Using Faust in JUCE Projects

Julius Smith
CCRMA, Stanford University

ADCx-2023, San Francisco

May 15, 2023

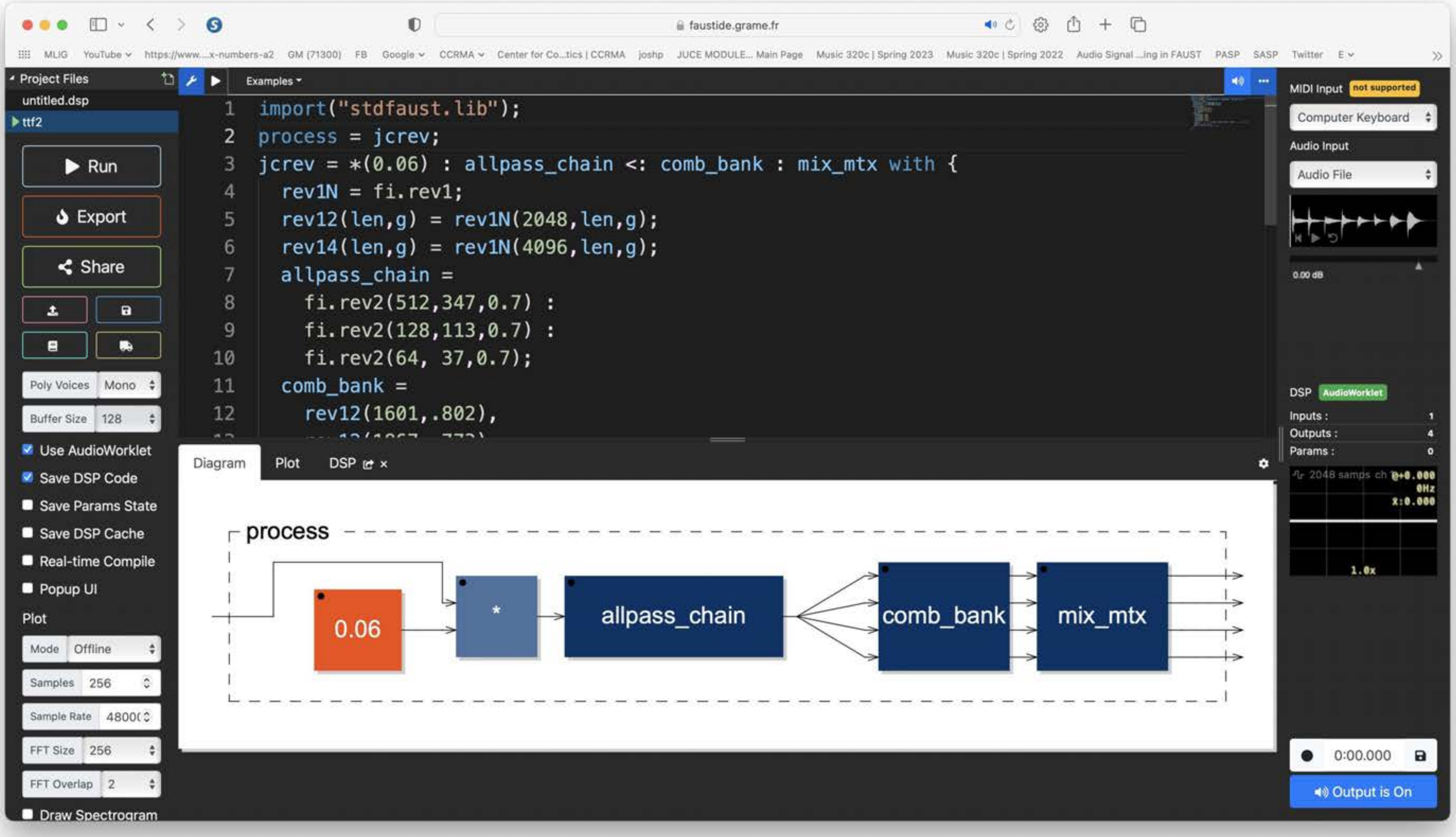
Faust
Faust → JUCE
Faust → AI → JUCE

Faust (Functional AUdio STream)

Why Faust?

- **Goal:** *Develop C++ for* audio signal processing ***ASAP***
- **Results** (for me):
  - **Faust** speeds C++ development by about an *order of magnitude*
  - **ChatGPT-4:** less effective due to *much more debugging time* (code *untested*, *out-of-date*, and occasionally *deranged*)
- In my tests, ChatGPT-4 typically performs *below college level* in *elementary signal processing, complex variables, calculus,* and *Newtonian physics*.
- **Writing Faust *by hand* is the clear winner for me right now**

What is Faust? (Functional AUdio STream)



```
import("stdfaust.lib");
process = jcrev;
jcrev = *(0.06) : allpass_chain <: comb_bank : mix_mtx with {
  rev1N = fi.rev1;
  rev12(len,g) = rev1N(2048,len,g);
  rev14(len,g) = rev1N(4096,len,g);
  allpass_chain =
    fi.rev2(512,347,0.7) :
    fi.rev2(128,113,0.7) :
    fi.rev2(64, 37,0.7);
  comb_bank =
    rev12(1601,.802),
```

Poly Voices: Mono
Buffer Size: 128
☑ Use AudioWorklet
☑ Save DSP Code
☐ Save Params State
☐ Save DSP Cache
☐ Real-time Compile
☐ Popup UI

Plot
Mode: Offline
Samples: 256
Sample Rate: 4800(
FFT Size: 256
FFT Overlap: 2
☐ Draw Spectrogram

MIDI Input: not supported
Computer Keyboard
Audio Input: Audio File
DSP: AudioWorklet
Inputs : 1
Outputs : 4
Params : 0

Faust Online Integrated Development Environment (IDE)

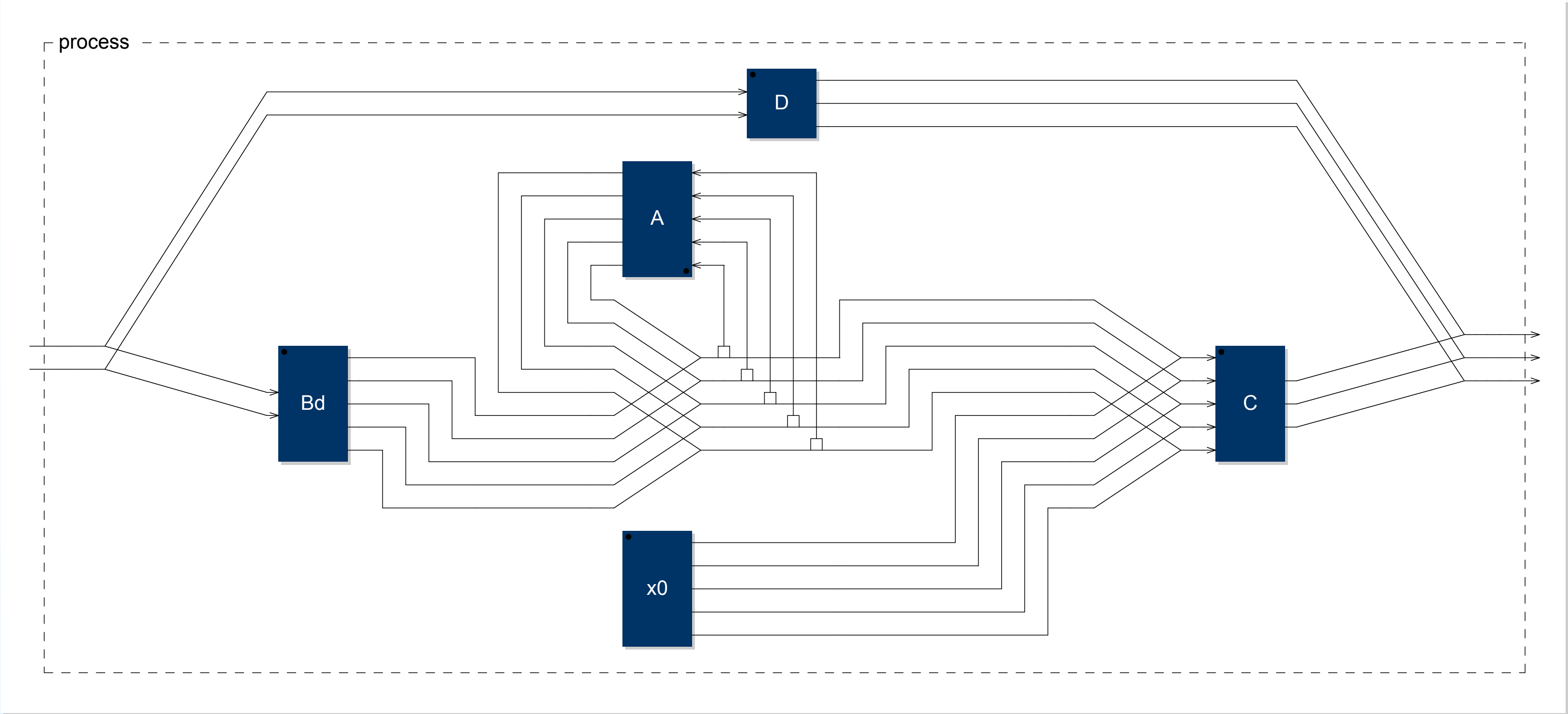
Faust Overview

- Excellent *domain-specific language* for specifying block diagrams
- A *block diagram* is an *audio processor* with *multiple I/O streams*
- Especially natural for *left-to-right signal flow with feedback loops*
- *Function definitions* compile into *block-diagrams*
  - Block diagrams easy to check (automatically drawn by compiler)
  - When it looks right, it’s usually right
  - No debugging on the sample level
  - This is mainly where I get my order-of-magnitude acceleration
- Generates C++ or any of several other back-end languages (C, C#, CMajor, Java, JAX, Julia, Rust, wasm, . . .)
- *GUI primitives* in the language enable useful *standalone programs*
- *Architecture files* provide support for many platforms (Linux, Mac, Windows, Browser, Unity, JUCE, Android, iOS, VCVRack, Owl, Bela, Teensy, ...)

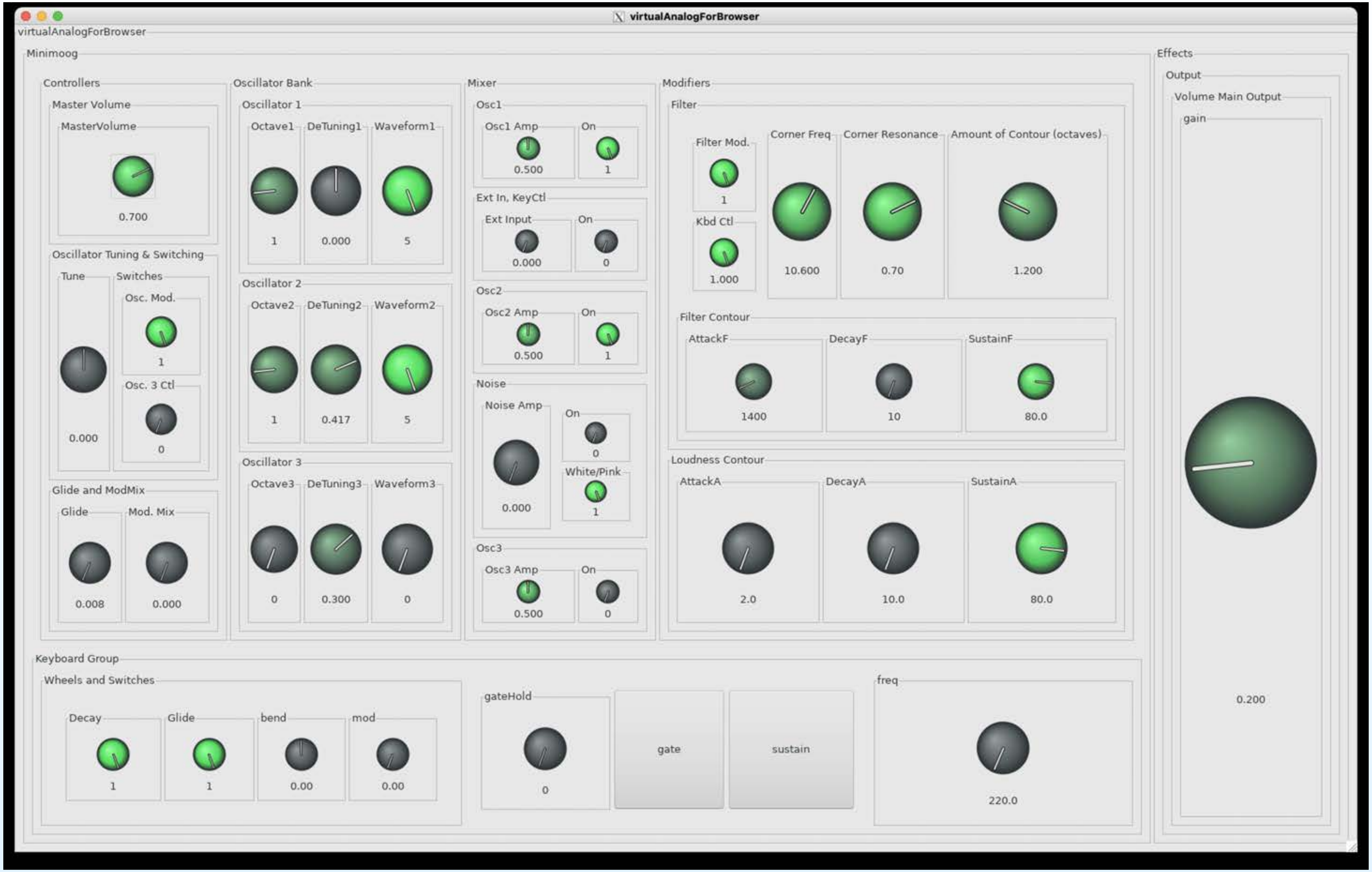
Generality: State Space Formulation in Faust

```
process = si.bus(p) <: D, (Bd : vsum(N)~(A), x0 : vsum(N) : C) :> si.bus(q)

Bd = par(i,p,mem) : B; // input delay needed for conventional definition
vsum(N) = si.bus(2*N) :> si.bus(N); // vector sum of two N-vectors
impulse = 1-1'; // unit step minus delayed unit step
x0 = par(i,N,i*impulse); // initial state = (0,1,2,3,...,N-1)
```



Faust Standalone App (≈Minimoog)



virtualAnalogForBrowser

Minimoog

Controllers

Master Volume
MasterVolume 0.700

Oscillator Tuning & Switching
Tune 0.000
Switches
Osc. Mod. 1
Osc. 3 Ctl 0

Glide and ModMix
Glide 0.008
Mod. Mix 0.000

Oscillator Bank

Oscillator 1
Octave1 1
DeTuning1 0.000
Waveform1 5

Oscillator 2
Octave2 1
DeTuning2 0.417
Waveform2 5

Oscillator 3
Octave3 0
DeTuning3 0.300
Waveform3 0

Mixer

Osc1
Osc1 Amp 0.500
On 1

Ext In, KeyCtl
Ext Input 0.000
On 0

Osc2
Osc2 Amp 0.500
On 1

Noise
Noise Amp 0.000
On 0
White/Pink 1

Osc3
Osc3 Amp 0.500
On 0

Modifiers

Filter
Filter Mod. 1
Kbd Ctl 1.000
Corner Freq 10.600
Corner Resonance 0.70
Amount of Contour (octaves) 1.200

Filter Contour
AttackF 1400
DecayF 10
SustainF 80.0

Loudness Contour
AttackA 2.0
DecayA 10.0
SustainA 80.0

Keyboard Group

Wheels and Switches
Decay 1
Glide 1
bend 0.00
mod 0.00

gateHold 0

freq 220.0

Effects

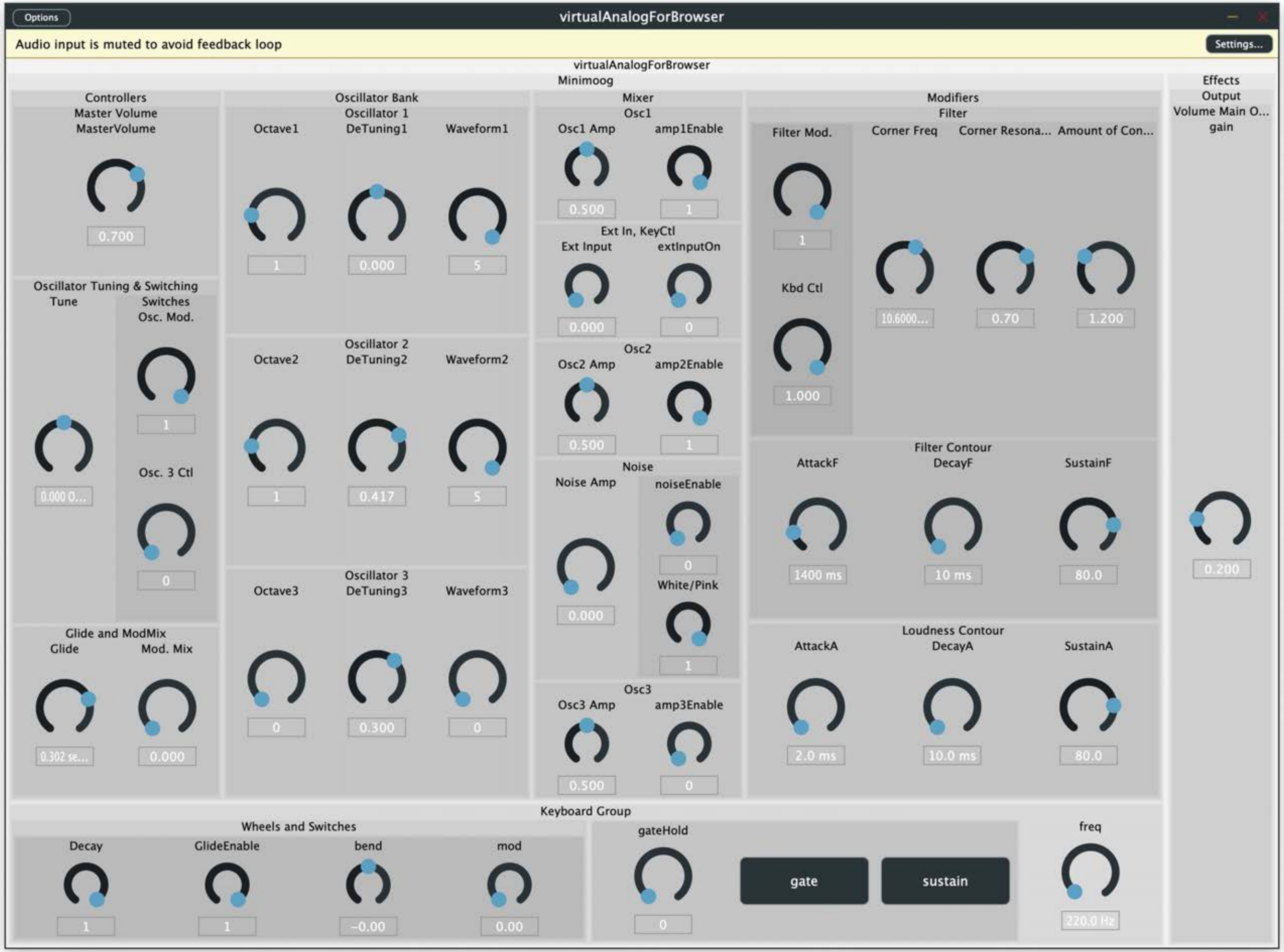
Output

Volume Main Output
gain 0.200

`faust2caqt`

Faust → JUCE

Faust App to JUCE Plugin via `faust2juce`



Audio input is muted to avoid feedback loop

MasterVolume: 0.700
Tune: 0.000 0...
Osc. Mod.: 1
Osc. 3 Ctl: 0
Glide: 0.302 se...
Mod. Mix: 0.000
Octave1: 1
DeTuning1: 0.000
Waveform1: 5
Octave2: 1
DeTuning2: 0.417
Waveform2: 5
Octave3: 0
DeTuning3: 0.300
Waveform3: 0
Osc1 Amp: 0.500
amp1Enable: 1
Ext Input: 0.000
extInputOn: 0
Osc2 Amp: 0.500
amp2Enable: 1
Noise Amp: 0.000
noiseEnable: 0
White/Pink: 1
Osc3 Amp: 0.500
amp3Enable: 0
Filter Mod.: 1
Kbd Ctl: 1.000
Corner Freq: 10.6000...
Corner Resona...: 0.70
Amount of Con...: 1.200
AttackF: 1400 ms
DecayF: 10 ms
SustainF: 80.0
AttackA: 2.0 ms
DecayA: 10.0 ms
SustainA: 80.0
gain: 0.200
Decay: 1
GlideEnable: 1
bend: -0.00
mod: 0.00
gateHold: 0
freq: 220.0 Hz

Faust to JUCE

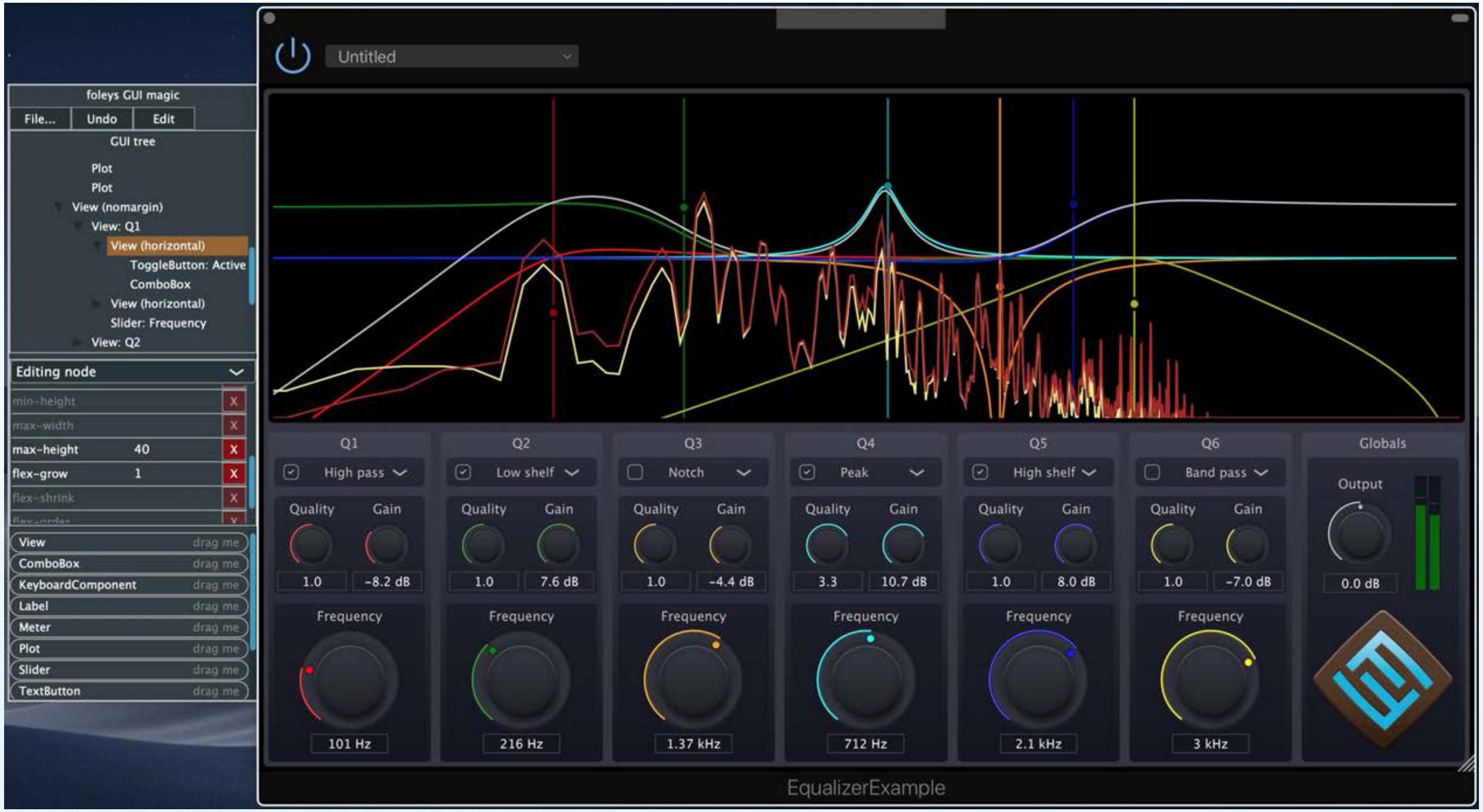
- Faust *standalone apps* are great for initial testing (`faust2caqt`, . . .)
- `faust2juce` (bash script) wraps a JUCE project around that
- For *audio plugins* (AU, VST, . . .), we often need *bypassable modules:*
    - Generate header files from Faust (`osc.h`, `vcf.h`, etc.)
    - Manage bypass in C++ (`juce::AudioProcessor::processBlock()`)
- More general GUIs:
    - Usually programmed directly in C++ (`juce::Component::paint()`)
    - JUCE Team is working on *Web-view based GUIs* (embedded browsers)
    - My own GUIs are “fast research prototypes”
        - Originally in Projucer’s (deprecated) GUI Editor (fine)
        - Now using Daniel Walz’s *Plugin GUI Magic* (PGM):

Faust

Faust ⟶ JUCE
- Minimoog to JUCE
- Faust to JUCE
- Plugin GUI Magic
- Faust Parameters

Faust ⟶ AI ⟶ JUCE

JUCE GUI using Foleys Plugin GUI Magic (PGM)



Create Faust parameter-wrappers and hook them up here

Faust Parameter Binding

Parameters are different in Faust and JUCE:

- *Faust parameters live in the DSP module* (e.g., `hslider` state)
  - Faust GUIs *point* to DSP state variables (e.g., `&fHslider0`)
  - Faust parameters (`FAUSTFLOAT`) *assumed atomic in hardware*
- JUCE audio plugin parameters live in both the `AudioProcessor` and (optional) `AudioProcessorEditor`
  - Each parameter has a set of *listeners*
  - Each GUI element has a set of *listeners*
  - GUI elements “attach” to parameters that listen to them
  - *“Raw parameter”* pointers also available (more like Faust)
  - Each parameter exists in *two or three places* that must be *synchronized*
- Question: Can a `juce::AudioProcessorParameter` *raw-pointer* “refer to” a *Faust state-variable* address?

ChatGPT Weighs In

Ask ChatGPT-4: Can JUCE Use Faust Parameters?

Prompt:

> ... can I make my [JUCE] gain parameter refer to a pre-existing memory location? ...

ChatGPT-4:

> Yes, you can make the gain parameter refer to a pre-existing memory location. You can create your own custom parameter class by inheriting from `juce::RangedAudioParameter` and overriding the `getValue`, `setValue`, and `getDefaultValue` methods.
>
> Here's an example of how to create a custom parameter class that refers to an existing memory location:

While the suggested code was incomplete, it had the right general idea, and was easily completed by referring to `juce_AudioParameterFloat.cpp` et al.

Use Faust Parameter References as JUCE Parameters?

I decided *no,* because:

- We already have `std::atomic<float>*` *raw pointers*
- Copies can be refreshed after every *set-parameter* event (such as a MIDI controller message)
- In `processBlock()`, each Faust `compute()` can run to the next MIDI timestamp followed by parameter refresh

Minimum Latency Faust Processing

JUCE Pseudo-Code (see also `juce::SynthesiserVoice::renderNextBlock()`):

```
processBlock (... audioBuffer, ... midiMessages) {
  int currentTime = 0;
  for (auto m : midiMessages ) {
    int nextEventTime = m.getTimeStampSamples(); // MIDI timestamp
    if (nextEventTime > currentTime) // compute audio to nextEventTime:
      faustModule.compute(nextEventTime - currentTime,
            faustBuffersAtCurrentTime()); // pointers into audioBuffer
    processMidiMessage(m); // JUCE parameters possibly set
    currentTime = nextEventTime; // refresh Faust parameters
    *faustParam1Address = *juceParamRawPointer1;
    *faustParam2Address = *juceParamRawPointer2;
    ...
  }
  int remainingSamples = audioBuffer.getNumSamples() - currentTime;
  faustModule.compute(remainingSamples, faustBuffersAtCurrentTime());
}
```

ChatGPT-3.5 Can “Sort of” Translate Faust to C++

Why would anyone want to translate Faust to C++
when Faust *compiles* to C++?
Possible reasons:

1. Squeeze out more performance (e.g., SIMD)
2. Convert pattern-matching “macro arguments” to arguments changeable at run time

My motivation was case 2:

- Faust’s `fi.lowpass(Order,cutoffHz)` requires `Order` be fixed at compile time (for recursive pattern matching)
- I needed *all* orders available between 1 and 50 or so (that’s a lot of `Lowpass` instances managed in some pointer array)
- Could I rewrite `fi.lowpass` in C++ using ChatGPT?
- Yes! ChatGPT-3.5 got me most of the way there in one iteration
- As a side task, Faust comments were converted to Doxygen format

Faust

Faust ⟶ JUCE

Faust ⟶ AI ⟶ JUCE

- ChatGPT-4 params
- Faust Param Refs
- Fast Faust Params
- Faust to C++
- ChatGPT Translations
- Conclusions

Easier ChatGPT Translations

The following were much easier than Faust-to-C++ translation using ChatGPT-3.5:

- MATLAB/Octave to Python
- MATLAB/Octave to C++
- Convert randomly commented C++ to Doxygen-format comments
- Write C++ *unit tests* for given C++ functions (debugging needed)

Conclusions

- Faust is an excellent source of signal-processing functions that are easily incorporated into JUCE projects
- Writing Faust manually, leveraging Faust Libraries, is still best
- ChatGPT-4 knows JUCE pretty well, but may use *old, obsolete API*
  (Most debugging time was upgrading the API and repairing “lapses”)
- We need **“JUCE-GPT”**
  == C++ assistant fine-tuned on latest JUCE Distribution
- More about *Butterworth Faust-to-C++ translation* and others on “Faust Day” at CCRMA this Saturday (free online and in-person):
  https://ccrma.stanford.edu/events/faust-day-ccrma-whats-with-faust-in-2023
- ChatGPT-4 is most useful right now for
  - general-purpose C++ development,
  - well-known language translations,
  - writing unit tests,
  - formatting comments,
  - and of course much more

In Closing

Prompt:

Write a limerick about JUCE

ChatGPT-4:

There once was a toolkit named JUCE,
For audio apps, it let loose.
With ~~its~~ code so concise,
It's ~~as~~ nice as spiced rice,
In ~~the~~ [our] world of sound, it's a deuce!

More ChatGPT Fun and Profit

Improved `createParameterLayout()`

Prompt:

In C++ using the JUCE framework, write `createParameterLayout()` to return a parameter layout based on the following abbreviated notation: `<type> <name> <min> <default> <max>`. Type bool only has a default. Type choice additionally has choices specified as a string enclosed in curly braces. For example, “int myInt 0 10 100” denotes an int ranging from 0 to 100 with a default value of 10:

```
int freqHz 20 440 8000
float gainDB -80 -10 0
bool enable true
choice waveform 0 1 3 {"Saw", "Triangle", "Square", "Sine"}
```

ChatGPT-4:

Now, you can create the parameter layout using the provided abbreviated notation: ...

Improved `createParameterLayout()`

ChatGPT-4:

> Now, you can create the parameter layout using the provided abbreviated notation:
>
> ```
> juce::String parameterData = R"(
> int freqHz 20 440 8000
> float gainDB -80 -10 0
> bool enable true
> choice waveform 0 1 3 {"Saw", "Triangle", "Square", "Sine"})";
>
> auto layout = createParameterLayout(parameterData);
> ```

Debugging was needed for `choice`.

The method was sound, but there was disorientation in the `choice` details.

Everything else was right on.

Summary of ChatGPT Experience to Date

Current take-aways (changing fast!):

- **ChatGPT-3.5 saved much time** translating Faust's Butterworth filter design functions to C++, maybe by an order of magnitude
- **ChatGPT-4 has a much better grip on the Faust language** than v3.5
- Even when way off overall, it often has good coding ideas and usable outputs
- **Unit tests** (or equivalent) with 100% coverage are *important and required*
- **Faust** is understandably **less well known** compared to mainstream languages
- **Writing Faust manually is normally faster than debugging ChatGPT's C++ in the field of signal-processing**