PHAUSTO: FAST AND ACCESSIBLE DSP PROGRAMMING FOR SOUND AND MUSIC CREATION IN PHARO.

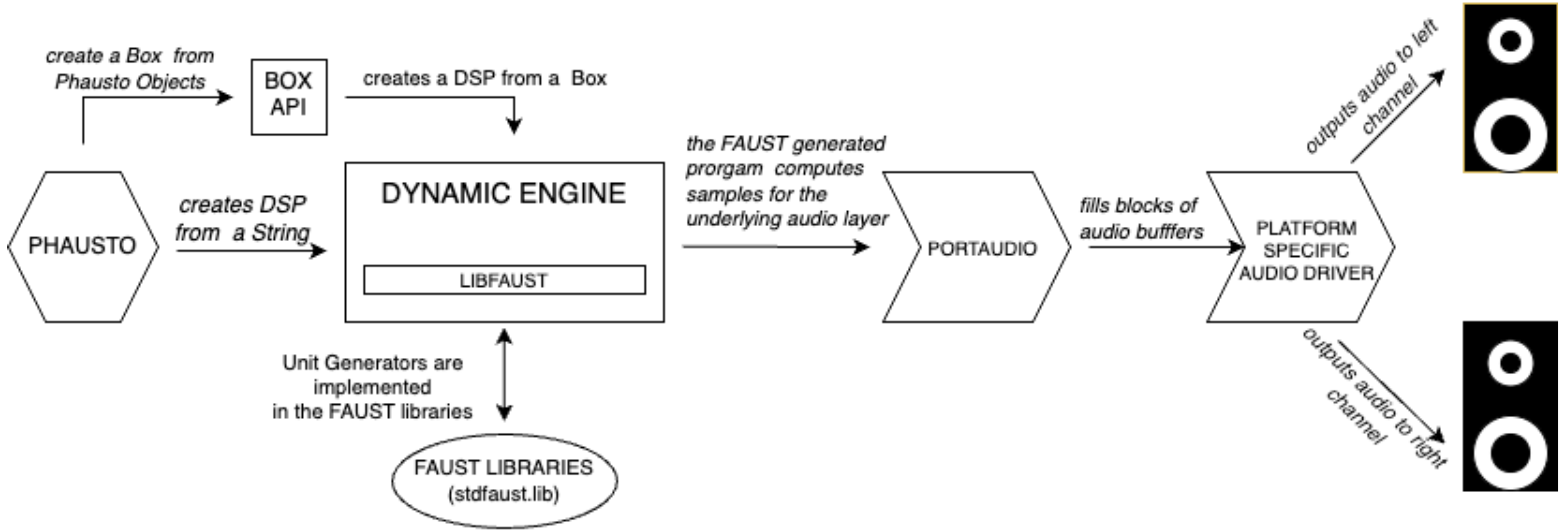
Domenico Cipriani, Sebastian Jordan Montano, Nahuel Palumbo, Stephane Ducasse

What is Phausto

- Phausto is a library and API that enables sound generation and audio Digital Signal Processing programming in Pharo.
- A dynamic library, created and maintained by Stephane Sletz at GRAME (Lyon), computes the output of the synthesisers and effects created with Phausto, using the FAUST compiler.
- The dynamic library also fills the blocks of the PortAudio buffer, which are finally transformed into sound by the platform specific audio drivers

Domenico Cipriani, 2024

Tuned for performance


create a Box from Phausto Objects
BOX API
creates a DSP from a Box
PHAUSTO
creates DSP from a String
DYNAMIC ENGINE
LIBFAUST
the FAUST generated prorgam computes samples for the underlying audio layer
PORTAUDIO
fills blocks of audio bufffers
PLATFORM SPECIFIC AUDIO DRIVER
outputs audio to left channel
outputs audio to right channel
Unit Generators are implemented in the FAUST libraries
FAUST LIBRARIES (stdfaust.lib)

Domenico Cipriani, 2024

What is Faust

- Faust is a functional programming language for sound synthesis and audio processing, created at GRAME - CNCM Research Department.

- It provides developers with an alternative to C++ for designing and deploying DSPs

- The core component of Faust is its compiler. It allows to "translate" any Faust digital signal processing (DSP) specification to a wide range of non-domain specific languages such as C++, C, LLVM bit code, WebAssembly, Rust, etc.

- Thanks to a wrapping system called "architectures," codes generated by Faust can be easily compiled into a wide variety of objects ranging from audio plug-ins to standalone applications and web apps. It is thanks to one of this architecture, the *dynamic-engine* that is possible too embed Faust in Pharo

Domenico Cipriani, 2024

Our target audience

1. Pharo programmers who wants to include sounds or sonic interaction in their Pharo applications, including procedural sound coming from Faust extensive physical modelling synthesis library.
2. Artists with little or no computer literacy who want to design and develop synthesisers and effects for their music creation fast and easily.
3. Students and beginners who want develop their own audio plug-ins, by exporting DSPs developed in Pharo to CMajor[3] patch thanks to the Faust -2CMajor export[4].

Domenico Cipriani, 2024

The dynamic-engine and the backend interpreter

- The dynamic engine provides the flexibility to choose to associate *libfaust* with an LLVM Intermediate Representation (IR) backend or with an interpreter backend

- The interpreter backend generates bytecode from the FAUST Imperative Representation, which is then successively executed by a stack-based virtual machine.

- The code generated by the interpreter backend is slower than the native code generated by LLVM, but it meets the performance needs of Phausto intended goals and audience. Indeed, we chose the interpreter backend for its lightweight nature and zero dependencies.

Domenico Cipriani, 2024

What is PortAudio

- The PortAudio is an Open Source Cross Platform C library and API for audio input and output originally, developed by Ross Bencina and Phil Burk. It was designed to simplify the development of real- time audio application, and it is part of a larger initiative called PortMusic that also includes MIDI capabilities.

- PortAudio handles the connection with the audio input and audio ports on the host platform. It internally manages audio stream buffers and requests audio processing from the client application via a callback that is associated with an opened stream.

- In the dynamic engine, this association is managed by the following function:

```
bool initDsp ( dsp * dsp , RendererType renderer , int sr , int bsize ) ;
```

- For instance, in Phausto we initialise a DSP by calling:

```
initDsp(aDSP, 0, 44100, 512);
```

Domenico Cipriani, 2024

Create a DSP from a string of Faust code

```
content := 'import("stdfaust.lib"); tempo = hslider("tempo", 4410, 300, 44100, 100); freq = hslider("freq", 440, 200, 900, 100); process = ba.pulsen(1, tempo) : pm.djembe(freq, 0.3, 0.4, 1) <: dm.freeverb_demo;'.
dsp := DSP create: content.
dsp init.
dsp start.
dsp openSliderFor: 'tempo'.
dsp sliderFor: 'freq'.
dsp sliderFor: 'Freeverb/0x00/RoomSize'.
dsp stop.
dsp destroy.
```

Domenico Cipriani, 2024

Phausto API

- Phausto API is designed to be modern and straightforward.

- The subdivision of Faust’s standard library functions into Phausto Unit Generators subclasses is deeply inspired by the ChucK programming language, which in turn takes this concept from MUSIC-N style programming music language.

- Unit Generators (UGens) are basic building blocks for signal processing algorithms that were first developed by Max Mathews and Joan E. Miller for the Music III program in 1960. UGens include processing modules such as oscillators, filters, envelopes and effects that can be connected to create synthesis instruments (sometimes referred as *patches*).

- UnitGenerators in Phausto are implemented using the Faust Box API.

Domenico Cipriani, 2024

Create a DSP from a from Boxes

```
djembe := Djembe new trigger: Pulsen new.
dsp := (djembe => FreeVerbDemo new) asDsp.
dsp init.
dsp start.
dsp stop.
```

Domenico Cipriani, 2024

The BOX API

- The FAUST C box API[14] allows for the programmatic creation of a box expression, which is used to create a DSP object. This stage is a new intermediate public entry point created in the *Semantic Phase* of FAUST's compilation chain.

- Boxes can be created by calling a specific function defined in *libfaust-box-c:*

```
Box CboxButton ( const char * label ) ;
```

- Boxes can be also be created from a string of Faust code

```
Box CDSPToBoxes(const char* name_app, const char* dsp_content, int argc, const char* argv[] , int* inputs,
    int* outputs, char* error_msg) ;
```

Domenico Cipriani, 2024

Algorithmic Composition 101

```
synth := (SquareOsc new freq: #SquareNote ) * (AREnv new trigger: #SquareGate). dsp := (synth connectTo: SatRev new) stereo asDsp.
dsp init.
dsp start
note := 72.
time := 2.
[24 timesRepeat: [dsp playNote: a prefix: 'Square' dur: 0.1. time wait. note = note + 1 . time := time * 0.85] ]fork
dsp stop.
dsp destroy.
```

Domenico Cipriani, 2024

Core implementation

- Phausto allows Pharo users to use functions and data structures from the FAUST via Foreign Function Interface (FFI) [to *libdynamic engine.dylib* and *libfaust.2.dylib.*

- *libdynamic engine.dylib* contains the API to instantiate, access and modify DSPs.

- *libfaust.2.dylib* is used to create Boxes which enables us to build a Pharo API for the FAUST programming language and its rich set of libraries.

- In order to use Phausto, the *librariesBundle* folder must be downloaded from the GitHub repository and placed next to the current Pharo image.

Domenico Cipriani, 2024

Conclusion and future work

- In its first year of development, Phausto has become stable and we have been label to provide bindings to more the 30% of the FAUST Box API. Additionally, 25% of the functions of the FAUST standard library have been implemented as classes and tested

- In the following months, our focus will be will be on porting all the functionalities of the FAUST programming language. This includes the ability of visualise the DSP block diagram as an .svg file and the possibility to save DSP written in Pharo as files, which can ber exported to different target languages, such as CMajor, C++ and WAST[21]

- In addition, we have planned to develop a new package called **TurboPhausto** specifically designed to program music on-the-fly with Coypu[22], the package we have developed in the last 3 years for live coding. **TurboPhausto i**s deeply inspired by the SuperDirt engine for SuperCollider

- In collaboration with the Evref team, we are working on a custom Playground and a set of custom UI elements (faders, rotary sliders, buttons, and more)for Phausto made with Toplo[23].

Domenico Cipriani, 2024