FaustPad : A free open-source mobile app for multi-touch interaction with Faust generated modules

Hyung-Suk Kim
Dept. of Electrical Engineering,
Stanford University
Palo Alto, CA 94305, USA
hskim08@stanford.edu

Julius O. Smith
Center for Computer Research in Music and Acoustics.
(CCRMA) Stanford University
Palo Alto, CA 94305, USA
jos@ccrma.stanford.edu

Abstract

In this paper we introduce a novel interface for mobile devices enabling multi-touch interaction with sound modules generated with Faust[1] (Functional Audio Stream) and run in SuperCollider. The interface allows a streamlined experience for experimentation and exploration of sound design.

Keywords

Faust, mobile music, SuperCollider, music app.

1 Introduction

Faust (Functional Audio Stream)[1] is a functional domain-specific language for real-time signal processing and synthesis. Programs written in Faust are translated in to highly efficient C++ code which can then be compiled into modules for various architectures and synthesis environments including SuperCollider, PureData and Chuck. Faust separates the specification from the underlying implementation and the UI, allowing the programmer to focus on the design of signal processing or synthesis modules.

Quick testing of a module is possible by compiling the program as a standalone program. However integrating a UI for a synthesis environment such as SuperCollider or PureData is not that trivial. In either case, the user interaction is limited to mouse-pointer interaction.

Mobile devices introduce a tangible, interactive experience compared to that of the traditional desktop. Recent mobile devices, i.e. smart phones, enable multi-touch interaction as well as other features, e.g. accelerometer, GPS, gyroscope, which allow novel methods of interacting with sound.

[1]http://faust.grame.fr/

Mobile devices have become computationally powerful enough for audio synthesis directly on the device. The MoMu toolkit [2] used by the Stanford Mobile Phone Orchestra (MoPho) is a toolkit for music synthesis on iOS devices. Unfortunately most audio SDKs for mobile devices are OS dependent, requiring separate learning for each new mobile OS. The computation power of mobile devices is still limited compared to desktop systems. Computationally complex modules are hard to run on mobile devices.

Mobile devices can be used as a music controller via OSC. TouchOSC, Control OSC[2] are examples. Such apps are general purpose OSC controllers that are highly customizable. The price of customizability is that it takes time to set up a module. This can become very cumbersome if the sound module has a lot of controls. Physical modeling synthesis programs can easily have over 20 parameters which can be very time consuming to setup.

In this paper we present an OSC music controller app, FaustPad,[3] that automatically creates the interface and necessary OSC settings based on the compiled results from Faust. This approach reduces the complicated setup process, allowing a streamlined experience for experimenting with the generated sound module.

2 Overview

Setting up a module for interaction requires the following steps: 1) compiling the program with Faust, 2) setting up the synthesis environment, and 3) setting up the mobile device.

[2]http://charlie-roberts.com/Control/

[3]Code and documentation for FaustPad, can be found at https://github.com/hskim08/FaustPad

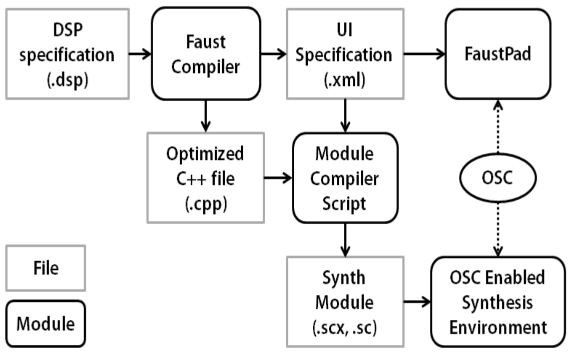


Figure 1: Overview flowchart of FaustPad setup. The file in parentheses may change depending on the environment used. Here we show the case for a SuperCollider Extension.

In this paper we cover setting up a Faust synthesis module with SuperCollider. In the last section we look at an example of installing all the modules in Faust-STK[3], the Synthesis Toolkit port for Faust.

We use Faust to create a SuperCollider extension which is loaded into `scsynth`, the SuperCollider server. FaustPad is used as an alternative `scsynth` client, replacing `sclang`, the SuperCollider interpreter.

3 Compiling with Faust

Recent Faust releases include scripts to compile programs into various environments. For SuperCollider extensions, we used `faust2supercollider` to create a SuperCollider class file (`.sc`) and a SuperCollider extension (`.scx`).

The Faust compiler can create an abstract "user interface" definition in XML format with the `--xml` option. This XML-file is used to create the UI for the mobile app.

For an in-depth description see "Audio Signal Processing in FAUST"[4].

4 SuperCollider

SuperCollider consists of two parts, the client, `sclang`, and the server, `scsynth`. The two parts communicate via OSC. FaustPad acts as an alternative client to `sclang`.

[4]https://ccrma.stanford.edu/~jos/spf/Using_FAUST_SuperCollider.html

```
  SynthDef.new("sitar", {
      arg freq = 440.0, gain = 1.0, gate =
0.0,  resonance  =  0.7,  damp  =  0.72,
roomsize = 0.54, wet = 0.141, pan_angle =
0.6, spatial_width = 0.5;
      Out.ar(0,  FaustSitar.ar(freq,  gain,
gate,  resonance,  damp,  roomsize,  wet,
pan_angle, spatial_width));
  }).load(s);
```

Figure 2: Output example of FaustScParser.

It is easy to import custom synthesis modules into SuperCollider as "extensions" by copying the `.sc` and `.scx` files into the SuperCollider extension folder.

Synth definition (SynthDef) files need to be defined and loaded to create instances of the synth modules. Control parameter names are also defined in the SynthDef files. The parameter names of the SynthDef files must match those of the app. This can be achieved by parsing the UI definition XML-file from Faust to create the SynthDef file. The Java project in the FaustPad github repository creates an executable jar-file, FaustScParser.jar, that does this.

In `sclang`, a SynthDef is compiled into byte code which is then sent to `scsynth` as an OSC message. As of the time of writing the SynthDef needs to be loaded once manually by the script created by the FaustScParser. Once loaded the SynthDef is saved on the server and will be loaded each time `scsynth` boots. We plan to port SynthDef compiling into FaustPad to further reduce the setup process.

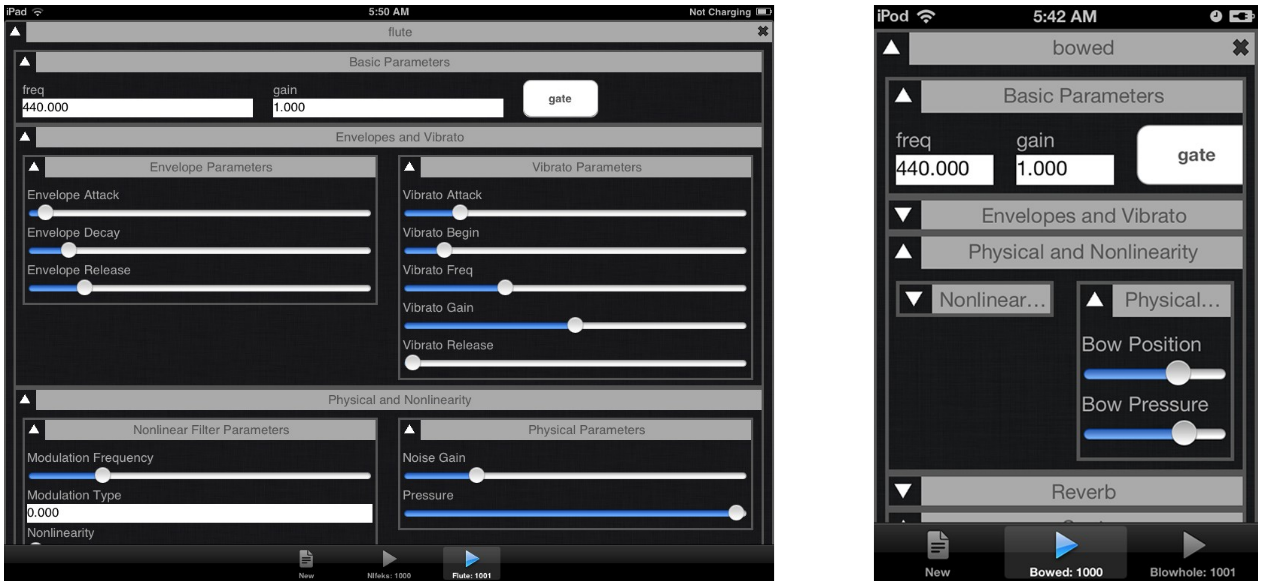
5 FaustPad

FaustPad is an OSC controller app with auto generated UIs tailored to Faust created modules. Ease of setup comes at the price of customizability. The app will only work for Faust generated modules. Given the expandability of Faust, it is a tradeoff worth making.

One advantage of using a mobile device is multi-touch. Multiple sliders and buttons can be modified simultaneously. FaustPad currently supports iOS devices only. An iPhone or iPod Touch can handle up to 5 touches and an iPad can handle up to 11 touches at once.

5.1 Building the UI

A typical Faust user interface definition has two parts, 1) the definition of widgets, i.e. the control parameters both passive and active, and 2) the layout



freq 440.000
gain 1.000
Modulation Type 0.000

freq 440.000
gain 1.000

Figure 3: Screenshots of FaustPad for iPad and iPod Touch. The FaustPad UI works for any orientation. In the iPod Touch screenshot, groups are folded to show only controls of interest.

definition. The separation of components and layout frees the UI builder from a static layout.

In the current implementation of FaustPad, the app maintains the given layout, creating foldable "group views". The automated building of the UI has two phases accordingly, 1) the creation of widgets and 2) the creation of groups which form the layout.

5.2 Layout principles

Faust generated modules may have many control parameters. This is especially true for physical modeling synthesis modules such as many of the modules in Faust-STK. For many parameters, once a value is set there may be little need to change it during a performance. In FaustPad, unused groups can be folded so that the user can focus on the parameters of interest.

FaustPad also allows multiple modules to be created. Each module is created in a separate tab. Tabs are always visible in the tab bar at the bottom of the UI, allowing quick change of synth module.

5.3 OSC messaging

Each UI component holds its own data, e.g. label, min/max values, from the UI description file. When there is a user interaction event, it sends an event to the OSC messaging object which parses the event and sends a properly formatted OSC message to the server. This separation of UI component and OSC messaging allows configurations for various OSC messaging protocols for different synthesis environments. The OSC messaging object also listens for incoming OSC messages that can be used for automation and feedback from the server.

6 Example: Faust-STK

We have tested FaustPad with Faust-STK, included in the Faust release. The example files come with a `Makefile` for each environment including SuperCollider. Creating the UI description files is less trivial, because the `Makefile` removes the files at the end of the compilation. This can be avoided by modifying `faust2supercollider` and `Makefile.sccompile`. After setting up the aforementioned scripts then loading all SynthDefs, we could use all 21 Faust-STK modules in FaustPad by simply copying the files to the FaustPad documents directory.

7 Extending FaustPad

The FaustPad user interface only adheres to the user interface description file and makes no assumption of the underlying structure until the moment a OSC message is sent in the OSC messaging object. Thus it is possible to extend the FaustPad to other OSC enabled synthesis environment such as PureData or Chuck. As of the time of writing, we are in the process of adding support for PureData via `pd-faust`[5].

[5]http://docs.pure-lang.googlecode.com/hg/pd-faust.html

8 Conclusion

With FaustPad we have explored methods for enabling a streamlined experience for interacting with Faust generated synthesis modules.

The separation of specification, device implementation and UI description of Faust enables support for multiple synthesis environments with multiple UIs. With FaustPad we explored the possibilities of using an alternative UI on a mobile device.

We have tested the experience by compiling Faust-STK synthesis modules for SuperCollider. Though there were some non-trivial modifications to code that were needed to obtain the UI description files, once made, it was easy to run and install the synthesis modules.

FaustPad is still in its early stages and there are more issues to be addressed. Such issues include extension of the interface to other environments such as PureData or Chuck, easy discovery and connection of server and device via technologies such as Bonjour/Zeroconf networking.

Connecting to PureData via pd-faust is of particular interest due to the expandability of PureData itself. With pd-faust it is possible to synchronize the interface to a MIDI-OSC transport for automation allowing live performance and audio recording use of FaustPad.

Finally, we acknowledge the fact that even though FaustPad is a free open-source project under a BSD-like license, it may not be open for anyone to develop due to the limitations of the iOS SDK. Porting to other mobile OS, many of which have become open-source, is another important future work to be done.

References

[1] Orlarey, Yann; Fober, Dominique; Letz, Stéphane. 2009. "Faust: an Efficient Functional Approach to DSP Programming". *New Computanionals Paradigms for Computer Music*. Edition Delatour. ISBN 978-2-7521-0054-2.

[2] Bryan, N. J., Herrera, J., Oh, J., and Wang, G. 2010. “MoMu: A Mobile Music Toolkit.” *In Proceedings of the International Conference on New Interfaces for Musical Expression*. Sydney 2010.

[3] Michon, Romain Smith, Julius O. III. 2011. "Faust-STK: a Set of Linear and Nonlinear Physical Models for the Faust Programming Language". *Proceedings of the 11th Int. Conference on Digital Audio Effects (DAFx-11)*: 199–204