FAUST
Programming language for audio and signal processing

Yann Orlarey

GRAME – Centre National de Création Musicale

November 28, 2013, GdT Programmation
1-Introduction
Introduction
What is FAUST?

FAUST stands for Functional AUdio STream:

- It is a Domain-Specific Language for real-time audio signal processing and synthesis.
- 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*:

- It is a *Domain-Specific Language* for real-time audio signal processing and synthesis.
- 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*:

- It is a *Domain-Specific Language* for real-time audio signal processing and synthesis.
- 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*:

- It is a *Domain-Specific Language* for real-time audio signal processing and synthesis.
- 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*:

- It is a *Domain-Specific Language* for real-time audio signal processing and synthesis.
- 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*:

- It is a *Domain-Specific Language* for real-time audio signal processing and synthesis.
- 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*:

- It is a *Domain-Specific Language* for real-time audio signal processing and synthesis.
- 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*:

- It is a *Domain-Specific Language* for real-time audio signal processing and synthesis.
- 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*:

- It is a *Domain-Specific Language* for real-time audio signal processing and synthesis.
- 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*:

- It is a *Domain-Specific Language* for real-time audio signal processing and synthesis.
- 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*:

- It is a *Domain-Specific Language* for real-time audio signal processing and synthesis.
- 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 goals of the FAUST project
Main Goals

Notation

- How to easily describe dsp algorithms with a high level, expressive and modular notation?
- By using a purely functional approach based on a block diagram algebra
- White noise formula
  - Mathematical notation:
    \[
    x(n) = x(n-1) \ast 1103515245 + 12345 \\
    y(n) = x(n) / 2147483647.0
    \]
  - Faust notation:
    \%(12345) ~ *(1103515245) : /2147483647.0 (%)
Main Goals

Notation

- How to easily describe dsp algorithms with a high level, expressive and modular notation?
- By using a purely functional approach based on a block diagram algebra

White noise formula

- Mathematical notation:
  \[ x(n) = x(n-1) \times 1103515245 + 12345 \]
  \[ y(n) = x(n) / 2147483647.0 \]
- Faust notation:
  +(12345) ~ *(1103515245) : /(2147483647.0)
Main Goals

Notation

- How to easily describe dsp algorithms with a high level, expressive and modular notation?
- By using a purely functional approach based on a block diagram algebra
- *White noise formula*
  
  - Mathematical notation:
    
    \[
    x(n) = x(n-1) \times 1103515245 + 12345 \\
    y(n) = x(n) / 2147483647.0
    \]
  
  - Faust notation:
    
    \[
    +(12345) \sim *(1103515245) : / (2147483647.0)
    \]
Main Goals

Notation

- How to easily describe dsp algorithms with a high level, expressive and modular notation?
- By using a purely functional approach based on a block diagram algebra
- White noise formula
  
  - **Mathematical notation**:
    
    \[ x(n) = x(n-1) \times 1103515245 + 12345 \]
    
    \[ y(n) = x(n) / 2147483647.0 \]

  - **Faust notation**:
    
    +(12345) ~ *(1103515245) : /(2147483647.0)
Main Goals

Notation

- How to easily describe dsp algorithms with a high level, expressive and modular notation?
- By using a purely functional approach based on a block diagram algebra
- *White noise formula*
  - **Mathematical notation**:
    \[
    x(n) = x(n-1) \times 1103515245 + 12345 \\
    y(n) = \frac{x(n)}{2147483647.0}
    \]
  - **Faust notation**:
    \[+(12345) \sim *(1103515245) : / (2147483647.0)\]
Main Goals

Efficiency

- How to implement these algorithms with an efficiency comparable to low level languages like C?

- By automatically translating FAUST programs to highly optimized imperative programs. Several backends are available:
  - C++
  - C
  - Java
  - Javascript
  - LLVM

- By automatic parallelization:
  - OpenMP
  - Work Stealing
Main Goals

Efficiency

- How to implement these algorithms with an efficiency comparable to low level languages like C ?
- By automatically translating FAUST programs to highly optimized imperative programs. Several backends are available :
  - C++
  - C
  - Java
  - Javascript
  - LLVM
- By automatic parallelization :
  - OpenMP
  - Work Stealing
Main Goals

Efficiency

- **How to implement these algorithms with an efficiency comparable to low level languages like C?**
- By automatically translating FAUST programs to highly optimized imperative programs. Several backends are available:
  - C++
  - C
  - Java
  - Javascript
  - LLVM
- By automatic parallelization:
  - OpenMP
  - Work Stealing
Main Goals

Efficiency

- How to implement these algorithms with an efficiency comparable to low level languages like C?
- By automatically translating FAUST programs to highly optimized imperative programs. Several backends are available:
  - C++
  - C
  - Java
  - Javascript
  - LLVM
- By automatic parallelization:
  - OpenMP
  - Work Stealing
Main Goals

Efficiency

- How to implement these algorithms with an efficiency comparable to low level languages like C?
- By automatically translating FAUST programs to highly optimized imperative programs. Several backends are available:
  - C++
  - C
  - Java
  - Javascript
  - LLVM
- By automatic parallelization:
  - OpenMP
  - Work Stealing
Main Goals

Efficiency

- How to implement these algorithms with an efficiency comparable to low level languages like C?
- By automatically translating FAUST programs to highly optimized imperative programs. Several backends are available:
  - C++
  - C
  - Java
  - Javascript
  - LLVM
- By automatic parallelization:
  - OpenMP
  - Work Stealing
Main Goals

Efficiency

- How to implement these algorithms with an efficiency comparable to low level languages like C?
- By automatically translating FAUST programs to highly optimized imperative programs. Several backends are available:
  - C++
  - C
  - Java
  - Javascript
  - LLVM
- By automatic parallelization:
  - OpenMP
  - Work Stealing
Main Goals

Efficiency

- How to implement these algorithms with an efficiency comparable to low level languages like C?
- By automatically translating FAUST programs to highly optimized imperative programs. Several backends are available:
  - C++
  - C
  - Java
  - Javascript
  - LLVM
- By automatic parallelization:
  - OpenMP
  - Work Stealing
Main Goals

Efficiency

- How to implement these algorithms with an efficiency comparable to low level languages like C?
- By automatically translating FAUST programs to highly optimized imperative programs. Several backends are available:
  - C++
  - C
  - Java
  - Javascript
  - LLVM

- By automatic parallelization:
  - OpenMP
  - Work Stealing
Main Goals

Efficiency

- How to implement these algorithms with an efficiency comparable to low level languages like C?
- By automatically translating FAUST programs to highly optimized imperative programs. Several backends are available:
  - C++
  - C
  - Java
  - Javascript
  - LLVM
- By automatic parallelization:
  - OpenMP
  - Work Stealing
Main Goals

Deployement

- How to transparently deploy these programs on a large variety of software and hardware plateforms, from desktop to mobile devices?
- By a separation of concerns between the audio computation itself (described by the FAUST code), and its relations to the external world (described by an architecture file). Recent additions:
  - Web Audio API
  - iOS
  - Android (Romain Michon)
  - LV2 (Albert Gräf)
  - AudioUnits (Reza Payami)
  - Raspberry PI
Main Goals

Deployement

- How to transparently deploy these programs on a large variety of software and hardware platforms, from desktop to mobile devices?

- By a separation of concerns between the audio computation itself (described by the FAUST code), and its relations to the external world (described by an architecture file). Recent additions:
  - Web Audio API
  - iOS
  - Android (Romain Michon)
  - LV2 (Albert Gräf)
  - AudioUnits (Reza Payami)
  - Raspberry PI
How to transparently deploy these programs on a large variety of software and hardware platforms, from desktop to mobile devices?

By a separation of concerns between the audio computation itself (described by the FAUST code), and its relations to the external world (described by an architecture file). Recent additions:

- Web Audio API
- iOS
- Android (Romain Michon)
- LV2 (Albert Gräf)
- AudioUnits (Reza Payami)
- Raspberry PI
How to transparently deploy these programs on a large variety of software and hardware platforms, from desktop to mobile devices?

By a separation of concerns between the audio computation itself (described by the FAUST code), and its relations to the external world (described by an architecture file). Recent additions:

- Web Audio API
- iOS
- Android (Romain Michon)
- LV2 (Albert Gräf)
- AudioUnits (Reza Payami)
- Raspberry PI
How to transparently deploy these programs on a large variety of software and hardware platforms, from desktop to mobile devices?

By a separation of concerns between the audio computation itself (described by the FAUST code), and its relations to the external world (described by an architecture file). Recent additions:

- Web Audio API
- iOS
- Android (Romain Michon)
- LV2 (Albert Gräf)
- AudioUnits (Reza Payami)
- Raspberry PI
How to transparently deploy these programs on a large variety of software and hardware platforms, from desktop to mobile devices?

By a separation of concerns between the audio computation itself (described by the FAUST code), and its relations to the external world (described by an architecture file). Recent additions:

- Web Audio API
- iOS
- Android (Romain Michon)
- LV2 (Albert Gräf)
- AudioUnits (Reza Payami)
- Raspberry PI
Main Goals

Deployement

- How to transparently deploy these programs on a large variety of software and hardware plateforms, from desktop to mobile devices?

- By a separation of concerns between the audio computation itself (described by the FAUST code), and its relations to the external world (described by an architecture file). Recent additions:
  - Web Audio API
  - iOS
  - Android (Romain Michon)
  - LV2 (Albert Gräf)
  - AudioUnits (Reza Payami)
  - Raspberry PI
Main Goals

Deployement

- How to transparently deploy these programs on a large variety of software and hardware platforms, from desktop to mobile devices?
- By a separation of concerns between the audio computation itself (described by the FAUST code), and its relations to the external world (described by an architecture file). Recent additions:
  - Web Audio API
  - iOS
  - Android (Romain Michon)
  - LV2 (Albert Gräf)
  - AudioUnits (Reza Payami)
  - Raspberry PI
Main Goals

Accessibility

- How to make the FAUST technology easily accessible, including to other applications and music languages?
- By providing, in addition to the FAUST compiler itself and the FaustWorks IDE:
  - Online Compiler (http://faust.grame.fr)
  - libfaust (embeddable Faust Compiler)
    - Faustgen for Max/MSP
    - Faustcompiles, etc. for Csound (V. Lazzarini)
  - Faustnode for the Web Audio API
  - FaustLive
  - FaustWeb (remote compilation service)
Main Goals
Accessibility

- How to make the FAUST technology easily accessible, including to other applications and music languages?
- By providing, in addition to the FAUST compiler itself and the FaustWorks IDE:
  - Online Compiler (http://faust.grame.fr)
  - libfaust (embeddable Faust Compiler)
    - Faustgen for Max/MSP
    - Faustcompile, etc. for Csound (V. Lazzarini)
    - Faustnode for the Web Audio API
    - FaustLive
  - FaustWeb (remote compilation service)
Main Goals

Accessibility

How to make the FAUST technology easily accessible, including to other applications and music languages?

By providing, in addition to the FAUST compiler itself and the FaustWorks IDE

- Online Compiler (http://faust.grame.fr)
- libfaust (embeddable Faust Compiler)
  - Faustgen for Max/MSP
  - Faustcompile, etc. for Csound (V. Lazzarini)
  - Faustnode for the Web Audio API
  - FaustLive
- FaustWeb (remote compilation service)
Main Goals

Accessibility

- **How to make the FAUST technology easily accessible, including to other applications and music languages?**
- By providing, in addition to the FAUST compiler itself and the FaustWorks IDE
  - Online Compiler (http://faust.grame.fr)
  - libfaust (embeddable Faust Compiler)
    - Faustgen for Max/MSP
    - Faustcompile, etc. for Csound (V. Lazzarini)
    - Faustnode for the Web Audio API
    - FaustLive
  - FaustWeb (remote compilation service)
How to make the FAUST technology easily accessible, including to other applications and music languages?

By providing, in addition to the FAUST compiler itself and the FaustWorks IDE:

- Online Compiler (http://faust.gram.fr)
- libfaust (embeddable Faust Compiler)
  - Faustgen for Max/MSP
  - Faustcompile, etc. for Csound (V. Lazzarini)
  - Faustnode for the Web Audio API
  - FaustLive
- FaustWeb (remote compilation service)
Main Goals

Accessibility

- How to make the FAUST technology easily accessible, including to other applications and music languages?
- By providing, in addition to the FAUST compiler itself and the FaustWorks IDE
  - Online Compiler (http://faust.grame.fr)
  - libfaust (embeddable Faust Compiler)
    - Faustgen for Max/MSP
    - Faustcompile, etc. for Csound (V. Lazzarini)
  - Faustnode for the Web Audio API
  - FaustLive
  - FaustWeb (remote compilation service)
Main Goals

Accessibility

- How to make the FAUST technology easily accessible, including to other applications and music languages?
- By providing, in addition to the FAUST compiler itself and the FaustWorks IDE
  - Online Compiler (http://faust.grame.fr)
  - libfaust (embeddable Faust Compiler)
    - Faustgen for Max/MSP
    - Faustcompile, etc. for Csound (V. Lazzarini)
    - Faustnode for the Web Audio API
  - FaustLive
  - FaustWeb (remote compilation service)
Main Goals

Accessibility

- How to make the FAUST technology easily accessible, including to other applications and music languages?
- By providing, in addition to the FAUST compiler itself and the FaustWorks IDE:
  - Online Compiler (http://faust.grame.fr)
  - libfaust (embeddable Faust Compiler)
    - Faustgen for Max/MSP
    - Faustcompile, etc. for Csound (V. Lazzarini)
    - Faustnode for the Web Audio API
    - FaustLive
  - FaustWeb (remote compilation service)
Main Goals

Accessibility

- **How to make the FAUST technology easily accessible, including to other applications and music languages?**
- By providing, in addition to the FAUST compiler itself and the FaustWorks IDE
  - Online Compiler (http://faust.grame.fr)
  - libfaust (embeddable Faust Compiler)
    - Faustgen for Max/MSP
    - Faustcompile, etc. for Csound (V. Lazzarini)
    - Faustnode for the Web Audio API
    - FaustLive
  - FaustWeb (remote compilation service)
Main Goals

Preservation

How to preserve these programs on the long term?

- Preservation by abstraction (projet ASTREE ANR-2008-CORD-003). We abstract the programming language and keep the mathematical semantics. We generate a complete mathematical description of a FAUST program.

- from faust expression
  \[ + (12345) \sim \ast (1103515245) : /(2147483647.0) \]

- we automatically infer the mathematical equations:
  \[ y(t) = 4.6566128752458 \ast 10^{-10} \ast r1(t) \text{ and } r1(t) = 12345 + 1103515245 \ast r1(t - 1) \]
Main Goals

Preservation

- How to preserve these programs on the long term?
- Preservation by abstraction (projet ASTREE ANR-2008-CORD-003). We abstract the programming language and keep the mathematical semantics. We generate a complete mathematical description of a FAUST program.

- from faust expression
  
  \[(12345) \sim (1103515245) : (2147483647.0)\]

- we automatically infer the mathematical equations:

  \[y(t) = 4.6566128752458 \times 10^{-10} \times r1(t)\]  
  
  \[r1(t) = 12345 + 1103515245 \times r1(t - 1)\]
How to preserve these programs on the long term?

Preservation by abstraction (projet ASTREE ANR-2008-CORD-003). We abstract the programming language and keep the mathematical semantics. We generate a complete mathematical description of a FAUST program.

- From Faust expression
  \[(12345) \sim (1103515245) : (2147483647.0)\]

  - We automatically infer the mathematical equations:
    \[y(t) = 4.6566128752458 \times 10^{-10} \times r1(t) \text{ and } r1(t) = 12345 + 1103515245 \times r1(t - 1)\]
How to preserve these programs on the long term?

Preservation by abstraction (projet ASTREE ANR-2008-CORD-003). We abstract the programming language and keep the mathematical semantics. We generate a complete mathematical description of a FAUST program.

From Faust expression:
\[(+\left(12345\right) \sim \left(1103515245\right) : \left(2147483647.0\right))\]

We automatically infer the mathematical equations:
\[y(t) = 4.6566128752458 \times 10^{-10} \times r1(t) \text{ and } \]
\[r1(t) = 12345 + 1103515245 \times r1(t - 1)\]
Introduction
What is a FAUST program?

A FAUST program describes a signal processor:

- A signal processor is a mathematical function that maps input signals to output signals:
  \[ P = S^n \rightarrow S^m \]
- A (periodically sampled) signal is a time to samples function:
  \[ S = N \rightarrow R \]
- Everything in FAUST is a signal processor:
  \[ + : S^2 \rightarrow S^1 \in P, \]
  \[ 3.14 : S^0 \rightarrow S^1 \in P, \ldots, \]
- Programming in FAUST is essentially combining signal processors:
  \[ \{ ; , <: :> \sim \} \subset P \times P \rightarrow P \]
A FAUST program describes a signal processor:

- A signal processor is a mathematical function that maps input signals to output signals:
  - $P = S^n \rightarrow S^m$
- A (periodically sampled) signal is a time to samples function:
  - $S = N \rightarrow R$
- Everything in FAUST is a signal processor:
  - $+: S^2 \rightarrow S^1 \in P$, $3.14 : S^0 \rightarrow S^1 \in P$, ...
- Programming in FAUST is essentially combining signal processors:
  - $\{ :, <: :, \rightarrow, \sim \} \subset P \times P \rightarrow P$
A FAUST program describes a signal processor:

- A signal processor is a mathematical function that maps input signals to output signals:
  \[ P = S^n \rightarrow S^m \]
- A (periodically sampled) signal is a time to samples function:
  \[ S = \mathbb{N} \rightarrow \mathbb{R} \]
- Everything in FAUST is a signal processor:
  \[ + : S^2 \rightarrow S^1 \in P, \]
  \[ 3.14 : S^0 \rightarrow S^1 \in P, \ldots, \]
- Programming in FAUST is essentially combining signal processors:
  \[ \{ , , <: :> \sim \} \subset P \times P \rightarrow P \]
Introduction
What is a FAUST program?

A FAUST program describes a *signal processor*:

- A *signal processor* is a mathematical function that maps input *signals* to output *signals*:
  
  \[ P = S^n \rightarrow S^m \]

- A (periodically sampled) *signal* is a *time to samples* function:
  
  \[ S = N \rightarrow R \]

- Everything in FAUST is a *signal processor*:
  
  \[ + : S^2 \rightarrow S^1 \in P, \]
  
  \[ 3.14 : S^0 \rightarrow S^1 \in P, \ldots, \]

- Programming in FAUST is essentially combining signal processors:
  
  \[ \{ : , <: , :> , ~ \} \subset P \times P \rightarrow P \]
Introduction

What is a FAUST program?

A FAUST program describes a *signal processor*:

- A *signal processor* is a mathematical function that maps input *signals* to output *signals*:
  - \( \mathbb{P} = S^n \rightarrow S^m \)

- A (periodically sampled) *signal* is a *time to samples* function:
  - \( S = \mathbb{N} \rightarrow \mathbb{R} \)

- Everything in FAUST is a *signal processor*:
  - \( + : S^2 \rightarrow S^1 \in \mathbb{P}, \)
  - \( 3.14 : S^0 \rightarrow S^1 \in \mathbb{P}, \ldots, \)

- Programming in FAUST is essentially combining signal processors:
  - \( \{ :, \ <:: \ :> \ :~ \} \subset \mathbb{P} \times \mathbb{P} \rightarrow \mathbb{P} \)
Introduction

What is a FAUST program?

A FAUST program describes a **signal processor**: 

- A **signal processor** is a mathematical function that maps input *signals* to output *signals*:  
  \[ P = S^n \rightarrow S^m \]
- A (periodically sampled) *signal* is a *time to samples* function:  
  \[ S = N \rightarrow \mathbb{R} \]
- Everything in FAUST is a **signal processor**:  
  \[ + : S^2 \rightarrow S^1 \in P, \]
  \[ 3.14 : S^0 \rightarrow S^1 \in P, \ldots, \]
- Programming in FAUST is essentially combining signal processors:  
  \[ \{ :, <: :> ~ \} \subset P \times P \rightarrow P \]
Introduction

Example of signal processor

- A digital signal processor, here a Lexicon 300, can be modeled as a *mathematical function* transforming *input signals* into *output signals*.
- FAUST allows to describe both the *mathematical computation* and the *user interface*. 
Introduction

Example of signal processor

- A digital signal processor, here a Lexicon 300, can be modeled as a *mathematical function* transforming *input signals* into *output signals*.

- FAUST allows to describe both the *mathematical computation* and the *user interface*. 
A digital signal processor, here a Lexicon 300, can be modeled as a *mathematical function* transforming *input signals* into *output signals*.

FAUST allows to describe both the *mathematical computation* and the *user interface*. 
Introduction
A simple FAUST program

Figure: Source code of a simple 1-voice mixer

```cpp
// 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: Resulting application
Introduction
Main characteristics

FAUST is based on several design principles:

- High-level Specification language
- Purely functional approach
- Textual, block-diagram oriented, syntax
- Efficient sample level processing
- Fully compiled code (sequential or parallel)
- Embeddable code (no runtime dependences, constant memory and CPU footprint)
- Easy deployment: single code multiple targets (from VST plugins to iPhone or standalone applications)
Introduction

Main characteristics

**FAUST is based on several design principles:**

- **High-level Specification language**
- Purely functional approach
- Textual, block-diagram oriented, syntax
- Efficient sample level processing
- Fully compiled code (sequential or parallel)
- Embeddable code (no runtime dependences, constant memory and CPU footprint)
- Easy deployment: single code multiple targets (from VST plugins to iPhone or standalone applications)
Introduction
Main characteristics

FAUST is based on several design principles:

- High-level Specification language
- Purely functional approach
  - Textual, block-diagram oriented, syntax
  - Efficient sample level processing
  - Fully compiled code (sequential or parallel)
  - Embeddable code (no runtime dependences, constant memory and CPU footprint)
- Easy deployment: single code multiple targets (from VST plugins to iPhone or standalone applications)
Introduction
Main characteristics

FAUST is based on several design principles:

- High-level Specification language
- Purely functional approach
- Textual, block-diagram oriented, syntax
- Efficient sample level processing
- Fully compiled code (sequential or parallel)
- Embeddable code (no runtime dependences, 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
- Purely functional approach
- Textual, block-diagram oriented, syntax
- Efficient sample level processing
  - Fully compiled code (sequential or parallel)
  - Embeddable code (no runtime dependences, 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
- Purely functional approach
- Textual, block-diagram oriented, syntax
- Efficient sample level processing
- Fully compiled code (sequential or parallel)
- Embeddable code (no runtime dependences, 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
- Purely functional approach
- Textual, block-diagram oriented, syntax
- Efficient sample level processing
- Fully compiled code (sequential or parallel)
- Embeddable code (no runtime dependences, constant memory and CPU footprint)
- Easy deployment: single code multiple targets (from VST plugins to iPhone or standalone applications)
Introduction

Main characteristics

FAUST is based on several design principles:

- High-level Specification language
- Purely functional approach
- Textual, block-diagram oriented, syntax
- Efficient sample level processing
- Fully compiled code (sequential or parallel)
- Embeddable code (no runtime dependences, constant memory and CPU footprint)
- Easy deployment: single code multiple targets (from VST plugins to iPhone or standalone applications)
2-Block Diagram Algebra
Block-Diagram Algebra

Programming by patching is familiar to musicians:
Block-Diagram Algebra

Today programming by patching is widely used in Visual Programming Languages like Max/MSP:

Figure: Block-diagrams can be a mess
Block-Diagram Algebra

Faust allows structured block-diagrams

Figure: A complex but structured block-diagram
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
The *parallel composition* \((A, B)\) is probably the simplest one. It places the two block-diagrams one on top of the other, without connections.

**Figure**: Example of parallel composition \((10, *)\)
The *sequential composition* \((A : B)\) connects the outputs of \(A\) to the inputs of \(B\). \(A[0]\) is connected to \([0]B\), \(A[1]\) is connected to \([1]B\), and so on.

**Figure:** Example of sequential composition \(((*,/) : +)\)
The *split composition* \((A <: B)\) operator is used to distribute \(A\) outputs to \(B\) inputs.

**Figure:** example of split composition \(((10, 20) <: (+, *, /))\)
The *merge composition* \((A :> B)\) is used to connect several outputs of \(A\) to the same inputs of \(B\).

*Figure: example of merge composition \(((10,20,30,40) :> *)\)*
The *recursive composition* $(A \sim B)$ is used to create cycles in the block-diagram in order to express recursive computations.

**Figure:** example of recursive composition $+(12345) \sim *(1103515245)$
3-Primitive operations
Faust Primitives

Arithmetic operations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>$S^2 \rightarrow S^1$</td>
<td>addition: $y(t) = x_1(t) + x_2(t)$</td>
</tr>
<tr>
<td>-</td>
<td>$S^2 \rightarrow S^1$</td>
<td>subtraction: $y(t) = x_1(t) - x_2(t)$</td>
</tr>
<tr>
<td>*</td>
<td>$S^2 \rightarrow S^1$</td>
<td>multiplication: $y(t) = x_1(t) \times x_2(t)$</td>
</tr>
<tr>
<td>∧</td>
<td>$S^2 \rightarrow S^1$</td>
<td>power: $y(t) = x_1(t)^{x_2(t)}$</td>
</tr>
<tr>
<td>/</td>
<td>$S^2 \rightarrow S^1$</td>
<td>division: $y(t) = x_1(t)/x_2(t)$</td>
</tr>
<tr>
<td>%</td>
<td>$S^2 \rightarrow S^1$</td>
<td>modulo: $y(t) = x_1(t) % x_2(t)$</td>
</tr>
<tr>
<td>int</td>
<td>$S^1 \rightarrow S^1$</td>
<td>cast into an int signal: $y(t) = (\text{int})x(t)$</td>
</tr>
<tr>
<td>float</td>
<td>$S^1 \rightarrow S^1$</td>
<td>cast into an float signal: $y(t) = (\text{float})x(t)$</td>
</tr>
</tbody>
</table>
## Faust Primitives

### Bitwise operations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>$S^2 \rightarrow S^1$</td>
<td>logical AND: $y(t) = x_1(t) &amp; x_2(t)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>logical OR: $y(t) = x_1(t)</td>
</tr>
<tr>
<td>xor</td>
<td>$S^2 \rightarrow S^1$</td>
<td>logical XOR: $y(t) = x_1(t) \land x_2(t)$</td>
</tr>
<tr>
<td>&lt;&lt;</td>
<td>$S^2 \rightarrow S^1$</td>
<td>arith. shift left: $y(t) = x_1(t) &lt;&lt; x_2(t)$</td>
</tr>
<tr>
<td>&gt;&gt;</td>
<td>$S^2 \rightarrow S^1$</td>
<td>arith. shift right: $y(t) = x_1(t) &gt;&gt; x_2(t)$</td>
</tr>
</tbody>
</table>
# Faust Primitives

## Comparison operations

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>$S^2 \rightarrow S^1$</td>
<td>less than: $y(t) = x_1(t) &lt; x_2(t)$</td>
</tr>
<tr>
<td>&lt;=</td>
<td>$S^2 \rightarrow S^1$</td>
<td>less or equal: $y(t) = x_1(t) \leq x_2(t)$</td>
</tr>
<tr>
<td>&gt;</td>
<td>$S^2 \rightarrow S^1$</td>
<td>greater than: $y(t) = x_1(t) &gt; x_2(t)$</td>
</tr>
<tr>
<td>&gt;=</td>
<td>$S^2 \rightarrow S^1$</td>
<td>greater or equal: $y(t) = x_1(t) \geq x_2(t)$</td>
</tr>
<tr>
<td>==</td>
<td>$S^2 \rightarrow S^1$</td>
<td>equal: $y(t) = x_1(t) = x_2(t)$</td>
</tr>
<tr>
<td>!=</td>
<td>$S^2 \rightarrow S^1$</td>
<td>different: $y(t) = x_1(t) \neq x_2(t)$</td>
</tr>
</tbody>
</table>
## Faust Primitives

### Trigonometric functions

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>acos</td>
<td>$S^1 \rightarrow S^1$</td>
<td>arc cosine: $y(t) = \text{acosf}(x(t))$</td>
</tr>
<tr>
<td>asin</td>
<td>$S^1 \rightarrow S^1$</td>
<td>arc sine: $y(t) = \text{asinf}(x(t))$</td>
</tr>
<tr>
<td>atan</td>
<td>$S^1 \rightarrow S^1$</td>
<td>arc tangent: $y(t) = \text{atanf}(x(t))$</td>
</tr>
<tr>
<td>atan2</td>
<td>$S^2 \rightarrow S^1$</td>
<td>arc tangent of 2 signals: $y(t) = \text{atan2f}(x_1(t), x_2(t))$</td>
</tr>
<tr>
<td>cos</td>
<td>$S^1 \rightarrow S^1$</td>
<td>cosine: $y(t) = \text{cosf}(x(t))$</td>
</tr>
<tr>
<td>sin</td>
<td>$S^1 \rightarrow S^1$</td>
<td>sine: $y(t) = \text{sinf}(x(t))$</td>
</tr>
<tr>
<td>tan</td>
<td>$S^1 \rightarrow S^1$</td>
<td>tangent: $y(t) = \text{tanf}(x(t))$</td>
</tr>
<tr>
<td>Syntax</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>exp</td>
<td>$S_1 \rightarrow S_1$</td>
<td>base-e exponential: $y(t) = \expf(x(t))$</td>
</tr>
<tr>
<td>log</td>
<td>$S_1 \rightarrow S_1$</td>
<td>base-e logarithm: $y(t) = \logf(x(t))$</td>
</tr>
<tr>
<td>log10</td>
<td>$S_1 \rightarrow S_1$</td>
<td>base-10 logarithm: $y(t) = \log10f(x(t))$</td>
</tr>
<tr>
<td>pow</td>
<td>$S_2 \rightarrow S_1$</td>
<td>power: $y(t) = \powf(x_1(t), x_2(t))$</td>
</tr>
<tr>
<td>sqrt</td>
<td>$S_1 \rightarrow S_1$</td>
<td>square root: $y(t) = \sqrtf(x(t))$</td>
</tr>
<tr>
<td>abs</td>
<td>$S_1 \rightarrow S_1$</td>
<td>absolute value (int): $y(t) = \text{abs}(x(t))$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>absolute value (float): $y(t) = \text{fabsf}(x(t))$</td>
</tr>
<tr>
<td>min</td>
<td>$S_2 \rightarrow S_1$</td>
<td>minimum: $y(t) = \text{min}(x_1(t), x_2(t))$</td>
</tr>
<tr>
<td>max</td>
<td>$S_2 \rightarrow S_1$</td>
<td>maximum: $y(t) = \text{max}(x_1(t), x_2(t))$</td>
</tr>
<tr>
<td>fmod</td>
<td>$S_2 \rightarrow S_1$</td>
<td>float modulo: $y(t) = \text{fmodf}(x_1(t), x_2(t))$</td>
</tr>
<tr>
<td>remainder</td>
<td>$S_2 \rightarrow S_1$</td>
<td>float remainder: $y(t) = \text{remainderf}(x_1(t), x_2(t))$</td>
</tr>
<tr>
<td>floor</td>
<td>$S_1 \rightarrow S_1$</td>
<td>largest int $\leq$: $y(t) = \floorf(x(t))$</td>
</tr>
<tr>
<td>ceil</td>
<td>$S_1 \rightarrow S_1$</td>
<td>smallest int $\geq$: $y(t) = \ceilf(x(t))$</td>
</tr>
<tr>
<td>rint</td>
<td>$S_1 \rightarrow S_1$</td>
<td>closest int: $y(t) = \rintf(x(t))$</td>
</tr>
</tbody>
</table>
Faust Primitives
Add new ones using Foreign Functions

\[\text{foreignexp} \]

- Reference to external C functions, variables and constants can be introduced using the foreign function mechanism.

- example:

  \[\text{asinh} = \text{ffunction}(\text{float asinhf (float), <math.h>, "});\]
## Faust Primitives

**Delays and Tables**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem</td>
<td>$S^1 \rightarrow S^1$</td>
<td>1-sample delay: $y(t + 1) = x(t)$, $y(0) = 0$</td>
</tr>
<tr>
<td>prefix</td>
<td>$S^2 \rightarrow S^1$</td>
<td>1-sample delay: $y(t + 1) = x_2(t)$, $y(0) = x_1(0)$</td>
</tr>
<tr>
<td>@</td>
<td>$S^2 \rightarrow S^1$</td>
<td>fixed delay: $y(t + x_2(t)) = x_1(t)$, $y(t &lt; x_2(t)) = 0$</td>
</tr>
<tr>
<td>rtable</td>
<td>$S^3 \rightarrow S^1$</td>
<td>read-only table: $y(t) = T[r(t)]$</td>
</tr>
<tr>
<td>rtable</td>
<td>$S^5 \rightarrow S^1$</td>
<td>read-write table: $T[w(t)] = c(t)$; $y(t) = T[r(t)]$</td>
</tr>
<tr>
<td>select2</td>
<td>$S^3 \rightarrow S^1$</td>
<td>select between 2 signals: $T[] = {x_0(t), x_1(t)}$; $y(t) = T[s(t)]$</td>
</tr>
<tr>
<td>select3</td>
<td>$S^4 \rightarrow S^1$</td>
<td>select between 3 signals: $T[] = {x_0(t), x_1(t), x_2(t)}$; $y(t) = T$</td>
</tr>
</tbody>
</table>
# Faust Primitives

## User Interface Primitives

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>button(<em>str</em>)</td>
<td>button(&quot;play&quot;)</td>
</tr>
<tr>
<td>checkbox(<em>str</em>)</td>
<td>checkbox(&quot;mute&quot;)</td>
</tr>
<tr>
<td>vslider(<em>str</em>, <em>cur</em>, <em>min</em>, <em>max</em>, <em>inc</em>)</td>
<td>vslider(&quot;vol&quot;, 50, 0, 100, 1)</td>
</tr>
<tr>
<td>hslider(<em>str</em>, <em>cur</em>, <em>min</em>, <em>max</em>, <em>inc</em>)</td>
<td>hslider(&quot;vol&quot;, 0.5, 0, 1, 0.01)</td>
</tr>
<tr>
<td>nentry(<em>str</em>, <em>cur</em>, <em>min</em>, <em>max</em>, <em>inc</em>)</td>
<td>nentry(&quot;freq&quot;, 440, 0, 8000, 1)</td>
</tr>
<tr>
<td>vgroup(<em>str</em>, <em>block-diagram</em>)</td>
<td>vgroup(&quot;reverb&quot;, ...)</td>
</tr>
<tr>
<td>hgroup(<em>str</em>, <em>block-diagram</em>)</td>
<td>hgroup(&quot;mixer&quot;, ...)</td>
</tr>
<tr>
<td>tgroup(<em>str</em>, <em>block-diagram</em>)</td>
<td>tgroup(&quot;parametric&quot;, ...)</td>
</tr>
<tr>
<td>vbargraph(<em>str</em>, <em>min</em>, <em>max</em>)</td>
<td>vbargraph(&quot;input&quot;, 0, 100)</td>
</tr>
<tr>
<td>hbargraph(<em>str</em>, <em>min</em>, <em>max</em>)</td>
<td>hbargraph(&quot;signal&quot;, 0, 1.0)</td>
</tr>
</tbody>
</table>
4-Architectures
Faust Architecture System
Motivations: Easy deployment (one Faust code, multiple targets)

Diagram:
- FAUST Compiler
  - VST
  - Max/MSP
  - Jack
  - Alsa
  - Ladspa
  - CoreAudio
  - libsndfile
  - PortAudio
  - ActionScript
  - Pure Data
  - SuperCollider
  - Pure
  - Matlab
  - CSound
  - Q
  - etc...

FAUST Specification
Faust Architecture System

Principle: separation of concerns between the audio computation and its usage

To provide easy deployment, the DSP code generated by compiling a Faust program should be pure audio computation, abstracted from any audio drivers or GUI toolkit.
The role of the architecture file is to provide the missing information: the audio drivers and the user interface. The new modular architecture file combines an Audio driver module and one or more User Interface modules.
The Faust compiler wraps the DSP code into the selected architecture file. For examples

```
faust -a jack-gtk.cpp noise.dsp
```

will wrap the DSP code of a noise generator into the architecture of jack-gtk standalone application.
Faust Architecture System

Examples of supported architectures

- Audio plugins:
  - AudioUnit
  - LADSPA
  - DSSI
  - LV2
  - Max/MSP
  - VST
  - PD
  - Csound
  - Supercollider
  - Pure
  - Chuck
  - Octave
  - Flash

- Audio drivers:
  - Jack
  - Alsa
  - CoreAudio
  - Web Audio API

- Graphic User Interfaces:
  - QT
  - GTK
  - Android
  - iOS
  - HTML5/SVG

- Other User Interfaces:
  - OSC
  - HTTPD
5-Compiler/Code Generation
6-Performances
Performance of the generated code
How the C++ code generated by FAUST compares with hand written C++ code

**STK vs FAUST (CPU load)**

<table>
<thead>
<tr>
<th>File name</th>
<th>STK</th>
<th>FAUST</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>blowBottle.dsp</td>
<td>3,23</td>
<td>2,49</td>
<td>-22%</td>
</tr>
<tr>
<td>blowHole.dsp</td>
<td>2,70</td>
<td>1,75</td>
<td>-35%</td>
</tr>
<tr>
<td>bowed.dsp</td>
<td>2,78</td>
<td>2,28</td>
<td>-17%</td>
</tr>
<tr>
<td>brass.dsp</td>
<td>10,15</td>
<td>2,01</td>
<td>-80%</td>
</tr>
<tr>
<td>clarinet.dsp</td>
<td>2,26</td>
<td>1,19</td>
<td>-47%</td>
</tr>
<tr>
<td>flutestk.dsp</td>
<td>2,16</td>
<td>1,13</td>
<td>-47%</td>
</tr>
<tr>
<td>saxophony.dsp</td>
<td>2,38</td>
<td>1,47</td>
<td>-38%</td>
</tr>
<tr>
<td>sitar.dsp</td>
<td>1,59</td>
<td>1,11</td>
<td>-30%</td>
</tr>
<tr>
<td>tibetanBowl.dsp</td>
<td>5,74</td>
<td>2,87</td>
<td>-50%</td>
</tr>
</tbody>
</table>

Overall improvement of about 41 % in favor of FAUST.
Performance of the generated code
How the C++ code generated by FAUST compares with hand written C++ code

STK vs FAUST (CPU load)

<table>
<thead>
<tr>
<th>File name</th>
<th>STK</th>
<th>FAUST</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>blowBottle.dsp</td>
<td>3,23</td>
<td>2,49</td>
<td>-22%</td>
</tr>
<tr>
<td>blowHole.dsp</td>
<td>2,70</td>
<td>1,75</td>
<td>-35%</td>
</tr>
<tr>
<td>bowed.dsp</td>
<td>2,78</td>
<td>2,28</td>
<td>-17%</td>
</tr>
<tr>
<td>brass.dsp</td>
<td>10,15</td>
<td>2,01</td>
<td>-80%</td>
</tr>
<tr>
<td>clarinet.dsp</td>
<td>2,26</td>
<td>1,19</td>
<td>-47%</td>
</tr>
<tr>
<td>flutestk.dsp</td>
<td>2,16</td>
<td>1,13</td>
<td>-47%</td>
</tr>
<tr>
<td>saxophony.dsp</td>
<td>2,38</td>
<td>1,47</td>
<td>-38%</td>
</tr>
<tr>
<td>sitar.dsp</td>
<td>1,59</td>
<td>1,11</td>
<td>-30%</td>
</tr>
<tr>
<td>tibetanBowl.dsp</td>
<td>5,74</td>
<td>2,87</td>
<td>-50%</td>
</tr>
</tbody>
</table>

Overall improvement of about 41 % in favor of FAUST.
Performance of the generated code
What improvements to expect from parallelized code?

Sonik Cube
Audio-visual installation involving a cube of light, reacting to sounds, immersed in an audio feedback room (Trafik/Orlarey 2006).
Performance of the generated code
What improvements to expect from parallelized code?

Sonik Cube

- 8 loudspeakers
- 6 microphones
- audio software, written in FAUST, controlling the audio feedbacks and the sound spatialization.
Performance of the generated code
What improvements to expect from parallelized code?

**Sonik Cube**

Compared performances of the various C++ code generation strategies according to the number of cores:

![Graph showing performance improvement with increasing number of cores]
7-Documentation Preservation
Automatic Mathematical Documentation
Motivations et Principles

- Binary and source code preservation of programs is not enough: quick obsolescence of languages, systems and hardware.
- We need to preserve the mathematical meaning of these programs independently of any programming language.
- The solution is to generate automatically the mathematical description of any Faust program.
Binary and source code preservation of programs is not enough: quick obsolescence of languages, systems and hardware.

We need to preserve the mathematical meaning of these programs independently of any programming language.

The solution is to generate automatically the mathematical description of any Faust program.
Binary and source code preservation of programs is not enough: quick obsolescence of languages, systems and hardware.

We need to preserve the mathematical meaning of these programs independently of any programming language.

The solution is to generate automatically the mathematical description of any Faust program.
Binary and source code preservation of programs is not enough: quick obsolescence of languages, systems and hardware.

We need to preserve the mathematical meaning of these programs independently of any programming language.

The solution is to generate automatically the mathematical description of any Faust program.
Automatic Mathematical Documentation

Tools provided

- The easiest way to generate the complete mathematical documentation is to call the \texttt{faust2mathdoc} script on a \texttt{Faust} file.
- This script relies on a new option of the \texttt{Faust} compile: \texttt{-mdoc}
- \texttt{faust2mathdoc noise.dsp}
The easiest way to generate the complete mathematical documentation is to call the `faust2mathdoc` script on a `Faust` file.

This script relies on a new option of the `Faust` compile:

```
-mdoc
```

```
faust2mathdoc noise.dsp
```
The easiest way to generate the complete mathematical documentation is to call the `faust2mathdoc` script on a `Faust` file.

This script relies on a new option of the `Faust` compile: `-mdoc`
The easiest way to generate the complete mathematical documentation is to call the `faust2mathdoc` script on a Faust file.

This script relies on a new option of the Faust compile: `-mdoc`

`faust2mathdoc noise.dsp`
Automatic Mathematical Documentation
Files generated by Faust2mathdoc noise.dsp

▼ noise-mdoc/
  ▼ cpp/
    ◁ noise.cpp
  ▼ pdf/
    ◁ noise.pdf
  ▼ src/
    ◁ math.lib
    ◁ music.lib
    ◁ noise.dsp
  ▼ svg/
    ◁ process.pdf
    ◁ process.svg
  ▼ tex/
    ◁ noise.pdf
    ◁ noise.tex
8-Resources
Resources

FAUST Distribution on Sourceforge

http://sourceforge.net/projects/faudiostream/

- git clone
git://faudiostream.git.sourceforge.net/gitroot/faudiostream/faudiostream faust
- cd faust; make; sudo make install
Resources

FAUST Distribution on Sourceforge

- [http://sourceforge.net/projects/faudiostream/](http://sourceforge.net/projects/faudiostream/)
- `git clone`
  - `git://faudiostream.git.sourceforge.net/gitroot/faudiostream/faudiostream faust`
- `cd faust; make; sudo make install`
Resources

FAUST Distribution on Sourceforge

- [http://sourceforge.net/projects/faudiostream/](http://sourceforge.net/projects/faudiostream/)
- `git clone git://faudiostream.git.sourceforge.net/gitroot/faudiostream/faudiostream faust`
- `cd faust; make; sudo make install`
Resources
FAUST Distribution on Sourceforge

- [http://sourceforge.net/projects/faudiostream/](http://sourceforge.net/projects/faudiostream/)
- `git clone`
  - `git://faudiostream.git.sourceforge.net/gitroot/faudiostream/faudiostream faust`
- `cd faust; make; sudo make install`
Resources

FaustWorks IDE on Sourceforge

- http://sourceforge.net/projects/faudiostream/files/FaustWorks-0.3.2.tgz/download
- git clone
git://faudiostream.git.sourceforge.net/gitroot/faudiostream/FaustWorks
- cd FaustWorks; qmake; make
Resources

FaustWorks IDE on Sourceforge

- http://sourceforge.net/projects/faudiostream/files/FaustWorks-0.3.2.tgz/download
- git clone
git://faudiostream.git.sourceforge.net/gitroot/faudiostream/FaustWorks
- cd FaustWorks; qmake; make
Resources

FaustWorks IDE on Sourceforge

- [http://sourceforge.net/projects/faudiostream/files/](http://sourceforge.net/projects/faudiostream/files/) FaustWorks-0.3.2.tgz/download
- `git clone`
  - `git://faudiostream.git.sourceforge.net/gitroot/faudiostream/FaustWorks`
- `cd FaustWorks; qmake; make`
Resources

FaustWorks IDE on Sourceforge

- [http://sourceforge.net/projects/faudiostream/files/FaustWorks-0.3.2.tgz/download](http://sourceforge.net/projects/faudiostream/files/FaustWorks-0.3.2.tgz/download)
- `git clone`
  - `git://faudiostream.git.sourceforge.net/gitroot/faudiostream/FaustWorks`
- `cd FaustWorks; qmake; make`
Resources

Using FAUST Online Compiler

- [http://faust.grame.fr](http://faust.grame.fr)
- No installation required
- Compile to C++ as well as binary (Linux, MacOSX and Windows)
Resources
Using FAUST Online Compiler

- [http://faust.grame.fr](http://faust.grame.fr)
- No installation required
- Compile to C++ as well as binary (Linux, MacOSX and Windows)
Resources

Using FAUST Online Compiler

- [http://faust.grame.fr](http://faust.grame.fr)
- No installation required
- Compile to C++ as well as binary (Linux, MacOSX and Windows)
Resources
Using FAUST Online Compiler

- [http://faust.grame.fr](http://faust.grame.fr)
- No installation required
- Compile to C++ as well as binary (Linux, MacOSX and Windows)
Resources

FAUST Quick Reference

Figure: Faust Quick Reference, Grame
Resources
Some research papers

9-Acknowledgments
Acknowledgments

OS Community
Fons Adriaensen, Thomas Charbonnel, Olivier Guillerminet, Pierre Guillot, Albert Gräf, Stefan Kersten, Victor Lazzarini, Kjetil Matheussen, Yan Michalevsky, Romain Michon, Rémy Muller, Reza Payami, Stephen Sinclair, Travis Skare, Julius Smith, Mike Solomon

Sponsors
French Ministry of Culture, Rhône-Alpes Region, City of Lyon, National Research Agency

Partners from the ASTREE project (ANR 2008 CORD 003 02)
Jérôme Barthélemy (IRCAM), Karim Barkati (IRCAM), Alain Bonardi (IRCAM), Raffaele Ciavarella (IRCAM), Pierre Jouvelot (Mines/ParisTech), Laurent Pottier (U. Saint-Etienne)

Former Students
Tiziano Bole, Damien Cramet, Sarah Denoux, Étienne Gaudrin, Matthieu Leberre, Mathieu Leroi, Nicolas Scaringella