# Searching for Gesture and Embodiment in Live Coding

Spencer Salazar
California Institute of the Arts
ssalazar@calarts.edu

**ABSTRACT**

A tension surrounds the role of gesture and embodiment in live coding performance. Considering historical approaches, we propose a reconsideration of text's current seat as the de facto universal medium for programming code. We further indicate alternative paths for the fusion of gesture and live coding. One such path, based on the mobile touchscreen platform and employing a sketching metaphor, has been developed in the form of the Auraglyph computer music programming environment. Auraglyph's unique approach is intended to imbue music programming with a greater sense of embodiment and gesture. We discuss a number of performances involving Auraglyph to assess the degree of its success in achieving these goals.

## 1 Introduction

With its success in the last decade and a half, the live coding performance movement has further highlighted the tension between musical gesture, embodiment, and electronic music creation. To some it may seem strange to combine the intellectual processes of computer programming with musical performance, which is conventionally a highly embodied and gestural activity. When laptop musicians can call forth worlds of sound with taps of their fingers, the role of gesture and the body in musical performance is muddled.

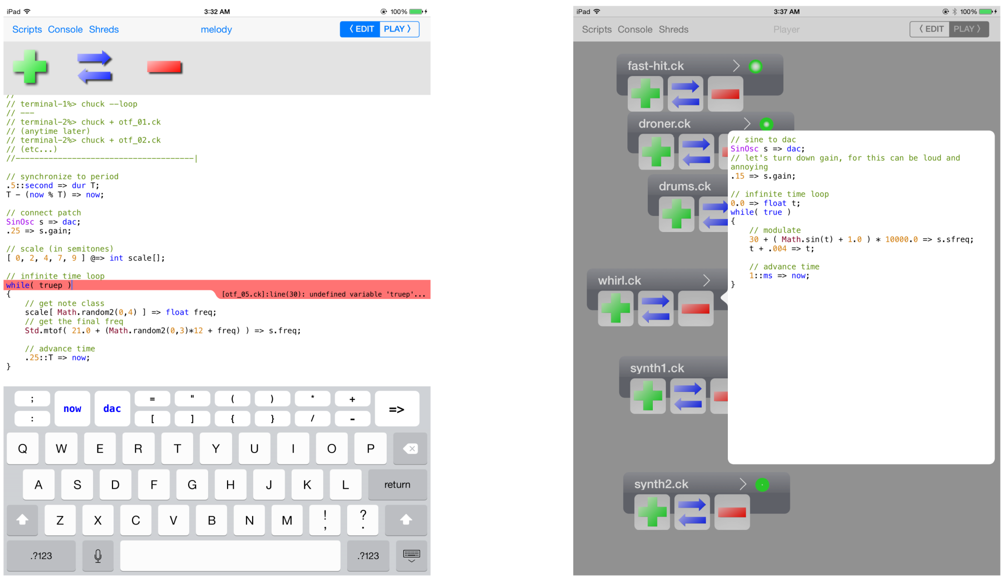
Yet human gestures, mediated by any number of electronic sensors, remain an attractive option for effecting musical action in live coding performance. Many live coding environments struggle to carry out musical actions if they cannot be readily formulated as programming instructions. These actions can be conceptually simple, such as modulating a parameter along some particular path, but if they lack an apparent algorithmic encoding, they are unavailable to the live coding practitioner. Gestural activities can also provide visual interest for the audience over the course of a performance, a function that continues to be beneficial in live electronic music.

However, we maintain that no intrinsic reason is evident that live coding and gesture need remain distinct, if complementary, activities in computer music performance. Recent avenues of programming language research show promise at bridging this gap. Herein we discuss a few such approaches to doing so. Specific attention is given to recent work in the author's Auraglyph research project, a sketch-based audio programming system rooted in both conventional music programming as well as modular synthesis practice.

## 2 Background

Issues related to embodiment and gesture are considered frequently in the discourse surrounding live coding. Collins indicates his and his collaborators' pursuit of "roles for more humanly active live coding," integrating live-coded algorithmic processes into musical improvisation and dance. He further notes historical instances of live coding-like behaviors within modern dance (Collins 2011). Stowell and McLean note the "lack of immediacy" in the actions of live coding performers who are "under pressure to do something interesting in the moment" (Stowell and McLean 2013). They resolve this tension by integrating instrumental and vocal performance modes with live coding performance, thereby combining "continuous organic expression with symbolic abstraction" but effectively dodging the need to solve these issues through programmatic means.

Baalman draws specific attention to the questions of embodiment in live coding in her performance *Code LiveCode Live* (2009), in which the sounds of her typing a live-coded program are manipulated in real-time by that very program. Baalman notes "the physical interaction of our body [. . . ] with the machine" as a primary instance in which live coding incorporates embodiment, in addition to the grammar and structure of the programming language itself, such as the muscle memory of typing idiomatic programming syntax (Baalman 2015). Armitage similarly introduced embodiment to live coding by



```
//
// terminal-1%> chuck --loop
// ---
// terminal-2%> chuck + otf_01.ck
// (anytime later)
// terminal-2%> chuck + otf_02.ck
// (etc...)
//--------------------------------------|

// synchronize to period
.5::second => dur T;
T - (now % T) => now;

// connect patch
SinOsc s => dac;
.25 => s.gain;

// scale (in semitones)
[ 0, 2, 4, 7, 9 ] @=> int scale[];

// infinite time loop
while( truep )
{                                   [otf_05.ck]:line(30): undefined variable 'truep'...
    // get note class
    scale[ Math.random2(0,4) ] => float freq;
    // get the final freq
    Std.mtof( 21.0 + (Math.random2(0,3)*12 + freq) ) => s.freq;

    // advance time
    .25::T => now;
}
```

```
// sine to dac
SinOsc s => dac;
// let's turn down gain, for this can be loud and
annoying
.15 => s.gain;

// infinite time loop
0.0 => float t;
while( true )
{
    // modulate
    30 + ( Math.sin(t) + 1.0 ) * 10000.0 => s.sfreq;
    t + .004 => t;

    // advance time
    1::ms => now;
}
```

Figure 1: *miniAudicle for iPad Editor (left) and Player mode (right).*

the active sonification of typing gestures through motorized actuators, thereby amplifying these gestures' presence in the minds of listeners (Armitage 2016). Hutchins positions live patching of modular synthesizers somewhere between the "abstract and unembodied" nature of text-based live coding and the embodiment of playing an acoustic instrument, noting also the general acceptance of modular synthesis live patching into the scope of live coding (Hutchins 2015).

Live coding's position seems unclear in the wider context of musical gesture research. In a comprehensive review of scholarly notions of musical gesture, Jensenius et al. broadly characterize gesture as a "bridge between movement and meaning" (presumably, that of a musical nature) (Jensenius et al. 2009). Kurtenbach and Hulteen, speaking from the perspective of general human-computer interaction research, explicitly reject the keyboard as a gestural interface (Kurtenbach and Hulteen 1990). Other definitions covered by Jensenius et al. variously consider gesture from the perspective of what can be observed by an audience or, more inclusively, any movement resulting in the production of sound.

In any case, it is not clear that any of these characterizations of gesture admit the movements typically involved in computer programming; these movements are not normally linked directly to a musical result, but rather to the transmission of instructions to a sound-producing machine. This exclusion might be better seen as a shortcoming of the aforementioned notions of gesture rather than of live coding; perhaps we must expand the prevailing definition of gesture in consideration of live coding in musical performance. However, more concerning to the author is the extent to which the scarcity of gestural opportunities in live coding limit musical possibilities in performance.

## 3 Prelude: ChucK on the iPad

Much of the author's original interest in live coding and gesture stems from an initial investigation into developing a programming interface for the ChucK programming language based on Apple iPad touchscreen tablet computers. These efforts are detailed extensively in (Salazar and Wang 2014) and (Salazar 2017) and are briefly summarized here to motivate further discussion. The result of this research was a coding environment called miniAudicle for iPad, so-named after the canonical desktop computer-based development environment for ChucK. miniAudicle for iPad included a touch-based editor for ChucK code, featuring a virtual keyboard for typing code, custom keys for common ChucK language syntax (such as the ChucK operator => and the `now` and `dac` keywords), and other small editing enhancements. To attempt to leverage the intrinsic affordances of touchscreen interaction, miniAudicle for iPad also included a "Player" mode in which pre-composed ChucK scripts could be called up, triggered, and turned off using ChucK's standard add/replace/remove on-the-fly programming actions. This offered a similar interaction to, for instance, a sampler pad that activates loops, in which loops were instead ChucK scripts that could be edited and re-triggered within a performance (Figure 1).

Overall the success of this project was mixed. Musicians and coders who used miniAudicle for iPad appreciated the casual nature of working with code on a touchscreen, but expressed concern over constructing more significant software works they were accustomed to making on desktop programming systems (Salazar 2017). The player mode was mostly ignored in user testing, leaving its promise for merging code and gesture largely unfulfilled. Other possibilities for working with physical interaction and code, such as building interfaces in TouchOSC to control ChucK-based synthesis, were explored, but essentially left coding and performance as two distinct activities.

## 4 Motivation

The experience of creating and using miniAudicle for iPad led to a number of conceptual developments that informed both the research immediately following, and might also provoke further explorations in the future. Fundamentally, mobile devices and touch-based interactions are a compelling medium for musical performance, as documented extensively in the research literature. But in introducing live coding to this medium, miniAudicle for iPad exposed long-standing tensions between algorithm and gesture in creative programming, as present in the experience of the author. Together, these suggest an interesting research direction that shifts away from plain text as the fundamental medium of musical live coding, towards hybrid systems incorporating both algorithmic and gestural capabilities.

### 4.1 Gestural Dynamics of Mobile Touchscreen Devices

Since their introduction into the mainstream computing ecosystem and the computer music research community, mobile and/or touchscreen devices have presented interesting opportunities for immensely physical, gestural, and embodied musical interaction.

Early research in this area includes the work of Tanaka (Tanaka 2004) and Geiger (Geiger 2006), who both sought to use the inputs of commodity personal digital assistant (PDA) devices for music creation. Research in the current generation of mobile technology includes the applications of Smule, an iPhone software developer (Wang et al. 2009). One example of these efforts is Ocarina, a virtual flute-like instrument for the iPhone which used nearly every input sensor and output of the device (Wang 2014). In Wang's words, he "started with a small and simple instrument, the ocarina, [. . . ] and fleshed it out onto the form factor of the iPhone." Ocarina's metaphor imagines the entire physical device as the entire instrument, as if its performer were in fact holding in their hands a real ocarina. The physical dimensions and interface affordances of mobile touchscreen devices in general allow a directness of interaction and strength of metaphor that is not generally feasible with desktop or laptop computers. This directness and physicality intrinsically supports gestural control, an evidently desirable property of musical interaction. In this sense, mobile touchscreen devices are an ideal commodity hardware platform for prototyping and development of new ideas in musical performance. Perhaps it might even be used musical live coding, if one is able to break away from text-based paradigms.

### 4.2 Algorithm and Gesture

Programming languages based entirely on text are, for the most part, limited to musical control that can be readily encoded as an algorithm. Along these lines, we propose a model for how a musical idea might typically be effected in the context of musical programming (Figure 2). Consider the curve in Figure 2, left, which might be imagined by a hypothetical performer and used to control a musical parameter such as volume or frequency over time.

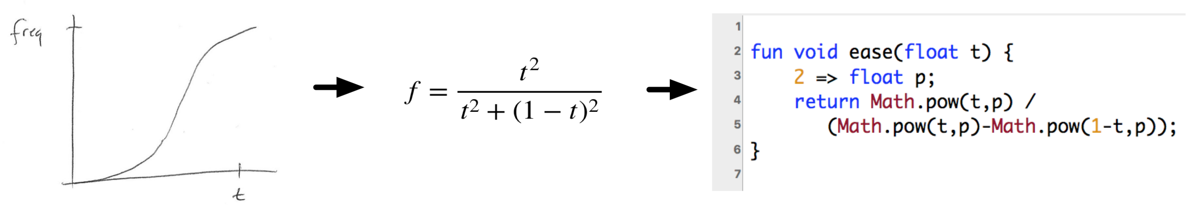
To effect this control pattern within a conventional text programming language, one first needs to determine a mathematical function that adequate describes the curve. In this case, the performer might suspect the mathematical function in Figure 2, center, adequately describes the desired curve. Next, the programmer must encode this function in an algorithm, which might resemble the ChucK code in Figure 2, left, which is intended to be evaluated over some normalized time (from 0 to 1) and scaled to the appropriate range. Upon typing this code, the performer can then execute it and evaluate the desirability of its musical result.

This entire process can be summarized like so:

```
Intent -> Mathematical formulation -> Algorithm -> Audible result -> Musical experience & evaluation
```

The transformations between the performer's intent and its mathematical formulation and between the mathematical forumalation and resulting code often involve significant intellectual effort. Depending on the experience of the performer, this degree of cognition may not be realistically achievable in a concert setting. In such cases, musical control in text-based live coding is limited to that which is arranged and memorized in advance or that which is simple enough to reason about within the constraints of a live performance.

In contrast, supposing that the appropriate musical parameter has some degree of conventional direct gestural control, such as a knob, slider, or similar, the musical intent can be directly translated to musical result by simply adjusting the control in the desired way over time. We propose a model for this process like so:



$$f = \frac{t^2}{t^2 + (1-t)^2}$$

```
fun void ease(float t) {
    2 => float p;
    return Math.pow(t,p) /
        (Math.pow(t,p)-Math.pow(1-t,p));
}
```

Figure 2: *The process of encoding a desired gesture as an algorithm (left to right): intention, mathematical formulation, programming code.*

```
Intent -> Gesture -> Audible result -> Musical experience & evaluation
```

The transformation between intent and gesture is typically intuitive and does not require much intellectual effort, although it is subject to the imprecisions of human dexterity. Furthermore, the directness of this interaction allows for tight feedback between action, effect, and experience, enabling the performer to fine-tune gestures as they are carried out.

This shortcoming of live coding is not limited to live performance situations, and is in fact a general limitation of many creative coding systems whether used for performance or used off-line. After a musician has determined a suitable algorithmic representation of their intent as described above, suppose then they decide they want to change their original idea to curve slightly differently, or to add a bump in the middle. Now they are faced with completely redesigning the mathematical formulation and algorithmic encoding of their musical intent. Whether they are performing live on stage or they have all the time in the world to determine an appropriate algorithm, this cognitive burden ultimately inhibits the musician's ability to explore different creative ideas.

Perhaps it is a theoretical imperative to enhance the concept of gesture to include activities associated with conventional live programming performance, which in many situations involves a great deal of skill and practice and may lead to moments of excitement, intensity, and virtuousity. And perhaps it is flawed to not simply start with the algorithm in the live-coding context rather than an abstract visualization of a musical parameter over time. Indeed, live coding is ideally suited to musical ideas conceived algorithmically that are difficult or impossible to execute with direct gestural control, such as "choose pitches according to the first 200 numbers in the Fibonacci series." However it is ultimately not necessary to choose one or the other; prior work has demonstrated that gesture and algorithm are not mutually exclusive characteristics of live coding.

## 4.3 Text as Abstraction

Text is an extremely convenient medium for programming, being a relatively simple data format that is readable and modifiable by a large variety of editing software programs. However, concerns of embodiment and gesture in live coding stem from its primary interaction of typing and editing text. Distancing a programming system from text is one possibility for reincorporating gesture into this mode of music performance. Text's preeminence as a medium for developing programming code belies the fact that it is an abstraction at the confluence of the programmer's intent and a suitable machine representation.

A programming language implementation will typically parse textual code into an abstract syntax tree which is then transformed into the corresponding machine code; this machine code may be directly executable by the CPU hardware or it may be bytecode for a software interpreter or virtual machine. Mature compilers such as GCC and clang include an additional intermediate representation in between these steps to provide a platform-agnostic basis for code analysis and optimization. In this sense, most programming code is an abstraction of the low-level machine representation the host hardware is capable of processing.

The programmer's intent is not normally a formal specification, but can be thought of as a description of the desired computational behavior in a representation natural to the programmer, like the written English descripton "add 1 to each item of this collection" or a particular curve following a certain path over time from 1 to 0. Advance thought or planning is often necessary to convert this intent to a code representation.

Therefore, text exists as a fixed format for bridging the programmer's intent to an executable hardware representation, but it need not be the only such bridge. Indeed, a wealth of graphically-oriented programming languages exist, including Pure Data, Max/MSP, Kyma, Reaktor, Quartz Composer, TouchDesigner, LabVIEW, Simulink, vvvv, UrMus, and many others;

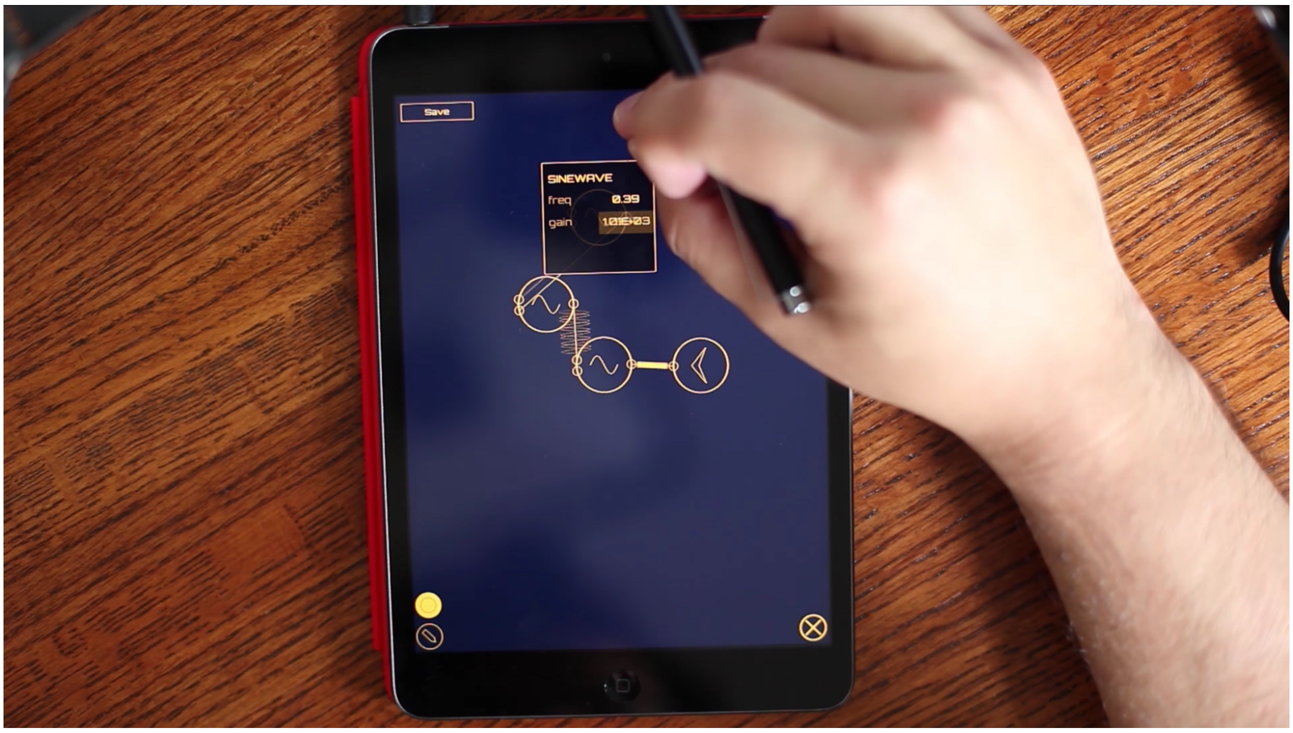


Figure 3: *Auraglyph in use.*

some of these languages retain text entry as a critical path to normal programming practice. Most of these systems follow a dataflow programming model, in which algorithms are encoded via a directed graph.

Imperative and functional programming paradigms have largely been left to textual representations, with some notable exceptions. The most well-known of these exceptions is Scratch, in which imperative programming code is constructed by dragging pre-defined blocks together in a visual canvas (Resnick et al. 2009). TouchDevelop works in a similar fashion, with composable programming blocks selected from an on-screen palette, but has been explicit designed for use on touchscreen-based mobile phone systems (Tillmann et al. 2011). Lisping (https://itunes.apple.com/us/app/lisping/id512138518) is an iOS application for directly editing the parse tree of Clojure and Scheme programs with touch; the Lisp family of programming languages is especially suited to this type of interaction given that source code in these languages is essentially a direct textual representation of the program's abstract syntax tree. Kronos is a functional programming language that can be edited in either text or visual formats; its semantics lend itself to representation and modification either as text or as a visual graph structure (Norilo 2016; Norilo 2012).

Some programming languages for creative computing have largely retained their textual heritage while introducing gestural elements. Field (http://openendedgroup.com/field/) is a Python-based programming environment in which graphical controls like sliders, graphs, and buttons can be placed directly inline with programming code. Field's canvas interface also allows code to be arranged spatially and temporally, with relationships between distinct code blocks defined in a number of graphical ways. Processing (https://processing.org/) features an enhancement called Tweak mode in which literal values within the code, like hard-coded geometric information or colors, can be adjusted on-the-fly while the corresponding sketch is running.

## 5 Sketching Live Code

Auraglyph (Figure 3) is a research music programming platform designed for touchscreen-based tablet computers with these motiviations in mind. Its goal is to make use of the gestural capabilities inherent in touch computing as well as distance itself from typed text as a necessary medium for programming. Auraglyph is a modular patching environment centered around a sketching metaphor. Within an on-screen canvas, extending infinitely in each direction, the programmer/performer directly draws programming structures which Auraglyph reifies into audio-rate and control-rate processing nodes. A dataflow graph structure is created by then drawing connections between these nodes. The nodes are further parameterized or controlled using touch controls. They can also be rearranged, disconnected, or deleted. The resulting audio and control graph generates audio in real-time in response to these modifications. The stylus can also be used to draw free-form structures that are left unchanged by the system, to use as annotations or decorations for the program.

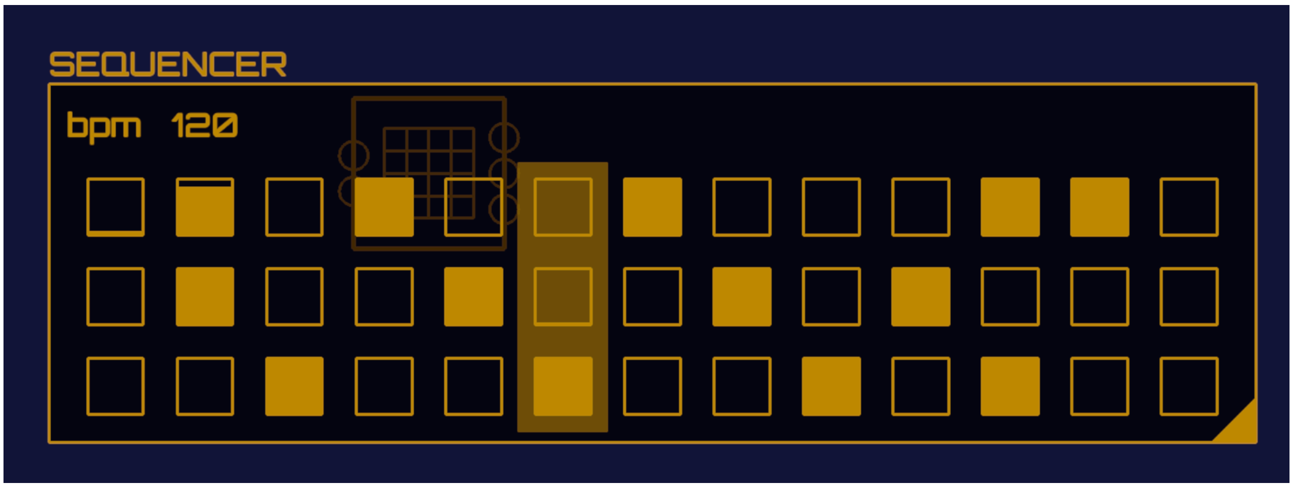


Figure 4: *Auraglyph's sequencer node.*

Performance practice in Auraglyph has developed in a number of ways. As a basic musical gesture, the properties of any node, e.g. an oscillator's frequency, a delay time, or an envelope's duration, can be adjusted using scrub controls that appear in an editor window after tapping the target node. These scrub controls are pressed and then moved up or down with a finger to adjust the value. Parameter values can be set directly by writing the desired value into an editor box, enabling immediate change. Most of these parameters are also able to be modulated by connecting any other node to the desired property. Node connections can be made or broken as a performative gesture to quickly activate or deactivate processing paths. Some nodes have extended interaction capabilities, such as a step sequencer which allows adjusting step values and durations (Figure 4) or a waveform editor which allows one to directly draw a waveform on screen.

An `Orientation` node supplies the device orientation in the form of pitch, roll, and yaw Euler angles, which can be used to manipulate musical properties of the synthesis graph. A `Stylus` node provides stylus x/y position in addition to pen pressure, angle, and rotation for use as a musical control (when these attributes are available). These capabilities facilitate the rapid development of new musical instruments using these controls and allow further integration of embodied gesture into a performance. However, interaction with these controls is not really "programming" per-se, in the sense that adjusting the physical properties of the device is not actually causing anything to be programmed, even if it complements live coding activities within a the same performance.

Auraglyph's visual component also lends itself to live performance scenarios. Its graphic design is intended to capture visual interest, characterized by trim monochrome lines and abstract icons. Furthermore, audio waveforms and control signals are visualized along each connection between two nodes, providing passive feedback to the programmer about the current system state (Figure 5). The relationships between these visualized control signals and waveforms can often give the effect that the program is a living, breathing system.

If the screen is projected, this also provides something interesting to look at for the audience; the waveform visualizations often change in interesting ways as the properties of related nodes are modified and as the waveform is transformed along the graph. In cases where the screen is projected, it is preferable to project video from a camera positioned above the performer's tablet, so that their hands and stylus are also visible. In previous Auraglyph-based performances in which the screen was not projected, audience members afterwards indicated they would have liked to see what was happening on-screen. A performer can also use free-form drawing to communicate non-musically with the audience, similar to how text-based live coders might use comments to speak to their audience.

Creative software is not simply a collection of features but an overall experience, supported by a shifting blend of its constituent parts. A sense of gesture and embodiment arises in Auraglyph from the features described above: the natural and free-flowing nature in which new programming structures are created and connected, the directness through which they are manipulated, and visual and aural feedback provided to the programmer.

## 5.1 Performance Practice

Auraglyph is experimental research software and as such does not have an extensive history of performances to reflect upon in the context of this discussion. A selection of the performances that have involved Auraglyph are discussed here to illuminate the advantages and limitations of its approach to integrating musical gesture and live coding. The first of these, *Pulse*, is an individual work, one of the first composed using Auraglyph, that has been performed several times since its debut in 2016. The second, *Eleven Rituals for iPad ensemble*, is a group piece performed by the Mobile Ensemble of

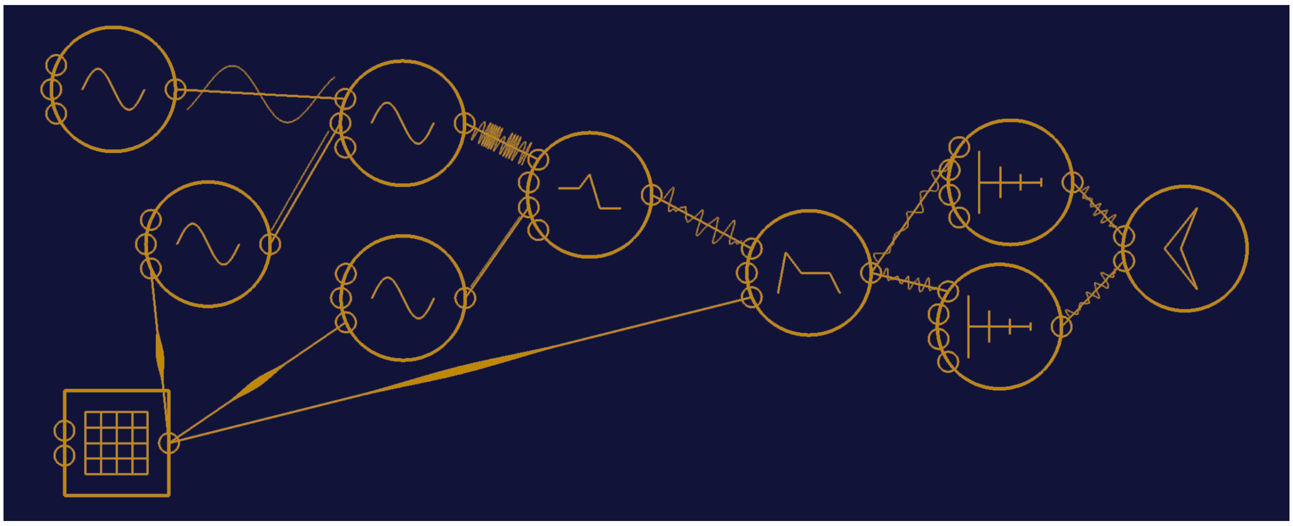


Figure 5: *Auraglyph's visualization of audio waveforms and system state.*

CalArts (MECA). The last performance considered here is one with the Electronic Arts Ensemble workshop at Stanford University, in which several improvisational works were performed in an ensemble of mixed electroacoustic instruments.

### 5.1.1 *Pulse*

*Pulse* is a work for an individual performer using Auraglyph, composed by the author. A video of *Pulse* being performed is available at https://www.youtube.com/watch?v=5JjXEvqPyCA. *Pulse* is performed by loading up an Auraglyph patch prepared in advance with most of the audio and control nodes already configured; with no intervention from the performer, the piece begins, almost immediately after loading the patch, by issuing a resounding, repeating bass pulse. The piece is structured in three parts; the first part mostly consists of drawing connections between nodes that were prepared in the patch ahead of time. These connections activate various parts of the audio graph in addition to integrating different pre-programmed note sequences.

The second part of *Pulse* consists of performing a melody over a repeating bassline. This melody is performed by writing individual MIDI note numbers into an editor window. While this sounds (and, in the video, appears) impractical, it seemed like the best way at the time to perform melodies in real-time with the capabilities provided by Auraglyph. As can be seen in the linked video of a *Pulse* performance, this method of performance is highly subject to error due to imprecision in Auraglyph's handwriting recognition algorithm; these errors are frustrating to experience as a performer.

The final section of *Pulse* involves substantial modification to the audio processing graph of the program. The bass pulse from the beginning of the piece is restored, and distorted with ring modulation. As the pulse repeats, a stereo feedback network is constructed, configured, and further distorted. Four individual feedback delay nodes are added to the chain of four pre-existing feedback nodes, gradually augmenting a persistent din. Then, a series of low-frequency sine oscillators are plugged into the delay times of individual feedback nodes, introducing further sonic disruptions in the form of variable pitch shifting; these pitch shift effects are further processed by the pitch shifting delays further down the chain. Each LFO is configured with a different frequency and modulation depth, and as each is added the overall texture develops into a sonic trash heap of chirps, pops, and unrelated tones. Visually, the graphical waveforms corresponding to these sounds chaotically dance around the screen due to the ring modulation and cascades of variable delay (Figure 6). Following this procedure, the piece is ended abruptly.

As is evident to *Pulse*'s performer and observers, actual physical work is required to perform the piece. Drawing the components of the corresponding Auraglyph program requires fairly precise input to ensure everything ends up in the right place with the correct connections and parameters. It became easier with practice to memorize the timing and placement of changes to the program, similar to performance with a musical instrument.

Moreover, it was eminently possible to make mistakes in performing this piece. The perceptive viewer will notice multiple mistakes in the video linked above, and many more errors were committed in earlier video takes of the same piece. Trueman, in discussing a philosophy of musical instrument design, urges that we view "challenge and difficulty as instrumental virtues" in order to "discover what is ultimately possible with computational-based performance" (Trueman 2007). In this sense, the risk of error, or even catastrophic failure, in performing with Auraglyph might also enable a higher ceiling for performative skill.

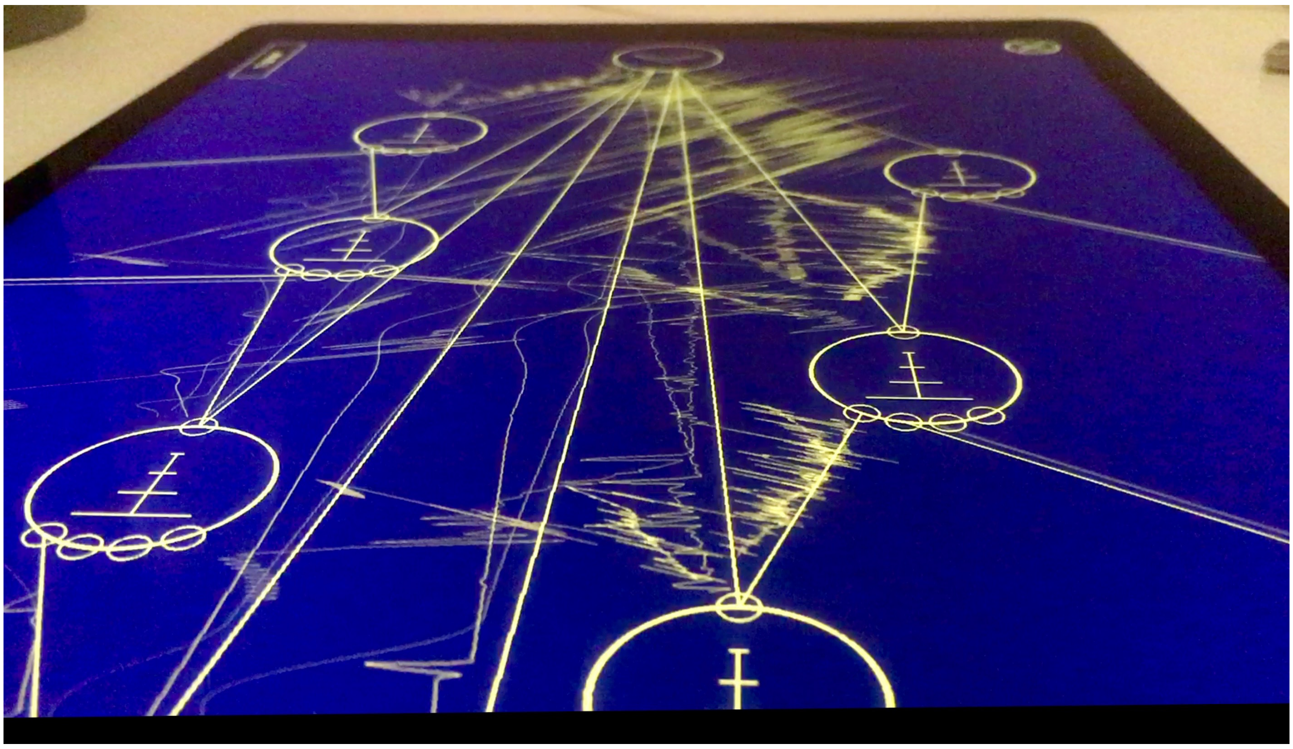


Figure 6: *The ending of* Pulse.

### 5.1.2 *Eleven Rituals for iPad ensemble*

*Eleven Rituals for iPad ensemble* is a piece for a group of roughly eight to twelve performers using Auraglyph on individual iPads. Each iPad outputs sound from its internal speaker. If amplification is deemed necessary in a given performance context, it is achieved using overhead room mics positioned above the group and connected to a house PA system. This enables the physical distribution of the performers and sonic presence of the individual iPads to create a sense of space.

Each member of the ensemble prepares three distinct sound textures in advance, using Auraglyph: a tonal texture, a rhythmic texture, and a noisy texture. What exactly these descriptors mean is left to the individual performer. A conductor/performer leads the ensemble through an eleven-part score, in which each part, or "ritual," proceeds through some arrangement of these textures, varying in duration but each on average lasting a little less than a minute. The conductor is able to direct when each performer begins and finishes each of these rituals as well as transition points within a ritual, such that small temporal offsets create a fluid texture distributed through time and space. Sudden shifts in sounds are occasionally used for contrast.

The primary gesture engaged in by an individual performer in this piece is turning the volume of a texture up or down at the right time. While this is a relatively simple operation, it must be done in synchrony with the rest of the ensemble, with each performer providing an appropriate acoustic presence. This entails a balance of listening to the sound of the ensemble, listening to one's own sound, and adjusting their patch accordingly.
This is not impossible to achieve with text-based live coding, but it seems particularly suited to gestural control. Other gestures used in *Eleven Rituals* included suddenly turning on or off specific textures in synchrony, and, at one specific moment in the piece, each performer chaotically modifying their patch simultaneously.

Beyond simply gesture, this work evoked a sense of embodiment due to the distribution of the performers and their sound through space, and the utilization of this space as a composition tool. The performers in the original rendition of this piece were seated at a large table set apart from the audience, but future renditions might position the table within the audience or even distribute performers among the audience members. These forms of embodiment are not exclusive to mobile, touch-based devices, and in fact the creative use of space, and use of multiple performers, might be employed by traditional laptop performers to heighten the sense of embodiment and increase engagement with audiences, to the extent these characteristics are desired (these are, of course, ideals exhibited by concepts such as the laptop orchestra).

### 5.1.3 Electronic Arts Ensemble

In the summer of 2017, the author participated as a guest lecturer/performer with the Electronic Arts Ensemble workshop at Stanford University, in which he utilized Auraglyph in a final concert at the end of the one-week workshop. The concert consisted chiefly of structured improvisations between eight performers of a heterogenous mix of electroacoustic instrumentation. Most of the works in the concert utilized non-traditional scores or instructions for the performer; among these were, for instance, Pauline Oliveros' *The Witness*, and an original work by one of the workshop participants in which performers interpreted assigned sections of a projected video recording.

In this context, Auraglyph's sketching metaphor proved to be something of a weakness. Drawing programming elements directly was too sluggish to integrate effectively into the often fast moving improvisations, leaving parameterization of existing programming nodes and modifying their connections as the primary musical gestures available. Sketching was also often too slow to fit with the other performers in the ensemble, while parameterization was too abrupt. Auraglyph mostly found success in pieces involving slow, building improvisations, allowing the author time to develop a program and control it effectively. Its reasonable that, with practice, an improvisational framework involving this method of interaction might arise; as yet, such a framework is not evident in the author's experience.

## 6 Conclusion

Gesture and embodiment continue to play an unclear role in live coding practice. It is our belief that the notion of gesture must expand to accommodate new ideas in musical performance created by live coding. But there is also immense potential in expanding live coding systems to enable gestural and embodied control as these are conventionally understood.

The gestural capabilities and embodiment present in mobile, touchscreen device interaction make it an interesting for computer-based musical expression, but, due to its poor support for text input and modification, it might seem an unlikely platform for live coding. However, if we allow that linear streams of text are just one of many possible representations of programming code and, furthermore, that text-based programming has poor support for many types of gestural control, mobile touchscreen devices, and perhaps other emerging forms of computing media, such as virtual and augmented reality, become attractive for live coding systems.

Auraglyph is a sketch-based music programming system developed with these ideas in mind. In Auraglyph, a programmer/performer draws programming nodes onto an open canvas, connecting them and parameterizing them to form a audio processing graph. The overall effect of these features is to imbue the programmer/performer a greater sense of gesture and embodiment when coding in Auraglyph. By examining a number of musical performances in which Auraglyph has been used, we can begin to assess the advantages and limitations of its approach to these concerns.

## References

Armitage, Joanne. 2016. "Revealing Timelines: Live Coding and Its Gestures." In *Proceedings of the International Conference on Live Coding*.

Baalman, Marije. 2015. "Embodiment of Code." In *Proceedings of the First International Conference on Live Coding*.

Collins, Nick. 2011. "Live Coding of Consequence." *Leonardo* 44 (3). MIT Press: 207–11.

Geiger, Günter. 2006. "Using the Touch Screen as a Controller for Portable Computer Music Instruments." In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 61–64. IRCAM—Centre Pompidou.

Hutchins, Charles Celeste. 2015. "Live Patch / Live Code." In *Proceedings of the First International Conference on Live Coding*.

Jensenius, Alexander Refsum, Marcelo M Wanderley, Rolf Inge Godøy, and Marc Leman. 2009. "Musical Gestures: Concepts and Methods in Research." In *Musical Gestures: Sound, Movement, and Meaning*. Routledge.

Kurtenbach, Gordon, and Eric A. Hulteen. 1990. "Gestures in Human-Computer Communication." In *The Art of Human-Computer Interface Design*, edited by Brenda Laurel and S. Joy Mountford. Addison-Wesley.

Norilo, Vesa. 2012. "Visualization of Signals and Algorithms in Kronos." In *Proceedings of the International Conference on Digital Audio Effects*.

———. 2016. "Kronos: A Declarative Metaprogramming Language for Digital Signal Processing." *Computer Music Journal*. MIT Press.

Resnick, Mitchel, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, et al. 2009. "Scratch: Programming for All." *Communications of the ACM* 52 (11). ACM: 60–67.

Salazar, Spencer. 2017. "Sketching Sound: Gestural Interaction in Expressive Music Programming." PhD thesis, Stanford, CA, USA: Stanford University.

Salazar, Spencer, and Ge Wang. 2014. “miniAudicle for iPad: Touchscreen-Based Music Software Programming.” In *Proceedings of the International Computer Music Conference*.

Stowell, Dan, and Alex McLean. 2013. “Live Music-Making: A Rich Open Task Requires a Rich Open Interface.” In *Music and Human-Computer Interaction*, 139–52. Springer.

Tanaka, Atau. 2004. “Mobile Music Making.” In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 154–56. National University of Singapore.

Tillmann, Nikolai, Michal Moskal, Jonathan de Halleux, and Manuel Fahndrich. 2011. “TouchDevelop: Programming Cloud-Connected Mobile Devices via Touchscreen.” In *Proceedings of the 10th Sigplan Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward ’11)*, 49–60. Portland, Oregon, USA: ACM.

Trueman, Dan. 2007. “Why a Laptop Orchestra?” *Organised Sound* 12 (2). Cambridge University Press: 171–79.

Wang, Ge. 2014. “Ocarina: Designing the iPhone’s Magic Flute.” *Computer Music Journal* 38 (2).

Wang, Ge, Georg Essl, Jeff Smith, Spencer Salazar, Perry Cook, Rob Hamilton, Rebecca Fiebrink, et al. 2009. “Smule= Sonic Media: An Intersection of the Mobile, Musical, and Social.” In *Proceedings of the International Computer Music Conference*, 16–21.