A JavaScript Pitch Shifting Library for EarSketch with Asm.js

Juan Carlos Martinez
Georgia Tech School of Music
840 McMillan St, Atlanta GA 30332
jcm7@gatech.edu

Jason Freeman
Georgia Tech School of Music
840 McMillan St, Atlanta GA 30332
jason.freeman@gatech.edu

ABSTRACT
A JavaScript pitch shifting library based on asm.js was developed for the EarSketch website. EarSketch is a Web Audio API-based educational website that teaches computer science principles through music technology and composition. Students write code in Python and JavaScript to manipulate and transform audio loops in a multi-track digital audio workstation paradigm. The pitch-shifting library provides a cross-platform, client-side pitch-shifting service to EarSketch to change the pitch of audio loop files without modifying their playback speed. It replaces a previous server-side pitch-shifting service with a noticeable increase in performance. This paper describes the implementation and performance of the library transpiled from a set of basic DSP routines written in C and converted to Asm JavaScript using emscripten.

1. INTRODUCTION
Since our team began to develop a browser-based version of EarSketch with Web Audio API in 2013, supporting pitch shifting has remained one of our most persistent technical challenges. Web Audio API does not support pitch-shifting natively (i.e. changing the pitch of an audio stream or file without changing its speed), and pitch-shifters built from Web Audio unit generators alone are limited in quality and generalizability [10]. We initially implemented pitch shifting using a ScriptProcessorNode but quickly ran into scalability problems well-known to ScriptProcessorNode, such as the inability to run multiple pitch shifting processes simultaneously without timing and performance artifacts and the inability to include pitch shifting effects in offline rendering contexts.

The approach we used in original production versions of EarSketch, then, relied on server-side audio processing to pitch-shift audio. The web client, using the offline rendering Web Audio functionality, created an audio WAV file and sent it to the server. Then, the server performed the pitch shifting transformation using the C-based Sox audio processing library [2] and returned the result to the client. This process is described in the UML [5] sequence diagram in figure 1.

This approach works, but with several caveats. It uses considerable bandwidth to transmit a rendered WAV file to the server and download the pitch-shifted result. This is particularly problematic since EarSketch is an educational environment and the schools which use it often have limited Internet bandwidth. It also places increased burden on the server to process pitch-shifting, especially as EarSketch's usage has grown.

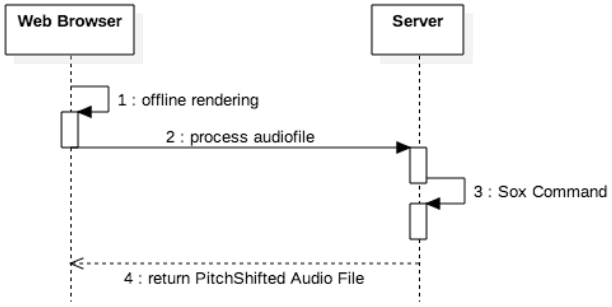

Figure 1. Backend Server Sequence UML Diagram

In the remainder of this paper, we discuss an updated approach to pitch shifting in EarSketch in which we pitch-shift audio files on the client side (Figure 2) using our own pitch shift implementation in C (transpiled to JavaScript with asm.js and emscripten). Like the server-side implementation, this approach works within the current limitations of Web Audio API but creates a client-side service for performing the pitch-shifting, eliminating the bandwidth and server resources previously required and speeding up performance dramatically with no loss in quality. We describe our implementation, evaluate its performance, and suggest it as a generalizable paradigm for addressing Web Audio API limitations through client-side offline rendering.

2. DESIGN
There are several approaches to develop a pitch shifting algorithm; one of them is to perform the pitch shifting transformation by using a Phase Vocoder [8]. Such an approach has been widely used since the 1980s as a pitch transformation tool having outstanding quality results. It is conducted in the frequency domain; therefore it could be slower than other methods in the time domain such as PSOLA [4]. Although PSOLA has good performance for audio samples with a single perceived pitch (monophonic), it is impractical for EarSketch where many audio samples are polyphonic. Taking into account these considerations, the phase vocoder technique was chosen as the best option since this method yields good music quality results in the polyphonic audio samples context.

The implementation of a JavaScript pitch-shifting algorithm raises two performance concerns. First, JavaScript is a dynamic

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

programming language running in a web browser without the capabilities of multithreading nor is it able to run some mathematical procedures at a low machine level. However, a native plugin is not a feasible option because EarSketch must be portable to different web browsers. The second issue is that Web Audio API does not provide a full FFT transform with the phase information (i.e. complex domain), meaning the FFT algorithm also has to be implemented entirely in JavaScript. To address these issues and implement pitch shifting with reasonable performance, we used Emscripten, a transpiler that efficiently converts C/C++ code into Asm.js JavaScript [9]. Asm is a very efficient low-level subset of JavaScript.

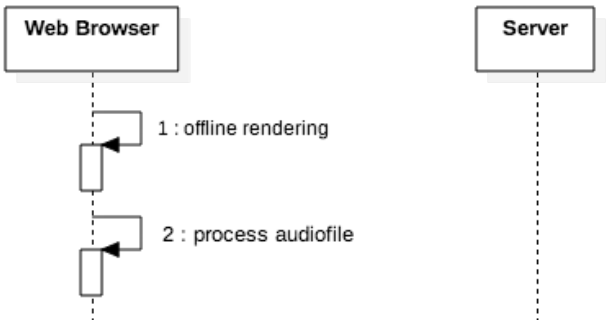

Figure 2. Client Approach Sequence UML Diagram

Our main design goal was to build a web-client solution portable to major web browsers that fully supported Web Audio API and that could be easily migrated to real-time implementation with AudioWorker in the future. In accordance to these goals, the solution design was to implement the Phase Vocoder algorithm business logic in plain JavaScript (i.e. not asm.js) and to migrate the basic DSP methods (i.e. FFT, Phase Computation, and Interpolation) from C to JavaScript asm.js. As a result of this design, the solution is portable as it is written in pure JavaScript, efficient as the key methods are written for asm JavaScript, and easy to migrate as the business logic is readable. An additional advantage of this design is the ease with which additional DSP transformations such as time stretching or spectral morphing can be added to the current functionality. Finally, to help improve performance in the business logic layer, the pitch shifting algorithm implemented in readable code uses Typed Array.

3. ALGORITHM

The basic pitch shift algorithm derives from the phase vocoder C implementation of Moore [3], adding a final step that performs variable-time compression in one block to improve performance, instead of doing it frame by frame. The basic process flow, based on the standard Short Time Fourier analysis, is shown in the figure 3, where the basic processing methods are highlighted, these last methods are the core of the transpiled library.

First, the FFT is conducted for each frame, and then the phase is extracted. Once previous steps are accomplished, a phase transformation is applied on a global array. The next step is to apply the inverse FFT to convert it to time domain and the segment is overlap-added to a global array, this way the phase vocoder ensures a smooth transition between frames. Finally, after the STFT processing takes place, a variable compressed version of the overlapped waveform is created maintaining the source WAV file's original total time.

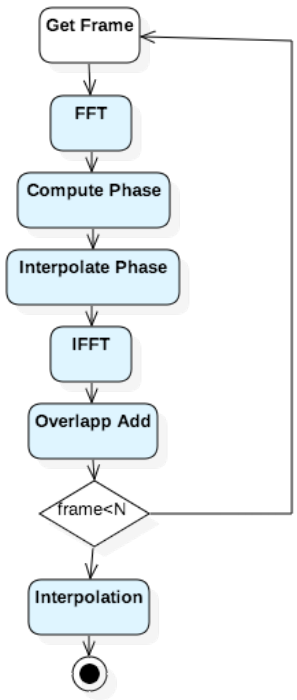

Figure 3. Phase Vocoder Flow Diagram

4. EVALUATION

Tables 1, 2 and 3 show some performance metrics comparing the implemented JavaScript pitch shifting algorithm to other methods. The client used for test 1 and 2 (table 1 and 2) was Google Chrome v45 running on a Mac laptop with OSX Yosemite, a 2.2 GHz Intel Core i7, and 4 GB RAM 1333 MHz DDR3. The internet connection during testing for table 1 has an upload/download average rate of around 40 Mbps. The client used for test 3 (table 3) was Google Chrome v48 and Mozilla Firefox v44 running on a Mac desktop with OSX El Capitan, a 2.8 GHz Intel Core i5, and 16 GB RAM 1863 MHz DDR3.

Table 1 includes the total time of an endpoint testing from EarSketch using two different audio files. The new approach using pure JavaScript generates a 4:1 ratio improvement compared to the previous server-side solution.

Table 1. Comparative Processing Time for Backend Server vs Client Pure Javascript

Audio Wav File Size	Backend Server Duration	JS Browser Duration
48 secs	16 secs	4 secs
96 secs	30 secs	7 secs

Table 2 shows the time differences between the C base code running native and the transpiled JavaScript version executed by the browser. In both implementations, the FFT plus the phase computations take around 80% out of the total time, which is the expected result for a phase vocoder. However, it is important to note that the FFT computation in the native version is 72 times faster than that of the JavaScript browser counterpart. Consequently, if Web Audio had a native FFT transform with phase information (i.e. complex domain), the performance of any

audio transformation in the frequency domain will be increased radically.

Table 2. Profiling of the total time spent by the main pitch shift methods in its two versions: JavaScript Browser and C Native

Method	C Avg Duration	JS Browser Avg Duration
FFT	165 ms / 30%	11941 ms / 51%
Phase Computations	282 ms / 50%	7931 ms / 34%

Finally, in table 3, a 34 second mono audio WAV file was processed with a 2 semitone variable-pitch shift in two different browsers Google Chrome and Mozilla Firefox using our JavaScript library, and the average processing time was measured. Then, we added the 'use asm' directive to our library and repeated the test. In both cases the performance of Firefox was superior to Chrome and when the 'use asm' directive is employed the Firefox average processing time is 40% lower than corresponding Chrome average time.

Table 3. Average processing time between Google Chrome and Mozilla Firefox

'use asm'	Google Chrome	Mozilla Firefox
no	1534 ms	1181 ms
yes	1371 ms	814 ms

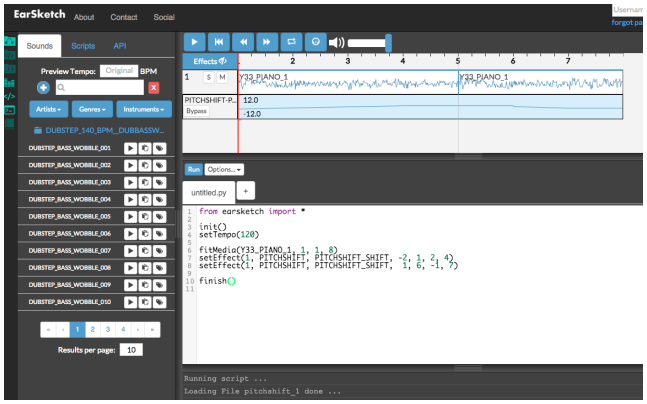

Figure 4. - EarSketch Sample Screen

5. DISCUSSION AND CONCLUSION

This paper describes the design, development and implementation of a pitch-shifting algorithm in JavaScript. The combination of a C/C++ transpiler as emscripten to create a basic DSP library plus the use of typed arrays in plain JavaScript results in a readable, portable, performance-efficient and maintainable code library in JavaScript. Additionally, it can be executed across different web browsers that fully support Web Audio API such as Firefox, Chrome and Safari. Although it is possible to create a very efficient code with this approach, the fact that Web Audio does not provide a full FFT transform with phase information limits significantly the development of state of the art audio transformations at the browser level. We hope that this library offers a generalizable paradigm for combining asm.js DSP functionality with Web Audio API applications in an organized, efficient, and extensible manner.

The JavaScript pitch shift library was successfully integrated in the EarSketch environment (Figure 4) and is currently in the production release. The source code for the library is released under a MIT license and is available at https://github.com/GTCMT/pitchshiftjs.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under DRL #1417835. Many thanks to the entire EarSketch project team (http://earsketch.gatech.edu/personnel). EarSketch is freely available at http://earsketch.gatech.edu/.

7. REFERENCES

[1] Freeman, Jason, Brian Magerko, and Regis Verdin. "EarSketch: A Web-based Environment for Teaching Introductory Computer Science Through Music Remixing." In Proceedings of the 46th ACM Technical Symposium on Computer Science Education, pp. 5-5. ACM, 2015.

[2] Sox – Sound eXchange. "Welcome to the home of SoX, the Swiss Army knife of sound processing programs." http://sox.sourceforge.net/ (accessed September 26, 2015).

[3] Moore, F. Richard. *Elements of computer music*. Prentice-Hall, Inc., 1990.

[4] Valbret, Hélene, Eric Moulines, and Jean-Pierre Tubach. "Voice transformation using PSOLA technique." In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, vol. 1, pp. 145-148. IEEE, 1992.

[5] Rumbaugh, James, Ivar Jacobson, and Grady Booch. Unified Modeling Language Reference Manual, The. Pearson Higher Education, 2004.

[6] Laroche, Jean. "Time and pitch scale modification of audio signals." In Applications of digital signal processing to audio and acoustics, pp. 279-309. Springer US, 2002.

[7] Mousa, Allam. "Voice conversion using pitch shifting algorithm by time stretching with PSOLA and re-sampling." *Journal of electrical engineering* 61, no. 1 (2010): 57-61.

[8] Moorer, James A. "The use of the phase vocoder in computer music applications." Journal of the Audio Engineering Society 26, no. 1/2 (1978): 42-45.

[9] Asm.js . "an extraordinarily optimizable, low-level subset of JavaScript." http://asmjs.org/spec/latest/ (last accessed September 26, 2015)

[10] Wilson, Chris. "Audio-Input-Effects." https://github.com/cwilso/Audio-Input-Effects/blob/master/js/jungle.js (last accessed September 26, 2015