Using Empirical Analysis of Music Corpora to Optimize Web Audio Playback

Tom Collins
Music Artificial Intelligence Algorithms, Inc.
P.O. Box 73004
Davis, CA 95617
tomthecollins@gmail.com

Christian Coulon
Music Artificial Intelligence Algorithms, Inc.
P.O. Box 73004
Davis, CA 95617
christianco@gmail.com

ABSTRACT

Due to feasibility issues and musical preferences, Web audio applications have tended to emphasize the use of synthesized instruments and short samples (e.g., drums) over large banks of longer files that sample other acoustic instruments such as a violin or piano. As the sounds generated by sampled acoustic instruments are quite realistic, they are likely to be of interest to many users of Web audio applications. Using the Tone.js Web Audio framework, this paper describes an initial investigation into load times when rendering music with such sampled instruments. A method is proposed for reducing load times, and hence optimizing Web Audio playback, based on empirical analysis of the note durations used across different music corpora. Experimental results for 400 randomly selected short music excerpts indicate that the proposed method does lead to significant load time reductions, from 3.87 s to 1.72 s. Researchers interested in replicating the results of these experiments or downloading and exploring our playback solution are pointed to http://tomcollinsresearch.net/research/wac/2016/

1. INTRODUCTION

One of many interesting application types to emerge from recent Web audio work is the browser-based interface for writing a song or piece of music [11, 3, 8]. Playback in such applications entails various challenges for the Web audio developer. One challenge is that to achieve instantaneous and accurate playback, the required audio content has to be loaded into the `AudioBuffer`, but this content may be fixed only moments before a user presses 'play'. Synthesizers and drum tracks tend not to place too many demands on the loading process, consisting typically of oscillator definitions and manipulations, and short small audio files respectively. If a user has the need for realistic violin or piano sounds, however, playback could involve the loading of longer, larger audio files belonging to a sampled instrument.

In this paper we investigate the above scenario, as well as the extent to which empirical analysis of music corpora – specifically knowledge of durational properties – can lead to the definition of sampled instruments with quicker load times, hence optimizing Web audio playback. For the sample bank duration distribution indicated by the dashed black line in Fig. 1, the mean duration of an audio file is 8.59 s, standard deviation 2.43 s. Such long samples ensure that if a song/piece requires a sustained instance of a note, then there is sufficient audio material to render it. An analysis of the note durations used by two different composers, as shown in Fig. 1, suggests that very few notes approach these long durations, however, with nearly 95% of notes lasting 1 s or less. It seems inefficient to load a long audio file if 95% of the time a truncated 1 s version of it will suffice. Loading such long files places unnecessary demands on the `AudioBuffer`, which may undermine instantaneous and accurate playback.

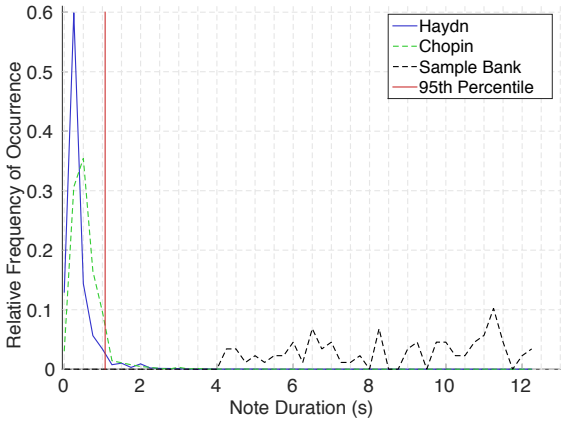

Figure 1: Empirical density functions for note durations (seconds) in pieces by Haydn (solid blue line), Chopin (dashed green line), and in an instrument sample bank (dashed black line). The vertical solid red line indicates the 95th percentile of the Haydn durations distribution, which happens to equal that of the Chopin durations distribution.

The premise of this paper is that for any given sampled instrument, we should maintain the original, long audio files alongside a bank of copies truncated to ∼1 s in duration. When a user presses play, it can be determined easily whether the use of MIDI note y in her/his song always lasts less than this ∼1 s threshold. If so, the audio can be loaded from the truncated bank of files rather than the original, long versions. The outcome could be reduced demand on the `AudioBuffer`, reduced load time, and an optimized playback. After reviewing some methods for playback optimiza-

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

tion already in use in desktop and Web contexts, we describe two experiments using the Tone.js Web Audio framework [6, 7] that investigate load times and the potential benefits of truncated sample banks.

2. EXISTING APPROACHES

The literature on sample-based synthesis cannot be reviewed thoroughly given the current scope. Heuristics exist in Desktop digital audio workstations for reducing the amount of audio material that has to be loaded to render a given sampled instrument. In Apple's Logic Pro [2], one WAV file might be loaded to render a piano playing MIDI note number y, and this same audio data is then pitch shifted to render any occurrences of the surrounding MIDI notes $y-2, y-1, y+1, y+2$. The amount of audio material that has to be loaded is reduced, therefore, here by a factor of five. A downside of pitch shifting is that it is often possible to tell that a note has been altered, due to the introduction of artifacts, which makes the playback sound less authentic.

Scheduling is another heuristic that can reduce instantaneous load demands, and has been discussed recently in the context of Web audio [9]. The idea of scheduling is to spread loading of audio material across the playback epoch, so that rather than all the required audio files being loaded at time zero, any given audio file is buffered a fixed time prior to being rendered. In this way, scheduling can reduce the wait time when a user presses 'play', because there is less demand on the `AudioBuffer` at time zero. As an example, in a metronomically exact playback of Fig. 2, an audio file for rendering pitch B3 in measure 41 is not required until 3.158 s.[1] Assuming that this audio file can be loaded within 0.5 s, say, then during playback we can wait until 2.658 s ($= 3.158 - 0.5$) to add this file to the buffer queue. A potential problem with scheduling is that if the connection speed slows during playback, certain audio files may not load in time and the song/piece will be rendered inaccurately.

Version r5 of the Tone.js framework [6, 7], built on the Web Audio API [1], provides a helpful method called `parseScore`, which can be used to render note information that is contained in a JS object using one or more synthesizers or sampled instruments. For example, the following code could be used to render the excerpt shown in Fig. 2 with a sampled string instrument:

```
1  var str = new Tone.PolySynth(8, Tone.Sampler, {
2   "A0": "sbank/021.wav", "A#0": "sbank/022.wav",
3   ..., "C8": "sbank/108.wav"
4  },
5  {
6   pitch: 0
7  }).toMaster();
8
9  Tone.Note.route(
10  "Str", function(ontime, pitch, dur, vel){
11   str.triggerAttackRelease(
12    pitch, dur, ontime, vel
13   );
14  }
15 );
```

[1]Counting quarter-note beats from zero at the beginning of measure 39, B3 appears on beat 8, and at a tempo of 152 BPM, this should sound at $3.158\,\text{s} = 8 \cdot 60/152$.

```
16 var Score = {
17  "Str" : [
18   [ 0, "E3", .197, .8 ],    // 1st note Vc. m.39
19   [ 0, "E4", .197, .8 ],    // 1st note Vla
20   [ 0, "E4", .197, .8 ],    // 1st note Vln. II
21   [ 0, "G#4", .444, .8 ],   // Tied note Vln. I
22   [ .197, "E2", .197, .8 ], // 2nd note Vc. m.39
23   ...
24   [ 3.158, "B2", 4.342, .4 ], // Tie Vc. m.41
25   ...
26   [ 7.105, "B3", .395, .4 ] // Last in Vla. m.43
27  ]
28 };
29 Tone.Note.parseScore(Score);
```

The variable `str` (lines 1-7) defines the sampled string instrument with reference to a bank of wav files. The call to `route` (lines 9-15) provides a handle for triggering notes with this instrument of specifiable start time, pitch, duration, and velocity. The `Score` variable (lines 16-28) is the JS object that contains these specifications, and finally `Score` is passed to `parseScore` (line 29).

In version r5 of Tone.js, all audio files specified in an instrument definition (lines 1-7 in the above example) are added to the buffer queue, even if some of these files are never used by the `Score` variable. Using Tone.js as we do here to define a sample bank, this queue-all approach would increase load times unnecessarily. In the code supporting our experiments, we introduce a check for required files and then refine the instrument definition accordingly. Preliminary investigations of whether compressed formats (e.g., MP3) with Tone.js lead to substantially quicker buffer times than uncompressed formats (e.g., WAV) were inconclusive: there is a compression/quality payoff to consider, and decompression of the MP3 (required for subsequent manipulations) can make total buffer times comparable.

While recognizing the relevance of different file transfer protocols, pitch shifting, qualitatively different note intensities, and scheduling, these matters will be put to one side in what follows, to focus on the effect of a truncated sample bank on buffer load times. Pitch shifting is possible in Tone.js (line 6 of the code) but it seems to be applicable to a whole instrument only – rather than to certain samples within an instrument – making it unclear if pitch shifting would lead to efficiency gains for this particular framework.

To our knowledge, little research has been conducted on using the Web Audio API to load samples for acoustic instruments such as violin or piano, nor has it been established whether empirical analysis of music corpora could result in reduced load times for such instruments. Therefore, these topics will be the focus of the experiments described below, complementing existing knowledge of real-world use of the Web Audio API.

3. EXPERIMENTS

3.1 Hypothesis

To motivate our experiments, we began by selecting twenty movements from the string quartets of Haydn, twenty mazurkas by Frédéric Chopin (1810–1849), and conducting a rudimentary analysis of the note durations in each corpus.[2]

[2]The Haydn pieces were op.17 nos.1, 2, 3, 5, 6 and the

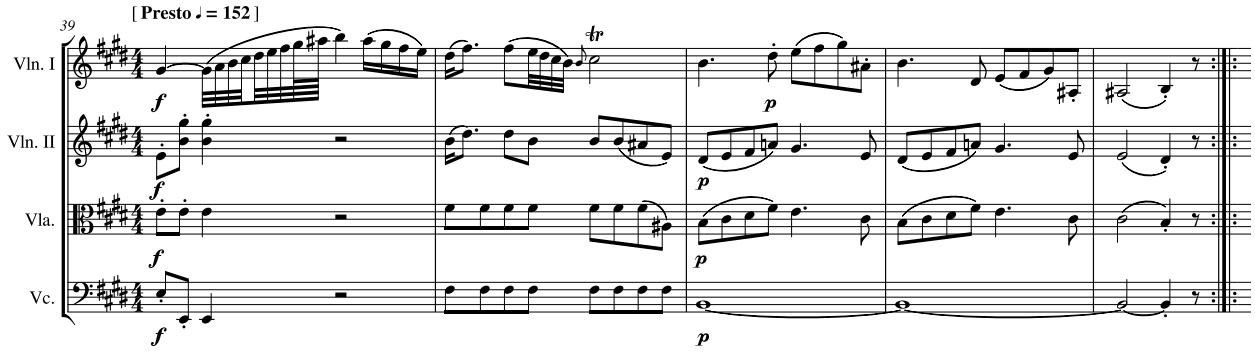

Figure 2: Excerpt from first movement of String Quartet in E major op.17 no.1 by Joseph Haydn (1732–1809).

The pieces selected cover a range of tempi, from largo or 50 beats per minute (BPM) through to presto or 172 BPM. Converting the opening tempo instruction and any subsequent tempo changes to a BPM value, it was determined how long in seconds each notated duration would last in a metronomically exact performance. One vector of these performed durations was calculated for the Haydn corpus, a second vector for the Chopin corpus, and these vectors form the basis of the empirical probability density functions given in Fig. 1. The vertical red line in Fig. 1 indicates the 95th percentile of the Haydn durations distribution (solid blue line). That is, 95% of performed durations in the Haydn corpus last 1.07 s or less. The Chopin durations distribution (dashed green line in Fig. 1) is flatter and with a larger mean than for Haydn, but it too has a 95th percentile of 1.07 s.

The above analysis leads to the hypothesis that a playback system based on an original sample bank plus a truncated copy (with files truncated to 1.07 s to be used whenever the duration of required notes allows) will have shorter load times on average than a playback system based only on the original, long audio files.

Cross-validation is the reason for choosing composers from different periods and pieces for different instrumental forces: the first corpus could be used to calculate a duration threshold with which material from the second corpus is rendered, and vice versa, to avoid training and testing systems on the same data. As mentioned above, however, the durations distributions turned out to have the same 95th percentile, and so we did not pursue a cross-validation approach in this initial investigation, instead using one threshold of 1.07 s.

3.2 Method

The experiments and supporting code are available from http://tomcollinsresearch.net/research/wac/2016/ for rerunning and/or download.

3.2.1 Stimuli

In Experiment 1, ten excerpts covering twelve quarter-note beats were selected at random from each of the twenty Haydn pieces, giving 200 stimuli in total. Twelve quarter-note beats was a sensible choice, because with or without scheduling, it might be necessary to load this amount of audio at a given time. The excerpts, while from complete classical pieces, were intended to represent passages of in-progress compositions in a playback context (i.e. the user has written something and presses 'play' to hear the results).

For the purposes of fair comparison and sake of simplicity, the same instrument (a grand piano sound) was used in each experiment. To create the truncated sample bank, the original, long samples were imported into a signal processing program, restricted to 47 250 ($\approx 1.07 \cdot 44\,100$) samples, and exported with an identifying tag appended to the file name.

3.2.2 Apparatus

A MacBook Pro running Google Chrome on OS X 10.9 with a 2.9 GHz processor and 8 GB RAM was used to browse the excerpts. The connection speed was 7616 KBits/s at the experiment's outset and similar, 7777 KBits/s, at the experiment's end. We could have run the experiment on a local host to remove the influence of connection speed on load times, but it is a more accurate reflection of real-world use of the Web Audio API to run it online at a connection speed typically available from US ISPs. Obtaining the same overall experimental outcomes several times satisfied us that connection speed did not constitute a nuisance influence.

3.2.3 Procedure

The excerpts were opened in a browser window one by one, and the time in seconds that it took to load the audio content was recorded automatically using a combination of HTML, JS, and PHP. The next excerpt loaded automatically after a fixed period of time (1 min), and cached files were cleared at the end of each experiment.

In Experiment 1a, Haydn excerpts were loaded using original, long audio files. In Experiment 1b, exactly the same excerpts were loaded but now using the original, long audio files alongside a bank of truncated files. Experiments 2a and 2b followed the same pattern, the only difference being Chopin excerpts were loaded instead of Haydn.

3.3 Results

The load times for Experiments 1 and 2 are shown in Fig. 3. Evidently, the load times for Experiment 1b are shorter than those for Experiment 1a (means of 1.72 s and 3.87 s respectively). In a paired t-test, this is a significant difference ($t(199) = 12.99,\ p < .001$). Also clear is that the load times for Experiment 2b are shorter than those for Experiment 2a (means of 1.62 s and 4.10 s respectively). Again a paired t-test reveals that this is a significant difference ($t(199) = 14.28,\ p < .001$). Outliers in Fig. 3 are due to excerpts requiring the long version of one or more samples.

Both Experiments 1b and 2b used the original, long audio

Chopin pieces were opp.17, 24, 30, 33, 41 [10, 5].

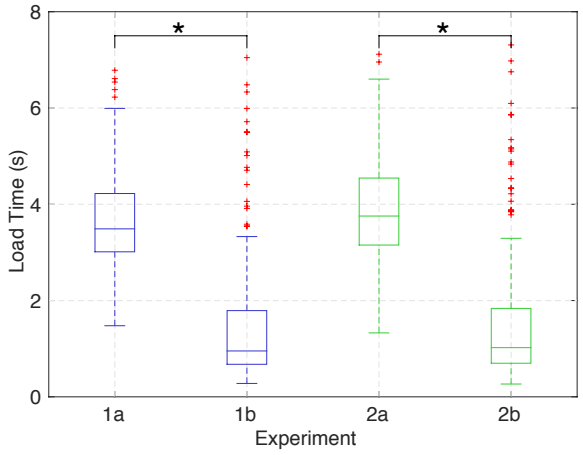

Figure 3: Boxplots showing the load-time distributions for Experiments 1a (Haydn with original, long samples), 1b (Haydn with original plus truncated samples), 2a (Chopin with original, long samples), and 2b (Chopin with original plus truncated samples). Red crosses are outlying load times – some not shown for the sake of clarity (> 8 s). Asterisks indicate significant differences between distributions.

files alongside a bank of truncated files, and the results show that this system reduces load times compared with Experiments 1a and 2a, respectively, which used the original, long audio files only. This finding confirms our hypothesis, that a playback system based on an original sample bank plus a truncated copy will have shorter load times on average than a playback system based only on the original, long audio files. Across experiments, the average load time reduction was 2.32 s, and for the faster system, `AudioBuffer` load time for a twelve-beat excerpt took an average 1.67 s.

4. DISCUSSION

Via online experiments involving the loading of randomly selected, twelve-beat excerpts of music, we investigated typical buffer times of the Tone.js Web Audio framework [6, 7] and, by extension, the Web Audio API [1]. The experiments focused on the playback scenario where a user needs realistic-sounding acoustic instruments to render an in-progress song/piece. Using a sample bank with mean sample duration 8.59 s, average load times were in the region ∼4 s. With much existing work on Web audio applications having used synthesizers and short samples (e.g., drums) [11, 8], this focus on realistic-sounding acoustic instruments and their load times complements existing knowledge regarding real-world use of the Web Audio API. With an average load time of ∼4 s for twelve beats' worth of musical material, users of browser-based interfaces for music creation may have to be patient regarding instantaneous and accurate playback with realistic-sounding acoustic instruments.

We proposed and tested a playback system based on a new method that places a truncated sample bank (sample duration 1.07 s) alongside the original, long samples of an acoustic instrument, and draws from the truncated bank whenever the note durations belonging to a music excerpt allow. The truncation factor of 8 ($\approx 8.59/1.07$) was determined by empirical analysis of durations observed in two different music corpora. Compared with a baseline system that used only the original, long samples, our proposed system led to more than halving load times, from ∼4 s to 1.67 s. While there is still work to be done in a Web Audio context to achieve instantaneous and accurate playback of realistic-sounding acoustic instruments, this reduction represents a substantial improvement.

Further work on the influence of pitch shifting and scheduling will help to determine whether `AudioBuffer` times can be reduced further, and whether solutions scale up successfully to longer excerpts and whole songs/pieces. More nuanced empirical analyses could also be conducted, for instance taking into account the use of playing techniques such as pedaling, and/or exploiting findings that lower pitches tend to have longer durations and higher pitches shorter durations [4] to create a tiered bank of truncated samples, rather than truncating all samples to one fixed duration irrespective of pitch.

5. ACKNOWLEDGMENTS

We would like to thank three anonymous reviewers for helpful comments on an earlier version of this paper.

6. REFERENCES

[1] P. Adenot and C. Wilson. Web Audio API GitHub repository. `https://webaudio.github.io/web-audio-api/`. Retrieved August 5, 2015.

[2] Apple Inc. Logic Pro. `http://www.apple.com/logic-pro/`. Retrieved October 1, 2015.

[3] J. Berkovitz. Noteflight: a Web-standards-based compositional community. In *Proceedings of the Web Audio Conference*, January 2015.

[4] Y. Broze and D. Huron. Does higher music tend to move faster? Evidence for a pitch-speed relationship. In *Proceedings of the International Conference on Music Perception and Cognition*, pages 159–165, July 2012.

[5] Center for Computer Assisted Research in the Humanities. Kern Scores. `http://kern.ccarh.org/`. Retrieved July 1, 2015.

[6] Y. Mann. Tone.js GitHub repository. `https://github.com/Tonejs/Tone.js`. Retrieved August 5, 2015.

[7] Y. Mann. Interactive music with Tone.js. In *Proceedings of the Web Audio Conference*, January 2015.

[8] J. Monschke. Web Audio Editor. `http://web-audio-editor.herokuapp.com/`. Retrieved September 17, 2015.

[9] J. Monschke. Building a collaborative music production environment using emerging Web standards. Master's thesis, Hochschule für Technik und Wirtschaft Berlin, Germany, 2014.

[10] C. S. Sapp. Online database of scores in the Humdrum file format. In *Proceedings of the International Society for Music Information Retrieval Conference*, pages 664–665, September 2005.

[11] A. Väänänen. AudioSauna. `http://www.audiosauna.com/`. Retrieved August 10, 2015.