FAUST2SMARTPHONE: A GENERATOR FOR MUSICAL MOBILE APPLICATION

Weng Ruolun

Dept. of Music Engineering
Shanghai Conservatory of Music
Shanghai, China
allen1991shcm@gmail.com

1. OVERVIEW

We introduce *faust2smartphone,* a tool to generate editable musical mobile application projects using the Faust programming language. *faust2smartphone* works as an extension of *faust2api.* Faust DSP objects can be easily embedded as a high level API so that developers can access various functions and elements across different mobile platforms.
Mobile devices are increasingly used as musical instruments in the context of interactive performances and installations.
Current real-time audio or DSP APIs provided by common development environments are written in different programming languages and not easily approachable by composers and sound engineers of interactive electronic music.
The Faust architectures and *faust2api* allow us to focus more on sound design in Faust. The Faust distribution already comes with a comprehensive series of tools to generate mobile applications such as *faust2ios, faust2android,* and *faust2smartkeyb, so* why we create a new one?
We already use *faust2ios* and *faust2android* in the framework of the *SmartFaust* project to generate applications with standard Faust user interfaces (e.g., sliders, buttons, etc). *faust2smartkeyb* is specifically designed to make smartphone-based musical instruments with a keyboard interface. It also requires the use of a specific metadata declaration. These two sets of tools are relatively closed environments, making customization and integration with other frameworks hard.
On the other hand, *faust2api* is a generic tool to generate a set of API files for different platforms including mobile devices. However, it only creates a raw files package with one C++ and one header file that needs to be re-generated each time a new project is started from scratch.
We wanted to extend the capabilities of *faust2api* by adding more specific functions to facilitate the development of musical mobile applications.
In this paper, we present *faust2smartphone* which provides the same features on iOS and Android (Windows phone are not supported yet).
For now, *faust2smartphone* is a separate branch and maintained on Github. Normally it should work with the latest version of the Faust official branch. You can find all the source of this project on https://github.com/RuolunWeng/faust2smartphone.git.

As illustrated in Figure 1, *faust2smartphone* inherits from *faust2api*.

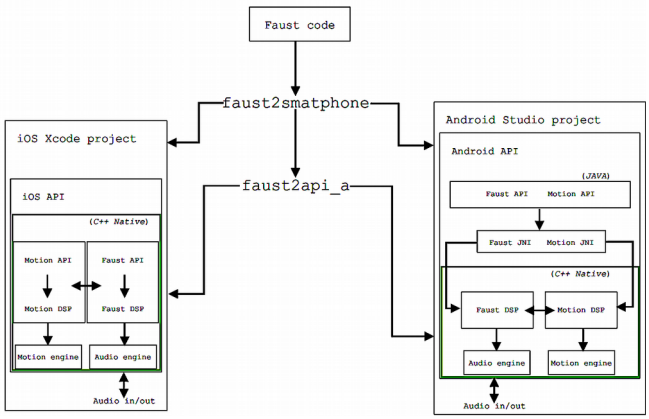

Figure 1: *Implementation of faust2smartphone.*

1.1. Simple mode

When *simple mode* is used, faust2api is automatically called and copies the generated files (e.g., *DspFaust.cpp* and *DspFaust.h*) to a template XCode or Android Studio project. That is what we call an "edit-ready" project, which bears the same name as the Faust code, embeds the Faust audio DSP engine and is ready to be used. This project is just a workplace to start, all the *faust2api* functions can be used and custom interfaces can be designed.

1.2. Motion mode

This special mode is based on *motion.lib* and can be used as a platform to prototype musical applications involving motion gestures. *motion.lib* uses the accelerometer, gyroscope, and rotation matrix signals provided by smartphones as an input. The output is the result of sensor's processing.
In this mode (see Figure 2), we have two DSP engines:

- DspFaust, which is the same as in simple mode and that is used for audio signal processing;
- DspFaustMotion, which is the pre-compiled engine for our motion processing.

This is an engine modified from the simple DspFaust structure in order to process motion rather than audio data, and hence is not driven by the audio driver like CoreAudio in iOS. The engine

runs at the sample rate of DspFaust divided by the buffer size of the DspFaust and a block size of 1. We think that this is enough for motion. Using audio processing rate for the sensors seems too expensive, that's also why we don't import motion.lib directly in the Faust code.

How to retrieve the sensor values and get the corresponding result from the DspFaustMotion engine? We decided to provide access to the inputs and outputs of the motion engine, which means that we can send the sensor's value and get the result through two new functions: `setInput()` and `getOutput()`.

Next question is how we check in the motion.lib which function the Faust code wants to call and how to affect the right controller. The first thing we need to do is a declaration in the metadata of the controller:

```
toto=hslider("toto[motion:ixp]",0,0,1,0.01)
```

"motion" is the keyword, followed by which function you want to call in the motion.lib.

By default, all the processes in motion.lib are muted to save CPU consumption; only if the program detects that you call the function, it will activate the corresponding process and affect this controller with the result calculated.

We have some other reserved keywords declarations :

```
toto=checkbox("touchgate");
tata=nentry("cue",0,0,5,1);
titi=hslider("screenx/screeny",0,0,1,0.01);
```

This suite works with a sub-mode of motion mode, we call it cueManager. We provide a simple interface for this mode to deal with the code composed with different cues. To active cueManager, you just need to add `-cuemanager` in the command line.

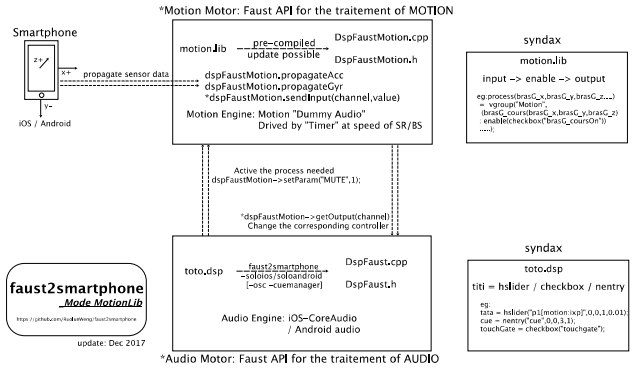

Figure 2: *Motion mode.*

1.3. Plug-in mode

This mode is not an audio VST plug-in generator. The idea is to have an engine which uses Faust code to process non-audio signals, which is simplified version of DspFaustMotion from the motion mode. We can use this for any parameter of videos or images.

For example, if we want to use the amplitude of an oscillator to control the alpha of the screen, the output of `os.osci(0.5)` can be connected to the alpha parameter. The user then needs to configure this manually in the script using the methods we already provide: `render()` and `getOutput()`.

2. APPLICATION

faust2smartphone has already been used in these productions:

"Audio Guide" is an application designed by Christophe Lebreton and me for blind person to experience a special sound map in the project created by GRAME and La Maison des Aveugles à Lyon. Based on the sound processing generated by *faust2smartphone,* we combine another framework in iOS, CoreLocation/CLBeacon for the Beacon part, which allows Bluetooth devices to broadcast or receive tiny and static pieces of data within short distances. Check the introduction online: http://www.grame.fr/events/carte-sonore-de-traces-en-traces.

A brand new creation named "Virtual Rhizome" at 2018 Biennale of Music in Lyon, created by Vincent-Raphaël Carinola and Christophe Lebreton, a solo performer armed by two smartphones, is diving into a virtual sound architecture that he must dispense and that changes every moment. We use the motion mode in *faust2smartphone,* with an interface modified from the cueManager sub-mode. You can check a video clip online: https://www.youtube.com/watch?v=cGZB44KI9Y0.

"sfPivoine" is a mobile application which I created for a participative performance selected by International Computer Music Conference(ICMC) 2018, "Pivone, for Pipa, Electronic music, Kunqu Opera and Smartphones of public". The spectators could have an immersive and augmented experience with their participations. This application merges the project generated by *faust2smartphone* and the simple audio-visual part using some animation and the Augmented Reality. The application is both available at App Store and Google Play.

3. REFERENCES

[1] R. Michon, J. Smith, S. Letz, C. Chafe and Y. Orlarey, " faust2api: a Comprehensive API Generator for Android and iOS," in Proceedings of the *Linux Audio Conference (LAC-17)*, Saint-Etienne, France, 2017.

[2] R. Michon, J. O. Smith, C. Chafe, M. Wright and G. Wang, "Nuance: Adding Multi-Touch Force Detection to the iPad," in *Proceedings of the Sound and Music Computing Conference (SMC-16)*, Hamburg, Germany, 2016.

[3] R. Michon, J. Smith and Y. Orlarey, "New Signal Processing Libraries for Faust," in *Proceedings of the Linux Audio Conference (LAC-17)*, Saint-Etienne, France, 2017.

[4] R. Michon, J. O. Smith and Y. Orlarey, "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, USA, 2015.

[5] R. Michon, "Faust2android: a Faust Architecture for Android," in *Proceedings of the 16th International Conference on Digital Audio Effects (DAFx-2013),* National University of Ireland, Maynooth, Ireland, Sept. 2-5, 2013.

[6] 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.

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