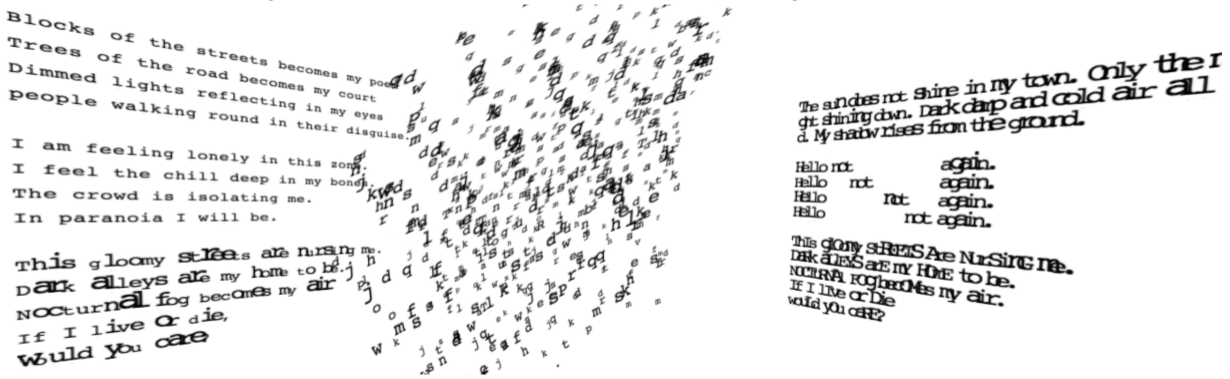
Web-Based Temporal Typography for Musical Expression and Performance

Sang Won Lee
Computer Science and Engineering
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
snaglee@umich.edu

Georg Essl
Electrical Engineering & Computer Science
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
gessl@umich.edu



Blocks of the streets becomes my pool
Trees of the road becomes my court
Dimmed lights reflecting in my eyes
People walking round in their disguise.

I am feeling lonely in this zone.
I feel the chill deep in my bone.
The crowd is isolating me.
In paranoia I will be.

This gloomy streets are nursing me.
Dark alleys are my home to be.
Nocturnal fog becomes my air
If I live or die,
Would you care

The sun does not shine in my town. Only the l
ght shining down. Dark damp and cold air all
d. My shadow rises from the ground.

Hello not again.
Hello not again.
Hello not again.
Hello not again.

This gloomy STREETS Are NurSinG me.
DarK alleyS aRE my HOME to be.
NOCTURNAL FOG becoMes my air.
If I live or Die
would you cARE?

Figure 1: *Live Writing : Gloomy Streets* using Web-Based Temporal Typography. The poem written by Pain

ABSTRACT

This paper introduces programmable text rendering that enables temporal typography in web browsers. Typing is seen not only as a dynamic but interactive process facilitating both scripted and live musical expression in various contexts such as audio-visual performance using keyboards and live coding visualization. With the programmable text animation , we turn plain text into a highly audiovisual medium and a musical interface which is visually expressive. We describe a concrete technical realization of the concept using Web Audio API, WebGL and GLSL shaders. We further show a number of examples that illustrate instances of the concept in various scenarios ranging from simple textual visualization to live coding environments. Lastly, we present an audiovisual music piece that involves live writing augmented by the visualization technique.

Author Keywords

Web Audio, Visualization, Live Coding, Live Writing

ACM Classification

H.5.5 [Information Interfaces and Presentation] Sound and Music Computing, H.5.2 [Information Interfaces and Presentation] User Interfaces — Screen design

1. INTRODUCTION

Textual performance interfaces and textual visualizations are common in computer music performance. Text is the basic building block of modern computer music software, for example, in live coding [9]. On the other hand, textual visualization is a powerful medium to convey the idea "literally." Turning plain text into a visual mean affords novel opportunities for musical expression and performance. Temporal typography - text that moves or changes over time - can be used to express music in visual, textual, and control manners. For musicians, this becomes an attractive channel by which they can communicate with the audience. In this paper, we introduce the foundation upon which the idea is built. We further elaborate the core ideas and describe the implementation structure. This is followed by a number of proof-of-concept examples and a music piece that shows the potential of this work in audiovisual performance music. Finally, we outline future research plans that are made possible by the system.

2. TEXTUAL VISUALIZATION AND LIVE CODING

Textual information is ubiquitous in digital music instruments, especially given the proliferation of music software tools and audiovisual performance. Text as an expressive medium for the purpose of better audience communication can be found in many forms, tag-cloud [16], audiovisual performances[4], audience participation music piece [10] and interactive installation [32]. Text is used as basic building blocks in most audio programming languages such as MAX [24], Pure Data [25], SuperCollider [21] and Chuck [33].

In live coding [9], in particular, the code text projection plays a significant role in communicating with the audience in live coding performance. This principle is captured well in the following statement of *TOPLAP*[1] manifesto: "Obscurantism is dangerous. Show us your screens." By screen sharing, the audience can draw possible connections between the algorithm (code text) and its outcome (generative

[1]http://www.toplap.org/

music). Live coders have made efforts to develop techniques to help audience understand live coding, for example, accessible variable naming conventions or writing code from a clean slate [5].

Instead of projecting the plain code text editor, live coding environments/language has been developed to have visual artifacts as part of the code outcome. These visuals are designed to help audience members associate them with generative sound. Some live coding music environments choose to overlay generative graphics on top of/behind/next to the code text and create visualization that can be mapped to the audio signal [14, 19, 27]. In contrast, there are live coding environments where visual information is embedded in the code text. *SchemeBricks* flashes code blocks when the instruction triggers certain sound events [23]. *ixi lang* utilizes programming syntax with which the spatial position of each symbol is interpreted as a rhythmic pattern of music [20]. *LOLC* visualizes code text from networked performers only at the moment of code execution (as opposed to revealing the whole process of typing), like chat messages in an instant messenger; the code generates visual patterns that represent tunes [19]. Swift and co-workers discuss the use of visual annotation one three aspect of live-coding: (1) the code text (*State of Code*), (2) the program state (*State of World*), and (3) the relationship between two [30]. In a recent update of *Gibber* [26], the authors added visualization directly on code text, which highlights a literal corresponding to a sound at the moment and changes text properties according to audiovisual output [28]. Indeed, expressive ways to visualize code text will be beneficial not only for more appealing visuals but also to help audience associate code text with music.

3. PROLIFERATION OF WEB BROWSER-BASED MUSIC APPLICATION

Modern web browsers have become a common platform for music making in many different contexts. A number of previous works of web-based music applications discuss the advantages of using web browser as a computer music platform. Common reasons are: (1) no installation required, (2) platform independent, (3) rich set of open-source libraries and (4) easy to distribute an application. Web browsers have been deployed in collaborative music making, networked musical systems [3, 6] and audience participation [13, 15, 34], to name a few usages. Recently, a Web UI toolkit has been introduced to support rapid prototyping of graphical user interfaces for music web apps [31].

The Web Audio API [29] accelerated emerging trends of web browser-based music applications. In [35], the viability of a web-browser as a computer music platform is evaluated in terms of timing and extensibility. There are many live coding environments built on the web browser using Web Audio API, including Gibber[26], Lich.js [22], wavepot.com [1], and livecodelab [11]. The Web Audio API also invited the development of high-level wrapper for better accessibility [8, 27]. Many of these web browser-based music applications will benefit from temporal typography that takes input from a variety of inputs.

4. DESIGN AND IMPLEMENTATION

The goal of this work is to realize a font-rendering system that supports expressive animations that can respond to both live and recorded music while facilitating the injection of a range of control mechanisms. The result is the support of expressive textual artistic expression. Its live properties is what differentiates it from common temporal typography techniques, i.e., kinetic typography [17]. Kinetic typography is usually created with a special software (such as Adobe After Effects) in a slow off-line authoring process and the outcome is fixed media (video) that does not change. Although this video outcome may create convincing visuals, it is not appropriate for live musicians who want real-time interactivity in text. Our work seeks to provide an on-line form of temporal typography. In addition, the approach is flexible in input and can take real-time input from any source such as sensors and live audio. WebGL takes advantages of accessibility, extensibility, and the growing popularity of web browsers while employing the power of OpenGL-based graphical rendering. The current implementation of temporal typography utilizes state-of-the-art web browser graphics libraries. It is written in javascript, WebGL [2] and Three.js [7] to draw text on HTML5 canvas objects.

One important technical factor in the implementation is the emphasis of the use, for animation, of a computer's graphics processing unit (GPU). The number of letters can be quite large for a certain context, potentially requiring substantial computational power. Making heavy use of the GPU for animations leaves the CPU available for audio processing and user input handling on the web browser. Implementing an animation algorithm in GPU can be done using OpenGL Shading Language (GLSL) shaders. GLSL is a high-level programming language syntactically close to C. There are two types of programmable shaders in GLSL: the *vertex shader* (operates on every vertex) and the *fragment shader* (runs per pixel). In this work, we update the position of vertices in the vertex shader, which can deform a letter in its shape, size, position, orientation, and rotation. The fragment shader is also used to change color and to manipulate texel (texture coordinate). For instance, the distorted letters of Step 4 from Figure-2 were created using sine wave to parametrically displace the texture coordinates. The main application written in javascript can communicate with GLSL shaders using uniform and attribute variables. For example, to change the font size based on the volume of audio, the WebGL program can set the value of a uniform variable, named `volume`, and then the variable is passed to the vertex shader to change the positions of the vertexes of letters. Attribute variable is used for the same purpose while attribute variables are defined per vertex.

There are four steps that constitute the final visualization (see Figure-2). The first step is to create a font texture atlas that allows for fast font rendering. The second step is to layout letters of the writing and generate the array of vertices. The texture associated with the array vertex is based on the texture coordinates of the font glyph from the texture atlas. For example, to draw "Hello, World" on screen, the second step would create a vertex array for 12 squares (which provide 48 vertices) and assign texture coordinates for each vertex. The third step is to generate data that is needed to animate the text (or part of the text) that will be passed to GLSL shaders. For example, if one wants to create a "Hello, World" that fades out over time, one need to create a uniform variable for the vertex shader and send timestamp values to the vertex shader using this variable. In the fourth step, the GLSL shaders create animation effects using whatever data is fed from the third step. For example, for fading out, the color of the fragment should be the product of the original color and a variable that goes from 1 to 0 over time based on the timestamp uniform variable passed from the javascript part. The first step determines the font-family and the resolution of the font that the visualization uses. The second step determines the placement of the text that the animation will be based on. The third and fourth steps are the placeholders

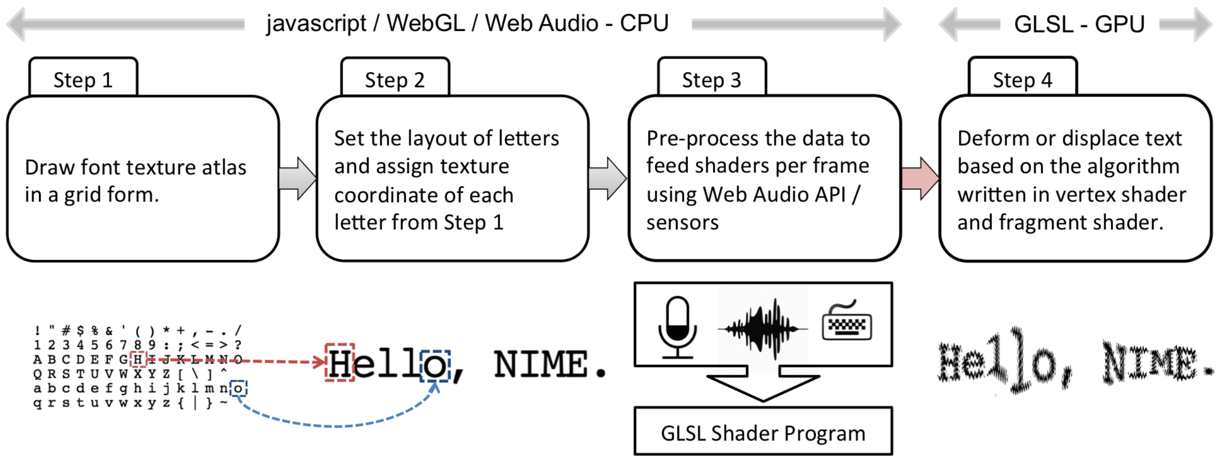

javascript / WebGL / Web Audio - CPU
GLSL - GPU
Step 1
Draw font texture atlas in a grid form.
Step 2
Set the layout of letters and assign texture coordinate of each letter from Step 1
Step 3
Pre-process the data to feed shaders per frame using Web Audio API / sensors
Step 4
Deform or displace text based on the algorithm written in vertex shader and fragment shader.
Hello, NIME.
GLSL Shader Program
Hello, NIME.

Figure 2: The procedure of programmable temporal typography

for the algorithmic visualization. Steps 1-3 of the algorithm of temporal typography are implemented in javascript while Step 4 is implemented in GLSL.

Interactive text in motion enabled by the algorithm above, does offer a wide range of expressivity. Algorithms written in javascript and GLSL shaders can access a variety of text properties, to name a few, font size, shape, position, color, texel (texture coordinate per fragments). On the other side, the algorithm can take input from audio signals, user input (keystrokes, mouse, camera, microphone, etc.), and pre-processed data. The algorithm in between will determine the mapping between these two sides and be able to create diverse audiovisual artifacts on top of the text.In conjunction with expressive text rendering, typing itself can be the site of the performance itself [12], leading to expressivity not only in sound but associated type. Typing has temporal characteristics which themselves can be made parameters in mappings. For example rapid typing can be visually and sonically differentiated from slow typing.

5. EXAMPLE APPLICATIONS

We present several examples of programmable temporal typography built on a web browser. These examples illustrate a range of possible application domains of temporal typography. All these examples are accessible at:

`http://www.sangwonlee.com/temporal-typography`

5.1 Example 1: Volume to Font Size

The first example is a straight-forward illustration of simple mapping. The z position (depth) of letters on screen is connected to the volume of the audio signal. Steps 1 and 2 are trivial tasks in preparing the font texture atlas and to place letters in the layout of usual writing. In Step 3, using Web Audio API, the javascript program plays an mp3 file and creates an audio analyzer node to retrieve an array of samples and to calculate the overall volume of the signal. Then the javascript program feeds shaders a uniform variable with the volume. For Step 4, the vertex shader will increase the z-position of letters (vertexes more specifically) in proportion to the volume value. Therefore, it creates an animation effect of increasing the font size in conjunction with the sound's volume. In point of fact, the letters are put closer to the camera so that they look larger than when in their original positions.

5.2 Example 2: Karaoke Lyric

Karaoke lyrics offer an example that explores audio signal in conjunction with timestamped scheduling data. The visualization displays the lyrics of the Beatles' song "Come Together" and the music plays along with the visualization. As the song progresses, the lyrics that correspond to the vocal at a particular moment are made to stand out. At the same time all of the text jitters based on the volume of the audio signal, using the same technique described in Example 1. The combination of two different visualizations enable richer expression of the relationship of lyrics to the music.

For the implementation of this example, the javascript program creates four types of input: overall volume, average frequency, the timestamp data per each word (or syllables), index of each letter as a vertex attribute. The first two variables are global values that all vertices share and are linked to two uniform variables in the shader. The last two variables are for the lyric progression, which is handled differently. First, timestamp data per word (or syllable) is annotated in advance. Second, while playing, the javascript will send two uniform variables - the starting-letter index and ending-letter index, specifying the range of words that should stand out at one moment. In the meantime when vertices are generated during Step 2, all vertices are given an index attribute of the associated letter. The four vertices that make one letter will have the same letter index. Lastly, vertex shader changes the z coordinates of all the vertices whose letter index falls into the two uniform variables passed from the script. As time passes, the draw function will update this starting index and ending index so that the next word will stand out at the proper time.

5.3 Example 3 - 5: Microphone Input

What the following three examples have in common is that they are based on input from a microphone; they differ in their visualization approach. Example 3 computes the fft spectrum of the audio signal coming from the microphone and changes the height of the letter. The low frequency sound will heighten the left side of the text while the high frequency sound will lengthen the right side of the text. Example 4 is similar, though that it makes the letter stroke convoluted with the weighted average frequency of the sound that the microphone is capturing. This technique is particularly interesting because it utilizes the fragment shader and manipulates texels (texture coordinates). The texel is adjusted so that a fragment color is set with the color of a point displaced from the original position. In Example 5 vertices of each letter are displaced based on the time domain data of the microphone audio signal. For all three example above, the javascript part passes an array of audio data (either in frequency domain or time domain) captured from the microphone to one of the shaders and displaces vertices' positions (or texture position).

5.4 Example 6: Live Coding Code Visualization

This example demonstrate how the *State of World* (or the program state) in live coding can be visualized on top of the code text using programmable temporal typography. This example simulates a live coding editor where textual visualization is connected to the audio signal that is associated with the code. The sound outcome is composed of two samples-drum and harp. The upper part of the text is the code that generates the drum sound and the lower part the harp. Looking at the whole code with two visualizations, one can easily distinguish the code text related to the drum sound.

We believe such textual visualization adds liveness to a live-coding performance. Typically, a live coder's typing is only related to certain events in the near future except the moment of the code run. However, with this kind of visualization, the audience will have a clearer idea of which code generates which sound regardless of which code the live coder edits. It also enhances the audience's anticipation. For example when the live coder modifies a parameter in the part where the drum sound is convoluted, the audience can expect that there will be a change in the drum pattern just by looking at where the live coder writes, as opposed to expecting that "something is going to happen." In addition, when a live coder writes a program from scratch, the audience will be able to quickly differentiate the code that is presently generating sound from code yet to be submitted so still unavailable in the program state.

The technique used in this example is the same as that used in Example 5, with the exception of there being two sets of data for two types of sound: two arrays of time domain audio data from two Web Audio analyser nodes playing two generative patterns. Each node sends an array of uniform float variables to the vertex shader. The vertex shader checks vertex attributes to identify which array (drum or harp) to use and displace the vertex y position based on the selected data. To realize multiple tracks of sound, multiple sets of data will simply be needed. However, it should be noted that optimization is needed due to the limit in the number of uniform/attribute variables that the shader can hold at once.

5.5 Example 7: Writing Interface

The last example demonstrates the potential of this work in an audiovisual performance where typing is the primary musical control. There are two types of data used in this example: microphone input and inter-keystroke interval. As a musician types a letter, it will appear on the screen as if it were a plain text editor with expressive visualization. The overall volume of the microphone input will change the size of the letter so that the typing sound level is visualized by the size of each letter. The average interval between keystrokes is used to determine the motion of the text and to change the intensity of the font color. Keystrokes are mapped to control visualization and to trigger certain musical events so that typing does generate, simultaneously, sound and visual with the textual content. This makes a unique audiovisual performance. The algorithm of sonification can be altered by typing a set of words selected in advance that will help a musician progress the music piece.

6. LIVE WRITING - GLOOMY STREETS.

As suggested in the previous section, the textual visualization introduces typing as a new performance interface that not only produces a music piece but also presents a piece of writing to audience. We presented an audiovisual music piece and strategically prefixed the title of the piece with "Live Writing", emphasizing the real-time writing shown to the audience (similar to what live coding does) and the live sound coming from the writing activity. While it is possible to completely improvise the writing as well (so-called live-poetry), the writing content of the piece is borrowed from a poem, "Gloomy Streets" by Pain-a multi-instrumental musician, composer, and lyricist. The music accompanied is composed by Lee with the poem in mind so that the composition will reinforce the content of the poem.

The piece is composed of three parts (or pages; see Figure-1). The mapping between inputs and sound is pre-programmed to alter whenever a performer presses a shortcut, writes a special letter, or the writing reaches a certain length. In the piece, interactivity of the piece mainly utilizes keystrokes to play a set of samples, to trigger the onset of synthesis algorithms and to change the mapping as the piece progress. Microphone input is used to capture the sound of typing. The amplified sound of typing helps convey the idea of live writing and introduces temporal dynamics of typing letters, words, and sentences. Towards the end of the piece, the algorithm incorporates more responsive and dynamic visualization that combines techniques used in the aforementioned examples with typing sonification and composed soundscape. The trackpad of the laptop is used to trigger synthesized tones and change the perspective of the camera so that the writing can be viewed from different perspectives, creating three-dimensional visuals of writing. The piece, which premiered at the University of Michigan's Performing Arts and Technology Showcase 2015, was well received. More detailed motivation and footage behind the piece is available at `www.sangwonlee.com/gloomy-streets`.

7. CONCLUSIONS AND DISCUSSION

In this paper, we introduced programmable text rendering that enables temporal typography on web browsers. We believe this is useful for textual performance interface and textual visualization, as it expands the range of musical expression and performance. The system is realized through the use of state-of-the-art Web Audio and Graphic libraries. The implementation of examples of distributed algorithms in GPU and CPU are given in detail. We outlined our future plan to explore the system in the context of a music performance, open source API, and a live coding environment.

We believe this will be useful in live-coding environments to help the audience better understand code text. We plan to build temporal typography enabled live coding editor that can be integrated with existing live coding languages on the web browser. The visualization of code text will be embedded in the editor and support textual visualization by default, as seen in Example 6.

In addition, typing that generates expressive animation can create a form of highly audiovisual performance that incorporates live writing, live poetry, sonification, and textual visualization. We plan to develop an accessible javascript library to support easy integration of programmable temporal typography in any web-based performance. We apply the concept of a control signal graph to the textual visualization so that one can connect any input to a property of text (e.g., connecting audio output to font size). Our goal is to develop the system in such a way that it can easily be incorporated in a wide range of web browser-based audiovisual performances for musical expression.

With a number of applications in practice, we believe there will emerge a set of interesting questions. Such questions may include: How will this text in motion enhance audience and musicians' engagement with the interface? How will it impact the reading comprehension of textual information and assist information visualization? Finally, we recently explored the program state visualization separate from code text in the context of collaborative live coding [18]. This raised the intriguing question of how the temporal typography changes the program state view, which can be now integrated into the code text for networked collaboration.

8. REFERENCES

[1] wavepot. `http://www.wavepot.com`.

[2] Webgl-opengles 2.0 for the web. `http://www.khronos.org/webgl`.

[3] A. Barbosa. Public sound objects: a shared environment for networked music practice on the web. *Organised Sound*, 10(03):233–242, 2005.

[4] J. Blonk and G. U. Levin. *Ars Electronica Festival*. Linz, Austria, 2005.

[5] A. R. Brown and A. C. Sorensen. aa-cell in practice: An approach to musical live coding. In *Proceedings of the International Computer Music Conference*. International Computer Music Association, 2007.

[6] P. Burk. Jammin'on the web-a new client/server architecture for multi-user musical performance. In *ICMC 2000*, 2000.

[7] R. Cabello. Three. js. *URL: https://github. com/mrdoob/three. js*, 2010.

[8] H. Choi and J. Berger. Waax: Web audio api extension. In *Proceedings of New Interfaces for Musical Expression*, pages 499–502, 2013.

[9] N. Collins, A. McLean, J. Rohrhuber, and A. Ward. Live coding in laptop performance. *Organised Sound*, 8(03):321–330, 2003.

[10] L. Dahl, J. Herrera, and C. Wilkerson. Tweetdreams: Making music with the audience and the world using real-time twitter data. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, 2011.

[11] D. Della Casa and G. John. Livecodelab 2.0 and its language livecodelang. In *Proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design*, pages 1–8. ACM, 2014.

[12] R. Fiebrink, G. Wang, and P. R. Cook. Don't forget the laptop: using native input capabilities for expressive musical control. In *Proceedings of the 7th international conference on New interfaces for musical expression*, 2007.

[13] J. Freeman. Web-based collaboration, live musical performance and open-form scores. *International Journal of Performance Arts and Digital Media*, 6(2):149–170, 2010.

[14] D. Griffiths. Fluxus. In A. Blackwell, A. McLean, J. Noble, and J. Rohrhuber, editors, *Collaboration and learning through live coding, Report from Dagstuhl Seminar 13382*, pages 149–150. 2013.

[15] A. Hindle. Swarmed: Captive portals, mobile devices, and audience participation in multi-user music performance. In *Proceedings of the 13th International Conference on New Interfaces for Musical Expression*, pages 174–179, 2013.

[16] O. Kaser and D. Lemire. Tag-cloud drawing: Algorithms for cloud visualization. *arXiv preprint cs/0703109*, 2007.

[17] J. C. Lee, J. Forlizzi, and S. E. Hudson. The kinetic typography engine: an extensible system for animating expressive text. In *Proceedings of the 15th annual ACM symposium on User interface software and technology*, 2002.

[18] S. W. Lee and G. Essl. Communication, control, and state sharing in collaborative live coding. In *Proceedings of New Interfaces for Musical Expression (NIME)*, London, United Kingdom, 2014.

[19] S. W. Lee and J. Freeman. Real-time music notation in mixed laptop–acoustic ensembles. *Computer Music Journal*, 37(4):24–36, 2013.

[20] T. Magnusson. ixi lang: a supercollider parasite for live coding. In *Proceedings of International Computer Music Conference*, pages 503–506. University of Huddersfield, 2011.

[21] J. McCartney. Rethinking the computer music language: Supercollider. *Computer Music Journal*, 26(4):61–68, 2002.

[22] C. McKinney. Quick live coding collaboration in the web browser. In *Proceedings of New Interfaces for Musical Expression*, London, U.K., 2014.

[23] A. McLean, D. Griffiths, N. Collins, and G. Wiggins. Visualisation of live code. *Proceedings of Electronic Visualisation and the Arts*, 2010.

[24] M. Puckette. Combining event and signal processing in the max graphical programming environment. *Computer music journal*, pages 68–77, 1991.

[25] M. Puckette et al. Pure data: another integrated computer music environment. *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41, 1996.

[26] C. Roberts and J. Kuchera-Morin. Gibber: Live coding audio in the browser. In *Proceedings of the International Computer Music Conference (ICMC)*, Ljubljana, Slovenia, 2012.

[27] C. Roberts, G. Wakefield, and M. Wright. The web browser as synthesizer and interface. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 313–318, 2013.

[28] C. Roberts, M. Wright, and J. Kuchera-Morin. Beyond editing: Extended interaction with textual code fragments. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, 2015.

[29] C. Rogers. Web audio api. 2012.

[30] B. Swift, A. C. Sorensen, H. Gardner, and J. Hosking. Visual code annotations for cyberphysical programming. In *1st International Workshop on Live Programming (LIVE)*. IEEE, 2013.

[31] B. Taylor, J. Allison, W. Conlin, Y. Oh, and D. Holmes. Simplified expressive mobile development with nexusui, nexusup and nexusdrop. In *Proceedings of the New Interfaces for Musical Expression conference*, 2014.

[32] C. Utterback and R. Achituv. Text rain. *SIGGRAPH Electronic Art and Animation Catalog*, 78, 1999.

[33] G. Wang, P. R. Cook, et al. Chuck: A concurrent, on-the-fly audio programming language. In *Proceedings of International Computer Music Conference*, pages 219–226, 2003.

[34] N. Weitzner, J. Freeman, Y.-L. Chen, and S. Garrett. massmobile: towards a flexible framework for large-scale participatory collaborations in live performances. *Organised Sound*, 18(01):30–42, 2013.

[35] L. Wyse and S. Subramanian. The viability of the web browser as a computer music platform. *Computer Music Journal*, 37(4):10–23, 2013.