Interactive Mobile Musical Application using faust2smartphone

WENG Ruolun
Shanghai Conservatory of Music
20 Fenyang Road, Shanghai, China
allen1991shcm@gmail.com

ABSTRACT

We introduce faust2smartphone, a tool to generate an edit-ready project for musical mobile application, which connects Faust programming language and mobile application's development. It is an extended implementation of faust2api. Faust DSP objects can be easily embedded as a high level API so that the developers can access various functions and elements across different mobile platforms. This paper provides several modes and technical details on the structures and implementation of this system as well as some applications and future directions for this tool.

Author Keywords

faust, musical mobile application, motion processing

CCS Concepts

• **Applied computing → Sound and music computing**; Performing arts; •**Software and its engineering → Software notations and tools → Development frameworks and environments → Application specific development environments**

1. BACKGROUND

Mobile devices are increasingly used as musical instruments in the context of interactive performances and installations. Current real-time audio or DSP (Digital Signal Processing) API (Application Programming Interface) provided by common development environments are written in different programming languages and not easily approachable by composers and sound engineers of interactive electronic music.

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.

1.1 Faust and faust2api

Faust[1] (Functional Audio Stream) is a functional programming language for sound synthesis and audio processing with a strong focus on the design of synthesizers, musical instruments, audio effects, etc. Faust targets high-performance signal processing applications and audio plug-ins for a variety of platforms and standards. The core component of Faust is its compiler. It allows to "translate" any Faust DSP specification to a wide range of non-domain specific languages such as C++, C, JAVA, JavaScript, LLVM bit code, WebAssembly, etc. In this regard, Faust can be seen as an alternative to C++ but is much simpler and intuitive to learn.

NIME'20, July 21-25, 2020, Royal Birmingham Conservatoire, Birmingham City University, Birmingham, United Kingdom.

Thanks to a wrapping system called "architectures," codes generated by Faust can be easily compiled into a wide variety of objects ranging from audio plug-ins to standalone applications or smartphone and web apps, etc. If you are the users of other programming languages such as Csound, Max, PureData, SuperCollider, and SOUL, Faust also provides the bridge linking to them.

faust2api[2], is a tool to generate custom DSP engines for Android and iOS using the Faust programming language. Faust DSP objects can easily be turned into MIDI-controllable polyphonic synthesizers or audio effects with built-in sensors support, etc. The various elements of the DSP engine can be accessed through a high-level API, made uniform across platforms and languages. At its highest level, faust2api is a command line program taking a Faust code as its main argument and generating a package containing a series of files implementing the DSP engine. Various flags can be used to customize the API. The only required flag is the target platform.

The goal of the faust2api is to provide a tool to easily generate custom APIs based on one or several Faust objects. On one hand, Faust DSP libraries implement hundreds of open source DSP algorithms that can be turned into C++, C, JAVA, JavaScript and LLVM bit code and embedded in your applications. On the other hand, Faust C++ libraries can carry out a wide range of tasks going from connecting Faust DSP objects to a specific audio engine (CoreAudio, OpenSL/ES, Alsa, JACK, etc.) or adding MIDI and polyphony support, sensor data handling, etc. to the same object. Most major Faust targets are supported: iOS, Android, OSX CoreAudio, ALSA, JACK, PortAudio, RTAudio, openFrameworks, JUCE.

1.2 Why using faust2smartphone

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 use faust2ios and faust2android in the framework of the "SmartFaust" project to generate mobile 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 to define the keyboard information. The SmartKeyboard UI allows to implement a wide range of controllers (basic keyboards, isomorphic keyboards, pads, X/Y controllers, etc.) on a touch-screen and can be configured directly in the Faust code. These two sets of tools are relatively closed environments, faust2ios and faust2android are more for general purposes, we take these two frameworks as the fundamental projects for Faust mobile applications, because they make it easy to quickly test your Faust code with a basic controllable interface. faust2smartkeyb is more oriented for the keyboard performance implementation. On the other hand, these specificity makes the customization and integration with other frameworks hard. faust2api is a generic tool to generate a set of API files for different platforms including mobile devices. However, it only creates a raw file package with one C++ and one header file that needs to be re-generated each time a new project is started from scratch. Comparing the standard audio signal processing workflow in JUCE and openFrameworks, faust2api help us to facilitate the sound programming and keep the possibility to integrate other third-party addons from their environments.

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 phones 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

2. OVERVIEW

Followed by the installation instruction of Faust and faust2smarphone, you are ready to explore the function by simply taping in your terminal "faust2smartphone -help" for the details. As faust2smartphone is designed for iOS and Android, "-ios, -iosmotion, -iosplugin" and "-android, -androidmotion, -androidplugin" will guide you to the target. As illustrated in Figure 1, faust2smartphone inherits from faust2api, so almost all the options for mobile systems are ready to be called, including: "-oscall/-oscalias" will activate the OSC (Open Sound Control) interface; "-soundfile" to active libsndfile support.

2.1 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.

2.2 Motion Mode

As illustrated in Figure 2, 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 , 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:

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

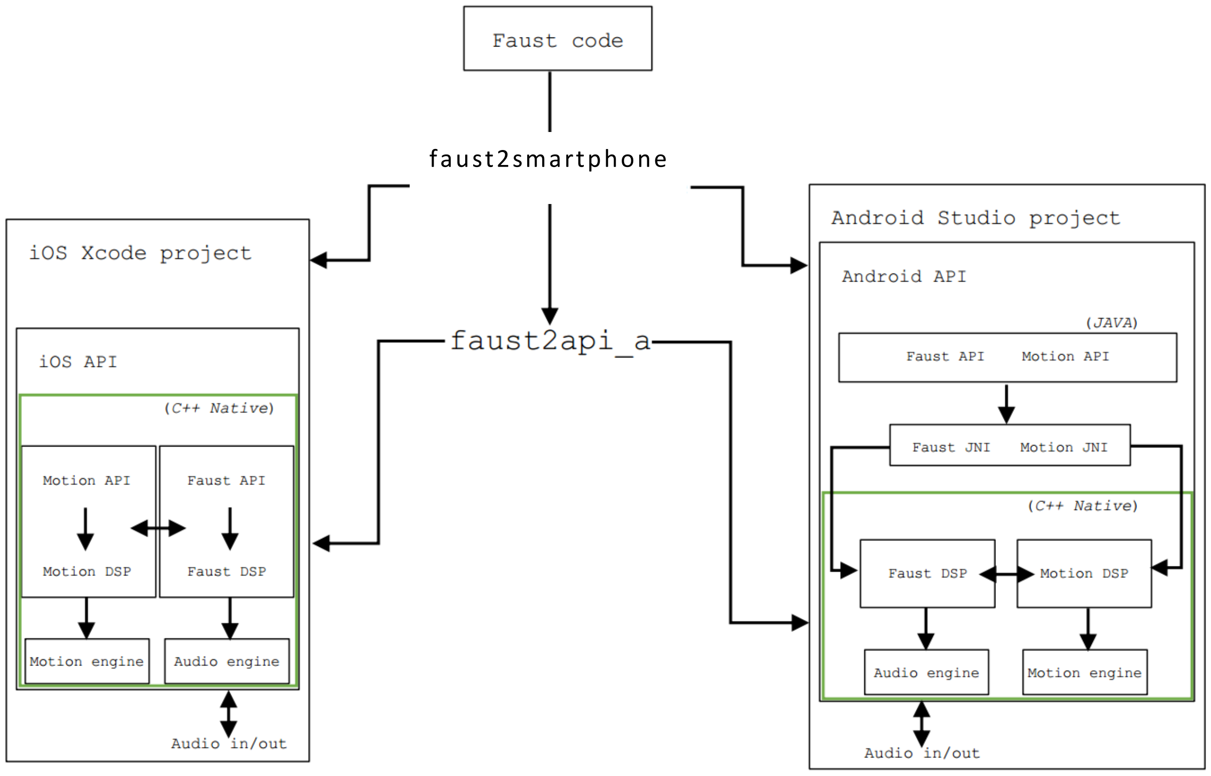

Figure 1. Implementation of faust2smartphone

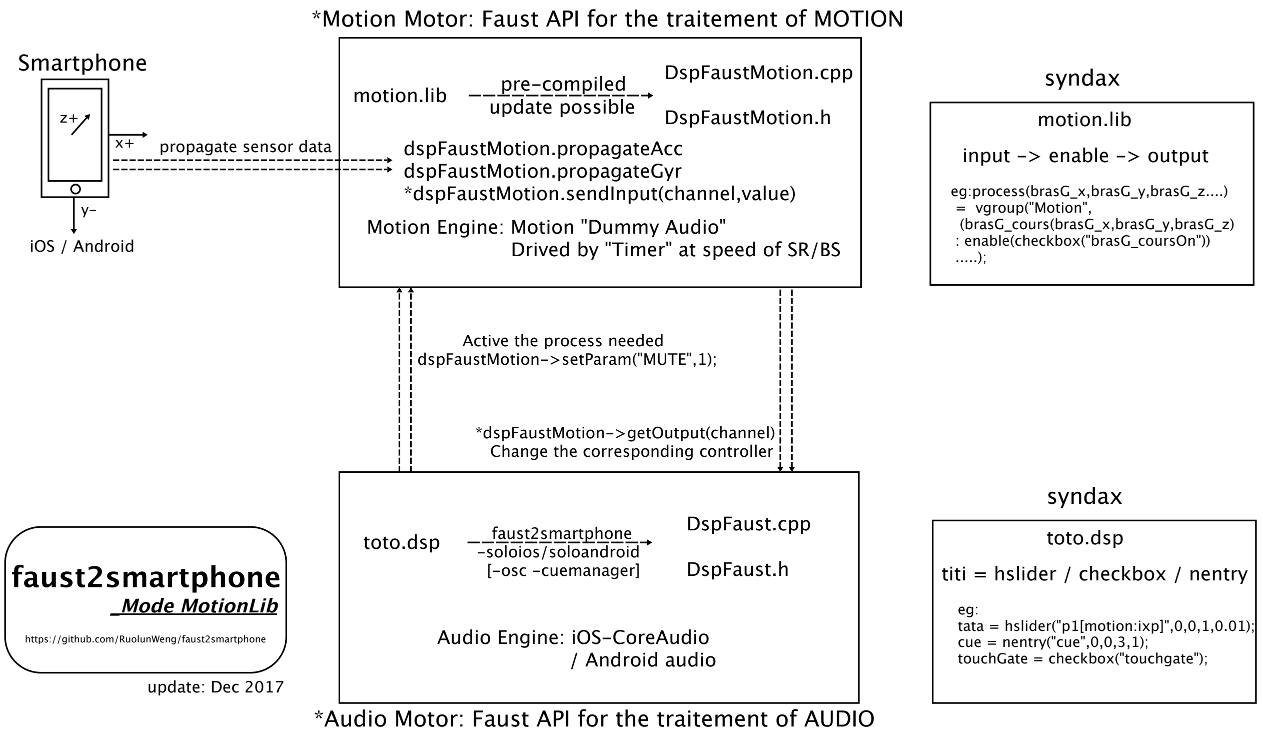
*Motion Motor: Faust API for the traitement of MOTION
Smartphone
z+
x+
y-
iOS / Android
propagate sensor data
motion.lib
pre-compiled
update possible
DspFaustMotion.cpp
DspFaustMotion.h
dspFaustMotion.propagateAcc
dspFaustMotion.propagateGyr
*dspFaustMotion.sendInput(channel,value)
Motion Engine: Motion "Dummy Audio"
Drived by "Timer" at speed of SR/BS
syndax
motion.lib
input -> enable -> output
eg:process(brasG_x,brasG_y,brasG_z....)
= vgroup("Motion",
(brasG_cours(brasG_x,brasG_y,brasG_z)
: enable(checkbox("brasG_coursOn"))
.....);
Active the process needed
dspFaustMotion->setParam("MUTE",1);
*dspFaustMotion->getOutput(channel)
Change the corresponding controller
faust2smartphone
Mode MotionLib
https://github.com/RuolunWeng/faust2smartphone
update: Dec 2017
toto.dsp
faust2smartphone
-soloios/soloandroid
[-osc -cuemanager]
DspFaust.cpp
DspFaust.h
Audio Engine: iOS-CoreAudio
/ Android audio
*Audio Motor: Faust API for the traitement of AUDIO
syndax
toto.dsp
titi = hslider / checkbox / nentry
eg:
tata = hslider("p1[motion:ixp]",0,0,1,0.01);
cue = nentry("cue",0,0,3,1);
touchGate = checkbox("touchgate");

Figure 2. Motion mode of faust2smartphone

where “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);
tit=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.

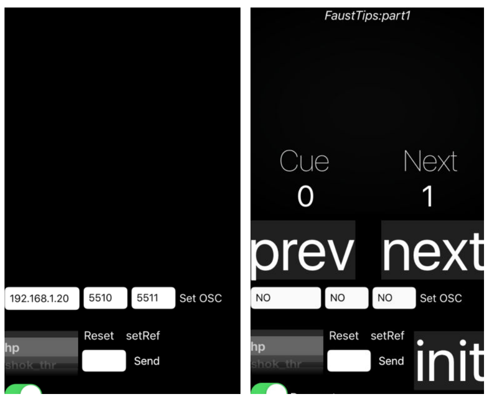

Figure 3. Interface of faust2smartphone

2.3 Plugin 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, the purpose is not to generate sound, but capture the signal digital value as envelope follower to affect other digital processing like video or lightening etc. The engine which is simplified version of DspFaustMotion from the motion mode will be computed by a simple timer, we can use the result for any parameter of post-processing.
For example, if we want to use the amplitude of an oscillator to control the alpha of the phone’s 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(), the template of plugin mode is a simple example to get started.

3. APPLICATIONS

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 in 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 AR (Augmented Reality). The application is both available at App Store and Google Play.
All of these projects are still maintained and envolving based on the previous feedback. “Audio Guide” is becoming the general experience option in the residence for the blind. Every year, the residents will update the content of the app with their new sound creation. An “open-day” in each July will also welcome the public for a special event. “Virtuel Rhizome” became one of the reference projects of mobile musical applications, implemented for its related sound installation, also generalised under the concept called “Smart Hand Computer” by Christophe Lebreton. “sfPivoine” keeps the trend of download activity by attacting many chinese culture lovers. With the positive impacts, there are also some critical options on the perfomance with smartphone, like the mixing with chinese opera and traditionnal instrument. They help us to keep in mind that we should always justify our choice of the application, technical frame must serve the global idea smoothly.

4. FUTURE WORKS

Elaborating the documentation of faust2smartphone is the essential task for the next step. In order to well guide people to use the project, we will also enrich the examples and instructions.
For the people who is interesting for the development of motion.lib, which is used in the motion mode, they are welcome to cooperate with us for the new functions based on the sensors etc.
Since there are many other frameworks, programming languages and Web development, more and more adapt to the mobile environment, how to identify or merge with them to make a more flexible project is also our goal.
For the applications, we will try to finish the performance version and publish them in the app store. Last but not least, we still need more user cases to test the workflow.

5. ACKNOWLEDGEMENT

Special thanks to Mr. LEBRETON Christophe, Mr. LETZ Stephane, Mr. ORLAREY Yann, Mr. POTTIER Laurent for their supports during the research.
The paper is also based on the speech of the 1st International Faust Conference, which is held in Mainz of Germany in 2018.

6. REFERENCES

[1] Faust Website: http://faust.grame.fr/
[2] 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.
[3] 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.
[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.