Some Web Oriented Applications of FAUST

Y. Orlarey, D. Fober, S. Letz, R. Michon

GRAME – Centre National de Création Musicale

Journée Web des Fonctions, Grame, 27 juin 2012

1-Introduction

Introduction

What is FAUST ?

FAUST stands for *Functional AUdio STream*:

- FAUST is a :
  - *DSL* for real-time audio signal processing and synthesis.
  - based on a purely *functional* and *synchrounous* approach.
- It can be used to develop:
  - *audio effects,*
  - *sound synthesizers*
  - real-time applications processing *signals.*
- Who uses FAUST ?
  - Developers of audio applications and plugins,
  - Sound engineers and musical assistants
  - Researchers in Computer Music

Introduction

What is FAUST ?

FAUST stands for *Functional AUdio STream*:

- FAUST is a :
  - *DSL* for real-time audio signal processing and synthesis.
  - based on a purely *functional* and *synchrounous* approach.
- It can be used to develop:
  - *audio effects*,
  - *sound synthesizers*
  - real-time applications processing *signals*.
- Who uses FAUST ?
  - Developers of audio applications and plugins,
  - Sound engineers and musical assistants
  - Researchers in Computer Music

Introduction

What is FAUST ?

FAUST stands for *Functional AUdio STream*:

- FAUST is a :
  - *DSL* for real-time audio signal processing and synthesis.
  - based on a purely *functional* and *synchrounous* approach.
- It can be used to develop:
  - *audio effects*,
  - *sound synthesizers*
  - real-time applications processing *signals*.
- Who uses FAUST ?
  - Developers of audio applications and plugins,
  - Sound engineers and musical assistants
  - Researchers in Computer Music

FAUST stands for *Functional AUdio STream*:

- FAUST is a :
  - *DSL* for real-time audio signal processing and synthesis.
  - based on a purely *functional* and *synchrounous* approach.
- It can be used to develop:
  - *audio effects*,
  - *sound synthesizers*
  - real-time applications processing *signals*.
- Who uses FAUST ?
  - Developers of audio applications and plugins,
  - Sound engineers and musical assistants
  - Researchers in Computer Music

Introduction

What is FAUST ?

FAUST stands for *Functional AUdio STream*:

- FAUST is a :
  - *DSL* for real-time audio signal processing and synthesis.
  - based on a purely *functional* and *synchrounous* approach.
- It can be used to develop:
  - *audio effects*,
  - *sound synthesizers*
  - real-time applications processing *signals*.
- Who uses FAUST ?
  - Developers of audio applications and plugins,
  - Sound engineers and musical assistants
  - Researchers in Computer Music

Introduction

Main caracteristics

FAUST is based on several design principles:

- High-level Specification language
- Simple and well defined formal semantics
- Expressive, block-diagram oriented, textual syntax
- Efficient sample level processing
- Fully compiled code
- Automatic parallelization
- Embeddable code (no runtime dependencies, no garbage collection, constant memory and CPU footprint)
- Easy deployment : single code multiple targets (from VST plugins to iPhone or standalone applications)

Introduction

Main caracteristics

FAUST is based on several design principles:

- High-level Specification language
- Simple and well defined formal semantics
- Expressive, block-diagram oriented, textual syntax
- Efficient sample level processing
- Fully compiled code
- Automatic parallelization
- Embeddable code (no runtime dependencies, no garbage collection, constant memory and CPU footprint)
- Easy deployment : single code multiple targets (from VST plugins to iPhone or standalone applications)

Introduction

Main caracteristics

FAUST is based on several design principles:

- High-level Specification language
- Simple and well defined formal semantics
- Expressive, block-diagram oriented, textual syntax
- Efficient sample level processing
- Fully compiled code
- Automatic parallelization
- Embeddable code (no runtime dependencies, no garbage collection, constant memory and CPU footprint)
- Easy deployment : single code multiple targets (from VST plugins to iPhone or standalone applications)

FAUST is based on several design principles:

- High-level Specification language
- Simple and well defined formal semantics
- Expressive, block-diagram oriented, textual syntax
- Efficient sample level processing
- Fully compiled code
- Automatic parallelization
- Embeddable code (no runtime dependencies, no garbage collection, constant memory and CPU footprint)
- Easy deployment : single code multiple targets (from VST plugins to iPhone or standalone applications)

FAUST is based on several design principles:

- High-level Specification language
- Simple and well defined formal semantics
- Expressive, block-diagram oriented, textual syntax
- Efficient sample level processing
- Fully compiled code
- Automatic parallelization
- Embeddable code (no runtime dependencies, no garbage collection, constant memory and CPU footprint)
- Easy deployment : single code multiple targets (from VST plugins to iPhone or standalone applications)

Introduction

Main caracteristics

FAUST is based on several design principles:

- High-level Specification language
- Simple and well defined formal semantics
- Expressive, block-diagram oriented, textual syntax
- Efficient sample level processing
- Fully compiled code
- Automatic parallelization
- Embeddable code (no runtime dependencies, no garbage collection, constant memory and CPU footprint)
- Easy deployment : single code multiple targets (from VST plugins to iPhone or standalone applications)

Introduction

Main caracteristics

FAUST is based on several design principles:

- High-level Specification language
- Simple and well defined formal semantics
- Expressive, block-diagram oriented, textual syntax
- Efficient sample level processing
- Fully compiled code
- Automatic parallelization
- Embeddable code (no runtime dependencies, no garbage collection, constant memory and CPU footprint)
- Easy deployment : single code multiple targets (from VST plugins to iPhone or standalone applications)

Introduction

Main caracteristics

FAUST is based on several design principles:

- High-level Specification language
- Simple and well defined formal semantics
- Expressive, block-diagram oriented, textual syntax
- Efficient sample level processing
- Fully compiled code
- Automatic parallelization
- Embeddable code (no runtime dependencies, no garbage collection, constant memory and CPU footprint)
- Easy deployment : single code multiple targets (from VST plugins to iPhone or standalone applications)

Introduction

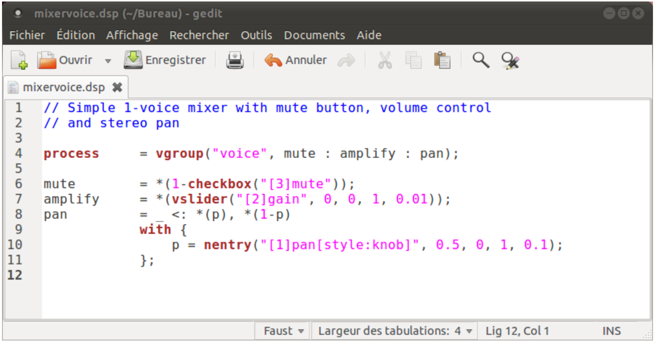
Main caracteristics

FAUST is based on several design principles:

- High-level Specification language
- Simple and well defined formal semantics
- Expressive, block-diagram oriented, textual syntax
- Efficient sample level processing
- Fully compiled code
- Automatic parallelization
- Embeddable code (no runtime dependencies, no garbage collection, constant memory and CPU footprint)
- Easy deployment : single code multiple targets (from VST plugins to iPhone or standalone applications)

Introduction

Example of FAUST program


mixervoice.dsp (~/Bureau) - gedit
Fichier Édition Affichage Rechercher Outils Documents Aide
Ouvrir
Enregistrer
Annuler
mixervoice.dsp
Faust
Largeur des tabulations: 4
Lig 12, Col 1
INS

```
// Simple 1-voice mixer with mute button, volume control
// and stereo pan

process     = vgroup("voice", mute : amplify : pan);

mute        = *(1-checkbox("[3]mute"));
amplify     = *(vslider("[2]gain", 0, 0, 1, 0.01));
pan         = _ <: *(p), *(1-p)
            with {
                p = nentry("[1]pan[style:knob]", 0.5, 0, 1, 0.1);
            };
```

Figure: Source code of a simple mixer channel

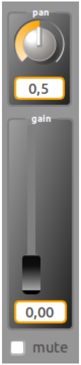


Figure: Resulting application

Introduction

Semantics

A FAUST program describes a *signal processor* :

- A (periodically sampled) *signal* is a *time* to *samples* function:
  - $\mathbb{S} = \mathbb{N} \to \mathbb{Z}$ (int signals)
  - $\mathbb{S} = \mathbb{N} \to \mathbb{R}$ (float signals)
- A *signal processor* is a *signals* to *signals* function:
  - $\mathbb{P} = \mathbb{S}^n \to \mathbb{S}^m$
- Everything in FAUST is a *signal processor* :
  - $+ : \mathbb{S}^2 \to \mathbb{S}^1 \in \mathbb{P}$,
  - $3.14 : \mathbb{S}^0 \to \mathbb{S}^1 \in \mathbb{P}, \ldots,$
- Programming in FAUST is essentially combining signal processors :
  - $\{ : \;\; , \;\; <: \;\; :> \;\; \sim \} \subset \mathbb{P} \times \mathbb{P} \to \mathbb{P}$

Introduction

Semantics

A FAUST program describes a *signal processor* :

- A (periodically sampled) *signal* is a *time* to *samples* function:
  - $\mathbb{S} = \mathbb{N} \rightarrow \mathbb{Z}$ (int signals)
  - $\mathbb{S} = \mathbb{N} \rightarrow \mathbb{R}$ (float signals)
- A *signal processor* is a *signals* to *signals* function:
  - $\mathbb{P} = \mathbb{S}^n \rightarrow \mathbb{S}^m$
- Everything in FAUST is a *signal processor* :
  - $+ : \mathbb{S}^2 \rightarrow \mathbb{S}^1 \in \mathbb{P}$,
  - $3.14 : \mathbb{S}^0 \rightarrow \mathbb{S}^1 \in \mathbb{P}, \ldots,$
- Programming in FAUST is essentially combining signal processors :
  - $\{: \;\; , \;\; <: \;\; :> \;\; \tilde{} \;\} \subset \mathbb{P} \times \mathbb{P} \rightarrow \mathbb{P}$

Introduction

Semantics

A FAUST program describes a *signal processor* :

- A (periodically sampled) *signal* is a *time* to *samples* function:
  - $\mathbb{S} = \mathbb{N} \to \mathbb{Z}$ (int signals)
  - $\mathbb{S} = \mathbb{N} \to \mathbb{R}$ (float signals)
- A *signal processor* is a *signals* to *signals* function:
  - $\mathbb{P} = \mathbb{S}^n \to \mathbb{S}^m$
- Everything in FAUST is a *signal processor* :
  - `+` $: \mathbb{S}^2 \to \mathbb{S}^1 \in \mathbb{P}$,
  - `3.14` $: \mathbb{S}^0 \to \mathbb{S}^1 \in \mathbb{P}, \ldots,$
- Programming in FAUST is essentially combining signal processors :
  - $\{$`:` `,` `<:` `:>` `~`$\} \subset \mathbb{P} \times \mathbb{P} \to \mathbb{P}$

Introduction

Semantics

A FAUST program describes a *signal processor* :

- A (periodically sampled) *signal* is a *time* to *samples* function:
  - $\mathbb{S} = \mathbb{N} \rightarrow \mathbb{Z}$ (int signals)
  - $\mathbb{S} = \mathbb{N} \rightarrow \mathbb{R}$ (float signals)
- A *signal processor* is a *signals* to *signals* function:
  - $\mathbb{P} = \mathbb{S}^n \rightarrow \mathbb{S}^m$
- Everything in FAUST is a *signal processor* :
  - $+ : \mathbb{S}^2 \rightarrow \mathbb{S}^1 \in \mathbb{P}$,
  - $\texttt{3.14} : \mathbb{S}^0 \rightarrow \mathbb{S}^1 \in \mathbb{P}, \ldots,$
- Programming in FAUST is essentially combining signal processors :
  - $\{ \texttt{:} \;\; \texttt{,} \;\; \texttt{<:} \;\; \texttt{:>} \;\; \texttt{\~{}} \} \subset \mathbb{P} \times \mathbb{P} \rightarrow \mathbb{P}$

Introduction

Semantics

A FAUST program describes a *signal processor* :

- A (periodically sampled) *signal* is a *time* to *samples* function:
  - $\mathbb{S} = \mathbb{N} \rightarrow \mathbb{Z}$ (int signals)
  - $\mathbb{S} = \mathbb{N} \rightarrow \mathbb{R}$ (float signals)
- A *signal processor* is a *signals* to *signals* function:
  - $\mathbb{P} = \mathbb{S}^n \rightarrow \mathbb{S}^m$
- Everything in FAUST is a *signal processor* :
  - `+` $: \mathbb{S}^2 \rightarrow \mathbb{S}^1 \in \mathbb{P}$,
  - `3.14` $: \mathbb{S}^0 \rightarrow \mathbb{S}^1 \in \mathbb{P}, \ldots,$
- Programming in FAUST is essentially combining signal processors :
  - $\{$`:` `,` `<:` `:>` `~`$\} \subset \mathbb{P} \times \mathbb{P} \rightarrow \mathbb{P}$

Introduction

Semantics

A FAUST program describes a *signal processor* :

- A (periodically sampled) *signal* is a *time* to *samples* function:
  - $\mathbb{S} = \mathbb{N} \to \mathbb{Z}$ (int signals)
  - $\mathbb{S} = \mathbb{N} \to \mathbb{R}$ (float signals)
- A *signal processor* is a *signals* to *signals* function:
  - $\mathbb{P} = \mathbb{S}^n \to \mathbb{S}^m$
- Everything in FAUST is a *signal processor* :
  - $\texttt{+} : \mathbb{S}^2 \to \mathbb{S}^1 \in \mathbb{P}$,
  - $\texttt{3.14} : \mathbb{S}^0 \to \mathbb{S}^1 \in \mathbb{P}, \ldots,$
- Programming in FAUST is essentially combining signal processors :
  - $\{\texttt{:}\ \ \texttt{,}\ \ \texttt{<:}\ \ \texttt{:>}\ \ \texttt{\~{}}\ \} \subset \mathbb{P} \times \mathbb{P} \to \mathbb{P}$

Block-Diagram Algebra

Programming by patching

Figure: the Moog modular synthesizer

Block-Diagram Algebra

Faust syntax is based on a *block diagram algebra*

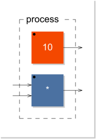
5 Composition Operators

- `(A,B)` parallel composition
- `(A:B)` sequential composition
- `(A<:B)` split composition
- `(A:>B)` merge composition
- `(A~B)` recursive composition

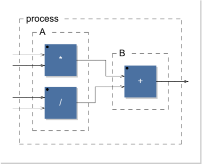
2 Constants

- `!` cut
- `_` wire

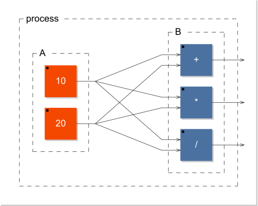
Block-Diagram Algebra



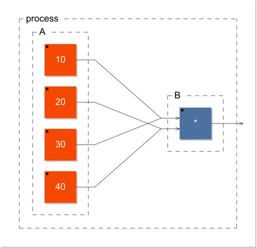
par: (10,*)



seq: ((*,/):+)

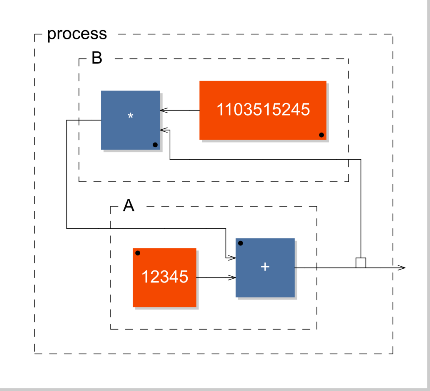


split: ((10,20) <: (+,*,/))



merge: ((10,20,30,40) :> *)

Block-Diagram Algebra



rec: `+(12345) ~ *(1103515245)`

Faust Architecture System

Motivations

- Easy deployment (one Faust code, multiple audio targets) is an essential feature of the Faust project
- This is why Faust programs say nothing about audio drivers or GUI toolkits to be used.
- There is a *separation of concerns* between the audio computation itself, and its usage.

Faust Architecture System

Motivations

- Easy deployment (one Faust code, multiple audio targets) is an essential feature of the Faust project
- This is why Faust programs say nothing about audio drivers or GUI toolkits to be used.
- There is a *separation of concerns* between the audio computation itself, and its usage.

Faust Architecture System

Motivations

- Easy deployment (one Faust code, multiple audio targets) is an essential feature of the Faust project
- This is why Faust programs say nothing about audio drivers or GUI toolkits to be used.
- There is a *separation of concerns* between the audio computation itself, and its usage.

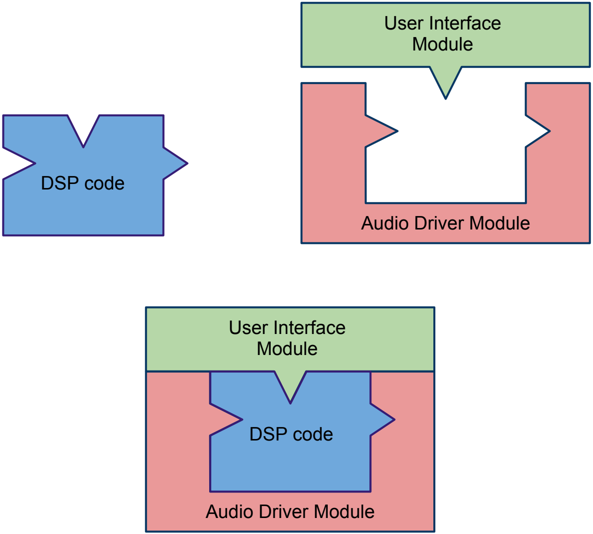
Faust Architecture System

Motivations

- Easy deployment (one Faust code, multiple audio targets) is an essential feature of the Faust project
- This is why Faust programs say nothing about audio drivers or GUI toolkits to be used.
- There is a *separation of concerns* between the audio computation itself, and its usage.

Faust Architecture System

The *architecture file* describes how to connect the audio computation to the external world.



Faust Architecture System

Examples of supported architectures

- Audio plugins :
  - LADSPA
  - DSSI
  - LV2
  - Max/MSP
  - VST
  - PD
  - CSound
  - Supercollider
  - Pure
  - Chuck
  - Octave
  - Flash
- Audio drivers :
  - Jack
  - Alsa
  - CoreAudio
- Graphic User Interfaces :
  - QT
  - GTK
  - iOS5
- Other User Interfaces :
  - OSC
  - HTTPD

2-HTTP based Audio Apps

3-Online Compiler

4-Javascript backend

5-Perspectives

Perspectives

- FAUST
  - Faciliter la publication Web
  - Faciliter la réutilisation (à la javascript)
  - Utiliser des URL pour les composants et les librairies
- Architecture Httpd
  - Développer l'interface utilisateur (HTML5/JS/CCS)
  - Différentier les accès administrateur et public
  - Integrer des QR Codes
- Compilateur en ligne
  - Séparer le compilateur en ligne et le site
  - Développer une API pour le compilateur en ligne
  - Intégrer la compilation vers Javascript

Perspectives

- FAUST
  - Faciliter la publication Web
  - Faciliter la réutilisation (à la javascript)
  - Utiliser des URL pour les composants et les librairies
- Architecture Httpd
  - Développer l'interface utilisateur (HTML5/JS/CCS)
  - Différentier les accès administrateur et public
  - Integrer des QR Codes
- Compilateur en ligne
  - Séparer le compilateur en ligne et le site
  - Développer une API pour le compilateur en ligne
  - Intégrer la compilation vers Javascript

Perspectives

- FAUST
  - Faciliter la publication Web
  - Faciliter la réutilisation (à la javascript)
  - Utiliser des URL pour les composants et les librairies
- Architecture Httpd
  - Développer l'interface utilisateur (HTML5/JS/CCS)
  - Différentier les accès administrateur et public
  - Integrer des QR Codes
- Compilateur en ligne
  - Séparer le compilateur en ligne et le site
  - Développer une API pour le compilateur en ligne
  - Intégrer la compilation vers Javascript

Perspectives

- FAUST
  - Faciliter la publication Web
  - Faciliter la réutilisation (à la javascript)
  - Utiliser des URL pour les composants et les librairies
- Architecture Httpd
  - Développer l'interface utilisateur (HTML5/JS/CCS)
  - Différentier les accès administrateur et public
  - Integrer des QR Codes
- Compilateur en ligne
  - Séparer le compilateur en ligne et le site
  - Développer une API pour le compilateur en ligne
  - Intégrer la compilation vers Javascript