Faust Plugins in (Sometimes Unexpected) Web-Based Hosts

Michel Buffa, Dorian Girard, Samuel Demont, Quentin Escobar, Ayoub Hofr

Proceedings of the International Faust Conference (IFC-24), Soundmit, Turin, Italy, November 21-22, 2024

FAUST PLUGINS IN (SOMETIMES UNEXPECTED) WEB-BASED HOSTS

Michel Buffa [*]

University Côte d'Azur
Nice, France
michel.buffa@univ-cotedazur.fr

Dorian Girard, Samuel Demont, Quentin Escobar, Ayoub Hofr

University Côte d'Azur
Nice, France
dorian.girard@etu-univ-cotedazur.fr
samuel.demont@etu-univ-cotedazur.fr
quentin.escobar@etu-univ-cotedazur.fr
ayoub.hofr@etu-univ-cotedazur.fr

ABSTRACT

In this short paper we will present the use of FAUST based Web Audio Modules plugins in two hosts: an open source DAW and in an collaborative, immersive, WebXR application in the Musical Metaverse.

1. INTRODUCTION

WAM Studio is an online Digital Audio Workstation (DAW) for creating audio projects, designed as multi-track musical compositions [1, 2]. It has been designed around the standard for Web Audio plugins and hosts called "Web Audio Modules" (or "WAM") [3]. Each track represents a different layer of content that can be recorded, edited, played back or integrated with audio files. Some tracks can control virtual instruments, containing only the notes to be played and metadata. Users can add or delete tracks, play them individually or together, and arm them for recording (Figure 1). The integration of FAUST based plugins is detailed in section 2.

The Musical Metaverse (MM) [4] is an immersive virtual space dedicated to musical activities, expanding on the broader concept of the Metaverse. It has recently gained popularity, reigniting interest in shared virtual environments within the realm of computer music. Applications include virtual concerts, educational tools, and collaborative musical performances. Transposing user experiences into conventional 2D applications generally does not work in an immersive environment, and new ergonomic and sensory approaches need to be employed [5]. The WAM application presented in this paper focuses on a persistent, real-time multi-participant immersive world for shared music creation, exploiting recent W3C web standards such as Web Audio, Web MIDI, WebXR, WebGL, WebGPU, WebAssembly, WebSockets, now implemented in the web browsers of the most common VR/XR headsets available on the market. In this application, users can build music installations by connecting Web Audio Modules plugins in a graph. Most of these WAM plugins are made with FAUST. An original approach has been developed for integrating existing WAMs in the 3D world, with a GUI generated on the fly, providing a user experience adapted for VR headset controllers.

[*] work has been supported by the French government, through the France 2030 investment plan managed by the Agence Nationale de la Recherche, as part of the "CA DS4H project, reference ANR-17-EURE-0004

2. FAUST BASED WEB AUDIO MODULES IN THE WAM STUDIO DAW

2.1. General Features

WAM-Studio is a powerful online Digital Audio Workstation (DAW) that utilizes cutting-edge technology to enable users to playback, record audio and MIDI tracks, employ high-quality plugins (effects and virtual instruments), manage latency, and perform offline rendering[1]. The source code is readily available (mono repository with front-end and back-end, including a simple Docker image for deployment) and the application is available online [1].

The project is developed in TypeScript, deliberately avoiding external frameworks, with the aim of making the code accessible to a wider audience and ensuring its long-term viability (minimal build tools). For those with a keen interest, WAM-Studio is like an "alarm clock to be disassembled» and will reveal the secrets of its design and implementation to the most curious, potentially uncovering insights into tasks that are not well-documented within the Web Audio and Web MIDI communities. It has been designed around the standard for Web Audio plugins and hosts called "Web Audio Modules" (or "WAM") [3] and serves as a compelling demonstration of its vast potential.

Each track represents a different layer of content that can be recorded, edited, played back or integrated with audio files. Some tracks can control virtual instruments, containing only the notes to be played and metadata. Users can add or delete tracks, play them individually or together, and arm them for recording. During recording, all other tracks play simultaneously, while armed tracks record new content.

In WAM-Studio, each track is a container for audio or MIDI related data, accompanied by an interactive representation of this data, editing and processing functions, and a few default parameters such as the track's volume and left/right panning. Figure 1 shows some audio tracks in WAM Studio with the associated audio buffer region (waveforms) and MIDI regions (squares) displayed and the default track controls/parameters on the left (mute/solo, record arming, volume, stereo panning, automation curve display, effects plugins). As many tracks can be displayed, scrolled during playback, zoomed and edited, we used the pixi.js library to efficiently manage drawing and interaction within an HTML canvas. This library uses GPU-accelerated WebGL rendering and offers

[1] https://wam-studio.i3s.univ-cotedazur.fr/https://wam-studio.i3s.univ-cotedazur.fr/, source code: https://github.com/Brotherta/wam-studiohttps://github.com/Brotherta/wam-studio

Figure 1: WAM Studio typical view, showing audio and MIDI tracks with some parameter automation curves and the plugin chain associated with the selected track.

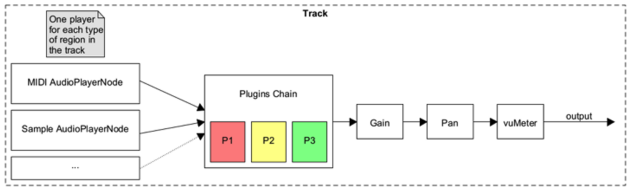

Figure 2: Audio graph of a track implementation.

many features for managing multiple layers on a single HTML5 canvas.

Figure 2 shows the audio graph corresponding to the processing chain of an audio track. The sound goes from left to right: first the "track player/recorder/editor" is implemented as an AudioWorklet node, using custom code to render an audio buffer or a MIDI region, then the sound goes through a chain of WAM plugins for adding audio effects, then the output signal has its gain and stereo pan adjusted, then we have another AudioWorklet node for rendering volume in a canvas (VU-meter).

Plugin chains are managed using a special WAM plugin that also acts as a "mini host" (Figure 3 and 4). We call it the WAM-bank (or the "WAM pedalboard") [6]. It connects to one or more plugin servers, which return(s) the list of available plugins as a JSON array of URIs (a WAM plugin can be loaded simply using a dynamic import and its URI, see [2]). From this list of URIs, WAM plugin descriptors are retrieved, which contain metadata about the plugins: name, version, provider, thumbnail URI, type (effect or instrument), available inputs and outputs etc. When the pedalboard plugin is displayed in the DAW, the chain of active plugins is empty, and plugins can be added to the processing chain, deleted, reordered and their parameters set.

Any configuration can be saved as a named preset (e.g. "crunch guitar sound 1", “Synthesizer with ambient audio effects”). Presets can be organized into sound banks ("rock", "funk", "blues"). Managing the organization and naming of banks and presets is the responsibility of the WAM-bank plugin. The parameters exposed by this plugin correspond to all the parameters of the active preset (i.e. the sum of the parameters of the preset’s plugins in the chain) and can be automated by the DAW.

Figure 3: WAM Bank is a special WAM that manages the chain of plugins associated with a track.

Figure 4: WAM plugins associated with a track

2.2. Rapid Development of WAM Plugins Using the Online IDE

All WAM plugins have a URI and can be dynamically imported into hosts using JavaScript dynamic import statements. The DAW uses JSON configuration files that contain a list of WAM plugin URIS. When one clicks on the FX icon of a track, an instance of the WAM-bank plugin manager is created, and acts as a Mini-Host for handling the chain of plugins (instruments, effects) that will be associated with the track.

Most instruments and audio effects in the current version of the DAW have been developed and exported using the FAUST IDE directly, without any further modifications [7]. A step by step tutorial about how to build WAM plugins with the FAUST IDE is available online [2]

In addition, many WAMs developed by the community of developers have been made available through the "WAM Community REST API", an endpoint from which WAMs can be requested online [8]. As of October 2024, dozens of WAM plugins are available.

Figure 6 shows the MIDI flute instrument in the GUI Builder of the FAUST IDE (the FAUST code comes from the example menu of the IDE). From this GUI Builder, an optional custom GUI can be designed (changing the position, sizes, look and feel of the buttons, changing the fonts used etc.), and by pressing the "Publish / preview WAM2" button, a WAM plugin is generated and published on a remote server. It can be tested directly from the IDE or downloaded as a ZIP file. Once published, its URI can be directly

[2]Create your own Web Audio Plugins with the FAUST IDE: `http://tinyurl.com/yckdyax4`http://tinyurl.com/yckdyax4

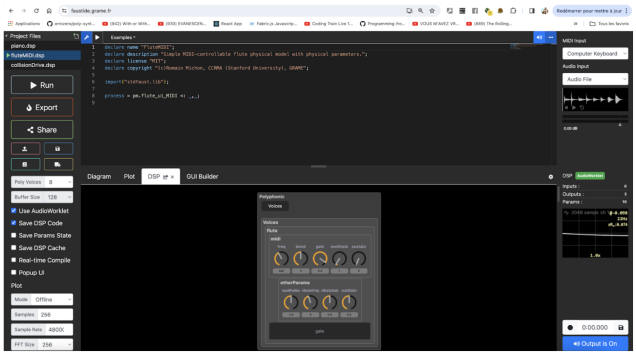

Figure 5: Auto-generated CSS based GUI of the MIDI flute instrument.

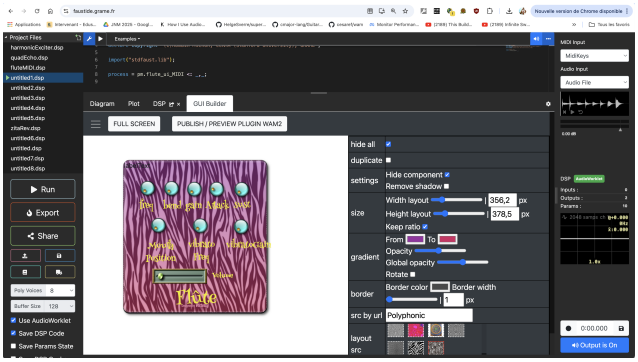

Figure 6: MIDI flute virtual instrument in the GUI Builder of the FAUST IDE

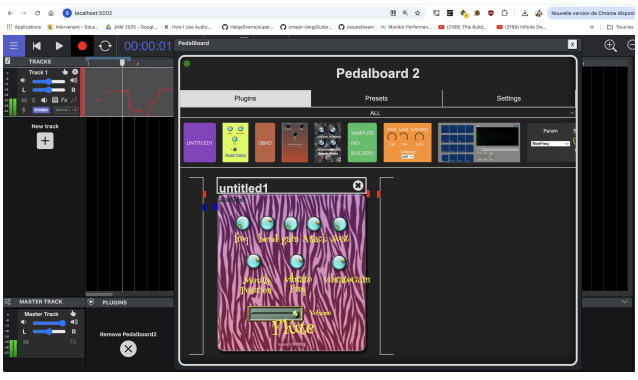

Figure 7: The Midi flute WAM in WAM Studio.

used in any WAM host. Figure 7 shows it in the WAM Studio DAW. The whole operation (compiling the source code in the IDE, making a custom GUI, exporting it and publishing it on a remote server) took less than two minutes. Then, making it available in the DAW is just a matter of adding one line in a configuration file.

The DAW includes a wide range of audio effects created with FAUST, such as a recreation of Eventide's famous Blackhole effect pedal, or Electro-Harmonix's Big Muff fuzz and Stone Phaser, for example, as well as numerous original effects covering the most classic needs: reverbs, modulation effects, stereo enhancers, distortions, etc. Several instruments of various types have also been integrated (flute, djembe, guitar, synthesizers).

2.3. FAUST WAMs for Optimal Performances in Host-Plugin Communications

Based on the faustwasm module, the FAUST distribution introduces a new script called faust2wam, a JavaScript tool that can generate self-contained FAUST WAMs within the Node.js environment or dynamically within browsers. Additionally, support has been added for polyphonic instruments and FAUST based spectral processors [8]. These new generation targets (web/wam2-ts, wam2-poly-ts, and wam2-fft-ts) are now available in the FAUST IDE Export window and are also used by the GUI Builder presented in the previous section.

During the WAM export and publishing process, the original FAUST code is compiled to WebAssembly, and the generated GUI is packaged as a Web Component. All generated files are placed in a single folder, which can be published online and downloaded. By default, this includes two different GUIs: the auto-generated default GUI (Figure 5) and the custom GUI, designed using the GUI Builder and leveraging widgets from the webaudiocontrols library (Figure 6). Two URIs are provided: one for the default GUI and the other for the custom GUI-based WAM.

More interesting in terms of performance is that the WebAssembly code runs inside an Audio Worklet, enabling custom DSP code execution. Audio Worklets were added to the Web Audio API in 2018, and FAUST was one of the first DSLs to support them as a target [9]. An Audio Worklet consists of two main parts: 1) the Audio Worklet Processor, where the core custom audio processing occurs. It's a class that extends AudioWorkletProcessor and processes audio in small chunks called frames this is where the FAUST WebAssembly code runs. 2) the Audio Worklet Node, which serves as the interface connecting the Audio Worklet Processor to the Web Audio API's audio graph. It bridges custom audio processing in the Worklet with the Web Audio API context.

More specifically, the faust2wam script uses WAMProcessor and WAMNode classes from the WAM SDK, which inherit from AudioWorkletProcessor and AudioWorkletNode, providing additional features. In particular, WAMProcessor supports high performance communication with a WAM host using Shared Array Buffers [3].

This approach eliminates the need to create events or cross the thread boundary between the GUI thread and the audio thread when a host needs to communicate with a WAM plugin, for parameter automation at the sample rate or for MIDI communication with a virtual instrument since both the host and plugin, based on Audio Worklets, have their processing parts running in the high-priority audio thread.

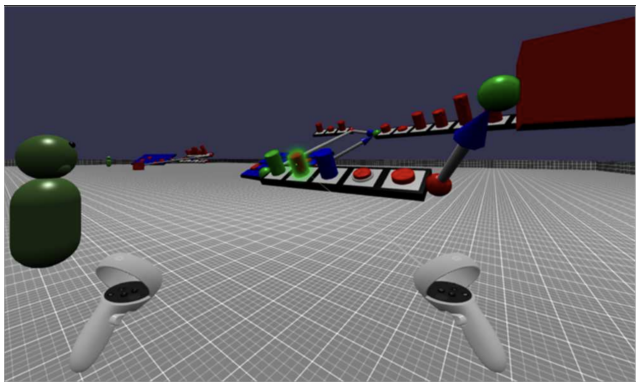

Figure 8: Multiple participants assemble 3D WAMs to build spatialized music installations.

3. FAUST BASED WEB AUDIO MODULES IN THE MUSICAL METAVERSE

3.1. WAM Jam Party, Making Music in an Immersive, Collaborative Environment

WAM Jam Party [10] is an immersive, collaborative application that runs in the Web Browser of VR headsets. This WAM-based application focuses on a persistent, real-time multi-participant immersive world for shared music creation, exploiting recent W3C web standards such as Web Audio, Web MIDI, WebXR, WebGL, WebGPU, WebAssembly, WebSockets, now implemented in the web browsers of the most common VR/XR headsets available on the market.

As stated in section 1, dozens of WAM plugins are now available through the WAM Community endpoint, comparable in quality and complexity to native applications [8]. Available WAMs include note generators such as: a piano roll, a programmable step sequencer, random note generators, chord generators, virtual instruments: synthesizers samplers, physical modeling of instruments (flute, clarinet, brass, djembe), audio effects (including all classics: flanger, chorus, reverb, distortion, fuzz, overdrive, etc.). The majority of the effects and instruments have been generated with the FAUST IDE.

With WAM Jam Party, users can connect to a URL using the VR headset web browser, and start building musical installation in the immersive world, by adding and connecting together note generators, virtual instruments and audio effects (Figure 8). Each of these component is an existing Web Audio Module.

The main inspiration was Sequencer Party [8], a collaborative online audio and MIDI editor, also based on WAMs. WAM Jam Party is a 3D adaptation of Sequencer Party concepts.

The first prototype of WAM Jam Party uses WAM introspection to build 3D interactive GUIs from existing 2D WAM plugins, BabylonJS for the 3D rendering of the immersive world, and a CRDT algorithm is used synchronizing incrementally the states of all the elements between clients. While client states are updated 3 times/s, the server saves seamlessly the ongoing session into a file every second, making the world persistent. Special care has been taken for generating the 3D representation of WAM plugins, using an optional JSON file for the mapping of the internal parameters into the 3D world. Each WAM is represented by default as a box with draggable cylinders, boxes or push buttons for its parameters

Figure 9: Big Muff fuzz, a FAUST-based WAM, and its 3D GUI.

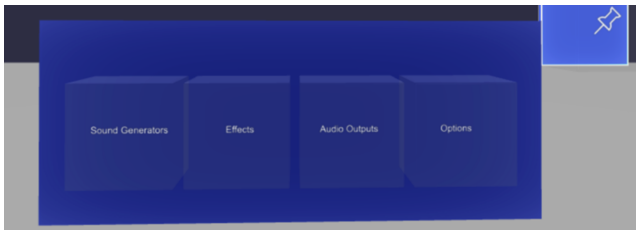

Figure 10: Main menu for adding WAMs in the 3D scene.

(Figure 9).

A dynamic menu made with the Mixed Reality Tool Kit (MRTK) available in BabylonJS, proposes a large set of WAMs enabling users to add elements in the 3D world (Figure 10). In addition, a green and a right sphere are located on the sides of the box, enabling users to connect WAMs together to form an audio graph (Figure 11). Connect a step sequencer to a synthesizer to some audio effects and you have a first basic music installation. Any element can be moved, oriented, or have its parameters adjusted using different controllers (see Figure 12).

In a multi participant session, each user can perform the same interactions (add, remove, move, rotate WAMs, connect and disconnect elements, adjust the parameters, move around), and modify other users' creations. The sound produced by each installation is spatialized and changes as you move your avatar in the 3D world.

Each user in the virtual environment is represented by a simple

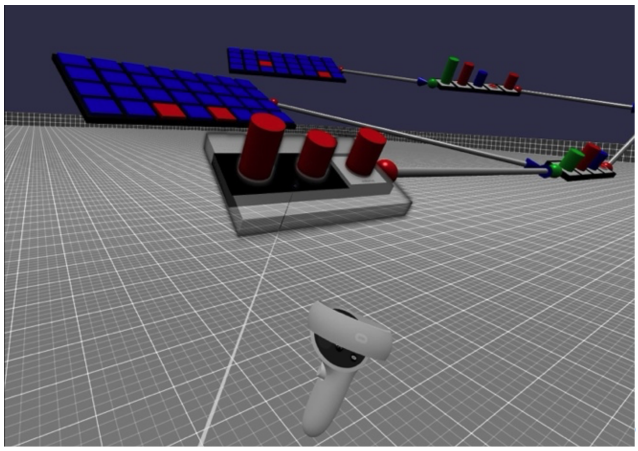

Figure 11: Example of a graph made of two note generators (step sequencers on the left), two instruments and audio effects.

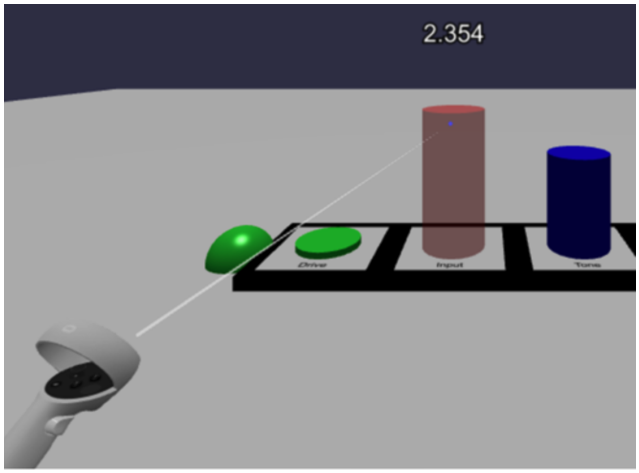

Figure 12: Parameters can be adjusted by clicking the controller trigger while aiming at a cylinder, and dragging it vertically.

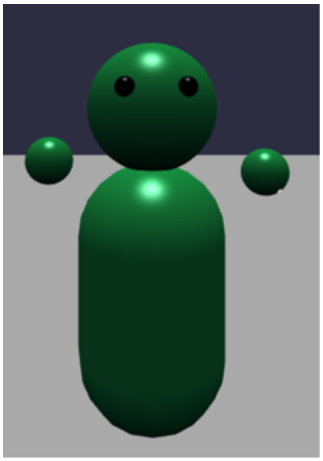

Figure 13: Avatar of a player.

Figure 14: A FAUST 2D GUI and its 3D version using 3D models of 2D widgets.

avatar consisting of a body, head, and two hands. The avatar's head orientation is directly mapped to the user's VR headset orientation, ensuring that the avatar's gaze direction aligns with the user's actual view within the virtual environment. The avatar's hands are positioned and oriented to match the user's controller movements. This visual representation provides a clear indication of the user's current actions, such as selecting WAMs or manipulating parameters.

3.2. How FAUST WAMs's 3D GUI Is Generated

Details about the UX design and interactions can be found in [10]. More interesting is the history of the first empiric tests conducted. FAUST allows us to declare basic user interface (UI) elements to control the parameters of a FAUST object [3]. After the compilation of a FAUST source code an abstract representation of the UI can be obtained for generating target dependant code. For example, the WAM GUI Builder uses it for generating an HTML based GUI. A first series of tests consisted in using this abstract representation for generating a 3D GUI, as shown in Figure 14. Unfortunately, this kind of 3D interfaces were not suited for a 3D manipulation in VR headsets. Users found many difficulties operating 3D knobs, sliders or switches with VR headset controllers. Also, for plugins with too many parameters, the 3D rendered view could be confusing with small labels and widgets.

Finally, the adopted solution that gave good results during user testing was to use ad hoc 3D shapes with interactions adapted to the VR usages. For example, 2D knobs and sliders become draggable cylinders or boxes, switches become a push button, etc. It also became rapidly obvious that as all original 2D elements could not be used in 3D (i.e a synthesizer with 60 parameters controlled by 2D knobs), the original abstract UI definition provided by FAUST was no more useful. Instead, the list and type of parameters provided by the WAM API of the generated plugin was sufficient for generating a 3D GUI on the fly. For example, the Big Muff pedal from Figure 9 has four parameters (volume, tone and sustain, plus the on/off switch), three of type float, and one of type boolean, so we can generate a 3D GUI with three draggable cylinders and a push button.

A configuration file also helped filtering unwanted parameter 3D controllers and customizing colours, shapes, labels of the wanted 3D parameters. A "convention over configuration" approach was used: a minimal configuration file with just the URI

[3] `https://faustdoc.grame.fr/manual/quick-start/#building-a-simple-user-interface`Building a Simple User Interface - Faust Documentation

of a WAM would lead to generating all parameters 3D controllers using default values, i.e red draggable cylinders for float parameters. But it is also possible to indicate different colors, labels, to hide some parameters by editing this configuration file. Furthermore, this parameter based generation works with all kind of WAMs, not only with FAUST based ones.

An interactive editor in which developers can enter an existing WAM URI and preview their 2D and 3D GUI (using a mouse of a VR headset) while customizing the different options (show/hide a parameter control, etc.) is under development, and could be integrated in the future in the FAUST GUI Builder.

4. CONCLUSIONS

This paper presents two innovative music applications utilizing Web Audio Modules plugins (WAMs), most of which are written in FAUST. We believe that FAUST and its online IDE offer one of the best approaches to developing Web Audio Modules. FAUST WAMs are at the heart of the applications presented: a web-based DAW and a 3D immersive application. While the first project shows that it is possible to recreate some of the most complex audio software on the Web (see "Why You Shouldn't Write a DAW" [4]), and plugins written in FAUST are a very important part of this, the second project opens up new possibilities and will certainly lead to the short-term availability of an editor for configuring and interactively generating reusable FAUST based 3D components.

5. ACKNOWLEDGMENTS

We would like to thank Antoine Vidal-Mazuy for his investment in WAM-Studio over the course of 2023 (he was its main coder and designer), and the team behind the Web Audio Modules, without which this DAW would never have existed. This work has been supported by the French government, through the France 2030 investment plan managed by the Agence Nationale de la Recherche, as part of the "UCA DS4H" project, reference ANR-17-EURE-0004.

6. REFERENCES

[1] Michel Buffa and Antoine Vidal-Mazuy, "Wam-studio, a digital audio workstation (daw) for the web," in *Companion Proceedings of the ACM Web Conference 2023*, 2023, pp. 543–548.

[2] Michel Buffa and Samuel Demont, "Can you DAW it Online?," in *IS2 2024 - IEEE International Symposium on the Internet of Sounds 2024 / 1st IEEE International Workshop on the Musical Metaverse (IEEE IWMM)*, Erlangen, Germany, Sept. 2024.

[3] Michel Buffa, Shihong Ren, Owen Campbell, Tom Burns, Steven Yi, Jari Kleimola, and Oliver Larkin, "Web audio modules 2.0: An open web audio plugin standard," in *Companion Proceedings of the Web Conference 2022*, 2022, pp. 364–369.

[4] David Rowland - ADC23, `https://www.youtube.com/watch?v=GMlnh6_9aTc`

[4] Luca Turchet, "Musical metaverse: vision, opportunities, and challenges," *Personal and Ubiquitous Computing*, vol. 27, no. 5, pp. 1811–1827, 2023.

[5] A. Gabriele A. Di Scipio A. Boem, M. Tomasetti and L. Turchet, "User needs in the musical metaverse: a case study with electroacoustic musicians," 2024.

[6] Michel Buffa, Pierre Kouyoumdjian, Quentin Beauchet, Yann Forner, and Michael Marynowic, "Making a guitar rack plugin-webaudio modules 2.0," in *Web Audio Conference 2022*, 2022.

[7] Shihong Ren, Stephane Letz, Yann Orlarey, Romain Michon, Dominique Fober, Michel Buffa, and Jerome Lebrun, "Using faust dsl to develop custom, sample accurate dsp code and audio plugins for the web browser," *Journal of the Audio Engineering Society*, vol. 68, no. 10, pp. 703–716, 2020.

[8] Michel Buffa, Shihong Ren, Tom Burns, Antoine Vidal-Mazuy, and Stéphane Letz, "Evolution of the web audio modules ecosystem," in *Web Audio Conference 2024*. Zenodo, 2024.

[9] Stéphane Letz, Yann Orlarey, and Dominique Fober, "Compiling faust audio dsp code to webassembly," in *Web Audio Conference*, 2017.

[10] Michel Buffa, Ayoub Hofr, and Dorian Girard, "Using Web Audio Modules for Immersive Audio Collaboration in the Musical Metaverse," in *IS2 2024 - IEEE International Symposium on the Internet of Sounds 2024*, Erlangen, Germany, Sept. 2024.

[11] Yann Orlarey, Stéphane Letz, and Dominique Fober, *New Computational Paradigms for Computer Music*, chapter "Faust: an Efficient Functional Approach to DSP Programming", Delatour, Paris, France, 2009.

[12] Julius O. Smith, "Signal processing libraries for Faust," in *Proceedings of Linux Audio Conference (LAC-12)*, Stanford, USA, May 2012.

[13] Albert Gräf, "pd-faust: An integrated environment for running Faust objects in Pd," in *Proceedings of the Linux Audio Conference (LAC-12)*, Stanford, USA, April 2012.

[14] Romain Michon and Julius O. Smith, "Faust-STK: a set of linear and nonlinear physical models for the Faust programming language," in *Proceedings of the 14th International Conference on Digital Audio Effects (DAFx-11)*, Paris, France, September 2011.