems are desirable". The codes extracted from the data set in someways obscure this theme as their atomic structure points often at particular features or behavior. For instance, a common code in the data set is ‘fast implementation’, which at first might simply indicate the general notion that the Lich.js Read Evaluation Print Loop (REPL) model works well at producing live coded music. When collecting codes a more general set of themes emerged based off codes such as “Fast sound generation cool”, “Easy setup most positive”, “Easy to use”, “Browser implementation easy.”, and “Browser implementation is novel.” These codes coalesce to form a theme that the browser implementation lowers frictions for users significantly. This theme is born out by specific quotes in the feedback as well, such as “Great to work in browser - easy startup, and sample exercises start well”, “I got results immediately”, and “Definitely the possibility to interact with others through a browser (was the most positive aspects of the experience).”. Considering the quantitative results from earlier, this theme is substantiated by the results for the fourth Likert item “The browser based implementation is useful”. This particular item had the strongest reported positive attitude and also held the highest consensus of the individual Likert items. The data, both quantitative and qualitative, strongly suggests that browser based music systems are of high interest to the respondents.

Web Audio is still a novel technology, and it was one of the compelling reasons to develop Lich.js. For these reasons it is unsurprising that many of the respondents (of a web based survey no less) respond favorably to a browser based implementation for a live coding language. On the other hand, “Anonymous live coding collaboration using Lich.js is difficult and potentially undesirable for some users”, was a theme that organically emerged from the analysis. Many responses were coded with closely relating intents such as “Couldn’t easily edit with other users”, “Collaborative features unclear”, and “Chat worked different than expected.” While the language was designed to facilitate collaboration, it was still one of the more esoteric features implemented. There was a significant amount of new information presented, and having an extra layer of needing to then coordinate or sustain an evolving dialog with other users increases that complexity. Several users reported technical difficulties, some of which was the result of poor documentation in the demo such as the issues with chat, and other problems like the problems reported with the display of multiple terminals could be pointed out as a design flaw. Not all of it was negative, and many other responses ranged from mixed to positive, such as “There was somebody else on at the same time, and it was kind of cool when our sounds meshed well. Sometimes.” and “Definetly the possibility to interact with others through a browser (was the most positive aspect of the experience).”, and “The chat is very nice, the sync is very precise.”

Beyond these comments though was another notion that the collaborative features were uninteresting, or actively undesirable to participants. Some responses include statements such as “I never tried that feature.”, “I didn’t communicate with anyone while trying Lich.js.”, and “Having multiple people playing around trying to figure everything out while I was trying to figure things out is a little frustrating.” A potential issue is that live coding collaboration with anonymous users may be disruptive when users are still learning or are unprepared. This may not be an issue for users that actively seek out this interaction, by inviting a friend or by expecting visitors. One respondent notes “No response to chat, so not sure if they were aware of me.” and another “It might be nice if there was a way to mute the other people on the server during experimentation.”. The wide range of responses including indifference is corroborated by the quantitative responses to “It was difficult to communicate and collaborate.” This range could be explained by a multiplicity of factors including technical issues, lack of interest, organizational requirements for arranging collaborations, chance interactions with other visitors, a lacking implementation and design for collaborative features, and an abundance of other features presented to new users.

Another major theme is that “The graphics implementation had several technical issues and many users were uninterested in this functionality.” Lich.js is designed as an audio and visual live coding language, although the current state of the language greatly favors audio features over graphical ones. Beyond this, is that the graphics functionality appears to be much more fragile and hardware dependent than the audio functionality. Users reported that “The graphics functionality didn’t run in my computer.” and “Didn’t get to try these, although I tried - perhaps some server crash?” The strongest response was that the users simply weren’t interested in the feature or that they never actually used it. Many responses include sentiments such as “Didn’t use it. Probably want to mark this as optional.”, “Never got that far.”, and “Sorry, didn’t go there yet ... not usually my thing.” In fact, indifference towards or the lack of interest in the graphics functionality is the strongest theme in the qualitative feedback. There were some users who found the live coded visual support to be of note such as the user who states “The ability to utilize the pattern streaming system with the graphics was incredibly intuitive and useful.”

While there was a healthy amount of criticism or noted issues with Lich.js, there was still a large and consistent theme that “Despite technical issues, the novelty and utility of Lich.js is recognized” Many users expressed interest in the combination of web technologies and a Haskell based language: “It opened up a door I thought was closed in web dev, which is using a FP language to work with complex models on the web browser.” Other users appreciated the patterns and scales, stating “I enjoyed writing so easily patterns and chords.” or that “Pattern sequencing was easy and fun.” Additionally, comments regarding the collaborative and graphical features previously mentioned were also lauded by some respondents. This theme lines up with the overall Likert scale results which shows positive, but not overly positive attitudes towards Lich.js, and this attitude is shown to have a high consensus.

Chapter 7

Conclusion

7.1 Brief Review

There will be a brief review of the narrative of the thesis so far, before moving on to discuss findings and conclusions. A brief history of network music and research was summarized, which attempted to capture the diverse background for this field of research. A small informal survey of practicing network musicians followed, which searched for initial themes regarding liveness and networked collaborations.

The next three chapters all proposed a new network music interface, each designed to address challenges and opportunities using different approaches. First, *Yig, the Father of Serpents*, proposed a two dimensional space for creating and manipulating feedback network lattices, and the unique behaviors these feedback systems exhibited. A formal comparative user study ended the chapter that compared not only *Yig* to an established network music instrument, *Auracle*, but also explored the effects of co-location and distribution on networked music collaborations.

Next, *Shoggoth* was presented, a 3D networked music environment for creating and manipulating musical patterns, collaboratively. A small informal analysis followed, where Curtis McKinney and Cole Ingraham, two performers from the band Glitch Lich, provided commentary on the utility, execution, and design of *Shoggoth*. Finally, a web based live coding language, named *Lich.js*, was presented. Lich.js attacked the problem of networked interfaces from a different approach to the highly graphical interfaces previously discussed. Afterwards a survey on Lich.js was considered, using quantitative and qualitative data to formulate a discussion on the design and merit of Lich.js, while also exploring broader topics regarding networked collaborations.

7.2 Summary of Findings

The search conducted in this thesis explored four key questions:

1. How do distribution, virtual spaces, communication, and autonomy impact a network music interface and collaboration?
2. What characteristics define a successful live networked music interface and collaboration?
3. Do computer musicians consider interfaces with virtualized environments to embody those characteristics?
4. Do musicians have a preference regarding co-location and distribution in collaborations?

Each question will be considered in turn, drawing upon the findings of the previous chapters.

7.2.1 How do distribution, virtual spaces, communication, and autonomy impact a network music interface and collaboration?

The three systems presented as a part of this thesis, Yig, Shoggoth, and Lich.js, all support distribution, virtual spaces, communication, and autonomy. These features have large impacts on their design, utility, ease of use, stability, and the music they produce. For instance, distribution was a requirement for the band Glitch Lich to allow for the performances listed in Chapter 1. During the years that those performances were given the band had members in the USA, UK, and China. That music could not have been made without this feature. Yet, all the systems were greatly complicated by the necessity of supporting not only collaborative performances, but distributed performances. This required more CPU overhead to render full audio for all the nodes in the network. It also produced a more complicated and error prone system. Many of the issues that users reported in Chapters 4, 5, and 6 can be attributable to problems introduced by attempting to support distributed performances. It also introduced unique design decisions when using techniques like feedback with real–time input, which can produce divergent results in the distributed renderings.

Yig, Shoggoth, and Lich.js all had some kind of support for a virtual space that the performance resided in. Using a virtual space immediately creates relationships between the performers. Relationships such as position, ownership, and co–dependancy arise out of having these kinds of spaces where the players can collaborate using a system with some kind of sense of spatial physical relationships or behavior. User evaluation showed interest in this usage of virtual spaces. The avatars for other players was reported to help give a sense of presence for the other performers, as well as convey what their actions were.

Communication is an important part of any collaboration, but it can be complicated in networked and distributed performances. Yig, Shoggoth, and Lich.js all relied on a simple chat system for basic communication. Users in the evaluations often ignored or were indifferent towards this feature. Glitch Lich used the chat not only as a means of communicating with each other, but directly with the audience as well. Some performances would also pull comments from twitter and display them in the chat during the performance to allow a live conversation amongst the band and the audience about the unfolding performance. There are many other possible methods that could be explored regarding communication in networked collaborations. This is a highly under–researched part of network music performance.

Autonomy is an important part of the Glitch Lich aesthetic, and it has a unique impact on collaborative performances. By introducing autonomy into a network interface, the performance becomes a collaboration not only amongst the performers, but also with the system itself. Occasionally this can produce undesirable results, which some of the users in the evaluations noted. Other times it can create a dynamic environment, with pleasantly unexpected behaviors. Unlike distribution or communication, autonomy is much more specific to the aesthetic of Glitch Lich, and other network collaborators may find this feature does not suite their aesthetic and design goals.

7.2.2 What characteristics define a successful live networked music interface and collaboration?

Views regarding interfaces and collaborations varied wildly amongst the many musicians represented in the studies in the thesis. While there was no singular theme that dominated discussions, many smaller, complimentary, and occasionally opposing, emerged. Communication, embodiment, individuality vs. the group, co-discovery, ownership, privacy, virtual awareness, technical friction, and graphical representation were all important themes discussed by the various collaborators involved. Communication was a particularly strong theme, and performers consistently noted that non-verbal communication was an integral part of networked performances. Others commented that virtual mediation of visual cues was also useful. Chat systems were often mentioned, yet few of the responses felt very strongly about using it. Even musicians invoked in distributed collaborations would forgo typing in chat to instead directly engage the other performers musically.

Another important theme was the capacity for independence, personality, and a sense of ownership. This theme had a dynamic relationship with the performers. Study participants often mentioned the desire to have mute buttons or private places in the virtual space to work. Users felt guilt for accidental virtual destruction and others felt joy from their digital 'griefing'. Additionally, several musicians noted that a motivating factor for engaging in network music at all was to benefit from hearing and interacting with their collaborators personality, through the various interfaces and in the music. This is further substantiated by the desire for some groups to allow for individual interfaces that communicate across an ensemble meta-structure, communicating using a specification or according to emergent rules.

Yet there were also a sentiment that network music was the place for group dynamics, not displays of individual virtuosity. The use of gestural controllers was not popular, and instead basic keyboards, mice, and sliders, were noted as the core input systems for the queried network ensembles. This may be in part due to another strong theme, shared discovery. Users noted that some of their most memorable moments were surprising interactions. These interactions were sometimes chance overlapping of intended action in the interface by players that created unforeseen behavior. They were also emergent themes and sounds that combined in unpredicted ways. This ties closely with another strong theme, improvisation. Because the musicians enjoyed co-discovery, often the music being performed by these kind of ensembles incorporates a certain amount of 'openness', and for that reason systems and interfaces that can allow for fast reactions were noted as important characteristics, so as to increase the speed of development of the group.

Lastly technical friction and fragility was a consistent theme in all the studies. An interface with many bugs that crashes often make not only development slow, but rehearsal and performance more difficult. The ambitions of the various network bands occasionally out paced the stability of the software they use and produce. Furthermore these complications made it more difficult for less experienced users to engage with the medium.

7.2.3 Do computer musicians consider interfaces with virtualized environments to embody those characteristics?

This thesis does not attempt to answer the question broadly, but rather proposes three system with various kinds of virtualized environments, and evaluates whether these systems meet the criteria and aesthetics laid out in the previous section. Considering the results from the various studies as a whole users were generally excited by the technology, and were interested in the musical potential they presented. There were however many notable issues identified in the evaluations regarding the three systems. *Yig, the Father of Serpents* performed slightly better than *Auracle* in the quantitative analysis of the study results. Participants reported higher attitudes regarding collaboration, usefulness, intuitive design as well as overall assessment. In no category quantifiably measured did *Auracle* out perform *Yig*. Yet the qualitative analysis revealed a diversity of opinions regarding both systems. While most users did not enjoy making music with *Auracle* as much as *Yig*, there were participants who held this position. Additionally some users found *Yig* to be confounding or had issues with bugs. Still, many users found *Yig* to facilitate compelling collaborations and encourage co-discovery.

Shoggoth had a less rigorous evaluation, but the two network musicians still reported useful information from the perspective of advanced users. *Shoggoth* was lauded for the range of sounds it could produce, as well as for the ease of pattern manipulation. Yet, *Shoggoth* was also directly criticized for having technical problems and a limited input system. *Lich.js* saw an open survey for evaluation. The language was designed to allow for facilitating collaborations with low technical friction. To do so it leveraged web technologies, a feature that was highly regarded by the participants. Quantitative analysis marked *Lich.js* as scoring generally favorable attitudes from the participants. Qualitative analysis revealed that the specific features of the language itself were assessed more diversely, with many users not able to fully complete the material in the demo before filling out their survey responses. Some users reported problems understanding the functional programming basis, but many others noted the intuitive design of the pattern system in *Lich.js*.

7.2.4 Do musicians have a preference regarding co-location and distribution in collaborations?

The evaluation in Chapter 4 directly addressed the issues of co-location and distribution in networked music collaborations. Quantitative analysis showed that there was actually a slight bias for distributed collaborators to report positive attitudes regarding the collaboration and software. Given the small sample size and the narrow margin by which distribution led co-location, no conclusive result can be or will be claimed. The qualitative analysis showed that while certain individuals had biases regarding the locality of the collaboration, these biases did not measuredly manifest themselves amongst the practitioner populations at large. These findings are further substantiated by the commentary in Chapter 3. This commentary from experienced practitioners is highly critical of distributed performance, and yet not every comment is deriding. The utility of the technology to facilitate performances when otherwise impossible is noted, as well as the beneficial effects of negating performance anxiety. The question remains open, and will require further research to better understand.

7.3 Discussion and Future Work

Yig, the Father of Serpents, *Shoggoth*, and *Lich.js* represent significant efforts to create interactive systems with aesthetically driven design that present new opportunities for collaboration. Each of these have aspects that were more successful and aspects that were disappointing. Glitch Lich performed more with Yig than Shoggoth or Lich.js, in large part because it is reliable, generally sounds good, and makes collaboration easy. Yig can also be quite limiting in the range of expression and lacks sophisticated techniques for dealing with rhythm and harmony. Shoggoth is the most successful system of the three for combining a collaborative virtual space that is also visually striking. The rhythmic sophistication exceeds Yig, but the tools for harmonic relationships are quite primitive. Additionally, of the three, Shoggoth is the most error and crash prone, contributing to Glitch Lich preferring Yig for performances on occasion.

Lich.js has a much wider range of expression than Yig or Shoggoth, as well as much deeper systems for handling rhythm, harmony, and patterns. For a live coding language Lich.js can be slow to pivot towards large structural changes during performances. While the web audio implementation is serviceable, it is still incredibly more limited and slower than native based alternatives. To make matters worse, Web Audio still varies wildly in performance between implementations. During Lich.js development Firefox was found to be the second fastest fully compliant implementation of the major browsers, but slow enough to not be reasonably useful for an actual performance. The Chrome Web Audio implementation is much faster than Firefox, which is why Glitch Lich performances always use Chrome. These performances have been troubled by an issue in Chrome that was reported to the issue tracker over two years ago from the writing of this (`https://bugs.chromium.org/p/chromium/issues/detail?id=379753#`), but which is still open. This bug causes ScriptProcessorNode objects to leak, leading every Lich.js performance to eventually crash the browser tab after anywhere from twenty to forty minutes of average activity.

Given the realities of these three systems, there is still work to do. Web Audio is not yet mature enough, and network music requirements exceed the realities at the moment. Creating Lich.js has demonstrated the utility of pure functional programming for network music, but also the limitations of a pseudo–Haskell language. There are some significant shortcomings in Lich.js, such as the lack of a strong type system, no support for type-classes, and no compiler enforcement for monadic IO. These kinds of live, collaborative, audio–visual systems are difficult to make in any language, and a naive pseudo–Haskell implementation can be quite awkward. Functional Reactive Programming (FRP) has the potential to elegantly solve many of the issues that Yig, Shoggoth, and Lich.js suffer from in their implementation. FRP uses two main abstractions, Events and Behaviors, for modeling time–varying values. This approach allows for declarative programs to describe interactive systems that exist in time. This is a great fit for the kinds of systems like Yig and Shoggoth. Another advantage is that FRP could be used to create a more similar syntax and set of semantics for describing different domains, such as audio or graphics, which in the previous systems had very different implementation details.

Of the three systems, Lich.js made spontaneous collaboration the most trouble free, which can be supported by the fact that the public Lich.js server (`chadmckinneyaudio.com/lich`) has been running for over a year without any downtime. Having a group simply go to a website is much easier than the old Glitch Lich experience of finding IP addresses, setting ports, starting programs, tweaking router settings, and debugging issues. Future systems will need to find a way to make native systems this fluid for collaborations.

Beyond implementation details, there are still many opportunities left for creating more sophisticated frameworks for collaboration and relationships between performers. While Yig, Shoggoth, and Lich.s were focused on collaboration, they only explored a small fraction of the possibilities for how to utilize the unique affordances of real–time collaborative electronic music performances. Additionally, each of these systems had very primitive representations of the performers and their actions. Future work will explore how to create systems that take further advantage of virtual representation. Finally, a methodology for creating systems that are both expressive and agile needs further investigation. Yig, Shoggoth, and Lich.js each balanced expression and agility with varying success. Future work will explore how using an FRP based implementation with expanded bandwidth for modalities of input, and more sophisticated frameworks for group dynamics might leverage a wide range of expression while still allowing for sharp transitions and regular development during performances.

Bibliography

Aagaard, J. (2013). Slub, live. `https://twitter.com/JohsAagaard`. [Accessed May 28th, 2014].

Aaron, S. (2014). Overtone. `http://overtone.github.io/`. [Accessed May 28th, 2014].

Aaron, S., & Blackwell, A. F. (2013). From sonic pi to overtone: Creative musical experiences with domain-specific and functional languages. In *Proceedings of the First ACM SIGPLAN Workshop on Functional Art, Music, Modeling &#38; Design*, FARM '13, pp. 35–46.

Adam, T., Agafonova, N., Aleksandrov, A., Altinok, O., Sanchez, P. A., & Aoki, S. (2011). Measurement of the neutrino velocity with the opera detector in the cngs beam. *Minos*, *1109*(November), 1–24.

Adams, M. (2011). *Lee de Forest: King of Radio, Television, and Film*. Springer.

Alleweldt, F., Kara, S., Fielder, A., Brown, I., Weber, V., & McSpedden-Brown, N. (2012). Cloud Computing Study. `http://ec.europa.eu/information_society/activities/cloudcomputing/docs/cc_study_parliament.pdf`. [Accessed June 20th, 2012].

Anonymous (1909). Distributing music over telephone lines. *Telephony*, 699–701.

Apple Inc. (2014). Apple app store. `https://itunes.apple.com/us/genre/ios-music/id6011?mt=8`. [Accessed May 28th, 2014].

Auslander, P. (2008). *Liveness: Performance in a Mediatized Culture*. Routledge.

Barbosa, A. (2003). Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation. *Leonardo Music Journal, 13: 53–59*.

Barbosa, A. (2006). *Computer-Supported Cooperative Work for Music Applications*. Ph.D. thesis, Universitat Pompeu Fabra.

Bell, R. (2011). An interface for realtime music using interpreted haskell. In *Proceedings of the Linux Audio Conference*, LAC '11.

Bencina, R. (2005). Oscgroups. `http://www.audiomulch.com/~rossb/code/oscgroups/`. [Accessed May 2010].

Bencina, R. (2013). oscpack. `http://www.audiomulch.com/~rossb/code/oscpack/`. [Accessed May 29th, 2014].

Benedetti, W. (2012). Taipei assassins triumph in 'league of legends' world finals. `http://www.nbcnews.com/technology/ingame/taipei-assassins-triumph-league-legends-world-finals-1C6448579`. [Accessed May 29th, 2014].

Berio, L., & Dalmonte, R. (2007). *Intervista sulla musica*. Laterza.

Berlioz, H., Malherbe, C., & Weingartner, F. (1900). *Symphonie Fantastique ; And, Harold in Italy:*. Dover Music Scores Series. Dover Publications.

Birmingham Laptop Ensemble (2011). Manifesto. `http://bilensemble.wordpress.com/manifesto/`. [Accessed June 25th, 2012].

Bischoff, J. (2003). Aperture. `http://www.transjam.com/aperture/aperture_client.html`. [Accessed June 7, 2012].

Bischoff, J., Gold, R., & Horton, J. (1978). Music for an interactive network of microcomputers. *Computer Music Journal*, *2*(3), pp. 24–29.

Bittanti, M., & Quaranta, D. (2006). *Gamescenes: art in the age of videogames*. Saggistica d'arte. Johan & Levi.

Bligh, J., Jennings, K., & Tangney, B. (2005). Designing interfaces for collaborative music composition. In *International Conference on Multimedia, Image Processing and Computer Vision*, pp. 218–222 Madrid.

Borgeat, P., Ballweg, H., & Romero, J. (2012). Benoitlib. `https://github.com/cappelnord/BenoitLib`. [Accessed May 29th, 2014].

Boulanger, R. (Ed.). (2000). *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*. No. v. 1. Mit Press.

Bozkurt, B. (2011a). Circuli. `http://www.earslap.com/projectslab/circuli`. [Accessed June 20, 2012].

Bozkurt, B. (2011b). Otomata. `http://www.earslap.com/projectslab/otomata`. [Accessed June 20, 2012].

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology, 3(2): 77–101*.

Brinner, B. (1995). *Knowing Music, Making Music: Javanese Gamelan and the Theory of Musical Competence and Interaction*. University of Chicago Press.

Brodsky, J. (2013). Jsaxus. `https://github.com/jonbro/jsaxus`. [Accessed May 29th, 2014].

Brougher, K., Strick, J., Wiseman, A., & Zilczer, J. (2005). *Visual music: synaesthesia in art and music since 1900*. Thames & Hudson.

Brown, A. (2010). Network jamming: Distributed performance using generative music. In *Proceedings of the 2010 conference New Interfaces for Musical Expression*, pp. 283–286.

Brown, C. (2003). Eternal network music. `http://www.transjam.com/eternal/eternal_client.html`. [Accessed June 7, 2012].

Brown, C., & Bischoff, J. (2002). Indigenous to the Net: Early Network Music Bands in the San Francisco Bay Area. `http://crossfade.walkerart.org/brownbischoff/IndigenoustotheNetPrint.html`. [Accessed 2 August 2010].

Burk, P. (1997). Transjam. `http://www.transjam.com/`. [Accessed May 29th, 2014].

Burk, P. (1998). Jsyn - a real-time synthesis api for java. In *International Computer Music Conference*.

Burk, P. L. (2000). Jammin' on the web - a new client / server architecture for multi-user musical performance. In *International Computer Music Conference*.

Burkart, P., & McCourt, T. (2006). *Digital Music Wars: Ownership and Control of the Celestial Jukebox*. Rowman & Littlefield Publishers.

Buyya, R., Broberg, J., & Goscinski, A. (2011). *Cloud Computing: Principles and Paradigms*. Wiley.

Bynens, M. (2010). Jsperf. `http://jsperf.com/`. [Accessed May 29th, 2014].

Cabello, R. (2010). Three.js. `http://threejs.org/`. [Accessed May 29th, 2014].

Cabello, R. (2013). Glsl sandbox. `http://mrdoob.com/139/GLSL_Sandbox`. [Accessed May 29th, 2014].

Cáeres, J.-p., Hamilton, R., Iyer, D., Chafe, C., & Wang, G. (2008). To the edge with china: Explorations in network performance. In *ARTECH 2008: Proceedings of the 4th International Conference on Digital Arts*.

Carôt, A. (2004). *Musical Telepresence – A Comprehensive Analysis Towards New Cognitive and Technical Approaches*. Ph.D. thesis, Universität zu Lübeck Germany.

Carôt, A., Krämer, U., & Schuller, G. (2006). Network music performance (nmp) in narrow band networks. In *Audio Engineering Society Convention 120*.

Carter, Z. (2014). Jison. `http://zaach.github.io/jison/docs/`. [Accessed May 29th, 2014].

Casa, D. D., McDonald, S., Stutters, J., & Ryan, T. C. (2013). Livecodlab. `http://www.sketchpatch.net/livecodelab/index.html`. [Accessed May 29th, 2014].

Cascone, K. (2003). Grain, Sequence, System (three levels of reception in the performance of laptop music). In Kleiner, M. S., & Szepanski, A. (Eds.), *Soundcultures*. Suhrkamp.

Center, P. R. (2014). Pew Research Center Internet Project Survey. `http://www.pewinternet.org/data-trend/internet-use/latest-stats/`. [Accessed August 29th, 2015].

Chafe, C. (2003). Distributed Internet Reverberation for Audio Collaboration. In *Audio Engineering Society International Conference*.

Chafe, C. (2009). Tapping into the internet as a musical/acoustical medium. In *Contemporary Music Review*, Vol. 28, pp. 413–420.

Chafe, C., Wilson, S., Leistikow, A., Chisholm, D., & Scavone, G. (2000). A simplified approach to high quality music and sound over ip. In *In Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00*, pp. 159–164.

Chafe, C., Wilson, S., & Walling, D. (2002). Physical model synthesis with application to internet acoustics. In *IEEE - Signal Procesing Society. Proceedings of the International Conference on Acoustics, Speech and Signal Processing*.

Chew, E., Sawchuk, A., Tanoue, C., & Zimmermann, R. (2005). Segmental tempo analysis of performances in user-centered experiments in the distributed immersive performance project. In *Sound and Music Computing*.

Chung, B. (2013). *Multimedia Programming with Pure Data: A comprehensive guide for digital artists for creating rich interactive multimedia applications using Pure Data*. Community experience distilled. Packt Publishing.

Ciccarelli, P., & Faulkner, C. (2004). *Networking Foundations*. John Wiley & Sons.

Cockos Incorporated (2004). Ninjam. `http://www.cockos.com/ninjam/`. [Accessed February 6th 2012].

Collins, K. (2008). *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design*. Mit Press.

Cooperstock, J., & Spackman, S. (2001). The recording studio that spanned a continent. In *Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC'01)*, pp. 161– Washington, DC, USA. IEEE Computer Society.

Cramer, F. (2005). Software dystopia: Netochka Nezvanova - Code as cult. *Words Made Flesh: Code, Culture, Imagination*. [Accessed June 20, 2012].

Crawley, M. (2012). *The R Book*. Wiley.

Cunningham, M., Kluver, B., Tudor, D., Moog, B., Coker, C., Kompfner, R., & Riley, T. (2008). *John Cage: Variations VII/ 9 Evenings in Theatre and Engineering*. ArtPix, Experiments in Art and Technology.

Cycling '74 (2014). Max/MSP. `http://cycling74.com/`. [Accessed May 29th, 2014].

De Campo, A., & Rohrhuber, J. (2004). Waiting and Uncertainty in Computer Music Networks. In *Proceedings of the 2004 International Computer Music Conference*.

De Jong, J. (2006). *Collective Talent: a Study of Improvisational Group Performance in Music*. Amsterdam University Press.

Dean, R. (2003). *Hyperimprovisation: Computer-interactive Sound Improvisation*. A-R Editions.

Delaney, T., & Madigan, T. (2015). *The Sociology of Sports: An Introduction, 2d ed.* McFarland, Incorporated, Publishers.

DeSanctis, G., & Gallupe, R. B. (1987). A foundation for the study of group decision support systems. *Management Science*, *33*(5), 589–610.

d'Escriván, J. (2006). To sing the body electric: Instruments and effort in the performance of electronic music. *Contemporary Music Review*, *25*(1-2), 183–191.

Deutsch, D. (2012). *The Psychology of Music*. Elsevier Science.

Dinamoe Labs (2013). Plink. `http://dinahmoelabs.com/plink`. [Accessed August 1st, 2016].

Doornekamp, I. (2013). Worp. `http://worp.zevv.nl/#Livecoding`. [Accessed May 28th, 2014].

Dourish, P., & Bellotti, V. (1992). Awareness and coordination in shared workspaces. In *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pp. 107–114.

Duckworth, W. (2013). *Virtual Music: How the Web Got Wired for Sound*. Taylor & Francis.

Duckworth, W. (1999a). Cathedral An Interactive Work for the Web. In *International Computer Music Conference*.

Duckworth, W. (1999b). Making music on the web. *Leonardo Music Journal*, *9*, 13–18.

Duckworth, W. (2001). Cathedral. `http://www.monroestreet.com/Cathedral/main.html`. [Accessed June 7, 2012].

eJamming Inc (2010). eJAMMING AUDiiO. `http://ejamming.com/`. [Accessed May 29th, 2014].

Essl, G. (2011). Automated ad hoc networking for mobile and hybrid music performance. In *Proceedings of the International Computer Music Conference*, pp. 28–34.

FaceBook (2015). Facebook. `https://www.facebook.com`. [Accessed August 29th, 2015].

Fahie, J. (2011). *A History of Wireless Telegraphy: Including Some Bare-Wire Proposals for Subaqueous Telegraphs*. Cambridge Library Collection – Technology. Cambridge University Press.

Fernando Lindner Ramos, M. d. O. C., & Manzolli, J. (2003). Virtual studio: distributed musical instruments on the web. In *Brazilian Symposium on Computer Music*.

Fiebrink, R., Wang, G., & Cook, P. (2007). Don't forget the laptop: Using native input capabilities for expressive musical control. In *Proceedings of the 2007 Conference on New interfaces For Musical Expression*, p. 3.

Finch, A. (2014). *The Unreal Game Engine: A Comprehensive Guide to Creating Playable Levels*. 3dtotal Team.

Fisher, S. S. (1991). *Virtual Environments: Personal Simulations and Telepresence*. Meckler, Westport, CT.

Flanagan, D. (2006). *JavaScript: The Definitive Guide*. O'Reilly.

Florit, G. (2013). livecoding.io. `http://livecoding.io`. [Accessed May 29th, 2014].

Föllmer, G. (2005). Netzmusik: Elektronische, Äthetische und soziale Strukturen einer partizipativen Musik. *Organised Sound*, *10*(3), 185–192.

Fontan, J., & Goberna, D. (2013). Glsl sandbox. `https://glsl.heroku.com/e#17427.0`. [Accessed May 29th, 2014].

Freeman, J. (2005). Graph theory. `http://turbulence.org/Works/graphtheory/index2.html`. [Accesssed June 11, 2012].

Freeman, J. (2007). Graph theory: interfacing audiences into the compositional process. In *Proceedings of the 7th international conference on New interfaces for musical expression*, NIME '07, pp. 260–263 New York, NY, USA. ACM.

Freeman, J., Varnik, K., Ramakrishnan, C., Neuhaus, M., Burk, P., & Birchfield, D. (2005). Auracle: a voice-controlled, networked sound instrument. *Organised Sound*, *10*(3), 221–231.

Galenson, D. (2009). *Conceptual Revolutions in Twentieth-Century Art*. Cambridge University Press.

GMBH, D. M. (2012). Digital Musician. `http://www.digitalmusician.net/`. [Accessed May 29th, 2014].

Google (2014a). Google play store. https://play.google.com/store/search?q=music&c=apps. [Accessed May 28th, 2014].

Google (2014b). Lich.js, Chrome Experiments Feature. https://www.chromeexperiments.com/experiment/lichjs. [Accessed August 29th, 2015].

Google (2015). Chrome. https://www.google.com/chrome/. [Accessed August 29th, 2015].

Google Inc. (2012). V8 engine. http://code.google.com/p/v8/. [Accessed June 20th, 2012].

Green, T., & Petre, M. (1996). Usability analysis of visual programming environments: A cognitive dimensions framework. *Journal of Visual Languages Computing*, *7*(2), 131–174.

Gresham-Lancaster, S. (2007). Is there no there there? video conferencing software as a performance medium. In *Music in the Global Village Conference*.

Griffiths, D. (2014). Fluxus. http://www.pawfal.org/fluxus. [Accessed May 27th, 2014].

Grover, C. (2011). *Flash CS5.5: The Missing Manual*. O'Reilly Media.

Grudin, J., & Poltrock, S. E. (1991). Computer-supported cooperative work and groupware. tutorial notes. In *Conference on Human Factors in Computing Systems*.

Grudin, J., & Poltrock, S. (2011). Taxonomy and theory in computer supported cooperative work. *Computer, S.W. Kozlo*(Oxford Univ. Press).

Gu, X., Dick, M., Kurtisi, Z., Noyer, U., & Wolf, L. (2005). Network-centric music performance: Practice and experiments. *IEEE Communications Magazine*, *43*, 86–93.

Gu, X., Dick, M., Noyer, U., & Wolf, L. (2004). Nmp - a new networked music performance system. In *Global Telecommunications Conference Workshops*, pp. 176 – 185.

Hajdu, G. (2004). Composition and improvisation on the net. In *Sound and Music Computing Conference*.

Hamilton, R. (2007a). Maps and legends: Designing fps-based interfaces for multi-user composition, improvisation and immersive performance. In Kronland-Martinet, R., Ystad, S., & Jensen, K. (Eds.), *Computer Music Modeling and Retrieval. Sense of Sounds, 4th International Symposium, CMMR 2007, Copenhagen, Denmark, August 27-31, 2007. Revised Papers*, Vol. 4969 of *Lecture Notes in Computer Science*, pp. 478–486. Springer.

Hamilton, R. (2007b). Maps and legends: Fps-based interfaces for composition and immersive performance. In *Proceedings of the 2012 International Computer Music Conference*, pp. 344–347.

Harris, J. (2011). *Globalization and Contemporary Art*. John Wiley & Sons.

Heiland-Allen, C. (2012). Clive. http://mathr.co.uk/blog/2012-12-25_clive.html. [Accessed May 28th, 2014].

Hersent, O., Boswarthick, D., & Elloumi, O. (2011). *The Internet of Things: Key Applications and Protocols*. John Wiley & Sons.

Holmes, T., & Holmes, T. (2002). *Electronic and Experimental Music: Pioneers in Technology and Composition*. Media and Popular Culture Series. Routledge.

H.P. Lovecraft (1931). *At The Mountains of Madness*. Arkham House.

Hugill, A. (2005). Internet music: An introduction. *Contemporary Music Review*, *24(6): 429–437*.

Hyde, A., & Harger, H. (1998). Radio astronomy. `http://radio-astronomy.net/`. [Accessed June 21st, 2012].

Iber, M. (1999). Integer (internet generated radio). `http://transmissionarts.org/work/fc0atb`. [Accessed June 7, 2012].

Ivanoff, I., & Jimenez, J. (2006). Flaxus. `http://www.i2off.org/flaxus/screen.html`. [Accessed May 29th, 2014].

J. Rohruber, A. de Campo (2011). The republic quark. `https://github.com/supercollider-quarks/Republic`. [Accessed May 29th, 2014].

Jin, D. (2010). *Korea's Online Gaming Empire*. Mit Press.

Johansen, R. (1988). *GroupWare: Computer Support for Business Teams*. The Free Press, New York, NY, USA.

Jones, F. (2006). The alphorn: Revival of an ancient instrument. *The Consort, Journal of the Dolmetsch Foundation*, *62*, 40–62.

Jordà, S. (1999). Faust Music On Line (FMOL) – An approach to Realtime Collective Composition on the Internet. *Leonardo Music Journal*, pp. 5–12.

Jordà, S. (2002). FMOL: Toward user-friendly, sophisticated new musical instruments. *Comput. Music J.*, *26*(3), 23–39.

Jordà, S. (2009). The reactable: Tabletop tangible interfaces for multithreaded musical performance. *Revista KEPES*, *5(14): 201–223*.

Joyent, Inc. (2012). Node.js. `http://nodejs.org/`. [Accessed June 20th, 212].

Kac, E. (1996). Ornitorrinco and Rara Avis: Telepresence Art on the Internet. *Leonardo*, *29*(5), 389–400.

Kac, E. (1997). Telepresence art. `http://ekac.org/telepresence.art._94.html`. [Accessed 11th, 2012].

Kac, E. (2005a). Satellite Art – An Interview with Name June Paik. *DIVA: Digital and Video Art Fair*. Originally published in Portuguese in the newspaper O Globo, Rio de Janeiro, Brazil, in July 10, 1988.

Kac, E. (2005b). *Telepresence & Bio Art: Networking Humans, Rabbits, & Robots*. Studies in Literature and Science. University of Michigan Press.

Karplus, K., & Strong, A. (1983). Digital synthesis of plucked-string and drum timbres. *Computer Music Journal*, *7*, 43–55.

Khronos Group (2013). Webgl - opengl es 2.0 for the web. `http://www.khronos.org/webgl/`. [Accessed May 29th, 2014].

Kickstarter Inc. (2014). Kick starter. `https://www.kickstarter.com/`. [Accessed May 28th, 2014].

Kirk, R., & Hunt, A. (1999). *Digital Sound Processing for Music and Multimedia*. Music Technology Series. Focal Press.

Kleimola, J. (2006). Latency issues in distributed musical performance. *Seminar*.

Knotts, S., & Hutchins, C. (2013). Network music festival. Information available at: `http://networkmusicfestival.org/`. [Accessed May 29th, 2014].

Koka, P., McCracken, M. O., Schwetman, H., Zheng, X., Ho, R., & Krishnamoorthy, A. V. (2010). Silicon-photonic network architectures for scalable, power-efficient multi-chip systems. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pp. 117–128 New York, NY, USA. ACM.

Konstantas, D., Orlarey, Y., Carbonel, O., & Gibbs, S. (1999). The distributed musical rehearsal environment. *Multimedia, IEEE*, *6*(3), 54 –64.

Konstantas, D. (1998). Overview of a telepresence environment for distributed musical rehearsals. In *Proceedings of the 1998 ACM symposium on Applied Computing*, SAC '98, pp. 456–457 New York, NY, USA. ACM.

Konstantas, D., Orlarey, Y., Gibbs, S., Carbonel, O., Moulin, J., Lyon, F., & Augustin, D. S. (1997). Distributed musical rehearsal. In *Procedings International Computer Music Conference*, No. 95.

Kurtisi, Z., Gu, X., & Wolf, L. (2006). Enabling network-centric music performance in wide-area networks. *Commun. ACM*, *49*(11), 52–54.

La Rosa, J. E. O. (2008). To un-button: Strategies in computer music performance to incorporate the body as re-mediator of electronic sound. Master's thesis, University of California, San Diego.

Lambert, P. (1997). *The Music Of Charles Ives*. Composers of the Twentieth Century Serie. Yale University Press.

Latta, C. (1991a). Notes from the netjam project. *Leonardo Music Journal*, *1*(1), pp. 103–105.

Latta, C. (1991b). Notes from the NetJam Project. *Computer Music Journal*, *15*.

Lazzaro, J., & Wawrzynek, J. (2001). A case for network musical performance. In *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '01, pp. 157–166 New York, NY, USA. ACM.

Lee, S. W., & Essl, G. (2013). Live coding the mobile music instrument. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pp. 28–34.

Leman, M. (2008). *Embodied Music Cognition and Mediation Technology*. Mit Press.

Levin, G., Gibbons, S., Shakar, G., Sohrawardy, Y., Gruber, J., Lehner, J., Schmidl, G., & Semlak, E. (2001). Dialtones (A Telesymphony) Final Report. `http://www.flong.com/projects/telesymphony/`. [Accessed June 9, 2012].

Li, Q., Jinmei, T., & Shima, K. (2007). *IPv6 Core Protocols Implementation*. The Morgan Kaufmann Series in Networking. Morgan Kaufmann.

Liljedahl, J. (2014). Kymatica software. `http://kymatica.com/Software/Software`. [Accessed May 28th, 2014].

Loeliger, J., & McCullough, M. (2012). *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. O'Reilly Media.

Loitsch, F. (2007). Exceptional continuations in javascript. In *Proceedings of the 2007 Workshop on Scheme and Functional Programming*, pp. 37–46.

Magnusson, T. (2010). An Epistemic Dimension Space for Musical Devices. In *Proceedings of the 2010 conference on New interfaces for musical expression*, pp. 43–46.

Magnusson, T. (2011). The ixi lang: A supercollider parasite for live coding. In "*Proceedings of the International Computer Music Conference*", pp. 198–200.

Magnusson, T., & Mendieta, E. H. (2007). The acoustic, the digital and the body: a survey on musical instruments. In *Proceedings of the international conference on New Interfaces for Musical Expression*, NIME '07, pp. 94–99. ACM.

Marinescu, D., & Marinescu, G. (2011). *Classical and Quantum Information*. Academic Press. Academic Press.

McCartney, J., et al. (2016). SuperCollider. `http://supercollider.github.io/`. [Accessed August 1st, 2016].

McIntosh, T., & Madan, E. (2012). Silophone. `http://www.silophone.net/`. [Accessed June 25, 2012].

McKinney, C. (2012). Yig, the Father of Serpents. `https://github.com/ChadMcKinney/Yig`. [Accessed May 29th, 2014].

McKinney, C. (2013a). libsc++. `https://github.com/ChadMcKinney/libscpp`. [Accessed May 29th, 2014].

McKinney, C. (2013b). Shoggoth. `https://github.com/ChadMcKinney/Shoggoth`. [Accessed May 29th, 2014].

McKinney, C. (2014a). Lich.js. `https://github.com/ChadMcKinney/Lich.js`. [Accessed May 29th, 2014].

McKinney, C. (2014b). Lich.js beta release and evaluation. `http://lurk.org/groups/livecode/messages/topic/7ehxjpBLXfBW9oY8pC4kYY`. [Accessed August 29th, 2015].

McKinney, C. (2014c). Lich.js demo. `www.chadmckinneyaudio.com/lich`. [Accessed August 29th, 2015].

McKinney, C. (2014d). Ot: Lich.js beta release and evaluation. `http://article.gmane.org/gmane.comp.audio.supercollider.user/108424/match=lich+js`. [Accessed August 29th, 2015].

McKinney, C. (2014e). Quick Live Coding Collaboration In The Web Browser. In *Proceedings of New Interfaces for Musical Expression*.

McKinney, C., & Collins, N. (2012a). Liveness In Network Music Performance. In "*Proceedings of the Live Interfaces: Performance, Art, Music Symposium*".

McKinney, C., & Collins, N. (2012b). Yig, the Father of Serpents: A Real-Time Network Music Performance Environment. In *Proceedings of the Sound and Music Computing conference*.

McKinney, C., & Collins, N. (2013). An Interactive 3D Network Music Space. In *Proceedings of New Interfaces for Musical Expression*.

McKinney, C., McKinney, C., O'Brien, B., & Ingraham, C. (2012). Glitch Lich: Lessons learned from the creation of a transcontinental laptop quartet. In *The Symposium on Laptop Ensembles and Orchestras*, pp. 116–122.

McKinney, C., & McKinney, C. (2012). Oscthulhu: Applying video game state based synchronization to network computer music..

McLean, A. (2011). *Artist-Programmers and Programming Languages for the Arts*. Ph.D. thesis, Department of Computing, Goldsmiths, University of London.

McLean, A., Griffiths, D., Collins, N., & Wiggins, G. (2010). visualisation-of-live-code. In *Proceedings Electronic Visualisation of the Arts*, pp. 26–30.

McLean, A., & Wiggins, G. (2011). Texture: Visual notation for the live coding of pattern. In *Proceedings of the International Computer Music Conference*, ICMC '11.

McReynolds, T., & Blythe, D. (2005). *Advanced Graphics Programming Using OpenGL*. The Morgan Kaufmann Series in Computer Graphics. Elsevier Science.

Microsoft (2014). Skype. `http://www.skype.com/`. [Accessed May 29, 2014].

Miell, D., MacDonald, R., & Hargreaves, D. (2005). *Musical Communication*. Oxford University Press, USA.

Mieszkowski, K. (2002) `http://www.salon.com/2002/03/01/netochka/`. [Accessed June 20, 2012].

Miletto, E. M., Pimenta, M. S., Bouchet, F., Sansonnet, J.-P., & Keller, D. (2011). Principles for music creation by novices in networked music environments. *Journal of New Music Research*, *40*(3), 205–216.

Mogees Ltd. (2014). Mogees. `http://mogees.co.uk/`. [Accessed May 28th, 2014].

Mojang (2013) `https://minecraft.net/`. [Accessed May 29th, 2014].

Morris, J. M. (2008). Structure in the Dimension of Liveness and Mediation. *Leonardo Music Journal*, 59–61.

Mothersele, D. (2014). Cyril. `http://cyrilcode.com/index.html`. [Accessed May 28th, 2014].

Myspace (2012). Myspace. `http://www.myspace.com/`. [Accessed June 20th, 2012].

Navarro-Prieto, R., & Cañas, J. (2001). Are visual programming languages better? the role of imagery in program comprehension. *International Journal of Human-Computer Studies*, *54*(6), 799–829.

Networks, P. (2011). Pando networks releases global internet speed study. `http://www.pandonetworks.com/Pando-Networks-Releases-Global-Internet-Speed-Study`. [Accessed Jun 11, 2012].

Neuhaus, M. (2004a). Auracle. `http://www.auracle.org/`. [Accessed May 29th 2014].

Neuhaus, M. (2004b). The broadcast works and audium. `http://www.auracle.org/docs/Neuhaus_Networks.pdf`. [Accessed June 25, 2012].

Neuhaus, M. (2004c). The broadcast works: Radionet. `http://www.kunstradio.at/ZEITGLEICH/CATALOG/ENGLISH/neuhaus2c-e.html`. [Accessed June 7th, 2012].

New York Miniaturist Ensemble (2005). Collaborative Composition Website. `http://nyme.org/collaborative.html`. [Accessed June 20th, 2012].

Nezvanova, N. (2000). The internet, a musical instrument in perpetual flux. *Comput. Music J.*, *24*(3), 38–41.

Nezvanova, N., & Föllmer, G. (2002). Interview with Netochka Nezvanova via email, February 2002. `http://www.hudba.de/interviews/Interview_Netochka.pdf`. [Accessed June 25, 2012].

Nicholls, D. (2002). *The Cambridge Companion to John Cage*. Cambridge Companions to Music. Cambridge University Press.

Nimtz, G., Heitmann, W., Roy-Brehonnet, F., & Jeune, B. (1997). *Superluminal Photonic Tunneling and Quantum Electronics*. Progress in quantum electronics. Elsevier Science.

Nunamaker, J. F., Dennis, A. R., Valacich, J. S., Vogel, D. R., & George, J. F. (1991). Electronic meeting systems to support group work. *Communications of the ACM*, *34*(7), 40–61.

Nyman, M. (1999). *Experimental Music: Cage and Beyond*. Music in the Twentieth Century. Cambridge University Press.

Obermeyer, F. (2013). Livecoder. `http://livecoder.net`. [Accessed May 29th, 2014].

Oh, J., Herrera, J., Bryan, N. J., Dahl, L., & Wang, G. (2010). Evolving the mobile phone orchestra. In *New Interfaces for Musical Expression*.

Oliver, B., Pierce, J., & Shannon, C. (1948). The philosophy of pcm. *Proceedings of the IRE*, *36*(11), 1324 – 1331.

Oliver, J., & Pickles, S. (2002). qa3pd. `http://julianoliver.com/q3apd/`. [Accessed May 29th, 2014].

Orchestra, L. (2012). Laptop orchestra. `http://laptoporchestra.net/`. [Accessed June 25th, 2012].

Osthoff, S. (2005). *From Mail Art to Telepresence: Communication at a Distance in the Works of Paulo Bruscky and Eduardo Kac*. MIT Press.

Packard, H. (2009). Cense. `http://www.hpl.hp.com/news/2009/oct-dec/cense.html`. [Accessed June 20th, 2012].

Paik, N. J. (1997). *Nam June Paik: videa 'n' videology, 1959-1973*. Tova Press.

Paper, A. W., Council, T., Bargar, R., Church, S., Systems, T., Keislar, D., Fish, M., Pavo, B. M., Microsoft, B. N., & Pennycook, B. (1998). Networking audio and music using internet2 and next-generation internet capabilities. In *Audio Engineering Society*.

Pasachoff, N. (1996). *Alexander Graham Bell: Making Connections*. Oxford University Press, USA.

Perevalov, D. (2013). *Mastering openFrameworks: Creative Coding Demystified*. Packt Publishing.

Perkis, T. (1999). The hub. *Electronic Musician Magazine*.

Perlis, V. (1974). *Charles Ives Remembered: An Oral History*. University of Illinois Press.

Phonotonic (2014). Interative music battle. `http://www.phonotonic.net/`. [Accessed May 28th, 2014].

Pilato, C., Collins-Sussman, B., & Fitzpatrick, B. (2008). *Version Control with Subversion*. O'Reilly Media.

Pilgrim, M. (2010). *HTML5: Up and Running*. O'Reilly Series. O'Reilly Media.

Pimenta, M., Miletto, E., Flores, L., & Hoppe, A. (2011). Cooperative mechanisms for networked music. *Future Generation Computer Systems*, *27*(1), 100 – 108.

Polak, E. (2012). The Nomadic Milk Project. `http://www.nomadicmilk.net/full/`. [Accessed June 20th, 2012].

Ponticelli, F., & McColl-Sylveste, L. (2008). *Professional haXe and Neko*. Programmer to Programmer. Wiley.

Postel, J. (1980). User datagram protocol. *USC/Information Sciences Institute*. RFC 768.

Postel, J. (1981). Transmission control protocol. *USC/Information Sciences Institute*. RFC 793.

Radioqualia (2004). Radio astronomy. `http://www.audiohyperspace.de/en/2004/09/radio-astronomy-2/`. [Accessed June 21st, 2012].

Ramakrishnan, C. (2004). The architecture of auracle: a realtime, distributed, collaborative instrument. In *Proceedings of the 2004 conference on New interfaces for musical expression*, pp. 100–103.

Reactable Systems (2014). Reactable. `http://www.reactable.com/products/mobile/`. [Accessed May 28th, 2014].

Reas, C., & Fry, B. (2007). *Processing: A Programming Handbook for Visual Designers and Artists*. Mit Press.

Rebelo, P. (2006). Haptic sensation and instrumental transgression. *Contemporary Music Review*, *25*(1-2), 27–35.

Renaud, A., & Caceres, J. P. (2010). Playing the network: The use of time delays as musical devices. In *Proceedings of the International Computer Music Conference*, pp. 244–250.

Rijnieks, K. (2013). *Cinder: Begin Creative Coding*. Packt Publishing, Limited.

Riot Games (2015). Worlds 2015 viewership. `http://www.lolesports.com/en_US/articles/worlds-2015-viewership`. [Accessed August 1st, 2016].

Roads, C. (1996). *The Computer Music Tutorial*. Mit Press.

Roberts, C. (2014). Gibber 2.0. `http://charlie-roberts.com/gibber/info/`. [Accessed May 29th, 2014].

Roberts, C., Wakefield, G., & Wright, M. (2014). The web browser as synthesizer and interface. In *Proceedings of the international conference on New Interfaces for Musical Expression*, pp. 313–318.

Rodden, T. (1992). A survey of cscw systems. *Interacting with Computers*, *3*, 319–353.

Rogers, C. (2013). Web Audio API. `http://www.w3.org/TR/webaudio/`. [Accessed May 29th, 2014].

Rohrhuber, J. (2007a). Network music. In Collins, N., & d'Escrivan, J. (Eds.), *Cambridge Companion to Electronic Music*, pp. 140–155. Cambridge University Press.

Rohrhuber, J. (2007b). Network music. In Collins, N., & d'Escrivan, J. (Eds.), *Cambridge Companion to Electronic Music*, pp. 143–144. Cambridge University Press.

Rohrhuber, J., de Campo, A., Wieser, R., van Kampen, J.-K., Ho, E., & Hölzl, H. (2007). Purloined Letters and Distributed Persons. In *Music in the Global Village Conference 2007*.

Rosenboom, D. (1976). *Biofeedback and the arts, results of early experiments*. Aesthetic Research Centre of Canada.

Ryan, J. (2010). *A History of the Internet and the Digital Future*. Reaktion Books.

Salomon, D. (2006). *Data Compression: The Complete Reference*. Molecular biology intelligence unit. Springer-Verlag London Limited.

Samp, K. (2013). Webgl Playground. `http://webglplayground.net/`. [Accessed May 29th, 2014].

Saper, C. (2001). *Networked Art*. University of Minnesota Press.

Sarkar, M. (2007). Tablanet: A real-time online musical collaboration system for indian percussion. Master's thesis, Massachusetts Institute of Technology.

Sawchuk, A. A., Chew, E., Zimmermann, R., Papadopoulos, C., & Kyriakakis, C. (2003). From remote media immersion to distributed immersive performance. In *Proceedings of the 2003 ACM SIGMM workshop on Experiential telepresence*, ETP '03, pp. 110–120 New York, NY, USA. ACM.

Schiemer, G., & Havryliv, M. (2006). Pocket gamelan: tuneable trajectories for flying sources in mandala 3 and mandala 4. In *Proceedings of the 2006 conference on New interfaces for musical expression*, NIME '06, pp. 37–42 Paris, France, France. IRCAM &#8212; Centre Pompidou.

Schooler, E. (1993). Distributed music: A foray into network music. `ftp://ftp.packetdesign.com/outgoing/casner/NetMusicFest.pdf`. [accessed Jun 4th, 2012].

Schroeder, F., Renaud, A. B., Rebelo, P., & Gualda, F. (2007). Addressing the Network: Performative Strategies for Playing Apart. In *Proceedings of the 2007 International Computer Music Conference*, pp. 133–140.

Scientific American (1891a). A long distance telephone concert. `http://archive.org/details/scientific-american-1891-02-28`. [Accessed June 15th, 2012].

Scientific American (1891b). Long distance telephone concerts. http://archive.org/details/scientific-american-1891-02-28. [Accessed June 15th, 2012].

Sexton, J. (2007). *Music, Sound and Multimedia: From the Live to the Virtual*. Music and the Moving Image Series. Edinburgh University Press.

Shreiner, D. (2009a). Pearson Education.

Shreiner, D. (2009b). *The Framebuffer*. OpenGL Series.

Simon, D. (1980). *Chambers: Scores by Alvin Lucier*. Wesleyan.

Smule (2014). Auto rap. Available at: http://www.smule.com/apps#autorap. [Accessed May 29th, 2014].

Sorensen, A. (2014). Extempore.. [Accessed May 28th, 2014].

Sorensen, A., & Gardner, H. (2010). Programming with time: Cyber-physical programming with impromptu. *ACM SIGPLAN Notices*, *45*(10), 822–834.

SoundCloud (2016). Soundcloud. https://soundcloud.com/. [Accessed August 1st, 2016].

St. Pierre, M., Stiles, J., & Bahn, C. (2006). Gps beatmap. http://faceremoval.com/face/content/video-gps-beatmap. [Accessed May 29th, 2014].

Stanford University (2012). The stanford laptop orchestra. http://slork.stanford.edu/. [Accessed June 25th, 2012].

Stelarc (2011). Internet eat. http://stelarc.org/?catID=20339. [Accessed August 1st, 2016].

Stelkens, J. (2003). peersynth: a p2p multi-user software synthesizer with new techniques for integrating latency in real time collaboration. In *in International Computer Music Conference*, pp. 319–322.

Summerfield, M. (2010). *Advanced Qt Programming: Creating Great Software with C++ and Qt 4*. Prentice Hall.

SurveyMonkey (2014). Survey monkey. https://www.surveymonkey.com. [Accessed August 29th, 2015].

Sweeney, T. (1999). Unreal networking architecture. http://udn.epicgames.com/Three/NetworkingOverview.html. [Accessed 16 May 2010].

Tanaka, A. (1999). Network Audio Performance and Installation. In *International Computer Music Conference*.

Tanaka, A. (2000). Speed of Sound. In *In: Machine Times. V2_organization*.

Tanaka, A. (2003). Seeking interaction, changing space. In *International Art and Communication Festival*.

Tanaka, A. (2006). *Interaction, Experience, and the Future of Music*, Vol. 35 of *Computer Supported Cooperative Work*, chap. 13, pp. 267–288. Springer.

Tanaka, A., & Bongers, B. (2001). Global string: A musical instrument for hybrid space. In Fleischmann, M., & Strauss, W. (Eds.), *Proceedings: Cast01 // Living in Mixed Realities*, pp. 177–181. MARS Exploratory Media Lab, FhG - Institut Medienkommunikation.

Tanzi, D. (2001). Observations about music and decentralized environments. *Leonardo*, *34*(5), 431–436.

Taylor, T. (2012). *Raising the Stakes: E-Sports and the Professionalization of Computer Gaming*. Mit Press.

the Processing.js team (2012). Processing.js. `http://processingjs.org/`. [Accessed June 20th, 2012].

The Res Rocket Surfer Project (2004). History of the rocketears. `http://www.jamwith.us/about_us/rocket_history.shtml`. [Accessed May 29th 2014].

Thorington, H. (2007). Interview: Scot gresham-lancaster. `http://turbulence.org/networked_music_review/2007/07/07/interview-scot-gresham-lancaster/`. [Accessed 1 June, 2010].

Tipler, P., & Llewellyn, R. (2007). *Modern Physics*. W.H. Freeman.

Truab, P. (2010). *Spatial Exploration: Physical, Abstracted, and Hybrid Spaces*. Ph.D. thesis, University of Virginia.

Twitter (2014). Twitter. `https://www.twitter.com`. [Accessed August 29th, 2015].

University of Colorado at Boulder (2012). The boulder laptop orchestra. `http://cismat.org/blork.html`. [Accessed June 25th, 2012].

Valve (2016). Defense of the ancients 2. `http://blog.dota2.com/`. [Accessed August 1st, 2016].

Wang, G. (2002). Chuck: Strongly-timed, concurrent, and on-the-fly audio programming language. `http://chuck.cs.princeton.edu/`. [Accessed June 20th, 2012].

Wang, G. (2007). A history of programming and music. In Collins, N., & d'Escrivan, J. (Eds.), *Cambridge Companion to Electronic Music*, pp. 69–70. Cambridge University Press.

Wang, G., & Cook, P. R. (2004). On-the-fly programming: Using code as an expressive musical instrument. In *Proceedings of the international conference on New Interfaces for Musical Expression*, pp. 138–143.

Wang, G., Essl, G., & Penttinen, H. (2008). Do Mobile Phones Dream of Electric Orchestras?. In *International Computer Music Conference*, pp. 24–29.

Wang, G., Misra, A., & Cook, P. R. (2006). Building Collaborative Graphical Interfaces in the Audicle. In *NIME '06: Proceedings of the 2006 conference on New interfaces for musical expression*, pp. 49–52 Paris, France, France. IRCAM — Centre Pompidou.

Wang, J., Yang, J.-Y., Fazal, I. M., Ahmed, N., Yan, Y., Huang, H., Ren, Y., Yue, Y., Dolinar, S., Tur, M., & Willner, A. E. (2012). Terabit free-space data transmission employing orbital angular momentum multiplexing. *Nature Photonics*, *advance online publication*, –.

Ward, A., Rohrhuber, J., Olofsson, F., McLean, A., Griffiths, D., Collins, N., & Alexander, A. (2004). Live Algorithm Programming and a Temporary Organisation for its Promotion. In Goriunova, O., & Shulgin, A. (Eds.), *read_me — Software Art and Cultures*.

Watson, K. (2016). International music summit business report..

Weinberg, G. (2003). *Interconnected Musical Networks: Bringing Expression and Thoughtfulness to Collaborative Group Playing*. Ph.D. thesis, Massachusetts Institute of Technology.

Weinberg, G. (2005). Interconnected Musical Networks: Toward a Theoretical Framework. *Computer Music Journal, 29(2): 23–29*.

Weiss, A. (2008). *Varieties of Audio Mimesis: Musical Evocations of Landscape*. Audio issues. Errant Bodies Press.

Wessel, D., & Wright, M. (2002). Problems and prospects for intimate musical control of computers. *Computer Music Journal*, *26*(3), 11–22.

Williamon, A. (2004). *Musical Excellence: Strategies and Techniques to Enhance Performance*. Oxford University Press, USA.

Wilson, C. (2013). A tale of two clocks. `http://www.html5rocks.com/en/tutorials/audio/scheduling/`. [Accessed May 29th, 2014].

Wilson-Bokowiec, J., & Bokowiec, M. A. (2006). Kinaesonics: The intertwining relationship of body and sound. *Contemporary Music Review*, *25*(1-2), 46–57.

Wöhrmann, R., & Ballet, G. (1999). Design and architecture of distributed sound processing and database systems for web-based computer music applications. *Comput. Music J.*, *23*(3), 73–84.

Wolfe, A., & Wolfe, C. (2014). Jasuto. `http://www.jasuto.com/`. [Accessed May 29th, 2014].

Woszczyk, W., Cooperstock, J. R., Roston, J., & Martens, W. (2005). Shake, rattle and roll: Getting immersed in multisensory, interactive music via broadband networks. *Journal of the Audio Engineering Society*, *53*(4), 336–344.

Wright, M. (2002). Open sound control specification. `http://opensoundcontrol.org/spec-1_1`. [Accessed May 29th, 2014].

Xu, A., & Cooperstock, J. (2000). Real time streaming of multi-channel audio data over internet 5120 (i - 3)..

Xu, G. (2003). *Gps: Theory, Algorithms and Applications*. Springer.

Yamagishi, S. (1998). Variations for WWW: Network Music by MAX and the WWW. In *International Computer Music Conference*.

Yonamine, N. (2013). Coffeecollider. `http://mohayonao.github.io/CoffeeCollider/`. [Accessed May 29th, 2014].

Z.B. Bishop and H.P. Lovecraft (1953). *The Curse of Yig*. Arkham House.

Zimmermann, R., Chew, E., Ay, S. A., & Pawar, M. (2008). Distributed musical performances: Architecture and stream management. *ACM Trans. Multimedia Comput. Commun. Appl.*, *4*(2), 14:1–14:23.

Appendices

Appendix A

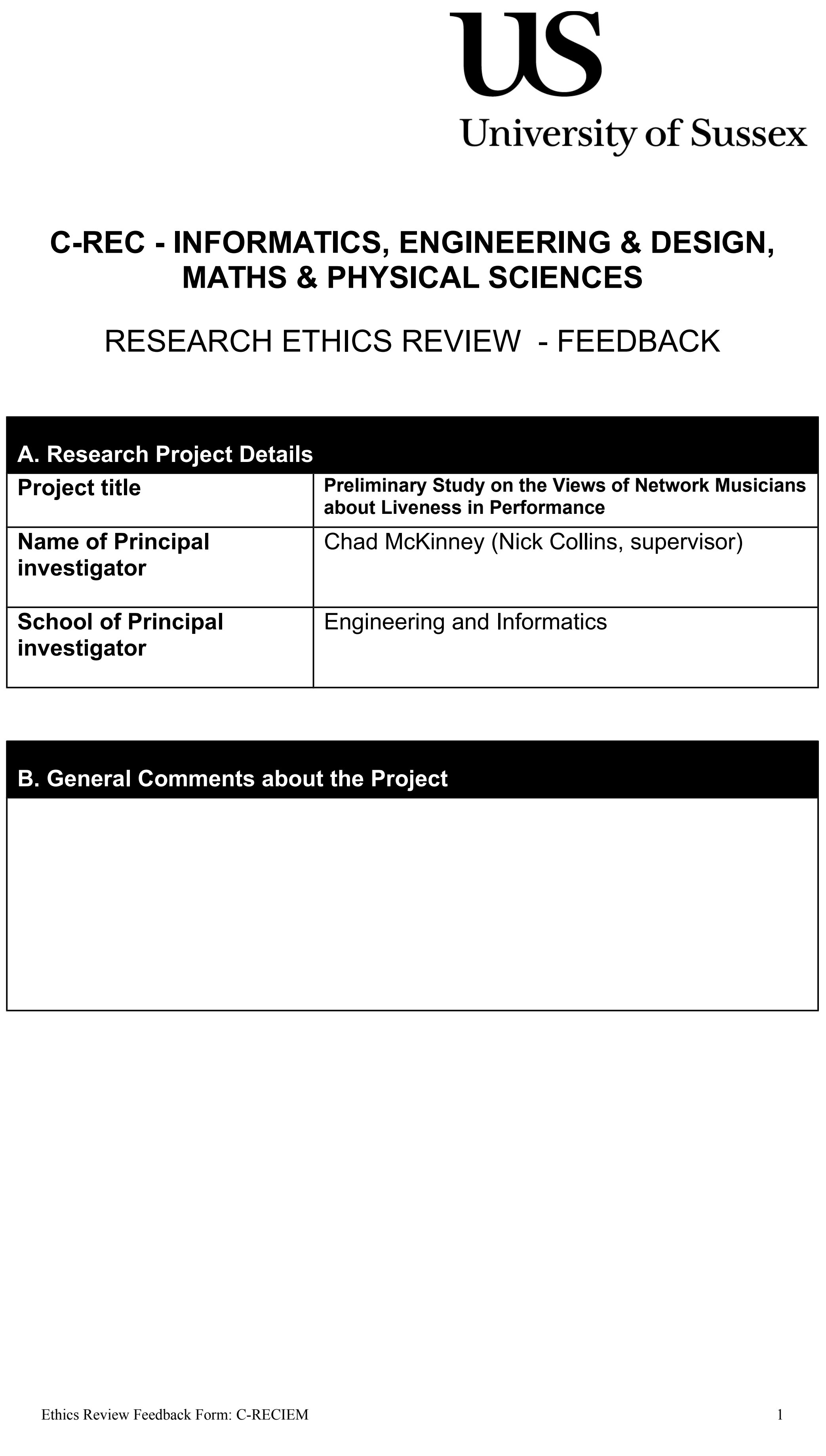
Liveness In Network Music Performance

A.1 Questionnaire on the Views of Network Musicians About Liveness in Performance

1. Do you wish your responses to this questionnaire to be fully anonymous (the default) or to be attributed to you personally if used in direct quotation?
2. How many members does your ensemble have?
3. What hardware (laptops, phones, kinect, instruments, etc..) does your ensemble use?
4. What are the kinds of software, languages, and environments does your ensemble use? Does everyone use the same collection or is there a mix?
5. Does your ensemble perform with members physical distributed among several locations?
6. What types of connections does your ensemble typically perform with, ie. Ethernet, wireless, etc..
7. Do you use any kind of visual element during performances? If so please describe the presentation?
8. Does your ensemble live code during performance? If so, do you show your screens?
9. How would you classify the genre or style of music that your ensemble performs?
10. How does your ensemble communicate with each other during performances?
11. Broadly, how do you feel network performance, and in particular your ensemble's approach to network music effects a sense of liveness as a performer?
12. How does your ensemble's structure and approach influence your sense of involvement in performance?
13. If your ensemble performs physically distributed, do you feel this effects a sense of liveness or connectivity?
14. How well do you think your ensemble projects involvement and effort by its members to a given audience?
15. How do your ensemble's channels of communication impact on group awareness? Do you find this to be successful and how do you compare it to more traditional ensembles using acoustic instruments?
16. Given the network music context, in using any controller interface for your music, how does the hardware effect the connection between effort and sonic output?
17. How do you feel your ensemble's visual presentation is effected by your networking setup? Does this effect your feeling of connection to the other performers during a performance?
18. As a performer (of any kind of music) do you have any regular psychological responses to performing (anxiety, excitement, etc...) and how does performing network music effect this response?

19. If you perform electronic music as a solo performer as well, could you please describe how your solo performance and networked performance work differs with respect to liveness?
20. What do you think has worked well for your ensemble, and what do you think has not, in regards to fostering a general sense of liveness during performance?
21. As an audience member for other network performances, do you feel observing a networked performance differs from actively engaging in it? If so, why?
22. As an audience member, do you feel that multi-location ensembles are affected by their physical distribution with regards to active engagement by all performers?
23. If you have any additional comments you would like to add, please note them here.

A.2 Ethics Approval



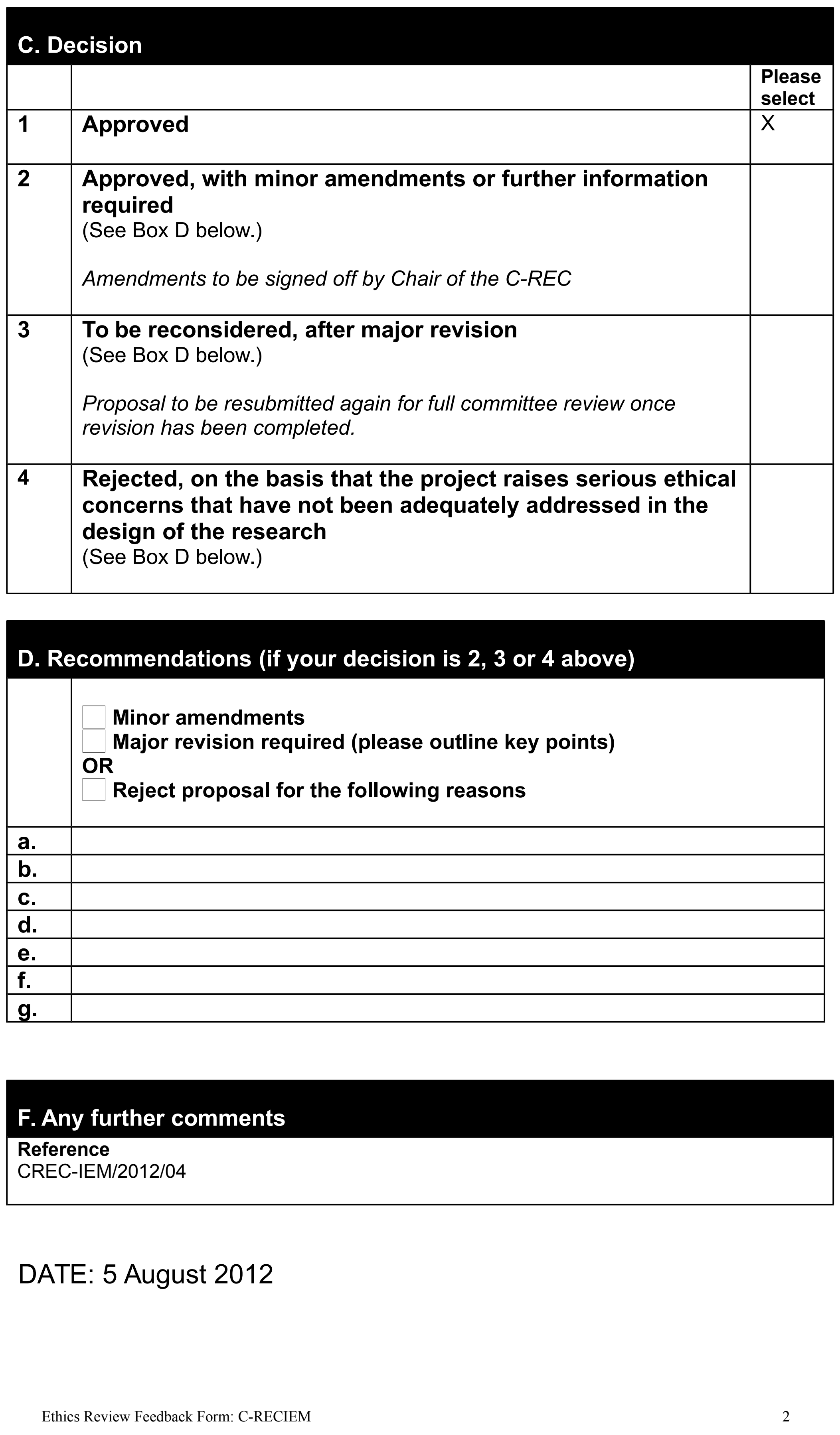
University of Sussex

C-REC - INFORMATICS, ENGINEERING & DESIGN, MATHS & PHYSICAL SCIENCES

RESEARCH ETHICS REVIEW - FEEDBACK

A. Research Project Details	
Project title	**Preliminary Study on the Views of Network Musicians about Liveness in Performance**
Name of Principal investigator	Chad McKinney (Nick Collins, supervisor)
School of Principal investigator	Engineering and Informatics

B. General Comments about the Project



C. Decision

		Please select
1	**Approved**	X
2	**Approved, with minor amendments or further information required** (See Box D below.) *Amendments to be signed off by Chair of the C-REC*	
3	**To be reconsidered, after major revision** (See Box D below.) *Proposal to be resubmitted again for full committee review once revision has been completed.*	
4	**Rejected, on the basis that the project raises serious ethical concerns that have not been adequately addressed in the design of the research** (See Box D below.)	

D. Recommendations (if your decision is 2, 3 or 4 above)

	☐ **Minor amendments** ☐ **Major revision required (please outline key points)** **OR** ☐ **Reject proposal for the following reasons**
a.	
b.	
c.	
d.	
e.	
f.	
g.	

F. Any further comments

Reference
CREC-IEM/2012/04

DATE: 5 August 2012

Ethics Review Feedback Form: C-RECIEM 2

Appendix B

Yig, The Father of Serpents

B.1 Yig Recording Comparison

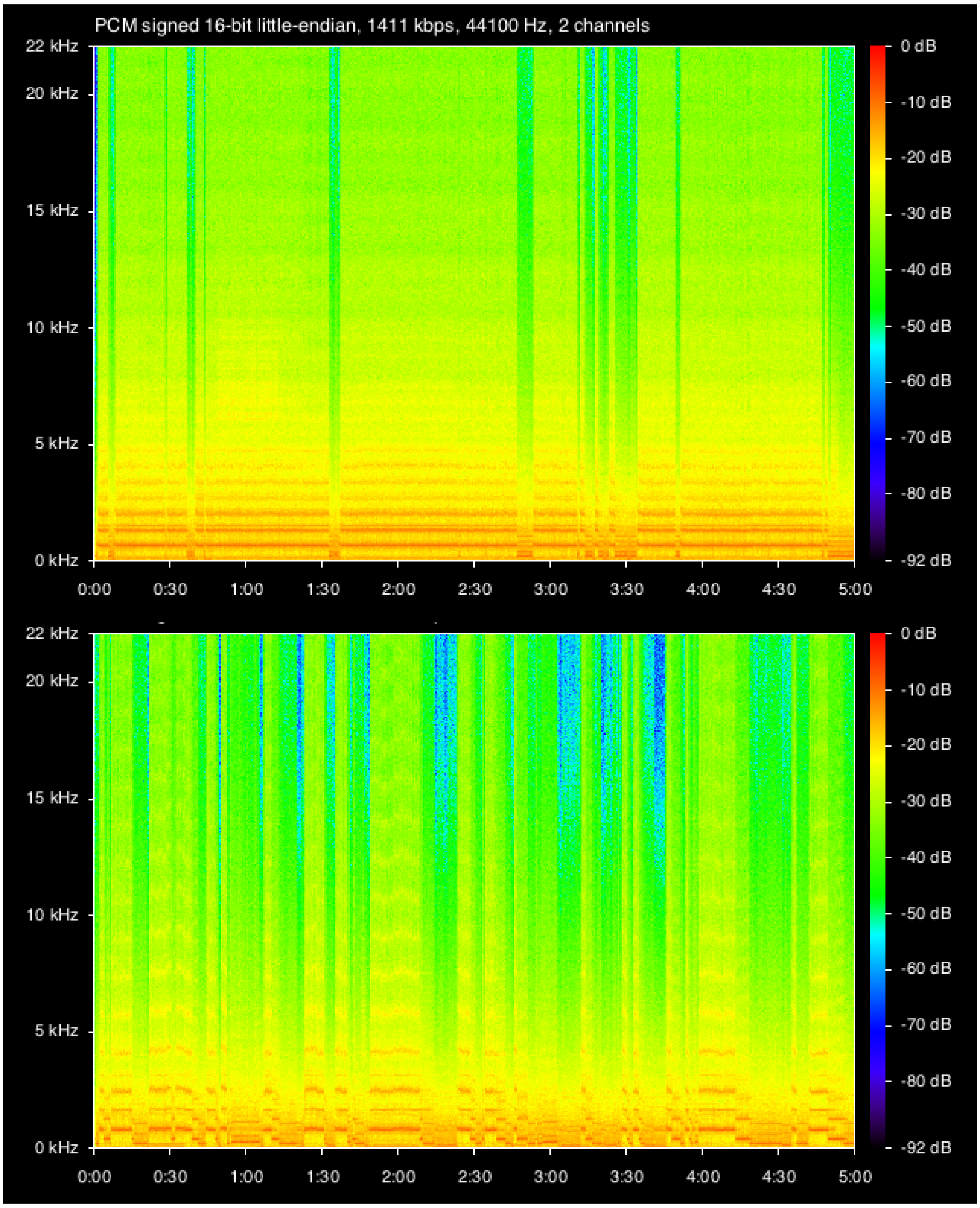
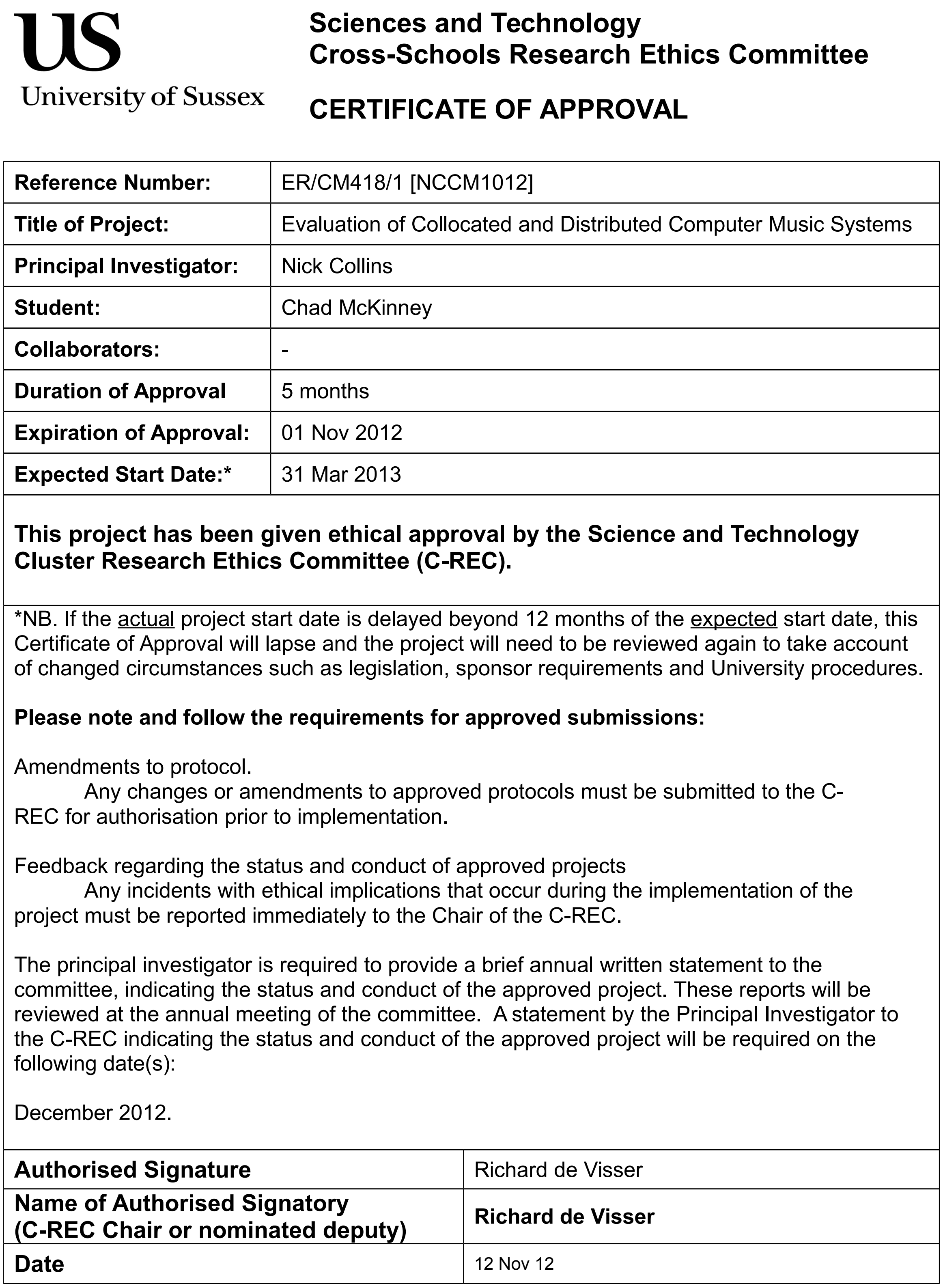


Figure B.1: Comparison of recordings from two nodes with identical states.

B.2 Ethics Approval



University of Sussex

Sciences and Technology
Cross-Schools Research Ethics Committee

CERTIFICATE OF APPROVAL

Reference Number:	ER/CM418/1 [NCCM1012]
Title of Project:	Evaluation of Collocated and Distributed Computer Music Systems
Principal Investigator:	Nick Collins
Student:	Chad McKinney
Collaborators:	-
Duration of Approval	5 months
Expiration of Approval:	01 Nov 2012
Expected Start Date:*	31 Mar 2013

This project has been given ethical approval by the Science and Technology Cluster Research Ethics Committee (C-REC).

*NB. If the actual project start date is delayed beyond 12 months of the expected start date, this Certificate of Approval will lapse and the project will need to be reviewed again to take account of changed circumstances such as legislation, sponsor requirements and University procedures.

Please note and follow the requirements for approved submissions:

Amendments to protocol.
Any changes or amendments to approved protocols must be submitted to the C-REC for authorisation prior to implementation.

Feedback regarding the status and conduct of approved projects
Any incidents with ethical implications that occur during the implementation of the project must be reported immediately to the Chair of the C-REC.

The principal investigator is required to provide a brief annual written statement to the committee, indicating the status and conduct of the approved project. These reports will be reviewed at the annual meeting of the committee. A statement by the Principal Investigator to the C-REC indicating the status and conduct of the approved project will be required on the following date(s):

December 2012.

Authorised Signature	Richard de Visser
Name of Authorised Signatory (C-REC Chair or nominated deputy)	**Richard de Visser**
Date	12 Nov 12

B.3 User Evaluation Questions

B.3.1 Background

- Nationality
- Gender
- Age
- Please list the instrument(s) you play, with the number of years you have been studying them: How would you describe your experience with computer music, if any (how many years have you been using computers in music, and what software are you familiar with)?
- How would you describe your experience with network music, if any?

B.3.2 Evaluation Questionnaire Session 1

- Which system did you use: Yig or Auracle?
- Was your experimental partner in the same room as you?
- With regards to the system you evaluated rate each item 1 – 7 by circling the selected value for each line. 1 being strongly disagree and 7 being strongly agree.
  - It is useful for music collaboration.
  - I don't notice any inconsistencies as I use it.
  - It does everything I expect it to.
  - It is user friendly.
  - It is flexible.
  - Using it is effortless.
  - I learned to use it quickly.
  - I quickly became skillful with it.
  - I am satisfied with it.
  - It is fun to use.
- With regards to the musical collaboration rate each item 1 – 7 by circling the selected value for each line. 1 being strongly disagree and 7 being strongly agree.
  - I felt involved with the collaboration.
  - I enjoyed the collaboration.
  - I understood what was going on.
  - I felt satisfied with the result.
  - I felt out of control.
  - The other participant ignored my contributions.
  - I felt aware of the other participant.
  - Coordination was difficult.

  - I would have played longer if given the option.
  - If I were performing for an audience just now, I would be satisfied with the performance.
  - The collaboration was gratifying.

B.3.3 Evaluation Questionnaire Session 2

- Which system did you use: Yig or Auracle?
- Was your experimental partner in the same room as you?
- With regards to the system you evaluated rate each item 1 – 7 by circling the selected value for each line. 1 being strongly disagree and 7 being strongly agree.
  - It is useful for music collaboration.
  - I don't notice any inconsistencies as I use it.
  - It does everything I expect it to.
  - It is user friendly.
  - It is flexible.
  - Using it is effortless.
  - I learned to use it quickly.
  - I quickly became skillful with it.
  - I am satisfied with it.
  - It is fun to use.
- With regards to the musical collaboration rate each item 1 – 7 by circling the selected value for each line. 1 being strongly disagree and 7 being strongly agree.
  - I felt involved with the collaboration.
  - I enjoyed the collaboration.
  - I understood what was going on.
  - I felt satisfied with the result.
  - I felt out of control.
  - The other participant ignored my contributions.
  - I felt aware of the other participant.
  - Coordination was difficult.
  - I would have played longer if given the option.
  - If I were performing for an audience just now, I would be satisfied with the performance.
  - The collaboration was gratifying.

B.3.4 Interview Questions

- Briefly, could you describe your general opinion of the collaboration?
- How do you think the collaboration would have differed if you were in separate locations or the same location?
- How did the software effect your ability to develop musical ideas?
- How would you compare the anxiety of a normal performance with the anxiety from the collaboration that just happened?
- Would you be interested in this kind of collaboration in the future? Why or why not?
- Can you describe the most negative aspects of the collaboration?
- Can you describe the most positive aspects of the collaboration?
- Are there any further comments you wish to make?

B.4 Yig And Auracle Evaluation Thematic Analysis Codes

B.4.1 Auracle – Distributed Codes

- Collaboration difficult
- Source of sounds confusing
- Simple
- Limited
- Small Button Better
- Still don't know the rules
- Weird that sound muted while recording voice
- Struggled for control
- Not really nervous
- Fun and not nervous
- Maybe nervous
- Maybe weird with an audience
- Further Interest in Network Music
- Initially boring
- Struggled to synchronize
- Unique experience
- Wanted more causality to actions
- Definitely enjoyable
- Hard to listen while getting used to interaction
- Had to get accustomed to relationships with the sound and gesture
- A little challenging, but probably good
- First five minutes just messing around
- Maybe more enjoyable with experience
- Glued to screen
- Same room would've made much difference
- Worth trying in the same room
- Can communicate visually quicker when in the same room
- Relieved to be in a different room
- Good to be distributed while getting used to making sound

- If facing each other you can laugh at each other
- Anxiety was different from a performance, learning vs attaining
- Didn't feel anxious
- Liked the map
- Just learning
- Interested in combining with live instrumentation
- Definitely interested in future collaboration
- Taking part demystifies
- Communication was the most difficult
- Just trying to listen and copy
- Delay made communication difficult
- Simpler and immediacy quite good
- Auracle didn't make any sense
- Didn't realize we were collaborating
- Could see visuals but couldn't understand
- The input never matched anything I'd expect from the output
- High pitch sounds actually make some kind of noise
- hard to collaborate when have no idea what's going on
- Easier to hear each other in the same location
- Hated using voice
- Using our voices together would've sucked but less than being lost when apart
- Awkward making noise alone, like a crazy person
- Graphic visualization is far too abstract
- Fun, interested in further collaboration
- Making network music could be more fun than chatting with friends
- Wanted more channels of communication in Auracle
- Could give more options for atmosphere
- Actions were less meaningful
- Would not work at all if you were in the same location
- Lag might make local collaboration difficult
- Can tell it's responding but little connection
- Laggy and unpredictable
- It's disappointing because it doesn't correlate
- Bothered me that it broke up long notes
- Would be better for performance to be familiar with software
- Vaguely nervous because it involved vocal performing
- Feels more like a toy than an instrument
- Didn't know what to do with voice
- Would make a poor performance medium
- Nervous during networked performances
- Not interested in using Auracle in the future
- Too slow and predictable
- Not too excited to use it again
- Idea isn't bad, just bad execution
- Worst thing was lack of real-time interaction between performers
- Didn't feel like there was much input from the performers
- Some nice moments but further apart than in Yig

B.4.2 Auracle – Co-located Codes

- Didn't know what I was doing
- Would be the same if distributed
- Didn't sing
- Made strange noises
- Didn't know what I was doing or to expect
- Didn't know what to expect
- Adaptive, but didn't know how to use
- Clearer than Yig
- Not sure what will come out
- A performance could get nerve wracking
- Lack of expectation in performance lessens nervousness
- Fun to use
- Didn't understand
- Network music is an interesting ideas
- Unsure if I want to do it more
- Most negative is not understanding
- Fun but surprising
- Fun and funny to see what collaborator would do
- Interesting to explore together without talking
- Would've been different if we knew more
- I had no idea what was going on
- Even not knowing was fun
- He knew what was going on more than me
- Maybe chat used more if in different locations
- Don't think would've used chat even if in different locations
- My network band doesn't use ours
- With an audience it would be different if we were in different locations
- We could hear our voices but not if separated
- The only enjoyment was from each other's voices, not from patch
- Doesn't get to developing musical ideas
- Doesn't really have an interface
- No anxiety whatsoever
- Wouldn't like singing on stage at all
- I get terrible stage fright
- I use my voice plenty, wouldn't like to use Auracle
- A vocal improviser would just ignore the system
- A script might help it be more predictable
- I've seen better vocal based systems
- I do network music already, but slightly different
- Would like to open up my interfaces to networking
- Would have to let go of pristine control when networking
- Worst moment when he changed the chat room
- Worst moment when he started singing "Will you be my bodyguard?"
- Yig more fun and had more communication
- I had no idea what was going on
- I'm really uncomfortable with singing
- That was awkward until I figured out I could make it feed back

- Probably would've used my voice more if distributed because he was laughing at me
- Being in the same location was prohibitive
- I think it is a composition, it does one thing
- The only thing that made me anxious was having to sing
- The only thing that made me anxious was not knowing the correspondence of action and sound
- It was fun, felt like playing
- Collaborator was awkward
- Collaborator knew too much
- Sometimes something spontaneously fun would happen
- I'd probably never say collaborating
- I thought we weren't collaborating at all
- Struggled to get past technical aspects
- Recorded each others output
- Found things together
- Hadn't worked it out
- Quite cool, the repeats and loops you got
- Probably would've used the chat more if distributed
- Probably would've ignored each other more
- When distributed would've collaborated more because of less distraction
- If distributed, would've been more isolated
- Co-located reminded me to be collaborative
- Quite well labeled but still couldn't make connections
- Would like to better understand
- Could make it on the computer better
- Easier to work out what you were doing
- A mute button would help
- Good thing about co-located is we can talk
- Some nervousness because the interviewer was there
- Crossed my min that "This is crap"
- Not the other user, but the interviewer made me nervous
- Just some network music in a class
- Really cool to play over the internet
- Interested in seeing someone else do it
- The technology was barbaric
- I couldn't understand what was going on
- Could just do things independently
- Would've been good to play by yourself first
- Learning was a bigger barrier to experiment
- On my own I would've gotten bored
- Improvising more fun with someone inspiring
- Visual contact is important for communication in improvising
- Expertise is important for improvisation
- Not knowing software is like using a different instrument
- Couldn't figure out visual feedback
- Learning software like learning an instrument

B.4.3 Yig – Distributed Codes

- Still don't know the rules
- Accidently discovered connections
- Wanted more informative interface
- Wanted better synth names
- Wanted to be able to mute
- Deleting seemed rude
- Halfway through began to understand
- Custom names only helpful with more time
- not really nervous
- Fun and not nervous
- Maybe nervous
- Maybe weird with an audience
- Further interest in network music
- Initially boring
- Unique experience
- Very nice results
- Wanted more causality to actions
- Limited, which is bad and good
- Easier to understand
- Definitely enjoyable
- Hard to listen while getting used to interaction
- Had to get accustomed to relationships with the sound and gesture
- A little challenging, but probably good
- First five minutes just messing around
- Maybe more enjoyable with experience
- Glued to screen
- Same room wouldn't have made much difference
- Worth trying in the same room
- Can communicate visually quicker when in the same room
- Relieved to be in a different room
- Good to be distributed while getting used to making sound
- If facing each other you can laugh at each other
- You could just try different things
- Could have been crazier but didn't know how to use it
- I hoped that turning an arrow would be more dramatic
- Seeing what the other person was doing worked really well
- Anxiety was different from a performance, learning vs attaining
- Didn't feel anxious
- Just learning
- Interested in combining with live instrumentation
- Definitely interested in future collaboration
- Taking part demystifies
- Communication was the most difficult
- Just trying to listen and copy
- Most positive when discovered how to connect synths
- Easier to get started
- Build off ideas really really really easy
- Don't need instrumental knowledge
- Figure out which combinations make different sounds
- Visuals were super easy to understand
- Can see the network working, can see the vibrations

- Felt better than previous musical performances
- More experimental, actually enjoyable
- I used to play hard core and hated being expected to emote
- Here you can sit down and focus
- Connecting causes strange surprises
- There was a learning curve
- Everything was easy to figure out except for deleting
- Fun, interested in further collaboration
- Making network music could be more fun than chatting with friends
- Most negative when all sounds were deleted
- Most negative when I couldn't get rid of a rhythmic sound
- With more time would've used chat more
- Didn't need more channels of communication
- Too busy to type "Hey, let's do this"
- Could give more options for atmosphere
- I wanted to make it higher, a happy place
- Really, really enjoyable
- We made something that was cool and you could enjoy it
- Felt happy with it but recreational too
- We're both running places and working to make a sound
- It was fun, musically, mechanically
- You could actually see what was going on
- No difference, except maybe text chat instead of talking
- The difficulty is familiarity with the synths
- Some synths didn't make sound, unsure why
- Would be better for performance to be familiar with software
- A visual medium may be helpful in improvised settings
- It was pretty easy to pick up
- Nervous during networked performances
- Interested in further collaboration using it
- Was fun
- Interested to be more familiar with it
- Worst thing was lack of real-time interaction between performers
- Yig crashed in the middle
- Liked the discovery of sound based on name
- Sometimes got lost
- Couldn't delete synths
- Having the score synchronized would be easier

B.4.4 Yig – Co-located Codes

- Good Fun
- Intuitive Collaboration
- Want to try with more people
- Didn't know what I was doing
- Would be the same if distributed
- Didn't know what to expect

- Adaptive, but didn't know how to use
- Unclear connection to actions
- Dragging loads of stuff
- Easier than Auracle
- Easier to manipulate
- More musical than Auracle
- A performance could get nerve wracking
- Noisiness makes ideas come easy
- Lack of expectation in performance lessens nervousness
- Fun to use
- Didn't understand
- Network music is an interesting ideas
- Unsure if I want to do it more
- Most negative is not understanding
- Fun but surprising
- Complicated without experience
- Didn't realize why I should connect multiple synths
- Fun and funny to see what collaborator would do
- Interesting to explore together without talking
- Like the output of sounds
- Would've been different if we knew more
- Second time using Yig, informed some decisions
- I had no idea what was going on
- Even not knowing was fun
- He knew what was going on more than me
- Just luck
- Different locations wouldn't have been different
- Maybe chat used more if in different locations
- Don't think would've used chat even if in different locations
- My network band doesn't use our chat
- With an audience it would be different if we were in different locations
- We could hear our voices but not if separated
- Feel like a rate in a maze
- More of a piece than an instrument
- Needs more transparent names for general use as a tool
- The names are beautifully poetic and amusing
- Prevented from being a tool
- Synths would jump to the side
- Some synths wouldn't select
- Crashed, if in front of an audience, a show stopper
- No anxiety whatsoever
- I do network music already, but slightly different
- Copies ReacTable and it's cool
- Would like to open up my interfaces to networking
- Would have to let go of pristine control when networking
- Worst moment when he changed the chat room
- Worst moment when he started singing "Will you be my bodyguard?"
- Yig more fun and had more communication
- We weren't fighting over where things should go
- I had no idea what was going on

- I knew exactly what to do, so I tried stuff I'd never done before
- I don't think being in different places would have made any difference
- I do think (co-location) makes a difference, but not much
- It gave me more ideas, opened more possibilities
- Didn't know whether it was supposed to not be responding
- No idea what to expect, but I guess that would be the software
- The only thing that made me anxious was not knowing the correspondence of action and sound
- It was fun, felt like playing
- Collaborator was awkward
- Collaborator knew too much
- There shouldn't be synths that don't make sound
- Not knowing if it was the software or me not doing something right was puzzling
- Sometimes something spontaneously fun would happen
- Things don't necessarily do what you want, that's the best part
- I'd probably never say collaborating
- I thought we weren't collaborating at all
- It was more fun trying to stop you from doing anything
- Struggled to get past technical aspects
- Accidental discovery during mutual movement
- A solo synth function would've been useful
- Keeping it simple would help
- A mute button would help
- Good thing about co-located is we can talk
- Some nervousness because the interviewer was there
- Crossed my mind that "This is crap"
- Not the other user, but the interviewer made me nervous
- Just some network music in a class
- Really cool to play over the internet
- Interested in seeing someone else do it
- The technology was barbaric
- I couldn't understand what was going on
- Could just do things independently
- Would've been good to play by yourself first
- Learning was a bigger barrier to experiment
- On my own I would've gotten bored
- Improvising more fun with someone inspiring
- Visual contact is important for communication in improvising
- Expertise is important for improvisation
- Not knowing software is like using a different instrument
- Found synth connections by accidental colliding
- Screens not a barrier
- Nice visual feed back
- Learning software like learning an instrument
- Exploratory aspect quite fun

B.5 Yig And Auracle Evaluation Likert Items

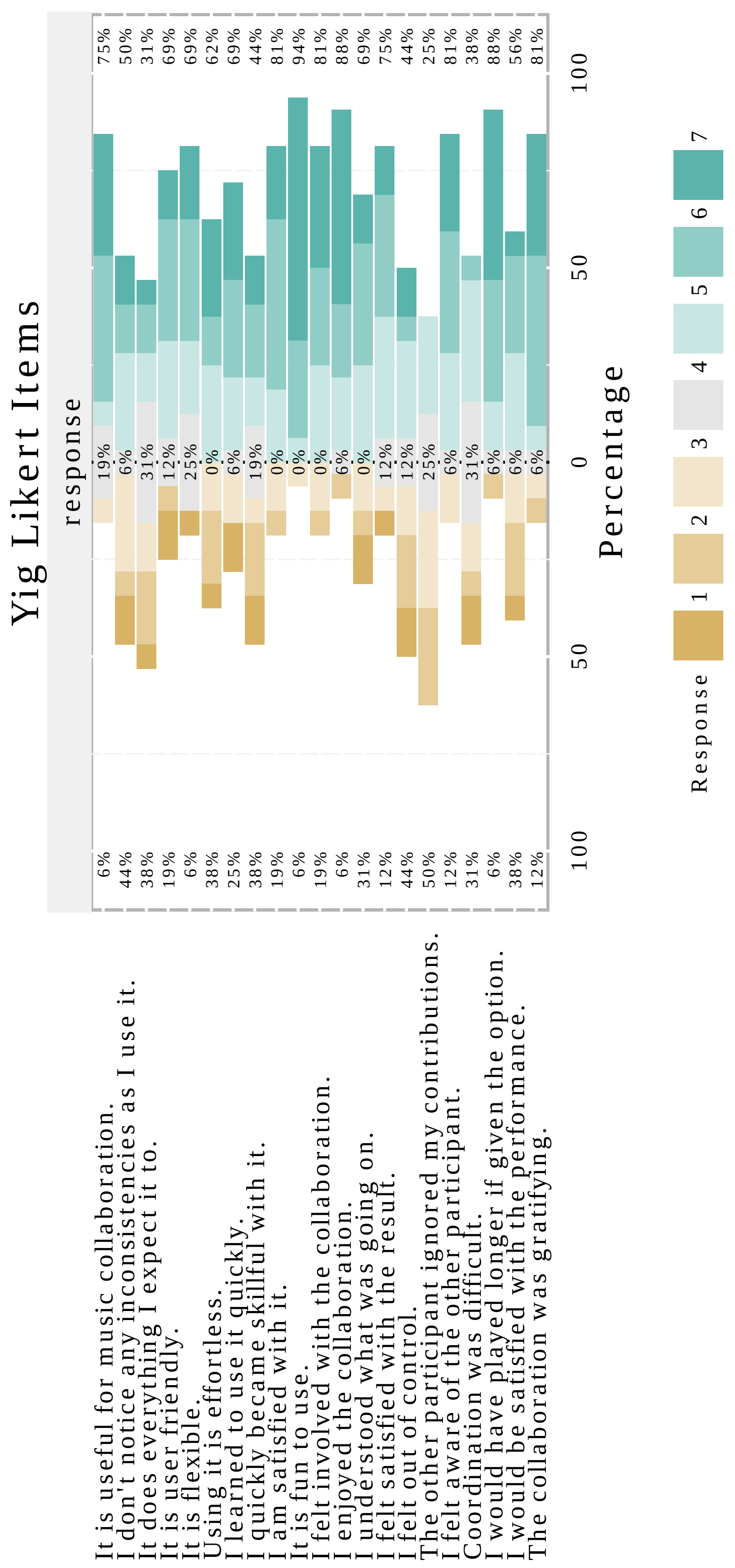


Figure B.2: Yig Likert Items.

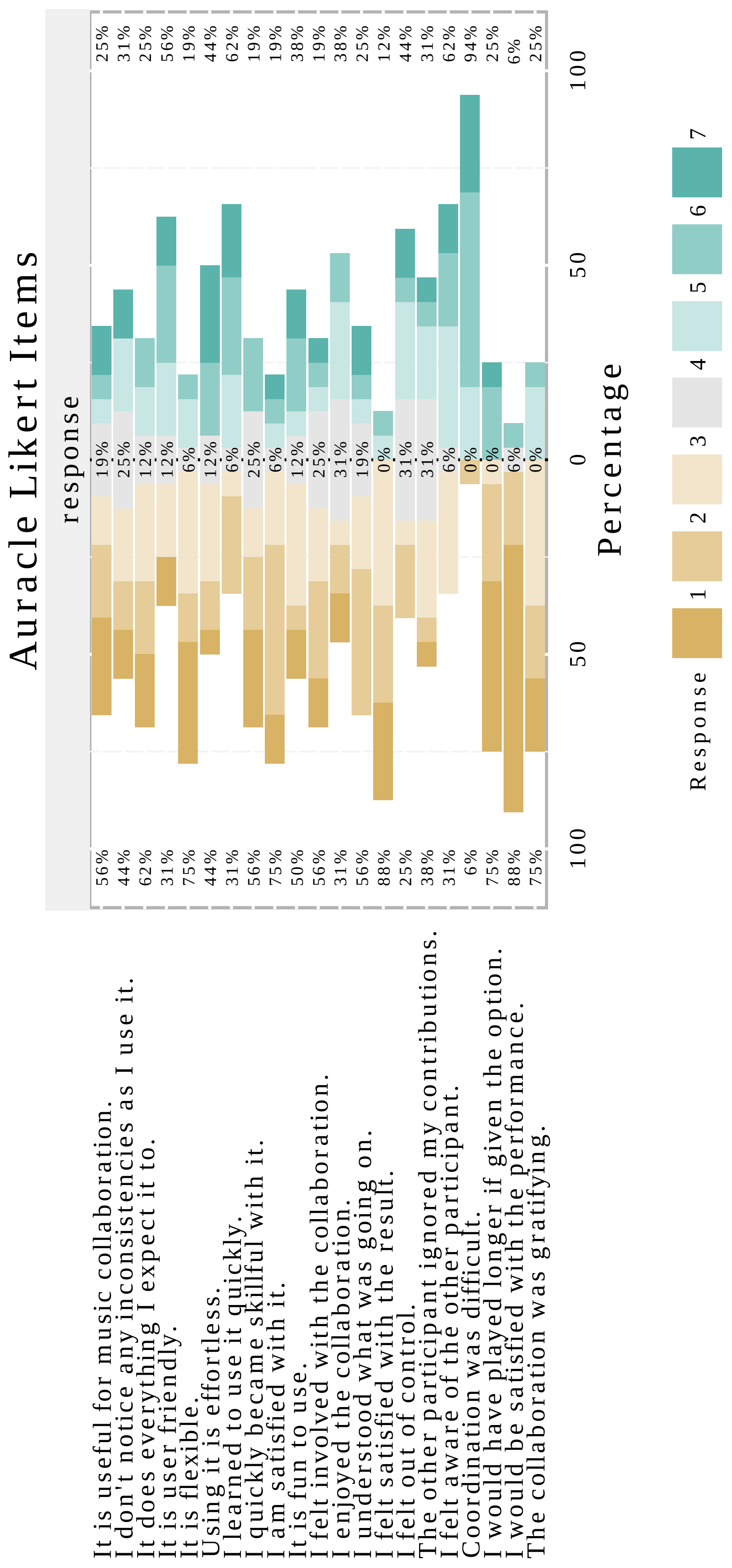


Figure B.3: Auracle Likert Items.

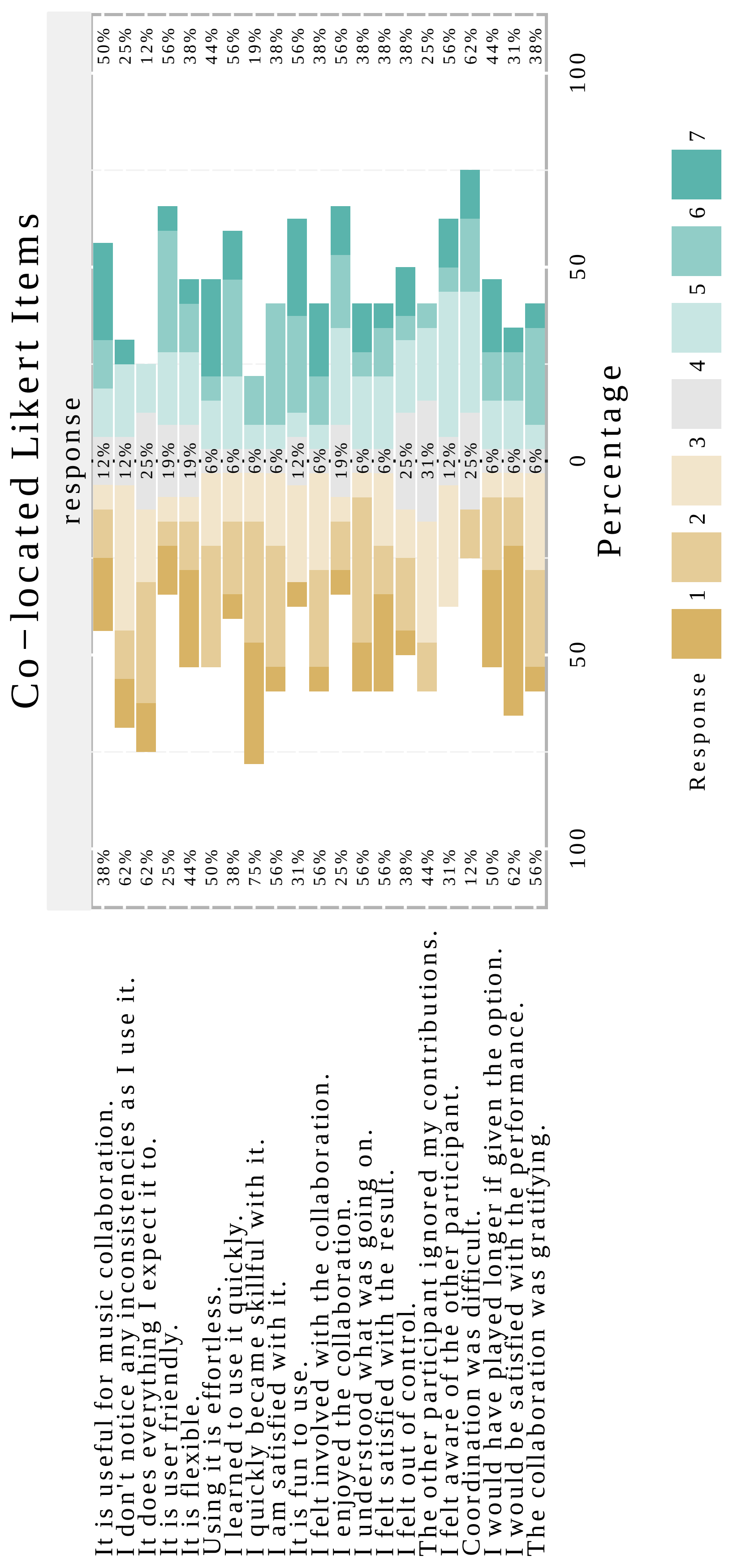


Figure B.4: Co-Located Likert Items.

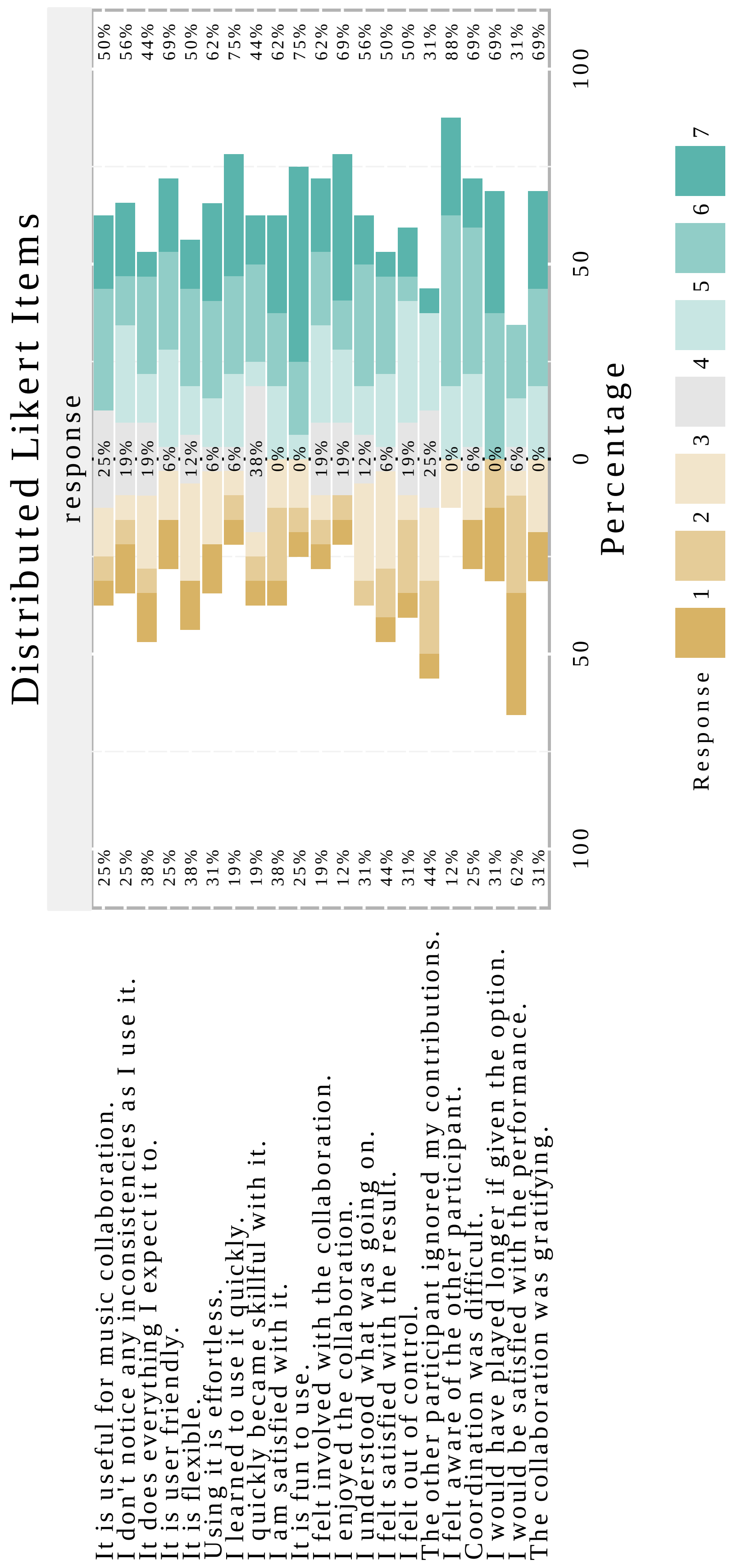


Figure B.5: Distributed Likert Items.

```
SynthDef.new("StutterSplice",{
   | amp=0.1,feedAmp=1,paramOne=0.5,paramTwo=0.5,
  audioIn=24,modIn=24,audioOut=20,gate=1 |
  var signal, env, directOut, feedOut,freq,freqMod, audioInput, modInput;
  var onsetFFT, trigEnv, trig, fft, fft2, fft3, delay, sil;
  env = EnvGen.ar(Env.asr(Rand(0.1,5),1,0.1,-4),gate : gate,doneAction : 2);
  audioInput = InFeedback.ar(audioIn) * env;
  modInput = InFeedback.ar(modIn) * env;
  paramOne = Lag.kr(paramOne);
  paramTwo = Lag.kr(paramTwo);
  trig = HPZ2.kr(paramOne + paramTwo);
  onsetFFT = FFT(LocalBuf(1024), audioInput);
  trig = Onsets.kr(onsetFFT, 0.1);
  trigEnv = EnvGen.ar(Env.asr(0.001,1,0.5,-4),gate : trig,doneAction : 0);
  signal = trigEnv * (audioInput + modInput) * 3;
  delay = BufDelayC.ar(
    LocalBuf(SampleRate.ir * 3, 3),
    signal,
    [
      Lag.kr(6.5 / paramOne.linlin(0, 1, 1, 16).round(1), 1),
      Lag.kr(6.5 / paramTwo.linlin(0, 1, 1, 16).round(1), 1),
      6.5 / 16
    ].reciprocal);
  fft = FFT(LocalBuf(2048), delay [ 0 ] );
  fft = PV_MagFreeze(
    fft,
    Decay.kr(DelayC.kr(CoinGate.kr(0.1,trig),paramOne/5)));
  fft = PV_BrickWall(fft, LFNoise1.ar(1, 0.5) + (paramOne - 0.5));
  fft = IFFT(fft);
  fft2 = FFT(LocalBuf(2048), delay [ 1 ] );
  fft2 = PV_MagFreeze(
    fft2,
    Decay.kr(DelayC.kr(CoinGate.kr(0.1,trig),paramTwo/5),0.2));
  fft2 = PV_BrickWall(fft2, LFNoise1.ar(1, 0.5) + (paramTwo - 0.5));
  fft2 = IFFT(fft2);
  fft3 = FFT(LocalBuf(2048), delay [ 2 ] );
  fft3 = PV_MagFreeze(
    fft3,
    Decay.kr(
      DelayC.kr(CoinGate.kr(0.1,trig),LFNoise1.kr(3.25,0.5,0.5)),
      1));
  fft3 = PV_BrickWall(fft3, 0.5);
  fft3 = IFFT(fft3);
  signal = LeakDC.ar(fft+fft2+(fft3 * 2) * 4);
  signal = Limiter.ar(signal);
  sil = Silence.ar;
  signal = Select.ar(CheckBadValues.ar(signal), [ signal,sil,sil,sil ] );
  Out.ar(0, [ signal, signal ] *amp*env);
  Out.ar(audioOut,signal*feedAmp*env);
});
```

Figure B.6: One of the synths used in Yig performances

Appendix C

An Interactive 3D Networked Music Space

C.1 User Evaluation Questions

I Rate the following statements with a number 1 through 7 according to how strongly you agree with them. 1 being strongly disagree and 7 being strongly agree.

- Shoggoth is useful for music collaboration
- I quickly became skillful with Shoggoth
- It was difficult to communicate and collaborate
- The 3D graphics interface implementation was useful
- I will use Shoggoth in a future performance

II How did the 3D graphics interface effect your ability to develop musical ideas?

III How did the 3D graphics interface effect your ability to communicate and collaborate?

IV Can you describe the most negative aspects of your experience with Shoggoth?

V Can you describe the most positive aspects of your experience with Shoggoth?

VI Are there any further comments you wish to make?

Appendix D

Quick Live Coding Collaboration in the Browser

D.1 User Evaluation Numbered Responses

Users were asked to rate the following statements with a number 1 through 7 according to how strongly they agree with them. 1 being strongly disagree and 7 being strongly agree.

I **Lich.js is useful for music collaboration**

(a) 5
(b) 5
(c) 7
(d) 1
(e) 7
(f) 4
(g) 4
(h) 6
(i) 1
(j) 7
(k) 6
(l) 6
(m) 5
(n) 3
(o) 5
(p) 5
(q) 7

II **I quickly became skillful with Lich.js**

(a) 4
(b) 3
(c) 3
(d) 3
(e) 6
(f) 5
(g) 6
(h) 5
(i) 1
(j) 7
(k) 5
(l) 4
(m) 3
(n) 3
(o) 3
(p) 5
(q) 5

III **It was difficult to communicate and collaborate**

(a) 4
(b) 1
(c) 4
(d) 7
(e) 4
(f) 7
(g) 4
(h) 2
(i) 7
(j) 4
(k) 4
(l) 2
(m) 1
(n) 5
(o) 2
(p) 2
(q) 2

IV **The browser based implementation is useful**

(a)	6	(j)	7
(b)	6	(k)	7
(c)	7	(l)	6
(d)	7	(m)	6
(e)	6	(n)	6
(f)	5	(o)	7
(g)	6	(p)	5
(h)	7	(q)	7
(i)	7		

V **I will use Lich.js in a future project**

(a)	7	(j)	6
(b)	5	(k)	5
(c)	5	(l)	7
(d)	4	(m)	2
(e)	4	(n)	7
(f)	1	(o)	5
(g)	2	(p)	5
(h)	7	(q)	2
(i)	6		

D.2 User Evaluation Questionnaire Responses

I **How did the language effect your ability to develop musical ideas?**

(a) The syntax is very intuitive, and i can write musical ideas with a few lines of code.

(b) I'm an old database warhorse (retired). Lich looks very intriguing as a non-lazy (fast) functional programming implementation of Haskell. However since I don't (yet) do FP haven't really settled into working with declarative patterns, I can't say. Monads are really going to be of major utility here that's not for a first Lich 101 course. Right now I'm prototyping music ideas w/ a spreadsheet, getting the algorithms right.

(c) Pattern sequencing was easy and fun. I have no Haskell experience, so syntax was somewhat of a stumbling block, but not too bad...

(d) The processing of patterns is clearly key for development of musical ideas, but I didn't get time to get into this very well - things suddenly get a lot more complex. I'd suggest adding some examples of "musical ideas" using patterns earlier, before the long series of mathematical demonstrations.

(e) The language features syntax and ugen that are similar to pre-existing systems, however, the framework allows much faster implementation of ideas.

(f) If ever feel like toying with music I now know a tool that I could use

(g) The language was natural and easy, and the API seems built to handle both simple and advanced music.

(h) So far due to my limited exposure of it no wild musical ideas came to me except it would be a fantastic way to do some procedural melodies in a game I am thinking about and it would work perfect for the sound effects. Combine that with needing no pre–generated files to send across the line and thats amazing. Now that I typed that more and more ideas are flooding in.

(i) I couldn't really understand it. I don't know Haskell, and I couldn't understand how functions were being created and applied.

(j) It really helps that I am fluent in SC thinking this might be a little difficult for an unfamiliar user.

(k) The language is quite extensive and allowed me to easilt express my musical ideas. I've worked with MaxMSP on the PC and lich.js provides functionality for me to do a lot of what I'm able to do in MaxMSP, in the browser itself (with a little effort that is)

(l) This is difficult to answer because i am a SuperColldier user, so the language was very familiar to me. I can say it was easy to develop musical ideas, I like the scaleList

(m)

(n) I see potential. I want to devote more time to learn this.

(o) I am a developer but I suck at functional programming. It's cool but the learning curve is super steep.

(p) prepare a page with all the music is a new way of thinking for me, I'm used to wave sequencers

(q) it made me feel capable of having rhythm and tune up the notes

II **If you used the graphics functionality, how did the language effect your ability to develop visual ideas?**

(a) The graphics functionality didn't run in my computer.

(b) n/a

(c) I have nothing intelligent to say on this.

(d) Didn't get to try these, although I tried - perhaps some server crash?

(e) The ability to utilize the pattern streaming system with the graphics was incredibly intuitive and useful.

(f) Didn't use it. Probably want to mark this as optional.

(g) It was good until it set the background to a color that made the text hard to read :) And I didn't quite understand the purpose of the graphics, were they only to help produce music (then they'd be discarded)? Or is the goal to produce a graphics and music composition, say for a game? The latter would be better than the former IMO, though game creation is a lot more complex. Sorry if this was covered but there was a lot of text to read.

(h) This part is fantastic and now thinking back to the last question for the musical ideas it would, if thought out carefully make a wonderful way for me to show the students I have each year and my team on how to rapidly sketch out an idea in a matter of minutes. Of course there are dozens of pieces of software that do this but quite a bit of the things we do revolve around making solutions for people using the boundaries of the web browser, this suddenly becomes a wonderful tool.

(i) Never got that far.

(j) Sorry, didn't go there yet ... not usually my thing

(k) Did not use the graphics functionality.

(l) Still not

(m) did not use

(n) I didn't work with graphics functionality.

(o) Hard to think about to use the FX with your music when learning

(p) interesting, though I prefer fullscreen shaders instead of using cubes and sphere, but that's interesting

(q) not bad

III **Can you describe the most negative aspects of your experience?**

(a) I couldn't make it work the graphics functionality.

(b) Please break the sample down into multiple, smaller samples. It's too much to digest as a wall of text. :)

(c) Having multiple people playing around trying to figure everything out while I was trying to figure things out is a little frustrating. It might be nice if there was a way to mute the other people on the server during experimentation... a solo Lich I suppose.

(d) Lost all state and ability to experiment once another user joined. No response to chat, so not sure if they were aware of me. The split window was confusing, and seemed a bit unstable - whole screen refreshed unpredictably, and I couldn't tell where my edit point was for a while.

(e) The initial colour and font selection was hard to read initially without asking chrome to zoom in further.

(f) As a vim user I dislike the fact that the editor is always in insert mode.

(g) It was a little long. I understand that there's a lot to cover, but as an elevator pitch it could use a little slimming down. Don't be afraid to skip intermediate steps; it felt more like a tutorial than a demo.

(h) When I had to stop and berate myself for realizing why CTRL-. was doing nothing, I am on a Mac!

(i) I couldn't figure out what was going on.

(j) The general problem with live coding is being sort of locked in with the speed of interaction and just the beat based lockstep thing rules out getting "atmospheric" or post-cagean ;-) I am not sure how you would pass interactions between collaborators, but I guess once you know the function names that might be possible, but not exactly obvious

(k) N/A

(l) The confusion between use coma (,) or not. The ctrl + point when I was jammin with others was difficult to listen again the partners code. It is difficult to undesrstand how produces wich sound. When you change to other window.

(m) I wish I could use functionality of javascript. and just have string commands/-javascript commands to the lich.js interface.

(n) mouseX, I couldn't figure out how to use.

(o) Not sucking at learning functional programming

(p) no OSC support, no fft analysis

(q) I prefer much more music thing than the visual, and i would like to record

IV **Can you describe the most positive aspects of your experience?**

(a) I enjoyed writing so easily patterns and chords.

(b) IT'S VERY COOL. Lich.js's dev cycle is just getting underway, playing with it real–time on a website leaves a good strong 1st impression It opened up a door I thought was closed in web dev, which is using a FP language to work with complex models on the web browser. And yes, I *hate* Java I've seen people bleed serotonin over Javascript callbacks other issues.

(c) All the options for sequencing are really inspiring. And I can see how the collaborative aspect would be fun when working with people who knew what they were doing :)

(d) Great to work in browser - easy startup, and sample exercises start well

(e) The speed at which you could implement complex synthesis and patterns was a joy. The fact that this can be done both alongside graphical developments and others is excellent.

(f) When editing and playing a sequence it seems to update on the next cycle instead of abruptly resetting or even worse - looping over.

(g) The implementation was functionally flawless and creatively impressive, and I can tell that a lot of hard work has been put into it.

(h) Everything besides my moment of stupidity.

(i) There was somebody else on at the same time, and it was kind of cool when our sounds meshed well. Happened. Sometimes.

(j) I got results immediately and really appreciated the ugen and list dumps. Like any good musical instrument or context it was quickly apparent that with practice the results would become more finely tuned and personal

(k) Most of my experience was positive. Lich.js is easy to use and has a lot of features which allowed me to program music. The demo was also very well made

(l) Definitely the possibility to interact with others through a browser. The chat is very nice, the sync is very precise. That you have not the possibility to save the code.

(m) Ease of setting up progression.

(n) Really fun stuff.

(o) hearing sound within 5 seconds of doing commands in the demo. Super cool

(p) motivates me to jump into live coding audio-visuals

(q) creating a sequencer and being able to record would do the trick for a great music compositor

V **Are there any further comments you wish to make?**

(a) Is a great language.

(b) 1) I didn't want to answer "It was difficult to communicate and collaborate." b/c I never tried that feature. 2) I've been impressed by what FP/Haskell brings to the table in terms of prototyping concision in app development. I'm looking at building a collaborative computer-assisted composition application, one along the lines of "Hyperscore", definitely geared toward pedagogy therapeutic goals. So what has been developed here could be nothing short of VERY VERY important.

(c) This is fun! It might be nice to have some higher-level synths included. After I send a chat message, it takes me back to the code editor, but there are lots of times I want to send more than one message and then I find myself suddenly typing msgs into the code editor.

More info/examples of the various audio effects would be nice in the tutorial. I admit to feeling slightly uncomfortable experimenting with them while other people are also using the system... I don't want to accidentally blow anybody's ears out.

The GUI had occasional minor problems with scrolling. I'm curious to see how it scales to having more than two people using it at once... do you eventually get to the point where each person can only see a couple of lines at a time? Hmmm. Not sure if this better than simply using a single shared editor, but it is really nice to see all the scrolling and highlighting that others are doing. On that note, it would be nice if executed code was flashed to help provide further awareness of collaborator activity.

(d) keep it up - great start!

(e) This is a very very promising project and I hope to see it being used by the community!

(f) Voice communication (with Push-to-Talk support and preferably the default option) might be preferred over text chat.

(g) I'm not sure I experienced anything that could be used to collaborate, other than the chat functionality. Is there an ability to stream your music or graphics to another user? And/or can you share edit code in real–time?

(h) How much work will go into this lich.js? Will there be room for others to add expansions for adding extensions? Like working with different inputs? I can go all day in this box :)

(i) People don't pick up programming as quickly as you think. I'm pretty experienced (35 years, +20 years PhD in CS), and I really couldn't figure out what was going on from the comments and examples.

(j) There is some huge potential here. Another developer and I are working on doing JavaScript in the browser which we thought this might be doing, but clearly this is on the server side which makes sense. I am assuming you are serving SC on that side. This is a spectacular set of ideas that will take more practice and focus to improve, but I really enjoyed my brief stint and will be showing it to my students. I am reminded of GIBBER which I am sure you are aware of.

(k) The only thing which I felt was lacking in lich.js is that there should be a way to just play lich.js code files directly. That way I could play a lich.js code file on a javascript event.

(l) Very nice system, I really like it.

(m)

(n) I want to try collaboration in the future. I didn't communicate with anyone while trying Lich.js.

(o) SUPER NEAT project. I do audio hacking and PureData performances using a NodeJS Websocket/OSP gateway. I am trying to figure out how to use Lich.js with my stuff. I forked the Lich.js project (aphexddb/Lich.js) so can tweak the ports and whatnot.

(p) this is after 20 mn using lich.js, subject to evolve

(q) I would love to learn making music in this kind of way

D.3 Lich.js Thematic Analysis Codes

- Syntax is intuitive
- Few lines of code
- Intriguing
- Functional Programming still new
- Still prototyping musical ideas
- Patterns are easy
- No Haskell experience
- Syntax is confusing
- Patterns are key
- Suddenly complex
- add musical examples
- Syntax is pre-existing
- fast implementation
- toying with music
- language is intuitive
- simple and advanced
- Good at procedural melodies
- perfect for sound effects
- lots of ideas
- language is confusing
- can't understand Haskell
- previous experience helpful
- difficult for new user
- Easy to express ideas
- Able to express ideas
- Browser implementation useful
- SuperCollider experience helpful
- musical ideas easy
- scale list good
- Has potential
- Need more time
- Learning curve is steep
- New way of thinking
- Felt capable of rhythm and melody
- Graphics didn't work
- Didn't use graphics
- Nothing to say about graphics
- Patterns and Graphics powerful
- Graphics made text unreadable
- Graphics fantastic
- useful for pedagogy
- Browser implementation is novel
- Hard to use graphics
- Prefer shaders to geometric shapes
- Graphics not bad
- Graphics most negative
- Demo too dense
- Multiple people frustrating
- Mute people useful
- Other users broke Lich
- Other users didn't respond
- Split window confusing
- Couldn't easily edit with other users
- Font color difficult to read
- Wanted VIM mode
- Demo too long
- Felt like tutorial, not demo
- Incorrect keybindings on mac
- Confused
- Live coding locked into beat based music
- Collaborative features unclear

- Syntax is confusing
- Difficult to listen with collaborators
- Difficult to understand who produces which sound
- Wanted native JS functionality
- Couldn't figure out mouseX
- Function programming difficult
- Lacked OSC and FFT
- Preferred music to graphics
- Patterns and Chords easy
- Very Cool
- Functional Programming useful
- Sequencing inspiring
- Collaboration would be fun with experienced users
- Browser implementation easy
- Development speed was a joy
- Music, graphics, and networking together is excellent
- Patterns update next cycle
- Functionally flawless
- Creatively impressive
- Moment of stupidity
- Collaboration sometimes cool
- Immediate results
- Benefit from practice
- Easy to use
- Feature Rich
- Demo well made
- Collaboration potential mostly positive
- Chat is very nice
- Sync is precise
- Easy setup most positive
- Really fun
- Fast sound generation cool
- Motivational for live coding
- Sequencer and recording would be useful
- A great language
- Never tried collaboration
- Impressed by functional programming
- Fun
- Built in synth would be nice
- Chat worked different than expected
- Wanted more info and examples
- Felt uncomfortable with other users online
- Didn't want to disturb other users
- GUI had minor problems
- Curious about scalability
- Shared editor might be better
- Wanted executed code flashing
- Great start
- Promising project
- Hope to see it used
- Would prefer voice communication over chat
- Nothing useful for collaboration but chat
- Unclear about collaborative features
- Interested in new features
- Difficult to learn
- Language is confusing (3?)
- Huge potential
- Spectacular set of ideas
- Will show to students

- Reminded of Gibber
- Wanted file playback
- Really liked it
- Didn't collaborate, but wanted to
- Really liked it
- Interested in using Lich.js
- Interested in learning more

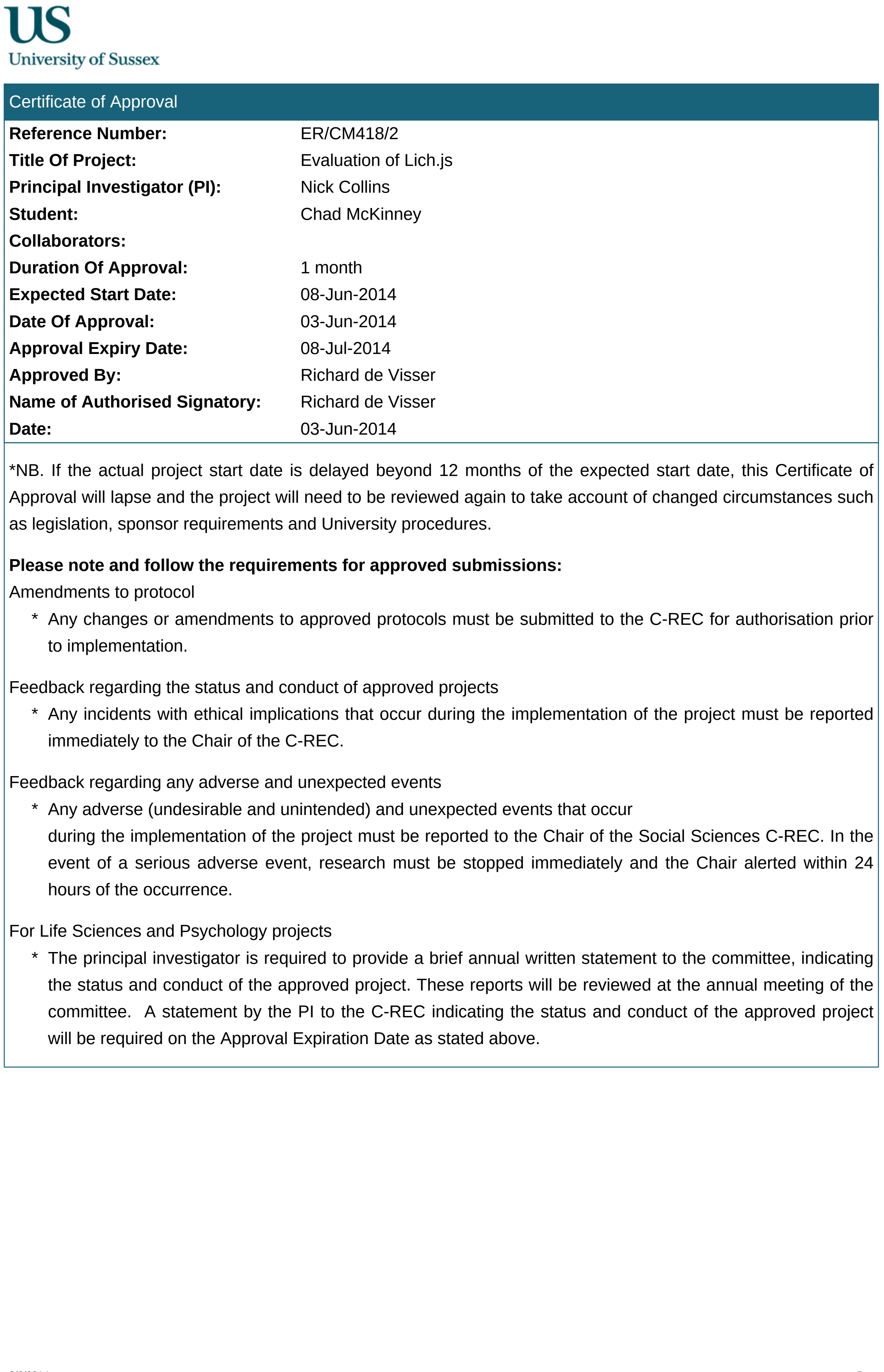
D.4 Lich.js Thematic Analysis Sub-Themes

- Lich.js is intuitive, fast, and easy
- Lich.js is unintuitive, slow, and difficult
- Lich.js is an instrument, requires practice, and can be played well or poorly
- Functional programming is foreign and difficult
- Functional programming is useful and powerful
- Previous computer music language experience is useful for learning Lich.js
- There is similarity to pre-existing systems
- The demo had too much overwhelming material
- The demo needed more material or was lacking
- The demo was well made
- Lich.js is a useful or potentially useful tool
- Browser based implementation is useful
- Lich.js is inspiring and is feature rich
- Lich.js is uninspiring and lacking functionality
- There is interest in the continuing development of Lich.js
- The graphics were not compelling and poorly implemented (much more dominant)
- The graphics were compelling (much less dominant)
- Collaboration was confusing, difficult, and poorly implemented
- Collaboration was frustrating, awkward or uncomfortable
- Collaboration would be compelling or useful, Given the right context
- Collaboration wasn't interesting or never tried to collaborate
- Communication was difficult
- Chat was useful
- Chat was not useful
- Lich.js worked well technically
- Lich.js worked poorly technically

D.5 Lich.js Thematic Analysis Major Themes

- Web based music systems are desirable
- Anonymous live coding collaboration using Lich.js is difficult and potentially undesirable for some users
- The graphics implementation had several technical issues and many users were uninterested in this functionality
- Despite technical issues, the novelty and utility of Lich.js is recognized

D.6 Ethics Approval



University of Sussex

Certificate of Approval

Reference Number:	ER/CM418/2
Title Of Project:	Evaluation of Lich.js
Principal Investigator (PI):	Nick Collins
Student:	Chad McKinney
Collaborators:	
Duration Of Approval:	1 month
Expected Start Date:	08-Jun-2014
Date Of Approval:	03-Jun-2014
Approval Expiry Date:	08-Jul-2014
Approved By:	Richard de Visser
Name of Authorised Signatory:	Richard de Visser
Date:	03-Jun-2014

*NB. If the actual project start date is delayed beyond 12 months of the expected start date, this Certificate of Approval will lapse and the project will need to be reviewed again to take account of changed circumstances such as legislation, sponsor requirements and University procedures.

Please note and follow the requirements for approved submissions:
Amendments to protocol

* Any changes or amendments to approved protocols must be submitted to the C-REC for authorisation prior to implementation.

Feedback regarding the status and conduct of approved projects

* Any incidents with ethical implications that occur during the implementation of the project must be reported immediately to the Chair of the C-REC.

Feedback regarding any adverse and unexpected events

* Any adverse (undesirable and unintended) and unexpected events that occur
during the implementation of the project must be reported to the Chair of the Social Sciences C-REC. In the event of a serious adverse event, research must be stopped immediately and the Chair alerted within 24 hours of the occurrence.

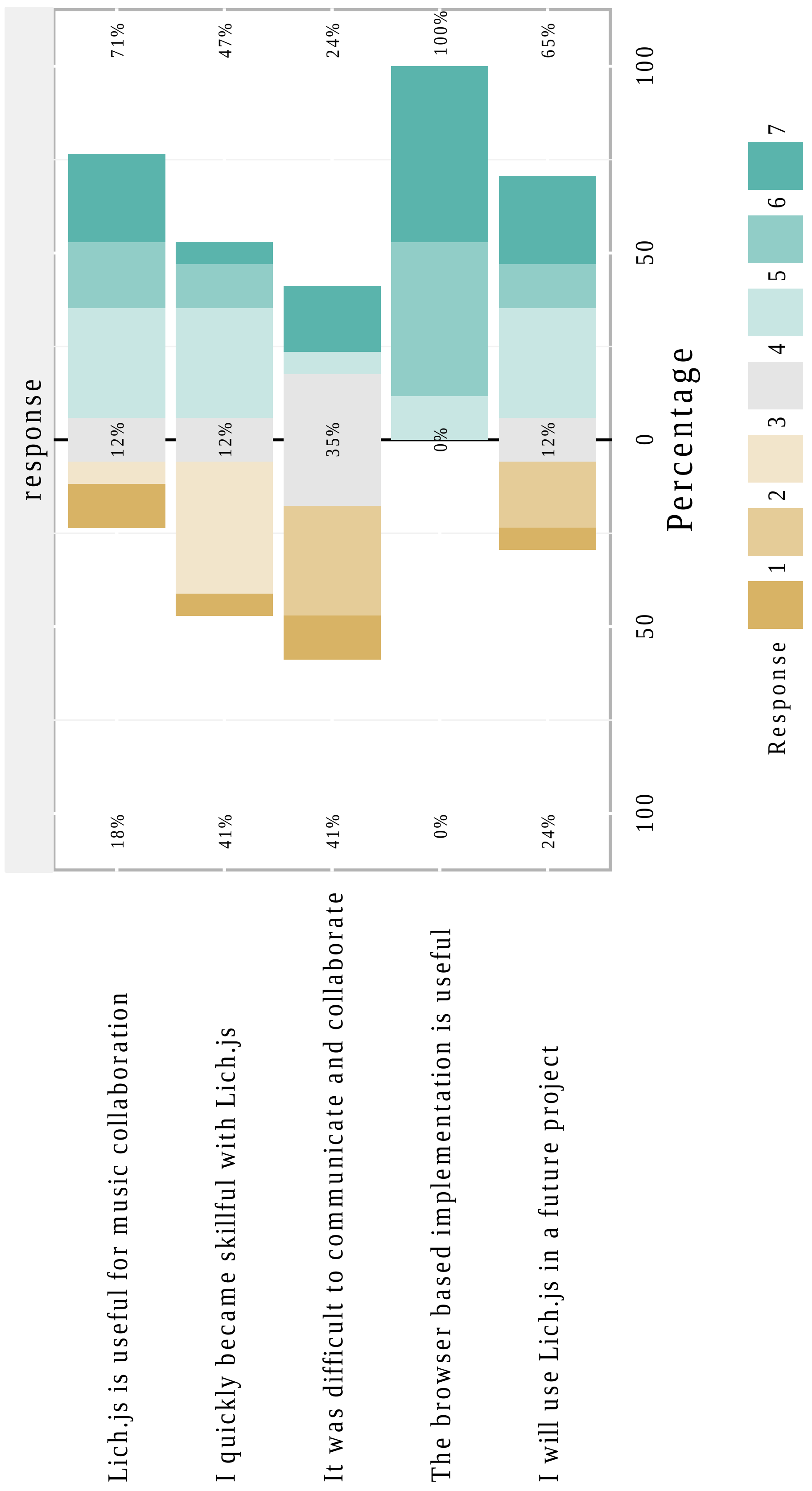
For Life Sciences and Psychology projects

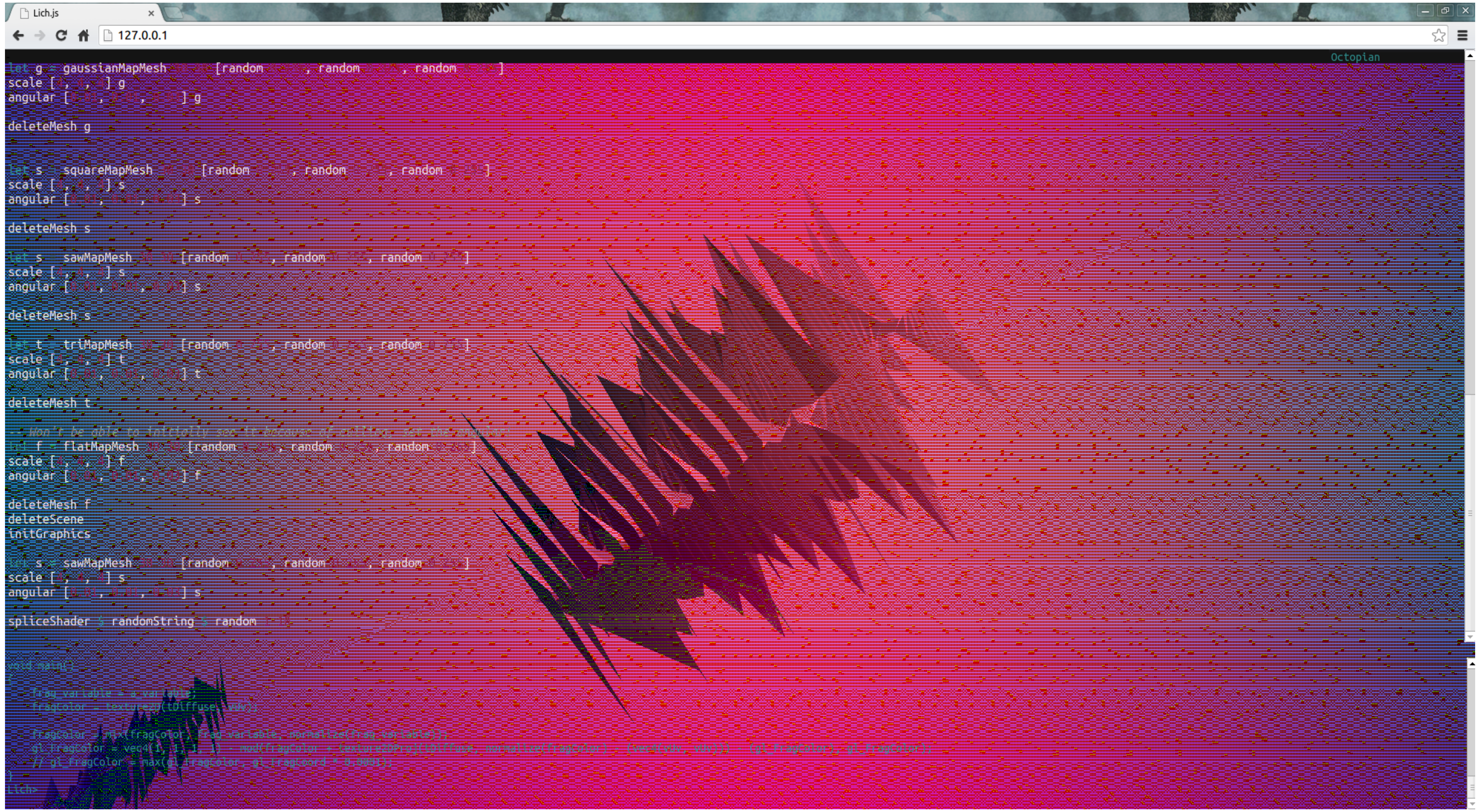
* The principal investigator is required to provide a brief annual written statement to the committee, indicating the status and conduct of the approved project. These reports will be reviewed at the annual meeting of the committee. A statement by the PI to the C-REC indicating the status and conduct of the approved project will be required on the Approval Expiration Date as stated above.

D.7 Lich and JavaScript Benchmark Test code

```
-- Fibonacci test code in Lich.js
let fib n = if n == 0 then 0 else if n == 1
  then 1 else (fib (n - 1)) + (fib (n - 2))
-- Fibonacci test code in JavaScript
function jsFib(n) {
  var s = 0;
  if(n == 0) return(s);
  if(n == 1) {
    s += 1;
    return(s);
  } else {
      return(jsFib(n - 1) + jsFib(n - 2));
    }
  }
}
-- Case test code in Lich.js
case "Zombie" of
    "String" -> "String"
    20.1 -> "20.1"
    30 -> "30"
    False -> "False"
    _ -> "WildCard"
-- Case test code in JavaScript
function jsZombie() {
  switch("Zombie") {
    case "String": return "String";
    case 20.1: return "20.1";
    case 30: return "30";
    case false: return "false";
    default: return "WildCard";
  }
}
-- Binary operator test code in Lich.js
let seven = 1 + 2 * 3
-- Binary operator test code in JavaScript
seven = 1 + 2 * 3
-- List test code in Lich.js
[1,9..9999]
-- List test code in JavaScript
function jsListRange() {
  var res = [];
  for(var i = 1; i < 9999; i+=9) {
    res.push(i);
  }
  return res;
}
```

Figure D.1: Test code in Lich.js and JavaScript


response
Lich.js is useful for music collaboration
I quickly became skillful with Lich.js
It was difficult to communicate and collaborate
The browser based implementation is useful
I will use Lich.js in a future project
18%
41%
41%
0%
24%
12%
12%
35%
0%
12%
71%
47%
24%
100%
65%
100
50
0
50
100
Percentage
Response
1
2
3
4
5
6
7



```
let g = gaussianMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] g
angular [0.01, 0.01, 0.03] g

deleteMesh g


let s = squareMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] s
angular [0.01, 0.01, 0.03] s

deleteMesh s

let s = sawMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] s
angular [0.01, 0.01, 0.03] s

deleteMesh s

let t = triMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] t
angular [0.01, 0.01, 0.03] t

deleteMesh t

-- Won't be able to initially see it because of culling, set the angular!
let f = flatMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] f
angular [0.01, 0.01, 0.03] f

deleteMesh f
deleteScene
initGraphics

let s = sawMapMesh 30 30 [random 0 255, random 0 255, random 0 255]
scale [4, 4, 4] s
angular [0.01, 0.01, 0.03] s

spliceShader $ randomString $ random 1 10


void main()
{
    frag_variable = a_variable;
    fragColor = texture2D(tDiffuse, vUv);

    fragColor = mix(fragColor, frag_variable, normalize(frag_variable));
    gl_FragColor = vec4(1, 1, 1, 1) - mod(fragColor + texture2DProj(tDiffuse, normalize(fragColor) - (vec4(vUv, vUv)) - (gl_FragColor), gl_FragColor);
    // gl_FragColor = max(gl_FragColor, gl_FragCoord * 0.0001);
}
Lich>
```

Figure D.2: Generative visuals created with Lich.js