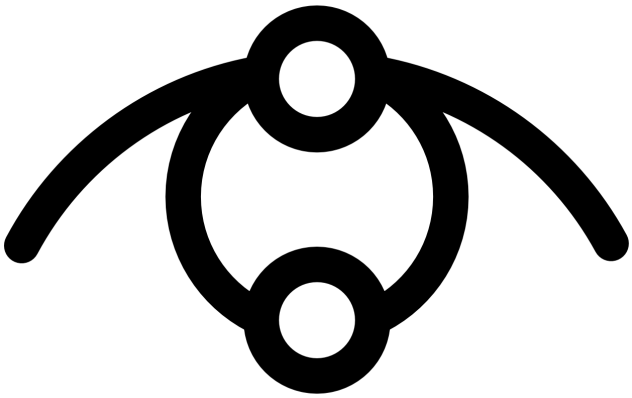
# The Resonarium Digital Waveguide Synthesizer

Gabriel Soule (gabesoule@gmail.com)

January 2025



*This is a living document, and remains a work in progress. It is has not yet been subjected to extensive review. As such, feedback, comments, and corrections are greatly appreciated.*

# Contents

# 0 A Preamble

*I love the guitar.*

*How can I qualify or justify this statement? The question is rhetorical. Some things simply are as they are, and I'm pretty sure that this one is as it is. We rarely choose our strange attractors.*

*I didn't learn the guitar as a young child, unfortunately. Musical education was inaccessible for me during that time. I finally found the resources to study music—time, money, and agency—later, as I wrapped up high school and entered university. What I wasn't able to find was the space and stability: I spent much of my time shortly after university without a stable home or studio, and I quickly learned that an itinerant lifestyle induces immediate problems with respect to regular musical practice.*

*Indeed, the notion of building a music studio when you spend most of your time living out of a backpack is a bit of a head-scratcher. You can't fit a studio into a backpack. You can't even fit a guitar into a backpack. You could maybe fit one into two separate backpacks, but you're not sure what the guitar might have to say about that.*

*What I could fit into my backpack at the time was a laptop, and according to a bunch of very clever guitar-havers in academia, physical guitar modeling is a relatively mature and well-studied art. And thus a strange seed germinated in my mind. Learning how to model a guitar (and then implementing it) proved to be, via a retrospectively deranged set of logical axioms I had fastened myself to, concretely more efficient than acquiring a guitar and a comprehensive set of guitar paraphernalia—insofar, at least, as the required number of backpacks was concerned, which, vis-à-vis those same principles, was the only metric worth optimizing at the time. I was intimately familiar with what happens when you give a mouse a cookie: he's probably going to want thousands of dollars worth of guitar pedals go with it. And so I removed my laptop from my singular backpack and got to work.*

*Often I'm not sure whether all this indirection makes me a terrible musician or a rather clever musician. I neatly and professionally resolve the problem by not thinking about it as much as possible.*

*Fortunately, things didn't go exactly as planned. Do they ever? The object of my endeavor, as presented in this paper, is not exactly what I intended to build. I readily confess with both pride and disappointment that it is not a dedicated and purpose-built guitar replacement. Not yet, at least. Like all sufficiently complex systems, it developed a life of its own as I built it, and tiny impulses to its evolutionary velocity compounded over time to induce a trajectory that I did not forsee. There are some things it does well that I expected it to do well. There*

*are some things it does poorly that I had hoped it would do well. And, of course, there are some things it does well, of which I did not—could not—imagine it would be capable. It is these unexpected facets of expression that delight me the most, and I try and lean into them when possible. Logic and heuristics dictate that interesting things in the world are those which I do not know that I do not know, and that is a vast ocean indeed.*

*So it goes with art. An artist is struck by an embryonic* idea*, and to reify her brainchild, she chooses a* medium *and channels into it her force of will. She (perhaps quite literally) chips away at the medium with refined confidence, projecting her vision upon and into it, and this all goes smoothly for a time—until eventually, inexorably, the balance of power suddenly shifts a little towards stable equilibrium. Maybe a tool slips. The medium is inhabited by an agency of its own, and it starts to talk to the artist:* hey, I see what you're doing, but that isn't going to work out like you thought... let's do it this way instead*. And from what was once a monophonic imposition of a artist's vision unto a passive subject, there now evolves a sort of dialogue between equals, a Newtonian ballet in which each impulse exerted upon the medium by the artist is reciprocated, in some way, by an impulsion upon the artist from the medium. What emerges from this process of co-creation is a mutualistic compromise that neither party predicted with perfect clarity, and yet that entropic uncertainty is what makes it good art. It's the happy little accidents that go on to change the world.*

*Maybe it's different for you, but least that's how it is for me. That's okay. That's what makes it fun. If I knew what I was doing I probably wouldn't be able to do it.*

*It still all fits in a single backpack. And most importantly, now that this is all said and done, I can finally study guitar now.*

# 1 Introduction

Physical modeling is a form of sound synthesis in which the vibration of physical objects is modeled via computer simulation [42],[43]. By unifying physics with the techniques of digital signal processing, one can synthesize timbres that faithfully emulate the modeled objects or instruments to a remarkable degree of fidelity.

Physical modeling is a well-refined art with a legacy of research that extends as far back to the 1970s. The physics that underpin such techniques, such as the properties of vibrating strings and bars, have been studied since antiquity. In the 2020s, physical modeling is a refined art, and many extant physical models are nearly indistinguishable from their physical referents.

However, the technique remains relatively niche within the greater community of composers, producers, and sound designers. While there exist commercially available collections of expressive physically modelled instruments [25,

13, 50], we believe that the technique remains relatively unexplored within the broader musical community, outside of specialist research groups.

Furthermore, we believe that the technology behind physical modeling can do much more than replicate familiar sounds from the pantheon of existing musical instruments. By separating this technology from physical reference, we can view the art of physical modeling as a distinct general-purpose synthesis paradigm in its own right, in the same category as FM synthesis, subtractive synthesis, and additive synthesis.

Hence, we present Resonarium, a comprehensive general-purpose physical modeling instrument and multi-effect, with an emphasis on real-time playability, organic expressivity via MIDI polyphonic expression (MPE)[49], and abstract, creative sound design. It can be invoked as both a standalone program, and a VST3 plugin, and is intended to fit comfortably within a musician's workflow.

The scope of this project was undefined at inception. So it goes with research. As the scope has congealed, however, development has been shepherded by the following principles:

1. **Focus on the sound, not the model.** The techniques of physical modeling have been refined over decades to accurately model specific real-world instances with remarkable fidelity. As such, many extant physical modeling instruments are explicitly designed with respect to a physical referent. In this work, we instead observe that the technology behind such these models can be abstractly detached from physical reference; instead, we employ it as a first-class synthesis technique in its own right. Through such a lens, one can synthesize exotic timbres that would be difficult if not impossible to reproduce through other sound design techniques. One can interpolate between real-life instrument models, or extrapolate far beyond such islands of wood-and-metal tangibility. Resonarium encourages the exploration of liminality, and the vast interstitia between what is and what could be.

   While Resonarium can effectively emulate real instruments, it is particularly effective as a tool of abstract sound design. We believe that this application has been relatively neglected by existing works, and we consider this to be our focal contribution to the extant collection of physical modeling research.

2. **Expressibility is paramount.** Resonarium should expose granular configurability to the user, and let her use the instrument as she wishes, regardless of the designer's original intent. When possible, musically interesting parameters should be individually accessible and modulable. There should be a flexible, polyphonic modulation system so that users can creatively sculpt the timbre of their creations over time. Most importantly, Resonarium should emulate the expressivity of real instruments, so that users can manipulate their creations through multiple dimensions of control. We encourage live performance and improvisation. Most importantly of all, the instrument should be *accessible*: the user should be able to learn

and use Resonarium without needing to read hundreds of pages of germane academic literature.

3. **Software instruments are instruments too.** An instrument is more than just a tool used to create art—often, it is the art. Instruments themselves are often objects of beauty. Their design kindles curiosity, inviting manipulation and play. Indeed, a musical instrument is a sort of meta-art: an artifact of creative expression through which further expression is incarnated and rendered. Software, on the other hand, so often places function over form—but we believe that digital instruments are instruments too, and deserve to inherit the inspired legacy of their real-world ancestors. While it is somewhat unreasonable to compare the aesthetics of digital software widgets to the artisanal craftsmanship of a real instrument, we have still done our best to build a user interface that is inspired, stylized, and inviting.

In pursuit of these objectives, we present: (1) a digital instrument supporting MPE with sample-rate parameter interpolation; (2) a distinct waveguide model design that unifies Karplus-Strong synthesis, banded waveguide synthesis, and modal synthesis; and (3) an semi-modular and comprehensive modulation system, among other proverbial bells and whistles[1].

We remark that Resonarium **is not complete**. There remain several enhancements that have not yet been implemented. Some modules are incomplete. There are, of course, bugs. This document can do little more than expose a frozen snapshot of Resonarium's development state circa late 2024. However, Resonarium is both stable and playable as of writing. We do not anticipate the scope, appearance, or capabilities of the software to radically pivot in the near future. This document will be amended as development proceeds.

While the feature set currently implemented in Resonarium might compose a solid "1.0" release, its performance and stability are insufficient for any sort of wide public release. Our priority, as of writing, is to squash bugs and optimize performance. A list of some known issues can be found on GitHub[2].

Resonarium is designed with expressibility in mind, we **strongly** recommend that a MPE controller, such as the Roli Seaboard [32] or Ableton Push 3 [3], is used with the software. Many packaged presets are designed with MIDI polyphonic expression in mind, and may not work as well (or sound as interesting) without access to these control signals[3].

The source code, along with audio samples, installation instructions, and some appropriate caveats, can be found on GitHub[4].

[1] Indeed, Resonarium can effectively synthesize chimes, flutes, and woodwinds.

[2] https://github.com/users/gabrielsoule/projects/1

[3] Many DAWs support manual automation of MPE parameters. Through this technique, it is possible to explore the expressive potential of Resonarium without using a physical MPE control surface.

[4] https://github.com/gabrielsoule/resonarium

## 1.1 Waveguide Synthesis

Let us start at the beginning, and discuss the *synthesizer*. Synthesizers come in all shapes and sizes, but at a sufficiently blurred level of abstraction, they all work in the same way. A bank of *oscillators* generates primitive waveforms, which then are routed into an *effect rack*. The oscillators are often simple periodic functions that vomit forth samples according to some rule: an oscillator might be programmed to create a sine wave, or a sawtooth wave, or it might output data from a wavetable in memory. Sometimes, oscillators are used to modulate other oscillators, as in the case of FM synthesis. The output of these oscillators often requires further refinement, which is handled by the effect rack: here, distortion, filtering, reverb, and other effects are applied as desired.

Resonarium, too, implements an effect rack: however, it differs in *how* the unprocessed sounds are generated. It does not employ oscillators in the usual sense. Instead, it is based on *physical modeling*, in which the physical properties of musically interesting objects, such as strings, bars, and bells, are modelled via computer simulation. A physical instrument will not produce sound on its own: the musician will have to strike, brush, shake, or otherwise stimulate the object in some way. Our computer models are no different: in order to produce sound, they must be *excited* by an external signal—in practice, this is often a short burst of noise, or even a single impulse. Thus, many of the built-in "oscillators" in Resonarium only produce simple, inharmonic noise: clicks, pops, and static. All the musical intrigue flows from the models' responses to the exciter signal.

The physical modeling techniques used in Resonarium are inspired by *digital waveguide synthesis* and *modal synthesis* [5, 42][5]. For obvious reasons, we expect the target audience of this particular version of this particular document to possess some familiarity with these techniques. Regardless, in the interest of pedagogical completeness, we present a cursory treatment of the art in section 3.1.

### 1.1.1 Resonarium as an Audio Effect

By tradition, the impulse signal in a physical modeling workflow is simple and inharmonic, but we remark that this need not be the case. Any signal can be used to excite a physical model, and Resonarium is explicitly furnished with both a sampler and a live external audio input. Therefore, Resonarium be used as a transformational "vocoder on steroids" applied to both recorded audio and live performance. Like a vocoder, Resonarium transfers the timbral characteristics of the instrument preset onto the "carrier" exciter signal. From this perspective, there is no clear distinction between a "instrument" and an "effect": any instrument preset can be used as an audio effect applied to an external carrier, and any audio effect preset can be used an instrument with the built-in exciter generators. We have not observed this design perspective in other physical modeling instruments. More discussion is given in section 6.

[5] https://ccrma.stanford.edu/~jos/pasp/Modal_Expansion.html

Some digital audio workstations require that plugins be *either* instruments *or* effects. In such cases, plugins marked as instruments cannot be placed in audio effect chains, and plugins marked as effects cannot be placed in track instrument slots. To mitigate this restriction, Resonarium can be built as both a synth and an effect; it compiles into two distinct plugins that are functionally identical aside from their VST3 classification tags[6].

## 1.2 Many Perspectives, One Phenomenon

Under the hood, a physical model is little more than a collection of DSP *components*, such as resonant filters and waveguide loops, arranged in a certain *topology* that corresponds to a particular instrument's timbre and expressive range. Parameters corresponding to the instrument itself, such as bow pressure or pick position, are mapped indirectly via some function to the raw parameters of the model's components. This induces a loose sort of *isomorphism* between an *instrument view* of a physical model, e.g. a virtual violin, and a *component view* of a physical model, e.g. a network of generalized filters and delay lines that that makes a sound suspiciously similar that of a violin. The parameter mapping function facilitates this isomorphism, translating the configuration state of the virtual instrument into the corresponding state of the underlying DSP components.

One can interpret the component view as a "lower-level" description of the phenomenon captured by the instrument view, in the same way that molecular physics is a lower-level description of the phenomena studied by chemists, which is in turn a lower-level description of the phenomena studied by biologists, and so on, all the way up to sociology and philosophy.

Many extant physical modeling programs are explicitly designed around the instrument view. Parameters belonging to the physical instrument may be exposed to the user, but the underlying DSP components are intentionally inaccessible. Such virtual instruments model their referents with remarkable accuracy, yet in a Heisenbergian bargain, what they gain in precision they lose in customization: a violin model designed to sound like a violin will always sound like a violin, if parameters are limited to "realistic" values.

### 1.2.1 Hello From The Other Side

Resonarium flips this all on its head and explores the other side of the isomorphism. Resonarium is not designed to model a particular instrument: we distance ourselves from this metaphor and instead directly expose the user to a cornucopia of physical modeling components. The state of these components, and the topology by which they are connected, is all configurable, both by the user, and through real-time modulation via control signals. The user, if she so chooses, can configure Resonarium to emulate an instrument of her choice, and distribute this configuration as a *preset*. But she is under no obligation to recreate familiar instruments from the real world. Instead, Resonarium encourages

[6]The effect edition is distinguished by a handsome blue-green color scheme.

both exploration of the liminal space *between* instruments, and that great expanse beyond. It is designed to explore physical modeling not as a strict means to an ossified simulacrum, but rather as a fluid and unique synthesis technique in its own right.

Imagine if all modern software synthesizers were strict emulations of famous hardware synths, such as the Minimoog or the DX7! This, by analogy, is the state of most physical modelling software today. Physical modeling was developed with respect to the real world, but there is no reason for it to remain there.

We believe that this detachment from physical reference is underrepresented in both research literature and commercially available software. Through such a lens, we hope to expose an uncharted dimension of sound design rich with potential, and make it accessible to musicians and producers.

## 1.3 Expressability and MPE

A key feature of this work is its expressability. Physical modeling is a reasonably mature domain, and many years of diligent research has yielded some models that are nearly indistinguishable from their metal-and-wood counterparts. However, the sound produced by an instrument is only half the puzzle. Most physical instruments possess several dimensions of expression: pressure, slide, vibrato, striking position, damping, and so on. During a performance, these modes of expression must be skillfully manipulated in real time by the musician. The *shape* of an instrument, therefore, and the physical interfaces by which it is manipulated by the musician, compose an inseparable component of the instrument's identity.

Therefore, designers of physically modelled instruments face a tenacious problem that may be neither mathematical nor mechanical: rather, it is a matter of *ergonomics*. A realistic computer model of an instrument should expose to the musician every mode of expression present in its referent; however, standard computer music peripherals and standards only capture a tiny subspace of this rich domain of interaction. The computer keyboard itself, for instance, can only communicate binary signals: note on and note off. The MIDI protocol is a meager improvement: while there is a notion of per-note *velocity*, and monophonic pitch bending, the protocol itself is still built around binary note on and note off messages.

With some faculty in soldering, component sourcing, and hardware hacking, it is feasible to build a custom hardware interface for a computer instrument; however, this approach is out of reach for all but the most dedicated enthusiasts. A large touchscreen device can detect slide or timbre, and tablets have been harnessed for this purpose [25], but many commercially available tablets do not detect pressure, and the smooth, fungible surface of a tablet yields no tactile feedback for the musician.

Extant physical models simulate their physical referents with remarkable efficacy. The bottleneck, then, is not the models themselves—rather, it is the

*human-machine interface*: a musician's ability to manipulate a multidimensional control space in real time.

### 1.3.1 MIDI Polyphonic Expression

A promising solution is *MIDI polyphonic expression* (MPE) [49]. MPE is an extension of the MIDI protocol that facilitates real-time continuous control of slide, pressure, and timbre for each active note in a performance. Since an expressive physically modeled instrument requires a real-time input channel for each dimension of expression, and MPE explicitly facilitates those input channels via an open protocol, MPE is an indispensable asset for such instruments. Furthermore, there exist several MPE controllers available for purchase (e.g. [3],[32]).

Since MPE synergizes so powerfully with physical modeling, one might expect a flourishing of MPE-compatible physically modeled instruments to have cropped up in recent years. This does not appear to be the case: the author is not aware of any general-purpose physical modeling tools that explicitly take advantage of the MPE protocol[7]. Hence, Resonarium is designed with first-class MPE expression in mind. MPE control channels can be mapped to any number of internal parameters, and thus Resonarium functions as a live instrument with all the expressive potential that a musician would expect.

### 1.3.2 Modulation and Automation

Resonarium is endowed with a comprehensive modulation system. In addition to a standard collection of ADSR envelopes, random number generators, and macros, Resonarium includes several multi-segment envelope generators (MSEGs). Since Resonarium can be built as a VST3 plugin, all parameters are exposed to host software and can be automated therein.

Many existing synthesizers and audio effects are impeded by an anemic modulation system that is more of an afterthought than a core feature. Often, only a handful of "obligatory" LFOs and envelopes are provided to the user. By contrast, the modulation system of this work is a key feature of the software. We have worked to implement an expressive and expansive modulation system that supports high modulation rates and a wide variety of modulation sources.

Furthermore, most parameters in Resonarium support *stereo modulation*. Digital audio generally flows through both a pair of stereo channels, and Resonarium's modulation sources are capable of modulating the left and right values of a particular parameter independently. This potentiates exceptional creativity in the stereo field, and obviates the need for several stereo-specific controls. We describe the modulation system in more detail in section 7.

[7] GeoShred [25] supports MPE via its AU3 plugin; however, GeoShred is strictly focused on the simulation of real, distinct instruments, whereas this work is focused on abstract sound design. We discuss GeoShred further, with respect to Resonarium's design principles, in Section 3.1

## 1.4 The Philanthropic Case for Physically Modeled Instruments

> *I am, somehow, less interested in the weight and convolutions of Einstein's brain than in the near certainty that people of equal talent have lived and died in cotton fields and sweatshops.*
>
> Stephen Jay Gould

I will briefly depart from the royal academic nosism to share a somewhat personal story about my interest in this particular field of study. It will, perhaps, illuminate my motivations for the development of this software with greater truth and candor than all the "practical" considerations elucidated above.

Neither my elementary school nor my high school offered music classes, and, for many reasons, I felt discouraged from musical exploration as a child. Regardless of the reasons, the outcome was clear: I was, in a sense, a musical "late bloomer," especially juxtaposed against many who dedicate themselves to musically relevant pursuits.

Thus, my gateway into music, even as an adolescent, was through my laptop computer. I had spent much of my childhood assuming that music was simply out of reach to people like me, and it was the universal accessibility of the computer that shattered this spell irrevocably.

I assert with confidence that few innovations have been more musically influential than the proliferation of affordable, portable computing devices (and associated musical software). The laptop computer has done for musical production what the printing press did for the written word in Europe, and I marvel at this observation regularly.

This claim may appear reckless at first blush, but consider: for the low cost of a couple hundred dollars, an aspiring musician with a laptop computer can acquire a set of production, mixing, mastering, and recording equipment that, in the mid to late 20th century, would have cost *hundreds of thousands* of dollars and consumed several rooms in a physical building—a laptop, by contrast, fits comfortably in a backpack. Rare hardware synths, analog compressors, reverb modules, arcane and alien multi-effects: all are available at the push of a button and can be stored within a medium the size of a fingernail. The nepotistic or financial barriers to the production of professional-quality music have not been lowered; they have been *atomized*. With a little practice, anyone in the world can boot up some software and weave into reality the sounds they dreamed of as a child.

There is, of course, a void in this otherwise rosy tapestry: accurate reproduction of expressive physical instruments. Sampled instruments, historically, are employed as digital simulacra; however, sampling an instrument only captures a reductive subspace of its expressive potential. This is where physically modeled instruments enter play, and it is this aspiration to *democratize* access to quality instruments — to enable anyone in the world to play any*thing* in the

world — that motivates my work on this software. It might be naïve to assume that computer instruments will equal or surpass their physical counterparts in expressivity and fidelity, and that the corresponding control surfaces will be more easily accessible. Nor do I claim that my work in this paper *resolves* this problem in its entirety. Rather, it is my earnest hope that it pushes the envelope ever so slightly in the right direction. So it goes with all such lofty aspirations. We are always obliged, at least, to try.

To that distant end I present this little contribution.

## 1.5 Comparison To and Inspiration From Existing Art

Resonarium is motivated by voids within the existing body of physical modeling instruments. As such, we conclude this chapter with a brief comparison to existing works in this domain. While there exists a plethora of physical modeling software, e.g. [13, 50], we particularly focus on self-contained *instruments* that target the commercial market. We describe how Resonarium diverges from and innovates beyond extant software in this domain, and, since art is rarely produced in an inspirational vacuum, ascribe credit where credit is due.

### 1.5.1 GeoShred

GeoShred [25] is an excellent collection of physically modelled instruments, which are an MPE-capable audio plugin, or an iPad application. We applaud GeoShred for its accessibility across multiple dimensions: iPads are relatively ubiquitous, especially within educational contexts, and the base software is affordably priced. We also commend the software for its expressivity: the iPad provides a two-dimensional control surface that facilitates glide and multi-touch, and MPE is properly supported in the audio plugin. Most importantly, the sound is realistic, precise, and universally a pleasure to listen to.

Where Resonarium differs from GeoShred is in its *perspective*. Recall from section 1.2 that there are two ways of looking at a modelled instrument: as a virtual manifestation of a physical referent, or as a network of DSP components. GeoShred firmly asserts itself on the *instrument* side of the isomorphism: though it allows the user to customize existing instrument models to create presets, it does not expose the user to the underlying DSP components, their network topology, and their raw configuration states.

Resonarium, by contrast, exposes several lower-level DSP components to the user, and it is up to her to explore and invent musically interesting configurations. To this end, Resonarium offers the user a collection of common synthesizer tools, such as a modulation system and a general-purpose effect chain. In other words, GeoShred can be considered an *instrument collection*, whereas Resonarium is more of a general-purpose *synthesizer*.

None of this is to imply that there exists a *deficiency* in GeoShred's capabilities. The two programs are different tools for different purposes, and complement each other well. The fidelity and precision of GeoShred's guitar

models are worthy of aspiration, and we hope one day to imbue Resonarium with similar capability.

### 1.5.2 Objekt

Objekt[31] is a physical modeling synthesizer from Reason Studios. Objekt was released in 2023 and is therefore is a relatively modern addition to the physical modeling ecosystem. Unlike GeoShred, Objekt adopts the *component view*: it exposes its underlying DSP components to the user, and it is up to the user to tinker with these parameters to produce something musically interesting. This design philosophy is similar to the one espoused by this work, and therefore there are a number of similarities between the two programs: for example, both synthesizers are oriented around an array of (optionally) coupled waveguide components.

Some of these similarities are not entirely coincidental. Objekt was released while this work was in development, and therefore inspired a couple of the features present in Resonarium, which we credit in the footnotes.[8]

Objekt, however, falls short with respect to both configurability and musical expression. Its modulation system is threadbare and difficult to work with. More importantly, it does not support MPE. Patches created via Objekt lack the expressive playability one would expect from their real-life counterparts, and therefore, can be substituted with a corresponding sampled instrument without any loss of expressive potential. Despite these shortcomings, Objekt is a well-designed piece of software with a beautiful user interface and plenty of space for creative exploration. Its synthesis capabilities are broad, from bells and bowls to flutes and whistles.

# 2 Resonarium, At a Glance

Figure 1 displays the Resonarium user interface. The software can run as a standalone application, or as a plugin in a digital audio workstation. It is written entirely in C++, and uses the JUCE framework for both UI and audio programming. We dissect the UI further in section 9, and elaborate on the software engineering in section 9.3.

Clearly visible are the five main components of the synthesizer:

- **Exciters.** The left-hand column contains the exciter modules. These modules generate audio signals that are used to excite the resonators. We describe them in section 6.

- **Resonators.** The upper centered panel exposes the resonator banks: the heart of this synthesizer. Audio from the exciters is routed to these

[8] In particular, both the *impulse train exciter* is inspired by Objekt's static and pulse noise modes, and the feedforward mixing from model bank $i$ to model bank $i+1$ is adapted from Objekt's signal flow as well (originally, the banks were parallel).


Figure 1: The main Resonarium window

physical models, which then resonance based on both the harmonic content of the exciter signal and the resonators' own internal configuration states.

- **Modulators.** The lower center collection of panels expose a phantasmagoria of modulation sources and germane paraphernalia, such as a modulation matrix, a modulation source browser, and a macro system. Both monophonic and polyphonic modulation is supported. We elaborate in section 1.3.2.

- **Effects.** The right-hand column contains a scrollable list of digital audio effects. Each individual synthesizer *voice*, i.e. each note, is assigned its own instance of the effect chain, which means effect chain parameters can be modulated polyphonically. Multiplying the effect chain in this way is computationally expensive, but is necessary for effective polyphonic modulation. We elaborate further in section section 8.

The signal flow flows roughly left to right. The *exciters* produce an excitation signal which is routed to the *resonator banks*. When excited, the resonator banks will produce sound, which is sent to the the effect chain on the right column of the interface. The processed signal is the final product, and is routed from the synthesizer into the wider world beyond.

# 3 Alice's Narrative and the Karplus-Strong Algorithm

The time has come to dissect the beating heart of Resonarium: its physical models and the mathematics therein. Study of this functionality will supply illuminative context for other components of the instrument, such as the exciters.

In research, there are several ways by new wisdom is incarnated and integrated. All roads lead to Rome, but not all roads are created equal.

One path is through skilled application of fundamental principles and mathematical insight. It is possible to start from the well-understood art of mechanical physics and build, step by mathematical step, a collection of physically modelled instruments based on these principles. For example, one can study the *ideal vibrating string* as a mathematical construct, and then apply DSP principles to derive a musical model of the underlying idealized physical phenomenon. Similar techniques have been profitably applied to woodwinds, bowed instruments, and more.

Another path is navigated via exploration, experimentation, and the frequent exploitation of happy accidents. It turns out that many of the DSP components used to build physical models are not particularly specialized: for some instruments, all one needs is filters and delay lines. While the mathematics that link the digital implementations of physical models to their idealized referents is, at times, intimidating, the implementations themselves can be comparatively simple. It is, in fact, not difficult to accidentally "invent" a physical model by aimlessly tinkering with DSP components. The mathematical justification behind such models can be then derived retroactively; in fact, this is more or less how the Karplus-Strong string model was originally discovered (section 3.2.)

Much has been written about the mathematics of physical modeling, and for a comprehensive treatment of the art, we direct the reader to the exemplary books by Smith [42, 41]. In the interest of pedagogical diversity, we will present the fundamentals of physical modeling through a slightly different lens. Basic physical modeling is not too difficult; in fact, one can naively derive an effective string model with nothing but an impulse generator, basic DSP components, and a light sprinkle of curiosity. We demonstrate this via the narrative below.

## 3.1 Alice and the Box

While enjoying an afternoon at the beach, Alice stumbles upon a mysterious device washed up amongst the kelp and flotsam. She assesses it to be a computer of some sort, and, since she has recently retired from an illustrious career as a placeholder in cryptographic thought experiments[9], she has plenty of free time to pursue her alternative interests. She takes the device home for further study.

She discovers that the device is a DSP component with a singular (and rather trivial) purpose. When triggered, it produces a digital impulse signal $\delta[n]$, which

[9] https://cryptocouple.com

Alice dutifully recalls as

$$\delta[n] = \begin{cases} 1, & \text{if } n = 0 \\ 0, & \text{if } n \neq 0 \end{cases}$$

Alice plugs the device into her speakers, and all she hears is a sharp *pop* when the impulse is triggered. Fortunately, Alice has been spending her retirement pursuing an incendiary passion in digital signal processing, so she is delighted by this otherwise mundane discovery.

### 3.1.1 Iteration I: Impulse Train Generation

Alice wonders if she can turn her impulse generator into something *musically interesting*. The task is daunting: how can she transform a harsh *pop*, corresponding to a single nonzero sample, into a pleasant and engaging sound? After some consideration, Alice decides to insert a *delay line* [36] into the signal flow, coupled with a feedback loop[10].

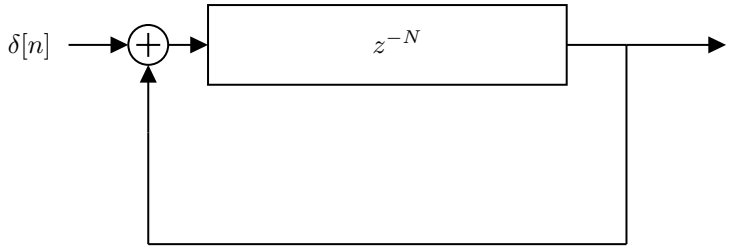



Figure 2: Alice's impulse train generator

Alice establishes a sample rate of $f_s$, and varies $N$. She observes that the impulse signal becomes "stuck" in the loop, cycling at a rate of $f = \frac{f_s}{N}$. Each time the impulse completes a full cycle, it is output to the speakers. In other words, Alice has created an *impulse train generator* [44][11] with a period of $P = \frac{1}{f} = \frac{N}{f_s}$.

For low values of $f$ (below approximately 40Hz), Alice hears a continuous sequence of clicks. She finds this marginally more interesting than the single click produced directly by the machine, but it's nothing remarkable. However, when she increases $f$ to audio-rate levels, the distinction between each successive click asymptotically blurs before vanishing, and what she hears is a sustained, musical tone with a fundamental frequency of $f$ Hz. Alice is delighted by this discovery: by injecting a delay with feedback between the mysterious device and her audio output, she has transformed a sharp, unpitched impulse into a sustained musical note.

[10] Alice's construction could also be described as a *feedback comb filter* under a particular parameterization.

[11] https://ccrma.stanford.edu/~jos/sasp/Impulse_Trains.html

### 3.1.2 Iteration II: First Order Decay

Alice is not entirely satisfied with her invention. The sound of her "instrument" is harsh; neither its amplitude nor timbre vary over time. To rectify this, she inserts a attenuating multiplier $g$ inside the delay loop. Now, the gain of the signal is multiplied by $g$ whenever it cycles through the feedback loop. If $g > 1$, the gain of the signal overflows and blows up Alice's speakers; however, for $g < 1$ the signal decays at a rate proportional to both $g$ and $f$.

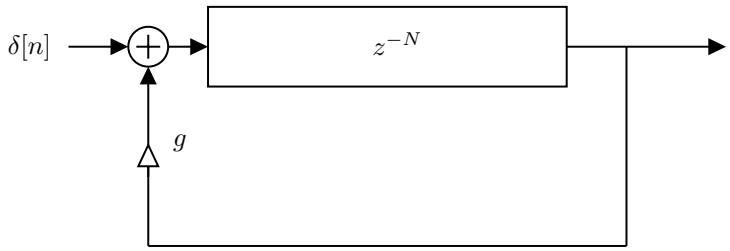



Figure 3: Alice's impulse train generator, with gain multiplier $g$ injected into the feedback loop.

Alice is delighted by this discovery: she can now produce a sharp plucked sound at any pitch of her choosing. The parameter $g$ is effectively an amplitude envelope, and by tuning it carefully, she can control the decay time of each note.

However, the *timbre* of the sound is still unchanging; furthermore, it is unpleasantly bright and harsh. Alice considers applying a low-pass filter to the output signal to attenuate the brightness, but this solution isn't perfect: the sound is be more pleasant to listen to, but its timbre remains constant. *Boring*, she thinks. Alice wants the timbre to audibly morph over the course of a single note. Clearly, she needs something more sophisticated than an exponentially decaying amplitude envelope.

### 3.1.3 Iteration III: Damping Filters

That evening, Alice retires to the living room to light a fire and enjoy a glass of fine sparkling wine. The flames in the fireplace contort themselves with wild agency, caressing the edges of their brick prison with flirtatious irreverence. The shape of the fire reminds her of a spectrum analyzer: its ephemeral tongues of flame render faithfully the resonances of some untamed and alien signal. She idly wonders what the fire would sound like, from this perspective. The "frequencies" are strongest in the middle of the fireplace; towards the edges, where fresh fuel is scarce, they undergo a ghostly blue attenuation, and then Alice—enraptured by this fantastical isomorphism between the shape of the flames and the timbre of music, and, no doubt, teetering at the hazy border of that fugue state that, induced by pyrohypnosis and abetted by light drink, can ensnare a clever mind for hours—is struck by an *idea* that leaps like an ember from the flames and grounds itself in the back of her skull.

The problem, as Alice sees it, is that $g$ attenuates all frequencies equally. If she could somehow attenuate the higher frequencies at a greater rate, then the timbre of the sound would be time-dependent. The harsh high-frequency buzzing would vanish quickly as a transient, rather than linger awkwardly with each note. *Maybe I can split the signal into a small number of bands,* she muses, *and apply a distinct $g_i$ to each band.* But as she gazes idly into her fiery spectroscope, the idea shimmers into crystalline focus, and she sees it in its entirety: for what is a *filter* if not an exhaustive array of gain multipliers, where each multiplier $g_i$ corresponds to a specific frequency? A good low-pass filter should precisely induce the frequency-dependent attenuation she needs. The solution is obvious. It has been there for hours, lurking in plain sight from the moment she lit the match.

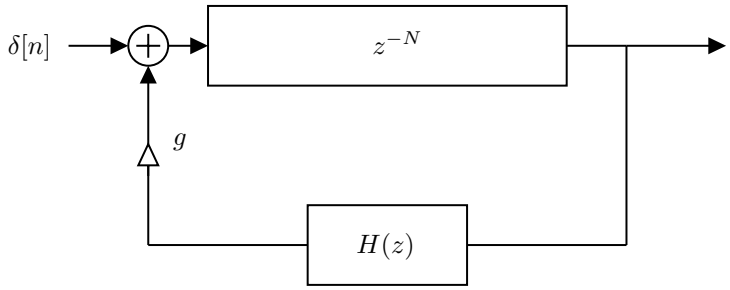



Figure 4: The third iteration of Alice's experiment, with a low-pass filter placed inside the feedback loop.

Invigorated, Alice places a low-pass filter inside the feedback loop, and turns the device on. The sound produced has a sharp transient with rich and bright harmonic content; however, the higher frequencies are quickly attenuated by the loop filter, and the note fades out with only a handful of lower harmonics present. Most interestingly, her experiment makes a sound suspiciously similar to that of a plucked string. It's not perfect, but it's remarkably familiar.

Alice immediately hires a handful of engineers and hardware procurement specialists, and goes on to launch a digitar company, from which she becomes grotesquely wealthy. She reconnects with Bob, and the two spend many a sunset sipping expensive champagne, engaging in light conversation, and otherwise enjoying the finer things life has to offer. Mallory attempts to repair their strained relationship, but there's simply too much bad blood from their prior careers in cryptography to make things work.

## 3.2 The Karplus-Strong Algorithm

There are two motivations for the inclusion of this narrative exercise. The first is to demonstrate that the fundamentals of physical modeling need not be intimidating. All one needs is basic filters, delays, and curiosity. Alice has

inadvertently rediscovered[12] the *Karplus-Strong algorithm* [19, 37], and, like Alice, Kevin Karplus and Alex Strong stumbled upon the configuration entirely by accident, whilst mucking about with an experimental wavetable synthesizer [42][13]. Though the discovery was accidental, its curious timbral resemblance to a plucked string is *not* a coincidence. It was later shown [45] that the Karplus-Strong algorithm (and other similar techniques) can be derived mathematically from the principles of the *ideal vibrating string*; in other words, the Karplus-Strong algorithm is a mathematically consistent *computational physical model* of an ideal string. The broader art of *waveguide synthesis* [40, 42] generalizes these ideas further; such techniques can emulate many classes of stringed instruments, as well as brass, woodwinds, and more.

We refer the reader to [42] for a comprehensive treatment of waveguide synthesis. For an abbreviated and hands-on approach, we strongly recommend the exercises in [37], which are accompanied by helpful Faust [13] code.

### 3.2.1 Karplus-Strong and Resonarium

The second purpose of Alice's narrative is to warm the reader up to the ideas present in Resonarium. This original intent of this work was to model guitars and other stringed instruments; therefore, Resonarium's object models are inspired by the Karplus-Strong model and its extensions [15, 42][14]. Alice's experiment, in fact, is a much reduced version of the object models used in this work.

## 4 The Resonators of Resonarium

> *You will never be anything but a philosopher—and what is that but an ass who plagues himself all his life, that he may be talked about after he is dead.*
>
> Madame Rosseau to Jean le Rond d'Alembert, who first solved the wave equation for an ideal vibrating string [22]

Figure 5 shows the *resonator bank* panel and tabs. Each rainbow-colored column corresponds to a single "waveguide loop" (much like Alice's string loop, only with several enhancements). Eight of these structures are collected into a *bank*, where they can then be coupled together under various topologies. Res-

[12] The original Karplus-Strong model uses a two-point average, $H(z) = 0.5 + 0.5z^{-1}$, as the loop filter. We do not specify the filter Alice uses; the purpose of the narrative is to develop intuition around the algorithm, rather than to prescribe a particular implementation. In practice, any filter can be used as long as its peak gain is no greater than 1.

[13] https://ccrma.stanford.edu/~jos/pasp/Karplus_Strong_Algorithms.html

[14] https://ccrma.stanford.edu/~jos/pasp/Extended_Karplus_Strong_Algorithm.html


Figure 5: The Resonator Bank UI panel

onarium is endowed with multiple such banks, and the user can switch between them via the tabs on the panel header.

This all induces an immediate problem with respect to nomenclature. We have made liberal use of the term *physical model*; however, this term is easily overloaded on several layers of abstraction and fails to capture granularity. Each of the waveguide loops in Resonarium could be rightly considered a physical model; however, we could also describe the bank of waveguide loops as a physical model (e.g. an instrument with several strings, or modes of vibration). If the user uses multiple banks of loops to model a particular instrument and saves her configuration as a preset, then the state of the entire synthesizer could be justifiably labeled as a physical model. To resolve this ambiguity, we will use the following definitions for the remainder of this document:

- We will call the individual waveguide loops *resonators*. It is not strictly accurate to describe them as "string models" or "digitar strings," since the models have diverged from their physical referents over the course of development. We describe their structure in section Figure 6.
- Eight *resonators* are gathered together into a single *resonator bank*. By default, this categorization is purely epistemological: the resonators operate in parallel, and do not interact. However, the bank can be parameterized by a *coupling mode*, which allows the resonators to interact with each other through a handful of distinct topologies. We describe the coupling modes in section 5.
- Resonarium is furnished with four separate resonator banks. Like the

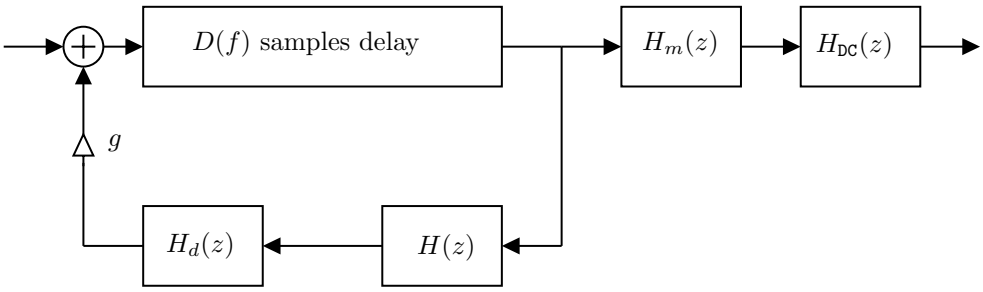



Figure 6: The internal topology of a single resonator $R_i$ tuned to frequency $f$

resonators themselves, they operate by default in parallel. However, there exists an option to feed-forward the signal from resonator bank $i$ as the excitation for resonator bank $i+1$, which can produce interesting musical textures.

With that out of the way, we are ready to discuss the resonators.

## 4.1 The Resonators

The structure of our resonator is presented in Figure 6. Recall that each resonator corresponds directly to one of the colorful columns in the resonator bank panel (Figure 5). Readers familiar with the digital waveguide literature will notice immediate similarities between this diagram and the extended Karplus-Strong algorithm [15]. However, there are distinct differences between EKS and Figure 6, in both implementation and purpose. Our resonator is a waveguide-inspired structure that unifies the principles of*Karplus-Strong string synthesis*, *modal synthesis* [5, 42][15], and *banded waveguide synthesis* [12]. A Resonarium instance can "interpolate" between these three modes of synthesis in real time, or in response to the musical control surface, via the modulation system.

- $H(z)$ is the damping filter. It is implemented as a *unity-gain second order state variable filter*, and is carefully designed such that its resonant frequency[16] is always equal to the waveguide loop's natural eigenfrequency (as determined by the length of the delay line). Notably, for standard second-order biquad filters, the resonant frequency does *not* strictly equal the cutoff frequency. Resonarium's loop filter automatically compensates for this discrepancy, as described section 4.4.
- $H_d$ is a *dispersion* filter, which corresponds to *stiffness* in the vibrating medium [42][17]. It induces a metallic, inharmonic timbre. It is implemented as an allpass filter, and we describe it in section section 4.6.

[15] https://ccrma.stanford.edu/~jos/pasp/Modal_Expansion.html

[16] The angular resonant frequency $\omega_r$ of a filter $H(z)$ is defined as $\omega_r = \arg\max_\omega |H(e^{j\omega})|$. This is the frequency for which the filter's corresponding amplitude response is greatest; i.e. the global maximum of its amplitude response plot.

[17] https://ccrma.stanford.edu/~jos/pasp/Stiff_String.html

- The delay line is interpolated via third-order Lagrange interpolation [42][18]. The delay length function, $D(f)$, is nontrivial, since the damping filter $H(z)$ exhibits a nonlinear phase response, which in turn induces tuning error in the waveguide model. This must be compensated for by $D(f)$.

- $g$ is a first-order decay multiplier, much like the $g$ in Alice's experiment. When $g$ is set to 1, the signal inside the loop does not decay[19], and thus the entire resonator performs much like an active *wavetable*: a periodic waveform is generated continuously, and modules in the subsequent effect chain can creatively shape that sound in musically pleasing ways.

- $H_m(z)$ is a multi-mode state-variable filter placed *after* the waveguide feedback loop. Hence, it is often called the *post filter*. The post filter can be tuned a *modal filter* [5], which facilitates modal synthesis within Resonarium. We discuss this further in section 4.7.

- $H_{\texttt{DC}}(z)$ is a DC blocker [42][20]. The accumulation of a DC offset inside a waveguide loop can induce a deleterious effect on stability. The DC blocker was originally placed *inside* the waveguide loop; however, this was observed to negatively impact the timbre of the sound produced.

## 4.2 Karplus-Strong Versus Banded Waveguides

The damping filter in Resonarium is carefully designed to facilitate live *interpolation* between a Karplus-Strong (KS) style of string synthesis and the *banded waveguide* technique. To explain this properly, we need to briefly cover the differences between KS synthesis and banded waveguide synthesis, and point out a few key similarities that enable us to unify both techniques within a single DSP component.

Banded waveguide synthesis and Karplus-Strong synthesis were originally intended to model different physical referents. The Karplus Strong algorithm, as we mentioned in section 3.2, can be formally derived from the physical principles of vibrating strings, and its extensions can effectively model stringed instruments, such as the guitar [42][21]. Banded waveguides, by contrast, were proposed by Essl and Cook [12] as a efficient way to model struck bar percussion instruments, such as marimbas and the like.

A banded waveguide model often is composed of several delay line feedback loops. Like Karplus-Strong feedback loops, there is a filter present; however, these filters are typically resonant bandpass filters that attenuate all frequencies except for a narrow band of sinusoids. Each feedback loop, therefore, corresponds to a single resonant mode within the physical referent.

[18] https://ccrma.stanford.edu/~jos/pasp/Lagrange_Interpolation.html

[19] In practice, this is not strictly true. The Lagrange interpolation in the fractional delay line induces a mild low-pass effect within the resonator, which gradually attenuates the signal inside the loop. Fortunately, this effect is mild.

[20] https://ccrma.stanford.edu/~jos/filters/DC_Blocker.html

[21] https://ccrma.stanford.edu/~jos/pasp/Extended_Karplus_Strong_Algorithm.html

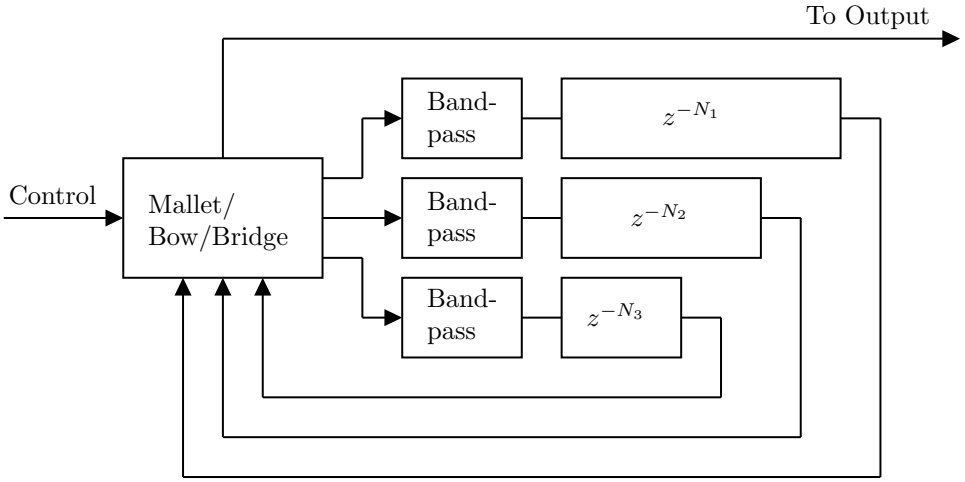



Figure 7: A simplified diagram of a banded waveguide model with three resonant modes, adapted from the original paper [12], and the STK source code by Cook [50] (which we use as a reference implementation). The delay lines in the figure are tuned to match the resonant frequencies of their respective bandpass loop filters.

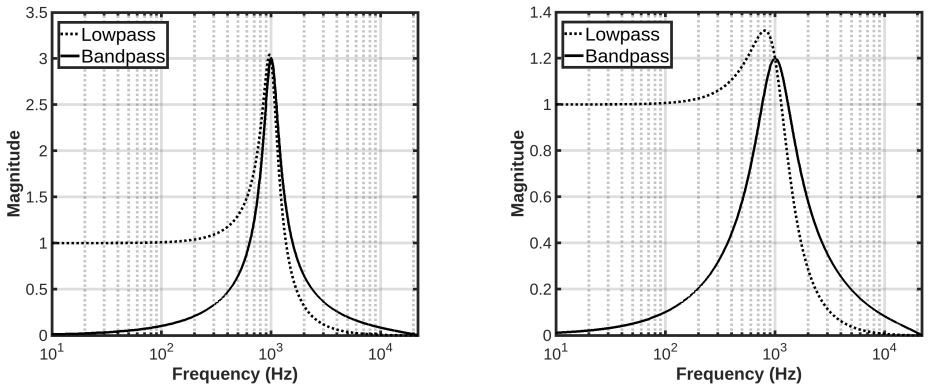
Figure 7 shows a diagram of a banded waveguide model with three resonant modes. The large block on the left often does some sort of inter-mode coupling and excitation modeling, and the mechanics within are beyond the scope of this section. Suppose, however, that the mallet/bow/bridge block is a no-op[22], and simply passes the signal wires through untouched. Then, the model effectively reduces to three parallel Karplus-Strong string models, with one key distinction: the loop filter is a resonant bandpass filter instead of a damping lowpass filter.

From this perspective, (parallel arrays of) Karplus-Strong string models and banded wavguide models are, from an implementation perspective, more alike than the underlying mathematical derivations might suggest. To concretely unify the two approaches, we observe that a second order lowpass filter [52, 7], with sufficiently high resonance, is musically indistinguishable from a resonant bandpass filter, under the right conditions. In fact, we have verified experimentally that there is no audible difference between a banded waveguide model implemented with resonant bandpass filters (as implemented in [50]), and a banded waveguide model using appropriately-tuned resonant lowpass filters[23]. Furthermore, a second-order lowpass filter with maximally flat (Butterworth) response can be effectively used as the damping filter for a Karplus-Strong in-

[22] By tinkering with the banded waveguide model in the STK [50], we have observed that if the bowing simulation is omitted, then the coupling effect is negligible. It can be bypassed entirely, and the individual resonances can be operated in parallel, without audibly degrading the timbre.

[23] The same observation applies to resonant highpass filters; however, highpass filters are used less frequently in string synthesis.

spired string model.




(a) $Q = 3$ (b) $Q = 1.2$

Figure 8: The frequency responses of two pairs of lowpass/bandpass second order digital biquad filters. The cutoff frequency is 1000 Hz in all figures, but resonance ($Q$) is varied between Figure 8a and Figure 8b.

Therefore, it is feasible to *interpolate* a single resonator between a Karplus-Strong string configuration and a banded waveguide configuration by exposing the loop filter resonance as a configurable parameter. Figure 8a superimposes the frequency response of a lowpass filter and a bandpass filter, with reasonably high resonance, set to the same cutoff frequency. It is readily observed that the frequency responses of both filters are remarkably similar; furthermore, they both have a peak gain of approximately $Q$.

## 4.3 The Problem With Resonant Lowpass Filters

Unfortunately, this apparent congruence is a mirage—a trick of the low resolution and high resonance. Figure 8b shows the two filters configured with a lower resonance value ($Q = 1.2$), which magnifies two underlying discrepancies. The bandpass filter is well-behaved: its resonant frequency is equal to the cutoff, while the amplitude at this frequency is precisely equal to $Q$. By contrast, the lowpass filter's peak frequency is somewhat less than the cutoff, and its peak gain is greater than $Q$—somewhere around 1.3.

This observation exposes two immediate problems with our plan to use a resonant lowpass filter inside a waveguide feedback loop. The first is readily obvious: any filter inside a feedback loop *must* have a peak gain of no more than 1. If this is not the case, then the signal inside the resonator may overflow to infinity, wreaking unmitigated havoc upon the unfortunate musician's speaker system and eardrums. Furthermore, the peak gain should be very close to if not equal to 1; otherwise, the signal inside the loop may be attenuated too quickly. It is thus necessary to have at hand an efficient function to calculate the filter's peak amplitude $A_p$, so that it can be normalized (typically via multiplication by

$A_p^{-1}$). While second order band-pass filters are well behaved, with $A_p = Q$[24], it is not immediately obvious how to compute the peak amplitude of the resonant lowpass filter.

### 4.3.1 Matching Waveguide Eigenfrequencies to Filter Resonant Frequencies

The second problem is more subtle. The resonant frequency of a waveguide loop is determined the length of the loop delay line; it "wants" to vibrate at this eigenfrequency $f_0 \approx f_s/N$. A resonant filter also has an eigenfrequency at $f_p$, its peak frequency. For sufficiently high $Q$ values, it will resonate at this frequency and attenuate all other frequencies to near zero. Therefore, when a resonant filter is placed *inside* a waveguide loop, care must be taken to ensure that the filter's eigenfrequency and the waveguide loop's eigenfrequency are equal. Otherwise, the filter will attenuate the waveguide loop's natural frequency, the signal inside the loop will drop to zero within a small number of periods, and the waveguide model will not produce anything musically interesting.

Recall that the delay lines in Figure 7 are tuned so that the feedback loop's period matches the resonant bandpass filter's peak frequency $f_p$. Fortunately, $f_c = f_p$ for such bandpass filters; therefore, the delay line length $N$ can be related to the filter's cutoff via $\frac{F_s}{N} = F_c$. Furthermore, such filters are easily normalized.

As discussed above, and shown qualitatively in Figure 8, this is not the case for resonant lowpass filters. Therefore, we must show how to parameterize a resonant lowpass filter with respect to its peak frequency, not its cutoff frequency; furthermore, the filter must be normalized with respect to its gain at that peak frequency. Fortunately, this is not particularly difficult, as we show below.

## 4.4 The Resonarium Loop Filter

The loop filter $H(z)$ in Resonarium is a second-order digital filter implemented in the *state-variable* form [52, 8, 38]. We prefer the state variable topology over the classical biquad construction [7] for two key reasons.

- The state variable topology is stable under rapid parameter modulation. Biquad topologies, by contrast, tend to blow up under such conditions, and become irrevocably stuck in internal states that can only be described as musically repugnant.
- The state variable topology produces low-pass, band-pass, and high-pass output simultaneously "for free". These three outputs can be mixed to create a *multi-mode* filter whose type can be smoothly interpolated in real time.

[24] Some out-of-the-box implementations of second order bandpass filters normalize the amplitude by default, such that $A_p = 1$ for all $Q$.

The digital state variable filter can be derived from analog prototypes via any extant numerical integration methods. Chamberlin employs the forward Euler method in [8], while Smith presents a forward-backward Euler realization in [38]. This work implements the digitization of Simper [33, 35, 34], which profitably applies the bilinear transform [41][25] to the analog prototype.

### 4.4.1 Peak Frequencies of Analog Second Order Filters

The analog transfer functions of second-order analog lowpass, bandpass, and highpass filters, respectively, are:

$$H_{\texttt{LP}}(s) = \frac{\omega_0^2}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2} \tag{1}$$

$$H_{\texttt{BP}}(s) = \frac{\frac{\omega_0}{Q}s}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2} \tag{2}$$

$$H_{\texttt{HP}}(s) = \frac{s^2}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2} \tag{3}$$

As shown by Simper [33], analog state-variable implementations of these filters can be digitized via the bilinear transform to procure digital equivalents defined as follows:

$$H_{\texttt{LP}}(z) = \frac{g^2 + 2g^2z^{-1} + g^2z^{-2}}{(g^2 + gk + 1) + 2(g^2 - 1)z^{-1} + (g^2 - gk + 1)z^{-2}} \tag{4}$$

$$H_{\texttt{BP}}(z) = \frac{g - gz^{-2}}{(g^2 + gk + 1) + 2(g^2 - 1)z^{-1} + (g^2 - gk + 1)z^{-2}} \tag{5}$$

$$H_{\texttt{HP}}(z) = \frac{1 - 2z^{-1} + z^{-2}}{(g^2 + gk + 1) + 2(g^2 - 1)z^{-1} + (g^2 - gk + 1)z^{-2}} \tag{6}$$

Here, $g$ is the warped frequency, given by $g = \tan\left(\frac{\pi f}{f_s}\right)$, and $k = 1/Q$. Since we wish to parameterize our lowpass filter by its desired peak frequency $f_p$, not its desired cutoff $f_c$, our first order of business is to derive an expression for $f_c$ as a function of $f_p$ and $Q$.

This can be done entirely in the analog domain, without reference to the digital transfer functions. The most straightforward strategy is to compute the real-valued maximum of the filter's amplitude response via simple calculus. To find the resonant frequency of a lowpass filter (eq. (1)), we first calculate the squared magnitude response:

$$|H_{\text{lp}}(j\omega)|^2 = \frac{\omega_0^4}{(\omega_0^2 - \omega^2)^2 + (\omega\omega_0/Q)^2} \tag{7}$$

To find the real-valued maximum of $|H(j\omega)|^2$, we differentiate with respect to $\omega$:

$$\frac{d}{d\omega}|H_{\texttt{LP}}(j\omega)|^2 = -2\frac{(2Q^2\omega^2 - 2Q^2\omega_0^2 + \omega_0^2)Q^2\omega\omega_0^4}{(Q^2\omega^4 - 2Q^2\omega^2\omega_0^2 + Q^2\omega_0^4 + \omega^2\omega_0^2)^2} \tag{8}$$

[25] https://ccrma.stanford.edu/~jos/fp/Digitizing_Analog_Filters_Bilinear.html

We then solve $\frac{d}{d\omega}|H_{\text{LP}}(j\omega)|^2 = 0$, which yields the peak frequency:

$$\omega_p = \omega_0 \sqrt{1 - \frac{1}{2Q^2}} \tag{9}$$

for $Q > 1/\sqrt{2}$. Clearly, $\lim_{Q\to\infty} \omega_p = \omega_0$, which is congruent with the qualitative observations about Figure 8 in section 4.3.

Armed with the peak frequency in terms of $\omega_0$, finding the peak real-valued is rather simple: we simply need to evaluate the transfer function at $\omega_p$. Armed with eq. (9), some straightforward algebra suffices to show that

$$A_p = |H_{\text{LP}}(\omega_p)| = \frac{2Q^2}{\sqrt{4Q^2 - 1}} \tag{10}$$

#### 4.4.2 Resonant Frequencies of Other Filters

The loop filter in Resonarium can also be configured as a bandpass or highpass filter. As discussed, the cutoff and resonant frequencies of a bandpass filter are identical, so no additional labor is required here. To compute the resonant frequency of the highpass filter, we successfully apply the same techniques as in the prior section. We omit the derivation for brevity.

### 4.5 Calculating the Delay Line Length

In the Resonarium synthesizer, eq. (9) and eq. (10) are both used to ensure that the resonant peak of the loop filter equals the resonant frequency of the waveguide loop itself. However, the phase delay of a second order filter is generally non-linear; therefore, our loop filter affects tuning by effectively increasing the length of the delay line by some function of the fundamental frequency.

Let us suppose that the loop filter has a resonant frequency of $f_p$ and induces a phase delay of $d$ samples at that point. As per section 4.3.1, the resonant frequency of the waveguide loop must match the resonant frequency of the filter. Naively, we would tune the waveguide loop to frequency $f$ by setting $N = f_s/f$; however, since the filter induces a delay of $d$ samples at $f$, we set $N = f_s/f - d$.

We observed that setting $d$ to the equal phase delay [41][26] of the digital transfer function (eq. (4)) at $f$ is sufficient, and facilitates accurate tuning of the waveguide loop.

### 4.6 The Dispersion Filter

Real-world strings often demonstrate nonzero *stiffness*. It can be shown that high-frequency waves travel faster through a medium than low-frequency ones. [42][27] This phenomenon induces *inharmonicity* within the waveguide, and its

[26] https://ccrma.stanford.edu/~jos/filters06/Phase_Delay.html
[27] https://ccrma.stanford.edu/~jos/pasp/Dispersive_1D_Wave_Equation.html

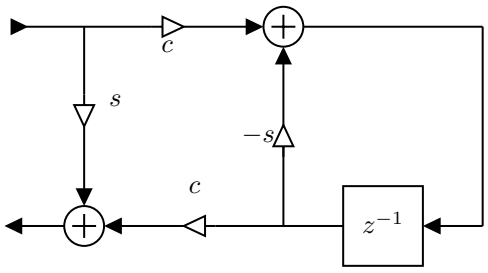



Figure 9: A first order normalized ladder allpass filter, where $c = \cos\theta$ and $s = \sin\theta$. This block diagram corresponds to $H_d(z)$ in Figure 6. Adapted from Figure 2 of [39].

effect on the timbre of a stringed instrument is non-negligible. Therefore, waveguide models that accurately model stiffness must induce non-linear phase delay within the waveguide loop. Allpass filters, which affect phase whilst maintaining a unity-gain amplitude response, are often used for this purpose. The inharmonic sound produced in this way can be often described as "metallic" or "bell-like".

Much has been written about the use of allpass filters in waveguide modeling [39, 2, 1, 30, 29, 20, 21]. When dispersion is utilized for the accurate reproduction of real-world instruments, it is often applied judiciously, and its effects on timbre are restrained[28]. It is often used to model the delicate and subtle inharmonic quality of stringed instruments such as the piano. By contrast, this work is not motivated by a desire to perfectly replicate a specific physical instrument. Hence, we seek to encourage musical experimentation by furnishing Resonarium with a powerful and exagerated dispersion parameter that radically alters the timbre of the parent waveguide.

Several different allpass filter designs were experimented with early in development. We found that the *first-order normalized ladder allpass filter* of [39] profitably induces a strong inharmonic effect within the waveguide model, with minimal computational expenditure: only a single unit delay is required.

Figure 9 shows a block diagram of a nonlinear first order ladder allpass filter, which corresponds to $H_d(z)$ in Figure 6. Let us examine two specific cases: $\theta = 0$ and $\theta = -\frac{\pi}{2}$. For $\theta = 0$, we have $c = 1$ and $s = 0$, for which $H_d(z)$ reduces to:

$$H_d(z) = z^{-1} \tag{11}$$

This represents a unit delay, which is effectively a no-op for our purposes. When $\theta = -\frac{\pi}{2}$, we obtain $c = 0$ and $s = -1$, reducing $H_d(z)$ to:

$$H_d(z) = -1 \tag{12}$$

[28] Supplementary audio recordings germane to [30] can be found at http://legacy.spa.aalto.fi/demos/ext-disp. These recordings demonstrate the effect of such tunable allpass filters on piano models. The effect is subtle, yet distinctive.

In this case, the polarity of the signal is reversed, but its amplitude and phase remain otherwise unchanged. In a standard linear signal chain, such inversion has no direct effect on the audible timbre of the signal. However, when placed inside the main feedback loop of our waveguide model, the resultant signal is is instead composed of *odd* harmonics only.

One of way of understanding this is to consider the waveguide loop as a feedback comb filter with extra steps. Indeed, the feedback comb filter (with a feedback coefficient of 1) is practically identical to Alice's simplified string model in Figure 2. The frequency respones, respectively, of a positive feedback comb filter $H_+(z)$, and a negative feedback comb filter $H_-(z)$, are:

$$H_+(e^{j\omega}) = \frac{1}{1 - e^{-j\omega N}} \qquad H_-(j\omega) = \frac{1}{1 + e^{-j\omega N}} \tag{13}$$

We are interested in the iconic "spikes" in the frequency response that lend the comb filter its name. Clearly, these occur when the transfer functions' respective denominators approach zero, hence, it suffices to solve

$$1 - e^{-j\omega N} = 0$$

for the positive feedback case, and

$$1 + e^{-j\omega N} = 0$$

for the negative feedback case.

Observe that $e^{-j\omega N} = 1$ when $\omega N = 2\pi k$ for $k \in \mathbb{Z}$, Hence, the positive feedback comb filter induces resonances at $\omega = \frac{2\pi k}{N}$. For the negative feedback comb filter, we need to show when $e^{-j\omega N} = -1$. This only occurs for $\omega N = \pi(2k+1)$, and therefore resonances occur when $\omega = \frac{2\pi k}{N}$. In practice, this means that negative feedback comb filters produce *odd* harmonics. Furthermore, by varying $\theta$ between 0 and $-\frac{\pi}{2}$, we achieve "smooth" interpolation between the positive and negative comb filter behaviors. Figure 10 displays this property in action via a recorded spectrogram. The lower valleys correspond to $\theta \approx -\frac{\pi}{2}$, while the higher peaks occur when $\theta \approx 0$. In Resonarium, zero dispersion maps to $\theta = 0$, while "maximum" dispersion maps to $\theta = -\frac{\pi}{2}$. Modulation of this parameter can effectively synthesize bells and metallic sounds, as well as woodwinds and other tube instruments.

We express some dissatisfaction with the current implementation of the dispersion filter. For instance, we are, in a sense "cheating": when dispersion is set to maximum, the phase response of the dispersion filter is zero! The "inharmonic" auditory effect is, in part, an illusory epiphenomenon of the smooth interpolation visible in Figure 10. This filter was one of the earliest implemented features of this work, and therefore a second pass over the subject is well overdue. During those halcyon times, we were entirely satiated by the ability to smoothly transition towards an odd-harmonic spectrum (and this sentiment, to some extent, lingers). To improve the physical fidelity of our implementation, we would like to further explore the "tuned" allpass designs of [30, 29]. We wish

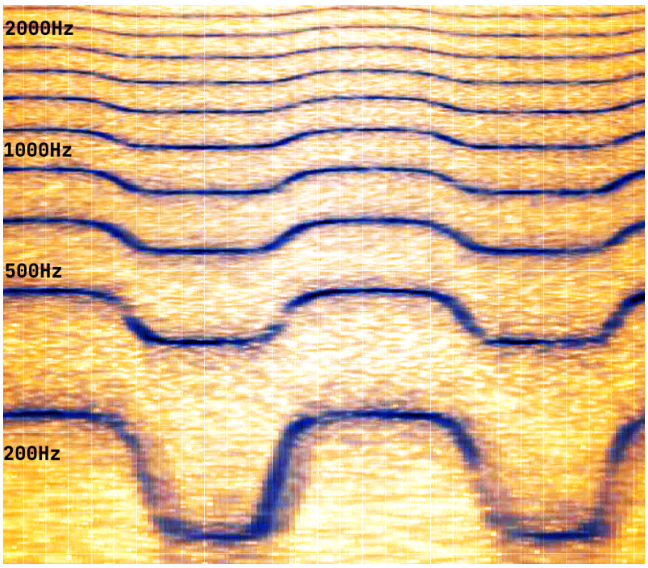



Figure 10: A single resonator (Figure 6) is tuned to 220Hz and excited with white noise, while $\theta$ is sinusoidally modulated between 0 and $-\frac{\pi}{2}$.

to faciliate easy tuning of presets with Resonarium; hence, an "ideal" dispersion filter would leave the fundamental frequency of each note untouched[29], whilst squeezing, stretching, or otherwise distorting the higher harmonics. We hope to research this further in the near future.

## 4.7 The Post Filter

We now discuss the *post filter*, denoted as $H_m(z)$ in Figure 6. The post filter is a *multi-mode state variable filter* that is located *after* the resonator loop (hence the name). The post filter serves two important purposes in this work: it can be used to further sculpt the sound produced by its preceding resonator, or it can faithfully implement *modal synthesis*.

The post filter is implemented with the same state-variable topology [33, 35, 34] as the loop filter of section 4.3.1. Unlike the loop filter, the post filter takes advantage of the multi-mode capability of the state variable topology. This is exposed via a *mode* parameter, through which the filter can be smoothly interpolated between lowpass, bandpass, and highpass configurations. Internally, this is implemented via selective mixing of the filter's lowpass, bandpass, and highpass output signals.

[29] In practice, this need not strictly be the case: if it is computationally efficient to compute the phase delay of the allpass filter at the waveguide loop's natural eigenfrequency, then the loop delay line can be shortened appropropriately to correct the tuning error. We apply this technique to the loop filter in section 4.4.

### 4.7.1 The Post Filter as a Sound Design Tool

The most obvious way to harness the post filter is to simply use it as intended. A skilled sound designer can tease an incredible amount of mileage out of a simple oscillator and a well-designed filter. A particularly fruitful technique is to use the resonator loop as a *wavetable* by setting the decay multiplier to its maximum value, bypassing it entirely. The resonator will then produce sound indefinitely[30], which can be sculpted in real time by the post filter modulated by user input (e.g. MPE pressure), or an envelope/LFO. This strategy can be fruitfully applied to the synthesis of sustained or bowed instruments, such as woodwinds or pipes, or towards abstract pads and ambient drones.

### 4.7.2 The Post Filter as a Modal Resonance Filter

The post filter can also be used as a resonant filter for *modal synthesis*[42][31]. In modal synthesis, complex vibrating structures are represented as a collection of simple resonant modes. Such a collection is implemented via a parallel bank of resonant bandpass filters, each of which corresponds to a single resonant mode of the physical system. Notably, there are no feedback loops in modal synthesis: an harmonically-rich excitation signal is sent into the parallel filter bank, the filters amplify their respective peak frequencies while attenuating all others, and the resultant signal is routed to the output.

In Resonarium, the entire waveguide loop of Figure 6 can be effectively disabled by reducing the decay multiplier to zero. By transforming the feedback loop into a no-op, the excitation signal is sent directly to the post filter, which can be configured in a resonant bandpass state. Furthermore, each post filter can be tuned to a specific harmonic offset (with respect to MIDI input), which facilitates the simulation of complex resonating bodies with distinct resonance modes, such as bells, bars, and bowls.

## 4.8 The Multi-Mode SVF as a Loop Filter

While the post filter can interpolate between lowpass, bandpass, and highpass configurations, the loop filter omits such flexibility: it can only snap discretely between these three distinct configuration states.

In an early version of this software, both the loop filter and the post filter were implemented as multi-mode filters. Recall that one notable feature of Resonarium is its ability to interpolate, in a sense, between Karplus-Strong synthesis [19, 37] and banded waveguide synthesis [12]. This is implemented through careful interpolation of the loop filter's coefficients, from a lowpass/highpass

[30] Unfortunately, this is not strictly true in practice, despite being theoretically accurate. The third order Lagrange interpolation used in the fractional delay line induces a mild attenuating effect, which becomes noticeable over time. Therefore, the resonator may need to be "refreshed" from time to time via excitation. We would like to research this attenuation in the future, and mitigate it via careful gain compensation or different interpolation techniques.

[31] https://ccrma.stanford.edu/~jos/pasp/Modal_Representation.html

state to a bandpass state. Armed with the multi-mode capabilties of the state variable filter, there are two ways to accomplish this functionality:

1. We can implement the loop filter as a multi-mode state variable filter (which, in practice, is a parallel bank of three distinct second-order filters hiding under a trench coat), and interpolate the mode from lowpass to bandpass by mixing the lowpass and bandpass outputs appropiately.

2. We can implement the loop filter as a lowpass filter, and increase the resonance until the filter is practically indistinguishable from a bandpass filter.

It was observed that both approaches are acoustically indistinguishable in a waveguide context. Furthermore, the filters in both cases must be tuned with respect to peak frequency, as demonstrated in section 4.4.

However, this calculation is an order of magnitude easier in the second case, where the loop filter is a single lowpass filter instead of three parallel second order filters. Furthermore, while applying the techniques of section 4.4 to the multi-mode filter is mathematically feasible, the resulting formulae suffer from numerical instability at extreme values, rendering the technique unworkable in practice.

To see this, recall that a multi-mode filter can be expressed as a linear combination of its second-order lowpass, bandpass, and highpass outputs. In the analog domain, these outputs are given by eq. (1), eq. (2), and eq. (3), respectively. Each output is scaled by one of the mixing coefficients $m_{\texttt{LP}}, m_{\texttt{BP}}, m_{\texttt{HP}} \in [0, 1]$, before being summed to produce the final result. In practice, $m_{\texttt{LP}}$ and $m_{\texttt{HP}}$ are mutually exclusive: one of the two will always be zero, as the multi-mode filter interpolates between lowpass and bandpass, or between bandpass and highpass.

Hence, the transfer function of the multi-mode filter can be split into two distinct cases. We will discuss the lowpass-bandpass case, as the bandpass-highpass case proceeds similarly. The transfer function of such a filter, in the analog domain, is:

$$H_{\texttt{LP-BP}}(s) = m_{\texttt{LP}} H_{\texttt{LP}}(s) + m_{\texttt{BP}} H_{\texttt{BP}}(s) \tag{14}$$

$$= \frac{m_{\texttt{LP}}\omega_0^2}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2} + \frac{m_{\texttt{BP}}\frac{\omega_0}{Q}s}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2} \tag{15}$$

$$= \frac{m_{\texttt{LP}}\omega_0^2 + m_{\texttt{BP}}\frac{\omega_0}{Q}s}{s^2 + \frac{\omega_0}{Q}s + \omega_0^2} \tag{16}$$

As in section 4.4, we differentiate with respect to $\omega$ and solve for zero to find the peak frequency $\omega_p$ of the combined filter:

$$\omega_p = \frac{\omega_0}{m_{\texttt{BP}}} \sqrt{-m_{\texttt{LP}}^2 + \frac{\sqrt{-m_{\texttt{LP}}^2 m_{\texttt{BP}}^2 + (m_{\texttt{LP}}^2 + m_{\texttt{BP}}^2)^2 Q^2}}{Q}} \tag{17}$$

Consequently, the peak gain at that frequency is given by

$$A_p = m_{\texttt{BP}}^2 Q \sqrt{\frac{1}{m_{\texttt{BP}}^2(1-2Q^2) + 2Q(-m_{\texttt{LP}}^2 Q + \sqrt{-m_{\texttt{LP}}^2 m_{\texttt{BP}}^2 + (m_{\texttt{LP}}^2 + m_{\texttt{BP}}^2)^2 Q^2})}} \tag{18}$$

These equations are rather cumbersome; indeed, the algebraic derivation is straightforward, but remarkably ponderous, and has been omitted here for brevity. The author emphatically recommends the use a computer algebra system. Furthermore, our implementation was hampered by numerical instability for mixing coefficients close to zero or one. For such extrema, tiny values in the square roots can erroneously flip negative due to floating point error—which, naturally, blows up the square root function. There are techniques for working with arbitrarily precise floating point numbers; however, simplicity and performance are paramount in this application, since both the peak gain and frequency are computed many times per audio block. As the multi-mode filter is acoustically indistinguishable form a resonant lowpass/highpass filter in our waveguide context, and an order of magnitude more frustrating to work with, it was abandoned in favor of the latter. We would like to return to this topic in the future, however, and further explore the behavior of parallel filter banks in waveguide loops.

# 5 The Resonator Banks

The resonators of the previous section (Figure 5) do not exist in isolation. Observe that eight[32] such structures are gathered together into a single *resonator bank*. By default, this categorization is acoustically inconsequential: the resonators operate in parallel, and there is no interaction between them.

However, a resonator bank can be parameterized by a *coupling mode*. In stringed instruments, for example, vibrations from each active string travel through the bridge and/or body of the instruments to the rest the strings. Such sympathetic vibration is subtle, but must be considered carefully for in order to accurately model the instrument in question. To implement this in a digital waveguide model, the output from each generalized string model is tapped, processed in some way, and then mixed into the exciter signals of the other strings in the parent instrument model. Germane literature suggests that the coupling signal should be strongly attenuated [45], and many implementations generally obey this principle [50][33].

However, this work is not motivated by a monomaniacal desire to faithfully model a specific instrument or family of instruments. Therefore, our interest

[32]This particular number was motivated by aesthetic constraints: eight columns fit neatly into the UI. Further, we mathematicians and computer scientists are inexplicably drawn to powers of two, like moths to flame or vultures to carrion. Four was too few, and sixteen would clearly be too many.

[33]In particular, we draw attention to the coupling technique implemented in the guitar model, `Guitar.h/cpp`. The gain of the coupling signal is extremely low compared to the feedback signal itself.

in a physically accurate bridge or body simulation is restrained; rather, we wish to procure a dramatic and exaggerated set of coupling topologies that radically and unpredictably re-imagine the timbre of the resonators. While we are eminently interested in developing coupling modes that accurately emulate physical referents, we do not wish to restrict the relevant parameter space to such subspaces of creative expression.

As of writing, three distinct coupling modes have been implemented: *parallel*, *interlinked*, and *cascade*. It was observed that these coupling modes fundamentally reshape the timbre of the resonators' output——moreso than any other exposed parameter. Indeed, many of the most distinctive timbres that can be synthesized by this work are exclusively realized through the manipulation of coupling topologies.

## 5.1 The Parallel Coupling Mode

The *parallel* coupling mode is the simplest of the bunch, and is defined by a total omission of any intra-resonator coupling whatsoever. We dedicate an entire section to this topic entirely in the interest of epistemological comprehensiveness. A diagram of the parallel topology, as implemented in this work, is depicted in Figure 11. Each $R_i$ block corresponds to an instance of Figure 6. Furthermore, the relative gain of each resonator can be adjusted via the corresponding multiplier $g_i$.

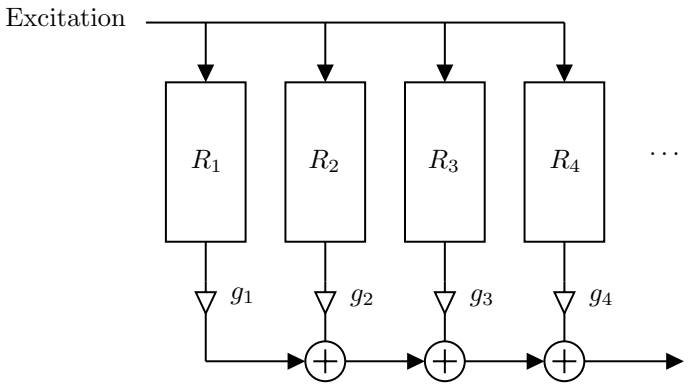



Figure 11: A diagram of the *parallel* coupling mode in this work. Only four resonators are depicted, but the eight-resonator structure implemented in practice can be trivially extrapolated.

## 5.2 The Interlinked Coupling Mode

The *interlinked* coupling mode directly connects the resonators to each other by a simple bridge filter [42]. This coupling mode produces rich, metallic, and inharmonic timbres, yielding fertile ground for exploration and experimentation. Its effect is dramatic and ineluctable.

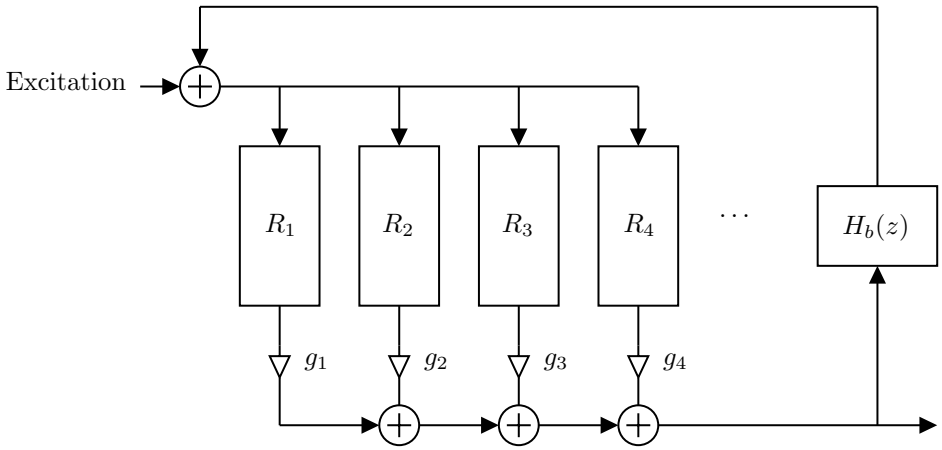



Figure 12: A diagram of the *interlinked* coupling mode, depicted with four connected resonators.

There is nothing distinct about the topology of Figure 12, compared to the extant literature. It is simply a generalization of the standard string coupling (as seen in, e.g., [42][34]). Most coupled string models have a feedback line that aggregates the outputs of all strings, passes the combined signal through a filter, and replicates it across the inputs of all strings. The difference lies in the filter implementation.

Many extant physical models apply strong attenuation to the coupling signal. The guitar model of the STK [50][35] applies a one-pole low-pass filter to the signal, before multiplying it by a factor of approximately 0.01. As one might imagine, the severity of this attenuation relegates the coupling signal to a minor role in the timbral characteristics of the resultant model. However, these subtle flourishes of realism are necessary to cross the "last mile" in accurate physical modeling: we cannot escape the uncanny valley of synthesized instruments without their aid.

A significant span of time was spent in search of a *dramatic* coupling filter that would radically distort the sound in a musically interesting way. These results were met with limited success. Very early in development, the author accidentally stumbled upon the bridge filter

$$H_b(z) = \frac{-2}{G} \tag{19}$$

where $G$ equals the sum of the output magnitudes of all active resonators. It turns out that this configuration is well-known, and corresponds to a lossless junction of several strings [42]. When enabled, this coupling filter induces a distinctly metallic timbre with inharmonic spacing between resonant spikes,

[34] https://ccrma.stanford.edu/~jos/pasp/Two_Ideal_Strings_Coupled.html
[35] The relevant code can be found in `Guitar.h`, `Guitar.cpp`, `Twang.h`, and `Twang.cpp`.

whose fundamental frequency does not strictly equal any eigenfrequency of any resonator.

This bridge filter constant, though simple, is delicate: if increased slightly, the system of resonators becomes immediately unstable; if decreased slightly, the signal within the resonators is attenuated too quickly and the result is not particularly interesting. This is a immediate consequence of its physics-based justification. Furthermore, the system relies on perfect phase cancellation between the coupling signal and each resonator feedback signal. Under this condition, it is not difficult to show that, for each resonator $R_i$, the magnitude of its feedback signal plus the coupling signal remains bounded between $-1$ and 1 (assuming both input signals are similarly restricted). A similar analysis can be conducted from a physics perspective to show that the bridge losslessly conserves energy.

This perfect phase cancellation is a necessary condition for model stability. If a delay is added to the coupling filter that is *not* strictly compensated for by a corresponding delay in each resonator's feedback loop, then a phase shift occurs between the coupling signal and the feedback signals. This results in imperfect destructive interference, and thus entire system becomes unstable.

### 5.2.1 Interpolating Coupling Modes

This is problematic, as we would like to implement a notion of *interpolation* between a parallel configuration and a coupled configuration. Ideally, it would be possible to smoothly "fade in" the coupling effect by adjusting some parameter associated with the coupling filter, such that the minimum value of the parameter induces an effect roughly equivalent to setting $H_b(z) = 0$.

Much time was invested in pursuit of this objective, with limited success. The obvious solution—real-time linear interpolation of $H_b(z)$ between 0 and $-2/G$—is not as musically interesting we had hoped. While the parallel configuration punches through as $H_b(z)$ approaches zero, and the distinctive *interlinked* timbre is audible as $H_b(z)$ approaches $-2/G$, the interstitia are excessively damped, and generally dominated by harsh, high-frequency resonances.

We experimented with a lowpass or highpass filter injected into the coupling signal path. By manipulating the filter's cutoff frequency, the user could, in principle, modulate the spectral content from complete attenuation to full transmission. This would provide a filter-based notion of "coupling intensity". The phase shift problem, however, immediately precludes the use of many filters with nonlinear phase response. Indeed, if an off-the-shelf second order filter is placed in the path of the coupling signal, then the resonator bank immediately descends into instability[36], despite the filter's unity gain characteristic. It is possible to employ a linear-phase FIR filter of order $\mathcal{O}$ in the coupling signal chain by manually adding a $\mathcal{O}/2$-sample delay to each resonator's feedback

[36]Even a single unit-sample delay, placed in the coupling signal flow, induces catastrophic instability within the coupled waveguide model. This is somewhat counterintuitive from a DSP perspective, since an isolated unit delay should not, in principle, modify the spectral characteristics of any sound.

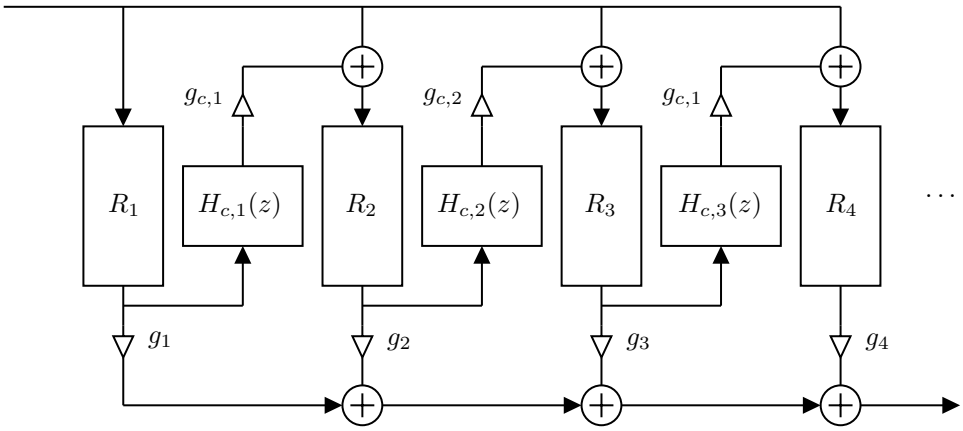



Figure 13: A diagram of the *cascade* coupling mode, depicted with four connected resonators.

line; however, this approach is expensive and cumbersome; furthermore, the sound produced is disappointingly similar to that of the interpolated approach described above.

Nevertheless, there is plenty of extant research in this domain [42][37]. We intend to implement a more sophisticated coupling filter, using a mathematically sound approach, in the near future.

## 5.3 The Cascade Coupling Mode

In the *cascade* coupling mode, the output signal from resonator $R_i$ is directed into the input of resonator $R_{i+1}$. Furthermore, a multi-mode state variable filter ([33], section 4.7) is instantiated between each pair of resonators, as depicted in Figure 13[38]. We are particularly pleased with this design. It can produce striking "generative" ambient patches, as resonant epiphenomena move slowly through the series of linked resonators. Cascade coupling is particularly suitable for experimentation and unstructured play. Rapid modulation of the cascade filters' parameters can induce unpredictable and novel timbres.

This coupling mode cannot induce instability: unlike the interlinked coupling mode, cascade coupling does not construct additional feedback loops between sets of resonators. However, successive resonators tuned to harmonically related frequencies tend to amplify resonances passing through their respective internal organs, which can produce dangerously loud—if stable—sinusoidal waveforms.

[37] https://ccrma.stanford.edu/~jos/pasp/Positive_Real_Functions.html

[38] In Figure 13, each instance of the cascade filter is distinct; hence, one may reasonably assume that the parameters of each filter are distinct as well. This is not the case as of writing; in Resonarium, all eight cascade filters share the same cutoff, resonance, and LP-BP-HP interpolation mode (though these three values are exposed to user modulation). This is primarily due to UI space constraints, and to temper what might be otherwise be an intractable and byzantine user interface.

There is no automated technique to mitigate this phenomenon. We therefore advise the user to approach the cascade coupling mode with at least a scintilla of trepidation: while the topology produces rich and evocative textures, soundscapes, and resonances, it can (and will) blow up a pair of speakers with abandon if given the headroom.

We intend to research the amplitude response of series-coupled waveguides further in the future, with an intent to develop an automated gain compensation system that would mitigate, if not nullify, these amplitude spikes. However, we prefer to offer the musician this creative option, whilst notifying her of the appropriate caveats. In general, we endeavour to expose unstable or undesirable configuration states, if such exposure also opens up vistas of musical exploration that would otherwise be inaccessible. Additional discussion on the philosophy of this work can be found in section 10.

### 5.4 Future Research on Coupling Modes

Though execution of this work, we have observed that waveguide coupling exposes unique timbres that we do not know how to otherwise synthesize. Coupled models induce epiphenomena which would be fundamentally inaccessible otherwise; without coupling, many of our most musically interesting presets would not exist. We wish to research coupling in two distinct directions:

- *towards realism*, in which we implement physically accurate bridge models that enhance Resonarium's capability to faithfully recreate the timbres of physical instruments, and
- *towards surrealism*, in which we explore coupling modes that expose musically unique timbres, without regard to any physical referent or referents.

For the first objective, we intend to follow in the footsteps of [45, 42][39], among other open-source implementations of such instruments. The second is more or less untrodden ground, and we are excited to see what future experimentation may unearth.

## 6 The Exciter Modules

A physical instrument doesn't do much on its own: unmolested, it will sit menacingly in a dusty corner of the room and do nothing. To produce sound, a transfer of energy from the musician to the instrument must occur: it must be plucked, struck, bowed, or otherwise physically manipulated.

Resonarium's resonator banks—which we define further in the next section of this document—also require some sort of energetic excitation to do anything musically interesting. To that end, Resonarium is furnished with several *exciter modules*, arranged within the leftmost column of the user interface. These are

[39] https://ccrma.stanford.edu/~jos/pasp/Positive_Real_Functions.html

simple oscillators that produce various types of mostly inharmonic noise: clicks, pops, noise bursts, static, and so on.

The envelope of the resonators' musical response directly corresponds to the temporal shape of the exciter signal: a sharp, impulsive excitation is reciprocated by a plucked sound, while a prolonged excitation (e.g. white noise) induces a bowed or rubbed effect.

Furthermore, each resonator is tuned to a specific frequency, rendering it particularly responsive to excitation signals with similar spectral material. When the exciter signal spectrally aligns with the resonator's eigenfrequencies, the resonator will produce a louder, resonant tone. Conversely, when the spectral composition of the excitation is dissimilar to the resonator's eigenfrequencies, the resonator's response will be modest and restrained.

In this way the excitation signal strongly influences the synthesizer's final musical product.

## 6.1 The Impulse Exciter


The *impulse exciter* is the simplest of the exciter modules: when a note is triggered, an impulse is generated and used to excite the resonators[40]. The *density* parameter generalizes the impulse to a length-$\ell$ *normalized rectangular pulse*, defined as

$$\delta_\ell[n] = \begin{cases} \frac{1}{\ell}, & \text{if } 0 \leq n < \ell \\ 0, & \text{otherwise.} \end{cases}$$

Increasing the density parameter induces a pleasant low-pass effect within the resonators. Furthermore, the impulse exciter is equipped with a standard second order filter, which enables additional sculpting of the impulse signal.

In the context of physical modeling, an impulse excitation often corresponds to plucking a string, or striking an object quickly. A low-pass filter applied to the impulse can emulate a "soft" striker, i.e. a mallet wrapped in felt.

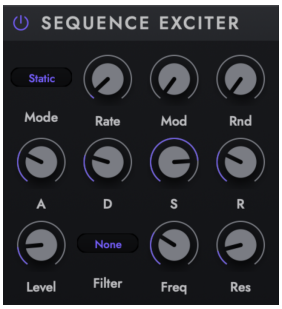
## 6.2 The Noise Exciter

The *noise exciter* does, more or less, exactly what one would it expect it to do. When activated, it generates white noise that can be sculpted with a multi-

[40] A digital *impulse* is a signal $\delta[n]$ where $\delta[n] = \begin{cases} 1, & \text{if } n = 0 \\ 0, & \text{if } n \neq 0 \end{cases}$


mode filter. Furthermore, the noise exciter is furnished with a built-in ADSR envelope. The noise exciter is effective for ambient "bowed" instruments and sustained pads.

## 6.3 The Impulse Train Exciter




The *impulse train exciter* synthesizes a continuous train of impulses at a parameterized rate, with respect to a built-in ADSR envelope. This exciter facilitates experimental and creative patches, such as metallic foley, rich and organic pads, or perhaps the sound of evening rainfall on a tin roof. Furthermore, the *rate* at which pulses are generated can be synchronized with the host tempo, which is effective for rhythmic musical sequences. This exciter can be configured with respect to of a handful of *modes*, which all operate distinctly.

- The *pulse* mode generates a sequence of one-sample impulses at the specified rate. Sample interpolation has not yet been implemented; hence, the pulse impulse train suffers from quantization error at higher frequencies. In practice, the rate parameter is bounded below a value at which audible symptoms of quantization error would manifest. We intend to implement interpolation in a future update.

- The *static* mode generates impulses randomly over time. At each time step, the exciter may or may not generate an impulse with random amplitude (the probability, of course, is mapped to an exposed parameter). This high entropy excitation signal is particularly useful for generative patches and experimental ambiences.

## 6.4 The Sampler Exciter

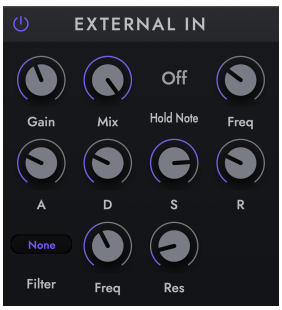

The *sampler exciter* is more or less self-explanatory. The user can drag and drop an audio file into the sample exciter panel, and the synthesizer will mix the sample into the exciter signal. The sampler is furnished with an analog-modeled ADSR envelope and a multi-purpose filter.

The sampler exciter blurs the divide between this work as an *instrument* and this work as an *audio effect*. From this perspective, Resonarium can be viewed as a sort of vocoder, capable of unrecognizably mutating existing audio signals. Applying "light resonance" to a sample can induce a shimmery, metallic delay, while other configurations can irreducibly transform an uninteresting and inharmonic sample into a glittering expanse of color and light.

Furthermore, samples often contain more harmonic intrigue than the white noise bursts generated by Resonarium's algorithmic exciters. Complex excitation signals are indispensable in the creation of organic, natural instrument presets.

The sample exciter remains a work in progress. Several improvements, such as loop selection, crossfading, pitch-shifting, and a responsive waveform UI are in development.

## 6.5 The External Input Exciter



If the sample exciter can be said to blur any distinction between an instrument and an audio effect, then the *external input exciter* dispenses with pleasantries and pulverizes it entirely. The external input exciter does not generate a signal; rather, it routes live input from any source into the resonator banks as an excitation signal.

This source is generally configured within the host software. With the appropriate routing topology, the user can direct MIDI into Resonarium, while audio is simultaneously streaming into the external input exciter. This audio can be from another synthesizer, a sampler, or a live input such as a microphone or a physical instrument.

Therefore, Resonarium can be invoked not as an instrument, but as an audio effect, much like a vocoder or a granulator. From this perspective, the externally-sourced exciter signal can be considered a "carrier", whose timbral characteristics are transformed by Resonarium's resonators. When processed, the carrier undergoes an unrecognizable metamorphosis: a dry drum recording can be transformed a shimmering choir of bells and chimes, or one instrument can be mutated into another. Since the external input exciter processes audio in real time, Resonarium can be used as an audio effect or pedal in a live performance.

The external input exciter is furnished with a carrier amplitude envelope that cannot be disabled. This is necessary, as Resonarium only terminates a voice once it detects silence: unless an envelope modulator is enabled and attached to a parameter, releasing a MIDI note has no effect on the corresponding voice's resonators. If the carrier signal does not contain sufficiently long periods of silence, Resonarium will not terminate its voices, and the synthesizer will generate audio indefinitely, long after the corresponding MIDI notes have been released.

Most DAWs support arbitrary MIDI routing. Since Resonarium was first developed as an MPE instrument, the software will not produce audio unless a MIDI note is active. Therefore, MIDI information must be channeled into Resonarium, even when used as a pure audio effect. This procedure varies by DAW, and can be somewhat cumbersome. To mitigate this, the External Input Exciter can generate a virtual MIDI note, which is always active regardless of the state of the host software. This allows Resonarium to be used as a pure audio effect without any convoluted MIDI routing.

# 7 Automation and Modulation

Resonarium is furnished with a comprehensive and semi-modular modulation system. As mentioned in the introduction, the state of the physical modeling ecosystem, with particular respect to polyphonic modulation, is unsatisfactory. This work, therefore, is strongly motivated by a desire to explore physical modeling principles in a semi-modular environment.

The modulation system of Resonarium has one trick up its proverbial sleeve, which we illuminate in section 7.2. Otherwise, the modulation system in this


Figure 14: Several modulation sources in action.

work is not particularly unique: in fact, all sufficiently advanced modulation systems work in the same way, and the notion of modulating parameters is neither novel nor perplexing.

Such systems are generally composed of three (collections of) components: modulation sources, modulation destinations, and connections between the two. Modulation sources, or modulators, broadcast a control signal that can be derived either from an algorithmic source, such as an LFO or envelope, or a MIDI source, such as pitch bend or MPE pressure. Modulation destinations are generally user-controllable parameters that control the instrument's internal state: pitch, distortion, and so on.

A *connection* is established by linking a modulation source to a modulation destination. Each connection is parameterized by some nonzero *modulation depth*, which tells the modulation system how strongly the modulation signal should affect the destination parameter. Many modulation systems recursively support the modulation of modulation depth itself. In such instances, the modulation depth of each active connection is itself a modulation destination, which can be modulated by another modulation connection. Such recursive composition facilitates the creation of complex and dynamic modulation patterns.

Resonarium implements four distinct types of algorithmic modulators. There are four instances of each of these types, for a total of sixteen algorithmic modulation sources. In particular:

- The **ADSR modulator** is a simple ADSR envelope that is polyphonically triggered at the start of a MIDI event. The curvature of each segment is nonlinear, and inspired by the exponential curves of analog envelopes. In the future, we plan to allow user modulation of the curvature of each segment.

- The **LFO modulator** generates a control signal from one of several distinct waveforms. Though the waveforms themselves are hard-coded, parameters such as depth and frequency are exposed to the user.

- The **random modulator** generates a random signal, which is visible in

the oscilloscope to the right. By default, the signal is a random step function whose rate is controlled by the corresponding rate parameter. The *chaos* parameter controls how closely the next random sample is correlated with the previous random sample, while the *smooth* parameter applies smooth interpolation between successive pairs of samples.

- The **multi-segment envelope generator (MSEG)** is a generalized envelope/LFO with a customizable waveform. The user defines waveforms by inserting or removing segments from a grid, and adjusting the curvature between consecutive pairs of points. We are particularly pleased with the MSEG and the creative expression it catalyzes. In a future update, we will implement a *curve browser*, where the user can load and save MSEG curve presets. A curve browser will obviate both the ADSR and LFO modules entirely, since presets matching those waveforms can simply be shipped as MSEG curves.

## 7.1 Polyphonic Modulation

Resonarium is a *polyphonic* instrument: multiple notes can be played at the same time. This is internally implemented via a pool of *voices*: each time a MIDI note is triggered, a voice instance is initialized and engaged. A voice is responsible for generating all the audio corresponding to a specific note event. When the note is no longer audible, the voice is disabled, reset, and recycled.

Since each note has its own internal state and progression, it is often desirable to modulate each note individually, on a per-note basis. For instance, if an ADSR envelope is attached to a parameter, and a note is triggered, we would like the envelope to modulate the state of that note alone, rather than the respective states of any other active notes. Similarly, if we connect an MPE modulation source to a parameter, we expect a distinct modulation signal to be applied to each note individually, since the MPE protocol supports polyphonic pressure, pitch bend, and timbre. Otherwise, MPE control signals would battle for dominance over a shared global state: if multiple notes are being played at once, which one gets to generate the monophonic MPE control signal for pressure or timbre?


Figure 15: A *random* modulation source, as implemented in this work.

To facilitate this, nearly all Resonarium's parameters are *optionally polyphonic*. A picture of a *random* modulator is depicted in Figure 15. To instan-

tiate a connection between this source and a destination, the user can drag one of the two purple circular arc icons from the source to a desired parameter. The single arc creates a *monophonic* connection, while the double arc creates a *polyphonic* connection. The difference is that the monophonic signal has a global state, shared between all voices, and the polyphonic signal is unique per voice.

It is difficult to create expressive, dynamic patches in *any* synth without a polyphonic modulation system. To our knowledge there is no extant physical modeling instrument with a semi-modular polyphonic modulation system, and we have greatly enjoyed the artistic potential of this feature.

## 7.2 There's More than One of Everything: Stereo Modulation in Resonarium

> *First rule in government spending: why build one when you can have two at twice the price?*
>
> S.R. Hadden

A distinctive feature of Resonarium's modulation system is its stereo modulation system. Each modulation connection contains a left and right control signal channel, and these signals can be generated independently. All modulation destinations apply the left modulation and the right modulation to the left and right channels, respectively, to processed audio.

Most notably, both the resonator and resonator bank parameters are stereo. This is implemented through simple duplication: there are, quite literally, two of everything under the proverbial hood. Each resonator (Figure 6) is actually *two* distinct resonators with independent states: one for the left signal, and one for the right signal. This style of duplication extends to all components within this work.

Since the resonators are deterministic, and modulation sources duplicate their control signals across both channels by default, this feature is effectively invisible until utilized in a patch. Stereo modulation signals can be generated by adjusting the *stereo* parameter of any of the three algorithmic modulation sources of section 7. In the random source, this parameter corresponds to the *correlation* of the left and right control signals: when set to 0%, the signals are identical, and when set to 100%, the signal are entirely independent. For the LFO source, the left and right control signals are always populated by the same waveform, but stereo parameter controls the amount of phase shift between the two channels.

Stereo modulation is remarkably effective in the pursuit of rich, multi-layered generate soundscapes, pads, and experimental patches. Furthermore, the implementation obviates the need for stereo-specific controls in the exciters, resonators, and effect chain, instead delegating this behavior to the modulation system in a modular fashion.

One notable problem with our present implementation is the absence of any visual indicators that might inform the user whether a particular parameter supports stereo modulation. Not all parameters in this work consume a stereo control signal; for some, the notion is contextually nonsensical. These "mono destinations" simply discard one of the two signals. We plan to differentiate these two cases with a visual indicator in a future update.

## 7.3 The Staggering Redundancy of Modulation Systems

We will continue to emphasize *ad nauseum* that all sufficiently capable modulation systems are functionally identical. There are sources, there are destinations, and the user creates connections between the two sets. This raises an interesting question: why spend so much time developing a modulation system at all? Most modern host applications are bundled with advanced modulation tools; furthermore, all audio plugin APIs enforce conformity between their client plugins' exposed parameters. Bitwig Studio[6], for example, is a DAW known for its remarkably complex modulation system. Since plugin parameters are exposed to the host program, it should be possible to delegate the messy business of modulation to the host, saving both development time and UI real estate. Modulation systems are not particularly interesting to develop from scratch, and the process is error-prone and tedious. The UI in particular is a quagmire of animated widgets and edge cases. Common wisdom instructs us to avoid such redundancy if possible, and yet here we are, writing about our custom implementation. What gives?

The problem, simply put, is polyphony: the polyphonic voices in a synthesizer plugin are not exposed to the host. Therefore, the host can only modulate the *global* state of any exposed plugin parameter; though it passes MIDI notes from the system to the plugin, it cannot exploit this privileged information by assigning particular parameter values to particular note events.

An immediate consequence of this observation is that each synthesizer must implement its own polyphonic modulation system. This is prodigiously inefficient; furthermore, it casts a long shadow on the expressive potential of many otherwise excellent software instruments. We have noted several otherwise excellent software synthesizers and audio effects that are hobbled by a fledgling or nonexistent modulation system. Were it possible to pull this functionality out of each plugin, and implement it within higher-level control software such as the plugin host, all extant digital instruments would be imbued with polyphonic modulation "for free".

### 7.3.1 If You're Happy And You Know It...

Clever Audio Plug-in (CLAP) is a nascent audio plugin API, intended as an open-source replacement to extant closed APIs such as Steinberg's VST3 standard. CLAP also addresses several longstanding inefficiencies in the plugin-host architecture, including—to our inexpressible delight—parameter modulation. CLAP facilitates per-voice communication between host and plugin, which

neatly counters much of the griping and whinging of section 7.3 without much fuss.

Unfortunately, CLAP is still early in the development process, and does not yet enjoy the fruits of mass adoption. As of 2024, it is only supported by a small handful of plugins and host applications[41]. Furthermore, JUCE has no official CLAP support.

This work is motivated by accessibility: we wish to create an easy-to-use, approachable, and general purpose physical modeling tool. Most synthesizer users expect polyphonic modulation to be part of the synthesizer, not the host program; furthermore, this capability has not yet been implemented in most mainstream digital audio workstations. At present, delegating polyphonic modulation to a CLAP-compatible host program is incongruous with this work's principle of accessibility.

# 8 The Effect Chain

Like most digital synthesizers, Resonarium is equipped with an effect chain. We implement several basic effect units commonly found in existing digital instruments, such as filters, delays, and reverbs. Many of our effects are adopted from existing open source works; there is nothing particularly unique about our effect chain that can not be found elsewhere. The perfunctory nature of these components begs the question: why implement an effect chain at all? Indeed, there exist hundreds specialized third-party effect plugins that are, in many cases, objectively better than those implemented in this work.

As with modulation, the reasoning has something to do with polyphonic expression. In Resonarium, each internal voice is furnished with a stateful copy of the effect chain and its enclosed modules; therefore, effect chain parameters can be modulated polyphonically. This is not possible with third party effect plugins, since Resonarium (like all VST/AU3 plugins) mixes the outputs of all voices into a single output signal. This signal is subsequently handled by the enclosing host software, which cannot associate specific signal components with their corresponding synthesizer voices and MIDI events. Hence, all effect chains constructed at the host level must be monophonic.

We are dissatisfied with this state of affairs, and are interested in a solution—but even CLAP (section 7.3.1) does not claim to support polyphonic hosted effect chains. Many extant synthesizer plugins would require substantial refactoring, since voice signals would need to be kept separate during all stages of processing, instead of being summed into a main bus and sent to a singular output buffer, as is the custom. One awkward solution would be a $N$-voice "meta-plugin" that acts as a proxy between the host application and a digital instrument. This plugin would instantiate $N$ instances of the instrument plugin, along with $N$ instances of an effect chain containing one or several third party effect plugins. Then, the meta-plugin would manage polyphony, and thus each of the $N$ instrument instances would only handle a single note at a time. But

[41] The list is expanding rapidly! https://cleveraudio.org/hosts-and-plug-ins/

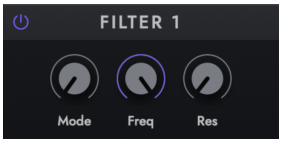
this solution is obviously inefficient; furthermore, automatically synchronizing state between each of the $N$ instances would be difficult if not impossible, since many plugins do not expose all their internal parameters to the host application.

We imagine that this problem has been tackled before. We eagerly accept suggestions and referrals to existing work in this particular domain.

A key disadvantage of a polyphonic effect chain is performance: for a synthesizer with $n$ active voices, a polyphonic effect chain incurs an $O(n)$ computational cost (versus $O(1)$ for a monophonic effect chain). Not all presets require polyphonic effects—for example, a patch that only employs unmodulated reverb may benefit little from a polyphonic effect chain. Resonarium, therefore, can be configured to use either a monophonic or polyphonic effect chain, depending on the user's creative intent and computational performance constraints.

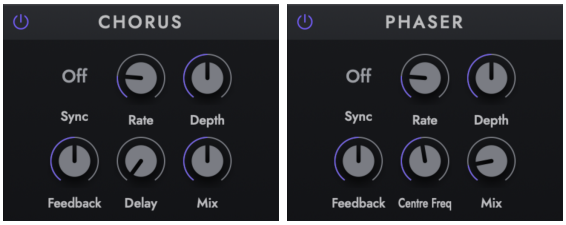
In the following sections, we briefly describe each module in Resonarium's effect chain.

## 8.1 Filters




Two filters are included in the effect chain: one at the beginning and one at the end. These filters are multi-mode second order state variable filters, identical to the *post filter* of section 4.7. As such, their parameters can handle audio-rate modulation without instability. Such extreme modulation often produces musically interesting results.

## 8.2 Chorus and Phaser




Both a chorus[42][42] module and a phaser[27] module are implemented in the effect chain. These effects are subtly distinct, but both induce a swirling, phasing sort of *unison*, as if many voices are being played at once. In the interest of expediency, we invoke the built-in JUCE[18] chorus[43] and phaser[44] widgets,

[42] https://ccrma.stanford.edu/~jos/pasp/Chorus_Effect.html
[43] https://docs.juce.com/master/classdsp_1_1Chorus.html
[44] https://docs.juce.com/master/classdsp_1_1Phaser.html

which are effective if unremarkable. In the future, we would like to improve these implementations, either by using more expressive open-source implementations, or implementing our own from scratch.

## 8.3 Distortion


Distortion [52, 42] is an indispensable tool in sound design, and we would be remiss not to include it in this work. Our distortion algorithm is nothing remarkable: we include several common waveshaping algorithms, such as the classic cubic nonlinearity[52, 37, 42], tangential soft-clipping, and hard clipping. We also adopt linear and sinusoidal wavefolding functions[9], adapted from the Vital synthesizer [51].

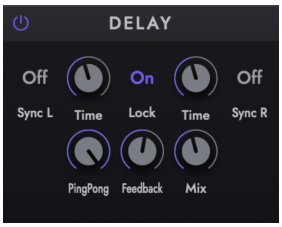
## 8.4 Amp Simulation


Recall that the original motivation for this work was guitar synthesis. Though this software has profitably strayed somewhat from its original directive, we still wish to give special attention to useful and relevant guitar effects. To this end, an amp module is implemented in the effect chain. Our expertise in amp modeling is limited (an amp, after all, won't fit into a backpack); therefore, we have adapted several brilliant open-source simulations from Airwindows [4].

We have thus far implemented five Airwindows amp models: LeadAmp, FireAmp, GrindAmp, BigAmp, and BassAmp. We have also implemented the DeRez2 bitcrusher as a related bonus. Further reading about these amp simulators can be found at [16]. We express our profound gratitude towards Chris Johnson for publishing such a wealth of DSP source code. This work would not be possible without people like him.

## 8.5 Delay

Resonarium is equipped with a standard stereo delay module. The left right channel delay lengths can be synchronized, or modulated independently. Fur-




thermore, a cross-channel "ping-pong" effect can be interpolated in or out.

We have experimented with placing a similar delay module in between the exciter and resonators, which enhances the musical potential of the instrument as a whole. We intend to implement this properly in the future. The obstacles here are aesthetic rather than technical: we are displeased with the current UI of the delay unit, and duplicating this UI as-is across multiple exciter modules would be unacceptable from a design perspective.

## 8.6 Reverb


Practically all modern electronic music productions employ reverb to emulate spatial context, improve realism, and generally enhance the quality of otherwise harsh and "in-your-face" electronic timbres. Much has been written about digital reverberation[42, 52], and we explored several open-source reverb implementations during development of this work. We ultimately adapted MVerb[24], an open source reverb based on the figure-of-eight topology of Dattorro[11].

# 9 UI Design and Implementation

> *C++, due to its rapid compilation speed, its ubiquity across web platforms, and its vast wealth of portable UI components; components; and documentation, is widely considered the dominant language and ecosystem for the development of intricate, responsive, and artistic user interfaces.*
>
> NOBODY

In practice, creative expression and aesthetics are inextricably joined at the hip. Indeed, the assertion is often tautological. Even artwork that rejects traditional notions of aesthetics, order, and beauty typically does so with respect to those same principles; its rejection of convention is, in the greater artistic context, implicitly a statement *about* convention. Music distinguishes itself from other creative media by means of its creative scope. A painter may care little about the color of his paintbrush handles, and a potter is agnostic to the shape, symmetry, and design of her kiln—in these instances; art is expressed through the process and the product, not the tool. Music, by contrast is performative: hence, the tool often *is* the art.

Musical instruments are imbued with an indisputable aesthetic magnetism. They look pretty, and that's part of the deal. Present a violin before a person who has never seen a violin, never heard a violin performance, who is entirely bereft of the mental context necessary to understand that a violin can produce music at all, and she will likely comment favorably on the object's pleasing symmetry, its seductive curves and impeccable craftsmanship, and its curious and alien design. She will want to touch it, to manipulate it. She does need to know what *music* is to understand that this is an object of rare and distinct beauty. So it goes for the tangled, intestinal pipeworks of a brass instrument, or a fractally intricate rack of synthesizer modules, glittering like a distant cityscape in the dark. The tools themselves are art; the fact that they happen to produce music is an *incredible* bonus.

It is worth asking whether this principle extends into the domain of *digital* instruments and associated software.

## 9.1 The Distinctive Art of Digital Synthesizer Design

Unlike physical instruments, digital instruments are nothing but software on a computer screen. Computer software—especially that which is designed for professionals—is generally recognized not for its aesthetic quality, but rather its efficiency, efficacy, and applicability. Such metrics are often incongruous with beauty for the sake of beauty.

Digital instruments, perhaps because of their shared genealogy with physical instruments, are a rare exception to this paradigm. In a monocropped design landscape dominated by flat, primitive shapes perforated by vast deserts of

whitespace, digital synthesizers are a last bastion of information-dense, skeuomorphic components and creative, unconventional design. This too is the case for digital audio effect units.

Figure 17 demonstrates two outstanding examples of synthesizer user interfaces. Figure 17a is distinctly not of this earth; it resembles a mysterious console on an alien spacecraft. It looks rubbery. One might expect the rectangular modules to wiggle when touched, like vegetable cubes suspended in a Jell-O salad. Figure 17b, meanwhile, is metallic, cold, and intricately technical. The common thread between both interfaces is that they are (a) weird and (b) indisputably enticing. They invite curiosity, wonder, and play. Indeed, the author was initially drawn to many years ago not just for its musical potential, but because the software *looks really cool*. Not all of will go on to become astronauts, but the next best thing might just be fiddling with knobs on a neat-looking synthesizer.


(a) Absynth (Native Instruments) [26] (b) Hive (u-he) [14]

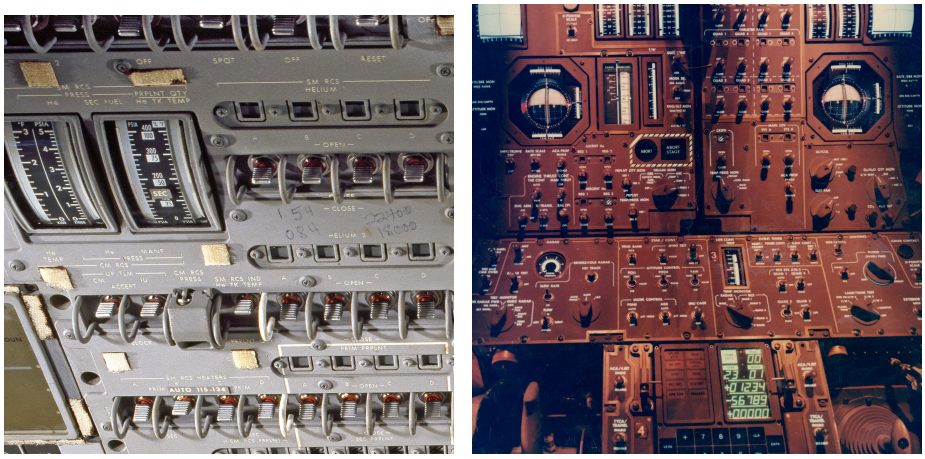
Figure 17: Two well-known digital synthesizers with distinctive graphic design.

## 9.2 The Design of Resonarium

This work attempts to inherit and perpetuate the unconventional legacy of synthesizer design. We intended to eschew "programmer art," as it were, and create an inciting and aesthetically consistent piece of software that invites interaction and experimentation. Many weeks of labor were invested in UI design, even though this effort has no effect on the timbre and quality of the sound.

We loosely source inspiration from aerospace control surfaces—in particular, those built for the Apollo lunar missions. Some reference images are displayed in Figure 18. The gray shades of our panels recall the industrial simplicity of the Apollo surfaces, while the black display insets that perforate Resonarium's user

interface are reminiscent of both electroluminescent displays and oscilloscopes commonly embedded in control surfaces of that era.



(a) The Apollo Command Module (CM) (b) The Apollo Lunar Module (LM)

Figure 18: The interface design of Apollo spacecraft. Observe the black inset instrument displays, the use of brackets to organize information, and the employ of Futura-style typefaces.

Finally, we draw attention to our use of Futura throughout Resonarium. Futura has a storied legacy; it is inextricably associated with aerospace and science fiction. It was a favorite of Stanley Kubrick, and enjoyed widespread use during the "golden age" of space exploration in the mid-to-late 20th century. It can still be found in the cockpits of many aircraft. To steer clear of any licensing quagmires, however, we employ a typeface called Jost [17], which is nearly indistinguishable from Futura.

We are only loosely inspired by mid-century aerospace controls. We do not seek to emulate the aesthetic perfectly. While the designs of Figure 18 are monochromatic and utilitarian, our interface does not shy away from vibrant color and motion, as in, e.g. the resonator banks of Figure 5.

### 9.2.1 The Logo

In principle, you can't spell *iconic* without an *icon*. Readers intimately familiar with the waveguide synthesis literature may observe that the icon for this project, pictured above, is a stylized diagram of the original Karplus-Strong string model: the venerable ancestor of many decades of fruitful research.

In practice, you just get *iconoclasts* instead. Many friends of the author, who are not intimately familiar with the waveguide synthesis literature, reject such nonsense about strings and models. They argue that the logo looks much more like a sea turtle with an extra head where one shouldn't be, and that this, in fact, should be the canonical interpretation. Beauty is in the eye of the beholder, so we leave any further subjective analysis to the reader.

There is another Easter egg hidden in the logo, which we immediately spoil in the following sentence. The radii of the three distinct circles in the logo (including the large semi-circle) are harmonically related. In fact, their ratio corresponds to the tuning of a minor triad! The voicing of the chord is a little odd, but no matter: any griping over such minutiae really misses the point of having a snazzy logo like this one.

## 9.3 Implementing the Design

The user interface of Resonarium was built entirely using JUCE's built-in UI tools, and written in straight C++. Building an effective, attractive, and performant UI using only C++ is about as uncomfortable and tedious as one might expect. In 2024, it is generally acknowledged that nobody in their right minds would try to build a complicated user interface in C++; fortunately, the art hangs on by the skin of its teeth because there are always a small number of people not in their right minds.

The JUCE UI framework does not support hardware acceleration: all drawing is done on the CPU. Some synthesizers, such as Vital [51], use OpenGL instead; however, OpenGL is deprecated on MacOS and the author is not familiar with low-level graphics programming in Metal or Vulkan. This reliance on the CPU inhibits the instantiation of visually complex widgets with many moving parts. Indeed, the performance impact of the oscilloscope panels[45] in this work are non-negligible, and we thus cap their refresh rates at visibly low levels. Furthermore, artistic effects such as glow effects and gradients are sluggish when rendered by the CPU. We are delighted to report that some people throw caution to the wind and implement them anyway [46].

We extend our profound gratitude towards Roland Rabien and his Gin [28] framework. Gin and JUCE, as it turns out, make a mean cocktail. This collection of plugin-agnostic JUCE utilities contains several UI widgets that we have adapted for this work; furthermore, the source code is a remarkable pedagogical resource for new plugin developers.

### 9.3.1 Debugging UIs in JUCE

A distinct challenge of Juce UI development is layout prototyping and debugging. JUCE does not natively support hot reloading, which means that any changes to the UI must be compiled manually. C++, unfortunately, is not known for its rapid compilation times.

[45] We refer to the black "scope" panels used to demonstrate the state of many modulator sources, as seen in, e.g. Figure 15

This project utilizes native JUCE components without third-party UI frameworks such as PluginGUIMagic, though we would like to integrate this framework in the future. For UI debugging and layout prototyping, we employ Melatonin Inspector [47], which provides a real-time component hierarchy viewer and property editor. While modifications made through the inspector cannot be baked into the source code, this tool has accelerated our UI development and debugging workflow by an order of magnitude, and we recommend it vigorously.

### 9.3.2 Web UIs in Audio Plugins

It is sometimes said that native desktop UIs are going the way of the dinosaur. Indeed, many modern desktop apps simply spin up a lightweight browser engine within a native window, and use web frameworks for both the front-end and back-end of the software. Juce 8 features improved support for web technologies [18][46], and it is now feasible to write plugin interfaces using popular frameworks such as React, or just straight HTML/CSS/JS.

The potential benefits of this approach are legion. It is an order of magnitude easier to design and implement complex animations, intricate widgets, and responsive UIs using the web stack, compared to JUCE's C++ UI framework. The JUCE UI tools are tainted by no small amount of "jank,"[47] and mitigating undefined or imperformant behavior in complex JUCE interfaces can be a Sisyphean effort. Furthermore, GPU acceleration can be accessed through web technologies.

However, we reserve some skepticism towards web-based user interfaces. Web components represent a new paradigm in the plugin development world. Both pedagogical resources and effective examples are scant. Some early gossip indicates that web views suffer from latency[48], as C++ plugin state (e.g. the value of a LFO, or the amplitude of a signal) must be packaged and routed through multiple layers of abstraction before it can be processed by some JavaScript hook and rendered as a HTML component.

We are also concerned about what a web-dominant future might mean for the conservation of styled, "maximalist" designs, as depicted in Figure 17. Information-rich skeuomorphism is an endangered mind-species within the global ideatic ecosystem, and audio plugins persevere as of a dwindling number of extant "nature reserves" for such threatened ideas. With modern web frameworks, it is almost *too easy* to mass-produce boilerplate interfaces using minimal, stylish cookie-cutter code and prepackaged components. We hope that this invasive species will not entirely supplant the stylized legacy of audio software design.

[46] https://juce.com/blog/juce-8-feature-overview-webview-uis/
[47] https://melatonin.dev/blog/dealing-with-jank-in-juce/
[48] https://forum.juce.com/t/webview-vs-juce-graphics/61156

# 10 The Future: Shortcomings, Improvements, and Further Research

> *The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.*
>
> Tom Cargill, Bell Labs

We stress that Resonarium is not a finished project. There are a host of enhancements that we wish to implement. Some are smaller tweaks and ergonomic improvements, others are grand in scope. We enumerate some of them here.

At the moment, however, our priority is stability and performance. This work has grown organically over many months, and the cumulative performance impact of dozens of small features is now longer negligible. We do not wish this work to suffer a slow death from a thousand paper cuts, so we currently prioritize internal refactoring over any shiny "2.0" features.

## 10.1 User Experience and Ease of Use

> *If you try to catch two rabbits, you will lose them both.*
>
> Unknown, Probably a Whole Bunch of People

In congruence with our principle of accessibility, a musician or sound designer should not need to be an expert on waveguide synthesis to effectively use Resonarium. Ideally, she should invoke her intuition about how an abstract "object" might "resonate," and apply this intuition to the instrument's controls.

Compared to extant work, however, Resonarium approaches physical modeling from a lower-level lens (section 1.2). We eschew the instrument metaphor, and instead directly expose a plethora of waveguide parameters. This granular perspective induces immediate challenges with respect to accessibility. Many people have a rough understanding of what happens when damping is applied to the resonant components of a piano, or what effect the pick position might have on a plucked string. By contrast, it is not immediately clear to the layperson what many of the parameters of Resonarium actually *do*.

This is an inexorable consequence of our adherence to the "component model" (section 1.2). We try to mitigate the impact of this albatross where possible. For instance, most parameters in Resonarium are associated with helpful tooltips that manifest on mouse over. Furthermore, we lean into allegory when naming parameters, and apply a moderate low-pass filter of metaphorical abstraction over the overwhelming complexity of implementation detail. For

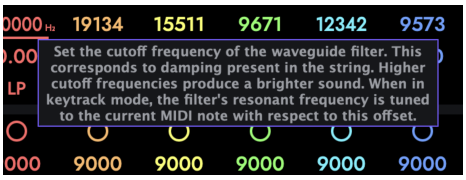



Figure 19: A handy tooltip that appears when the user hovers over a loop filter cutoff parameter.

example, we employ the term "resonator" to describe *a thing that resonates at a certain frequency*, even though "tuned feedback comb filter" or "waveguide" loop might be technically more appropriate.

We enjoy such metaphors: they are tractable sources of illumination for an otherwise obscure and unfamiliar method of sound synthesis. However, metaphors by nature smooth, tame, and obfuscate layers of granular complexity. Resonarium is built as a professional tool, and thus we intend to surface as many musically interesting parameter spaces as possible. When a parameter space contains regions of instability (i.e. a potential runaway feedback loop), but also contains regions of musical intrigue, our philosophy directs us to expose this parameter space to the musician, with the appropriate warnings and caveats.

Perfectly accommodating the needs of casual users whilst facilitating the niche demands of power users is a problem with no well-defined solution. We are designing one instrument, not two; therefore, a compromise somewhere along this spectrum must be struck. At present, we are somewhat dissatisfied by where Resonarium lands on this spectrum. While we endeavor to create a "sandbox" that encourages open-ended exploration and experimentation, we have observed that some of the most musically interesting configurations or rely on narrow configuration bands that are difficult to discover and navigate.

## 10.2 Undefined Behavior and Unstable Configurations

> *With a sufficient number of users of an API, it does not matter what you promise in the contract: all observable behaviors of your system will be depended on by somebody.*
>
> Hyrum Wright

We remark that the philosophical directive of the previous section, in which we favor musical expression over guaranteed stability, remains an awkward compromise. Ideally, it should be impossible for any parameter configuration to induce instability within the model. However, as of writing there exist a handful of edge cases with the potential to blow up speaker cabinets, and even some configuration states that are mathematically stable (i.e. no infinite-gain feed-

back loops) still produce a dangerously loud output signal. However, pruning these undesirable states would curtail the musical expressivity of the instrument; hence, in congruence with our philosophy we choose to leave such configuration spaces navigable by the musician.

A future area of research that interests us greatly would be the "taming" of such configuration states. We would prefer to develop mathematical bounds on a configuration space that guarantee a rough envelope of behavior, rather than seal off the parameters entirely. A prescient example is the high-amplitude resonance spikes in the *cascade* coupling mode, as we discuss in section 5.3. The problem is not necessarily that Resonarium can be coaxed to produce dangerously loud signals (this is possible with any synth: just crank every gain knob to its maximum)—rather, it is that such amplitude spikes are *unexpected*. It is not clear to the layperson, for example, why several parallel waveguide loops amplify each other if and only if their eigenfrequencies are harmonically related. Software design principles dictate that good software should be *predictable*: the user should be able to anticipate the effects of her actions, and such effects should be deterministic and consistent. Resonarium's Gordian knot of sixty-four feedback loops, and the acoustic epiphenomena they produce, somewhat miss the mark here.

It is tempting at this point to declare a bloody crusade of normalization throughout the codebase, and sterilize these eccentricities with prejudice. However, it would be both wise—and frugal—to show restraint here. Resonarium is not just another software product: it is an *instrument*, and instruments can be tempermental! We invoked Hyrum's epigram earlier for good reason: it is not uncommon for a sound designer to incorporate the effects of undefined behavior within her synthesizer into a fundamental part of her music. Some of the most creative synthesizer patches are built by pushing the underlying synth's parameters to their unforeseen extremes[49]. By definition, innovation cannot be predicted with any accuracy, for knowledge of when and how a thing will be invented in the future induces the ability to build it in the present. Many of the greatest innovations in history have been accidental. Creativity thrives not within the comfortable margins of well-defined behavior, but in that dark and unmapped expanse beyond the border. This is as true for music as it is for science as it is for life.

## 10.3 Improved Modulation

The modulation system in Resonarium is a cornerstone of this work, and we are pleased by the creative potential that it engenders. However, there are several improvements that need to be made before we can consider it complete.

[49] Roland released the TB-303 bassline generator in 1981 to dismal sales and withdrew it from production shortly after. Today it is known as one of the most iconic and recognizable electronic instruments of all time, not for its ability to create realistic bass notes, but rather for its squelchy, resonant "acid" sound that defined an entire *genre* of house music. Imagine the loss of creative potential energy were the designers of the TB-303 to, in the interest of ensuring that their bassline generator was used only for basslines, artificially limit the resonance, cutoff, and pitch of their filters to "reasonable" levels?

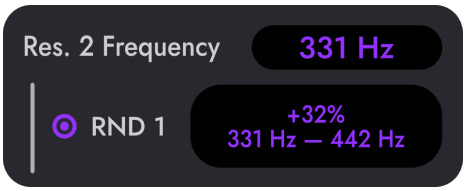



Figure 20: A mockup of a modulation depth control widget, intended as a reference during implementation. If only it were this easy to implement in JUCE!

The modulation system is somewhat rough as of writing: there are bugs, there is jank, and there are unimplemented quality-of-life features that would make everything much easier. We enumerate some of them in this section.

### 10.3.1 Turtles All the Way Down: Modulating Modulation

One obvious omission is an ability to modulate the modulation depth of an existing connection. As of writing, the modulation depth of a modulation connection is not itself exposed as a modulation destination. Therefore, a modulation signal cannot control the "strength" of another modulation connection. This inhibits the creation of complex and dynamic modulation patterns, and weakens the efficacy of the macro system as a higher-order control system for existing patches.

There is nothing mathematically difficult about this feature: all extant obstacles exist at the implementation level. A non-negligible refactoring of the modulation system would be required to elegantly support this feature; furthermore, several complex UI components would need to be implemented. This is rather tedious using Juce's C++ UI libraries. We intend to implement this in the future once Resonarium's core feature set is stable.[50]

### 10.3.2 Precision Depth Control

As of writing, it is difficult to precisely control modulation depth with respect to parameter value. There is no way to directly specify a modulation depth in terms of precise upper and lower bounds. If the user wishes to, say, set a frequency parameter to oscillate between 100 Hz and 200 Hz, she must manually adjust the modulation depth by clicking and dragging until the desired range is "close enough". This is undesirable, especially in the context of physical modeling, where small changes in parameter values can induce unpredictable downstream phenomena.

Improvements to this system are in development; they will be deployed in the near future.

[50] Many algorithmic sources in Resonarium contain a modulable "depth" parameter, which controls the amplitude of the modulation signal. In many cases, this parameter suffices as a modulation depth control.

## 10.4 Macroscopic State Interpolation

Recall that Resonarium is generalist, not specialist, technology: there are no "hard-coded" mechanics designed to emulate a particular instrument or family of instruments. Two distinct "instruments" (i.e. *presets* or *patches*) are differentiated entirely by the state of Resonarium's several hundred parameters.

It is therefore theoretically feasible to "interpolate" between two or more distinct presets, simply via parameter interpolation *en masse*. This would expose a great deal of musical potential: for example, it would be possible to create a "meta-preset" that would polyphonically morph between a flute preset and a waterphone preset in real time, depending on the musician's MPE gestures.

This is already possible, to an extent, via the existing modulation system and clever use of macros. One can create a macro that controls several parameters at once, and modulate the macro with a single control signal to synchronously modulate those several parameters. However, this is a cumbersome solution, and an explicit approach would be more elegant and user-friendly.

Let us define a *resonator bank state* as the set of all parameters that define the configuration state of a single resonator bank. We envision a feature called the "instrument table," which is to the resonator bank state as a wavetable is to an individual waveform. Just as a wavetable is composed of several distinct waveforms, often with interpolation between successive keyframes, an instrument table would be composed of several distinct resonator bank states. The user can then interpolate between these states in real time, and the instrument table will smoothly interpolate between the corresponding resonator bank states.

Development of this system has begun, and early results are promising. However, a proper implementation of the instrument table requires an extensive re-work of the internal state management system, as well as a significant number of UI overhauls and new widgets. None of this is conceptually difficult—but it is somewhat time-consuming. As of writing, the instrument table is not implemented in the work, but we plan to include it in a future update.

## 10.5 SIMD Optimization

Programming DSP software is not entirely unlike programming for a real-time operating system (RTOS), such as those used in spacecraft. In such extreme environments, programs are executed in a strict, deterministic order, and must complete their execution within a fixed time frame. While the stakes are arguably lower in synthesizer design, the principle is the same: the audio thread waits for no one. All audio code must terminate within a fixed period, or else the audio stream will be corrupted and "glitch". Such errors are universally unacceptable in all musical contexts: recording, production, live performance, and so on.

Resonarium is reasonably optimized when possible. The audio thread is sacrosanct, and we take care to minimize per-sample processing. Parameters that support sample-rate modulation are linearly interpolated across successive

audio blocks, which is both cheap and effective.

However, the practical reality of running up to 64 string models per voice is difficult to avoid. As such, improving performance is a priority. Many existing open-source synthesizers, such as Vital [51] and Surge XT [48], employ SIMD processing to improve polyphonic performance. However, we do not harness parallelism in this work. Much of our physical modeling code is entangled in serial dependency chains inside feedback loops, which does not lend itself naturally to parallelization. However, there is no interaction or feedback between distinct synthesizer voices. This exposes an enticing target for parallelization, and we intend to experiment with this in the future.

### 10.6 Machine Learning and Physical Modeling

No discussion on any novel technology in 2024 would be complete without some mention of machine learning, or artificial intelligence. Earlier, we expressed our frustration with Resonarium's temperamental configuration states. It is difficult to predict the effect of a particular parameter configuration, and the discovery of musically interesting presets can devolve into a tedious slog of trial and error. Machine learning techniques may be well-suited to this problem. We have recently reviewed some literature on the intersection of machine learning and physical modelling, such as differentiable models. Furthermore, we are aware of some efforts at CCRMA to generate synth patches using machine learning, which is more or less exactly what we envision for this work.

Recent work has directly applied differential DSP to physical modeling [23][51]. We would like to explore this research further, and see if it can be used in conjunction with this work.

## 11 Credits and Acknowledgements

Creative work is rarely composed in a vacuum. Indeed, Resonarium would not have been possible without the open contributions of numerous talented individuals, spread over several decades of fruitful research.

Our state-variable filter is adapted from Jatin Chowdhury's implementation, which can be found in the excellent `chowdsp-utils` library [10].

In addition, we are indebted to the `Gin` library [28]. Gin is a brilliant collection of general-purpose JUCE utilities that would be at home in any JUCE plugin. We adapted several components from Gin, including the modulation matrix, knob widgets, and preset serialization. None of these components are particularly unique or conceptually difficult, yet they are all a must-have for any modern audio plugin. We thank Roland Rabien sparing us from many hours of tedious tinkering.

As stated in section 9.3.1, we would have not been able to build such an intricate JUCE UI without the aid of Melatonin Inspector [47]. We also appreciate Melatonin Blur [46] for its CPU-based glows and drop shadows.

[51] https://ccrma.stanford.edu/events/jin-woo-lee-differentiable-physical-modeling-sound-synthesis-design-invers

This work was accelerated by Faust [13]. While we do not directly use generated Faust code in this work, Faust was invaluable for prototyping. Physical models can be unpredictable and difficult to debug: when a problem occurs, it is often unclear whether the issue lies in the model design or its C++ implementation. A parallel Faust implementation allows us to eliminate the second possibility, and precisely triangulate the source of any unexpected behavior. In the future, we would like to compare the performance of Faust-generated code with our hand-written C++ code.

Similarly, the Synthesis Toolkit (STK)[50] was used as a reference implementation for banded waveguide synthesis, which accelerated the development of the hybridized resonator model used in this work.

Finally, we extend our profound gratitude to those involved in digital waveguide synthesis research over the past decades. This work would not have been possible without the foundational research of Julius Smith, Perry Cook, and the rest of the physical modeling crew both at CCRMA and beyond. We are particularly grateful to Smith for publishing his DSP books online, for free, so that anyone—regardless of financial or academic status—can explore this art.

# References


[1] J.S. Abel, V. Valimaki, and J.O. Smith. "Robust, Efficient Design of Allpass Filters for Dispersive String Sound Synthesis". en. In: *IEEE Signal Processing Letters* 17.4 (Apr. 2010), pp. 406–409. ISSN: 1070-9908, 1558-2361. DOI: 10.1109/LSP.2010.2040924. URL: http://ieeexplore.ieee.org/document/5395664/ (visited on 06/19/2024).

[2] Jonathan S Abel and Julius O Smith. "Robust Design of Very High-Order Allpass Dispersion Filters". en. In: (2006).

[3] Ableton. *Push – a standalone expressive instrument — Ableton*. URL: https://www.ableton.com/en/push/ (visited on 10/02/2024).

[4] *Airwindows — handsewn bespoke digital audio*. URL: https://www.airwindows.com/ (visited on 10/18/2024).

[5] Stefan Bilbao. *Modal Synthesis*. URL: https://ccrma.stanford.edu/~bilbao/booktop/node14.html (visited on 10/11/2024).

[6] *Bitwig — Home — Bitwig*. www.bitwig.com. URL: https://www.bitwig.com/ (visited on 11/07/2024).

[7] Robert Bristow-Johnson. *Audio EQ Cookbook*. Audio EQ Cookbook. URL: https://www.w3.org/TR/audio-eq-cookbook/ (visited on 10/11/2024).

[8] Hal Chamberlin. *Musical applications of microprocessors*. The Hayden microcomputer series. Rochelle Park, N.J: Hayden Book Co, 1980. ISBN: 978-0-8104-5753-9.

[9] Jatin Chowdhury. *Wavefolder*. Wavefolder. URL: https://ccrma.stanford.edu/~jatin/ComplexNonlinearities/Wavefolder.html (visited on 10/18/2024).

[10] *Chowdhury-DSP/chowdsp_utils*. original-date: 2020-08-28T17:39:27Z. Oct. 20, 2024. URL: https://github.com/Chowdhury-DSP/chowdsp_utils (visited on 10/20/2024).

[11] Jon Dattorro. "1: Reverberator and Other Filters". In: ().

[12] Georg Essl and Perry R Cook. "BANDED WAVEGUIDES: TOWARDS PHYSICAL MODELING OF BOWED BAR PERCUSSION INSTRUMENTS". In: (1999).

[13] Grame-CNCM. *Faust Programming Language*. Faust Programming Language. URL: https://faust.grame.fr/ (visited on 10/03/2024).

[14] *Hive 2: Sleek, streamlined, supercharged — u-he*. URL: https://u-he.com/products/hive/ (visited on 11/05/2024).

[15] David A. Jaffe and Julius O. Smith. "Extensions of the Karplus-Strong Plucked-String Algorithm". In: *Computer Music Journal* 7.2 (1983), p. 56. ISSN: 01489267. DOI: 10.2307/3680063. URL: https://www.jstor.org/stable/3680063?origin=crossref (visited on 10/09/2024).

[16] Chris Johnson. *Airwindowpedia*. URL: https://www.airwindows.com/wp-content/uploads/Airwindopedia.txt (visited on 10/18/2024).

[17] *Jost**. URL: https://indestructibletype.com/Jost.html (visited on 10/22/2024).

[18] *JUCE Framework*. JUCE. URL: https://juce.com/ (visited on 10/18/2024).

[19] Kevin Karplus and Alex Strong. "Digital Synthesis of Plucked-String and Drum Timbres". In: *Computer Music Journal* 7.2 (1983), p. 43. ISSN: 01489267. DOI: 10.2307/3680062. URL: https://www.jstor.org/stable/3680062?origin=crossref (visited on 10/09/2024).

[20] Jari Kleimola. "Dispersion Modulation using Allpass Filters". en. In: (2008).

[21] Jari Kleimola et al. "SOUND SYNTHESIS USING AN ALLPASS FILTER CHAIN WITH AUDIO-RATE COEFFICIENT MODULATION". en. In: (2009).

[22] Charles Knight. *The National Cyclopaedia of Useful Knowledge*. The National Cyclopaedia of Useful Knowledge v. 1-2. Little, 1853. URL: https://books.google.com/books?id=iNhPAAAAMAAJ.

[23] Jin Woo Lee et al. *Differentiable Modal Synthesis for Physical Modeling of Planar String Sound and Motion Simulation*. 2024. arXiv: 2407.05516 [eess.AS]. URL: https://arxiv.org/abs/2407.05516.

[24] martineastwood. *martineastwood/mverb*. original-date: 2014-07-25T20:08:21Z. Oct. 14, 2024. URL: https://github.com/martineastwood/mverb (visited on 10/18/2024).

[25] MoForte. *About Geoshred*. moForte. 2024. URL: https://www.moforte.com/about/ (visited on 10/02/2024).

[26] *Native Instruments Absynth — Vintage Synth Explorer*. URL: https://www.vintagesynth.com/native-instruments/absynth (visited on 11/05/2024).

[27] *Phaser (effect)*. In: *Wikipedia*. Page Version ID: 1240632659. Aug. 16, 2024. URL: https://en.wikipedia.org/w/index.php?title=Phaser_(effect)&oldid=1240632659 (visited on 10/18/2024).

[28] Roland Rabien. *FigBug/Gin*. original-date: 2018-01-17T14:06:47Z. Oct. 19, 2024. URL: https://github.com/FigBug/Gin (visited on 10/20/2024).

[29] J. Rauhala and V. Valimaki. "Tunable dispersion filter design for piano synthesis". en. In: *IEEE Signal Processing Letters* 13.5 (May 2006), pp. 253–256. ISSN: 1070-9908. DOI: 10.1109/LSP.2006.870376. URL: http://ieeexplore.ieee.org/document/1618690/ (visited on 06/19/2024).

[30] Jukka Rauhala and Vesa Välimäki. "Dispersion Modeling in Waveguide Piano Synthesis Using Tunable Allpass Filters". en. In: (2006).

[31] *Reason Studios*. URL: https://www.reasonstudios.com/devices/objekt (visited on 12/04/2024).

[32] Roli. *ROLI — Instruments for Creators*. ROLI. URL: https://roli.com/us (visited on 10/02/2024).

[33] Andy Simper. *Linear Trapezoidal Integrated State Variable Filter With Low Noise Optimisation*. en. 2011.

[34] Andy Simper. *Simutaneous solving of all outputs of Linear SVF using trapezoidal integration in state space form*. 2021. URL: https://cytomic.com/files/dsp/SvfLinearTrapAllOutputs.pdf (visited on 07/27/2024).

[35] Andy Simper. *SvfLinearTrapOptimised2*. 2013. URL: https://cytomic.com/files/dsp/SvfLinearTrapOptimised2.pdf (visited on 10/02/2024).

[36] Julius Smith, Nelson Lee, and RealSimple Project. "Computational Acoustic Modeling with Digital Delay". In: ().

[37] Julius Smith and REALSIMPLE Project. "Making Virtual Electric Guitars and Associated Effects Using Faust". en. In: ().

[38] Julius O Smith. "Digital state-variable filters". In: *Center for Computer Research in Music and Acoustics (CCRMA), Department of Music, Stanford University, Stanford, California* 94305 (2014).

[39] Julius O Smith and Romain Michon. "NONLINEAR ALLPASS LADDER FILTERS IN FAUST". en. In: (2011).

[40] Julius O. Smith. "A Basic Introduction to Digital Waveguide Synthesis (for the Technically Inclined)". In: (Feb. 2023). URL: https://ccrma.stanford.edu/~jos/swgt/swgt.pdf (visited on 10/10/2024).

[41] Julius O. Smith. *Introduction to Digital Filters with Audio Applications*. http://www.w3k.org/books/: W3K Publishing, 2007. ISBN: 978-0-9745607-1-7.

[42] Julius O. Smith. *Physical audio signal processing: for virtual musical instruments and audio effects*. eng. Stanford, Calif: Stanford University, CCRMA, 2010. ISBN: 978-0-9745607-2-4.

[43] Julius O. Smith. "Physical Modeling Using Digital Waveguides". In: *Computer Music Journal* 16.4 (1992), p. 74. ISSN: 01489267. DOI: 10.2307/3680470. URL: https://www.jstor.org/stable/3680470?origin=crossref (visited on 06/20/2024).

[44] Julius O. Smith. *Spectral audio signal processing*. In collab. with Stanford University. Stanford, Calif: Stanford University, CCRMA, 2011. 654 pp. ISBN: 978-0-9745607-3-1.

[45] Julius O. Smith. "Techniques for Digital Filter Design and System Identification with Application to the Violin". PhD thesis. Stanford University (CCRMA), June 1983. URL: https://ccrma.stanford.edu/files/papers/stanm14.pdf (visited on 10/10/2024).

[46] Sudara. *sudara/melatonin_blur*. original-date: 2023-11-07T13:28:24Z. Oct. 15, 2024. URL: https://github.com/sudara/melatonin_blur (visited on 10/20/2024).

[47] *sudara/melatonin_inspector: A JUCE module that gives you the ability to inspect and visually edit (non-destructively) components in your UI.* URL: https://github.com/sudara/melatonin_inspector (visited on 10/20/2024).

[48] *surge-synthesizer/surge*. original-date: 2018-09-20T22:32:25Z. Oct. 27, 2024. URL: https://github.com/surge-synthesizer/surge (visited on 10/27/2024).

[49] The MIDI Association. *MIDI Polyphonic Expression*. 2022. URL: https://drive.google.com/file/d/1QGjiK2QPPbii8YmES3vDEZlP63OGLiKr/view.

[50] *thestk/stk*. original-date: 2013-09-17T12:54:47Z. Oct. 10, 2024. URL: https://github.com/thestk/stk (visited on 10/11/2024).

[51] Matt Tytel. *mtytel/vital*. original-date: 2021-02-10T20:36:35Z. Oct. 17, 2024. URL: https://github.com/mtytel/vital (visited on 10/18/2024).

[52] Udo Zolzer. *DAFX: digital audio effects*. 2nd ed. Chichester, West Sussex, England: Wiley, 2011. ISBN: 978-0-470-66599-2.