Faustine User Manual

CRI – Mines ParisTech
Centre de Recherche en Informatique

September 2013

Contents

Chapter 1

Introduction

FAUSTINE is an interpreter for multi-rate and vector exented FAUST programs testing, written in OCaml, at CRI of MINES ParisTech, and covered by the GNU Public License V3 (see LICENSE.txt).

FAUST (*Functional Audio Stream*) is a functional programming language specifically designed for real-time signal processing and synthesis. FAUST targets high-performance signal processing applications and audio plug-ins for a variety of platforms and standards.

1.1 Design Principles

Various principles have guided the design of FAUSTINE:

- FAUSTINE is a *test bed* interpreter for faust programs, especially for vector extension. It aims at providing a framework to test *vector* and *multi-rate* ideas quite easily, without having to deal with the burdens of the compiler. FAUSTINE is written in OCaml.

- FAUSTINE programs are interpreted upon FAUST language and extensions, onto input files (wav or csv), and producing output files (also wav or csv). The interpreter relies on a FAUST preprocessor to translate FAUST programs into equivalent flatten programs containing only core FAUST functions except GUI ones.

- In most cases, FAUSTINE is inefficient but, still, it often allows to get a idea of time consumption location.

- Faustine depends on g++ and ocamlopt compilers. It embeds libsndfile and a slighlty modified version of libsndfile-ocaml.
- For the moment, Faustine only handles dynamic type-checking but future work should address static type-checking.
- Faustine current extension supports four multirate and vector functions: `vectorize`, `serialize`, `[ ]` (pick vector element), and `#` (concatenate two vector elements).
- So far, several vector libraries have been developed: complex.lib, fft.lib, fft2d.lib and morpho.lib.

Chapter 2

Compiling and installing Faustine

FAUSTINE's git repository can be cloned calling:

```
git clone https://scm.cri.ensmp.fr/git/
    Faustine.git
```

2.1 Organization of the distribution

FAUSTINE directory should contain the following elements:

benchmarks/	benchmark result files
Changes.txt	what's new with each release
configure	compilation configuration script
examples/	vector examples (fft, image processing...)
INSTALL.txt	Faustine installation instructions
interpreter/	Faustine's interpreter source code
lib/	library files in Faustine (fft.lib, morpho.lib...)
LICENSE.txt	license and copyright notice
Makefile	main Makefile to compile and install
README.txt	this file

2.2 Compiling and Installing

FAUSTINE has no dependencies outside standard libraries, except OCaml and g++ compilers and 'make'-like standard commands. Therefore the compilation should be straightforward. Configure is necessary for libsndfile embedded library. To compile the Faustine interpreter do:

```
    cd Faustine/
    ./configure
    make
    sudo make install
```

If the compilation was successful you can test the interpreter before installing it:

```
    make test
```

Chapter 3

How to use Faustine

Chapter 4

How to maintain and extend Faustine