Using the FAUST DSP language and the libfaust JIT compiler with JUCE

Oliver Larkin
Contemporary Music Research Centre,
University of York / Oli Larkin Plug-ins
contact@olilarkin.co.uk

FAUST

Functional **AU**dio **ST**ream

FAUST

- A domain-specific functional programming language for audio signal processing
- Developed by Yann Orlarey, Stéphane Letz and Dominique Fober at Grame in Lyon, with many international contributors
- The FAUST compiler can generate C, C++, JAVA, JavaScript, ASM.js or LLVM IR from the FAUST language
- Comes with a large collection of scripts and “architecture files” to compile FAUST code to various binary formats (e.g. Max MSP object, VST plug-in)
- The compiler is GPL licensed and works on Linux, OS X, Windows
- Code generated by the compiler inherits the license of the input source code and can typically be used to create closed-source/commercial projects

Functional Programming

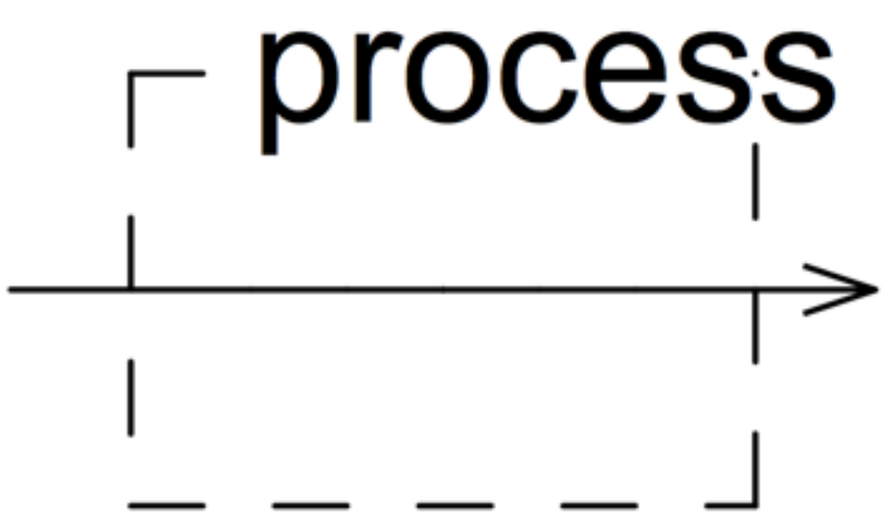
In computer science, functional programming is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions ***and avoids changing-state and mutable data****. It is a declarative programming paradigm, which means* ***programming is done with expressions****. In functional code, the output value of a function depends only on the arguments that are input to the function, so calling a function f twice with the same value for an argument x will produce the same result f(x) each time*

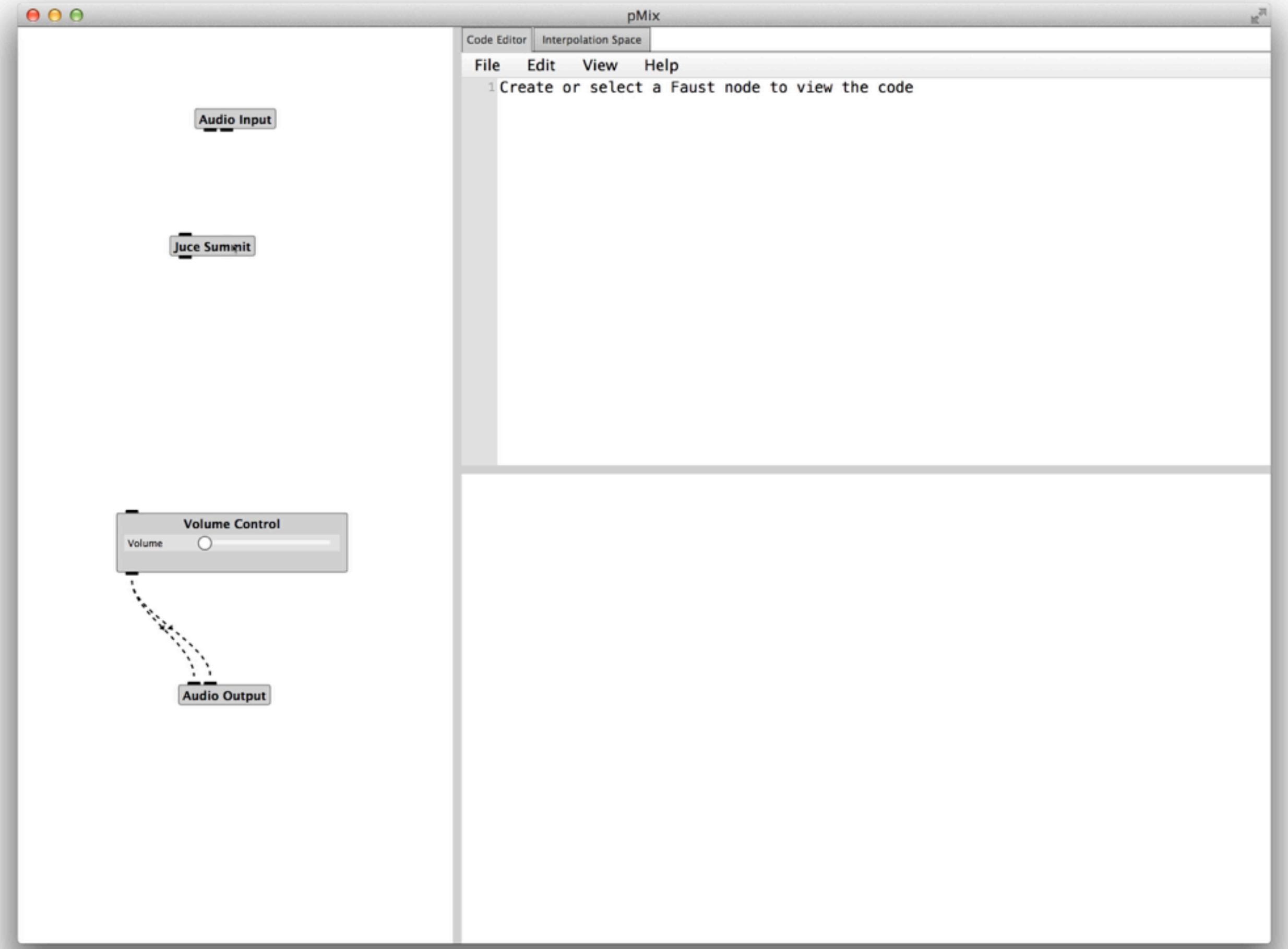
Source: https://en.wikipedia.org/wiki/Functional_programming

FAUST Programs

- A FAUST program describes a signal processor: a mathematical function that is applied to an input signal and then output

```
process = _;
```




pMix
Code Editor
Interpolation Space
File
Edit
View
Help
1 Create or select a Faust node to view the code
Audio Input
Juce Summit
Volume Control
Volume
Audio Output

Syntax - Composition

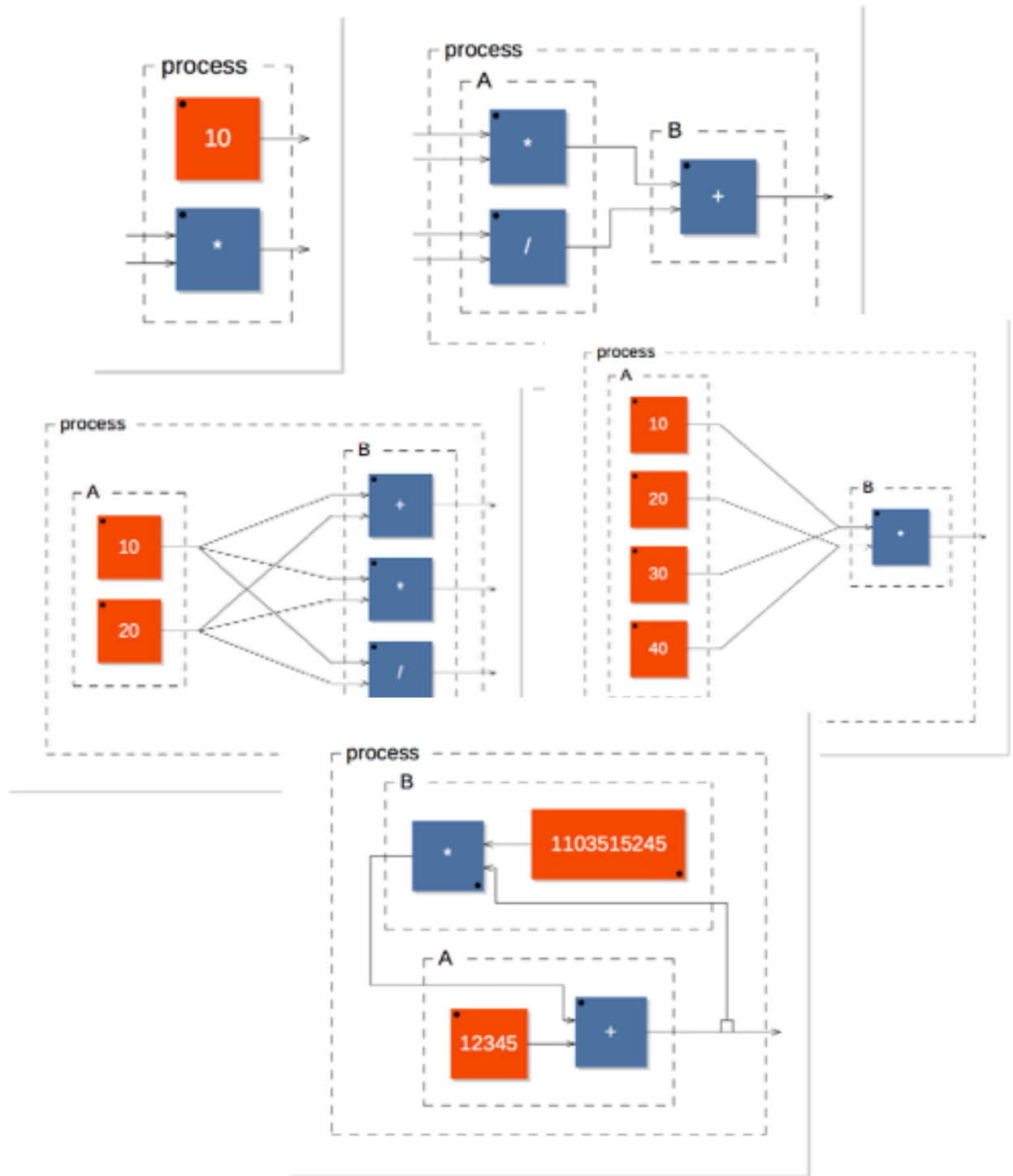
, Parallel composition

: Sequential composition

<: Split composition

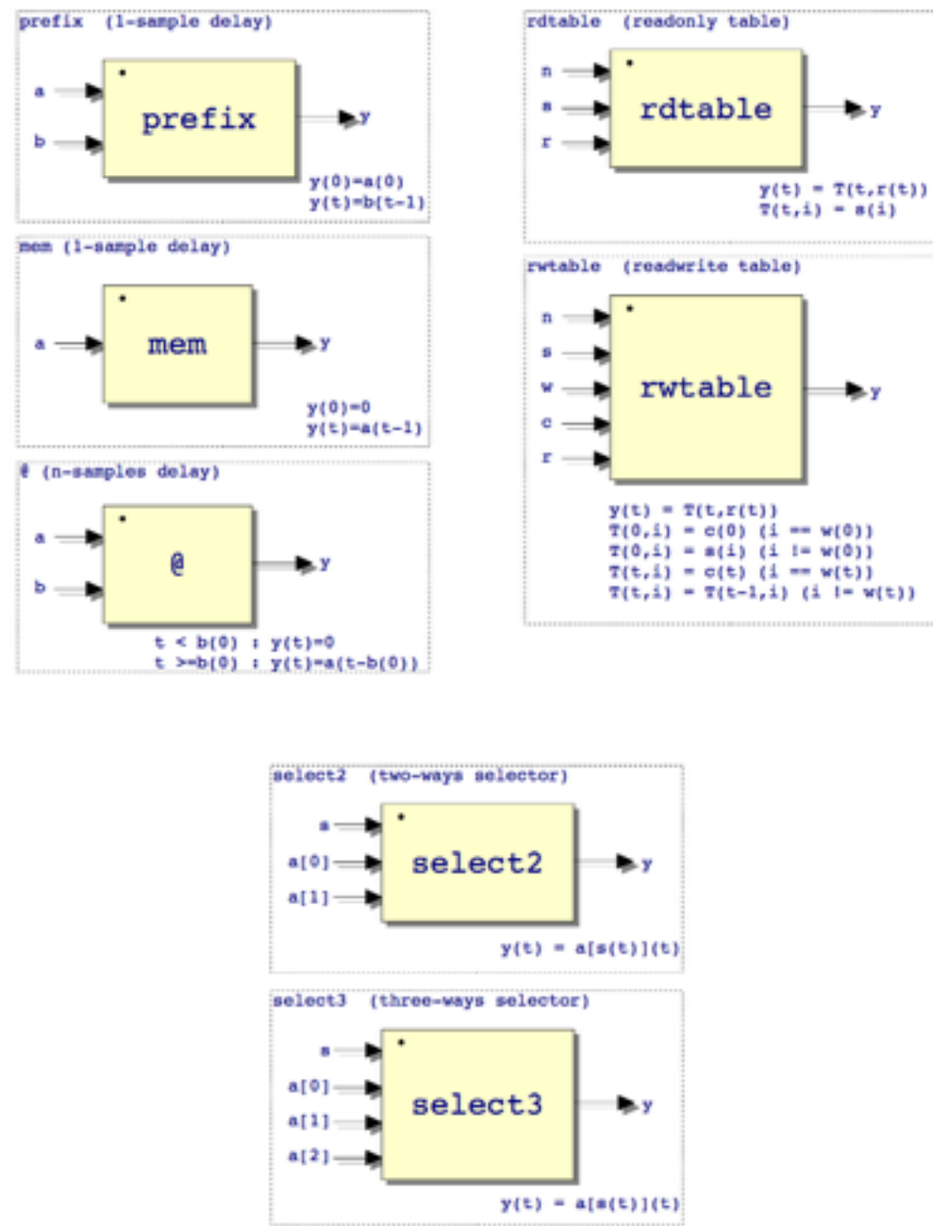
:> Merge composition

~ Recursive composition



Language primitives

- Numerical Primitives (float/int)
- Mathematical operators
- Comparison operators
- Bitwise operators
- C math.h Functions sin(), cos(), tanh() etc
- Delays
- Table read/write
- Signal selectors
- Foreign functions
- UI elements



UI Specification

Syntax	Example
button(*str*)	button("play")
checkbox(*str*)	checkbox("mute")
vslider(*str*,*cur*,*min*,*max*,*step*)	vslider("vol",50,0,100,1)
hslider(*str*,*cur*,*min*,*max*,*step*)	hslider("vol",0.5,0,1,0.01)
nentry(*str*,*cur*,*min*,*max*,*step*)	nentry("freq",440,0,8000,1)
vgroup(*str*,*block-diagram*)	vgroup("reverb", ...)
hgroup(*str*,*block-diagram*)	hgroup("mixer", ...)
tgroup(*str*,*block-diagram*)	vgroup("parametric", ...)
vbargraph(*str*,*min*,*max*)	vbargraph("input",0,100)
hbargraph(*str*,*min*,*max*)	hbargraph("signal",0,1.0)
attach	attach(x, vumeter(x))

Libraries

- **math.lib** (GRAME) - Math Library
- **maxmsp.lib** (GRAME) - MaxMSP compatibility Library
- **music.lib** (GRAME) - Delays, random, noise, spatialisers
- **oscillator.lib** (JOS, CCRMA) - Oscillator Library
- **filter.lib** (JOS, CCRMA) - Filter Library
- **effect.lib** (JOS, CCRMA) - Audio Effect Library
- **instrument.lib** (Romain Michon, CCRMA) - Faust-STK Tools Library

Workflows

Command Line

```
oli$ faust Flanger.dsp -a module.cpp -i -o Flanger.cpp
```

Some common arguments:

- -a = specify an “architecture file”
- -i = inline architecture files
- -o = specify an output file
- -svg = generate svg files for the block diagram
- -single/-double = floating point precision
- -vec = generate code for auto-vectorization

```
#ifndef FAUSTFLOAT
#define FAUSTFLOAT float
#endif



#ifndef FAUSTCLASS
#define FAUSTCLASS mydsp
#endif

class mydsp : public dsp {

  private:

    FAUSTFLOAT fHslider0;
    int fSamplingFreq;

  public:

    void static metadata(Meta* m) {
    }

    virtual int getNumInputs() {
      return 0;

    }
    virtual int getNumOutputs() {
      return 1;

    }
    virtual int getInputRate(int channel) {
      int rate;
      switch (channel) {
            default: {
                  rate = -1;
                  break;
            }

      }
      return rate;

    }

    virtual int getOutputRate(int channel) {
      int rate;
      switch (channel) {
            case 0: {
                  rate = 1;
                  break;
            }
            default: {
                  rate = -1;
                  break;
            }

      }
      return rate;

    }

    static void classInit(int samplingFreq) {

    }

    virtual void instanceInit(int samplingFreq) {
      fSamplingFreq = samplingFreq;
      fHslider0 = FAUSTFLOAT(0.);

    }

    virtual void init(int samplingFreq) {
      classInit(samplingFreq);
      instanceInit(samplingFreq);
    }

    virtual void buildUserInterface(UI* interface) {
      interface->openVerticalBox("0x00");
      interface->addHorizontalSlider("value", &fHslider0, 0.f, -1.f, 1.f, 0.1f);
      interface->closeBox();

    }

    virtual void compute(int count, FAUSTFLOAT** inputs, FAUSTFLOAT** outputs) {
      FAUSTFLOAT* output0 = outputs[0];
      float fSlow0 = float(fHslider0);
      for (int i = 0; (i < count); i = (i + 1)) {
            output0[i] = FAUSTFLOAT(fSlow0);

      }

    }


};
```

Architecture Files

- Architecture files are C++ (etc) boiler plate code that interface the FAUST C++ class with a particular API, such as VST, AU, IOS, Android.

- You can write your own to interface with your platform

faust2xxx scripts

```
faust2alqt
faust2alsa
faust2alsaconsole
faust2android
faust2api
faust2asmjs
faust2au
faust2caqt
faust2csound
faust2dssi
faust2eps
faust2firefox
faust2gen
faust2graph
faust2graphviewer
faust2ios
faust2jack
faust2jackconsole
faust2jackinternal
faust2jackserver
faust2jaqt
faust2ladspa
faust2lv2
faust2lv2synth
faust2mathdoc
faust2mathviewer
faust2max6
faust2md
faust2msp
faust2netjackconsole
faust2netjackqt
faust2octave
faust2owl
faust2paqt
faust2pdf
faust2plot
faust2png
faust2puredata
faust2raqt
faust2ros
faust2rosgtk
faust2rpialsaconsole
faust2rpinetjackconsole
faust2sc
faust2sig
faust2sigviewer
faust2supercollider
faust2svg
faust2vst
faust2vsti
faust2w32msp
faust2w32puredata
faust2w32vst
faust2webaudioasm
```

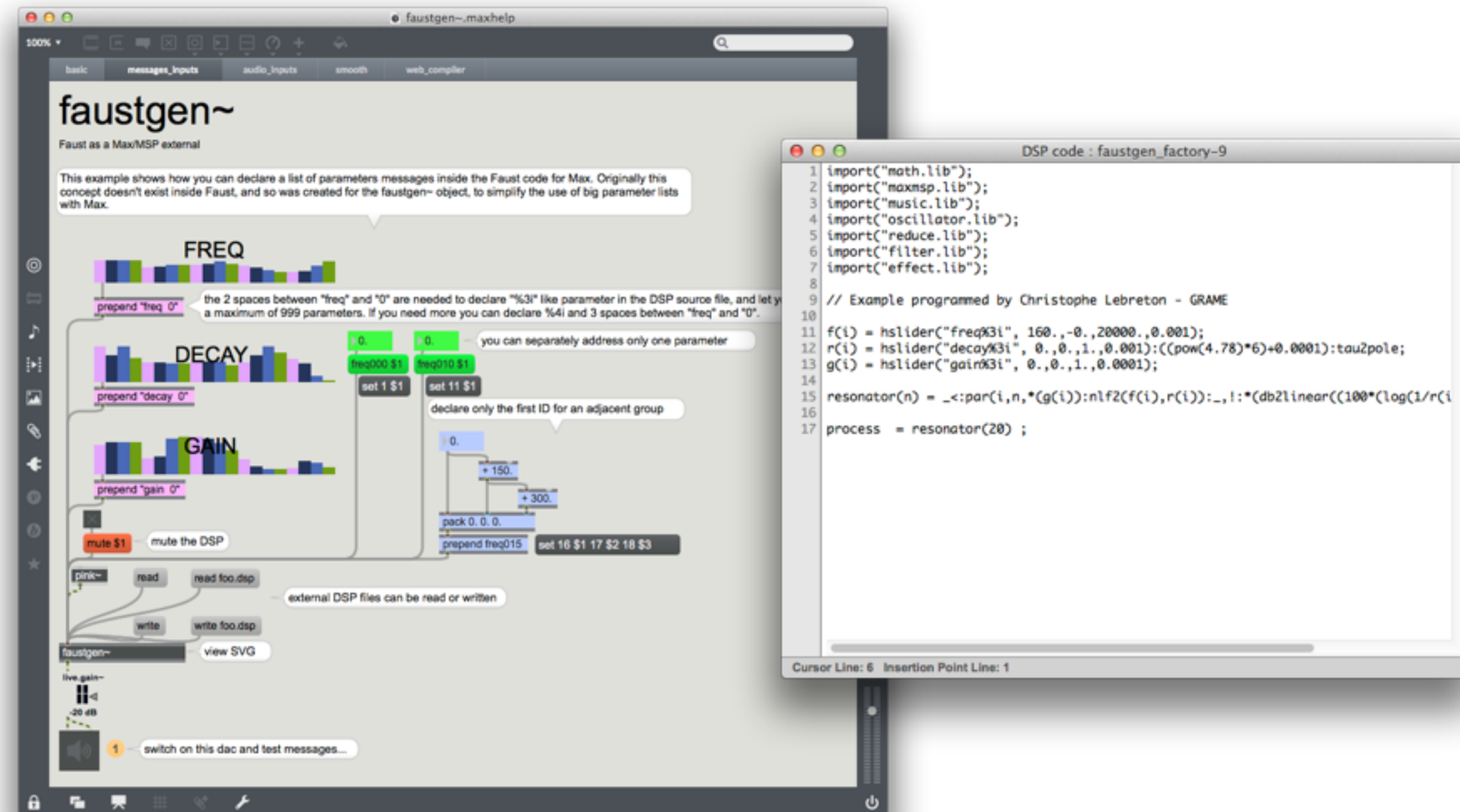
FAUST2 & libfaust

- FAUST2 uses LLVM to provide an embeddable compiler “libfaust”
- Check out the FAUST2 branch from the GIT repository
- More dependencies (LLVM) required to build it, but relatively easy via package managers on OSX and Linux

Faust Live

- A standalone QT FAUST host for OSX/Win/Linux based on libfaust
- Use alongside a text editor - it JIT compiles FAUST **.dsp** files when they change
- Can export **.dsp** file to a variety of architectures using the Faust online compiler
- Works well with Jack Audio Router

faustgen~ Max External



Online Compiler

Faust Online Compiler

faust.grame.fr/onlinecompiler/goto_codeFaust.php

Drop your .dsp file here or write your Faust code below

Faust Code | Compiled Code | Diagrams | Automatic Doc | Exec File

```
declare name     "CZ OSC";
declare version   "1.0";
declare author    "Oli Larkin";
declare license   "BSD";

//import("music.lib");
import("filter.lib");

SMOOTH_TIME = 0.005; // secs

//freq  = hslider("freq [unit:Hz] ", 55, 20, 1000, 1);
dcw = hslider("dcw [OWL:PARAMETER_A]", 0, 0, 99, 1) : smooth(tau2pole(SMOOTH_TIME));
detune = hslider("detune [OWL:PARAMETER_B]", 0, 0, 100, 1) : *(0.01) : smooth(tau2pole(SMOOTH_TIME)) ;

lutsize = 1 << 9;
costable = float(time)*(2.0*PI)/float(lutsize) : cos;
sintable = float(time)*(2.0*PI)/float(lutsize) : sin;

voltz2hz(v) = 440. * pow(2, v);

VOLTAGE_MULTIPLIER = -4.40;
VOLTAGE_OFFSET = 0.028;
OCTAVE_OFFSET = 4;

owl2voltz(v) = (v - VOLTAGE_OFFSET) * VOLTAGE_MULTIPLIER;

// linear interpolation table lookup
tablelookup(normalizedPhase) = s1 + d * (s2 - s1)
  with {
    phase = normalizedPhase *(float(lutsize));
    i = int(phase);
    d = decimal(phase);
    s1 = rdtable(lutsize+1, sintable, i);
    s2 = rdtable(lutsize+1, sintable, i+1);
  };

czresolosc(dcw, freq) = phasor(freq) : *(dcw2freqScalar) : fmod(_, 1.) : tablelookup : *(window)
  with {
    phasor(freq) = freq/float(SR) : (+ : decimal) ~ _;
    window = 1.-(phasor(freq));
    dcw2freqScalar = ((1/16) + (dcw/99)) * 16;
};

process(x) = czresolosc(dcw, voltz2hz(owl2voltz(x)-OCTAVE_OFFSET)), czresolosc(dcw, voltz2hz(owl2voltz(x) -(OCTAVE_OFFSET) +(detune)));
```

Strengths/Weaknesses

Strengths

- Very concise and expressive syntax
- Very quick workflow
- Projects easily portable to a variety of platforms via “architectures”
- Good performance/optimisation possibilities
- Vast library of high quality DSP included
- Generate documentation and diagrams from code

Weaknesses (in current version)

- Currently everything is always on (experimental "with-mute" branch on git addresses this)
- No multi-rate processing (no FFTs or oversampling, yet). No fast convolution.
- Complex syntax / steep learning curve (?)
- Sometimes cryptic errors when syntax is wrong!

Further Info

- FAUST website
  http://faust.grame.fr/

- FAUST @ sourceforge
  http://sourceforge.net/projects/faudiostream/

- https://en.wikipedia.org/wiki/FAUST_(programming_language)

- Julius Smith: Audio Signal Processing in FAUST
  https://ccrma.stanford.edu/~jos/aspf/aspf.pdf

- Tiziano Bole: Faust Tutorial 2
  http://faust.grame.fr/images/faust-tutorial2.pdf

- Romain Michon: Faust Worksop 2015: Online Course
  https://ccrma.stanford.edu/~rmichon/faustWorkshops/course2015/

juce_faustllvm module

juce_faustllvm module

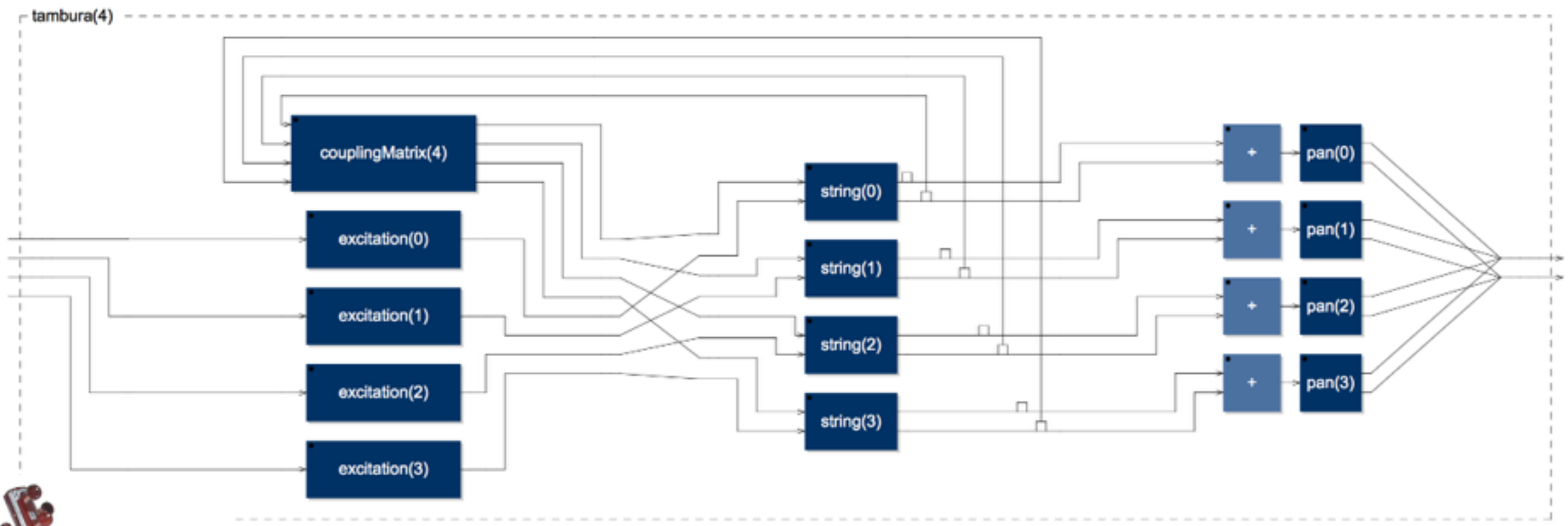
- Based on faustgen~ MaxMSP external by GRAME
- Provides a JUCE AudioPluginInstance/AudioProcessor where the DSP can be JIT compiled from FAUST code
- Parameters based on FAUST UI specification
- Show a generic JUCE widget GUI from FAUST UI spec (not yet implemented)
- Includes supporting files, such as FAUST syntax highlighter

Usage

```
#include "../JuceLibraryCode/JuceHeader.h"

int main (int argc, char* argv[])
{
  // Create instance
  FaustAudioPluginInstance pi;

  // Must initialize with Faust Libraries directory and path for SVG files
  pi.initialize(File::getCurrentWorkingDirectory(), File::getCurrentWorkingDirectory());

  // update sourcecode (just setting every sample to value of parameter)
  pi.setSourceCode("process=hslider(\"value\", 0, -1, 1, 0.1);", true /*compile*/);

  pi.getFactory()->generateSVG();

  pi.prepareToPlay(44100., 8);

  AudioSampleBuffer audioBuffer(1, 8);
  MidiBuffer midiBuffer;

  pi.setParameter(0, 0.5);

  pi.processBlock(audioBuffer, midiBuffer);

  return 0;
}
```

My FAUST Projects

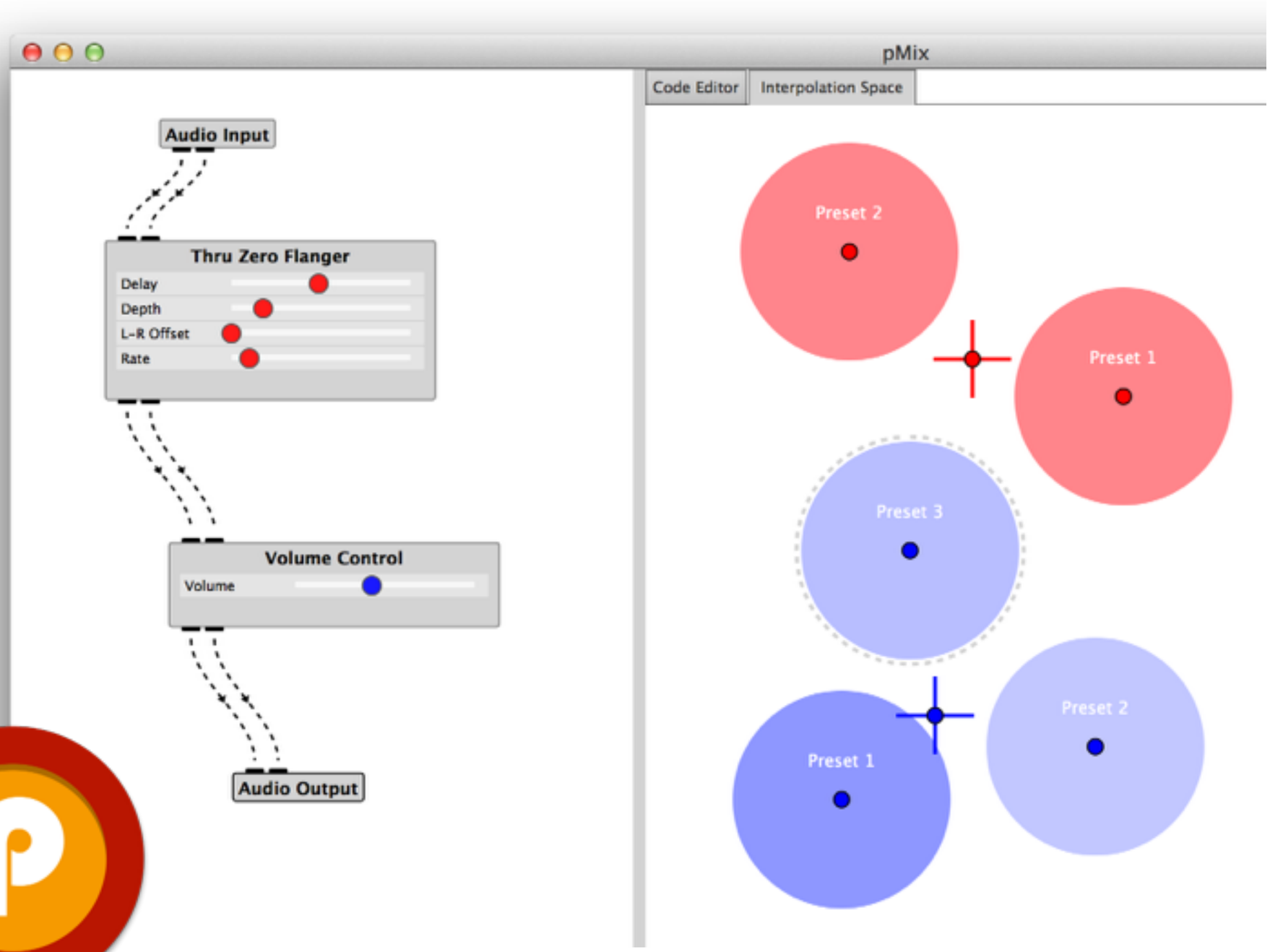
Tambura Physical Model



```
// s = string index
// c = comb filter index (of 9 comb filters in risset string)
tambura(NStrings) = (  couplingMatrix(NStrings), par(s, NStrings, excitation(s)) : inter
                     ) // string itself with excitation + fbk as input
                     ~ par(s, NStrings, (!,_)) // feedback only the right waveguide
                     : par(s, NStrings, (+:pan(s)) // add left/right waveguides and pan
                     ) :> _,_ //stereo output
 with {
```

https://soundcloud.com/olilarkin/sets/tambura-model

pMix v2


pMix
Code Editor
Interpolation Space
Audio Input
Thru Zero Flanger
Delay
Depth
L-R Offset
Rate
Volume Control
Volume
Audio Output
Preset 2
Preset 1
Preset 3
Preset 2
Preset 1

OWL FX Library

https://github.com/olilarkin/OL-OWLPatches

Dual Frequency Shifter
Stereo Frequency Shifter
Thru Zero Flanger
Dual Pitch Shifter
Weird Phaser
Blipper
CZOscillator
…

Downloads

- **juce_faustllvm** and **pMix2** are Open Source and GPL licensed

  https://github.com/CMRCYork/

Thanks! Questions?

contact@olilarkin.co.uk
www.olilarkin.co.uk | www.cmrcyork.org

twitter: @olilarkin