Revista Vórtex | Vortex Music Journal | ISSN 2317–9937 | http://vortex.unespar.edu.br/
D.O.I.: https://doi.org/10.33871/23179937.2021.9.2.23

EXC!TE SNARE DRUM

— Making an Audio Plugin with Pure Data inside

Max Neupert, Clemens Wegener, Philipp Schmalfuß, Sebastian Stang

The Center for Haptic Audio Interaction Research / Bauhaus-Universität Weimar | Germany

Abstract: This report describes how we made a VST3 plugin containing Pure Data and integrated libpd into VCV Rack, iPlug2 and the VST3SDK. The plugin is a real-time snare drum synthesizer using an exciter-resonator model. We discovered an undesirable effect in Pd where the computationally cheap 4-point interpolation on delread4~ creates audible artefacts, effecting our wave-guide. Our solution to this issue was to implement our own interpolation object based on advice from Cyrille Henry posted to the Pd mailing list in 2008. The implementation was taken from Julius O. Smith's Digital Audio Resampling reference book.

Keywords: libpd, VCV Rack, VST3, wave-guide, interpolation.

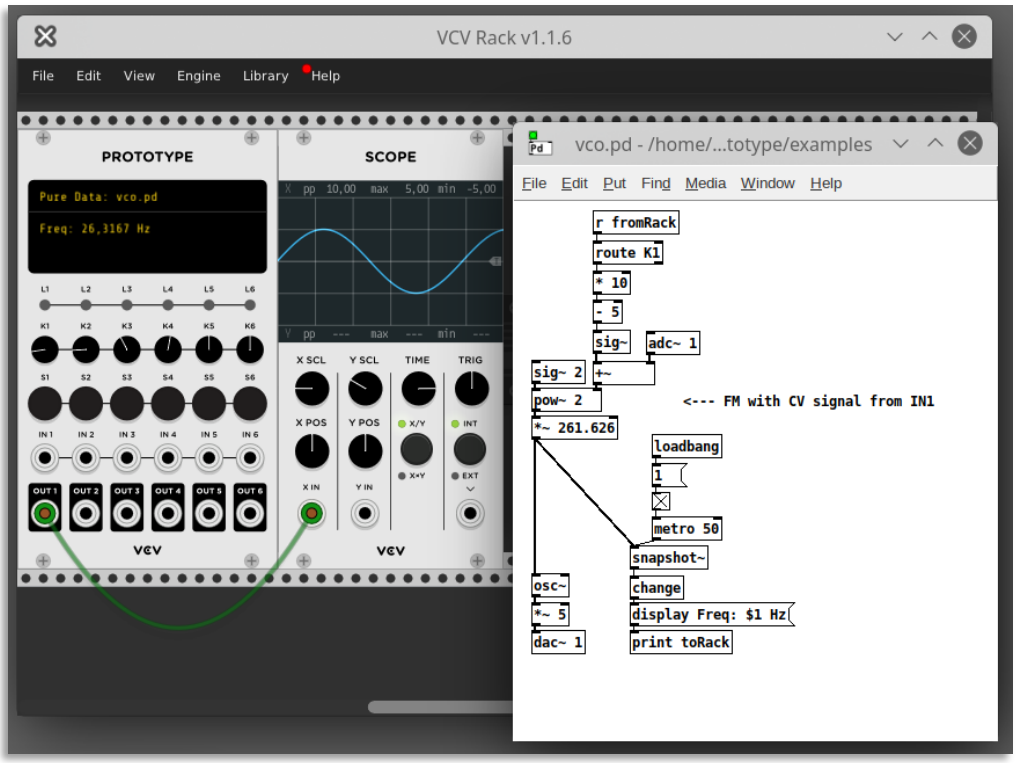
Received on July 1st, 2021. Approved on: November 1st, 2021. Available online on: 30/11/2021. Guest Editor: Dr. Alexandre Torres Porres.

At *The Center for Haptic Audio Interaction Research (CHAIR)* we are developing 'acoustic interfaces' to excite digital resonators (NEUPERT and WEGENER, 2019). The plan emerged to reduce time-to-market by publishing software before our hardware product reached maturity because hardware development, production, and sales turned out to be much more challenging compared to software development. We looked into different cross-platform audio plugin environments in which we could package our Pure Data patches with libpd (BRINKMANN et al., 2011, 2016) while adding our own graphical user interface. With Camomile (GUILLOT, 2018) there is a solution for VST plugins, but the GUI is limited to a subset of recreated Pd-GUI elements which are automatically generated from the patch by the Camomile framework. Camomile is using the industry dominating JUCE framework. JUCE was acquired by ROLI in 2014 and sold off to PACE in 2020. The JUCE framework is open source and dual-licensed, but we were worried about the stability of the parent companies and their corporate interests. The dual license choice to either commit to a costly plan or implement a mandatory splash-screen were options we considered both to be unappealing. Hence, when we found out about the liberally-licensed iPlug2, we were intrigued. Since there is an existent method to use the FAUST language for writing DSP code in iPlug2 (LARKIN, 2018), extending it to libpd seemed like a logical step for us. IPlug2 is feature-rich; it can export to many plugin formats and allows for vector graphics interfaces. We went ahead and added libpd support to it.

1. Intermezzo: VCV Rack Prototype

At the time when we were integrating libpd into the iPlug2 framework, Andrew Belt, the author of 'Rack', was looking for someone to integrate Pd into a special prototyping module for his modular 'virtual control voltage' software synthesizer. We immediately offered our help and the VCV Rack community raised funds allowing us to focus on this task. Our libpd integration for the Prototype module is open source and available at https://github.com/VCVRack/VCV-Prototype. In order to compile libpd into the prototype module we added a static target to libpd which is now up-streamed to the main branch of libpd.

FIGURE 1 – Pure Data inside VCV Rack Prototype Module.[1]

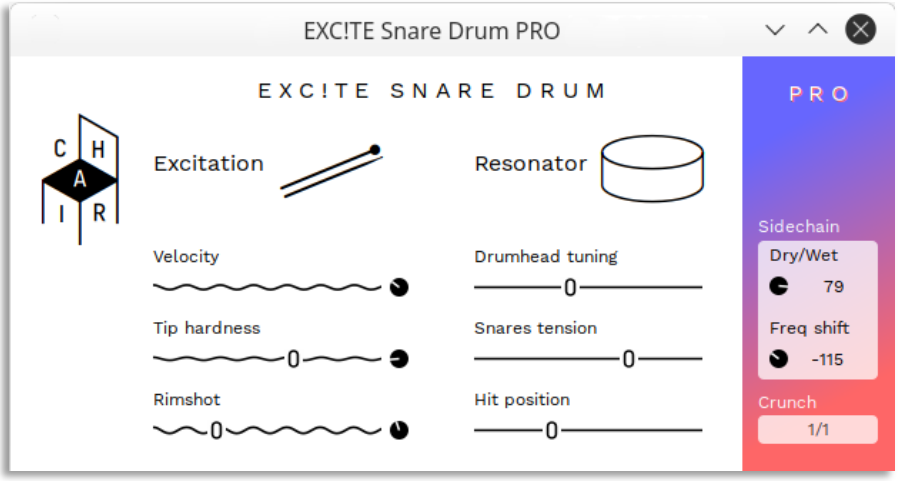


2. Going VST3SDK

After having invested a fair bit of development into iPlug2, we, unfortunately, had to abandon it because our development platform is Linux and the Linux support of iPlug2 sadly is in a work-in-progress state. Instead, we started using the Steinberg VST3SDK directly. However, it is inferior to iPlug2 in many ways: it does not directly support vector graphics and the only output format is VST3 (and .AU and .AAX with a wrapper layer). Its Linux support is somewhat incomplete (no hover labels, for example), but at least it works well enough to allow for development. Steinberg Media Technologies GmbH (a subsidiary of Yamaha Corporation, based in Hamburg, Germany) assigned no dedicated manpower to maintain the VST3SDK. According to them, work on it is mostly done by their staff in their free time. Steinberg presumably open sourced it in the hope that volunteers would take care of its further development. There is a moderate level of activity on both the Steinberg developer forum and their github issue tracker, although it is not as active of a community as compared to the JUCE framework.

[1] All images were provided by the authors.

FIGURE 2 – Interface of the VST3 plugin.



3. The Plugin

'EXC!TE Snare Drum' is a physical modeling snare drum synthesizer. It features an in-built exciter which is triggered by MIDI note input and feeds into a wave-guide resonator. The resonator algorithm may be summarized as four cascaded delay lines and rotation matrices nested inside a fifth delay line with an inverted signal (SCHMALFUSS et al. 2020). The Pro version of the plugin adds side-chain audio input which feeds directly into the resonator, bypassing the exciter. The side-chain signal can be shifted in frequency so that feedback is suppressed and the harmonic spectrum can be manipulated. Finally, the Pro version includes a 'crunch'-parameter which in fact is running the patch at quarter, half, normal, and double sampling-rate relative to 48 kHz. Lower sampling-rates give a dirtier, lo-fi snares sound and are lighter on the CPU. A higher sampling-rate offers a more detailed sound at a slightly higher CPU load. Our plugin is available as a VST3 instrument for Linux, Windows, and on MacOS additionally in AudioUnit format. The Pro version is a 20€ download. There is no activation, no copy protection, and no telemetry.

4. Limits of Pure Data's 4-point interpolation

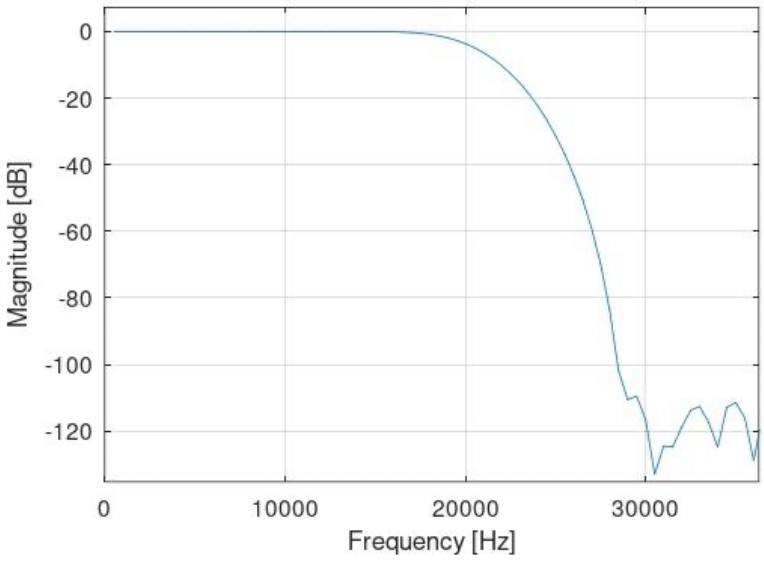
While testing the plugin in different plugin-hosts, on different computers and operating systems at different sampling-rates we discovered a weird bug: The sound changed drastically with

the identical settings with a mere sample-rate change from 44.1 to 48 kHz. On further investigation it turned out that the sound changed depending on if the delay time referred to a fractional or non-fractional number of samples. Eventually we could pinpoint the issue on the delread4~ object, precisely on the four-point interpolation implemented in the object. In regular use this interpolation is fine. It is a computationally efficient solution. However, in some cases, like in our wave-guide or super slow sample playback, artefacts from the interpolation will become audible. Users of Pd have complained several times about aliasing problems in tabread4~ on the Pd mailing list. Cyrille Henry had proposed back in 2008[2] to use a Whittaker-Shannon interpolation (Sinc function) instead.

5. Interpolation with the Sinc function

This is our contribution: An alternative to delread4~. We called it delreadsinc~. It uses an alternative fractional interpolation method documented in SMITH 2020.

FIGURE 3 – Anti-aliasing and filter curve: The implementation uses 23 points of the Sinc function with a Blackman-Harris window applied. It shows good anti-aliasing characteristics as shown in the graph for 48kHz sampling rate. The transition band ends at 28kHz with 87dB of rejection. Because aliasing is symmetric w.r.t. 24kHz this will "back-alias" on 20kHz with 87dB of rejection.



[2] https://lists.puredata.info/pipermail/pd-list/2008-06/063221.html

It is computationally more expensive than Miller's 4-point interpolation, but it allows for very slow sample playback with little artifacts and will not noticeably change the timbre whether the delay time translates to a fractional or non-fractional number of samples inside the delay-line. Alternatively, the issue could be circumvented by upsampling before the interpolation, but this is computationally more expensive than using the Sinc function.

TABLE 1 – CPU load of delreadsinc~ compared to an upsampled delread4~

Upsampling	**delread4~**	**delreadsinc~**
none	12% (sounds distorted)	**15%**
2 x	21% (still not good)	(not needed, measured 26%)
4 x	**33%**	(not needed, measured 39%)

To make delread4~ sound as smooth as delreadsinc~ it must be upsampled four-fold. Table 1 shows the CPU-load of a patch containing five delreads running in Pd with 48 kHz. Substituting delreadsinc~ with a four-fold uplsampled delread4~ costs more than double the CPU load.[3] In conclusion, we can say that delreadsinc~ adds an acceptable performance overhead for the additional quality it provides. The delreadsinc~ object reads a delay line from a delwrite~. An alternative delay line implemented as an external would need to duplicate the delwrite~ functions into a new external object. Since we wanted to stay 'vanilla' (no externals) with our Plugin code, we instead made a feature branch in a forked repository. We hope that this code may become part of vanilla Pd at some point, but there are difficult considerations to make: Would it be desirable to switch between different interpolation methods? If so, what's the name of that object? Do we accept code duplication for the interpolation methods in tabread4~/delread4~ and tabreadsinc~/delreadsinc~, or would it be expandable to interpolation plugins? The drive to make this implementation clean, simple, yet expandable is opening a can of worms which remained unanswered and inconclusive in the discussions on the mailing list. For us the work is sufficient with our fork of vanilla Pd which provides delreadsinc~ to libpd which is running inside our VST3.

[3] A discussion of the subject happened on the github issue tracker https://github.com/pure-data/pure-data/issues/1305

6. Closing thoughts

In an industry collaboration job sometime in 2019, we were able to sell a simple MaxMSP standalone executable as if it was a custom-made software. It looked professional and impressed the client with its massive download of 120 MB, thanks to all the bloat which comes with the MaxMSP runtime. The customer was happy — and so were we. This experience left us pondering: Is that how it works? Does software you paid for allow you to profit by giving you the power to produce (export) a 'product', while free software 'just' lets you do free stuff?

No. That's definitely not the case, as the liberally licensed Pure Data allows it to be part of almost anything. Thanks to the work on libpd this has become easier than ever. It is not as streamlined and accessible to *users* (as opposed to *programmers*) as the standalone export in MaxMSP, but it is a start.

Imagine developing something quickly in Pd and then wrap it into something else, like in our case: a VSTi. We actually have released our synthesis patches openly through Deken, but even though the synthesis is open source there are enough people who value a shiny VST with a nice GUI. Thanks to them we can grow towards being sustainable. Sustainability for a community means that the investment of learning a software is rewarded with the ability to turn this skill into something others value. For some that can be a position in education, for others it may be a software product. For such a product to become more than a mere patch it needs to be transformed into a neatly wrapped package. Our package is a plugin, but it could be just as well a (mobile) app (IGLESIA, 2016).

ACKNOWLEDGMENT

We would like to thank Dan Wilcox for mentoring our libpd integration in VCV Prototype, the VCV Rack community for the successful fundraiser, Cyrille Henry for pointing to the Sinc function, Charles Henry for the detailed feedback and structuring advice on our delreadsinc~ implementation, and Brian Clark for copy editing.

REFERENCES

BRINKMANN, Peter et al. Embedding Pure Data with libpd. Proceedings of the 4th Pure Data Convention (Pd~Con), Weimar, 2011.

BRINKMANN, Peter et al. libpd: Past, Present, and Future of Embedding Pure Data. Proceedings of the 5th Pure Data Convention (Pd~Con), New York City, 2016

GUILLOT, Pierre. *Camomile: Creating Audio Plugins with Pure Data.* Proceedings of the Linux Audio Conference (LAC), Berlin, 2018.

IGLESIA, Daniel. *The mobility is the message: The development and uses of MobMuPlat.* Proceedings of the 5th Pure Data Convention (Pd~Con), New York City, 2016

LARKIN, Oliver. *FAUST in iPlug2: Creative Coding Audio Plugins.* Proceedings of the 1st International Faust Conference (IFC-18), Mainz, 2018.

NEUPERT, Max and WEGENER, Clemens. *Interacting with digital resonators by acoustic excitation.* Proceedings of the 16th Sound & Music Computing Conference (SMC), Malaga, 2019.

SCHMALFUSS, Philipp et al. *Efficient Snare Drum Model for Acoustic Interfaces with Piezoelectric Sensors.* Proceedings of the 23rd International Conference on Digital Audio Effects (DAFx-20), Vienna, 2020.

SMITH, Julius O. *Digital Audio Resampling Home Page "Implementation" section*, http://www-ccrma.stanford.edu/~jos/resample/Theory_Operation.html, 2020.

ABOUT THE AUTHORS

The Center for Haptic Audio Interaction Research is dedicated to bring natural interaction to synthesis by enabling acoustic excitation of virtual resonators. Current research and development activities include the Tickle tactile instrument and piezoelectric strings.

Max Neupert is media artist (PhD) and industrial designer. Neupert is teaching / researching at the Media Environments chair at Bauhaus-Universität Weimar. ORCID: https://orcid.org/0000-0001-7581-3450. E-mail: max@chair.audio

Clemens Wegener is musicologist (BA) and computer scientist (MA). Wegener is part of the Interface Design Group at Bauhaus-Universität Weimar. ORCID: https://orcid.org/0000-0001-6110-5488. E-mail: clemens@chair.audio

Philipp Schmalfuß is composer and electroacoustic musician. He is the author of the audiolab abstraction library for Pd. ORCID: https://orcid.org/0000-0002-9406-734X. E-mail: philipp@chair.audio

Sebastian Stang is an IT specialist with a wide range of experience from DSP programming, embedded systems, Cmake work-flows to online systems. E-mail: sebastian@chair.audio