New Signal Processing Libraries for Faust

Romain Michon, Julius Smith
CCRMA
Stanford University
Stanford, CA 94305-8180
USA
{rmichon,jos}@ccrma.stanford.edu

Yann Orlarey
GRAME
Centre National de Création Musicale
11 Cours de Verdun (Gensoul)
69002, Lyon
France
orlarey@grame.fr

Abstract

We present a completely re-organized set of signal processing libraries for the FAUST programming language. They aim at providing a clearer classification of the different FAUST DSP functions, as well as better documentation. After giving an overview of this new system, we provide technical details about its implementation. Finally, we evaluate it and give ideas for future directions.

Keywords

FAUST, Digital Signal Processing, Computer Music Programming Language

1 Introduction

FAUST is a functional programming language for real time Digital Signal Processing (DSP) targeting high-performance audio applications and plug-ins for a wide range of platforms and standards. [Orlarey et al., 2009]

One of FAUST's strength lies in its DSP libraries implementing a large collection of reference implementations ranging from filters to audio effects and sound generators, etc.

When FAUST was created, it had a limited number of DSP libraries that were organized in a "somewhat" coherent way: `math.lib` contained mathematical functions, and `music.lib` everything else (filters, effects, generators, etc.). Later, the libraries `filter.lib`, `oscillator.lib`, and `effect.lib` were developed [Smith, 2008], [Smith, 2012], which had significant overlap in scope with `music.lib`.

A year ago, we decided to fully reorganize the FAUST libraries to

- provide more clarity,
- organize functions by category,
- standardize function names,
- create a dynamic documentation of their content.

In this paper, we give an overview of the organization of the new FAUST libraries, as well as technical details about their implementation. We then evaluate them through the results of a workshop on FAUST that was taught at the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University in 2016, and we provide ideas for future directions.

2 Global Organization and Standards

2.1 Overview

The new FAUST libraries[1] are organized in different files presented in Figure 1. Each file contains several subcategories allowing to easily find functions for specific uses. While some libraries host fewer functions than others, they were created to be easily updated with new elements. The content of the old (and now deprecated) FAUST libraries was spread across these new files, making backward compatibility a bit hard to implement (see §2.4).

More specifically, the old `music.lib` was removed since it contained much overlap in scope with `oscillator.lib`, `effect.lib`, and `filter.lib`.

`effect.lib` was divided into several "specialized" libraries: `compressors.lib`, `misceffects.lib`, `phaflangers.lib`, `reverbs.lib`, and `vaeffects.lib`. Similarly, the content of `oscillator.lib` is now spread between `noises.lib` and `oscillators.lib`. Finally, `demo.lib` hosts demo functions, typically adding user-interface elements with illustrative parameter defaults.

2.2 Prefixes

Each FAUST library has a recommended two-letter namespace prefix defined in the "meta library" `stdfaust.lib`. For example, `stdfaust.lib` contains the lines

[1] `http://faust.grame.fr/library.html`. All the URLs in this paper were verified on 01/30/17.

analyzer.lib
- Amplitude Tracking
- Spectrum-Analyzers
- Mth-Octave Spectral Level
- Arbitrary-Crossover Filter
- Banks and Spectrum Analyzers

basics.lib
- Conversion Tools
- Counters and Time/Tempo Tools
- Array Processing and Pattern Matching
- Selectors (Conditions)
- Other Misc Functions

compressors.lib
Compressors and limiters library.

delays.lib
- Basic Delay Functions
- Lagrange Interpolation
- Thiran Allpass Interpolation

demos.lib
- Analyzers
- Filters
- Effects
- Generators

envelopes.lib
Envelope generators library.

filters.lib
- Basic Filters
- Comb Filters
- Direct-Form Sections
- Direct-Form Second-Order
- Biquad Sections
- Ladder/Lattice
- Virtual Analog Filters
- Simple Resonator
- Butterworth Filters
- Elliptic (Cauer) Filters
- Filters for Parametric Equalizers (Shelf, Peaking)
- Arbitrary-Crossover Filter-Banks

maths.lib
- Constants
- Functions

misceffects.lib
- Dynamic
- Filtering
- Time Based
- Pitch Shifting
- Meshes

noises.lib
Noise generators library.

oscillators.lib
- Wave-Table-Based Oscillators
- LFOs
- Low Frequency Sawtooths
- Bandlimited Sawtooth
- Bandlimited Pulse, Square, and Impulse Trains
- Filter-Based Oscillators
- Waveguide-Resonators

phaflangers.lib
Phasers and flangers library

reverbs.lib
Reverbs library.

routes.lib
Signal routing library.

signals.lib
Misc signal tools library.

spats.lib
Spatialization tools library.

synths.lib
Misc synthesizers library.

vaeffects.lib
Virtual analog effects library.

Figure 1: Overview of the organization of the new FAUST libraries.

```
fi = library("filters.lib");
os = library("oscillators.lib");
```

so that functions from `oscillator.lib` can be invoked using the `os` prefix and functions from `filter.lib` through `fi`:

```
import("stdfaust.lib");
process = os.sawtooth(440) : fi.lowpass
    (2,2000);
```

It is of course possible to avoid prefixes using the `import` directive:

```
import("filters.lib");
import("oscillators.lib");
process = sawtooth(440) : lowpass
    (2,2000);
```

The libraries presently avoid name collisions, so it is possible to load all functions from all libraries into one giant namespace soup:

```
import("all.lib");
process = sawtooth(440) : lowpass
    (2,2000);
```

Alternatively, all FAUST-defined functions can be loaded into a single namespace separate from the user's namespace:

```
sf = library("all.lib"); // standard
    faust namespace
process = sf.sawtooth(440) : sf.lowpass
    (2,2000);
```

Further details can be found in the documentation for the libraries.[2]

2.3 Standard Functions

The FAUST libraries implement dozens of functions, and it can be hard for new users to find standard elements for basic uses. For example, `filter.lib` contains seven different lowpass filters, and it's probably not obvious to someone with little experience in signal processing which one should be used.

To address this problem, the new FAUST libraries declare "standard" functions (see Figure 2) that are automatically added to the library documentation.[3] Standard functions are organized by categories, independently from the library where they are declared (see §3). They should cover the needs of most users used to computer music programming environments such as PureData,[4] SuperCollider,[5] etc.

2.4 Backward Compatibility

With such major changes, providing a decent level of backward compatibility proved to be quite complicated. The old FAUST libraries (`effect.lib`, `filter.lib`, `math.lib`, `music.lib` and `oscillator.lib`) can still be used and will remain accessible for about one year.

In order to make this possible, we had to find a way to make them cohabit with the new libraries without creating conflicts. Thus, we decided to use plurals for the name of the new libraries, allowing to concurrently use our new `filters.lib` with the old `filter.lib`, for example.

If one of the old libraries is imported in a FAUST program, the FAUST compiler now throws a warning indicating the use of a deprecated library.

2.5 Other "Non-Standard" Libraries

A few "non-standard" libraries for very specific applications remain accessible but are not documented (see §3):

- `hoa.lib`: high order ambisonics library
- `instruments.lib`: library used by the FAUST-STK [Michon and Smith, 2011]
- `maxmsp.lib`: compatibility library for Max/MSP
- `tonestacks.lib`: tonestack emulation library used by Guitarix[6]
- `tubes.lib`: guitar tube emulation library used by Guitarix

3 Automatic Documentation

The new FAUST libraries use a new automatic documentation system based on the `faust2md` (FAUST to MarkDown) script which is now part of the FAUST distribution. It allows to easily write MarkDown comments within the code of the libraries by respecting the standards described below.

Library headers and descriptions can be created with

```
//##### Library Name ##### // Some
    Markdown text.
//########################
```

Libraries can be organized into sections using the following syntax:

```
//===== Section Name ===== // Some
    Markdown text.
//========================
```

Each function in a library should be documented as such:

```
//---- Function Name ---- // Some
    Markdown text.
//----------------------
```

The libraries documentation can be conveniently generated by running:

```
make doclib
```

[2] http://faust.grame.fr/library.html

[3] http://faust.grame.fr/library.html\#standard-functions.

[4] https://puredata.info.

[5] http://supercollider.github.io.

[6] http://guitarix.org.

Analysis Tools	
`an.amp_follower`	Amplitude follower
`an.mth_oct[...]`	Octave analyzers
Basic Elements	
`ba.beat`	Pulse generator
`si.block`	Block a signal
`ba.bpf`	Break Point Function
`si.bus`	Bus of n signals
`ba.bypass1`	Bypass (mono)
`ba.bypass2`	Bypass (stereo)
`ba.count`	Counts in a list
`ba.countdown`	Samples count down
`ba.countup`	Samples count up
`de.delay`	Integer delay
`de.fdelay`	Fractional delay
`ba.impulsify`	Signal to impulse
`ba.sAndH`	Sample and hold
`ro.cross`	Cross n signals
`si.smoo`	Smoothing
`si.smooth`	Controllable smoothing
`ba.take`	Element from a list
`ba.time`	Timer
Conversion	
`ba.db2linear`	dB to linear
`ba.linear2db`	Linear to dB
`ba.midikey2hz`	MIDI key to Hz
`ba.pole2tau`	Pole to t60
`ba.samp2sec`	Samples to seconds
`ba.sec2samp`	Seconds to samples
`ba.tau2pole`	t60 to pole
Effects	
`ve.autowah`	Auto-wah
`co.compressor`	Compressor
`ef.cubicnl`	Distortion
`ve.crybaby`	Crybaby
`ef.echo`	Echo
`pf.flanger`	Flanger
`ef.gate_mono`	Signal gate
`co.limiter`	Limiter
`pf.phaser2`	Phaser
`re.fdnrev0`	Reverb (FDN)
`re.freeverb`	Reverb (Freeverb)
`re.jcrev`	Reverb (simple)
`re.zita_rev1`	Reverb (Zita)
`sp.panner`	Panner
`ef.transpose`	Pitch shift
`sp.spat`	Panner
`ef.speakerbp`	Speaker simulator
`ef.stereo_width`	Stereo width
`ve.vocoder`	Vocoder
`ve.wah4`	Wah

Envelopes	
`en.adsr`	ADSR envelope
`en.ar`	AR envelope
`en.asr`	ASR envelope
`en.smoothEnv`	Exponential envelope
Filters	
`fi.bandpass`	Bandpass (Butterworth)
`fi.resonbp`	Bandpass (resonant)
`fi.bandstop`	Bandstop (Butterworth)
`fi.tf2`	Biquad Filters
`fi.allpass_fcomb`	Comb (allpass)
`fi.fb_fcomb`	Comb (feedback)
`fi.ff_fcomb`	Comb (feedforward)
`fi.dcblocker`	DC blocker
`fi.filterbank`	Filterbank
`fi.fir`	FIR (arbitrary order)
`fi.high_shelf`	High shelf
`fi.highpass`	Highpass (Butterworth)
`fi.resonhp`	Highpass (resonant)
`fi.iir`	IIR (arbitrary order)
`fi.levelfilter`	Level filter
`fi.low_shelf`	Low shelf
`fi.lowpass`	Lowpass (Butterworth)
`fi.resonlp`	Lowpass (resonant)
`fi.notchw`	Notch filter
`fi.peak_eq`	Peak equalizer
Generators	
`os.impulse`	Impulse
`os.imptrain`	Impulse train
`os.phasor`	Phasor
`no.pink_noise`	Pink noise
`os.pulsetrain`	Pulse train
`os.lf_imptrain`	Low-freq pulse train
`os.sawtooth`	Sawtooth wave
`os.lf_saw`	Low-freq sawtooth
`os.osc`	Sine (filter-based)
`os.oscsin`	Sine (table-based)
`os.square`	square wave
`os.lf_square`	Low-freq square
`os.triangle`	Triangle
`os.lf_triangle`	Low-freq triangle
`no.noise`	White noise
Synths	
`sy.additiveDrum`	Additive drum
`sy.dubDub`	Filtered sawtooth
`sy.combString`	Comb string
`sy.fm`	FM
`sy.sawTrombone`	Lowpassed sawtooth
`sy.popFiltPerc`	Popping filter

Figure 2: Standard FAUST functions with their corresponding prefix when used with `stdfaust.lib`.

at the root of the FAUST distribution. This will generate an `html` and a `pdf` file in the `/documentation` folder using `pandoc`.[7]

4 Evaluation and Future Directions

The new FAUST libraries were beta tested during the *CCRMA Faust Summer Workshop* at Stanford University.[8] In previous editions of the workshop, students had to go through the library files to get the documentation of specific functions. During last year's workshop, thanks to the new libraries documentation, students were able to find information about functions simply by doing a search in the documentation file. Additionally, none of them encountered problems while using the new libraries which was very satisfying.

The FAUST libraries are meant to grow with time, and we hope that this new format will facilitate the integration of new contributions. Eventually, we plan to divide `filters.lib` into more subcategories, like we did for the old `oscillator.lib`. Finally, `physmodels.lib` which is a new library for physical modeling of musical instruments is currently under development.

5 Conclusions

The new FAUST libraries provide a platform to easily prototype DSP algorithms using the FAUST programming language. Their new organization, in combination with their automatically generated documentation, simplifies the search for specific elements covering a wide range of uses. New "standard functions" help to point new users to useful elements to implement various kind of synthesizers, audio effects, etc. Finally, we hope that this new format will encourage new contributions.

6 Acknowledgments

Thanks to Albert Gräf for his contributions to the design of the new libraries, and for single-handedly implementing a solid backward-compatibility scheme!

References

Romain Michon and Julius O. Smith. 2011. 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.

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

Julius Orion Smith. 2008. Virtual electric guitars and effects using Faust and Octave. In *Proceedings of the Linux Audio Conference (LAC-08)*, pages 123–127, KHM, Cologne, Germany.

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

[7] `http://pandoc.org`.

[8] `https://ccrma.stanford.edu/~rmichon/faustWorkshops/2016`.