Yig, the Father of Serpents: A Real-Time Network Music Performance Environment

Chad McKinney
University of Sussex, UK
C.Mckinney@sussex.ac.uk

Nick Collins
University of Sussex, UK
N.Collins@sussex.ac.uk

ABSTRACT

Approaches to network music performance are often focused on creating systems with minimal latency and maximal synchronicity. In this article we present *Yig, the Father of Serpents*, a new program for performing network music that is designed with these principles in mind, but also offers an argument for a different approach. In Yig, users may have identical states yet the audio rendering could be different. In this paper an introduction to the interface is followed by a brief description of the technical development of the software. Next, the instrument is classified and analyzed using existing frameworks and some philosophy behind divergence in network music is explained. The article concludes with an numeration of potential software improvements and suggestions towards future work using divergence

1. INTRODUCTION

Yig, the Father of Serpents is a new program 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* [1] as well as the ancient symbol of the ouroboros [1], both as metaphors for feedback and recursion. 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 [2] [3] [4].

Here instead we have designed a system that, while using sophisticated techniques for low latency and high levels of synchronicity, embraces divergence in the network' sonic result. 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 created cable creation. 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 [5]. This concert presents the six member ensemble as bifurcated sub ensembles networking locally. The two trios communicated with each other via a phone line 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 for their own projects.

In the next section we 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.

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

2. 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 [6] [7]. 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 [8]. 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.

2.1 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 app Jasuto, although in Yig only disconnections are proximity based while cables are created during collisions [9]. 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 is impossible to reproduce with physical equipment

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.

2.2 GUI Development

Yig is written in C++ with heavy reliance on the Qt framework [10]. Qt was chosen because of its strong cross platform GUI application programming interface (API) which

Figure 1. Screen shot of Yig, the Father of Serpents.

greatly simplifies and streamlines the development process. Creating Yig 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.

Qt provided invaluable during development, however the early decision to use the QGraphicsView and QGraphicsScene paradigm for the creation and manipulation of graphics elements has proven 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 is much worse than anticipated. With only six synth objects, which are very simple circle graphics, the system idles at 20 percent CPU usage on a two year old Macbook Pro. When the synths are moved, triggering redrawing of the items and collision detection, the CPU can jump dramatically, upwards to 70 percent. We attempted many approaches to increase performance, such as multithreading and pre-caching reusable graphics items, however the results are still slower than preferable.

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.

2.3 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 sin-

gular package for distribution. Libscsynth was chosen over other options, including a custom audio engine, because it is open source, lightweight, robust, and high quality [11]. However, most importantly, we wanted Yig to be an instrument that allowed 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 def template to create new sounds.

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 CheckBadValues.ar in most, if not all synths, to prevent unstable events from destroying a performance. Currently there is no global tempo synchronization; In some performances Demand rate ugens have been used to create sequenced rhythm, however you cannot guarantee a shared downbeat between instances.

3. 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 network, however the audio fidelity of the system is not effected and often these events go unnoticed by audience members and even the performers themselves.

3.1 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 new system written by Curtis McKinney with some work by Chad McKinney that is still undergoing active development but may see an open beta release in the near future. 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 entire network. Because the server runs on a rented machine on the open internet, it bypasses many of the problems 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 [12]. 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 a message is sent, it is likely that it will 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 interpretting 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.

3.2 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. Technology should not attempt to restrict the direction of aesthetics in music, but rather attempt to facilitate the needs of musicians.

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 we prefer to view it as a unique resource.

4. EVALUATION

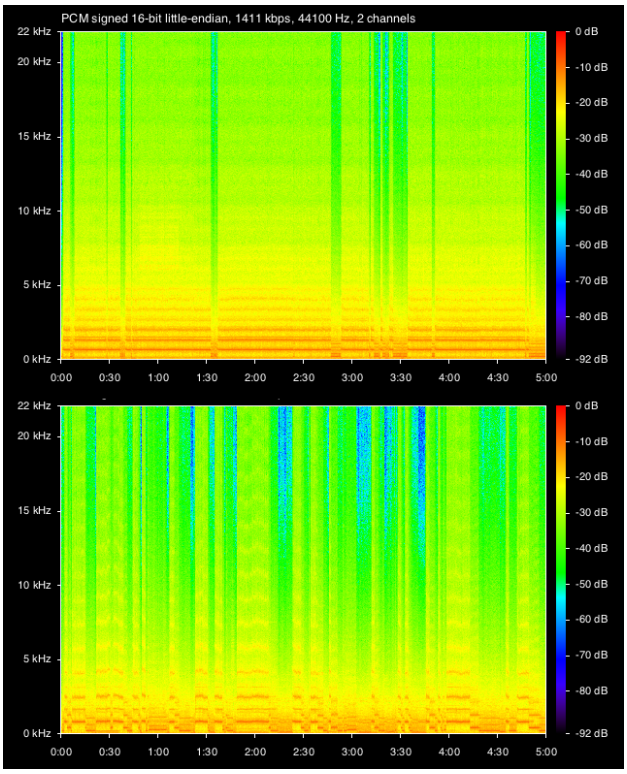

Figure 2. Comparison of recordings from two nodes with identical states. Listen at: `http://chadmckinneyaudio.com/SMC2012`

4.1 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.

The spectrogram analysis of the two recordings is provided here in Figure 2. The two recordings show some clear differences: both recordings have similar harmonic content, but the punctuations do not align in number or placement. 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.

4.2 Epistemic Dimension Space and Internet Music Taxonomy

It may be useful to understand Yig within the context of previous attempts to classify and analyze similar instruments. We have chosen two useful frameworks for this analysis, Andrew Hugill's internet music taxonomy [13], as well as Thor Magnusson's epistemic dimension space [14]. Hugill's taxonomy provides five categories that broadly define several established approaches to creating music using the internet.

I. Uses the Network to Connect Physical Spaces or Instruments
II. Created or Performed in Virtual Environments, or Uses Virtual Instruments
III. Translates into Sound Aspects of the Network Itself
IV. Uses the Internet to Enable Collaborative Composition or Performance
V. Delivered via the Internet, with Varying Degrees of User Interactivity

Table 1. Andrew Hugill's five internet music types.

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.

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 made to other software. Once mapped in the dimension space, underlying design implications are

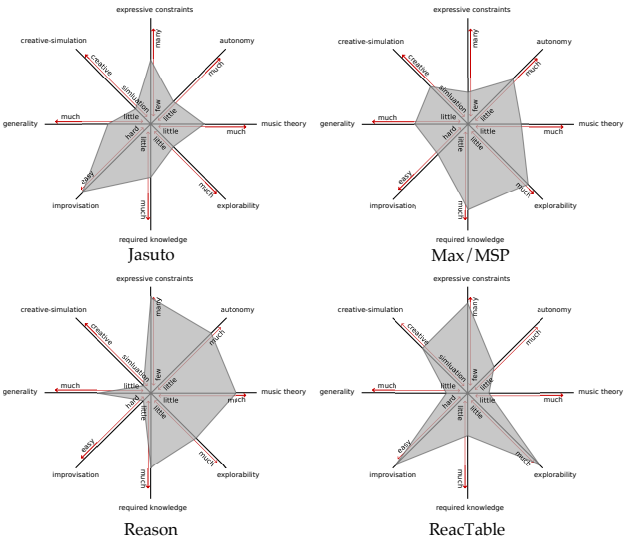

Figure 3. Jasuto, Max/MSP, Reason, and the ReacTable plotted in Magnusson's epistemic dimension space.

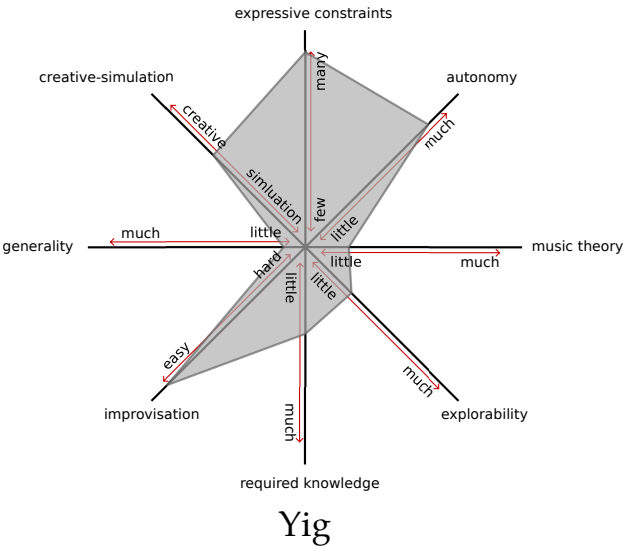

Figure 4. Yig in the epistemic dimension space.

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 explorability 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.

4.3 Demonstrations and Performances

Yig has benefited from several performances as well as a demo at the Digital Music Research Network (DMRN) in 2011 at Queen Mary, University of London. The demon-

stration supplied two laptops for the attendees to play over a local network. This set up provided an opportunity for the attendees to see the networking in realtime as well as hear the separate audio for each laptop. The demonstration ran for over an hour during the poster session, encountering twelve users, and did not crash or malfunction during the event. Reactions to Yig by the DMRN participants were wide ranging from intrigue to confusion. We intend to run a follow up study to more accurately assess user reactions.

Since the DMRN event at Queen Mary, Yig has seen two performances with the laptop band Glitch Lich [15]. The first was at a small noise show in London for a modest audience. Yig performed well, until the middle section of the work, when the computer running the local audio for the performance crashed. However in spite of the crash, we were able to quickly restart Yig and continue where we had left off.

After the crash at the noise show, revisions were made to Yig to prevent further problems. These revisions including checking all synths for bad values, using try/catch blocks in the server buffer fill call, and introducing a few more read/write locks for insured thread safety. The next performance was at the 2012 Network Music Festival in Birmingham and included updated synths as well as an entirely separate visualization program that runs in parallel to the Yig client. Although rehearsal revealed some issues with the wireless connection for the venue, the day ended without any crashes or bugs. This is of note as the group performed from three different locations (Boulder, Gainesville, and Birmingham) and the performance was well received.

5. CONCLUSIONS AND FUTURE WORK

Now that the fundamentals of the program have been established more work can be done 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. Additionally, a formal Human Computer Interaction (HCI) study on Yig is planned to take place in the autumn which will provide useful information for further improvements to the interface. Finally, we intend to release Yig as open source and freely downloadable under the GNU General Public License version 3 to coincide with the Sound and Music Computing conference this year.

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.

6. REFERENCES

[1] Z.B. Bishop and H.P. Lovecraft, *The Curse of Yig*. Arkham House, 1953.

[2] A. Barbosa, "Displaced Soundscapes: A Survey of Network Systems for Music and Sonic Art Creation," *Leonardo Music Journal*, vol. 13: 53–59, 2003.

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

[4] F. Schroeder, A. B. Renaud, P. Rebelo, and F. Gualda, "Addressing the Network: Performative Strategies for Playing Apart," in *Proceedings of the 2007 International Computer Music Conference*, 2007, pp. 133–140.

[5] C. Brown and J. Bischoff, "Indigenous to the Net: Early Network Music Bands in the San Francisco Bay Area," August 2002, available from: http://crossfade.walkerart.org/brownbischoff/IndigenoustotheNetPrint.html [Accessed 2 August 2010].

[6] Cycling '74, "Max/MSP," 2012, available from: http://cycling74.com/ [Accessed 12 May 2012].

[7] M. Puckette and Pure Data Community Developers, "Pure Data," 2012, available from: http://puredata.info/ [Accessed 12 March 2012].

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

[9] C. Wolfe, "Jasuto," 2012, available from: http://www.jasuto.com/site/ [Accessed 26 March 2012].

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

[11] S. Wilson, D. Cottle, and N. Collins, Eds., *The SuperCollider Book*. Cambridge, MA: MIT Press, 2011.

[12] T. Sweeney, "Unreal networking architecture," 1999, available from: http://udn.epicgames.com/Three/NetworkingOverview.html [Accessed 16 May 2010].

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

[14] T. Magnusson, "An Epistemic Dimension Space for Musical Devices," in *Proceedings of the 2010 conference on New interfaces for musical expression*, 2010, pp. 43–46.

[15] Glitch Lich, "Glitch lich," 2012, available from: http://www.glitchlich.com/ [March 12th 2012].