UC | Chile

International Computer Music Conference 2021 Santiago, Chile

Proceedings of the International Computer Music Conference 2021

The virtuoso computer: redefining limits

Hosted by:

Pontificia Universidad Católica de Chile
Santiago, Chile, July 25th–31st

Proceedings of the International Computer Music Conference 2021

Pontificia Universidad Católica de Chile | July 25th-31st, 2021, Santiago, Chile

The virtuoso computer: redefining limits

Rodrigo F. Cádiz, editor

ISBN: 978-0-9845274-9-6

Published by

International Computer Music Association, Inc.

1819 Polk St., Suite 330

San Francisco, CA 94109

USA

icma@umich.edu

http://www.computermusic.org

UC | Chile

International Computer Music Conference 2021 Santiago, Chile

Pi-Shaker: A New Workflow for Augmented Instruments

Vesa Norilo
University of the Arts
Helsinki, Finland
vno11100@uniarts.fi

Andrew R. Brown
Griffith University
Brisbane, Australia
andrew.r.brown@griffith.edu.au

ABSTRACT

We present a project that explores the application of efficient digital signal processing techniques for interactive music applications across a range of devices and platforms, focusing on visual programming. As a test case, we implement physically informed models of sound synthesis and sound spatialisation that can respond in real time to performative gestures. We compare the strengths and weakness of implementations in several languages and how they can be integrated to best take advantage of these differences.

1. INTRODUCTION

With the increasing power of small scale microprocessors, the types of real time digital signal processing tasks typical of desktop computers are increasingly possible with handheld devices. One of the remaining challenges is to port the power of desktop class signal processing tool chains to these devices. In this paper we describe a project to apply the power of the Kronos signal processing language to drive a gesturally responsive handheld digital instrument.

Hand held interactive digital instruments, like the eShaker [1], demonstrate the possibility for stand-alone devices to perform similarly to the traditional combination of gestural controllers attached to desktop computers, such as the bEADS shaker [2] and the T-Stick [3].

For this project we developed a prototype handheld digital instrument, the Pi-Shaker, using the small and inexpensive Raspberry Pi Zero computer. An image of the Pi-Shaker prototype is shown in Figure 1.

For audio signal processing we implemented versions of the well-established PhISEM percussion model [4] in Pure Data (Pd) [5] and developed workflows for compiling Pd extensions using the visual Veneer environment and Kronos langauge for synthesis and reverb.

2. BACKGROUND

The porting of digital audio languages to mobile and small scale devices has developed in accord with the growing popularity and power of these devices for both DIY electronics [6] [7] and commercial mobile phones [8] [9] [10].

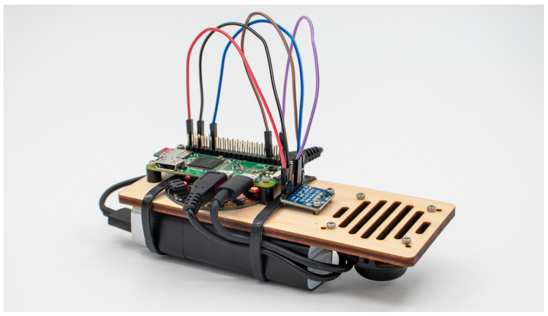


Figure 1. Pi-Shaker prototype

Real time synthesis based on physical models is ideal for mapping gestural and spatial strategies to DSP responsiveness, yet can be somewhat demanding computationally. Our criteria for evaluating the Pi-Shaker included three interaction gestures corresponding to Schaeffer/Chion's categories of sounds: Impulse (hit), Iterative (shake) and Sustained (tilt) [11]. We also drew upon the more extensive framework for gestural mapping to physical modelled sound outlined in the PHYSMISM project [12]. Because we are using low cost devices, our evaluations also explored the differing computing performance and algorithmic affordances of implementations using native Pd and Veneer-generated externals.

2.1 Kronos, the DSP Compiler

Kronos is a metaprogrammable static circuit compiler targeted at musical signal processing [13]. It generates efficient mixed-rate circuits based on the paradigm of discrete reactive systems [14]. The language is designed to constrain user programs to static data flow, which enables the compiler to perform significant analysis and optimization. Build artifacts do not depend on any runtime libraries, apart from optionally using the C runtime for mathematical functions. As such, it is inherently suitable for targeting embedded systems.

The Kronos compiler is based on LLVM [15], a middle-to backend code generation framework that is capable of cross-compilation. This means we can run the compilation process on a powerful computer, and deploy the standalone binaries to a more resource-constrained device such as the Raspberry Pi Zero.

2.2 Veneer, the Web Patcher

Veneer [16] is a web application that provides a patching interface to the Kronos programming language. It can inte-

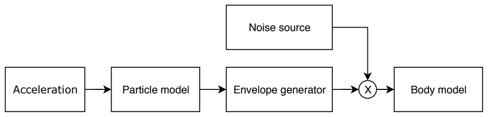


Figure 2. Instrument model overview

grate in real time with a native execution engine, or use an embedded execution engine based on WebAssembly and the just-in-time compiler in the browser itself.

Veneer is designed for interactive programming. Any changes to the user program are compiled in real time. Further, programs can have elements the user can directly manipulate, such as sliders, dials, pads and number boxes.

2.3 Producing Native Binaries from Web Applications

Notably, the execution engine embedded in Veneer [16] does not contain the full LLVM [15] code generator, but uses Binaryen, a lighter-weight WebAssembly code generator. Integrating a full cross-compiler to a web application seems impractical as of this writing due to the size of the LLVM codebase.

An interesting solution to this problem is remote compilation. A prior example is the Faust Online Compiler [17], which combines an integrated development environment (IDE) built as a web application, with a server-based compiler. The user may write a program in the browser, submit it for compilation over the network, and finally download the finished build artifacts.

3. METHODOLOGY

3.1 Algorithm

Our instrument model is based by the well-known physically informed method for synthesizing percussion instruments such as shakers, maracas and tambourines [4].

Based on the incoming excitation, the system generates a random impulse train that simulates particle collisions. Each collision is given an exponentially decaying envelope.

The sum of all the envelopes is multiplied by a noise source for phase randomization, and the enveloped noise is processed with a body model, such as a resonator bank, to provide the timbral characteristics for a range of percussion instruments. An overview of the algorithm is shown in Figure 2.

As shown in previous research [4], an infinite number of exponentially decaying overlapping envelopes can be generated by convolution of the impulse train, efficiently implemented as a simple one-pole filter.

3.2 Hardware implementation

The hardware elements used to build the Pi-Shaker include the Raspberry Pi Zero W computer, audio output was sent via the i2s protocol to an Adafruit MAX08357 Amp Breakout board, then to a 40mm speaker. Gestural data was captured by the accelerometer on an Adafuit Playground Express board and sent via MIDI over serial to the Raspberry Pi. All components were powered by a rechargeable battery over USB. The Raspeberry Pi ran Pd patches using

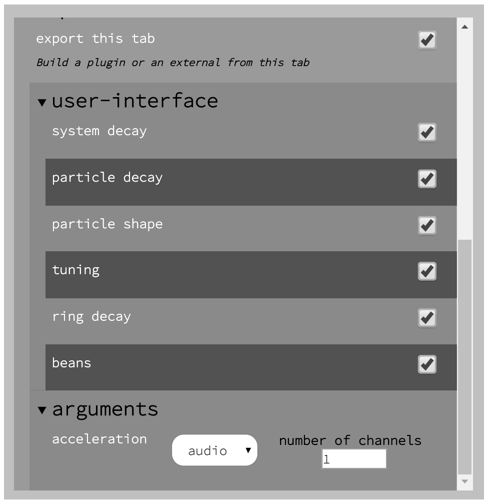


Figure 3. Veneer function export settings

externals created as described in Section 3.3. The Circuit Playground was programmed in the Arduino IDE to map the X, Y and Z accelerometer outputs into three MIDI control change messages sent to the Raspberry Pi.

3.3 Compiler as a Web Service

To facilitate deployment of Veneer patches on a variety of platforms, we developed an online compilation service accessible via a central web server. The server contains a job queue and an artifact repository, based on CouchDB [18]. Build agents claim jobs from the queue and deploy artifacts, which become available for users to download.

A build agent consists of a simple Python script, CMake-driven build process, the Kronos static compiler, development kits related to the platform being built for, and glue code written in both kronoslang and C++ to provide the mapping between the discrete reactive signal model in Kronos and the host paradigm.

3.3.1 Kronos–Pd interface

In the case of Pure Data [19], we must resolve the mapping from Kronos clock domains to either messages or signals. In addition, signals can have many data types per frame in Kronos, while Pd maintains a one-to-one correspondence between 32-bit floating point channels and patch cords.

Because Kronos programs are generic, some user assistance is required. For a function to be exported from Veneer to Pd, the user must specify the basic mapping for each argument. Currently, the export interface supports signal inputs with an arbitrary number of channels, control parameters with default values, higher and lower bounds, and constants. Each signal input is represented by one or more Pd signal inlets, while each control parameter generates a message inlet.

Separately, each interactive widget, such as a slider or a knob, can be exported. Veneer looks for any named widgets and provides an option to export them. In Pd, widgets become message inlets, and their current value is the default setting. The export definition interface is demonstrated in Figure 3.

Table 1. Cross-compilation environment

config	key	value
env	`RASPBIAN_ROOTFS`	*path to Raspbian file system*
agent	`toolchain`	*path to toolchain file*
kc	`--mcpu`	`arm1176jzf-s`
kc	`--mtriple`	`arm-linux-gnueabihf`

The Kronos-side glue code written for the build agent splits multi-channel inputs and outputs into multiple monophonic channels to facilitate the Pure Data model. Each control parameter becomes a message inlet, with the arrival of messages driving reactive computation. Control rate clocks are derived by subdividing the main audio clock.

3.3.2 Cross-compilation for the Raspberry Pi

We provide the Raspberry Pi builds by cross-compiling from Ubuntu Linux. As a basis, the toolchain prepared by Stefan Profanter [1] was used to cross-compile the C++ part of each external. The system libraries and headers were obtained from the Raspbian Buster distribution and its Pure Data package. The Kronos code is compiled by the Kronos static compiler `kc`, which is inherently a cross-compiler on Ubuntu Linux, due to the system package for LLVM [15] supporting the Arm processors found in Raspberry Pi.

Cross-compilation settings are summarized in Table 1. The build agent script must be configured to cross-compile with a combination of environment variables and a CMake toolchain file, while `kc` requires the appropriate target triple and cpu architecture setting. The target triple setting ensures that the Kronos compiler respects the hardware floating point calling convention that is the system default on the Raspberry Pi. Further, LLVM would produce software-emulated floating point code by default, which runs too slowly to be useful. The correct processor setting, targeting the Raspberry Pi Zero and above, allows our external to use the hardware vector and floating point unit, improving performance by an order of magnitude.

The resulting cross-compiling agent was captured in a Docker [20] image for the ease of redeployment.

4. CASE STUDIES

We tested the synthesizer with various known configurations from literature [4]. A tambourine model Pd patch is shown in Figure 4 and a Veneer patch in Figure 5. These are the basis for our performance comparisons detailed in section 5.

4.1 Tambourine

The Pd patch models the PhISEM algorithm outlined by Cook [4]. In the Veneer patch, the nodes that participate in impulse train generation are in orange, while the envelope generators and noise modulation are colored pink. The blue nodes compute gain compensation based on the probability of collisions.

In Veneer the impulse train can be generated efficiently by comparing the output of a random generator (`Drift`) to a threshold value, and masking the overall system energy envelope with the result of that comparison (logical

[1] `https://github.com/Pro/raspi-toolchain`

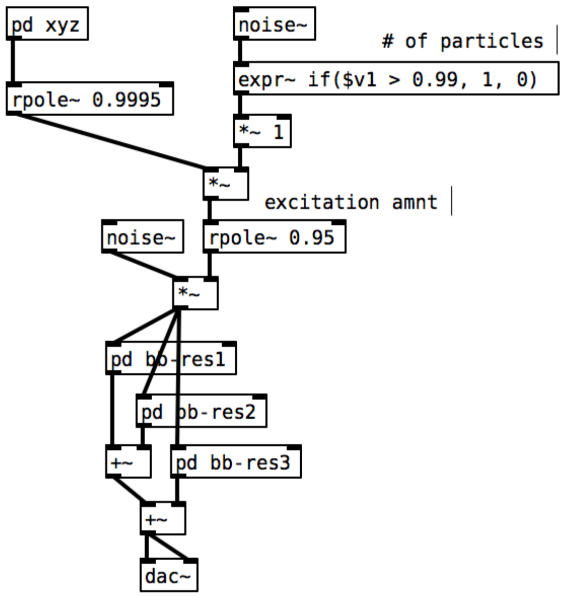


Figure 4. Particle percussion synthesis in Pure Data

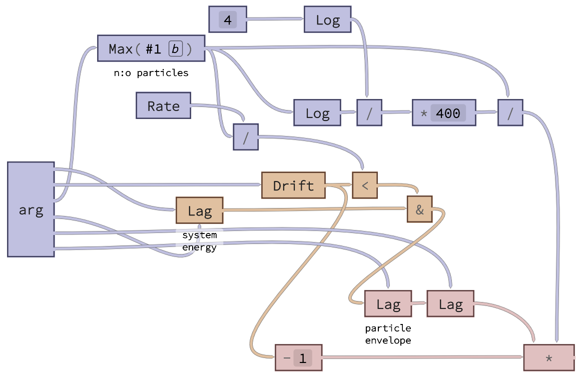


Figure 5. Particle percussion synthesis in Veneer

and, `&`). The `Lag` nodes represent one-pole filters, parametrized by the length of time between onset and 90% loss in the exponential decay.

In this implementation, a resonator bank is derived from a parameter matrix by a higher lever function, `Map`.

The audio signal from the particle model is captured by the mapping function, as shown by the connection from `PHISEM` to `Resonator`, in a visual analogy to a closure. This creates a fanout filter bank structure, subsequently mixed in the `Sum` node. Notably, the `seed` parameter determines the pseudo-random sequence, and therefore the timing of any collision events relative to the start of the synthesis run.

By hand-tweaking parameter values we were able to achieve clear distinctions between three gestures types — hit, shake and tilt — that can be the basis for interactions based on percussion instruments such as hand-clap, sleigh bells, and rain stick. Audio examples from these case studies are

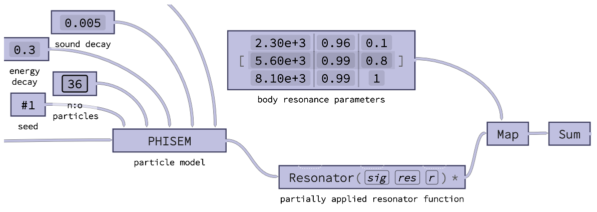
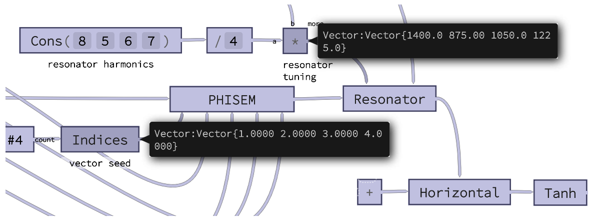


Figure 6. Tambourine settings and body model



```
Vector:Vector{1400.0 875.00 1050.0 122
5.0}
```

```
Vector:Vector{1.0000 2.0000 3.0000 4.0
000}
```

Figure 7. Excerpt from the "Mbiracas", a vectored particle synthesizer

available online [2].

4.2 "Mbiracas"

As a new development, we wanted to explore the design space of augmented particle-collision instruments afforded by Kronoslang. *Mbiracas* is the first result; it features four distinct particle models and resonators, mixed into a nonlinear `Tanh` waveshaper.

We can utilize the Kronos generic programming facilities to accomplish such a setup easily; by seeding the particle model with a *vector* rather than scalar, higher-dimensional circuits are generated by the compiler, and we obtain four distinct particle streams.

Automatic type derivation causes the resonator signal path to upgrade to four channels as well; we can define distinct resonators for each channel by supplying vectored parameters. An excerpt of the Veneer patch is shown in Figure 7. We use a short index vector as the pseudo-random seed and a harmonic series as the resonator frequency vector. The patch is shown while interrogating these vector values.

4.3 Reverberation

Previously we showed how to vectorize an existing algorithm to explore further sonic possibilities. This approach is particularly well suited to reverberation; the feedback-delay network (FDN) [21], an excellent method for synthetic reverberation, is essentially a vectored comb filter.

We developed a FDN reverberator to exercise the capabilities of Veneer and Kronos in programmatic circuit generation as well as test the performance of the generated code on the Raspberry Pi.

We chose an 8-channel reverberator with additional diffusion provided by allpass filters in series with each delay line. The allpass filters provide a convenient means of controlling the reflection density of the algorithm without adjusting the delay line lengths.

[2] http://explodingart.com/icmc2020/audio_examples/

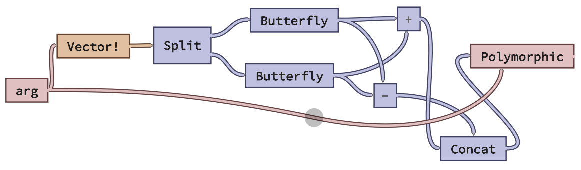


Figure 8. "Butterfly", the recursive FDN matrix implementation in Veneer

Our implementation uses a Hadamard feedback matrix. This type of matrix yields a rapidly increasing reflection as well as symmetry properties useful for optimization, known as the Fast Walsh-Hadamard transform (FWT), reducing the computational complexity from $O(n^2)$ to $O(nlogn)$, where n is the square of vector width. Figure 8 shows the implementation of FWT in Veneer. The patch is polymorphic and recursive; if the argument is a vector, it is recursively split in half until the terminating scalar case is reached. The transform is completed by concatenating the sum and difference of each recursive sub-transform, yielding the characteristic Butterfly-shaped circuit.

5. RESULTS AND DISCUSSION

We compared the tambourine (shaker) models for both Pd and Kronos implementations on a laptop computer, Raspberry Pi 3, and the Raspberry Pi Zero W. Results are summarised in Figure 9. The numbers on the bars indicate how many instances of each instrument could theoretically run in real time on each device. The heights of the bars are normalized per device to help observe the relative performance of each patch on devices of different CPU class.

We ran several instances in parallel to obtain more robust CPU use numbers; 20 in case of the laptop computer, 4 for Raspberry Pi3 and 3 for the Raspberry Pi Zero. An empty Pd patch was also measured as a baseline. We computed the theoretical CPU load attributed to just our code by subtracting the baseline. The displayed number is the theoretical maximum number of real-time instances each device could support assuming linear scaling and perfectly efficient 100% CPU load. This is merely to contextualize the numbers as in practice neither linear scaling or robust operation under full CPU load is realistic.

The patch, Pd external and script files used for benchmarking are available online [3].

5.1 Benchmarks

On a Macbook laptop with an Intel i5 processor, our Kronos implementation is ca. 3 times more efficient than our native Pd patch. On the Rasperry Pi family, it is twice as efficient. We believe the difference is due to the fact that i5 can extract more instruction level parallelism from statically compiled code subjected to interprocedural optimisation, and the fact that processors with shallow pipelines, such as the Arm chips, suffer less from the unpredictable dynamic dispatch of the native Pd patch.

[3] https://github.com/kronoslang/icmc2020_benchmarks

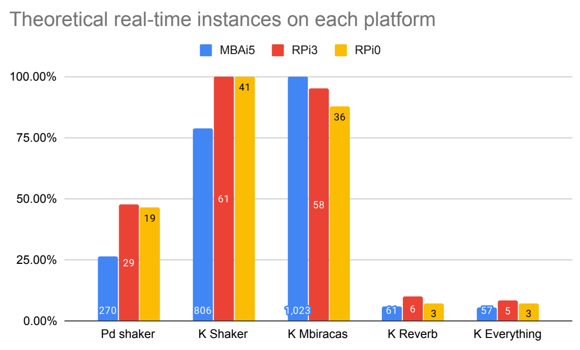


Figure 9. Summary of benchmark results

On the Mac, the vectored mbiracas runs slightly faster than the shaker. On Raspberry Pi devices, the shaker is slightly faster. The shaker has three resonators in series compared to four parallel resonators on the mbiracas; Kronos generates parallel vector code for mbiracas, and on the modern i5 processor vector math runs fast enough for four parallel filters to handily outperform three serial ones. The Raspberry Pi has a weaker vector unit: parallel filters still provide a boost, but to a lesser degree.

Understandably the reverb demands a much heavier load than the particle models, but it is encouraging that this vector-based algorithm with 8 channels of diffuse reverberation can be implemented at all via the Pd externals. This opens up new signal processing options to environments where Kronos/Veneer do not run natively.

5.2 Comparison of Relevant Language Features

In the traditional division of musical programming tasks into ugen, instrument and score [22], Kronos addresses the former two, while Pure Data excels at the latter two. As the present study deals with fairly low-level algorihmic details, we would expect Kronos to have some advantages.

Notably, we were able to share an identical particle model implementation between the scalar and vector cases. In most cases, Pure data would require the programmer to explicitly duplicate subpatches to achieve vectored processing. This useful abstraction in Kronos came with no loss of efficiency: the generated code made full use of the hardware vector units.

In addition, the metaprogramming capabilities of Kronos were useful in implementing the reverberation algorithm, especially the fast Walsh-Hadamard transform (see Section 4.3). A similar implementation in Pure data would have required a lot of manually specified nodes and connections, absent a metaprogramming or scripting facility outside of the core language.

On the other hand, the reach and expandability of Pure data enabled us to easily deploy our implementation on a variety of platforms without dealing with the intricacies of hardware.

5.3 Future work

Whilst the development of the Pi-Shaker and Kronos-Pd workflow open up new opportunities for gesturally controlled hand held instruments, clearly more rigorous testing and exploration of the possibilities opened up in this project are required and we look forward to undertaking that work. The Veneer compiling pipeline could also be extended to other platforms such as to Arduino-class micro controllers, especially those using the Arm M4 or higher processors with dedicated floating point hardware.

6. CONCLUSIONS

In the paper we have presented the Pi-Shaker project that explores the application of efficient digital signal processing techniques for interactive music applications. Notably, Kronos and Veneer were successfully used to extend the signal processing capabilities of Pure Data without abandoning the visual patching metaphor. We outlined implementations of physically informed models of sound synthesis and sound spatialisation using a novel workflow featuring Kronos and Pure Data. These were shown to be efficient and respond effectively in real time to a range of performative gestures. As such, we demonstrate the potential of this platform as a basis for future development of handheld digital instruments.

Acknowledgments

Vesa Norilo's work was supported by the Academy of Finland, award number SA311535.

7. REFERENCES

[1] A. Piepenbrink, "Embedded Digital Shakers: Handheld Physical Modeling Synthesizers," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, T. M. Luke Dahl, Douglas Bowman, Ed. Blacksburg, Virginia, USA: Virginia Tech, Jun. 2018, pp. 362–363. [Online]. Available: http://www.nime.org/proceedings/2018/nime2018_paper0080.pdf

[2] P. L. Williams and D. Overholt, "bEADS: Extended Actuated Digital Shaker," in *Proceedings of New Interfaces for Musical Expression*. Copenhagen: NIME, 2017, pp. 13–18.

[3] J. Malloch and M. M. Wanderley, "The T-Stick: From musical interface to musical instrument," in *Proceedings of the 7th international conference on New interfaces for musical expression*. ACM, 2007, pp. 66–70.

[4] P. Cook, "Physically informed sonic modeling (phism): Synthesis of percussive sounds," *Computer Music Journal*, pp. 38–49, 1997.

[5] M. Puckette, "Pure Data: Another integrated computer music environment," in *Second Intercollege Computer Music Concerts*. Tachikawa: Kunitachi College of Music, 1996, pp. 37–41. [Online]. Available: http://www-crca.ucsd.edu/~msp/software.html

[6] S. Dimitrov and S. Serafin, "Audio Arduino – an ALSA (Advanced Linux Sound Architecture) Audio Driver for FTDI-based Arduinos," in *Proceedings of the International Conference on New Interfaces*

for Musical Expression, Oslo, Norway, 2011, pp. 211–216. [Online]. Available: http://www.nime.org/proceedings/2011/nime2011_211.pdf

[7] R. Michon, Y. Orlarey, S. Letz, and D. Fober, "Real Time Audio Digital Signal Processing With Faust and the Teensy," in *Proceedings of Sound, Music, Computing*, 2019, p. 7.

[8] A. Tanaka, "Mobile music making," in *Proceedings of the 2004 Conference on New Interfaces for Musical Expression*, ser. NIME '04. Singapore, Singapore: Singapore: National University of Singapore, 2004, pp. 154–156. [Online]. Available: http://dl.acm.org/citation.cfm?id=1085884.1085918

[9] C. Heinrichs and A. McPherson, "Performance-Led Design of Computationally Generated Audio for Interactive Applications." in *Proceedings of the Tenth International Conference on Tangible, Embedded, and Embodied Interaction*. Eindhoven, Netherlands: ACM, 2016, pp. 697–700.

[10] D. Iglesia, "The Mobility is the Message: the Development and Uses of MobMuPlat." in *Proceedings of the 5th International Pure Data Convention*. New York: NYU music, 2016, p. 6.

[11] M. Chion, *Guide to sound objects: Pierre Schaeffer and musical research*. Paris: Institut National de L'audiovisuel, 2009. [Online]. Available: http://www.ears.dmu.ac.uk

[12] S. Gelineck and S. Serafin, "A Practical Approach towards an Exploratory Framework for Physical Modeling," *Computer Music Journal*, vol. 34, no. 2, pp. 51–65, 2010. [Online]. Available: http://www.mitpressjournals.org/doi/10.1162/comj.2010.34.2.51

[13] Norilo, Vesa, "Kronos: A Declarative Metaprogramming Language for Digital Signal Processing," *Computer Music Journal*, vol. 39, no. 4, pp. 30–48, 2015.

[14] P. Van Roy, "Programming Paradigms for Dummies: What Every Programmer Should Know," in *New Computational Paradigms for Music*, G. Assayag and A. Gerzso, Eds. Paris: Delatour France, IRCAM, 2009, pp. 9–49.

[15] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," *International Symposium on Code Generation and Optimization 2004 CGO 2004*, vol. 57, no. c, pp. 75–86, 2004.

[16] Norilo, Vesa Petri, "Veneer: Visual and Touch-based Programming for Audio," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, M. Queiroz and A. X. Sedó, Eds. Porto Alegre, Brazil: UFRGS, 2019, pp. 319–324.

[17] R. Michon and Y. Orlarey, "The Faust online compiler: a web-based IDE for the Faust programming language," in *Proceedings of the 10th Linux Audio Conference (LAC-12)*. Stanford University, 2012.

[18] J. Anderson, J. Lehnardt, and N. Slater, *CouchDB: the definitive guide: time to relax*. O'Reilly Media, Inc., 2010.

[19] M. Puckette, "Pure data: another integrated computer music environment," in *Proceedings of the 1996 International Computer Music Conference*, 1996, pp. 269–272.

[20] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, 2014.

[21] J. Jot and A. Chaigne, "Digital Delay Networks for Designing Artificial Reverberators," in *the 90th AES Convention*, vol. 3030, 1991, p. preprint no. 3030.

[22] C. Roads, *The Computer Music Tutorial*. Cambridge: MIT Press, 1996.