EarSketch: Teaching computational music remixing in an online Web Audio based learning environment

Anand Mahadevan
School Of Music
Georgia Institute of Technology
Atlanta, GA 30332
amahadevan32@gatech.edu

Jason Freeman
School Of Music
Georgia Institute of Technology
Atlanta, GA 30332
jason.freeman@gatech.edu

Brian Magerko
Digital Media Program
Georgia Institute of Technology
Atlanta, GA 30332
magerko@gatech.edu

Juan Carlos Martinez
School Of Music
Georgia Institute of Technology
Atlanta, GA 30332
jcm7@gatech.edu

ABSTRACT

EarSketch is a novel approach to teaching computer science concepts via algorithmic music composition and remixing in the context of a digital audio workstation paradigm. This project includes a Python/Javascript coding environment, a digital audio workstation view, an audio loop browser, a social sharing site and an integrated curriculum. EarSketch is aimed at satisfying both artistic and pedagogical goals of introductory courses in computer music and computer science. This integrated platform has proven particularly effective at engaging culturally and economically diverse students in computing through music creation. EarSketch makes use of the Web Audio API as its primary audio engine for playback, effects processing and offline rendering of audio data. This paper explores the technical framework of EarSketch in greater detail and discusses the opportunities and challenges associated with using the Web Audio API to realize the project.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education*; H.5.3 [**Information Interfaces and Presentation**]: Web based interaction

Keywords

Remixing, Music composition, CS education, Web audio, Social media sharing.

Copyright © 2015 for the individual papers by the papers' authors. Copying permitted for private and academic purposes. This volume is published and copyrighted by its editors.
Web Audio Conference '2015 Iracam and Mozilla, Paris France
.

1. INTRODUCTION

EarSketch seeks to increase and broaden participation in computing by creating an engaging and culturally relevant learning experience using a STEAM (science, technology, engineering, arts and mathematics) approach [14]. Students write code (in Python or Javascript) to creatively and algorithmically manipulate audio samples from a loop library while learning computing fundamentals such as loops, lists, conditions and functions. EarSketch offers a tightly integrated creative and pedagogical environment that captivates students by making abstract computing concepts relevant in a context that borrows from the paradigm of digital audio workstations and music production while remaining readily accessible to those without any prior experience with music or music technology [12].

Other recent projects have also tried to engage students in computing by connecting coding to artistic and creative contexts. Alice, for example, is a 3D programming environment that allows students to create animated stories through code [8] while learning computer science principles such as object-oriented programming. Scratch, on the other hand, teaches programming through the creation and sharing of games, animations and simulations [15]. EarSketch is inspired by efforts such as these, but in addition to focusing on a different artistic domain (music), it also focuses on authenticity [9] in its design by ensuring its approach is both industry and culturally relevant to students. Its API and visual design borrow heavily from pervasive integrated development environments such as Eclipse and digital audio workstations such as Live, it teaches students to code in some of the most popular programming languages in the world (Python and Javascript), and it enables students to create music in popular styles such as hip hop and dub step while using loops created by music industry veterans [4].

Since development of EarSketch began in 2011, it has been accessed by over 10000 users at over a dozen schools, summer camps, academic courses, and other educational programs ranging from elementary school through college. It has also been incorporated into a music technology MOOC [3] taught by one of the authors that has reached over 35,000 students.

A 2013 pilot study at an Atlanta-area high school demon-

strated the success of EarSketch in improving student engagement and content knowledge in computing. Approximately 100 student participants completed a pre and post content knowledge assessment and a retrospective pre-post student engagement survey, and a subset of the students also participated in a focus group. The results showed significant increases in student content knowledge and engagement, regardless of ethnicity or gender, and showed that for many engagement constructs, female students increased significantly more than male students from pre to post [4].

The primary focus of this paper is on the technical implementation of the EarSketch learning environment. We discuss the original technical framework for EarSketch, our motivations for reimplementing EarSketch as a browser-based application using Web Audio, the software architecture, and various constraints, limitations, and development challenges.

2. TECHNICAL DESIGN AND IMPLEMENTATION

2.1 Motivation for web based version

Initially, EarSketch was built on top of Reaper, a commercial but inexpensive DAW (digital audio workstation) similar to those used in professional studios. Reaper supports a wide range of multi track audio recording and editing features and includes a large library of effects. It requires minimal system resources (CPU, RAM, disk space), which makes it practical for school computer labs that often use outdated technology. Most importantly to us, Reaper is also extremely extensible: its Python API, ReaScript, provides low-level access to its internal data structures and functions [12].

Using Reaper, we were able to quickly develop a version of EarSketch to use in pilot studies. However, this approach created some design problems and logistical challenges. EarSketch existed as a collection of linked but separate tools: code editor + digital audio workstation + online curriculum + audio library + social sharing site. These elements were loosely connected but operated as separate desktop applications and web sites. This created a rather convoluted workflow for students to follow as it involved too many discrete components that required constant context switching between one another.

This version of EarSketch also created installation and setup challenges: school IT labs had to seek separate approval to install each component application, and our installer was often intercepted by security software. Finally, EarSketch was dependent on the Reaper DAW, which despite being very cheap, was still a financial barrier to some cash-strapped schools.

Based on these experiences, in 2013 we decided to implement a new version of EarSketch entirely in the web browser, EarSketch 2.0. A web-based paradigm addresses all the design and logistical challenges of our previous version and offers the opportunity to create a truly integrated learning environment for students. The rapid development of the Web Audio API in recent years made it possible for us to consider a web-based version of EarSketch that just two years earlier had seemed impractical.

2.2 EarSketch API design

In both versions of EarSketch, the EarSketch API provides functions that closely mimic popular operations in a DAW workflow such as placing audio clips on a multi-track timeline, adding effects and effect automation breakpoints, and step-sequencing rhythms. Our customized API abstracts a number of low-level intricacies and function calls.

2.2.1 *Placing audio on the timeline*

Figure 2 illustrates a basic EarSketch script that initializes the DAW and places an audio clip onto a track. The code is written in Python and begins by importing the EarSketch module for Python. (This step is not required while coding in Javascript.) The project tempo is then set to 120 bpm using the `setTempo()` function.

Next, an audio file is placed on the timeline using the `fitMedia()` function. Audio files are specified by constants. EarSketch comes bundled with over 2000 audio loops in a variety of genres like hip-hop, soul, rock, techno and house that we commissioned from Richard Devine, an experimental electronic musician and sound designer, and Young Guru, an audio engineer and DJ best known for his long-running collaboration with Jay Z. Users can also upload their own audio files to add to their EarSketch sound library. Audio content can be previewed in a sound browser (Figure 1) and the corresponding constant can be pasted into the code editor directly from the browser. EarSketch automatically time stretches each audio clip to match the specified tempo and loops it as necessary to fill the duration specified. The remaining arguments to `fitMedia()` specify a track number, start measure, and end measure. Floating point arguments specify midpoints in measures.

The script in Figure 2 also demonstrates how to place an effect on a particular track. In this case, the GAIN parameter of the VOLUME effect is being automated from -60db to 0db between the first and fifth measure.

Figure 3 shows the result of running the script in the DAW view. In the DAW view, users can play the music they created in code and perform basic transport and mixing operations. By design, though, they cannot edit the multi-track audio and effects content directly: they must edit their code to change the music. Users can export their project to Reaper if they wish to edit it further.

2.2.2 *Beat sequencing*

The script in Figure 2 does more than just place audio on a timeline and apply effects. EarSketch's `makebeat()` function allows users to create rhythmic beats and phrases by using strings to piece together contents of different audio files at a 16th note resolution. Borrowing from Thor Magnusson's ixi lang [10] and Freeman and Van Troyer's LOLC [5], our API uses a string representation to sequence individual sixteenth notes over a full measure. The notation is fairly straightforward; a number represents a single audio file or an index in a list of audio files, a "+" sign extends the duration of the preceding sound by a sixteenth note, and a "–" sign indicates a rest. The resultant pattern is depicted in Track 2 of Figure 3. Through beat sequencing, EarSketch teaches students computer science concepts such as strings, string operations, lists, and list indices.

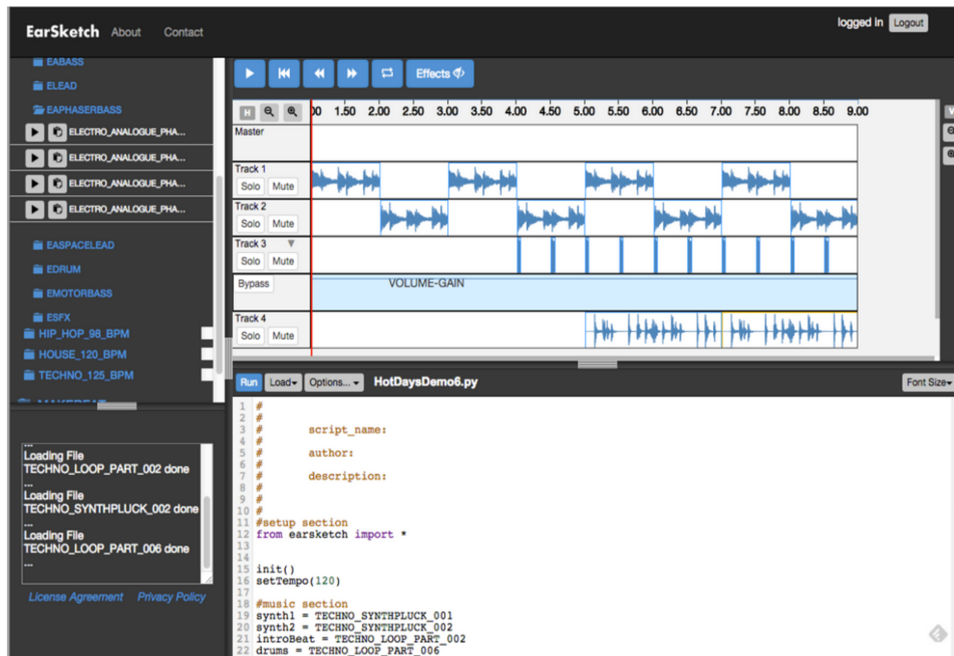

Figure 1: The EarSketch Interface

2.2.3 Analysis

EarSketch also provides functionality for analyzing the content of an audio clip or a track or a segment of a clip or track, computing features such as spectral centroid and RMS amplitude. The calculated information can be effectively used in a pedagogical and compositional context. For example, students can create a noise gate by arithmetically comparing RMS amplitude values of various clips (or segments of clips) with some threshold and accordingly silencing those segments that are above that threshold. Through analysis, we also teach students concepts such as conditionals, sorting, and mapping. The implementation of most of our analysis features makes use of the Fast Fourier Transform (FFT). The FFT routines we use are part of the dsp.js library [2].

Detailed specifications of the EarSketch API can be found on our curriculum website (`http://earsketch.gatech.edu/category/learning/reference/earsketch-api`).

Run | Load | Options... | **WAC_demo.py**

```
1  from earsketch import *
2
3  init()
4  setTempo(120)
5
6  # melody
7  fitMedia(HOUSE_ROADS_PIANO_007, 1, 1, 5)
8  # effects
9  setEffect(1, VOLUME, GAIN, -60, 1, 0, 5)
10 setEffect(1, DELAY, DELAY_TIME, 250)
11
12 # Sequence a beat
13 beatElement = OS_KICK03
14 beatString = "0+++0+++0+0+0-0-"
15 for index in range(1,5):
16       makeBeat(beatElement, 2, index, beatString)
17
18 finish()
```

Figure 2: Example of a basic EarSketch script

2.3 Implementation

2.3.1 Python / Javascript editor

EarSketch's code editor uses CodeMirror [7], a text editor implemented specifically for the browser with specialized language modes and features such as syntax highlighting and auto-completion.

When users code in Javascript, the code is simply handed over to the backend, also implemented in Javascript, for evaluation. Python code requires an intermediate agent to interpret the Python code into Javascript. For this purpose we use Skulpt, a Javascript, browser-based implementation of Python that runs completely on the client side [6]. Skulpt's extensibility enabled us to easily add support for the EarSketch API functions.

2.3.2 Client-Server model

Our approach to distributing tasks across client and server is influenced by our desire to minimize required server-side resources, optimize performance in low-bandwidth school settings, and minimize delay between executing code and hearing the resulting music. Following from this, most of the computationally intensive and time sensitive tasks – audio playback, effects processing and DAW view rendering – occur on the client machine. The server is responsible for hosting all of the audio loops and samples and for performing operations on those samples, such as time stretching samples to match project tempo, that are currently difficult to do efficiently with the Web Audio API. On the server side, we use SoX [13] to time stretch and perform format conversions. A Tomcat servlet manages audio sample requests, maintaining a fixed thread pool of SoX instances, queuing requests, and caching converted files to improve efficiency. The client also caches requested audio samples to avoid redundant transfers. To improve performance in low-bandwidth settings, the server can also transfer audio files

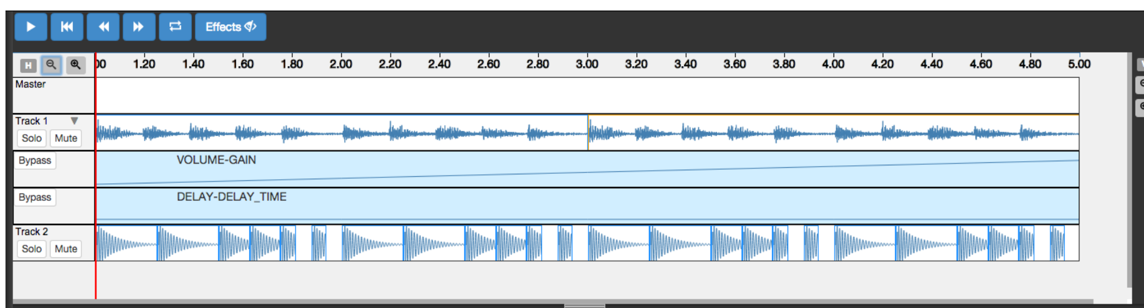

Figure 3: DAW visualization of example script from Figure 2

in a lossy format (Ogg Vorbis) instead of as WAV files.

The server also handles operations such as managing user accounts, facilitating sharing projects, and saving and loading scripts to and from the server. These are handled by a Tomcat servlet with a MySQL database. The client side is managed by Angular JS, a popular MVC Javascript framework [1]. The client and server communicate via a RESTful API.

2.3.3 *Playback engine*

In this section, we describe the process of audio playback on the client side using the Web Audio API. Once the server dispatches the relevant audio clips to the client, each is decoded, and its buffer and relevant metadata (such as timeline location) is stored in EarSketch-specific data structures. Since the loading process is asynchronous, playback waits until all the sound buffers are filled with the decoded PCM data.

Next, playback is scheduled. The scheduling of playback is relative because playback begins only once the user presses the DAW's play button. The `start()` and `stop()` APIs provided by web audio make all of this possible as we can specify the time domain offsets as parameters to these functions [16]. This property is also exploited to facilitate scrubbing around the timeline and pausing and resuming playback. Since the web audio API runs on its own thread, its scheduler is much more precise than traditional Javascript timer routines like `setTimeout()`. If a particular audio clip is scheduled to play at the same time as part of different tracks, we still need only one audio buffer with the contents of that clip, since different buffer sources can read the same audio buffer. Hence, this asserts the fact that duplicate audio buffers are redundant despite the same audio file being overlapped in time [14].

2.3.4 *Effects and Automation*

The Web Audio API includes an assortment of AudioNodes that make it straightforward to rack up custom effects.

EarSketch uses nodes such as GainNode, DelayNode, WaveshaperNode, and BiquadFilterNode to realize its 15 effects (refer to `http://earsketch.gatech.edu/category/learning/reference/every-effect-explained` for a complete list of effects). Web Audio AudioNodes have specific configurable parameters to modify how the node processes incoming audio. For example, the DelayNode has an attribute called `delayTime` that specifies by how milliseconds the input audio is delayed.

Many effects can be created by routing nodes, much like other patching environments for audio synthesis like MAX / MSP and Pure Data. Chris Wilson's web audio playground application [17] demonstrates how audio effects can be built from scratch using similar techniques. Once the nodes are connected, the entire graph can be encapsulated into a custom Javascript object. EarSketch takes advantage of such abstractions in order to chain multiple effects on a single track. The custom effects that are built are simply wired together in a linked list with the head being the source audio and the tail being the Audio Context's destination. To implement effect automation envelopes, we leverage Web Audio API's ability to schedule changes to audio parameters residing within AudioNodes. Not only can we schedule the values of AudioParams at a particular instant of time, but we can also customize the rate at which the parameter changes [16]. For the sake of simplicity, EarSketch uses a linear ramp to interpolate between automation points on the timeline. However, more complex automation curves can easily be implemented. The routing graph template that EarSketch employs is slightly more complicated, as the abstracted effect nodes need to support dry/wet mixes as well as effect bypass capability. Figure 4 is the resultant node graph of a delay node.

We have explained how EarSketch uses encapsulated node graphs to perform effect automation. For more complicated cases like LFO (low frequency oscillator) based effects, automation has to be performed on a set of basic AudioParams. Consider a tremolo effect for example - the amount of modulation caused by the LFO is actually controlled by a gain node, though that might not seem intuitive at first. The gain node shapes the amplitude of the LFO and as a result, controls the depth of modulation. The same logic can be carried forward for effects like Phaser, Chorus, Flanger and so on. In a nutshell, it can be easily understood from the previous discussions that the Web Audio API can offer us significant low-level parameter control by simply tweaking a few basic parameters of AudioNodes.

Some effects, however, demand different approaches. Let us consider pitchshift as an example. It cannot be modeled using Web Audio's playbackRate parameter as it entails an inevitable and undesirable change in tempo as well. EarSketch initially made use of the ScriptProcessorNode to tackle this issue. The script processor gives us access to a frame of buffer data that can directly be manipulated using native JavaScript code [14]. Control is transferred to a callback function (audioprocess event) where we can process a frame of data. The frequency of the call back depends on the size of the buffer data to be manipulated. During the audioprocess event, the buffer is sent to another JavaScript

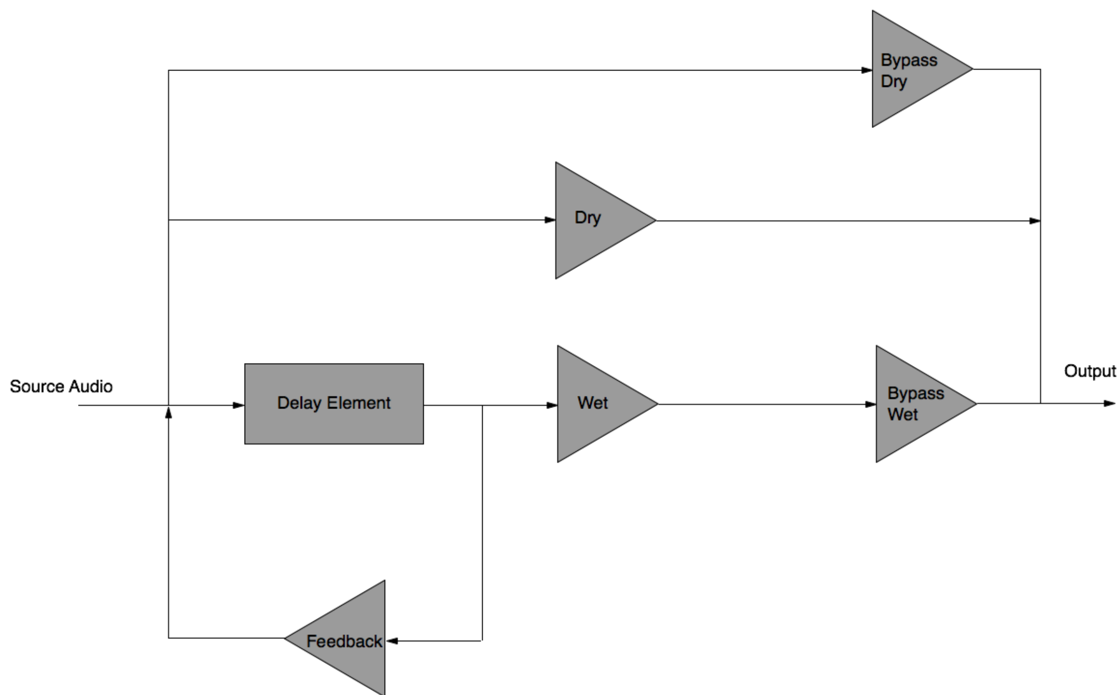

Figure 4: EarSketch delay effect's routing graph (Triangles represent gain nodes).

based phase vocoder module that performs frequency domain transformation, harmonic content retention and resampling. This is a computationally intensive process and ends up hogging the Javascript thread. The outcome is not pleasing when the effect is applied to multiple tracks in real time and results in glitchy audio playback.

Due to these limitations, we are considering different approaches to these script processor dependent effects. Our current solution for pitchshift is prior rendering of pitch-shifted clips by SoX on the server. This offline operation is tantamount to requesting a new audio loop from the server and therefore works against our client-server distribution goals (see section 2.3.2) that seek to minimize server-side resource consumption. It does, however, offer a solution that does not compromise audio quality and serves as an interim implementation until we identify an appropriate client-side solution.

2.3.5 Offline rendering

EarSketch also supports offline rendering of audio created by the user. The Web Audio API's OfflineAudioContext is employed to perform this task. Once the audio is rendered, we have access to the PCM buffer data, which we use to export to a WAV file that can be downloaded by the user to his or her desktop. Since the offline context can encode the audio faster than real time, an entire song can often be completed in just a few seconds. It is worth mentioning here that scriptProcessor based effects (see section 2.3.4) are not audible when the audio file is rendered simply because those callbacks occur in real time.

2.4 Export to Reaper

Since the EarSketch DAW is view-only, with simple transport and mixing controls but no way to edit audio and effects content, we offer users the ability to export their EarSketch projects to the Reaper DAW for further editing or for recording additional tracks. We collate the information from all our internal data structures containing pertinent information about the project tempo, audio buffers and their placement on tracks, effects, automation points, etc. and convert them to a Reaper project file (.rpp), a format that loosely resembles XML. The EarSketch server generates a ZIP file with the Reaper project and all the source audio files referenced by the project.

3. BETA TESTING, USER FEEDBACK, AND FUTURE WORK

We released an initial public beta version of EarSketch online in March 2014. Since then, it has been used in academic courses, summer camps, teacher training workshops and a massive open online course (MOOC). We have received informal, largely positive feedback from students and teachers about the user experience, especially as compared to the desktop version of EarSketch. We have also received numerous suggestions and feature requests, especially from teachers, which we have incorporated into the current version of the web site. For example, we added features such as font size options to help teachers use EarSketch on video projections during classroom lectures. We also unified the code editor, sound browser, console and DAW view into a single view, each with resizable dimensions. (We had previously had some of these components on separate tabs.) This emulates the experience of popular IDEs like Eclipse and Visual Basic and also DAWs like Ableton Live while avoiding constant tab switching and facilitating more direct

associations of music with code.

In November and December 2014, a member of the EarSketch research team with expertise in usability testing conducted a more formal usability study with high school students using EarSketch. Preliminary data has yielded additional suggestions, including revamping the sound browser interface to enable easier navigation and search and enabling personalization of the the UI theme and color schemes to help students feel more invested in the interface.

In the coming year, we are working to integrate more components of EarSketch into the new single-window interface design. We plan to embed the EarSketch curriculum, teacher training materials, and social media features directly into the interface instead of maintaining them on distinct web sites.

In order to maintain the robustness of our software, our team is also developing a unit testing framework and improved error reporting and logging mechanisms. We are also exploring ways to bring the EarSketch approach to computational music and computer science education to additional programming contexts, including a visual programming environment built on Blockly [11] and a mobile-friendly programming environment. We hope these additional programming contexts will help bring EarSketch to younger students and to more informal learning environments.

4. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. CNS #1138469 and DRL #1417835. It is also supported by the Scott Hudgens Family Foundation. Many thanks to the entire EarSketch project team (`http://earsketch.gatech.edu/personnel`).

EarSketch is freely available at `http://earsketch.gatech.edu`.

5. REFERENCES

[1] G. Brat Tech LLC and community. angular/angular.js, 2009.

[2] C. Brook. corbanbrook/dsp.js https://github.com/corbanbrook/dsp.js, 2010.

[3] J. Freeman. Survey of music technology https://www.coursera.org/course/musictech, October 2014.

[4] J. Freeman, B. Magerko, J. Permar, C. Summers, E. Fruchter, M. Reilly, and T. McKlin. Engaging underrepresented groups in high school introductory computing through computational remixing with EarSketch. In *SIGCSE 2014 - Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, pages 85–90. Association for Computing Machinery, 2014. 85.

[5] J. Freeman and A. Van Troyer. Collaborative textual improvisation in a laptop ensemble. *Computer Music Journal*, (2):8, 2011.

[6] S. Graham. https://github.com/skulpt/skulpt, 2010.

[7] M. Haverbeke. codemirror/CodeMirror https://github.com/codemirror/codemirror, 2007.

[8] C. Kelleher, R. Pausch, and S. Kiesler. Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1455–1464, New York, NY, USA, 2007. ACM.

[9] H.-S. Lee and N. Butler. Making authentic science accessible to students. *International Journal of Science Education*, 25(8):923–948, Aug. 2003.

[10] T. Magnusson. The IXI lang: A SuperCollider parasite for live coding. International Computer Music Conference, pages 503–506. San Francisco, Huddersfield, International Computer Music Association, Centre for Research in New Music University of Huddersfield, 2011.

[11] A. Marron, G. Weiss, and G. Wiener. A decentralized approach for programming interactive applications with JavaScript and blockly. In *Proceedings of the 2Nd Edition on Programming Systems, Languages and Applications Based on Actors, Agents, and Decentralized Control Abstractions*, AGERE! '12, pages 59–70, New York, NY, USA, 2012. ACM.

[12] S. McCoid, J. Freeman, B. Magerko, C. Michaud, T. Jenkins, T. Mcklin, and H. Kan. EarSketch: An integrated approach to teaching introductory computer music. *Organised Sound*, 18(2):146–160, Aug. 2013.

[13] L. Norskog. SoX http://sox.sourceforge.net/sox.html, 1991.

[14] N. Park and Y. Ko. Computer education's teaching-learning methods using educational programming language based on STEAM education. In J. J. Park, A. Zomaya, S.-S. Yeo, and S. Sahni, editors, *Network and Parallel Computing*, number 7513 in Lecture Notes in Computer Science, pages 320–327. Springer Berlin Heidelberg, Jan. 2012.

[15] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: Programming for all. *Commun. ACM*, 52(11):60–67, Nov. 2009.

[16] B. Smus. *Web Audio API*. "O'Reilly Media, Inc.", Mar. 2013.

[17] C. Wilson. cwilso/WebAudio https://github.com/cwilso/webaudio, 2012.