



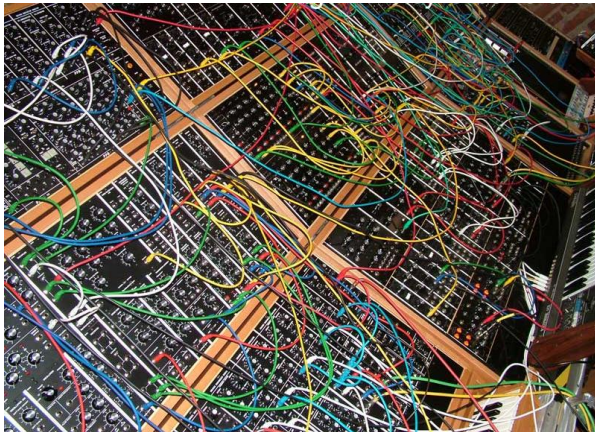
FAUST Tutorial for Functional Programmers

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

ICFP 2017 / FARM 2017

What is Faust ?

What is Faust?



A programming language (DSL) to build electronic music instruments

Some Music Programming Languages

- 4CED
- Adagio
- AML
- AMPLE
- Antescofo
- Arctic
- Autoklang
- Bang
- Canon
- CHANT
- Chuck
- CLCE
- CMIX
- Cmusic
- CMUSIC
- Common Lisp Music
- Common Music
- Common Music Notation
- Csound
- CyberBand
- DARMS
- DCOMP
- DMIX
- Elody
- EsAC
- Euterpea
- Extempore
- Faust
- Flavors Band
- Fluxus
- FOIL
- FORMES
- FORMULA
- Fugue
- Gibber
- GROOVE
- GUIDO
- HARP
- Haskore
- HMSL
- INV
- invokator
- KERN
- Keynote
- Kyma
- LOCO
- LPC
- Mars
- MASC
- Max
- MidiLisp
- MidiLogo
- MODE
- MOM
- Moxc
- MSX
- MUS10
- MUS8
- MUSCOMP
- MuseData
- MusES
- MUSIC 10
- MUSIC 11
- MUSIC 360
- MUSIC 4B
- MUSIC 4BF
- MUSIC 4F
- MUSIC 6
- MCL
- MUSIC III/IV/V
- MusicLogo
- Music1000
- MUSIC7
- Musictex
- MUSIGOL
- MusicXML
- Musitex
- NIFF
- NOTELIST
- Nyquist
- OPAL
- OpenMusic
- Organum1
- Outperform
- Overtone
- PE
- Patchwork
- PILE
- Pla
- PLACOMP
- PLAY1
- PLAY2
- PMX
- POCO
- POD6
- POD7
- PROD
- Puredata
- PWGL
- Ravel
- SALIERI
- SCORE
- ScoreFile
- SCRIPT
- SIREN
- SMDL
- SMOKE
- SSP
- SSSP
- ST
- Supercollider
- Symbolic Composer
- Tidal

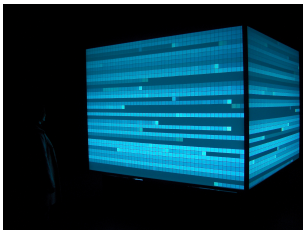
Brief Overview to Faust

- Faust offers end-users a high-level alternative to C to develop audio applications for a large variety of platforms.
- The role of the Faust compiler is to synthesize the most efficient implementations for the target language (C, C++, LLVM, Javascript, etc.).
- Faust is used on stage for concerts and artistic productions, for education and research, for open sources projects and commercial applications :

What Is Faust Used For ?

Artistic Applications

Sonik Cube (Trafik/Orlarey), Smartfaust (Gracia), etc.



Open-Source projects

Guitarix: Hermann Meyer



WebAudio Applications

YC20 Emulator

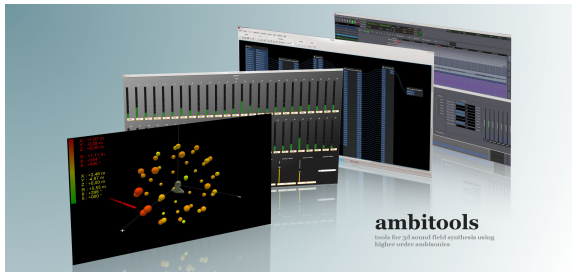
Thanks to the HTML5/WebAudio API and Asm.js it is now possible to run synthesizers and audio effects from a simple web page !



Sound Spatialization

Ambitools: Pierre Lecomte, CNAM

Ambitools (Faust Award 2016), 3-D sound spatialization using Ambisonic techniques.



Medical Applications

Brain Stethoscope: Chris Chafe, CCRMA-Stanford

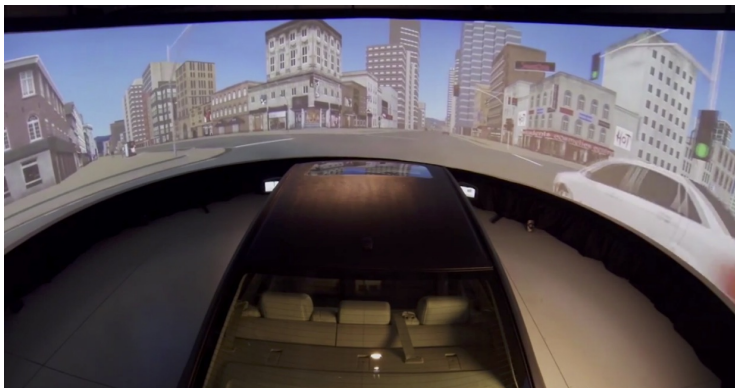
Brain stethoscope turns seizures into music in hopes of giving the listener an empathetic and intuitive understanding of the neural activity.



Simulation Applications

Stanford Car Simulator: Romain Michon, CCRMA-Stanford

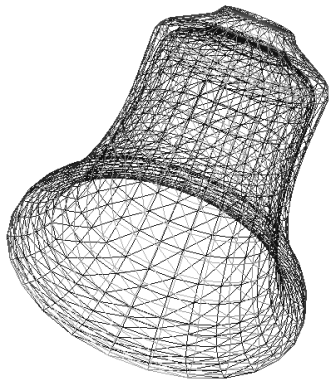
Stanford Car Simulator, simulation of the sound of a car engine in Faust.



Simulation Applications

Bell simulations: Romain Michon and Chris Chafe, CCRMA-Stanford

CAD description of a bell turned into a procedural audio simulation in Faust



Hearing Aids Applications

Soniccloud, USA

Soniccloud: every cell phone call can be perfectly calibrated for an individual's unique Hearing Fingerprint – across 10 sonic dimensions.

The team at SonicCloud has had an outstanding experience working with Faust. Specifically, we have been able to optimize code to run our DSP algorithms in real-time without having to hand-optimize C/C++ code or write assembler. (Soniccloud)



Musical Instruments

Moforte, USA

Moforte (USA) designs musical instruments for iPad and iOS using Faust



Exercise 1: Djembe

Exercise 1: Djembe

Faust Online Compiler:

- <https://faust.grame.fr/onlinecompiler>

Faust Libraries Documentation:

- <http://faust.grame.fr/libraries.html>

Faust Code:

```
import("stdfaust.lib");  
  
process = button("play")  
          : pm.djembe(330,0.8,0.5,1);
```

Faust Signals and Time Model

Faust Signals and Time Model

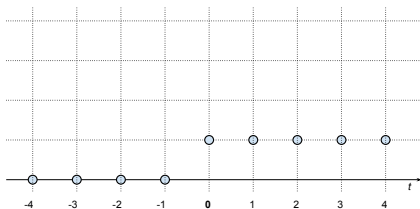
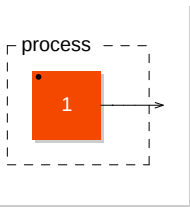
Faust programs operate on periodically sampled *signals*.

- A *signal* $s \in \mathbb{S}$ is a *time to sample* function.
- Two kinds of signals: $\mathbb{S} = \mathbb{S}_{\mathbb{Z}} \cup \mathbb{S}_{\mathbb{R}}$
 - ▶ Integer signals: $\mathbb{S}_{\mathbb{Z}} = \mathbb{Z} \rightarrow \mathbb{Z}$
 - ▶ Floating-point signals: $\mathbb{S}_{\mathbb{R}} = \mathbb{Z} \rightarrow \mathbb{R}$
- The value of a Faust signal is always 0 before time 0 :
 - ▶ $\forall s \in \mathbb{S}, s(t < 0) = 0$
- A Faust program denotes a *signal processor* $p \in \mathbb{P}$, a (continuous) function that maps a group of n input *signals* to a group of m output *signals* :
 - ▶ $\mathbb{P} = \mathbb{S}^n \rightarrow \mathbb{S}^m$

Faust Signals and Time Model

Example, a "constant" signal

`process = 1;`



$$y(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}$$

Primitive Signal Processors

Faust Primitives

Arithmetic operations

Syntax	Type	Description
+	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	addition: $y(t) = x_1(t) + x_2(t)$
-	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	subtraction: $y(t) = x_1(t) - x_2(t)$
*	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	multiplication: $y(t) = x_1(t) * x_2(t)$
\wedge	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	power: $y(t) = x_1(t)^{x_2(t)}$
/	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	division: $y(t) = x_1(t)/x_2(t)$
%	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	modulo: $y(t) = x_1(t) \% x_2(t)$
int	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	cast into an int signal: $y(t) = (int)x(t)$
float	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	cast into an float signal: $y(t) = (float)x(t)$

Faust Primitives

Bitwise operations

Syntax	Type	Description
<code>&</code>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical AND: $y(t) = x_1(t) \& x_2(t)$
<code> </code>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical OR: $y(t) = x_1(t) x_2(t)$
<code>xor</code>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	logical XOR: $y(t) = x_1(t) \wedge x_2(t)$
<code><<</code>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	arith. shift left: $y(t) = x_1(t) << x_2(t)$
<code>>></code>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	arith. shift right: $y(t) = x_1(t) >> x_2(t)$

Faust Primitives

Comparison operations

Syntax	Type	Description
<	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	less than: $y(t) = x_1(t) < x_2(t)$
<=	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	less or equal: $y(t) = x_1(t) \leq x_2(t)$
>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	greater than: $y(t) = x_1(t) > x_2(t)$
>=	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	greater or equal: $y(t) = x_1(t) \geq x_2(t)$
==	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	equal: $y(t) = x_1(t) == x_2(t)$
!=	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	different: $y(t) = x_1(t) \neq x_2(t)$

Faust Primitives

Trigonometric functions

Syntax	Type	Description
<code>acos</code>	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	arc cosine: $y(t) = \text{acosf}(x(t))$
<code>asin</code>	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	arc sine: $y(t) = \text{asinf}(x(t))$
<code>atan</code>	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	arc tangent: $y(t) = \text{atanf}(x(t))$
<code>atan2</code>	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	arc tangent of 2 signals: $y(t) = \text{atan2f}(x_1(t), x_2(t))$
<code>cos</code>	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	cosine: $y(t) = \text{cosf}(x(t))$
<code>sin</code>	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	sine: $y(t) = \text{sinf}(x(t))$
<code>tan</code>	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	tangent: $y(t) = \text{tanf}(x(t))$

Faust Primitives

Other Math operations

Syntax	Type	Description
exp	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	base-e exponential: $y(t) = \expf(x(t))$
log	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	base-e logarithm: $y(t) = \logf(x(t))$
log10	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	base-10 logarithm: $y(t) = \log10f(x(t))$
pow	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	power: $y(t) = \text{powf}(x_1(t), x_2(t))$
sqrt	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	square root: $y(t) = \text{sqrtf}(x(t))$
abs	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	absolute value (int): $y(t) = \text{abs}(x(t))$ absolute value (float): $y(t) = \text{fabsf}(x(t))$
min	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	minimum: $y(t) = \min(x_1(t), x_2(t))$
max	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	maximum: $y(t) = \max(x_1(t), x_2(t))$
fmod	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	float modulo: $y(t) = \text{fmodf}(x_1(t), x_2(t))$
remainder	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	float remainder: $y(t) = \text{remainderf}(x_1(t), x_2(t))$
floor	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	largest int \leq : $y(t) = \text{floorf}(x(t))$
ceil	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	smallest int \geq : $y(t) = \text{ceilf}(x(t))$
rint	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	closest int: $y(t) = \text{rintf}(x(t))$

Faust Primitives

Delays and Tables

Syntax	Type	Description
mem	$\mathbb{S}^1 \rightarrow \mathbb{S}^1$	1-sample delay: $y(t+1) = x(t), y(0) = 0$
@	$\mathbb{S}^2 \rightarrow \mathbb{S}^1$	delay: $y(t+x_2(t)) = x_1(t), y(t < x_2(t)) = 0$
rdtable	$\mathbb{S}^3 \rightarrow \mathbb{S}^1$	read-only table: $y(t) = T[r(t)]$
rwtable	$\mathbb{S}^5 \rightarrow \mathbb{S}^1$	read-write table: $T[w(t)] = c(t); y(t) = T[r(t)]$
select2	$\mathbb{S}^3 \rightarrow \mathbb{S}^1$	select between 2 signals: $T[] = \{x_0(t), x_1(t)\}; y(t) = T[s(t)]$
select3	$\mathbb{S}^4 \rightarrow \mathbb{S}^1$	select between 3 signals: $T[] = \{x_0(t), x_1(t), x_2(t)\}; y(t) = T$

Faust Primitives

User Interface Primitives

Syntax	Example
<code>button(<i>str</i>)</code>	<code>button("play")</code>
<code>checkbox(<i>str</i>)</code>	<code>checkbox("mute")</code>
<code>vslider(<i>str</i>,<i>cur</i>,<i>min</i>,<i>max</i>,<i>inc</i>)</code>	<code>vslider("vol",50,0,100,1)</code>
<code>hslider(<i>str</i>,<i>cur</i>,<i>min</i>,<i>max</i>,<i>inc</i>)</code>	<code>hslider("vol",0.5,0,1,0.01)</code>
<code>nentry(<i>str</i>,<i>cur</i>,<i>min</i>,<i>max</i>,<i>inc</i>)</code>	<code>nentry("freq",440,0,8000,1)</code>
<code>vgroup(<i>str</i>,<i>block-diagram</i>)</code>	<code>vgroup("reverb", ...)</code>
<code>hgroup(<i>str</i>,<i>block-diagram</i>)</code>	<code>hgroup("mixer", ...)</code>
<code>tgroup(<i>str</i>,<i>block-diagram</i>)</code>	<code>vgroup("parametric", ...)</code>
<code>vbargraph(<i>str</i>,<i>min</i>,<i>max</i>)</code>	<code>vbargraph("input",0,100)</code>
<code>hbargraph(<i>str</i>,<i>min</i>,<i>max</i>)</code>	<code>hbargraph("signal",0,1.0)</code>

Exercise 2: Adding rhythm and sliders to the Djembe

Exercise 2: Adding rhythm to the Djembe

Faust Code:

```
import("stdfaust.lib");

process = button("play"), // try checkbox("play")
      ba.pulsen(100, 4800) : *
      : pm.djembe(330,0.8,0.5,1);
```

Exercise 3: Adding sliders to the Djembe

Faust Code:

```
import("stdfaust.lib");

process = checkbox("play"),
          ba.pulsen(100, 4800) : *
          : pm.djembe(
              hslider("freq", 300, 100, 1000, 1),
              hslider("position", 0.8, 0, 1, 0.1),
              hslider("sharpness", 0.5, 0, 1, 0.1),
              hslider("gain", 0.5, 0, 1, 0.1)
          );
```

Exercise 4: Adding an echo to the Djembe

Faust Code:

```
import("stdfaust.lib");

echo = +~(@ (22100) : * (hslider("fb", 0, 0, 1, 0.01)));

process = checkbox("play"),
          ba.pulsen(100, 4800) : *
          : pm.djembe(
              hslider("freq", 300, 100, 1000, 1),
              hslider("position", 0.8, 0, 1, 0.1),
              hslider("sharpness", 0.5, 0, 1, 0.1),
              hslider("gain", 0.5, 0, 1, 0.1)
          )
          : echo;
```

Programming by composition

Programming by Composition

Block-Diagram Algebra

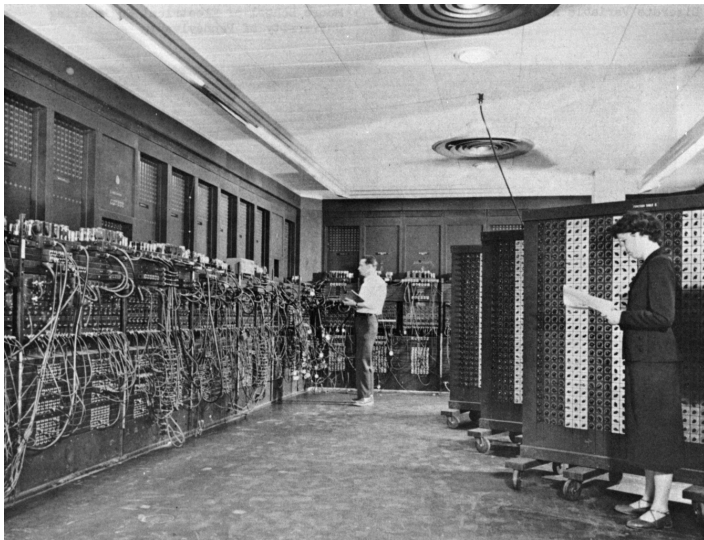
Programming by patching is familiar to musicians :



Programming by Composition

Block-Diagram Algebra

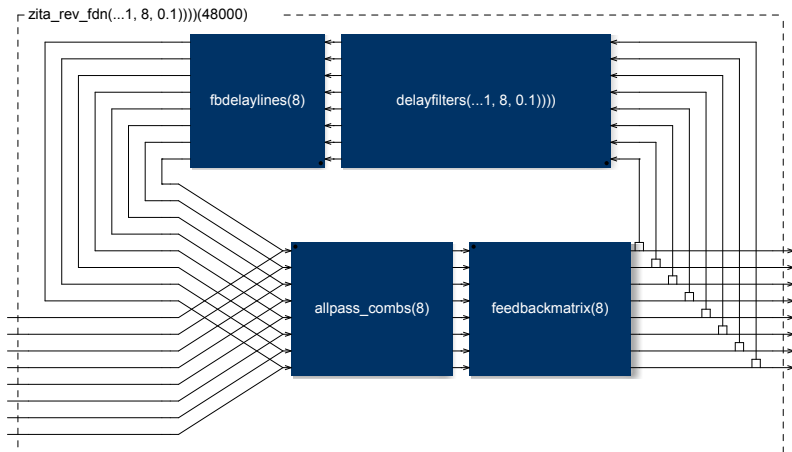
Programming by patching, the ENIAC computer :



Programming by Composition

Block-Diagram Algebra

Faust allows structured block-diagrams, here part of the zita reverb.



Programming by Composition

Composition Operations

- $(A < : B)$ split composition (associative, priority 1)
- $(A : > B)$ merge composition (associative, priority 1)
- $(A : B)$ sequential composition (associative, priority 2)
- (A, B) parallel composition (associative, priority 3)
- $(A \sim B)$ recursive composition (priority 4)

Programming by Composition

Same Expression in Lambda-Calculus, FP and Faust

Lambda-Calculus

$\lambda x. \lambda y. (x+y, x*y) \ 2 \ 3$

FP/FL (John Backus)

$[+,*] : \langle 2,3 \rangle$

Faust

$2,3 \prec : +,*$

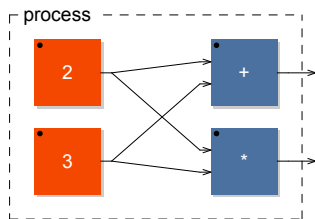
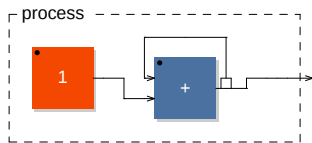


Figure: block-diagram of
 $2,3 \prec : +,*$

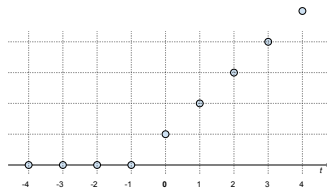
Programming by Composition

A Very Simple Example

```
process = 1 : +~_;
```



$$y(t) = \begin{cases} 0 & t < 0 \\ 1 + y(t-1) & t \geq 0 \end{cases}$$



Programming by Composition

Parallel Composition (associative, priority 3)

The *parallel composition* (A, B) is probably the simplest one. It places the two block-diagrams one on top of the other, without connections.

$$(A, B): (\mathbb{S}^n \rightarrow \mathbb{S}^m) \rightarrow (\mathbb{S}^{n'} \rightarrow \mathbb{S}^{m'}) \rightarrow (\mathbb{S}^{n+n'} \rightarrow \mathbb{S}^{m+m'})$$

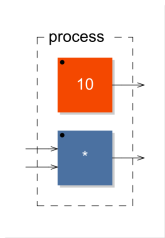


Figure: Example of parallel composition $(10, *)$

Programming by Composition

Sequential Composition (associative, priority 2)

The *sequential composition* $(A:B)$ connects the outputs of A to the corresponding inputs of B .

$$(A:B): (\mathbb{S}^n \rightarrow \mathbb{S}^m) \rightarrow (\mathbb{S}^m \rightarrow \mathbb{S}^p) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^p)$$

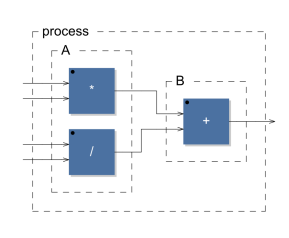


Figure: Example of sequential composition $((*, /):+)$

Programming by Composition

Split Composition (associative, priority 1)

The *split composition* ($A <: B$) operator is used to distribute the outputs of A to the inputs of B

$$(A <: B): (\mathbb{S}^n \rightarrow \mathbb{S}^m) \rightarrow (\mathbb{S}^{k.m} \rightarrow \mathbb{S}^p) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^p)$$

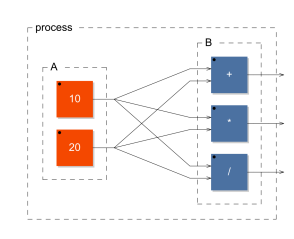


Figure: example of split composition $((10, 20) <: (+, *, /))$

Programming by Composition

Merge Composition (associative, priority 1)

The *merge composition* ($A :> B$) is used to connect several outputs of A to the same inputs of B . Signals connected to the same input are added.

$$(A :> B) : (\mathbb{S}^n \rightarrow \mathbb{S}^{k.m}) \rightarrow (\mathbb{S}^m \rightarrow \mathbb{S}^p) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^p)$$

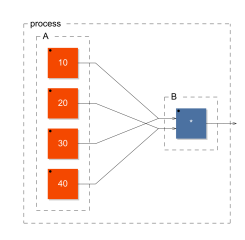


Figure: example of merge composition $((10, 20, 30, 40) :> *)$

Programming by Composition

Recursive Composition (priority 4)

The *recursive composition* ($A \sim B$) is used to create cycles in the block-diagram in order to express recursive computations.

$$(A \sim B): (\mathbb{S}^{n+n'} \rightarrow \mathbb{S}^{m+m'}) \rightarrow (\mathbb{S}^{m'} \rightarrow \mathbb{S}^{n'}) \rightarrow (\mathbb{S}^n \rightarrow \mathbb{S}^{m+m'})$$

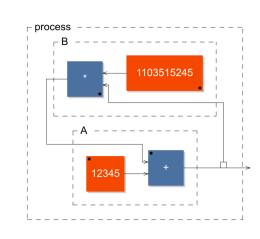
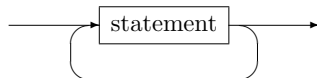


Figure: example of recursive composition $+(12345) \sim *(1103515245)$

Faust Program

Faust Program

program



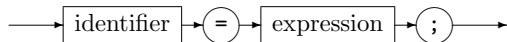
- A Faust program is essentially a list of *statements*. These statements can be :
 - ▶ metadata *declarations*,
 - ▶ file *imports*
 - ▶ *definitions*
- Example :

```
declare name      "noise";  
declare copyright "(c)GRAME_2006";  
import("music.lib");  
process = noise * vslider("volume", 0, 0, 1, 0.1);
```

Definitions

Simple Definitions

definition



- A *definition* associates an identifier with an expression it stands for.

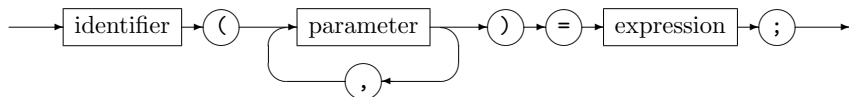
Example :

```
random = +(12345) ~ *(1103515245);
```

Definitions

Functions' definitions

definition



- Definitions with formal parameters correspond to functions' definitions.
Example :

```
linear2db(x) = 20*log10(x);
```

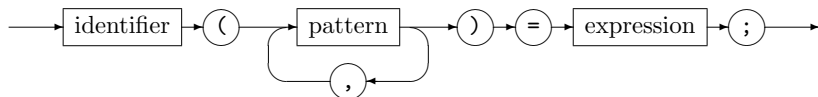
- Alternative notation using a *lambda-abstraction*:

```
linear2db = \ (x).(20*log10(x));
```

Definitions

Pattern Matching Definitions

definition



- Formal parameters can also be full expressions representing patterns.

Example :

```
duplicate(1,exp) = exp;  
duplicate(n,exp) = exp, duplicate(n-1,exp);
```

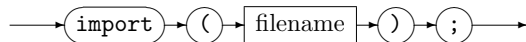
- Alternative notation :

```
duplicate = case {  
    (1,exp) => exp;  
    (n,exp) => duplicate(n-1,exp);  
};
```

Statement

Import file

fileimport

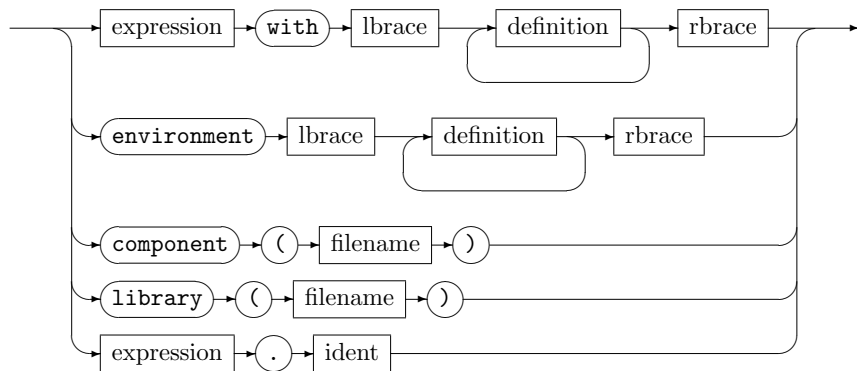


- allows to import definitions from other source files.
- for example `import("math.lib");` imports the definitions from `"math.lib"` file, a set of additional mathematical functions provided as foreign functions.

Expressions

Environments

envexp

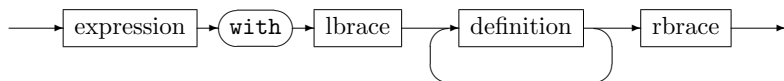


- Each Faust expression has an associated *lexical environment*

Environments

With Expression

with expression



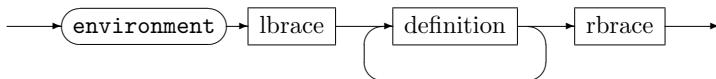
- With expression allows to specify a *local environment*, a private list of definitions that will be used to evaluate the left hand expression
- example pink noise filter :

```
pink = f : + ~ g with {  
    f(x) = 0.04957526213389*x  
          - 0.06305581334498*x@1  
          + 0.01483220320740*x@2;  
    g(x) = 1.80116083982126*x  
          - 0.80257737639225*x@1;  
};
```

Environments

Environment

environment



- an **environment** is used to group together related definitions :

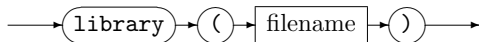
```
constant = environment {  
    pi = 3.14159;  
    e = 2,718 ;  
    ....  
};
```

- definitions of an environment can be easily accessed : `constant.pi`

Environments

Library

library

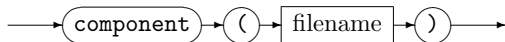


- allows to create an environment by reading the definitions from a file.
- example : `library("filter.lib")`
- definitions are accessed like this : `library("filter.lib").smooth`

Environments

Component

component



- allows to reuse a full Faust program as a simple expression.
- example :

```
component ("osc.dsp") <: component ("freeverb.dsp")
```

- equivalence between :

```
component ("freeverb.dsp")
```

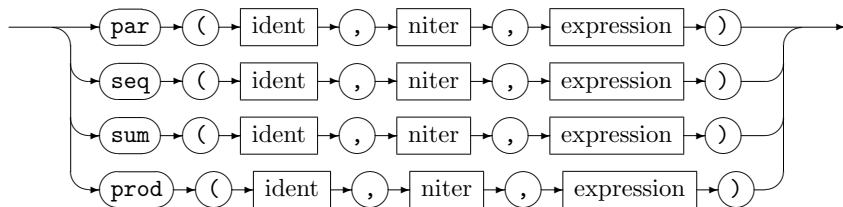
and

```
library ("freeverb.dsp").process
```

Expressions

Iterations

diagiteration



- Iterations are analog to `for(...)` loops
- provide a convenient way to automate some complex block-diagram constructions.

Expressions

Iterations

The following example shows the use of `seq` to create a 10-bands filter:

```
process = seq(i, 10,  
              vgroup("band_□i",  
                    bandfilter( 1000*(1+i) )  
              )  
);
```

Exercise 5: Djembe automation

Exercise 5: Djembe automation

Faust Code:

```
import("stdfaust.lib");

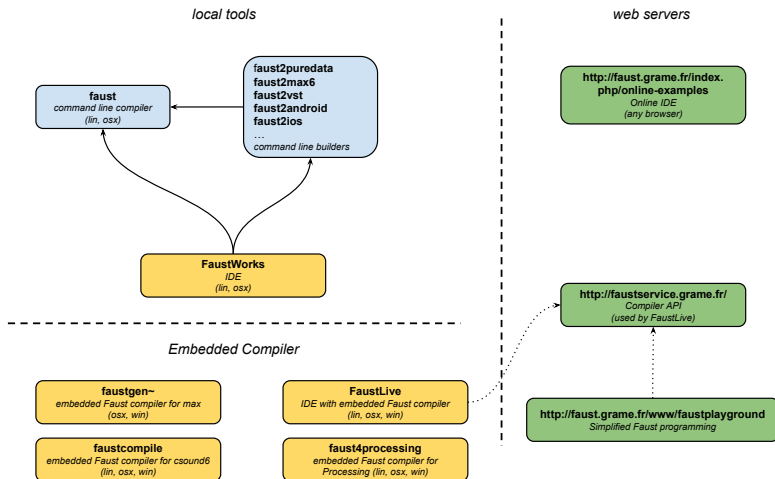
saw(f) = f/ma.SR : (+,1:fmod)~_ ;

process = checkbox("play"),
          ba.pulsen(100, 4800) : *
          : pm.djembe(
              hslider("freq", 300, 100, 1000, 1),
              saw(hslider("fpos",1,0.05,20,0.01)),
              saw(hslider("fsharp",1,0.05,20,0.01)),
              saw(hslider("fgain",1,0.05,20,0.01))
          )
          ;
```


Faust Ecosystem

Faust Ecosystem

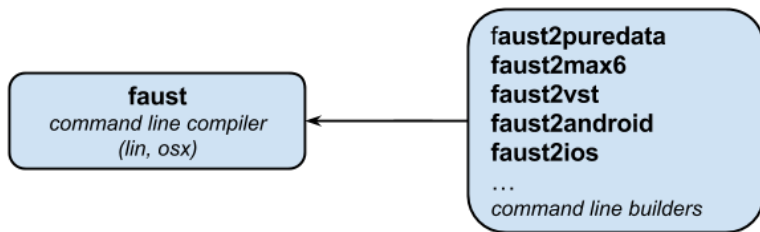
Overview



Faust Ecosystem

Command-line Tools

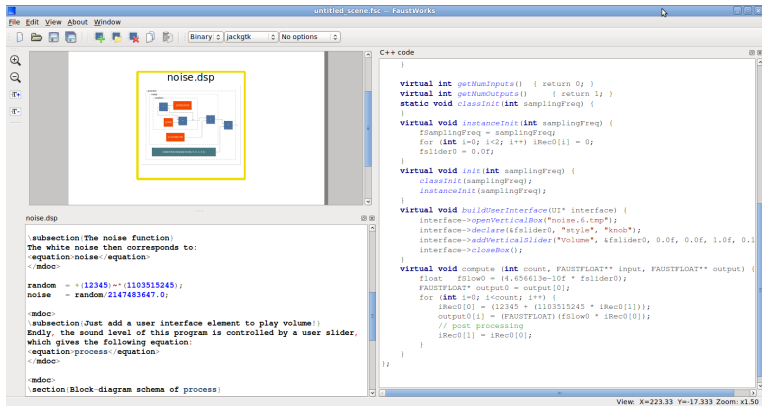
Simplify the compilation workflow : full automated process to build Android and iOS applications, VST plugins, Max/MSP externals, Csound opcodes, etc.



Faust Ecosystem

FaustWorks

FaustWorks can simplify Faust learning in particular by providing "realtime" code and diagram generation:



The screenshot displays the FaustWorks application interface. The top window, titled "untitled_scene.fsc - FaustWorks", shows a block diagram of a noise function. The diagram includes a "random" block, a "gain" block, and a "slider" block, all interconnected. Below the diagram, the Faust code for "noise.dsp" is visible. The code defines a noise function with a random signal source, a gain of 1/2147483647.0, and a user interface element for volume control. The bottom window, titled "C++ code", shows the generated C++ code for the same function. The code includes headers for FaustWorks, defines the number of inputs and outputs, and implements the initialization and computation logic. The computation logic involves a random signal source, a gain, and a user interface element for volume control.

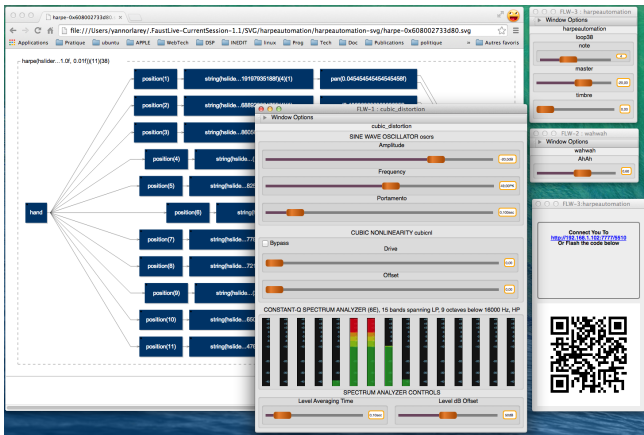
```
noise.dsp
<section>The noise function</section>
The white noise then corresponds to:
<equation>noise</equation>
<mdoc>
random
= +(12345)~*(1103515245);
noise
= random/2147483647.0;
</mdoc>
<section>Just add a user interface element to play volume!</section>
Endly, the sound level of this program is controlled by a user slider,
which gives the following equation:
<equation>process</equation>
<mdoc>
</mdoc>
<section>Block-diagram schema of process</section>
```

```
C++ code
virtual int getNumInputs() { return 0; }
virtual int getNumOutputs() { return 1; }
static void classInit(int samplingFreq) {
}
virtual void instanceInit(int samplingFreq) {
    fSamplingFreq = samplingFreq;
    for (int i=0; i<2; i++) iRec0[i] = 0;
    fslider0 = 0.0f;
}
virtual void init(int samplingFreq) {
    classInit(samplingFreq);
    instanceInit(samplingFreq);
}
virtual void buildUserInterface(UI* interface) {
    interface->openVerticalBox("noise.6.tmp");
    interface->declare(&fslider0, "style", "knob");
    interface->addVerticalSlider("Volume", &fslider0, 0.0f, 0.0f, 1.0f, 0.1);
    interface->closeBox();
}
virtual void compute (int count, FAUSTFLOAT** input, FAUSTFLOAT** output) {
    float fSlow0 = (4.656613e-10f * fslider0);
    FAUSTFLOAT* output0 = output[0];
    for (int i=0; i<count; i++) {
        iRec0[0] = (12345 + (1103515245 * iRec0[1]));
        output0[i] = (FAUSTFLOAT)(fSlow0 * iRec0[0]);
        // post processing
        iRec0[1] = iRec0[0];
    }
}
```

Faust Ecosystem

FaustLive

FaustLive speeds up the Edit/Compile/Run. It provides advanced cooperation features :



Faust Ecosystem

Faustgen

Faustgen speeds up the Edit/Compile/Run within the Max framework:

The image displays two windows from the Faustgen application running within the Max/MSP framework.

DSP code : fst_cub~

```
declare name "fst_cub~";
declare version "1.33";
//declare author "Julius O. Smith ";
declare license "STK-4.3";

import("math.lib");
import("music.lib");
import("effect.lib");
import("filter.lib");

//----- cubicnl(drive,offset) -----
// Cubic nonlinearity distortion
//
// USAGE: cubicnl(drive,offset), where
// drive = distortion amount, between 0 and 1
// offset = constant added before nonlinearity to give even harmonics
// Note: offset can introduce a nonzero mean - feed
// cubicnl output to dcblocker to remove this.
//
// REFERENCES:
// https://ccrma.stanford.edu/~jos/pasp/Cubic_Soft_Clipper.html
// https://ccrma.stanford.edu/~jos/pasp/Nonlinear_Distortion.html

// direct from effect Lib and combine with a filter. A dc remover will be in

sf1 = vslider("freq-low-cut",130,20,1000,1):smooth(0.99);
sf2 = vslider("freq-high-cut",5000,1000,10000,1):smooth(0.99);

drive = vslider("drive", 0, 0, 1, 0.01);
offset = vslider("offset", 0, 0, 1, 0.01);

process_ = cubicnl(drive,offset) * speakerba(sf1 sf2) * mvecho;
Insertion Point Line: 1
```

fg_fst_cub~

faustgen~: Error in sequential composition (A.B) The number of outputs (1) of A = _(1,...

fst_cub~
Cubic nonlinearity distortion

1 vibes-a1.aif select a audiofile

2 or drag & drop soundfile

prepend open 0.22 0.94 1555 hz 84.30000 hz

splay~ drive \$1 offset \$1 freq-low-cut \$1 freq-high-cut \$1

☐ mute dsp of faust object directly

mute \$1 read fst_cub.dsp

faustgen~ fst_cub~

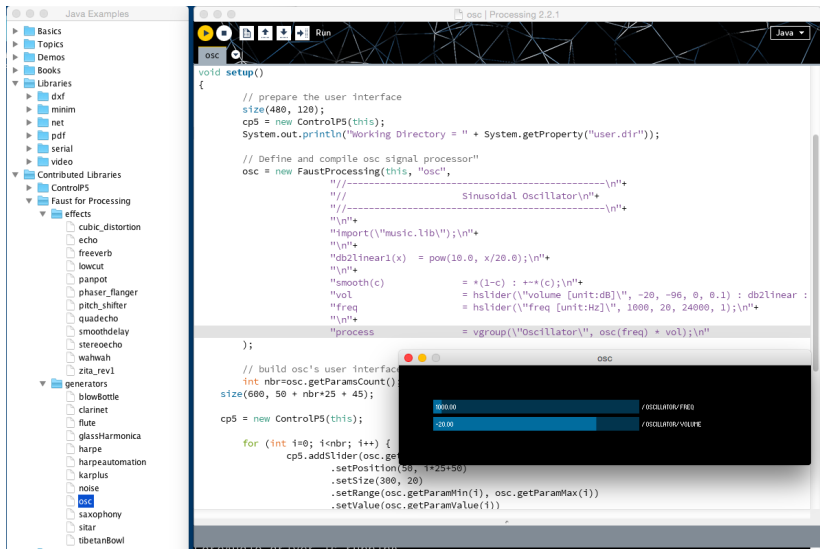
live gain~

-7.0 dB

Faust Ecosystem

Faust4processing

Faust4processing provides an embedded Faust compiler for Processing:



Faust Ecosystem

PMIX (Oliver Larkin)

PMIX speeds up the Edit/Compile/Run within VST host:

The screenshot displays the pMix software interface, which is used for editing and running Faust patches. The interface is divided into several sections:

- Left Panel:** Contains a visual representation of the Faust patch. It shows a **Midi Input** and **Audio Input** at the top. Below them is a **Kisana** module with parameters: **level** (red dot), **note** (two red dots), **note** (red dot), **note** (red dot), **master** (red dot), and **timbre** (red dot). Below the **Kisana** module is a **WahWah** module with a **wahwah** parameter (a slider). At the bottom is an **Audio Output** module. Dashed lines indicate the signal flow between these modules.
- Right Panel:** Contains a **Code Editor** with a menu bar (File, Edit, View, Help) and a tab labeled **Interpolation Space**. The code editor shows the following Faust code:

```
1 //-----
2 //   Kisana : 3-loops string instrument
3 //   (based on Karplus-Strong)
4 //
5 //-----
6
7 declare name    "Kisana";
8 declare author  "Yann Orlarey";
9
10 import("music.lib");
11
12
13
14 KEY = 60; // basic midi key
15 NCY = 15; // note cycle length
16 CCY = 15; // control cycle length
17 RPS = 360; // general tempo (beat per sec)
```

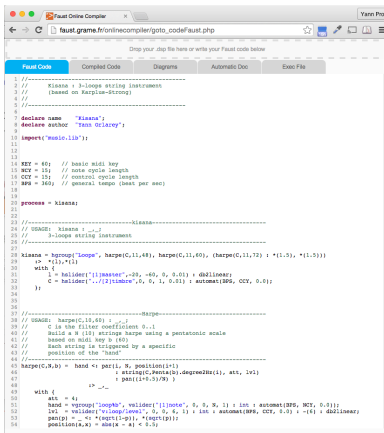
Below the code editor is a **Diagram** section showing a hierarchical view of the patch. It includes a **process** block containing a **group(Total)** block, which in turn contains three **harpeholder...** blocks. These blocks are connected to a network of arithmetic and signal processing blocks, including multipliers (labeled 1.0 and 1.4) and summing junctions.

Faust Ecosystem

Online compiler

The Online compiler can be used from a web browser to compile Faust programs for a variety of systems, including the Web.

<http://faust.grame.fr/onlinecompiler>



```
1 //-----
2 // Kiana = 3-voice string instrument
3 // (based on Karplus-Strong)
4 //
5 //-----
6
7 declare name "Kiana";
8 declare author "Evan Orlitzky";
9
10 import("music.lib");
11
12
13
14 KEY = 60; // basic midi key
15 MCV = 15; // note cycle length
16 CCR = 15; // control cycle length
17 BPM = 360; // general tempo (beat per sec)
18
19
20 process = kiana:
21
22
23 //-----Kiana-----
24 // USAGE: kiana ~,~
25 // 3-voice string instrument
26 //-----
27
28 kiana = hgroup("loop", harpe(C,11,68), harpe(C,11,60), harpe(C,11,72) : *(1.5), *(1.5))
29   >> "f1","f1"
30   with {
31     l = balder["lijaster",-20,-60,0,0.01] : d2linear;
32     C = balder["...l2]umbr",0,0,1,0.01] : automa(BPM,CCR,0.0);
33   };
34
35
36
37 //-----Harpe-----
38 // USAGE: harpe(C,10,60) ~,~
39 // C is the filter coefficient 0..1
40 // Build a N (10) strings harpe using a pentatonic scale
41 // based on midi key b (60)
42 // Each string is triggered by a specific
43 // position of the "hand"
44 //-----
45 harpe(C,N,b) = hand <- par1(L,N,position[1+1]
46   : atxling(C,pena[b].degree2Hz(i), atx, lvl)
47   : pan((1+0.5)/N))
48   >> ~,~
49   with {
50     atx = 4;
51     hand = vgroup("loopk", valider["lijnote",0,0,N,1] : lat : automa(BPM,CCR,0.0));
52     lvl = valider["looplevel",0,0,0,1] : lat : automa(BPM,CCR,0.0) : [-6] : d2linear;
53     pan(p) = ~,~ <- (log((1-p)),*(exp(19)))
54     position[i,x] = abs(x - a) < 0.5;
```

Faust Ecosystem

Faust playground

The Web as a gigantic Lego box to reuse and recompose audio applications. <http://faust.grame.fr/faustplayground>

The screenshot shows the Faust Playground web application in a browser window. The browser's address bar displays `faust.grame.fr/faustplayground/`. The application's top navigation bar includes buttons for LIBRARY, LOAD, EDIT, SAVE, EXPORT, and HELP, along with icons for fullscreen, checkmark, and trash. The main interface is divided into several sections:

- Top Left:** A text box labeled "Patch" with a description: "The application is called: Patch. Only alphabet letters and numbers are accepted. Spaces, apostrophes and accents are automatically replaced. The name cannot start with a number. It must be between 1 and 50 characters." Below this is a text input field containing "Patch".
- Top Center:** A "Choose export" section with a text input field containing `http://faustservice.grame.fr`, a "Refresh server" button, and two dropdown menus both set to "android". Below these is an "Export" button.
- Top Right:** A "Download" section featuring a QR code and a "Download" button.
- Bottom Left:** A "Change the name of the application" button.
- Bottom Center:** A "FEEDBACK" control panel with a slider and a "Download" button.
- Bottom Right:** A speaker icon indicating audio output.

The background of the application is a blue grid pattern. At the bottom left, there is a logo for "GRAME" and the text "Extension of the WebAudio Playground by Chris Wilson".

Thanks! Questions?