Time Stretching & Pitch Shifting with Web Audio API: Where are we at?

ABSTRACT

Audio time stretching and pitch shifting are operations that all major commercial and/or open source Digital Audio Workstations, DJ Mixing Software and Live Coding Suites offer. Those operations allow users to change tempo while maintaining the pitch and vice-versa. Unfortunately, there are few (and experimental) client-side Javascript implementations of those two operations.

In this paper, we review the current state of art for client-side implementations of time stretching and pitch shifting, their limitations, and describe two new implementations for two well-known algorithms: (1) Phase Vocoder with Identity Phase Lock and (2) a modified version of Overlap & Add.

Additionally, we discuss some issues related to Web Audio API (WAA) and Frequency-based audio process, regarding latency and audio quality in pitch shifting and hoping to raise awareness over some potential (and welcome) changes in WAA.

1. INTRODUCTION

Time stretching and pitch shifting are two operations widely available in commercial and open source musical applications like Ableton Live[1], Traktor Pro[2] and Ardour[3]. Currently, creative frameworks implemented in JavaScript and Web Audio API (WAA) like Flocking [**?**] and Tone.js [**?**] only offer time stretching through re-sampling of audio buffers, changing both the duration (i.e.: tempo) and pitch at the same time. In digital audio workstations (DAW) like EarSketch [**?**], time stretching and pitch shifting is performed server-side, with SoX[4], making real-time interaction impossible.

This paper presents an overview of the current implementations for both algorithms using JavaScript in the browser environment and two new implementations. Additionally, we discuss the impact of certain decisions in WAA design, and philosophy, regarding the absence of frequency-based operators, like Fast Fourier Transform (FFT), cause an impact in performance.

For both implementations, we had the following non-functional requirements:

- **Constant memory usage**, with fixed size circular arrays, in order to minimize the impact of the garbage collector, by maintaining a constant memory profile.
- **Intuitive API and documentation** because time stretching, for the current Web Audio API community, seems to be a difficult subject.
- **Minimize assumptions** regarding sample rate, number of channels and the type of audio (percussive or harmonic) being processed.

The outline of the paper is the following. First, we review the basic theory for time, and frequency, time stretch and pitch shifting algorithms. Then, we describe the existing Javascript implementations of such algorithms. After that, we present our two new implementations for two well know algorithms: (1) Phase Vocoder with Identity Phase Locking[**?**] and (2) a modified Overlap & Add (OLA) algorithm. Finally, we discuss the trade-offs between existing implementations and problems with WAA regarding these two operations.

2. THEORETICAL BACKGROUND

For both tasks, there are algorithms that work in time domain, like Overlap and Add (OLA) [], Waveform Similarity based OLA (WSOLA) [**?**] and delay line modulation [**?**], and others that work in the frequency domain, like the Phase Vocoder [**?**] and Spectral Modelling [**?**]. In this section, we give an overview of three popular methods: OLA, WSOLA and Phase Vocoder.

2.1 OLA

Algorithms in both time and frequency domains, for Time Stretching and Pitch Shifting, are more sophisticated versions of OLA. There are three main steps for OLA:

1. Chop the input signal into several (analysis) frames of size N, each overlapping with the previous one in H_a (i.e.: the analysis hop size) samples.
2. Apply a window to each analysis frame.

[1]https://www.ableton.com/
[2]http://bit.ly/1izsxHy
[3]http://ardour.org/
[4]http://sox.sourceforge.net/

Web Audio Conference WAC-2016, April 4–6, 2016, Atlanta, USA

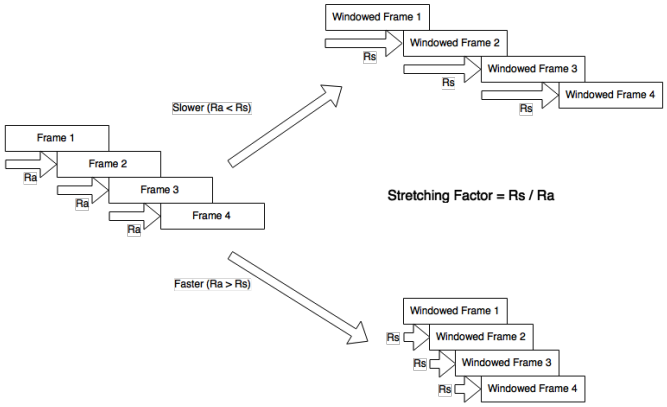

Figure 1: Example showing what happens, regarding hop sizes, for slowdowns and speed-ups of the audio.

3. Add the (synthesis) frame to the output, by overlapping it with H_s (i.e.: the synthesis hop size) samples of the "tail" of the output signal.

So, the main idea of OLA is to retrieve frames from the input signal, each one overlapping with consecutive input frames in H_a samples, and overlapping each windowed input frame with the output signal, in H_s samples. The relation between both overlap/hop sizes determines the stretching factor:

$$\alpha = \frac{H_s}{H_a}$$

Usually, one of the hop sizes is fixed while the other is a function of α. We should note that α can be change at each new frame. OLA does not preserve phase relations between consecutive frames and, as such, there are noticeable artefacts in the output signal: modulation of harmonic structures (e.g.: human voice) and reverberation. The temporal complexity of OLA is $O(N)$. In figure **??**, it can be seen what happens to both H_a and H_s when changing the α. To perform pitch shifting with OLA, one could couple a re-sampler to the OLA time stretcher. In order to allow simultaneous time stretching and pitch shifting, let β_t be the new desired pitch and R_t the current sampling rate used in the re-sampler. Then, the new stretching factor α_{t+1} and the new sample rate R_{t+1} are defined as

$$\alpha_{t+1} = \frac{\alpha_t}{R_t}$$

$$R_{t+1} = R_t * \beta_{t+1}$$

2.2 WSOLA

First introduced in [**?**], WSOLA applies a delay $\delta \in [-\delta_{max} : \delta_{max}]$ to each analysis frame, such that the waveforms of two overlapping synthesis frames are as similar as possible in the overlapping regions[5]. The delay can be obtained by calculating the cross-correlation between the overlapping regions of each synthesis sample. This algorithm removes the reverberation and modulation but, for transient rich sounds (percussion, guitar riffs), there might be some missing transients and stuttering (i.e.: repeated transients). Regarding the temporal complexity of WSOLA, if the cross-correlation is implemented with FFT Convolution [**?**], we get $O(Nlog_2N)$. Otherwise, if it is a "naive" implementation of the convolution, we get $O(N^2)$. To perform pitch shifting, one can use the same method described for OLA.

[5]WSOLA is equivalent to OLA when $\delta_{max} = 0$.

2.3 Phase Vocoder

Phase Vocoder is a well documented [**?**, **?**, **?**, **?**, **?**, **?**, **?**] algorithm used for Time Stretching [**?**, **?**], Pitch Shifting [**?**] and other audio effects like robotization [**?**]. In general, it has higher computational costs than time domain methods like WSOLA but offers high quality audio, without missing transients. Each iteration of the Phase Vocoder has the following steps:

1. Chop the input signal into several (analysis) frames of size N, each overlapping with the previous one in H_a (i.e.: the analysis hop size) samples.
2. Apply a window to each analysis frame.
3. Calculate the forward Fourier Transform. TODO
4. Calculate the magnitude and phase spectra of the frequency frame obtained in step 1. TODO
5. Compute the phase difference between the current phase spectra and the previous one (i.e.: instantaneous frequency). TODO
6. Calculate the difference between the instantaneous frequency and the centre frequency of each frequency bin. TODO
7. Due to the fact that the phase values are given in modulo 2π and, as such, phase jumps will exists due to the nature of the polar representation, we need to calculate the principal determination of the difference obtained in the previous step. TODO
8. Compute the Inverse Fourier Transform. TODO
9. Add the (synthesis) frame to the output, by overlapping it with H_s (i.e.: the synthesis hop size) samples of the "tail" of the output signal.

This algorithm ensures *horizontal phase coherence* (i.e.: phase continuity for the same frequency bin, in consecutive frequency frames) is guaranteed. However, *vertical phase coherence* (i.e.: phase relations between different frequency bins, in the same frame) is usually destroyed in the phase correction process. The loss of *vertical phase coherence* results in a distinct artefact: *phasiness* (i.e.: metallic tunnel sound). The temporal complexity of the Phase Vocoder is $O(N\,log_2N)$.

To maintain both vertical and horizontal coherence, we can apply a technique known as *Identity Phase-Locking* [**?**]. The main idea is that frequency bins that are not spectral peaks contribute to the partials of the nearest spectral peak. In order to find the spectral peaks, one can use a simple heuristic: if a frequency bin has the maximum magnitude when compared with four neighbour bins, then it is a spectral peak. After identifying all spectral peaks, we need to infer what is the "region of influence" of each peak (i.e.: for a given peak, what are the non-peak bins that contribute to the peak partial) (see section IIIC of [**?**]). After identifying both the spectral peaks and the "regions of influence", the

usual phase correction method is applied to the peak bins. The phase of a non-peak bin will be equal to the phase of its corresponding peak bin.

To pitch shift with a Phase Vocoder, there are two methods available. The first one is re-sampling, in the same manner as OLA and WSOLA. The second one is adding an additional step to the phase correction, as described in [**?**].

3. EXISTING IMPLEMENTATIONS

In this section we give an overview regarding time stretching and pitch shifting implementations with web technologies like JavaScript and Web Audio API, as well as native web browser implementations, available through the Audio Element[**?**], through its *playbackRate* attribute [**?**].

For pitch shift only implementations in JavaScript, there is pitchshift.js[6] and jungle.js[7]. The first one is a port of a c++ implementation[8] of the Phase Vocoder for pitch shifting [**?**] while the second one is an implementation of the algorithm presented in [**?**]. In Vexwarp[9], time stretching is performed with the basic phase vocoder algorithm [**?**, **?**]. With this application, it is not possible to perform real-time processing of the input signal.

Soundtouch.js[10] is a port of the C++ library SoundTouch[11], an WSOLA implementation. This library performs both time stretching and pitch shifting (through re-sampling) with some "tricks":

- Cross-correlation is computed with an interleaved array with all audio channels.
- Instead of implementing the "naive" approach to cross-correlation, the developers used an hierarchical algorithm.

This port is tightly coupled regarding buffers, buffer management, stretcher, re-sampler and parameter adjustments, as well as some hard coded parameters, making integration a difficult task.

There is another WSOLA implementation: tempo.js[12], the result of the compilation of a port of the SoX tempo effect, using Emscripten [**?**]. Currently, it has no documentation and the only demo available uses deprecated APIs that are no longer available.

In WAVES project[13], the Audio library [**?**] supports time stretching and pitch shifting through granular synthesis and re-sampling, offering two classes to perform both tasks: GranularEngine and SegmentEngine. The first one causes significant transient smearing. The second class requires the developer to pass a JSON object detailing the segmentation of the input audio buffer.

Regarding the native implementations in web browsers, all major browsers, like Opera, Safari, Chrome and Firefox, implement time stretching to be used with the Audio tag, controlling the stretching factor with the *playbackRate* attribute of the Audio tag/object. The stretching factor in the current implementations seems to be limited to the range $\alpha \in [0.5 : 4]$, where 0.5 is the slowest speed and 4 is the fastest one. Currently, we can only detail the implementation of Firefox and Chromium due to the closed-source nature of the mentioned web browsers. In Firefox, time stretch is performed by the SoundTouch library. Chromium uses a custom WSOLA implementation. For both browsers, when slowing down, there is some stuttering (more noticeable in Firefox). When speeding up, namely for playbackRate values bigger than 1.2, there are some missing transients, for percussive sounds.

4. NEW IMPLEMENTATIONS

In this section, we detail our two implementations for time stretching, OLA-TS.js (modified OLA) and PhaseVocoder.js (Phase Vocoder with Identity Phase-Locking), as well as some helper classes we created to ease the integration of the time stretchers.

Both implementations operate on a single audio channel and can be included in a ScriptProcessor or AudioWorker for real-time interaction/processing, or they can be used in batch processsing, in a similar way as Vexwarp, in order to integrate in frameworks and applications like Ciseaux[14] and EarSketch. They do not include pitch shifting capabilities but that be easily changed by coupling a re-sampler to the helper classes. In order to maintain a static memory footprint, we used an existing circular buffer implementation, CBuffer[15]. Even though our time stretchers are implementations of (totally) different algorithms, there is a common API to both stretchers:

- *process(Array inputFrame, CBuffer outputFrame)*: given a (mono) frame, performs a phase vocoder iteration and pushes H_s samples in the output CBuffer.
- *get_ha*: returns the current analysis hop size. This is a useful function to calculate the increment to the "read head" of the input signal, when play an audio file.
- *get_hs*: returns the current synthesis hop size. This is a useful function to calculate the increment to the output signal position which an be used to guide the cursor in the UI of an audio player using OLA-TS.js or PhaseVocoder.js as time stretchers.
- *clear_buffers*: clear all internal buffers, like the overlapping buffer. This can be useful for audio players that need to create a noticeable stop in the transition to the next file in a playlist, in order to avoid using the phase of the previous song to adjust the phase of the next song.
- *set_alpha(Number newAlpha)*: given the new stretching factor, computes the new values for H_s, H_a (both integers) and invokes the function pointed by *overlap_fn*.
- *get_alpha*: returns the last specified stretching factor.
- *overlap_fn*: public field pointing to a function that, given a stretching factor α, will return a new overlapping factor.

[6] http://bit.ly/1MQ093A
[7] https://github.com/cwilso/Audio-Input-Effects
[8] http://downloads.dspdimension.com/smbPitchShift.cpp
[9] http://www.vexflow.com/vexwarp/
[10] https://github.com/also/soundtouch-js
[11] http://www.surina.net/soundtouch/
[12] http://bit.ly/1G96QGb
[13] http://wavesjs.github.io/
[14] https://github.com/mohayonao/ciseaux
[15] https://github.com/trevnorris/cbuffer

- *get_real_alpha*: there are stretching factors that do not allow H_s and H_a to be integers and this might present a problem because the input signal "read head" is an integer number (H_a is used to increment the "read head"). When the developer uses *set_alpha* to specify a new stretching factor, both H_s and H_a are rounded to the closest integer. As a result, there will be a difference between the specified α and the real α. This difference can cause problems in use cases like a DJ application that automatically synchronizes two songs to a master tempo. If there is a divergence between the specified α and the real α, after a certain amount of time, this divergence can cause beats of both songs to not align properly. As such, we created this function in order to allow the developer to create adequate controllers to adjust the speed of the audio players to avoid the problem described in the previous sentence.

PhaseVocoder.js use, by default, fourier.js, a FFT library offering methods implemented in both asm.js[16] and "raw" JavaScript. In order to use a different FFT library, the developer can use the following methods: *set_stft_fn(Function stftCallback)*, *set_istft_fn(Function istftCallback)* and *init_fft_fn(Function initCallback)*:

- *stftCallback(Array inputFrame, Array windowFrame, Number wantedSize, Object out)*: *inputFrame* is a sequence of samples; *windowFrame* is the discretization of the window function; *wantedSize* is the desired size for the real and imaginary arrays of the output[17]; *out* is a JSON object with four arrays: real, imaginary, magnitude and phase, all of them describing the result of the forward short time Fourier transform. This function will be invoked for each time frame being processed by PhaseVocoder.js.

- *istftCallback(Array real, Array imag, Array windowFrame, Array timeFrame)*: *real* and *imag* is the real and imaginary array describing a frequency frame; *timeFrame* is the result of the inverse short time Fourier transform. This function will be invoked when synthesizing a frequency frame, after the phase adaptation.

- *initCallback(Number frameSize)*: this function is invoked after being added to a PhaseVocoder.js instance.

OLA-TS.js diverges from the basic OLA algorithm in the following: the window has an exponent that is a function of the stretching factor, $W'(n) = W(n)^{\beta(\alpha)}$. The overlapping factor is too a function of α, $Ovl(\alpha)$. We include two default functions for both the overlapping factor and the exponent. The exponent function can be redefined by changing the public field *beta_fn*. The default functions for the exponent and overlapping factor were designed through experimentation. Both of them are a series of step functions design to minimize the modulation described in section 2.1.

Both OLA-TS.js and PhaseVocoder.js use the following formulas to define the analysis and synthesis hop sizes:

$$H_a = \frac{N}{Ovl(\alpha)}$$

$$H_s = \alpha * H_a$$

[16]http://asmjs.org

[17]*wantedSize* can not be bigger than *windowFrame.length*.

Figure 2: Demo page for PhaseVocoder.js and OLA-TS.js

In order to properly stretch a input signal, the developer should use a predetermined sequence of instructions. To make integration in other applications easier, we implemented helper classes to manage the buffering and the "read heads" for the input buffers.Both implementations and their helper classes are public and open source, each one with its Git repository[18][19]. Additionally, we implemented a small demo page where the user can drag & drop several songs and play them simultaneously, controlling the volume and the stretching factor.

5. DISCUSSION

In this section, we compare our implementations with the existing ones, reviewed in section 3.

Unlike Vexwarp, PhaseVocoder.js allows real-time and block-wise processing of the input signal, with a minimized impact in transient smearing and no metallic sound, due to the maintenance of the *vertical phase coherence*. Unfortunately, due to the number of FFTs used (for $\alpha > 1$, there are 4 forward FFT and 4 inverse FFT per output buffer), the computational costs are greater than soundtouch.js. But, in order to avoid missing transients in soundtouch.js, one needs to decrease the frame size and/or increase thee cross-correlation range, which will increase the computational costs of soundtouch.js and, as a consequence, can make PhaseVocoder.js competitive.

OLA-TS.js allows $O(N)$ time stretching, unlike PhaseVocoder.js, $O(N\,log_2 N)$, and soundtouch.js, $O(N^2)$, but can introduce noticeable modulation in harmonic structures like the voice of a human singer, for frame sizes bellow 4096 samples, with a sample rate of 44100 Hz. Additonally, it may require some manual tinkering with the overlap and beta parameters to reduce modulation[20]. Due to the recommended frame size (4096), OLA-TS.js is adequate for applications where there are smooth adjustments in the stretching factor.

To make the it simple for the reader to understand some of the main features and problems with each overviewed and/or discussed implementation, we have create table 1, listing, for each implementation, its algorithm, audio artefacts and the effect(s) implemented.

5.1 Issues with Web Audio API, regarding time stretching and pitch shifting

Currently, WAA offers two classes to work in frequency domain: (1) *AnalyserNode*, allowing the developer to obtain the magnitude spectra (but no phase spectra) of a time sequence, (2) PeriodicWave, an interface to define a periodic waveform for an *OscillatorNode* in order to perform synthesis using a similar method to the one described in [**?**]. So, there is no way to retrieve the full frequency description in a ScriptProcessor or AudioWorker. This situation requires the developers to use JavaScript implements of the FFT in order to implement high quality time stretching

[18]https://github.com/echo66/OLA-TS.js

[19]https://github.com/echo66/PhaseVocoderJS

[20]We have already two default functions to adjust both parameters.

Table 1: Comparison of Time Stretching/Pitch Shift JavaScript and Native implementations

Name	T. Stretch/P. Shift	Algorithm	Audio Artifacts
SoundTouch.js	Both	WSOLA + Re-Samplig	Missing Transients
WAVES Audio library	Both	G. Synthesis + Re-Sampling	Smeared Transients
pitchshift.js	Pitch Shift	Phase Vocoder	No
jungle.js	Pitch Shift	Delay-Line Modulation	Reverberation
Vexwarp	Time Stretch	Phase Vocoder	Metal Tunnel
tempo-sox.js	Time Stretch	WSOLA	Unknown
PhaseVocoder.JS	Time Stretch	Phase Vocoder	No
OLA-TS.JS	Time Stretch	Modified OLA	Some modulation in harm. structures
Firefox Audio	Time Stretch	WSOLA	Missing Transients
Chromium Audio	Time Stretch	WSOLA	No

with Phase Vocoders, instead of relying on native implementations exposed in a JavaScript API like the WAA. This leads to an increased overhead in computational costs because JavaScript is an interpreted language. This overhead can cause sudden audio dropouts when using algorithms like the Phase Vocoder, Spectral Modelling or Percussive-Harmonic Separation [**?**] within a ScriptProcessor[21]. And, in a Phase Vocoder, the bulk of the computational cost is due to the FFT. This absence caused significant discussion within WAA community[22].

6. CONCLUSION AND FUTURE WORK

In this paper, we presented the existing time stretching and pitch shifting implementations using web technologies, as well as two new implementations. Then, we compared the implementations regarding computational costs and the existence of audio artefacts. Additionally, we comment on the current state of the WAA specification regarding frequency operations and how does that affect time stretching and pitch shifting.

Our next steps regarding OLA-TS.js and PhaseVocoder.js are clear: integrate this implementation in creative frameworks and applications like Flocking and EarSketch. Additionally, we plan on integrating a re-sampler in the helper classes in order to provide pitch shifting. For PhaseVocoder.js, we plan to include new audio effects like robotization and whiperization.

7. ACKNOWLEDGMENTS

8. REFERENCES

[1] Colin Clark and Adam Tindale. Flocking: A Framework for Declarative Music-Making on the Web. In *Proceedings of the 40st International Computer Music Conference*, Athens, 2014.

[2] A. de Gotzen, N. Bernardini, and D. Arfib. Traditional implementations of a phase-vocoder: The tricks of the trade. In *Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00)*, Verona, Italy, December 2000.

[3] S. Disch and U. Zolzer. Modulation and Delay Line Based Digital Audio Effects. In *Proceedings of the 2nd COST G-6 Workshop on Digital Audio Effects (DAFx99)*, NTNU, Trondheim, December 1999.

[4] M. Dolson. The phase vocoder: A tutorial. *Computer Music Journal*, 10(4):14–27, 1986.

[5] T. Karrer, E. Lee, and J. Borchers. Phavorit: A phase vocoder for real-time interactive time-stretching. In *ICMC '06: Proceedings of the International Computer Music Conference*, pages 708–715, New Orleans, USA, November 2006.

[6] J. Laroche and M. Dolson. Phase-vocoder: about this phasiness business. In *Applications of Signal Processing to Audio and Acoustics, 1997. 1997 IEEE ASSP Workshop on*, pages 4 pp.–, 1997.

[7] J. Laroche and M. Dolson. Improved phase vocoder time-scale modification of audio. *IEEE Transactions on Speech and Audio Processing*, 7(3):323–332, 1999.

[8] J. Laroche and M. Dolson. New phase-vocoder techniques for pitch-shifting, harmonizing and other exotic effects. In *Applications of Signal Processing to Audio and Acoustics, 1999 IEEE Workshop on*, pages 91–94, 1999.

[9] E. Lee, T. Karrer, and J. Borchers. An Analysis of Startup and Dynamic Latency in Phase Vocoder-Based Time-Stretching Algorithms. In *ICMC '07: Proceedings of the International Computer Music Conference*, volume 2, pages 73–80, Copenhagen, Denmark, August 2007.

[10] A. Mahadevan, J. Freeman, and B. Magerko. EarSketch: Teaching computational music remixing in an online Web Audio based learning environment. In *Proceedings of the 1st Web Audio Conference*, Paris, 2015.

[11] J. A. Moorer. The Synthesis of Complex Audio Spectra by Means of Discrete Summation Formulas. *Journal of Audio Engineering Society*, 24(9):717–727, 1976.

[12] S. Pieters, A. van Kesteren, P. Jagenstedt, D. Denicola, I. Hickson, R. Berjon, S. Faulkner, T. Leithead, E. D. Navara, E. O'Connor, T. Atkins, S. Pieters, Y. Weiss, and M. Marquis. HTML 5.1, W3C Working Draft, 2015.

[13] M. Roelands and W. Verhelst. Waveform similarity based overlap-add (WSOLA) for time-scale modification of speech: structures and evaluation. In *EUROSPEECH*, 1993.

[21] In Google Chromium/Chrome, if you play, at the same time, 3 or more songs in our demo page, you will notice several audio dropouts. In Firefox, those dropouts still happen but for more than 6 songs.

[22] https://github.com/WebAudio/web-audio-api/issues/468

[14] N. Schnell, V. Saiz, K. Barkati, and S. Goldszmith. Of Time Engines and Masters - An API for Scheduling and Synchronizing the Generation and Playback of Event Sequences and Media Streams for the Web Audio API. In *Proceedings of the 1st Web Audio Conference*, Paris, 2015.

[15] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. 2011.

[16] H. Tachibana, N. Ono, H. Kameoka, and S. Sagayama. Harmonic/percussive sound separation based on anisotropic smoothness of spectrograms. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 22(12):2059–2073, Dec. 2014.

[17] Teoli and C. Mills. Web Audio playbackRate explained, 2014.

[18] Yotam Mann. Interactive Music with Tone.js. In *Proceedings of the 1st Web Audio Conference*, Paris, 2015.

[19] A. Zakai. Emscripten: an LLVM-to-JavaScript compiler. In *26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA*, Portland, OR, USA, October 2011.

[20] U. Zolzer. *DAFX: Digital Audio Effects, second edition*. Wiley, New Jersey, 2011.