Mobile Music With the Faust Programming Language

Romain Michon,[1,2] Yann Orlarey,[1] Stéphane Letz[1], Dominique Fober[1] and Catinca Dumitrascu[1]

[1] GRAME-CNCM, Lyon (France)
[2] CCRMA, Stanford University, Stanford (USA)
michon@grame.fr

Abstract. The FAUST programming language has been playing a role in the mobile music landscape for the past ten years. Multiple tools to facilitate the development of musical smartphone applications for live performance such as `faust2ios`, `faust2android`, `faust2api`, and `faust2smartkeyb` have been implemented and used in the context of a wide range of large scale musical projects. Similarly, various digital musical instruments leveraging these tools and based on the concept of augmenting mobile devices have been created. This paper gives an overview of the work done on these topics and provide directions for future developments.

Keywords: FAUST, Mobile Music, Digital Lutherie

1 Introduction

The field of mobile music has been active for the past fifteen years [1]. It started with early experiments on programmable smartphones around 2004 [2, 3] but it really took off in 2007 when the iPhone was released and smartphones started to spread out to quickly become a standard [4]. The FAUST[3] project [5] through its core developer team at GRAME-CNCM[4] involved itself in this action in 2010 with initial experiments on running FAUST programs on iOS devices. Since then, a panoply of tools to generate standalone smartphone applications and audio engines for different mobile platforms (i.e., Android and iOS) have been developed and used as part of a wide range of musical and pedagogical projects.

In this paper, we give an overview of the work that has been done around mobile music in the context of the FAUST programming language. We present `faust2ios`, `faust2android`, `faust2api`, and `faust2smartkeyb` which are tools that can be used to create musical mobile apps at a high level using FAUST. Work carried out on the idea of augmenting mobile devices with passive and active elements to turn them into specific musical instruments is described. An

[3] `https://faust.grame.fr` (All URLs presented in this paper were verified on May 2, 2019.)
[4] `http://www.grame.fr`

overview of various musical projects such as *SmartFaust*, *SmartMômes*, and *Geek-Bagatelles* is presented. Finally, we talk about current developments and future directions for this type of work.

2 `faust2ios`

Pushed by the interest around mobile music in the early 2010s (see §7), we worked at GRAME-CNCM on a tool to convert FAUST programs into ready-to-use iOS applications: `faust2ios`. As any other FAUST "architecture,"[5] the user interface of such apps is based on the UI description provided in the FAUST code, and is therefore typically made out of sliders, knobs, buttons, groups, etc.

Figure 1 presents a screenshot of sfCapture,[6] an app made with `faust2ios` as part of the *SmartFaust* project (see §7.1).

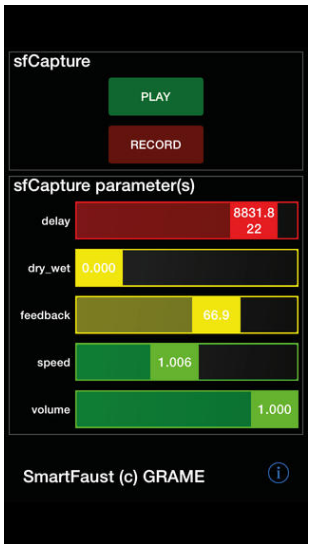

Fig. 1. Screen-shot of sfCapture, an App Made with `faust2ios`.

`faust2ios` works as a command line tool taking a FAUST program as its main argument and producing in return either a ready-to-install iOS app or the Xcode project corresponding to this app. For example, running the following command in a terminal:

```
faust2ios myFaustProgram.dsp
```

[5] Architectures in the FAUST vocabulary refer to wrappers allowing to turn a FAUST program into a specific object such as standalone desktop program, an audio plug-in, a smartphone app, an audio engine for a specific platform, etc.

[6] `https://itunes.apple.com/us/app/sfcapture/id799532659?mt=8`

will produce an iOS app corresponding to the FAUST program implemented in `myFaustProgram.dsp`.

Various features can be added to the generated app such as MIDI, OSC and polyphony support simply by using specific flags (options) when running `faust2ios`. Regular FAUST options are also available to generate parallelized DSP[7] code, change sample resolution, etc. Any parameter of a FAUST program can be assigned to a specific axis of a built-in motion sensor (i.e., accelerometer, gyroscope, etc.) of the smartphone simply by using metadata. Complex non-linear mappings can be implemented using this mechanism.[8]

Implementing `faust2ios` was relatively straightforward since the FAUST compiler can generate C++ code and that iOS applications can be implemented in Objective-C which allows for the direct use of C++.

3 `faust2android`

Motivated by the success of `faust2ios` (see §2) among composers and developers at GRAME-CNCM, we started the development of a similar system for the Android platform in 2013 [6]. This proved to be way more challenging than we anticipated, mostly because Android was never designed with real-time audio applications in mind. First, the fact that JAVA is used as the preferred programming language to develop Android apps was problematic since it doesn't perform well in the context of real-time DSP. Hence, the audio portion of the app must be implemented in C++ and the higher level elements in JAVA. This implies the use of wrappers between these two languages which is not straightforward to implement. Another issue with Android was that despite the use of low-level native code for the DSP portion of the app, audio latency used to be dreadful around 2013 (greater than 200ms), discarding any potential use in a musical context.

Despite these difficulties, the first version of `faust2android` was released in the first quarter of 2013 [6]. It had similar features than `faust2ios` (see §2) and worked in a very similar way as a command line tool. Figure 2 presents a screenshot of an app generated with `faust2android`.

As time passed and the market for real-time audio applications on smartphone grew up, Google slowly addressed the audio latency issue of Android and acceptable performances matching that of the iOS platform (less than 20ms) were achieved by 2016. Additionally, Google released in 2017 a new C++ API for real-time audio on Android which significantly simplified the design of apps involving this kind of element.[9]

On the `faust2android` front, various new features were added to replace the standard FAUST user interface of Android apps by advanced interfaces more usable in a musical context such as piano keyboards, X/Y controllers, etc. [7] These opened the path to `faust2smartkeyb` which is presented in §5.

[7] *Digital Signal Processing*

[8] `https://faust.grame.fr/doc/manual#sensors-control-metadatas`

[9] This new API is currently not used by `faust2android`, which predates its release.

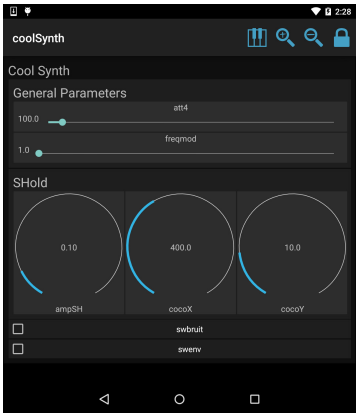

Fig. 2. Example of Interface Generated by `faust2android` Containing Groups, Sliders, Knobs and Checkboxes.

4 `faust2api`

With `faust2ios` (see §2) and `faust2android` (see §3) appeared the need for a generic system to generate audio engines with a high-level API similar across languages (i.e., JAVA, C++, etc.) using FAUST: `faust2api` [8]. The main goal of this tool was to offer iOS and Android developers with little background in audio DSP a simple way to generate ready-to-use engines for sound synthesis and processing.

`faust2api` is a command line tool working in a similar way than `faust2ios` and `faust2android`. It takes a FAUST program as its main argument and accept more or less the same options than `faust2ios` and `faust2android`. The format of the generated engines varies between platforms but the same API can be used to configure and control it.

`faust2api` was released in 2017 and was used as the basis for `faust2smartkeyb` (see §5). `faust2ios` and `faust2android` were simplified by using `faust2api` to carry out real-time audio DSP tasks. Because of its large success among developers, the concept of `faust2api` was spread to most of FAUST's targets and it can now be used to generate audio engines for desktop applications, plug-ins, etc.

5 `faust2smartkeyb`

With the latest developments of `faust2android` (see §3), we started exploring the idea of replacing the standard FAUST user interface made out of sliders, buttons, groups, etc. with more advanced interfaces, better adapted to a use in a live music performance context and to touch-screens. We extended this idea with SMARTKEYBOARD which is a highly configurable keyboards matrix where keys can be seen both as discrete buttons and continuous X/Y controllers. For

example, a keyboard matrix of size 1x1 (a single keyboard with a single key) will fill up the screen which can then be used as a multi-touch X/Y controller.

This type of interface is available as part of the `faust2smartkeyb` command line tool [9] which allows us to turn a FAUST program into an iOS or an Android app with a SMARTKEYBOARD interface. The interface can be configured directly from the FAUST code using a metadata. For example, the following program:

```
declare interface "SmartKeyboard{
  'Number of Keyboards':'2'
}";
import("stdfaust.lib");
f = nentry("freq",200,40,2000,0.01);
g = nentry("gain",1,0,1,0.01);
t = button("gate");
envelope = t*g : si.smoo;
process = os.sawtooth(f)*envelope <: _,_;
```

implements a synthesizer based on a sawtooth wave oscillator and a simple exponential envelope controlled by two parallel piano keyboards on the touch-screen (see Figure 3). Connection between the interface and the DSP part is carried out by the use of standard parameter names. Hence, `freq` is automatically associated to the pitch on the keyboard, `gain` to velocity, and `gate` to note-on/off events.[10]

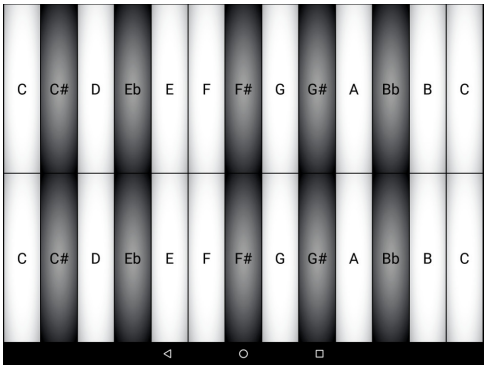

Fig. 3. Simple SMARTKEYBOARD Interface.

Complex behaviors can be implemented to handle polyphony, monophony (e.g., voice stealing, priority to upper or lower keys, etc.), and continuous pitch control (e.g., quantization, "pitch rounding" to be in tune and allow for vibrato and glissandi to be performed at the same time, etc.).

[10] `https://faust.grame.fr/doc/manual#standard-polyphony-parameters`

In the following example, a completely different app is implemented where a single key on a single keyboard is used to control a simple synthesizer producing a constant sound (no key on/off):

```
declare interface "SmartKeyboard{
  'Number of Keyboards':'1',
  'Max Keyboard Polyphony':'0',
  'Keyboard 0 - Number of Keys':'1',
  'Keyboard 0 - Send Freq':'0',
  'Keyboard 0 - Static Mode':'1',
  'Keyboard 0 - Piano Keyboard':'0',
  'Keyboard 0 - Send Numbered X':'1',
  'Keyboard 0 - Send Numbered Y':'1'
}";
import("stdfaust.lib");
//////// parameters ////////
x0 = hslider("x0",0.5,0,1,0.01) : si.smoo;
y0 = hslider("y0",0.5,0,1,0.01) : si.smoo;
y1 = hslider("y1",0,0,1,0.01) : si.smoo;
q = hslider("q[acc: 0 0 -10 0 10]",30,10,50,0.01) : si.smoo;
//////// mapping ////////
impFreq = 2 + x0*20;
resFreq = y0*3000+300;
//////// putting it together ////////
process = os.lf_imptrain(impFreq) : fi.resonlp(resFreq,q,1) :
ef.cubicnl(y1,0)*0.95 <: _,_;
```

Here, `x0` corresponds to the X position of the first finger to touch the screen, `y0` its Y position and `y1` the Y position of the second finger to touch the screen. The q parameter of the resonant lowpass filter is controlled by the X axis of the built-in accelerometer with a linear mapping.[11]

An exhaustive list of the SmartKeyboard configuration keywords can be found in its corresponding documentation[12] and tutorials demonstrating how to implement various types of behaviors can be found on the Faust tutorial page.[13]

6 Digital Lutherie and Smartphones

In parallel of the development of the various tools presented in the previous sections, an important work has been carried out at GRAME-CNCM and at CCRMA[14] (Stanford University) around the concept of augmenting mobile devices to implement advanced musical instruments. The core idea of this project

[11] `https://faust.grame.fr/doc/manual#sensors-control-metadatas`
[12] `https://ccrma.stanford.edu/~rmichon/smartKeyboard/`
[13] `https://ccrma.stanford.edu/~rmichon/faustTutorials/#making-faust-based-smartphone-musical-instruments`
[14] *Center for Computer Research in Music and Acoustics*

was to use mobile devices as the platform for computing and sound synthesis/processing of physical Digital Musical Instruments (DMIs) built around this type of device. Two kinds of "smartphone augmentations" were developed in this context:

- **passive augmentations** [10] based on digitally fabricated elements leveraging existing sensors on the device, allowing us to hold it in a specific way, or modifying the acoustical properties of its built-in speaker and microphone, etc.,
- **active augmentations** [11] implying the use of additional sensors connected to the mobile device through the use of a microcontroller, etc.

Figure 4 presents an overview of the type of passive augmentations that have been explored as part of this project.

Fig. 4. A Few Examples of Passive Smartphone Augmentations.

The BladeAxe [12] is a good example of an active mobile device augmentation. It provides a plucking system based on piezo to capture sound excitations created by the performer on plastic tines to drive waveguide physical models running on an iPad in an app implemented with `faust2smartkeyb` (see §5). This allows for a very natural and intuitive control of the plucking since the sound of each excitation is different.

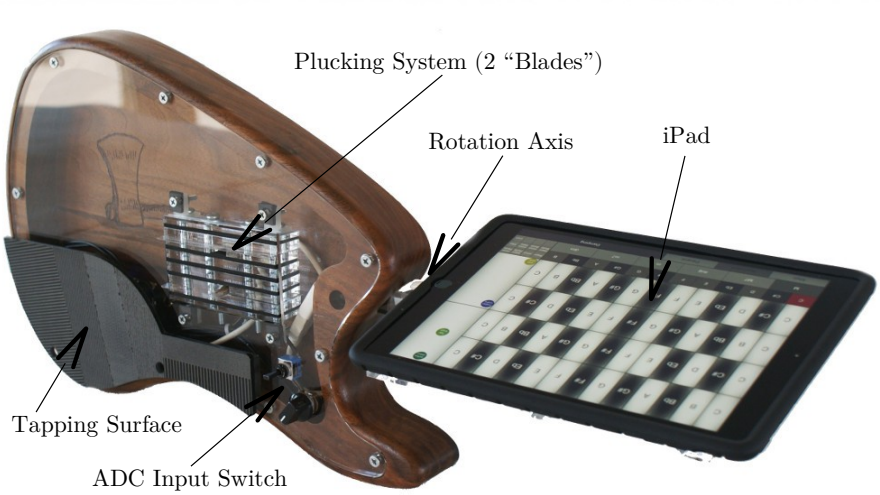

Fig. 5. The BladeAxe.

7 Performances and Pedagogy

7.1 From SmartFaust to Geek-Bagatelles

`faust2ios` and `faust2android` (see §2,3) served as the platform for the development by GRAME-CNCM of a series of large scale musical projects involving the use of smartphones as early as 2013. The first of them – *SmartFaust* – was a participatory concert for smartphones which was commissioned to composer Xavier Garcia for the 2014 Biennale Musique en Scène and funded by the INEDIT ANR[15] project.

Garcia worked closely with a developer/computer music assistant (Christophe Lebreton) to the development of a series of iOS and Android applications using `faust2ios` and `faust2android`. The instruments/applications and their corresponding musical pieces were co-written simultaneously. Another remarkable feature of these instruments is the lack of graphical interface: only motion sensors were used. The performer never needs to look at the phone to play it: everything is done between the hand and the ear!

The fruit of this work was performed for the first time at the Subsistances in Lyon (France) in March 2014. The concert was organized in two sections: the performance of three pieces for “chorus” of Smartphones and soloists, and then a fourth piece involving the audience.

After this first performance, *SmartFaust* met a large success and started an Asian tour with participatory concerts and workshops that were organized in June 2015 in Wuhan, Hong-Kong and Chengdu. In this context, new pieces for

[15] *Agence Nationale de Recherche*: French National Research Agency

Fig. 6. Left: *SmartFaust* Performance at the Subsistances in Lyon (France) on March 16, 2014. Right: *SmartMômes* Performance at the Saint-Étienne (France) City Hall in March 2016.

the *SmartFaust* apps corpus were written, in particular by composer Qin Yi in Shanghai.

The original *SmartFaust* project also gave birth to other performances such as:

- *SmartFaust on Air* at the 2015 Design Biennale in Saint-Etienne (France),
- participatory concerts in the TGV[16] in partnership with the SNCF,[17]
- sound installations with the *Smartland Divertimento* piece presented at the Museum of the Confluences at the 2016 Biennale Musique en Scène (Lyon, France).

The latter, proposed by Christophe Lebreton and composer Stéphane Borrel, is like a bush of smartphones that communicate with each other and sparkle independently, a bit like fireflies.

The most recent project of this series was created as part of the ONE project (Orchestra Network for Europe) with the Picardy Orchestra (France). It was finalized in September 2014, approved by the European Commission in April 2015, and finally resulted in a commission to composer Bernard Cavanna for a piece for orchestra and smartphones: *Geek-bagatelles, introspections sur quelques fragments de la IXe symphonie de Beethoven*. The performance was premiered on November 20, 2016 by the Picardy Orchestra at the Paris Philharmonie. It combined a chorus of 20 smartphones and an orchestra of 38 musicians. The audience participated as well thanks to the *Geek-Bagatelles* app on their smartphone.

The performance was a success and a tour was initiated in the countries part of the ONE network, each time with a new orchestra and a new amateur smartphones chorus formed for the occasion.

[16] High speed train system in France
[17] French National Railway Company

Fig. 7. *Geek-Bagatelles* Performance at the Paris Philharmonie on November 20, 2016.

7.2 SmartMômes

After the initial performance of *SmartFaust*, Mômeludies which is a nonprofit promoting the creation and the diffusion of new musics towards kids commissioned composer Xavier Garcia a new piece for smartphones: *SmartMômes* (see Figure 6). They asked him to teach a series of workshops on this topic in multiple middle schools as well. As a publisher, Mômeludies also published the score of *SmartMômes*.

Because of the interest around the pedagogical aspect of this approach, GRAME-CNCM organized a series of *SmartFaust* workshops during which the FaustPlayground[18] was used to create musical smartphone apps using Faust at a very high level with a Graphical User Interface.

8 Current and Future Directions

The various tools and technologies presented in the previous sections of this paper reached a certain level of maturity and are now broadly used at GRAME-CNCM and elsewhere. They significantly contributed to the success of most of the recent musical productions of our center thanks to their universal aspect and to their tangibility. Performing with independent standalone and tangible DMIs is quite appealing in a world where everything tends to become completely virtual. Hence, while we keep adding new features to our toolkit for mobile development, we also started exploring new paths to work with embedded systems for low latency/high quality audio. Indeed, microcontrollers are now powerful enough to run complex sound synthesis and processing algorithms in real-time. Similarly, embedded computers such as the Raspberry Pi (RPI) when used without operating system ("bare-metal"), FPGAs[19], GPUs[20] and other low-level

[18] `https://faust.grame.fr/faustplayground`
[19] *Field Programmable Gate Arrays*
[20] *Graphical Processor Units*

DSPs offer new possibilities to create embedded/embodied instruments at a low cost and with un-paralleled performances. While we currently investigate the use of FAUST on FPGAs and bare-metal RPI, FAUST targets have already been implemented for microncontrollers [13] and DSPs such as the SHARC Audio Module.[21]

These new developments recently allowed us to create a new programmable musical instruments: the *Gramophone* (see Figure 8) that we plan to use for pedagogical purpose and for future musical productions at GRAME-CNCM. Based on Teensy 3.6 board[22] for sensor acquisition and sound synthesis, it can be powered by its internal battery for about ten hours, it is equipped with a powerful speaker and amplifier, and it hosts a wide range of sensors (i.e., accelerometer, gyroscope, compas, force sensing resistors, knobs, buttons, photo-resistor, etc.) that can be assigned to FAUST parameters directly from the FAUST code using metadata. It is better than a smartphone in many ways as it offers more affordances and it is more flexible and much louder. While it is still being developed, we plan to release the first version in Fall 2019.

Fig. 8. The Gramophone.

9 Conclusion

After fifteen years, mobile music has reshaped the computer music landscape partly by reintroducing the concept of standaloneness/independence in DMIs and by making this type of instrument more approachable by the general public. FAUST played a role in this revolution by providing high level tools to develop musical apps for live performance. GRAME-CNCM took advantage of these technologies to place mobile music at the heart of various large scale musical

[21] `https://wiki.analog.com/resources/tools-software/sharc-audio-module/faust`

[22] `https://www.pjrc.com/store/teensy36.html`

productions/projects. By offering the possibility to easily create orchestras of DMIs, mobile music opened the way to new paths for creation that we intend to keep exploring by developing new programmable instruments taking advantage of recent developments in embedded real-time signal processing such as the Gramophone.

References

1. Gaye, L., Holmquist, L.E., Behrendt, F., Tanaka, A.: Mobile Music Technology: Report on an Emerging Community. In: Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-06), Paris (2006)
2. Tanaka, A.: Mobile Music Making. In: Proceedings of the International Conference on New Interfaces for Musical Expression (NIME04), National University of Singapore (2004)
3. Schiemer, G., Havryliv, M.: Pocket Gamelan: Tuneable Trajectories for Flying Sources in Mandala 3 and Mandala 4. In: Proceedings of the International Conference on New Interfaces for Musical Expression (NIME06), Paris (2006)
4. Wang, G.: Ocarina: Designing the iPhone's Magic Flute. Computer Music Journal, 38(2), 8–21 (2014)
5. Orlarey, Y., Letz, S., Fober, D.: New Computational Paradigms for Computer Music, chapter "Faust: an Efficient Functional Approach to DSP Programming." Delatour, Paris (2009)
6. Michon, R.: faust2android: a Faust Architecture for Android. In: Proceedings of the 16th International Conference on Digital Audio Effects (DAFx-13), Maynooth, Ireland (2013)
7. Michon, R., Smith, J.O., Orlarey Y.: MobileFaust: a Set of Tools to Make Musical Mobile Applications with the Faust Programming Language. In: Proceedings of the International Conference on New Interfaces for Musical Expression, Baton Rouge (2015)
8. Michon, R., Smith, J.O., Letz, S., Chafe C., Orlarey, Y.: faust2api: a Comprehensive API Generator for Android and iOS. In: Proceedings of the Linux Audio Conference (LAC-17), Saint-Étienne, France (2017)
9. Michon, R., Smith, J.O., Chafe, C., Wang, G., Wright, M.: faust2smartkeyb: a Tool to Make Mobile Instruments Focusing on Skills Transfer in the Faust Programming Language. Proceedings of the International Faust Conference (IFC-18), Mainz, Germany (2018)
10. Michon, R., Smith, J.O., Wright, M., Chafe, C., Granzow, J., Wang, G.: Passively Augmenting Mobile Devices Towards Hybrid Musical Instrument Design. In: Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-17), Copenhagen (2017)
11. Michon, R., Smith, J.O., Wright, M., Chafe, C., Granzow, J., Wang, G.: Mobile Music, Sensors, Physical Modeling, and Digital Fabrication: Articulating the Augmented Mobile Instrument. Applied Sciences, 7(12), 1311 (2017)
12. Michon, R., Smith, J.O., Wright, M., Chafe, C.: Augmenting the iPad: the BladeAxe. In: Proceedings of the International Conference on New Interfaces for Musical Expression (NIME-16), Brisbane, Australia (2016)
13. Michon, R., Orlarey, Y. Letz, Y., Fober D.: Real Time Audio Digital Signal Processing With Faust and the Teensy. In: Proceedings of the Sound and Music Computing Conference (SMC-19), Malaga, Spain (2019) – Paper not published yet but accepted to the conference