DEMONSTRATION OF FAUST SIGNAL PROCESSING LANGUAGE

Yann Orlarey, Dominique Fober, Stephane Letz

DEMONSTRATION OF FAUST SIGNAL PROCESSING LANGUAGE

Yann Orlarey, Dominique Fober, Stephane Letz
Grame, Centre National de Creation Musicale
9, rue du Garet, BP 1185
69001 Lyon Cedex 01
France

ABSTRACT

FAUST is a compiled language designed for real-time audio signal processing. It is a free software published under the terms of the GNU General Public License. Sources and binaries are available at source forge :

http://faudiostream.sourceforge.net.

The demonstration will give an overview of the main features of the language and the compiler through several simple and practical examples.

1. INTRODUCTION

The name FAUST [1] stands for *Functional AUdio STream*. It designates a language that combines two programming models : *functional programming* and *block diagram composition*. Therefore you can think of FAUST as a *structured block diagram language* with a textual syntax.

FAUST is intended for programmers that need to develop efficient C/C++ audio code from a high level specification. A single FAUST specification can be used for multiple targets from plugins (various plugins standards are supported) to standalone audio applications. Thanks to some specific compilation techniques and powerful optimizations, the C++ code generated by the Faust compiler is usually very fast and can generally compete with (and sometimes outperform) hand-written C/C++ code.

2. THE LANGUAGE

The primitive elements of the language are : *mathematical operations* on signals (the C functions on numbers have a FAUST equivalent on signals), *delays*, *tables* and *user interface widgets*. These primitive building blocks are combined using five high level composition operations. They operate on block-diagrams by establishing specific patterns of connections between their inputs and outputs. Moreover powerful abstraction mechanisms based on the Lambda-Calculus allows for *parametric* block-diagrams (block-diagrams with *variable parts* that can be instantiated).

Programming with FAUST is somehow like working with electronic circuits and analog signals. A FAUST program is a list of definitions that describes a *signal processor* : a piece of code that produces output signals according to its inputs signals (and maybe some user interface parameters). For example a very simple FAUST program that transforms a stereo signal into a mono signal is :

```
process = +;
```

It defines the `process` block-diagram (the FAUST equivalent for `main` in C) as the `+` primitive operator that add two signals.

3. THE COMPILER

In order to generate the most efficient code, the compilation is based on a *semantic* approach. Instead of compiling directly the block-diagram defined by the user, the idea is to compile its *mathematical meaning* (what it actually computes).

This mathematical meaning is inferred by the compiler using symbolic techniques. It propagates virtual signals in the block diagram and collects the resulting output signal equations. These output equations are then simplified and normalized before being used to generate the C++ code. Two different block diagrams can then result in the same C++ program provided that they have the same mathematical meaning.

The compiler can optionally wrap the generated code into an architecture depended file in order to directly produce a VST plugin, a MAX/MSP external object or a Jack-Gtk application for Linux. Eight different targets are currently available : Max/MSP, VST, LADSPA, OSS, Jack, Port-Audio, Audio files. Other targets can be added simply by providing the corresponding architecture file.

4. REFERENCES

[1] Yann Orlarey, Dominique Fober, Stephane Letz "Syntactical and Semantical Aspects of Faust", *Soft Computing*, Volume 8, Number 9, Springer-Verlag GmbH, September 2004.