A FAUST-based re-creation of the power amp stage for WebAudio-based simulations of guitar tube amplifiers

Michel Buffa, Jerome Lebrun

A FAUST-BASED RE-CREATION OF THE POWER AMP STAGE FOR WEBAUDIO-BASED SIMULATIONS OF GUITAR TUBE AMPLIFIERS

Michel Buffa

Université Côte d'Azur, CNRS, INRIA
Sophia Antipolis, France
michel.buffa@univ-cotedazur.fr

Jerome Lebrun

Université Côte d'Azur, CNRS
Sophia Antipolis, France
lebrun@i3s.unice.fr

ABSTRACT

In this paper, we detail our on-going browser-based re-creations of famous tube guitar amplifiers and describe the JavaScript implementations we have been developing using the WebAudio API. The tricky part of such amplifiers is the power stage (Power Amp) which contains a parametric negative feedback loop. We show the limits of the high-level WebAudio API layer, and how FAUST allows us to re-implement the Power Amp part more faithfully. Finally we also compare FAUST vs JavaScript development, and mention future optimizations.

1. INTRODUCTION

Since 2015, we have been designing and developing faithful simulations of iconic tube guitar amplifiers like the Marshall JCM 800 or the Mesa-Boogie 2:90, used by many famous artists (AC/DC, Guns and Roses, Deep Purple, Metallica, etc.), *that can run in a web browser* [1].
This work consisted in emulating the different parts of the electronic circuit of this amplifier using WebAudio, implementing the necessary signal processing algorithms using the available API, and finding adequate solutions to circumvent some limitations specific to the web browser environment (thread priority, latency, JavaScript API limitations). Finally, we extensively compared (quantitatively and qualitatively) our realization with the state of the art, i.e. native simulations, mostly commercial, written in C++, and not having the constraints of webapps. The results exceeded our expectations. Since then, we have continued this work by refining the models used in the simulation, and we designed a framework to reproduce different electronic architectures present in various other tube amplifiers found in many musicians' equipment [2]. We can now simulate for example a Fender, a Vox or a Mesa Boogie amplifier, etc. or even create new original designs. These customizable simulations have been tested by professional guitarists, are being used by music schools on an experimental basis and are the subject of a marketing contract by the CNRS in order to be included as plugins [3, 4] in an online commercial digital audio workstation.
So far, our simulations were fully developed in JavaScript and based on the WebAudio API.

2. WEBAUDIO

WebAudio is a W3C-standard JavaScript API[1] that relies on building an "audio graph" by connecting processing nodes one to another. The signal is sequentially processed through the block diagram where each node crossed can modify the signal (e.g. a filter node may be used to remove high-pitched sounds, etc ...). Some particular nodes can also be used as a sound source (audio file, wave generator, link to an HTML5 element <audio> or <video>).

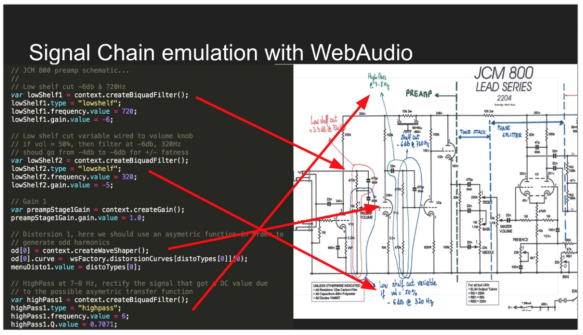

Fig. 1. *High level simulation. The lamps and filters are identified and simulated part by part using the WebAudio API.*

To our knowledge, there is no previous research work that has tried to simulate a complete guitar amplifier using WebAudio. When we started, we made the choice of relying on higher level psycho-acoustic emulations, in which the "logical" parts are well identified (filters, lamps, etc.) and perceptually emulated using separate templates for each part (Fig. 1). This work has been presented in [1] and validated by professional guitar players during different campaigns of user tests. This may be in theory less accurate than a 'global' physical simulation because some effects and interactions such as feedback loops from overloaded tubes or the damping factor of the loudspeaker impedance on the power amplifier may not be precisely taken into account. However, this approach is much simpler and more tunable. It can be implemented real-time within a browser and is

[1] https://www.w3.org/TR/webaudio/

surely more adapted to WebAudio, its ecosystem and its limitations.
In our approach, we precisely analyzed the electronic schematics of amplifiers: lamps, filters structures, power supply, output transformers were identified and simulated part by part by JavaScript code relying on the WebAudio API. WebAudio provides a set of "high-level" nodes (such as the WaveShaper node and the BiquadFilter node) that can be used to model the lamps and filters, and it has been shown that when properly used, waveshaping techniques[2], combined with proper oversampling and filtering, can give quite good results [5].

One big advantage of JavaScript/WebAudio API is the really flexible and dynamic way you can manipulate the audiograph (by changing its topology in real-time, even while playing guitar) or the parameters of the different nodes, in particular, you can reshape in real-time the transfer functions used by the waveshaper nodes.
Another advantage of using WebAudio high level nodes, is that most of the audio processing is done in their C++ implementation that lies in the web browser internals, so performance is generally not an issue. Our simulation has a very low CPU impact and runs audio processing threads with the highest priority. On a MacBook Pro from 2016, we could run up to 15 amp simulations in parallel (all stages: preamp, tonestack, power amp, speaker simulation), in a DAW, without noticing any glitche and with a CPU stress level of less than 50%.

However, until recently, one main limitation of the WebAudio API design was that default signal processing is constrained to block-processing of chunks of 128 samples at a time, and until very recently it was not possible to do stream processing at the sample-wise level without introducing glitches and latency. Not being able to perform processing with time-granularity below this 3ms limit caused a lot of issues in the implementation of the PowerAmp part of the amplifier schematic. Indeed, in its classic Push/Pull configuration, the power stage includes a crucial negative feedback loop that could not be faithfully simulated without introducing customized solutions to mitigate this major limitation.
This issue has been recently partially solved with the arrival of the AudioWorklet node and a new WebAssembly standard in 2018[3]. Writing custom DSP code in JavaScript, or coding in WebAssembly by hand, is nevertheless quite tedious. Fortunately, FAUST quickly proposed WebAudio/WebAssembly as a compilation target and proved to be an ideal framework for developing powerful custom code (we even did some personal contributions to FAUST's WebAudio support) [6].

[2] The most straightforward method for obtaining signal distortion with digital devices is to apply an instantaneous nonlinear transformation, using a so-called "transfer function" from the input signal to the output variable. This type of timbre alteration is coined *waveshaping* [11, 12].

[3] WebAssembly is a W3C standard: a portable binary-code format for executable programs, firstly to be used on the Web, but also on native environments. WebAssembly aims to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms. See https://webassembly.org.

In this paper, we will detail how a PowerAmp works (Section 3), introduce our initial solution to mitigate the block-processing limitations of WebAudio (Section 4), and devote the rest of the article to detail our FAUST and WebAssembly-based approaches to achieve more faithful low-latency simulations (Section 5). We will conclude by comparing the advantages and disadvantages of the two approaches (JavaScript + WebAudio high level nodes vs. FAUST/WebAssembly) with special care on the performance measurements (latency, cpu usage, etc.) (Section 6).

3. Tube Guitar Amplifier Design

3.1. Overview

A guitar amplifier is composed of different parts: usually, a preamplifier stage, a so-called "tonestack" stage that includes bass, midrange and treble controls, and a power stage (the Power Amp). See Figure 2 and 3 below..

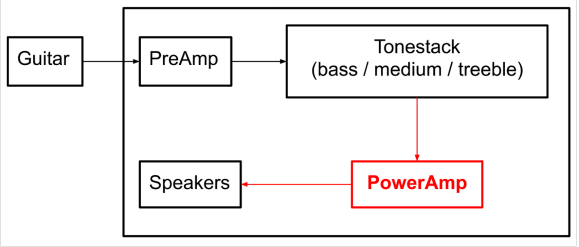

Fig. 2. *The typical stages of guitar amplifiers and their associated signal paths. Some brands (as Fender) may switch the PreAmp and the Tonestack stages.*

The preamplifier beefs up the high-impedance low-level signal coming from the guitar pickup microphones to a lower impedance mid voltage signal that can drive the power stage. The preamplifier also shapes the tone of the signal; high settings of the preamplifier lead to 'overload' that creates crunch/distorted sounds. The power amplifier with the help of the output transformer outputs a very low impedance, high current signal adequate to efficiently drive a loudspeaker and to produce loud amplified sounds. Another clear aim of guitar amps is of course to create desirable distortions too.

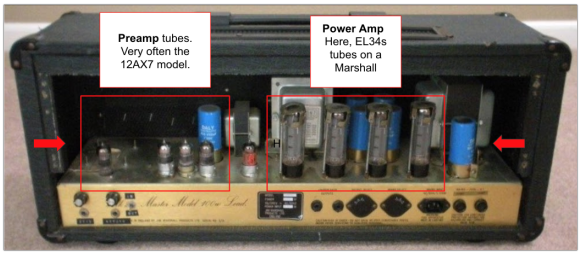

Fig. 3. *The back of a Marshall JCM800 power head. We can see the two amplification stages.*

To get a finer understanding of this critical stage, we will further detail how a PowerAmp works, its role in the signal chain and why the power amplifier is so tricky to emulate due to the presence of some feedback loop in the circuitry.

3.2. The PowerAmp stage

Usually solely controlled by the Master Volume and Presence knobs (Fig 4.), the Power Amp stage has a profound impact on the sound and overall dynamics. Indeed, the type of sound you can get from the preamp stage alone remains on the lean side in terms of distortions.

Fig 4. *The Power Amp stage is controlled by the Master Volume and Presence knobs.*

Referring to Kuehnel [7] and Denton [8], when you turn up the volume of a tube amp using the Master Volume knob, the power tubes get more and more distorted and the sound gets thicker and thicker, and also less controlled. And as you get closer to power saturation (i.e. clipping the power tubes and/or saturating the output transformer core causing the famous grinding effect, a mixture of saturation/oscillations, but also possibly starving the power supply - more about this later in this section), the output dynamics get tighter, resulting in muddy distortions with a heavier sound. The English slang to characterize these sounds is difficult to translate: in French, one would speak of "thick", "heavy", "metallic", "abrasive" (translation of gritty), "rich" sounds.

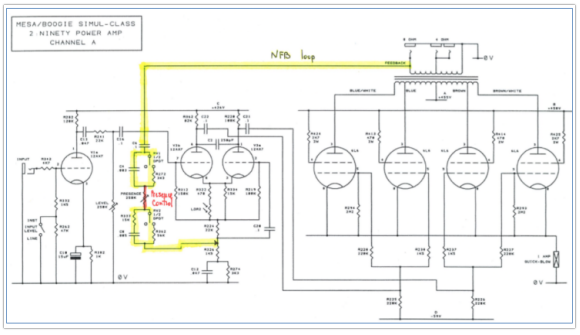

Fig 5. *For illustration, the Power Amp stage of a Mesa/Boogie 2:90. The Negative Feedback Loop (NFB)/Presence with its RC network is highlighted. The JCM 800 has similar topology.*

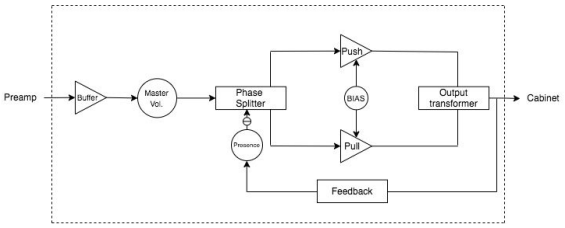

Fig 6. *An abstraction of a classic Push/Pull Power Amp.*

In Power Amps, the Negative Feedback Loop (NFB)/Presence control (see Figs. 5 and 6) are typically introduced to extend the usable frequency bandwidth by limiting/reshaping the unwanted distortions originating from the non-linearities of the output transformer at the price of a slightly lower total gain. The negative feedback loop (NFB) takes a portion of the signal from the output transformer secondary winding, dephasing it with a 180 degree phase shift and re-injects it back into the splitter stage (see the highlighted plot in Fig. 5 and the "Feedback" loop in Fig. 6). Amplifiers without NFBs are usually more powerful (higher gain) with more distortion, but their coloration also tends to become rougher and unpredictable when pushed into saturation. The NFB loop smoothes out the sound by reducing the level of distortion in the parts of the spectrum where it is most disturbing (typ. mid-range) [7].

Unknown to concert stage or HiFi power amps, a 'Presence' control is added to the NFB loop of guitar amps to allow control on the coloration of the NFB and thus to provide a simple but efficient way of adjusting the brightness and sharpness of the sound at the Power Amp stage. The Presence may be looked at as a global time/frequency control on the brightness of timbre[4] by altering the signal fed back from the negative feedback loop. The effects of negative feedback can be reduced for certain frequencies as the Presence knob controls the resistor part of a RC network, hence a tunable filter made of Resistors and Capacitors (RC) in the loop. With less negative feedback in the high frequencies, the sound becomes brighter, louder, more vivid and dynamic. The behavior of this control is very different from those of the tonestack (bass, midrange, treble), which merely equalizes the output of the preamp. Namely, unlike conventional treble control which is mainly subtractive, Presence control is pseudo-additive in that it limits the fundamental subtractive aspect of the NFB loop. It should be noticed also that the NFB/Presence has a major influence at the temporal level [9] as the RC networks controlled by the Presence knob induce some frequency-dependent group-delay in the NFB loop. This may explain the blurring/sharpening effect that Presence has on the attack slopes of the notes played, acting as a "softening/anti-softening" pedal. This clearly motivates our introduction of a curve-based parametric Presence control to choose in which frequency band one wants its brightness.

To design properly a NFB/Presence circuit, lots of parameters are involved and a lot of care must be taken when adjusting them. With real amplifiers' power amps, manufacturers are very conservative: the allowed range of Presence control is restricted so as to avoid unwanted oscillations that may be destructive to the speaker/cabinet. As a consequence, only subtle alterations (mostly upper-mids/lower-high range brightness) are possible, the sound signature being mainly carved through the preamp

[4] Fender: Be in the Moment: The Presence Control Explained: What is it and how can it help energize your live sound? (https://www.fender.com/articles/tech-talk/be-in-the-moment-the-presence-control-explained)

settings. In our simulations, the curve-based parametric Presence control allows us to fully carve the time/frequency coloration at the Power Amp level to add punch or special effects to the output from the Preamp and Tonestack.
Some other important controls on the Power Amp are the Master volume to adapt gain with the Preamp output level, and the Bias setting of the tubes may be changed to introduce some light non-linearities/asymmetries in the tube response curves. In real-world tube Power Amps, Bias allows us to change the class of operation of the tubes (typ. class AB for push-pull) from almost class A to almost class B. And of course the Presence setting for adjusting the overall tone/brightness is quite important and can lead to destructive positive feedback if not designed correctly (freq ranges, etc.). Historically, some amps have a Power Amp section which is more important in their sound signature, in general they are the most "vintage" ones, think of the Fenders, the first Marshalls, the Vox AC30 …
Improving the real-time dynamics in simulations is crucial to get them more realistic[5]. Namely, when pushed to their limits by large transients, classic guitar tube amplifiers (esp. with tube rectifiers based power supply) have a tendency towards harmonic saturation with typical dynamic range compression/expansion effects (known as 'sag' and 'bloom'), temporal lag (known as 'squish') or spatialisation artifacts (known as 'swirl')[6]. Sag is consecutive to some drooping of the supply voltage in the Preamp stage in response to large transients. The recovery from Sag as voltage supply returns to normal is coined as 'Bloom' and is again a well appreciated effect. Now, Squish is linked to some temporal hysteresis induced by some increased time lag in the feedback loop - also in response to large transients. Finally, Swirl is linked to saturation of the core in the output transformer from overload, leading to phase inconsistencies and spatial blurring of the chord played (ala Univox Uni-Vibe pedal effect). All those effects proved to be quite tricky to emulate in real-time[7].

We introduced a dedicated method to approximate these advanced temporal, nonlinear behaviours of tubes (typ. 'sag'/'bloom', 'squish'/'swirl' effects). Namely, in the WebAudio API, the slope of the tube transfer function curves can be driven real-time by the power of input signal enveloppe emulating hysteresis phenomena. Consequently, to mimic this hysteresis and the sag in the response of a tube Preamp stage, we implemented it with simple dynamical real-time changes in the slopes of the transfer functions in our preamp simulation[8]. By properly adjusting the threshold at which we get the squish, only the higher power transients from the envelope amplitude will trigger controlled change of slope. This is still an experimental feature that needs to be properly adjusted and evaluated. Figure 7 shows a curve animated in real time with hysteresis behaviour in the transfer function triggered by the input signal envelope.

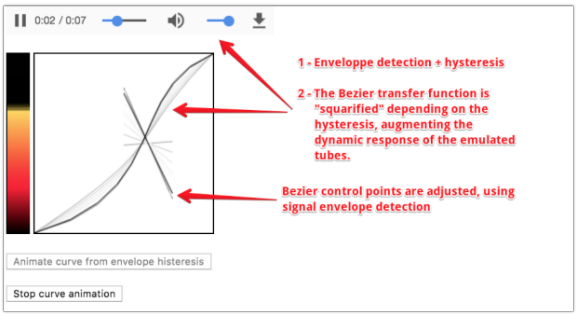

Fig 7. *A tool we developed in order to adjust the real-time dynamics of the tube simulation based on waveshapers*[9].

The Bias control corresponds to the idle voltage around which the tubes are amplifying signals and thus controls their linearity/asymmetry characteristics[10]. With Bias, one can change the amount of (negative) voltage offset applied to the signal at the grid of the tubes. Colder settings will make the pentodes/tetrodes draw less current, decreasing the overall output volume and potentially introducing cross-over distortion due to the class AB getting closer to class B (which may be what a guitarist looks after to achieve a more dirty/loose tone). Hotter settings will make the pentodes/tetrodes draw more current moving closer to class A, with a thicker, louder perceived output level and eventually up to power supply "sagging" (depending on the Sagging control setting), adding compression or, in extreme cases, saturation and cleaning up the tone from potential cross-over distortion.

4 - Implementing the Power Amp stage with WebAudio high level nodes, dealing with limitations

In the current implementation, the presence filter is fully customizable and can be controlled in real time using a graphic equalizer (Fig. 8) to select the frequency range of the filter.
The min and max values of the negative gain in the NFB can also be adjusted using a slider. These tools (NFB and Presence) are quite sensitive (at the edge of creating positive feedback with oscillations and Larsen effects so we provide controls in the GUI to adjust/restrict the admissible range) but this novel presence control provides an utterly powerful and spectacular tool to shape the final sound.
Being able to adjust all the parameters (gains, filter parameters, transfer function of the waveshaper, etc.) is crucial to fine tune this stage using WebAudio nodes. We strongly advise the player to watch this YouTube video[11] which shows the differences in sound and dynamics with and without this loop (Power Amp on / off) in our simulation.

[5] Video: https://www.youtube.com/watch?v=zBhn7odezUQ
[6] More details on: http://www.aikenamps.com/index.php/what-is-sag
[7] Some examples at: http://www.diale.org/tube_emulation.html
[8] See this in our amp simulation in this video: https://www.youtube.com/watch?v=zBhn7odezUQ

[9] https://jsbin.com/zotaver/edit
[10] Aiken - The last word on Biasing: https://www.aikenamps.com/the-last-word-on-biasing
[11] WebAudio implementation demo of the PowerAmp stage: https://www.youtube.com/watch?v=-NdMdJQx2Bw

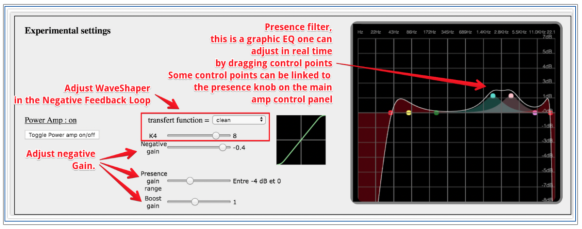

Fig 8. *The settings of the NFB power amplifier can be changed/adjusted dynamically on the fly, even while playing, without generating any audio glitches/dropouts.*

It sounds and plays very well (we made several user evaluations [1, 2] that showed that guitarists, even professional ones, liked the way the dynamics of a real amp was simulated). As said in previous sections, we simulate tubes using filters and waveshaper nodes with properly adjusted transfer functions. As shown in Fig 8. The green transfer function curve on the left of the graphic EQ (presence bank of filters) is rather linear, meaning that only little distortion is added to the signal, and only at high amplitudes. Being able to adjust this function is also crucial to avoid oscillations. On top of that, the Bias control has to be properly adapted to the selected tube transfer functions to guarantee coherent coloration/output volume (making it easier to compare different tubes and choose the right one).

Nevertheless, we encountered many issues in the implementation of our signal loops in WebAudio. Proper simulations of NFB and Presence have been quite difficult to achieve due to some limitations of the WebAudio API and divergences/bugs in how browsers generally parse the WebAudio graphs with loops. In the WebAudio API specs, loops in the graph are required to include at least one delay node. Without this delay node, Firefox stops rendering the graph, while Chrome does not complain but adds, behind the scenes, a 3 ms delay (the minimal delay from a frame audio buffer of 128 sample-frames as within WebAudio API, the signal is always processed in packets of 128 samples, which means that the minimum value for a delay is 128 / sample rate, i.e. about 128/44100 = 3ms). With the current limitations in WebAudio, and quite strangely, this 3ms delay in the loop to conform to the specs, brings slightly different coloring of the amps between FF and Chrome. We have reported these errors and discussed them with the implementers (Raymond Toy from Google and Paul Adenot from Firefox) but so far, nothing has been fixed (September 2020). For a good implementation of NFB, a stable delay of fewer samples would have been preferable (higher delays increase latency and the softening of attacks). Now, to faithfully implement loops like the NFB with its RC networks inducing shorter delays, finer precision at the level of some sample-frames is required.
To circumvent these limitations and allow a proper sample-wise accuracy in the processing of loops, we decided to re-implement the NFB loop (and other critical parts in general) in FAUST as AudioWorklets, ending up re-implementing all the processing nodes present in the circuit: filters, gain and wave shaper, in this language.

5. FAUST IMPLEMENTATION OF THE POWER AMP STAGE

A guitar amplifier is composed of different parts: usually, a Preamplifier stage, a so-called "Tonestack" stage that includes bass, midrange and treble controls, and a Power stage (the Power Amp). Our general approach consisted in looking at the different parts we needed to re-create. However, as mentioned in the previous section, the poweramp stage is quite tricky to implement as it includes a negative feedback loop. In our initial purely WebAudio implementation [1, 2], we were using a bank of biquad filters in series (for the presence implementation), a waveshaper node with appropriate biquad filters (for simulating tubes), some gain nodes (master volume at the input of the stage, negative gain in the NFB loop and a few others for fine tuning the signal level at different locations in the audio graph), and a delay node (in the NFB loop).
Aware of the great difficulties encountered in simulating the power amp in JavaScript/WebAudio, we first tried to re-create as faithfully as possible the signal chain that gave good results. For example, using the same types of filters with the same parameters, the same transfer function with the core issue of simulating properly a waveshaper in FAUST, the same gain values, etc. And of course, we looked closely at the behavior of the NFB loop whose delay hopefully should be lower than the 3ms barrier imposed by the previous implementation.
This way, we can replace the current power amp implementation (made of a dozen of high level WebAudio nodes) with a single FAUST generated AudioWorklet node [12].
WebAudio comes with a set of classic biquad filters types: lowpass, highpass, bandpass, lowshelf, highshelf, peaking, notch and allpass. All these filters have a fixed set of parameters: frequency, gain, Q and detune, whereas some of these parameters are not relevant to some type of filters (as Q for lowshelf/highshelf). FAUST does not come with similar filters out of the box. After trying to adapt existing FAUST filters to behave like WebAudio filter ones, and after talking with FAUST and WebAudio implementers the conclusion was that for a really faithfull behavior, it would be better to start from the original C++ implementation of the WebAudio filter API, taken from the Chromium browser source code. The FAUST team did the port and provided us with the so-called webaudio.lib that is now available in the FAUST distribution.
For the power amp tubes, we looked at the way FAUST developers simulated tubes (e.g. in the Guitarix project[13] source repository, in particular in the guitarix.lib file), or waveshapers (as in several distortion plugins such as the ones by Oleg Kapitonov[14] or by Nick Thompson's Creative Intent Temper Distortion plugin[15]). We found out that most tube simulations relied on C/C++ code and could not be used out of the box (typ. guitarix tube simulations), as we

[12] We already did that in the past by replacing the tonestack stage by some FAUST implementations [2].
[13] http://guitarix.org/
[14] https://github.com/olegkapitonov/Kapitonov-Plugins-Pack
[15] https://github.com/creativeintent/temper

must be able to run these in a Web browser, and the FAUST toolchain still does not support hybrid FAUST/C source code when the compilation target is set to WebAudio/WebAssembly.

```
tanh(x) = x * (27 + x * x) / (27 + 9 * x * x);
transfer(x) = tanh(k * x) / tanh(k);
```

Fig. 9: *FAUST transfer function, uses an approximation of tanh for computational improvements*[16].

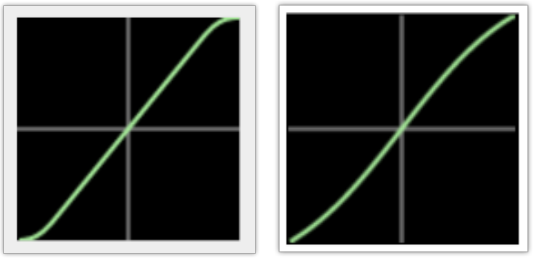

Fig. 10: *Left, the transfer function for our JS implementation, Right, the one from the FAUST implementation*[17] *with k=1.*

The Temper distortion simulation, however, used a 100% FAUST based implementation of a simple waveshaper that produced a warm, adjustable, distortion sound that could easily fit our needs. We used it as a starting point. Furthermore, the code contained the definition of a transfer function curve that could be adjusted (i.e using a more subtle/less aggressive curve). Fig. 9 shows the transfer function from the Temper Distortion waveshaper code, that is very close to the one we used in our JavaScript implementation (see Fig. 10. for comparisons), the parameter *k* driving the S shape of the curve.

Figs. 11 and 12 show the final diagram of the FAUST implementation of the PowerAmp based on the Temper Distortion source code. Differences from the Temper Distortion source code are mainly the introduction of the Presence (made of two peaking filters, at 2kHz and 4kHz) in the feedback loop, the introduction of an adjustable negative feedback gain, the removing of some unnecessary elements (i.e. a resonant lowpass filter at the beginning of our signal chain).

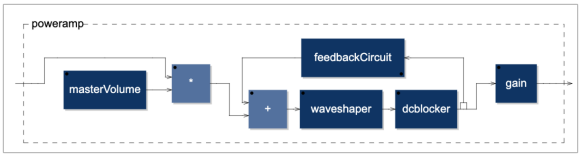

Fig. 11: *Diagrams of the final implementation.*

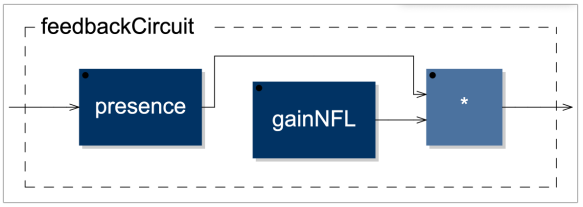

Fig. 12: *The feedback circuit. The Presence filter is obtained using a set of peaking filters ported in FAUST from the WebAudio API implementation.*

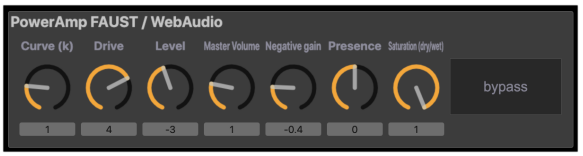

Fig. 13: *GUI generated by the FAUST IDE, some extra parameters (negative feedback gain, waveshaper; drive, curve, saturation) are tweakable, enabling fine tuning of the NFB loop.*

Finally we added GUI elements (knobs) in order to fine tune in real time different parameters (Fig. 13), in particular the ones that control the waveshaper (drive, curve, distorsion), the NFB gain and the Presence filters.
The current implementation (mainly based on the code from the Temper Distortion) of the waveshaper does not rely on pre-calculated point tables, but on a transfer function applied to each sound sample, which leaves room for optimization. The dynamic time response of the tubes is approximated using an amplitude follower placed in the signal chain just before the waveshaper that drives an allpass filter (and which aims at modifying the DC offset, and thus the slope of the curve). We still have to evaluate whether this approach is as efficient or flexible as the method we used previously in the JavaScript implementation, which changes the slope but also the S-shape of the curve (see section 3, Fig. 7).

Once our FAUST-based Power Amp re-creation was functional and adjustable, we could proceed to the evaluation phase.

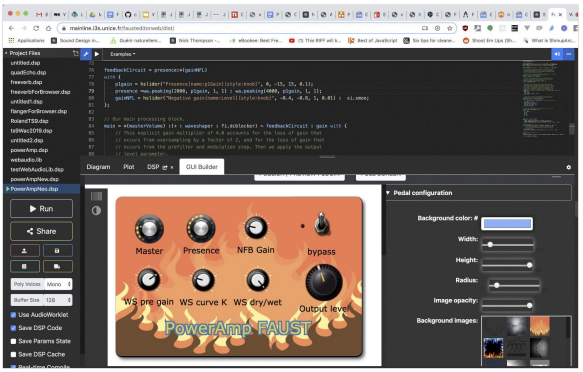

Fig. 14: *For testing purposes, we created a WebAudio plugin from the FAUST code: using the WAP GUI Builder we developed, integrated in the FAUST IDE.*

[16] See http://www.musicdsp.org/showone.php?id=238
[17] Check our online comparison tool: https://jsbin.com/qifexor/edit?js,console,output,

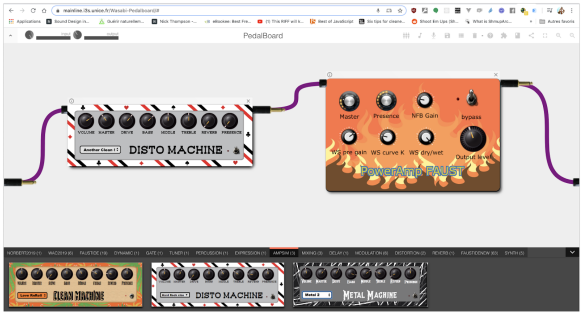

Fig. 15: *The PowerAmp plugin in our pedalboard application, with a version of our AmpSim in which we bypassed our Power Amp pure-JavaScript version and the cabinet simulator stages.*

6. Evaluation

The first step of the evaluation was to listen to the global overall sound when we used the Power Amp in standalone mode, tweaking the different parameters (Master volume, Presence, negative gain, transfer function parameters), and to check its behavior compared to our previous WebAudio/JavaScript implementation. We did some trials using dry guitar sound samples, but also with a real guitar as inputs. The general feeling is that the two main settings, Master volume and Presence, reacted very closely with both implementations. In addition, the classic effects of oscillation and positive feedback could be obtained again when pushing some parameters close to the limit values (Presence, NFB gain).

In a second step we used the FAUST IDE to create a WebAudio plugin from the FAUST implementation of the Power Amp (Fig. 14) and we chained a special version of our tube guitar amplifier simulations in which we bypassed the embedded power amp and cabinet simulation stages (Fig. 15), and compared with the full featured, JavaScript based simulations. Results can be seen/heard in a video we published online[18], or in the online pedalboard WebAudio application[19].

The differences in terms of sound/timbre and playing dynamics are small and subtle. However, we noticed that the FAUST implementation was much more stable and versatile when adjusting the internal parameters of the feedback loop. The processing in our JavaScript implementation was based on blocks of 128 samples, inducing an undesirable delay of 3ms in the back-fed signal as opposed to only two samples with the FAUST loop[20]. In the FAUST implementation, the measurement tools (Fig. 16) proved the sample-wise nature of the processing with a delay of just one-sample for the NFB/Presence loop.[21] This also explains the increased stability of this loop. Now, in terms of aggregated latency for the Power Amp, we did measurements of the "end-to-end" latency, from guitar to cabinet and obtained consistently better values for latency with the new FAUST implementation: around 20-21 ms compared to the 23-24 ms latency of our previous finely-tuned JavaScript implementation (both using a Firefox Nightly 75.0a1 browser with an external Focusrite Scarlett and a Macbook Pro 16 under 10.14). This confirms a saving of 3ms in accordance with the difference of processing of loops between FAUST (sample-wise) and WebAudio API/JavaScript (block based).

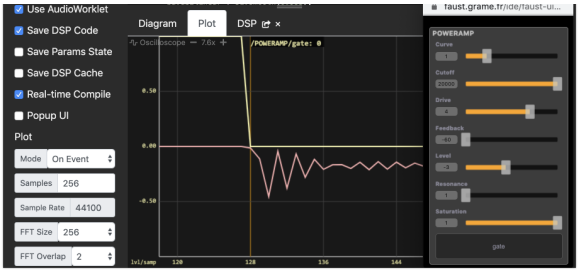

Fig. 16: The latency can be measured achieving sample-wise accuracy using the tools embedded in the IDE. Yellow: a gate signal, Pink: the output from the PowerAmp. X-axis is in samples.

In addition, the use of the FAUST IDE for coding, debugging and profiling this Power Amp plugin has been evaluated by six audio plugin developers with different levels of expertise with the FAUST language. They had to follow a tutorial that guided them and after that, they had to answer a detailed survey. The IDE has been considered as "very useful" for prototyping rapidly an audio plugin, prior to polishing its GUI by hand, editing the HTML/CSS code. Details of this evaluation are available in [13].

7. Perspectives and Conclusions

FAUST has proven to be particularly suitable for porting the JavaScript code of an existing Power Amp in a very similar way, freeing it from the limitations, inconsistencies and constraints of the WebAudio API. We performed preliminary qualitative measurements and measured the latency of the processing, which proved to be almost identical. We did not investigate some aspects that can still be improved/optimized, such as the implementation of a full-featured waveshaper in FAUST (based on pre-calculated tables and interpolation to describe the transfer function, for example), or on more complex tube models. At the evaluation level, additional systematic measurements have to be made and the power amp behavior should also be compared with native implementations, for example, those of the TPA-1 by Ignit Amps[22] plugins, or the TSE X50 by TSE audio[23] (in which the poweramp stage can be isolated).

[18] https://youtu.be/uNp-0hzveeo

[19] https://mainline.i3s.unice.fr/Wasabi-Pedalboard/#, check the PowerAmp tab at bottom, drag'n'drop in the main area.

[20] As measured in the FAUST IDE, using `process = button("gate") <: ((poweramp), _);` style code and the embedded visualization tools.

[21] As measured in the FAUST IDE, using `process = button("gate") <: ((poweramp), _);` style code and the embedded visualisation tools.

[22] http://www.igniteamps.com/#tpa-1

[23] https://www.tseaudio.com/software/x50v2

ACKNOWLEDGMENTS

This work was partially supported by the French Research National Agency (ANR) and the WASABI [21] team (contract ANR-16-CE23-0017-01).

REFERENCES

[1] Buffa, M. and Lebrun, J. 2017. Real Time Tube Guitar Amplifier Simulation using WebAudio. In *Proc. 3rd Web Audio Conference* (WAC'17), London, UK. ISSN 2663-5844

[2] Buffa, M. and Lebrun, J. 2018. WebAudio Virtual Tube Guitar Amps and Pedal Board Design. In *Proc. 4th Web Audio Conference* (WAC'18), Berlin, Germany. ISSN 2663-5844

[3] Buffa, M., Lebrun, J., Kleimola, J., Larkin, O., and Letz, S. 2018. Towards an Open Web Audio Plug-in Standard. In *Companion Proc. The Web Conference 2018* (WWW '18). Lyon, France. (April 23--27, 2018). 759-766. doi:10.1145/3184558.3188737

[4] Buffa, M., Lebrun, J., Kleimola, J., Larkin, O., Pellerin, G. and Letz, S. 2018. WAP: Ideas for a Web Audio Plug-in Standard. In *Proc. 4th Web Audio Conference* (WAC'18), Berlin, Germany. ISSN 2663-5844

[5] Pakarinen, J. and Yeh, D.T. 2009. A Review of Digital Techniques for Modeling Vacuum-Tube Guitar Amplifiers. *Computer Music Journal*, 33(2), 85-100. doi:10.1162/comj.2009.33.2.85

[6] Ren, S., Letz, S., Orlarey, Y., Michon, R., Fober, D. and al.. FAUST online IDE: dynamically compile and publish FAUST code as WebAudio Plugins. In *Proc. 5th Web Audio Conference* (WAC'19), Trondheim, Norway. ISSN 2663-5844

[7] Kuehnel, R. 2005. *Circuit Analysis of a Legendary Tube Amplifier: The Fender Bassman 5F6-A*. Pentode Press. 2005. ISBN 978-0976982258

[8] Denton, D. 2013. *Electronics for Guitarists*. Springer-Verlag. 2013. ISBN 978-1-4614-4087-1

[9] Letz, S., Orlarey, Y., and Fober, D. 2017. Compiling Faust Audio DSP Code to WebAssembly. In Proc. *3rd Web Audio Conference* (WAC'17), London, UK. ISSN 2663-5844

[10] Alves, L.N. and Aguiar, R.L., 2005. On the Effect of Time Delays in Negative Feedback Amplifiers. In Proc. *2005 IEEE Int. Symp. Circuits and Systems* (IEEE ISCAS 2005), Kobe, Japan. doi: 10.1109/ISCAS.2005.1464755

[11] Arfib, D. 1979. Digital Synthesis of Complex Spectra by Means of Multiplication of Nonlinear Distorted Sine Waves. *J. AES,* 27(10):757–768.

[12] LeBrun, M. 1979. Digital Waveshaping Synthesis. *J. AES,* 27(4):250–266.

[13] Ren, S., Letz, S., Orlarey, Y., Michon, R., Fober, D., Buffa, M., and Lebrun, J. 2020. Using Faust DSL to Develop Custom, Sample Accurate DSP Code and Audio Plugins for the Web Browser. *J. AES,* in Press, doi:10.17743/jaes.2020.0014