PHAUSTO:
THE SOUND WITHIN PHARO

Domenico Cipriani - 2024

WHAT IS PHAUSTO?

- Phausto is a multi-platform library and API that enables the programming Digital Signal Processors (DSPs) and sound generation in Pharo

- The audio is generated through FFI calls to a *dynamic engine* that computes audio signal by leveraging the power on an embedded FAUST compiler and feeds the buffer of a PortAudio callback

- Phausto has been developed with three main goals:
    1. To enrich Pharo applications with sound;
    2. To allow sound artists and musician to program synthesisers and effects and compose music with Pharo;
    3. To teach DSP programming to beginners and offer a fast prototyping platform for musician and audio developers

WHY FAUST?

- FAUST is a a functional programming language for sound synthesis and audio processing created at the GRAME-CNCM Research Department in Lyon.

- FAUST is considered the state of the art in the research and development of the implementation of time-domain algorithms the can be represented as block diagrams, such as virtual analog synthesisers, filters, waveguide physical models and reverbs

- FAUST standard libraries offers a ready to use extensive collection of sound generators, physical models, DSP helper functions and effects, all resulting from cutting edge audio research supported by a large community

THE UNIT GENERATORS

- Unit Generators (UGens) are basic building blocks for signal processing algorithms first developed by Max Matthews and John E.Muller for the Music III program n 1960.

- Phausto organizes and implement the functions and the semantics of FAUST standard library into Unit Generators subclasses deeply inspired by the ChucK programming language.

- Unit Generators include oscillators, filters, physical models, envelopes and effects such as delays, reverbs and flangers.

INSTALL PHAUSTO

- First, download the packed faustLibraries for your platform, open the package, and copy of the librariesBundle folder into *documents/Pharo/images/yourPhaustimage*

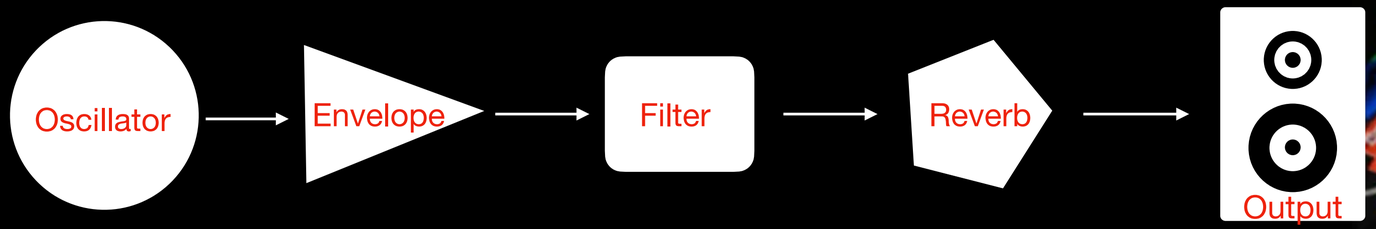
```
Metacello new
  baseline: 'Phausto';
  repository: 'github://lucretiomsp/phausto:main';
  load
```

MODULAR DSP PROGRAMMING

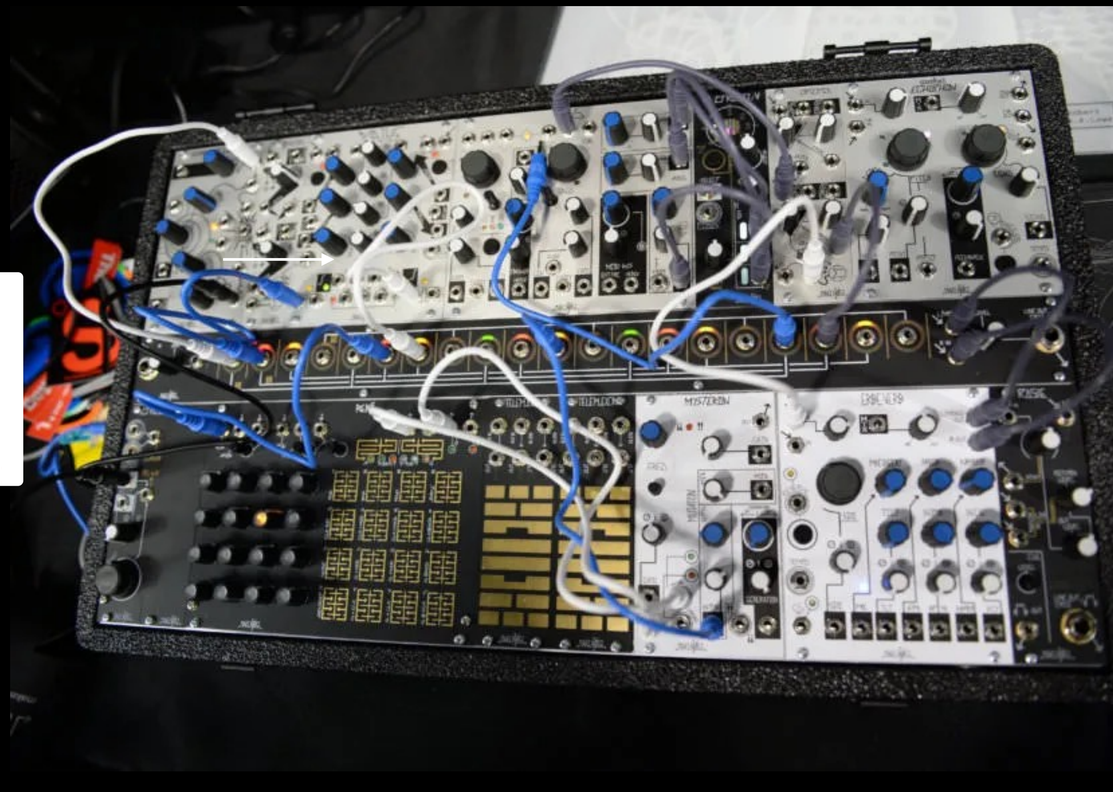
- **Digital signal processing** (**DSP**) is the use of digital processing, to perform a wide variety of signal processing operations focused on analyzing, modifying and synthesizing *signals*, such as sound, images, potential fields, seismic signals.

- Phausto offers an approach to develop and design synthesisers and effect that is inspired by modular synthesiser patching.

- In Phausto, we connect Unit Generator setting their members value or using the **ChucK** operator => .



dsp := SineOsc new => ADSREnv new => ResonLp new => SatRev new.



Domenico Cipriani - 2024

HELLO PHAUSTO

```
"create a Sine wave Oscillator"
sine := SineOsc new.
"creates a stereo DSP from the Oscillator"
dsp := sine stereo asDsp.
"initialize the DSP"
dsp init.
"start the sound"
dsp start.
"stop the sound"
dsp stop.
"destroy the dsp when you no longer need it"
dsp destroy.
```

Domenico Cipriani - 2024

GET FUNKY

```
“Create two pulse generators, the first has its period changed by a LowFrequency Oscillator”
pulse1 := Pulsen new period: (LFOTriPos new freq: 0.2; offset: 0.05; amount: 4) .
pulse2 := Pulsen new period: 0.35.
“Create a djembe, triggered by pulse1”
djembe := Djembe new trigger: pulse1.
marimbaFreq := LFORandomPos new offset: 20; amount: 600; freq: (1 /0.35).
“Create a marimba, triggered by pulse2 and with the frequency modulated by an LFO with a random shape”
marimba := Marimba new trigger: pulse2; freq: marimbaFreq .
“Sum the marimba and the djembe and creates a dsp”
dsp := (djembe + marimba) stereo asDsp.

dsp init.
dsp start.
dsp stop.
```

Domenico Cipriani - 2024

TAKE CONTROL

```
"create a Pulse wave Oscillator with a frequency of 232 hz"
pulse := PulseOsc new freq: 232.
"creates a stereo DSP from the Oscillator"
dsp := pulse stereo asDsp.
"initialize the DSP"
dsp init.
"start the sound"
dsp start.

"create and open  a slider to control the DutyCycle of the PulseOscillator"
f := dsp openSliderFor: 'PulseOscDuty'.
"stop the sound"
dsp stop.
"destroy the dsp when you no longer need it"
dsp destroy.
```

Domenico Cipriani - 2024

TAKE CONTROL (WITH TOPLO)

“Create a djembe triggered by a Pulse train”
djembe := Djembe new trigger: Pulsen new.
“Connect the Djembe to a GreyHole reverb effect and create a stereo DSP”
dsp := (djembe => GreyHole new) stereo asDsp.

“Initialise the DSP”
dsp init.
“Start the sound”
dsp start.

“Open a BlSpace with a knob for all the dsp parameters”
ICDarkKnob5 openForAllParameters: dsp.
“Stop the dsp”
dsp stop.

Domenico Cipriani - 2024

SYNTAX IN A PIT STOP

Domenico Cipriani - 2024

```
"Create your modular synth connecting Unit Generators with the ChucK operator =>"
synth := SquareOsc new => AREnv new => GreyHole new.
"creates a stereo DSP from your "
dsp := synth stereo asDsp.
"initialize the DSP"
dsp init.
"start the sound"
dsp start.
"check which parameters you can modify"
dsp traceAllParams.
"change the value of a parameter in real time"
dsp setValue: 900 parameter: 'SquareOscFreq'.
"rig the envelope"
dsp trig: 'AREnvGate'.
"stop the sound"
dsp stop.
"destroy the dsp when you no longer need it"
dsp destroy.
```

THE CHUCK OPERATOR =>

** it is our playful homage to the creators of ChucK!*

- We friendly adopted* the binary operator => from the ChucK programming language to simplify the connection between Unit Generators.
- The chuckInto: method simplifies and abstract the connections between Unit Generators adhering to the principles of modular synthesis patching

```
UnitGenerator => anotherUnitGenerator

        ^ self chuckInto: anotherUnitGenerator
```

```
UnitGenerator chuckInto: aBox

        aBox patchedWith: self.

        ^ aBox
```

```
UnitGenerator new patchedWith: aBox

        ^ Error new signal:  'You cannnot connect a' ,
aBox className asString , ' with a ' , self className asString
```

```
Envelope new patchedWith: aUnitGenerator

        aUnitGenerator asBox * self asBox
```

```
ResonatorFilter patchedWith: anInput

        self input: anInput
```

Domenico Cipriani - 2024

THE TOOLKIT

- The Toolkit is a comprehensive collection of sound generators, effects and utilities designed to facilitate and accelerate the creation and development of Phausto sounds.

- Inspired by Perry Cook’s and Gary Scavone’s Standard Tool Kit (STK), it is conceived to extend the functionalities of the FAUST libraries.

- At the moment iy includes a Mono Delay with Feedback, an Incrementer, a Resetter, a 4 inputs selectors and a basic Sample Player

Domenico Cipriani - 2024

TURBOPHAUSTO

- TurboPhausto has been inspired by the SuperDirt engine for SuperCollider and it is thought to be the default audio client for Coypu, the pHaro package for programming music on-the-fly

- It will feature a collection of synthesisers, drum machines, instruments, sample players and effect with a default API design

- ZXXZX EXAMPLE
-

Domenico Cipriani - 2024

UNMUTE YOUR PHARO

```
ThreadSafeTranscript subclass: #TurboTranscript
	instanceVariableNames: ''
	classVariableNames: ''
	package: 'PhaustoESUG24'
```

```
TurboTranscript >>>open

	| path sp myDsp |

	"Create an instance of the SamplePlayer"
	sp := SamplePlayer new.
	"Specify a path to an audio files"
	sp pathToFile: path , ‘bonapetit.wav'.
	path := FileLocator documents asAbsolute pathString , '/phDemoSamples/'.

	"Give the sampler a unique name"
	sp name: 'voice'.
	"Compile it  as a DSP, initialize it and start it"
	myDsp := sp stereo asDsp.
	myDsp init.	myDsp start.
	^ self openLabel: ' T U R B O T R A N S C R I P T'
```

Domenico Cipriani - 2024

ALGORITHMIC COMPOSITIONS

```
"PsgPlus is a TurboPhausto synth inspired by Sega Master System PSG (Programmable
Sound Generator,  a clone of the SN76489 chip used in the Texas Instruments TI-99/4A and
TI-99/8 home computers.”

dsp := ( PsgPlus new  => DelayMonoFB new )stereo asDsp.
dsp init.
dsp start.

"sonification of Collection subclasses"
Collection subclasses do: [ :c | dsp playNote: c selectors size prefix: 'PsgPlus'  dur: 0.12.
                (Delay forSeconds: 0.16) wait    ].
```

Domenico Cipriani - 2024

THE FUTURE ISN'T WRITTEN YET

- All the functions of FAUST standard libraries should be implementedas Phausto Unit Generators classes or as methods
- Support for external MIDI Input (keyboards and controllers) with Pharo Sound
- Extend the Toolkit
- Design a basic set of instruments and effect for TurboPhausto
- Implement a robust and modern UI with Toplo.